Linux From Scratch

版本 20200622-systemd，中文翻译版

发布于 2020 年 6 月 22 日

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

Linux From Scratch: 版本 20200622-systemd，中文翻译版: 发布于 2020 年 6 月 22 日

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

目录

序言

前言

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

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

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

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

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

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

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

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

--
Gerard Beekmans
gerard@linuxfromscratch.org

本书面向的读者

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

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

LFS 的另一个关键优势是，它允许您更好地控制您的系统，而不用依赖于其他人的 Linux 实现。您就像坐在驾驶座上一样，完全掌控系统的各个方面。

LFS 允许您创建非常紧凑的 Linux 系统。在安装传统的 Linux 发行版时，您往往不得不安装一大堆可能永远不会用到，甚至完全无法理解其必要性的程序。它们会浪费系统资源。您可能以为，有了现代的大容量硬盘和高速 CPU，就不需要考虑资源耗费的问题。然而，在一些情况下，即使不考虑其他问题，仅仅存储空间的约束就十分紧张。可引导 CD，USB 启动盘或者嵌入式系统就是典型代表。在这些领域中，LFS 是十分有用的。

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

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

此外，构建 LFS 系统还有很多好处，这里就不一一列举了。在所有原因中，最重要的是，在您编译和使用 LFS 的实践中，您将了解很多威力巨大的信息和知识。

LFS 的目标架构

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

对于构建 LFS 来说，构建 64 位系统相较于 32 位系统而言只会获得很小的收益。例如，在使用安装了 Core i7-4790 CPU 的系统测试构建 LFS-9.1 时，我们得到的实验数据为：

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

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

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

阅读本书需要的背景知识

构建 LFS 系统不是一项简单的任务。它需要您运用足够丰富的 Unix 系统管理知识来解决构建过程中的问题，并正确执行本书给出的命令。特别是，您至少需要拥有使用命令行 (shell) 来复制或移动文件和目录，列出目录或文件的内容，以及切换当前工作目录的能力。另外，我们希望您能够拥有一定水平的使用和安装 Linux 软件的知识。

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

在您开始构建 LFS 系统之前，我们建议您阅读下列材料：

- Software-Building-HOWTO http://www.tldp.org/HOWTO/Software-Building-HOWTO.html

  这是一份关于在 Linux 环境编译和安装 “一般的” Unix 软件包的详细指南。虽然这份文档比较老，但是它较好地总结了编译和安装软件的基本技巧。
- Beginner's Guide to Installing from Source http://moi.vonos.net/linux/beginners-installing-from-source/

  这份指南很好地总结了从源代码编译软件的基本技能和技巧。

LFS 和标准

LFS 的结构尽可能遵循 Linux 的各项标准。主要的标准有：

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

  LSB 由 4 个独立的标准组成：Core、Desktop、Runtime Language 和 Imaging。除了通用要求外，还有架构特定的要求。另外，还有两个处于试用阶段的标准：Gtk3 和 Graphics。LFS 试图遵循 LSB 对前一节讨论的那些架构的要求。

注意

许多人不认同 LSB 的要求。定义 LSB 的主要目的是保证专有软件能够在满足 LSB 的系统上正常运行。然而 LFS 是基于源代码的，用户拥有完全的控制权，有权选择不安装 LSB 要求的软件包。

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

LSB 要求的，由 LFS 提供的软件包

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

LSB 要求的，由 BLFS 提供的软件包

LSB Core:	At, Batch (At 的一部分), Cpio, Ed, Fcrontab, LSB-Tools, NSPR, NSS, PAM, Pax, Sendmail (或 Postfix，或 Exim), time
LSB Desktop:	Alsa, ATK, Cairo, Desktop-file-utils, Freetype, Fontconfig, Gdk-pixbuf, Glib2, GTK+2, Icon-naming-utils, Libjpeg-

	turbo, Libpng, Libtiff, Libxml2, MesaLib, Pango, Xdg-utils, Xorg
LSB Runtime Languages:	Python, Libxml2, Libxslt
LSB Imaging:	CUPS, Cups-filters, Ghostscript, SANE
LSB Gtk3 和 LSB Graphics (试用):	GTK+3

LSB 要求的，LFS 和 BLFS 均不提供的软件包

LSB Core:	无
LSB Desktop:	Qt4 (但提供了 Qt5)
LSB Runtime Languages:	无
LSB Imaging:	无
LSB Gtk3 和 LSB Graphics (试用):	无

本书选择软件包的逻辑

我们之前指出，LFS 的目标是构建一个完整且基本可用的系统。这包含所有重复构建 LFS 系统所需的软件包，以及在 LFS 提供的相对小的基础上根据用户需求，继续定制更完备的系统所必须的软件包。因此，LFS 并不是最小可用系统。LFS 中一些重要的软件包甚至不是必须安装的。下面列出了选择每个软件包的理由。

· Acl

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

· Attr

这个软件包包含管理文件系统对象的扩展属性的程序。

· Autoconf

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

· Automake

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

· Bash

这个软件包为系统提供一个 LSB core 要求的 Bourne Shell 接口。与其他 shell 软件包相比，它更加常用，且在基本 shell 功能的基础上有更好的扩展能力，因此在各种 shell 软件包中选择了它。

· Bc

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

· Binutils

该软件包包含链接器、汇编器，以及其他处理目标文件的工具。编译 LFS 系统以及运行在 LFS 之上的大多数软件包都需要该软件包中的程序。

· Bison

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

· Bzip2

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

· Check

这个软件包包含通用的文本宏处理器。它被其他程序用于构建工具。

· Coreutils

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

· D-Bus

这个软件包包含一些用于提供消息总线系统的程序，是一种应用程序之间通信的简单方式。

· DejaGNU

这个软件包包含用于测试其他程序的框架。

· Diffutils

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

· E2fsprogs

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

· Expat

这个软件包包含一个相对轻量级的 XML 解析库。Perl 模块 XML::Parser 需要该软件包。

· Expect

这个软件包包含一个自动和其他交互程序交互的脚本执行程序。一般用它测试其他程序。该软件包只被安装在临时工具链中。

· File

这个软件包包含用于判定给定文件类型的工具。一些软件包的构建脚本需要它。

· Findutils

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

· Flex

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

· Gawk

这个软件包包含用于操作文本文件的程序。它是 awk (Aho-Weinberg-Kernighan) 的 GNU 版本。它被许多其他软件包的编译脚本使用。

· GCC

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

· GDBM

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

· Gettext

这个软件包包含用于许多其他软件包的国际化和本地化的工具和库。

· Glibc

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

· GMP

这个软件包包含一些数学库，提供了用于任意精度算术的函数。编译 GCC 需要该软件包。

· Gperf

这个软件包包含一个能够根据键值集合生成完美散列函数的程序。Eudev 需要该软件包。

· Grep

这个软件包包含在文本中搜索指定模式的程序。它被多数软件包的编译脚本所使用。

· Groff

这个软件包包含用于处理和格式化文本的程序。它们的一项重要功能是生成 man 页面。

· GRUB

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

· Gzip

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

· Iana-etc

这个软件包包含网络服务和协议的描述数据。网络功能的正确运作需要该软件包。

· Inetutils

这个软件包包含基本网络管理程序。

· Intltool

这个软件包包含能够从源代码中提取可翻译字符串的工具。

· IProute2

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

· Kbd

这个软件包包含键盘映射文件，用于非美式键盘的键盘工具，以及一些控制台字体。

· Kmod

这个软件包包含用于管理 Linux 内核模块的程序。

· Less

这个软件包包含一个很好的文本文件查看器，它支持在查看文件时上下滚动。此外，Man-DB 使用该软件包来显示 man 页面。

· Libcap

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

· Libelf

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

· Libffi

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

· Libpipeline

Libpipeline 包含一个能够灵活、方便地操作子进程流水线的库。Man-DB 软件包要求这个库。

· Libtool

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

· Linux Kernel

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

· M4

这个软件包包含通用的文本宏处理器。它被其他程序用于构建工具。

· Make

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

· Man-DB

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

· Man-pages

这个软件包包含基本的 Linux man 页面的实际内容。

· Meson

这个软件包提供一个自动编译软件的工具。它的设计目标是最小化软件开发者不得不用于配置构建系统的时间。

· MPC

这个软件包包含用于复数算术的函数。GCC 需要该软件包。

· MPFR

这个软件包包含用于多精度算术的函数。GCC 需要该软件包。

· Ninja

这个软件包包含一个注重执行速度的小型构建系统。它被设计为读取高级构建系统输出的配置文件，并以尽量高的速度运行。

· Ncurses

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

· Openssl

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

· Patch

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

· Perl

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

· Pkg-config

这个软件包提供一个查询已经安装的库和软件包的元数据信息的程序。

· Procps-NG

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

· Psmisc

这个软件包包含一些显示当前运行的系统进程信息的程序，对系统管理非常有用。

· Python 3

这个软件包提供了一种解释性语言支持，它围绕代码可读性这一重点而设计。

· Readline

这个软件包包含一组库，提供命令行编辑和历史记录支持。Bash 需要该软件包。

· Sed

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

· Shadow

这个软件包包含用于安全地处理密码的程序。

· Systemd

这个软件包包含一个init程序，和一些附加的引导和系统控制支持。它能够替代 Sysvinit。许多商业发行版使用该软件包。

· Tar

这个软件包提供存档和提取功能，几乎每个 LFS 软件包都需要它才能被提取。

· Tcl

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

· Texinfo

这个软件包包含用于阅读、编写和转换 info 页面的程序。它被用于许多 LFS 软件包的安装过程中。

· Util-linux

这个软件包包含许多工具程序，其中有处理文件系统、终端、 分区和消息的工具。

· Vim

这个软件包包含一个编辑器，由于它与经典的 vi 编辑器相兼容，且拥有许多强大的功能，我们选择这个编辑器。编辑器的选择是非常主观的，如果希望的话，读者可以选择其他编辑器。

· XML::Parser

这个软件包是和 Expat 交互的 Perl 模块。

· XZ Utils

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

· Zlib

这个软件包包含一些程序使用的压缩和解压缩子程序。

· Zstd

这个软件包包含一些程序使用的压缩和解压缩子程序。它具有较高的压缩比，以及很宽的压缩比/速度折衷范围。

排版约定

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

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

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

有时，我们会将一个逻辑行拆分成两行或者多行，此时行末需要使用反斜线。

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

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

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

以上格式的文本 (等宽字体) 展示屏幕输出，通常是某个命令执行的结果。这种格式也用于展示文件名，例如 `/etc/ld.so.conf`。

强调的文本

以上格式的文本在本书中被用于一些目的。主要目的是强调重点。

http://www.linuxfromscratch.org/

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

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

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

`<需要替换的文本>`

不应该直接输入或复制粘贴这种尖括号包含的文本，而应该将其替换成合适内容。

`[可选的文本]`

方括号包含的文本是可选的，根据您的需要决定是否输入。

`passwd(5)`

以上格式被用于引用特定的手册 (man) 页面。数字表明页面来自系统手册中的某一节。例如，**passwd** 有两个 man 页面。LFS 安装命令将它们安装在 `/usr/share/man/man1/passwd.1` 和 `/usr/share/man/man5/passwd.5`。当本书使用 `passwd(5)` 时，它特指 man 页面 `/usr/share/man/man5/passwd.5`。**man passwd** 会显示它找到的第一个匹配 “passwd” 的 man 页面，即 `/usr/share/man/man1/passwd.1`。对于本例，您需要执行 **man 5 passwd** 才能阅读指定的 man 页面。多数 man 页面在各个章节中并不存在重名。因此，**man <程序名>** 一般是足够的。

本书结构

本书被分为以下三个部分。

第一部分 - 引言

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

第二部分 - 准备工作

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

第三部分 - 构建 LFS 交叉工具链和临时工具

第三部分提供在最终构建 LFS 系统时需要使用的工具的构建方法。

第四部分 - 构建 LFS 系统

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

第五部分 - 附录

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

勘误表

用于构建 LFS 系统的软件处于不断更新和改进的过程中。在本书发布后，一些软件可能公布安全警告和漏洞修复补丁。为了确认本书提供的软件包版本或者构建命令是否需要修正，以反映最新的安全补丁或其他漏洞修复，请在开始构建 LFS 之前阅读 http://www.linuxfromscratch.org/lfs/errata/systemd/。您应该关注勘误表列出的所有修正项，并在构建过程中注意对本书的相关章节进行修正。

第 I 部分 引言

第 1 章 引言

1.1. 如何构建 LFS 系统

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

您也可以选择不安装一个单独的发行版，而是使用某个商业发行版的 LiveCD。

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

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

第 6 章向您展示如何使用刚刚构建的交叉工具链，交叉编译一些基本工具。

之后在第 7 章中，进入一个 "chroot" 环境，并使用之前构建的工具，再构建一些额外的工具，这些额外工具对于构建和测试最终的系统是必要的。

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

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

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

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

1.2. 自上次发布以来的更新

在这一版本中，我们对 LFS 手册进行了大规模重构，使用一些技巧以避免对宿主系统的更改，并使构建过程更直截了当。

以下是自本书上一次发布之后，发生变化的软件包的清单。

已升级：

·

· Automake-1.16.2
· Bc 2.7.2
· Bison-3.6.4
· Coreutils-8.32
· E2fsprogs-1.45.6
· File-5.39

· Gawk-5.1.0
· GCC-10.1.0
· Gettext-0.20.2
· IANA-Etc-20200429
· IPRoute2-5.7.0
· Kmod-27
· Libcap-2.36
· Libelf-0.180 (来自 elfutils)
· Linux-5.7.2
· Man-DB-2.9.2
· Man-pages-5.07
· Meson-0.54.3
· Openssl-1.1.1g
· Perl-5.30.3
· Procps-ng-3.3.16
· Psmisc-23.3
· Python-3.8.3
· Systemd-245
· Tzdata-2020a
· Util-Linux-2.35.2
· Vim-8.2.0814
· XZ-Utils-5.2.5
· Zstd-1.4.5

已添加：

·

已移除：

·

1.3. 更新日志

这是 Linux From Scratch 手册的 20200622-systemd 版本，发布于 2020 年 6 月 22 日。如果该版本已经发布了六个月或更久，可能已经发布了更好的新版本。如果要查询是否有新版本，通过 http://www.linuxfromscratch.org/mirrors.html 访问一个 LFS 镜像站。

下面是本书自上一版本发布以来的更新日志。

更新日志记录：

· 2020 年 6 月 22 日
  · [renodr] - 修复 systemd-udevd 中的段错误。

- 2020 年 6 月 17 日
  - [bdubbs] — 更新到 meson-0.54.3。修复 #4673。
  - [bdubbs] — 更新到 man-pages-5.07。修复 #4669。
  - [bdubbs] — 更新到 linux-5.7.2。修复 #4662。
  - [bdubbs] — 更新到 iproute2-5.7.0。修复 #4668。
  - [bdubbs] — 更新到 file-5.39。修复 #4671。
  - [bdubbs] — 更新到 elfutils-0.180。修复 #4670。
  - [bdubbs] — 更新到 bison-3.6.4。修复 #4672。
- 2020 年 6 月 16 日
  - [bdubbs] — 将第 5 章拆分成三章。实现新的交叉编译 LFS 工具链和其他工具的构建方法，以简化将新系统与宿主系统隔离的过程。这将会是 LFS-10.0的起点。
- 2020 年 6 月 3 日
  - [renodr] — 使用补丁修复 systemd 无法被 GCC-10 编译的问题，而不是指定 CFLAGS。
  - [renodr] — 更新到 perl-5.30.3 (安全更新)。修复 #4664。
  - [renodr] — 更新到 dbus-1.12.18 (安全更新)。修复 #4665。
  - [renodr] — 更新到 man-db-2.9.2。修复 #4663。
  - [renodr] — 更新到 libcap-2.36。修复 #4666。
  - [renodr] — 更新到 bison-3.6.3。修复 #4667。
- 2020 年 5 月 31 日
  - [pierre] — 修复 bash 测试套件：增加一些从 /bin 指向 /tools 的符号链接，以 tty 所有者的权限创建 tester 用户，使用 su << EOF 运行 bash 测试，并显式定义 stdin (在 thomas 和 bdubbs 的帮助下完成)
- 2020 年 5 月 29 日
  - [xry111] — 将 flex 移到第 6 章更早的位置，这样 binutils 就可以使用它。
  - [xry111] — 从第 5 章删除 bzip2 和 flex
  - [xry111] — 将 zstd 移到第 6 章更早的位置，这样 file 和 GCC 就能使用它。
  - [bdubbs] — 以非特权用户身份运行 sed 和 findutils 测试。修复 #4661。
- 2020 年 5 月 28 日
  - [bdubbs] — 在开始第 6 章时添加一个非特权用户，名为 tester，以运行一些测试。该用户在这一章结束时被删除。
  - [bdubbs] — 更新到 zstd-1.4.5。修复 #4660。
  - [bdubbs] — 更新到 util-linux-2.35.2。修复 #4659。
  - [bdubbs] — 更新到 bison-3.6.2。修复 #4657。
  - [pierre] — 更新到 linux-5.6.15。修复 #4658。
- 2020 年 5 月 27 日
  - [pierre] — Bash: 记录测试结果。
- 2020 年 5 月 26 日

- [pierre] — Bash: 不要使用 "su -c command" 命令切换到 nobody 用户:这会移除控制终端并且导致一些测试失败。使用 "su << EOF" 代替它。
- [pierre] — 使用 "--bind" 挂载 /dev/pts，这样 "tty"就可以知道它在终端中运行。修复 coreutils 的一项测试。
- [pierre] — 增加补丁以修复 gold 测试套件中的一项失败测试，这个测试套件的某些测试确实需要 -fcommon。
- [pierre] — 修复 automake 测试套件中的一项失败测试。
- [pierre] — 更新到 vim-8.2.0814。
- [pierre] — 建立从 /tools/lib/locale 到 /usr/lib/locale/locale-archive 的符号链接，使得一些程序可以找到安装的 locale。修复 bison 和 man-db 的测试失败。

· 2020 年 5 月 21 日

- [pierre] — 修复使用内核配置 CONFIG_STACK_PROTECTOR_STRONG=y 时，在引导早期阶段发生的崩溃。

· 2020 年 5 月 16 日

- [bdubbs] — 更新到 meson-0.54.2。修复 #4656。
- [bdubbs] — 更新到 Python-3.8.3。修复 #4655。
- [bdubbs] — 更新到 bison-3.6.1。修复 #4654。
- [bdubbs] — 更新到 linux-5.6.13。修复 #4653。

· 2020 年 5 月 9 日

- [pierre] — 向 systemd 构建系统传递 -Wno-format-overflow，防止使用 GCC 10 时出现的错误。

· 2020 年 5 月 9 日

- [pierre] — 在第二遍构建 GCC 时打补丁，从而允许交叉编译 (仅用于新的交叉编译方法)。

· 2020 年 5 月 8 日

- [bdubbs] — 更新到 vim-8.2.0716。
- [bdubbs] — 更新到 bison-3.6。修复 #4652。
- [bdubbs] — 更新到 gcc-10.1.0。修复 #4651。
- [bdubbs] — 更新到 libcap-2.34。修复 #4650。
- [bdubbs] — 更新到 bc-2.7.2。修复 #4648。
- [bdubbs] — 更新到 linux-5.6.11。修复 #4649。

· 2020 年 5 月 1 日

- [bdubbs] — 更新到 tzdata-2020a。修复 #4644。
- [bdubbs] — 更新到 meson-0.54.1。修复 #4646。
- [bdubbs] — 更新到 iana-etc-20200429。修复 #4645。
- [bdubbs] — 更新到 linux-5.6.8。修复 #4630。

· 2020 年 4 月 23 日

- [ken] — 更新到 openssl-1.1.1g (安全修补)。修复 #4643.

· 2020 年 4 月 20 日

- [pierre] — 在 "调整工具链" 时，将 -isystem 改为 -idirafter。这允许在查找公共头文件目录前，先查找 g++ 内部头文件，这与正常的搜索顺序一致，但缺点是如果 /usr 和 /tools 中存在相同的头文件，就会找到 /tools 中的。几乎修复 #4641。

· 2020 年 4 月 19 日

- [pierre] — 对于本书的两个版本，都在第 5 章构建 util-linux。为 util-linux 库和头文件创建从 /usr 指向 /tools 的链接。将 util-linux 的 pkg-config 文件从 /tools 复制到 /usr，并将其中所有出现 /tools 的地方改为 /usr。另外，将 eudev 移到 util-linux 之前。修复 #4637，#4638，以及 #4642。
- [pierre] — 在第 5 章重新引入 flex，这样 ar 和 ranlib (binutils 中的工具) 可以链接到 libfl。这也允许运行 bison 的测试。修复 #4631。
- [pierre] — 添加配置开关 --with-curses，防止 readline.pc 将 termcap 作为内部库引用。修复 #4635。
- [pierre] — 通过将 gettext 移到 bison 之前，允许 bison 使用来自 gettext 的 libtextstyle.so 库。修复 #4634。
- [pierre] — 通过将 libcap 移到 shadow 之前，允许一些 shadow 程序使用 "setcap"。修复 #4633。
- [pierre] — 修复 shadow 软件包中一些程序硬编码的错误路径。修复 #4632。

· 2020 年 4 月 15 日

- [renodr] — 安装 systemd 的 man 页面。修复 #4627。
- [bdubbs] — 更新到 gawk-5.1.0。修复 #4629。
- [bdubbs] — 更新到 gettext-0.20.2。修复 #4628。
- [bdubbs] — 更新到 man-pages-5.06。修复 #4626。
- [bdubbs] — 更新到 bc-2.6.1。修复 #4625。
- [bdubbs] — 更新到 bison-3.5.4。修复 #4623。
- [bdubbs] — 更新到 iproute2-5.6.0。修复 #4622。
- [bdubbs] — 更新到 linux-5.6.4。修复 #4615。

· 2020 年 4 月 1 日

- [bdubbs] — 更新到 vim-8.2.0486。处理 #4500。
- [bdubbs] — 更新到 elfutils-0.179。修复 #4621。
- [bdubbs] — 更新到 meson-0.54.0。修复 #4620。
- [bdubbs] — 更新到 e2fsprogs-1.45.6。修复 #4619。
- [bdubbs] — 更新到 automake-1.16.2。修复 #4618。
- [bdubbs] — 更新到 xz-5.2.5。修复 #4617。
- [bdubbs] — 更新到 openssl-1.1.1f。修复 #4616。
- [bdubbs] — 更新到 perl-5.30.2。修复 #4614。

· 2020 年 3 月 29 日

- [bdubbs] — 文本更新，感谢 Kevin Buckley 的建议。

· 2020 年 3 月 19 日

- [renodr] — 更新到 systemd-245。修复 #45936

· 2020 年 3 月 18 日
  · [renodr] — 对内核配置进行微调，以符合 Linux-5.5 中配置选项在菜单中的位置。
· 2020 年 3 月 29 日
  · [bdubbs] — 文本更新，感谢 Kevin Buckley 的建议。
· 2020 年 3 月 19 日
  · [renodr] — 更新到 systemd-245。修复 #4593。
· 2020 年 3 月 18 日
  · [renodr] — 对内核配置进行微调，以符合 Linux-5.5 中配置选项在菜单中的位置。
· 2020 年 3 月 15 日
  · [bdubbs] — 更新到 gcc-9.3.0。修复 #4613。
  · [bdubbs] — 更新到 bc-2.6.0。修复 #4612。
  · [bdubbs] — 更新到 bison-3.5.3。修复 #4611。
  · [bdubbs] — 更新到 linux-5.5.9。修复 #4610。
  · [bdubbs] — 更新到 coreutils-8.32。修复 #4609。
· 2020 年 3 月 2 日
  · [bdubbs] — 更新到 Python-3.8.2。修复 #4606。
  · [bdubbs] — 更新到 meson-0.52.2。修复 #4605。
  · [bdubbs] — 更新到 man-db-2.9.1。修复 #4604。
  · [bdubbs] — 更新到 kmod-27。修复 #4603。
  · [bdubbs] — 更新到 procps-3.3.16。修复 #4602。
  · [bdubbs] — 更新到 psmisc-23.3。修复 #4601。
  · [bdubbs] — 更新到 libcap-2.33。修复 #4608。
  · [bdubbs] — 更新到 linux-5.5.7。修复 #4598。
· 2020 年 3 月 1 日
  · [bdubbs] — LFS-9.1 发布。

1.4. 相关资源

1.4.1. FAQ

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

1.4.2. 邮件列表

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

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

1.4.3. IRC

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

1.4.4. 镜像站

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

1.4.5. 联系信息

请直接将您的问题和评论发送到某个 LFS 邮件列表 (上面已经给出)。

1.5. 如何求助

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

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

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

1.5.1. 需要提供的信息

除了简要描述您遇到的问题外，您应该在求助邮件中附带下列必要信息。

- LFS 手册的版本 (这本书的版本是 20200622-systemd)
- 构建 LFS 时使用的宿主发行版名称和版本
- 宿主系统需求脚本的输出
- 出现问题的软件包或书内章节
- 程序输出的原始错误消息，或者出现的症状
- 您是否进行了超出本书内容的操作

注意

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

1.5.2. 配置脚本的问题

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

1.5.3. 编译错误

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

```
gcc -DALIASPATH=\"/mnt/lfs/usr/share/locale:.\"
-DLOCALEDIR=\"/mnt/lfs/usr/share/locale\"
-DLIBDIR=\"/mnt/lfs/usr/lib\"
-DINCLUDEDIR=\"/mnt/lfs/usr/include\" -DHAVE_CONFIG_H -I. -I.
-g -O2 -c getopt1.c
gcc -g -O2 -static -o make ar.o arscan.o commands.o dir.o
expand.o file.o function.o getopt.o implicit.o job.o main.o
misc.o read.o remake.o rule.o signame.o variable.o vpath.o
default.o remote-stub.o version.o opt1.o
-lutil job.o: In function `load_too_high':
/lfs/tmp/make-3.79.1/job.c:1565: undefined reference
to `getloadavg'
collect2: ld returned 1 exit status
make[2]: *** [make] Error 1
make[2]: Leaving directory `/lfs/tmp/make-3.79.1'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory `/lfs/tmp/make-3.79.1'
make: *** [all-recursive-am] Error 2
```

对于本例来说，许多人会只附带靠下的一行：

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

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

可以在线阅读一篇关于如何在网络上提问的精彩文章：http://catb.org/~esr/faqs/smart-questions.html。在您提问时，阅读并遵从这篇文章的建议，可以增加您得到帮助的可能性。

第 II 部分 准备工作

第 2 章 准备宿主系统

2.1. 概述

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

2.2. 宿主系统需求

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

比下列最低版本更古老的版本可能正常工作，但作者没有进行测试。

- **Bash-3.2** (/bin/sh 必须是到 bash 的符号链接或硬连接)
- **Binutils-2.25** (比 2.34 更新的版本未经测试，不推荐使用)
- **Bison-2.7** (/usr/bin/yacc 必须是到 bison 的链接，或者是一个执行 bison 的小脚本)
- **Bison-2.7** (/usr/bin/yacc 必须是到 bison 的链接，或者是一个执行 bison 的小脚本)
- **Coreutils-6.9**
- **Diffutils-2.8.1**
- **Findutils-4.2.31**
- **Gawk-4.0.1** (/usr/bin/awk 必须是到 gawk 的链接)
- **GCC-6.2** 包括 C++ 编译器, **g++** (比 10.1.0 更新的版本未经测试，不推荐使用)
- **Glibc-2.11** (比 2.31 更新的版本未经测试，不推荐使用)
- **Grep-2.5.1a**
- **Gzip-1.3.12**
- **Linux Kernel-3.2**

  内核版本的要求是为了符合第 6 章中编译 glibc 时开发者推荐的配置选项。udev 也要求一定的内核版本。

  如果宿主内核比 3.2 更早，您需要将内核升级到较新的版本。升级内核有两种方法，如果您的发行版供应商提供了 3.2 或更新的内核软件包，您可以直接安装它。如果供应商没有提供一个足够新的内核包，或者您不想安装它，您可以自己编译内核。编译内核和配置启动引导器 (假设宿主使用 GRUB) 的步骤在第 10 章中。
- **M4-1.4.10**
- **Make-4.0**
- **Patch-2.5.4**
- **Perl-5.8.8**
- **Python-3.4**
- **Sed-4.1.5**
- **Tar-1.22**
- **Texinfo-4.7**
- **Xz-5.0.0**

重要

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

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

```
cat > version-check.sh << "EOF" #!/bin/bash #
Simple script to list version numbers of critical development tools export
LC_ALL=C bash --version | head -n1 | cut -d" " -f2-4 MYSH=$(readlink -f
/bin/sh)  echo "/bin/sh -> $MYSH" echo $MYSH | grep -q bash || echo
"ERROR: /bin/sh does not point to bash" unset MYSH echo -n "Binutils: "; ld
--version | head -n1 | cut -d" " -f3- bison --version | head -n1 if [ -h
/usr/bin/yacc ]; then echo "/usr/bin/yacc -> `readlink -f /usr/bin/yacc`";
elif [ -x /usr/bin/yacc ]; then echo yacc is `/usr/bin/yacc --version | head
-n1` else echo "yacc not found" fi bzip2 --version 2>&1 <
/dev/null | head -n1 | cut -d" " -f1,6- echo -n "Coreutils: "; chown
--version | head -n1 | cut -d")" -f2 diff --version | head -n1 find
--version | head -n1 gawk --version | head -n1 if [ -h /usr/bin/awk ]; then
echo "/usr/bin/awk -> `readlink -f /usr/bin/awk`"; elif [ -x /usr/bin/awk
]; then echo awk is `/usr/bin/awk --version | head -n1` else echo "awk not
found" fi
```

```
gcc --version | head -n1 g++ --version | head -n1 ldd --version |
head -n1 | cut -d" " -f2- # glibc version grep --version | head -n1 gzip
--version | head -n1 cat /proc/version m4 --version | head -n1 make
--version | head -n1 patch --version | head -n1 echo Perl `perl -V:version`
python3 --version sed --version | head -n1 tar --version | head -n1 makeinfo
--version | head -n1 # texinfo version xz --version | head -n1
```

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

bash version-check.sh
```

2.3. 分阶段构建 LFS

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

2.3.1. 第 1–4 章

这些章节是在宿主系统完成的。在重启后，注意下列事项:

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

2.3.2. 第 5–6 章

· /mnt/lfs 分区需要重新挂载。

· 第 5 章的所有步骤必须由用户 lfs 完成。在执行第 5 章的任务时，必须先执行 **su - lfs** 命令。

· 编译过程的一般说明中的过程是关键的。如果在安装软件包时感觉不对劲，确认之前解压的源码包已经被删除，然后重新解压源码包的文件，重新执行该软件包对应章节的所有命令。

2.3.3. 第 7–10 章

· /mnt/lfs 分区需要重新挂载。

· 从 “改变所有权” 到 “进入 Chroot 环境” 的一些操作必须以 root 身份完成，且 LFS 环境变量必须为 root 用户设定。

· 在进入 chroot 环境时，LFS 环境变量必须为 root 设置好。之后就不需要 LFS 变量。

· 虚拟文件系统必须挂载好。在进入 chroot 环境之前，请切换到一个宿主系统的虚拟终端，以 root 身份执行第 7.3.2 节 “挂载和填充 /dev”和第 7.3.3 节 “挂载虚拟内核文件系统”中的命令。

2.4. 创建新的分区

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

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

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

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

注意

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

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

2.4.1. 其他分区问题

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

2.4.1.1. 根分区

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

2.4.1.2. 交换 (Swap) 分区

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

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

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

2.4.1.3. Grub Bios 分区

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

注意

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

2.4.1.4. 常用分区

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

- boot – 高度推荐。这个分区可以存储内核和其他引导信息。为了减少大磁盘可能引起的问题，建议将 /boot 分区设为第一块磁盘的第一个分区。为它分配 200 MB 就绰绰有余。
- /home – 高度推荐。独立的 /home 分区可以在多个发行版或 LFS 系统之间共享 home 目录和用户设置。它的尺寸一般很大，取决于硬盘的可用空间。
- /usr – 一个独立的 /usr 分区一般被用于瘦客户端或无盘站的服务器，LFS 一般不需要。为它划分 10GB 足够满足多数系统的要求。
- /opt – 这个目录往往被用于在 BLFS 中安装 Gnome 或 KDE 等大型软件，以免把大量文件塞进 /usr 目录树。如果将它划分为独立分区，5 到 10 GB 一般就足够了。
- /tmp – 一个独立的 /tmp 分区是很少见的，但在配置瘦客户端时很有用。如果分配了这个分区，大小一般不会超过几个 GB。
- /usr/src – 将它划分为独立分区，可以用于存储 BLFS 源代码，并在多个 LFS 系统之间共享它们。它也可以用于编译 BLFS 软件包。30-50 GB 的分区可以提供足够的空间。

如果您希望在启动时自动挂载任何一个独立的分区，就要在 `/etc/fstab` 文件中说明。有关指定分区的细节将在 第 10.2 节 “创建 /etc/fstab 文件” 中讨论。

2.5. 在分区上建立文件系统

现在我们建立好了空白分区，可以在分区上建立文件系统。LFS 可以使用 Linux 内核能够识别的任何文件系统，最常见的是 ext3 和 ext4。文件系统的选型是一个复杂的问题，要综合考虑分区的大小，以及其中所存储文件的特征。例如：

ext2
: 适用于不经常更新的小分区，例如 /boot。

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

ext4
: 是 ext 文件系统家族的最新成员，它具有纳秒精度时间戳、 超大 (16 TB) 文件支持等新功能，速度也更快。

其他文件系统，包括 FAT32, NTFS, ReiserFS, JFS 和 XFS 在特定场合也很有用。关于这些文件系统的更多信息，可以在 http://en.wikipedia.org/wiki/Comparison_of_file_systems 找到。

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

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

命令中 `<xxx>` 应该替换成 LFS 分区的名称。

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

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

命令中 `<yyy>` 应该替换成 `swap` 分区的名称。

2.6. 设置 $LFS 环境变量

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

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

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

小心

无论何时，如果您离开并重新进入了工作环境，一定要确认 LFS 的设定值和您离开工作环境时相同。(例如，使用 **su** 切换到 root 或者其他用户时。) 请执行以下命令，检查 LFS 的设置是否正确：

```
echo $LFS
```

确认该命令的输出是您构建 LFS 的位置，如果您使用本书提供的例子，那么输出应该是 /mnt/lfs。如果输出不正确，使用前文给出的命令，将 $LFS 设置成正确的目录名。

注意

确保 LFS 始终正确的一种方法是：编辑您的主目录中的 .bash_profile，以及/root/.bash_profile，为它们加入上述设置并导出 LFS 变量的 export 命令。还要确认 /etc/passwd 中为每个需要使用 LFS 变量的用户指定的 shell 都是 bash，以保证每次登录时都执行 .bash_profile 中的命令。

另外还要考虑登录宿主系统的方式，如果您使用图形显示管理器登录，再启动虚拟终端，那么 .bash_profile 一般不会被虚拟终端执行。此时，应该将 export 命令加入到您使用的用户和 root 用户的 .bashrc 文件中。另外，如果以非交互模式启动 bash，有的发行版不会执行 .bashrc 中的指令。此时一定要在使用环境变量前添加 export 命令。

2.7. 挂载新的分区

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

输入以下命令以创建挂载点，并挂载 LFS 文件系统：

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

将 <xxx> 替换成 LFS 分区的代号。

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

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

将 <xxx> 和 <yyy> 替换成对应的分区代号。

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

警告

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

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

如果您使用了多个分区，它们都需要添加到 fstab 中。

如果您使用了 swap 分区，使用 **swapon** 命令启用它：

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

将 <zzz> 替换成 swap 分区的名称。

现在我们准备好了工作环境，可以下载软件包了。

第 3 章 软件包和补丁

3.1. 概述

本章包含了构建基本的 Linux 系统时需要下载的软件包列表。我们给出的版本号对应于已经确定可以正常工作的版本，本书是基于这些版本编写的。我们强烈反对使用更新的版本，因为特定版本可用的构建命令未必适用于新版本。最新版本的软件包可能有需要排查的问题，我们会在本书的开发过程中进行排查，将解决方案找到并固定下来。

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

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

为了创建这个目录，在开始下载软件包之前，以root身份执行：

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

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

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

存在多种获取构建 LFS 必须的软件包和补丁的方法：

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

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

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

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

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

使用上面的各种方法获取文件后，都可以执行这项检查。

3.2. 全部软件包

下载或者用其他方法获取下列软件包。

- **Acl (2.2.53) - 513 KB:**

主页：https://savannah.nongnu.org/projects/acl
下载地址：http://download.savannah.gnu.org/releases/acl/acl-2.2.53.tar.gz
MD5 校验和：`007aabf1dbb550bcddde52a244cd1070`

· **Attr (2.4.48) - 457 KB:**
主页：https://savannah.nongnu.org/projects/attr
下载地址：http://download.savannah.gnu.org/releases/attr/attr-2.4.48.tar.gz
MD5 校验和：bc1e5cb5c96d99b24886f1f527d3bb3d

· **Autoconf (2.69) - 1,186 KB:**
主页：http://www.gnu.org/software/autoconf/
下载地址：http://ftp.gnu.org/gnu/autoconf/autoconf-2.69.tar.xz
MD5 校验和：50f97f4159805e374639a73e2636f22e

· **Automake (1.16.2) - 1,510 KB:**
主页：http://www.gnu.org/software/automake/
下载地址：http://ftp.gnu.org/gnu/automake/automake-1.16.2.tar.xz
MD5 校验和：6cb234c86f3f984df29ce758e6d0d1d7

· **Bash (5.0) - 9,898 KB:**
主页：http://www.gnu.org/software/bash/
下载地址：http://ftp.gnu.org/gnu/bash/bash-5.0.tar.gz
MD5 校验和：2b44b47b905be16f45709648f671820b

· **Bc (2.7.2) - 185 KB:**
主页：https://git.yzena.com/gavin/bc
下载地址：https://github.com/gavinhoward/bc/releases/download/2.7.2/bc-2.7.2.tar.xz
MD5 校验和：28235ceaf2280b909591ace7a3a4f051

· **Binutils (2.34) - 21,131 KB:**
主页：http://www.gnu.org/software/binutils/
下载地址：http://ftp.gnu.org/gnu/binutils/binutils-2.34.tar.xz
MD5 校验和：664ec3a2df7805ed3464639aaae332d6

· **Bison (3.6.4) - 2,415 KB:**
主页：http://www.gnu.org/software/bison/
下载地址：http://ftp.gnu.org/gnu/bison/bison-3.6.4.tar.xz
MD5 校验和：08bf8aa8334d7f817b7b24509ef412bf

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

· **Check (0.14.0) - 753 KB:**
主页：https://libcheck.github.io/check
下载地址：https://github.com/libcheck/check/releases/download/0.14.0/check-0.14.0.tar.gz
MD5 校验和：270e82a445be6026040267a5e11cc94b

· **Coreutils (8.32) - 5,418 KB:**
主页：http://www.gnu.org/software/coreutils/
下载地址：http://ftp.gnu.org/gnu/coreutils/coreutils-8.32.tar.xz
MD5 校验和：022042695b7d5bcf1a93559a9735e668

· **D-Bus (1.12.18) - 2,048 KB:**
主页：https://www.freedesktop.org/wiki/Software/dbus
下载地址：https://dbus.freedesktop.org/releases/dbus/dbus-1.12.18.tar.gz
MD5 校验和：4ca570c281be35d0b30ab83436712242

· **DejaGNU (1.6.2) - 514 KB:**
主页：http://www.gnu.org/software/dejagnu/
下载地址：http://ftp.gnu.org/gnu/dejagnu/dejagnu-1.6.2.tar.gz
MD5 校验和：`e1b07516533f351b3aba3423fafeffd6`

· **Diffutils (3.7) - 1,415 KB:**
主页：http://www.gnu.org/software/diffutils/
下载地址：http://ftp.gnu.org/gnu/diffutils/diffutils-3.7.tar.xz
MD5 校验和：`4824adc0e95dbbf11dfbdfaad6a1e461`

· **E2fsprogs (1.45.6) - 7,753 KB:**
主页：http://e2fsprogs.sourceforge.net/
下载地址：https://downloads.sourceforge.net/project/e2fsprogs/e2fsprogs/v1.45.6/e2fsprogs-1.45.6.tar.gz
MD5 校验和：`cccfb706d162514e4f9dbfbc9e5d65ee`

· **Elfutils (0.180) - 8,867 KB:**
主页：https://sourceware.org/ftp/elfutils/
下载地址：https://sourceware.org/ftp/elfutils/0.180/elfutils-0.180.tar.bz2
MD5 校验和：`23feddb1b3859b03ffdbaf53ba6bd09b`

· **Expat (2.2.9) - 413 KB:**
主页：https://libexpat.github.io/
下载地址：https://prdownloads.sourceforge.net/expat/expat-2.2.9.tar.xz
MD5 校验和：`d2384fa607223447e713e1b9bd272376`

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

· **File (5.39) - 932 KB:**
主页：https://www.darwinsys.com/file/
下载地址：ftp://ftp.astron.com/pub/file/file-5.39.tar.gz
MD5 校验和：`1c450306053622803a25647d88f80f25`

注意
File (5.39) 可能已经不能从列表中给出的位置下载。该地址的网站管理员在发布新版本后有时会删除旧版本。访问 http://www.linuxfromscratch.org/lfs/download.html#ftp 中的备用下载地址，可能找到正确的版本。

· **Findutils (4.7.0) - 1,851 KB:**
主页：http://www.gnu.org/software/findutils/
下载地址：http://ftp.gnu.org/gnu/findutils/findutils-4.7.0.tar.xz
MD5 校验和：`731356dec4b1109b812fecfddfead6b2`

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

· **Gawk (5.1.0) - 3,081 KB:**
主页：http://www.gnu.org/software/gawk/
下载地址：http://ftp.gnu.org/gnu/gawk/gawk-5.1.0.tar.xz
MD5 校验和：8470c34eeecc41c1aa0c5d89e630df50

· **GCC (10.1.0) - 72,844 KB:**
主页：https://gcc.gnu.org/
下载地址：http://ftp.gnu.org/gnu/gcc/gcc-10.1.0/gcc-10.1.0.tar.xz
MD5 校验和：7d48e00245330c48b670ec9a2c518291

· **GDBM (1.18.1) - 920 KB:**
主页：http://www.gnu.org/software/gdbm/
下载地址：http://ftp.gnu.org/gnu/gdbm/gdbm-1.18.1.tar.gz
MD5 校验和：988dc82182121c7570e0cb8b4fcd5415

· **Gettext (0.20.2) - 9,292 KB:**
主页：http://www.gnu.org/software/gettext/
下载地址：http://ftp.gnu.org/gnu/gettext/gettext-0.20.2.tar.xz
MD5 校验和：0cf5f68338d5d941bbf9ac93b847310f

· **Glibc (2.31) - 16,286 KB:**
主页：http://www.gnu.org/software/libc/
下载地址：http://ftp.gnu.org/gnu/glibc/glibc-2.31.tar.xz
MD5 校验和：78a720f17412f3c3282be5a6f3363ec6

· **GMP (6.2.0) - 1,966 KB:**
主页：http://www.gnu.org/software/gmp/
下载地址：http://ftp.gnu.org/gnu/gmp/gmp-6.2.0.tar.xz
MD5 校验和：a325e3f09e6d91e62101e59f9bda3ec1

· **Gperf (3.1) - 1,188 KB:**
主页：http://www.gnu.org/software/gperf/
下载地址：http://ftp.gnu.org/gnu/gperf/gperf-3.1.tar.gz
MD5 校验和：9e251c0a618ad0824b51117d5d9db87e

· **Grep (3.4) - 1,520 KB:**
主页：http://www.gnu.org/software/grep/
下载地址：http://ftp.gnu.org/gnu/grep/grep-3.4.tar.xz
MD5 校验和：111b117d22d6a7d049d6ae7505e9c4d2

· **Groff (1.22.4) - 4,044 KB:**
主页：http://www.gnu.org/software/groff/
下载地址：http://ftp.gnu.org/gnu/groff/groff-1.22.4.tar.gz
MD5 校验和：08fb04335e2f5e73f23ea4c3adbf0c5f

· **GRUB (2.04) - 6,245 KB:**
主页：http://www.gnu.org/software/grub/
下载地址：https://ftp.gnu.org/gnu/grub/grub-2.04.tar.xz
MD5 校验和：5aaca6713b47ca2456d8324a58755ac7

· **Gzip (1.10) - 757 KB:**
主页：http://www.gnu.org/software/gzip/
下载地址：http://ftp.gnu.org/gnu/gzip/gzip-1.10.tar.xz
MD5 校验和：691b1221694c3394f1c537df4eee39d3

· **Iana-Etc (20200429) - 574 KB:**
主页：http://freecode.com/projects/iana-etc
下载地址：http://anduin.linuxfromscratch.org/LFS/iana-etc-20200429.tar.gz
MD5 校验和：f9f7cda56c0ebe6ac2fa69a0be5d5400

· **Inetutils (1.9.4) - 1,333 KB:**
主页：http://www.gnu.org/software/inetutils/
下载地址：http://ftp.gnu.org/gnu/inetutils/inetutils-1.9.4.tar.xz
MD5 校验和：87fef1fa3f603aef11c41dcc097af75e

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

· **IPRoute2 (5.7.0) - 747 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/iproute2-5.7.0.tar.xz
MD5 校验和：da22ab8562eda56ae232872fa72e4870

· **Kbd (2.2.0) - 1,090 KB:**
主页：http://ftp.altlinux.org/pub/people/legion/kbd
下载地址：https://www.kernel.org/pub/linux/utils/kbd/kbd-2.2.0.tar.xz
MD5 校验和：d1d7ae0b5fb875dc082731e09cd0c8bc

· **Kmod (27) - 537 KB:**
下载地址：https://www.kernel.org/pub/linux/utils/kernel/kmod/kmod-27.tar.xz
MD5 校验和：3973a74786670d3062d89a827e266581

· **Less (551) - 339 KB:**
主页：http://www.greenwoodsoftware.com/less/
下载地址：http://www.greenwoodsoftware.com/less/less-551.tar.gz
MD5 校验和：4ad4408b06d7a6626a055cb453f36819

· **Libcap (2.36) - 112 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-2.36.tar.xz
MD5 校验和：3d8cd4a87650cdee130691cb110c2ce2

· **Libffi (3.3) - 1,275 KB:**
主页：https://sourceware.org/libffi/
下载地址：ftp://sourceware.org/pub/libffi/libffi-3.3.tar.gz
MD5 校验和：6313289e32f1d38a9df4770b014a2ca7

· **Libpipeline (1.5.2) - 971 KB:**
主页：http://libpipeline.nongnu.org/
下载地址：http://download.savannah.gnu.org/releases/libpipeline/libpipeline-1.5.2.tar.gz
MD5 校验和：169de4cc1f6f7f7d430a5bed858b2fd3

· **Libtool (2.4.6) - 951 KB:**
主页：http://www.gnu.org/software/libtool/
下载地址：http://ftp.gnu.org/gnu/libtool/libtool-2.4.6.tar.xz
MD5 校验和：1bfb9b923f2c1339b4d2ce1807064aa5

· **Linux (5.7.2) - 110,047 KB:**
主页：https://www.kernel.org/
下载地址：https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.7.2.tar.xz
MD5 校验和：3aec12c426030b27553006ea515a91a1

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

· **M4 (1.4.18) - 1,180 KB:**
主页：http://www.gnu.org/software/m4/
下载地址：http://ftp.gnu.org/gnu/m4/m4-1.4.18.tar.xz
MD5 校验和：730bb15d96fffe47e148d1e09235af82

· **Make (4.3) - 2,263 KB:**
主页：http://www.gnu.org/software/make/
下载地址：http://ftp.gnu.org/gnu/make/make-4.3.tar.gz
MD5 校验和：fc7a67ea86ace13195b0bce683fd4469

· **Man-DB (2.9.2) - 1,844 KB:**
主页：https://www.nongnu.org/man-db/
下载地址：http://download.savannah.gnu.org/releases/man-db/man-db-2.9.2.tar.xz
MD5 校验和：86c7b99ce5969d9b20bf9aeae8d86e0b

· **Man-pages (5.07) - 1,677 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-5.07.tar.xz
MD5 校验和：4423c3e8d8e8224382ff244449c29468

· **Meson (0.54.3) - 1,652 KB:**
主页：https://mesonbuild.com
下载地址：https://github.com/mesonbuild/meson/releases/download/0.54.3/meson-0.54.3.tar.gz
MD5 校验和：4ec36e59f182ac6f890585e8baff6fe4

· **MPC (1.1.0) - 685 KB:**
主页：http://www.multiprecision.org/
下载地址：https://ftp.gnu.org/gnu/mpc/mpc-1.1.0.tar.gz
MD5 校验和：4125404e41e482ec68282a2e687f6c73

· **MPFR (4.0.2) - 1,409 KB:**
主页：https://www.mpfr.org/
下载地址：http://www.mpfr.org/mpfr-4.0.2/mpfr-4.0.2.tar.xz
MD5 校验和：320fbc4463d4c8cb1e566929d8adc4f8

· **Ncurses (6.2) - 3,346 KB:**
主页：http://www.gnu.org/software/ncurses/
下载地址：http://ftp.gnu.org/gnu/ncurses/ncurses-6.2.tar.gz
MD5 校验和：e812da327b1c2214ac1aed440ea3ae8d

· **Ninja (1.10.0) - 206 KB:**
主页：https://ninja-build.org/
下载地址：https://github.com/ninja-build/ninja/archive/v1.10.0/ninja-1.10.0.tar.gz
MD5 校验和：cf1d964113a171da42a8940e7607e71a

· **OpenSSL (1.1.1g) - 9,572 KB:**
主页：https://www.openssl.org/
下载地址：https://www.openssl.org/source/openssl-1.1.1g.tar.gz
MD5 校验和：76766e98997660138cdaf13a187bd234

· **Patch (2.7.6) - 766 KB:**
主页：https://savannah.gnu.org/projects/patch/
下载地址：http://ftp.gnu.org/gnu/patch/patch-2.7.6.tar.xz
MD5 校验和：78ad9937e4caadcba1526ef1853730d5

· **Perl (5.30.3) - 12,088 KB:**
主页：https://www.perl.org/
下载地址：https://www.cpan.org/src/5.0/perl-5.30.3.tar.xz
MD5 校验和：0af2ab0f01ec13e37cc13a27de930936

· **Pkg-config (0.29.2) - 1,970 KB:**
主页：https://www.freedesktop.org/wiki/Software/pkg-config
下载地址：https://pkg-config.freedesktop.org/releases/pkg-config-0.29.2.tar.gz
MD5 校验和：f6e931e319531b736fadc017f470e68a

· **Procps (3.3.16) - 840 KB:**
主页：https://sourceforge.net/projects/procps-ng
下载地址：https://sourceforge.net/projects/procps-ng/files/Production/procps-ng-3.3.16.tar.xz
MD5 校验和：e8dc8455e573bdc40b8381d572bbb89b

· **Psmisc (23.3) - 305 KB:**
主页：http://psmisc.sourceforge.net/
下载地址：https://sourceforge.net/projects/psmisc/files/psmisc/psmisc-23.3.tar.xz
MD5 校验和：573bf80e6b0de86e7f307e310098cf86

· **Python (3.8.3) - 17,494 KB:**
主页：https://www.python.org/
下载地址：https://www.python.org/ftp/python/3.8.3/Python-3.8.3.tar.xz
MD5 校验和：3000cf50aaa413052aef82fd2122ca78

· **Python 文档 (3.8.3) - 6,404 KB:**
下载地址：https://www.python.org/ftp/python/doc/3.8.3/python-3.8.3-docs-html.tar.bz2
MD5 校验和：2568df23eb5ad90aabab4b1e84b99fd9

· **Readline (8.0) - 2,907 KB:**
主页：https://tiswww.case.edu/php/chet/readline/rltop.html
下载地址：http://ftp.gnu.org/gnu/readline/readline-8.0.tar.gz
MD5 校验和：7e6c1f16aee3244a69aba6e438295ca3

· **Sed (4.8) - 1,317 KB:**
主页：http://www.gnu.org/software/sed/
下载地址：http://ftp.gnu.org/gnu/sed/sed-4.8.tar.xz
MD5 校验和：6d906edfdb3202304059233f51f9a71d

· **Shadow (4.8.1) - 1,574 KB:**
下载地址：https://github.com/shadow-maint/shadow/releases/download/4.8.1/shadow-4.8.1.tar.xz
MD5 校验和：4b05eff8a427cf50e615bda324b5bc45

· **Systemd (245) - 8,784 KB:**
主页：https://www.freedesktop.org/wiki/Software/systemd/
下载地址：https://github.com/systemd/systemd/archive/v245/systemd-245.tar.gz
MD5 校验和：04f02d9841ea5992a16f6b03c873da28

· **Systemd Man 页面 (245) - 512 KB:**
主页：https://www.freedesktop.org/wiki/Software/systemd/
下载地址：http://anduin.linuxfromscratch.org/LFS/systemd-man-pages-245.tar.xz
MD5 校验和：ecf8cc4baa33b91ad4212d28e88f8edd

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

· **Tar (1.32) - 2,055 KB:**
主页：http://www.gnu.org/software/tar/
下载地址：http://ftp.gnu.org/gnu/tar/tar-1.32.tar.xz
MD5 校验和：83e38700a80a26e30b2df054e69956e5

· **Tcl (8.6.10) - 9,907 KB:**
主页：http://tcl.sourceforge.net/
下载地址：https://downloads.sourceforge.net/tcl/tcl8.6.10-src.tar.gz
MD5 校验和：97c55573f8520bcab74e21bfd8d0aadc

· **Tcl 文档 (8.6.10) - 1,171 KB:**
下载地址：https://downloads.sourceforge.net/tcl/tcl8.6.10-html.tar.gz
MD5 校验和：a012711241ba3a5bd4a04e833001d489

· **Texinfo (6.7) - 4,237 KB:**
主页：http://www.gnu.org/software/texinfo/
下载地址：http://ftp.gnu.org/gnu/texinfo/texinfo-6.7.tar.xz
MD5 校验和：d4c5d8cc84438c5993ec5163a59522a6

· **Time Zone Data (2020a) - 388 KB:**
主页：https://www.iana.org/time-zones
下载地址：https://www.iana.org/time-zones/repository/releases/tzdata2020a.tar.gz
MD5 校验和：96a985bb8eeab535fb8aa2132296763a

· **Util-linux (2.35.2) - 5,030 KB:**
主页：http://freecode.com/projects/util-linux
下载地址：https://www.kernel.org/pub/linux/utils/util-linux/v2.35/util-linux-2.35.2.tar.xz
MD5 校验和：248a4d0810c9193e0e9a4bb3f26b93d8

· **Vim (8.2.0814) - 14,595 KB:**
主页：https://www.vim.org
下载地址：http://anduin.linuxfromscratch.org/LFS/vim-8.2.0814.tar.gz
MD5 校验和：`02b8b91bd2a9a97879fc60616f4eb767`

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

· **XML::Parser (2.46) - 249 KB:**
主页：https://github.com/chorny/XML-Parser
下载地址：https://cpan.metacpan.org/authors/id/T/TO/TODDR/XML-Parser-2.46.tar.gz
MD5 校验和：`80bb18a8e6240fcf7ec2f7b57601c170`

· **Xz Utils (5.2.5) - 1,122 KB:**
主页：https://tukaani.org/xz
下载地址：https://tukaani.org/xz/xz-5.2.5.tar.xz
MD5 校验和：`aa1621ec7013a19abab52a8aff04fe5b`

· **Zlib (1.2.11) - 457 KB:**
主页：https://www.zlib.net/
下载地址：https://zlib.net/zlib-1.2.11.tar.xz
MD5 校验和：`85adef240c5f370b308da8c938951a68`

· **Zstd (1.4.5) - 1,928 KB:**
主页：https://facebook.github.io/zstd/
下载地址：https://github.com/facebook/zstd/releases/download/v1.4.5/zstd-1.4.5.tar.gz
MD5 校验和：`dd0b53631303b8f972dafa6fd34beb0c`

以上软件包的总大小：约 416 MB

3.3. 必要的补丁

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

· **Bash 上游修复补丁 - 22 KB:**
下载地址：http://www.linuxfromscratch.org/patches/lfs/development/bash-5.0-upstream_fixes-1.patch
MD5 校验和：`c1545da2ad7d78574b52c465ec077ed9`

· **Binutils 对 gold 测试套件的修复补丁 - 5.0 KB:**
下载地址：http://www.linuxfromscratch.org/patches/lfs/development/binutils-2.34-gcc10_gold_test_fix-1.patch
MD5 校验和：`d18aaf9b25830cb8f7a5d44aa3febe28`

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

· **Coreutils 国际化修复补丁 - 166 KB:**
下载地址：http://www.linuxfromscratch.org/patches/lfs/development/coreutils-8.32-i18n-1.patch
MD5 校验和：cd8ebed2a67fff2e231026df91af6776

· **Glibc FHS 补丁 - 2.8 KB:**
下载地址：http://www.linuxfromscratch.org/patches/lfs/development/glibc-2.31-fhs-1.patch
MD5 校验和：9a5997c3452909b1769918c759eff8a2

· **GCC 交叉构建补丁 - 4.3 KB:**
下载地址：http://www.linuxfromscratch.org/patches/lfs/development/gcc-10.1.0-cet_fix-1.patch
MD5 校验和：f37d8a6b4c943a6f8eaf541923fed838

· **Kbd 退格/删除修复补丁 - 12 KB:**
下载地址：http://www.linuxfromscratch.org/patches/lfs/development/kbd-2.2.0-backspace-1.patch
MD5 校验和：f75cca16a38da6caa7d52151f7136895

· **Systemd GCC-10 补丁 - 8 KB:**
下载地址：http://www.linuxfromscratch.org/patches/lfs/development/systemd-245-gcc_10-fixes-2.patch
MD5 校验和：5eaac1d3a66118c40814d3b81ad36b71

以上补丁的总大小：约 221.7 KB

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

第 4 章 最后准备工作

4.1. 概述

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

4.2. 在 LFS 文件系统中创建最小目录布局

在 LFS 分区中需要进行的第一项任务是，创建一个最小目录树，使得在第 6 章中编译的程序可以被安装到它们的最终位置。这样，在第 8 章中重新构建它们时，就能直接覆盖这些临时程序。

以 root 身份，执行以下命令创建所需的目录布局：

```
mkdir -pv $LFS/{usr,lib,var,etc,bin,sbin}
case $(uname -m) in
  x86_64) mkdir -pv $LFS/lib64 ;;
esac
```

在 第 6 章 中，会使用交叉编译器编译程序 (细节参见工具链技术说明一节)。为了将这个交叉编译器和其他程序分离，它会被安装在一个专门的目录。执行以下命令创建该目录：

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

4.3. 添加 LFS 用户

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

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

命令行各选项的含义：

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

-g lfs
: 添加用户 lfs 到组 lfs。

-m
: 为用户 lfs 创建一个主目录。

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

lfs
: 要创建的用户的名称。

为了以 lfs 身份登录系统 (尽管以 root 身份登录时可以不用输入密码，直接切换到用户 lfs)，为 lfs 设置密码：

```
passwd lfs
```

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

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

如果您按照本书的建议，建立了一个单独的工作目录，那么将这个目录的所有者也设为 lfs：

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

注意

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

下面以 lfs 的身份登录。可以通过虚拟控制台或者显示管理器登录，也可以使用下面的命令切换用户：

```
su - lfs
```

参数 “-” 使得 **su** 启动一个登录 shell，而不是非登录 shell。您可以阅读 bash(1) 和 **info bash** 详细了解它们的区别。

4.4. 配置环境

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

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

在以 lfs 用户登录时，初始的 shell 一般是一个登录 shell。它读取宿主系统的 /etc/profile 文件 (可能包含一些设置和环境变量)，然后读取 .bash_profile。我们在 .bash_profile 中使用 **exec env -i.../bin/bash** 命令，新建一个除了 HOME, TERM 以及 PS1 外没有任何环境变量的 shell，替换当前 shell，防止宿主环境中不必要和有潜在风险的环境变量进入编译环境。通过使用以上技巧，我们创建了一个干净环境。

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

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

.bashrc 中设定的含义：

`set +h`
set +h 命令关闭 **bash** 的散列功能。一般情况下，散列是很有用的 —— **bash** 使用一个散列表维护各个可执行文件的完整路径，这样就不用每次都在 PATH 指定的目录中搜索可执行文件。然而，在构建 LFS 时，我们希望总是使用最新安装的工具。因此，需要关闭散列功能，使得 shell 在运行程序时总是搜索 PATH。这样，shell 总是能够找到 $LFS/tools 目录中那些最新编译的工具，而不是使用之前记忆的另一个目录中的程序。

`umask 022`
将用户的文件创建掩码 (umask) 设定为 022，保证只有文件所有者可以写新创建的文件和目录，但任何人都可读取、执行它们。(如果 open(2) 系统调用使用默认模式，则新文件将具有权限码 644，而新目录具有权限码 755)。

`LFS=/mnt/lfs`
LFS 环境变量必须被设定为之前选择的挂载点。

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

`LFS_TGT=(uname -m)-lfs-linux-gnu`
LFS_
TGT变量设定了一个非默认，但与宿主系统兼容的机器描述符。该描述符被用于构建交叉编译器和交叉编译临时工具链。工具链技术说明包含了关于这个描述符的更多信息。

`PATH=/usr/bin`
许多现代 Linux 发行版合并了 /bin 和 /usr/bin。在这种情况下，标准 PATH 变量只需要被设定为 /usr/bin，即可满足第 6 章的环境。否则，后续命令将会增加 /bin 到搜索路径中。

`if [ ! -L /bin ]; then PATH=/bin:$PATH; fi`
如果 /bin 不是符号链接，则它需要被添加到 PATH 变量中。

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

`export LFS LC_ALL LFS_TGT PATH`
前面的命令设定了一些变量，为了让所有子 shell 都能使用这些变量，需要导出它们。

重要

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

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

在第 7 章一章开始后，就不再使用 lfs 用户，您 (如果希望的话) 可以复原 /etc/bash.bashrc 文件。

注意我们将会在第 8.34 节 “Bash-5.0”中构建的 LFS Bash 软件包未被配置为读取或执行 /etc/bash.bashrc，因此它在完整的 LFS 系统中没有作用。

最后，为了完全准备好编译临时工具的环境，指示 shell 读取刚才创建的配置文件：

```
source ~/.bash_profile
```

4.5. 关于 SBU

许多人想在编译和安装各个软件包之前，了解这一过程大概需要多少时间。由于 Linux From Scratch 可以在许多不同系统上构建，我们无法直接给出估计时间。例如，最大的软件包 (Glibc) 在最快的系统上只要大约 20 分钟就能构建好，而在一些较慢的系统上需要 3 天！因此，我们不提供实际时间，而是以标准构建单位 (SBU) 衡量时间。

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

例如，考虑一个编译时间是 4.5 SBU 的软件包。如果在某个系统上，需要 10 分钟来编译和安装第一轮的 Binutils，那么大概需要 45 分钟才能构建这个软件包。幸运的是，多数软件包构建时间比 Binutils 少。

一般来说，SBU 不是完全准确的。这是由于它受到许多因素的影响，包括宿主系统的 GCC 版本。SBU 只能用来估计安装一个软件包可能需要的时间，估计结果的误差在个别情况下可能达到几十分钟。

注意

对于许多拥有多个处理器 (或处理器核心) 的现代系统，可以显著缩短软件包的编译时间，设置环境变量或者直接告诉 **make** 命令有多少个可用的处理器，即可进行并行构建。例如，一块 Intel i5-6500 CPU 可以支持 4 个同时运行的进程，可以设定：

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

或者直接用以下命令构建：

```
make -j4
```

用这种方式使用多个处理器时，SBU 值将会发生变化，有时甚至变得比正常值还大。某些情况下，还会导致 make 命令失败。另外，分析构建过程的的输出也会变得困难，因为不同进程的输出行会交错在一起。如果在构建过程中出现问题，需要使用单处理器进行构建，才能更好地分析错误消息。

4.6. 关于测试套件

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

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

注意

在第 5 章和第 6 章中不可能运行测试套件，因为这些程序是使用交叉编译器编译的，根本不该在构建它们的宿主系统运行。

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

一些软件包的测试套件总是失败，但开发者已经知道失败原因，并判定这些失败并不重要。参照 http://www.linuxfromscratch.org/lfs/build-logs/development/ 中的构建日志，来检查这些失败是否符合预期。本书中的所有测试都可以在该网址查询。

第 III 部分 构建 LFS 交叉工具链和临时工具

重要的提前阅读资料

概述

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

重要

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

下一节将给出构建过程的技术说明，再下一节包含**非常重要**的通用说明。

工具链技术说明

本节综合地解释构建方法中的逻辑和技术细节。您现在并不需要立刻理解本节的所有内容，在实际进行构建的过程中，可以更清晰地理解本节的信息。在整个构建过程中，您随时可以回来翻阅本节。

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

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

关于交叉编译

交叉编译涉及一些概念，值得专门用一节讨论。尽管您可以在初次阅读时跳过本节，但强烈建议您之后回头阅读本节，以完全掌握构建过程。

首先我们定义讨论交叉编译时常用的术语：

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

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

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

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

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

这样，我们可以为机器 C 使用 cc2 在快速的机器 B 上构建所有其他程序。注意除非 B 能运行为 C 编译的程序，我们无法测试编译得到的程序，直到在 C 上运行它。例如，如果要测试 ccC，我们可以增加第四个阶段：

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

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

LFS 的交叉编译实现

注意

几乎所有构建系统都使用形如 CPU-供应商-内核-操作系统，称为三元组的名称表示目标机器。好奇的读者可能感到奇怪，为什么一个“三元组”却包含四个部分。这是历史遗留的：最早，三个部分就足以无歧义地描述一台机器。但是随着新的机器和系统不断出现，最终证明三个部分是不够的。然而，“三元组”这个术语保留了下来。有一种简单方法可以获得您的机器的三元组，即运行许多软件包附带的 **config.guess** 脚本。解压缩 Binutils 源码，然后运行脚本：**./config.guess**，观察输出。例如，对于 32 位 Intel 处理器，输出应该是 i686-pc-linux-gnu，而对于 64 位系统输出应该是 x86_64-pc-linux-gnu。

另外注意平台的动态链接器的名称，它又被称为动态加载器 (不要和 Binutils 中的普通链接器 **ld** 混淆)。动态链接器由 Glibc 提供，它寻找并加载程序所需的共享库，为程序运行做好准备，然后运行程序。在 32 位 Intel 机器上动态链接器的名称是 `ld-linux.so.2` (在 64 位系统上是 `ld-linux-x86-64.so.2`)。一个确定动态链接器名称的准确方法是从宿主系统找一个二进制可执行文件，然后执行：**readelf -l <二进制文件名> | grep interpreter** 并观察输出。包含所有平台的权威参考可以在 Glibc 源码树根目录的 `shlib-versions` 文件中找到。

为了将本机伪装成交叉编译目标机器，我们在 `LFS_TGT` 变量中，对宿主系统三元组的 "vendor" 域进行修改。我们还会在构建交叉链接器和交叉编译器时使用 `--with-sysroot` 选项，指定查找所需的 host 系统文件的位置。这保证在第 6 章中的其他程序在构建时不会链接到宿主 (build) 系统的库。前两个阶段是必要的，第三个阶段可以用于测试：

阶段	Build	Host	Target	操作描述
1	pc	pc	lfs	在 pc 上使用 cc-pc 构建交叉编译器 cc1
2	pc	lfs	lfs	在 pc 上使用 cc1 构建 cc-lfs
3	lfs	lfs	lfs	在 lfs 上使用 cc-lfs 重新构建并测试它本身

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

现在，关于交叉编译，还有更多要处理的问题：C 语言并不仅仅由一个编译器实现，它还规定了一个标准库。在本书中，我们使用 GNU C 运行库，即 glibc。它必须为 lfs 目标机器使用交叉编译器 cc1 编译。但是，编译器本身使用一个库，实现汇编指令集并不支持的一些复杂指令。这个内部库称为 libgcc，它必须链接到 glibc 库才能实现完整功能！另外，C++ 标准库 (libstdc++) 也必须链接到 glibc。为了解决这个”先

有鸡还是先有蛋“的问题，只能先构建一个降级的 cc1，它的 libgcc 缺失线程和异常等功能，再用这个降级的编译器构建 glibc (这不会导致 glibc 缺失功能)，再构建 libstdc++。但是这种方法构建的 libstdc++ 和 libgcc 一样，会缺失一些功能。

讨论还没有结束：上面一段的结论是 cc1 无法构建功能完整的 libstdc++，但这是我们在阶段 2 构建 C/C++ 库时唯一可用的编译器！当然，在阶段 2 中构建的编译器 cc-lfs 将会可以构建这些库，但是 (1) GCC 构建系统不知道这个编译器在 pc 上可以使用，而且 (2) 它是一个本地编译器，因此在 pc 上使用它可能产生链接到 pc (宿主系统) 库的风险。因此我们必须在进入 chroot 后再次构建 libstdc++。

构建过程的其他细节

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

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

Binutils 将汇编器和链接器安装在两个位置，一个是 $LFS/tools/bin，另一个是 $LFS/tools/$LFS_TGT/bin。这两个位置中的工具互为硬链接。链接器的一项重要属性是它搜索库的顺序，通过向 **ld** 命令加入 --verbose 参数，可以得到关于搜索路径的详细信息。例如，**ld --verbose | grep SEARCH** 会输出当前的搜索路径及其顺序。此外，通过编译一个样品 (dummy) 程序并向链接器 **ld** 传递 --verbose 参数，可以知道哪些文件被链接。例如，**gcc dummy.c -Wl,--verbose 2>&1 | grep succeeded** 将显示所有在链接过程中被成功打开的文件。

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

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

基于我们上面论述的原因，这些输出非常重要。这说明 GCC 的配置脚本没有在 PATH 变量指定的目录中搜索工具。然而，在 **gcc** 的实际运行中，未必会使用同样的搜索路径。为了查询 **gcc** 会使用哪个链接器，需要执行以下命令：**$LFS_TGT-gcc -print-prog-name=ld**。

通过向 **gcc** 传递 -v 参数，可以知道在编译样品程序时发生的细节。例如，**gcc -v dummy.c** 会输出预处理、编译和汇编阶段中的详细信息，包括 **gcc** 的包含文件搜索路径和顺序。

下一步安装“净化的” (sanitized) Linux API 头文件。这允许 C 标准库 (Glibc) 与 Linux 内核提供的各种特性交互。

下一步安装 Glibc。在构建 Glibc 时需要着重考虑是编译器，二进制工具，以及内核头文件。编译器一般不成问题，Glibc 总是使用传递给配置脚本的 --host 参数相关的编译器。例如，在我们的例子中，使用的编译器是 **$LFS_TGT-gcc**。但二进制工具和内核头文件的问题比较复杂。安全起见，我们使用配置脚本提供的开关，保证正确的选择。在 **configure** 脚本运行完成后，可以检查 build 目录中的 config.make 文件，了解全部重要的细节。注意参数 CC="$LFS_TGT-gcc" (其中 $LFS_TGT 会被展开) 控制构建系统使用正确的二进制工具，而参数 -nostdinc 和 -isystem 控制编译器的包含文件搜索路径。这些事项凸显了 Glibc 软件包的一个重要性质 —— 它的构建机制是相当自给自足的，通常不依赖于工具链默认值。

正如前文所述，接下来构建 C++ 标准库，然后是第 6 章中那些需要自身才能构建的程序后。在安装这些软件包时使用 DESTDIR 变量，将它安装到 LFS 文件系统中。

在第 6 章一节的末尾，构建 lfs 本地编译器。首先使用和其他程序相同的 DESTDIR 第二次构建 binutils，然后第二次构建 GCC，构建时忽略 libstdc++ 和其他不重要的库。由于 GCC 配置脚本的一些奇怪逻辑，CC_FOR_TARGET 变量在 host 系统和 target 相同，但与 build 不同时，被设定为 **cc**。因此我们必须显式地在配置选项中指定 CC_FOR_TARGET=$LFS_TGT-gcc。

在第 7 章中，进入 chroot 环境后，首先安装 libstdc++。之后临时性地安装工具链的正常工作所必须的程序。还要构建测试其他程序时必须的程序。此后，核心工具链成为自包含的本地工具链。在第 8 章中，构建、测试并最后一次安装所有软件包，它们组成功能完整的系统。

编译过程的一般说明

在构建软件包时，本书提供的命令基于下列假设：

- 某些软件包在编译前需要打补丁，然而补丁只在绕过特定问题时才需要。补丁常常在本章和下一章都要使用，然而有时只在其中一章使用。因此，如果发现本书给出的步骤中没有使用某个下载好的补丁，这是正常的，不必担心。在应用补丁时可能会出现关于 offset 或者 fuzz 的警告信息。不用担心这些警告，补丁还是会成功应用到源码上的。
- 在编译大多数软件包时，屏幕上都会出现一些警告。这是正常的，可以放心地忽略。这些警告就像它们描述的那样，是关于一些过时的，但并不是错误的 C 或 C++ 语法。C 标准经常改变，一些软件包仍然在使用旧的标准。这并不是一个严重的问题，但确实会触发警告。
- 最后确认 LFS 环境变量是否配置正确：

```
echo $LFS
```

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

- 最后强调两个重要事项：

重要

本书中的命令假设宿主系统需求中的所有内容，包括符号链接，都被正确设置：

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

重要

再次强调构建过程：

1. 把所有的源码包和补丁放在一个能够从 chroot 环境访问的目录，例如 /mnt/lfs/sources/。
2. 切换到放着源码包的目录。
3. 对于每个软件包：
   a. 使用 **tar** 程序，解压需要构建的软件包。在第 5 章和第 6 章中解压软件包时，确认您以用户 lfs 的身份登录。
   b. 切换到解压源码包时产生的目录。
   c. 根据书中的指示构建软件包。
   d. 切换回包含所有源码包的目录。
   e. 除非另有说明，删除解压出来的目录。

第 5 章 编译交叉工具链

5.1. 概述

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

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

5.2. Binutils-2.34 - 第一遍

Binutils 包含汇编器、链接器以及其他用于处理目标文件的工具。

估计构建时间: 1 SBU
需要硬盘空间: 611 MB

5.2.1. 安装交叉工具链中的 Binutils

注意

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

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

Binutils 文档推荐在一个专用的目录中构建 Binutils：

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

注意

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

现在，准备编译 Binutils：

```
../configure --prefix=$LFS/tools       \
             --with-sysroot=$LFS        \
             --target=$LFS_TGT          \
             --disable-nls              \
             --disable-werror
```

配置选项的含义：

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

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

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

`--disable-nls`
该选项禁用临时工具不需要的国际化功能。

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

然后编译该软件包：

```
make
```

安装该软件包：

```
make install
```

这个软件包的细节在第 8.18.2 节 “Binutils 的内容”中可以找到。

5.3. GCC-10.1.0 - 第一遍

GCC 软件包包含 GNU 编译器集合，其中有 C 和 C++ 编译器。

估计构建时间: 11 SBU
需要硬盘空间: 3.7 GB

5.3.1. 安装交叉工具链中的 GCC

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

注意

对于本章内容有一些很常见的误解。该软件包的构建过程就像之前 (软件包构建说明) 解释的那样，首先解压 GCC 压缩包，切换到解压出的目录中，再执行下面的命令。

```
tar -xf ../mpfr-4.0.2.tar.xz
mv -v mpfr-4.0.2 mpfr
tar -xf ../gmp-6.2.0.tar.xz
mv -v gmp-6.2.0 gmp
tar -xf ../mpc-1.1.0.tar.gz
mv -v mpc-1.1.0 mpc
```

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

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

GCC 文档建议在一个专用目录中构建 GCC:

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

准备编译 GCC:

```
../configure                                       \
    --target=$LFS_TGT                              \
    --prefix=$LFS/tools                            \
    --with-glibc-version=2.11                      \
    --with-sysroot=$LFS                            \
    --with-newlib                                  \
    --without-headers                              \
    --enable-initfini-array                        \
    --disable-nls                                  \
    --disable-shared                               \
    --disable-multilib                             \
    --disable-decimal-float                        \
    --disable-threads                              \
    --disable-libatomic                            \
    --disable-libgomp                              \
    --disable-libquadmath                          \
    --disable-libssp                               \
    --disable-libvtv                               \
    --disable-libstdcxx                            \
    --enable-languages=c,c++
```

配置选项的含义：

`--with-glibc-version=2.11`
该选项确保该软件包与宿主的 glibc 版本兼容。它被设定为宿主系统需求中要求的最小 glibc 版本一致。

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

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

`--enable-initfini-array`
这个开关强制启用一些内部数据结构，它们是必要的，但是在构建交叉编译器时，无法被检测到。

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

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

`--disable-decimal-float`, `--disable-threads`, `--disable-libatomic`, `--disable-libgomp`, `--disable-libquadmath`, `--disable-libssp`, `--disable-libvtv`, `--disable-libstdcxx`
这些开关禁用对于十进制浮点数、线程、libatomic、libgomp、 libquadmath、libssp、libvtv 和 C++ 标准库的支持。在构建交叉编译器时它们的编译会失败，而且在交叉编译临时 libc 时并不需要它们。

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

执行以下命令编译 GCC:

```
make
```

安装该软件包:

```
make install
```

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

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

该软件包的更多细节在第 8.26.2 节 “GCC 的内容”可以找到。

5.4. Linux-5.7.2 API 头文件

Linux API 头文件 (在 linux-5.7.2.tar.xz 中) 导出内核 API 供 Glibc 使用。

估计构建时间: 0.1 SBU
需要硬盘空间: 1 GB

5.4.1. 安装 Linux API 头文件

Linux 内核需要导出一个应用程序编程接口 (API) 供系统的 C 运行库 (例如 LFS 中的 Glibc) 使用。这通过净化内核源码包中提供的若干 C 头文件完成。

确保软件包中没有遗留陈旧的文件:

```
make mrproper
```

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

```
make headers
find usr/include -name '.*' -delete
rm usr/include/Makefile
cp -rv usr/include $LFS/usr
```

5.4.2. Linux API 头文件的内容

安装的头文件: /usr/include/asm/*.h, /usr/include/asm-generic/*.h, /usr/include/drm/*.h, /usr/include/linux/*.h, /usr/include/misc/*.h, /usr/include/mtd/*.h, /usr/include/rdma/*.h, /usr/include/scsi/*.h, /usr/include/sound/*.h, /usr/include/video/*.h, 以及 /usr/include/xen/*.h
安装的目录: /usr/include/asm, /usr/include/asm-generic, /usr/include/drm, /usr/include/linux, /usr/include/misc, /usr/include/mtd, /usr/include/rdma, /usr/include/scsi, /usr/include/sound, /usr/include/video, 以及 /usr/include/xen

简要描述

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

/usr/include/xen/*.h	Linux API Xen 头文件

5.5. Glibc-2.31

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

估计构建时间: 4.2 SBU
需要硬盘空间: 750 MB

5.5.1. 安装 Glibc

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

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

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

```
patch -Np1 -i ../glibc-2.31-fhs-1.patch
```

Glibc 文档推荐在一个专用目录中构建 Glibc:

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

下面，准备编译 Glibc:

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

配置选项的含义:

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

`--enable-kernel=3.2`
该选项告诉 Glibc 编译出支持 3.2 版或者更新的 Linux 内核，这样就不会使用那些为更老内核准备的替代方案。

`--with-headers=$LFS/usr/include`
该选项告诉 Glibc 在编译过程中，使用 $LFS/usr/include 目录中的头文件，这样它就知道内核拥有哪些特性，并据此对自身进行优化。

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

在当前阶段，可能出现下列警告：

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

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

注意

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

编译该软件包：

```
make
```

安装该软件包：

警告

如果 LFS 没有正确设定，而且您不顾本书的建议以 root 用户身份进行构建，下面的命令会将新构建的 glibc 安装到您的宿主系统中，这很可能导致宿主系统完全无法使用。因此，请再次检查该环境变量是否已经为 lfs 用户设定好。

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

make install 选项的含义：

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

小心

现在我们不可避免地要停下确认新工具链的各基本功能 (编译和链接) 能如我们所预期的那样工作。执行以下命令进行完整性检查:

```
echo 'int main(){}' > dummy.c
$LFS_TGT-gcc dummy.c
readelf -l a.out | grep '/ld-linux'
```

如果一切正常，那么应该没有错误消息，而且最后一行命令应该输出下列格式的内容:

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

注意，对于 32 位机器，解释器的名字将会是 /lib/ld-linux.so.2。

如果输出不像上面描述的那样，或者根本没有输出，就说明出了问题。检查并重新跟踪各个步骤，找到出问题的地方并修正它。在继续构建之前，必须解决这个问题。

检验步骤顺利完成后，清理测试文件:

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

注意

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

现在我们的交叉工具链已经构建完成，可以完成 limits.h 头文件的安装。为此，运行 GCC 开发者提供的一个工具:

```
$LFS/tools/libexec/gcc/$LFS_TGT/10.1.0/install-tools/mkheaders
```

该软件包的详细信息可以在第 8.8.3 节 “Glibc 的内容”中找到。

5.6. GCC-10.1.0 中的 Libstdc++，第一遍

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

估计构建时间: 0.4 SBU
需要硬盘空间: 952 MB

5.6.1. 安装目标系统的 Libstdc++

注意

Libstdc++ 是 GCC 源代码的一部分。您应该先解压 GCC 源码包并切换到解压出来的 gcc-10.1.0 目录。

为 Libstdc++ 创建一个单独的构建目录，并进入它:

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

准备编译 Libstdc++:

```
../libstdc++-v3/configure           \
    --host=$LFS_TGT                 \
    --build=$(../config.guess)      \
    --prefix=/usr                   \
    --disable-multilib              \
    --disable-nls                   \
    --disable-libstdcxx-pch         \
    --with-gxx-include-dir=/tools/$LFS_TGT/include/c++/10.1.0
```

配置选项的含义:

`--host=...`
指定使用我们刚刚构建的交叉编译器，而不是 /usr/bin 中的宿主系统编译器。

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

`--with-gxx-include-dir=/tools/$LFS_TGT/include/c++/10.1.0`
C++ 编译器应该在这个位置搜索标准头文件。在正常的构建过程中，这项信息被顶层目录构建系统自动传递给 Libstdc++ **configure** 脚本。然而我们没有使用顶层构建系统，必须明确给出这项信息。

运行以下命令编译 Libstdc++:

```
make
```

安装这个库:

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

关于该软件包的详细信息可以在第 8.26.2 节 “GCC 的内容”中找到。

第 6 章 交叉编译临时工具

6.1. 概述

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

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

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

6.2. M4-1.4.18

M4 软件包包含一个宏处理器。

估计构建时间: 0.1 SBU
需要硬盘空间: 22 MB

6.2.1. 安装 M4

首先，进行 glibc-2.28 要求的一些修补：

```
sed -i 's/IO_ftrylockfile/IO_EOF_SEEN/' lib/*.c
echo "#define _IO_IN_BACKUP 0x100" >> lib/stdio-impl.h
```

准备编译 M4：

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

编译该软件包：

```
make
```

安装该软件包：

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

该软件包的详细信息可以在第 8.15.2 节 “M4 的内容”找到。

6.3. Ncurses-6.2

Ncurses 软件包包含终端无关的字符屏幕处理库。

估计构建时间: 0.7 SBU
需要硬盘空间: 48 MB

6.3.1. 安装 Ncurses

首先，保证在配置时优先查找 **gawk** 命令:

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

然后，运行以下命令，在宿主系统构建“tic”程序:

```
mkdir build
pushd build
  ../configure
  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-debug              \
            --without-ada                \
            --without-normal             \
            --enable-widec
```

新出现的配置选项的含义:

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

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

`--enable-widec`
该选项使得宽字符库 (例如 `libncursesw.so.6.2`) 被构建，而不构建常规字符库 (例如 `libncurses.so.6.2`)。宽字符库在多字节和传统 8 位 locale 中都能工作，而常规字符库只能在 8 位 locale 中工作。宽字符库和普通字符库在源码层面是兼容的，但二进制不兼容。

`--without-normal`
该选项禁止多数静态库的构建和安装。

编译该软件包：

```
make
```

安装该软件包：

```
make DESTDIR=$LFS TIC_PATH=$(pwd)/build/progs/tic install
ln -s libncursesw.so $LFS/usr/lib/libncurses.so
```

将共享库移动到它们应该位于的 /lib 目录中：

```
mv -v $LFS/usr/lib/libncursesw.so.6* $LFS/lib
```

由于库文件被移动到其他位置，一个符号链接现在指向不存在的文件。重新生成它：

```
ln -sfv ../../lib/$(readlink $LFS/usr/lib/libncursesw.so) $LFS/usr/lib/libncursesw.so
```

该软件包的详细信息可以在第 8.28.2 节 “Ncurses 的内容”中找到。

6.4. Bash-5.0

Bash 软件包包含 Bourne-Again SHell。

估计构建时间: 0.4 SBU
需要硬盘空间: 64 MB

6.4.1. 安装 Bash

准备编译 Bash:

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

配置选项的含义:

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

编译该软件包:

```
make
```

安装该软件包:

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

将可执行文件移动到正确位置:

```
mv $LFS/usr/bin/bash $LFS/bin/bash
```

为那些使用 **sh** 命令运行 shell 的程序考虑，创建一个链接:

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

该软件包的详细信息可以在第 8.34.2 节 “Bash 的内容”中找到。

6.5. Coreutils-8.32

Coreutils 软件包包含用于显示和设定系统基本属性的工具。

估计构建时间: 0.6 SBU
需要硬盘空间: 170 MB

6.5.1. 安装 Coreutils

准备编译 Coreutils:

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

配置选项的含义:

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

编译该软件包:

```
make
```

安装该软件包:

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

将程序移动到它们最终安装时的正确位置。尽管在临时环境中这不必要，但我们必须这样做，因为一些程序会硬编码它们的位置:

```
mv -v $LFS/usr/bin/{cat,chgrp,chmod,chown,cp,date,dd,df,echo} $LFS/bin
mv -v $LFS/usr/bin/{false,ln,ls,mkdir,mknod,mv,pwd,rm}        $LFS/bin
mv -v $LFS/usr/bin/{rmdir,stty,sync,true,uname}               $LFS/bin
mv -v $LFS/usr/bin/{head,nice,sleep,touch}                    $LFS/bin
mv -v $LFS/usr/bin/chroot                                     $LFS/usr/sbin
mkdir -pv $LFS/usr/share/man/man8
mv -v $LFS/usr/share/man/man1/chroot.1                        $LFS/usr/share/man/man8/chroot.8
sed -i 's/"1"/"8"/'                                           $LFS/usr/share/man/man8/chroot.8
```

该软件包的详细信息可以在第 8.52.2 节 “Coreutils 的内容”中找到。

6.6. Diffutils-3.7

Diffutils 软件包包含显示文件或目录之间差异的程序。

估计构建时间: 0.2 SBU
需要硬盘空间: 26 MB

6.6.1. 安装 Diffutils

准备编译 Diffutils:

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

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.54.2 节 “Diffutils 的内容”中找到。

6.7. File-5.39

File 软件包包含用于确定给定文件类型的工具。

估计构建时间: 0.1 SBU
需要硬盘空间: 20 MB

6.7.1. 安装 File

准备编译 File:

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

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.13.2 节 “File 的内容”中找到。

6.8. Findutils-4.7.0

Findutils 软件包包含用于查找文件的程序。这些程序能够递归地搜索目录树，以及创建、维护和搜索文件数据库 (一般比递归搜索快，但在数据库最近没有更新时不可靠)。

估计构建时间: 0.2 SBU
需要硬盘空间: 40 MB

6.8.1. 安装 Findutils

准备编译 Findutils:

```
./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
```

将可执行文件移动到最终安装时的正确位置:

```
mv -v $LFS/usr/bin/find $LFS/bin
sed -i 's|find:=${BINDIR}|find:=/bin|' $LFS/usr/bin/updatedb
```

该软件包的详细信息可以在第 8.56.2 节 “Findutils 的内容”中找到。

6.9. Gawk-5.1.0

Gawk 软件包包含操作文本文件的程序。

估计构建时间: 0.2 SBU
需要硬盘空间: 46 MB

6.9.1. 安装 Gawk

首先，确保不要安装一些没有必要的文件：

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

准备编译 Gawk：

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

编译该软件包：

```
make
```

安装该软件包：

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

该软件包的详细信息可以在第 8.55.2 节 “Gawk 的内容”中找到。

6.10. Grep-3.4

Grep 软件包包含在文件内容中进行搜索的程序。

估计构建时间: 0.2 SBU
需要硬盘空间: 26 MB

6.10.1. 安装 Grep

准备编译 Grep:

```
./configure --prefix=/usr   \
            --host=$LFS_TGT \
            --bindir=/bin
```

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.33.2 节 “Grep 的内容”中找到。

6.11. Gzip-1.10

Gzip 软件包包含压缩和解压缩文件的程序。

估计构建时间: 0.1 SBU
需要硬盘空间: 10 MB

6.11.1. 安装 Gzip

准备编译 Gzip:

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

编译该软件包:

```
make
```

安装该软件包:

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

将可执行文件移动到最终安装时的正确位置:

```
mv -v $LFS/usr/bin/gzip $LFS/bin
```

该软件包的详细信息可以在第 8.60.2 节 “Gzip 的内容”中找到。

6.12. Make-4.3

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

估计构建时间: 0.1 SBU
需要硬盘空间: 16 MB

6.12.1. 安装 Make

准备编译 Make:

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

新出现的配置选项的含义:

`--without-guile`
尽管我们在进行交叉编译，配置脚本如果找到宿主系统的 guile，仍然会试图使用它。这导致编译失败，因此使用该选项防止使用 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可以在第 8.64.2 节 “Make 的内容”中找到。

6.13. Patch-2.7.6

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

估计构建时间: 0.1 SBU
需要硬盘空间: 13 MB

6.13.1. 安装 Patch

准备编译 Patch：

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

编译该软件包：

```
make
```

安装该软件包：

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

该软件包的详细信息可以在第 8.65.2 节 “Patch 的内容”中找到。

6.14. Sed-4.8

Sed 软件包包含一个流编辑器。

估计构建时间: 0.1 SBU
需要硬盘空间: 21 MB

6.14.1. 安装 Sed

准备编译 Sed:

```
./configure --prefix=/usr   \
            --host=$LFS_TGT \
            --bindir=/bin
```

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.29.2 节 “Sed 的内容”中找到。

6.15. Tar-1.32

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

估计构建时间: 0.2 SBU
需要硬盘空间: 39 MB

6.15.1. 安装 Tar

准备编译 Tar:

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

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的详细信息可以在第 8.67.2 节 “Tar 的内容”中找到。

6.16. Xz-5.2.5

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

估计构建时间: 0.1 SBU
需要硬盘空间: 16 MB

6.16.1. 安装 Xz

准备编译 Xz:

```
./configure --prefix=/usr                     \
            --host=$LFS_TGT                   \
            --build=$(build-aux/config.guess) \
            --disable-static                  \
            --docdir=/usr/share/doc/xz-5.2.5
```

编译该软件包:

```
make
```

安装该软件包:

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

确保所有关键文件位于正确的目录中:

```
mv -v $LFS/usr/bin/{lzma,unlzma,lzcat,xz,unxz,xzcat}  $LFS/bin
mv -v $LFS/usr/lib/liblzma.so.*                       $LFS/lib
ln -svf ../../lib/$(readlink $LFS/usr/lib/liblzma.so) $LFS/usr/lib/liblzma.so
```

该软件包的详细信息可以在第 8.11.2 节 “Xz 的内容”中找到。

6.17. Binutils-2.34 - 第二遍

Binutils 包含汇编器、链接器以及其他用于处理目标文件的工具。

估计构建时间: 1.2 SBU
需要硬盘空间: 492 MB

6.17.1. 安装 Binutils

再一次地，创建一个单独的构建目录:

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

准备编译 Binutils:

```
../configure                   \
    --prefix=/usr              \
    --build=$(../config.guess) \
    --host=$LFS_TGT            \
    --disable-nls              \
    --enable-shared            \
    --disable-werror           \
    --enable-64-bit-bfd
```

配置选项的含义:

`--enable-shared`
将 `libbfd` 构建为共享库。

`--enable-64-bit-bfd`
启用 64 位支持 (在那些字长较短的平台上)。在 64 位系统上可能并不需要，但无害。

编译该软件包:

```
make
```

安装该软件包:

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

该软件包的更多信息可以在第 8.18.2 节 “Binutils 的内容”中找到。

6.18. GCC-10.1.0 - 第二遍

GCC 软件包包含 GNU 编译器集合，其中有 C 和 C++ 编译器。

估计构建时间: 11 SBU
需要硬盘空间: 3.6 GB

6.18.1. 安装 GCC

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

```
tar -xf ../mpfr-4.0.2.tar.xz
mv -v mpfr-4.0.2 mpfr
tar -xf ../gmp-6.2.0.tar.xz
mv -v gmp-6.2.0 gmp
tar -xf ../mpc-1.1.0.tar.gz
mv -v mpc-1.1.0 mpc
```

如果是在 x86_64 上构建，修改 64 位库文件的默认目录名为 “lib”:

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

修复 GCC-10.1 在使用交叉编译器构建时出现的问题:

```
patch -Np1 -i ../gcc-10.1.0-cet_fix-1.patch
```

再次创建一个独立的构建目录:

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

创建一个符号链接，以允许 libgcc 在构建时启用 POSIX 线程支持:

```
mkdir -pv $LFS_TGT/libgcc
ln -s ../../../libgcc/gthr-posix.h $LFS_TGT/libgcc/gthr-default.h
```

在开始构建 GCC 前，记得清除所有覆盖默认优化开关的环境变量。

现在准备编译 GCC:

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

配置选项的含义:

-with-build-sysroot=$LFS
通常，使用 --host 即可保证使用交叉编译器构建 GCC，这个交叉编译器知道它应该在 $LFS 中查找头文件和库。但是，GCC 构建系统使用其他一些工具，它们不知道这个位置。因此需要该选项告诉它们在 $LFS 中查找需要的文件，而不是在宿主系统中查找。

--enable-initfini-array
该选项在使用 x86 本地编译器构建另一个本地编译器时自动启用。然而我们使用交叉编译器进行编译，因此必须显式启用它。

编译该软件包:

```
make
```

安装该软件包:

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

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

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

关于本软件包的更多信息可以在第 8.26.2 节 “GCC 的内容”中找到。

第 7 章 进入 Chroot 并构建其他临时工具

7.1. 概述

本章展示如何构建临时系统最后缺失的部分：首先，安装一些软件包的构建机制所必须的工具，然后安装三个用于运行测试的软件包。这样，就解决了所有的循环依赖问题，我们可以使用“chroot”环境进行构建，它与宿主系统除正在运行的内核外完全隔离。

为了隔离环境的正常工作，必须它与正在运行的内核之间建立一些通信机制。我们通过所谓的虚拟内核文件系统达成这一目的，它们必须在进入 chroot 环境时挂载。您可能希望用 **findmnt** 命令检查它们是否挂载好。

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

7.2. 改变所有者

注意

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

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

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

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

7.3. 准备虚拟内核文件系统

内核对外提供了一些文件系统，以便自己和用户空间进行通信。它们是虚拟文件系统，并不占用磁盘空间，其内容保留在内存中。

首先创建这些文件系统的挂载点：

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

7.3.1. 创建初始设备节点

在内核引导系统时，它需要一些设备节点，特别是 console 和 null 两个设备。它们需要创建在硬盘上，这样在内核填充 /dev 前，或者 Linux 使用 init=/bin/bash 内核选项启动时，也能使用它们。运行以下命令创建它们：

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

7.3.2. 挂载和填充 /dev

用设备文件填充 /dev 目录的推荐方法是挂载一个虚拟文件系统 (例如 tmpfs) 到 /dev，然后在设备被发现或访问时动态地创建设备文件。这个工作通常由 Udev 在系统引导时完成。然而，我们的新系统还没有 Udev，也没有被引导过，因此必须手工挂载和填充 /dev。这可以通过绑定挂载宿主系统的 /dev 目录就实现。绑定挂载是一种特殊挂载类型，它允许在另外的位置创建某个目录或挂载点的映像。运行以下命令进行绑定挂载：

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

7.3.3. 挂载虚拟内核文件系统

现在挂载其余的虚拟内核文件系统：

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

在某些宿主系统上，/dev/shm 是一个指向 /run/shm 的符号链接。我们已经在 /run 下挂载了 tmpfs 文件系统，因此在这里只需要创建一个目录。

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

7.4. 进入 Chroot 环境

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

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

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

从现在开始，就不再需要使用 LFS 环境变量，因为所有工作都被局限在 LFS 文件系统内。这是由于 Bash 被告知 $LFS 现在是根目录 (/)。

注意 /tools/bin 不在 PATH 中。这意味着交叉工具链在 chroot 环境中不被再使用。这还需要保证 shell 不“记忆”执行过的程序的位置 —— 因此需要传递 +h 参数给 **bash** 以关闭散列功能。

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

注意

本章剩余部分和后续各章中的命令都要在 chroot 环境中运行。如果您因为一些原因 (如重新启动计算机) 离开了该环境，必须确认虚拟内核文件系统如第 7.3.2 节 “挂载和填充 /dev”和第 7.3.3 节 “挂载虚拟内核文件系统”所述挂载好，然后重新进入 chroot 环境，才能继续安装 LFS。

7.5. 创建目录

现在可以在 LFS 文件系统中创建完整的目录结构。执行以下命令，创建一棵标准目录树:

```
mkdir -pv /{bin,boot,etc/{opt,sysconfig},home,lib/firmware,mnt,opt}
mkdir -pv /{media/{floppy,cdrom},srv,var}
mkdir -pv /usr/{,local/}{bin,include,lib,sbin,src}
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}
install -dv -m 1777 /tmp /var/tmp
install -dv -m 0750 /root

mkdir -v /var/{log,mail,spool}
ln -sv /run /var/run
ln -sv /run/lock /var/lock
mkdir -pv /var/{opt,cache,lib/{color,misc,locate},local}
```

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

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

7.5.1. FHS 兼容性注记

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

7.6. 创建必要的文件和符号链接

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

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

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

```
echo "127.0.0.1 localhost $(hostname)" > /etc/hosts
```

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

执行以下命令创建 /etc/passwd 文件：

```
cat > /etc/passwd << "EOF"
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/dev/null:/bin/false
daemon:x:6:6:Daemon User:/dev/null:/bin/false
messagebus:x:18:18:D-Bus Message Daemon User:/var/run/dbus:/bin/false
systemd-bus-proxy:x:72:72:systemd Bus Proxy:/:/bin/false
systemd-journal-gateway:x:73:73:systemd Journal Gateway:/:/bin/false
systemd-journal-remote:x:74:74:systemd Journal Remote:/:/bin/false
systemd-journal-upload:x:75:75:systemd Journal Upload:/:/bin/false
systemd-network:x:76:76:systemd Network Management:/:/bin/false
systemd-resolve:x:77:77:systemd Resolver:/:/bin/false
systemd-timesync:x:78:78:systemd Time Synchronization:/:/bin/false
systemd-coredump:x:79:79:systemd Core Dumper:/:/bin/false
nobody:x:99:99:Unprivileged User:/dev/null:/bin/false
EOF
```

我们以后再设置 root 用户的实际密码。

执行以下命令，创建 /etc/group 文件：

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:daemon
sys:x:2:
kmem:x:3:
tape:x:4:
tty:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
adm:x:16:
messagebus:x:18:
systemd-journal:x:23:
input:x:24:
mail:x:34:
kvm:x:61:
systemd-bus-proxy:x:72:
systemd-journal-gateway:x:73:
systemd-journal-remote:x:74:
systemd-journal-upload:x:75:
systemd-network:x:76:
systemd-resolve:x:77:
systemd-timesync:x:78:
systemd-coredump:x:79:
wheel:x:97:
nogroup:x:99:
users:x:999:
EOF
```

这里创建的用户组并不属于任何标准 —— 它们一部分是为了满足第 9 章中 Udev 配置的需要，另一部分借鉴了一些 Linux 发行版的通用惯例。另外，某些测试套件需要特定的用户或组。Linux Standard Base (LSB，可以在 http://www.linuxbase.org 查看) 标准只推荐以组 ID 0 创建用户组 root，以及以组 ID 1 创建用户组 bin，其他组名和组 ID 由系统管理员自由分配，因为好的程序不会依赖组 ID 数字，而是使用组名。

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

```
echo "tester:x:$(ls -n $(tty) | cut -d" " -f3):101::/home/tester:/bin/bash" >> /etc/passwd
echo "tester:x:101:" >> /etc/group
install -o tester -d /home/tester
```

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

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

注意这里使用了 +h 参数。它告诉 **bash** 不要使用内部的路径散列机制。如果没有指定该参数，**bash** 会记忆它执行过程序的路径。为了在安装新编译好的程序后马上使用它们，在本章和下一章中总是使用 +h。

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

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

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

注意

文件 /run/utmp 记录当前登录的用户，它由引导脚本动态创建。

7.7. GCC-10.1.0 中的 Libstdc++，第二遍

在构建第二遍的 GCC时，我们不得不暂缓安装 C++ 标准库，因为当时没有编译器能够编译它。我们不能使用那一节构建的编译器，因为它是一个本地编译器，不应在 chroot 外使用，否则可能导致编译产生的库被宿主系统组件污染。

估计构建时间: 1.1 SBU
需要硬盘空间: 1.1 GB

7.7.1. 安装目标系统的 Libstdc++

注意

Libstdc++ 是 GCC 源代码的一部分。您应该先解压 GCC 压缩包并切换到解压出来的 gcc-10.1.0 目录。

创建一个符号链接，允许在 GCC 源码树中构建 Libstdc++:

```
ln -s gthr-posix.h libgcc/gthr-default.h
```

为 Libstdc++ 创建一个单独的构建目录，并切换到该目录:

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

准备编译 Libstdc++:

```
../libstdc++-v3/configure           \
    CXXFLAGS="-g -O2 -D_GNU_SOURCE" \
    --prefix=/usr                   \
    --disable-multilib              \
    --disable-nls                   \
    --disable-libstdcxx-pch
```

配置选项的含义:

`CXXFLAGS="-g -O2 -D_GNU_SOURCE"`
这些编译开关在构建完整的 GCC 时，由顶层目录 Makefile 传递。

`--disable-libstdcxx-pch`
这个开关防止安装预编译包含文件，它在当前阶段没有必要。

运行以下命令编译 Libstdc++:

```
make
```

安装这个库:

```
make install
```

该软件包的详细信息可以在第 8.26.2 节 “GCC 的内容”中找到。

7.8. Bison-3.6.4

Bison 软件包包含语法分析器生成器。

估计构建时间: 0.3 SBU
需要硬盘空间: 47 MB

7.8.1. 安装 Bison

准备编译 Bison:

```
./configure --prefix=/usr \
            --docdir=/usr/share/doc/bison-3.6.4
```

配置选项的含义:

--docdir=/usr/share/doc/bison-3.6.4
该选项告诉构建系统将 Bison 文档安装到带有版本号的目录中。

编译该软件包:

```
make
```

安装该软件包:

```
make install
```

该软件包的详细信息可以在第 8.32.2 节 “Bison 的内容”中找到。

7.9. Gettext-0.20.2

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

估计构建时间: 1.7 SBU
需要硬盘空间: 303 MB

7.9.1. 安装 Gettext

对于我们的临时工具，只要安装 Gettext 中的三个程序即可。

准备编译 Gettext:

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

配置选项的含义:

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

编译该软件包:

```
make
```

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

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

该软件包的详细信息可以在第 8.31.2 节 “Gettext 的内容”中找到。

7.10. Perl-5.30.3

Perl 软件包包含实用报表提取语言。

估计构建时间: 1.5 SBU
需要硬盘空间: 261 MB

7.10.1. 安装 Perl

准备编译 Perl:

```
sh Configure -des -Dprefix=/usr
```

新出现的配置选项的含义:

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

编译该软件包:

```
make
```

安装该软件包:

```
make install
```

该软件包的详细信息可以在第 8.40.2 节 “Perl 的内容”中找到。

7.11. Python-3.8.3

Python 3 软件包包含 Python 开发环境。它被用于面向对象编程，编写脚本，为大型程序建立原型，或者开发完整的应用。

估计构建时间: 1.2 SBU
需要硬盘空间: 397 MB

7.11.1. 安装 Python

注意

该软件包包含两个以“python”开头的压缩包。我们应该解压的包是 `Python-3.8.3.tar.xz` (注意首字母是大写的)。

准备编译 Python:

```
./configure --prefix=/usr --without-ensurepip
```

配置选项的含义:

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

编译该软件包:

```
make
```

安装该软件包:

```
make install
```

关于该软件包的详细信息可以在第 8.49.2 节 “Python 3 的内容”中找到。

7.12. Texinfo-6.7

Texinfo 软件包包含阅读、编写和转换 info 页面的程序。

估计构建时间: 0.3 SBU
需要硬盘空间: 105 MB

7.12.1. 安装 Texinfo

准备编译 Texinfo：

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

注意

在配置过程中，一项测试报告与 TextXS_la-TestXS.lo 相关的错误。这和 LFS 没有关系，应该忽略该错误。

编译该软件包：

```
make
```

安装该软件包：

```
make install
```

该软件包的详细信息可以在第 8.68.2 节 “Texinfo 的内容”中找到。

7.13. Util-linux-2.35.2

Util-linux 软件包包含一些工具程序。

估计构建时间: 0.7 SBU
需要硬盘空间: 129 MB

7.13.1. 安装 Util-linux

首先创建一个目录，允许 **hwclock** 程序存储数据:

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

准备编译 Util-linux:

```
./configure ADJTIME_PATH=/var/lib/hwclock/adjtime    \
            --docdir=/usr/share/doc/util-linux-2.35.2 \
            --disable-chfn-chsh  \
            --disable-login      \
            --disable-nologin    \
            --disable-su         \
            --disable-setpriv    \
            --disable-runuser    \
            --disable-pylibmount \
            --disable-static     \
            --without-python
```

配置选项的含义:

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

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

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

编译该软件包:

```
make
```

安装该软件包:

```
make install
```

该软件包的详细信息可以在第 8.73.2 节 “Util-linux 的内容”中找到。

7.14. 清理和备份临时系统

libtool .la 文件仅在链接到静态库时有用。在使用动态共享库时它们没有意义，甚至有害，特别是对于非 autotools 构建系统。仍然在 chroot 环境中，运行命令中删除它们:

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

注意

本节中的其余步骤都是可选的。不过，一旦您开始在第 8 章中安装软件包，临时工具就会被覆盖。因此，按照下面描述的步骤备份临时工具可能是个好主意。其余步骤只有在您的磁盘空间非常紧张时才需要执行。

以下步骤在 chroot 环境之外进行。换句话说，您需要在继续操作之前退出 chroot 环境。这样做的主要原因是:

- 保证在操作时，目标文件不会被使用。
- 访问 chroot 环境之外的文件系统位置，以写入或读取备份档案，备份档案不应存放在 $LFS 目录树中，以保证安全。

退出 chroot 环境，并解除挂载内核虚拟文件系统:

注意

以下给出的所有步骤都以 root 身份执行。请非常小心地执行命令，错误的命令可能修改您的宿主系统。特别注意环境变量 LFS 会自动为用户 lfs 设定，但可能没有为 root 设定。无论何时，只要准备以 root 身份执行命令，一定要确认 LFS 变量正确设定。第 2.6 节 “设置 $LFS 环境变量”已经讨论了这个问题。

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

7.14.1. 移除无用内容

如果 LFS 分区比较小，好消息是，有些无用的内容可以删除。到现在为止，已经构建的可执行文件和库包含大约 90MB 的无用调试符号。

从二进制文件移除调试符号:

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

以上命令会跳过一些文件，并报告说无法识别它们的格式。这些文件大多数都是脚本文件，而不是二进制文件。

注意不要对库文件使用 --strip-unneeded 选项。这会损坏静态库，结果工具链软件包都要重新构建。

为了节约更多空间 (略高于 35 MB)，删除文档:

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

现在，您应该保证 chroot 分区有至少 5 GB 的可用空间，以在下一阶段构建和安装 Glibc 和 GCC。如果空间足够构建和安装 Glibc，那么构建和安装剩余的软件包就不成问题。您可以使用命令 **df -h $LFS** 查询磁盘可用空间。

7.14.2. 备份

现在已经建立了必要的工具，可以考虑备份它们。如果对之前构建的软件包进行的各项检查都没有发现问题，即可判定您的临时工具状态良好，可以将它们备份起来供以后重新使用。如果在后续章节发生了无法挽回的错误，通常来说，最好的办法是删除所有东西，然后 (更小心地) 从头开始。不幸的是，这也会删除所有临时工具。为了避免浪费时间对已经构建成功的部分进行返工，可以准备一个备份。

确认在 root 的主目录中，有至少 600 MB 的可用存储空间 (源代码压缩包也会被包含在备份档案中)。

运行以下命令，创建备份档案：

```
cd $LFS &&
tar -cJpf $HOME/lfs-temp-tools-20200622-systemd.tar.xz .
```

如果您不想将备份存储在 root 的主目录中，将 $HOME 替换为您选择的其他位置。

7.14.3. 还原

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

```
cd $LFS &&
rm -rf ./* &&
tar -xpf $HOME/lfs-temp-tools-20200622-systemd.tar.xz
```

再一次复查环境是否配置正确，即可继续构建系统。

重要

如果您在移除调试符号，进行备份，或从备份进行恢复时退出了 chroot 环境，记得按照第 7.3 节 “准备虚拟内核文件系统”的描述重新挂载内核虚拟文件系统，并重新进入 chroot 环境 (参阅第 7.4 节 “进入 Chroot 环境”)，再继续进行构建。

第 IV 部分 构建 LFS 系统

第 8 章 安装基本系统软件

8.1. 概述

在本章中，我们将真正开始构造 LFS 系统。

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

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

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

注意

对于拥有可用的测试套件的软件包，第 8 章中给出的 SBU 值和需要的磁盘空间包含了运行测试套件需要的时间和磁盘空间。SBU 值根据仅使用单个 CPU 核心 (-j1) 进行操作时测得的时间计算。

8.1.1. 关于库

一般来说，LFS 作者不推荐构建和安装静态库。它们是为了某些在现代 Linux 系统中早已过时的原因而存在的。另外，将静态库链接到程序中可能是有害的。如果需要更新这个库以解决安全问题，所有使用该静态库的程序都要重新链接到新版本的库。程序对静态库的使用并不总是显然的，甚至可能无法查明有哪些程序需要重新链接 (以及如何重新链接)。

在本章的安装过程中，我们删除或者禁止安装多数静态库。一般来说，传递 `--disable-static` 选项给 **configure** 就可以禁用静态库。然而，某些情况下需要其他手段。在极个别情况下，特别是对于 Glibc 和 GCC，静态库对于一般的软件包构建过程仍然很关键，就不能禁用静态库。

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

8.2. 软件包管理

经常有人请求将软件包管理加入 LFS 手册。包管理器可以跟踪文件的安装过程，简化移除或升级软件包的工作。如同处理二进制程序和库文件一样，包管理器也会处理配置文件的安装过程。在您开始想入非非前，不 —— 本节不会讨论或者推荐任何一个特定的包管理器。本节对软件包管理的流行技术及其工作原理进行综述。对您来说，完美的包管理器可能是其中的某个技术，也可能是几个技术的结合。本节还会简要介绍在升级软件包时可能遇到的问题。

LFS 或 BLFS 不介绍任何包管理器的原因包括：

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

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

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

8.2.1. 升级问题

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

· 如果需要升级 Glibc (例如从 Glibc-2.31 升级到 Glibc-2.32)，最安全的方法是重新构建 LFS。尽管您或许能按依赖顺序重新构建所有软件包，但我们不推荐这样做。

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

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。在决定使用某种包管理器前，请研读这些技术，特别是要了解特定技术的不足。

8.2.2.1. 这都在我的脑袋里!

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

8.2.2.2. 安装到独立目录

这是一种最简单的软件包管理方式，它不需要任何额外的软件来控制软件包的安装。每个软件包都被安装在单独的目录中。例如，软件包 foo-1.1 将会被安装在 /usr/pkg/foo-1.1，然后创建一个符号链接 /usr/pkg/foo 指向 /usr/pkg/foo-1.1。在安装新版本 foo-1.2 的时候，把它安装到 /usr/pkg/foo-1.2，然后把之前的符号链接替换为指向新版本的符号链接。

PATH、LD_LIBRARY_PATH、MANPATH、INFOPATH 和 CPPFLAGS 等环境变量需要被扩充，以包含 /usr/pkg/foo。一旦软件包的数量较多，这种架构就会变得无法管理。

8.2.2.3. 符号链接风格的软件包管理

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

安装过程需要伪装，使得软件包认为它处于 /usr 中，尽管它实际上被安装在 /usr/pkg 目录结构中。这种安装过程一般是超出常规的。例如，考虑安装软件包 libfoo-1.1。下面的指令可能不能正确安装该软件包：

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

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

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

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

8.2.2.4. 基于时间戳的方案

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

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

8.2.2.5. 追踪安装脚本

在这种方式中，安装脚本执行的命令被记录下来。有两种技术可以进行记录：

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

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

8.2.2.6. 创建软件包档案

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

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

创建包含依赖关系信息的软件包文件十分复杂，超出了 LFS 的范畴。

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

8.2.2.7. 基于用户的软件包管理

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

8.2.3. 在多个系统上部署 LFS

LFS 系统的一项优势是，没有依赖于磁盘系统中文件位置的文件。将构建好的 LFS 系统复制到另一台具有相同硬件架构的计算机很简单，只要用 **tar** 命令把包含根目录的 LFS 分区打包 (未压缩的情况下，一个基本的 LFS 系统需要 250 MB)，然后通过网络或者 CD-ROM 复制到新的系统上，再展开即可。这时，个别配置文件需要修改。可能需要更新的配置文件有：`/etc/hosts`，`/etc/fstab`，`/etc/passwd`，`/etc/group`，`/etc/shadow`，以及 `/etc/ld.so.conf`。

由于系统硬件和原始内核配置的区别，可能需要为新系统重新配置并构建内核。

注意

有一些报告反映称，在架构相近但不完全一致的计算机之间拷贝 LFS 系统时出现问题。例如，Intel 系统使用的指令集和 AMD 处理器不完全相同，且较新的处理器可能包含旧处理器没有的指令。

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

8.3. Man-pages-5.07

Man-pages 软件包包含 2,200 多个 man 页面。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 31 MB

8.3.1. 安装 Man-pages

执行以下命令安装 Man-pages:

```
make install
```

8.3.2. Man-pages 的内容

安装的文件: 若干 man 页面

简要描述

man 页面	描述 C 语言函数、重要的设备文件以及主要配置文件

8.4. Tcl-8.6.10

Tcl 软件包包含工具命令语言，它是一个可靠的通用脚本语言。Except 软件包是用 Tcl 语言编写的.

估计构建时间: 0.9 SBU
需要硬盘空间: 76 MB

8.4.1. 安装 Tcl

为了支持 GCC 和 Binutils 等软件包测试套件的运行，需要安装这个软件包和接下来的两个 (Expect 与 DejaGNU)。为了测试目的安装三个软件包看似浪费，但是只有运行了测试，才能放心地确定多数重要工具可以正常工作，即使测试不是必要的。我们必须安装这些软件包，才能执行本章中的测试套件。

首先，运行以下命令解压文档:

```
tar -xf ../tcl8.6.10-html.tar.gz --strip-components=1
```

准备编译 Tcl:

```
SRCDIR=$(pwd)
cd unix
./configure --prefix=/usr           \
            --mandir=/usr/share/man \
            $([ "$(uname -m)" = x86_64 ] && echo --enable-64bit)
```

配置选项的含义:

`$([ "$(uname -m)" = x86_64 ] && echo --enable-64bit)`
$(<shell 命令>) 会被 shell 替换为 shell 命令的输出。这里，如果在 32 位机器上，输出是空的，而如果在 64 位机器上，输出是 `--enable-64bit`。

构建该软件包:

```
make

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

sed -e "s|$SRCDIR/unix/pkgs/tdbc1.1.1|/usr/lib/tdbc1.1.1|" \
    -e "s|$SRCDIR/pkgs/tdbc1.1.1/generic|/usr/include|"    \
    -e "s|$SRCDIR/pkgs/tdbc1.1.1/library|/usr/lib/tcl8.6|" \
    -e "s|$SRCDIR/pkgs/tdbc1.1.1|/usr/include|"            \
    -i pkgs/tdbc1.1.1/tdbcConfig.sh

sed -e "s|$SRCDIR/unix/pkgs/itcl4.2.0|/usr/lib/itcl4.2.0|" \
    -e "s|$SRCDIR/pkgs/itcl4.2.0/generic|/usr/include|"    \
    -e "s|$SRCDIR/pkgs/itcl4.2.0|/usr/include|"            \
    -i pkgs/itcl4.2.0/itclConfig.sh

unset SRCDIR
```

“make”命令之后的若干“sed”命令从配置文件中删除构建目录，并用安装目录替换它们。构建 LFS 的后续过程不对此严格要求，但如果之后构建使用 Tcl 的软件包，则可能需要这样的操作。

运行以下命令，以测试编译结果：

```
make test
```

安装该软件包：

```
make install
```

将安装好的库加上写入权限，以便将来移除调试符号：

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

安装 Tcl 的头文件。下一个软件包 Expect 需要它们才能构建。

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

创建一个必要的符号链接：

```
ln -sfv tclsh8.6 /usr/bin/tclsh
```

8.4.2. Tcl 的内容

安装的程序: tclsh (到 tclsh8.6 的链接) 和 tclsh8.6
安装的库: libtcl8.6.so 和 libtclstub8.6.a

简要描述

tclsh8.6	Tcl 命令行 shell
tclsh	一个指向 tclsh8.6 的链接
`libtcl8.6.so`	Tcl 运行库
`libtclstub8.6.a`	Tcl 端桩库

8.5. Expect-5.45.4

Expect 软件包包含通过脚本控制的对话，自动化 **telnet**，**ftp**，**passwd**，**fsck**，**rlogin**，以及 **tip** 等交互应用的工具。Expect 对于测试这类程序也很有用，它简化了这类通过其他方式很难完成的工作。DejaGnu 框架是使用 Expect 编写的。

估计构建时间: 0.1 SBU
需要硬盘空间: 3.9 MB

8.5.1. 安装 Expect

准备编译 Expect:

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

配置选项的含义:

`--with-tcl=/usr/lib`
需要使用该选项告知 **configure** 配置脚本 **tclConfig.sh** 的位置。

`--with-tclinclude=/usr/include`
该选项显式指定查找 Tcl 内部头文件的位置。

构建该软件包:

```
make
```

运行以下命令，以测试编译结果:

```
make test
```

安装该软件包:

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

8.5.2. Expect 的内容

安装的程序: expect
安装的库: libexpect-5.45.so

简要描述

expect	根据一个脚本与其他交互程序交流
`libexpect-5.45.so`	包含一些函数，使得 Expect 可以作为 Tcl 扩展使用，也可以直接在 C 或 C++ 中使用 (不使用 Tcl)

8.6. DejaGNU-1.6.2

DejaGnu 包含使用 GNU 工具运行测试套件的框架。它是用 **expect** 编写的，后者又使用 Tcl (工具命令语言)。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 4.6 MB

8.6.1. 安装 DejaGNU

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
```

构建并安装该软件包：

```
make install
install -v -dm755  /usr/share/doc/dejagnu-1.6.2
install -v -m644   doc/dejagnu.{html,txt} /usr/share/doc/dejagnu-1.6.2
```

如果要测试该软件包，执行：

```
make check
```

8.6.2. DejaGNU 的内容

安装的程序: runtest

简要描述

runtest 一个寻找正确的 **expect** shell，并运行 DejaGNU 的封装脚本。

8.7. Iana-Etc-20200429

Iana-Etc 软件包包含网络服务和协议的数据。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 4.7 MB

8.7.1. 安装 Iana-Etc

对于该软件包，我们只需要将文件复制到正确的位置：

```
cp services protocols /etc
```

8.7.2. Iana-Etc 的内容

安装的文件: /etc/protocols 和 /etc/services

简要描述

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

8.8. Glibc-2.31

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

估计构建时间: 19 SBU
需要硬盘空间: 2.6 GB

8.8.1. 安装 Glibc

某些 Glibc 程序使用与 FHS 不兼容的 /var/db 目录存放运行时数据。应用下列补丁，使得这些程序在 FHS 兼容的位置存储运行时数据:

```
patch -Np1 -i ../glibc-2.31-fhs-1.patch
```

Glibc 文档推荐在专用目录中构建它:

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

准备编译 Glibc:

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

配置选项的含义:

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

`--enable-kernel=3.2`
该选项告诉构建系统 Glibc 可能被与 3.2 这样老版本的内核一起使用。这样，Glibc 会生成代码，在后续版本引入的系统调用不可用时绕过它们。

`--enable-stack-protector=strong`
该选项通过加入额外代码，对栈溢出攻击等导致的缓冲区溢出进行检查，以提高系统安全性。

`--with-headers=/usr/include`
该选项指定构建系统搜索内核 API 头文件的位置。

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

编译该软件包:

```
make
```

重要

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

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

```
case $(uname -m) in
  i?86)   ln -sfnv $PWD/elf/ld-linux.so.2        /lib ;;
  x86_64) ln -sfnv $PWD/elf/ld-linux-x86-64.so.2 /lib ;;
esac
```

注意

我们需要上面的符号链接，以便在当前的 chroot 构建环境中运行测试套件。后续的安装过程将会覆盖它。

```
make check
```

您可能看到一些失败结果。Glibc 的测试套件和宿主系统之间有某种依赖关系。下面列出是在一些版本的 LFS 上发现的，最常见的问题：

- 已知 misc/tst-ttyname 在 LFS chroot 环境中会失败。
- 已知 nss/tst-nss-files-hosts-multi 可能失败，原因尚未查明。
- rt/tst-cputimer{1,2,3} 测试依赖于宿主系统的内核。已知内核版本 4.14.91–4.14.96，4.19.13–4.19.18，以及 4.20.0–4.20.5 会导致它们失败。
- 如果 CPU 不是较新的 Intel 或 AMD 处理器，数学测试有时会失败。

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

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

修正生成的 Makefile，跳过一个在 LFS 的不完整环境中会失败的完整性检查：

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

安装该软件包：

```
make install
```

安装 **nscd** 的配置文件和运行时目录：

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

安装 **nscd** 的 systemd 支持文件：

```
install -v -Dm644 ../nscd/nscd.tmpfiles /usr/lib/tmpfiles.d/nscd.conf
install -v -Dm644 ../nscd/nscd.service /lib/systemd/system/nscd.service
```

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

可以用 **localedef** 程序安装单独的 locale。例如，下面的第一个 **localedef** 命令将 /usr/share/i18n/locales/cs_CZ 中的字符集无关 locale 定义和 /usr/share/i18n/charmaps/UTF-8.gz 中的字符映射定义组合起来，并附加到 /usr/lib/locale/locale-archive 文件。以下命令将会安装能够覆盖测试所需的最小 locale 集合:

```
mkdir -pv /usr/lib/locale
localedef -i POSIX -f UTF-8 C.UTF-8 2> /dev/null || true
localedef -i cs_CZ -f UTF-8 cs_CZ.UTF-8
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
localedef -i de_DE -f UTF-8 de_DE.UTF-8
localedef -i el_GR -f ISO-8859-7 el_GR
localedef -i en_GB -f 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_MX -f ISO-8859-1 es_MX
localedef -i fa_IR -f UTF-8 fa_IR
localedef -i fr_FR -f ISO-8859-1 fr_FR
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i fr_FR -f UTF-8 fr_FR.UTF-8
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i 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.SIJS 2> /dev/null || true
localedef -i ja_JP -f UTF-8 ja_JP.UTF-8
localedef -i ru_RU -f KOI8-R ru_RU.KOI8-R
localedef -i ru_RU -f UTF-8 ru_RU.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
```

另外，安装适合您自己国家、语言和字符集的 locale。

或者，也可以一次安装 glibc-2.31/localedata/SUPPORTED 中列出的所有 locale (包括上面列出的所有 locale，以及其他很多)。执行下面这个需要很长时间的命令:

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

如果需要，再使用 **localedef** 命令创建和安装 glibc-2.31/localedata/SUPPORTED 中没有列出的 locale，当然您不太可能需要它们。

注意

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

8.8.2. 配置 Glibc

8.8.2.1. 创建 nsswitch.conf

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

执行以下命令创建新的 /etc/nsswitch.conf:

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

passwd: files
group: files
shadow: files

hosts: files dns
networks: files

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

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

8.8.2.2. 添加时区数据

输入以下命令，安装并设置时区数据:

```
tar -xf ../../tzdata2020a.tar.gz

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

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

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

zic 命令的含义:

zic -L /dev/null ...
该命令创建没有闰秒的 POSIX 时区。一般的惯例是将它们安装在 zoneinfo 和 zoneinfo/posix 两个目录中。必须将 POSIX 时区安装到 zoneinfo，否则若干测试套件会报告错误。在嵌入式系统上，如果存

储空间十分紧张，而且您永远不会更新时区信息，您可以不使用 posix 目录，以节约 1.9 MB，但个别程序或测试套件可能会失败。

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

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

一种确定本地时区的方法是运行脚本：

```
tzselect
```

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

确定时区后，执行以下命令，创建 /etc/localtime：

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

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

8.8.2.3. 配置动态加载器

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

运行以下命令，创建一个新的 /etc/ld.so.conf：

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

EOF
```

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

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

EOF
mkdir -pv /etc/ld.so.conf.d
```

8.8.3. Glibc 的内容

安装的程序: catchsegv, gencat, getconf, getent, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, makedb, mtrace, nscd, pcprofiledump, pldd, sln, sotruss, sprof, tzselect, xtrace, zdump, 以及 zic

安装的库: ld-2.31.so, libBrokenLocale.{a,so}, libSegFault.so, libanl.{a,so}, libc.{a,so}, libc_nonshared.a, libcrypt.{a,so}, libdl.{a,so}, libg.a, libm.{a,so}, libmcheck.a, libmemusage.so, libmvec.{a,so}, libnsl.{a,so}, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libpcprofile.so, libpthread.{a,so}, libpthread_nonshared.a, libresolv.{a,so}, librt.{a,so}, libthread_db.so, 以及 libutil.{a,so}

安装的目录: /usr/include/arpa, /usr/include/bits, /usr/include/gnu, /usr/include/net, /usr/include/netash, /usr/include/netatalk, /usr/include/netax25, /usr/include/neteconet, /usr/include/netinet, /usr/include/netipx, /usr/include/netiucv, /usr/include/netpacket, /usr/include/netrom, /usr/include/netrose, /usr/include/nfs, /usr/include/protocols, /usr/include/rpc, /usr/include/sys, /usr/lib/audit, /usr/lib/gconv, /usr/lib/locale, /usr/libexec/getconf, /usr/share/i18n, /usr/share/zoneinfo, /var/cache/nscd, 以及 /var/lib/nss_db

简要描述

catchsegv	在程序因为段错误而终止时创建栈跟踪
gencat	生成消息目录
getconf	显示文件系统指定的系统配置变量值
getent	从管理数据库取得条目
iconv	转换给定文件的字符集
iconvconfig	创建可快速加载的 **iconv** 模块配置文件
ldconfig	配置动态链接器运行时绑定
ldd	报告给定程序或共享库依赖于哪些共享库
lddlibc4	辅助 **ldd** 处理对象文件
locale	给出当前 locale 的一些信息
localedef	编译 locale 规范
makedb	从文本输入创建简单的数据库
mtrace	读取并解析内存跟踪文件，以人类可读的形式输出内存跟踪信息
nscd	一个缓存最常见命名服务请求的守护进程
pcprofiledump	显示基于程序计数器的性能剖析数据
pldd	列出正在运行的进程使用的共享库
sln	静态链接的 **ln** 程序
sotruss	跟踪特定命令对共享库中子程序的调用
sprof	读取并显示共享库性能剖析数据
tzselect	询问用户系统所在的位置并报告对应的时区

xtrace	显示正在执行的函数以跟踪程序执行
zdump	输出当前时间在多个时区中的表示
zic	时区编译器
ld-2.31.so	动态链接器/加载器
libBrokenLocale	被 Glibc 内部用作使某些不正确的程序 (例如某些 Motif 程序) 正常运行的粗糙手段，参阅 glibc-2.31/locale/broken_cur_max.c 中的注释了解更多信息
libSegFault	**catchsegv** 使用的段错误信号处理程序
libanl	异步的命名查找库
libc	主要的 C 运行库
libcrypt	密码学库
libdl	动态链接接口库
libg	没有功能的空库，曾经是 g++ 的运行库。
libm	数学库
libmcheck	链接到该库时启用内存分配检查
libmemusage	被 **memusage** 用于收集程序内存使用信息
libnsl	网络服务库
libnss	命名服务开关库，包含用于解析域名、用户名、组名、代号、 服务、协议等的函数。
libpcprofile	可以预加载它，以对程序进行基于程序计数器的性能剖析
libpthread	POSIX 线程库
libresolv	包含用于创建、发送和解析因特网域名服务数据包的函数。
librt	包含 POSIX.1b 实时扩展要求的多数接口
libthread_db	包含用于构建多线程程序调试的函数器
libutil	包含许多 Unix 工具使用的 “标准” 函数

8.9. Zlib-1.2.11

Zlib 软件包包含一些程序使用的压缩和解压缩子程序。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 5.1 MB

8.9.1. 安装 Zlib

准备编译 Zlib:

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

编译该软件包:

```
make
```

运行以下命令以测试编译结果

```
make check
```

安装该软件包:

```
make install
```

共享库需要被移动到 /lib 中，因此 .so 文件需要在/usr/lib 目录中重建:

```
mv -v /usr/lib/libz.so.* /lib
ln -sfv ../../lib/$(readlink /usr/lib/libz.so) /usr/lib/libz.so
```

8.9.2. Zlib 的内容

安装的库: libz.{a,so}

简要描述

libz 包含一些程序使用的压缩和解压缩函数

8.10. Bzip2-1.0.8

Bzip2 软件包包含用于压缩和解压缩文件的程序。使用 **bzip2** 压缩文本文件可以获得比传统的 **gzip** 优秀许多的压缩比。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 7.7 MB

8.10.1. 安装 Bzip2

应用一个补丁，以安装该软件包的文档：

```
patch -Np1 -i ../bzip2-1.0.8-install_docs-1.patch
```

以下命令保证安装的符号链接是相对的：

```
sed -i 's@\(ln -s -f \)$(PREFIX)/bin/@\1@' Makefile
```

确保 man 页面被安装到正确位置：

```
sed -i "s@(PREFIX)/man@(PREFIX)/share/man@g" Makefile
```

执行以下命令，准备编译 Bzip2：

```
make -f Makefile-libbz2_so
make clean
```

make 命令参数的含义：

`-f Makefile-libbz2_so`
该命令使用一个不同的 `Makefile` 文件构建 Bzip2，对于我们的例子来说就是使用 `Makefile-libbz2_so` 文件。它创建一个共享库 `libbz2.so`，并将 Bzip2 工具链接到这个库。

编译并测试该软件包：

```
make
```

安装软件包中的程序：

```
make PREFIX=/usr install
```

安装链接到共享库的 **bzip2** 二进制程序到 `/bin` 目录，创建必要的符号链接，并进行清理：

```
cp -v bzip2-shared /bin/bzip2
cp -av libbz2.so* /lib
ln -sv ../../lib/libbz2.so.1.0 /usr/lib/libbz2.so
rm -v /usr/bin/{bunzip2,bzcat,bzip2}
ln -sv bzip2 /bin/bunzip2
ln -sv bzip2 /bin/bzcat
```

8.10.2. Bzip2 的内容

安装的程序: bunzip2 (链接到 bzip2), bzcat (链接到 bzip2), bzcmp (链接到 bzdiff), bzdiff, bzegrep (链接到 bzgrep), bzfgrep (链接到 bzgrep), bzgrep, bzip2, bzip2recover, bzless (链接到bzmore), 以及 bzmore
安装的库: libbz2.{a,so}
安装的目录: /usr/share/doc/bzip2-1.0.8

简要描述

bunzip2	解压缩 bzip 压缩文件
bzcat	解压缩到标准输出
bzcmp	对 bzip 压缩过的文件运行 **cmp**
bzdiff	对 bzip 压缩过的文件运行 **diff**
bzegrep	对 bzip 压缩过的文件运行 **egrep** 命令
bzfgrep	对 bzip 压缩过的文件运行 **fgrep** 命令
bzgrep	对 bzip 压缩过的文件运行 **grep** 命令
bzip2	使用 Burrows-Wheeler 块排序文本压缩算法和 Huffman 编码压缩文件；其压缩率优于更常见的使用 “Lempel-Ziv” 算法的压缩工具，如 **gzip**
bzip2recover	试图从损坏的 bzip2 压缩文件中恢复数据
bzless	对 bzip 压缩过的文件运行 **less** 命令
bzmore	对 bzip 压缩过的文件运行 **more** 命令
`libbz2`	这个库实现基于 Burrows-Wheeler 算法的无损块排序数据压缩

8.11. Xz-5.2.5

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

估计构建时间: 0.2 SBU
需要硬盘空间: 15 MB

8.11.1. 安装 Xz

准备编译 Xz:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/xz-5.2.5
```

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

```
make check
```

安装该软件包，并保证所有重要文件都位于正确的目录中:

```
make install
mv -v   /usr/bin/{lzma,unlzma,lzcat,xz,unxz,xzcat} /bin
mv -v /usr/lib/liblzma.so.* /lib
ln -svf ../../lib/$(readlink /usr/lib/liblzma.so) /usr/lib/liblzma.so
```

8.11.2. Xz 的内容

安装的程序: lzcat (到 xz 的链接), lzcmp (到 xzdiff 的链接), lzdiff (到 xzdiff 的链接), lzegrep (到 xzgrep 的链接), lzfgrep (到 xzgrep 的链接), lzgrep (到 xzgrep 的链接), lzless (到 xzless 的链接), lzma (到 xz 的链接), lzmadec, lzmainfo, lzmore (到 xzmore 的链接), unlzma (到 xz 的链接), unxz (到 xz 的链接), xz, xzcat (到 xz 的链接), xzcmp (到 xzdiff 的链接), xzdec, xzdiff, xzegrep (到 xzgrep 的链接), xzfgrep (到 xzgrep 的链接), xzgrep, xzless, 以及 xzmore
安装的库: liblzma.so
安装的目录: /usr/include/lzma 和 /usr/share/doc/xz-5.2.5

简要描述

lzcat 解压到标准输出
lzcmp 在 LZMA 压缩文件上执行 **cmp**
lzdiff 在 LZMA 压缩文件上执行 **diff**
lzegrep 在 LZMA 压缩文件上执行 **egrep**
lzfgrep 在 LZMA 压缩文件上执行 **fgrep**
lzgrep 在 LZMA 压缩文件上执行 **grep**

lzless	在 LZMA 压缩文件上执行 **less**
lzma	使用 LZMA 格式压缩或解压缩文件
lzmadec	一个轻量、快速的 LZMA 压缩文件解码器
lzmainfo	显示 LZMA 压缩文件头中存储的信息
lzmore	在 LZMA 压缩文件上执行 **more**
unlzma	使用 LZMA 格式解压缩文件
unxz	使用 XZ 格式解压缩文件
xz	使用 XZ 格式压缩或解压缩文件
xzcat	解压到标准输出
xzcmp	在 XZ 压缩文件上执行 **cmp**
xzdec	一个轻量、快速的 XZ 压缩文件解码器
xzdiff	在 XZ 压缩文件上执行 **diff**
xzegrep	在 XZ 压缩文件上执行 **egrep**
xzfgrep	在 XZ 压缩文件上执行 **fgrep**
xzgrep	在 XZ 压缩文件上执行 **grep**
xzless	在 XZ 压缩文件上执行 **less**
xzmore	在 XZ 压缩文件上执行 **more**
`liblzma`	实现基于 Lempel-Zip-Markov 链的无损块排序数据压缩算法的库

8.12. Zstd-1.4.5

Zstandard 是一种实时压缩算法，提供了较高的压缩比。它具有很宽的压缩比/速度权衡范围，同时支持具有非常快速的解压缩。

估计构建时间: 0.6 SBU
需要硬盘空间: 16 MB

8.12.1. 安装 Zstd

编译该软件包

```
make
```

该软件包没有测试套件。

安装该软件包:

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

删除静态库，并将共享库移动到 /lib 。另外， .so 符号链接也要在 /usr/lib 中重建:

```
rm -v /usr/lib/libzstd.a
mv -v /usr/lib/libzstd.so.* /lib
ln -sfv ../../lib/$(readlink /usr/lib/libzstd.so) /usr/lib/libzstd.so
```

8.12.2. Zstd 的内容

安装的程序: zstd, zstdcat (到 zstd 的链接), zstdgrep, zstdless, zstdmt (到 zstd 的链接), 以及 unzstd (到 zstd 的链接)
安装的库: libzstd.so

简要描述

zstd 使用 ZSTD 格式压缩或解压缩文件
zstdgrep 在 ZSTD 压缩文件上运行 **grep**
zstdless 在 ZSTD 压缩文件上运行 **less**
libzstd 基于 ZSTD 算法实现无损数据压缩的库

8.13. File-5.39

File 软件包包含用于确定给定文件类型的工具。

估计构建时间: 0.1 SBU
需要硬盘空间: 13 MB

8.13.1. 安装 File

准备编译 File:

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

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.13.2. File 的内容

安装的程序: file
安装的库: libmagic.so

简要描述

file 通过进行文件系统、魔数和语言等测试，尝试对每个给定的文件进行分类

libmagic 包含 **file** 程序使用的魔数识别子程序

8.14. Readline-8.0

Readline 软件包包含一些提供命令行编辑和历史记录功能的库。

估计构建时间: 0.1 SBU
需要硬盘空间: 15 MB

8.14.1. 安装 Readline

重新安装 Readline 会导致旧版本的库被重命名为 <库名称>.old。这一般不是问题，但某些情况下会触发**ldconfig**的一个链接 bug。运行下面的两条 sed 命令防止这种情况:

```
sed -i '/MV.*old/d' Makefile.in
sed -i '/{OLDSUFF}/c:' support/shlib-install
```

准备编译 Readline:

```
./configure --prefix=/usr    \
            --disable-static \
            --with-curses    \
            --docdir=/usr/share/doc/readline-8.0
```

配置选项的含义

--with-curses
该选项告诉 Readline 它可以在 curses 库中查找 termcap 库函数，而不是单独的termcap 库。这样就能生成正确的 readline.pc 文件。

编译该软件包:

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

make 命令选项的含义

SHLIB_LIBS="-lncursesw"
该选项强制 Readline 链接到 libncursesw 库。

该软件包不包含测试套件。

安装该软件包:

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

下面将动态库移动到更合适的位置，并修正访问权限和符号链接:

```
mv -v /usr/lib/lib{readline,history}.so.* /lib
chmod -v u+w /lib/lib{readline,history}.so.*
ln -sfv ../../lib/$(readlink /usr/lib/libreadline.so) /usr/lib/libreadline.so
ln -sfv ../../lib/$(readlink /usr/lib/libhistory.so ) /usr/lib/libhistory.so
```

如果您希望的话，可以安装文档:

```
install -v -m644 doc/*.{ps,pdf,html,dvi} /usr/share/doc/readline-8.0
```

8.14.2. Readline 的内容

安装的库: libhistory.so 和 libreadline.so
安装的目录: /usr/include/readline 和 /usr/share/doc/readline-8.0

简要描述

`libhistory` 提供一个查询之前输入行的一致用户接口

`libreadline` 提供一组在程序的交互会话中操纵输入的文本的命令。

8.15. M4-1.4.18

M4 软件包包含一个宏处理器。

估计构建时间: 0.4 SBU
需要硬盘空间: 31 MB

8.15.1. 安装 M4

首先，进行 Glibc-2.28 及更新版本要求的一些修补：

```
sed -i 's/IO_ftrylockfile/IO_EOF_SEEN/' lib/*.c
echo "#define _IO_IN_BACKUP 0x100" >> lib/stdio-impl.h
```

准备编译 M4：

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

编译该软件包：

```
make
```

运行以下命令以测试编译结果：

```
make check
```

安装该软件包：

```
make install
```

8.15.2. M4 的内容

安装的程序: m4

简要描述

m4 复制给定文件，并展开它们包含的宏 这些宏可能是内置或用户定义的，可以接受任意个参数。除了展开宏外，**m4** 还包含用于包含指定文件、 运行 Unix 命令、进行整数运算、处理文本、 递归执行等功能的内建函数。**m4** 程序可以被用作编译器前端，也可以被单独用作宏处理器。

8.16. Bc-2.7.2

Bc 软件包包含一个任意精度数值处理语言。

估计构建时间: 0.1 SBU
需要硬盘空间: 3.2 MB

8.16.1. 安装 Bc

准备编译 Bc:

```
PREFIX=/usr CC=gcc CFLAGS="-std=c99" ./configure.sh -G -O3
```

配置选项的含义:

`CC=gcc CFLAGS="-std=c99"`
该选项指定编译时使用的 C 编译器和标准。

`-O3`
该选项指定编译时使用的优化等级。

`-G`
忽略在没有 GNU bc 存在时无法工作的测试。

编译该软件包:

```
make
```

为了测试 bc，运行:

```
make test
```

安装该软件包:

```
make install
```

8.16.2. Bc 的内容

安装的程序: bc 和 dc

简要描述

bc 一个命令行计算器
dc 一个逆波兰式命令行计算器

8.17. Flex-2.6.4

Flex 软件包包含一个工具，用于生成在文本中识别模式的程序。

估计构建时间: 0.4 SBU
需要硬盘空间: 36 MB

8.17.1. 安装 Flex

准备编译 Flex:

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

编译该软件包:

```
make
```

如果要测试编译结果 (需要约 0.5 SBU), 执行:

```
make check
```

安装该软件包:

```
make install
```

个别程序还不知道 **flex**，并试图去运行它的前身 **lex**。为了支持这些程序，创建一个名为 `lex` 的符号链接，它运行 `flex` 并启动其模拟 **lex** 的模式:

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

8.17.2. Flex 的内容

安装的程序: flex, flex++ (到 flex 的链接), 以及 lex (到 flex 的链接)
安装的库: libfl.so
安装的目录: /usr/share/doc/flex-2.6.4

简要描述

flex 一个用于生成在文本文件中识别模式的程序的工具，它允许灵活地指定查找模式的规则，消除了开发专用程序的需要。

flex++ flex 的扩展，用于生成 C++ 代码和类。它是一个指向 **flex** 的符号链接

lex 一个以 **lex** 仿真模式运行 **flex** 的符号链接

`libfl` `flex` 库

8.18. Binutils-2.34

Binutils 包含汇编器、链接器以及其他用于处理目标文件的工具。

估计构建时间: 6.6 SBU
需要硬盘空间: 4.7 GB

8.18.1. 安装 Binutils

进行简单测试，确认伪终端 (PTY) 在 chroot 环境中能正常工作：

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

该命令应该输出：

```
spawn ls
```

如果输出不是上面这样，而是下面的消息，就说明环境没有为 PTY 的正常工作设置好。在运行 Binutils 和 GCC 的测试套件前必须解决这个问题。

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

删除一项导致测试套件无法完成的测试，并且修复 gold 测试套件中的一些需要为 GCC 10 进行调整的其他测试：

```
sed -i '/@\tincremental_copy/d' gold/testsuite/Makefile.in
patch -Np1 -i ../binutils-2.34-gcc10_gold_test_fix-1.patch
```

Binutils 文档推荐在一个专用的构建目录中构建 Binutils：

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

准备编译 Binutils：

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

配置选项的含义：

`--enable-gold`
构建 gold 链接器，并且将它 (和默认链接器一起) 安装为 ld.gold。

`--enable-ld=default`
构建传统的 bfd 链接器，并且将它安装为 ld (默认链接器) 和 ld.bfd。

`--enable-plugins`
启用链接器插件支持。

`--enable-64-bit-bfd`
(在字长较小的宿主平台上) 启用 64 位支持。在 64 位平台上可能不需要，但无害。

`--with-system-zlib`
使用安装好的 zlib 库，而不是构建附带的版本。

编译该软件包：

```
make tooldir=/usr
```

make 命令选项的含义：

`tooldir=/usr`
一般来说，工具目录 (最终存放该软件包中可执行文件的目录) 被设定为 `$(exec_prefix)/$(target_alias)`。例如，在 x86_64 机器上，它将展开为 `/usr/x86_64-unknown-linux-gnu` 。因为 LFS 是定制系统，不需要 `/usr` 中的特定目标工具目录。如果系统用于交叉编译 (例如，在 Intel 机器上编译软件包，生成可以在 PowerPC 机器上执行的代码)，就会使用 `$(exec_prefix)/$(target_alias)` 目录。

重要

本节中，Binutils 的测试套件被认为是十分关键的，在任何情况下都不能跳过。

测试编译结果：

```
make -k check
```

安装该软件包：

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

8.18.2. Binutils 的内容

安装的程序:	addr2line, ar, as, c++filt, dwp, elfedit, gprof, ld, ld.bfd, ld.gold, nm, objcopy, objdump, ranlib, readelf, size, strings, 以及 strip
安装的库:	libbfd.{a,so} 和 libopcodes.{a,so}
安装的目录:	/usr/lib/ldscripts

简要描述

addr2line	将程序中的地址翻译成文件名和行号； 给定一个内存地址以及可执行程序的名字，该程序使用可执行文件中的调试信息，确定与该地址相关的源代码文件和行号。
ar	创建、修改、提取档案文件
as	一个能够汇编 gcc 输出的汇编代码并生成目标文件的汇编器
c++filt	被链接器用于 demangle C++ 和 Java 符号，防止重载函数冲突。
dwp	DWARF 封装工具
elfedit	更改 ELF 文件的 ELF 头
gprof	显示函数调用图性能分析数据
ld	一个链接器，将一些对象文件和档案文件组合为一个单独的文件，重定位它们的数据，并绑定符号引用
ld.gold	ld 的一个裁减版，只支持 ELF 目标文件格式

ld.bfd	**ld** 的硬链接
nm	列出给定目标文件中的符号
objcopy	将一种目标文件翻译成另一种
objdump	显示给定目标文件的信息，通过命令行选项指定要显示哪些信息； 这些信息对开发编译工具的程序员很有用
ranlib	生成档案文件内容的索引，并将索引存入档案文件； 索引列出档案文件中所有可重定位目标文件定义的符号
readelf	显示 ELF 格式二进制文件的信息
size	列出给定文件各个段的大小和文件总大小
strings	对于每个给定文件，输出其中长度不小于给定长度 (默认是 4) 的可打印字符序列；对于目标文件，它默认只输出可加载的已初始化数据段中的字符串，对于其他文件，它扫描整个文件
strip	移除目标文件中的符号
`libbfd`	二进制文件描述符库
`libctf`	紧凑 ANSI-C 类型格式调试支持库
`libctf-nobfd`	libctf 的变体，它不需要 libbfd 的功能
`libopcodes`	一个用于处理操作码 —— 处理器指令的 “可读文本”版本的库； 它被 **objdump** 等构建工具所使用

8.19. GMP-6.2.0

GMP 软件包包含提供任意精度算术函数的数学库。

估计构建时间: 1.1 SBU
需要硬盘空间: 51 MB

8.19.1. 安装 GMP

注意

如果您在为 32 位 x86 构建 LFS，但您的 CPU 能够运行 64 位代码，而且 您指定了 CFLAGS 环境变量，配置脚本会试图为 64 位 CPU 进行配置并且失败。为了避免这个问题，像下面这样执行 configure 命令:

```
ABI=32 ./configure ...
```

注意

GMP 的默认设定会生成为本机处理器优化的库。如果您希望获得适合功能没有本机强大的 CPU 的库，执行以下命令，以生成通用库:

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

准备编译 GMP:

```
./configure --prefix=/usr    \
            --enable-cxx     \
            --disable-static \
            --docdir=/usr/share/doc/gmp-6.2.0
```

新的配置选项的含义:

--enable-cxx
该参数启用 C++ 支持

--docdir=/usr/share/doc/gmp-6.2.0
该变量指定文档的正确位置

编译该软件包，并生成 HTML 文档:

```
make
make html
```

重要

我们认为，本节中 GMP 的测试套件被认为是关键的。无论如何都不要跳过测试过程。

测试编译结果:

```
make check 2>&1 | tee gmp-check-log
```

小心

GMP 中的代码是针对本机处理器高度优化的。在偶然情况下，检测处理器的代码会错误识别 CPU 的功能，导致测试套件或使用 GMP 的其他程序输出消息 “Illegal instruction” (非法指令)。如果发生这种情况，需要加入选项 --build=x86_64-unknown-linux-gnu 并重新构建 GMP。

务必确认测试套件中的 197 个测试全部通过。运行以下命令检验结果：

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

安装该软件包及其文档：

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

8.19.2. GMP 的内容

安装的库: libgmp.so 和 libgmpxx.so
安装的目录: /usr/share/doc/gmp-6.2.0

简要描述

libgmp	包含任意精度数学函数
libgmpxx	包含 C++ 任意精度数学函数

8.20. MPFR-4.0.2

MPFR 软件包包含多精度数学函数。

估计构建时间: 0.8 SBU
需要硬盘空间: 36 MB

8.20.1. 安装 MPFR

准备编译 MPFR:

```
./configure --prefix=/usr        \
            --disable-static     \
            --enable-thread-safe \
            --docdir=/usr/share/doc/mpfr-4.0.2
```

编译该软件包，并生成 HTML 文档:

```
make
make html
```

重要

本节中 MPFR 的测试套件被认为是非常关键的，无论如何不能跳过。

测试编译结果，并确认所有测试都能通过:

```
make check
```

安装该软件包及其文档:

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

8.20.2. MPFR 的内容

安装的库: libmpfr.so
安装的目录: /usr/share/doc/mpfr-4.0.2

简要描述

`libmpfr` 包含多精度数学函数

8.21. MPC-1.1.0

MPC 软件包包含一个任意高精度，且舍入正确的复数算术库。

估计构建时间: 0.3 SBU
需要硬盘空间: 21 MB

8.21.1. 安装 MPC

准备编译 MPC:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/mpc-1.1.0
```

编译该软件包，并生成 HTML 文档:

```
make
make html
```

运行以下命令以测试编译结果:

```
make check
```

安装该软件包

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

8.21.2. MPC 的内容

安装的库: libmpc.so
安装的目录: /usr/share/doc/mpc-1.1.0

简要描述

`libmpc` 包含复数数学运算函数

8.22. Attr-2.4.48

Attr 软件包包含管理文件系统对象扩展属性的工具。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 4.2 MB

8.22.1. 安装 Attr

准备编译 Attr:

```
./configure --prefix=/usr     \
            --disable-static  \
            --sysconfdir=/etc \
            --docdir=/usr/share/doc/attr-2.4.48
```

编译该软件包:

```
make
```

测试套件必须在支持扩展属性的文件系统，如 ext2、ext3 或 ext4 上运行。运行下列命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

需要将共享库移动到 /lib 目录，因此 /usr/lib 中的 .so 符号链接也需要重新建立:

```
mv -v /usr/lib/libattr.so.* /lib
ln -sfv ../../lib/$(readlink /usr/lib/libattr.so) /usr/lib/libattr.so
```

8.22.2. Attr 的内容

安装的程序: attr, getfattr, 以及 setfattr
安装的库: libattr.so
安装的目录: /usr/include/attr 和 /usr/share/doc/attr-2.4.48

简要描述

attr	在文件系统对象上扩展属性
getfattr	查询文件系统对象的扩展属性
setfattr	设定文件系统对象的扩展属性
libattr	包含处理扩展属性的库函数

8.23. Acl-2.2.53

Acl 软件包包含管理访问控制列表的工具，访问控制列表能够更细致地自由定义文件和目录的访问权限。

估计构建时间: 0.1 SBU
需要硬盘空间: 6.2 MB

8.23.1. 安装 Acl

准备编译 Acl:

```
./configure --prefix=/usr         \
            --disable-static      \
            --libexecdir=/usr/lib \
            --docdir=/usr/share/doc/acl-2.2.53
```

编译该软件包:

```
make
```

Acl 的测试套件必须在构建了链接到 Acl 库的 Coreutils 后才能在支持访问控制的文件系统上运行。如果想运行它们，在构建好 Coreutils 后再返回这里，并执行 **make check**。

安装该软件包:

```
make install
```

共享库需要被移动到 /lib 目录，因此 /usr/lib 中的 .so 符号链接需要重新建立:

```
mv -v /usr/lib/libacl.so.* /lib
ln -sfv ../../lib/$(readlink /usr/lib/libacl.so) /usr/lib/libacl.so
```

8.23.2. Acl 的内容

安装的程序: chacl, getfacl, 以及 setfacl
安装的库: libacl.so
安装的目录: /usr/include/acl 和 /usr/share/doc/acl-2.2.53

简要描述

chacl 修改文件或目录的访问控制列表
getfacl 获取文件访问控制列表
setfacl 设定文件访问控制列表
libacl 包含操作访问控制列表的库函数

8.24. Libcap-2.36

Libcap 软件包为 Linux 内核提供的 POSIX 1003.1e 权能字实现用户接口。这些权能字是 root 用户的最高特权分割成的一组不同权限。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 11 MB

8.24.1. 安装 Libcap

防止一个静态库的安装:

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

编译该软件包:

```
make lib=lib
```

make 命令选项的含义:

`lib=lib`
在 x86_64 上，该参数将库文件目录设定为 `/lib`，而不是 `/lib64`。它在 x86 上没有作用。

运行以下命令以测试编译结果:

```
make test
```

安装该软件包，并进行清理工作:

```
make lib=lib PKGCONFIGDIR=/usr/lib/pkgconfig install
chmod -v 755 /lib/libcap.so.2.36
mv -v /lib/libpsx.a /usr/lib
rm -v /lib/libcap.so
ln -sfv ../../lib/libcap.so.2 /usr/lib/libcap.so
```

8.24.2. Libcap 的内容

安装的程序: capsh, getcap, getpcaps, 以及 setcap
安装的库: libcap.so 和 libpsx.a

简要描述

capsh	一个用于演示和限制 Linux 权能字的 shell 封装器
getcap	检验文件权能字
getpcaps	查询进程的权能字
setcap	设定文件权能字
`libcap`	包含操作 POSIX 1003.1e 权能字的库函数
`libpsx`	包含为 pthread 库相关的系统调用提供 POSIX 语义的函数

8.25. Shadow-4.8.1

Shadow 软件包包含安全地处理密码的程序。

估计构建时间: 0.2 SBU
需要硬盘空间: 45 MB

8.25.1. 安装 Shadow

注意

如果您希望强制使用强密码，参考 http://www.linuxfromscratch.org/blfs/view/svn/postlfs/cracklib.html 以在构建 Shadow 前安装 CrackLib。然后，将 `--with-libcrack` 添加到下面的 **configure** 命令中。

禁止该软件包安装 **groups** 程序和它的 man 页面，因为 Coreutils 会提供更好的版本。同样，避免安装第 8.3 节 “Man-pages-5.07”软件包已经提供的 man 页面：

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

不使用默认的 crypt 加密方法，使用更安全的 SHA-512 方法加密密码，该方法也允许长度超过 8 个字符的密码。另外，还需要把过时的用户邮箱位置 `/var/spool/mail` 改为当前普遍使用的 `/var/mail` 目录：

```
sed -e 's:#ENCRYPT_METHOD DES:ENCRYPT_METHOD SHA512:' \
    -e 's:/var/spool/mail:/var/mail:'                 \
    -i etc/login.defs
```

注意

如果您选择构建有 Cracklib 支持的 Shadow，执行以下命令：

```
sed -i 's:DICTPATH.*:DICTPATH\t/lib/cracklib/pw_dict:' etc/login.defs
```

进行微小的改动，使 useradd 使用 1000 作为第一个组编号：

```
sed -i 's/1000/999/' etc/useradd
```

准备编译 Shadow：

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

配置选项的含义：

touch /usr/bin/passwd
我们需要保证 `/usr/bin/passwd` 存在，因为它的位置会被硬编码到一些程序中，如果它不存在的话，会使用错误的默认位置。

`--with-group-name-max-length=32`
最长用户名可以有 32 个字符。设定组名称最大长度为相同值。

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
```

8.25.2. 配置 Shadow

该软件包包含用于添加、修改、删除用户和组，设定和修改它们的密码，以及进行其他管理任务的工具。如果希望查阅关于 password shadowing 的详细解释，阅读解压得到源代码目录树中的 `doc/HOWTO` 文件。如果使用 Shadow 支持，请注意所有需要验证密码的程序 (如显示管理器、FTP 程序、pop3 守护进程等) 都必须和 Shadow 兼容。换句话说，它们必须能使用 Shadow 加密的密码。

如果要对用户密码启用 Shadow 加密，执行以下命令：

```
pwconv
```

如果要对组密码启用 Shadow 加密，执行：

```
grpconv
```

Shadow 附带的 **useradd** 配置文件有一些需要解释的事项。首先，**useradd** 的默认操作是创建一个用户，以及一个名字和用户名相同的组。默认情况下，用户 ID (UID) 和组 ID (GID) 会从 1000 开始。这意味着，如果您不向 **useradd** 传递参数，每个用户都会属于一个不同的组。如果您不希望这样，就要传递 `-g` 参数给 **useradd**。默认参数保存在 `/etc/default/useradd` 文件中。您可以编辑其中的两个参数，以满足您的特定需求。

/etc/default/useradd 参数解释

`GROUP=1000`
该参数设定 /etc/group 文件中使用的第一个组编号。您可以将它修改为您希望的任何值。注意，**useradd** 绝不会重用 UID 或 GID。如果该参数指定的数字已经被使用了，它就会使用下一个可用的数字。另外，如果您第一次使用 **useradd** 时没有编号 1000 的组，也没有使用 `-g` 选项，您就会在终端看到一条消息：`useradd: unknown GID 1000`。您可以忽略这条消息，它会使用组编号 1000。

`CREATE_MAIL_SPOOL=yes`
该参数使得 **useradd** 为新创建的用户建立邮箱文件。**useradd** 会使得 `mail` 为邮箱文件属组，并为邮箱文件赋予 0660 权限码。如果您不希望 **useradd**创建这些邮箱文件，执行以下命令：

```
sed -i 's/yes/no/' /etc/default/useradd
```

8.25.3. 设定 root 密码

为用户 root 选择一个密码，并执行以下命令设定它：

```
passwd root
```

8.25.4. Shadow 的内容

安装的程序: chage, chfn, chgpasswd, chpasswd, chsh, expiry, faillog, gpasswd, groupadd, groupdel, groupmems, groupmod, grpck, grpconv, grpunconv, lastlog, login, logoutd, newgidmap, newgrp, newuidmap, newusers, nologin, passwd, pwck, pwconv, pwunconv, sg (到 newgrp 的链接), su, useradd, userdel, usermod, vigr (到 vipw 的链接), 以及 vipw

安装的目录: /etc/default

简要描述

chage 用于修改强制性密码更新的最大天数

chfn 用于修改用户全名和其他信息

chgpasswd 用于批量更新组密码

chpasswd 用于批量更新用户密码

chsh 用于改变用户的默认登录 shell

expiry 检查并强制保证当前密码过期策略

faillog 检查失败登录日志，设定锁定账户的最大失败次数，或重置失败次数

gpasswd 用于增加或删除组的用户和管理员

groupadd 以指定名称创建组

groupdel 删除指定的组

groupmems 允许用户在不需要超级用户权限的情况下，管理自己的组成员列表

groupmod 用于修改给定的组名称或 GID

grpck 验证组文件 /etc/group 和 /etc/gshadow 的完整性

grpconv 根据普通组文件创建或更新加密组文件

grpunconv 根据 /etc/gshadow 文件更新 /etc/group 文件，并删除前者

lastlog 报告所有用户或给定用户最后一次登录的信息

login 被系统用于允许用户登录

logoutd 是一个限制登录时间和端口的守护进程

newgidmap 用于设定用户命名空间的 gid 映射

newgrp 用于在登录会话中修改当前 GID

newuidmap 用于设定用户命名空间的 uid 映射

newusers 用于批量创建或更新用户账户

nologin 显示一条账户不可用的消息；它被设计为用来当作被禁用的账户的默认 shell

passwd 用于修改用户或组账户的密码

pwck 检验密码文件 /etc/passwd 和 /etc/shadow 的完整性

pwconv 从普通密码文件创建或更新加密密码文件

pwunconv 根据 /etc/shadow 更新 /etc/passwd 并删除前者

sg 在用户 GID 设为给定组 ID 的情况下，执行给定命令

su	用替换的用户和组 ID 运行 shell
useradd	以指定名称创建新用户，或更新新用户默认信息
userdel	删除给定用户
usermod	修改给定用户的登录名称、用户标识符 (UID)、shell、初始组、home 目录等信息
vigr	编辑 /etc/group 或 /etc/gshadow 文件
vipw	编辑 /etc/passwd 或 /etc/shadow 文件

8.26. GCC-10.1.0

GCC 软件包包含 GNU 编译器集合，其中有 C 和 C++ 编译器。

估计构建时间: 103 SBU (with tests)
需要硬盘空间: 4.4 GB

8.26.1. 安装 GCC

在 x86_64 上构建时，修改存放 64 位库的默认路径为 “lib”：

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

GCC 文档推荐在专用的构建目录中构建 GCC:

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

准备编译 GCC:

```
../configure --prefix=/usr            \
             LD=ld                    \
             --enable-languages=c,c++ \
             --disable-multilib       \
             --disable-bootstrap      \
             --with-system-zlib
```

请注意，对于其他语言，还有一些尚未满足的依赖项。阅读BLFS 手册，以了解如何构建 GCC 支持的所有语言。

新的配置选项的含义:

`LD=ld`
该选项使得配置脚本使用之前在本章中构建的 ld，而没有该选项时会使用交叉编译构建的版本。

`--with-system-zlib`
该选项使得 GCC 链接到系统安装的 Zlib 库，而不是它自带的 Zlib 副本。

编译该软件包:

```
make
```

重要

本节中 GCC 的测试套件被认为是关键的，无论如何不能跳过。

已知 GCC 测试套件中的一组测试可能耗尽默认栈空间，因此运行测试前要增加栈空间:

```
ulimit -s 32768
```

以非特权用户身份测试编译结果，但出错时继续执行其他测试：

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

输入以下命令查看测试结果的摘要：

```
../contrib/test_summary
```

如果只想看摘要，将输出用管道送至 **grep -A7 Summ**。

可以将结果与 http://www.linuxfromscratch.org/lfs/build-logs/development/ 和 https://gcc.gnu.org/ml/gcc-testresults/ 的结果进行比较。

已知有 6 个关于 get_time 的测试会失败。它们似乎与 en_HK locale 有关。

少量意外的失败有时无法避免，GCC 开发者一般知道这类问题，但尚未解决它们。我们可以继续安全地构建系统，除非测试结果和以上 URL 的结果截然不同。

安装该软件包，并移除一个不需要的目录：

```
make install
rm -rf /usr/lib/gcc/$(gcc -dumpmachine)/10.1.0/include-fixed/bits/
```

GCC 构建目录目前属于用户 tester，这会导致安装的头文件目录 (及其内容) 具有不正确的所有权。将所有者修改为 root 用户和组：

```
chown -v -R root:root \
    /usr/lib/gcc/*linux-gnu/10.1.0/include{,-fixed}
```

创建一个 FHS 因 “历史原因” 要求的符号链接。

```
ln -sv ../usr/bin/cpp /lib
```

创建一个兼容性符号链接，以支持在构建程序时使用链接时优化 (LTO)：

```
install -v -dm755 /usr/lib/bfd-plugins
ln -sfv ../../libexec/gcc/$(gcc -dumpmachine)/10.1.0/liblto_plugin.so \
        /usr/lib/bfd-plugins/
```

现在最终的工具链已经就位，重要的是再次确认编译和链接像我们期望的一样正常工作。我们通过进行一些完整性检查，进行确认：

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

There should be no errors, and the output of the last command will be (allowing for platform-specific differences in the dynamic linker name):

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

Now make sure that we're setup to use the correct start files:

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

The output of the last command should be:

```
/usr/lib/gcc/x86_64-pc-linux-gnu/10.1.0/../../../../lib/crt1.o succeeded
/usr/lib/gcc/x86_64-pc-linux-gnu/10.1.0/../../../../lib/crti.o succeeded
/usr/lib/gcc/x86_64-pc-linux-gnu/10.1.0/../../../../lib/crtn.o succeeded
```

以上结果可能随您的机器体系结构不同而略微不同。差异在于 /usr/lib/gcc 之后的目录名。我们关注的重点是，gcc 应该找到所有三个 crt*.o 文件，它们应该位于 /usr/lib 目录中。

Verify that the compiler is searching for the correct header files:

```
grep -B4 '^ /usr/include' dummy.log
```

This command should return the following output:

```
#include <...> search starts here:
 /usr/lib/gcc/x86_64-pc-linux-gnu/10.1.0/include
 /usr/local/include
 /usr/lib/gcc/x86_64-pc-linux-gnu/10.1.0/include-fixed
 /usr/include
```

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

Next, verify that the new linker is being used with the correct search paths:

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

References to paths that have components with '-linux-gnu' should be ignored, but otherwise the output of the last command should be:

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

在 32 位系统上可能显示一些不同的目录。例如，下面是 i686 机器上的输出：

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

Next make sure that we're using the correct libc:

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

The output of the last command should be:

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

Make sure GCC is using the correct dynamic linker:

```
grep found dummy.log
```

The output of the last command should be (allowing for platform-specific differences in dynamic linker name):

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

If the output does not appear as shown above or is not received at all, then something is seriously wrong. Investigate and retrace the steps to find out where the problem is and correct it. Any issues will need to be resolved before continuing with the process.

Once everything is working correctly, clean up the test files:

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

最后移动一个位置不正确的文件：

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

8.26.2. GCC 的内容

安装的程序:	c++, cc (到 gcc 的链接), cpp, g++, gcc, gcc-ar, gcc-nm, gcc-ranlib, gcov, gcov-dump, 以及 gcov-tool
安装的库:	libasan.{a,so}, libatomic.{a,so}, libcc1.so, libgcc.a, libgcc_eh.a, libgcc_s.so, libgcov.a, libgomp.{a,so}, libitm.{a,so}, liblsan.{a,so}, liblto_plugin.so, libquadmath.{a,so}, libssp.{a,so}, libssp_nonshared.a, libstdc++.{a,so}, libstdc++fs.a, libsupc++.a, libtsan.{a,so}, 以及 libubsan.{a,so}
安装的目录:	/usr/include/c++, /usr/lib/gcc, /usr/libexec/gcc, 以及 /usr/share/gcc-10.1.0

简要描述

c++	C++ 编译器
cc	C 编译器
cpp	C 预处理器，编译器使用它展开源文件中的 #include、#define 及类似指令
g++	C++ 编译器
gcc	C 编译器
gcc-ar	**ar** 的一个包装器，它在命令行中添加一个插件。这个程序只被用于提供链接时优化功能，对于默认的构建选项来说没有作用。
gcc-nm	**nm** 的一个包装器，它在命令行中添加一个插件。这个程序只被用于提供链接时优化功能，对于默认的构建选项来说没有作用。
gcc-ranlib	**ranlib** 的一个包装器，它在命令行中添加一个插件。这个程序只被用于提供链接时优化功能，对于默认的构建选项来说没有作用。

gcov	一个覆盖率测试工具；用于分析程序并确定在哪里优化最有效
gcov-dump	离线 gcda 和 gcno 性能剖析数据显示工具
gcov-tool	离线 gcda 性能剖析预处理工具
libasan	地址完整性检查库
libatomic	GCC 内建原子操作运行库
libcc1	C 预处理库
libgcc	包含 **gcc** 的运行时支持
libgcov	在 GCC 被指示启动性能剖析时，这个库被链接到程序中
libgomp	OpenMP API 的 GNU 实现，用于 C/C++ 和 Fortran 的跨平台共享内存并行编程
liblsan	内存泄露清理检查库
liblto_plugin	GCC 的链接时优化 (LTO) 插件，使得 GCC 可以进行跨越编译单元的优化
libquadmath	GCC 四精度数学 API 库
libssp	包含 GCC 的栈溢出保护功能支持子程序
libstdc++	C++ 标准库
libstdc++fs	ISO/IEC TS 18822:2015 文件系统库
libsupc++	包含 C++ 编程语言支持子程序
libtsan	线程完整性检查库
libubsan	未定义行为清理检查库

8.27. Pkg-config-0.29.2

pkg-config 软件包提供一个在软件包安装的配置和编译阶段，向构建工具传递头文件和/或库文件路径的工具。

估计构建时间: 0.4 SBU
需要硬盘空间: 30 MB

8.27.1. 安装 Pkg-config

准备编译 Pkg-config:

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

新的配置选项的含义:

`--with-internal-glib`
该选项允许 pkg-config 使用它内部的 Glib 版本，因为 LFS 不提供外部的 Glib。

`--disable-host-tool`
该选项防止创建一个指向 pkg-config 程序的不需要的硬链接。

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.27.2. Pkg-config 的内容

安装的程序: pkg-config
安装的目录: /usr/share/doc/pkg-config-0.29.2

简要描述

pkg-config 返回某个库或软件包的元数据信息

8.28. Ncurses-6.2

Ncurses 软件包包含终端无关的字符屏幕处理库。

估计构建时间: 0.4 SBU
需要硬盘空间: 32 MB

8.28.1. 安装 Ncurses

输入以下命令，使构建系统不安装一个 configure 脚本未处理的静态库:

```
sed -i '/LIBTOOL_INSTALL/d' c++/Makefile.in
```

准备编译 Ncurses:

```
./configure --prefix=/usr           \
            --mandir=/usr/share/man \
            --with-shared           \
            --without-debug         \
            --without-normal        \
            --enable-pc-files       \
            --enable-widec
```

新的配置选项的含义:

`--enable-widec`
该选项使得宽字符库 (例如 `libncursesw.so.6.2`) 被构建，而不构建常规字符库 (例如 `libncurses.so.6.2`)。宽字符库在多字节和传统 8 位 locale 中都能工作，而常规字符库只能在 8 位 locale 中工作。宽字符库和普通字符库在源码层面是兼容的，但二进制不兼容。

`--enable-pc-files`
该参数使得构建系统生成并安装 pkg-config 使用的 .pc 文件。

`--without-normal`
该选项禁止构建系统编译并安装多数静态库。

编译该软件包:

```
make
```

该软件包有测试套件，但只能在安装该软件包后才能运行。测试用例位于 `test/` 中。阅读其中的 `README` 文件了解更多细节。

安装该软件包:

```
make install
```

将共享库移动到期望的 `/lib` 目录:

```
mv -v /usr/lib/libncursesw.so.6* /lib
```

由于共享库被移走了，一个符号链接指向了不存在的文件。重新创建它:

```
ln -sfv ../../lib/$(readlink /usr/lib/libncursesw.so) /usr/lib/libncursesw.so
```

许多程序仍然希望链接器能够找到非宽字符版本的 Ncurses 库。通过使用符号链接和链接脚本，诱导它们链接到宽字符库：

```
for lib in ncurses form panel menu ; do
    rm -vf                    /usr/lib/lib${lib}.so
    echo "INPUT(-l${lib}w)" > /usr/lib/lib${lib}.so
    ln -sfv ${lib}w.pc        /usr/lib/pkgconfig/${lib}.pc
done
```

最后，确保那些在构建时寻找 -lcurses 的老式程序仍然能够构建：

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

如果需要的话，安装 Ncurses 文档：

```
mkdir -v       /usr/share/doc/ncurses-6.2
cp -v -R doc/* /usr/share/doc/ncurses-6.2
```

注意

上述指令没有创建非宽字符的 Ncurses 库，因为从源码编译的软件包不会在运行时链接到它。然而，已知的需要链接到非宽字符 Ncurses 库的二进制程序都需要版本 5。如果您为了满足一些仅有二进制版本的程序，或者满足 LSB 兼容性，必须安装这样的库，执行以下命令再次构建该软件包：

```
make distclean
./configure --prefix=/usr    \
            --with-shared    \
            --without-normal \
            --without-debug  \
            --without-cxx-binding \
            --with-abi-version=5
make sources libs
cp -av lib/lib*.so.5* /usr/lib
```

8.28.2. Ncurses 的内容

安装的程序:	captoinfo (链接到 tic), clear, infocmp, infotocap (链接到 tic), ncursesw6-config, reset (链接到 tset), tabs, tic, toe, tput, 以及 tset
安装的库:	libcursesw.so (指向 libncursesw.so 的符号链接和链接脚本), libformw.so, libmenuw.so, libncursesw.so, libncurses++w.a, libpanelw.so, 以及它们的库名称没有 “w” 的非宽字符替代品。
安装的目录:	/usr/share/tabset, /usr/share/terminfo, 以及 /usr/share/doc/ncurses-6.2

简要描述

captoinfo	将 termcap 描述转换成 terminfo 描述
clear	如果可能的话，清空屏幕
infocmp	比较或输出 terminfo 描述

infotocap	将 terminfo 描述转化为 termcap 描述
ncursesw6-config	提供 ncurses 的配置信息
reset	以终端默认值重新初始化终端
tabs	清除并设置终端的制表符宽度
tic	Terminfo 条目描述编译器，将 terminfo 文件从源代码格式翻译为 ncurses 库子程序需要的二进制格式 [terminfo 文件包含特定终端的功能信息。]
toe	列出所有可用的终端类型，并给出每种类型的主要名称和描述
tput	使 shell 可以使用终端相关的功能；也可以重置或初始化终端，或者报告它的长名称
tset	可以被用于初始化终端
`libcursesw`	指向 `libncursesw` 的链接
`libncursesw`	包含在终端屏幕上以多种复杂方式显示文本的函数；使用这些函数的典型例子是运行内核的 **make menuconfig** 时显示的目录
`libformw`	包含实现表单的函数
`libmenuw`	包含实现目录的函数
`libpanelw`	包含实现面板的函数

8.29. Sed-4.8

Sed 软件包包含一个流编辑器。

估计构建时间: 0.5 SBU
需要硬盘空间: 32 MB

8.29.1. 安装 Sed

准备编译 Sed:

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

编译该软件包，并生成 HTML 文档:

```
make
make html
```

运行以下命令以测试编译结果:

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

安装该软件包及其文档:

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

8.29.2. Sed 的内容

安装的程序: sed
安装的目录: /usr/share/doc/sed-4.8

简要描述

sed 一次性过滤和转换文本文件

8.30. Psmisc-23.3

Psmisc 软件包包含显示正在运行的进程信息的程序。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 4.7 MB

8.30.1. 安装 Psmisc

准备编译 Psmisc:

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

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
```

最后，移动 **killall** 和 **fuser** 到 FHS 指定的位置:

```
mv -v /usr/bin/fuser   /bin
mv -v /usr/bin/killall /bin
```

8.30.2. Psmisc 的内容

安装的程序: fuser, killall, peekfd, prtstat, pslog, pstree, 以及 pstree.x11 (到 pstree 的链接)

简要描述

fuser	报告使用给定文件或文件系统的进程 ID (PID)
killall	根据名称杀死进程；它向所有运行给定命令的进程发送信号
peekfd	根据给定 PID，查看正在运行进程的文件描述符
prtstat	打印某个进程的信息
pslog	报告某个进程当前使用的日志路径
pstree	以树形格式列出正在运行的进程
pstree.x11	除了在退出前等待用户确认外，和 **pstree**相同

8.31. Gettext-0.20.2

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

估计构建时间: 2.8 SBU
需要硬盘空间: 231 MB

8.31.1. 安装 Gettext

准备编译 Gettext:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/gettext-0.20.2
```

编译该软件包:

```
make
```

输入以下命令以测试编译结果 (需要较长时间, 大约 3 SBU):

```
make check
```

安装该软件包:

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

8.31.2. Gettext 的内容

安装的程序: autopoint, envsubst, gettext, gettext.sh, gettextize, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, recode-sr-latin, 以及 xgettext
安装的库: libasprintf.so, libgettextlib.so, libgettextpo.so, libgettextsrc.so, libtextstyle.so, 以及 preloadable_libintl.so
安装的目录: /usr/lib/gettext, /usr/share/doc/gettext-0.20.2, /usr/share/gettext, 以及 /usr/share/gettext-0.19.8

简要描述

autopoint	将标准 Gettext 微架构文件复制到源代码包
envsubst	替换 shell 格式化字符串中的环境变量
gettext	通过查询消息目录中的翻译，将中性语言消息翻译成用户的语言
gettext.sh	主要用于 gettext 的 shell 函数库
gettextize	将所有标准 Gettext 文件复制到软件包顶层目录中，以开始国际化该软件包
msgattrib	根据属性过滤翻译目录中的消息，或修改这些属性
msgcat	连接并合并给定的 .po 文件

msgcmp	比较两个 .po 文件，以检查它们是否包含相同的 msgid 字符串集合
msgcomm	找出给定的多个 .po 中的公共消息
msgconv	将翻译目录转换成另一种字符编码
msgen	创建英文翻译目录
msgexec	对翻译目录中的所有翻译执行命令
msgfilter	对翻译目录中的所有翻译应用过滤器
msgfmt	根据翻译目录创建二进制消息目录
msggrep	找出翻译目录中所有匹配给定模式，或属于给定源代码文件的消息
msginit	创建一个新的 .po 文件，以用户环境中的值初始化其元信息
msgmerge	将两个原始翻译文件组合成一个文件
msgunfmt	反编译二进制消息目录，生成原始翻译文本
msguniq	去除翻译目录中重复的翻译
ngettext	显示某条语法形式依赖于数字的文本消息的母语翻译
recode-sr-latin	将塞尔维亚语文本从西里尔字符转换为拉丁字符
xgettext	从给定源代码文件中提取可翻译消息，以生成最初的翻译模板
libasprintf	定义 autosprintf 类，使得 C 格式化输出子程序在 C++ 程序中可用，能够与 <string> 字符串和 <iostream> 流一起使用
libgettextlib	一个内部库，包含若干 Gettext 程序的公共子程序； 不建议普遍使用
libgettextpo	用于编写处理 .po 文件的专用程序； 这个库在 Gettext 发行的标准程序 (如 **msgcomm**, **msgcmp**, **msgattrib**, 以及 **msgen**) 不能满足要求时使用
libgettextsrc	一个内部库，包含若干 Gettext 程序使用的公共子程序； 没有设计为普遍使用
libtextstyle	文本样式库
preloadable_libintl	一个被设计为由 LD_PRELOAD 预加载的库，帮助 libintl 记录未翻译的消息

8.32. Bison-3.6.4

Bison 软件包包含语法分析器生成器。

估计构建时间: 5.6 SBU
需要硬盘空间: 50 MB

8.32.1. 安装 Bison

准备编译 Bison：

```
./configure --prefix=/usr --docdir=/usr/share/doc/bison-3.6.4
```

编译该软件包：

```
make
```

已知测试在使用多个处理器时可能失败。如果要测试编译结果 (需要约 5.5 SBU)，运行命令：

```
make -j1 check
```

安装该软件包：

```
make install
```

8.32.2. Bison 的内容

安装的程序: bison 和 yacc
安装的库: liby.a
安装的目录: /usr/share/bison

简要描述

bison 根据一组规则，创建一个用于分析文本文件结构的程序； Bison 是 Yacc (Yet Another Compiler Compiler) 的替代品。

yacc **bison** 的一个封装器，被那些仍然调用 **yacc** 而非 **bison** 的程序使用，它调用 **bison** 时附加 `-y` 选项。

`liby` Yacc 库包含与 Yacc 兼容的 `yyerror` 和 `main` 函数实现；它并不是很有用，但 POSIX 需要它

8.33. Grep-3.4

Grep 软件包包含在文件内容中进行搜索的程序。

估计构建时间: 0.7 SBU
需要硬盘空间: 37 MB

8.33.1. 安装 Grep

准备编译 Grep:

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

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.33.2. Grep 的内容

安装的程序: egrep, fgrep, 以及 grep

简要描述

egrep 打印与扩展正则表达式匹配的行
fgrep 打印与固定字符串匹配的行
grep 打印与基本正则表达式匹配的行

8.34. Bash-5.0

Bash 软件包包含 Bourne-Again SHell。

估计构建时间: 1.8 SBU
需要硬盘空间: 48 MB

8.34.1. 安装 Bash

整合上游进行的一些修复:

```
patch -Np1 -i ../bash-5.0-upstream_fixes-1.patch
```

准备编译 Bash:

```
./configure --prefix=/usr                      \
            --docdir=/usr/share/doc/bash-5.0 \
            --without-bash-malloc              \
            --with-installed-readline
```

配置选项的含义:

`--with-installed-readline`
该选项告诉 Bash 使用系统中已经安装的 `readline` 库，而不是它自己的 readline 版本。

编译该软件包:

```
make
```

如果不运行测试套件，跳到 “安装该软件包”。

为了准备进行测试，确保 `tester` 用户可以写入源代码目录:

```
chown -Rv tester .
```

现在以 `tester` 用户的身份运行测试:

```
su tester << EOF
PATH=$PATH make tests < $(tty)
EOF
```

安装该软件包，并把主要的可执行文件移动到 `/bin`:

```
make install
mv -vf /usr/bin/bash /bin
```

执行新编译的 **bash** 程序 (替换当前正在执行的版本):

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

注意

上面使用的参数使得 **bash** 进程是一个可交互的登录 shell，并且仍然禁用散列功能，这样新程序一旦可用就会被找到。

8.34.2. Bash 的内容

安装的程序: bash, bashbug, 以及 sh (到 bash 的链接)
安装的目录: /usr/include/bash, /usr/lib/bash, 和 /usr/share/doc/bash-5.0

简要描述

bash 一个广泛使用的命令解释器； 它在执行命令前对命令行进行多种展开和替换操作，这些操作使得它成为强大的工具。

bashbug 一个 shell 脚本，用于帮助用户按照电子邮件标准格式编写关于 **bash** 的 bug 报告

sh 一个指向 **bash** 程序的符号链接； 当以 **sh** 命令运行时，**bash** 试图尽可能地模仿 **sh** 的历史版本，以符合 POSIX 标准

8.35. Libtool-2.4.6

Libtool 软件包包含 GNU 通用库支持脚本。它在一个一致、可移植的接口下隐藏了使用共享库的复杂性。

估计构建时间: 1.9 SBU
需要硬盘空间: 43 MB

8.35.1. 安装 Libtool

准备编译 Libtool:

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

编译该软件包:

```
make
```

为了测试编译结果，执行:

```
make check
```

注意

在多核系统上，可以显著减少 libtool 的测试时间。为了使用多个核心，在上述命令中附加 **TESTSUITEFLAGS=-j<N>** 参数。例如，使用 -j4 可以减少超过 60% 的测试时间。

在 LFS 构建环境中，已知有五个测试因为循环依赖而失败，但所有测试在 automake 安装后都能通过。

安装该软件包:

```
make install
```

8.35.2. Libtool 的内容

安装的程序: libtool 和 libtoolize
安装的库: libltdl.so
安装的目录: /usr/include/libltdl 和 /usr/share/libtool

简要描述

libtool	提供通用化库文件构建支持服务
libtoolize	提供为软件包增加 **libtool** 支持的标准方法
`libltdl`	隐藏用 dlopen 加载库时可能遇到的若干困难

8.36. GDBM-1.18.1

GDBM 软件包包含 GNU 数据库管理器。它是一个使用可扩展散列的数据库函数库，工作方法和标准 UNIX dbm 类似。该库提供用于存储键值对、通过键搜索和获取数据，以及删除键和对应数据的原语。

估计构建时间: 0.1 SBU
需要硬盘空间: 11 MB

8.36.1. 安装 GDBM

首先，修复一个 GCC-10 首先发现的问题：

```
sed -r -i '/^char.*parseopt_program_(doc|args)/d' src/parseopt.c
```

准备编译 GDBM：

```
./configure --prefix=/usr    \
            --disable-static \
            --enable-libgdbm-compat
```

配置选项的含义：

--enable-libgdbm-compat
该选项启用 libgdbm 兼容性库的构建。LFS 之外的一些软件包需要它提供的老式 DBM 子程序。

编译该软件包：

```
make
```

运行以下命令以测试编译结果：

```
make check
```

安装该软件包：

```
make install
```

8.36.2. GDBM 的内容

安装的程序: gdbm_dump, gdbm_load, 以及 gdbmtool
安装的库: libgdbm.so 和 libgdbm_compat.so

简要描述

gdbm_dump	将 GDBM 数据库转储到文件
gdbm_load	从转储文件重建 GDBM 数据库
gdbmtool	测试和修改 GDBM 数据库
libgdbm	包含用于操作散列数据库的函数
libgdbm_compat	包含老式 DBM 函数的兼容性库

8.37. Gperf-3.1

Gperf 根据一组键值，生成完美散列函数。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 6.4 MB

8.37.1. 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
```

已知同时执行多个测试 (-j 选项的值大于 1) 会导致测试失败。执行以下命令测试编译结果:

```
make -j1 check
```

安装该软件包:

```
make install
```

8.37.2. Gperf 的内容

安装的程序: gperf
安装的目录: /usr/share/doc/gperf-3.1

简要描述

gperf 根据键值集合生成完美散列函数

8.38. Expat-2.2.9

Expat 软件包包含用于解析 XML 文件的面向流的 C 语言库。

估计构建时间: 0.1 SBU
需要硬盘空间: 14 MB

8.38.1. 安装 Expat

准备编译 Expat:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/expat-2.2.9
```

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

如果需要，安装该软件包的文档:

```
install -v -m644 doc/*.{html,png,css} /usr/share/doc/expat-2.2.9
```

8.38.2. Expat 的内容

安装的程序: xmlwf
安装的库: libexpat.so
安装的目录: /usr/share/doc/expat-2.2.9

简要描述

xmlwf 是一个用于检验 XML 文档是否良好的非验证性工具

`libexpat` 包含解析 XML 文档的 API 函数

8.39. Inetutils-1.9.4

Inetutils 软件包包含基本网络程序。

估计构建时间: 0.3 SBU
需要硬盘空间: 29 MB

8.39.1. 安装 Inetutils

准备编译 Inetutils：

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

配置选项的含义：

`--disable-logger`
该选项防止 Inetutils 安装 **logger** 程序，它被脚本文件用于向系统日志守护程序传递消息。这里不安装它，因为 Util-linux 会安装更新的版本。

`--disable-whois`
该选项防止构建过时的 **whois** 客户端，BLFS 手册中有一个更好的 **whois** 客户端。

`--disable-r*`
这些参数禁用过时的程序，由于安全问题，它们不应该被继续使用。它们提供的功能可以由 BLFS 手册中的 openssh 软件包代替。

`--disable-servers`
该选项禁用 Inetutils 软件包包含的若干网络服务程序，它们在基本的 LFS 系统中注定是不合适的。其中一些服务程序从本质上就不安全，只有在可信的网络环境中才能被认为是安全的。要注意的是，对于其中许多服务程序，都能找到更好的替代品。

编译该软件包：

```
make
```

运行以下命令以测试编译结果：

```
make check
```

注意

其中一项测试 libls.sh 在初始的 chroot 环境中可能失败，但在 LFS 系统构建完成后再重新运行时即可通过。另外，一项名为 ping-localhost.sh 的测试在宿主系统不支持 ipv6 时会失败。

安装该软件包：

```
make install
```

移动一些程序，这样在 /usr 文件系统不可用时也能使用它们:

```
mv -v /usr/bin/{hostname,ping,ping6,traceroute} /bin
mv -v /usr/bin/ifconfig /sbin
```

8.39.2. Inetutils 的内容

安装的程序: dnsdomainname, ftp, ifconfig, hostname, ping, ping6, talk, telnet, tftp, 以及 traceroute

简要描述

dnsdomainname 显示系统的 DNS 域名

ftp 文件传输程序

hostname 报告或设定主机名称

ifconfig 管理网络接口

ping 发送回显请求数据包并报告响应用时

ping6 用于 IPv6 网络的 **ping** 版本

talk 用于和其他用户聊天

telnet TELNET 协议的接口

tftp 简单文件传输程序

traceroute 追踪您的数据包从您工作的主机到网络上另一台主机的路径，显示中间通过的跳跃 (网关)。

8.40. Perl-5.30.3

Perl 软件包包含实用报表提取语言。

估计构建时间: 8.5 SBU
需要硬盘空间: 240 MB

8.40.1. 安装 Perl

该版本的 Perl 会构建 Compress::Raw::ZLib 和 Compress::Raw::BZip2 模块。默认情况下 Perl 会使用内部的源码副本构建它们。执行以下命令，使得 Perl 使用系统中已经安装好的库：

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

为了能够完全控制 Perl 的设置，您可以在以下命令中移除 “-des” 选项，并手动选择构建该软件包的方式。或者，直接使用下面的命令，以使用 Perl 自动检测的默认值：

```
sh Configure -des -Dprefix=/usr                \
                  -Dvendorprefix=/usr          \
                  -Dman1dir=/usr/share/man/man1 \
                  -Dman3dir=/usr/share/man/man3 \
                  -Dpager="/usr/bin/less -isR"  \
                  -Duseshrplib                 \
                  -Dusethreads
```

配置选项的含义：

-Dvendorprefix=/usr
: 这保证 **perl** 知道如何告知软件包应该在哪里安装它们的 perl 模块。

-Dpager="/usr/bin/less -isR"
: 这保证该软件包使用**less** 对输出进行分页，而不是使用 **more**。

-Dman1dir=/usr/share/man/man1 -Dman3dir=/usr/share/man/man3
: 由于 Groff 还没有安装，**Configure** 认为我们不需要 Perl 的 man 页面。这些参数覆盖这个判断。

-Duseshrplib
: 构建 libperl 共享库，一些 perl 模块需要它。

-Dusethreads
: 构建带有线程支持的 perl。

编译该软件包：

```
make
```

为了测试编译结果 (需要约 11 SBU)，执行以下命令：

```
make test
```

安装该软件包，并清理环境变量：

```
make install
unset BUILD_ZLIB BUILD_BZIP2
```

8.40.2. Perl 的内容

安装的程序: corelist, cpan, enc2xs, encguess, h2ph, h2xs, instmodsh, json_pp, libnetcfg, perl, perl5.30.3 (指向 perl 的硬链接), perlbug, perldoc, perlivp, perlthanks (指向 perlbug 的硬链接), piconv, pl2pm, pod2html, pod2man, pod2text, pod2usage, podchecker, podselect, prove, ptar, ptardiff, ptargrep, shasum, splain, xsubpp, 以及 zipdetails
安装的库: 很多，无法在这里全部列出
安装的目录: /usr/lib/perl5

简要描述

corelist Module::CoreList 的命令行前端
cpan 通过命令行与 Perl 综合归档网络 (CPAN) 交互
enc2xs 从 Unicode 字符映射或 Tcl 编码文件构建 Encode 模块使用的 Perl 扩展
encguess 猜测一些文件的编码格式
h2ph 将 `.h` C 头文件转化为 `.ph` Perl 头文件
h2xs 将 `.h` C 头文件转化为 Perl 扩展
instmodsh 用于检验安装好的 Perl 模块的 shell 脚本，可以从安装好的模块创建压缩包
json_pp 在特定输入输出格式之间转化数据
libnetcfg 可以被用于配置 `libnet` Perl 模块
perl 由 C 语言、**sed**、**awk** 和 **sh** 的最好特性结合成的一门瑞士军刀式语言
perl5.30.3 指向 **perl** 的硬链接
perlbug 用于创建关于 Perl 或者它附带的模块的 bug 报告，并用邮件发送它们
perldoc 显示集成在 Perl 安装目录树或某个 Perl 脚本中的一页 pod 格式文档
perlivp Perl 安装检验程序；它可以被用于确认 Perl 和它的库都安装正确
perlthanks 用于生成发送给 Perl 开发者的感谢信
piconv 字符编码转换器 **iconv** 的 Perl 版本
pl2pm 一个用于将 Perl4 `.pl` 文件转换成 Perl5 `.pm` 模块的粗糙工具
pod2html 将 pod 格式的文件转换为 HTML 格式
pod2man 将 pod 数据转换为格式化的 *roff 输入
pod2text 将 pod 数据转化为格式化的 ASCII 文本
pod2usage 输出文件中嵌入的 pod 文档中的使用方法信息
podchecker 检查 pod 格式文档文件的语法
podselect 显示 pod 文档中的指定章节
prove 用于运行使用 Test::Harness 模块的测试
ptar 一个 Perl 编写的类似 **tar** 的程序
ptardiff 一个比较压缩档案和未压缩版本的 Perl 程序
ptargrep 一个在 tar 档案中的文件内容上进行模式匹配的 Perl 程序
shasum 打印或检查 SHA 校验和

splain	被用于 Perl 的强制性详细警告诊断
xsubpp	将 Perl XS 代码转换为 C 代码
zipdetails	显示 Zip 文件内部结构的详细信息

8.41. XML::Parser-2.46

XML::Parser 模块是 James Clark 的 XML 解析器 Expat 的 Perl 接口。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 2.4 MB

8.41.1. 安装 XML::Parser

准备编译 XML::Parser：

```
perl Makefile.PL
```

编译该软件包：

```
make
```

执行以下命令以测试编译结果：

```
make test
```

安装该软件包：

```
make install
```

8.41.2. XML::Parser 的内容

安装的模块: Expat.so

简要描述

`Expat` 提供 Expat 的 Perl 接口

8.42. Intltool-0.51.0

Intltool 是一个从源代码文件中提取可翻译字符串的国际化工具。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 1.5 MB

8.42.1. 安装 Intltool

首先修复由 perl-5.22 及更新版本导致的警告:

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

注意

上面使用的正则表达式由于大量的反斜线，看上去比较奇怪。它的功能是在序列 '\${' 的花括号之前增加一个反斜线，得到 '\$\{'。

准备编译 Intltool:

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

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

```
make check
```

安装该软件包:

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

8.42.2. Intltool 的内容

安装的程序: intltool-extract, intltool-merge, intltool-prepare, intltool-update, 以及 intltoolize
安装的目录: /usr/share/doc/intltool-0.51.0 和 /usr/share/intltool

简要描述

intltoolize	准备为软件包应用 intltool
intltool-extract	生成可供 **gettext** 读取的头文件
intltool-merge	将翻译好的字符串合并为多种类型的文件
intltool-prepare	更新 pot 文件并将它们和翻译文件合并
intltool-update	更新 po 模板文件并将它们和翻译文件合并

8.43. Autoconf-2.69

Autoconf 软件包包含生成能自动配置软件包的 shell 脚本的程序。

估计构建时间: 不到 0.1 SBU (计入测试为约 3.3 SBU)
需要硬盘空间: 79 MB

8.43.1. 安装 Autoconf

首先，修复 Perl 5.28 引入的 bug:

```
sed -i '361 s/{/\\{/' bin/autoscan.in
```

准备编译 Autoconf:

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

编译该软件包:

```
make
```

目前由于 bash-5 和 libtool-2.4.3 的变化，测试套件无法正常工作。如果无论如何要运行测试，执行命令:

```
make check
```

安装该软件包:

```
make install
```

8.43.2. Autoconf 的内容

安装的程序: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate, 以及 ifnames
安装的目录: /usr/share/autoconf

简要描述

autoconf 产生自动配置软件源码包，使其适用于多种类 Unix 系统的 shell 脚本；它产生的脚本可以独立运行 —— 运行它们不需要 **autoconf** 程序。

autoheader 一个创建 C #define 预处理指令的模板，以供配置脚本使用的程序

autom4te M4 宏处理器的封装器

autoreconf 在 **autoconf** 和 **automake** 模板文件发生变化时，按照正确顺序自动运行 **autoconf**、**autoheader**、**aclocal**、**automake**、**gettextize**, 以及 **libtoolize**，以便节省时间。

autoscan 帮助用户为软件包创建`configure.in` 文件； 它检验目录树中的源代码文件，在其中找出一般的移植性问题，然后创建一个 `configure.scan` 文件，作为软件包的原始 `configure.in` 文件

autoupdate 修改 `configure.in` 文件，将其中过时的 **autoconf** 宏名改为新的宏名。

ifnames 帮助用户为软件包编写 `configure.in`； 它打印软件包在 C 预处理器条件中使用的所有标识符 [如果一个软件包已经被设定为有一定的可移植性，该程序可以帮助确定 **configure** 需要进行哪些测试。它也会填充 **autoscan** 生成的 `configure.in` 中留下的空隙。]

8.44. Automake-1.16.2

Automake 软件包包含自动生成 Makefile，以便和 Autoconf 一同使用的程序。

估计构建时间: 不到 0.1 SBU (计入测试为约 8 SBU)
需要硬盘空间: 107 MB

8.44.1. 安装 Automake

修复一个失败的测试:

```
make -j4 check
```

准备编译 Automake:

```
./configure --prefix=/usr --docdir=/usr/share/doc/automake-1.16.2
```

编译该软件包:

```
make
```

由于单个测试点中存在的内部时延，即使仅有一个处理器，也应该使用 make 命令的 -j4 选项加速测试。输入以下命令测试编译结果:

```
make -j4 check
```

安装该软件包:

```
make install
```

8.44.2. Automake 的内容

安装的程序: aclocal, aclocal-1.16 (与 aclocal 互为硬链接), automake, 以及 automake-1.16 (与 automake 互为硬链接)
安装的目录: /usr/share/aclocal-1.16，/usr/share/automake-1.16，以及 /usr/share/doc/automake-1.16.2

简要描述

aclocal	根据 configure.in 文件内容生成 aclocal.m4。
aclocal-1.16	指向 **aclocal** 的硬链接
automake	一个根据 Makefile.am 文件，自动生成 Makefile.in 文件的工具 [为了创建软件包中的所有 Makefile.in 文件，在顶层目录运行该程序。它通过扫描 configure.in 文件，找到每个适用的 Makefile.am 文件，并生成对应的 Makefile.in 文件。]
automake-1.16	指向 **automake** 的硬链接

8.45. Kmod-27

Kmod 软件包包含用于加载内核模块的库和工具。

估计构建时间: 0.1 SBU
需要硬盘空间: 13 MB

8.45.1. 安装 Kmod

准备编译 Kmod:

```
./configure --prefix=/usr          \
            --bindir=/bin          \
            --sysconfdir=/etc      \
            --with-rootlibdir=/lib \
            --with-xz              \
            --with-zlib
```

配置选项的含义:

`--with-xz, --with-zlib`
它们允许 Kmod 处理压缩过的内核模块。

`--with-rootlibdir=/lib`
该选项保证一些和库有关的文件被放置在正确的目录中。

编译该软件包:

```
make
```

该软件包不包含能在 LFS chroot 环境下运行的测试套件。测试套件至少需要 git 程序的支持，且有些测试在 git 仓库外不会运行。

安装该软件包，并创建与 Module-Init-Tools (曾经用于处理 Linux 内核模块的软件包) 兼容的符号链接:

```
make install

for target in depmod insmod lsmod modinfo modprobe rmmod; do
  ln -sfv ../bin/kmod /sbin/$target
done

ln -sfv kmod /bin/lsmod
```

8.45.2. Kmod 的内容

安装的程序: depmod (到 kmod 的链接), insmod (到 kmod 的链接), kmod, lsmod (到 kmod 的链接), modinfo (到 kmod 的链接), modprobe (到 kmod 的链接), 以及 rmmod (到 kmod 的链接)
安装的库: libkmod.so

简要描述

depmod 根据现有模块的符号信息创建依赖关系文件；**modprobe** 使用依赖关系文件自动加载需要的模块

insmod 在正在运行的内核中安装可加载模块
kmod 加载或卸载内核模块
lsmod 列出当前加载的模块
modinfo 检验与某个内核模块相关的目标文件，打印它能够收集到的一切信息
modprobe 使用一个 **depmod** 创建的依赖关系文件，自动加载相关模块
rmmod 从正在运行的内核中卸载模块
`libkmod` 这个库被其他程序用于加载和卸载内核模块

8.46. Elfutils-0.180 中的 Libelf

Libelf 是一个处理 ELF (可执行和可链接格式) 文件的库。

估计构建时间: 1.0 SBU
需要硬盘空间: 121 MB

8.46.1. 安装 Libelf

Libelf 是 elfutils-0.180 软件包的一部分。请使用 elfutils-0.180.tar.bz2 作为源代码包。

准备编译 Libelf:

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

编译该软件包:

```
make
```

执行下列命令以测试编译结果:

```
make check
```

只安装 Libelf:

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

8.46.2. Libelf 的内容

安装的库: libelf.so
安装的目录: /usr/include/elfutils

8.47. Libffi-3.3

Libffi 库提供一个可移植的高级编程接口，用于处理不同调用惯例。这允许程序在运行时调用任何给定了调用接口的函数。

估计构建时间: 1.9 SBU
需要硬盘空间: 10 MB

8.47.1. 安装 Libffi

注意

和 GMP 类似，libffi 在构建时会使用特定于当前处理器的优化。如果是在为另一台计算机构建系统，请导出 CFLAGS 和 CXXFLAGS 环境变量，为您的架构指定较为通用的构建目标。否则，所有链接到 libffi 的程序都可能触发非法指令异常。

准备编译 libffi：

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

配置选项的含义：

`--with-gcc-arch=native`
保证 gcc 为当前系统进行优化。如果不使用该选项，构建系统会猜测系统架构，在某些系统上可能生成不正确的代码。如果要将生成的代码从本地系统复制到指令集功能较弱的系统中，需要使用目标系统架构作为该选项的参数值。关于不同系统架构的信息，参阅 gcc 手册中提供的的 x86 选项。

编译该软件包：

```
make
```

运行以下命令以测试编译结果：

```
make check
```

安装该软件包：

```
make install
```

8.47.2. Libffi 的内容

安装的库: libffi.so

简要描述

`libffi` 包含 libffi API 函数

8.48. OpenSSL-1.1.1g

OpenSSL 软件包包含密码学相关的管理工具和库。它们被用于向其他软件包提供密码学功能，例如 OpenSSH，电子邮件程序和 Web 浏览器 (以访问 HTTPS 站点)。

估计构建时间: 2.1 SBU
需要硬盘空间: 150 MB

8.48.1. 安装 OpenSSL

准备编译 OpenSSL:

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

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

```
make test
```

一项名为 30-test_afalg.t 的测试在某些内核配置下会失败 (它假定选择了一些未说明的内核配置选项)。

安装该软件包:

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

如果需要的话，安装文档:

```
mv -v /usr/share/doc/openssl /usr/share/doc/openssl-1.1.1g
cp -vfr doc/* /usr/share/doc/openssl-1.1.1g
```

8.48.2. OpenSSL 的内容

安装的程序: c_rehash 和 openssl
安装的库: libcrypto.{so,a} 和 libssl.{so,a}
安装的目录: /etc/ssl, /usr/include/openssl, /usr/lib/engines 以及 /usr/share/doc/openssl-1.1.1g

简要描述

c_rehash 一个 Perl 脚本，扫描一个目录中的所有文件，并添加它们的符号链接，符号链接名为对应文件的散列值。

openssl 一个命令行工具，用于从 shell 使用 OpenSSL的密码学库的一些密码学函数。它可以被用于 **man 1 openssl**描述的许多功能。

`libcrypto.so` 实现不同 Internet 标准使用的许多密码学算法。该库提供的服务被 OpenSSL 的 SSL、TLS 和 S/MIME 实现使用，也被用于实现 OpenSSH、OpenPGP，以及其他密码学标准。

`libssl.so` 实现传输层安全 (TLS v1) 协议。它提供了丰富的 API，这些 API 的文档可以通过执行 **man 3 ssl**查阅。

8.49. Python-3.8.3

Python 3 软件包包含 Python 开发环境。它被用于面向对象编程，编写脚本，为大型程序建立原型，或者开发完整的应用。

估计构建时间: 1.2 SBU
需要硬盘空间: 246 MB

8.49.1. 安装 Python 3

准备编译 Python:

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

配置选项的含义:

`--with-system-expat`
该选项允许链接到系统已经安装的 Expat。

`--with-system-ffi`
该选项允许链接到系统已经安装的 libffi。

`--with-ensurepip=yes`
该选项启用 **pip** 和 **setuptools** 包管理程序的构建。

编译该软件包:

```
make
```

运行 **make test** 以测试编译结果。一些需要网络连接或额外软件包的测试会被跳过。名为 test_normalization 的测试会由于网络配置不完整而失败。如果需要更全面的测试结果，可以在 BLFS 中重新安装 Python 3 时再次进行测试。

安装该软件包:

```
make install
chmod -v 755 /usr/lib/libpython3.8.so
chmod -v 755 /usr/lib/libpython3.so
ln -sfv pip3.8 /usr/bin/pip3
```

安装命令的含义:

chmod -v 755 /usr/lib/libpython3.{8.,}so
修正库文件访问权限，使之和其他库文件一致。

如果需要的话，安装预先格式化的文档：

```
install -v -dm755 /usr/share/doc/python-3.8.3/html

tar --strip-components=1  \
    --no-same-owner       \
    --no-same-permissions \
    -C /usr/share/doc/python-3.8.3/html \
    -xvf ../python-3.8.3-docs-html.tar.bz2
```

文档安装命令的含义：

`--no-same-owner` 和 `--no-same-permissions`
保证安装的文件拥有正确的所有者和权限码。在没有这些选项的时候，tar 会以上游开发者使用的用户和权限码安装文件。

8.49.2. Python 3 的内容

安装的程序: 2to3, idle3, pip3, pydoc3, python3, 以及 python3-config
安装的库: libpython3.8.so 和 libpython3.so
安装的目录: /usr/include/python3.8, /usr/lib/python3 以及 /usr/share/doc/python-3.8.3

简要描述

2to3 是一个 Python 程序，读取 Python 2.x 源代码并对它进行一系列修正，转换成合法的 Python 3.x 源代码。

idle3 一个封装脚本，启动支持 Python 语法的 GUI 文本编辑器。要运行这个脚本，必须在 Python 之前安装 Tk，从而构建 Tkinter Python 模块。

pip3 Python 包安装器。您可以使用 pip 安装来自 Python 软件包目录或其他目录的包。

pydoc3 是 Python 文档工具。

python3 是一个解释性、交互性、面向对象的程序设计语言。

8.50. Ninja-1.10.0

Ninja 是一个注重速度的小型构建系统。

估计构建时间: 0.3 SBU
需要硬盘空间: 78 MB

8.50.1. 安装 Ninja

在运行时，ninja 一般尽量并行运行更多进程。默认情况下最大进程数是系统 CPU 核心数加 2 得到的值。某些情况下，这样会导致 CPU 过热，或者耗尽系统内存。如果使用命令行执行 ninja，可以传递 -jN 参数以限制并行进程数，但某些软件包内嵌了 ninja 的执行过程，且并不传递 -j 参数。

应用下面这个可选的修改，用户即可通过一个环境变量 NINJAJOBS 限制并行进程数量。**例如**设置：

```
export NINJAJOBS=4
```

会限制 ninja 使用 4 个并行进程。

如果您希望 Ninja 能够使用环境变量 NINJAJOBS，执行以下命令，添加这一功能：

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

构建 Ninja：

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

构建选项的含义：

--bootstrap
这个参数强制 ninja 为当前系统重新构建自身。

运行以下命令以测试编译结果：

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

安装该软件包：

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

8.50.2. Ninja 的内容

安装的程序: ninja

简要描述

ninja 是 ninja 构建系统。

8.51. Meson-0.54.3

Meson 是一个开放源代码构建系统，它的设计保证了非常快的执行速度，和尽可能高的用户友好性。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 33 MB

8.51.1. 安装 Meson

执行以下命令编译 Meson：

```
python3 setup.py build
```

该软件包不包含测试套件。

安装该软件包：

```
python3 setup.py install --root=dest
cp -rv dest/* /
```

安装选项的含义：

`--root=dest`
默认情况下 **python3 setup.py install** 将若干文件 (如 man 页面) 安装到 Python Eggs 中。在指定了根目录位置时，**setup.py** 将这些文件安装到符合标准的目录树中。我们即可直接复制该目录树，使得这些文件位于标准指定的位置。

8.51.2. Meson 的内容

安装的程序: meson
安装的目录: /usr/lib/python3.8/site-packages/meson-0.54.3-py3.8.egg-info 和 /usr/lib/python3.8/site-packages/mesonbuild

简要描述

meson 一个高产出的构建系统

8.52. Coreutils-8.32

Coreutils 软件包包含用于显示和设定系统基本属性的工具。

估计构建时间: 2.6 SBU
需要硬盘空间: 157 MB

8.52.1. 安装 Coreutils

POSIX 要求 Coreutils 中的程序即使在多字节 locale 中也能正确识别字符边界。下面应用一个补丁，以解决 Coreutils 不满足该要求的问题，并修复其他一些国际化相关的 bug:

```
patch -Np1 -i ../coreutils-8.32-i18n-1.patch
```

注意

在之前，这个补丁中找出了许多 bug。在向 Coreutils 维护者报告新 bug 前，请检查它们在不使用该补丁的情况下是否还会重现。

阻止一个在某些机器上会无限循环的测试:

```
sed -i '/test.lock/s/^/#/' gnulib-tests/gnulib.mk
```

现在准备编译 Coreutils:

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

配置选项的含义:

autoreconf
该命令重新生成配置文件，使之与 automake 的最新版本一致。

FORCE_UNSAFE_CONFIGURE=1
该环境变量允许以 root 用户身份构建该软件包。

--enable-no-install-program=kill,uptime
这个开关的目的是防止 Coreutils 安装那些被其他软件包安装的二进制程序。

编译该软件包:

```
make
```

如果不运行测试套件，直接跳到 “安装该软件包”。

现在测试套件已经可以运行了。首先运行那些设计为由 root 用户运行的测试:

```
make NON_ROOT_USERNAME=tester check-root
```

之后我们要以 tester 用户身份运行其余测试。然而，某些测试要求测试用户属于至少一个组。为了不跳过这些测试，我们添加一个临时组，并使得 tester 用户成为它的成员:

```
echo "dummy:x:102:tester" >> /etc/group
```

修正访问权限，使得非 root 用户可以编译和运行测试：

```
chown -Rv tester .
```

现在运行测试：

```
su tester -c "PATH=$PATH make RUN_EXPENSIVE_TESTS=yes check"
```

删除临时组：

```
sed -i '/dummy/d' /etc/group
```

安装该软件包：

```
make install
```

将程序移动到 FHS 要求的位置：

```
mv -v /usr/bin/{cat,chgrp,chmod,chown,cp,date,dd,df,echo} /bin
mv -v /usr/bin/{false,ln,ls,mkdir,mknod,mv,pwd,rm} /bin
mv -v /usr/bin/{rmdir,stty,sync,true,uname} /bin
mv -v /usr/bin/chroot /usr/sbin
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
```

```
mv -v /usr/bin/{head,nice,sleep,touch} /bin
```

8.52.2. Coreutils 的内容

安装的程序: [, b2sum, base32, base64, basename, basenc, cat, chcon, chgrp, chmod, chown, chroot, cksum, comm, cp, csplit, cut, date, dd, df, dir, dircolors, dirname, du, echo, env, expand, expr, factor, false, fmt, fold, groups, head, hostid, id, install, join, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mktemp, mv, nice, nl, nohup, nproc, numfmt, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, realpath, rm, rmdir, runcon, seq, sha1sum, sha224sum, sha256sum, sha384sum, sha512sum, shred, shuf, sleep, sort, split, stat, stdbuf, stty, sum, sync, tac, tail, tee, test, timeout, touch, tr, true, truncate, tsort, tty, uname, unexpand, uniq, unlink, users, vdir, wc, who, whoami, 以及 yes
安装的库: libstdbuf.so (在 /usr/libexec/coreutils 中)
安装的目录: /usr/libexec/coreutils

简要描述

[这是一个真实存在的命令，/usr/bin/[，它和 **test** 命令的功能相同。
base32 根据 base32 标准 (RFC 4648) 编码和解码数据
base64 根据 base64 标准 (RFC 4648) 编码和解码数据
b2sum 打印或检查 BLAKE2 (512 位) 校验和
basename 从文件名移除所有路径和一个给定后缀
basenc 使用一些算法编码或解码数据

cat 将文件合并到标准输出
chcon 修改文件和目录的 SELinux 安全上下文
chgrp 修改文件和目录所属的组
chmod 修改给定文件的访问权限为指定模式； 模式可以是所需修改的符号表示，或新权限的八进制码
chown 修改拥有文件的用户或组
chroot 将给定目录作为 / 目录，运行命令
cksum 输出每个给定文件的循环冗余检查 (CRC) 校验和及字节数
comm 比较两个排好序的文件，将两个文件特有的部分和它们共有的部分显示为三列
cp 复制文件
csplit 将给定文件分割为若干新文件，根据给定模式或行号进行分割，并输出每个新文件的字节数
cut 根据给定的域或位置，打印输入的分节和选定部分
date 以给定格式显示当前时间，或设定系统时间
dd 以给定块大小和个数复制文件，同时可以进行转换
df 报告每个已挂载文件系统 (或包含给定文件的文件系统) 的总大小和可用空间
dir 列出给定目录的内容 (和 **ls** 命令相同)
dircolors 输出用于设定 LS_COLOR 环境变量的命令，以修改 **ls** 的配色方案
dirname 从文件名中删掉非目录的后缀
du 报告当前目录使用的磁盘空间，给出当前目录下所有子目录和文件占用的空间
echo 显示给定字符串
env 在修改的环境中运行命令
expand 将制表符转换成空格
expr 计算表达式
factor 打印所有给定整数的质因数
false 什么也不做；总是以失败状态码退出；
fmt 重新格式化给定文件的段落
fold 折叠给定文件中的行
groups 报告用户所属的组
head 打印文件的前 10 (或给定行数) 行
hostid 以十六进制格式打印主机数字标识符
id 报告当前用户或给定用户的有效用户 ID、组 ID 和所属的组
install 复制文件并设定它们的访问权限，以及 (如果可能) 它们的所有者和属组
join 将两个文件中拥有相同域的行合并
link 以给定文件名创建硬链接
ln 在文件之间创建硬链接或软 (符号) 链接
logname 报告当前用户登录名

ls 列出给定目录内容
md5sum 报告或检查消息摘要 5 (MD5) 校验和
mkdir 以给定名称创建目录
mkfifo 以给定名称创建先进先出表 (FIFO)，在 UNIX 惯用语中又称为 “命名管道”
mknod 以给定名称创建设备节点；设备节点可能是字符特殊文件、 块特殊文件或 FIFO
mktemp 安全地创建临时文件，常用在脚本中
mv 移动或重命名文件或目录
nice 以修改的调度优先级运行程序
nl 标出给定文件的行
nohup 执行命令并使其忽略挂机信号，同时将输出重定向到日志文件
nproc 打印进程可用的处理单元数目
numfmt 在数字和人类可读字符串之间互相转换
od 以八进制或其他格式转储文件
paste 合并给定文件，将它们的对应行连接起来，以制表符分割
pathchk 检查文件名的有效性和可移植性
pinky 是轻量级 finger 客户端，报告给定用户的一些信息
pr 对文件进行分页和分栏以便打印
printenv 打印环境变量
printf 以给定格式打印给定参数，很像 C printf 函数
ptx 用文中的每个关键字，根据给定文件内容生成重排索引
pwd 报告当前工作目录名
readlink 报告给定符号链接的值
realpath 打印解析过的目录
rm 删除文件或目录
rmdir 如果目录是空的，删除它们
runcon 以给定 SELinux 安全上下文运行命令
seq 以给定的范围和增量打印等差数列
sha1sum 打印或检查 160 位安全散列算法 1 (SHA1) 校验和
sha224sum 打印或检查 224 位安全散列算法校验和
sha256sum 打印或检查 256 位安全散列算法校验和
sha384sum 打印或检查 384 位安全散列算法校验和
sha512sum 打印或检查 512 位安全散列算法校验和
shred 将给定文件多次用复杂模式覆盖，增加恢复数据的难度
shuf 打乱文件中的行
sleep 等待给定时间
sort 对给定文件的行进行排序

split	根据大小或行数，将指定文件分割成若干部分
stat	显示文件或文件系统状态
stdbuf	以修改的标准流缓冲操作运行命令
stty	设置或报告终端行设定
sum	打印每个指定文件的校验和及块个数
sync	刷新文件系统缓冲；它将修改过的块强制写入磁盘，并更新超级块
tac	逆序连接给定文件
tail	输出给定文件的最后 10 (或指定行数) 行
tee	读取标准输入，并将内容同时写入标准输出和给定文件
test	比较两个值，或检查文件类型
timeout	在限定时间内运行命令
touch	修改文件时间戳，将每个给定文件的访问和修改时间设为当前时间；以零长度创建当前不存在的文件
tr	从标准输入变换、压缩或删除给定字符
true	什么也不做；总是以成功状态码退出
truncate	将文件截断或扩展到指定大小
tsort	进行拓扑排序； 根据给定文件的部分顺序信息输出完整的排序列表
tty	报告标准输入的终端文件名
uname	报告系统信息
unexpand	将空格转换成制表符
uniq	在连续的相同行中只保留一行，删除其他所有行
unlink	删除给定文件
users	报告当前登录系统的用户名
vdir	和 **ls -l** 相同
wc	报告给定文件的行数、单词数和字节数
who	报告当前登录的用户
whoami	报告与当前有效用户 ID 相关的用户名
yes	不停输出 “y” 或给定字符串，直到被杀死
`libstdbuf`	**stdbuf** 使用的库

8.53. Check-0.14.0

Check 是一个 C 语言单元测试框架。

估计构建时间: 0.1 SBU (计入测试为约 3.3 SBU)
需要硬盘空间: 12 MB

8.53.1. 安装 Check

准备编译 Check:

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

构建该软件包:

```
make
```

现在编译已经完成，执行以下命令执行 Check 测试套件:

```
make check
```

注意 Check 测试套件可能需要相对较长 (可能高达 4 SBU) 的时间。

安装该软件包:

```
make docdir=/usr/share/doc/check-0.14.0 install
```

8.53.2. Check 的内容

安装的程序: checkmk
安装的库: libcheck.{a,so}

简要描述

checkmk 用于生成 C 语言单元测试的 awk 脚本，生成的单元测试可以和 Check 单元测试框架一起使用

`libcheck.{a,so}` 包含使得测试程序能够调用 Check 的函数

8.54. Diffutils-3.7

Diffutils 软件包包含显示文件或目录之间差异的程序。

估计构建时间: 0.4 SBU
需要硬盘空间: 33 MB

8.54.1. 安装 Diffutils

准备编译 Diffutils:

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

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.54.2. Diffutils 的内容

安装的程序: cmp, diff, diff3, 以及 sdiff

简要描述

cmp 比较两个文件并报告它们是否相同，或哪些字节不同
diff 比较两个文件或目录，并报告文件中哪些行不同
diff3 逐行比较三个文件
sdiff 合并两个文件，并交互地输出结果

8.55. Gawk-5.1.0

Gawk 软件包包含操作文本文件的程序。

估计构建时间: 0.4 SBU
需要硬盘空间: 43 MB

8.55.1. 安装 Gawk

首先，确保不安装某些不需要的文件：

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

准备编译 Gawk：

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

编译该软件包：

```
make
```

运行以下命令以测试编译结果：

```
make check
```

安装该软件包：

```
make install
```

如果需要的话，安装文档：

```
mkdir -v /usr/share/doc/gawk-5.1.0
cp    -v doc/{awkforai.txt,*.{eps,pdf,jpg}} /usr/share/doc/gawk-5.1.0
```

8.55.2. Gawk 的内容

安装的程序: awk (link to gawk), gawk, 以及 awk-5.1.0
安装的库: filefuncs.so, fnmatch.so, fork.so, inplace.so, intdiv.so, ordchr.so, readdir.so, readfile.so, revoutput.so, revtwoway.so, rwarray.so, 以及 time.so (均位于 /usr/lib/gawk 中)
安装的目录: /usr/lib/gawk, /usr/libexec/awk, /usr/share/awk, 以及 /usr/share/doc/gawk-5.1.0

简要描述

awk	到 **gawk** 的链接
gawk	一个操作文本文件的程序；是 **awk** 的 GNU 实现
gawk-5.1.0	与 **gawk** 互为硬链接

8.56. Findutils-4.7.0

Findutils 软件包包含用于查找文件的程序。这些程序能够递归地搜索目录树，以及创建、维护和搜索文件数据库 (一般比递归搜索快，但在数据库最近没有更新时不可靠)。

估计构建时间: 0.7 SBU
需要硬盘空间: 51 MB

8.56.1. 安装 Findutils

准备编译 Findutils:

```
./configure --prefix=/usr --localstatedir=/var/lib/locate
```

配置选项的含义:

`--localstatedir`
该选项将 **locate** 数据库的位置改为 `/var/lib/locate`，以与 FHS 兼容。

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

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

安装该软件包:

```
make install
```

BLFS 及 BLFS 之外的一些软件包预期 **find** 程序在 `/bin` 中，因此要保证它被放置在那里:

```
mv -v /usr/bin/find /bin
sed -i 's|find:=${BINDIR}|find:=/bin|' /usr/bin/updatedb
```

8.56.2. Findutils 的内容

安装的程序: find, locate, updatedb, 以及 xargs
安装的目录: /var/lib/locate

简要描述

find 在给定目录树中搜索满足给定条件的文件

locate 在文件名数据库中进行搜索，报告包含特定字符串或匹配特定模式的文件名

updatedb 更新 **locate** 数据库；它扫描整个文件系统 (包括当前挂载的其他文件系统，除非被告知不这样做)，并把找到的所有文件名加入数据库

xargs 可以将给定命令作用于一个列表中的所有文件

8.57. Groff-1.22.4

Groff 软件包包含处理和格式化文本的程序。

估计构建时间: 0.5 SBU
需要硬盘空间: 96 MB

8.57.1. 安装 Groff

Groff 期望环境变量 PAGE 包含默认纸张大小。对于美国用户来说，PAGE=letter 是正确的。对于其他地方的用户，PAGE=A4 可能更好。尽管在编译时配置了默认纸张大小，可以通过写入 “A4” 或 “letter” 到 /etc/papersize 文件，覆盖默认值。

准备编译 Groff:

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

该软件包不支持并行构建。编译该软件包:

```
make -j1
```

该软件包不包含测试套件。

安装该软件包:

```
make install
```

8.57.2. Groff 的内容

安装的程序: addftinfo, afmtodit, chem, eqn, eqn2graph, gdiffmk, glilypond, gperl, gpinyin, grap2graph, grn, grodvi, groff, groffer, grog, grolbp, grolj4, gropdf, grops, grotty, hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pdfmom, pdfroff, pfbtops, pic, pic2graph, post-grohtml, preconv, pre-grohtml, refer, roff2dvi, roff2html, roff2pdf, roff2ps, roff2text, roff2x, soelim, tbl, tfmtodit, 以及 troff
安装的目录: /usr/lib/groff, /usr/share/doc/groff-1.22.4, 以及 /usr/share/groff

简要描述

addftinfo 读取 troff 字体文件并为其添加 **groff** 系统使用的一些额外字体规格信息
afmtodit 创建供 **groff** 和 **grops** 使用的字体文件
chem 产生化学结构式的 groff 预处理器
eqn 将 troff 输入文件中嵌入的公式描述编译成 **troff** 理解的命令
eqn2graph 将 troff EQN (公式) 转换成裁减好的图像
gdiffmk 标出 groff/nroff/troff 文件的区别
glilypond 将 lilypond 语言写成的乐谱转换为 groff 语言
gperl groff 预处理器，允许在 groff 文件中增加 perl 代码
gpinyin groff 的预处理器，允许在 groff 文件中增加汉语拼音
grap2graph 将 grap 图形转换成裁减好的位图图像

grn	用于 gremlin 文件的**groff** 预处理器
grodvi	**groff** 的驱动程序，生成 TeX dvi 格式
groff	groff 文档格式化系统的前端；一般来说，它运行 **troff** 程序和一个适用于选定设备的后处理器
groffer	在 X 和 tty 终端显示 groff 文件和 man 页面
grog	读取文件，并猜测 **groff** 选项 -e, -man, -me, -mm, -ms, -p, -s, 以及 -t 中哪一个在打印文件时是必要的，并报告包含这些选项的 **groff** 命令
grolbp	是一个用于 Canon CAPSL 打印机 (LBP-4 和 LBP-8 系列激光打印机) 的 **groff** 驱动程序
grolj4	是一个生成用于 HP LaserJet 4 打印机的 PCL5 格式的 **groff** 驱动程序
gropdf	将 **troff** 输出转换成 PDF
grops	将 **troff** 输出转换成 PostScript
grotty	将 **troff** 输出转换成用于打字机类设备的形式
hpftodit	根据 HP 标签的字体规格文件，创建用于 **groff -Tlj4**的字体文件
indxbib	创建用于给定文件文献数据库的反向索引，以供 **refer**、**lookbib** 以及 **lkbib** 使用
lkbib	在文献数据库中搜索包含指定关键字的引用，并报告找到的所有引用
lookbib	在标准错误输出上显示命令提示符 (除非标准输入不是终端)，读取包含一组关键字的行，在给定文件的文献数据库中搜索包含这些关键字的引用，将它们打印到标准输出，重复这一过程直到输入结束
mmroff	**groff** 的简单预处理器
neqn	将公式格式化为美国标准信息交换代码 (ASCII) 输出
nroff	一个使用 **groff** 仿真 **nroff** 命令的脚本
pdfmom	一个 groff 包装器，提供从 mom 宏包编码的文件转换为 PDF 文档的功能
pdfroff	用 groff 创建 PDF 文档
pfbtops	将 .pfb 格式的 PostScript 字体转换为 ASCII
pic	将 troff 或 TeX 输入文件中嵌入的图片描述编译成 TeX 或 **troff** 理解的命令
pic2graph	将 PIC 图示转换成裁切好的图像
post-grohtml	将 GNU **troff** 的输出翻译成 HTML
preconv	将输入文件的编码转换成 GNU **troff** 理解的格式
pre-grohtml	将 GNU **troff** 的输出翻译成 HTML
refer	将文件内容复制到标准输出，除了在 .[和 .] 之间的行被解释为文献引用，.R1 和 .R2 之间的行被解释为处理文献引用的方式
roff2dvi	将 roff 文件转换成 DVI 格式
roff2html	将 roff 文件转换成 HTML 格式
roff2pdf	将 roff 文件转换成 PDF
roff2ps	将 roff 文件转换成 ps 文件
roff2text	将 roff 文件转换成文本文件
roff2x	将 roff 文件转换成其他格式

soelim	读取文件，将 .so 文件 形式的行替换为该文件的内容
tbl	将 troff 输入中嵌入的表格描述编译成 **troff** 理解的命令
tfmtodit	创建用于 **groff -Tdvi** 的字体文件
troff	和 UNIX **troff** 高度兼容； 它应该由 **groff** 命令调用，后者也会以正确的顺序和选项运行预处理器和后处理器

8.58. GRUB-2.04

GRUB 软件包包含 “大统一” (GRand Unified) 启动引导器。

估计构建时间: 0.8 SBU
需要硬盘空间: 154 MB

8.58.1. 安装 GRUB

准备编译 GRUB:

```
./configure --prefix=/usr          \
            --sbindir=/sbin        \
            --sysconfdir=/etc      \
            --disable-efiemu       \
            --disable-werror
```

新的配置选项的含义:

`--disable-werror`
该选项允许在有较新的 Flex 版本导致的警告时完成构建。

`--disable-efiemu`
该选项通过禁用 LFS 不需要的特性和测试程序，最小化需要构建的内容。

编译该软件包:

```
make
```

该软件包不包含测试套件。

安装该软件包:

```
make install
mv -v /etc/bash_completion.d/grub /usr/share/bash-completion/completions
```

使用 GRUB 引导您的 LFS 系统的方法将在 第 10.4 节 “使用 GRUB 设定引导过程” 中讨论。

8.58.2. GRUB 的内容

安装的程序: grub-bios-setup, grub-editenv, grub-file, grub-fstest, grub-glue-efi, grub-install, grub-kbdcomp, grub-macbless, grub-menulst2cfg, grub-mkconfig, grub-mkimage, grub-mklayout, grub-mknetdir, grub-mkpasswd-pbkdf2, grub-mkrelpath, grub-mkrescue, grub-mkstandalone, grub-ofpathname, grub-probe, grub-reboot, grub-render-label, grub-script-check, grub-set-default, grub-sparc64-setup, 以及 grub-syslinux2cfg
安装的目录: /usr/lib/grub, /etc/grub.d, /usr/share/grub, 以及 /boot/grub (在初次运行 grub-install 时安装)

简要描述

grub-bios-setup grub-install 使用的辅助程序

grub-editenv	用于编辑环境块的工具
grub-file	检验文件是否是给定类型
grub-fstest	调试文件系统驱动程序的工具
grub-glue-efi	处理 ia32 和 amd64 EFI 镜像，根据 Apple 格式粘合它们
grub-install	在您的驱动器上安装 GRUB
grub-kbdcomp	将 xkb 布局转化为 GRUB 能够识别的格式的脚本
grub-macbless	Mac 风格的 bless 命令，用于 HFS 或 HFS+ 文件系统
grub-menulst2cfg	将经典的 GRUB `menu.lst` 转化为 `grub.cfg` 以供 GRUB 2 使用
grub-mkconfig	生成 GRUB 配置文件
grub-mkimage	创建 GRUB 可引导镜像
grub-mklayout	生成 GRUB 键盘布局文件
grub-mknetdir	准备 GRUB 网络启动目录
grub-mkpasswd-pbkdf2	生成用于引导菜单的加密 PBKDF2 密码
grub-mkrelpath	生成相对于根目录的系统路径名称
grub-mkrescue	为软盘或 CDROM/DVD 创建 GRUB 可引导镜像
grub-mkstandalone	生成独立 (包含所有模块) 的镜像
grub-ofpathname	打印 GRUB 设备路径的帮助程序
grub-probe	对给定路径或设备探测信息
grub-reboot	仅为下次启动设置 GRUB 默认引导项
grub-render-label	为 Apple Mac 设置 Apple .disk_label
grub-script-check	在 GRUB 配置脚本中检查语法错误
grub-set-default	设置 GRUB 默认引导项
grub-sparc64-setup	grub-setup 使用的帮助程序
grub-syslinux2cfg	将 syslinux 配置文件转换为 grub.cfg 格式

8.59. Less-551

Less 软件包包含一个文本文件查看器。

估计构建时间: 不到 0.1 SBU
需要硬盘空间: 4.1 MB

8.59.1. 安装 Less

准备编译 Less:

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

配置选项的含义:

`--sysconfdir=/etc`
该选项告诉该软件包创建的程序在 /etc 查找配置文件

编译该软件包:

```
make
```

该软件包不包含测试套件。

安装该软件包:

```
make install
```

8.59.2. Less 的内容

安装的程序: less, lessecho, 以及 lesskey

简要描述

less 一个文件查看器或分页器；它显示给定文件的内容，使得用户可以进行滚动、查找字符串，或跳到标记

lessecho 用于展开元字符，例如 Unix 系统上文件名中的 * 和 ?

lesskey 用于指定 **less** 的按键绑定

8.60. Gzip-1.10

Gzip 软件包包含压缩和解压缩文件的程序。

估计构建时间: 0.1 SBU
需要硬盘空间: 19 MB

8.60.1. 安装 Gzip

准备编译 Gzip:

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

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

移动一个必须放置在根文件系统的程序:

```
mv -v /usr/bin/gzip /bin
```

8.60.2. Gzip 的内容

安装的程序: gunzip, gzexe, gzip, uncompress (与 gunzip 互为硬链接), zcat, zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore, 以及 znew

简要描述

gunzip	解压缩 gzip 压缩的文件
gzexe	创建自解压可执行文件
gzip	使用 Lempel-Ziv (LZ77) 编码压缩文件
uncompress	解压压缩文件
zcat	将给定 gzip 压缩文件解压到标准输出
zcmp	在 gzip 压缩文件上运行 **cmp**
zdiff	在 gzip 压缩文件上运行 **diff**
zegrep	在 gzip 压缩文件上运行 **egrep**
zfgrep	在 gzip 压缩文件上运行 **fgrep**
zforce	为给定所有文件中的 gzip 压缩文件确保 .gz 扩展名，这样 **gzip** 就不会重复压缩它们； 在文件传输过程中文件名被截断时，这个命令很有用
zgrep	在 gzip 压缩文件上运行 **grep**
zless	在 gzip 压缩文件上运行 **less**

zmore 在 gzip 压缩文件上运行 **more**

znew 将 **compress** 格式压缩文件重新压缩为 **gzip** 格式 —— 转换 .Z 文件为 .gz 文件

8.61. IPRoute2-5.7.0

IPRoute2 软件包包含基于 IPv4 的基本和高级网络程序。

估计构建时间: 0.2 SBU
需要硬盘空间: 14 MB

8.61.1. 安装 IPRoute2

该软件包中的 **arpd** 程序依赖于 LFS 不安装的 Berkeley DB，因此不会被构建。然而，用于 **arpd** 的一个目录和它的 man 页面仍会被安装。运行以下命令以防止它们的安装。如果需要使用 **arpd** 二进制程序，参考 BLFS 手册中的 Berkeley DB 编译说明，它位于 http://www.linuxfromscratch.org/blfs/view/svn/server/db.html。

```
sed -i /ARPD/d Makefile
rm -fv man/man8/arpd.8
```

还需要禁用两个需要 http://www.linuxfromscratch.org/blfs/view/svn/postlfs/iptables.html 的模块。

```
sed -i 's/.m_ipt.o//' tc/Makefile
```

编译该软件包：

```
make
```

该软件包没有能够工作的测试套件。

安装该软件包：

```
make DOCDIR=/usr/share/doc/iproute2-5.7.0 install
```

8.61.2. IPRoute2 的内容

安装的程序: bridge, ctstat (到 lnstat 的链接), genl, ifcfg, ifstat, ip, lnstat, nstat, routef, routel, rtacct, rtmon, rtpr, rtstat (到 lnstat 的链接), ss, 以及 tc
安装的目录: /etc/iproute2, /usr/lib/tc, 以及 /usr/share/doc/iproute2-5.7.0,

简要描述

bridge 配置网桥

ctstat 连接状态工具

genl 通用网络连接工具前端

ifcfg 一个封装 **ip** 命令的脚本 [注意它需要 **arping** 和 **rdisk** 程序，它们来自于 iputils 软件包，可以在 http://www.skbuff.net/iputils/找到。]

ifstat 显示网络接口统计，包括接口上发送和接收的数据包数量

ip 该软件包的主程序。它包含几种不同的功能：
ip link <device> 允许用户查看和修改设备状态
ip addr 允许用户查看网络地址及其属性，添加新地址或删除旧地址
ip neigh 允许用户查看 ARP 近邻绑定及其属性，增加新近邻项，或删除旧项
ip rule 允许用户查看或修改路由策略

ip route 允许用户查看路由表或修改路由表规则
ip tunnel 允许用户查看 IP 隧道及其属性，或修改它们
ip maddr 允许用户查看多播地址及其属性，或修改它们
ip mroute 允许用户设定、 修改或删除多播路由
ip monitor 允许用户连续地监视设备、 地址和路由的状态

lnstat 提供 Linux 网络统计；它是旧的**rtstat** 的更通用、功能更完备的替代品

nstat 显示网络统计

routef **ip route** 的一个组件。用于刷新路由表

routel **ip route** 的一个组件。用于显示路由表

rtacct 显示 /proc/net/rt_acct 的内容

rtmon 路由监视工具

rtpr 将 **ip -o** 的输出转换为可读形式

rtstat 路由状态工具

ss 与 **netstat** 命令相似；显示活动连接

tc 流量管制可执行程序；用于实现服务质量 (QOS) 和服务类型 (COS) 协议
tc qdisc 允许用户设定排队规则
tc class 允许用户设定基于排队规则调度的调度类
tc estimator 允许用户预计进入网络的流量
tc filter 允许用户设定 QOS/COS 数据包过滤
tc policy 允许用户设定 QOS/COS 策略

8.62. Kbd-2.2.0

Kbd 软件包包含按键表文件、控制台字体和键盘工具。

估计构建时间: 0.1 SBU
需要硬盘空间: 36 MB

8.62.1. 安装 Kbd

退格和删除键的行为在 Kbd 软件包的不同按键映射中不一致。以下补丁修复 i386 按键映射中的这个问题:

```
patch -Np1 -i ../kbd-2.2.0-backspace-1.patch
```

在应用补丁后，退格键生成编码为 127 的字符，删除键生成广为人知的 escape 序列。

删除多余的 **resizecons** 程序 (它需要已经不存在的 svgalib 提供视频模式文件 —— 一般使用 **setfont** 即可调整控制台大小) 及其 man 页面。

```
sed -i '/RESIZECONS_PROGS=/s/yes/no/' configure
sed -i 's/resizecons.8 //' docs/man/man8/Makefile.in
```

准备编译 Kbd:

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

配置选项的含义:

`--disable-vlock`
该选项防止构建 vlock 工具，因为它需要 chroot 环境中不可用的 PAM 库。

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

注意

对于一些语言 (如白俄罗斯文)，Kbd 软件包没有提供有用的的键盘映射。它提供的白俄罗斯文“by” 键盘映射假设使用 ISO-8859-5 编码，但通常应该使用的是 CP1251 编码的键盘映射。使用白俄罗斯文等文字的用户需要单独下载可工作的键盘映射。

如果需要的话，安装文档:

```
mkdir -v            /usr/share/doc/kbd-2.2.0
cp -R -v docs/doc/* /usr/share/doc/kbd-2.2.0
```

8.62.2. Kbd 的内容

安装的程序: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, kbdinfo, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (到 psfxtable 的链接), psfgettable (到 psfxtable 的链接), psfstriptable (到 psfxtable 的链接), psfxtable, setfont, setkeycodes, setleds, setmetamode, setvtrgb, showconsolefont, showkey, unicode_start, 以及 unicode_stop

安装的目录: /usr/share/consolefonts, /usr/share/consoletrans, /usr/share/keymaps, /usr/share/doc/kbd-2.2.0, 以及 /usr/share/unimaps

简要描述

chvt	修改当前虚拟终端
deallocvt	取消未使用的虚拟终端分配
dumpkeys	转储键盘转换表
fgconsole	打印活动虚拟终端的个数
getkeycodes	打印内核扫描码到键码的映射表
kbdinfo	获取终端状态信息
kbd_mode	报告或设置键盘模式
kbdrate	设置键盘重复和延迟率
loadkeys	加载键盘翻译表
loadunimap	加载内核 unicode 到字体的映射表
mapscrn	一个过时程序，曾用于将用户定义输出字符映射表加载到终端驱动程序；现在该任务由 **setfont** 完成
openvt	在新的虚拟终端 (VT) 启动程序
psfaddtable	向控制台字体增加 Unicode 字符表
psfgettable	提取控制台字体中嵌入的 Unicode 字符表
psfstriptable	删除控制台字体中嵌入的 Unicode 字符表
psfxtable	处理控制台字体的 Unicode 字符表
setfont	修改控制台上的增强图形适配器 (EGA) 和视频图像阵列 (VGA) 字体
setkeycodes	加载内核扫描码到键码的映射表项；在键盘上有特殊按键时很有用
setleds	设置键盘标志位和发光二极管 (LED)
setmetamode	定义键盘转换键 (meta-key) 处理
setvtrgb	设定所有虚拟终端的控制台颜色映射
showconsolefont	显示当前 EGA/VGA 控制台屏幕字体
showkey	报告键盘按键的扫描码、键码和 ASCII 编码
unicode_start	将键盘和控制台设定为 UNICODE 模式 [不要使用该程序，除非您的键盘映射文件是 ISO-8859-1 编码的。对于其他编码，该工具产生错误结果。]
unicode_stop	使键盘和控制台退出 UNICODE 模式

8.63. Libpipeline-1.5.2

Libpipeline 软件包包含用于灵活、方便地处理子进程流水线的库。

估计构建时间: 0.2 SBU
需要硬盘空间: 10 MB

8.63.1. 安装 Libpipeline

准备编译 Libpipeline:

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

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.63.2. Libpipeline 的内容

安装的库: libpipeline.so

简要描述

`libpipeline` 用于安全地在子进程之间构建流水线

8.64. Make-4.3

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

估计构建时间: 0.5 SBU
需要硬盘空间: 13 MB

8.64.1. 安装 Make

准备编译 Make:

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

编译该软件包:

```
make
```

如果要测试编译结果，运行命令:

```
make check
```

安装该软件包:

```
make install
```

8.64.2. Make 的内容

安装的程序: make

简要描述

make 自动确定软件包中需要(重新)构建的部分，并执行对应命令

8.65. Patch-2.7.6

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

估计构建时间: 0.2 SBU
需要硬盘空间: 12 MB

8.65.1. 安装 Patch

准备编译 Patch：

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

编译该软件包：

```
make
```

运行以下命令以测试编译结果：

```
make check
```

安装该软件包：

```
make install
```

8.65.2. Patch 的内容

安装的程序: patch

简要描述

patch 根据补丁文件修改文件 (补丁文件一般是使用 **diff** 程序创建的差异清单。通过将这些差异应用到原始文件，**patch** 即可创建应用补丁的文件版本。)

8.66. Man-DB-2.9.2

Man-DB 软件包包含查找和阅读 man 页面的程序。

估计构建时间: 0.5 SBU
需要硬盘空间: 40 MB

8.66.1. 安装 Man-DB

准备编译 Man-DB:

```
sed -i '/find/s@/usr@@' init/systemd/man-db.service.in

./configure --prefix=/usr                         \
            --docdir=/usr/share/doc/man-db-2.9.2 \
            --sysconfdir=/etc                     \
            --disable-setuid                      \
            --enable-cache-owner=bin              \
            --with-browser=/usr/bin/lynx          \
            --with-vgrind=/usr/bin/vgrind         \
            --with-grap=/usr/bin/grap
```

配置选项的含义:

sed -i '/find/s@/usr@@' init/systemd/man-db.service.in
修改硬编码的，指向 **find** 工具的路径，它被我们安装到 /bin 目录。

`--disable-setuid`
该选项防止将 **man** 程序 setuid 到用户 man。

`--enable-cache-owner=bin`
该选项使得系统范围的缓存文件所有者为用户 bin。

`--with-...`
这三个选项设定一些默认程序。**lynx** 是基于文本的 web 浏览器 (安装过程可在 BLFS 中查阅)，**vgrind** 将程序源代码转换成 Groff 输入，**grap** 用于在 Groff 文档中画图。**vgrind** 和 **grap** 在阅读 man 手册页面时一般用不到。它们不是 LFS 或 BLFS 的一部分，但如果需要的话，您应该可以在完成 LFS 的构建后自行安装它们。

编译该软件包:

```
make
```

运行以下命令以测试编译结果:

```
make check
```

安装该软件包:

```
make install
```

8.66.2. LFS 中的非英文 man 手册页面

下表展示了 Man-DB 假定的安装在 /usr/share/man/<ll> 中的 man 手册页面的编码字符集。另外，Man-DB 还能正确地判断出这些页面是否为 UTF-8 编码。

表 8.1. 传统 8 位 man 手册页面的预期字符编码

语言 (代码)	编码	语言 (代码)	编码
丹麦语 (da)	ISO-8859-1	克罗地亚语 (hr)	ISO-8859-2
德语 (de)	ISO-8859-1	匈牙利语 (hu)	ISO-8859-2
英语 (en)	ISO-8859-1	日语 (ja)	EUC-JP
西班牙语 (es)	ISO-8859-1	朝鲜语 (ko)	EUC-KR
爱沙尼亚语 (et)	ISO-8859-1	立陶宛语 (lt)	ISO-8859-13
芬兰语 (fi)	ISO-8859-1	拉脱维亚语 (lv)	ISO-8859-13
法语 (fr)	ISO-8859-1	马其顿语 (mk)	ISO-8859-5
爱尔兰语 (ga)	ISO-8859-1	波兰语 (pl)	ISO-8859-2
加利西亚语 (gl)	ISO-8859-1	罗马尼亚语 (ro)	ISO-8859-2
印度尼西亚语 (id)	ISO-8859-1	俄语 (ru)	KOI8-R
冰岛语 (is)	ISO-8859-1	斯洛伐克语 (sk)	ISO-8859-2
意大利语 (it)	ISO-8859-1	斯洛文尼亚语 (sl)	ISO-8859-2
挪威巴克摩语 (nb)	ISO-8859-1	拉丁文书写的塞尔维亚语 (sr@latin)	ISO-8859-2
荷兰语 (nl)	ISO-8859-1	塞尔维亚语 (sr)	ISO-8859-5
挪威尼诺斯克语 (nn)	ISO-8859-1	土耳其语 (tr)	ISO-8859-9
挪威语 (no)	ISO-8859-1	乌克兰语 (uk)	KOI8-U
葡萄牙语 (pt)	ISO-8859-1	越南语 (vi)	TCVN5712-1
瑞典语 (sv)	ISO-8859-1	简体中文 (zh_CN)	GBK
白俄罗斯语 (be)	CP1251	简体中文，新加坡 (zh_SG)	GBK
保加利亚语 (bg)	CP1251	繁体中文，香港特别行政区 (zh_HK)	BIG5HKSCS
捷克语 (cs)	ISO-8859-2	繁体中文 (zh_TW)	BIG5
希腊文 (el)	ISO-8859-7		

注意

用该表之外的语言编写的 man 手册页面不被支持。

8.66.3. Man-DB 的内容

安装的程序: accessdb, apropos (link to whatis), catman, lexgrog, man, mandb, manpath, 以及 whatis
安装的库: libman.so 和 libmandb.so (都在 /usr/lib/man-db 中)
安装的目录: /usr/lib/man-db, /usr/libexec/man-db, and /usr/share/doc/man-db-2.9.2

简要描述

accessdb 将 **whatis** 数据库内容转储为人类可读格式

apropos	搜索 **whatis** 数据库，显示包含给定字符串的系统命令的简要描述
catman	创建或更新预先格式化的 man 手册页面
lexgrog	显示给定 man 手册页面的单行摘要信息
man	格式化并显示请求的 man 手册页面
mandb	创建或更新 **whatis** 数据库
manpath	显示 $MANPATH 的内容，或者 (如果 $MANPATH 未设定) 根据 man.conf 和用户环境确定的合适搜索路径
whatis	搜索 **whatis** 数据库，显示包含给定关键词的系统命令的简要描述
`libman`	包含 **man** 运行时支持
`libmandb`	包含 **man** 运行时支持

8.67. Tar-1.32

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

估计构建时间: 2.5 SBU
需要硬盘空间: 39 MB

8.67.1. 安装 Tar

准备编译 Tar:

```
FORCE_UNSAFE_CONFIGURE=1  \
./configure --prefix=/usr \
            --bindir=/bin
```

配置选项的含义:

FORCE_UNSAFE_CONFIGURE=1
该选项强制以 root 用户身份运行 mknod 测试。一般认为以 root 用户身份运行该测试是危险的，不过由于是在仅仅部分构建好的系统上运行测试，可以覆盖掉这个安全措施。

编译该软件包:

```
make
```

执行以下命令测试编译结果 (需要约 3 SBU):

```
make check
```

安装该软件包:

```
make install
make -C doc install-html docdir=/usr/share/doc/tar-1.32
```

8.67.2. Tar 的内容

安装的程序: tar
安装的目录: /usr/share/doc/tar-1.32

简要描述

tar 创建称为 tarball 的档案文件，从档案文件中提取文件，或列出档案文件内容

8.68. Texinfo-6.7

Texinfo 软件包包含阅读、编写和转换 info 页面的程序。

估计构建时间: 0.7 SBU
需要硬盘空间: 104 MB

8.68.1. 安装 Texinfo

准备编译 Texinfo：

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

配置选项的含义：

--disable-static
在本例中，顶层配置脚本会抱怨说这是一个无法识别的选项，但 XSParagraph 配置脚本能够识别它，并禁止将静态库 XSParagraph.a 安装到 /usr/lib/texinfo。

编译该软件包：

```
make
```

运行以下命令以测试编译结果：

```
make check
```

安装该软件包：

```
make install
```

可选地，安装属于 TeX 环境的组件：

```
make TEXMF=/usr/share/texmf install-tex
```

make 命令参数的含义：

TEXMF=/usr/share/texmf
TEXMF Makefile 变量包含之后可能安装的 TeX 软件包的 TeX 目录树根位置。

Info 文档系统使用一个纯文本文件保存目录项的列表。该文件位于 /usr/share/info/dir。不幸的是，由于一些软件包 Makefile 中偶然出现的问题，它有时会与系统实际安装的 info 页面不同步。如果需要重新创建 /usr/share/info/dir 文件，可以运行以下命令完成这一工作：

```
pushd /usr/share/info
  rm -v dir
  for f in *
    do install-info $f dir 2>/dev/null
  done
popd
```

8.68.2. Texinfo 的内容

安装的程序: info, install-info, makeinfo (到 texi2any 的链接), pdftexi2dvi, pod2texi, texi2any, texi2dvi, texi2pdf, 以及 texindex
安装的库: MiscXS.so, Parsetexi.so, 以及 XSParagraph.so (都在 /usr/lib/texinfo 中)
安装的目录: /usr/share/texinfo 和 /usr/lib/texinfo

简要描述

info	用于阅读和 man 页面类似的 info 页面，man 页面一般只解释可用的命令行选项，而 info 页面更为深入 [例如，可以对比 **man bison** 和 **info bison**。]
install-info	用于安装 info 页面；该命令更新 **info** 索引文件
makeinfo	将给定 Texinfo 源代码文档转换成 info 页面、纯文本或 HTML
pdftexi2dvi	将给定 Texinfo 文档格式化为可移植文档格式 (PDF) 文件
pod2texi	将 Pod 转换成 Texinfo 格式
texi2any	将 Texinfo 文档转换成其他几种格式
texi2dvi	将给定 Texinfo 文档格式化为可打印的设备无关文件
texi2pdf	将给定 Texinfo 文档格式化为可移植文档格式 (PDF) 文件
texindex	用于排序 Texinfo 索引文件

8.69. Vim-8.2.0814

Vim 软件包包含强大的文本编辑器。

估计构建时间: 3.0 SBU
需要硬盘空间: 198 MB

Vim 的替代品

如果您喜爱其他编辑器 —— 例如 Emacs、Joe、或者 Nano —— 参考 http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html 中建议的安装说明。

8.69.1. 安装 Vim

首先，修改 vimrc 配置文件的默认位置为 /etc:

```
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
```

准备编译 Vim:

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

编译该软件包:

```
make
```

为了准备运行测试套件，需要使得 tester 用户拥有写入源代码目录树的权限:

```
chown -Rv tester .
```

现在，以 tester 用户身份运行测试:

```
su tester -c "LANG=en_US.UTF-8 make -j1 test" &> vim-test.log
```

测试套件会将大量二进制数据输出到屏幕。这可能扰乱当前终端设置。为了避免这个问题，像上面的命令一样，将输出重定向到日志文件。测试成功完成后，日志文件末尾会包含 “ALL DONE”。

安装该软件包:

```
make install
```

许多用户习惯于使用命令 **vi**，而不是 **vim**。为了在用户习惯性地输入 **vi** 时能够执行 **vim**，为二进制程序和各种语言的 man 页面创建符号链接:

```
ln -sv vim /usr/bin/vi
for L in  /usr/share/man/{,*/}man1/vim.1; do
    ln -sv vim.1 $(dirname $L)/vi.1
done
```

默认情况下，Vim 的文档安装在 /usr/share/vim 中。下面创建符号链接，使得可以通过 /usr/share/doc/vim-8.2.0814 访问符号链接，这个路径与其他软件包的文档位置格式一致:

```
ln -sv ../vim/vim82/doc /usr/share/doc/vim-8.2.0814
```

如果在安装 LFS 系统后安装了 X 窗口系统，可能需要在安装 X 后重新编译 Vim。Vim 提供的 GUI 版本编辑器需要 X 和一些额外的软件包才能安装。关于这一安装过程的更多信息，参考 Vim 文档和 BLFS 手册中位于 http://www.linuxfromscratch.org/blfs/view/svn/postlfs/vim.html 的 Vim 安装页面。

8.69.2. 配置 Vim

默认情况下，**vim** 在不兼容 vi 的模式下运行。这对于过去使用其他编辑器的用户来说可能显得陌生。以下配置包含的 “nocompatible” 设定是为了强调编辑器使用了新的行为这一事实。它也提醒那些想要使用 “compatible” 模式的用户，必须在配置文件的一开始改变模式。这是因为它会修改其他设置，对这些设置的覆盖必须在设定模式后进行。执行以下命令创建默认 **vim** 配置文件：

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

" Ensure defaults are set before customizing settings, not after
source $VIMRUNTIME/defaults.vim
let skip_defaults_vim=1

set nocompatible
set backspace=2
set mouse=
syntax on
if (&term == "xterm") || (&term == "putty")
  set background=dark
endif

" End /etc/vimrc
EOF
```

`set nocompatible` 设定使得 **vim** 以一种更有用的方式 (也是默认方式) 行动，而不是兼容于 vi 的旧模式。删除其中的 “no” 可以保持旧的 **vi** 行为。`set backspace=2` 设定允许退格越过换行，自动缩进，以及插入模式的起始位置。参数 `syntax on` 启用 vim 符号高亮功能。参数 `set mouse=` 允许在 chroot 中或通过远程连接工作时使用鼠标正确地粘贴文本。最后，if 语句为 `set background=dark` 纠正 **vim** 对于某些终端模拟器背景色的猜测。这能够提供更适合这些程序黑色背景的配色方案。

关于其他可用选项的文档可以通过执行以下命令获得：

```
vim -c ':options'
```

注意

默认情况下 Vim 只安装英语拼写检查文件。如果希望安装您使用的语言的拼写检查文件，需要获取使用于您的语言和字符编码的 `*.spl` 和可选的 `*.sug` 文件，从 ftp://ftp.vim.org/pub/vim/runtime/spell/ 下载它们，并保存到 `/usr/share/vim/vim82/spell/`。

为了使用这些拼写检查文件，需要在 `/etc/vimrc` 中进行配置，例如：

```
set spelllang=en,ru
set spell
```

关于更多信息，参考以上 URL 位置中合适的 README 文件。

8.69.3. Vim 的内容

安装的程序: ex (到 vim 的链接), rview (到 vim 的链接), rvim (到 vim 的链接), vi (到 vim 的链接), view (到 vim 的链接), vim, vimdiff (到 vim 的链接), vimtutor, 以及 xxd

安装的目录: /usr/share/vim

简要描述

ex	以 ex 模式启动 **vim**
rview	是 **view** 的受限模式； 不能启动 shell 命令，且不能挂起 **view**
rvim	是 **vim** 的受限模式； 不能启动 shell 命令，且不能挂起 **vim**
vi	到 **vim** 的链接
view	以只读模式启动 **vim**
vim	文本编辑器
vimdiff	用 **vim** 编辑两个或三个文件版本，并显示差异
vimtutor	教会用户使用 **vim** 的基本快捷键和命令
xxd	创建文件的十六进制转储；它也可以从十六进制转储创建文件，因此可用于二进制补丁

8.70. Systemd-245

Systemd 软件包包含控制系统引导、运行和关闭的程序。

估计构建时间: 1.9 SBU
需要硬盘空间: 249 MB

8.70.1. 安装 systemd

首先，应用一个补丁，修复使用 GCC-10 编译时出现的问题，并修复一个段错误：

```
patch -Np1 -i ../systemd-245-gcc_10-fixes-2.patch
```

创建一个符号链接，绕过本书不会安装的 xsltproc：

```
ln -sf /tools/bin/true /usr/bin/xsltproc
```

设定好 man 页面：

```
tar -xf ../systemd-man-pages-245.tar.xz
```

删除在 chroot 环境中无法构建的测试：

```
sed '179,$ d' -i src/resolve/meson.build
```

从默认的 udev 规则中删除不必要的组 render：

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

准备安装 systemd：

```
mkdir -p build
cd       build

LANG=en_US.UTF-8                  \
meson --prefix=/usr               \
      --sysconfdir=/etc           \
      --localstatedir=/var        \
      -Dblkid=true                \
      -Dbuildtype=release         \
      -Ddefault-dnssec=no         \
      -Dfirstboot=false           \
      -Dinstall-tests=false       \
      -Dkmod-path=/bin/kmod       \
      -Dldconfig=false            \
      -Dmount-path=/bin/mount     \
      -Drootprefix=               \
      -Drootlibdir=/lib           \
      -Dsplit-usr=true            \
      -Dsulogin-path=/sbin/sulogin  \
      -Dsysusers=false            \
      -Dumount-path=/bin/umount   \
      -Db_lto=false               \
      -Drpmmacrosdir=no           \
      -Dhomed=false               \
      -Duserdb=false              \
      -Dman=true                  \
      ..
```

meson 选项的含义：

`-D*-path=*`
这些开关提供了 systemd 在运行时需要，但尚未安装的二进制程序的位置。

`-Ddefault-dnssec=no`
这个开关禁用实验性的 DNSSEC 支持。

`-Dfirstboot=false`
这个开关防止 systemd 安装用于初始化设定系统的服务。在 LFS 中所有工作都会手工完成，因此不需要它们。

`-Dinstall-tests=false`
这个开关防止 systemd 安装编译好的测试文件。

`-Dldconfig=false`
这个开关防止一个 systemd 单元的安装，它在引导时运行 **ldconfig**，这对于 LFS 等源代码发行版来说没有意义，还会增加引导时间。如果您需要这个功能，可以删除这个开关。

`-Droot*`
这个开关保证核心程序和共享库被安装到根分区下的子目录中。

`-Dsplit-usr=true`
这个开关确保 systemd 能够在 /bin、/lib 和 /sbin 目录不是指向 /usr 中对应目录的符号链接的情况下工作。

`-Dsysusers=false`
这个开关防止 systemd 安装负责设定 /etc/group 和 /etc/passwd 文件的服务。我们在上一章已经创建了这两个文件。

`-Drpmmacrosdir=no`
该选项禁止安装用于 systemd 的 RPM 宏，因为 LFS 并不支持 RPM。

`-D{userdb,homed}=false`
移除两个守护程序，它们的依赖项超出了 LFS 的范围。

编译该软件包：

```
LANG=en_US.UTF-8 ninja
```

安装该软件包：

```
LANG=en_US.UTF-8 ninja install
```

删除一个不再必要的符号链接：

```
rm -f /usr/bin/xsltproc
```

创建 /etc/machine-id 文件，**systemd-journald**需要它：

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

设定启动目标单元的基本结构：

```
systemctl preset-all
```

已知一个服务单元会导致并非由 systemd-networkd 提供网络配置的系统出现问题，禁用它：

```
systemctl disable systemd-time-wait-sync.service
```

防止 systemd 重设最大 PID 值，它会导致 BLFS 中一些软件包和单元出现问题：

```
rm -f /usr/lib/sysctl.d/50-pid-max.conf
```

8.70.2. systemd 的内容

安装的程序: bootctl, busctl, coredumpctl, halt (到 systemctl 的符号链接), hostnamectl, init, journalctl, kernel-install, localectl, loginctl, machinectl, networkctl, portablectl, poweroff (到 systemctl 的符号链接), reboot (到 systemctl 的符号链接), resolvconf (到 resolvectl 的符号链接), resolvectl, runlevel (到 systemctl 的符号链接), shutdown (到 systemctl 的符号链接), systemctl, systemd-analyze, systemd-ask-password, systemd-cat, systemd-cgls, systemd-cgtop, systemd-delta, systemd-detect-virt, systemd-escape, systemd-hwdb, systemd-id128, systemd-inhibit, systemd-machine-id-setup, systemd-mount, systemd-notify, systemd-nspawn, systemd-path, systemd-repart, systemd-resolve (到 resolvectl 的符号链接), systemd-run, systemd-socket-activate, systemd-stdio-bridge, systemd-tmpfiles, systemd-tty-ask-password-agent, systemd-umount (到 systemd-mount 的符号链接), telinit (到 systemctl 的符号链接), timedatectl, 以及 udevadm

安装的库: libnss_myhostname.so.2, libnss_mymachines.so.2, libnss_resolve.so.2, libnss_systemd.so.2, libsystemd.so, libsystemd-shared-245.so (在 /lib/systemd 中), 以及 libudev.so

安装的目录: /etc/binfmt.d, /etc/init.d, /etc/kernel, /etc/modules-load.d, /etc/sysctl.d, /etc/systemd, /etc/tmpfiles.d, /etc/udev, /etc/xdg/systemd, /lib/systemd, /lib/udev, /usr/include/systemd, /usr/lib/binfmt.d, /usr/lib/kernel, /usr/lib/modules-load.d, /usr/lib/sysctl.d, /usr/lib/systemd, /usr/lib/tmpfiles.d, /usr/share/doc/systemd-245, /usr/share/factory, /usr/share/systemd, /var/lib/systemd, 以及 /var/log/journal

简要描述

bootctl	用于查询固件和启动管理器设置
busctl	用于探查和监视 D-bus 总线
coredumpctl	用于从 systemd 日志获取核心转储
halt	一般调用 **shutdown** 命令并传递 `-h` 选项，除非在运行级别已经为 0 时，则通知内核停止系统；在系统停止运行时它在 `/var/log/wtmp` 中进行记录
hostnamectl	用于查询和修改系统机器名和相关设置
init	在内核初始化硬件后第一个启动的进程，它接管引导过程，并根据它的配置文件启动所有进程。在本例中，它启动 systemd。
journalctl	用于查询 systemd 日志的内容
kernel-install	用于在 /boot 中添加或删除内核和 initramfs 映像文件。在 LFS 中，这项工作是手工完成的。
localectl	用于查询和修改系统 locale 和键盘布局设置
loginctl	用于探查和控制 systemd 登录管理器的状态
machinectl	用于探查和控制 systemd 虚拟机和容器注册管理器的状态
networkctl	用于探查和配置 systemd-networkd 管理的网络连接状态
portablectl	用于在本地系统附加或移除可移植服务

poweroff	告诉内核停止系统运行并关闭计算机 (见 **halt**)
reboot	告诉内核重启系统 (见 **halt**)
resolvconf	为 **systemd-resolved** 注册 DNS 服务器和域设置
resolvectl	向网络名称解析管理器发送控制命令，或解析域名，IPv4 和 IPv6 地址，DNS 记录，以及服务。
runlevel	报告当前系统运行级别和上一个运行级别，上一个运行级别被记录在 /var/run/utmp 中
shutdown	安全地关闭系统，向所有进程发送信号，并通知所有登录用户
systemctl	用于探查和控制 systemd 系统和服务管理器的状态
systemd-analyze	用于分析当前运行系统的引导性能，找出导致问题的 systemd 单元
systemd-ask-password	用于以命令行指定的消息向用户询问系统密码
systemd-cat	用于将进程的标准输出和错误输出重定向到系统日志
systemd-cgls	用树的形式递归地显示指定 Linux 控制组层次结构的内容
systemd-cgtop	显示本地 Linux 控制组层次结构中占用资源最多的，可以按 CPU 、内存和磁盘 I/O 负载排序
systemd-delta	用于确定并比较那些覆盖了 /usr 中默认值的 /etc 中的配置文件
systemd-detect-virt	确定系统是否在虚拟化环境中运行，并据此调节 udev。
systemd-escape	用于转义字符串，以便将其包含在 systemd 单元名中
systemd-hwdb	用于管理硬件数据库 (hwdb)
systemd-id128	生成和打印 id128 串
systemd-inhibit	用于在关机、休眠或待机抑制锁被锁定的情况下运行程序，在进程结束前防止关闭系统等动作。
systemd-machine-id-setup	被系统安装工具用于在安装时以随机生成的 ID 初始化 /etc/machine-id 中的机器 ID
systemd-mount	一个用于临时挂载或自动挂载驱动器的工具。
systemd-notify	被守护脚本用于通知 init 系统关于状态变化的信息
systemd-nspawn	用于在轻量级命名空间容器中运行命令或操作系统
systemd-path	用于查询系统和用户路径
systemd-repart	在 systemd 被作为 OS 映像使用 (例如在容器中) 时，用于在分区表中添加分区或增长分区大小。
systemd-resolve	用于解析域名，IPv4 和 IPv6 地址，DNS 资源记录，以及服务
systemd-run	用于创建一个临时的 .service 或 .scope 单元，并在其中运行指定命令。这对于验证 systemd 单元很有用。
systemd-socket-activate	用于监听 socket 服务，并在 socket 成功连接时启动进程。
systemd-tmpfiles	根据 tmpfiles.d 目录中的配置文件给定的文件格式和位置，创建、修改和清理易失性、临时性文件和目录
systemd-umount	卸载挂载点

systemd-tty-ask-password-agent	列出或处理等待中的 systemd 密码请求
telinit	告诉 **init** 切换到某个运行级别
timedatectl	用于查询和修改系统时钟及其设置
udevadm	通用 udev 管理工具，它控制 udevd 守护进程，提供 udev 数据库的信息，监视 uevent 事件，等待 uevent 事件结束，测试 udev 配置，或对于给定设备触发 uevent 事件
`libsystemd`	主要的 systemd 工具库
`libudev`	用于访问 udev 设备信息的库

8.71. D-Bus-1.12.18

D-bus 是一个消息总线系统，即应用程序之间互相通信的一种简单方式。D-Bus 提供一个系统守护进程 (负责 “添加了新硬件” 或 “打印队列发生改变” 等事件)，并对每个用户登录会话提供一个守护进程 (负责一般用户程序的进程间通信)。另外，消息总线被构建在一个通用的一对一消息传递网络上，它可以被任意两个程序用于直接通信 (不需通过消息总线守护进程)。

估计构建时间: 0.2 SBU
需要硬盘空间: 18 MB

8.71.1. 安装 D-Bus

准备编译 D-Bus:

```
./configure --prefix=/usr                       \
            --sysconfdir=/etc                   \
            --localstatedir=/var                \
            --disable-static                    \
            --disable-doxygen-docs              \
            --disable-xml-docs                  \
            --docdir=/usr/share/doc/dbus-1.12.18 \
            --with-console-auth-dir=/run/console
```

配置选项的含义:

`--with-console-auth-dir=/run/console`
该选项指定 ConsoleKit 认证目录位置。

编译该软件包:

```
make
```

该软件包有测试套件，但需要 LFS 没有包含的一些软件包。阅读 BLFS 手册中的 http://www.linuxfromscratch.org/blfs/view/svn/general/dbus.html，以查阅运行测试套件的说明。

安装该软件包:

```
make install
```

需要将共享库移动到 `/lib`，因此 `/usr/lib` 中的 `.so` 符号链接需要重新建立:

```
mv -v /usr/lib/libdbus-1.so.* /lib
ln -sfv ../../lib/$(readlink /usr/lib/libdbus-1.so) /usr/lib/libdbus-1.so
```

创建符号链接，使 D-Bus 和 systemd 使用同一个 `machine-id` 文件:

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

将 socket 文件从过时的 /var/run 移动到 /run:

```
sed -i 's:/var/run:/run:' /lib/systemd/system/dbus.socket
```

8.71.2. D-Bus 的内容

安装的程序: dbus-cleanup-sockets, dbus-daemon, dbus-launch, dbus-monitor, dbus-run-session, dbus-send, dbus-test-tool, dbus-update-activation-environment, 以及 dbus-uuidgen
安装的库: libdbus-1.{a,so}
安装的目录: /etc/dbus-1, /usr/include/dbus-1.0, /usr/lib/dbus-1.0, /usr/share/dbus-1, /usr/share/doc/dbus-1.12.18, 以及 /var/lib/dbus

简要描述

dbus-cleanup-sockets	用于清理目录中遗留的套接字
dbus-daemon	是 D-Bus 消息总线守护程序
dbus-launch	从 shell 脚本启动 **dbus-daemon**
dbus-monitor	监视通过一个 D-Bus 消息总线的消息
dbus-run-session	从 shell 脚本启动一个 **dbus-daemon** 的会话总线实例，并在该会话中启动给定程序
dbus-send	向 D-Bus 消息总线发送消息
dbus-test-tool	是一个帮助软件包测试 D-Bus 的工具
dbus-update-activation-environment	更新将会为 D-Bus 会话服务设置的环境变量
dbus-uuidgen	产生通用唯一识别码
`libdbus-1`	包含用于和 D-Bus 消息总线通信的 API 函数

8.72. Procps-ng-3.3.16

Procps-ng 软件包包含监视进程的程序。

估计构建时间: 0.1 SBU
需要硬盘空间: 16 MB

8.72.1. 安装 Procps-ng

准备编译 procps-ng:

```
./configure --prefix=/usr                            \
            --exec-prefix=                           \
            --libdir=/usr/lib                        \
            --docdir=/usr/share/doc/procps-ng-3.3.16 \
            --disable-static                         \
            --disable-kill                           \
            --with-systemd
```

配置选项的含义:

--disable-kill
该选项禁用 **kill** 命令的构建，Util-linux 软件包将安装它。

编译该软件包:

```
make
```

在 LFS 系统上，测试套件需要一些定制修改。删除一个在没有使用 tty 设备输入时会失败的测试，并修正另外两个。执行以下命令运行测试套件:

```
sed -i -r 's|(pmap_initname)\\\$|\1|' testsuite/pmap.test/pmap.exp
sed -i '/set tty/d' testsuite/pkill.test/pkill.exp
rm testsuite/pgrep.test/pgrep.exp
make check
```

安装该软件包:

```
make install
```

最后，将必要的库移动到 /usr 尚未挂载时也能访问的位置。

```
mv -v /usr/lib/libprocps.so.* /lib
ln -sfv ../../lib/$(readlink /usr/lib/libprocps.so) /usr/lib/libprocps.so
```

8.72.2. Procps-ng 的内容

安装的程序: free, pgrep, pidof, pkill, pmap, ps, pwdx, slabtop, sysctl, tload, top, uptime, vmstat, w, 以及 watch
安装的库: libprocps.so
安装的目录: /usr/include/proc 和 /usr/share/doc/procps-ng-3.3.16

简要描述

free 报告系统中可用和已用内存 (包括物理内存和交换空间) 的容量

pgrep 根据名称和其他属性查找进程
pidof 报告给定程序的 PID
pkill 根据名称和其他属性向进程发送信号
pmap 报告给定进程的内存映射
ps 列出正在运行的进程
pwdx 报告一个进程的当前工作目录
slabtop 实时显示内核 slab 缓存详细信息
sysctl 在运行时修改内核参数
tload 打印当前系统平均负载示意图
top 列出 CPU 占用最大的进程列表；它实时地提供处理器活动的连续概况
uptime 报告系统运行时间、登录用户数目和系统平均负载
vmstat 报告虚拟内存统计，给出进程、内存、分页、块输入输出 (IO)、 陷阱和 CPU 活动信息
w 显示当前登录用户和它们的登录地点、时间
watch 重复执行给定命令，显示其输出的第一页；这使得用户可以观察输出随时间的变化
`libprocps` 包含该软件包大多数程序使用的函数

8.73. Util-linux-2.35.2

Util-linux 软件包包含若干工具程序。这些程序中有处理文件系统、终端、分区和消息的工具。

估计构建时间: 1.3 SBU
需要硬盘空间: 254 MB

8.73.1. 安装 Util-linux

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

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

准备安装 Util-linux:

```
./configure ADJTIME_PATH=/var/lib/hwclock/adjtime   \
            --docdir=/usr/share/doc/util-linux-2.35.2 \
            --disable-chfn-chsh  \
            --disable-login      \
            --disable-nologin    \
            --disable-su         \
            --disable-setpriv    \
            --disable-runuser    \
            --disable-pylibmount \
            --disable-static     \
            --without-python
```

--disable 和 --without 选项防止一些警告，它们与一些 LFS 中不存在，或与其他软件包安装的程序不兼容的构建组件相关。

编译该软件包:

```
make
```

如果希望的话，以非 root 用户身份运行测试套件:

警告

以 root 用户身份运行测试套件可能对系统造成损害。为了运行它，内核配置选项 CONFIG_SCSI_DEBUG 必须在当前运行的系统中可用，且必须被构建为内核模块。直接将其构建为内核的一部分会导致系统无法引导。为了测试的完整覆盖，必须安装其他 BLFS 软件包。如果希望的话，可以在重启进入完整的 LFS 系统后，执行以下命令运行测试:

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

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

安装该软件包:

```
make install
```

8.73.2. Util-linux 的内容

安装的程序: addpart, agetty, blkdiscard, blkid, blkzone, blockdev, cal, cfdisk, chcpu, chmem, choom, chrt, col, colcrt, colrm, column, ctrlaltdel, delpart, dmesg, eject, fallocate, fdformat, fdisk, fincore, findfs, findmnt, flock, fsck, fsck.cramfs, fsck.minix, fsfreeze, fstrim, getopt, hexdump, hwclock, i386, ionice, ipcmk, ipcrm, ipcs, isosize, kill, last, lastb (到 last 的链接), ldattach, linux32, linux64, logger, look, losetup, lsblk, lscpu, lsipc, lslocks, lslogins, lsmem, lsns, mcookie, mesg, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, mountpoint, namei, nsenter, partx, pivot_root, prlimit, raw, readprofile, rename, renice, resizepart, rev, rfkill, rtcwake, script, scriptreplay, setarch, setsid, setterm, sfdisk, sulogin, swaplabel, swapoff (到 swapon 的链接), swapon, switch_root, taskset, ul, umount, uname26, unshare, utmpdump, uuidd, uuidgen, uuidparse, wall, wdctl, whereis, wipefs, x86_64, 以及 zramctl

安装的库: libblkid.so, libfdisk.so, libmount.so, libsmartcols.so, 以及 libuuid.so

安装的目录: /usr/include/blkid, /usr/include/libfdisk, /usr/include/libmount, /usr/include/libsmartcols, /usr/include/uuid, /usr/share/doc/util-linux-2.35.2, 以及 /var/lib/hwclock

简要描述

addpart 告知 Linux 内核有新的分区

agetty 打开 tty 端口，提示输入登录名，再启动 **login** 程序

blkdiscard 丢弃设备上的扇区

blkid 一个命令行工具，用于定位和打印块设备属性

blkzone 在给定块设备上运行 zone 命令

blockdev 允许用户从命令行调用块设备 ioctl

cal 显示简单的日历

cfdisk 操作给定设备的分区表

chcpu 修改 CPU 状态

chmem 配置内存

choom 显示和调整 OOM-killer 分数

chrt 操纵进程实时属性

col 过滤掉反向换行符

colcrt 为缺失加粗、半行等功能的终端过滤 **nroff** 输出

colrm 过滤掉给定列

column 将给定文件格式化为多栏

ctrlaltdel 将 Ctrl+Alt+Del 键组合的功能设定为硬复位或软复位

delpart 要求 Linux 内核删除分区

dmesg 转储内核引导消息

eject 弹出可移动媒体

fallocate 为文件预先分配空间
fdformat 低级格式化软盘
fdisk 操作给定设备的分区表
fincore 统计给定文件在内存中占用的页面数
findfs 根据通用唯一识别码 (UUID) 查找文件系统
findmnt 是 libmount 库的命令行接口，可以处理 mountinfo、fstab 和 mtab 文件
flock 获取文件锁，并在持有锁的情况下运行命令
fsck 用于检查或修复文件系统
fsck.cramfs 用于对给定设备上的 Cramfs 文件系统进行一致性检查
fsck.minix 用于对给定设备上的 Minix 文件系统进行一致性检查
fsfreeze 是内核驱动 ioctl 操作 FIFREEZE/FITHAW 的简单包装
fstrim 丢弃已挂载文件系统上未使用的块
getopt 解析给定命令行的选项
hexdump 以十六进制或其他给定格式转储文件
hwclock 读取或设置系统硬件时钟，它又被称为实时时钟 (RTC) 或基本输入输出系统 (BIOS) 时钟
i386 到 setarch 的符号链接
ionice 设定程序的 IO 调度类和优先级
ipcmk 创建多种 IPC 资源
ipcrm 删除给定的进程间通信 (IPC) 资源
ipcs 提供 IPC 状态信息
isosize 报告 ISO 9660 文件系统的大小
kill 向进程发送信号
last 显示哪些用户最后登录 (或登出)，在 /var/log/wtmp 文件中反向搜索；它也会显示系统引导、关闭和运行级别变化记录
lastb 显示 /var/log/btmp 记录的失败登录企图
ldattach 为串口线附加行规则
linux32 到 setarch 的符号链接
linux64 到 setarch 的符号链接
logger 将给定消息记入系统日志
look 显示以给定字符串开始的行
losetup 设定和控制回环设备
lsblk 以树状格式列出所有或给定块设备的信息
lscpu 打印 CPU 体系结构信息
lsipc 打印系统当前部署的 IPC 设施的信息
lslocks 列出本地系统锁
lslogins 列出用户、组和系统账户的信息

lsmem 列出可用内存的范围和它们的在线状态
lsns 列出命名空间
mcookie 为 **xauth** 创建魔术 cookie (128位随机十六进制数)
mesg 控制其他用户能否向当前用户终端发送消息
mkfs 在设备 (一般是硬盘分区) 上创建文件系统
mkfs.bfs 创建 Santa Cruz Operations (SCO) bfs 文件系统
mkfs.cramfs 创建 cramfs 文件系统
mkfs.minix 创建 Minix 文件系统
mkswap 将给定文件或设备初始化为交换空间
more 在屏幕上分页文本的过滤器
mount 将给定设备上的文件系统挂载到文件系统树结构中的给定目录
mountpoint 检查目录是否为挂载点
namei 显示给定目录名中的符号链接
nsenter 在其他程序的命名空间中运行程序
partx 告知内核磁盘分区的存在性和编号
pivot_root 将当前进程的根文件系统设为给定文件系统
prlimit 获取和设定进程资源限制
raw 将 Linux raw 字符设备绑定到块设备
readprofile 读取内核性能分析信息
rename 重命名给定文件，将给定字符串替换为另一个字符串
renice 修改正在运行的进程的优先级
resizepart 要求 Linux 内核改变分区大小
rev 反转给定文件的每一行
rkfill 用于启用或禁用无线设备的工具
rtcwake 进入睡眠状态，直到给定的唤醒时间
script 记录终端会话打字机文档
scriptreplay 根据计时信息重放终端会话打字机文档
setarch 在新程序环境中修改系统报告的体系结构，并设置进程执行域信息
setsid 在新会话中运行给定程序
setterm 设定终端属性
sfdisk 一个分区表修改器
sulogin 允许 `root` 登录；一般在系统进入单用户模式时由 **init** 执行
swaplabel 允许修改交换空间 UUID 和标签
swapoff 禁止在文件或设备上进行分页交换
swapon 启用文件或设备上的分页交换，或列出当前用于交换的设备和文件
switch_root 将另一个文件系统切换为挂载树的根

tailf	跟踪日志文件的增长；显示日志文件的最后 10 行，并在日志文件有新记录时继续显示它们
taskset	获取或设置进程 CPU 亲和性
ul	将下划线转换为在当前终端中表示下划线的 escape 序列的过滤器
umount	断开文件系统与系统文件目录树的连接
uname26	到 setarch 的符号链接
unshare	在某些命名空间与父进程脱离的情况下运行程序
utmpdump	以更加用户友好的格式显示给定登录文件
uuidd	UUID 库使用的守护进程，用于安全、确保唯一性地生成 UUID
uuidgen	创建新的 UUID。每个新的 UUID 可以被合理地认为在本地系统和其他系统上，在过去和未来，都是唯一的
uuidparse	用于解析统一标识符的工具
wall	显示文件或标准输入 (默认值) 的内容到所有登录用户的终端
wdctl	显示硬件看门狗电路状态
whereis	报告给定命令二进制文件、源代码文件和 man 页面的位置
wipefs	从设备上擦除文件系统签名
x86_64	到 setarch 的符号链接
zramctl	设定和控制 zram (压缩内存盘) 设备的程序
`libblkid`	包含设备识别和标识提取子程序
`libfdisk`	包含操作分区表的子程序
`libmount`	包含挂载和解挂块设备的子程序
`libsmartcols`	包含以表格形式在屏幕上输出的辅助子程序
`libuuid`	包含为对象生成唯一标识符，使它在本地系统以外也可以访问的子程序

8.74. E2fsprogs-1.45.6

E2fsprogs 软件包包含处理 ext2 文件系统的工具。此外它也支持 ext3 和 ext4 日志文件系统。

估计构建时间: 4.4 SBU on a spinning disk, 1.7 SBU on an SSD
需要硬盘空间: 106 MB

8.74.1. 安装 E2fsprogs

E2fsprogs 文档推荐在源代码目录树中的一个子目录内构建该软件包：

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

准备编译 E2fsprogs：

```
../configure --prefix=/usr           \
             --bindir=/bin           \
             --with-root-prefix=""   \
             --enable-elf-shlibs     \
             --disable-libblkid      \
             --disable-libuuid       \
             --disable-uuidd         \
             --disable-fsck
```

配置选项的含义：

--with-root-prefix="" 和 --bindir=/bin
某些程序 (例如 **e2fsck** 程序) 被认为是关键的。在 /usr 尚未挂载等情况下，这些程序仍然必须可用。它们应该放置在 /lib 和 /sbin 等目录中。如果没有向 e2fsprogs 配置脚本传递该参数，这些程序会被安装到 /usr 目录。

--enable-elf-shlibs
该选项表示创建该软件包中一些程序使用的共享库。

--disable-*
该选项防止 e2fsprogs 构建和安装 libuuid 和 libblkid 库，uuidd 守护程序，以及 **fsck** 包装器，因为 Util-linux 会安装更新的版本。

编译该软件包：

```
make
```

执行以下命令，以运行测试：

```
make check
```

在机械硬盘上，测试使用的时间略超过 4 SBU。在 SSD 上测试时间短很多 (只要约 1.5 SBU)。

安装该软件包：

```
make install
```

将安装好的静态库变为可写的，以便之后移除调试符号：

```
chmod -v u+w /usr/lib/{libcom_err,libe2p,libext2fs,libss}.a
```

该软件包安装了一个 gzip 压缩的 .info 文件，却没有更新系统的 dir 文件。执行以下命令解压该文件，并更新系统 dir 文件:

```
gunzip -v /usr/share/info/libext2fs.info.gz
install-info --dir-file=/usr/share/info/dir /usr/share/info/libext2fs.info
```

如果需要，执行以下命令创建并安装一些额外的文档:

```
makeinfo -o      doc/com_err.info ../lib/et/com_err.texinfo
install -v -m644 doc/com_err.info /usr/share/info
install-info --dir-file=/usr/share/info/dir /usr/share/info/com_err.info
```

8.74.2. E2fsprogs 的内容

安装的程序: badblocks, chattr, compile_et, debugfs, dumpe2fs,e2freefrag, e2fsck, e2image, e2label, e2mmpstatus, e2scrub, e2scrub_all, e2undo, e4crypt, e4defrag, filefrag, fsck.ext2, fsck.ext3, fsck.ext4, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mkfs.ext4, mklost+found, resize2fs, 以及 tune2fs

安装的库: libcom_err.so, libe2p.so, libext2fs.so, 以及 libss.so

安装的目录: /usr/include/e2p, /usr/include/et, /usr/include/ext2fs, /usr/include/ss, /usr/lib/e2fsprogs, /usr/share/et, 以及 /usr/share/ss

简要描述

badblocks 在一个设备 (一般是磁盘分区) 上搜索坏块

chattr 在 ext2 文件系统上修改文件属性，也适用于 ext3 文件系统，即ext2 文件系统的日志版本

compile_et 一个错误表编译器； 它将包含错误编号名称和消息的表转化成 C 源代码，以和 com_err 库一起使用

debugfs 一个文件系统调试器；可以检验并修改 ext2 文件系统的状态

dumpe2fs 打印给定设备上文件系统的超级块和块组信息

e2freefrag 报告可用空间碎片信息

e2fsck 用于检查或修复 ext2 文件系统和 ext3 文件系统

e2image 用于将 ext2 文件系统关键数据保存到文件

e2label 显示或修改给定设备上的 ext2 文件系统标签

e2mmpstatus 检查 ext4 文件系统的 MMP 状态

e2scrub 检查某个已挂载的 ext2，ext3 或 ext4 文件系统

e2scrub_all 检查所有已挂载的 ext2，ext3 或 ext4 文件系统

e2undo 重放设备上找到的 ext2/ext3/ext4 文件系统撤销日志 undo_log [可以用于撤销 e2fsprogs 程序的失败操作。]

e4crypt Ext4 文件系统加密工具

e4defrag ext4 文件系统在线碎片整理器

filefrag 报告特定文件碎片化程度

fsck.ext2 默认情况下检查 ext2 文件系统，是 **e2fsck** 的硬链接

fsck.ext3 默认情况下检查 ext3 文件系统，是 **e2fsck** 的硬链接

fsck.ext4 默认情况下检查 ext4 文件系统，是 **e2fsck** 的硬链接

logsave 将命令输出保存到日志文件

lsattr 列出 ext2 文件系统上的文件属性

mk_cmds 将包含命令名称和帮助信息的表格转换成 C 源代码文件，以便和 libss 子系统库一起使用

mke2fs 在给定设备上创建 ext2 或 ext3 文件系统

mkfs.ext2 默认情况下创建 ext2 文件系统，是 **mke2fs** 的硬链接

mkfs.ext3 默认情况下创建 ext3 文件系统，是 **mke2fs** 的硬链接

mkfs.ext4 默认情况下创建 ext4 文件系统，是 **mke2fs** 的硬链接

mklost+found 用于在创建 lost+found 目录；它在 ext2 文件系统上为该目录预先分配磁盘块，以减轻 **e2fsck** 的负担

resize2fs 可以用于扩大或压缩 ext2 文件系统

tune2fs 调整 ext2 文件系统的可调参数

libcom_err 公用错误显示子程序

libe2p 被 **dumpe2fs**、**chattr**, 和 **lsattr** 使用

libext2fs 包含允许用户级程序操纵 ext2 文件系统的子程序

libss 被 **debugfs** 使用

8.75. 关于调试符号

许多程序和库在默认情况下被编译为带有调试符号的二进制文件 (通过使用 **gcc** 的 `-g` 选项)。这意味着在调试这些带有调试信息的程序和库时，调试器不仅能给出内存地址，还能给出子程序和变量的名称。

然而，插入这些调试符号会显著增大程序或库的体积。下面是一些表现调试符号占用空间的例子：

· 一个有调试符号的 **bash** 二进制程序：1200 KB

· 一个没有调试符号的 **bash** 二进制程序：480 KB

· 带有调试符号的 Glibc 和 GCC 文件 (`/lib` 和 `/usr/lib` 目录中)：87 MB

· 没有调试符号的 Glibc 和 GCC 文件：16 MB

以上文件大小的值可能随编译器和 C 运行库的版本而变化，但在比较带调试符号和不带调试符号的程序时，它们文件大小的差距通常达到 2 至 5 倍。

由于大多数用户永远不会用调试器调试系统软件，可以通过移除它们的调试符号，回收大量磁盘空间。下一节展示如何从系统程序和库中移除所有调试符号。

8.76. 再次移除调试符号

本节是可选的。如果系统不是为程序员设计的，也没有调试系统软件的计划，可以通过从二进制程序和库移除调试符号，将系统的体积减小约 2 GB。除了无法再调试全部软件外，这不会造成任何不便。

大多数使用以下命令的用户不会遇到什么困难。但是，如果打错了命令，很容易导致新系统无法使用，因此在运行 **strip** 命令前，最好备份 LFS 系统的当前状态。

首先将一些库的调试符号保存在单独的文件中。之后在 BLFS 中，如果使用 valgrind 或 gdb 运行退化测试，则需要这些调试信息的存在。

```
save_lib="ld-2.31.so libc-2.31.so libpthread-2.31.so libthread_db-1.0.so"

cd /lib

for LIB in $save_lib; do
    objcopy --only-keep-debug $LIB $LIB.dbg
    strip --strip-unneeded $LIB
    objcopy --add-gnu-debuglink=$LIB.dbg $LIB
done

save_usrlib="libquadmath.so.0.0.0 libstdc++.so.6.0.28
             libitm.so.1.0.0 libatomic.so.1.2.0"

cd /usr/lib

for LIB in $save_usrlib; do
    objcopy --only-keep-debug $LIB $LIB.dbg
    strip --strip-unneeded $LIB
    objcopy --add-gnu-debuglink=$LIB.dbg $LIB
done

unset LIB save_lib save_usrlib
```

现在即可移除程序和库的调试符号：

```
find /usr/lib -type f -name \*.a \
   -exec strip --strip-debug {} ';'

find /lib /usr/lib -type f -name \*.so* ! -name \*dbg \
   -exec strip --strip-unneeded {} ';'

find /{bin,sbin} /usr/{bin,sbin,libexec} -type f \
    -exec strip --strip-all {} ';'
```

这里会有很多文件被报告为格式无法识别。这些警告可以安全地忽略。它们表明那些文件是脚本文件，而不是二进制文件。

8.77. 清理系统

最后，清理在执行测试的过程中遗留的一些文件：

```
rm -rf /tmp/*
```

现在需要登出，并使用新的 chroot 命令行重新进入 chroot 环境。从现在起，在退出并重新进入 chroot 环境时，要使用下面的修改过的 chroot 命令：

```
logout

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

这里不再使用 +h 选项，因为所有之前安装的程序都已经替换成了最终版本，可以进行散列。

如果解除了虚拟内核文件系统的挂载，必须通过手动或重启系统的方式重新挂载它们，保证在进入 chroot 时它们已经挂载好。第 7.3.2 节 “挂载和填充 /dev”和第 7.3.3 节 “挂载虚拟内核文件系统”已经说明了这一过程。

在本章的前几节中，有几个静态库的安装没有被禁止，目的是满足一些软件包的退化测试需要。这些库来自于 binutils、bzip2、e2fsprogs、flex、libtool 和 zlib。如果您希望的话，可以现在删除它们：

```
rm -f /usr/lib/lib{bfd,opcodes}.a
rm -f /usr/lib/libctf{,-nobfd}.a
rm -f /usr/lib/libbz2.a
rm -f /usr/lib/lib{com_err,e2p,ext2fs,ss}.a
rm -f /usr/lib/libltdl.a
rm -f /usr/lib/libfl.a
rm -f /usr/lib/libz.a
```

在 /usr/lib 和 /usr/libexec 目录中还有一些扩展名为 .la 的文件。它们是 "libtool 档案" 文件。正如我们已经讨论过的，它们在链接到共享库，特别是使用 autotools 以外的构建系统时，是不必要，甚至有害的。执行以下命令删除它们：

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

如果希望了解更多关于 libtool 档案文件的信息，参阅 BLFS 章节 "About Libtool Archive (.la) files"。

最后，移除上一章开始时创建的临时 'tester' 用户账户。

```
userdel -r tester
```

第 9 章 系统配置

9.1. 概述

本章讨论配置文件和 systemd 服务。第一，展示设定网络通常需要的配置文件。

· 第 9.2 节 “一般网络配置”。

· 第 9.2.3 节 “配置系统主机名”。

· 第 9.2.4 节 “自定义 /etc/hosts 文件”/

第二，讨论影响设备正确配置的问题。

· 第 9.3 节 “设备和模块管理概述”。

· 第 9.4 节 “管理设备”。

第三，展示如何配置系统时钟和键盘布局。

· 第 9.5 节 “配置系统时钟”。

· 第 9.6 节 “配置 Linux 控制台”。

第四，简要介绍用户登录系统时使用的脚本和配置文件。

· 第 9.7 节 “配置系统 Locale”。

· 第 9.8 节 “创建 /etc/inputrc 文件”。

最后，讨论如何配置 systemd 行为。

· 第 9.10 节 “Systemd 使用和配置”。

9.2. 一般网络配置

本节只适用于需要配置网卡的情况。

9.2.1. 网络接口配置文件

从 209 版本开始，systemd 提供一个名为 **systemd-networkd** 的网络配置守护进程，它能够用于基础网络配置。另外，自 213 版本起，可以用 **systemd-resolved** 代替静态 `/etc/resolv.conf` 文件处理域名解析。这两个服务在默认情况下都是启用的。

systemd-networkd (以及 **systemd-resolved**)的配置文件可以放置在 `/usr/lib/systemd/network` 或 `/etc/systemd/network` 中。`/etc/systemd/network` 中的配置文件优先级高于 `/usr/lib/systemd/network` 中的配置文件。有三种类型的配置文件：`.link`、 `.netdev` 和 `.network` 文件。要获得它们的详细描述和内容示例，参阅 `systemd-link(5)`、 `systemd-netdev(5)` 和 `systemd-network(5)` man 手册页面。

9.2.1.1. 网络设备命名

Udev 一般根据系统物理特征为网卡分配接口名，例如 enp2s1。如果您不确定接口名是什么，可以在引导您的系统后，运行 **ip link** 命令。

对于多数系统，每种连接类型只有一个网络接口。例如，有线连接的经典接口名是 eth0，而无线连接的接口名一般是 wifi0 或 wlan0。

如果您偏爱经典或自定义网络接口名，可以使用三种不同方式：

· 覆盖 udev 提供默认策略的 .link 文件：

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

· 手动创建命名架构，例如将网络接口命名为“internet0”、“dmz0” 或 “lan0”。为此，在 /etc/systemd/network 中创建 .link 文件，为您的一个、一些或全部网络接口直接选择名称，或选择更好的命名架构。例如：

```
cat > /etc/systemd/network/10-ether0.link << "EOF"
[Match]
# 将 MAC 地址替换为适用于您的网络设备的值
MACAddress=12:34:45:78:90:AB

[Link]
Name=ether0
EOF
```

参阅 man 页面 systemd.link(5) 获得更多信息。

· 在 /boot/grub/grub.cfg 的内核命令行中传递选项 net.ifnames=0。

9.2.1.2. 静态 IP 配置

以下命令为静态 IP 设置创建一个基本的配置文件 (使用 systemd-networkd 和 systemd-resolved)。

```
cat > /etc/systemd/network/10-eth-static.network << "EOF"
[Match]
Name=<网络设备名>

[Network]
Address=192.168.0.2/24
Gateway=192.168.0.1
DNS=192.168.0.1
Domains=<您的域名>
EOF
```

如果您有多个 DNS 服务器，可以在配置文件中创建多个 DNS 项。如果您希望使用静态 /etc/resolv.conf 文件，则不要在配置文件中包含 DNS 和 Domains 项。

9.2.1.3. DHCP 配置

以下命令为 IPv4 DHCP 配置创建基本配置文件：

```
cat > /etc/systemd/network/10-eth-dhcp.network << "EOF"
[Match]
Name=<网络设备名>

[Network]
DHCP=ipv4

[DHCP]
UseDomains=true
EOF
```

9.2.2. 创建 /etc/resolv.conf 文件

如果要将系统连接到 Internet，它需要某种域名服务 (DNS) 名称解析方式，以将 Internet 域名解析为 IP 地址，或将 IP 地址解析为域名。最好的方法是将 ISP 或网络管理员提供的 DNS 服务器 IP 地址写入 /etc/resolv.conf。

9.2.2.1. systemd-resolved 配置

注意

如果使用其他方式配置网络接口 (例如 ppp 或 network-manager 等)，或使用了某种本地解析器 (如 bind，dnsmasq，或者 unbound 等)，或其他任何生成 /etc/resolv.conf 的软件 (如 resolvconf)，则不应使用 **systemd-resolved** 服务。

在使用 **systemd-resolved** 进行 DNS 配置时，它创建文件 /run/systemd/resolve/resolv.conf。在 /etc 中创建符号链接以使用生成的文件：

```
ln -sfv /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

9.2.2.2. 静态 resolv.conf 配置

如果希望使用静态的 /etc/resolv.conf 执行以下命令创建它：

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

domain <您的域名>
nameserver <您的主要域名服务器 IP 地址>
nameserver <您的次要域名服务器 IP 地址>

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

可以省略 domain 语句，或使用一条 search 语句代替它。阅读 resolv.conf 的 man 页面了解更多细节。

将 **<域名服务器的 IP 地址>** 替换为您的网络环境下最合适的 DNS 服务器 IP 地址。这里往往会写入不止一个 DNS 服务器 (需要次要服务器作为后备)。如果您只需要或只希望使用一个 DNS 服务器，可以删除文件中的第二个 nameserver 行。可以写入本地网络路由器的 IP 地址。也可以使用 Google 公用 DNS 服务器作为域名服务器，它们的 IP 地址在下面给出。

注意

Google 公用 DNS 服务器的 IPv4 地址是 8.8.8.8 和 8.8.4.4，IPv6 地址是 2001:4860:4860::8888 和 2001:4860:4860::8844。

9.2.3. 配置系统主机名

在引导过程中，/etc/hostname 被用于设定系统主机名。

执行以下命令，创建 /etc/hostname 文件，并输入一个主机名：

```
echo "<lfs>" > /etc/hostname
```

<lfs> 需要被替换为赋予该计算机的名称。不要在这里输入全限定域名 (FQDN)，它应该被写入 /etc/hosts 文件。

9.2.4. 自定义 /etc/hosts 文件

选择一个全限定域名 (FQDN)，和可能的别名，以供 /etc/hosts 文件使用。如果使用静态 IP 地址，您还需要确定要使用的 IP 地址。hosts 文件条目的语法是：

```
IP_地址 主机名.域名 别名
```

除非该计算机可以从 Internet 访问 (即拥有一个注册域名，并分配了一个有效的 IP 地址段 —— 多数用户没有分配有效 IP)，确认使用的 IP 地址属于私网 IP 范围。有效的范围是：

```
私网地址范围                          公共前缀长度
10.0.0.1 - 10.255.255.254           8
172.x.0.1 - 172.x.255.254           16
192.168.y.1 - 192.168.y.254         24
```

x 可以是 16-31 之间的任何数字。y 可以是 0-255 之间的任何数字。

有效的私网 IP 地址的一个例子是 192.168.1.1。与之对应的 FQDN 可以是 lfs.example.org。

即使没有网卡，也要提供一个有效的 FQDN。某些程序，如 MTA，需要它才能正常工作。

执行以下命令，创建 /etc/hostname 文件：

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

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

# End /etc/hosts
EOF
```

其中 <192.168.0.2>，<FQDN>，以及<主机名> 的值需要为特定使用环境和需求进行修改 (如果系统或网络管理员分配了 IP 地址，且本机将被连接到现有的网络中)。可以略去别名 (alias)，它们不是必要的。

::1 是 127.0.0.1 在 IPv6 中的对应，即 IPv6 回环接口。

9.3. 设备和模块管理概述

在第 8 章中，我们在构建 systemd 时安装了 Udev 软件包。在我们详细讨论它的工作原理之前，首先按时间顺序简要介绍历史上曾经使用过的设备管理方式。

传统的 Linux 系统通常使用静态地创建设备，即在 /dev 下创建大量设备节点 (有时有数千个节点)，无论对应的硬件设备是否真的存在。一般通过 **MAKEDEV** 脚本完成这一工作，它包含以相关的主设备号和次设备号，为世界上可能存在的每个设备建立节点的大量 **mknod** 命令。

使用 Udev，则只有那些被内核检测到的设备才会获得为它们创建的设备节点。由于这些设备节点在每次引导系统时都会重新创建，它们被储存在 devtmpfs 文件系统中 (一个虚拟文件系统，完全驻留在系统内存)。设备节点不需要太多空间，它们使用的系统内存可以忽略不计。

9.3.1. 历史

在 2000 年 2 月，一个称为 devfs 的新文件系统被合并到 2.3.46 版内核中，并在 2.4 系列稳定内核中可用。尽管它本身曾经存在于内核源代码中，但这种设备节点动态创建方法从未得到内核核心开发者的大力支持。

devfs 实现机制的主要问题是它处理设备的检测、创建和命名的方式，其中最致命的或许最后一项，即设备节点命名方式。通常认为，如果设备名称是可配置的，那么设备命名策略应该由系统管理员，而不是某个 (某些)特定开发者决定。devfs 还受到其设计中固有的竞争条件的严重影响，在不对内核进行大量修改的前提下无法修复这一问题。由于缺乏维护，它早已被标记为过时特性，最终在 2006 年 6 月被从内核中移除。

在不稳定的 2.5 系列内核开发过程中，加入了一个新的虚拟文件系统，称为 sysfs，并在 2.6 系列稳定内核中发布。sysfs 的工作是将系统硬件配置信息导出给用户空间进程，有了这个用户空间可见的配置描述，就可能开发一种 devfs 的用户空间替代品。

9.3.2. Udev 实现

9.3.2.1. Sysfs

前面已经简要提到了 sysfs 文件系统。有些读者可能好奇，sysfs 是如何知道系统中存在哪些设备，以及应该为它们使用什么设备号的。答案是，那些编译到内核中的驱动程序在它们的对象被内核检测到时，直接将它们注册到 sysfs (内部的 devtmpfs)。对于那些被编译为模块的驱动程序，注册过程在模块加载时进行。只要 sysfs 文件系统被挂载好 (位于 /sys)，用户空间程序即可使用驱动程序注册在 sysfs 中的数据，Udev 就能够使用这些数据对设备进行处理 (包括修改设备节点)。

9.3.2.2. 设备节点的创建

内核通过 devtmpfs 直接创建设备文件，任何希望注册设备节点的驱动程序都要通过 devtmpfs (经过驱动程序核心) 实现。当一个 devtmpfs 实例被挂载到 /dev 时，设备节点将被以固定的名称、访问权限和所有者首次创建。

很快，内核会向 **udevd** 发送一个 uevent。根据 /etc/udev/rules.d，/lib/udev/rules.d，以及 /run/udev/rules.d 目录中文件指定的规则，**udevd** 将为设备节点创建额外的符号链接，修改其访问权限，所有者，或属组，或者修改该对象的 **udevd** 数据库条目 (名称)。

以上三个目录中的规则都被编号，且这三个目录的内容将合并处理。如果 **udevd** 找不到它正在创建的设备对应的规则，它将会沿用 devtmpfs 最早使用的配置。

9.3.2.3. 模块加载

编译为内核模块的设备驱动程序可能有内建的别名。别名可以通过 **modinfo** 程序查询，它通常和该模块支持的设备的总线相关标识符有关。例如，snd-fm801 驱动程序支持厂商 ID 为 0x1319，设备 ID 为 0x0801 的 PCI 设备，其别名为“pci:v00001319d00000801sv*sd*bc04sc01l*”。对于多数设备，总线驱动程序会通过 sysfs 导出应该处理该设备的驱动程序别名，例如 /sys/bus/pci/devices/0000:00:0d.0/modalias 文件应该包含字符串“pci:v00001319d00000801sv00001319sd00001319bc04sc01i00”。Udev 附带的默认规则会导致 **udevd** 调用 **/sbin/modprobe** 时传递 MODALIAS uevent 环境变量 (它的值应该和 sysfs 中modalias 文件的内容相同)，从而加载那些在通配符扩展后别名与这个字符串匹配的模块。

在本例中，这意味着除了 snd-fm801 外，过时 (且不希望) 的 forte 如果可用，也会被加载。之后将介绍防止加载不希望的驱动程序的方法。

内核本身也能够在需要时为网络协议，文件系统，以及 NLS 支持加载模块。

9.3.2.4. 处理热插拔/动态设备

当您插入一个设备，例如通用串行总线 (USB) MP3 播放器时，内核能够发现该设备现在已经被连接到系统，并生成一个 uevent 事件。之后 **udevd** 像前面描述的一样，处理该 uevent 事件。

9.3.3. 加载模块和创建设备时的问题

在自动创建设备节点时，可能出现一些问题。

9.3.3.1. 内核模块没有自动加载

Udev 只加载拥有总线特定别名，且总线驱动程序正确地向 sysfs 导出了必要别名的模块。如果情况不是这样，您应该考虑用其他方法加载模块。在 Linux-5.7.2 中，已知 Udev 可以加载编写正确的 INPUT，IDE，PCI，USB，SCSI，SERIO，以及 FireWire 驱动程序。

为了确定您需要的设备驱动程序是否包含 Udev 支持，以模块名为参数运行 **modinfo** 命令。然后试着在 /sys/bus 中找到设备对应的目录，并检查其中是否有 modalias 文件。

如果 modalias 文件存在于 sysfs 中，说明驱动程序支持该设备，并能够直接和设备交互，但却没有正确的别名。这是驱动程序的 bug，您需要不通过 Udev 直接加载驱动，并等待这个问题日后被解决。

如果 modalias 文件不存在于 /sys/bus 下的对应目录中，说明内核开发者尚未对该总线类型增加 modalias 支持。在 Linux-5.7.2 中，ISA 总线不受支持。只能等待这个问题在日后被解决。

Udev 根本不会尝试加载“包装器” 驱动程序，比如 snd-pcm-oss 等，或 loop 等非硬件驱动程序。

9.3.3.2. 内核模块没有自动加载，且 Udev 不尝试加载它

如果“包装器”仅仅用于增强其他模块的功能 (例如，snd-pcm-oss 增强 snd-pcm 的功能，使 OSS 应用程序能够使用声卡)，需要配置 **modprobe**，使其在 Udev 加载被包装的模块时，自动加载包装器。为此，需要将“softdep”行添加到对应的 /etc/modprobe.d/<filename>.conf 中。例如：

```
softdep snd-pcm post: snd-pcm-oss
```

注意“softdep”命令也允许 pre: 依赖项，或混合使用 pre: 和 post: 依赖项。参阅 modprobe.d(5) man 手册页面，了解更多关于“softdep”语法和功能的信息。

9.3.3.3. Udev 加载了不希望的模块

不要构建该模块，或者在 /etc/modprobe.d/blacklist.conf 文件中禁用它。以 forte 为例，下面一行禁用了该模块：

```
blacklist forte
```

被禁用的模块仍然可以通过直接执行 **modprobe** 手动加载。

9.3.3.4. Udev 创建了错误的设备或错误的符号链接

这一般是由于一条规则意外地匹配了某个设备。例如，一个写得不好的规则可能同时匹配到 SCSI 磁盘 (正确的) 和对应厂商的 SCSI 通用设备 (不正确的)。找到引起问题的规则，并通过 **udevadm info** 的帮助，将它进一步细化。

9.3.3.5. Udev 规则工作不可靠

这可能是前一个问题的另一个表现形式。如果不是，而且您的规则使用了 sysfs 属性，这个问题可能由内核计时问题引发，这类问题需要在新的内核版本中修复。目前，您可以创建一条规则以等待被使用的 sysfs 属性，并将它附加到 /etc/udev/rules.d/10-wait-for-sysfs.rules 文件中 (如果不存在就创建一个文件)，绕过这个问题。如果您通过这种方法解决了问题，请通知 LFS 开发邮件列表。

9.3.3.6. Udev 没有创建设备

以下内容假设驱动程序已经被编译到内核中，或作为模块被加载，而且您已经检查过并确认 Udev 没有创建命名错误的设备。

如果驱动程序没有将它的信息导出到 sysfs，Udev 就无法获得创建设备节点必需的信息。这种问题往往出现在内核源代码树以外的第三方驱动程序中。这时，需要在 /lib/udev/devices 中使用正确的主设备号和次设备号，创建一个静态设备节点 (参考内核文档中的 devices.txt 或第三方驱动厂商提供的文档)，该静态设备节点将被复制到 /dev，**udev** 会自动完成复制。

9.3.3.7. 重启后设备命名顺序随机变化

这是由于 Udev 从设计上就是并行加载模块的，因此无法预测加载顺序。这个问题永远也不会被 “修复”。您不应该指望内核提供稳定的设备命名，而是应该创建您自己的规则，以根据设备的一些稳定属性，例如设备序列号或 Udev 安装的一些 *_id 工具的输出，来创建具有稳定名称的符号链接。可以参考第 9.4 节 “管理设备”和第 9.2 节 “一般网络配置”中的例子。

9.3.4. 扩展阅读

以下链接包含了一些额外的帮助文档:

- A Userspace Implementation of devfs http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf
- The sysfs Filesystem http://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf

9.4. 管理设备

9.4.1. 处理重复设备

正如第 9.3 节 “设备和模块管理概述”中所述，那些功能相同的设备在 /dev 中的顺序是随机的。例如，如果您有一个 USB 摄像头和一个电视棒，有时 /dev/video0 会指向摄像头，/dev/video1 指向电视棒，而有时在重启后这个顺序正好颠倒过来。对于所有除了声卡和网卡以外的设备，该问题都可以通过创建自定义持久化符号链接的 Udev 规则来解决。对于网卡的解决方案在第 9.2 节 “一般网络配置”中单独描述，而声卡配置可以在 BLFS 中找到。

对于您的每个可能有这类问题的设备 (即使在您当前使用的 Linux 发行版上并没有问题)，找到 /sys/class 或 /sys/block 中的对应目录。对于视频设备，目录可能是 /sys/class/video4linux/videoX。找出能够唯一确认该设备的属性 (通常是厂商和产品 ID，或者序列号):

```
udevadm info -a -p /sys/class/video4linux/video0
```

然后编写创建符号链接的规则，例如:

```
cat > /etc/udev/rules.d/83-duplicate_devs.rules << "EOF"

# 摄像头和电视棒的持久化符号链接
KERNEL=="video*", ATTRS{idProduct}=="1910", ATTRS{idVendor}=="0d81", SYMLINK+="webcam"
KERNEL=="video*", ATTRS{device}=="0x036f",  ATTRS{vendor}=="0x109e", SYMLINK+="tvtuner"

EOF
```

结果是，/dev/video0 和 /dev/video1 仍然会随机指向电视棒和摄像头 (因此不应直接使用它们)，但符号链接 /dev/tvtuner 和 /dev/webcam 总会指向正确设备。

9.5. 配置系统时钟

本节讨论如何配置 **systemd-timedated** 系统服务，它的作用是配置系统时钟和时区。

如果您不确定您的硬件时钟是否设置为 UTC，运行 **hwclock --localtime --show** 命令，它会显示硬件时钟给出的当前时间。如果这个时间和您的手表显示的一致，则说明硬件时钟被设定为本地时间。相反，如果 **hwclock** 输出的时间不是本地时间，则硬件时钟很可能被设定为 UTC 时间。根据您的时区，在 **hwclock** 显示的时间上加减对应的小时数，进行进一步的验证。例如，如果您现在处于莫斯科时区，即 GMT -0700，在本地时间上加 7 小时，再进行比较。

systemd-timedated 读取 /etc/adjtime，并根据其内容将硬件时钟设定为 UTC 或本地时间。

果您的硬件时钟设置为本地时间，以下列内容创建 /etc/adjtime 文件：

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

如果 /etc/adjtime 在初次引导时不存在，**systemd-timedated** 会假设硬件时钟使用 UTC，并据此调整该文件。

您也可以使用 **timedatectl** 工具告诉 **systemd-timedated** 您的硬件时钟是 UTC 还是本地时间：

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

timedatectl 也能修改系统时间和时区。

如果要修改系统时间，执行以下命令：

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

硬件时钟也会同时被更新。

要修改当前时区，执行以下命令：

```
timedatectl set-timezone TIMEZONE
```

您可以通过运行以下命令查看可用的时区列表：

```
timedatectl list-timezones
```

注意

注意 **timedatectl** 命令只能用于 systemd 引导的系统。

9.5.1. 网络时钟同步

从版本 213 开始，systemd 附带了一个名为 **systemd-timesyncd** 的守护程序，可以用于将系统时间与远程 NTP 服务器同步。

该守护程序没有被设计为替代现有成熟的 NTP 守护程序，而是一个仅仅实现了 SNTP 协议的客户端，可以用于一些不太复杂的任务，或是资源紧张的系统。

从 systemd 版本 216 开始，**systemd-timesyncd** 守护进程被默认启用。如果希望禁用它，执行以下命令：

```
systemctl disable systemd-timesyncd
```

可以在 /etc/systemd/timesyncd.conf 中修改 **systemd-timesyncd** 使用的服务器。

注意，当系统时钟设定为本地时间时，**systemd-timesyncd** 不会更新硬件时钟。

9.6. 配置 Linux 控制台

本节讨论如何配置 **systemd-vconsole-setup** 系统服务，它负责配置虚拟控制台字体和控制台键盘映射。

systemd-vconsole-setup 服务从 /etc/vconsole.conf 文件中读取配置信息。它根据配置确定使用的键映射和控制台字体。一些与特定语言相关的 HOWTO 文档可以帮助您进行配置，参阅 http://www.tldp.org/HOWTO/HOWTO-INDEX/other-lang.html。浏览 **localectl list-keymaps** 输出的可用控制台键映射列表。在 /usr/share/consolefonts 目录中寻找可用的控制台字体。

/etc/vconsole.conf 文件的每一行都应该符合格式：变量名="值"，

KEYMAP
: 该变量指定键盘的键映射表。如果没有设定，默认为 us。

KEYMAP_TOGGLE
: 该变量可以用于配置第二切换键盘映射，没有默认设定值。

FONT
: 该变量指定虚拟控制台使用的字体。

FONT_MAP
: 该变量指定控制台字体映射。

FONT_UNIMAP
: 该变量指定 Unicode 字体映射。

下面的例子可以用于德文键盘和控制台：

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

在系统运行时，可以使用 **localectl** 工具修改 KEYMAP 变量值：

```
localectl set-keymap MAP
```

注意

请注意 **localectl** 只能在 systemd 引导的系统上使用。

也可以通过指定 **localectl** 工具的参数，修改 X11 键盘布局，模型，变体和选项设置：

```
localectl set-x11-keymap 布局 [模型] [变体] [选项]
```

如果需要列出可用的 **localectl set-x11-keymap** 参数值，可以使用下列参数运行 **localectl** 命令：

list-x11-keymap-models
列出已知的 X11 键盘映射模型。

list-x11-keymap-layouts
列出已知的 X11 键盘映射布局。

list-x11-keymap-variants
列出已知的 X11 键盘映射变体。

list-x11-keymap-options
列出已知的 X11 键盘映射选项。

注意

上面给出的参数都需要 BLFS 中的 XKeyboard-Config 软件包。

9.7. 配置系统 Locale

下面将创建的 `/etc/locale.conf` 设定本地语言支持需要的若干环境变量，正确设定它们可以带来以下好处：

· 程序输出被翻译成本地语言

· 字符被正确分类为字母、数字和其他类别，这对于使 **bash** 正确接受命令行中的非 ASCII 本地非英文字符来说是必要的

· 根据所在地区惯例排序字母

· 适用于所在地区的默认纸张尺寸

· 正确格式化货币、时间和日期值

将下面的 `<ll>` 替换为所需语言的双字符代号 (例如“en”)，`<CC>` 替换为国家或地区的双字符代号 (例如“GB”)，`<charmap>` 替换为您选定的 locale 的标准字符映射。另外，还可以加入 “@euro” 等可选修饰符。

Glibc 支持的所有 locale 可以用以下命令列出：

```
locale -a
```

字符映射可能有多个别名，例如“ISO-8859-1”也可以称为“iso8859-1”或者“iso88591”。某些程序不能正确处理一些别名 (例如，只识别“UTF-8”，不能识别“utf8”)，因此在多数情况下，为了保险起见，最好使用 locale 的规范名称。为了确定规范名称，执行以下命令，将 `<locale 名>` 替换成 **locale -a** 对于您希望的 locale 的输出 (以“en_GB.iso88591”为例)。

```
LC_ALL=<locale 名> locale charmap
```

对于“en_GB.iso88591”locale，以上命令输出：

```
ISO-8859-1
```

这样就最终确定 locale 应设置为“en_GB.ISO-8859-1”。在将以上启发方法获得的 locale 添加到 Bash 启动文件之前，一定要进行下列测试：

```
LC_ALL=<locale 名> locale language
LC_ALL=<locale 名> locale charmap
LC_ALL=<locale 名> locale int_curr_symbol
LC_ALL=<locale 名> locale int_prefix
```

以上命令应该输出语言名称，选定 locale 使用的字符编码，本地货币符号，以及所在国家或地区的国际电话区号。如果以上某个命令失败并输出类似下面这样的消息，意味着您的 locale 在第 6 章中没有安装，或者不被 Glibc 的默认安装支持。

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

如果出现了这种消息，您应该用 **localedef** 命令安装所需的 locale，或重新选择一个不同的 locale。后文假设 Glibc 没有输出类似错误消息。

某些 LFS 以外的软件包可能缺乏对您选择的 locale 的支持，例如 X 库 (X 窗口系统的一部分)，它在您的 locale 与它内部文件中的字符映射表名不完全匹配时，会输出以下错误消息：

```
Warning: locale not supported by Xlib, locale set to C
```

某些情况下 Xlib 期望字符映射以带有规范连字符的大写形式给出，例如应该使用 “ISO-8859-1”而不是“iso88591”。有时也可以通过去除 locale 规范中的字符映射部分找到合适的规范，可以通过运行 **locale charmap** 确认。例如，您需要将“de_DE.ISO-8859-15@euro”替换成“de_DE@euro”，以获得 Xlib 能够识别的 locale。

其他软件包在 locale 名不符合它们的期望时可能工作不正常(但未必输出错误消息)。在这种情况下，探索一下其他 Linux 发行版是如何支持您的 locale 的，可以得到一些有用的信息。

在确定了正确的 locale 设置后，创建 /etc/locale.conf 文件：

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

您也可以使用 systemd 的 **localectl** 工具修改 /etc/locale.conf。如果希望使用 **localectl** 创建以上例子中的 locale.conf，运行：

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

您也可以指定其他语言相关的环境变量，例如 LANG，LC_CTYPE，LC_NUMERIC，或 **locale** 输出的其他环境变量，用空格将它们分割即可。例如，将 LANG 设置为 en_US.UTF-8，LC_CTYPE 设置为 en_US：

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

注意

请注意 **localectl** 只能在使用 systemd 引导的系统中使用。

“C” (默认 locale) 和“en_US”(推荐美式英语用户使用的 locale) 是不同的。“C” locale 使用 US-ASCII 7 位字符集，并且将最高位为 1 的字节视为无效字符。因此，**ls** 等命令会将它们替换为问号。另外，如果试图用 Mutt 或 Pine 发送包含这些字符的邮件，会发出不符合 RFC 标准的消息 (发出邮件的字符集会被标为“未知 8 位”)。因此，您只能在确信自己永远不会使用 8 位字符时才能使用“C” locale。

9.8. 创建 /etc/inputrc 文件

inputrc 文件是 Readline 库的配置文件，该库在用户从终端输入命令行时提供编辑功能。它的工作原理是将键盘输入翻译为特定动作。Readline 被 Bash 和大多数其他 shell，以及许多其他程序使用。

多数人不需要 Readline 的用户配置功能，因此以下命令创建全局的 /etc/inputrc 文件，供所有登录用户使用。如果您之后决定对于某个用户覆盖掉默认值，您可以在该用户的主目录下创建 .inputrc 文件，包含需要修改的映射。

关于更多如何编写 inputrc 文件的信息，参考 **info bash** 中 Readline Init File 一节。**info readline** 也是一个很好的信息源。

下面是一个通用的全局 inputrc 文件，包含解释一些选项含义的注释。注意注释不能和命令写在同一行。执行以下命令创建该文件：

```
cat > /etc/inputrc << "EOF"
# Begin /etc/inputrc
# Modified by Chris Lynn <roryo@roryo.dynup.net>

# Allow the command prompt to wrap to the next line
set horizontal-scroll-mode Off

# Enable 8bit input
set meta-flag On
set input-meta On

# Turns off 8th bit stripping
set convert-meta Off

# Keep the 8th bit for display
set output-meta On

# none, visible or audible
set bell-style none

# All of the following map the escape sequence of the value
# contained in the 1st argument to the readline specific functions
"\eOd": backward-word
"\eOc": forward-word

# for linux console
"\e[1~": beginning-of-line
"\e[4~": end-of-line
"\e[5~": beginning-of-history
"\e[6~": end-of-history
"\e[3~": delete-char
"\e[2~": quoted-insert

# for xterm
"\eOH": beginning-of-line
"\eOF": end-of-line

# for Konsole
"\e[H": beginning-of-line
"\e[F": end-of-line

# End /etc/inputrc
EOF
```

9.9. 创建 /etc/shells 文件

shells 文件包含系统登录 shell 的列表，应用程序使用该文件判断 shell 是否合法。该文件中每行指定一个 shell，包含该 shell 相对于目录树根 (/) 的路径。

例如 **chsh** 使用该文件判断一个非特权用户是否可以修改自己的登录 shell。如果命令没有在 /etc/shell 中找到，就会拒绝修改操作。

这个文件对某些程序是必要的。例如 GDM 在找不到 /etc/shells 时不会填充登录界面，FTP 守护进程通常禁止那些使用未在此文件列出的终端的用户登录。

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

/bin/sh
/bin/bash

# End /etc/shells
EOF
```

9.10. Systemd 使用和配置

9.10.1. 基础设置

/etc/systemd/system.conf 文件包含一组控制 systemd 基本功能的选项。默认文件中所有条目都被注释掉，并标明了默认值。可以在这里修改日志级别，以及其他一些基本日志设定。参阅 systemd-system.conf(5) man 手册页面了解每个选项的详细信息。

9.10.2. 禁用引导时自动清屏

Systemd 的默认行为是在引导过程结束时清除屏幕。如果希望的话，您可以运行以下命令，修改这一行为：

```
mkdir -pv /etc/systemd/system/getty@tty1.service.d

cat > /etc/systemd/system/getty@tty1.service.d/noclear.conf << EOF
[Service]
TTYVTDisallocate=no
EOF
```

您总是可以用 root 身份运行 **journalctl -b** 命令，查阅引导消息。

9.10.3. 禁止将 tmpfs 挂载到 /tmp

默认情况下，/tmp 将被挂载 tmpfs 文件系统。如果不希望这样，可以执行以下命令覆盖这一行为：

```
ln -sfv /dev/null /etc/systemd/system/tmp.mount
```

或者，如果希望使用一个单独的 /tmp 分区，在 /etc/fstab 中为其添加一个条目。

警告

如果使用了单独的 /tmp 分区，不要创建上面的符号链接。这会导致根文件系统 (/) 无法重新挂载为可读写，使得系统在引导后不可用。

9.10.4. 配置文件自动创建和删除

有一些创建或删除文件、目录的服务：

- systemd-tmpfiles-clean.service
- systemd-tmpfiles-setup-dev.service
- systemd-tmpfiles-setup.service

它们的系统配置文件位于 /usr/lib/tmpfiles.d/*.conf。本地配置文件位于 /etc/tmpfiles.d。/etc/tmpfiles.d 中的文件覆盖 /usr/lib/tmpfiles.d 中的同名文件。参阅 tmpfiles.d(5) man 手册页面，了解配置文件格式的细节。

注意 /usr/lib/tmpfiles.d/*.conf 文件的语法较难理解。例如，删除 /tmp 目录下文件的默认规则是文件 /usr/lib/tmpfiles.d/tmp.conf 的一行：

```
q /tmp 1777 root root 10d
```

类别字段 q 表示创建一个带有配额的子卷，它实际上只适用于 btrfs 文件系统。它引用类别 v，类别 v 又引用类别 d (目录)。对于类别 d，会在目录不存在时自动创建它，并根据配置文件调整其权限和所有者。如果 age 参数被指定，该目录中较老的文件会被自动清理。

如果默认参数不符合您的期望，您可以将文件复制到 /etc/tmpfiles.d 目录，再编辑复制得到的副本。例如：

```
mkdir -p /etc/tmpfiles.d
cp /usr/lib/tmpfiles.d/tmp.conf /etc/tmpfiles.d
```

9.10.5. 覆盖系统服务默认行为

Systemd 单元的参数可以通过在 /etc/systemd/system 中创建一个包含配置文件的目录而覆盖。例如：

```
mkdir -pv /etc/systemd/system/foobar.service.d

cat > /etc/systemd/system/foobar.service.d/foobar.conf << EOF
[Service]
Restart=always
RestartSec=30
EOF
```

参阅 systemd.unit(5) man 手册页面获取更多信息。在创建配置文件后，执行 **systemctl daemon-reload** 和 **systemctl restart foobar**，激活对服务进行的修改。

9.10.6. 调试引导过程

与 SysVinit 或 BSD 风格 init 系统不同，systemd 使用统一格式处理不同种类的引导文件 (或称为单元)。命令 **systemctl** 能够启用、禁用单元文件，或控制、查询单元文件的状态。以下是一些常用的命令：

- **systemctl list-units -t <service> [--all]**: 列出已加载的服务 (service) 类型单元文件。
- **systemctl list-units -t <target> [--all]**: 列出已加载的引导目标 (target) 类型单元文件。
- **systemctl show -p Wants <multi-user.target>**: 显示所有依赖于 multi-user 引导目标的单元，引导目标 (target)是一种和 SysVinit 中运行级别 (runlevel) 地位相同的特殊单元文件。
- **systemctl status <servicename.service>**: 显示名为 servicename 的服务的状态。如果没有同名的其他类型单元文件，可以省略 .service 后缀。其他类型的单元文件有 .socket 文件 (它创建一个监听套接字，提供和 inetd/xinetd 类似的功能)。

9.10.7. 使用 systemd 日志

(默认情况下) 在使用 systemd 引导的系统上，systemd-journald 服务负责处理日志，它取代了传统的 Unix syslog 守护进程。如果您希望的话，也可以添加一个普通 syslog 守护进程，它和 systemd-journald 可以一起工作。systemd-journald 程序将日志项储存为二进制格式，而不是纯文本日志文件。为了解析日志文件，需要使用 systemd 提供的 **journalctl** 命令。下面是该命令的常见用法:

- **journalctl -r**: 按时间顺序，倒序显示所有日志内容。
- **journalctl -u UNIT**: 显示与给定单元文件 UNIT 关联的日志。
- **journalctl -b[=ID] -r**: 按时间倒序，显示自上次引导以来 (或编号为 ID 的引导中) 的所有日志。
- **journalctl -f**: 提供类似 tail -f 的功能 (不断将新日志项输出到屏幕)。

9.10.8. 处理核心转储

核心转储在调试崩溃的程序时非常有用，特别是对于守护进程崩溃的情况。在 systemd 引导的系统上，核心转储由 **systemd-coredump** 处理。它会在日志中记录核心转储，并且将核心转储文件本身存储到 /var/lib/systemd/coredump 中。如果要获取和处理核心转储文件，可以使用 **coredumpctl** 工具。下面给出它的常用命令的示例:

- **coredumpctl -r**: 按时间顺序，倒序显示所有核心转储记录。
- **coredumpctl -1 info**: 显示最近一次核心转储的信息。
- **coredumpctl -1 debug**: 将最后一次核心转储加载到 GDB 中。

核心转储可能使用大量磁盘空间。为了限制核心转储使用的最大磁盘空间，可以在 /etc/systemd/coredump.conf.d 中创建一个配置文件。例如:

```
mkdir -pv /etc/systemd/coredump.conf.d

cat > /etc/systemd/coredump.conf.d/maxuse.conf << EOF
[Coredump]
MaxUse=5G
EOF
```

参阅 man 手册页面 systemd-coredump(8)，coredumpctl(1)，以及 coredump.conf.d(5) 了解更多信息。

9.10.9. 持续运行进程

从 systemd 的 230 版本开始，在用户会话结束时，所有用户进程都被杀死，即使使用了 nohup 或 daemon() 、setsid 等函数也不例外。这是开发者有意做出的修改，将传统的宽松环境改为更加严格的环境。如果您需要让持续运行的程序 (例如 **screen** 或 **tmux**) 在用户会话结束后保持运行，这项新的行为会导致问题。有三种方法使得这类驻留进程在用户会话结束后继续运行:

- 仅为选定的用户启用进程驻留：普通用户有执行命令 **loginctl enable-linger** 启用进程驻留的权限，管理员可以使用带 user 参数的该命令，为特定用户启用进程驻留。在启用进程驻留后，可以使用 **systemd-run** 命令启动持续运行的进程。例如，**systemd-run --scope --user /usr/bin/screen**。如果您为您的用户启用了进程驻留，则 user@.service 将持续运行，甚至在所有登录会话关闭后仍然运行，而且会在系统引导时自动启动。这种方法的好处是可以显式地允许或禁止进程在用户会话结束后继续运行，但却破坏了和 **nohup** 等工具，和使用 daemon() 函数的工具的兼容性。
- 为整个系统启用进程驻留：您可以在将KillUserProcesses=no 设置行加入 /etc/systemd/logind.conf，为所有用户全局地启用进程驻留。它的好处是允许所有用户继续使用旧方法，但无法进行明确控制。
- 在编译时禁用该功能：您可以在构建 systemd 时传递参数 -Ddefault-kill-user-process=no 给 **meson**，使得 systemd 默认启用进程驻留。这完全禁用了 systemd 在会话结束时杀死用户进程的功能。

第 10 章 使 LFS 系统可引导

10.1. 概述

现在应该配置 LFS 系统，使其可以引导了。本章讨论创建 /etc/fstab 文件，为新的 LFS 系统构建内核，以及安装 GRUB 引导加载器，使得系统引导时可以选择进入 LFS 系统。

10.2. 创建 /etc/fstab 文件

一些程序使用 /etc/fstab 文件，以确定哪些文件系统是默认挂载的，和它们应该按什么顺序挂载，以及哪些文件系统在挂载前必须被检查 (确定是否有完整性错误)。参考以下命令，创建一个新的文件系统表：

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

# 文件系统     挂载点       类型     选项                转储  检查
#                                                             顺序

/dev/<xxx>     /            <fff>    defaults            1     1
/dev/<yyy>     swap         swap     pri=1               0     0

# End /etc/fstab
EOF
```

将 <xxx>、 <yyy> 和 <fff> 替换为适用于您的系统的值，例如 sda2、sda5 和 ext4。参阅 **man 5 fstab** 了解该文件中 6 个域的详细信息。

在挂载来源于 MS-DOS 或 Windows 的文件系统 (如 vfat、ntfs、smbfs、cifs、iso9660、udf) 时，需要一个特殊的挂载选项 —— utf8，才能正常解析文件名中的非 ASCII 字符。对于非 UTF-8 locale，选项 iocharset 的值应该和您的 locale 字符集设定一致，但改写成内核可以识别的写法。该选项能够正常工作的前提是，将相关的字符集定义 (在内核配置选项的 File Systems -> Native Language Support 子菜单中) 编译到内核中，或构建为内核模块。然而，如果使用了 UTF-8 locale，对应的 iocharset=utf8 会导致文件系统变得大小写敏感。为了避免这个问题，在使用 UTF-8 locale 时，需要用特殊选项 utf8 代替 iocharset=utf8。另外，vfat 和 smbfs 文件系统还需要“codepage”选项，它应该被设定为您的语言在 MS-DOS 下的代码页编号。例如，为了挂载一个 USB 闪存盘，一个 ru_RU.KOI8-R 用户应该在 /etc/fstab 中对应于闪存盘的行添加下列挂载选项：

```
noauto,user,quiet,showexec,codepage=866,iocharset=koi8r
```

相应的，ru_RU.UTF-8 用户应该使用下列选项：

```
noauto,user,quiet,showexec,codepage=866,utf8
```

注意此时使用的 iocharset 默认为 iso8859-1 (这保证文件系统是大小写不敏感的)，而 utf8 选项告诉内核使用 UTF-8 编码转换文件名，这样它们就能在 UTF-8 locale 中被正确解析。

也可以在内核配置中，为一些文件系统指定默认 codepage 和 iocharset 选项值。相关的配置参数名为“Default NLS Option” (CONFIG_NLS_DEFAULT)，“Default Remote NLS Option” (CONFIG_SMB_NLS_DEFAULT)，“Default codepage for FAT” (CONFIG_FAT_DEFAULT_CODEPAGE)，以及“Default iocharset for FAT” (CONFIG_FAT_DEFAULT_IOCHARSET)。无法在编译内核时为 ntfs 文件系统指定这些默认值。

在某些硬盘上，通过将 barrier=1 挂载选项加入 /etc/fstab，可以使得 ext3 文件系统在发生电源故障时更可靠。为了检查磁盘驱动器是否支持该选项，在可用的磁盘驱动器上运行 hdparm。例如：

```
hdparm -I /dev/sda | grep NCQ
```

如果输出内容不为空，说明该选项可用。

注意：基于逻辑卷管理 (LVM) 的分区不能使用 barrier 选项。

10.3. Linux-5.7.2

Linux 软件包包含 Linux 内核。

估计构建时间: 4.4 - 66.0 SBU (一般约 6 SBU)
需要硬盘空间: 960 - 4250 MB (一般约 1100 MB)

10.3.1. 安装内核

构建内核需要三步 —— 配置、编译、安装。阅读内核源代码树中的 `README` 文件，了解不同于本手册的内核配置方法。

运行以下命令，准备编译内核:

```
make mrproper
```

该命令确保内核源代码树绝对干净，内核开发组建议在每次编译内核前运行该命令。尽管内核源代码树在解压后应该是干净的，但这并不完全可靠。

下面通过菜单界面配置内核，阅读 http://www.linuxfromscratch.org/hints/downloads/files/kernel-configuration.txt 了解关于内核配置的一般信息。BLFS 的某些软件包需要特定内核配置，阅读 http://www.linuxfromscratch.org/blfs/view/svn/longindex.html#kernel-config-index 了解它们。另外在 http://www.kroah.com/lkn/ 也有一些关于配置和构建内核的信息。

注意

一个较好的初始内核配置可以通过运行 **make defconfig** 获得。它会考虑您的当前系统体系结构，将基本内核配置设定到较好的状态。

一定要按照以下列表启用/禁用/设定其中列出的内核特性，否则系统可能不能正常工作，甚至根本无法引导：

```
General setup -->
   [*] Control Group support [CONFIG_CGROUPS]
   [ ] Enable deprecated sysfs features to support old userspace tools [CONFIG_SYSFS_DEPREC
   [*] Configure standard kernel features (expert users) [CONFIG_EXPERT] --->
      [*] open by fhandle syscalls [CONFIG_FHANDLE]
   [ ] Auditing support [CONFIG_AUDIT]
Processor type and features  --->
   [*] Enable seccomp to safely compute untrusted bytecode [CONFIG_SECCOMP]
Firmware Drivers  --->
   [*] Export DMI identification via sysfs to userspace [CONFIG_DMIID]
Networking support  --->
  Networking options  --->
   <*> The IPv6 protocol [CONFIG_IPV6]
Device Drivers  --->
  Generic Driver Options  --->
   [ ] Support for uevent helper [CONFIG_UEVENT_HELPER]
   [*] Maintain a devtmpfs filesystem to mount at /dev [CONFIG_DEVTMPFS]
   Firmware Loader --->
      [ ] Enable the firmware sysfs fallback mechanism [CONFIG_FW_LOADER_USER_HELPER]
File systems  --->
   [*] Inotify support for userspace [CONFIG_INOTIFY_USER]
   <*> Kernel automounter support (supports v3, v4, and v5) [CONFIG_AUTOFS_FS]
  Pseudo filesystems  --->
   [*] Tmpfs POSIX Access Control Lists [CONFIG_TMPFS_POSIX_ACL]
   [*] Tmpfs extended attributes [CONFIG_TMPFS_XATTR]
```

注意

尽管 “The IPv6 Protocol” (IPv6 协议支持) 并不是严格要求的，但是 systemd 开发者强烈推荐启用它。

注意

如果您的硬件平台使用 UEFI，则 “make defconfig” 命令也会自动加入一些 EFI 相关的内核选项。

为了允许从宿主系统的 UEFI 引导环境引导 LFS 内核，您必须选择一个内核选项：

```
Processor type and features  --->
   [*]   EFI stub support  [CONFIG_EFI_STUB]
```

在 LFS 中管理 UEFI 环境的较完整说明包含在 lfs-uefi.txt 中，它位于 http://www.linuxfromscratch.org/hints/downloads/files/lfs-uefi.txt。

上述配置选项的含义：

Support for uevent helper
如果启用了该选项，它可能干扰 Udev/Eudev 的设备管理。

Maintain a devtmpfs
该选项会使内核自动创建设备节点，即使 Udev 没有运行。Udev 之后才在这些设备节点的基础上运行，管理它们的访问权限并为它们建立符号链接。所有 Udev/Eudev 用户都需要启用该选项。

```
make menuconfig
```

以上命令中可选的 make 环境变量及含义：

LANG=<host_LANG_value> LC_ALL=
它们根据宿主使用的 locale 建立 locale 设定。在 UTF-8 Linux 文本终端下，有时必须这样做才能正确绘制基于 ncurses 的配置菜单接口。
在这种情况下，一定要将 <host_LANG_value> 替换成宿主环境中的 $LANG 变量值。您也可以使用宿主环境中 $LC_ALL 或 $LC_CTYPE 的值代替。

某些情况下，**make oldconfig** 更为合适。阅读 README 文件了解更多信息。

如果希望的话，也可以将宿主系统的内核配置文件 .config 拷贝到解压出的 linux-5.7.2 目录 (前提是可以找到该文件)。然而我们不推荐这样做，一般来说，浏览整个配置目录，并从头创建内核配置是更好的选择。

编译内核映像和模块：

```
make
```

如果要使用内核模块，可能需要在 /etc/modprobe.d 中写入模块配置。讨论模块和内核配置的信息位于第 9.3 节 “设备和模块管理概述”和 linux-5.7.2/Documentation 目录下的内核文档中。另外 modprobe.d(5) 也可以作为参考。

如果内核配置使用了模块，安装它们：

```
make modules_install
```

在内核编译完成后，需要进行额外步骤完成安装，一些文件需要拷贝到 /boot 目录中。

小心

如果宿主系统有单独的 /boot 分区，需要将这些文件拷贝到该分区中。最简单的方法是将宿主系统的 /boot (在 chroot 之外) 绑定到 /mnt/lfs/boot 再拷贝文件，在宿主系统中，以 root 身份执行：

```
mount --bind /boot /mnt/lfs/boot
```

指向内核映像的路径可能随机器平台的不同而变化。下面使用的文件名可以依照您的需要改变，但文件名的开头应该保持为 vmlinuz，以保证和下一节描述的引导过程自动设定相兼容。下面的命令假定是机器是 x86 体系结构：

```
cp -iv arch/x86/boot/bzImage /boot/vmlinuz-5.7.2-lfs-20200622-systemd
```

System.map 是内核符号文件，它将内核 API 的每个函数入口点和运行时数据结构映射到它们的地址。它被用于调查分析内核可能出现的问题。执行以下命令安装该文件：

```
cp -iv System.map /boot/System.map-5.7.2
```

内核配置文件 .config 由上述的 **make menuconfig** 步骤生成，包含编译好的内核的所有配置选项。最好能将它保留下来以供日后参考：

```
cp -iv .config /boot/config-5.7.2
```

安装 Linux 内核文档：

```
install -d /usr/share/doc/linux-5.7.2
cp -r Documentation/* /usr/share/doc/linux-5.7.2
```

需要注意的是，在内核源代码目录中可能有不属于 root 的文件。在以 root 身份解压源代码包时 (就像我们在 chroot 环境中所做的那样)，这些文件会获得它们之前在软件包创建者的计算机上的用户和组 ID。这一般不会造成问题，因为在安装后通常会删除源代码目录树。然而，Linux 源代码目录树一般会被保留较长时间，这样创建者当时使用的用户 ID 就可能被分配给本机的某个用户，导致该用户拥有内核源代码的写权限。

注意

之后在 BLFS 中安装软件包时往往需要修改内核配置。因此，和其他软件包不同，我们在安装好内核后可以不移除源代码树。

如果要保留内核源代码树，切换到内核源代码目录，执行 **chown -R 0:0**，以保证 linux-5.7.2 目录中所有文件都属于 root。

警告

有的内核文档建议创建符号链接 /usr/src/linux 指向内核源代码目录，这仅仅适用于 2.6 系列之前的内核。在 LFS 系统上绝对不要创建它，因为在构建完基本 LFS 系统后，它可能在您构建其他软件包时引起问题。

警告

在系统 include 目录 (即 /usr/include) 中的内核头文件应该总是与构建 Glibc 时使用的内核头文件一致，即保持为第 5.4 节 “Linux-5.7.2 API 头文件”中安装的净化头文件。换句话说，永远不要用原始内核头文件，或其他版本内核的净化头文件替换它们。

10.3.2. 配置 Linux 内核模块加载顺序

多数情况下 Linux 内核模块可以自动加载，但有时需要指定加载顺序。负责加载内核模块的程序 **modprobe** 和 **insmod** 从 /etc/modprobe.d 下的配置文件中读取加载顺序，例如，如果 USB 驱动程序 (ehci_hcd、ohci_hcd 和 uhci_hcd) 被构建为模块，则必须按照先加载 echi_hcd，再加载 ohci_hcd 和 uhci_hcd 的正确顺序，才能避免引导时出现警告信息。

为此，执行以下命令创建文件 /etc/modprobe.d/usb.conf:

```
install -v -m755 -d /etc/modprobe.d
cat > /etc/modprobe.d/usb.conf << "EOF"
# Begin /etc/modprobe.d/usb.conf

install ohci_hcd /sbin/modprobe ehci_hcd ; /sbin/modprobe -i ohci_hcd ; true
install uhci_hcd /sbin/modprobe ehci_hcd ; /sbin/modprobe -i uhci_hcd ; true

# End /etc/modprobe.d/usb.conf
EOF
```

10.3.3. Linux 的内容

安装的文件: config-5.7.2, vmlinuz-5.7.2-lfs-20200622-systemd, 以及 System.map-5.7.2
安装的目录: /lib/modules 和 /usr/share/doc/linux-5.7.2

简要描述

config-5.7.2	包含所有内核配置选项的值
vmlinuz-5.7.2-lfs-20200622-systemd	Linux 系统的引擎，在启动计算机时，它是操作系统中被最早加载的部分。它检测并初始化计算机硬件，将它们以目录树的形式提供给软件，并将单个 CPU 封装成多任务系统，使多个用户程序看上去在同时执行。
System.map-5.7.2	地址和符号列表；它将内核函数和数据结构映射为入口点和地址

10.4. 使用 GRUB 设定引导过程

10.4.1. 概述

警告

如果您不小心错误地配置了 GRUB，可能导致您的系统完全无法使用，除非使用 CD-ROM 或可引导的 USB 存储器等备用引导设备。本节不是引导您的 LFS 系统的唯一方案，您可能只要修改现有的启动加载器 (如 Grub-Legacy、GRUB2 或 LILO) 配置即可引导 LFS。

您务必保证自己拥有一个紧急引导磁盘，它在计算机不可用 (无法引导) 时能够 “抢修” 计算机。如果您现在还没有引导设备，您可以执行以下命令创建一个。在运行下列命令前，您需要跳到 BLFS，安装包含 **xorriso** 的 libisoburn 软件包：

```
cd /tmp
grub-mkrescue --output=grub-img.iso
xorriso -as cdrecord -v dev=/dev/cdrw blank=as_needed grub-img.iso
```

注意

为了在使用 UEFI 的宿主系统上引导 LFS，在构建内核时，需要启用 CONFIG_EFI_STUB 功能，正如上一节所述。不过，可以使用 GRUB2 在没有该功能的情况下引导 LFS，前提是在系统 BIOS 设定中关闭 UEFI 模式和安全引导 (Secure Boot) 功能。关于在 UEFI 环境下引导 LFS 的的详细信息，可以参阅 lfs-uefi.txt hint，它位于 http://www.linuxfromscratch.org/hints/downloads/files/lfs-uefi.txt。

10.4.2. GRUB 命名惯例

GRUB 使用一种独特的命名结构，为驱动器和分区命名。分区名的形式为 (hdn,m)，这里 n 是硬盘驱动器编号，m 是分区编号。硬盘驱动器编号从 0 开始，但分区号对于主分区来说从 1 开始，而对于扩展分区来说从 5 开始。例如，分区 sda1 在 GRUB 中的名字是 (hd0, 1)，而 sda3 的名字是 (hd1, 3)。和 Linux 不同，GRUB 不认为 CD-ROM 驱动器属于硬盘驱动器。例如，如果在 hdb 上有一个 CD-ROM 驱动器，而 hdc 上有第二个硬盘驱动器，则第二个硬盘驱动器仍然名为 hd1。

10.4.3. 设定 GRUB 配置

GRUB 的工作方式是，将数据写入硬盘的第一个物理磁道。这里不属于任何文件系统，在启动时，第一个物理磁道中的程序从引导分区加载 GRUB 模块，默认在 /boot/grub 中查找模块。

引导分区的位置由负责进行配置的用户自己决定，作者推荐创建一个小的 (建议大小为 200 MB) 分区，专门存放引导信息。这样，不同的 Linux 系统 (无论是 LFS 还是商业发行版) 在启动时和启动后都能访问相同的引导文件。如果您选择这样做，您需要挂载这个单独的分区，将 /boot 中已有的文件 (例如上一节中构建的内核) 移动到新的分区中。之后，解除该分区的挂载，并将它挂载为 /boot。另外，还要注意更新 /etc/fstab。

直接使用 LFS 分区也是可以的，但这样在配置多系统启动时比较麻烦。

根据以上信息，确定 LFS 根分区 (或 boot 分区，如果使用了独立的 boot 分区) 的名称。下面假设 LFS 根分区 (或 boot 分区) 是 sda2。

将 GRUB 文件安装到/boot/grub 并设定引导磁道：

警告

以下命令会覆盖当前启动引导器，如果这不是您希望的，不要运行该命令。例如，如果您使用第三方启动引导器管理主引导记录 (MBR)。

```
grub-install /dev/sda
```

注意

如果系统是使用 UEFI 引导的，**grub-install** 会试图为 x86_64-efi 目标安装文件，但它们并未在第 6 章中安装。如果出现了这类问题，请在以上命令中添加 `--target i386-pc` 选项。

10.4.4. 创建 GRUB 配置文件

生成 /boot/grub/grub.cfg:

```
cat > /boot/grub/grub.cfg << "EOF"
# Begin /boot/grub/grub.cfg
set default=0
set timeout=5

insmod ext2
set root=(hd0,2)

menuentry "GNU/Linux, Linux 5.7.2-lfs-20200622-systemd" {
        linux   /boot/vmlinuz-5.7.2-lfs-20200622-systemd root=/dev/sda2 ro
}
EOF
```

注意

从 GRUB的视角来看，内核文件的位置相对于它使用的分区。如果您使用了单独的 /boot 分区，需要从上面的 linux 行删除 /boot，然后修改 set root 行，指向 /boot 分区。

GRUB 是一个很强大的程序，它提供了非常多的选项，可以支持多种设备、操作系统和分区类型，还有很多用于定制启动屏幕、 声音、鼠标输入等的选项。这些选项的细节超过了本书的范围，不予讨论。

小心

有一个命令 grub-mkconfig 被用于自动创建配置文件。它使用 /etc/grub.d 中的脚本创建新配置文件，这会覆盖您手动编写的 grub.cfg。这些脚本主要是为非源代码发行版设计的，在 LFS 中不推荐使用。但是，如果您安装了商业发行版，它很可能在发行版中被运行，记得备份 grub.cfg 以防它被覆盖。

第 11 章 尾声

11.1. 收尾工作

很好！现在新的 LFS 系统已经安装好了！我们祝愿您的全新的，自定义的 Linux 系统能够成功启动！

创建一个 /etc/lfs-release 文件似乎是一个好主意。通过使用它，您 (或者我们，如果您向我们寻求帮助的话) 能够容易地找出当前安装的 LFS 系统版本。运行以下命令创建该文件：

```
echo 20200622-systemd > /etc/lfs-release
```

后续安装在系统上的软件包可能需要两个描述当前安装的系统的文件，这些软件包可能是二进制包，也可能是需要构建的源代码包。

另外，最好创建一个文件，根据 Linux Standards Base (LSB) 的规则显示系统状态。运行命令创建该文件：

```
cat > /etc/lsb-release << "EOF"
DISTRIB_ID="Linux From Scratch"
DISTRIB_RELEASE="20200622-systemd"
DISTRIB_CODENAME="<your name here>"
DISTRIB_DESCRIPTION="Linux From Scratch"
EOF
```

第二个文件基本上包含相同的信息，systemd 和一些图形桌面环境会使用它。运行命令创建该文件：

```
cat > /etc/os-release << "EOF"
NAME="Linux From Scratch"
VERSION="20200622-systemd"
ID=lfs
PRETTY_NAME="Linux From Scratch 20200622-systemd"
VERSION_CODENAME="<your name here>"
EOF
```

您可以修改 'DISTRIB_CODENAME' 域，体现您的系统的独特性。

11.2. 增加 LFS 用户计数

现在您已经完成了本书中的构建过程，那么问题来了，您希望被计为一名 LFS 用户吗？前往 http://www.linuxfromscratch.org/cgi-bin/lfscounter.php 并输入您的姓名和第一次使用的 LFS 版本，即可注册成为 LFS 用户。

下面重启计算机进入 LFS。

11.3. 重启系统

现在已经安装好了本书中的所有软件，可以重新启动进入 LFS 了。然而，您应该注意一些可能出现的问题。您根据本书构建的系统是很小的，可能缺失一些功能，导致您无法继续使用。您可以在当前的 chroot 环境中安装一些 BLFS 手册提供的额外软件包，以便在重启进入新的 LFS 系统后更容易工作。下面是一些建议您考虑的软件包：

- 字符模式浏览器，例如 Lynx，这样您可以在一个虚拟终端中阅读 BLFS 手册，同时在另一个虚拟终端构建软件包。
- GPM 软件包允许您在 虚拟终端中进行复制粘贴操作。
- 如果静态 IP 配置不符合您的网络环境要求，安装 dhcpcd 或 dhcp 的客户端部分等在这种环境下有用的软件包。
- 可以安装 sudo，这样就能使用非 root 用户构建软件包，再很容易地切换到 root 身份进行安装。
- 如果您想从具有舒适的 GUI 环境的远程计算机访问新系统，安装 openssh。
- 为了更方便地从网络下载文件，安装 wget。
- 如果您有 GUID 分区表 (GPT) 磁盘，可能需要安装 gptfdisk 或 parted。
- 最后，可以再次检查下列配置文件。
  - /etc/bashrc
  - /etc/dircolors
  - /etc/fstab
  - /etc/hosts
  - /etc/inputrc
  - /etc/profile
  - /etc/resolv.conf
  - /etc/vimrc
  - /root/.bash_profile
  - /root/.bashrc

现在，正如我们之前保证的，您可以引导全新的 LFS 系统了！首先退出 chroot 环境：

```
logout
```

解除虚拟文件系统的挂载：

```
umount -v $LFS/dev/pts
umount -v $LFS/dev
umount -v $LFS/run
umount -v $LFS/proc
umount -v $LFS/sys
```

如果为 LFS 创建了多个分区，在解除 LFS 文件系统的挂载前，先解除其他挂载点：

```
umount -v $LFS/usr
umount -v $LFS/home
umount -v $LFS
```

解除 LFS 文件系统本身的挂载：

```
umount -v $LFS
```

现在重新启动系统：

```
shutdown -r now
```

如果 GRUB 引导加载器如同本书前文所述安装，并配置正确，GRUB 目录应该已经配置为自动引导 LFS SVN-20200622 启动项。

重启完成后，LFS 系统就可以使用了，您可以安装更多软件包以满足自己的需求。

11.4. 下面该做什么?

感谢您阅读本书。我们希望您觉得本书对您有用，而且您能够从构建系统的过程中学到一些知识。

现在 LFS 系统已经安装好了，您可能想问“然后呢？”为了解答这个问题，我们为您汇集了一份有用资源的列表。

- 维护

  所有软件都会定期发布 Bug 报告和安全注意事项。由于 LFS 系统是从源代码构建的，您必须自己留意它们。有一些跟踪它们的在线网站，下面列出一些:

  - CERT (计算机应急响应小组)

    CERT 有一个邮件列表，发布许多操作系统和应用程序的安全警告。访问 http://www.us-cert.gov/cas/signup.html 阅读邮件列表订阅信息。

  - Bugtraq

    Bugtraq 是一个计算机安全公示邮件列表，它发布新发现的安全问题，偶尔还会提供可能的修复方式。访问 http://www.securityfocus.com/archive 阅读订阅信息。

- Beyond Linux From Scratch

  Beyond Linux From Scratch 手册涵盖了许多不属于 LFS 范畴的软件的安装过程，项目主页是 http://www.linuxfromscratch.org/blfs/。

- LFS Hints

  LFS Hints 是一组由 LFS 社区志愿者提交的帮助文档，它位于 http://www.linuxfromscratch.org/hints/downloads/files/。

- 邮件列表

  如果您需要帮助，希望跟踪 LFS 开发进度，或者希望参与该项目，访问第 1 章 - 邮件列表了解一下 LFS 邮件列表。

- Linux 文档计划

  Linux 文档计划 (The Linux Documentation Project, TLDP) 的目标是通过协作解决 Linux 文档的所有问题，它包含大量 HOWTO 文档、指南和 man 页面。它的网址是 http://www.tldp.org/。

第 V 部分 附录

附录 A. 缩写和术语

ABI 应用程序二进制接口（Application Binary Interface）
ALFS Automated Linux From Scratch
API 应用程序编程接口（Application Programming Interface）
ASCII 美国标准信息交换代码（American Standard Code for Information Interchange）
BIOS 基本输入输出系统（Basic Input/Output System）
BLFS Beyond Linux From Scratch
BSD Berkeley 软件发行版（Berkeley Software Distribution）
chroot 切换根目录（change root）
CMOS 互补金属氧化物半导体（Complementary Metal Oxide Semiconductor）
COS 服务类型（Class Of Service）
CPU 中央处理器（Central Processing Unit）
CRC 循环冗余检查（Cyclic Redundancy Check）
CVS 并行版本系统（Concurrent Versions System）
DHCP 动态主机配置协议（Dynamic Host Configuration Protocol）
DNS 域名服务（Domain Name Service）
EGA 增强图形适配器（Enhanced Graphics Adapter）
ELF 可执行与可链接格式（Executable and Linkable Format）
EOF 文件结束（End of File）
EQN 公式（equation）
ext2 第二代增强文件系统（second extended file system）
ext3 第三代增强文件系统（third extended file system）
ext4 第四代增强文件系统（fourth extended file system）
FAQ 常见问题（Frequently Asked Questions）
FHS 文件系统目录结构标准（Filesystem Hierarchy Standard）
FIFO 先进先出（First-In, First Out）
FQDN 全限定域名（Fully Qualified Domain Name）
FTP 文件传输协议（File Transfer Protocol）
GB 吉字节（Gigabytes）
GCC GNU 编译器集合（GNU Compiler Collection）
GID 组标识符（Group Identifier）
GMT 格林尼治标准时间（Greenwich Mean Time）
HTML 超文本标记语言（Hypertext Markup Language）
IDE 集成驱动电子设备（Integrated Drive Electronics）
IEEE 电子电气工程师学会（Institute of Electrical and Electronic Engineers）

IO 输入/输出（Input/Output）
IP 因特网协议（Internet Protocol）
IPC 进程间通信（Inter-Process Communication）
IRC 互联网中继聊天（Internet Relay Chat）
ISO 国际标准化组织（International Organization for Standardization）
ISP 互联网服务提供商（Internet Service Provider）
KB 千字节（Kilobytes）
LED 发光二极管（Light Emitting Diode）
LFS Linux From Scratch
LSB Linux 标准规范（Linux Standard Base）
MB 兆字节（Megabytes）
MBR 主引导记录（Master Boot Record）
MD5 消息摘要算法第五版（Message Digest 5）
NIC 网络接口卡（Network Interface Card）
NLS 本地语言支持（Native Language Support）
NNTP 网络新闻传输协议（Network News Transport Protocol）
NPTL 原生 POSIX 线程库（Native POSIX Threading Library）
OSS 开放音频系统（Open Sound System）
PCH 预编译头文件（Pre-Compiled Headers）
PCRE Perl 兼容的正则表达式（Perl Compatible Regular Expression）
PID 进程标识符（Process Identifier）
PTY 伪终端（pseudo terminal）
QOS 服务质量（Quality of Service）
RAM 随机访问存储器（Random Access Memory）
RPC 远程过程调用（Remote Procedure Call）
RTC 实时时钟（Real Time Clock）
SBU 标准构建单位（Standard Build Unit）
SCO Santa Cruz 作业公司（The Santa Cruz Operation）
SHA1 安全散列算法第一版（Secure-Hash Algorithm 1）
TLDP Linux 文档计划（The Linux Documentation Project）
TFTP 简单文件传输协议（Trivial File Transfer Protocol）
TLS 线程本地存储（Thread-Local Storage）
UID 用户标识符（User Identifier）
umask 用户文件创建掩码（user file-creation mask）
USB 通用串行总线（Universal Serial Bus）
UTC 协调世界时（Coordinated Universal Time）

UUID 通用唯一识别码（Universally Unique Identifier）
VC 虚拟控制台（Virtual Console）
VGA 视频图形阵列（Video Graphics Array）
VT 虚拟终端（Virtual Terminal）

附录 B. 致谢

我们希望感谢以下人员和组织对 Linux From Scratch 项目作出的贡献:

· Gerard Beekmans <gerard@linuxfromscratch.org> – LFS 创始人

· Bruce Dubbs <bdubbs@linuxfromscratch.org> – LFS 执行编辑

· Jim Gifford <jim@linuxfromscratch.org> – CLFS 共同负责人

· Pierre Labastie <pierre@linuxfromscratch.org> – BLFS 编辑及 ALFS 负责人

· DJ Lucas <dj@linuxfromscratch.org> – LFS 和 BLFS 编辑

· Ken Moffat <ken@linuxfromscratch.org> – BLFS 编辑

· 在 LFS 和 BLFS 的相关邮件列表中还有无数朋友， 他们为本书进行提供了宝贵的建议，对本书中的安装说明进行了测试， 提供了问题报告和安装说明， 还分享了在安装各种软件包时获得的宝贵经验。 他们的工作使得本书得以发布。

翻译人员

· Manuel Canales Esparcia <macana@macana-es.com> – 西班牙语 LFS 翻译项目

· Johan Lenglet <johan@linuxfromscratch.org> – 2008 年以前的 LFS 法语翻译项目

· Jean-Philippe Mengual <jmengual@linuxfromscratch.org> – 2008-2016 年的 LFS 法语翻译项目

· Julien Lepiller <jlepiller@linuxfromscratch.org> – 2017 年后的 LFS 法语翻译项目

· Anderson Lizardo <lizardo@linuxfromscratch.org> – 葡萄牙语 LFS 翻译项目

· Thomas Reitelbach <tr@erdfunkstelle.de> – 德语 LFS 翻译项目

· Anton Maisak <info@linuxfromscratch.org.ru> – 俄语 LFS 翻译项目

· Elena Shevcova <helen@linuxfromscratch.org.ru> – 俄语 LFS 翻译项目

镜像站维护者

北美镜像站

· Scott Kveton <scott@osuosl.org> – lfs.oregonstate.edu 镜像站

· William Astle <lost@l-w.net> – ca.linuxfromscratch.org 镜像站

· Eujon Sellers <jpolen@rackspace.com> – lfs.introspeed.com 镜像站

· Justin Knierim <tim@idge.net> – lfs-matrix.net 镜像站

南美镜像站

· Manuel Canales Esparcia <manuel@linuxfromscratch.org> – lfsmirror.lfs-es.info 镜像站

· Luis Falcon <Luis Falcon> – torredehanoi.org 镜像站

欧洲镜像站

· Guido Passet <guido@primerelay.net> – nl.linuxfromscratch.org 镜像站

· Bastiaan Jacques <baafie@planet.nl> – lfs.pagefault.net 镜像站

- Sven Cranshoff <sven.cranshoff@lineo.be> – lfs.lineo.be 镜像站
- Scarlet Belgium – lfs.scarlet.be 镜像站
- Sebastian Faulborn <info@aliensoft.org> – lfs.aliensoft.org 镜像站
- Stuart Fox <stuart@dontuse.ms> – lfs.dontuse.ms 镜像站
- Ralf Uhlemann <admin@realhost.de> – lfs.oss-mirror.org 镜像站
- Antonin Sprinzl <Antonin.Sprinzl@tuwien.ac.at> – at.linuxfromscratch.org 镜像站
- Fredrik Danerklint <fredan-lfs@fredan.org> – se.linuxfromscratch.org 镜像站
- Franck <franck@linuxpourtous.com> – lfs.linuxpourtous.com 镜像站
- Philippe Baque <baque@cict.fr> – lfs.cict.fr 镜像站
- Vitaly Chekasin <gyouja@pilgrims.ru> – lfs.pilgrims.ru 镜像站
- Benjamin Heil <kontakt@wankoo.org> – lfs.wankoo.org 镜像站
- Anton Maisak <info@linuxfromscratch.org.ru> – linuxfromscratch.org.ru 镜像站

亚洲镜像站

- Satit Phermsawang <satit@wbac.ac.th> – lfs.phayoune.org 镜像站
- Shizunet Co.,Ltd. <info@shizu-net.jp> – lfs.mirror.shizu-net.jp 镜像站
- Init World <http://www.initworld.com/> – lfs.initworld.com 镜像站

澳大利亚镜像站

- Jason Andrade <jason@dstc.edu.au> – au.linuxfromscratch.org 镜像站

曾经的项目组成员

- Christine Barczak <theladyskye@linuxfromscratch.org> – LFS 手册编辑
- Archaic <archaic@linuxfromscratch.org> – LFS 技术作家/编辑， HLFS 项目领导者，BLFS 编辑，Hints 和补丁项目维护者
- Matthew Burgess <matthew@linuxfromscratch.org> – LFS 项目领导者，LFS 技术作家/编辑
- Nathan Coulson <nathan@linuxfromscratch.org> – LFS-Bootscripts 维护者
- Timothy Bauscher
- Robert Briggs
- Ian Chilton
- Jeroen Coumans <jeroen@linuxfromscratch.org> – 网站开发者，FAQ 维护者
- Manuel Canales Esparcia <manuel@linuxfromscratch.org> – LFS/BLFS/HLFS XML 和 XSL 维护者
- Alex Groenewoud – LFS 技术作家
- Marc Heerdink
- Jeremy Huntwork <jhuntwork@linuxfromscratch.org> – LFS 技术作家，LFS LiveCD 维护者
- Bryan Kadzban <bryan@linuxfromscratch.org> – LFS 技术作家
- Mark Hymers

· Seth W. Klein – FAQ 维护者
· Nicholas Leippe <nicholas@linuxfromscratch.org> – Wiki 维护者
· Anderson Lizardo <lizardo@linuxfromscratch.org> – 网站后台脚本维护者
· Randy McMurchy <randy@linuxfromscratch.org> – BLFS 项目领导者，LFS 编辑
· Dan Nicholson <dnicholson@linuxfromscratch.org> – LFS 和 BLFS 编辑
· Alexander E. Patrakov <alexander@linuxfromscratch.org> – LFS 技术作家，LFS 国际化编辑，LFS LiveCD 维护者
· Simon Perreault
· Scot Mc Pherson <scot@linuxfromscratch.org> – LFS NNTP 网关维护者
· Douglas R. Reno <renodr@linuxfromscratch.org> – Systemd 编辑
· Ryan Oliver <ryan@linuxfromscratch.org> – CLFS 项目共同负责人
· Greg Schafer <gschafer@zip.com.au> – LFS 技术作家， 新一代启用 64 位构建方法设计者
· Jesse Tie-Ten-Quee – LFS 技术作家
· James Robertson <jwrober@linuxfromscratch.org> – Bugzilla 维护者
· Tushar Teredesai <tushar@linuxfromscratch.org> – BLFS 手册编辑，Hints 和补丁计划领导者
· Jeremy Utley <jeremy@linuxfromscratch.org> – LFS 技术作家，Bugzilla 维护者，LFS-Bootscripts 维护者
· Zack Winkles <zwinkles@gmail.com> – LFS 技术作家

附录 C. 依赖关系

LFS 中构建的每个软件包都依赖于一个或多个其他软件包， 才能正确地构建和安装。某些软件包甚至存在循环依赖， 即第一个软件包依赖于第二个软件包，而第二个软件包反过来又依赖第一个。 由于这些依赖关系的存在，在 LFS 中构建软件包的顺序非常关键。 本页面的目的就是记录 LFS 中每个软件包构建时的依赖关系。

对于我们构建的每个软件包，我们都列出了三种甚至四种依赖关系。 第一种列出了编译和安装该软件包需要的其他软件包。 第二种列出了不属于第一种情况， 但在运行该软件包测试套件时需要的其他软件包。 第三种列出了在构建和安装前， 需要该软件包已经构建并安装到最终位置的其他软件包。 多数情况下，这是因为它们会在脚本中硬编码指向二进制程序的路径。 如果不按照特定顺序构建，则最终的系统中某个脚本可能包含路径 /tools/bin/[二进制程序]，这显然是我们不希望的。

第四种列出的依赖关系是 LFS 中没有提到的可选软件包， 但它们对用户可能很有用。这些软件包本身可能还有必要或可选的依赖关系。 对于这些依赖关系，推荐的方法是在完成 LFS 手册后， 安装可选依赖项，再重新构建相关的 LFS 软件包。 BLFS 提到了几个软件包的重新安装方法。

Acl

安装依赖于: Attr, Bash, Binutils, Coreutils, GCC, Gettext, Grep, M4, Make, Perl, Sed, 以及 Texinfo
测试依赖于: Automake, Diffutils, Findutils, 以及 Libtool
必须在下列软件包之前安装: Coreutils, Sed, Tar, 以及 Vim
可选依赖项: 无

Attr

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Grep, M4, Make, Perl, Sed, 以及 Texinfo
测试依赖于: Automake, Diffutils, Findutils, 以及 Libtool
必须在下列软件包之前安装: Acl 和 Libcap
可选依赖项: 无

Autoconf

安装依赖于: Bash, Coreutils, Grep, M4, Make, Perl, Sed, 以及 Texinfo
测试依赖于: Automake, Diffutils, Findutils, GCC, 以及 Libtool
必须在下列软件包之前安装: Automake
可选依赖项: Emacs

Automake

安装依赖于: Autoconf, Bash, Coreutils, Gettext, Grep, M4, Make, Perl, Sed, 以及 Texinfo
测试依赖于: Binutils, Bison, Bzip2, DejaGNU, Diffutils, Expect, Findutils, Flex, GCC, Gettext, Gzip, Libtool, 以及 Tar
必须在下列软件包之前安装: 无
可选依赖项: 无

Bash

安装依赖于: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Patch, Readline, Sed, 以及 Texinfo
测试依赖于: Shadow
必须在下列软件包之前安装: 无
可选依赖项: Xorg

Bc

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Grep, 以及 Make
测试依赖于: Gawk
必须在下列软件包之前安装: Linux Kernel
可选依赖项: 无

Binutils

安装依赖于: Bash, Binutils, Coreutils, Diffutils, File, Flex, Gawk, GCC, Glibc, Grep, Make, Perl, Sed, Texinfo, 以及 Zlib
测试依赖于: DejaGNU 和 Expect
必须在下列软件包之前安装: 无
可选依赖项: Debuginfod

Bison

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Perl, 以及 Sed
测试依赖于: Diffutils, Findutils, 以及 Flex
必须在下列软件包之前安装: Kbd 和 Tar
可选依赖项: Doxygen (测试套件)

Bzip2

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make, 以及 Patch
测试依赖于: 无
必须在下列软件包之前安装: File
可选依赖项: 无

Check

安装依赖于: GCC, Grep, Make, Sed, 以及 Texinfo
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 无

Coreutils

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, GMP, Grep, Libcap, Make, Patch, Perl, Sed, 以及 Texinfo
测试依赖于: Diffutils, E2fsprogs, Findutils, Shadow, 以及 Util-linux
必须在下列软件包之前安装: Bash, Diffutils, Findutils, 以及 Man-DB
可选依赖项: Perl 模块 Expect 和 IO:Tty (测试套件)

DejaGNU

安装依赖于: Bash, Coreutils, Diffutils, GCC, Grep, Make, 以及 Sed
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 无

Diffutils

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed, 以及 Texinfo
测试依赖于: Perl
必须在下列软件包之前安装: 无
可选依赖项: 无

E2fsprogs

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Gzip, Make, Sed, Texinfo, 以及 Util-linux
测试依赖于: Procps-ng 和 Psmisc
必须在下列软件包之前安装: 无
可选依赖项: 无

Eudev

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Gperf Make, Sed, 以及 Util-linux
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 无

Expat

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, 以及 Sed
测试依赖于: 无
必须在下列软件包之前安装: Python 和 XML::Parser
可选依赖项: 无

Expect

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Patch, Sed, 以及 Tcl

测试依赖于: 无

必须在下列软件包之前安装: 无

可选依赖项: 无

File

安装依赖于: Bash, Binutils, Bzip2, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, Xz, 以及 Zlib

测试依赖于: 无

必须在下列软件包之前安装: 无

可选依赖项: 无

Findutils

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, 以及 Texinfo

测试依赖于: DejaGNU, Diffutils, 以及 Expect

必须在下列软件包之前安装: 无

可选依赖项: 无

Flex

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Patch, Sed, 以及 Texinfo

测试依赖于: Bison 和 Gawk

必须在下列软件包之前安装: Binutils, IPRoute2, Kbd, Kmod, 以及 Man-DB

可选依赖项: 无

Gawk

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, GMP, Grep, Make, MPFR, Patch, Readline, Sed, 以及 Texinfo

测试依赖于: Diffutils

必须在下列软件包之前安装: 无

可选依赖项: libsigsegv

GCC

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, GMP, Grep, M4, Make, MPC, MPFR, Patch, Perl, Sed, Tar, Texinfo, 以及 Zstd
测试依赖于: DejaGNU, Expect, 以及 Shadow
必须在下列软件包之前安装: 无
可选依赖项: GNAT 和 ISL

GDBM

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Grep, Make, 以及 Sed
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 无

Gettext

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Sed, 以及 Texinfo
测试依赖于: Diffutils, Perl, 以及 Tcl
必须在下列软件包之前安装: Automake 和 Bison
可选依赖项: 无

Glibc

安装依赖于: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Gzip, Linux API Headers, Make, Perl, Python, Sed, 以及 Texinfo
测试依赖于: File
必须在下列软件包之前安装: 无
可选依赖项: 无

GMP

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, M4, Make, Sed, 以及 Texinfo
测试依赖于: 无
必须在下列软件包之前安装: MPFR 和 GCC
可选依赖项: 无

Gperf

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, 以及 Make
测试依赖于: Diffutils 和 Expect
必须在下列软件包之前安装: 无
可选依赖项: 无

Grep

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Patch, Sed, 以及 Texinfo
测试依赖于: Gawk
必须在下列软件包之前安装: Man-DB
可选依赖项: Pcre 和 libsigsegv

Groff

安装依赖于: Bash, Binutils, Bison, Coreutils, Gawk, GCC, Glibc, Grep, Make, Patch, Sed, 以及 Texinfo
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Man-DB 和 Perl
可选依赖项: Ghostscript

GRUB

安装依赖于: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Texinfo, 以及 Xz
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 无

Gzip

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, 以及 Texinfo
测试依赖于: Diffutils 和 Less
必须在下列软件包之前安装: Man-DB
可选依赖项: 无

Iana-Etc

安装依赖于: Coreutils, Gawk, 以及 Make
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Perl
可选依赖项: 无

Inetutils

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed, Texinfo, 以及 Zlib
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Tar
可选依赖项: 无

Intltool

安装依赖于: Bash, Gawk, Glibc, Make, Perl, Sed, 以及 XML::Parser
测试依赖于: Perl
必须在下列软件包之前安装: 无
可选依赖项: 无

IProute2

安装依赖于: Bash, Bison, Coreutils, Flex, GCC, Glibc, Make, Libcap, Libelf, 以及 Linux API 头文件
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: 无
可选依赖项: Berkeley DB 和 Iptables

Kbd

安装依赖于: Bash, Binutils, Bison, Check, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, Patch, 以及 Sed
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: 无
可选依赖项: 无

Kmod

安装依赖于: Bash, Binutils, Bison, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, Pkg-config, Sed, Xz-Utils, 以及 Zlib
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Eudev
可选依赖项: 无

Less

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, 以及 Sed
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Gzip
可选依赖项: Pcre

Libcap

安装依赖于: Attr, Bash, Binutils, Coreutils, GCC, Glibc, Perl, Make, 以及 Sed
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: IProute2 和 Shadow
可选依赖项: Linux-PAM

Libelf

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, 以及 Make
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: IProute2 和 Linux Kernel
可选依赖项: 无

Libffi

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Make, 以及 Sed
测试依赖于: DejaGnu
必须在下列软件包之前安装: Python
可选依赖项: 无

Libpipeline

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, 以及 Texinfo
测试依赖于: Check
必须在下列软件包之前安装: Man-DB
可选依赖项: 无

Libtool

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, 以及 Texinfo
测试依赖于: Autoconf, Automake, 以及 Findutils
必须在下列软件包之前安装: 无
可选依赖项: 无

Linux Kernel

安装依赖于: Bash, Bc, Binutils, Coreutils, Diffutils, Findutils, GCC, Glibc, Grep, Gzip, Kmod, Libelf, Make, Ncurses, OpenSSL, Perl, 以及 Sed
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: 无
可选依赖项: 无

M4

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, 以及 Texinfo
测试依赖于: Diffutils
必须在下列软件包之前安装: Autoconf 和 Bison
可选依赖项: libsigsegv

Make

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, 以及 Texinfo
测试依赖于: Perl 和 Procps-ng
必须在下列软件包之前安装: 无
可选依赖项: 无

Man-DB

安装依赖于: Bash, Binutils, Bzip2, Coreutils, Flex, GCC, GDBM, Gettext, Glibc, Grep, Groff, Gzip, Less, Libpipeline, Make, Sed, 以及 Xz
测试依赖于: Util-linux
必须在下列软件包之前安装: 无
可选依赖项: 无

Man-Pages

安装依赖于: Bash, Coreutils, 以及 Make
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: 无
可选依赖项: 无

Meson

安装依赖于: Ninja 和 Python
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Systemd
可选依赖项: 无

MPC

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, GMP, Make, MPFR, Sed, 以及 Texinfo
测试依赖于: 无
必须在下列软件包之前安装: GCC
可选依赖项: 无

MPFR

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, GMP, Make, Sed, 以及 Texinfo
测试依赖于: 无
必须在下列软件包之前安装: Gawk 和 GCC
可选依赖项: 无

Ncurses

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Patch, 以及 Sed
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Bash, GRUB, Inetutils, Less, Procps-ng, Psmisc, Readline, Texinfo, Util-linux, 以及 Vim
可选依赖项: 无

Ninja

安装依赖于: Binutils, Coreutils, GCC, 以及 Python
测试依赖于: 无
必须在下列软件包之前安装: Meson
可选依赖项: Asciidoc, Doxygen, Emacs, 以及 re2c

Openssl

安装依赖于: Binutils, Coreutils, Gcc, Make, 以及 Perl
测试依赖于: 无
必须在下列软件包之前安装: Linux
可选依赖项: 无

Patch

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, 以及 Sed
测试依赖于: Diffutils
必须在下列软件包之前安装: 无
可选依赖项: Ed

Perl

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, GDBM, Glibc, Grep, Groff, Make, Sed, 以及 Zlib
测试依赖于: Iana-Etc 和 Procps-ng
必须在下列软件包之前安装: Autoconf
可选依赖项: 无

Pkg-config

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Popt, 以及 Sed
测试依赖于: 无
必须在下列软件包之前安装: Kmod
可选依赖项: 无

Popt

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, 以及 Make
测试依赖于: Diffutils 和 Sed
必须在下列软件包之前安装: Pkg-config
可选依赖项: 无

Procps-ng

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Make, 以及 Ncurses
测试依赖于: DejaGNU
必须在下列软件包之前安装: 无
可选依赖项: 无

Psmisc

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, 以及 Sed
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: 无
可选依赖项: 无

Python

安装依赖于: Bash, Binutils, Coreutils, Expat, GCC, Gdbm, Gettext, Glibc, Grep, Libffi, Make, Ncurses, Sed, 以及 Util-linux
测试依赖于: GDB 和 Valgrind
必须在下列软件包之前安装: Ninja
可选依赖项: Berkeley DB, OpenSSL, SQLite, 以及 Tk

Readline

安装依赖于: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed, 以及 Texinfo
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Bash 和 Gawk
可选依赖项: 无

Sed

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, 以及 Texinfo
测试依赖于: Diffutils 和 Gawk
必须在下列软件包之前安装: E2fsprogs, File, Libtool, 以及 Shadow
可选依赖项: 无

Shadow

安装依赖于: Acl, Attr, Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Libcap, Make, 以及 Sed
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: Coreutils
可选依赖项: Cracklib 和 PAM

Sysklogd

安装依赖于: Binutils, Coreutils, GCC, Glibc, Make, 以及 Patch
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: 无
可选依赖项: 无

Systemd

安装依赖于: Acl, Attr, Bash, Binutils, Coreutils, Diffutils, Expat, Gawk, GCC, Glibc, Gperf, Grep, Intltool, Libcap, Meson, Sed, 以及 Util-linux
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 很多，参见 BLFS systemd 页面

Sysvinit

安装依赖于: Binutils, Coreutils, GCC, Glibc, Make, 以及 Sed
测试依赖于: 没有可用的测试套件
必须在下列软件包之前安装: 无
可选依赖项: 无

Tar

安装依赖于: Acl, Attr, Bash, Binutils, Bison, Coreutils, GCC, Gettext, Glibc, Grep, Inetutils, Make, Sed, 以及 Texinfo
测试依赖于: Autoconf, Diffutils, Findutils, Gawk, 以及 Gzip
必须在下列软件包之前安装: 无
可选依赖项: 无

Tcl

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, 以及 Sed
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 无

Texinfo

安装依赖于: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Patch, 以及 Sed
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: 无

Util-linux

安装依赖于: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Eudev, 以及 Zlib
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: Libcap-ng

Vim

安装依赖于: Acl, Attr, Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, 以及 Sed
测试依赖于: 无
必须在下列软件包之前安装: 无
可选依赖项: Xorg, GTK+2, LessTif, Python, Tcl, Ruby, 以及 GPM

XML::Parser

安装依赖于: Bash, Binutils, Coreutils, Expat, GCC, Glibc, Make, 以及 Perl
测试依赖于: Perl
必须在下列软件包之前安装: Intltool
可选依赖项: 无

Xz

安装依赖于: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, 以及 Make.
测试依赖于: 无
必须在下列软件包之前安装: Eudev, File, GRUB, Kmod, 以及 Man-DB
可选依赖项: 无

Zlib

安装依赖于: Bash, Binutils, Coreutils, GCC, Glibc, Make, 以及 Sed
测试依赖于: 无
必须在下列软件包之前安装: File, Kmod, Perl, 以及 Util-linux
可选依赖项: 无

Zstd

安装依赖于: Binutils, Coreutils, GCC, Glibc, Gzip, Make, 以及 Sed
测试依赖于: 无
必须在下列软件包之前安装: GCC
可选依赖项: 无

附录 D. LFS 授权许可

D.1. Creative Commons License

Creative Commons Legal Code

Attribution-NonCommercial-ShareAlike 2.0

重要

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions
   a. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.
   b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.
   c. "Licensor" means the individual or entity that offers the Work under the terms of this License.
   d. "Original Author" means the individual or entity who created the Work.
   e. "Work" means the copyrightable work of authorship offered under the terms of this License.

f. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

g. "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, Noncommercial, ShareAlike.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

b. to create and reproduce Derivative Works;

c. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

d. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works;

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(e) and 4(f).

4. Restrictions.The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any reference to such Licensor or the Original Author, as requested.

b. You may distribute, publicly display, publicly perform, or publicly digitally perform a Derivative Work only under the terms of this License, a later version of this License with the same License Elements as this License, or a Creative Commons iCommons license that contains the same License Elements as this License (e.g. Attribution-NonCommercial-ShareAlike 2.0 Japan). You must include a copy of, or the Uniform Resource Identifier for, this License or other license specified in the previous sentence with every copy or phonorecord of each Derivative Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Derivative Works that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder, and You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Derivative Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Derivative Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Derivative Work itself to be made subject to the terms of this License.

c. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

d. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

e. For the avoidance of doubt, where the Work is a musical composition:

   i. Performance Royalties Under Blanket Licenses. Licensor reserves the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work if that performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

   ii. Mechanical Rights and Statutory Royalties. Licensor reserves the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions), if Your distribution of such cover

version is primarily intended for or directed toward commercial advantage or private monetary compensation. 6. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

f. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

5. Representations, Warranties and Disclaimer

   UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

重要

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, neither party will use the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time.

Creative Commons may be contacted at http://creativecommons.org/.

D.2. The MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

索引

软件包

程序

库

脚本

其他