Linux From Scratch（简体中文版）

Version 8.4-systemd

项目创建者：Gerard Beekmans
总编：Bruce Dubbs
编辑：DJ Lucas
翻译：Linux 中国 - LCTT - LFS 翻译小组

Linux From Scratch（简体中文版）: Version 8.4-systemd

by 项目创建者：Gerard Beekmans, 总编：Bruce Dubbs, 编辑：DJ Lucas, and 翻译：Linux 中国 - LCTT - LFS 翻译小组

Table of Contents

序章

前言

自 1998 始，我逐渐从学习 Linux 到对其深入了解。仅仅是在我安装完我的第一个 Linux 发行版后，我便迅速地为 Linux 所蕴含的整套理念及其背后的哲学思想所着迷。

总是会有很多的方法来完成一个任务。这对 Linux 的发行版而言也是这样。这些年来，这么多的发行版层出不迭，有的还在，有的已经今非昔比，有的现在也只能在记忆中感怀。它们的行事风格大相径庭，以满足不同受众的需求。也正是因为如此之多不同的发行版想要到达的是同一个目标，我开始意识到自己可以不必在受限于别人做好的发行版。在开始探索 Linux 之前，我们只能忍受各种操作系统的问题，因为我们别无选择。无论你喜欢还是不喜欢，它就在那里。但 Linux 让你拥有了选择的可能。如果你有什么不喜欢的，你可以自由地，甚至倍受鼓舞地去改变它。

我尝试过很多发行版，但还是无法决定该用哪个。这些系统，从它们自身的角度来看，它们都是很棒的系统。这里也并不是在探讨对错问题，这只不过是个人习惯的问题。在各种可选项中，显然没有哪个单一的系统对我而言是完美的。于是我开始着手创建自己的 Linux 系统，以完全满足自己的喜好。

为了使系统真正地属于我，我毅然决定开始从源代码编译所有东西，而不是使用预编译好的二进制包。这个「完美」的 Linux 系统能够拥有不同系统的优点，而不会有它们与之俱来的不足。起初，这种想法让人想都不敢想，但我一直保持着可以构建出这样一个系统的信念。

在梳理了诸如循环依赖和编译错误之类的问题后，我最终构建出了一个订制的 Linux 系统。它运转自如，用起来与当时的那些 Linux 系统一样完美，但它是由我亲手打造的。能组装出这样一个系统让我很满意，而与之相比更棒的，恐怕唯有在软件的各个方面都亲历亲为了，不过这也可算是第二棒的事了。

当我将自己的目标和经验与 Linux 社区的其它成员分享时，引发了大家对这些想法的不断关注。很快人们就清楚地意识到，这样一个定制的 Linux 系统不仅仅可以满足用户的特定需求，还可为程序员和系统管理员提供一个理想的学习机会，以增强他们（原有）的 Linux 技能。出于这个被放大了的兴趣，Linux From Scratch 项目诞生了。

这本 Linux From Scratch 手册是该项目的重中之重。它为你提供了设计和构建自己的系统所需的背景知识和指令。虽然本手册给出了一个模板，一个最后能正常工作的系统，但你可以自由地改变指令以适应自身的需求，从某种程度上说，这是也本项目的一个重要部分。你仍然可以控制一切，我们只是在你起航之前助你一臂之力。

我衷心地希望你能享受构建属于自己的 Linux From Scratch 系统的过程，并对坐拥一个真正属于自己的系统所带来诸多好处能做到甘之如饴。

--
Gerard Beekmans
gerard@linuxfromscratch.org

致读者

说到为什么要读这本书，我想，原因一定有很多。有些人可能会提出这样的疑问：「都已经有现成的 Linux 系统可以下载和安装了，你为什么还要多此一举的从从头构建一个呢？」

此项目存在的一个重要原因是帮助你了解 Linux 的内部是如何工作的。通过构建 LFS 系统，你可以更好的理解 Linux 的工作原理，以及程序是如何协同工作和相互依赖的。最棒的是，这个学习经历能给你提供自定义 Linux 系统以满足你自己独特需求的能力。

另一个重要的原因是，你对系统本身有更多的控制权，而无需去关心别人的 Linux 是如何实现的。在 LFS 下，你就是主宰，系统的各个方面都需要你亲力亲为。

LFS 可以让你创建极其精简的 Linux 系统。在安装那些常规的 Linux 系统时，你往往会迫不得已而安装那些你用不到的（甚至你都不知道它们是干什么用的）程序。这些程序也许会浪费你的硬件资源。或许你可能会说，现在计算机的资源那么丰富，稍微浪费一些又有什么关系呢。但是你依旧要考虑到可引导 CD、U 盘或者是一些嵌入式的环境，它们对资源高度敏感，这也恰恰是 LFS 所擅长的地方。

另一个优势便是，自定义的 Linux 系统有着更高的安全性。通过从源码构建一个完整的系统，你有权审核所有的代码以及所打入的安全补丁。也就没必要去在等别人编译修复该安全漏洞的二进制程序了。而且，除非你亲自检查了补丁文件并且做了完整的测试，否则你又怎么能确信，新的二进制程序确实是正确的编译且解决了问题呢？

从头构建一个基本可用的 Linux 系统是本书的目标，你即使不打算这么做，也还是能从这本书的内容中有所收获。

支撑你从头构建属于自己的 LFS 系统的原因实在是太多了。但说到最后，其中最重要的一个原因还是教学。随着你的 LFS 经验愈趋熟稔，这些信息和知识真正给你带来的力量，你也便了然于胸了。

LFS 的目标架构

LFS 当前主要支持 AMD/Intel 的 x86（32 位）和 x86_64（64 位）构架的 CPU。另外，本文档也涉及一些更改以允许 LFS 顺利地在 Power PC 和 ARM CPU 上运行。为了顺利构建 LFS，除了后面几页的内容外，你主要需要一个可以在当前 CPU 上正常运行的 Linux 系统，例如：早先版本的 LFS、Ubuntu、Red Hat/Fedora、SuSE 或者是在你的架构上可以运行的其它发行版。另外注意 32 位的发行版是可以在 64 位的 AMD/Intel 处理器上作为宿主机正常安装和运作的。

这里也说明一下关于 64 位系统的一些情况。与 32 位相比，大体上程序运行的速度虽稍微地快了那么一点点，但体积也稍微的大了那么一点点。以在 Core 2 Duo 处理器上运行的 LFS-6.5 系统为例，以下便是实测数据：

```
Architecture    Build Time      Build Size
32-bit          198.5 minutes   648 MB
64-bit          190.6 minutes   709 MB
```

正如你看到的，64 位程序仅仅比 32 位程序快了 4%，而体积大了 9%。由此可见，单纯的追逐 64 位系统其实并没有太大的必要。但是，假如你的电脑的内存超过了 4 GB 又或者说需要操作大于 4 GB 的数据，64 位系统的优势就比较明显了。

> **Note**
>
> 上述讨论只适用于对比相同硬件下的构建过程。现代的 64 位系统会比老的 64 位系统更快。LFS 作者建议，如果有机会，尽量在 64 位系统上进行构建。

假如按照本文的默认方式构建，那么你将得到一个「纯」64 位系统——这意味着你仅能够执行 64 位的程序。构建「multi-lib」并不是不可以，但是这意味着很多的程序都需要编译两次：一次为 32 位程序编译，一次为 64 位程序编译。不过，本文档并不涉及这部份的内容，因为这些内容会干扰用户学习如何构建一份最基本的 Linux 系统。你可以通过阅读 Cross Linux From Scratch 的相关内容获得有关该话题的更多帮助。

LFS 和标准

LFS 的结构尽可能的遵循 Linux 的标准。主要的标准有：

- POSIX.1-2008.
- 文件系统层次结构标准（FHS）3.0 版本
- Linux 标准规范（LSB） 5.0 （2015）版本

  LSB 有四个独立的标准：核心（Core）、桌面（Desktop）、运行时语言（Runtime Languages）和成像（Imaging）。除了通用要求，还有架构特定要求。此外还有两个领域在试行：分别是 Gtk3 和图形（Graphics）。LFS 试图遵从前一节中所讨论的架构要求。

> **Note**
>
> 很多人不认同 LSB 的要求。定义它的主要目的是确保私有软件能够在兼容的系统上安装并正常运行。由于 LFS 是基于源代码的，用户对于需要什么软件包有完全的控制权，有很多人就选择不安装 LSB 规范要求的软件包。

完全可以创建一个能够通过 LSB 认证测试的完整 LFS 系统，但这需要很多 LFS 范围之外的额外软件包。在 BLFS 中有这些额外软件包的安装说明。

由 LFS 提供，用于满足 LSB 要求的软件包

LSB 核心：	Bash, Bc, Binutils, Coreutils, Diffutils, File, Findutils, Gawk, Grep, Gzip, M4, Man-DB, Ncurses, Procps, Psmisc, Sed, Shadow, Tar, Util-linux, Zlib
LSB 桌面：	无
LSB 运行时语言：	Perl

LSB 成像：	无
LSB Gtk3 和 LSB 图形（试用）：	无

由 BLFS 提供，用于满足 LSB 要求的软件包

LSB 核心：	At, Batch (At 的一部分), Cpio, Ed, Fcrontab, Initd-tools, Lsb_release, NSPR, NSS, PAM, Pax, Sendmail (或 Postfix 或 Exim), time
LSB 桌面：	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 运行时语言：	Python, Libxml2, Libxslt
LSB 成像：	CUPS, Cups-filters, Ghostscript, SANE
LSB Gtk3 和 LSB 图形（试用）：	GTK+3

LFS 和 BLFS 没有提供，用于满足 LSB 要求的软件包

LSB 核心：	无
LSB 桌面：	Qt4 （以及 Qt5 除外）
LSB 运行时语言：	无
LSB 成像：	无
LSB Gtk3 和 LSB 图形（试行）：	无

本书中的软件包逻辑

正如前文所述，LFS 的目标是构建一个完整可用的基础级系统。这其中便包含了，所有地软件包需要复制自身，直至为用户提供出一个相对最小的基础，然后根据用户的选择，在这个基础上定制出一个更完善的系统。这便意味着 LFS 并不是最小可用系统的代名词。其中包含的几个重要软件包，也并非严格需求的。下面的列表介绍了本书中选择每个软件包的理由。

- Acl

  这个软件包中，包含管理访问控制列表（ACL）的工具，用于定义文件和目录更细粒度的自主访问权。

- Attr

  这个软件包中包含的程序，用于管理文件系统对象的扩展属性。

- Autoconf

  这个软件包中包含的程序，能根据开发者的模板自动生成配置源代码的 shell 脚本。通常需要在更新构建过程之后重新构建一个软件包。

- Automake

  这个软件包中，包含了通过模板生成 Make 文件的程序。通常需要在更新构建过程之后重新构建一个软件包。

- Bash

  这个软件包用于能满足 LSB 核心的需求，为系统提供 Bourne Shell 接口。选择它而非其它 shell 软件包的理由，是因为它的通用性以及在基本 shell 功能上的扩展能力。

- Bc

  这个软件包提供了一种任意精度的数值处理语言。在构建 Linux 内核时需要它。

- Binutils

  这个软件包中，包含了一个链接器、一个汇编器和其它处理对象文件的工具。编译 LFS 以及 BLFS 系统中的大部分软件包，包需要该软件包中的程序。

- Bison

  这个软件包中有 GNU 版的 yacc（Yet Another Compiler Compiler），用于构建一些其它的 LFS 程序。

- Bzip2

  这个软件包中，包含用来压缩和解压缩文件的程序。在解压缩很多 LFS 软件包的时候需要它。

- Check

  这个软件包中，包含一个用于测试其它程序的工具。仅安装在临时工具链中。

- Coreutils

  这个软件包，包含了一些查看和管理文件及目录的重要程序。在命令行里管理文件时和每个 LFS 软件包的安装过程中需要它。

- D-Bus

  这个软件包中，包含了一个用于提供消息总线的程序，消息总线是一种应用程序间通信的简单方式。

- DejaGNU

  这个软件包中，包含一个测试其它程序的框架。仅安装在临时工具链中。

- Diffutils

  这个软件包中，包含了一些显示文件和目录差异的程序。这些程序可以用来创建补丁，也用于很多软件包的构建过程。

- E2fsprogs

  这个软件包中，包含了一些处理 ext2、ext3 和 ext4 文件系统的工具。这些是 Linux 上支持的最常用而且完全经过考验的文件系统。

- Expat

  这个软件包中，包含了一个相对小的 XML 解析库。Perl 模块 XML::Parser 需求该软件包。

- Expect

  这个软件包中，包含的了一个生成与其它程序交互的脚本对话框的程序。通常用来测试其它软件包。仅安装在临时工具链中。

- File

  这个软件包中，包含一个能判断给定文件的类型的工具。一些软件包需要用它来构建。

- Findutils

  这个软件包中，包含了一些在文件系统中查找文件的程序。在很多软件包构建脚本中会用到。

- Flex

  这个软件包中，包含了一个能生成识别文本模式程序的工具。是 lex（lexical analyzer）程序的 GNU 版本。构建很多 LFS 软件包需要用到。

- Gawk

  这个软件包中，包含了一些操作文本文件的程序。是 awk（Aho-Weinberg-Kernighan）的 GNU 版本。在很多软件包的构建脚本中会用到。

- Gcc

  这个软件包是 GNU 编译器工具集。它包括 C 和 C++ 的编译器以及其它构建 LFS 时用不到的软件包。

- GDBM

  这个软件包中，包含 GNU 数据库管理库。LFS 的另一个软件包 Man-DB 会用到。

- Gettext

  这个软件包中，包含了很多软件包国际化和本地化需要用到的工具和库。

- Glibc

  这个软件包中，包含了主要的 C 语言库。缺少它 Linux 的程序便无法运行了。

- GMP

  这个软件包中，包含了能提供任意精度数值运算的数学库。编译 Gcc 时会用到。

- Gperf

  这个软件包中，包含了一个能从一个键集生成完美哈希函数的程序。Eudev 会用到。

- Grep

这个软件包中，包含了一些在文件中搜索的程序。大部分软件包的构建脚本会用到。

- Groff

  这个软件包中，包含了处理和格式化文本的程序。其中一个重要的功能是格式化 man 页面。

- GRUB

  这个软件包是 Grand Unified Boot Loader。是诸多可用的引导加载器中的一个，但是最灵活。

- Gzip

  这个软件包中，包含了一些压缩和解压缩文件的程序。解压很多 LFS 以及 BLFS 的软件包时需要用到。

- Iana-etc

  这个软件包中，包含提供了网络服务和协议的数据。启用合适的网络功能时会用到。

- Inetutils

  这个软件包中，包含基本网络管理的程序。

- Intltool

  这个软件包中，包含能从源文件中抽取可翻译字符串的工具。

- IProute2

  这个软件包中，包含了一些基本和高级的 IPv4 和 IPv6 网络的程序。由于它的 IPv6 功能，所以选择它来取代其它的常见网络工具包（net-tools）。

- Kbd

  这个软件包中，包含一些键盘映射文件，用于非 US 键盘的键盘工具以及一些控制台字体。

- Kmod

  这个软件包中，包含一些用于管理 Linux 内核模块的程序。

- Less

  这个软件包中，包含了一个很好的文本文件查看器，允许查看文件的时候向上或向下滚动。Man-DB 用它来查看 man 页面。

- Libcap

  这个软件包中，包含实现了可以用于 Linux 内核的，从用户空间到 POSIX 1003.1e 的接口。

- Libelf

  elfutils 项目为 ELF 文件和 DWARF 数据提供了库和工具。该软件包中的大部分实用程序都可以在别的软件包中使用，但是该库需求使用默认（且最有效的）配置构建 Linux 内核。

- Libffi

  这个软件包实现了一个可移植的、高级编程接口，以适应各种调用惯例。在编译时，一些程序可能还不知道给函数传递的是什么参数。例如，一个解释器可能会在运行时才被告知函数调用中所传递的参数的个数和类型。Libffi 可以应用于这些程序，充当从解释程序到被编译代码的桥梁。

- Libpipeline

  这个软件包中，包含一个以灵活和便捷的方式操作子进程流水线的库。Man-DB 软件包会用到。

- Libtool

  这个软件包中，包含了一些 GNU 通用库支持脚本。它降低了在一致、可移植的接口上使用共享库的复杂度。在其他 LFS 软件包的测试套件里需要它。

- Linux 内核

  这个软件包就是操作系统。即我们常说的「GNU/Linux」中的「Linux」。

- M4

  这个软件包中，包含一个普通的文本宏处理器，作为其它程序的构建工具使用。

- Make

  这个软件包中，包含了一个指导软件包构建的程序。LFS 中的几乎每个包都需要它。

- Man-DB

这个软件包中，包含了一些查找和查看 man 页面的程序。由于其更好的国际化功能，用来代替 man 软件包。它提供了 man 程序。

- Man-pages

  这个软件包中，包含基本的 Linux man 页面的真正内容。

- Meson

  这个软件包提供了自动化构建软件的工具。Meson 的主要目标是最小化软件开发者，在其构建的系统上，花费的配置时间。

- MPC

  这个软件包中，包含复数运算的函数。Gcc 需要它。

- MPFR

  这个软件包中，包含多精度运算的函数。Gcc 需要它。

- Ninja

  这个软件包中，包含了一个专注于速度的小型构建系统。其输入文件由更高层次的构建系统生成，并且尽可能快速地运行构建过程。

- Ncurses

  这个软件包中，包含了一些处理字符界面的不依赖特定终端的库。通常用来为菜单系统提供光标控制。一些 LFS 的软件包会用到。

- Openssl

  这个软件包中，提供了与加密相关管理工具和库。提供的这些加密功能对于他软件包（包括 Linux 内核）而言非常有用。

- Patch

  这个软件包中，包含了一个通过 patch 文件来修改或新建文件的程序，patch 文件通常是由 diff 程序创建的。一些 LFS 软件包的构建过程会需要它。

- Perl

  这个软件包中，包含了一个运行时语言 PERL 的解析器。一些 LFS 软件包的安装和测试套件会需要它。

- Pkg-config

  这个软件包提供了一个返回已安装库或软件包的元数据的程序。

- Procps-NG

  这个软件包中，包含了一些监视进程的程序。这些程序对系统管理非常有用，也被用于 LFS 的启动脚本。

- Psmisc

  这个软件包中，包含了一些显示运行中进程信息的程序。这些程序对系统管理非常有用。

- Python 3

  这个软件包提供了一种解释型语言，该语言的设计理念强调代码的可读性。

- Readline

  这个软件包提供了一些命令行编辑和历史功能的库。Bash 会使用它。

- Sed

  这个软件包提供了不通过文本编辑器而直接编辑文本的功能。大部分 LFS 软件包的配置脚本需要它。

- Shadow

  这个软件包中，包含了一些以安全方式处理密码的程序。

- systemd

  作为 Sysvinit 的替代品，这个包提供了 init 程序，以及一些其它的启动和系统控制的功能。很多 Linux 商业发行版都用它。

- Tar

  这个软件包提供了归档和提取 LFS 中的几乎所有软件包的能力。

- Tcl

这个软件包中，包含了在很多 LFS 软件包测试套件中使用的工具命令语言。仅安装在临时工具链中。

- Texinfo

  这个软件包中，包含了一些读、写以及转换信息页面的程序。在很多 LFS 软件包的安装过程中会使用它。

- Util-linux

  该软件包中，包含了许多工具。其中有处理文件系统、控制台、分区和消息的工具。

- Vim

  这个软件包中，包含一个编辑器。由于 vi 编辑器的经典以及大量的强大功能而选择它。对很多用户来说一个编辑器是一个非常个人的选择，如果需要的话也可以选择其它编辑器。

- XML::Parser

  这个软件包是和 Expat 交互的 Perl 模块。

- XZ Utils

  这个软件包中，包含了一些压缩和解压缩文件的程序。通常它的压缩率最高，在解压 XZ 或者 LZMA 格式的软件包时非常有用。

- Zlib

  这个软件包中，包含了一些程序所使用的压缩和解压缩功能。

前提条件

构建一个 LFS 系统并不是一个简单的任务。它要求对 Unix 系统管理有一定水平的了解，以便可以解决问题并正确地执行所列出的命令。特别是，至少你应该拥有使用命令行（shell），复制或移动文件和目录、列出目录和文件内容、切换当前目录的能力。同时也希望你拥有使用和安装 Linux 软件的基本知识。

因为 LFS 书中假设你至少有这些基本的技能，所以众多的 LFS 提供的论坛看起来在这方面也给你提供不了太多的帮助。你会发现对于这些基本知识的问题并不会得到解答，或者会简单的建议你去看一下 LFS 主要的提前阅读列表。

构建 LFS 系统之前，我们建议阅读以下内容：

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

  这是一个在 Linux 上编译和安装「通用」Unix 软件包的综合指南。尽管成文已有一段时间，但其对于编译和安装软件中所需的基本技能，还算一个不错的总结。

- 初学者源码安装指南 http://moi.vonos.net/linux/beginners-installing-from-source/

  该指南很好的总结了从源码构建软件时所需要的基础技能和技巧。

排版约定

为了能更更容易理解，这里有一些全书使用的排版约定。本段，会包含一些来自 Linux From Scratch 中排版格式的示例。

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

这种形式的文本，设计出来就是为了让你照着输入的，除非周围文本中另有说明。部分的解释段落和会用它，以指出引用的那条命令。（译注：由于文字排版原因，其中的内容即便是注释，也不会翻译，下同。）

某些情况下，会在一行的行末添加反斜，将一个逻辑行的内容分写成两个或更多的物理行。

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

注意，反斜杠的后面必须紧跟一个回车符。其它的空白字符，例如空格和 Tab 将导致错误的结果。

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

这种形式的文本（等宽文本）用于显示屏幕输出，通常是所运行命令的输出结果。这种形式也用于文件名的显示，例如： `/etc/ld.so.conf`。

强调

这种形式的文本在本书中有多重目的。主要目的是强调重要的内容或项目。

http://www.linuxfromscratch.org/

这种格式用来表示 LFS 社区内部及外部网页的超链接。包括 HOWTO，下载地址和网站等。

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

这种格式在创建配置文件中会使用。第一个命令告诉系统新建一个文件 $LFS/etc/group，然后后面的行中不论在输入了什么都会输入到文件中，直至遇到文件终结符（EOF）。因此，这整个部分会如看到的那样输入。

<#######>

这种格式用来封装需要替换为合适内容的文本，或者复制粘贴操作。

[#####]

这种格式用来封装可选项文本。

`passwd(5)`

这种格式用来表示特定的手册 （man） 页面。括号内的数字表示该手册的特定部分。例如 passwd 有两个手册页面。在 LFS 安装说明中，这两个手册页面会表示为 `/usr/share/man/man1/passwd.1` 以及 `/usr/share/man/man5/passwd.5`。当书中使用 `passwd(5)` 时它特指 `/usr/share/man/man5/passwd.5`。man passwd 会输出它找到匹配「passwd」的第一个手册页面，也就是 `/usr/share/man/man1/passwd.1`。在这个例子中，你需要执行 man 5 passwd 才能阅读指定的手册页面。应该注意的是，大部分的手册页面在不同部分不会有重复的页面名字。因此，man ***<program name>*** 通常就足够了。

本书结构

本书分为以下几个部分。

第一部分 介绍

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

第二部分 构建的准备工作

第二部分描述了构建的一些准备工作，包括分区，下载软件包，编译一些临时工具。

第三部分 构建 LFS 系统

第三部分引导用户开始 LFS 系统的构建——逐一的编译和安装所有的软件包，设置启动脚本，安装内核。生成的 Linux 系统是继续编译其它一系列软件的基础，通过那些软件来扩展系统，系统才能更好地满足我们的需求。在本书的最后，我们还给出了一个便于使用的引用列表，包括安装好的程序、库和一些重要文件。

勘误表

构建 LFS 系统的软件是在不断的更新和改进的，也许在这本 LFS 书发布以后就会出现一些安全警告或者是修复了一些 bug。请在着手构建之前访问 http://www.linuxfromscratch.org/lfs/errata/systemd/，检查此版本 LFS 软件包的版本或说明，是否需要修改以解决安全漏洞或其它 bug。你应该关注本书的一切变化，且将其应用到 LFS 系统的构建中去。

如果 LFS 的中文手册翻译的有问题，或是你对 LFS 中文手册有自己的建议和意见。可以通过中文手册的 Github 库 LFS-BOOK https://github.com/LCTT/LFS-BOOK 找到并告诉我们。

Part I. 介绍

Chapter 1. 介绍

1.1. 如何构建 LFS 系统

LFS 系统需要在一个已经安装好的 Linux 发行版（比如 Debian、OpenMandriva、Fedora 或 OpenSUSE 等）中构建。这个已有的 Linux 系统（即宿主）作为构建新系统的起始点,提供了必要的程序，包括编译器、链接器和 shell。请在安装发行版的过程中选择「Development（开发）」选项以便使用这些开发工具。

你可以选择在电脑上安装一个独立的发行版，亦可以临时使用商业发行版的 LiveCD。

本手册的 第 2 章 描述了如何创建一个新的 Linux 本地分区和文件系统。这就是编译和安装全新的 LFS 系统的地方。第 3 章 介绍了构建 LFS 系统所需要下载的软件包和补丁，以及如何将它们保存到新的文件系统中。第 4 章 讨论了如何设置恰当的工作环境。还请务必仔细阅读 第 4 章，它会告诉您在继续 第 5 章 以及后面的工作之前，尤其需要注意的几件事请。

第 5 章 解释了为构建编译套件（工具链）需要安装的多个软件包，在 Chapter#6中将会使用这套工具构建正真的系统。其中有一些包需要解决循环依赖——比如你需要一个编译器来编译出一个编译器。

第 5 章 还展示了如何构建第一阶段的工具链，包括 Binutils 和 GCC（第一阶段的主要意义就是重新安装这两个核心软件包）。下一步将来构建 Glibc，即 C 语言库。Glibc 将使用第一阶段构建的工具链编译。然后，第二阶段的工具链编译就完成了。这次，工具链将会被动态链接到新建立的 Glibc。第 5 章 中剩余的其他包都将使用第二阶段的工具链编译。当这些过程完成之后，除了正在运行的内核外，LFS 的安装过程就不再已赖于宿主发行版了。

把新系统从宿主发行版中分离出来的工作看起来可能有点多余。Section#5.2, “工具链技术说明”给出了为什么要这么做的完整技术说明。

在 Chapter#6中，将构建完整的 LFS 系统。我们将会使用 chroot（切换成 root 用户的）命令来进入一个虚拟环境并启动一个新的 shell，且整个虚拟环境就将作为 LFS 分区的根目录。这个过程，和重起电脑并指示内核将 LFS 分区挂载为根分区十分相像。只是系统并没有真正重启，而是通过 chroot 实现的，而这样做的原因是想要实现一个正真可以启动的系统还有很多工作没有完成。使用「chrooting」最大的好处在于，构建 LFS 的同时可以继续使用宿主系统。在等待包编译的过程中，你可以继续正常的使用电脑。

在完成基本的安装过程之后，您还需要阅读 第 7 章 对系统做简单的配置，然后阅读 第 8 章 配置内核和引导器（bootloader）。第 9 章 包含了一些 LFS 后续使用的建议。完成本手册的所有步骤之后，就可以重启电脑，进入新的 LFS 系统了。

大致流程就是如此。每一步的细节都会在后续的章节中逐一阐明。在你开始 LFS 的旅途之后，混沌的过程便会渐渐明晰。

1.2. 自上一版手册发布后的变更

下面列举了上一版手册发布后的一些软件包更新。

更新：

-
- Bash 5.0
- Binutils-2.32
- Bison-3.3.2
- DejaGNU-1.6.2
- D-Bus-1.12.12
- Diffutils-3.7
- E2fsprogs-1.44.5
- File-5.36
- GDBM-1.18.1
- Glibc-2.29
- Grep-3.3
- Groff-1.22.4
- Gzip-1.10

- IPRoute2-4.20.0
- Kmod-26
- Libcap-2.26
- Libelf-0.176 (from elfutils)
- Libpipeline-1.5.1
- Linux-4.20.12
- Man-DB-2.8.5
- Meson-0.49.2
- MPFR-4.0.2
- Ninja-1.9.0
- Openssl-1.1.1a
- Perl-5.28.1
- Psmisc-23.2
- Python-3.7.2
- Readline-8.0
- Sed-4.7
- systemd-240
- Tar-1.31
- Tcl-8.6.9
- Tzdata-2018i
- Util-Linux-2.33.1

新增：

-

移除：

-

1.3. 更新日志

这是 Linux From Scratch 手册的 8.4-systemd 版本，发布于 2019 年 3 月 1 日。如果距离这个时间已超过 6 个月，那么应该已经有更新和更好的版本了。你可以访问 http://www.linuxfromscratch.org/mirrors.html 的任意镜像站点来获取他们。

下面是本书上一次发布之后的更新列表。

更新日志条目：

- 2019-03-01
  - [bdubbs] - LFS-8.4 发布。
- 2019-02-25
  - [bdubbs] - 更新至 linux-4.20.12。修复 #4425。
  - [bdubbs] - 更新至 elfutils-0.176。修复 #4426。
  - [bdubbs] - 更新至 file-5.36。修复 #4429。
  - [renodr] - 为 systemd-240 添加安全补丁，以修复 D-Bus 上 PID1 崩溃的问题。修复 #4428。
- 2019-02-19
  - [bdubbs] - 添加一可选修改至 ninja 的构建过程，以允许使用环境变量 NINJAJOBS。
- 2019-02-14
  - [bdubbs] - 更新至 linux-4.20.8。修复 #4423。
  - [bdubbs] - 修复第 5 章中构建 Python 时，某些宿主机可能使用其相关 header 导致的错误。
- 2019-02-11
  - [bdubbs] - 更新至 linux-4.20.7。修复 #4421。

- [bdubbs] - 更新至 kmod-26。修复 #4422。
- 2019-02-08
  - [renodr] - 更新宿主机系统要求。
- 2019-02-06
  - [bdubbs] - 简化第 5 章中 glibc 的说明。感谢 Romain Geissler 的报告。
- 2019-02-05
  - [bdubbs] - 更新至 bison-3.3.2。修复 #4419。
  - [bdubbs] - 更新至 meson-0.49.2。修复 #4420。
- 2019-02-02
  - [bdubbs] - 修复 psmisc URL。修复 #4418。
  - [bdubbs] - 更新至 binutils-2.32。修复 #4417。
- 2019-02-01
  - [bdubbs] - 更新至 bison-3.3.1。修复 #4412。
  - [bdubbs] - 更新至 glibc-2.29。修复 #4415。
  - [bdubbs] - 更新至 libpipeline-1.5.1。修复 #4413。
  - [bdubbs] - 更新至 linux-4.20.6。修复 #4409。
  - [bdubbs] - 更新至 meson-0.49.1。修复 #4410。
  - [bdubbs] - 更新至 mpfr-4.0.2。修复 #4416。
  - [bdubbs] - 更新至 ninja-1.9.0。修复 #4414。
- 2019-01-27
  - [pierre] - 通过添加在构建说明中一条 sed 命令，来修复 tar-1.31 的一个 bug 的说明。同时移除有关测试失败的陈旧注释。
- 2019-01-20
  - [renodr] - 使用 Docbook XSL 非命名空间版本的样式重新生成 systemd man 手册的 tarball。
- 2019-01-11
  - [renodr] - 为 systemd-240 添加安全补丁。该补丁用于修正 CVE-2018-16865 和 CVE-2018-16864（日志中的内存损坏导致堆栈溢出/任意代码可执行） 。尽可能快的应用该补丁。修复 #4408。
- 2019-01-10
  - [bdubbs] - 更新至 linux-4.20.1。修复 #4398。
  - [bdubbs] - 更新至 diffutils-3.7。修复 #4401。
  - [bdubbs] - 更新至 tar-1.31。修复 #4402。
  - [bdubbs] - 更新至 man-db-2.8.5。修复 #4403。
  - [bdubbs] - 更新至 bash-5.0。修复 #4404。
  - [bdubbs] - 更新至 readline-8.0。修复 #4405。
  - [bdubbs] - 更新至 iproute2-4.20.0。修复 #4406。
  - [bdubbs] - 更新至 util-linux-2.33.1。修复 #4407。
- 2019-01-01
  - [bdubbs] - 更新至 gzip-1.10。修复 #4400。
  - [bdubbs] - 更新至 tzdata-2018i。修复 #4399。
- 2018-12-27
  - [renodr] - 更新至 linux-4.19.12。修复 #4389。
  - [renodr] - 更新至 e2fsprogs-1.44.5。修复 #4390。
  - [renodr] - 更新至 bison-3.2.4。修复 #4391。
  - [renodr] - 更新至 sed-4.7。修复 #4392。
  - [renodr] - 更新至 grep-3.3。修复 #4393。
  - [renodr] - 更新至 systemd-240。包含一个针对 systemd-tmpfiles 的关键修复（权限提升）。修复 #4394。

- [renodr] - 更新至 Python-3.7.2。修复 #4395。
- [renodr] - 更新至 groff-1.22.4。修复 #4396。

- 2018-12-12
  - [renodr] - 添加一则注意提示 libffi 编译时将根据 CPU 优化。与 GMP 类似，如果将安装被移动到另一个系统，则会导致非法操作错误。
- 2018-12-11
  - [bdubbs] - 更新至 dbus-1.12.12。修复 #4385。
  - [bdubbs] - 更新至 linux-4.19.8。修复 #4387。
  - [bdubbs] - 更新至 meson-0.49.0。修复 #4388。
- 2018-12-01
  - [bdubbs] - 移动 /etc/bash_completions.d/grub 至更好的位置。修复 #4385。
  - [bdubbs] - 更新至 dejagnu-1.6.2。修复 #4382。
  - [bdubbs] - 更新至 linux-4.19.6。修复 #4383。
  - [bdubbs] - 更新至 perl-5.28.1。修复 #4384。
- 2018-11-25
  - [bdubbs] - 更新至 bison-3.2.2。修复 #4380。
- 2018-11-24
  - [dj] - 更新至 linux-4.19.4。修复 #4381.
  - [dj] - 更新至 systemd-239-6b4878d.
  - [dj] - 添加「wheel」用户组至 systemd 用户组。
  - [dj] - 添加 touch 至被移动的 coreutils 程序列表，并阐述移动的必要性是为了满足 FHS 合规性。
- 2018-11-21
  - [renodr] - 添加「wheel」用户组以满足 systemd 的需求。修复 #4376。
  - [renodr] - 添加一条 sed 命令以修复 autoconf 测试套件的 bug。修复 #4372。
  - [renodr] - 更新至 tcl-8.6.9。安全更新。修复 #4375。
  - [renodr] - 更新至 openssl-1.1.1a。这是一个安全更新。修复 #4379。
  - [renodr] - 更新至 systemd-239-25d1ba1。将修复三个 systemd 的安全问题。修复 #4377。
  - [renodr] - 更新至 linux-4.19.3。修复 #4373。
  - [renodr] - 更新至 elfutils-0.175。修复 #4374。
- 2018-11-19
  - [bdubbs] - 更新至 libcap-2.26。修复 #4378。
- 2018-11-09
  - [bdubbs] - 更新至 meson-0.48.2。修复 #4371。
  - [bdubbs] - 更新至 bison-3.2.1。修复 #4370。
- 2018-11-06
  - [bdubbs] - 更新至 bison-3.2。修复 #4367。
  - [bdubbs] - 更新至 linux-4.19.1。修复 #4368。
  - [bdubbs] - 更新至 tzdata-2018g。修复 #4366。
  - [bdubbs] - 更新至 util-linux-v2.33。修复 #4353。
- 2018-10-29
  - [dj] - 更新至 gdbm-1.18.1。修复 #4364。
  - [dj] - 更新至 Python-3.7.1。修复 #4361。
- 2018-10-27
  - [dj] - 更新至 iproute2-4.19.0。修复 #4363。
  - [dj] - 更新至 file-5.35。修复 #4359。
  - [dj] - 更新至 tzdata-2018f。修复 #4358。

- [dj] - 更新至 meson-0.48.1。修复 #4357。
- [dj] - 更新至 linux-4.19。修复 #4356。

- 2018-10-10
  - [dj] - 移除 systemd 说明中指向 /toold/lib64 的错误链接。修复 #4355。
  - [dj] - 添加 systemd-239-meson-0.48.0_fixes-1.patch 以解决 meson 的构建问题。
  - [dj] - 更新至 meson-0.48.0。修复 #4351。
  - [dj] - 更新至 linux-4.18.12。修复 #4352。
- 2018-09-30
  - [dj] - 恢复第 5 章 Util-Linux 的构建，以避免对 systemd 的相互依赖。
  - [dj] - 移动 Util-Linux 和 E2fsprogs 的安装至 Procps 之后，以满足 systemd 版 LFS 手册的构建顺序。这将不会影响到 SysV 版的手册。
- 2018-09-20
  - [bdubbs] - 清理不需要的符号链接。重排软件包安装顺序，尽可能晚的安装第 6 章中版本特定的软件包。现在就无需在第 5 章构建 util-linux 了，将其移除。修复 #4345 和 #4349。
  - [bdubbs] - 更新至 elfutils-0.174 (libelf)。修复 #4348。
  - [bdubbs] - 更新至 psmisc-23.2。修复 #4347。
  - [bdubbs] - 更新至 openssl-1.1.1。修复 #4346。
  - [bdubbs] - 更新至 linux-4.18.9。修复 #4344。
- 2018-09-02
  - [bdubbs] - 更新至 bison-3.1。修复 #4342。
  - [bdubbs] - 更新至 meson-0.47.2。修复 #4341。
  - [bdubbs] - 更新至 gdbm-1.18。修复 #4340。
  - [bdubbs] - 更新至 e2fsprogs-1.44.4。修复 #4338。
- 2018-09-01
  - [bdubbs] - LFS-8.3 发布。

1.4. 资源

1.4.1. FAQ

如果在构建 LFS 系统的过程中遇到任何的错误，或者是有任何的疑问，亦或者认为本手册存在任何的拼写错误，请先阅读常见问题列表（FAQ）：http://www.linuxfromscratch.org/faq/。

1.4.2. 邮件列表

`linuxfromscratch.org` 的服务器上部署了一些和 LFS 项目开发工作相关的邮件列表。其中包括主要开发列表和支持列表，以及一些其他相关话题的讨论列表。如果 FAQ 不能解决您的问题，那么您可以在此搜索邮件列表：http://www.linuxfromscratch.org/search.html。

关于这些列表的订阅、归档、和其他的信息，都可以访问 http://www.linuxfromscratch.org/mail.html 获取。

1.4.3. IRC 频道

有些 LFS 的社区成员也会在 IRC 上为别人提供帮助。在使用这种方法之前，请保证问题在 LFS 的 FAQ 或者是邮件列表中没有提及。您可以通过 `irc.freenode.net` 查找 #LFS-support 加入讨论。

1.4.4. 镜像站点

LFS 项目在世界范围内有许多镜像站点,方便大家访问我们的网站以及下载所需文件。请访问 LFS 站点 http://www.linuxfromscratch.org/mirrors.html 查看最新的镜像站点列表。

1.4.5. 联系信息

请直接通过某个 LFS 邮件列表（上文已列出）提出问题和评论。

1.5. 帮助

如果在使用本书的过程中有疑问或碰到问题，可以先去看下 FAQ 页面http://www.linuxfromscratch.org/faq/#generalfaq。那里已经解决了很多经常遇到的问题。如果你的问题在那里找不到答案，可以先尝试找出问题的原因。下面页面里的提示可以提供一些帮你定位问题的帮助：http://www.linuxfromscratch.org/hints/downloads/files/errors.txt。

如果在 FAQ 里找不到你遇到的问题，还可以在这个邮件列表里搜索一下：http://www.linuxfromscratch.org/search.html。

我们还有一个很棒的 LFS 社区，大家都很乐意通过邮件列表和 IRC 提供协助（参看本书的 Section#1.4, “资源”）。不过，我们每天收到的支持问题中有很多其实可以通过查看 FAQ 和搜索邮件列表轻松解决。所以为了让我们能最大可能地提供更好的协助，希望你碰到问题能自己先研究一下。这样可以让我们有精力去关注更罕见问题。如果你自己搜索不到解决方式，请在你的帮助请求里收集所有相关信息（下面提到的）。

1.5.1. 需要提供的信息

除了对遇到的问题的简短描述外，帮助请求里还需要包含的以下的关键信息：

- 所用手册的版本（当前版本为 8.4-systemd）
- 用来构建 LFS 的宿主机器的 Linux 发行版以及版本
- 本手册 宿主系统需求 中脚本的所有输出信息
- 出现问题的软件包或本手册的章节
- 精确的错误信息或表现形式
- 注明你是否已经脱离了本书的内容

> **Note**
>
> 脱离本手册的内容并不意味着我们就一定不会帮你。毕竟，LFS 还是属于个人爱好。坦率地告知别人对已验证的流程作出的任何改动，有助于我们评估和找到你问题的可能原因。

1.5.2. 配置脚本问题

如果在运行 configure 脚本时遇到问题，查看一下 `config.log` 文件。这个文件中可能会包含 configure 脚本运行时没有输出到屏幕上的错误信息。想寻求帮助的话请包含相关行。

1.5.3. 编译问题

屏幕上的显示输出和各个文件的内容都有助于定位编译发生问题的原因。configure 和 make 命令的输出信息都有用。然而通常并不需要在寻求帮助时将这些信息一股脑的贴出来，但是一定要包含足够的相关信息。下面的例子就是在执行 make 命令出错后应该贴出来的输出信息：

```
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
```

在此例子中，很多人可能仅仅就贴出最后的一部分：

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

这样的信息并不能用于诊断问题，实际上它仅仅是告知别人出了问题，却没有指出到底是哪里出现问题。这也就是为什么要提供更加完整的信息。比如在上面的例子中，就告诉别人所执行的命令及相应的错误信息。

这个链接 http://catb.org/~esr/faqs/smart-questions.html 中的文章告知人们如何在互联网上寻求帮助，可以去看一下，且遵循其中的提问原则进行提问，可以提高获得帮助的概率（亦让别人对你伸出援手更加方便）。

Part II. 准备构建

Chapter 2. 准备宿主系统

2.1. 简介

本章将检查宿主机上用于构建 LFS 的工具，没有的话，安装它们。然后，会准备用于安装 LFS 的分区。包括建立分区、为分区设置文件系统，挂载分区。

2.2. 宿主系统要求

你的主机系统应具有以下指定最低版本的软件。 对于大多数现代 Linux 发行版来说，这应该不是问题。 另请注意，不少发行版可能会将软件的头文件单独打包，并采用「<package-name>-devel」或「<package-name>-dev」的形式命名。如果你的发行版提供了这样的软件包，请务必安装。

所列软件包的早期版本可能也能用，但尚未经过测试。

- Bash-3.2 （/bin/sh 应该是 bash 的符号链接或硬链接）
- Binutils-2.25 （不建议使用 2.32 以后的版本，尚未经过测试）
- Bison-2.7 （/usr/bin/yacc 应该是指向 bison 或执行 bison 的小型脚本的链接）
- Bzip2-1.0.4
- Coreutils-6.9
- Diffutils-2.8.1
- Findutils-4.2.31
- Gawk-4.0.1 （/usr/bin/awk 应该是 gawk 的链接）
- GCC-5.2 包括 C++ 编译器，g++ （8.2.0 以后的版本不建议使用，尚未经过测试）
- Glibc-2.11 （2.29 以后的版本不建议使用，尚未经过测试）
- Grep-2.5.1a
- Gzip-1.3.12
- Linux Kernel-3.2

  之所以指定使用这一版本的内核，是因为这是第 6 章构建 glibc 的需求，开发人员如是建议。udev 也有这个需求。

  如果宿主机内核早于 3.2，则需使用更新版本的内核将其替换。有两种方法可以解决这个问题。首先，查看你的 Linux 供应商是否提供 3.2 或更高版本内核的软件包。如果提供的话，你便能如愿安装了。如果你的供应商不提供合适的内核（的软件包），或者是你不想直接安装，你也可以自己编译内核。编译内核和配置引导的说明（假设宿主机使用 GRUB）位于 第 8 章。
- 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

> **Important**
>
> 请注意，依据本手册的指导构建 LFS 系统需确保上述提到的符号链接无误。虽然指向其他软件（例如 dash，mawk 等）的符号链接可能也可以，但 LFS 开发团队尚未对其进行测试或支持，并且这样做可能导致对于个别软件包在执行命令或者应用某些补丁的时候出现偏差。

查看宿主机系统是否已具备所有软件包的适当版本和编译程序的能力，请运行以下命令：

```
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 的设计是建立在一个会话中的。这也就是说假定系统在处理的过程中不会关机。但是实际上这并不意味着 LFS 的构建需要一气呵成。但是如果 LFS 恢复时的状态和之前不同，则某些程序必须在此之后重新构建。

2.3.1. 第 1-4 章

当在主机系统各种完成此章节的内容，重启时，需要注意一下的内容：

- 程序在 2.4 节以后都是以 root 用户完成的。所以需要重新将 LFS 的环境变量设置为 ROOT。

2.3.2. 第 5 章

- /mnt/lfs 分区必须挂载。
- 第 5 章涉及的所有指令都应该由 lfs 用户执行。所以在执行第 5 章的任务之前请先执行 su - lfs。
- Section#5.3, “通用编译指南”中提及的程序是十分重要的。如果在安装任何软件包时存在问题，请确保所有之前已经解压的软件包已经删除。重新将它们解压后再完成此章节中的所有命令。

2.3.3. 第 6-8 章

- /mnt/lfs 分区必须挂载。
- 当进入 chroot 环境，LFS 环境变量必须设置为 root 用户。否则不得使用 LFS 变量。
- 必须挂载虚拟文件系统。这可以在进入 chroot 之前或者之后通过改变主机的虚拟终端，以 root 用户执行 Section#6.2.2, “挂载和激活 /dev”和 Section#6.2.3, “挂载虚拟文件系统”中的命令。

2.4. 创建新分区

与绝大多数其它操作系统相同，安装 LFS 通常需要专门的分区。构建 LFS 系统比较推荐的方法是使用可用的空分区，或者如果条件允许，最好是在未分区的空间里新建分区。

最小化的系统需要大约 6 GB 的分区，这足以存储所有的源码包及满足编译的需求。但如果要将 LFS 作为主要的 Linux 系统，可能需要安装其它附加的软件，这将需要额外的空间。考虑到了日后所需的空间，一个 20 GB 的分区是比较合理的。LFS 系统本身并不会占用这么多的空间，但大分区将能提供充裕的临时储存空间，并为完成 LFS 以后添加附加功能留有余地。编译软件包可能需要较大的磁盘空间，但这些空间可以在软件包安装后回收。

由于编译过程中所需的内存（RAM）可能不足，用一个小型的磁盘分区作为 swap 空间是个不错的想法。内核会在此分区中储存较少使用的数据，从而为活动进程提供更多的内存。LFS 系统可以与宿主系统共用 swap 分区，这样就没有必要再新建一个了。

启动磁盘分区程序，通过如 cfdisk 或 fdisk 加上新分区所在的磁盘名——例如 /dev/sda 若是主盘为 IDE 的话。你需要创建一个 Linux 本地分区，并按需创建 swap 分区。如果你还不知道如何使用这些程序，请参考 cfdisk(8) 或 fdisk(8)。

> **Note**
>
> 对于有经验的用户，也可自行定制分区分案。新版 LFS 系统支持软件 RAID 阵列或 LVM 然而，这些方案需要用到 initramfs，这涉及到比较复杂的话题。并不建议首次尝试 LFS 的用户使用这样的分区方法。

请记住新分区的位置（例如，sda5）。本书中将称其为 LFS 分区。还需要记住 swap 分区的位置。这些将会在/etc/fstab 文件中用到。

2.4.1. 其它分区问题

LFS 邮件列表中经常有人问到关于系统分区的建议。这一话题非常主观。大多数发行版默认情况下会使用整个磁盘，仅为交换分区保留一小部分空间，但由于种种原因，这并不适合 LFS。这样做会降低灵活性，使得多个发行版或几个 LFS 版本之间共享数据变得困难，也让备份更耗时，还会导致文件系统结构分配不合理而浪费磁盘空间。

2.4.1.1. 根分区

为 LFS 根分区（不要与 `/root` 目录混淆）分配 10 GB 的空间是比较好的折中方案。这为构建 LFS 和大多数 BLFS 提供了足够的空间，也小到可以轻易创建多个分区用于实验。

2.4.1.2. 交换分区

大多数发行版会自动创建交换分区。一般来说，交换分区的推荐大小为物理内存的两倍左右，然而鲜少需要这样做。若是磁盘空间有限，可以设置为 2 GB的交换分区并查看磁盘的交换量。

发生内存交换其实并不好。通常，你只需要观察磁盘活动以及系统对命令的响应程度就能知道这个系统是否在进行交换。通常在使用非常不合理的命令时才会发生交换，如尝试编辑一个大小为 5 GB 的文件时。如果交换在你的系统上是常态，那最好的办法就是为你的系统添置更多的物理内存。

2.4.1.3. Grub Bios 分区

如果是在 GPT（GUID Partition Table，GUID 分区表）下分 boot 磁盘，那么必然会产生一个约 1 MB 左右的小分区，如果之前没有的话。这个分区可能还没有被格式化，但是它必须存在，GRUB 会在安装引导器的时候用到。如果使用 fdisk，该分区通常会被标记为「BIOS Boot」。而如果使用的是 gdisk，则分区代码应为 EF02 。

> **Note**
>
> Grub Bios 分区必须位于 BIOS 用于引导系统的磁盘驱动器上。却不一定要和 LFS 的根分区位于同一个磁盘驱动器。一个系统里的磁盘可能会用到不同类型的分区表。该磁盘仅与 boot 磁盘的分区表类型有关。

2.4.1.4. 常用分区

在分配磁盘时，有些分区不是必须的，但却值得你考虑。以下列表并不全面，仅供参考。

- /boot – 强烈推荐。此分区用于存储内核和其它启动信息。为了减少大容量磁盘启动时的潜在问题，尽量将该分区设为磁盘驱动器上第一个物理分区。100 MB 的空间就十分充裕了。
- /home – 强烈推荐。home 目录可用于跨发行版或多个 LFS 版本之间共享用户自定义内容。应该将尽量多的磁盘都分配给 home 分区。
- /usr – 独立的 /usr 分区常见于瘦客户端服务器或无盘工作站。LFS 通常不需要。5 GB 大小足以应付大部分安装。
- /opt – 该目录常用于在 BLFS 中，安装多个像 Gnome 或 KDE 这样无需将文件嵌入 /usr 层次结构的大型软件包。如果使用的话，5 到 10 GB 的空间就足够了。
- /tmp – 独立的 /tmp 分区是比较少见的，但这在配置瘦客户端时会有用。如果使用的话，很少超过几 GB。
- /usr/src – 该分区非常用于存储 BLFS 源文件并在构建不同版本的 LFS 中共享。它也可用于构建 BLFS 软件包。30-50 GB 的分区可以提供足够的空间。

任何你需要在启动时自动挂载的单独分区都需要写入到 `/etc/fstab` 文件中。有关如何指定分区的细节将在 Section#8.2, “创建 /etc/fstab 文件”中讨论 。

2.5. 在分区上创建文件系统

现在已经有了一个空的分区，可以创建文件系统了。LFS 可以使用 Linux 内核能识别的任何文件系统，但最常用的类型是 ext3 和 ext4。当然，至于具体采用哪种文件系统，应该取决于所要存储的文件特点和分区的大小。例如：

ext2
: 适用于那些分区容量不是太大，更新也不频繁的情况，例如 /boot 分区。

ext3
: 是 ext2 的改进版本，其支持日志功能，能够帮助系统从非正常关机导致的异常中恢复。是常用作一般需求的文件系统。

ext4
: 是 ext 文件系统的最新版。提供了很多新的特性，包括纳秒级时间戳、创建和使用巨型文件(16TB)、以及速度的提升。

其它文件系统，包括 FAT32、NTFS、ReiserFS、JFS 和 XFS 等都在专门领域有其独到的作用。关于这些文件系统更多的信息可以参考http://en.wikipedia.org/wiki/Comparison_of_file_systems。

LFS 假设要将根分区（/）的文件系统的类型设置为 ext4。运行如下指令，为 LFS 分区创建 ext4 文件系统：

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

请替换 <*xxx*> 为实际的 / 分区设备名。

如果你已经有了现成的 swap 分区,不需要重新格式化。如果是新建的 swap 分区，需要用下面的命令初始化：

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

请替换 <*yyy*> 为实际的 swap 分区设备名。

2.6. 设置 $LFS 变量

在整本书里，会多次用到环境变量 LFS。你应该确保这个变量在整个 LFS 构建过程中总是定义了的。它应该被设置为你将要构建的 LFS 系统的目录名——这里我们使用 /mnt/lfs 作为例子，但是选择哪一个目录是你的自由。如果你把 LFS 构建到一个单独的分区里，这个目录将成为那个分区的挂载点。选择一个本地目录并用以下命令设置该变量：

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

设置这个变量的好处在于有些命令可以直接写成类似 mkdir -v $LFS/tools这样。在处理这条命令时，会自动替换「$LFS」为「/mnt/lfs」（或任何这个变量所指的地方）。

Caution

不论何时，当你离开又重新进入这个工作环境时都不要忘了检查 LFS 是否设置（比如当你使用 su 切换到 root 或是另一个用户时）。使用如下命令检查 LFS 变量是否正确设置：

```
echo $LFS
```

请确保输出的是你构建 LFS 的那个目录的路径，如果你是按照例子设置的，那就是 /mnt/lfs。如果你输出的路径不正确，请使用本页前一部分提供的命令把 $LFS 重新设置到正确的目录。

Note

确保 LFS 变量一直为设置的一个方法是编辑你 home 目录下的 .bash_profile 文件和 /root/.bash_profile，并输入上述 export 命令。另外，在 /etc/passwd 文件中，为所有需要的用户指定 LFS 变量的 shell 需要是 bash，以确保 /root/.bash_profile 文件作为登录过程的一部分，并入其中。

另一个考虑是登入宿主系统的方法。如果是通过图形显示管理器登入，那么就不会像寻常那样在虚拟终端启动时，采用用户的 .bash_profile 文件。在这种情况下，需要将 export 命令添加至 root 用户的 .bashrc 中。另外，有些发行版还有指令，在非交互式 bash 的调用中不让 .bashrc 指令运行。在非交互式使用的测试前请保证 export 命令已经添加。

2.7. 挂载新分区

至此，文件系统已经创建妥当，下一步就是访问这些分区了。为此，需要将这些建立的分区挂载到选定的挂载点。本书假定的挂载点为环境变量 LFS 指向的地址，如前文所述。

创建挂载点并用下面的命令挂载 LFS 文件系统：

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

请用 LFS 分区替代 *<xxx>* 。

如果 LFS 使用了多个分区（如，/ 和 /usr），需要挂载：

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

请替换 *<xxx>* 和 *<yyy>* 为合适的分区名。

请确保在挂载新分区时没有使用过于严格的权限参数（如 nosuid 或 nodev 选项）。运行不带任何参数的 mount 命令来查看在挂载 LFS 分区时设置了什么参数。如果设置了 nosuid 和/或 nodev 参数，就需要重新挂载了。

Warning

上述说明假定你在 LFS 的过程中不会重启你的计算机。如果你关闭了你的系统，你将需要在每次重新开始构建过程时重新挂载 LFS 分区，或者修改你宿主系统的 /etc/fstab 文件，让每次重新启动后都自动挂载它。示例如下：

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

如果你还需使用其它可选分区，那就确保将它们也一并添加。

如果你要使用 swap 分区，请使用 swapon 命令确保它是可用的：

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

请替换 *<zzz>* 为 swap 分区名。

至此你已经建立了自己的工作空间，是时候下载软件包了。

Chapter 3. 软件包和补丁

3.1. 简介

本章列出了一张软件包的清单，你需要下载它们来构建一个基础的 Linux 系统。列出的软件版本号便是该软件经过确认可以正常工作的版本，也是成书时我们使用的版本。我们强烈建议不要使用更新的本版，因为某个版本的编译指令并不一定适用于更新版本。最新的软件包会包含许多问题，需要特别对应。这些对应方法会在本书的开发版本中解决并固定下来。

我们无法保证下载的地址是一直有效的。如果在本书发布后下载地址变了，大部分软件包可以用 Google （http://www.google.com/）解决。如果连搜索也失败了，那不妨试一试 http://www.linuxfromscratch.org/lfs/packages.html#packages 中提到的其他下载地址。

下载好的软件包和补丁需要保存在某处，这个某处最好是一个在整个构建过程中都能便捷地访问到的地方。另外还需要一个工作目录，用于解压并构建源码。`$LFS/sources` 可以同时作为软件包、补丁的存放所和工作目录。通过这个目录，所有需要的元素都将存储在 LFS 分区中，并且在整个构建过程中都可以访问。

在开始下载任务之前，先用 `root` 用户执行下面的命令，创建这个目录：

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

设置目录的写权限和粘滞模式。「粘滞模式」是指，即便多个用户对某个目录有写权限，但仅有文件的所有者，能在粘滞目录中删除该文件。运行以下命令将开启目录的写权限和粘滞模式：

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

一个简单的一口气下载所有软件包和补丁的方法是使用 wget-list 作为 wget 的输入。例如：

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

另外，从 LFS-7.0 开始，多了一个单独的文件 md5sums，可以在正式开始前校验所有的文件是否都正确。将这个文件复制到 `$LFS/sources` 目录中并执行：

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

3.2. 所有软件包

下载或使用别的方式获取如下软件包：

- Acl (2.2.53) - 513 KB:
下载： 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： 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： http://ftp.gnu.org/gnu/autoconf/autoconf-2.69.tar.xz
MD5 校验和： `50f97f4159805e374639a73e2636f22e`
- Automake (1.16.1) - 1,499 KB:
主页： http://www.gnu.org/software/automake/
下载： http://ftp.gnu.org/gnu/automake/automake-1.16.1.tar.xz
MD5 校验和： `53f38e7591fa57c3d2cee682be668e5b`
- Bash (5.0) - 9,898 KB:
主页： http://www.gnu.org/software/bash/
下载： http://ftp.gnu.org/gnu/bash/bash-5.0.tar.gz
MD5 校验和： `2b44b47b905be16f45709648f671820b`
- Bc (1.07.1) - 411 KB:
主页： http://www.gnu.org/software/bc/
下载： http://ftp.gnu.org/gnu/bc/bc-1.07.1.tar.gz
MD5 校验和： `cda93857418655ea43590736fc3ca9fc`

- Binutils (2.32) - 20,288 KB:
主页： http://www.gnu.org/software/binutils/
下载： http://ftp.gnu.org/gnu/binutils/binutils-2.32.tar.xz
MD5 校验和： 0d174cdaf85721c5723bf52355be41e6

- Bison (3.3.2) - 2,060 KB:
主页： http://www.gnu.org/software/bison/
下载： http://ftp.gnu.org/gnu/bison/bison-3.3.2.tar.xz
MD5 校验和： c9b552dee234b2f6b66e56b27e5234c9

- Bzip2 (1.0.6) - 764 KB:
下载： http://anduin.linuxfromscratch.org/LFS/bzip2-1.0.6.tar.gz
MD5 校验和： 00b516f4704d4a7cb50a1d97e6e8e15b

- Check (0.12.0) - 747 KB:
主页： https://libcheck.github.io/check
下载： https://github.com/libcheck/check/releases/download/0.12.0/check-0.12.0.tar.gz
MD5 校验和： 31b17c6075820a4341195929411 86f70

- Coreutils (8.30) - 5,234 KB:
主页： http://www.gnu.org/software/coreutils/
下载： http://ftp.gnu.org/gnu/coreutils/coreutils-8.30.tar.xz
MD5 校验和： ab06d68949758971fe744db66b572816

- D-Bus (1.12.12) - 2,029 KB:
主页： https://www.freedesktop.org/wiki/Software/dbus
下载： https://dbus.freedesktop.org/releases/dbus/dbus-1.12.12.tar.gz
MD5 校验和： ea11069521beeee4e47f0086596a43c8

- DejaGNU (1.6.2) - 514 KB:
主页： http://www.gnu.org/software/dejagnu/
下载： 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： http://ftp.gnu.org/gnu/diffutils/diffutils-3.7.tar.xz
MD5 校验和： 4824adc0e95dbbf11dfbdfaad6a1e461

- E2fsprogs (1.44.5) - 7,448 KB:
主页： http://e2fsprogs.sourceforge.net/
下载： https://downloads.sourceforge.net/project/e2fsprogs/e2fsprogs/v1.44.5/e2fsprogs-1.44.5.tar.gz
MD5 校验和： 8d78b11d04d26c0b2dd149529441fa80

- Elfutils (0.176) - 8,444 KB:
主页： https://sourceware.org/ftp/elfutils/
下载： https://sourceware.org/ftp/elfutils/0.176/elfutils-0.176.tar.bz2
MD5 校验和： 077e4f49320cad82bf17a997068b1db9

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

- File (5.36) - 856 KB:
主页： https://www.darwinsys.com/file/
下载： ftp://ftp.astron.com/pub/file/file-5.36.tar.gz
MD5 校验和： 9af0eb3f5db4ae00fffc37f7b861575c

Note

File (5.36) 可能已经不能从列出的地址下载了。该站点管理员在新版本发布后删除了旧版本。合适版本的替代下载地址为： http://www.linuxfromscratch.org/lfs/download.html#ftp。

- Findutils (4.6.0) - 3,692 KB:
主页： http://www.gnu.org/software/findutils/
下载： http://ftp.gnu.org/gnu/findutils/findutils-4.6.0.tar.gz
MD5 校验和： 9936aa8009438ce185bea2694a997fc1

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

- Gawk (4.2.1) - 2,916 KB:
主页： http://www.gnu.org/software/gawk/
下载： http://ftp.gnu.org/gnu/gawk/gawk-4.2.1.tar.xz
MD5 校验和： 95cf553f50ec9f386b5dfcd67f30180a

- GCC (8.2.0) - 61,974 KB:
主页： https://gcc.gnu.org/
下载： http://ftp.gnu.org/gnu/gcc/gcc-8.2.0/gcc-8.2.0.tar.xz
MD5 校验和： 4ab282f414676496483b3e1793d07862

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

- Gettext (0.19.8.1) - 7,041 KB:
主页： http://www.gnu.org/software/gettext/
下载： http://ftp.gnu.org/gnu/gettext/gettext-0.19.8.1.tar.xz
MD5 校验和： df3f5690eaa30fd228537b00cb7b7590

- Glibc (2.29) - 16,129 KB:
主页： http://www.gnu.org/software/libc/
下载： http://ftp.gnu.org/gnu/glibc/glibc-2.29.tar.xz
MD5 校验和： e6c279d5b2f0736f740216f152acf974

- GMP (6.1.2) - 1,901 KB:
主页： http://www.gnu.org/software/gmp/
下载： http://ftp.gnu.org/gnu/gmp/gmp-6.1.2.tar.xz
MD5 校验和： f58fa8001d60c4c77595fbbb62b63c1d

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

- Grep (3.3) - 1,440 KB:
主页： http://www.gnu.org/software/grep/
下载： http://ftp.gnu.org/gnu/grep/grep-3.3.tar.xz
MD5 校验和： 05d0718a1b7cc706a4bdf8115363f1ed

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

- GRUB (2.02) - 5,970 KB:
主页： http://www.gnu.org/software/grub/
下载： https://ftp.gnu.org/gnu/grub/grub-2.02.tar.xz
MD5 校验和： 8a4a2a95aac551fb0fba860ceabfa1d3

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

- Iana-Etc (2.30) - 201 KB:
主页： http://freecode.com/projects/iana-etc
下载： http://anduin.linuxfromscratch.org/LFS/iana-etc-2.30.tar.bz2
MD5 校验和： 3ba3afb1d1b261383d247f46cb135ee8
- Inetutils (1.9.4) - 1,333 KB:
主页： http://www.gnu.org/software/inetutils/
下载： 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： https://launchpad.net/intltool/trunk/0.51.0/+download/intltool-0.51.0.tar.gz
MD5 校验和： 12e517cac2b57a0121cda351570f1e63
- IPRoute2 (4.20.0) - 691 KB:
主页： https://www.kernel.org/pub/linux/utils/net/iproute2/
下载： https://www.kernel.org/pub/linux/utils/net/iproute2/iproute2-4.20.0.tar.xz
MD5 校验和： f3dab4c812812bbb5873cb90f471bcbf
- Kbd (2.0.4) - 1,008 KB:
主页： http://ftp.altlinux.org/pub/people/legion/kbd
下载： https://www.kernel.org/pub/linux/utils/kbd/kbd-2.0.4.tar.xz
MD5 校验和： c1635a5a83b63aca7f97a3eab39ebaa6
- Kmod (26) - 540 KB:
下载： https://www.kernel.org/pub/linux/utils/kernel/kmod/kmod-26.tar.xz
MD5 校验和： 1129c243199bdd7db01b55a61aa19601
- Less (530) - 332 KB:
主页： http://www.greenwoodsoftware.com/less/
下载： http://www.greenwoodsoftware.com/less/less-530.tar.gz
MD5 校验和： 6a39bccf420c946b0fd7ffc64961315b
- Libcap (2.26) - 66 KB:
主页： https://sites.google.com/site/fullycapable/
下载： https://www.kernel.org/pub/linux/libs/security/linux-privs/libcap2/libcap-2.26.tar.xz
MD5 校验和： 968ac4d42a1a71754313527be2ab5df3
- Libffi (3.2.1) - 920 KB:
主页： https://sourceware.org/libffi/
下载： ftp://sourceware.org/pub/libffi/libffi-3.2.1.tar.gz
MD5 校验和： 83b89587607e3eb65c70d361f13bab43
- Libpipeline (1.5.1) - 965 KB:
主页： http://libpipeline.nongnu.org/
下载： http://download.savannah.gnu.org/releases/libpipeline/libpipeline-1.5.1.tar.gz
MD5 校验和： 4c8fe6cd85422baafd6e060f896c61bc
- Libtool (2.4.6) - 951 KB:
主页： http://www.gnu.org/software/libtool/
下载： http://ftp.gnu.org/gnu/libtool/libtool-2.4.6.tar.xz
MD5 校验和： 1bfb9b923f2c1339b4d2ce1807064aa5
- Linux (4.20.12) - 101,841 KB:
主页： https://www.kernel.org/
下载： https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.20.12.tar.xz
MD5 校验和： edd3015435d60598b99cf6aaf223710e

Note

Linux 内核更新相对比较频繁，大多数时候是因为发现了安全漏洞。当前最新的 4.20.x 内核版本应该都可以使用，除非在勘误页有其他说明。
对于网速较慢或带宽费用较高的用户如果希望更新 Linux 内核，可以将基线版本的软件包和补丁分开下载。针对次要版本中一连串的补丁程度的更新，这种操作方式也许能让你少费些时间，或者说功夫。

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

- Make (4.2.1) - 1,375 KB:
主页： http://www.gnu.org/software/make/
下载： http://ftp.gnu.org/gnu/make/make-4.2.1.tar.bz2
MD5 校验和： 15b012617e7c44c0ed482721629577ac

- Man-DB (2.8.5) - 1,746 KB:
主页： https://www.nongnu.org/man-db/
下载： http://download.savannah.gnu.org/releases/man-db/man-db-2.8.5.tar.xz
MD5 校验和： c5c6c3434be14a5527d43b5ad0f09a13

- Man-pages (4.16) - 1,592 KB:
主页： https://www.kernel.org/doc/man-pages/
下载： https://www.kernel.org/pub/linux/docs/man-pages/man-pages-4.16.tar.xz
MD5 校验和： ad9f1ff81276fe8d90d077484d6d4b5e

- Meson (0.49.2) - 1,310 KB:
主页： https://mesonbuild.com
下载： https://github.com/mesonbuild/meson/releases/download/0.49.2/meson-0.49.2.tar.gz
MD5 校验和： 0267b0871266056184c484792572c682

- MPC (1.1.0) - 685 KB:
主页： http://www.multiprecision.org/
下载： 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： http://www.mpfr.org/mpfr-4.0.2/mpfr-4.0.2.tar.xz
MD5 校验和： 320fbc4463d4c8cb1e566929d8adc4f8

- Ninja (1.9.0) - 187 KB:
主页： https://ninja-build.org/
下载： https://github.com/ninja-build/ninja/archive/v1.9.0/ninja-1.9.0.tar.gz
MD5 校验和： f340be768a76724b83e6daab69009902

- Ncurses (6.1) - 3,288 KB:
主页： http://www.gnu.org/software/ncurses/
下载： http://ftp.gnu.org/gnu/ncurses/ncurses-6.1.tar.gz
MD5 校验和： 98c889aaf8d23910d2b92d65be2e737a

- OpenSSL (1.1.1a) - 8,160 KB:
主页： https://www.openssl.org/
下载： https://openssl.org/source/openssl-1.1.1a.tar.gz
MD5 校验和： 963deb2272d6be7d4c2458afd2517b73

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

- Perl (5.28.1) - 12,083 KB:
主页： https://www.perl.org/
下载： https://www.cpan.org/src/5.0/perl-5.28.1.tar.xz
MD5 校验和： fbb590c305f2f88578f448581b8cf9c4

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

- Procps (3.3.15) - 884 KB:
主页： https://sourceforge.net/projects/procps-ng
下载： https://sourceforge.net/projects/procps-ng/files/Production/procps-ng-3.3.15.tar.xz
MD5 校验和： `2b0717a7cb474b3d6dfdeedfbad2eccc`

- Psmisc (23.2) - 297 KB:
主页： http://psmisc.sourceforge.net/
下载： https://sourceforge.net/projects/psmisc/files/psmisc/psmisc-23.2.tar.xz
MD5 校验和： `0524258861f00be1a02d27d39d8e5e62`

- Python (3.7.2) - 16,648 KB:
主页： https://www.python.org/
下载： https://www.python.org/ftp/python/3.7.2/Python-3.7.2.tar.xz
MD5 校验和： `df6ec36011808205beda239c72f947cb`

- Python Documentation (3.7.2) - 6,072 KB:
下载： https://docs.python.org/ftp/python/doc/3.7.2/python-3.7.2-docs-html.tar.bz2
MD5 校验和： `107ade7bb17efd104a22b2d457f4cb67`

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

- Sed (4.7) - 1,268 KB:
主页： http://www.gnu.org/software/sed/
下载： http://ftp.gnu.org/gnu/sed/sed-4.7.tar.xz
MD5 校验和： `777ddfd9d71dd06711fe91f0925e1573`

- Shadow (4.6) - 1,639 KB:
下载： https://github.com/shadow-maint/shadow/releases/download/4.6/shadow-4.6.tar.xz
MD5 校验和： `b491fecbf1232632c32ff8f1437fd60e`

- systemd (240) - 7,412 KB:
主页： https://www.freedesktop.org/wiki/Software/systemd/
下载： https://github.com/systemd/systemd/archive/v240/systemd-240.tar.gz
MD5 校验和： `0e4f91b513d4b04e2c10a5173e5a87b2`

- systemd Man Pages(240) - 460 KB:
主页： https://www.freedesktop.org/wiki/Software/systemd/
下载： http://anduin.linuxfromscratch.org/LFS/systemd-man-pages-240.tar.xz
MD5 校验和： `ca49a25e1cf330b02adc07218f430dae`

> **Note**
>
> Linux From Scratch 团队自制了 systemd man 手册的源码包。之所以这么做，是为了避免不必要的依赖。

- Tar (1.31) - 2,052 KB:
主页： http://www.gnu.org/software/tar/
下载： http://ftp.gnu.org/gnu/tar/tar-1.31.tar.xz
MD5 校验和： `bc9a89da1185ceb2210de12552c43ce2`

- Tcl (8.6.9) - 9,772 KB:
主页： http://tcl.sourceforge.net/
下载： https://downloads.sourceforge.net/tcl/tcl8.6.9-src.tar.gz
MD5 校验和： `aa0a121d95a0e7b73a036f26028538d4`

- Texinfo (6.5) - 4,399 KB:
主页： http://www.gnu.org/software/texinfo/
下载： http://ftp.gnu.org/gnu/texinfo/texinfo-6.5.tar.xz
MD5 校验和： `3715197e62e0e07f85860b3d7aab55ed`

- Time Zone Data (2018i) - 369 KB:
主页： https://www.iana.org/time-zones
下载： https://www.iana.org/time-zones/repository/releases/tzdata2018i.tar.gz
MD5 校验和： `b3f0a1a789480a036e58466cd0702477`

- Util-linux (2.33.1) - 4,542 KB:
主页： http://freecode.com/projects/util-linux
下载： https://www.kernel.org/pub/linux/utils/util-linux/v2.33/util-linux-2.33.1.tar.xz
MD5 校验和： `6fcfea2043b5ac188fd3eed56aeb5d90`

- Vim (8.1) - 10,995 KB:
主页： https://www.vim.org
下载： ftp://ftp.vim.org/pub/vim/unix/vim-8.1.tar.bz2
MD5 校验和： `1739a1df312305155285f0cfa6118294`

- XML::Parser (2.44) - 232 KB:
主页： https://github.com/chorny/XML-Parser
下载： https://cpan.metacpan.org/authors/id/T/TO/TODDR/XML-Parser-2.44.tar.gz
MD5 校验和： `af4813fe3952362451201ced6fbce379`

- Xz Utils (5.2.4) - 1030 KB:
主页： https://tukaani.org/xz
下载： https://tukaani.org/xz/xz-5.2.4.tar.xz
MD5 校验和： `003e4d0b1b1899fc6e3000b24feddf7c`

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

这些安装包的总计：约 381 MB

3.3. 需要的补丁

除了下载软件包外，还需要几个补丁。这些补丁修正了软件包中应该由维护者来解决的问题。补丁也会对软件包做一些小调整方便大家使用。构建 LFS 系统需要下面的补丁：

- Bzip2 文档补丁 - 1.6 KB:
下载： http://www.linuxfromscratch.org/patches/lfs/8.4/bzip2-1.0.6-install_docs-1.patch
MD5 校验和： `6a5ac7e89b791aae556de0f745916f7f`

- Coreutils 国际化修复补丁 - 168 KB:
下载： http://www.linuxfromscratch.org/patches/lfs/8.4/coreutils-8.30-i18n-1.patch
MD5 校验和： `a9404fb575dfd5514f3c8f4120f9ca7d`

- Glibc FHS 补丁 - 2.8 KB:
下载： http://www.linuxfromscratch.org/patches/lfs/8.4/glibc-2.29-fhs-1.patch
MD5 校验和： `9a5997c3452909b1769918c759eff8a2`

- Kbd Backspace/Delete 键修复补丁 - 12 KB:
下载： http://www.linuxfromscratch.org/patches/lfs/8.4/kbd-2.0.4-backspace-1.patch
MD5 校验和： `f75cca16a38da6caa7d52151f7136895`

- systemd 安全补丁 - 14 KB:
下载： http://www.linuxfromscratch.org/patches/lfs/8.4/systemd-240-security_fixes-2.patch
MD5 校验和： `10abebce8ff5d9fd402623ace39b5ab8`

这些补丁文件总大小: 约 198.4 KB

除了以上所要求的补丁外，还有一些由 LFS 社区创建的可选补丁。这些可选补丁或是解决了一些小问题，或是打开某个默认关闭的功能。请查阅补丁数据库 http://www.linuxfromscratch.org/patches/downloads/，获取适合你系统需求的额外补丁。

Chapter 4. 最后的准备工作

4.1. 简介

在本章，我们还需要为构建临时系统做一些额外的准备工作。我们会在 $LFS 中新建一个用于安装临时工具的目录，增加一个非特权用户用于降低风险，并为该用户创建合适的构建环境。我们还会解释用于测量构建 LFS 软件包花费时间的单位，或称「SBUs」，并给出一些关于软件包测试套件的信息。

4.2. 创建目录 $LFS/tools

所有 第 5 章 中编译的软件都会安装到 $LFS/tools 中，以确保和 Chapter#6中编译的软件相互分离。这里编译的软件是临时工具，并不会成为最终的 LFS 系统中的一部分。将这些软件保存在单独的目录中，用完后方便弃置。这样做也可以防止这些程序在宿主机生成目录中突然停止工作（在 第 5 章 中很容易发生意外）。

以 root 用户运行以下的命令来创建需要的文件夹：

```
mkdir -v $LFS/tools
```

下一步是在宿主系统中创建 /tools 的符号链接。将其指向 LFS 分区中新建的目录。同样以 root 用户运行下面的命令：

```
ln -sv $LFS/tools /
```

> **Note**
>
> 上面的命令是正确的。ln 命令有一些语法变种，所以在报出你觉得可能是错误的信息之前检查一下 info coreutils ln 和 ln(1)。

创建的符号链接使得编译的工具链总是指向 /tools 文件夹，也就是说编译器、汇编器以及链接器无论是在第五章中（我们仍然使用宿主机的一些工具的时）还是之后（当我们「chrooted」到 LFS 分区时）都可以工作。

4.3. 添加 LFS 用户

当以 root 用户登录时，犯一个小错误可能会破坏或摧毁整个系统。因此，我们建议在本章中以非特权用户编译软件包。你当然可以使用你自己的用户名，但为了使其更容易建立一个干净的工作环境，创建一个名为 lfs 的新用户作为新组（同样命名为 lfs）的成员，并在安装过程中使用这个用户。以 root 用户运行以下命令来添加新用户：

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

命令行选项释义：

-s /bin/bash
: 将 bash 设置为 lfs 用户的默认 shell。

-g lfs
: 这个选项将用户 lfs 添加到组 lfs 中。

-m
: 为 lfs 用户创建主目录。

-k /dev/null
: 这个参数通过改变输入位置为特殊的空（null）设备，以防止可能从框架目录（默认是 /etc/skel）复制文件。

lfs
: 这是创建的组和用户的实际名称。

要以 lfs 用户身份登录（相较于以 root 身份登录的情况下切换到 lfs 用户时，无需为 lfs 用户设置密码），需要给 lfs 用户一个密码：

```
passwd lfs
```

通过更改目录所有者为 lfs，为用户 lfs 赋予了访问 $LFS/tools 目录的所有权限：

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

如果你按照建议创建了单独的工作目录，给 lfs 用户赋予这个目录的所有权：

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

下一步，以 lfs 用户身份登录。可以能通过一个虚拟控制台、显示控制器，或者下面的切换用户命令完成：

```
su - lfs
```

这个「-」授意 su 启动登录 shell，而非 non-login shell。关于这两种 shell 类型的区别，可以在 bash(1) 和 info bash 中查看详细详情。

4.4. 设置环境

通过为 bash shell 创建两个开机启动的文件，设置合适的工作环境。当以 lfs 用户身份登录时，运行以下命令创建一个新的 .bash_profile 文件：

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

当以 lfs 用户身份登录时，初始 shell 通常是一个 login 的 shell，它先读取宿主机的 /etc/profile 文件（很可能包括一些设定和环境变量），然后是 .bash_profile 文件。.bash_profile 中的命令 exec env -i.../bin/bash 用一个除了 HOME，TERM 和 PS1 变量外，其他环境完全为空的新 shell 代替运行中的 shell。这能确保不会有潜在的和意想不到的危险环境变量，从宿主机泄露到构建环境中。这样做主要是为了确保环境的干净。

新的 shell 实例是一个 non-login 的 shell，不会读取 /etc/profile 或者 .bash_profile 文件，而是读取 .bashrc。现在，创建 .bashrc 文件：

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

set +h 命令关闭了 bash 的哈希功能。哈希通常是个好用的功能——bash 用一个哈希表来记录可执行文件的完整路径，以规避对 PATH 进行检索的时间和对同一个可执行文件的重复寻找。然而，新工具在安装后，应马上使用。通过关闭哈希功能，程序执行的时候就会一直搜索 PATH。如此，新编译的工具一旦可用，shell 便能在马上在文件夹 $LFS/tools 中找到它们，而不会去记录存在于不同地方的旧版该程序。

设置用户文件新建时的掩码（umask）为 022，以确保新建的文件和目录只有其所有者可写，但任何人都可读可执行（假设系统调用的 open(2) 使用的是默认模式，新文件将使用 664 权限模式、文件夹为 755 模式）。

LFS 变量应设置成选定的挂载点。

LC_ALL 变量控制某些程序的本地化，使它们的消息遵循特定国家的惯例。设置 LC_ALL 为「POSIX」或「C」（两者是等价的），确保在 chroot 环境中一切能如期望的那样进行。

LFS_TGT 变量设置了一个虽非默认，但在构建交叉编译器、连接器和交叉编译临时工作链时，用得上到的兼容的机器说明。Section#5.2, “工具链技术说明中包含更多信息。

通过把 /tools/bin 放在标准 PATH 变量的前面，使得所有在 第 5 章 中安装的程序，一经安装 shell 便能马上使用。与之配合的关闭哈希功能，能在第五章环境中的程序在可用的情况下，限制使用宿主机中旧程序的风险。

最后，启用刚才创建的用户配置，为构建临时工具完全准备好环境：

```
source ~/.bash_profile
```

4.5. 关于 SBUs

在着手之前,不少人想知道编译和安装一个软件包到底需要多长的时间。因为 Linux From Scratch 能够运行于众多的硬件上，所以具体的编译时间无法一概而论。举一个简单的例子：在最快的系统上，编译最大的软件包（Glibc）需要约 20 分钟，但在慢的系统上有可能需要 3 天！所以，这里使用标准构建单元（SBU）来代替具体的编译时间。

SBU 衡量方式如下。我们编译的第一个软件包是 第 5 章 的 Binutils。将这个软件包在编译时所需要的时间，作为标准构建单元（SBU）。其它软件的构建时间，都以其为标准进行比较。

举个例子，假如编译某个软件耗时 4.5 SBUs。这意味着如果这个系统，在编译 Binutils 需要 10 分钟，大概需要约 45 分钟来构建此软件包。很幸运，大部分的系统在生成 Binutils 时花的时间都要比这个系统要短。

一般来说，SBU 的结果并不完全准确，因为影响编译的因素太多，包括宿主机系统中 GCC 的版本也会影响到。所以更多的时候，这仅仅是提供一个编译和安装时间的预估。然而，在某些情况下，这个数字可能偏差约十几分钟。

Note

对于大多数带有多个处理器（或内核）的现代操作系统而言，可以通过设置环境变量或者是告知 make 程序具体可用的处理器数目，通过「并行 make」来减少编译的时间。例如，对于 Core2Duo 可以通过以下参数实现两个处理器同时编译：

```
export MAKEFLAGS='-j 2'
```

或者直接这样来构建：

```
make -j2
```

以该方式使用多处理器时，SBU 值可能比书中的正常值还要大。某些情况下，make 过程仅仅是简单的就失败了。分析错误日志也十分困难：因为不同处理器之间的执行路线是交错的。如果你在构建过程中遇到了问题，为了正确分析错误信息，最好返回单处理器。

4.6. 关于测试套件

很多软件包都提供相应的测试套件。为新构建的软件包运行测试套件是非常好的习惯，因为这样做可以「保证」所有功能都已编译正确。经由一系列的测试，套件往往能够检查出软件包的功能是否都如开发人员预想的那样。然而，这并不能保证所测试的软件包万无一失。

有一些测试套件要相较而言更为重要。例如，核心工具链软件包——GCC, Binutils 和 Glibc——对于对于一个系统的正常运转起到至关重要的作用。要完成 GCC 和 Glibc 的测试套件可能要花费很长的时间，特别是对于硬件比较慢的设备来说，但还是强烈推荐完成它们！

Note

经验表明，在 第 5 章 中运行测试套件并不是什么好主意。在那章有何无法回避的现实，就是宿主机或多或少会对测试产生影响，经常导致一些令人摸不着头脑的错误信息。因为在 第 5 章 中构建的这些工具只是零时的，最终我们并不需要它们，所以我们并不推荐普通读者在 第 5 章 中运行测试套件。虽然为测试者和开发者提供了测试套件的说明，但是这依旧是可选项。

对 Binutils 和 GCC 执行测试套件时可能会使伪终端（PTYs）耗尽。造成大量的测试失败。造成问题的原因有很多，但最有可能的原因是宿主系统没能正确设置 `devpts` 文件系统。针对这个问题在 http://www.linuxfromscratch.org/lfs/faq.html#no-ptys 中有更详尽的讨论。

还有一些测试套件运行错误，是开发人员已知且视为不重要的。查看 http://www.linuxfromscratch.org/lfs/build-logs/8.4/ 中的日志，确认这些失败信息是否都是意料之中的。该网址中涉及的内容会贯穿全书所有的测试。

Chapter 5. 构建临时系统

5.1. 简介

本章将向您展示如何构造一个最小的 Linux 系统。该系统将包含刚好足够构建 Chapter#6中最终 LFS 系统所需的工具，以及一个比最小环境具有更好用户便利性的工作环境。

构建这个最小系统有两个步骤。第一步，是构建一个与宿主系统无关的新工具链（编译器、汇编器、链接器、库和一些有用的工具）。第二步则是使用该工具链，去构建其它的基础工具。

本章中编译得到的文件将被安装在目录 `$LFS/tools` 中，以确保在下一章中安装的文件和宿主系统生成的目录相互分离。由于此处编译的软件包都是临时性的，因此，我们不愿意它们污染后面即将构成的 LFS 系统。

5.2. 工具链技术说明

这一节解释总体构建方法之中的某些基本原理和技术细节。本节中的所有问题并无需马上消化。在进行实际构建的过程中，绝大部分的信息会变得愈加清晰。过程中随时都可以查阅本小节的内容。

纵览 第 5 章 的目标是生成一个临时的系统，这个系统中包含一个已知的较好工具集，并且工具集可以独立于宿主系统。通过使用 chroot，其余各章中的命令将被包含在此环境中，以保证目标 LFS 系统能够洁净且无故障地生成。该构建过程的设计就是为了使得新读者承担最少的风险，同时还能有最好的指导价值。

> **Note**
>
> 在继续前，请留心工作平台的名称，它时常被称为目标三元组。要确定目标三元组的名称有一个简单的法子，那就是运行许多软件包的源码附带的脚本 config.guess。解压 Binutils 的源码并运行脚本：**`./config.guess`** 并注意它的输出。举一个例子，Intel 的 32 位处理器输出会是 i686-pc-linux-gnu。而在一个 64 位系统上，则会是 x86_64-pc-linux-gnu。
>
> 也请留心平台的动态链接器的的名称，它时常被称为动态加载器（不要与 Binutils 中的标准链接器 ld 混为一谈）。动态链接器由 Glibc 提供，用于寻找和加载程序所需的共享库，为程序的运行做准备，以及运行程序。32 位的 Intel 机器，动态链接器的名称是 `ld-linux.so.2`（64 位系统则是 `ld-linux-x86-64.so.2`）。确定动态链接名的一个确定的方法，就是检查随机二进制文件，通过在宿主机运行：**`readelf -l <name of binary> | grep interpreter`** 并注意它的输出。覆盖全平台的权威参考在 Glibc 源码树根目录的 `shlib-versions` 文件中。

下面是 第 5 章 构建方法的几个关键技术点：

- 通过改变 `LFS_TGT` 变量的目标系统三段式中的「vendor（供应商）」字段，稍微调整一下工作平台的名称，以保证第一遍构建 Binutils 和 GCC 时能够生成兼容的交叉链接器和交叉编译器。此处的交叉链接器和交叉编译器生成的二进制文件与当前的硬件兼容，而不是用于其它的硬件架构。
- 临时库经交叉编译获得。由于交叉编译原本就不应该依赖于宿主系统，因此，通过降低宿主系统的头文件或库进入新工具的可能性，该方法可去除目标系统中潜在的污染。交叉编译的方式，还可以在 64 位硬件平台上同时构建出 32 位和 64 位库。
- 谨慎地操作 GCC 源码告诉编译器，哪个是即将被采用的目标动态链接器。

Binutils 是首个安装的包，因为无论是执行 GCC 还是 Glibc 的 configure 时，都将围绕关汇编器和链接器实施多项特性测试，来判断哪些软件特性要启用或是禁用。其重要性可能更甚于你对它的第一印象。GCC 或 Glibc 的错误配置会导致工具链出现难以捉摸的问题，可能直到整个构建过程接近尾声时，这些问题才会显现出来。不过，通常情况下，在你进行大量的无用功之前，一次测试套件的失败便会将该错误高亮出来。

Binutils 将其汇编器和链接器安装在两处，`/tools/bin` 和 `/tools/$LFS_TGT/bin`。有一处的工具是另一处的硬链接。链接器的一个重要方面是它的库搜索顺序。可以给 ld 传递参数 *`--verbose`* 获取详细信息。例如，**`ld --verbose | grep SEARCH`** 将说明当前的搜索路径及其顺序。通过编译一个 dummy（假）程序并向链接器传递 *`--verbose`* 参数，可显示 ld 都链接了哪些文件。例如，**`gcc dummy.c -Wl,--verbose 2>&1 | grep succeeded`** 将显示链接过程中成功打开的所有文件。

下一个安装的包是 GCC。下面便是运行 GCC 的 configure 的输出示例：

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

基于上述原因，这很重要。这还说明了 GCC 的配置脚本并不会搜索 PATH 目录来寻找使用什么工具。不过，在 gcc 自身的实际运行中，并不需要使用同样的搜索路径。运行：**`gcc -print-prog-name=ld`** 可获知 gcc 使用是何种标准链接器。

在编译 dummy（假）程序时，向 gcc 传递命令行选项 *`-v`* 可获得详细信息。例如，**`gcc -v dummy.c`** 将显示预处理器、编译和汇编阶段的详细信息，包括 gcc 的 include 搜索路径及其顺序。

下一个安装的是经过净化的 Linux API 头文件。这些头文件可使得标准 C 库（Glibc）与 Linux 内核提供的特性进行交行交互。

下一个安装的软件包是 Glibc。构建 Glibc 时，最重要的几个注意点是编译器、二进制工具和内核头文件。编译器通常不成问题，因为 Glibc 将一直使用传递给它配置脚本的，有关 *`--host`* 参数的编译器。如，在我们的这个场景中，编译器就是 i686-lfs-linux-gnu-gcc。而二进制工具和内核头文件可能就要复杂一些了。因此，请不要冒险，并利用可用的配置开关来强制正确的选择。configure 运行完毕，目录 `glibc-build` 下的文件 `config.make` 包含有所有的重要细节。需要注意的是，*`CC="i686-lfs-gnu-gcc"`* 用来控制使用哪个二进制工具，*`-nostdinc`* 和 *`-isystem`* 标志用来控制编译器的 include 搜索路径。这些条目强调了 Glibc 包的一个重要方面，即其构建机制是非常自给自足的，通常并不依赖默工具链的默认设置。

在第二遍编译 Binutils 过程中，我们能够利用配置开关 *`--with-lib-path`* 来控制 ld 的库搜索路径。

第二遍编译 GCC 时，也需要修改其源代码，以告诉 GCC 使用新的动态链接器。如果不加修改，将导致 GCC 的自身程序中嵌入源自宿主系统目录 `/lib` 中的动态链接器，这违背了独立于宿主系统的目的。正是基于前面的这个出发点，核心工具链是自包含和自托管的。第 5 章 其它的软件包都将在 `/tools` 中的新 Glibc 的基础上进行构建。

在进入 Chapter#6中的 chroot 环境前，将安装的首个主要的软件包是 Glibc，这是因为它天生具有前面提及的自给自足特点。一旦将 Glibc 安装到 `/usr` 中，我们将快速改变工具链的默认设置，然后继续构建目标 LFS 系统的其余部分。

5.3. 通用编译指南

编译软件包时，本指南假设你已知晓这几点：

- 有几个软件包，在编译之前需要打补丁来规避一些问题。有的补丁在本章和下一章中都需要，但有些只有其中一章需要。因此，如果某章看起来缺少某个补丁的下载说明时，不用担心。安装补丁的时候也许会遇到关于 offset 或者 fuzz 的警告信息。别担心这些警告，补丁还是会成功安装的。
- 在大部分软件包的编译过程中，屏幕上都可能出现几个警告。这都很正常，可以安全地忽略。这些警告正如它们描述的那样，是对使用过时的 C 或 C++ 语法的警告，而不是这些语法不可用。C 语言的标准经常改变，一些软件包仍然在使用旧的标准。这并不是一个问题，不过确实会弹出警告。
- 最后确认一次，是否正确设置了 LFS 环境变量：

```
echo $LFS
```

  确认输出显示的是 LFS 分区挂载点的路径，在我们的例子中，也就是 `/mnt/lfs`。

- 最后，有两个重要事项必须强调：

> **Important**
>
> 编译指南假定你已经正确地设置了 Host System Requirements 和符号链接：
>
> - shell 使用的是 bash
> - sh 是到 bash 的符号链接。
> - /usr/bin/awk 是到 gawk 的符号链接。
> - /usr/bin/yacc 是到 bison 的符号链接，或者是一个执行 bison 的小脚本。

Important

再次强调构建的过程：

1. 把所有源文件和补丁放到 chroot 环境可访问的目录，例如 `/mnt/lfs/sources/`。但是千万不能把源文件放在 `/mnt/lfs/tools/` 中。
2. 进入到源文件目录。
3. 对于每个软件包：
   a. 用 tar 程序解压要编译的软件包。在第五章中，确保解压软件包时你使用的是 lfs 用户。
   b. 进入到解压后创建的目录中。
   c. 根据指南说明编译软件包。
   d. 回退到源文件目录。
   e. 除非特别说明，删除解压出来的目录。

5.4. Binutils-2.32 - 第 1 遍

Binutils 软件包包含一个链接器、一个汇编器、以及其它处理目标文件的工具。

大致构建用时: 1 SBU
所需磁盘空间: 580 MB

5.4.1. 安装交叉编译的 Binutils

> **Note**
>
> 返回前面章节重新阅读注意事项。了解标记为重要的注意事项能在后面帮你省去很多麻烦。

第一个编译的软件包是 Binutils 软件包，这点很重要，因为 Glibc 和 GCC 都会对可用的链接器和汇编器执行各种测试，以决定启用它们自身的哪些功能。

Binutils 手册建议，在源码目录之外一个专门的编译目录里面编译 Binutils：

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

> **Note**
>
> 为了衡量在本书中其余部分所使用 SBU 值，我们要测量一下这个软件包从配置到包括第一次安装在内的编译时间。为了轻松的做到这点，会用类似 **`time { ./configure ... && ... && make install; }`** 的方式将命令包裹在 time 命令中。

> **Note**
>
> 第五章中，大致的构建 SBU 值和所需磁盘空间不包括测试套件数据。

现在准备编译 Binutils：

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

配置选项的含义：

`--prefix=/tools`
告诉配置脚本将 Binutils 程序安装到 `/tools` 文件夹。

`--with-sysroot=$LFS`
用于交叉编译，告诉编译系统在 $LFS 中查找所需的目标系统库。

`--with-lib-path=/tools/lib`
指定需要配置使用的链接器的库路径。

`--target=$LFS_TGT`
因为 LFS_TGT 变量中的机器描述和 config.guess 脚本返回的值略有不同，这个选项会告诉 configure 脚本调整 Binutils 的构建系统来构建一个交叉链接器。

`--disable-nls`
这会禁止国际化（i18n），因为国际化对临时工具来说没有必要。

`--disable-werror`
这会防止来自宿主编译器的警告事件导致停止编译。

继续编译软件包：

```
make
```

现在编译完成了。通常现在我们会运行测试套件，但在这个初期阶段，测试套件框架（Tcl、Expect 和 DejaGNU）还没有就绪。在此进行测试的收效甚微，因为第一遍编译的程序很快会被第二遍的代替。

如果是在 x86_64 上构建，创建符号链接，以确保工具链的完整性：

```
case $(uname -m) in
  x86_64) mkdir -v /tools/lib && ln -sv lib /tools/lib64 ;;
esac
```

安装软件包:

```
make install
```

该软件包的详细信息位于 Section#6.16.2, “Binutils 内容”。

5.5. GCC-8.2.0 - 第 1 遍

GCC 软件包包括 GNU 编译器集，其中有 C 和 C++ 的编译器。

大致构建用时: 11 SBU
所需磁盘空间: 2.9 GB

5.5.1. 安装交叉编译的 GCC

现在 GCC 需要 GMP、MPFR 和 MPC 软件包。在你的主机发行版中可能并不包括这些软件包，它们将和 GCC 一起编译。将每个解压软件包到 GCC 的目录下，并重命名解压后得到的目录，以便 GCC 编译过程中能自动使用这些软件：

> **Note**
>
> 读者经常对本章节产生误解。过程与之前的章节（软件包构建说明）中提到的一样。首先从源目录中解压 gcc 的源码包，然后进入创建的目录中。接着才可以执行下面的指令。

```
tar -xf ../mpfr-4.0.2.tar.xz
mv -v mpfr-4.0.2 mpfr
tar -xf ../gmp-6.1.2.tar.xz
mv -v gmp-6.1.2 gmp
tar -xf ../mpc-1.1.0.tar.gz
mv -v mpc-1.1.0 mpc
```

下面的指令将会修改 GCC 默认的动态链接器的位置，安装到 `/tools` 目录中的。并将 `/usr/include` 从 GCC 的 include 检索路径中移除。执行：

```
for file in gcc/config/{linux,i386/linux{,64}}.h
do
  cp -uv $file{,.orig}
  sed -e 's@/lib\(64\)\?\(32\)\?/ld@/tools&@g' \
      -e 's@/usr@/tools@g' $file.orig > $file
  echo '
#undef STANDARD_STARTFILE_PREFIX_1
#undef STANDARD_STARTFILE_PREFIX_2
#define STANDARD_STARTFILE_PREFIX_1 "/tools/lib/"
#define STANDARD_STARTFILE_PREFIX_2 ""' >> $file
  touch $file.orig
done
```

如果上面的内容看起来有些难以理解，那让我们慢慢消化吧。首先，我们复制文件 `gcc/config/linux.h`，`gcc/config/i386/linux.h`，和 `gcc/config/i368/linux64.h`，给复制的文件加上「.orig」后缀。然后第一个 sed 表达式在每个「/lib/ld」，「/lib64/ld」或者「/lib32/ld」实例前面增加「/tools」，第二个 sed 表达式替换「/usr」的硬编码实例。然后，我们添加这改变默认 startfile 前缀到文件末尾的定义语句。注意「/tools/lib/」后面的「/」是必须的。最后，我们用 touch 更新复制文件的时间戳。当与 cp -u 一起使用时，可以防止命令被无意中运行两次造成对原始文件意外的更改。

最后，在 x86_64 的主机上，为 64 位的库设置默认目录名至「lib」：

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

GCC 手册建议在源目录之外一个专门的编译目录中编译 GCC：

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

准备编译 GCC:

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

配置选项的含义:

--with-newlib
由于还没有可用的 C 库，这确保编译 libgcc 时定义了常数 inhibit_libc。这可以防止编译任何需要 libc 支持的代码。

--without-headers
创建一个完成的交叉编译器的时候，GCC 要求标准头文件和目标系统兼容。对于我们的目的来说，不需要这些头文件。这个选项可以防止 GCC 查找它们。

--with-local-prefix=/tools
GCC 会查找本地已安装的 include 文件的系统位置。默认是 `/usr/local`。把它设置为 `/tools` 能把主机位置中的 `/usr/local` 从 GCC 的搜索路径中排除。

--with-native-system-header-dir=/tools/include
GCC 默认会在 `/usr/include` 中查找系统头文件。和 sysroot 选项一起使用，会转换为 `$LFS/usr/include`。在后面两个章节中头文件会被安装到 `$LFS/tools/include` 。这个选项确保 gcc 能正确找到它们。第二次编译 GCC 时，同样的选项可以保证不会去寻找主机系统的头文件。

--disable-shared
这个选项强制 GCC 静态链接到它的内部库。我们这样做是为了避免与主机系统可能出现的问题。

--disable-decimal-float, --disable-threads, --disable-libatomic, --disable-libgomp, --disable-libmpx, --disable-libquadmath, --disable-libssp, --disable-libvtv, --disable-libstdcxx
这些选项取消了对十进制浮点数扩展、线程
化、libatomic、libgomp、libmpx、libitm、libquadmath、libsanitizer、libssp、libvtv、libcilkrts 和 C++ 标准库的支持。这些功能在编译交叉编译器的时候会导致编译失败，对于交叉编译临时 libc 来说也没有必要。

--disable-multilib
在 x86_64 机器上，LFS 还不支持 multilib 配置。这个选项对 x86 来说无害。

--enable-languages=c,c++
这个选项确保只编译 C 和 C++ 编译器。这些是现在唯一需要的语言。

运行命令编译 GCC:

```
make
```

现在编译完成了。在这里，通常会运行测试套件，但正如前面提到的，测试套件框架还没有准备好。在此进行测试的并没有太多好处，因为第一遍编译的程序很快会被取代。

安装软件包：

```
make install
```

该软件包的详细信息请参见： Section#6.21.2, “GCC 软件包内容”。

5.6. Linux-4.20.12 API 头文件

Linux API 头文件（在 linux-4.20.12.tar.xz 里）会将内核 API 导出给 Glibc 使用。

大致构建用时: 0.1 SBU
所需磁盘空间: 937 MB

5.6.1. 安装 Linux API 头文件

Linux 内核需要展示供系统 C 库（在 LFS 中是 Glibc）使用的应用程序编程接口（API）。这通过在 Linux 内核源代码 tar 包中包括一些 C 头文件来完成。

确认这里没有陈旧的文件且不依赖于之前的操作：

```
make mrproper
```

从源代码中提取用户可见的内核头文件。把他们保存在一个临时本地文件夹中然后复制到所需的位置，因为解压过程会移除目标文件夹中任何已有的文件。

```
make INSTALL_HDR_PATH=dest headers_install
cp -rv dest/include/* /tools/include
```

该软件包的详细信息请参见： Section#6.7.2, “Linux API 头文件内容”。

5.7. Glibc-2.29

Glibc 软件包包含了主要的 C 函数库。这个库提供了分配内存、搜索目录、打开关闭文件、读写文件、操作字符串、模式匹配、基础算法等基本程序。

大致构建用时: 5.1 SBU
所需磁盘空间: 885 MB

5.7.1. 安装 Glibc

Glibc 手册建议在源文件夹之外的一个专用文件夹中编译 Glibc：

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

下一步，准备编译 Glibc：

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

配置选项的含义:

`--host=$LFS_TGT, --build=$(../scripts/config.guess)`
这些选项的组合效果是 Glibc 的构建系统配置它自己用 `/tools` 里面的交叉链接器和交叉编译器交叉编译自己。

`--enable-kernel=3.2`
这告诉 Glibc 编译能支持 3.2 以及之后的内核库。更早的内核版本不受支持。

`--with-headers=/tools/include`
告诉 Glibc 利用刚刚安装在 tools 文件夹中的头文件编译自身，此能够根据内核的具体特性提供更好的优化。

在这个过程中，可能会出现下面的警告：

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

msgfmt 程序的缺失或者不兼容通常是无害的。这个 msgfmt 程序是 Gettext 软件包的一部分，主机发行版应该提供了。

> **Note**
>
> 有报告说用「parallel make」编译这个软件包的时候会失败。如果出现这种情况，用「-j1」选项重新运行 make 命令。

编译软件包:

```
make
```

安装软件包:

```
make install
```

Caution

到了这里，必须停下来确认新工具链的基本功能(编译和链接)都是像预期的那样正常工作。运行下面的命令进行全面的检查：

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

如果一切工作正常的话，这里应该没有错误，最后一个命令的输出形式会是：

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

注意 32 位机器上对应的解释器名字是 `/tools/lib/ld-linux.so.2`。

如果输出不是像上面那样或者根本就没有输出，那么可能某些地方出错了。调查并回溯这些步骤，找出问题所在并改正它。在继续之前必须解决这个问题。

一旦一切都顺利，清理测试文件：

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

Note

在后面的编译 Binutils 章节时会再一次检查工具链是否正确编译。如果 Binutils 编译失败，说明之前安装 Binutils、GCC、或者 Glibc 时某些地方出现了错误。

该软件包的详细信息请参见： Section#6.9.3, “Glibc 软件包内容”。

5.8. GCC-8.2.0 中的 Libstdc++

Libstdc++ 是标准的 C++ 库。需要用它来编译 C++ 代码（GCC 的一部分是用 C++ 写的），但是在构建 gcc-第 1 遍 时，我们需要推迟它的安装进程，因为依赖的 glibc，还未部署在 /tools 目录中。

Libstdc++ 是标准的 C++ 库。g++ 编译器正确运行需要它。

大致构建用时: 0.5 SBU
所需磁盘空间: 803 MB

5.8.1. 安装目标 Libstdc++

> **Note**
>
> Libstdc++ 是 GCC 源文件的一部分。你首先应该解压 GCC 的压缩包，然后进入 `gcc-8.2.0` 文件夹。

为 Libstdc++ 另外创建一个用于构建的文件夹并进入该文件夹：

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

准备编译 Libstdc++：

```
../libstdc++-v3/configure           \
    --host=$LFS_TGT                 \
    --prefix=/tools                 \
    --disable-multilib              \
    --disable-nls                   \
    --disable-libstdcxx-threads     \
    --disable-libstdcxx-pch         \
    --with-gxx-include-dir=/tools/$LFS_TGT/include/c++/8.2.0
```

配置选项的含义：

`--host=...`
指示使用我们刚才编译的交叉编译器，而不是 `/usr/bin` 中的。

`--disable-libstdcxx-threads`
由于我们还没有编译 C 线程库，C++ 的也还不能编译。

`--disable-libstdcxx-pch`
此选项防止安装预编译文件，此步骤并不需要。

`--with-gxx-include-dir=/tools/$LFS_TGT/include/c++/8.2.0`
这是 C++ 编译器搜索标准 include 文件的位置。在一般的编译中，这个信息自动从顶层文件夹中传入 Libstdc++ configure 选项。在我们的例子中，必须明确给出这信息。

编译 libstdc++：

```
make
```

安装库：

```
make install
```

该软件包的详细信息请参见： Section#6.21.2, “GCC 软件包内容”。

5.9. Binutils-2.32 - 第 2 遍

Binutils 软件包包含一个链接器、一个汇编器、以及其它处理目标文件的工具。

大致构建用时: 1.1 SBU
所需磁盘空间: 598 MB

5.9.1. 安装 Binutils

再次新建一个单独的编译文件夹：

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

准备编译 Binutils：

```
CC=$LFS_TGT-gcc                \
AR=$LFS_TGT-ar                 \
RANLIB=$LFS_TGT-ranlib         \
../configure                   \
    --prefix=/tools            \
    --disable-nls              \
    --disable-werror           \
    --with-lib-path=/tools/lib \
    --with-sysroot
```

新配置选项的含义:

CC=$LFS_TGT-gcc AR=$LFS_TGT-ar RANLIB=$LFS_TGT-ranlib
因为这是真正的原生编译 Binutils，设置这些变量能确保编译系统使用交叉编译器和相关的工具，而不是宿主系统中已有的。

--with-lib-path=/tools/lib
这告诉配置脚本在编译 Binutils 的时候指定库搜索目录，此处将 `/tools/lib` 传递到链接器。

--with-sysroot
sysroot 功能使链接器可以找到包括在其命令行中的其它共享对象明确需要的共享对象。否则的话，在某些主机上一些软件包可能会编译不成功。

编译软件包：

```
make
```

安装软件包：

```
make install
```

现在，为下一章的「Re-adjusting」阶段准备链接器：

```
make -C ld clean
make -C ld LIB_PATH=/usr/lib:/lib
cp -v ld/ld-new /tools/bin
```

make 参数的含义：

-C ld clean
告诉 make 程序移除所有 `ld` 子目录中编译过的文件。

-C ld LIB_PATH=/usr/lib:/lib
这个选项重新编译 `ld` 子目录中的所有文件。在命令行中指定 Makefile 的 `LIB_PATH` 变量可以使我们能够重写临时工具的默认值并指向正确的最终路径。该变量的值指定链接器的默认库搜索路径。下一章中会用到这个准备。

该软件包的详细信息请参见： Section#6.16.2, “Binutils 内容”。

5.10. GCC-8.2.0 - 第 2 遍

GCC 软件包包括 GNU 编译器集，其中有 C 和 C++ 的编译器。

大致构建用时: 14 SBU
所需磁盘空间: 3.4 GB

5.10.1. 安装 GCC

我们第一次编译 GCC 的时候安装了一些内部系统头文件。其中的一个 `limits.h` 会反过来包括对应的系统头文件 `limits.h`，在我们的例子中，是 `/tools/include/limits.h` 。但是，第一次编译 gcc 的时候 `/tools/include/limits.h` 并不存在，因此 GCC 安装的内部头文件只是部分的自包含文件，并不包括系统头文件的扩展功能。这足以编译临时 libc，但是这次编译 GCC 要求完整的内部头文件。使用和正常情况下 GCC 编译系统使用的相同的命令创建一个完整版本的内部头文件：

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

再一次更改 GCC 的默认动态链接器的位置，使用安装在 `/tools` 的那个。

```
for file in gcc/config/{linux,i386/linux{,64}}.h
do
  cp -uv $file{,.orig}
  sed -e 's@/lib\(64\)\?\(32\)\?/ld@/tools&@g' \
      -e 's@/usr@/tools@g' $file.orig > $file
  echo '
#undef STANDARD_STARTFILE_PREFIX_1
#undef STANDARD_STARTFILE_PREFIX_2
#define STANDARD_STARTFILE_PREFIX_1 "/tools/lib/"
#define STANDARD_STARTFILE_PREFIX_2 ""' >> $file
  touch $file.orig
done
```

如果是在 x86_64 环境上构建，为 64 位库改变默认目录名至「lib」：

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

和第一次编译 GCC 一样，它要求 GMP、MPFR 和 MPC 软件包。解压 tar 包并把它们重名为到所需的文件夹名称：

```
tar -xf ../mpfr-4.0.2.tar.xz
mv -v mpfr-4.0.2 mpfr
tar -xf ../gmp-6.1.2.tar.xz
mv -v gmp-6.1.2 gmp
tar -xf ../mpc-1.1.0.tar.gz
mv -v mpc-1.1.0 mpc
```

再次创建独立的编译文件夹：

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

在开始编译 GCC 之前，记住取消所有会覆盖默认优化选项的环境变量。

准备编译 GCC:

```
CC=$LFS_TGT-gcc                                    \
CXX=$LFS_TGT-g++                                   \
AR=$LFS_TGT-ar                                     \
RANLIB=$LFS_TGT-ranlib                             \
../configure                                       \
    --prefix=/tools                                \
    --with-local-prefix=/tools                     \
    --with-native-system-header-dir=/tools/include \
    --enable-languages=c,c++                       \
    --disable-libstdcxx-pch                        \
    --disable-multilib                             \
    --disable-bootstrap                            \
    --disable-libgomp
```

新配置选项的含义:

--enable-languages=c,c++
这个选项确保编译了 C 和 C++ 编译器。

--disable-libstdcxx-pch
不为 libstdc++ 编译预编译的头文件(PCH)。这会花费很多时间，却对我们没有用处。

--disable-bootstrap
对于原生编译的 GCC，默认是做一个「引导」构建。这不仅会编译 GCC 一次，而是会编译很多次。使用初次编译的程序去编译其自身第二次，然后同样地进行第三次。比较第二次和第三次迭代确保其能完美地复制自身。这也能预示编译是正确地。但是，LFS 的构建方法能够提供一个稳定的编译器，避免每次都需要重新引导。

编译软件包:

```
make
```

安装软件包:

```
make install
```

作为画龙点睛，这里创建一个符号链接。很多程序和脚本执行 cc 而不是 gcc 来保证程序的通用性，并且在所有的 Unix 类型的系统上都能用，而非仅局限于安装了 GCC 的 Unix 类型的系统。运行 cc 使得系统管理员不用考虑要安装那种 C 编译器：

```
ln -sv gcc /tools/bin/cc
```

Caution

到了这里，必须停下来确认新工具链的基本功能（编译和链接）都是像预期的那样正常工作。运行下面的命令进行全面的检查：

```
echo 'int main(){}' > dummy.c
cc dummy.c
readelf -l a.out | grep ': /tools'
```

如果一切工作正常的话，这里应该没有错误，最后一个命令的输出形式会是：

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

注意 32 位机器的动态链接是 /tools/lib/ld-linux.so.2。

如果输出不是像上面那样或者根本就没有输出，那么可能某些地方出错了。调查并回溯这些步骤，找出问题所在并改正它。在继续之前必须解决这个问题。首先，使用 gcc 而不是 cc 再次进行全面的检查。如果能运行，就符号链接 /tools/bin/cc 就不见了。像上面介绍的那样新建符号链接。下一步，确认 PATH 是正确的。这能通过运行 echo $PATH 检验，验证 /tools/bin 在列表的前面。如果 PATH 是错误的，这意味着你可能不是以 lfs 用户的身份登录或者在前面 Section#4.4, “设置环境” 中某些地方出现了错误。

一旦一切都顺利，清理测试文件：

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

该软件包的详细信息请参见： Section#6.21.2, “GCC 软件包内容”。

5.11. Tcl-8.6.9

Tcl 软件包包含工具命令语言（Tool Command Language）相关程序。

大致构建用时: 0.9 SBU
所需磁盘空间: 66 MB

5.11.1. 安装 Tcl

此软件包和后面两个包（Expect 和 DejaGNU）用来为 GCC 和 Binutils 还有其他的一些软件包的测试套件提供运行支持。仅仅为了测试目的而安装三个软件包，看上去有点奢侈，虽然因为大部分重要的工具都能正常工作而并不需要去做测试。尽管在本章中并没有执行测试套件（并不做要求），但是在 Chapter#6中都要求执行这些软件包自带的测试套件。

注意，这里的 Tcl 软件包用的是最小化安装的版本，仅仅是为了运行 LFS 测试。需要完整版的软件包，可参考 BLFS 的 Tcl 流程。

配置 Tcl 准备编译：

```
cd unix
./configure --prefix=/tools
```

编译软件包：

```
make
```

现在编译已经完成。之前说过，不强求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Tcl 的话可以用下面的命令：

```
TZ=UTC make test
```

Tcl 测试套件在宿主机某些特定条件下会失败，原因很难推测。不过测试套件失败并不奇怪，也不是什么严重的错误。参数 *TZ=UTC* 设定了时区和相应的世界标准时间（UTC），但是只在测试套件运行期间才有效。这个可以保证时钟测试能正常运行。关于 TZ 环境变量的细节请参阅本手册 第 7 章。

安装软件包：

```
make install
```

让安装的库文件可写，这样之后可以删除调试符号。

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

安装 Tcl 的头文件。后面的软件包 Expect 在编译的时候会用到。

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

现在创建几个必要的软链接：

```
ln -sv tclsh8.6 /tools/bin/tclsh
```

5.11.2. Tcl 软件包内容

安装的程序: tclsh (link to tclsh8.6) and tclsh8.6
安装的库: libtcl8.6.so, libtclstub8.6.a

简要介绍

tclsh8.6	Tcl 命令终端
tclsh	软链接到 tclsh8.6
`libtcl8.6.so`	Tcl 库
`libtclstub8.6.a`	Tcl Stub 库

5.12. Expect-5.45.4

Expect 软件包包含一个实现用脚本和其他交互式程序进行对话的程序。

大致构建用时: 0.1 SBU
所需磁盘空间: 3.9 MB

5.12.1. 安装 Expect

首先，强制 Expect 的 configure 配置脚本使用 `/bin/stty` 替代宿主机系统里可能存在的 `/usr/local/bin/stty` 。这样可以保证我们的测试套件工具在工具链的最后一次构建能够正常。

```
cp -v configure{,.orig}
sed 's:/usr/local/bin:/bin:' configure.orig > configure
```

现在配置 Expect 准备编译：

```
./configure --prefix=/tools       \
            --with-tcl=/tools/lib \
            --with-tclinclude=/tools/include
```

配置脚本参数的含义：

`--with-tcl=/tools/lib`
这个选项可以保证 configure 配置脚本会从临时工具目录里找 Tcl 的安装位置，而不是在宿主机系统中寻找。

`--with-tclinclude=/tools/include`
这个选项会给 Expect 显式地指定 Tcl 内部头文件的位置。通过这个选项可以避免 configure 脚本不能自动发现 Tcl 头文件位置的情况。

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Expect 的话可以用下面的命令：

```
make test
```

请注意 Expect 测试套件已知在某些宿主机特定情况下有过失败的情况，我们还没有完全把握。不过，在这里测试套件运行失败并不奇怪，也不认为是关键问题。

安装软件包：

```
make SCRIPTS="" install
```

make 参数的含义：

`SCRIPTS=""`
这个变量可以避免安装额外的、没有需求的 Expect 脚本。

5.12.2. Expect 软件包内容

安装的程序: expect
安装的库: libexpect-5.45.so

简要介绍

expect	基于脚本和其他交互式程序通信。
`libexpect-5.45.so`	包含一些函数允许 Expect 用作 Tcl 扩展或直接用于 C/C++（不用 Tcl）。

5.13. DejaGNU-1.6.2

DejaGNU 软件包包含了测试其他程序的框架。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 3.2 MB

5.13.1. 安装 DejaGNU

配置 DejaGNU 准备编译：

```
./configure --prefix=/tools
```

编译安装软件包：

```
make install
```

要测试编译结果，执行：

```
make check
```

5.13.2. DejaGNU 软件包内容

安装的程序: runtest

简要介绍

runtest 一个封装脚本用于定位合适的 expect 终端然后执行 DejaGNU。

5.14. M4-1.4.18

M4 软件包包含一个宏处理器。

大致构建用时: 0.2 SBU
所需磁盘空间: 20 MB

5.14.1. 安装 M4

首先，对应 glibc-2.28 的需求做一些修复：

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

配置 M4 准备编译：

```
./configure --prefix=/tools
```

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 M4 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.14.2, “M4 软件包内容”。

5.15. Ncurses-6.1

Ncurses 软件包包含用于不依赖于特定终端的字符屏幕处理的库。

大致构建用时: 0.6 SBU
所需磁盘空间: 41 MB

5.15.1. 安装 Ncurses

在安装之前，须要确保 gawk 在第一次配置时已经找到：

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

配置 Ncurses 准备编译：

```
./configure --prefix=/tools \
            --with-shared   \
            --without-debug \
            --without-ada   \
            --enable-widec  \
            --enable-overwrite
```

配置脚本参数的含义：

`--without-ada`
这个选项会保证 Ncurse 不会编译对宿主机系统里可能存在的 Ada 编译器的支持，而这在我们 chroot 切换环境后就不再可用。

`--enable-overwrite`
这个选项会告诉 Ncurses 安装它的头文件到 `/tools/include` 目录，而不是 `/tools/include/ncurses` 目录，保证其他软件包可以正常找到 Ncurses 的头文件。

`--enable-widec`
这个选项会控制编译宽字符库（比如，`libncursesw.so.6.1`）而不是默认的普通库（比如，`libncurses.so.6.1`）。这些宽字符库在多字节和传统的 8 位环境下使用，而普通库只能用于 8 位环境。宽字符库和普通库的源代码是兼容的，但并不是二进制兼容。

编译软件包：

```
make
```

这个软件包有测试套件，但是只有在安装后才能执行。测试用例在 `test/` 目录里。查看该目录下的 `README` 文件了解更多细节。

安装软件包：

```
make install
ln -s libncursesw.so /tools/lib/libncurses.so
```

关于这个软件包的详细资料请参见： Section#6.24.2, “Ncurses 软件包内容”。

5.16. Bash-5.0

Bash 软件包包含 Bourne-Again Shell。

大致构建用时: 0.4 SBU
所需磁盘空间: 67 MB

5.16.1. 安装 Bash

配置 Bash 准备编译：

```
./configure --prefix=/tools --without-bash-malloc
```

配置脚本参数的含义：

`--without-bash-malloc`
这个选项会禁用 Bash 的内存分配功能（`malloc`），这个功能已知会导致段错误。而禁用这个功能后，Bash 将使用 Glibc 的 `malloc` 函数，这样会更稳定。

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Bash 的话可以用下面的命令：

```
make tests
```

安装软件包：

```
make install
```

为使用 sh 终端的程序创建一个软链接：

```
ln -sv bash /tools/bin/sh
```

关于这个软件包的详细资料请参见： Section#6.34.2, “Bash 软件包内容”。

5.17. Bison-3.3.2

Bison 软件包包含一个语法生成器。

大致构建用时: 0.3 SBU
所需磁盘空间: 37 MB

5.17.1. 安装 Bison

配置 Bison 准备编译：

```
./configure --prefix=/tools
```

编译软件包：

```
make
```

为了测试结果，输入：

```
make check
```

安装软件包：

```
make install
```

该软件包的详细信息参见： Section#6.31.2, “Bison 软件包内容”。

5.18. Bzip2-1.0.6

Bzip2 软件包包含压缩和解压缩的程序。用 bzip2 压缩文本文件能获得比传统的 gzip 更好的压缩比。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 5.5 MB

5.18.1. 安装 Bzip2

Bzip2 软件包里没有 configure 配置脚本。用下面的命令编译和测试：

```
make
```

安装软件包：

```
make PREFIX=/tools install
```

关于这个软件包的详细资料请参见： Section#6.22.2, “Bzip2 软件包内容”。

5.19. Coreutils-8.30

Coreutils 软件包包含用于显示和设置基本系统特性的工具。

大致构建用时: 0.8 SBU
所需磁盘空间: 148 MB

5.19.1. 安装 Coreutils

配置 Coreutils 准备编译：

```
./configure --prefix=/tools --enable-install-program=hostname
```

配置脚本参数的含义：

`--enable-install-program=hostname`
这个选项会允许构建和安装 hostname 程序——默认是不安装的，但是 Perl 测试套件需要它。

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Coreutils 的话可以用下面的命令：

```
make RUN_EXPENSIVE_TESTS=yes check
```

参数 *RUN_EXPENSIVE_TESTS=yes* 会告诉测试套件额外运行对某些系统开销相对大一些（主要是 CPU 运算能力和内存消耗）的测试用例，但是通常对 Linux 来说不是问题。

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.54.2, “Coreutils 软件包内容”。

5.20. Diffutils-3.7

Diffutils 软件包包含显示文件和目录差异的程序。

大致构建用时: 0.2 SBU
所需磁盘空间: 26 MB

5.20.1. 安装 Diffutils

配置 Diffutils 准备编译：

```
./configure --prefix=/tools
```

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Diffutils 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.56.2, “Diffutils 软件包内容”。

5.21. File-5.36

File 软件包包括一个判断给定的某个或某些文件文件类型的工具。

大致构建用时: 0.1 SBU
所需磁盘空间: 18 MB

5.21.1. 安装 File

配置 File 准备编译：

```
./configure --prefix=/tools
```

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 File 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.12.2, “File 软件包内容”。

5.22. Findutils-4.6.0

Findutils 软件包包含查找文件的程序。这些程序提供递归搜索目录树、创建、管理以及搜索数据库（通常比递归式的 find 要快，但如果数据库最近没有更新的话结果不可靠）。

大致构建用时: 0.3 SBU
所需磁盘空间: 36 MB

5.22.1. 安装 Findutils

首先，对应 glibc-2.28 的需求做一些修复：

```
sed -i 's/IO_ftrylockfile/IO_EOF_SEEN/' gl/lib/*.c
sed -i '/unistd/a #include <sys/sysmacros.h>' gl/lib/mountlist.c
echo "#define _IO_IN_BACKUP 0x100" >> gl/lib/stdio-impl.h
```

配置 Findutils 准备编译：

```
./configure --prefix=/tools
```

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Findutils 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.58.2, “Findutils 软件包内容”。

5.23. Gawk-4.2.1

Gawk 软件包包含用于操作文本文件的程序。

大致构建用时: 0.2 SBU
所需磁盘空间: 43 MB

5.23.1. 安装 Gawk

配置 Gawk 准备编译：

```
./configure --prefix=/tools
```

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Gawk 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.57.2, “Gawk 软件包内容”。

5.24. Gettext-0.19.8.1

Gettext 软件包包含用于国际化和本土化的工具。这允许用 NLS（Native Language Support，本地语言支持）编译程序，使得能以用户的本地语言输出信息。

大致构建用时: 0.9 SBU
所需磁盘空间: 173 MB

5.24.1. 安装 Gettext

对于我们这次用到的临时工具集，我们只需要编译安装 Gettext 软件包里的 3 个程序。

配置 Gettext 准备编译：

```
cd gettext-tools
EMACS="no" ./configure --prefix=/tools --disable-shared
```

配置脚本参数的含义：

EMACS="no"
这个选项会禁止配置脚本侦测安装 Emacs Lisp 文件的位置，已知在某些系统中会引起错误。

--disable-shared
这次我们不需要安装任何的 Gettext 动态库，所以不需要编译。

编译软件包：

```
make -C gnulib-lib
make -C intl pluralx.c
make -C src msgfmt
make -C src msgmerge
make -C src xgettext
```

因为只编译了 3 个程序，在不编译 Gettext 的额外支持库的情况下，测试套件是不可能成功运行的。所以在这个阶段建议不要尝试运行测试套件。

安装 msgfmt、msgmerge 和 xgettext 程序：

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

关于这个软件包的详细资料请参见： Section#6.47.2, “Gettext 软件包内容”.

5.25. Grep-3.3

Grep 软件包包含用于在文件中搜索的程序。

大致构建用时: 0.2 SBU
所需磁盘空间: 24 MB

5.25.1. 安装 Grep

配置 Grep 准备编译：

```
./configure --prefix=/tools
```

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Grep 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.33.2, “Grep 软件包内容”。

5.26. Gzip-1.10

Gzip 软件包包含用于压缩和解压文件的程序。

大致构建用时: 0.1 SBU
所需磁盘空间: 10 MB

5.26.1. 安装 Gzip

配置 Gzip 准备编译：

```
./configure --prefix=/tools
```

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Gzip 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.62.2, “Gzip 软件包内容”。

5.27. Make-4.2.1

Make 软件包包含一个用于编译软件的程序。

大致构建用时: 0.1 SBU
所需磁盘空间: 13 MB

5.27.1. 安装 Make

首先，解决一个 glibc-2.27 或更高版本带来的问题：

```
sed -i '211,217 d; 219,229 d; 232 d' glob/glob.c
```

配置 Make 准备编译：

```
./configure --prefix=/tools --without-guile
```

配置脚本参数的含义：

`--without-guile`
这个选项会保证 Make-4.2.1 不会链接宿主系统上可能存在的 Guile 库，而在下一章里通过 chroot 切换环境后便不再有效了。

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Make 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.66.2, “Make 软件包内容”。

5.28. Patch-2.7.6

Patch 软件包包含一个通过打「补丁」创建或修改文件的程序，补丁文件通常由 diff 程序生成。

大致构建用时: 0.2 SBU
所需磁盘空间: 12 MB

5.28.1. 安装 Patch

配置 Patch 准备编译：

```
./configure --prefix=/tools
```

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Patch 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.67.2, “Patch 软件包内容”。

5.29. Perl-5.28.1

Perl 软件包包含实用信息抽取与报告语言。

大致构建用时: 1.6 SBU
所需磁盘空间: 275 MB

5.29.1. 安装 Perl

配置 Perl 准备编译：

```
sh Configure -des -Dprefix=/tools -Dlibs=-lm -Uloclibpth -Ulocincpth
```

配置选项的含义：

-des
这是三个选项地的组合：-d 对所有选项使用默认值；-e 确保完成所有任务；-s 静默，不产生非必要输出。

-Uloclibpth amd -Ulocincpth
这些条目中未定义的变量，会促使配置过程去寻找宿主系统中已经存在的本地安装的组件。

编译软件包：

```
make
```

虽然 Perl 软件包自带测试套件，最好还是等下一章中它被完整安装之后再运行。

这次我们只需要安装一小部分的应用和库。

```
cp -v perl cpan/podlators/scripts/pod2man /tools/bin
mkdir -pv /tools/lib/perl5/5.28.1
cp -Rv lib/* /tools/lib/perl5/5.28.1
```

关于这个软件包的详细资料请参见： Section#6.40.2, “Perl 软件包内容”。

5.30. Python-3.7.2

软件包 Python 3 包含了 Python 的开发环境。对于面向对象编程，书写脚本，构建大型程序的原型，或者开发整个应用程序而言，非常有用。

大致构建用时: 1.5 SBU
所需磁盘空间: 371 MB

5.30.1. 安装 Python

这个软件包首先构建 Python 解释器，然后是一些标准的 Python模块。构建模块的主要脚本是用 Python 编写的，并使用宿主机 `/usr/include` 和 `/usr/lib` 目录的硬编码路径。以此防止他们被使用，输入：

```
sed -i '/def add_multiarch_paths/a \        return' setup.py
```

配置 Python 准备编译：

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

配置选项的含义：

`--without-ensurepip`
该选项用于禁用现阶段好不需要的 Python 安装程序。

编译软件包：

```
make
```

编译完成。测试套件需求 TK 和 X Windows，此时无法运行。

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.51.2, “Python 3 软件包内容.”

5.31. Sed-4.7

Sed 软件包包含一个流编辑器。

大致构建用时: 0.2 SBU
所需磁盘空间: 20 MB

5.31.1. 安装 Sed

配置 Sed 准备编译：

```
./configure --prefix=/tools
```

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Sed 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.28.2, “Sed 软件包内容”。

5.32. Tar-1.31

Tar 软件包包含一个归档程序。

大致构建用时: 0.3 SBU
所需磁盘空间: 38 MB

5.32.1. 安装 Tar

配置 Tar 准备编译：

```
./configure --prefix=/tools
```

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Tar 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.69.2, “Tar 软件包内容”。

5.33. Texinfo-6.5

Texinfo 软件包包含用于读、写以及转换信息页的程序。

大致构建用时: 0.3 SBU
所需磁盘空间: 104 MB

5.33.1. 安装 Texinfo

配置 Texinfo 准备编译：

```
./configure --prefix=/tools
```

> **Note**
>
> 作为配制过程的一部分，有一个测试会指出 TestXS_la-TestXS.lo 有一处错误。这与 LFS 没有关系，可以忽略。

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Texinfo 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.70.2, “Texinfo 软件包内容”。

5.34. Util-linux-2.33.1

Util-linux 软件包包含了各种各样的小工具。

大致构建用时: 1 SBU
所需磁盘空间: 147 MB

5.34.1. 安装 Util-linux

配置 Util-linux 准备编译：

```
./configure --prefix=/tools                \
            --without-python               \
            --disable-makeinstall-chown    \
            --without-systemdsystemunitdir \
            --without-ncurses              \
            PKG_CONFIG=""
```

配置脚本参数的含义：

--without-python
这个选项会禁止使用宿主系统中可能安装了的 Python。这样可以避免构建一些不必要的捆绑应用。

--disable-makeinstall-chown
这个选项会禁止在安装的时候使用 chown 命令。这对我们安装到 /tools 目录没有意义而且可以避免使用 root 用户安装。

--without-ncurses
这个选项会禁止在构建的过程中使用 ncurses 库。在往 /tools 目录安装时没有必要使用 ncurses 库，而且这样做还能避免一些宿主发行版带来的问题。

--without-systemdsystemunitdir
对于使用 systemd 的系统，这个软件包会尝试安装 systemd 特定文件到 /tools 下一个不存在的目录里。这个选项可以避免这个不必要的动作。

PKG_CONFIG=""
设定这个环境变量可以避免增加一些宿主机上存在却不必要的功能。请注意这里设定环境变量的方式和 LFS 其他部分放在命令前面的方式不同。在这里是为了展示一下使用 configure 脚本配置时设定环境变量的另一种方式。

编译软件包：

```
make
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.75.3, “Util-linux 软件包内容”。

5.35. Xz-5.2.4

Xz 软件包包含用于压缩和解压文件的程序。它提供 lzma 和更新的 xz 压缩格式功能。和传统的 gzip 或 bzip2 命令相比，用 xz 压缩文本文件能获得更好的压缩率。

大致构建用时: 0.2 SBU
所需磁盘空间: 18 MB

5.35.1. 安装 Xz

配置 Xz 准备编译：

```
./configure --prefix=/tools
```

编译软件包：

```
make
```

现在编译已经完成。之前说过，不要求为本章中所构建的临时工具运行测试套件。不过你仍然要测试 Xz 的话可以用下面的命令：

```
make check
```

安装软件包：

```
make install
```

关于这个软件包的详细资料请参见： Section#6.45.2, “Xz 软件包内容”。

5.36. 清理无用内容

本小节里的步骤是可选的，但如果你的 LFS 分区容量比较小，知道有些不必要的内容可以被删除也是挺好的。目前编译好的可执行文件和库大概会有 70MB 左右不需要的调试符号。可以通过下面的命令移除这些符号：

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

这两个命令会跳过一些文件，并提示不可识别的文件格式。大多数是脚本文件而不是二进制文件。同样还可以用宿主系统里的 strip 命令为 /tools 目录下的 strip 二进制文件清理无用内容。

注意不要对库文件使用 *--strip-unneeded* 选项。静态库会被损坏导致整个工具链将会需要全部重新编译。

更节省更多空间，还可以删除帮助文档：

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

删除不需要的文件：

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

这个时候，你应该在 $LFS 分区中为下个阶段编译安装 Glibc 和 GCC 预留至少 3GB 剩余空间。如果你可以编译安装 Glibc，那其他的就不会有问题了。

5.37. 改变属主

> **Note**
>
> 本书余下部分的命令都必须以 root 用户身份执行而不再是 lfs 用户。另外，再次确认下 $LFS 变量在 root 用户环境下也有定义。

当前，$LFS/tools 目录属于 lfs 用户，这是一个只存在于宿主系统上的帐号。如果继续保持 $LFS/tools 目录的现状，其中的文件将属于一个没有相关联帐号的用户 ID。这很危险，因为随后创建的用户有可能会分配到相同的用户 ID，从而变成 $LFS/tools 目录及其中所有文件的属主，以致留下恶意操作这些文件的可能。

为了解决这个问题，你可以在随后新的 lfs 系统里创建 /etc/passwd 文件时增加一个 lfs 用户，并注意给它分配和宿主系统里相同的用户和组 ID。不过更好的方式是，通过下面的命令将 $LFS/tools 目录的属主改为 root 用户：

```
chown -R root:root $LFS/tools
```

尽管 $LFS/tools 目录可以在 LFS 系统构建完成后删除，但仍然可以保留下来用于 构建额外的相同版本 LFS 系统。备份 $LFS/tools 目录到底有多少好处取决于你个人。

> **Caution**
>
> 如果你想保留临时工具用来构建新的 LFS 系统，现在就要备份好。本书随后第六章中的指令将对当前的工具做些调整，导致在构建新系统时会失效。

Part III. 构建 LFS 系统

Chapter 6. 安装基本的系统软件

6.1. 简介

在本章中，我们会进入构建环境开始认真地构建 LFS 系统了。我们将 chroot 到之前准备好的临时迷你 Linux 系统，做一些最后的准备工作，然后就开始安装软件包。

安装软件很简单。尽管很多时候安装指令能更短而且更具通用性，但我们还是选择为每个软件包都提供完整的指令，以减小引起错误的可能性。了解 Linux 系统如何工作的关键就是知道每个软件包的作用以及为什么你（或系统）需要它。

我们不建议在编译时使用优化。这虽然可以让程序运行得快那么一点点，但是却也有可能增加编译难度，以及在运行时出问题。如果在打开优化后编译失败，请试一下关闭优化编译看看。就算打开优化通过了编译，考虑到源代码和编译工具之间的复杂交互，仍然存在编译不正确的风险。还有需要注意 `-march` 和 `-mtune` 选项除了本书指定的值都未经测试。这有可能导致工具链软件包（Binutils、GCC 和 Glibc）发生问题。对比使用编译优化带来的好处与风险，这样做经常是得不偿失。第一次构建 LFS 系统还是推荐不要使用自定义优化。这样构建出来的系统一样会运行得很快，于此同时还很稳定。

本章里安装软件包的顺序需要严格遵守，这是为了保证不会有程序意外地依赖与 `/tools` 路径的硬链相关的目录。同样的理由，不要同时编译不同的软件包。并行地编译也许能节省一点时间（特别是在双 CPU 电脑上），但是它可能会导致程序里存在包含到 `/tools` 目录的硬链接，这样的话在这个目录移除后程序就不能正常工作了。

在安装指令之前，每个页面都提供了关于软件包的信息，包括其中所包含内容的精确描述，构建需求的大致时间，以及在过程中需求磁盘空间的大小。在安装指令之后，是一个该软件包即将安装的程序和库（及概要说明）的列表。

> **Note**
>
> 第六章里软件包的 SBU 数值和所需磁盘空间包含了可能存在的测试套件数据。

6.1.1. 关于库

总的来说，LFS 的编辑们并不推荐构建和安装静态库。许多静态库的初衷已经赶不上现在的 Linux 系统了。而且将静态库链接到程序还有不好之处。假设库更新需要移除一个安全问题，所有使用该静态库的程序都需要重新链接到新的库。由于静态库并不会总那么明显，有哪些相关的程序（以及需要链接的程序）很可能都不知道。

第六章的程序，我们移除或禁止了大部分静态库的安装。通常通过在 configure 命令中使用 `--disable-static` 项，便可以做到。有些情况下，可能用到其他代替的办法。当然也有少数情况，特别是 glibc 和 gcc 使用的静态库在软件包的构建过程中是必不可少的。

更多关于库的讨论，请参考 BLFS 中 库：静态还是共享？ 章节。

6.2. 准备虚拟内核文件系统

内核会挂载几个文件系统用于自己和用户空间程序交换信息。这些文件系统是虚拟的，并不占用实际磁盘空间，它们的内容会放在内存里。

开始先创建将用来挂载文件系统的目录：

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

6.2.1. 创建初始设备节点

在内核引导系统的时候，它依赖于几个设备节点，特别是 `console` 和 `null` 两个设备。设备点必须创建在硬盘上以保证在 udevd 启动前是可用的，特别是在使用 *`init=/bin/bash`* 启动 Linux 时。运行以下命令创建设备节点：

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

6.2.2. 挂载和激活 /dev

通常激活 /dev 目录下设备的方式是在 /dev 目录挂载一个虚拟文件系统（比如 tmpfs），然后允许在检测到设备或打开设备时在这个虚拟文件系统里动态创建设备节点。这个通常是在启动过程中由 udev 完成。由于我们的新系统还没有 udev，也没有被引导，有必要手动挂载和激活 /dev 这可以通过绑定挂载宿主机系统的 /dev 目录来实现。绑定挂载是一种特殊的挂载模式，它允许在另外的位置创建某个目录或挂载点的镜像。运行下面的命令来实现：

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

6.2.3. 挂载虚拟文件系统

现在挂载剩下的虚拟内核文件系统：

```
mount -vt devpts devpts $LFS/dev/pts -o gid=5,mode=620
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
mount -vt tmpfs tmpfs $LFS/run
```

挂载选项 devpts 的含义：

gid=5
以确保所有由 devpts 创建的设备节点属于 GID 5 之下。这个 ID 日后将用于 tty 组。我们使用 GID 来代替组名，原因是宿主系统可能为 tty 组使用其他的 ID。

mode=0620
以确保所有 devpts 创建的设备节点的属性为 0620 （属主用户可读写，组成员可写）。与上一个选项同时使用，可以保证 devpts 所创建的设备节点能满足 grantpt() 函数的要求，这就意味着不需要 Glibc （默认不安装的）帮助程序 pt_chown 了。

在某些宿主机系统里，/dev/shm 是一个指向 /run/shm 的软链接。这个 /run 下的 tmpfs 文件系统已经在之前挂载了，所以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
```

6.3. 软件包管理

软件包管理经常被请求加入到 LFS 手册中。软件包管理器可以追踪安装的文件，方便软件包的移除和升级。不仅是二进制执行文件和库文件，包管理器还会处理配置文件的安装。在你想太多之前，答案是不——本节不讨论也不安利任何特定的软件包管理器，只是总结了一下关于软件包管理的常用技术和工作原理。对你而言最完美的软件包管理器可能就在这些技术之中，也可能由这些技术的其中几个组合而成。本节还简要的提了一些在升级软件包时可能遇到的问题。

为什么 LFS 或 BLFS 手册里不采用任何软件包管理器的一些原因：

- 使用软件包管理偏离了本手册的主要目标——教大家 Linux 系统是如何构建出来的。
- 存在很多软件包管理的解决方案，每一个都有自己的长处和缺点。很难选择一种适合所有人的方式。

关于软件包管理有很多资料，可以访问 Hints Project 看看是否可以解决你的需求。

6.3.1. 升级问题

软件包管理器可以在软件新版本发布后轻松升级。一般来说 LFS 和 BLFS 手册里的指令是可以用来升级版本。下面是一些在你准备升级软件包时需要注意的事情，特别是运行中系统需更加注意。

- 如果需要升级 Glibc 到新版本（比如，从 glibc-2.19 升级到 glibc-2.20），重新构建整个 LFS 会比较安全。虽然你 也许 能够按依赖关系重新编译所有的软件包，不过我们不建议这样做。
- 如果某个包含的动态库的软件包升级了，而且库名字有所改变，那么所有动态链接到这个库的软件包都需要重新链接新的库。（请注意软件包版本和库名字并不存在相关性）。举个例子，说软件包 foo-1.2.3 安装了一个叫 libfoo.so.1 的动态库。然后你将软件包升级到了新版本 foo-1.2.4 安装的动态库叫 libfoo.so.2 了。在这种情况下，所有动态链接到 libfoo.so.1 的软件包都需要重新编译链接到 libfoo.so.2。注意在所有依赖软件包重新编译完成之前，请勿删除之前的库文件。

6.3.2. 软件包管理技术

下面介绍一些常见的软件包管理技巧。在决定用哪种包管理方式之前，先研究一下各种不同的技术，特别是某些方案的不足之处。

6.3.2.1. 所有一切都在我脑袋里！

是的，这也算一种软件包管理技术。有些人觉得不需要管理软件包，是因为他们非常熟悉软件包，知道每个包都安装了哪些文件。也有些用户不需要管理软件包，是因为他们会在某个软件包有更改后重建整个系统。

6.3.2.2. 在独立目录里安装

这是一种简单的软件包管理方式，不需要其他额外的软件来管理软件的安装。每一个软件包都被装到一个独立的目录里。例如，软件包 foo-1.1 安装到目录 `/usr/pkg/foo-1.1` 中并创建一个软链接 `/usr/pkg/foo` 指向 `/usr/pkg/foo-1.1`。在安装新版本 foo-1.2 的时候，它会被装到目录 `/usr/pkg/foo-1.2` 中，然后用指向新版本的软链替代之前的软链接。

类似 `PATH`、`LD_LIBRARY_PATH`、`MANPATH`、`INFOPATH` 和 `CPPFLAGS` 之类的环境变量变量需要包含 `/usr/pkg/foo` 目录。在管理大量软件包时，这种方式就不可行了。

6.3.2.3. 软链接方式软件包管理

这是前一种软件包管理技术的变种。每个软件包的安装方式都和之前的方式类似。但不是建立目录的软链接，而是把每个文件都链接到 `/usr` 目录结构里。这样就不需要扩展环境变量了。通过自动创建这些可由用户自行创建的软链，许多软件包管理器就采用了这种方式进行管理的。其中比较流行的有 Stow、Epkg、Graft 和 Depot。

这种安装方式需要伪装，这样软件包会认为自己被装到了 `/usr` 目录下，而实际上它被装到了 `/usr/pkg` 目录下。在这种方式下，安装并不是一件琐碎的小事。例如，假如你准备安装一个软件包 libfoo-1.1。下面的指令可能不会正确地安装：

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

安装本身倒是没有问题，但是可能一些依赖包不会像你期望的那样链接到 libfoo 库。如果你要编译一个链接 libfoo 的软件，你可能会注意到它实际上链接到的是 `/usr/pkg/libfoo/1.1/lib/libfoo.so.1` 而不是你所期望的 `/usr/lib/libfoo.so.1`。正确的方式是使用 `DESTDIR` 策略来伪装软件包的安装过程。这种方式需要像下面这样操作：

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

大多数软件包支持这种方式，但也有一些例外。对于不兼容的软件包，你可能需要自己手动安装，或许你会发现将这些有问题的包安装到 `/opt` 目录下会更简单些。

6.3.2.4. 基于时间戳

在这种方式里，在安装之前会创建一个时间戳文件。在安装之后，用一行简单的 find 命令加上合适的参数就可以生成在时间戳文件创建之后所安装的所有文件列表。有一个采用这种方式的包管理器叫做 install-log。

这种方式的优点是非常简单，但是它有两个缺陷。比如，在安装过程中，所安装文件采用的是其它时间戳而不是当前时间，那这些文件将不能被软件包管理器跟踪到。还有，这种方式只能在一次安装一个软件包的情况下使用。如果在不同的终端里同时安装两个不同的软件包，此时的安装日志就不可靠了。

6.3.2.5. 追踪安装脚本

在这种方式里，安装脚本所使用的命令都会被记录下来。有两种技术，一种是：

设定环境变量 `LD_PRELOAD` 指向一个在安装前预加载的库。在安装过程中，这个库会追踪软件包安装脚本里所包含的各种执行文件比如 cp、install、mv，以及追踪会修改文件系统的系统调用。要让这种方式有效的话，所有的执行文件需要动态链接到没有 suid 或 sgid 标志位的库。预加载这个库可能会引起安装过程中一些意外的副作用。因此，建议做一些测试以保证软件包管理器不会造成破坏并且记录了所有适当的文件。

第二种技术是使用 strace 命令，它会记录下安装脚本执行过程中所有的系统调用。

6.3.2.6. 创建软件包存档

在这种方式里，像之前的软链接软件包管理方式里所描述的那样，软件包被伪装安装到一个独立的目录树里。在安装完成后，会将已安装文件打包成一个软件包存档。然后这个存档会用来在本地机器或其他机器上安装软件包。

这种方式为商业发行版中的大多数包管理器所采用。例子有 RPM（顺带提一下，这也是 Linux 标准规范 中指定的包管理器），pkg-utils，Debian 的 apt，和 Gentoo 的 Portage 系统。如何在 LFS 系统里采用这种包管理方式的简单描述，请参看 http://www.linuxfromscratch.org/hints/downloads/files/fakeroot.txt.

创建带有依赖关系的软件包存档非常复杂，已经超出 LFS 手册范围了。

Slackware 使用一个基于 tar 的系统来创建软件包存档。这套系统不像那些更复杂的包管理器，有意地不处理包依赖关系。关于 Slackware 包管理器的详细信息，请参看 http://www.slackbook.org/html/package-management.html。

6.3.2.7. 基于用户的软件包管理

这种方式，是 LFS 特有的，由 Matthias Benkmann 所设计，可以在 Hints Project 中能找到。在这种方式里，每个软件包都由一个单独的用户安装到标准的位置。文件属于某个软件包可以通过检查用户 ID 轻松识别出来。关于这种方式的利弊比较复杂，就不再本节中赘述了。详细的信息请参看 http://www.linuxfromscratch.org/hints/downloads/files/more_control_and_pkg_man.txt。

6.3.3. 在多个系统上部署 LFS

LFS 系统的一个优点是没有那种需要依赖其在磁盘系统中的位置的文件。克隆一份 LFS 到和宿主机器相似配置的机器上，简单到只要对包含根目录的 LFS 分区（对于一个基本的 LFS 构建不压缩的话大概有 250MB）使用 tar 命令打包，然后通过网络传输或光盘拷贝到新机器上展开即可。在这之后，需要调整一些配置文件。需要更新的配置文件包括： `/etc/hosts`, `/etc/fstab`, `/etc/passwd`, `/etc/group`, `/etc/shadow`, 和 `/etc/ld.so.conf`。

根据系统硬件和原始内核配置文件的差异，可能还需要重新编译一下内核。

> **Note**
>
> 据报告，当这样的复制发生在两个相近却又不完全相同的架构时会发生问题。例如，Intel 系统的指令集就和 AMD 处理器的不同，还有一些较新版的处理器可能会有一些在较早版本中不能支持的指令。

最后，通过 Section#8.4, “使用 GRUB 设置启动过程”中介绍的方法来使新系统可以引导。

6.4. 进入 Chroot 环境

现在可以切换到 chroot 环境开始构建和安装最终的 LFS 系统了。以 `root` 用户运行下面的命令进入此环境，从现在开始，就只剩下准备的那些临时工具了：

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

给 env 命令传递 *`-i`* 选项会清除这个 chroot 切换进去的环境中的所有变量。随后，只需重新设定 `HOME`、`TERM`、`PS1` 、和 `PATH` 变量。*`TERM=$TERM`* 将会把 `TERM` 设定成 chroot 外环境相同的值。许多程序需要这个变量才能正常工作，比如 vim 和 less。如果还需设定其他变量，如 `CFLAGS` 或 `CXXFLAGS`，正好在这一起设置了。

在这之后，`LFS` 变量就不再需要了，因为后面所有工作都将被限定在 LFS 文件系统中。因为我们已经告诉 Bash 终端 `$LFS` 就是当前的根目录（`/`）目录。

注意要将 `/tools/bin` 放在 `PATH` 变量的最后。意思是在每个软件的最后版本编译安装好后就不再使用临时工具了。这还需要让 shell 不要「记住」每个可执行文件的位置——这样的话，还要给 bash 加上 *`+h`* 选项来关闭其哈希功能。

注意一下 bash 的提示符是 `I have no name!` 这是正常的，因为这个时候 `/etc/passwd` 文件还没有被创建。

> **Note**
>
> 非常重要，从本章开始，后续章节中的命令都要在 chroot 环境下运行。如果因为某种原因（比如说重启）离开了这个环境，请保证要按照 Section#6.2.2, “挂载和激活 /dev”和 Section#6.2.3, “挂载虚拟文件系统” 中所说的那样挂载虚拟内核文件系统，并在继续构建之前重新运行 chroot 进入环境。

6.5. 创建目录

现在准备创建 LFS 文件系统里的一些目录结构。使用下面的命令创建一个标准的目录树：

```
mkdir -pv /{bin,boot,etc/{opt,sysconfig},home,lib/firmware,mnt,opt}
mkdir -pv /{media/{floppy,cdrom},sbin,srv,var}
install -dv -m 0750 /root
install -dv -m 1777 /tmp /var/tmp
mkdir -pv /usr/{,local/}{bin,include,lib,sbin,src}
mkdir -pv /usr/{,local/}share/{color,dict,doc,info,locale,man}
mkdir -v  /usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -v  /usr/libexec
mkdir -pv /usr/{,local/}share/man/man{1..8}

case $(uname -m) in
 x86_64) mkdir -v /lib64 ;;
esac

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}
```

一般目录默认会按 755 的权限创建，但是这并不适用于所有的目录。在上面的命令里，有两个改动——一个是 `root` 的 home 目录，另一个是存放临时文件的目录。

第一个模式改动能保证不是所有人都能进入 `/root` 目录——同样一般用户也需要为他/她的 home 目录设置这样的模式。第二个模式改动能保证所有用户对目录 `/tmp` 和 `/var/tmp` 都是可写的，但又不能移除其他用户的文件。后面的这个限制是由所谓的「粘滞位」实现的，即位掩码 1777 中的最高位（1）。

6.5.1. 关于 FHS 兼容性

这个目录树是基于文件系统目录结构标准（FHS）（参看 https://wiki.linuxfoundation.org/en/FHS）。FHS 标准还规定了要有 `/usr/local/games` 和 `/usr/share/games` 目录。我们只创建了我们需要的目录。然而，如果你更喜欢严格遵守 FHS 标准，创建这些目录也无妨。

6.6. 创建必要的文件和符号链接

有些程序里会使用写死的路径调用其它暂时还未安装的程序。为了满足这种类型程序的需要，我们将创建一些符号链接，在完成本章内容后这些软件会安装好，并替代之前的符号链接：

```
ln -sv /tools/bin/{bash,cat,chmod,dd,echo,ln,mkdir,pwd,rm,stty,touch} /bin
ln -sv /tools/bin/{env,install,perl,printf}         /usr/bin
ln -sv /tools/lib/libgcc_s.so{,.1}                  /usr/lib
ln -sv /tools/lib/libstdc++.{a,so{,.6}}             /usr/lib

install -vdm755 /usr/lib/pkgconfig

ln -sv bash /bin/sh
```

每个链接的目的：

/bin/bash
: 许多 bash 脚本指定了 `/bin/bash`。

/bin/cat
: 这个路径在 Glibc 的配置脚本里写死了。

/bin/dd
到 dd 的路径将写死在/usr/bin/libtool 实用工具中。

/bin/echo
这个是为了满足 Glibc 测试套件里的一个测试用例，它会检测 /bin/echo。

/usr/bin/env
这个路径名硬编码到许多软件包的构建过程中。

/usr/bin/install
到 install 的路径将写死在 /usr/lib/bash/Makefile.inc 文件中。

/bin/ln
到 ln 的路径将写死在 /usr/lib/perl5/5.28.1/<target-triplet>/Config_heavy.pl 文件中。

/bin/pwd
某些 configure 脚本，特别是 Glibc 的，写死了这个路径。

/bin/rm
到 rm 的路径将写死在 /usr/lib/perl5/5.28.1/<target-triplet>/Config_heavy.pl 文件中。

/bin/stty
这个路径在 Expect 软件中写死了，所以在 Binutils 和 GCC 测试套件中会需要它。

/usr/bin/perl
许多 Perl 脚本写死了这个路径调用 perl 执行程序。

/usr/lib/libgcc_s.so{,.1}
Glibc 需要这个让 pthreads 库正常工作。

/usr/lib/libstdc++{,.6}
在 Glibc 的一些测试套件和对于 GMP 的 C++ 支持中会需要。

/bin/sh
许多 shell 脚本写死了位置 /bin/sh。

由于历史原因，Linux 在文件 /etc/mtab 中维护一个已挂载文件系统的列表。而现代内核改为在内部维护这个列表，并通过 /proc 文件系统输出给用户。为了满足一些依赖 /etc/mtab 文件的应用程序，我们要创建下面的符号链接：

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

为了让 root 用户能正常登录，而且「root」的名字能被正常识别，必须在文件 /etc/passwd 和 /etc/group 中写入相应的内容。

运行下面的命令创建 /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
```

root 用户的实际密码（这里的「x」只是占位符）将在后面创建。

运行下面的命令创建 /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
```

这里创建的用户组没有参照任何标准——它们一部分是为了满足本章中配置 udev 的需要，还有一部分来自一些现存 Linux 发行版的通用设定。另外，某些测试套件也依赖特定用户或组。而 Linux 标准规范 （LSB，参见 http://www.linuxbase.org）只要求以组 ID（GID）为 0 创建用户组 root 以及以 GID 为 1 创建用户组 bin。系统管理员可以自由分配其它所有用户组名字和 GID，因为优秀的程序不会依赖 GID 数字，而是使用组名。

为了移除「I have no name!」的提示符，可以打开一个新 shell。由于完整的 Glibc 已经在 第 5 章 里装好了，而且已经创建好了 /etc/passwd 和 /etc/group 文件，用户名和组名就可以正常解析了：

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

注意这里使用了 *+h* 参数。这样会告诉 bash 不要使用它内建的路径哈希功能。而不加这个参数的话，bash 将会记住曾经执行过程序的路径。为了在新编译安装好程序后就能马上使用，参数 *+h* 将在本章中一直使用。

程序 login、agetty 和 init（还有一些其它的）会使用一些日志文件来记录信息，比如谁在什么时候登录了系统。不过，在日志文件不存在的时候这些程序一般不会写入。下面初始化一下日志文件并加上合适的权限：

```
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 会记录所有的登录和登出动作。文件 /var/log/lastlog 会记录每个用户的最后一次登录时间。文件 /var/log/faillog 会记录失败的登录尝试。文件 /var/log/btmp 会记录非法的登录尝试。

Note

文件 /run/utmp 会记录当前已登录的用户。这个文件会在启动脚本中动态创建。

6.7. Linux-4.20.12 API 头文件

Linux API 头文件（在 linux-4.20.12.tar.xz 里）会将内核 API 导出给 Glibc 使用。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 941 MB

6.7.1. Linux API 头文件的安装

Linux 内核需要提供一个应用编程接口（API）供系统的 C 库（LFS 中的 Glibc）调用。这通过整理 Linux 内核源码包中的多个 C 头文件来完成。

确保在之前的动作里没有留下旧文件和依赖关系：

```
make mrproper
```

现在要从源代码里解压出用户需要的内核头文件。因为解压过程会删除目标目录下所有文件，所以我们会先输出到一个本地中间目录后再拷贝到需要的地方。而且里面还有一些隐藏文件是给内核开发人员用的，而 LFS 不需要，所以会将它们从中间目录里删除。

```
make INSTALL_HDR_PATH=dest headers_install
find dest/include \( -name .install -o -name ..install.cmd \) -delete
cp -rv dest/include/* /usr/include
```

6.7.2. Linux API 头文件内容

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, 和 /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, 和 /usr/include/xen

简要介绍

`/usr/include/asm/*.h`	Linux API ASM 头文件
`/usr/include/asm-generic/*.h`	Linux API ASM 通用头文件
`/usr/include/drm/*.h`	Linux API DRM 头文件
`/usr/include/linux/*.h`	Linux API Linux 头文件
`/usr/include/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

6.8. Man-pages-4.16

Man-pages 软件包里包含了超过 2,200 份 man 手册页面。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 28 MB

6.8.1. 安装 Man-pages

运行下面的命令安装 Man-pages：

```
make install
```

6.8.2. Man-pages 内容

安装的文件: 各种 man 手册页面

简要介绍

`man pages` 描述 C 编程语言函数，重要的设备文件，以及主要的配置文件

6.9. Glibc-2.29

Glibc 软件包包含了主要的 C 函数库。这个库提供了分配内存、搜索目录、打开关闭文件、读写文件、操作字符串、模式匹配、基础算法等基本程序。

大致构建用时: 22 SBU
所需磁盘空间: 3.2 GB

6.9.1. 安装 Glibc

> **Note**
>
> Glibc 构建系统是独立的，并且会完美地安装，即使编译器指定的文件和链接器仍然指向 `/tools`。在 Glibc 安装之前，不能调整指定的文件和链接器，因为 Glibc 的 autoconf 测试会给出错误结果并阻碍目标实现干净的构建。

有些 Glibc 程序会用到和 FHS 不兼容的 `/var/db` 目录来存储它们的运行时数据。打上如下的补丁让这些程序在 FHS 兼容的位置存储它们的运行时数据。

```
patch -Np1 -i ../glibc-2.29-fhs-1.patch
```

首先创建一个兼容性符号链接，以避免在最终的 glibc 中存在 /tool 中的引用：

```
ln -sfv /tools/lib/gcc /usr/lib
```

决定 GCC 的 include 目录，并顺应 LSB 规范创建一个软链接。此外，x86_64 情况下，为了使动态加载器正确运作，再创建一个兼容性的软链接：

```
case $(uname -m) in
    i?86)    GCC_INCDIR=/usr/lib/gcc/$(uname -m)-pc-linux-gnu/8.2.0/include
             ln -sfv ld-linux.so.2 /lib/ld-lsb.so.3
    ;;
    x86_64) GCC_INCDIR=/usr/lib/gcc/x86_64-pc-linux-gnu/8.2.0/include
             ln -sfv ../lib/ld-linux-x86-64.so.2 /lib64
             ln -sfv ../lib/ld-linux-x86-64.so.2 /lib64/ld-lsb-x86-64.so.3
    ;;
esac
```

删除一个以前的构建尝试可能遗留下来的文件：

```
rm -f /usr/include/limits.h
```

Glibc 文档里建议在 Glibc 特定编译目录下编译：

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

配置 Glibc 准备编译：

```
CC="gcc -isystem $GCC_INCDIR -isystem /usr/include" \
../configure --prefix=/usr                          \
             --disable-werror                       \
             --enable-kernel=3.2                    \
             --enable-stack-protector=strong        \
             libc_cv_slibdir=/lib
unset GCC_INCDIR
```

选项和新参数的含义：

`CC="gcc -isystem $GCC_INCDIR -isystem /usr/include"`
设置 gcc 和系统的 include 目录以避免在调试符号中引入无效路径。

`--disable-werror`
选项像禁用了传递给 GCC 的 -Werror 选项。为了测试套件，此项必不可少。

`--enable-stack-protector=strong`
该选项通过添加额外代码检查缓冲区溢出（例如，堆栈粉碎攻击）来增强系统安全性。

libc_cv_slibdir=/lib
该变量为所有系统设置正确的库。我们不希望使用 lib64 。

编译软件包：

```
make
```

> **Important**
>
> 在本小节里，运行 Glibc 的测试套件是很关键的。在任何情况下都不要跳过这个测试。

通常会有一些测试不能通过。若是下面列出的这些失败，通常就可以安心的无视了。

```
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
```

> **Note**
>
> 在 chroot 环境的这个阶段需要上面的符号链接以运行测试。在接下来的安装中，这些链接将会被覆盖。

```
make check
```

你可能会看到一些失败的测试项。Glibc 的测试套件对宿主系统有一定的依赖。下面是一些 LFS 版本中最常见的问题：

- misc/tst-ttyname 已知会在 LFS 的 chroot 环境中失败。
- inet/tst-idna_name_classify 已知会在 LFS 的 chroot 环境中失败。
- posix/tst-getaddrinfo4 和 posix/tst-getaddrinfo5 可能在某些架构上失败。
- nss/tst-nss-files-hosts-multi 由于未知的原因，测试可能失败。
- rt/tst-cputimer{1,2,3} 测试取决于宿主机的系统内核。众所周知，内核版本 4.14.91–4.14.96、4.19.13–4.19.18 和 4.20.0–4.20.5 是会导致这些测试失败的。
- 如果你系统的 CPU 不是相对较新的 Intel 或 AMD 处理器，数学运算测试有时候会失败。

虽然这只是无关紧要的消息，在安装 Glibc 时会报告找不到 `/etc/ld.so.conf` 文件。下面的方式可以避免这个警告：

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

修复生成的 Makefile 以跳过一个没有必要的完整性检查，该检查在部分 LFS 环境中会失败：

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

安装软件包：

```
make install
```

为 nscd 安装配置文件并创建运行时目录：

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

为 nscd 安装系统支持文件：

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

接着，安装可以使系统响应不同语言的地域设置。没有一个地域是必需的，但是如果缺少了其中的某些，可能会导致在未来的软件包测试套件中，跳过了重要的测试项目。

单独的语言环境可以用 localedef 程序安装。例如，下面第一个 localedef 命令将 /usr/share/i18n/locales/cs_CZ 字符表定义组合在一起，并将结果附加到 /usr/share/i18n/charmaps/UTF-8.gz 文件末尾。下面的命令将安装能完美覆盖测试所需语言环境的最小集合：

```
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，安装适合你自己国家/地区、语言和字符集的语言环境。

或者，也可以一次性安装在 glibc-2.29/localedata/SUPPORTED 文件里列出的所有语言环境（包括以上列出的所有语言环境以及其它更多），执行下面这个非常耗时的命令：

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

你需要的语言环境几乎不太可能没有列在 glibc-2.29/localedata/SUPPORTED 文件中，但如果真的没有可以使用 localedef 命令创建和安装。

> **Note**
>
> 当前，Glibc 在解决国际化域名时使用 libidn2。此为运行时依赖。如果需要此功能，可以参考 BLFS libidn2 页 相关的安装指令。

6.9.2. 配置 Glibc

6.9.2.1. 添加 nsswitch.conf

由于 Glibc 的默认状态在网络环境下工作的并不好，所以需要创建 /etc/nsswitch.conf 文件。

创建新的 /etc/nsswitch.conf 通过以下命令：

```
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
```

6.9.2.2. 添加时区数据

通过以下命令安装并启动时区数据：

```
tar -xf ../../tzdata2018i.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 ...
: 这会创建没有时间补偿的 posix 时区数据。一般将它们同时放在 zoneinfo 和 zoneinfo/posix。另外需要将 POSIX 时区数据放到 zoneinfo 目录下，否则很多测试套件会报错。在嵌入式平台，如果存储空间紧张而且你也不准备更新时区，也可以不用 posix 目录从而节省 1.9MB，但是一些应用程序或测试套件也许会出错。

zic -L leapseconds ...
: 这会创建包含时间补偿的 right 时区数据。在嵌入式平台，空间比较紧张而且你也不打算更新时区或者不需要准确时间，你可以忽略 right 目录从而节省 1.9MB。

zic ... -p ...
: 这会创建 posixrules 文件。我们使用纽约是因为 POSIX 要求夏令时规则与 US 标准一致。

一种确定本地时区的方式是运行下面的脚本：

```
tzselect
```

在询问了几个关于位置的问题后，脚本会输出所在时区的名字（比如 America/Edmonton)）。在 /usr/share/zoneinfo 文件中也有其它一些可用时区，比如 Canada/Eastern 或 EST5EDT，这些时区并没有被脚本列出来但也是可以使用的。

然后运行下面的命令创建 /etc/localtime 文件：

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

将命令中的 <*xxx*> 替换成你所在实际时区的名字（比如 Canada/Eastern）。

6.9.2.3. 配置动态库加载器

默认情况下，动态库加载器（/lib/ld-linux.so.2）会搜索目录 /lib 和 /usr/lib 查找程序运行时所需的动态库文件。如果库文件不在 /lib 和 /usr/lib 目录下，需要把它所在目录加到 /etc/ld.so.conf 文件里，保证动态库加载器能找到这些库。通常有两个目录包含额外的动态库，/usr/local/lib 和 /opt/lib，把这两个目录加到动态库加载器的搜索路径中。

运行下面的命令创建一个新文件 /etc/ld.so.conf

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

EOF
```

如果需要的话，动态库加载器也可以查找目录并包含里面配置文件的内容。通常在这个包含目录下的文件只有一行字指向库目录。运行下面的命令增加这个功能：

```
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
```

6.9.3. Glibc 软件包内容

安装的程序:	catchsegv, gencat, getconf, getent, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, makedb, mtrace, nscd, pldd, sln, sotruss, sprof, tzselect, xtrace, zdump, 和 zic
安装的库:	ld-2.29.so, libBrokenLocale.{a,so}, libSegFault.so, libanl.{a,so}, libc.{a,so}, libc_nonshared.a, libcidn.so, libcrypt.{a,so}, libdl.{a,so}, libg.a, libieee.a, libm.{a,so}, libmcheck.a, libmemusage.so, libnsl.{a,so}, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libnss_nis.so, libnss_nisplus.so, libpthread.{a,so}, libpthread_nonshared.a, libresolv.{a,so}, librpcsvc.a, librt.{a,so}, libthread_db.so, 和 libutil.{a,so}
安装的库:	/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/rpcsvc, /usr/include/sys, /usr/lib/audit, /usr/lib/gconv, /usr/lib/locale, /usr/libexec/getconf, /usr/share/i18n, /usr/share/zoneinfo, /var/cache/nscd, 和 /var/lib/nss_db

简要介绍

catchsegv	可以在程序因为段错误终止的时候创建栈调用历史
gencat	生成消息条目
getconf	显示文件系统相关的系统配置变量的值
getent	获取系统数据库的内容
iconv	字符集转换
iconvconfig	创建 iconv 快速加载模块配置文件
ldconfig	配置动态链接器的运行时环境
ldd	报告某个程序或动态库所依赖的动态库
lddlibc4	协助 ldd 处理某些目标文件
locale	输出当前语言环境的大量信息
localedef	编译语言环境规格
makedb	根据输入的文本创建简单数据库
mtrace	读取并解析内存跟踪文件，然后用方便人阅读的格式显示一个摘要
nscd	一个后台服务程序，提供最常用名字服务请求的缓存
pldd	列出运行中进程正在使用的动态共享目标
sln	一个静态链接的 ln 程序
sotruss	跟踪指定命令里的动态库函数调用
sprof	读取并显示共享目标分析数据

tzselect	询问用户该系统的地理位置并给出相应的时区描述
xtrace	跟踪程序执行过程并打印当前执行的函数
zdump	时区数据输出工具
zic	时区数据编译工具
ld-2.29.so	用于动态库执行的辅助程序
libBrokenLocale	Glibc 内部的一个粗暴破解用来修复损坏程序（比如，一些 Motif 应用）。查看文件 glibc-2.29/locale/broken_cur_max.c 里的注释来了解更多信息
libSegFault	段错误信号处理函数，catchsegv 会用到
libanl	一个异步名字查找库
libc	主要的 C 库
libcidn	Glibc 内部用于在函数 getaddrinfo() 中处理国际化域名
libcrypt	密码学函数库
libdl	动态链接接口函数库
libg	不包含函数的一个空库。以前是 g++ 的运行时库
libieee	链接该模块会强制使用电气与电子工程师协会（IEEE) 定义的数学函数错误处理规则。默认的是 POSIX.1 错误处理
libm	数学运算函数库
libmcheck	链接这个库后会打开内存分配检查
libmemusage	Used by memusage 命令用它来协助收集应用程序里内存使用信息
libnsl	网络服务函数库
libnss	名称服务切换函数库，包含了解析主机名、用户名、组名、别称、服务、协议等等的函数
libpthread	POSIX 线程函数库
libresolv	包含了创建、发送和解析互联网域名服务器封包的函数
librpcsvc	包含了提供杂项 RPC 服务的函数
librt	包含了实现 POSIX.1b 实时扩展里规定的大部分接口的函数
libthread_db	包含了方便构建多线程程序调试工具的函数
libutil	包含各种 Unix 应用程序中用到的「标准」函数的代码

6.10. 调整工具链

现在最后的 C 语言库已经装好了，是时候调整工具链，让新编译的程序链接到这些新的库上。

首先，备份 /tools 链接器，然后用我们在第五章调整过的链接器代替它。我们还会创建一个链接，链接到 /tools/$(uname -m)-pc-linux-gnu/bin 的副本：

```
mv -v /tools/bin/{ld,ld-old}
mv -v /tools/$(uname -m)-pc-linux-gnu/bin/{ld,ld-old}
mv -v /tools/bin/{ld-new,ld}
ln -sv /tools/bin/ld /tools/$(uname -m)-pc-linux-gnu/bin/ld
```

接下来，修改 GCC 参数文件，让它指向新的动态连接器。只需删除所有「/tools」的实例，这样应该可以留下到达动态链接器的正确路径。还要调整参数文件，这样 GCC 就知道怎样找到正确的头文件和 Glibc 启动文件。一个 sed 命令就能完成这些：

```
gcc -dumpspecs | sed -e 's@/tools@@g'                   \
    -e '/\*startfile_prefix_spec:/{n;s@.*@/usr/lib/ @}' \
    -e '/\*cpp:/{n;s@$@ -isystem /usr/include@}' >      \
    `dirname $(gcc --print-libgcc-file-name)`/specs
```

直观地检查参数文件来确认预期的变化已确实完成是个好办法。

确保已调整的工具链的基本功能（编译和链接）都能如期进行是非常必要的。怎样做呢？执行下面这条命令：

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

如果没有任何错误，上条命令的输出应该是（不同的平台上的动态链接器可能名字不同）：

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

注意在 64 位系统中，虽然我们的动态链接位于 /lib 中，但却可以通过 /lib64 中的符号链接访问。

> **Note**
>
> 32 位系统的解释器应该是 /lib/ld-linux.so.2.

现在确保我们已经设置好了启动文件：

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

上一条命令的输出应该是：

```
/usr/lib/../lib/crt1.o succeeded
/usr/lib/../lib/crti.o succeeded
/usr/lib/../lib/crtn.o succeeded
```

确保链接器能找到正确的头文件：

```
grep -B1 '^ /usr/include' dummy.log
```

这条命令应该返回如下输出：

```
#include <...> search starts here:
 /usr/include
```

接下来，确认新的链接器已经在使用正确的搜索路径：

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

应该忽略指向带有 '-linux-gnu' 的路径，上条命令的输出应该是：

```
SEARCH_DIR("/usr/lib")
SEARCH_DIR("/lib")
```

然后我们要确定我们使用的是正确的 libc：

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

上条命令的输出应该为：

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

最后，确保 GCC 使用的是正确的动态链接器：

```
grep found dummy.log
```

上条命令的结果应该是（不同的平台上链接器名字可以不同）：

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

如果显示的结果不一样或者根本没有显示，那就出了大问题。检查并回溯之前的步骤，找到出错的地方并改正。最有可能的原因是参数文件的调整出了问题。在进行下一步之前所有的问题都要解决。

确保一切正常之后，清除测试文件：

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

6.11. Zlib-1.2.11

Zlib 软件包包括一些程序所使用的压缩和解压缩例程。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 4.4 MB

6.11.1. 安装 Zlib

准备编译 Zlib：

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

编译软件包：

```
make
```

输入命令查看结果：

```
make check
```

安装软件包:

```
make install
```

共享库需要移动到 /lib，因此需要重建 .so 里面的 /usr/lib 文件：

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

6.11.2. Zlib 软件包内容

安装的库: libz.{a,so}

简要介绍

libz 包括一些程序所使用的压缩和解压缩功能。

6.12. File-5.36

File 软件包包括一个判断给定的某个或某些文件文件类型的工具。

大致构建用时: 0.1 SBU
所需磁盘空间: 18 MB

6.12.1. 安装 File

准备编译 File：

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

编译软件包：

```
make
```

输入命令检查结果：

```
make check
```

安装软件包：

```
make install
```

6.12.2. File 软件包内容

安装的程序: file
安装的库: libmagic.so

简要介绍

file 为每个文件归类；这可以通过一些测试达到——文件系统测试，魔术数字测试还有语言测试。
`libmagic` 包含程序 file 进行魔术数字识别的例程。

6.13. Readline-8.0

Readline 软件包是提供命令行编辑和历史功能的库的集合。

大致构建用时: 0.1 SBU
所需磁盘空间: 15 MB

6.13.1. 安装 Readline

重装 Readline 会使旧的库移动到 <libraryname>.old。通常来说这并不是什么问题，但一些情况下可能引起 ldconfig 链接错误。可以通过下面的两个 sed 命令避免这个问题：

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

准备编译 Readline：

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

编译软件包：

```
make SHLIB_LIBS="-L/tools/lib -lncursesw"
```

make 选项的含义：

SHLIB_LIBS="-L/tools/lib -lncursesw"
该选项强制 Readline 链接到 libncursesw 库。

该软件包没有测试套件。

安装软件包：

```
make SHLIB_LIBS="-L/tools/lib -lncursesw" install
```

现在移动动态库到更合适的位置并修正一些文件权限和符号链接：

```
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
```

如果需要的话，安装帮助文档：

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

6.13.2. Readline 软件包内容

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

简要介绍

`libhistory` 为重新查看历史行提供一致的用户界面

`libreadline` 提供一组用于操纵文本进入交互式会话程序的命令

6.14. M4-1.4.18

M4 软件包包含一个宏处理器。

大致构建用时: 0.4 SBU
所需磁盘空间: 33 MB

6.14.1. 安装 M4

首先，对应 glibc-2.28 的需求做一些修复：

```
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
```

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

用以下命令测试结果：

```
make install
```

6.14.2. M4 软件包内容

安装的程序: m4

简要介绍

m4 复制给定的文件并扩展其中包括的宏[这些宏或者是内建的或是用户定义的，可以有任何数目的参数。除了进行宏扩展，m4 还有用于包含命名文件、运行 Unix 命令、进行整数运算、操作文本、递归等内建函数。m4 程序可以作为一个编译器的前端，也可以作为一个宏处理器使用。]

6.15. Bc-1.07.1

Bc 软件包包括一个任意精度数值处理的语言。

大致构建用时: 0.1 SBU
所需磁盘空间: 4.1 MB

6.15.1. 安装 Bc

首先，更改内部脚本用 sed 代替 ed：

```
cat > bc/fix-libmath_h << "EOF"
#! /bin/bash
sed -e '1   s/^/{"/' \
    -e     's/$/",/' \
    -e '2,$ s/^/"/'  \
    -e   '$ d'       \
    -i libmath.h

sed -e '$ s/$/0}/' \
    -i libmath.h
EOF
```

创建临时符号链接，以便软件包能找到 readline 库，并确认所需的 libncurses 库可用。虽然现在这些库位于 /tools/lib 中，但在本章节的最后系统将改用 /usr/lib。

```
ln -sv /tools/lib/libncursesw.so.6 /usr/lib/libncursesw.so.6
ln -sfv libncursesw.so.6 /usr/lib/libncurses.so
```

修复一个由于在 LFS 早期阶段缺少文件所导致的 configure 问题：

```
sed -i -e '/flex/s/as_fn_error/: ;; # &/' configure
```

准备编译 Bc：

```
./configure --prefix=/usr           \
            --with-readline         \
            --mandir=/usr/share/man \
            --infodir=/usr/share/info
```

配置选项的含义：

--with-readline
该选项告诉 Bc 使用系统中已经安装的 `readline` 库而不是使用自带的 readline 版本。

编译软件包：

```
make
```

运行下面的命令来测试 bc。这会输出好多内容，因此你可能希望重定向到一个文件。测试中会有很小的比例（10 分之 12,144）最后一位数字会有舍入误差。

```
echo "quit" | ./bc/bc -l Test/checklib.b
```

安装软件包：

```
make install
```

6.15.2. Bc 软件包内容

安装的程序: bc 和 dc

简要介绍

bc 一个命令行计算器
dc 逆波兰命令行计算器

6.16. Binutils-2.32

Binutils 软件包包含一个链接器、一个汇编器、以及其它处理目标文件的工具。

大致构建用时: 6.9 SBU
所需磁盘空间: 4.9 GB

6.16.1. 安装 Binutils

通过一个简单测试验证在 chroot 环境下 PTY 工作正常：

```
expect -c "spawn ls"
```

这个命令应该输出以下内容：

```
spawn ls
```

假如输出包括下面的信息，那么表示没有为 PTY 操作设置好环境。在运行 Binutils 和 GCC 的测试套件之前需要解决这个问题：

```
The system has no more ptys.
Ask your system administrator to create more.
```

Binutils 的文档建议在一个专用的编译目录中编译 Binutils：

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

配置 Binutils 准备编译：

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

配置参数的意义：

`--enable-gold`
构建 gold 链接器并将其安装为 ld.gold（紧挨着默认链接器）。

`--enable-ld=default`
构建原来的 bdf 链接器并将其安装为 ld（默认链接器）和 ld.bfd。

`--enable-plugins`
为链接器启用插件支持。

`--enable-64-bit-bfd`
启用 64 位支持（针对字宽较窄的主机）。64 位系统可能没什么必要，但也不会有什么坏处。

`--with-system-zlib`
使用安装的 zlib 库替代自带的版本构建。

编译软件包：

```
make tooldir=/usr
```

make 参数的含义:

`tooldir=/usr`
一般来说，tooldir (最终存放可执行文件的目录) 设置为 `$(exec_prefix)/$(target_alias)`。例如，x86_64 机器会把它扩展为 `/usr/x86_64-unknown-linux-gnu`。因为这是个自定制的系统，并不需要 `/usr`中的特定目标目录。如果系统用于交叉编译（例如，在 Intel 机器上编译能生成在 PowerPC 机器上运行的代码的软件包）会使用 `$(exec_prefix)/$(target_alias)`。

> **Important**
>
> 本章节中的 Binutils 测试套件至关重要，任何情况下都不能跳过。

查看结果：

```
make -k check
```

已知一个测试 debug_msg.sh 会失败。

安装软件包：

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

6.16.2. Binutils 内容

安装的程序:	addr2line, ar, as, c++filt, elfedit, gprof, ld, ld.bfd, ld.gold, nm, objcopy, objdump, ranlib, readelf, size, strings, 和 strip
安装的库:	libbfd.{a,so} 和 libopcodes.{a,so}
安装目录:	/usr/lib/ldscripts

简要介绍

addr2line	转换程序地址为文件名称和行号；给定一个地址和可执行文件的名称，它使用可执行文件中的调试信息来判断与该地址关联的源文件以及行号。
ar	创建、更改以及抽取归档文件。
as	一个将 gcc 的输出汇编到目标文件的汇编器。
c++filt	链接器用来过滤 C++ 和 Java 符号以及防止重载函数冲突。
elfedit	更新 ELF 文件的 ELF 文件头。
gprof	显示调用关系图配置数据。
ld	一个将多个目标文件和归档文件合并为单一文件，重定位数据及绑定符号引用的链接器。
ld.gold	一个阉割版的 ld，仅支持 elf 对象文件格式
ld.bfd	到 ld 的硬链接。
nm	列出指定目标文件中出现的符号。
objcopy	转换某种类型的目标文件到另一种类型。
objdump	显示给定目标文件的信息，用选项可以控制显示特定信息；显示的信息对于使用编译工具的程序员非常有用。
ranlib	生成归档文件内容的索引并保存到归档文件；索引列出了所有归档文件成员——可重定位的目标文件定义的符号。
readelf	显示 ELF 类型的二进制文件的信息。
size	列出所给目标文件各部分大小和总的大小。
strings	对每个给定文件，输出不低于指定长度（默认是4）的可打印字符序列；对于目标文件，它默认只打印初始化和引导部分的字符串，而对于其它类型的文件扫描整个文件。
strip	从目标文件中去除符号。
`libbfd`	二进制文件描述库。
`libopcodes`	一个库用于处理操作码——「可读文本」版的处理器指令；用于构建类似 objdump 的工具。

6.17. GMP-6.1.2

GMP 软件包包含一些数学库。这里有对任意精度数值计算很有用的函数。

大致构建用时: 1.3 SBU
所需磁盘空间: 61 MB

6.17.1. 安装 GMP

> **Note**
>
> 如果你是为 32 位的 x86 系统编译，但是你的 CPU 可以运行 64 位代码并且环境中你有指定的 CFLAGS，那么配置脚本会尝试配置为 64 位并导致失败。用下面的 方式执行配置命令来避免这个问题：
>
> ```
> ABI=32 ./configure ...
> ```

> **Note**
>
> GMP 的默认设定会为主机的处理器优化库。如果你不需要完美符合主机 CPU 的库，可以通过下方命令创建通用库，这样的话契合度会差一些：
>
> ```
> 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-6.1.2
```

新配置选项的含义：

--enable-cxx
这个参数启用 C++ 支持

--docdir=/usr/share/doc/gmp-6.1.2
这个变量指定保存文档的正确位置

编译软件包并生成 HTML 文档：

```
make
make html
```

> **Important**
>
> 该章节 GMP 的测试套件至关重要，任何情况下都不能跳过。

查看结果：

```
make check 2>&1 | tee gmp-check-log
```

> **Caution**
>
> GMP 中的代码对于其构建的处理器进行了高度优化。有时，检测处理器的代码会误认系统的功能，并在测试中报错，或是在其他应用使用 GMP 库的时候显示消息「Illegal instruction（非法指令）」。在这种情况下，GMP 需要重新配置选项 --build=x86_64-unknown-linux-gnu 并重新构建。

确认测试套件中所有的 190 个测试都通过了。通过输入下面的命令检查结果：

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

安装软件包和文档：

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

6.17.2. GMP 内容

安装的库: libgmp.so 和 libgmpxx.so
安装目录: /usr/share/doc/gmp-6.1.2

简要介绍

`libgmp`	包括精度数学函数
`libgmpxx`	包括 C++ 精度属性函数

6.18. MPFR-4.0.2

MPFR 软件包包含多精度数学函数。

大致构建用时: 1.0 SBU
所需磁盘空间: 37 MB

6.18.1. 安装 MPFR

准备编译 MPFR：

```
./configure --prefix=/usr        \
            --disable-static     \
            --enable-thread-safe \
            --docdir=/usr/share/doc/mpfr-4.0.2
```

编译软件包并生成 HTML 文档：

```
make
make html
```

> **Important**
>
> 该章节 MPFR 的测试套件至关重要，任何情况下都不能跳过。

检查结果确认通过了所有的测试：

```
make check
```

安装软件包以及文档：

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

6.18.2. MPFR 软件包内容

安装的库: libmpfr.so
安装目录: /usr/share/doc/mpfr-4.0.2

简要介绍

`libmpfr` 包含多精度数学函数

6.19. MPC-1.1.0

MPC 软件包包含一个能以任意高精度进行复数数值计算和对结果进行正确四舍五入的库。

大致构建用时: 0.3 SBU
所需磁盘空间: 22 MB

6.19.1. 安装 MPC

准备编译 MPC：

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/mpc-1.1.0
```

编译软件包并生成 HTML 文档：

```
make
make html
```

用以下命令检查结果：

```
make check
```

安装软件包以及它的帮助文档：

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

6.19.2. MPC 软件包内容

安装的库: libmpc.so
安装目录: /usr/share/doc/mpc-1.1.0

简要介绍

`libmpc` 包含复数数学函数

6.20. Shadow-4.6

Shadow 软件包包含以安全方式处理密码的程序。

大致构建用时: 0.2 SBU
所需磁盘空间: 46 MB

6.20.1. 安装 Shadow

> **Note**
>
> 如果你喜欢强制使用更强的密码，在编译 Shadow 之前可以根据 http://www.linuxfromscratch.org/blfs/view/8.4/postlfs/cracklib.html 安装 CrackLib。然后在下面的 configure 命令中增加 *--with-libcrack*。

禁用对 groups 程序以及相应 man 手册的安装，Coreutils 已经提供了更棒的版本。同时也避免了安装已由 Section#6.8, “Man-pages-4.16”安装过的手册页：

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

比起默认的 crypt 方法，用更安全的 SHA-512 方法加密密码，它允许密码长度超过 8 个字符。也需要把 Shadow 默认使用的用户邮箱由陈旧的 `/var/spool/mail` 位置改为正在使用的 `/var/mail` 位置：

```
sed -i -e 's@#ENCRYPT_METHOD DES@ENCRYPT_METHOD SHA512@' \
       -e 's@/var/spool/mail@/var/mail@' etc/login.defs
```

> **Note**
>
> 如果你选择编译支持 Cracklib 的 Shadow，运行下面的命令：
>
> ```
> sed -i 's@DICTPATH.*@DICTPATH\t/lib/cracklib/pw_dict@' etc/login.defs
> ```

做一个小改动，用 useradd 1000 生成第一个组号：

```
sed -i 's/1000/999/' etc/useradd
```

准备编译 Shadow：

```
./configure --sysconfdir=/etc --with-group-name-max-length=32
```

配置选项的含义：

--with-group-name-max-length=32
最长用户名为 32 个字符，使组名称也是如此。

编译软件包：

```
make
```

该软件包没有测试套件。

安装软件包：

```
make install
```

移动位置错误的程序到正确的位置：

```
mv -v /usr/bin/passwd /bin
```

6.20.2. 配置 Shadow

该软件包包含增加、更改、以及删除用户和组的工具；设置和修改密码；执行其它特权级任务。软件包解压后的 `doc/HOWTO` 文件有关于 password shadowing 的完整解释。如果使用 Shadow 支持，记住需要验证密码（显示管理器、FTP 程序、pop3 守护进程等）的程序必须和 Shadow 兼容。也就是说，它们要能使用 Shadow 加密的密码。

运行下面的命令启用 shadow 密码：

```
pwconv
```

运行下面的命令启用 shadow 组密码：

```
grpconv
```

用于 useradd 工具的 Shadow 配置有一些需要解释的注意事项。首先，useradd 工具的默认操作是创建用户以及和用户名相同的组。默认情况下，用户 ID(UID) 和组 ID(GID) 的数字从 1000 开始。这意味着如果你不传递参数给 useradd，系统中的每个用户都会属于一个不同的组。如果不需要这样的结果，你需要传递参数 *-g* 到 useradd。默认参数保存在 `/etc/default/useradd` 文件中。你需要修改该文件中的两个参数来实现你的特定需求。

`/etc/default/useradd` 参数解释

`GROUP=1000`
该参数设定 /etc/group 文件中使用的起始组序号。你可以把它更改为任何你需要的数字。注意 useradd 永远不会重用 UID 或 GID。如果该参数指定的数字已经被使用了，将会使用它之后的下一个可用数字。另外注意如果你系统中没有序号为 1000 的组，第一次使用 useradd 而没有参数 *-g* 的话，你会在终端中看到一个提示信息：`useradd: unknown GID 1000`。你可以忽视这个信息，它会使用组号 1000。

`CREATE_MAIL_SPOOL=yes`
这个参数会为 useradd 新添加的用户创建邮箱文件。useradd 会使组 `mail` 拥有该文件的所有权，并赋予组 0660 的权限。如果你希望 useradd 不创建这些邮箱文件，你可以运行下面的命令：

```
sed -i 's/yes/no/' /etc/default/useradd
```

6.20.3. 设置 root 密码

运行下面的命令为用户 root 设置密码：

```
passwd root
```

6.20.4. Shadow 软件包内容

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 (链接到 newgrp), su, useradd, userdel, usermod, vigr (链接到 vipw), 和 vipw

安装目录: /etc/default

简要介绍

chage	用来更改强制性密码更新的最大天数
chfn	用来更改用户的全名以及其它信息
chgpasswd	用来以批处理模式更新组密码
chpasswd	用来以批处理模式更新用户密码
chsh	用来更改用户登录时默认使用的 shell
expiry	检查并强制执行当前密码过期策略
faillog	用来检查登录失败的日志文件，设置锁定用户的最大失败次数，或者重置失败次数
gpasswd	用来给组增加、删除成员以及管理员
groupadd	用指定的名称创建组
groupdel	用指定的名称删除组
groupmems	允许用户管理他/她自己的组成员列表而不需要超级用户权限。
groupmod	用于更改指定组的名称或 GID
grpck	验证组文件 `/etc/group` 和 `/etc/gshadow` 的完整性
grpconv	从普通组文件创建或升级为 shadow 组文件
grpunconv	从 `/etc/group` 更新到 `/etc/gshadow` 然后删除前者
lastlog	报告所有用户或指定用户的最近一次登录

login	用于系统让用户登录进来
logoutd	用于强制限制登录时间和端口的守护进程
newgidmap	用于设置用户命名空间的 gid 映射
newgrp	用于在一次登录会话中更改当前 GID
newuidmap	用于设置用户命名空间的 uid 映射
newusers	用于批量创建或更新用户账户
nologin	显示一个账户不可用的信息；它用于来作为不可登录的账户的默认 shell
passwd	用来更改用户或组账户的密码
pwck	验证密码文件 `/etc/passwd` 和 `/etc/shadow` 的完整性
pwconv	从普通密码文件创建或升级 shadow 密码文件
pwunconv	从 `/etc/passwd` 更新到 `/etc/shadow` 然后删除前者
sg	当用户的 GID 被设置为指定组的 GID 时执行一个特定命令
su	用替换的用户和组 ID 运行 Shell
useradd	用指定的名称新建用户或更新新用户的默认信息
userdel	删除指定的用户账户
usermod	用于更改指定用户的登录名称、UID、shell、初始组、home 目录，等
vigr	编辑 `/etc/group` 或 `/etc/gshadow` 文件
vipw	编辑 `/etc/passwd` 或 `/etc/shadow` 文件

6.21. GCC-8.2.0

GCC 软件包包括 GNU 编译器集，其中有 C 和 C++ 的编译器。

大致构建用时: 92 SBU (with tests)
所需磁盘空间: 3.9 GB

6.21.1. 安装 GCC

如果是在 x86_64 上实施构建，更改 64 位库的默认目录名为「lib」：

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

删除之前创建的符号链接，因为最终 gcc 将被安装在这：

```
rm -f /usr/lib/gcc
```

GCC 的文档建议在源代码目录之外一个专用的编译目录中编译 GCC：

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

准备编译 GCC：

```
SED=sed                               \
../configure --prefix=/usr            \
             --enable-languages=c,c++ \
             --disable-multilib       \
             --disable-bootstrap      \
             --disable-libmpx         \
             --with-system-zlib
```

注意，对于其它的编程语言，现在还有一些前提条件没有准备好。可以查看 BLFS Book 了解如何编译 GCC 支持的所有语言的指令。

新配置选项的含义：

SED=sed
: 设置环境变量防止访问到硬编码的 /tools/bin/sed 路径。

--disable-libmpx
: 这个选项告知 GCC 不要编译 mpx （Memory Protection Extensions：内存保护扩展）。此特性在一些处理器上存在问题，已经在下一个版本的 GCC 中移除。

--with-system-zlib
: 这个选项告诉 GCC 链接系统安装的 Zlib 库，而不是它内部自带的库。

编译软件包：

```
make
```

> **Important**
>
> 本章节中 GCC 的测试套件至关重要，任何情况下都不能跳过。

GCC 测试套件中一个测试集的会耗尽堆栈空间，因此运行测试之前要增加栈大小：

```
ulimit -s 32768
```

删除已知会导致问题的测试：

```
rm ../gcc/testsuite/g++.dg/pr83239.C
```

以非特权用户测试编译结果，不要因为出现错误就停下来：

```
chown -Rv nobody .
su nobody -s /bin/bash -c "PATH=$PATH make -k check"
```

要查看测试套件结果的概要，运行：

```
../contrib/test_summary
```

如果仅查看摘要，则使用管道 **grep -A7 Summ** 选项控制输出将输出：

```
../contrib/test_summary | grep -A7 Summ
```

结果可以和 http://www.linuxfromscratch.org/lfs/build-logs/8.4/ 以及 http://gcc.gnu.org/ml/gcc-testresults/ 上的相比较。

一些意料之外的错误总是难以避免。GCC 开发者通常会意识到这些问题，但还没有解决。除非测试结果和上面 URL 中的相差很大，不然就可以安全继续。

> **Note**
>
> 在某些内核配置和 AMD 处理器组合的时候，肯能会在 gcc.target/i386/mpx 测试（旨在测试最新英特尔处理器上的 MPX 选项）中出现 1100 个失败。AMD 处理器的话可以安心的忽略这些。

安装软件包：

```
make install
```

创建 FHS 因为「历史」原因而需要的软链接。

```
ln -sv ../usr/bin/cpp /lib
```

很多软件包用命令 cc 调用 C 编译器。为了满足这些软件包，创建一个符号链接：

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

增加一个兼容符号链接启用编译程序时进行链接时间优化（Link Time Optimization，LTO）：

```
install -v -dm755 /usr/lib/bfd-plugins
ln -sfv ../../libexec/gcc/$(gcc -dumpmachine)/8.2.0/liblto_plugin.so \
        /usr/lib/bfd-plugins/
```

现在我们最终的工具链已经准备就绪了，再一次确认编译和链接都能像预期那样正常工作很重要。我们通过做和前面章节做过的相同的完整性检查做到这点：

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

如果没有任何错误，上条命令的输出应该是（不同的平台上的动态链接器可能名字不同）：

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

现在确保我们已经设置好了启动文件：

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

上一条命令的输出应该是：

```
/usr/lib/gcc/x86_64-pc-linux-gnu/8.2.0/../../../../lib/crt1.o succeeded
/usr/lib/gcc/x86_64-pc-linux-gnu/8.2.0/../../../../lib/crti.o succeeded
/usr/lib/gcc/x86_64-pc-linux-gnu/8.2.0/../../../../lib/crtn.o succeeded
```

取决于你机器的架构，上面的结果可能有稍微不同，差异通常是 /usr/lib/gcc 后目录的名称。这里重要的一点是 gcc 能在 /usr/lib 目录下找到所有的三个 crt*.o 文件。

确保链接器能找到正确的头文件：

```
grep -B4 '^ /usr/include' dummy.log
```

这条命令应该返回如下输出：

```
#include <...> search starts here:
 /usr/lib/gcc/x86_64-pc-linux-gnu/8.2.0/include
 /usr/local/include
 /usr/lib/gcc/x86_64-pc-linux-gnu/8.2.0/include-fixed
 /usr/include
```

同时，注意你的目标系统三段式后面的目录名称可能和上面的不同，这取决于你的架构。

接下来，确认新的链接器已经在使用正确的搜索路径：

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

应该忽略指向带有 '-linux-gnu' 的路径，上条命令的输出应该是：

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

32 位的系统可能有一些不同的目录。例如，下面是一台 i686 机器的输出：

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

然后我们要确定我们使用的是正确的 libc：

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

上条命令的输出应该为：

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

最后，确保 GCC 使用的是正确的动态链接器：

```
grep found dummy.log
```

上条命令的结果应该是（不同的平台上链接器名字可以不同）：

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

如果显示的结果不一样或者根本没有显示，那就出了大问题。检查并回溯之前的步骤，找到出错的地方并改正。最有可能的原因是参数文件的调整出了问题。在进行下一步之前所有的问题都要解决。

确保一切正常之后，清除测试文件：

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

最后，移动位置放错的文件：

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

6.21.2. GCC 软件包内容

安装的程序:	c++, cc (链接到 gcc), cpp, g++, gcc, gcc-ar, gcc-nm, gcc-ranlib, 和 gcov
安装的库:	libasan.{a,so}, libatomic.{a,so}, libgcc.a, libgcc_eh.a, libgcc_s.so, libgcov.a, libgomp.{a,so}, libiberty.a, libitm.{a,so}, liblto_plugin.so, libquadmath.{a,so}, libssp.{a,so}, libssp_nonshared.a, libstdc++.{a,so}, libsupc++.a, 和 libtsan.{a,so}
安装目录:	/usr/include/c++, /usr/lib/gcc, /usr/libexec/gcc, 和 /usr/share/gcc-8.2.0

简要介绍

c++	C++ 编译器
cc	C 编译器
cpp	C 预处理器；编译器用来扩展源文件中 #include、#define 以及类似语句
g++	C++ 编译器
gcc	C 编译器

gcc-ar	增加插件到命令行的 ar 的封装。这个程序只用于添加 "链接时间优化"，在使用默认编译选项时不起作用
gcc-nm	增加插件到命令行的 nm 的封装。这个程序只用于添加 "链接时间优化"，在使用默认编译选项时不起作用
gcc-ranlib	增加插件到命令行的 ranlib 的封装。这个程序只用于添加 "链接时间优化"，在使用默认编译选项时不起作用
gcov	一个覆盖测试工具；用于分析程序以决定在哪里进行优化有最大的效果
libasan	Address Sanitizer（地址消毒剂）运行时库。
libgcc	包含用于 gcc 的运行时支持
libgcov	当指示 GCC 启用分析时该库会被链接到程序中
libgomp	用于 C/C++、Fortran 语言的多平台共享内存并行编程的 OpenMP API 的 GNU 实现
libiberty	包含多种 GNU 程序所使用的例程，包括 getopt, obstack, strerror, strtol, 和 strtoul
liblto_plugin	GCC 的链接时间优化（LTO）插件，允许 GCC 跨编译单元进行优化
libquadmath	GCC 四精度数学库 API
libssp	包含支持 GCC 堆栈溢出保护功能的例程
libstdc++	标准 C++ 库
libsupc++	为 C++ 编程语言提供支持例程
libtsan	Thread Sanitizer（数据速率检测工具） 运行时库

6.22. Bzip2-1.0.6

Bzip2 软件包包含压缩和解压缩的程序。用 bzip2 压缩文本文件能获得比传统的 gzip 更好的压缩比。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 2.3 MB

6.22.1. 安装 Bzip2

使用能为这个软件包安装帮助文档的补丁：

```
patch -Np1 -i ../bzip2-1.0.6-install_docs-1.patch
```

下面的命令确保安装的符号链接是相对链接：

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

确认 man 页面安装到了正确的位置：

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

准备编译 Bzip：

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

make 参数的含义：

`-f Makefile-libbz2_so`
这会使用不同的 `Makefile` 文件编译 Bzip2，在这里是 `Makefile-libbz2_so`，它会创建动态 `libbz2.so` 库，并把它链接到 Bzip2 工具。

编译并测试软件包：

```
make
```

安装程序：

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

安装使用动态链接库的 bzip2 二进制文件到 `/bin` 目录，创建一些必须的符号链接并清理：

```
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
```

6.22.2. Bzip2 软件包内容

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), 和 bzmore
安装的库: libbz2.{a,so}
安装目录: /usr/share/doc/bzip2-1.0.6

简要介绍

bunzip2	解压 bzip 压缩的文件
bzcat	解压到标准输出
bzcmp	对 bzip 压缩的文件运行 cmp 命令
bzdiff	对 bzip 压缩的文件运行 diff 命令
bzegrep	对 bzip 压缩的文件运行 egrep 命令
bzfgrep	对 bzip 压缩的文件运行 fgrep 命令
bzgrep	对 bzip 压缩的文件运行 grep 命令
bzip2	使用哈夫曼编码的 Burrows-Wheeler 块排序文本压缩算法压缩文件；压缩率比传统的用「Lempel-Ziv」算法的压缩器要好，比如 gzip。

bzip2recover	尝试从损坏的 bzip 压缩文件中恢复数据
bzless	对 bzip 压缩的文件运行 less 命令
bzmore	对 bzip 压缩的文件运行 more 命令
libbz2	用 Burrows-Wheeler 算法实现的无损的块排序数据压缩库

6.23. Pkg-config-0.29.2

pkg-config 软件包包含一个在配置和 make 文件运行时把 include 路径和库路径传递给编译工具的工具。

大致构建用时: 0.3 SBU
所需磁盘空间: 30 MB

6.23.1. 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
```

新配置选项的含义：

`--with-internal-glib`
这会让 pkg-config 使用它自己内部版本的 Glib，因为在 LFS 中没有可用的外部版本。

`--disable-host-tool`
此选项取消创建到 pkg-config 程序的不必要的硬链接。

编译软件包：

```
make
```

用以下命令检查结果：

```
make check
```

安装软件包：

```
make install
```

6.23.2. pkg-config 软件包内容

安装的软件: pkg-config
安装目录: /usr/share/doc/pkg-config-0.29.2

简要介绍

pkg-config 返回指定库或软件包的元信息

6.24. Ncurses-6.1

Ncurses 软件包包含用于不依赖于特定终端的字符屏幕处理的库。

大致构建用时: 0.3 SBU
所需磁盘空间: 42 MB

6.24.1. 安装 Ncurses

不要安装静态库，它不受配置控制：

```
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
```

新配置选项的含义：

--enable-widec
这个选项会编译宽字符库（例如 libncursesw.so.6.1）来代替普通字符库（例如 libncurses.so.6.1）。宽字符库可用于多字节和传统的 8 位本地字符，而常规的库只能用于 8 位本地字符。宽字符库和常规的库是源文件兼容的，而不是二进制文件兼容的。

--enable-pc-files
该选项为 pkg-config 生成和安装 .pc 文件。

--without-normal
该选项用于禁用构建和安装大多数的静态库。

编译软件包：

```
make
```

该软件包有个测试套件，但只能在安装完软件包后运行。测试程序在 test/ 目录中。查看该目录中的 README 文件获取更详细信息。

安装软件包：

```
make install
```

移动共享库到期望的 /lib 文件夹：

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

由于库已经被移走了，符号链接指向了一个不存在的文件。重建符号链接：

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

很多应用程序仍然希望编辑器能找到非宽字符的 Ncurses 库。通过符号链接和链接器脚本欺骗这样的应用链接到宽字符库：

```
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，确保在编译时会查找 -lcurses 的旧应用程序仍然可以编译：

```
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 的帮助文档：

```
mkdir -v       /usr/share/doc/ncurses-6.1
cp -v -R doc/* /usr/share/doc/ncurses-6.1
```

Note

上面的指令并不会创建非宽字符 Ncurses 库，因为没有从源文件中编译安装的软件包会在运行时链接它们。然而，已知的仅有二进制应用程序并能链接到非等宽字符的库，需要第 5 版的支持。如果你由于一些仅有二进制的应用程序或要和 LSB 兼容而必须要有这样的库，用下面的命令重新编译软件包：

```
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
```

6.24.2. Ncurses 软件包内容

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, 和 tset
安装的库:	libcursesw.so (到 libncursesw.so 的符号链接和链接器脚本), libformw.so, libmenuw.so, libncursesw.so, libncurses++w.a, libpanelw.so, 以及库名称中没有「w」的对应的非宽字符部分。
安装目录:	/usr/share/tabset, /usr/share/terminfo, 和 /usr/share/doc/ncurses-6.1

简要介绍

captoinfo	转换 termcap 描述为 terminfo 描述
clear	如果可以的话清空屏幕
infocmp	比较或输出 terminfo 描述
infotocap	转换 terminfo 描述为 termcap 描述
ncursesw6-config	P为 ncurses 提供配置信息
reset	重新初始化终端为默认设置
tabs	清空终端并设置制表符长度
tic	将 terminfo 文件从源文件格式转换到二进制格式的 terminfo 条目描述编译器需要 ncurses 例程 [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，给出每个主名称和描述
tput	可以在 shell 中使用终端特定的功能值；也可用来重置或初始化终端或者报告它的完整名称
tset	可以用来初始化终端
`libcursesw`	到 `libncursesw` 的链接
`libncursesw`	包含在一个终端屏幕以多种复杂方式显示文本的函数；使用这些功能的一个好的例子是内核 make menuconfig 时的菜单显示
`libformw`	包含实现表单的函数
`libmenuw`	包含实现菜单的函数
`libpanelw`	包含实现面板的函数

6.25. Attr-2.4.48

attr 软件包包含管理文件系统对象的扩展属性的工具。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 4.2 MB

6.25.1. 安装 Attr

准备编译 Attr：

```
./configure --prefix=/usr     \
            --disable-static  \
            --sysconfdir=/etc \
            --docdir=/usr/share/doc/attr-2.4.48
```

编译软件包：

```
make
```

测试需要在支持扩展属性的文件系统上运行，例如 ext2、ext3、或者 ext4。如果同时运行多个测试会导致测试失败（-j 选项大于 1）。输入命令检查结果：

```
make check
```

安装软件包：

```
make install
```

需要移动共享库到 /lib，因此需要重建 .so 中的 /usr/lib 文件：

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

6.25.2. Attr 软件包内容

安装的程序: attr, getfattr, 和 setfattr
安装的库: libattr.so
安装目录: /usr/include/attr 和 /usr/share/doc/attr-2.4.48

简要介绍

attr	扩展文件系统对象的属性
getfattr	获取文件系统对象的扩展属性
setfattr	设置文件系统对象的扩展属性
libattr	包含管理扩展属性的库函数

6.26. Acl-2.2.53

Acl 软件包包含管理访问控制列表的工具，访问控制列表用于定义文件和目录更细粒度的自定义访问权限。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 6.4 MB

6.26.1. 安装 Acl

准备编译 Acl：

```
./configure --prefix=/usr         \
            --disable-static      \
            --libexecdir=/usr/lib \
            --docdir=/usr/share/doc/acl-2.2.53
```

编译软件包：

```
make
```

在用 Acl 库构建 Coreutils 后，Acl 测试才能在支持访问控制的文件系统上运行。如果需要的话，可以在本章后面构建完 Coreutils 之后回到这个软件包运行 make check 进行测试。

安装软件包：

```
make install
```

需要移动共享库到 /lib，因此需要重建 /usr/lib 中的 .so 文件：

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

6.26.2. Acl 软件包内容

安装的程序: chacl, getfacl, 和 sefacl
安装的库: libacl.so
安装目录: /usr/include/acl 和 /usr/share/doc/acl-2.2.53

简要介绍

chacl	更改文件或目录的访问控制列表
getfacl	获取文件访问控制列表
sefacl	设置文件访问控制列表
libacl	包括用于管理访问控制列表的库函数

6.27. Libcap-2.26

Libcap 软件包实现了可用在 Linux 内核上的对 POSIX 1003.1e 功能的用户空间接口。这些功能将所有强大 root 权限划分为不同的权限组合。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 1.4 MB

6.27.1. 安装 Libcap

避免安装静态库：

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

编译软件包

```
make
```

这个软件包没有测试套件。

安装软件包：

```
make RAISE_SETFCAP=no lib=lib prefix=/usr install
chmod -v 755 /usr/lib/libcap.so.2.26
```

make 选项的含义：

RAISE_SETFCAP=no
这个选项跳过尝试对自身使用 setcap。这可以避免内核或文件系统不支持扩展功能时出现安装错误。

lib=lib
此参数用于在 x86_64 位系统中，将库文件安装到 `$prefix/lib` 而不是 `$prefix/lib64`。此参数不影响 x86 系统。

需要移动共享库到 `/lib`，因此需要重建 `/usr/lib` 中的 `.so` 文件：

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

6.27.2. Libcap 软件包内容

安装的程序: capsh, getcap, getpcaps, 和 setcap
安装的库: libcap.so

简要介绍

capsh	使用和控制功能支持的 shell 封装
getcap	检查文件功能
getpcaps	显示查询进程的功能
setcap	设置文件的容量
`libcap`	包括用于管理 POSIX 1003.1e 功能的库函数

6.28. Sed-4.7

Sed 软件包包含一个流编辑器。

大致构建用时: 0.3 SBU
所需磁盘空间: 32 MB

6.28.1. 安装 Sed

首先修复 LFS 环境中的问题，然后移除一个失败的测试：

```
sed -i 's/usr/tools/'                 build-aux/help2man
sed -i 's/testsuite.panic-tests.sh//' Makefile.in
```

准备编译 Sed：

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

编译软件包并生成 HTML 文档：

```
make
make html
```

输入命令查看结果：

```
make check
```

安装软件包和它的文档：

```
make install
install -d -m755           /usr/share/doc/sed-4.7
install -m644 doc/sed.html /usr/share/doc/sed-4.7
```

6.28.2. Sed 软件包内容

安装的程序: sed
安装目录: /usr/share/doc/sed-4.7

简要介绍

sed 过滤器，一次性转换文本文件

6.29. Psmisc-23.2

Psmisc 软件包包含用于显示运行中进程信息的程序。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 4.5 MB

6.29.1. 安装 Psmisc

准备编译 Psmisc：

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

编译软件包：

```
make
```

该软件包没有测试套件。

安装软件包：

```
make install
```

最后，将程序 killall 和 fuser 移动到 FHS 指定的位置：

```
mv -v /usr/bin/fuser   /bin
mv -v /usr/bin/killall /bin
```

6.29.2. Psmisc 软件包内容

安装的程序: fuser, killall, peekfd, prtstat, pstree, 和 pstree.x11 (链接到 pstree)

简要介绍

fuser	报告使用指定文件或文件系统的进程的进程 ID（PID）
killall	根据名称杀死进程；它发送信号到所有的运行任何给定命令的进程
peekfd	根据 PID 查看正在运行进程的文件描述符
prtstat	打印关于某个进程的信息
pstree	以树形结构显示运行中的进程
pstree.x11	和 pstree 命令相同，但退出时它会等待确认

6.30. Iana-Etc-2.30

Iana-Etc 软件包为网络服务和协议提供数据。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 2.3 MB

6.30.1. 安装 Iana-Etc

下面的命令将 IANA 提供的原始数据转换为 `/etc/protocols` 和 `/etc/services` 数据文件的正确格式：

```
make
```

该软件包没有测试套件。

安装软件包：

```
make install
```

6.30.2. Iana-Etc 软件包内容

安装的文件: /etc/protocols 和 /etc/services

简要介绍

`/etc/protocols` 描述 TCP/IP 子系统中可用的多种 DARPA 网络协议
`/etc/services` 提供友好文本名称和背后分配的端口号以及协议类型之间的映射

6.31. Bison-3.3.2

Bison 软件包包含一个语法生成器。

大致构建用时: 0.3 SBU
所需磁盘空间: 37 MB

6.31.1. 安装 Bison

准备编译 Bison：

```
./configure --prefix=/usr --docdir=/usr/share/doc/bison-3.3.2
```

编译软件包：

```
make
```

考虑到 bison 和 flex 的检查有循环依赖。如有需要，在下一节安装 flex 之后，可以使用 make check 命令重新编译并检查 bison 软件包。

安装软件包：

```
make install
```

6.31.2. Bison 软件包内容

安装的程序: bison 和 yacc
安装的库: liby.a
安装目录: /usr/share/bison

简要介绍

bison 根据一系列规则生成用于分析文本结构的程序；Bison 是 Yacc（Yet Another Compiler Compiler） 的替代品。

yacc bison 的封装，用于仍然调用 yacc 而不是 bison 的程序；它会调用带有 *-y* 选项的 bison

`liby` Yacc 库包含和 Yacc 兼容的 `yyerror` 和 `main` 程序的实现；这个库并不是很有用，但是 POSIX 要求有它

6.32. Flex-2.6.4

Flex 软件包包括一个用于生成识别文本模式的程序的工具。

大致构建用时: 0.4 SBU
所需磁盘空间: 35 MB

6.32.1. 安装 Flex

首先，修复一个 glibc-2.26 引入的问题：

```
sed -i "/math.h/a #include <malloc.h>" src/flexdef.h
```

构建过程假设能使用程序 help2man 的 --help 选项来创建 man 手册。但这显然是不存在的，所以我们使用环境变量来跳过这步。现在，准备编译 Flex：

```
HELP2MAN=/tools/bin/true \
./configure --prefix=/usr --docdir=/usr/share/doc/flex-2.6.4
```

编译软件包：

```
make
```

用以下命令测试结果（大约 0.5 SBU）：

```
make check
```

安装软件包：

```
make install
```

一些程序还不知道 flex 并尝试运行它的预处理器 lex。为了支持这些程序，创建以 `lex` 仿真模式运行 `flex` 的符号链接 lex：

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

6.32.2. Flex 软件包内容

安装的程序: flex, flex++ (链接到 flex), 和 lex (链接到 flex)
安装的库: libfl.so
安装目录: /usr/share/doc/flex-2.6.4

简要介绍

flex	一个用于生成能识别文本模式程序的工具；它允许指定多种用于模式发现的规则，从而消除了开发专门程序的需要
flex++	flex 的扩展，用于生成 C++ 代码和类。是到 flex 的符号链接
lex	一个以 flex 仿真模式运行 lex 的符号链接。
`libfl`	`flex` 库

6.33. Grep-3.3

Grep 软件包包含用于在文件中搜索的程序。

大致构建用时: 0.4 SBU
所需磁盘空间: 37 MB

6.33.1. 安装 Grep

准备编译 Grep ：

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

编译软件包：

```
make
```

用以下命令测试结果：

```
make -k check
```

安装软件包：

```
make install
```

6.33.2. Grep 软件包内容

安装的程序: egrep, fgrep, 和 grep

简要介绍

egrep 打印匹配扩展正则表达式的行
fgrep 打印匹配固定字符串列表的行
grep 打印匹配基本正则表达式的行

6.34. Bash-5.0

Bash 软件包包含 Bourne-Again Shell。

大致构建用时: 1.7 SBU
所需磁盘空间: 62 MB

6.34.1. 安装 Bash

准备编译 Bash：

```
./configure --prefix=/usr                     \
            --docdir=/usr/share/doc/bash-5.0 \
            --without-bash-malloc            \
            --with-installed-readline
```

新配置选项的含义：

--with-installed-readline
该选项告诉 Bash 使用系统中已经安装的 `readline` 库而不是使用自带的 readline 版本。

编译软件包：

```
make
```

如果不需要运行测试套件的话跳转到「安装软件包」。

准备测试，确保 `nobody` 用户可以写源文件树：

```
chown -Rv nobody .
```

现在，以 `nobody` 用户身份运行测试：

```
su nobody -s /bin/bash -c "PATH=$PATH HOME=/home make tests"
```

安装软件包并将主要的可执行文件移动至 `/bin`：

```
make install
mv -vf /usr/bin/bash /bin
```

运行新编译的 bash 程序（替换正在运行的那个）：

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

> **Note**
>
> 参数使 bash 进程成为一个可交互的登录 shell 并停用散列使得新程序可用的时候就能发现。

6.34.2. Bash 软件包内容

安装的程序: bash, bashbug, 和 sh (链接到 bash)
安装目录: /usr/share/doc/bash-5.0

简要介绍

bash	广泛使用的命令解释器；在执行一个命令之前进行多种扩展和替换，使得该解释器成为一个强大的工具
bashbug	一个 shell 脚本，用于帮助用户撰写和发送标准格式的关于 bash 的 bug 报告邮件
sh	到bash 程序的符号链接；当以 sh 调用时，在符合 POSIX 标准的情况下，bash 尽可能地模仿历史版本上 sh 的启动过程

6.35. Libtool-2.4.6

Libtool 软件包包含 GNU 通用库支持脚本。它用一致的、可移植的接口封装复杂的共享库。

大致构建用时: 1.5 SBU
所需磁盘空间: 43 MB

6.35.1. 安装 Libtool

准备编译 Libtool：

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

编译软件包：

```
make
```

用以下命令测试结果(大约 11.0 SBU)：

```
make check
```

> **Note**
>
> 在具有多个内核的系统上，libtool 的测试时间可以显著削减。为此，请在上面那行命令中添加 TESTSUITEFLAGS=-j<N>。例如，使用 -j4 或将减少 60% 的测试时间。

在 LFS 构建环境中已知有五个测试由于循环依赖会失败，但如果安装完 automake 之后重新检查，所有测试就都能通过。

安装软件包：

```
make install
```

6.35.2. Libtool 软件包内容

安装的程序: libtool 和 libtoolize
安装的库: libltdl.so
安装目录: /usr/include/libltdl 和 /usr/share/libtool

简要介绍

libtool	提供通用库编译支持服务
libtoolize	提供添加 libtool 支持到软件包的一个标准方法
`libltdl`	埋藏 dlopen 库的诸多难处

6.36. GDBM-1.18.1

GDBM 软件包包含 GNU 数据库管理器。是使用扩展散列，工作方法和标准 UNIX dbm 类似的数据库函数库。该库提供存储键/数据对、通过键搜索和检索数据、以及删除键和数据的原语。

大致构建用时: 0.1 SBU
所需磁盘空间: 11 MB

6.36.1. 安装 GDBM

准备编译 GDBM：

```
./configure --prefix=/usr    \
            --disable-static \
            --enable-libgdbm-compat
```

配置选项的含义：

`--enable-libgdbm-compat`
该选项启用编译 libgdbm 兼容性库，因为一些 LFS 之外的软件包可能需要它提供的旧的 DBM 例程。

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包：

```
make install
```

6.36.2. GDBM 软件包内容

安装的程序: gdbm_dump, gdbm_load, 和 gdbmtool
安装的库: libgdbm.so 和 libgdbm_compat.so

简要介绍

gdbm_dump	转储 GDBM 数据库到文件
gdbm_load	从转储文件重建一个 GDBM 数据库
gdbmtool	测试和更改 GDBM 数据库
`libgdbm`	包含操作散列数据库的函数
`libgdbm_compat`	包含旧的 DBM 函数的兼容性库

6.37. Gperf-3.1

Gperf 为键集合生成完美的哈希函数。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 6.3 MB

6.37.1. 安装 Gperf

准备编译 Gperf：

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

编译软件包：

```
make
```

该测试已知在运行多任务同时测试（即 -j 选项大于 1）时会失败。用以下命令测试结果：

```
make -j1 check
```

安装软件包：

```
make install
```

6.37.2. Gperf 软件包内容

安装的程序: gperf
安装目录: /usr/share/doc/gperf-3.1

简要介绍

gperf 为键集合生成完美哈希

6.38. Expat-2.2.6

Expat 软件包包含一个用于解析 XML 的面向流的 C 库。

大致构建用时: 0.1 SBU
所需磁盘空间: 11 MB

6.38.1. 安装 Expat

首先修复一个 LFS 环境中，回归测试的问题：

```
sed -i 's|usr/bin/env |bin/|' run.sh.in
```

准备编译 Expat：

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/expat-2.2.6
```

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包：

```
make install
```

如果需要的话，安装帮助文档：

```
install -v -m644 doc/*.{html,png,css} /usr/share/doc/expat-2.2.6
```

6.38.2. Expat 软件包内容

安装的程序: xmlwf
安装的库: libexpat.so
安装目录: /usr/share/doc/expat-2.2.6

简要介绍

xmlwf 用于检查 XML 文档是否格式良好的非验证工具
`libexpat` 包含用于解析 XML 的 API 函数

6.39. Inetutils-1.9.4

Inetutils 软件包包含基本的网络程序。

大致构建用时: 0.3 SBU
所需磁盘空间: 29 MB

6.39.1. 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，脚本使用该程序传递消息到系统日志守护进程。因为 Util-linux 安装了一个更新版本因此不能安装这个。

`--disable-whois`
该选项禁用编译过时的 Inetutils whois 客户端。BLFS 指南中有更好的 whois 客户端说明。

`--disable-r*`
为了安全，该参数使编译过时的程序不能被使用。提供该功能的程序在手册 BLFS 中的 openssh 会有所提及。

`--disable-servers`
禁用安装作为 Inetutils 软件包一部分的多种网络服务程序。这些服务程序被认为不适用于基础的 LFS 系统。其中的一些本来就不安全，或者说仅在可信网络中才被认为安全。注意这些服务程序有更好的可用替代品。

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

> **Note**
>
> 测试 libls.sh 可能会在初始的 chroot 环境中失败，但是在 LFS 系统构建完成后重新运行就会通过了。测试 ping-localhost.sh 会因为宿主系统不支持 IPv6 而失败。

安装软件包：

```
make install
```

移动一些程序使得 `/usr` 在不可访问时仍保持可用：

```
mv -v /usr/bin/{hostname,ping,ping6,traceroute} /bin
mv -v /usr/bin/ifconfig /sbin
```

6.39.2. Inetutils 软件包内容

安装的程序: dnsdomainname, ftp, ifconfig, hostname, ping, ping6, talk, telnet, tftp, 和 traceroute

简要介绍

dnsdomainname 显示系统的 DNS 域名

ftp	简要介绍
hostname	报告或设置主机名称
ifconfig	管理网络接口
ping	发送请求应答包并报告响应用时
ping6	用于 IPv6 网络的 ping 版本
talk	用于和另一个用户交互
telnet	TELNET 协议接口
tftp	简单文件传输程序
traceroute	跟踪从你的工作主机发送到另一个网络上的主机的数据包通过的路径，显示中间通过的跳（网关）。

6.40. Perl-5.28.1

Perl 软件包包含实用信息抽取与报告语言。

大致构建用时: 7.1 SBU
所需磁盘空间: 274 MB

6.40.1. 安装 Perl

首先创建其中一个 Perl 配置文件和可选测试套件引用的基本 `/etc/hosts` 文件：

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

该版本的 Perl 会编译 Compress::Raw::Zlib 和 Compress::Raw::BZip2 模块。Perl 默认会使用内部的源码用于构建。用以下的命令使 Perl 使用系统中已安装的库：

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

为了能完全控制 Perl 的设置，你可以在下面的命令中移除「-des」选项并手动设置编译该软件包的方式。相应的，用下面的命令来使用 Perl 自动检测到的默认值：

```
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
这能确保 perl 知道，该如何告知软件包应该将它们的 perl 模块安装在哪里。

-Dpager="/usr/bin/less -isR"
这能确保使用的是 **less** 而非 **more**。

-Dman1dir=/usr/share/man/man1 -Dman3dir=/usr/share/man/man3
由于 Groff 还没有安装，Configure 会认为我们不希望为 Perl 安装 man 手册。用这些参数更改这个判断。

-Duseshrplib
构建某些 perl 模块需要的共享 libperl。

-Dusethreads
构建支持线程的 perl。

编译软件包：

```
make
```

用以下命令测试结果（大概 11 SBU）：

```
make -k test
```

> **Note**
>
> 因为使用了最新的 gdbm 从而会导致一项测试失败。

安装软件包并清理：

```
make install
unset BUILD_ZLIB BUILD_BZIP2
```

6.40.2. Perl 软件包内容

安装的程序: corelist, cpan, enc2xs, encguess, h2ph, h2xs, instmodsh, json_pp, libnetcfg, perl, perl5.28.1 (到 perl 的硬链接), perlbug, perldoc, perlivp, perlthanks (到 perlbug 的硬链接), piconv, pl2pm, pod2html, pod2man, pod2text, pod2usage, podchecker, podselect, prove, ptar, ptardiff, ptargrep, shasum, splain, xsubpp, 和 zipdetails
安装的库: 过多，不能在这完整列出
安装目录: /usr/lib/perl5

简要介绍

corelist	Module::CoreList 的命令行前端
cpan	用命令行与综合 Perl 归档网络（Comprehensive Perl Archive Network,CPAN）交互
enc2xs	从 Unicode 字符映射或 Tcl 编码文件为 Encode 模块编译 Perl 扩展
encguess	猜测一个或多个文件的编码类型
h2ph	转换 `.h` C 头文件为 `.ph` Perl 头文件
h2xs	转换 `.h` C 头文件为 Perl 扩展
instmodsh	用于检查安装的 Perl 模块的 shell 脚本，并且能从一个安装的模块中创建 tar 包
json_pp	在特定输入输出格式之间转换数据
libnetcfg	可用于配置 `libnet` Perl 模块
perl	将 C,sed，awk 以及 sh 一些最好的特性结合到一个单一的强大语言
perl5.28.1	到 perl 的硬链接
perlbug	用于生成关于 Perl、或者一起发布的模块的 bug 报告，并用邮件通知
perldoc	用嵌入到 Perl 安装目录或 Perl 脚本中的 pod 格式显示文档
perlivp	Perl 安装验证程序；能用于验证 Perl 和它的库是否正确安装
perlthanks	用于生成发送到 Perl 开发者的感谢邮件
piconv	Perl 版本的字符编码转换程序 iconv
pl2pm	用于将 Perl4 `.pl` 文件转换为 Perl5 `.pm` 模块的工具
pod2html	将文件从 pod 格式转换为 HTML 格式
pod2man	将 pod 格式数据转换为格式化的 *roff 输入
pod2text	将 pod 数据转换为格式化的 ASCII 文本
pod2usage	从文件中嵌入的 pod 文档显示使用信息
podchecker	检查 pod 格式的文档文件语法
podselect	显示 pod 文档选中的章节
prove	运行对 Test::Harness 模块测试的命令行工具
ptar	用 Perl 写的类似 tar 的程序
ptardiff	用于比较提取的文档和未提取的 Perl 程序
ptargrep	用于对 tar 归档文件中的内容进行模式匹配的 Perl 程序
shasum	打印或检查 SHA 校验码
splain	用于 Perl 中的强制冗长警告诊断
xsubpp	转换 Perl XS 代码为 C 代码
zipdetails	显示 Zip 文件内部结构的详细信息

6.41. XML::Parser-2.44

XML::Parser 模块是到 James Clark 的 XML 解析器的 Perl Expat 接口。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 2.1 MB

6.41.1. 安装 XML::Parser

准备编译 XML::Parser：

```
perl Makefile.PL
```

编译软件包：

```
make
```

用以下命令测试结果：

```
make test
```

安装软件包：

```
make install
```

6.41.2. XML::Parser 软件包内容

安装的模块: Expat.so

简要介绍

`Expat` 提供 Perl Expat 接口

6.42. Intltool-0.51.0

Intltool 是一个用于从源文件中抽取可翻译字符串的国际化工具。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 1.5 MB

6.42.1. 安装 Intltool

首先修复 perl-5.22 和其后版本导致的警告：

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

准备编译 Intltool：

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

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包：

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

6.42.2. Intltool 软件包内容

安装的程序: intltool-extract, intltool-merge, intltool-prepare, intltool-update, 和 intltoolize
安装的目录: /usr/share/doc/intltool-0.51.0 and /usr/share/intltool

简要介绍

intltoolize	准备使用 intltool 的软件包
intltool-extract	生成 gettext 能读取的头文件
intltool-merge	合并翻译后的字符串到多种文件格式
intltool-prepare	更新 pot 文件并把它们和翻译文件合并
intltool-update	更新 po 模板文件并把它们和翻译文件合并

6.43. Autoconf-2.69

Autoconf 软件包包含用于生成自动配置源代码的 shell 脚本的程序。

大致构建用时: 少于 0.1 SBU （包含测试大于 2.7 SBU）
所需磁盘空间: 17 MB

6.43.1. 安装 Autoconf

准备编译 Autoconf：

首先，修复一个 Perl 5.28 引入的问题。

```
sed '361 s/{/\\{/' -i bin/autoscan.in
```

准备编译 Autoconf：

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

编译软件包：

```
make
```

由于 bash-5 和 libtool-2.4.3 的原因，测试套件当前无法使用。如果要强行测试，输入：

```
make check
```

安装软件包：

```
make install
```

6.43.2. Autoconf 软件包内容

安装的程序: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate, 和 ifnames
安装目录: /usr/share/autoconf

简要介绍

autoconf	生成能为多种类 Unix 系统自动配置软件源码包的 shell 脚本；它生成的配置脚本是独立的——运行脚本不需要 autoconf 程序
autoheader	一个能生成给配置脚本使用的 C #define 语句模板文件的工具
autom4te	M4 宏处理器的封装
autoreconf	对 autoconf 和 automake 模板文件作了更改后能以正确顺序自动运行 autoconf，autoheader，aclocal，automake，gettextize，以及 libtoolize 以节省时间
autoscan	为软件包帮助生成 `configure.in` 文件；它检查目录树中的源文件，查找常见移植问题，生成作为软件包的初步 `configure.scan` 文件的 `configure.in` 文件
autoupdate	更改 `configure.in` 文件，仍然通过旧名称调用 autoconf 宏来使用当前宏名称
ifnames	帮助为软件包写 `configure.in` 文件；打印软件包在 C 预处理器中使用的标识符[如果已经设置软件包具有某些可移植性，该程序能帮助决定需要检查哪些 configure。还能填充 `configure.in` 生成的 autoscan 文件中的空格。]

6.44. Automake-1.16.1

软件包包含了生成可与 Autoconf 一同使用的 Makefile 的程序。

大致构建用时: 少于 0.1 SBU （包含测试大于 6.9 SBU）
所需磁盘空间: 107 MB

6.44.1. Automake 的安装

准备编译 Automake：

```
./configure --prefix=/usr --docdir=/usr/share/doc/automake-1.16.1
```

编译软件包：

```
make
```

因为各个测试之间存在内部延时，故建议就算是在单核处理器的设备上，也使用 -j4 编译选项加速测试过程。用以下命令测试结果：

```
make -j4 check
```

已知 LFS 环境中 subobj.sh 测试会失败。

安装软件包：

```
make install
```

6.44.2. 关于 Automake 软件包内容

安装的程序: aclocal, aclocal-1.16 (到 aclocal 的硬链接), automake, 和 automake-1.16 (到 automake 的硬链接)
安装的目录: /usr/share/aclocal-1.16, /usr/share/automake-1.16, 和 /usr/share/doc/automake-1.16.1

简要介绍

aclocal	基于 `aclocal.m4` 文件的内容生成 `configure.in` 文件
aclocal-1.16	到 aclocal 的硬链接
automake	一个从 `Makefile.in` 文件自动生成 `Makefile.am` 文件的工具 [要生成一个软件包里所有的 `Makefile.in` 文件，在最上层的目录运行这个程序。通过扫描 `configure.in` 文件，它能自动找到每个对应的 `Makefile.am` 文件，并生成对应的 `Makefile.in` 文件。]
automake-1.16	到 automake 的硬链接

6.45. Xz-5.2.4

Xz 软件包包含用于压缩和解压文件的程序。它提供 lzma 和更新的 xz 压缩格式功能。和传统的 gzip 或 bzip2 命令相比，用 xz 压缩文本文件能获得更好的压缩率。

大致构建用时: 0.2 SBU
所需磁盘空间: 16 MB

6.45.1. 安装 Xz

准备编译 Xz：

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/xz-5.2.4
```

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包并确保所需的文件都在正确目录中：

```
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
```

6.45.2. Xz 软件包内容

安装的程序: lzcat (链接到 xz), lzcmp (链接到 xzdiff), lzdiff (链接到 xzdiff), lzegrep (链接到 xzgrep), lzfgrep (链接到 xzgrep), lzgrep (链接到 xzgrep), lzless (链接到 xzless), lzma (链接到 xz), lzmadec, lzmainfo, lzmore (链接到 xzmore), unlzma (链接到 xz), unxz (链接到 xz), xz, xzcat (链接到 xz), xzcmp (链接到 xzdiff), xzdec, xzdiff, xzegrep (链接到 xzgrep), xzfgrep (链接到 xzgrep), xzgrep, xzless, 和 xzmore
安装的库: liblzma.so
安装的目录: /usr/include/lzma 和 /usr/share/doc/xz-5.2.4

简要介绍

lzcat	解压标准输出
lzcmp	对 LZMA 压缩文件运行 cmp 命令
lzdiff	对 LZMA 压缩文件运行 diff 命令
lzegrep	对 LZMA 压缩文件运行 egrep 命令
lzfgrep	对 LZMA 压缩文件运行 fgrep 命令
lzgrep	对 LZMA 压缩文件运行 grep 命令
lzless	对 LZMA 压缩文件运行 less 命令
lzma	用 LZMA 格式压缩或解压文件
lzmadec	用于 LZMA 压缩文件的轻便解码器
lzmainfo	显示存储在 LZMA 压缩文件头部的信息
lzmore	对 LZMA 压缩文件运行 more 命令
unlzma	用 LZMA 格式解压文件
unxz	用 XZ 格式解压文件
xz	用 XZ 格式压缩或解压文件
xzcat	解压到标准输出
xzcmp	对 XZ 压缩文件运行 cmp 命令
xzdec	用于 XZ 压缩文件的轻便解码器

xzdiff	对 XZ 压缩文件运行 diff 命令
xzegrep	对 XZ 压缩文件运行 egrep 命令
xzfgrep	对 XZ 压缩文件运行 fgrep 命令
xzgrep	对 XZ 压缩文件运行 grep 命令
xzless	对 XZ 压缩文件运行 less 命令
xzmore	对 XZ 压缩文件运行 more 命令
liblzma	用 Lempel-Ziv-Markov 链算法实现无损块排序数据压缩的库

6.46. Kmod-26

Kmod 软件包包含用于加载内核模块的库和工具

大致构建用时: 0.1 SBU
所需磁盘空间: 13 MB

6.46.1. 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
这些选项使 Kmod 能处理压缩的内核模块。

--with-rootlibdir=/lib
该选项确保和不同库相关的文件放置到正确的目录。

编译软件包：

```
make
```

这个软件包没有附带可在 LFS chroot 环境中运行测试套件。至少需要 git 程序并进行一些测试保证不会在 git 仓库外运行。

安装软件包并创建符号链接使兼容 Module-Init-Tools(之前处理 Linux 内核模块的软件包)：

```
make install

for target in depmod insmod lsmod modinfo modprobe rmmod; do
  ln -sfv ../bin/kmod /sbin/$target
done

ln -sfv kmod /bin/lsmod
```

6.46.2. Kmod 软件包内容

安装的程序: depmod (链接到 kmod), insmod (链接到 kmod), kmod, lsmod (链接到 kmod), modinfo (链接到 kmod), modprobe (链接到 kmod), 和 rmmod (链接到 kmod)
安装的库: libkmod.so

简要介绍

depmod	基于从已有的模块集上发现的符号创建依赖文件；modprobe 用该依赖文件自动加载所需模块
insmod	在运行的内核上安装可加载模块
kmod	加载或卸载内核模块
lsmod	列出当前已加载模块
modinfo	检查和内核模块相关联的目标文件并显示搜索到的任何信息
modprobe	用 depmod 创建的依赖文件自动加载相关模块
rmmod	从运行中的内核卸载模块
libkmod	其它程序使用该库加载或卸载内核模块

6.47. Gettext-0.19.8.1

Gettext 软件包包含用于国际化和本土化的工具。这允许用 NLS（Native Language Support，本地语言支持）编译程序，使得能以用户的本地语言输出信息。

大致构建用时: 2.0 SBU
所需磁盘空间: 210 MB

6.47.1. 安装 Gettext

首先，抑制两个 test-lock 的调用在某些机器上发生无限循环的问题：

```
sed -i '/^TESTS =/d' gettext-runtime/tests/Makefile.in &&
sed -i 's/test-lock..EXEEXT.//' gettext-tools/gnulib-tests/Makefile.in
```

然后修正配置文件：

```
sed -e '/AppData/{N;N;p;s/\.appdata\./.metainfo./}' \
    -i gettext-tools/its/appdata.loc
```

准备编译 Gettext：

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/gettext-0.19.8.1
```

编译软件包：

```
make
```

用以下命令测试结果（需要较长一段时间，大概 3 SBUs）：

```
make check
```

安装软件包：

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

6.47.2. Gettext 软件包内容

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, 和 xgettext
安装的库: libasprintf.so, libgettextlib.so, libgettextpo.so, libgettextsrc.so, 和 preloadable_libintl.so
安装的目录: /usr/lib/gettext, /usr/share/doc/gettext-0.19.8.1, 和 /usr/share/gettext

简要介绍

autopoint	复制标准 Gettext 基础文件到源码包
envsubst	shell 格式字符串的替代环境变量
gettext	通过查看信息目录中的转换将原来语言信息转换为用户语言
gettext.sh	主要作为 gettext 的一个 shell 函数库
gettextize	复制所有标准 Gettext 文件到指定软件包的顶层目录以开始国际化
msgattrib	根据属性过滤翻译目录的信息并操作属性
msgcat	连接和合并给定 .po 文件
msgcmp	比较两个 .po 文件以检查两者是否包含相同的 msgid 字符串集合
msgcomm	查找给定 .po 文件共同的信息
msgconv	转换翻译目录到不同的字符编码
msgen	创建一个英语翻译目录
msgexec	对翻译目录的所有翻译运行命令
msgfilter	对翻译目录的所有翻译应用过滤器

msgfmt	从翻译目录生成一个二进制信息目录
msggrep	从翻译目录中抽取所有符合指定模式或属于特定源文件的信息
msginit	创建一个新的 .po 文件，根据用户环境中的值初始化元信息
msgmerge	合并两个原译到一个单独的文件
msgunfmt	反向编译一个二进制信息目录为原译文本
msguniq	统一重复翻译为一个翻译目录
ngettext	显示语法形式取决于多种母语翻译的文本信息的本地语言翻译
recode-sr-latin	对 Serbian 文本从 Cyrillic 重新编码为 Latin 脚本
xgettext	从指定源文件中抽取可翻译信息行用于生成第一个翻译模板
libasprintf	定义 autosprintf 类，该类使 C 格式化输出在 C++程序中能和 <string> 字符串以及 <iostream> 流一起使用
libgettextlib	包括多种 Gettext 程序使用的常用例程的私有库；并不用于一般用途
libgettextpo	用来写处理 .po 文件的特殊程序；当 Gettext 附带的标准应用（例如 msgcomm，msgcmp，msgattrib, 以及 msgen）不足够时会使用这个库
libgettextsrc	包括多种 Gettext 程序使用的常用例程的私有库；并不用于一般用途
preloadable_libintl	LD_PRELOAD 帮助 libintl 记录未翻译信息时使用的库

6.48. Libelf 源自 Elfutils-0.176

Libelf 是处理 ELF（可执行与可链接格式）文件的库。

大致构建用时: 1.3 SBU
所需磁盘空间: 105 MB

6.48.1. 安装 Libelf

Libelf 是软件包 elfutils-0.176 种的一部分。使用 elfutils-0.176.tar.bz2 作为源码包。

编译 Libelf 前的准备：

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

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

仅安装 Libelf：

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

6.48.2. Libelf 软件包的内容

安装的库: libelf.so

6.49. Libffi-3.2.1

Libffi 库为各种调用规范提供了一个可移植的，高级编程接口。允许程序员在运行时，通过调用接口描述调用任意指定函数。

大致构建用时: 0.3 SBU
所需磁盘空间: 7.6 MB

6.49.1. 安装 Libffi

> **Note**
>
> Libffi 与 GMP 相似，构建时会根据使用的处理器优化。如果需要构建的是另一个系统，设定 CFLAGS 和 CXXFLAGS 为你的架构指定成通用构建。如果不这样做，所有指向 libffi 的链接将触发非法操作错误。

修改 Makefile 将头文件安装到标准的 `/usr/include` 目录，而非 `/usr/lib/libffi-3.2.1/include`。

```
sed -e '/^includesdir/ s/$(libdir).*$/$(includedir)/' \
    -i include/Makefile.in

sed -e '/^includedir/ s/=.*$/=@includedir@/' \
    -e 's/^Cflags: -I${includedir}/Cflags:/' \
    -i libffi.pc.in
```

编译 libffi 前的准备：

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

配置选项的含义：

`--with-gcc-arch=native`
确保 GCC 为当前系统进行优化。如果没有指定则会进行猜测，这样生成的代码在某些系统上是不正确的。如果生成的代码想要从本机系统复制到功能较弱的系统，则应该以功能较弱的系统作为参数。有关替代系统类型的详细信息，请参阅 gcc 手册中的 x86 选项。

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包：

```
make install
```

6.49.2. Libffi 的软件包内容

安装的库: libffi.so

简要介绍

`libffi` 包含 libffi 的 API 函数。

6.50. OpenSSL-1.1.1a

OpenSSL 软件包包含管理工具和加密相关的库。其中提供的这些加密功能对于其他软件包，比方说 OpenSSH，email 应用和网页浏览器（访问 HTTPS 站点）来说，十分有用。

大致构建用时: 1.7 SBU
所需磁盘空间: 141 MB

6.50.1. 安装 OpenSSL

编译 OpenSSL 前的准备：

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

编译软件包：

```
make
```

查看测试结果，输入：

```
make test
```

安装软件包：

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

如果有需要，安装文档：

```
mv -v /usr/share/doc/openssl /usr/share/doc/openssl-1.1.1a
cp -vfr doc/* /usr/share/doc/openssl-1.1.1a
```

6.50.2. OpenSSL 软件包内容

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 和 /usr/share/doc/openssl-1.1.1a

简要介绍

c_rehash 用于扫描一个目录中的所有文件并为它们的 hash 值添加符号链接的 Perl 脚本。

openssl 一个从 shell 使用 OpenSSL 的加密库的各种加密功能的命令行工具。功能广泛，十分有用，参考 man 1 openssl。

`libcrypto.so` 实现了各种互联网标准中极大部分加密算法。该库提供的服务，被 OpenSSL 用于实现 SSL， TLS 和 S/MIME，并且它们还被用于实现 OpenSSH，OpenPGP，以及其他加密标准。

`libssl.so` 实现传输层安全（TLS v1）协议。提供丰富的 API，文档参见 man 3 ssl。

6.51. Python-3.7.2

软件包 Python 3 包含了 Python 的开发环境。对于面向对象编程，书写脚本，构建大型程序的原型，或者开发整个应用程序而言，非常有用。

大致构建用时: 1.0 SBU
所需磁盘空间: 392 MB

6.51.1. 安装 Python 3

编译 Python 前的准备：

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

配置选项的含义：

--with-system-expat
该参数用于启用 Expat 系统版本的链接。

--with-system-ffi
该参数用于启用 libffi 系统版本的链接。

--with-ensurepip=yes
该参数用于启用 pip 和 setuptools 打包程序的构建。

编译软件包：

```
make
```

测试套件需要 TK 和 X Windows 会话，直至 BLFS 中重新安装 Python 3 之前都执行不了。

安装软件包：

```
make install
chmod -v 755 /usr/lib/libpython3.7m.so
chmod -v 755 /usr/lib/libpython3.so
```

安装命令的含义：

chmod -v 755 /usr/lib/libpython3.{7m.,}so
修复库的权限问题，同其他库保持一致。

如果需要，安装预格式化好的文档：

```
install -v -dm755 /usr/share/doc/python-3.7.2/html

tar --strip-components=1  \
    --no-same-owner       \
    --no-same-permissions \
    -C /usr/share/doc/python-3.7.2/html \
    -xvf ../python-3.7.2-docs-html.tar.bz2
```

文档安装命令的含义：

`--no-same-owner` and --no-same-permissions
确保安装文件的归属和权限是正确的。没有这个选项的话，运行 tar 时会以上游创建者的身份安装软件包内的文件。

6.51.2. Python 3 软件包内容

安装的程序: 2to3, idle3, pydoc3, python3, python3-config, pyvenv
安装的库: libpython3.7m.so 和 libpython3.so
安装的目录: /usr/include/python3.7m, /usr/lib/python3 和 /usr/share/doc/python-3.7.2

简要介绍

2to3 一个用于读取 Python 2.x 源代码并实施一系列的修复，将其转化称有效的 Python 3.x 代码的 Python 程序。

idle3	一个用于打开 Python 自带的 GUI 编辑器的封装脚本。为了让该脚本能运行，你必须在安装 Python 前先安装 Tk，这样 Tkinter Python 模块才会构建。
pydoc3	Python 的文档工具。
python3	一种解释性的，交互式的，面向对象的编程语言。
pyvenv	在一个或多个目标目录中，创建虚拟 Python 环境。

6.52. Ninja-1.9.0

Ninja 是一个专注于速度的小型构件系统。

大致构建用时: 0.2 SBU
所需磁盘空间: 65 MB

6.52.1. 安装 Ninja

在运行时，ninja 通常会并行最大数量的进程。默认值是系统的核心数乘以二。有些时候会导致 CPU 过热，或者内存容量不足。如果是命令行运行，通过传递 -jN 参数可以限制并行的进程数，但是有些软件包虽然潜入了 ninja 的执行却不会传递 -j 参数。

通过使用下方 可选 过程，让用户能够通过环境变量 NINJAJOBS 来限制并行进程的数量。例如， 设定：

```
export NINJAJOBS=4
```

将限制 ninja 最多仅四个进程并行。

如果需要，运行以下命令以添加使用环境变量 NINJAJOBS 的功能：

```
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
```

构建参数的含义：

`--bootstrap`
这个参数强迫 ninja 重新构建自身以适应当前系统。

查看测试结果，输入：

```
python3 configure.py
./ninja ninja_test
./ninja_test --gtest_filter=-SubprocessTest.SetWithLots
```

安装软件包：

```
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
```

6.52.2. Ninja 软件包内容

安装的程序: ninja

简要介绍

ninja Ninja 构件系统。

6.53. Meson-0.49.2

Meson 是一个开源代码构建系统，不仅速度非常快，而且更重要的是对用户极其友好。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 24 MB

6.53.1. 安装 Meson

通过下列命令编译 Meson：

```
python3 setup.py build
```

这个软件包还没有测试套件。

安装软件包：

```
python3 setup.py install --root=dest
cp -rv dest/* /
```

安装参数的含义：

`--root=dest`
默认情况下执行命令 python3 setup.py install 会将所有文件（比如 man 手册）安装到 Python Eggs。指定根路径则 setup.py会将这些文件安装到标准层次目录。然后只需要复制层次目录，文件即处于标准位置。

6.53.2. Meson 软件包内容

安装的程序: meson, mesonconf, mesonintrospect, mesontest, and wraptool
安装的目录: /usr/lib/python3.7/site-packages/meson-0.49.2-py3.7.egg

简要介绍

meson	高生产率构建系统
mesonconf	Meson 构建系统配置工具
mesonintrospect	Meson 构建系统信息提取工具
mesontest	Meson 构建系统测试工具
wraptool	源码依赖下载器

6.54. Coreutils-8.30

Coreutils 软件包包含用于显示和设置基本系统特性的工具。

大致构建用时: 2.6 SBU
所需磁盘空间: 190 MB

6.54.1. 安装 Coreutils

POSIX 要求 Coreutils 中的程序即使在多字节语言环境也能正确识别字符边界。下面的补丁修复这个不兼容性以及其它一些和国际化相关的错误。

```
patch -Np1 -i ../coreutils-8.30-i18n-1.patch
```

> **Note**
>
> 之前在这个补丁中发现了很多错误。当向 Coreutils 维护者报告新错误的时候，请先检查没有该补丁是否可以重现该错误。

抑制测试在某些机器上可能出现的无线循环：

```
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
: 此命令更新已有的配置文件，以使其和 automake 最新生成的一致。

FORCE_UNSAFE_CONFIGURE=1
: 该环境变量允许以 root 用户权限编译软件包。

--enable-no-install-program=kill,uptime
: 该选项的目的是防止 Coreutils 安装其它软件包后面会安装的二进制包。

编译软件包：

```
FORCE_UNSAFE_CONFIGURE=1 make
```

如果不运行测试套件的话跳到「安装软件包」

现在可以运行测试套件了。首先，运行需要以 root 用户运行的测试：

```
make NON_ROOT_USERNAME=nobody check-root
```

我们会以 nobody 用户运行剩下的测试。但是，一些测试要求用户属于多个组。为了不跳过这些测试，我们会添加一个临时的组并添加用户 nobody 作为它的成员：

```
echo "dummy:x:1000:nobody" >> /etc/group
```

修复一些权限问题以便非 root 用户可以编译和运行测试：

```
chown -Rv nobody .
```

现在运行测试。确保 **su** 环境中的 PATH 环境变量包含了 /tools/bin。

```
su nobody -s /bin/bash \
          -c "PATH=$PATH make RUN_EXPENSIVE_TESTS=yes check"
```

已知测试程序 test-getlogin 在部分构建的系统环境（如 chroot 环境）中会失败，章节结束后运行便会通过。已知测试程序 tty.sh 也会失败。

移除临时组：

```
sed -i '/dummy/d' /etc/group
```

安装软件包：

```
make install
```

移动程序到 FHS 指定的位置：

```
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\"/1 /usr/share/man/man8/chroot.8
```

```
mv -v /usr/bin/{head,nice,sleep,touch} /bin
```

6.54.2. Coreutils 软件包内容

安装的程序: [, base32, base64, basename, 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, and yes
安装的库: libstdbuf.so
安装目录: /usr/libexec/coreutils

简要介绍

base32	根据 base32（RFC 4648）规范编码和解码数据
base64	根据 base64（RFC 4648）规范编码和解码数据
basename	从文件名称中抽取路径和后缀
cat	将文件连接到标准输出
chcon	改变文件和目录的安全上下文
chgrp	更改文件和目录的组所有者
chmod	更改每个文件为指定模式的权限；模式可以是要实现更改的符号表示 或者表示新权限的十进制数字
chown	更改文件和目录的用户和/或组所有者
chroot	使用指定目录作为 / 目录运行命令
cksum	输出指定文件的循环冗余检验（CRC）校验码和字节数目
comm	比较两个排序后的文件，以三列输出不用的行和相同的行
cp	复制文件
csplit	将指定文件分割为几个新的文件，根据指定的模式或者行数分割并 输出每个新文件的字节数
cut	根据指定的域或位置，选择输出行的组成
date	根据指定格式显示当前时间，或设置系统日期
dd	用给定的块大小和数目复制文件，同时可以进行一些转换
df	报告所有挂载的文件系统中可用（和已用）磁盘空间，或只是 含有指定文件的文件系统
dir	列出指定目的的内容（和 ls 命令相同）
dircolors	输出设置 LS_COLOR 环境变量的命令，用于更改 ls 使用的颜色模式
dirname	从一个文件名称中抽取非目录后缀
du	报告当前目录使用的磁盘空间，根据指定的每个文件夹（包括子文件夹） 或每个指定的文件
echo	显示给定的字符串
env	在更改后的环境中运行命令
expand	把 tab 键转换为空格

expr	计算表达式
factor	输出所有指定整数的质数因子
false	什么都不做；总是以指示失败的状态码退出
fmt	重新格式化给定文件中的段落
fold	折叠指定文件的行
groups	报告一个用户的组成员
head	输出指定文件的前十行（或指定数目行数）
hostid	报告主机的数字标识符（以十六进制）
id	报告当前用户或指定用户的有效 用户 ID、组 ID 以及组成员
install	复制文件的同时设置权限模式，如果可以的话包括用户和组
join	从两个单独的文件中连接有相同域的行
link	用指定的名称创建到一个文件的硬链接
ln	在文件之间建立硬链接或软（符号）链接
logname	报告当前用户的登录名
ls	列出给定目录的内容
md5sum	报告或检查消息摘要 5（MD5） 校验码
mkdir	用指定的名称新建目录
mkfifo	用指定的名称在 UNIX 中创建先进先出（FIFO） 的「命名管道」
mknod	用指定的名称创建设备结点；设备结点是一个特殊字符文件、特殊块文件或先进先出
mktemp	以安全方式新建临时文件；在脚本中使用
mv	移动或重命名文件或目录
nice	以更改后的调度优先级运行程序
nl	标记指定文件的行号
nohup	以不能被挂起方式运行命令，输出重定向到一个日志文件
nproc	输出进程可用的处理单元数目
numfmt	转换数字为人可读字符串或者相反
od	以十进制或其他格式转储文件
paste	合并指定文件，用 tab 字符分隔，以行并列方式连续合并
pathchk	检查文件名是否可用
pinky	轻量级的 finger 客户端；报告指定用户的信息
pr	对文件进行分页分行用于打印
printenv	输出环境
printf	根据指定格式打印指定参数，类似于 C 语言的 printf 函数
ptx	用文中的每个关键字，根据文件的内容中建立重排索引
pwd	报告当前工作目录的名称
readlink	报告指定符号链接的值
realpath	打印解释后路径
rm	删除文件或目录
rmdir	如果目录为空则删除
runcon	以指定安全上下文运行命令
seq	用指定的范围和增长步长输出一序列数字
sha1sum	打印或检查 160-bit 安全哈希算法1（SHA1） 校验码
sha224sum	打印或检查 224-bit 安全哈希算法校验码
sha256sum	打印或检查 256-bit 安全哈希算法校验码
sha384sum	打印或检查 384-bit 安全哈希算法校验码
sha512sum	打印或检查 512-bit 安全哈希算法校验码

shred	用复杂形式多次重写指定文件，使得难以恢复其中的数据
shuf	打乱文本行
sleep	暂停指定时间
sort	排序给定文件的行
split	根据大小或行数分割文件为多个块
stat	显示文件或文件系统状态
stdbuf	用改变后的缓冲操作在标准流上运行命令
stty	设置或报告终端行设置
sum	打印指定文件的校验码和块数目
sync	清空文件系统缓存；强制更改块到磁盘并更新超级块
tac	反向输出给定文件
tail	输出每个给定文件的最后十行（或给定数目的行）
tee	从标准输入读入并写出到标准输出和指定文件
test	比较值并检查文件类型
timeout	有限时间内运行命令
touch	更改文件时间戳，设置指定文件的访问和修改时间为当前时间； 如果文件不存在则创建空文件
tr	从标准输入转换、压缩并删除指定字符
true	不做任何事情，总是成功；总是以表示成功的状态码退出
truncate	压缩或扩展文件到特定大小
tsort	进行拓扑排序；根据指定文件的部分排序写出完全有序列表
tty	报告链接到标准输入的终端文件名称
uname	报告系统信息
unexpand	转换空格为 tab 键
uniq	忽略所有除非出现连续相同的行
unlink	移除指定文件
users	报告当前登录的用户名
vdir	和 ls -l 相同
wc	报告给定文件的行数、单词数和字节数，以及给定多个文件时总的行数
who	报告谁登录了
whoami	报告和当前有效用户 ID 关联的用户名
yes	重复输出「y」或指定的字符串直到被杀死
`libstdbuf`	stdbuf 使用的库

6.55. Check-0.12.0

Check 是 C 语言的单元测试框架。

大致构建用时: 0.1 SBU （包含测试大于 3.0 SBU）
所需磁盘空间: 12 MB

6.55.1. 安装 Check

配置 Check 准备编译：

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

构建软件包：

```
make
```

现在编译完成了。运行 Check 的测试套件，输入以下命令：

```
make check
```

注意，Check 的测试套件可能会占用挺长（上至 4 SBU）的时间。

安装软件包，并修复脚本：

```
make install
sed -i '1 s/tools/usr/' /usr/bin/checkmk
```

6.55.2. Check 软件包内容

安装的程序: checkmk
安装的库: libcheck.{a,so}

简要介绍

checkmk 用于生成 C 语言单元测试的 Awk 脚本，这些用例可以配合 Check 单元测试框架使用
`libcheck.{a,so}` 包含允许测试程序调用 Check 的功能

6.56. Diffutils-3.7

Diffutils 软件包包含显示文件和目录差异的程序。

大致构建用时: 0.3 SBU
所需磁盘空间: 36 MB

6.56.1. 安装 Diffutils

准备编译 Diffutils：

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

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包：

```
make install
```

6.56.2. Diffutils 软件包内容

安装的程序: cmp, diff, diff3, and sdiff

简要介绍

cmp 比较两个文件并报告字节差异
diff 比较两个文件或目录并报告文件中的行差异
diff3 逐行比较三个文件
sdiff 比较两个文件并交互式输出结果

6.57. Gawk-4.2.1

Gawk 软件包包含用于操作文本文件的程序。

大致构建用时: 0.3 SBU
所需磁盘空间: 44 MB

6.57.1. 安装 Gawk

首先，确保哪些不需要的文件没有被安装：

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

准备编译 Gawk：

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

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包：

```
make install
```

如果需要的话，安装帮助文档：

```
mkdir -v /usr/share/doc/gawk-4.2.1
cp    -v doc/{awkforai.txt,*.{eps,pdf,jpg}} /usr/share/doc/gawk-4.2.1
```

6.57.2. Gawk 软件包内容

安装的程序: awk (链接到 gawk), gawk, 和 awk-4.2.1
安装的库: filefuncs.so, fnmatch.so, fork.so, inplace.so, ordchr.so, readdir.so, readfile.so, revoutput.so, revtwoway.so, rwarray.so, testext.so, and time.so
安装的目录: /usr/lib/gawk, /usr/libexec/awk, /usr/share/awk, 和 /usr/share/doc/gawk-4.2.1

简要介绍

awk	到 gawk 的链接
gawk	用于操作文本文件的程序；awk 的 GNU 实现
gawk-4.2.1	到 gawk 的硬链接

6.58. Findutils-4.6.0

Findutils 软件包包含查找文件的程序。这些程序提供递归搜索目录树、创建、管理以及搜索数据库（通常比递归式的 find 要快，但如果数据库最近没有更新的话结果不可靠）。

大致构建用时: 0.6 SBU
所需磁盘空间: 51 MB

6.58.1. 安装 Findutils

首先，抑制测试中可能在某些机器中出现的无限循环：

```
sed -i 's/test-lock..EXEEXT.//' tests/Makefile.in
```

然后，对应 glibc-2.28 或更高版本 glibc 的需求做一些修复：

```
sed -i 's/IO_ftrylockfile/IO_EOF_SEEN/' gl/lib/*.c
sed -i '/unistd/a #include <sys/sysmacros.h>' gl/lib/mountlist.c
echo "#define _IO_IN_BACKUP 0x100" >> gl/lib/stdio-impl.h
```

准备编译 Findutils：

```
./configure --prefix=/usr --localstatedir=/var/lib/locate
```

配置选项的含义：

--localstatedir
该选项改变 locate 数据库的位置为 FHS 兼容的 `/var/lib/locate`。

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包：

```
make install
```

一些 BLFS 及之上的软件包希望 find 程序在 `/bin`，因此确保位置正确：

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

6.58.2. Findutils 软件包内容

安装的程序: find, locate, updatedb, 和 xargs

简要介绍

find	查找指定目录树中匹配特定要求的文件
locate	搜索文件名称数据库并报告包含给定字符串或匹配给定模式的文件名称
oldfind	老版本的 find，使用一个不同的算法
updatedb	更新 locate 数据库； 它搜索整个文件系统（包括已挂载的其它文件系统，除非指定排除） 并把找到的每个文件名插入到数据库
xargs	对一系列文件运行给定命令

6.59. Groff-1.22.4

Groff 软件包包含用于处理和格式化文本的程序。

大致构建用时: 0.4 SBU
所需磁盘空间: 94 MB

6.59.1. 安装 Groff

Groff 希望环境变量 PAGE 包含默认的页面大小，对于美国的用户，为 *PAGE=letter*，对于其它地方，*PAGE=A4* 更合适。尽管在编译的时候配置了默认页面大小，后面通过 echo「A4」或「letter」到 `/etc/papersize` 文件仍然可以修改。

准备编译 Groff：

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

该软件不支持并行构建。编译软件包：

```
make -j1
```

该软件包没有测试套具。

安装软件包：

```
make install
```

6.59.2. Groff 软件包内容

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, 和 troff
安装的目录: /usr/lib/groff 和 /usr/share/doc/groff-1.22.4, /usr/share/groff

简要介绍

addftinfo	读 troff 字体文件并添加一些额外的 groff 系统使用的字体信息
afmtodit	创建用于和 groff 以及 grops 一起使用的字体文件
chem	Gorff 预处理生成化学结构图
eqn	编译嵌入了 troff 输入文件的方程的描述为 troff 能理解的命令
eqn2graph	转换 troff EQN （equation 方程）为裁剪图像
gdiffmk	标记 groff/nroff/troff 文件的差异
glilypond	将 lilypond 语言写成的乐谱转换至 groff语言
gperl	Groff 的预处理器，允许在 groff 中添加 perl 代码
gpinyin	Groff 的预处理器，允许在 groff 文件中添加类似欧洲语言的中文拼音。
grap2graph	转换 grap 图为裁剪位图图像
grn	用于 gremlin 文件的 groff 预处理器
grodvi	生成 TeX dvi 格式的 groff 驱动
groff	groff 文档格式化系统前端；一般运行 troff 程序和适合选定文件的后处理器
groffer	在 X 和 tty 终端显示 groff 文件以及 man 页面
grog	读文件并猜测打印文件需要的 groff 选项 `-e`，`-man`，`-me`，`-mm`，`-ms`，`-p`，`-s`,以及 `-t`，并报告包含这些选项的 groff 命令
grolbp	用于 Canon CAPSL 打印机（LBP-4 和 LBP-8 系列激光打印机） 的 groff 驱动
grolj4	生成适合于 HP LaserJet 4 打印机的 PCL5 格式输出的 groff 驱动
gropdf	翻译 GNU troff 的输出至 PDF
grops	转换 GNU troff 输出为 PostScript
grotty	转换 GNU troff 输出为适合于打字机设备的格式

hpftodit	从 HP标签字体规格文件创建和 groff -Tlj4 一起使用的字体文件
indxbib	用和 refer，lookbib, 以及 lkbib 一起使用的指定文件为文献数据库创建倒排索引
lkbib	搜索文献数据库中包含指定键的引用并报告找到的任何引用
lookbib	在标准错误中输出提示（除非标准输入不是终端），从标准输入读取包含一系列关键字的行，在指定的文件中搜索文献数据库中包含那些关键字的引用，在标准输出中打印找到的任何引用，循环这些过程直到输入结束
mmroff	groff 的简单预处理器
neqn	为美国标准信息交换码（ASCII） 输出格式化方程
nroff	用 nroff 模仿 groff 命令的脚本
pdfmom	groff 周围的封装，用于简化从带有 mom 宏的文件格式生成 PDF 文件的过程。
pdfroff	用 groff 创建 pdf 文档
pfbtops	转换 `.pfb` 格式中的 PostScript 字体为 ASCII
pic	编译嵌入了 troff 或 TeX 输入文件的图像的描述为 TeX 或 troff 能理解的命令
pic2graph	转换 PIC 图为裁剪图像
post-grohtml	转换 GNU troff 输出为 HTML
preconv	转换输入文件编码为 GNU troff 能理解的编码
pre-grohtml	转换 GNU troff 的输出为 HTML
refer	复制文件内容到标准输出，其中 .[和 .] 之间的行解释为引用，.R1 和 .R2 之间的行解释为如何处理引用的命令
roff2dvi	转化 roff 文件到 DVI 格式
roff2html	转换 roff 文件到 HTML 格式
roff2pdf	转换 roff 文件到 PDFs
roff2ps	转换 roff 文件为 ps 文件
roff2text	转换 roff 文件为文本文件
roff2x	转换 roff 文件到其它格式
soelim	读文件并用相应的 file 内容替换 .so file 格式的行
tbl	编译嵌入了 troff 输入文件的表的描述为 troff 能理解的命令
tfmtodit	创建和 groff -Tdvi 一起使用的字体文件
troff	和 Unix troff 高度兼容；通常应该使用 groff 命令调用，它也会以恰当的顺序和选项运行预处理器和后处理器

6.60. GRUB-2.02

GRUB 软件包包含多重启动管理器（GRand Unified Bootloader）。

大致构建用时: 0.6 SBU
所需磁盘空间: 147 MB

6.60.1. 安装 GRUB

准备编译 GRUB：

```
./configure --prefix=/usr          \
            --sbindir=/sbin        \
            --sysconfdir=/etc      \
            --disable-efiemu       \
            --disable-werror
```

新配置选项的含义：

`--disable-werror`
允许忽视有更新 Flex 版本提示的警告以完成构建。

`--disable-efiemu`
这些选项通过停用 LFS 不需要的功能和测试程序最小化构建。

编译软件包：

```
make
```

该软件包没有测试套件。

安装软件包：

```
make install
mv -v /etc/bash_completion.d/grub /usr/share/bash-completion/completions
```

会在 Section#8.4, “使用 GRUB 设置启动过程”介绍通过 GRUB 启动你的 LFS 系统。

6.60.2. GRUB 软件包内容

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, 和 grub-syslinux2cfg
安装的目录: /usr/lib/grub, /etc/grub.d, /usr/share/grub, 和 boot/grub (when grub-install is first run)

简要介绍

grub-bios-setup	grub-install 的帮助程序
grub-editenv	编辑环境块的工具
grub-file	检查 FILE 是否为指定的类型
grub-fstest	调试文件系统驱动的工具
grub-glue-efi	处理 ia32 和 amd64 的 EFI 镜像并按照 Apple 的格式将他们粘在一起
grub-install	在你的驱动器上安装 GRUB
grub-kbdcomp	转换 xkb 布局为 GRUB 可识别样式的脚本
grub-macbless	以 Mac 的风格保护 HFS 或 HFS+ 文件
grub-menulst2cfg	为和 GRUB 2 一起使用，转换引导装载程序（GRUB Legacy） `menu.lst` 为 `grub.cfg`
grub-mkconfig	生成 grub 配置文件
grub-mkimage	创建 GRUB 可启动镜像
grub-mklayout	生成 GRUB 键盘布局文件

grub-mknetdir	准备一个 GRUB 网络启动目录
grub-mkpasswd-pbkdf2	生成一个用于启动菜单的加密 PBKDF2 密码
grub-mkrelpath	生成相对于根目录的系统路径名称
grub-mkrescue	创建适用于软盘或 CDROM/DVD 的可启动 GRUB 镜像
grub-mkstandalone	生成一个单独镜像
grub-ofpathname	打印 GRUB 设备路径的帮助程序
grub-probe	对指定路径或设备检测设备信息
grub-reboot	只为下次启动设置默认 GRUB 启动选项
grub-render-label	为 Apple Mac 赋予 Apple 的 .disk_label
grub-script-check	检查 GRUB 配置脚本是否有语法错误
grub-set-default	为 GRUB 设置默认启动选项
grub-sparc64-setup	grub-setup 的帮助程序
grub-syslinux2cfg	将 syslinux 的配置文件转换至 grub.cfg 格式

6.61. Less-530

Less 软件包包含一个文本文件查看器。

大致构建用时: 少于 0.1 SBU
所需磁盘空间: 3.9 MB

6.61.1. 安装 Less

准备编译 Less：

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

配置选项的含义：

`--sysconfdir=/etc`
该选项告诉软件包创建的程序在 `/etc` 中查找配置文件。

编译软件包：

```
make
```

该软件包没有测试套件。

安装软件包：

```
make install
```

6.61.2. Less 软件包内容

安装的程序: less, lessecho, 和 lesskey

简要介绍

less	文件查看器或分页器；显示指定文件的内容，允许用户滚动、查找字符串以及跳转到标记。
lessecho	在 Unix 系统文件名中扩展元字符，例如 * 和 ?
lesskey	用于指定绑定到 less 的键

6.62. Gzip-1.10

Gzip 软件包包含用于压缩和解压文件的程序。

大致构建用时: 0.1 SBU
所需磁盘空间: 20 MB

6.62.1. 安装 Gzip

准备编译 Gzip：

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

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

已知有两个测试在 LFS 环境中会失败：help-version 和 zmore。

安装软件包：

```
make install
```

移动一些需要放在根文件系统的程序：

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

6.62.2. Gzip 软件包内容

安装的程序: gunzip, gzexe, gzip, uncompress (到 gunzip 的硬链接), zcat, zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore, 和 znew

简要介绍

gunzip	解压 gzip 压缩的文件
gzexe	创建自解压可执行文件
gzip	用 Lempel-Ziv(LZ77) 编码压缩指定文件
uncompress	解压压缩文件
zcat	解压指定 gzip 压缩的文件到标准输出
zcmp	对 gzip 压缩的文件运行 cmp 命令
zdiff	对 gzip 压缩的文件运行 diff 命令
zegrep	对 gzip 压缩的文件运行 egrep 命令
zfgrep	对 gzip 压缩的文件运行 fgrep 命令
zforce	强制所有指定的 gzip 压缩文件的扩展名为 `.gz`，这样 gzip 就不会对它们再次压缩；这在文件传输中文件名被截断时非常有用
zgrep	对 gzip 压缩的文件运行 grep 命令
zless	对 gzip 压缩的文件运行 less 命令
zmore	对 gzip 压缩的文件运行 more 命令
znew	从 compress 格式到 gzip 格式重新压缩文件format——`.Z` 到 `.gz`

6.63. IPRoute2-4.20.0

IPRoute2 软件包包含基于 IPV4 网络的基本和高级程序。

大致构建用时: 0.2 SBU
所需磁盘空间: 13 MB

6.63.1. 安装 IPRoute2

此软件包中包含的 arpd 因为依赖于 Berkeley DB，但是此软件并没有包含于 LFS 中，所以将不会进行编译。但是 arpd 的目录依旧会被安装。运行以下的命令来阻止这一动作。如果需要 arpd 的二进制文件，请查看 BLFS Book 的网页 http://www.linuxfromscratch.org/blfs/view/8.4/server/databases.html#db 以了解编译 Berkeley DB 都需要哪些指令。

```
sed -i /ARPD/d Makefile
rm -fv man/man8/arpd.8
```

此外，还需要禁用两个模块，它依赖于 http://www.linuxfromscratch.org/blfs/view/8.4/postlfs/iptables.html.

```
sed -i 's/.m_ipt.o//' tc/Makefile
```

编译软件包：

```
make
```

此软件包不包含可用的测试套件。

安装软件包：

```
make DOCDIR=/usr/share/doc/iproute2-4.20.0 install
```

6.63.2. IPRoute2 软件包内容

安装的程序: bridge, ctstat (链接到 lnstat), genl, ifcfg, ifstat, ip, lnstat, nstat, routef, routel, rtacct, rtmon, rtpr, rtstat (链接到 lnstat), ss, 和 tc
安装的目录: /etc/iproute2, /usr/lib/tc, 和 /usr/share/doc/iproute2-4.20.0,

简要介绍

bridge 配置网桥

ctstat 连接状态工具

genl 通用 netlink 多用途前端

ifcfg ip 命令的 shell 脚本封装 [注意它需要 iputils 软件包中的 arping 和 rdisk 程序，可以在 http://www.skbuff.net/iputils/ 找到 iputils 软件包。]

ifstat 显示接口统计信息，包括接口发送和接收的包的数目

ip 主要的可执行程序。它有多种不同功能：
ip link ***<device>*** 允许用户查看设备状态或更改
ip addr 允许用户查看地址和属性、增加新地址、删除旧地址
ip neighbor 允许用户查看邻居和它们的特性、增加新邻居、删除旧邻居
ip rule 允许用户查看路由策略并更改
ip route 允许用户查看路由表并更改路由表规则
ip tunnel 允许用户查看 IP 隧道及其特性、并进行更改
ip maddr 允许用户查看多播地址及其特性、并进行更改
ip mroute 允许用户设置、更改或删除多播路由
ip monitor 允许用户持续监视设置、地址和路由状态

lnstat 提供 Linux 网络统计信息；是更通用、功能更完备的替代旧 rtstat 的程序

nstat 显示网络统计信息

routef ip route 的组件。用于清空路由表

routel ip route 的组件。用于列出路由表

rtacct 显示 `/proc/net/rt_acct` 的内容

rtmon 路由监视工具

rtpr	转换 ip -o 输出为可读形式
rtstat	路由状态工具
ss	类似于 netstat 命令；显示活动连接
tc	拥塞控制可执行程序；用于实现服务质量（Quality Of Service,QOS）和服务等级（Class Of Service,COS） tc qdisc 允许用户设置排队规则 tc class 允许用户基于排队规则调度策略设置等级 tc estimator 允许用户估计到一个网络的网络流量 tc filter 允许用户设置 QOS/COS 包过滤 tc policy 允许用户设置 QOS/COS 策略

6.64. Kbd-2.0.4

Kbd 软件包包含键表文件、控制台字体和键盘工具。

大致构建用时: 0.1 SBU
所需磁盘空间: 30 MB

6.64.1. 安装 Kbd

在 Kbd 软件包中退格键（Backspace）和删除键（Delete）的行为和键映射并不一致。下面的补丁修复了 i386 键映射中的这个问题：

```
patch -Np1 -i ../kbd-2.0.4-backspace-1.patch
```

打补丁后，退格键生成编码为 127 的字符，删除键会生成一个著名的转义序列。

移除冗余的 resizecons 程序（它要求功能不全的 svglib 提供视频模式文件——用于正常使用 setfont 设置控制台字体大小）以及帮助手册。

```
sed -i 's/\(RESIZECONS_PROGS=\)yes/\1no/g' configure
sed -i 's/resizecons.8 //' docs/man/man8/Makefile.in
```

准备编译 Kbd：

```
PKG_CONFIG_PATH=/tools/lib/pkgconfig ./configure --prefix=/usr --disable-vlock
```

配置选项的含义：

--disable-vlock
该选项防止编译 vlock 工具，因为它要求 chroot 环境中不可用的 PAM 库。

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包：

```
make install
```

> **Note**
>
> 由于通常使用的 CP1251 键映射假设使用 ISO-8859-5 编码，Kbd 软件包不能为某些语言（例如，白俄罗斯） 提供可用的键映射。使用这样的语言需要单独下载能工作的键映射。

如果需要的话，安装帮助文档：

```
mkdir -v       /usr/share/doc/kbd-2.0.4
cp -R -v docs/doc/* /usr/share/doc/kbd-2.0.4
```

6.64.2. Kbd 软件包内容

安装的程序: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, kbdinfo, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (链接到 psfxtable), psfgettable (链接到 psfxtable), psfstriptable (链接到 psfxtable), psfxtable, setfont, setkeycodes, setleds, setmetamode, setvtrgb, showconsolefont, showkey, unicode_start, 和 unicode_stop
安装目录: /usr/share/consolefonts, /usr/share/consoletrans, /usr/share/keymaps, /usr/share/doc/kbd-2.0.4, 和 /usr/share/unimaps

简要介绍

chvt 更改前台虚拟终端
deallocvt 重新分配未使用的虚拟终端

dumpkeys	转储键盘转换表
fgconsole	输出活动虚拟终端的数目
getkeycodes	输出内核扫描码到键码的映射表
kbdinfo	获取关于某个终端的状态信息
kbd_mode	报告或设置键盘模式
kbdrate	设置键盘重复和延迟速度
loadkeys	加载键盘转换表
loadunimap	加载内核 Unicode 到字体映射表
mapscrn	用于加载用户定义的输出字符映射表到控制台驱动的过时程序；现在通过 setfont 完成
openvt	在一个新的虚拟终端（VT）启动程序
psfaddtable	添加 Unicode 字符表到控制台字体
psfgettable	从控制台字体抽取嵌入的 Unicode 字符表
psfstriptable	从控制台字体移除嵌入的 Unicode 字符表
psfxtable	为控制台字体处理 Unicode 字符表
setfont	更改控制台的增强图形适配器（Enhanced Graphic Adapter,EGA）和视频图形阵列（Video Graphics Array,VGA）的字体
setkeycodes	加载内核扫描码到键码映射表条目；键盘上有异常键时非常有用
setleds	设置键盘标记和 LED 灯
setmetamode	定义键盘元键处理
setvtrgb	设置所有虚拟终端中的控制台颜色映射
showconsolefont	显示当前 EGA/VGA 控制台屏幕字体
showkey	报告键盘上按键的扫描码、键码以及 ASCII 码
unicode_start	设置键盘和控制台为 UNICODE 模式[别用该程序，除非你的键映射文件是 ISO-8859-1 编码。对于其它编码，该工具会输出错误结果。]
unicode_stop	从 UNICODE 模式恢复键盘和控制台为原来模式

6.65. Libpipeline-1.5.1

Libpipeline 软件包包含以灵活方便方式管理管道和子进程的库。

大致构建用时: 0.1 SBU
所需磁盘空间: 9.0 MB

6.65.1. 安装 Libpipeline

准备编译 Libpipeline：

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

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包：

```
make install
```

6.65.2. Libpipeline 软件包内容

安装的库: libpipeline.so

简要介绍

`libpipeline` 该库用于在子进程之间安全地建立管道

6.66. Make-4.2.1

Make 软件包包含一个用于编译软件的程序。

大致构建用时: 0.5 SBU
所需磁盘空间: 13 MB

6.66.1. 安装 Make

再来一次，解决由 glibc-2.27 或更新版本 glibc 导致的错误：

```
sed -i '211,217 d; 219,229 d; 232 d' glob/glob.c
```

准备编译 Make：

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

编译软件包：

```
make
```

测试套件需要知道支持 perl 文件的位置。我们使用一个环境变量来达成这个目的。用以下命令测试结果：

```
make PERL5LIB=$PWD/tests/ check
```

安装软件包：

```
make install
```

6.66.2. Make 软件包内容

安装的程序: make

简要介绍

make 自动检测软件包的哪些部分需要（重新）编译，然后运行相应命令

6.67. Patch-2.7.6

Patch 软件包包含一个通过打「补丁」创建或修改文件的程序，补丁文件通常由 diff 程序生成。

大致构建用时: 0.2 SBU
所需磁盘空间: 13 MB

6.67.1. 安装 Patch

准备编译 Patch：

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

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包：

```
make install
```

6.67.2. Patch 软件包内容

安装的程序: patch

简要介绍

patch 根据补丁文件修改文件[补丁文件通常是由 diff 程序产生的差异列表。通过对源文件应用这些差异生成打补丁后的版本。]

6.68. Man-DB-2.8.5

Man-DB 软件包包含用于查找和查看 man 页面的程序。

大致构建用时: 0.3 SBU
所需磁盘空间: 36 MB

6.68.1. 安装 Man-DB

准备编译 Man-DB：

```
./configure --prefix=/usr                        \
            --docdir=/usr/share/doc/man-db-2.8.5 \
            --sysconfdir=/etc                    \
            --disable-setuid                     \
            --enable-cache-owner=bin             \
            --with-browser=/usr/bin/lynx         \
            --with-vgrind=/usr/bin/vgrind        \
            --with-grap=/usr/bin/grap
```

配置选项的含义：

`--disable-setuid`
为用户 man 禁止 man.

`--enable-cache-owner=bin`
用于将系统范围内的 cache 文件的拥有者设置为 bin 用户。

`--with-...`
这些参数用于设置一些默认程序。lynx 是一个基于文本的网络浏览器（查看 BLFS 获取安装指令），vgrind 将程序源码转换为 Groff 输入，grap 在 Groof 文档排版图中非常有用。查看手册页通常并不需要 vgrind 和 grap 程序。它们并不是 LFS 或 BLFS 的一部分，但是如果需要的话你自己应该能够在完成 LFS 之后安装它们。

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包：

```
make install
```

6.68.2. LFS 中的非英语手册页

下面的表格显示了 Man-DB 假定手册页安装到 `/usr/share/man/<ll>` 会使用的字符编码。除此之外，Man-DB 能正确判断安装到该目录的手册页是否采用 UTF-8 编码。

Table#6.1.#传统 8 位手册页预期字符编码

语言 (缩写)	编码	语言 (缩写)	编码
Danish (da)	ISO-8859-1	Croatian (hr)	ISO-8859-2
German (de)	ISO-8859-1	Hungarian (hu)	ISO-8859-2
English (en)	ISO-8859-1	Japanese (ja)	EUC-JP
Spanish (es)	ISO-8859-1	Korean (ko)	EUC-KR
Estonian (et)	ISO-8859-1	Lithuanian (lt)	ISO-8859-13
Finnish (fi)	ISO-8859-1	Latvian (lv)	ISO-8859-13
French (fr)	ISO-8859-1	Macedonian (mk)	ISO-8859-5
Irish (ga)	ISO-8859-1	Polish (pl)	ISO-8859-2
Galician (gl)	ISO-8859-1	Romanian (ro)	ISO-8859-2
Indonesian (id)	ISO-8859-1	Russian (ru)	KOI8-R

语言 (缩写)	编码	语言 (缩写)	编码
Icelandic (is)	ISO-8859-1	Slovak (sk)	ISO-8859-2
Italian (it)	ISO-8859-1	Slovenian (sl)	ISO-8859-2
Norwegian Bokmal (nb)	ISO-8859-1	Serbian Latin (sr@latin)	ISO-8859-2
Dutch (nl)	ISO-8859-1	Serbian (sr)	ISO-8859-5
Norwegian Nynorsk (nn)	ISO-8859-1	Turkish (tr)	ISO-8859-9
Norwegian (no)	ISO-8859-1	Ukrainian (uk)	KOI8-U
Portuguese (pt)	ISO-8859-1	Vietnamese (vi)	TCVN5712-1
Swedish (sv)	ISO-8859-1	Simplified Chinese (zh_CN)	GBK
Belarusian (be)	CP1251	Simplified Chinese, Singapore (zh_SG)	GBK
Bulgarian (bg)	CP1251	Traditional Chinese, Hong Kong (zh_HK)	BIG5HKSCS
Czech (cs)	ISO-8859-2	Traditional Chinese (zh_TW)	BIG5
Greek (el)	ISO-8859-7	#	#

Note

手册页不支持不在列表中的语言。

6.68.3. Man-DB 软件包内容

安装的程序: accessdb, apropos (链接到 whatis), catman, lexgrog, man, mandb, manpath, 和 whatis
安装的库: libman.so 和 libmandb.so
安装的目录: /usr/lib/man-db, /usr/lib/tmpfiles.d, /usr/libexec/man-db, 和 /usr/share/doc/man-db-2.8.5

简要介绍

accessdb 以人类可读形式转储 whatis 数据库
apropos 查询 whatis 数据库并显示包含指定字符串的系统命令的简要介绍
catman 创建或更新预格式化手册页
lexgrog 显示指定手册页的一行概要信息
man 格式化并显示要求的手册页
mandb 创建或更新 whatis 数据库
manpath 基于 man.conf 中的设置和用户环境显示 $MANPATH 或 (如果没有设置 $MANPATH) 合适的搜索路径的内容
whatis 查询 whatis 数据库并显示包含以给定关键字为独立字的系统命令的简要介绍
`libman` 包含 man 的运行时支持
`libmandb` 包含 man 的运行时支持

6.69. Tar-1.31

Tar 软件包包含一个归档程序。

大致构建用时: 1.7 SBU
所需磁盘空间: 45 MB

6.69.1. 安装 Tar

修复 tar-1.31 中引入的 bug：

```
sed -i 's/abort.*/FALLTHROUGH;/' src/extract.c
```

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
强制以 root 用户运行 mknod 的测试。通常认为以 root 用户运行该测试是危险的，但由于是在部分构建的系统上运行，这样并没有问题。

编译软件包：

```
make
```

用以下命令测试结果（大概 3 SBU）：

```
make check
```

安装软件包：

```
make install
make -C doc install-html docdir=/usr/share/doc/tar-1.31
```

6.69.2. Tar 软件包内容

安装的程序: tar
安装的目录: /usr/share/doc/tar-1.31

简要介绍

tar 创建归档文件，从归档文件提取文件，列出归档文件的内容，归档文件也被成为 tar 包

6.70. Texinfo-6.5

Texinfo 软件包包含用于读、写以及转换信息页的程序。

大致构建用时: 0.9 SBU
所需磁盘空间: 129 MB

6.70.1. 安装 Texinfo

修复在回归检测中导致大量失败项的文件：

```
sed -i '5481,5485 s/({/(\\{/' tp/Texinfo/Parser.pm
```

准备编译 Texinfo：

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

配置选项的含义：

--disable-static
顶级的配置脚本会告诉你这是一个未能识别的选项，但是 XSParagraph 的配置脚本能够识别它，并能用其来禁用安装静态 XSParagraph.a 至 /usr/lib/texinfo 的操作。

编译软件包：

```
make
```

用以下命令测试结果：

```
make check
```

安装软件包：

```
make install
```

可选地安装 TeX 中的组件：

```
make TEXMF=/usr/share/texmf install-tex
```

make 参数的含义：

TEXMF=/usr/share/texmf
如果后面会安装 TeX 软件包，TEXMF makefile 变量保存了作为 TeX 树的根位置。

该信息文档系统使用一个纯文本文件来存放菜单条目清单。文件保存在 /usr/share/info/dir。不幸的是，由于不同软件包 Makefile 的偶然问题，有时候会和系统中安装的信息页不同步。如果需要重建 /usr/share/info/dir 文件，下面的可选命令能完成任务：

```
pushd /usr/share/info
rm -v dir
for f in *
  do install-info $f dir 2>/dev/null
done
popd
```

6.70.2. Texinfo 软件包内容

安装的程序: info, install-info, makeinfo (链接到 texi2any), pdftexi2dvi, pod2texi, texi2any, texi2dvi, texi2pdf, 和 texindex
安装的库: XSParagraph.so
安装的目录: /usr/share/texinfo 和 /usr/lib/texinfo

简要介绍

info	用于读取和 man 页面相似的信息页，但和只是解释所有可用命令行选项相比更加深入，[例如比较 man bison 和 info bison。]
install-info	用于安装信息页；它会更新 info 索引文件中的条目
makeinfo	翻译给定的 Texinfo 源文档为信息页、纯文本或 HTML
pdftexi2dvi	用于格式化给定的 Texinfo 文档为 PDF 文件

pod2texi	转换 Pod 为 Texinfo 格式
texi2any	翻译 Texinfo 源文档为多种其它格式
texi2dvi	用于格式化给定 Texinfo 文档为可打印的设备无关文件
texi2pdf	用于格式化给定 Texinfo 文档为 PDF 文件
texindex	用于对 Texinfo 索引文件进行排序

6.71. Vim-8.1

Vim 软件包包含了一个强大的文本编辑器。

大致构建用时: 1.3 SBU
所需磁盘空间: 169 MB

> **Vim 的安装**
>
> 如果你钟情于其它的编辑器，比如 Emacs、Joe，或 Nano。请参考 http://www.linuxfromscratch.org/blfs/view/8.4/postlfs/editors.html 里的安装指导。

6.71.1. Vim 的安装

首先，把配置文件 `vimrc` 从默认位置移动到 `/etc`：

```
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
```

让 Vim 做好编译准备：

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

编译安装包：

```
make
```

检验结果：

```
LANG=en_US.UTF-8 make -j1 test &> vim-test.log
```

这个测试套件会输出一堆二进制数据到屏幕上。这会导致当前设置下的终端出现问题。把输出重定向到一个日志文件就可以解决这个问题。测试成功的话就会输出「ALL DONE」。

安装软件包：

```
make install
```

许多用户习惯于使用 vi 而不是 vim。为了当人们在习惯性的输入 vi 时能执行 vim，需要给二进制文件和 man 页建立符号连接：

```
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 的说明文档被安装在 `/usr/share/vim` 里。下面的这个符号链接使得可以通过 `/usr/share/doc/vim-8.1` 访问该文档，让它的位置与其它软件包的文档位置保持一致：

```
ln -sv ../vim/vim81/doc /usr/share/doc/vim-8.1
```

如果要把一个 X Window 系统安装在 LFS 系统上，可能得在安装完 X 系统后再重新编译 Vim。Vim 带有一个 GUI 版本，这个版本需要安装 X 和一些额外的库。想了解更多信息，请参考 Vim 文档和 BLFS http://www.linuxfromscratch.org/blfs/view/8.4/postlfs/vim.html 中 Vim 安装指导页。

6.71.2. 设置 Vim

默认情况下，vim 是以不兼容 vi 的模式运行的。这对于过去使用其它编辑器的用户可能是个新问题。下面列出了「非兼容性」设置以突出显示使用的新特性。它也提醒着那些想换成「兼容」模式的人，这是配置文件里第一个该被设置的地方。这非常有必要，因为它会改变其它的设置，而且覆写必须在这个设置之后。以下面的方式，创建一个默认的 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
```

设置 *`set nocompatible`* 让 vim 比 vi 兼容模式更有用。删掉「no」以保留旧的 vi 特性。设置 *`set backspace=2`* 让退格跨越换行、自动缩进和插入的开始。*`syntax on`* 参数使 vim 能高亮显示语法。设置 *`set mouse`* 让你能在 chroot 和远程连接的时候用鼠标粘帖文本。最后，带有 *`set background=dark`* 的 if 语句矫正了 vim 对于某些终端模拟器的背景颜色的估算。这让某些写在黑色背景上的程序的高亮色能有更好的调色方案。

用下面的命令可以获得其它选项的文档：

```
vim -c ':options'
```

> **Note**
>
> 默认情况下，Vim 只安装了英文的拼写检查文件。要想安装你想要的语言的拼写检查文件，请从 ftp://ftp.vim.org/pub/vim/runtime/spell/ 下载你所用语言的 `*.spl` 文件，可下可不下的 `*.sug` 文件和文字编码。并把它们保存到 `/usr/share/vim/vim81/spell/`。
>
> 要使用这些文件，需要设置 `/etc/vimrc` 里的某些项，例如：
>
> ```
> set spelllang=en,ru
> set spell
> ```
>
> 想要了解更多信息，请阅读上方 URL 里对应 README 文件。

6.71.3. Vim 软件包内容

安装的软件: ex (链接到 vim), rview (链接到 vim), rvim (链接到 vim), vi (链接到 vim), view (链接到 vim), vim, vimdiff (链接到 vim), vimtutor, 和 xxd

安装的目录: /usr/share/vim

简要介绍

ex	以 ex 模式启动 vim
rview	是 view 的一个受限版本；不能启动 shell 命令，而且 view 无法暂停
rvim	是 vim 的一个受限版本；不能启动 shell 命令，而且 vim 无法暂停
vi	链接到 vim
view	以只读模式启动 vim
vim	就是这个编辑器
vimdiff	用 vim 编辑一个文件的两到三个版本并显示它们的区别
vimtutor	vim 基本热键和命令的教学

xxd	将给定文件进行十六进制转储，还可以还原；所以它可以被用于打二进制补丁

6.72. systemd-240

systemd 软件包包含用于控制启动、运行和关闭系统的程序。

大致构建用时: 2.4 SBU
所需磁盘空间: 226 MB

6.72.1. 安装 systemd

应有该补丁以修复两个严重的安全漏洞：

```
patch -Np1 -i ../systemd-240-security_fixes-2.patch
```

创建符号链接以解决丢失 xsltproc 的问题：

```
ln -sf /tools/bin/true /usr/bin/xsltproc
```

由于我们还未安装 Util-Linux 的最终版本，所以在适当的位置创建指向库的链接：

```
for file in /tools/lib/lib{blkid,mount,uuid}*; do
    ln -sf $file /usr/lib/
done
```

设置 man 手册：

```
tar -xf ../systemd-man-pages-240.tar.xz
```

移除不能在 chroot 下构建的测试：

```
sed '177,$ d' -i src/resolve/meson.build
```

从默认的 udev 规则中删除没有必要的组 `render`：

```
sed -i 's/GROUP="render", //' rules/50-udev-default.rules.in
```

systemd 编译前准备：

```
mkdir -p build
cd       build

PKG_CONFIG_PATH="/usr/lib/pkgconfig:/tools/lib/pkgconfig" \
LANG=en_US.UTF-8                   \
meson --prefix=/usr                \
      --sysconfdir=/etc            \
      --localstatedir=/var         \
      -Dblkid=true                 \
      -Dbuildtype=release          \
      -Ddefault-dnssec=no          \
      -Dfirstboot=false            \
      -Dinstall-tests=false        \
      -Dkill-path=/bin/kill        \
      -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                \
      ..
```

选项的含义：

`-D*-path=*`
此类参数提供了 systemd 在运行时所需的二进制的位置，这些二进制文件有些还未安装，有些的 pkgconfig 文件现在还在`/tools/lib/pkgconfig` 中。

`-Ddefault-dnssec=no`
此参数关闭了实验性 DNSSEC 的支持。

-Dfirstboot=false
此参数将防止负责首次设置系统的 systemd 服务被安装。这对 LFS 没什么用，因为 LFS 的一切都是手动完成的。

-Dinstall-tests=false
此参数将防止编译测试的安装。

-Dldconfig=false
此参数将防止启动时运行 ldconfig 的 systemd 单元被安装，对于 LFS 这样的 原生发行版而言毫无作用还会拖慢启动时间。如果你也是这么想的就删掉它。

*-Droot**
此类参数用于确保核心程序和共享库能够安装到根分区的子目录。

-Dsplit-usr=true
此参数用于确保 systemd 可以在 /bin，/lib 和 /sbin 目录工作，而非符号链接指向的 /usr 副本。

-Dsysusers=false
此参数用于防止负责于设置 `/etc/group` 和 `/etc/passwd` 文件的 systemd 服务被安装。这两个文件都早在前章就创建了。

编译软件包：

```
LANG=en_US.UTF-8 ninja
```

安装软件包：

```
LANG=en_US.UTF-8 ninja install
```

移除不必要的目录：

```
rm -rfv /usr/lib/rpm
rm -f /usr/bin/xsltproc
```

创建 `/etc/machine-id` 需要的 systemd-journald 文件：

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

创建 `/lib/systemd/systemd-user-sessions` 脚本，以允许非特权用户无需 systemd-logind 亦可登录：

```
cat > /lib/systemd/systemd-user-sessions << "EOF"
#!/bin/bash
rm -f /run/nologin
EOF
chmod 755 /lib/systemd/systemd-user-sessions
```

6.72.2. systemd 软件包内容

安装的程序:	bootctl, busctl, coredumpctl, halt, hostnamectl, init, journalctl, kernel-install, localectl, loginctl, machinectl, networkctl, poweroff, reboot, runlevel, shutdown, systemctl, systemd-analyze, systemd-ask-password, systemd-cat, systemd-cgls, systemd-cgtop, systemd-delta, systemd-detect-virt, systemd-escape, systemd-hwdb, systemd-inhibit, systemd-machine-id-setup, systemd-mount, systemd-notify, systemd-nspawn, systemd-path, systemd-resolve, systemd-run, systemd-socket-activate, systemd-stdio-bridge, systemd-tmpfiles, systemd-tty-ask-password-agent, telinit, timedatectl, 和 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-240.so, 和 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-240, /usr/share/factory, /usr/share/systemd, /var/lib/systemd, 和 /var/log/journal

简要介绍

bootctl	用于查询固件和启动管理设置
busctl	用于自检和监控 D-Bus 总线

coredumpctl	用于检索 systemd 日志生成的核心转储
halt	通常调用带 *-h* 参数的 shutdown 命令，除非已经是运行等级 0，然后告诉内核暂停系统；它会在 `/var/log/wtmp` 文件中标记正准备关闭系统。
hostnamectl	用于查询或更改系统名称以及相关的设置
init	内核初始化硬件后接管启动过程而启动的第一个进程，随后会根据其配置文件，启动其中所有的进程。
journalctl	用于查询 systemd 日志的内容
kernel-install	用于向和从 /boot 中添加或移除内核以及 initramfs 镜像
localectl	用于查询和更改系统区域设置和键盘布局设置
loginctl	用于自检和控制 systemd 登录管理器的状态
machinectl	用于自检和控制 systemd 虚拟机和容器注册管理器的状态
networkctl	用于检查 systemd-networkd 所看到的网络链接状态
poweroff	告诉内核暂停系统并关闭计算机 （查看 halt）
reboot	告诉内核重启系统 （查看 halt）
runlevel	报告之前和当期的运行等级、即 `/var/run/utmp` 文件中的最后一个运行等级记录
shutdown	以安全的方式关闭系统、向所有进程发送信号并通知所有登录的用户
systemctl	用于自检和控制 systemd 系统和服务管理器的状态
systemd-analyze	用于确定当前引导中的系统启动性能
systemd-ask-password	通过命令行中的问题消息用于向用户查询系统密码或口令
systemd-cat	用于连接进程日志的 STDOUT 和 STDERR
systemd-cgls	以树的形式递归显示指定 Linux 控制组层次结构的内容
systemd-cgtop	按照 CPU、内存和磁盘 I/O 负载的顺序显示本地 Linux 控制组层次结构的顶层控制组
systemd-delta	用于识别和比较 `/etc` 中覆盖 `/usr` 对应部分的配置文件
systemd-detect-virt	在虚拟化环境中检测执行情况
systemd-escape	用于在 systemd 单元名称中包含转义字符串
systemd-hwdb	用户管理硬件数据库（hwdb）
systemd-inhibit	用于在关机、睡眠或空闲休眠锁时执行程序
systemd-machine-id-setup	启动时系统安装程序用随机生成的 ID 初始化保存到 `/etc/machine-id` 的机器 ID
systemd-mount	是临时挂载或自动挂载驱动器的工具。
systemd-notify	守护进程脚本用于通知 init 系统状态更改
systemd-nspawn	用于在轻量级容器空间中运行命令或操作系统
systemd-path	用于查询系统和用户路径
systemd-resolve	用于解析域名，IPv4 和 IPv6 地址，DNS 资源记录，以及服务
systemd-run	用于创建并运行一个临时 .service 或 .scope 单元并在其中运行指定命令
systemd-socket-activate	用于监听设备套接字并在连接时启动进程的工具。
systemd-tmpfiles	基于配置文件格式和 `tmpfiles.d` 指定的位置创建、删除以及清理易变的和临时文件和目录
systemd-tty-ask-password-agent	用于列出或执行正在等待的 systemd 密码请求
telinit	告诉 init 要更改的运行等级
timedatectl	用于查询和更改系统时间和设置
udevadm	通用 udev 管理工具：控制 udevd 守护进程、从 udev 数据库提供信息、监控 uevent、等待 uevent 完成、检测 udev 配置、为指定设备触发 uevents

libsystemd	systemd 主要的实用工具库
libudev	用于获取 udev 设备信息的库

6.73. D-Bus-1.12.12

D-Bus 是一个消息总线系统，应用之间相互通信的简单方式。D-Bus 支持系统守护进程（例如添加新硬件设备或打印队列更改事件）和每个用户的登录会话守护进程 （例如用户应用程序之间的一般进程间通信）。另外，消息总线在通用一对一消息传递框架之上构建，该框架使得任意两个应用可以直接通信（而不需要通过消息总线守护进程）。

大致构建用时: 0.2 SBU
所需磁盘空间: 18 MB

6.73.1. 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-1.12.12 \
            --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 验证目录的位置。

编译软件包：

```
make
```

该软件包没有测试套件，但要求 LFS 中没有的几个软件包。运行测试套件的命令可以在 BLFS 指南 http://www.linuxfromscratch.org/blfs/view/8.4/general/dbus.html 中找到。

安装软件包：

```
make install
```

需要移动共享库到 `/lib`，因此需要重建 `/usr/lib` 中的 `.so` 文件：

```
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，使得 D-Bus 和 systemd 可以使用相同的 `machine-id` 文件：

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

6.73.2. D-Bus 软件包内容

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, and 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-1.12.12, 和 /var/lib/dbus

简要介绍

dbus-cleanup-sockets	用于清理目录中残留的套接字
dbus-daemon	D-Bus 消息总线守护进程
dbus-launch	从 shell 脚本启动 dbus-daemon
dbus-monitor	监控 D-Bus 消息总线中传送的消息
dbus-run-session	从 shell 脚本中启动一个 dbus-daemon 会话总线实例并在会话中启动指定程序
dbus-send	发送消息到 D-Bus 消息总线
dbus-test-tool	帮助软件包测试 D-Bus 的工具

dbus-update-activation-environment	更新 D-Bus 会话服务的环境变量
dbus-uuidgen	生成一个通用唯一 ID
`libdbus-1`	包含用于和 D-Bus 消息总线交互的 API 函数

6.74. Procps-ng-3.3.15

Procps-ng 软件包包含监视进程的程序。

大致构建用时: 0.1 SBU
所需磁盘空间: 17 MB

6.74.1. 安装 Procps-ng

准备编译 procps-ng：

```
./configure --prefix=/usr                            \
            --exec-prefix=                           \
            --libdir=/usr/lib                        \
            --docdir=/usr/share/doc/procps-ng-3.3.15 \
            --disable-static                         \
            --disable-kill                           \
            --with-systemd
```

配置选项的含义：

--disable-kill
该选项将不会编译已经由 Util-linux 软件包安装了的 kill 命令。

编译软件包：

```
make
```

对于 LFS，测试套件需要自定义某些更改。移除当脚本不使用 tty 设备时失败的测试，同时修复另两个失败。用下面的命令运行测试套件：

```
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
```

安装软件包：

```
make install
```

最后，如果 /usr 没有挂载的话，移动重要文件到一个可以找到的位置。

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

6.74.2. Procps-ng 软件包内容

安装的程序: free, pgrep, pidof, pkill, pmap, ps, pwdx, slabtop, sysctl, tload, top, uptime, vmstat, w, 和 watch
安装的库: libprocps.so
安装目录: /usr/include/proc 和 /usr/share/doc/procps-ng-3.3.15

简要介绍

free	报告系统中空闲和使用的内存容量（包括物理和交换内存）
pgrep	根据名称和其它属性查找进程
pidof	报告指定程序的 PID
pkill	根据名称和其它属性给进程发送信号
pmap	报告指定进程的内存映射情况
ps	列出正在运行的进程
pwdx	报告进程的当前工作目录
slabtop	实时显示内核 slab 缓存信息
sysctl	运行时修改内核参数
tload	打印当前系统平均负荷曲线图
top	显示最 CPU 密集型进程列表；它可以实时地连续查看处理器活动

uptime	报告系统运行时长、登录用户数目以及系统平均负荷
vmstat	报告虚拟内存统计信息、给出关于进程、内存、分页、块输入/输出(IO)、陷阱以及 CPU 活动的信息
w	显示当前登录的用户、以及登录地点和时间
watch	重复运行指定命令，显示输出的第一个整屏；这允许用户查看随着时间的输出变化
`libprocps`	包含该软件包大部分程序使用的函数

6.75. Util-linux-2.33.1

Util-linux 软件包其它实用程序。包括处理文件系统、控制台、分区以及消息等工具。

大致构建用时: 1.5 SBU
所需磁盘空间: 214 MB

6.75.1. FHS 兼容性注意事项

FHS 推荐使用 /var/lib/hwclock 目录而不是通常的 /etc 目录作为 adjtime 文件的位置。首先新建目录用于存储 hwclock 程序：

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

6.75.2. 安装 Util-linux

删除早前创建的符号链接：

```
rm -vf /usr/include/{blkid,libmount,uuid}
```

准备编译 Util-linux：

```
./configure ADJTIME_PATH=/var/lib/hwclock/adjtime   \
            --docdir=/usr/share/doc/util-linux-2.33.1 \
            --disable-chfn-chsh  \
            --disable-login      \
            --disable-nologin    \
            --disable-su         \
            --disable-setpriv    \
            --disable-runuser    \
            --disable-pylibmount \
            --disable-static     \
            --without-python
```

--disable 和 --without 选项用于防止出现关于 LFS 中缺少构建组件需要的软件包或和其它软件包安装的程序不一致的警告。

编译软件包：

```
make
```

如果需要的话，以非 root 用户运行测试套件：

> **Warning**
>
> 以 root 用户运行测试套件会对系统有害。为了运行测试套件，必须保证当前运行的系统中用于内核的 CONFIG_SCSI_DEBUG 选项可用，且是以一个模块的方式编译。把它构建到内核中将阻止启动。为了全面覆盖，还必须安装其它的 BLFS 软件包。如果需要的话，可以在重启进入完整的 LFS 系统后用以下命令运行该测试：
>
> ```
> bash tests/run.sh --srcdir=$PWD --builddir=$PWD
> ```

```
chown -Rv nobody .
su nobody -s /bin/bash -c "PATH=$PATH make -k check"
```

安装软件包：

```
make install
```

6.75.3. Util-linux 软件包内容

安装的程序:	addpart, agetty, blkdiscard, blkid, blockdev, cal, cfdisk, chcpu, choom, chrt, col, colcrt, colrm, column, ctrlaltdel, delpart, dmesg, eject, fallocate, fdformat, fdisk, findfs, findmnt, flock, fsck, fsck.cramfs, fsck.minix, fsfreeze, fstrim, getopt, hexdump, hwclock, i386, ionice, ipcmk, ipcrm, ipcs, isosize, kill, last, lastb (link to last), ldattach, linux32, linux64, logger, look, losetup, lsblk, lscpu, lsipc, lslocks, lslogins, mcookie, mesg, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, mountpoint, namei, nsenter, partx, pg, pivot_root, prlimit, raw, readprofile, rename, renice, resizepart, rev, rtcwake, script, scriptreplay, setarch, setsid, setterm, sfdisk, sulogin, swaplabel, swapoff (link to swapon), swapon, switch_root, tailf, taskset, ul, umount, uname26, unshare, utmpdump, uuidd, uuidgen, wall, wdctl, whereis, wipefs, x86_64, 和 zramctl
安装的库:	libblkid.so, libfdisk.so, libmount.so, libsmartcols.so, 和 libuuid.so
安装的库:	/usr/include/blkid, /usr/include/libfdisk, /usr/include/libmount, /usr/include/libsmartcols, /usr/include/uuid, /usr/share/doc/util-linux-2.33.1, 和 /var/lib/hwclock

简要介绍

addpart	通知 Linux 内核有新的分区
agetty	打开一个 tty 端口，提示输入登录名，然后调用 login 程序
blkdiscard	丢弃设备上的扇区
blkid	用于定位和打印块设备属性的命令行工具
blockdev	允许用户在命令行中调用块设备的 ioctls
cal	显示一个简单的日历
cfdisk	管理指定设备的分区表
chcpu	更改 CPU 的状态
choom	显示和调整 OOM-killer 分数
chrt	管理进程的的实时属性
col	过滤掉反向换行符
colcrt	为终端过滤缺少某些功能，例如加粗和半行的 nroff 输出
colrm	过滤掉指定的列
column	格式化指定文件为多列
ctrlaltdel	设置 Ctrl+Alt+Del 组合键的功能为硬或软复位
delpart	请求 Linux 内核移除一个分区
dmesg	转储内核启动信息
eject	弹出可移除媒体
fallocate	为文件预分配空间
fdformat	低级别格式化软盘
fdisk	管理指定设备的分区表
findfs	通过标签或通用唯一标识符（UUID）查找文件系统
findmnt	libmount 库中用于和 mountinfo、fstab 和 mtab 文件工作的命令行接口
flock	请求一个文件锁，然后用所持有的锁执行命令
fsck	用于检查或者修复文件系统
fsck.cramfs	在指定设备的 Cramfs 文件系统上进行一致性检查
fsck.minix	在指定设备的 Minix 文件系统上进行一致性检查
fsfreeze	FIFREEZE/FITHAW ioctl 内核驱动操作的简单封装
fstrim	丢弃已挂载的文件系统中未使用的块
getopt	解析给定命令行中的选项
hexdump	以十六进制或其它指定格式转储给定文件

hwclock	读取或设置系统硬件时钟，也称为实时时钟（RTC）或基本输入输出系统（BIOS）时钟
i386	到 setarch 的符号链接
ionice	为某个程序获取或设置 io 调度类和优先级
ipcmk	创建多种 IPC（进程间通信） 资源
ipcrm	移除指定的进程间通信（IPC）资源
ipcs	提供 IPC 状态信息
isosize	报告 iso9660 文件系统的大小
kill	向进程发送信号
last	通过反向查找 /var/log/wtmp 文件显示上次登录（注销）的用户；同时还显示系统启动、关闭以及运行时级别更改
lastb	根据 /var/log/btmp 中的日志显示失败的登录尝试
ldattach	向行中添加行规则
linux32	到 setarch 的符号链接
linux64	到 setarch 的符号链接
logger	输入给定的信息到系统日志
look	显示以指定字符串开头的行
losetup	设置和控制环路设备
lsblk	以类似树的形式列出所有或指定块设备的信息
lscpu	打印 CPU 架构信息
lsipc	打印目前使用的 IPC 设备的信息
lslocks	打印 CPU 架构信息
lslogins	列出关于用户、组和系统账号的信息
mcookie	为 xauth 生成 magic cookies（128位随机十六进制数）
mesg	控制其它用户是否可以向当前用户终端发送信息
mkfs	在设备上构建文件系统（通常是一个硬盘分区）
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	初始化指定设备或文件作为交换空间使用
more	用于每次显示文本一页的过滤器
mount	在文件系统树中挂载文件系统到给定设备的指定目录
mountpoint	检查目录是否是一个挂载点
namei	显示给定路径名称的符号链接
nsenter	在其他进程的命名空间中运行程序
partx	告诉内核磁盘上存在的分区和编号
pg	每次显示一屏文本文件
pivot_root	使指定文件系统作为当前进程的新的根文件系统
prlimit	获取或设置进程资源限制
raw	绑定 Linux 原始字符设备到一个块设备
readprofile	读取内核分析信息
rename	重命名指定文件，用另一个字符串替换指定字符串
renice	更改运行中进程的优先级
resizepart	请求 Linux 内核重新设置分区大小
rev	反转指定文件的行
rtcwake	用于进入系统睡眠状态知道指定的唤醒时间
script	生成终端会话的打字稿

scriptreplay	用定时信息播放打字稿
setarch	在新程序环境中更改报告架构并设置个性标签
setsid	在新会话中运行指定程序
setterm	设置终端属性
sfdisk	磁盘分区表管理器
sulogin	允许 root 登录；通常当系统进入单用户模式时由 init 调用
swaplabel	允许更改交换空间 UUID 和标签
swapoff	停用设备和文件的分页和交换机制
swapon	启用设备和文件的分页和交换机制并列出当前使用的设备和文件
switch_root	切换到另一个文件系统并把当前路径作为挂载树的根
tailf	跟踪日志文件的的增长；显示日志文件的最后 10 行然后继续显示日志文件中 添加的任何新条目
taskset	获取或设置一个进程的 CPU 亲和力
ul	将强调转换为转义序列以表示强调正在使用的终端的过滤器
umount	断开文件系统到系统文件树的连接
uname26	一个到 setarch 的软链接
unshare	用一些父进程非共享的名字空间运行程序
utmpdump	以更友好的格式显示指定登录文件的内容
uuidd	UUID 库用于生成基于时间的安全和保证唯一的 UUID 的守护进程
uuidgen	创建新的 UUID。在所有创建的 UUID 中，在本地系统或其它系统，在之前和以后，每个新的 UUID 都可以被认为是唯一的
wall	在终端上显示所有当前登录用户的文件内容，或者默认的标准输出
wdctl	显示硬件看门狗状态
whereis	报告指定命令的二进制文件、源代码或者 man 手册的位置
wipefs	从设备中擦除文件系统签名
x86_64	到 setarch 的符号链接
zramctl	用于设置和控制 zram（压缩后的 ram 磁盘）的程序
libblkid	包含用于设备识别和标记提取的例程
libfdisk	包含操作分区表的例程
libmount	包含用于块设备挂载和卸载的例程
libsmartcols	包含以表格形式进行屏幕输出的例程
libuuid	包含用于生成在本地系统之上可访问对象的唯一标识符的例程

6.76. E2fsprogs-1.44.5

E2fsprogs 软件包包含用于处理 ext2 文件系统的工具。它也支持 ext3 和 ext4 日志文件系统。

大致构建用时: 1.6 SBU
所需磁盘空间: 96 MB

6.76.1. 安装 E2fsprogs

E2fsprogs 的文档建议在源码目录下新建目录进行编译：

```
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
```

环境变量和配置选项的含义：

--with-root-prefix="" 和 *--bindir=/bin*
有些程序（例如 e2fsck）属于重要程序。比如，当 /usr 没有挂载的时候，仍然要求这些程序可用。它们放在类似 /lib 和 /sbin 的目录中。如果没有传递这个参数到 E2fsprogs 的配置参数中，程序就会被安装在 /usr 目录。

--enable-elf-shlibs
创建该软件包中一些程序会使用的共享库。

*--disable-**
这会阻止 E2fsprogs 编译和安装 libuuid 和 libblkid 库、uuidd 守护进程、以及 fsck 封装包。因为 Util-Linux 安装了更新的版本。

编译软件包：

```
make
```

输入命令运行测试：

```
make check
```

E2fsprogs 的其中一个测试程序会试图分配 256M 的内存。如果你没有比这更多的 RAM，确保为测试启用了足够的交换空间。阅读 Section#2.5, “在分区上创建文件系统”以及 Section#2.7, “挂载新分区”查看创建和启用交换空间的详细信息。

安装二进制文件、文档以及共享库：

```
make install
```

安装静态库和头文件：

```
make install-libs
```

使安装的静态库可写，以便后面可以移除调试符号：

```
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 文件但并没有更新系统级的 dir 文件。解压该文件并用下面的命令更新系统的 dir 文件：

```
gunzip -v /usr/share/info/libext2fs.info.gz
install-info --dir-file=/usr/share/info/dir /usr/share/info/libext2fs.info
```

如果需要的话，用下面的命令创建和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
```

6.76.2. E2fsprogs 软件包内容

安装的程序:	badblocks, chattr, compile_et, debugfs, dumpe2fs,e2freefrag, e2fsck, e2image, e2label, e2undo, e4defrag, filefrag, fsck.ext2, fsck.ext3, fsck.ext4, fsck.ext4dev, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mkfs.ext4, mkfs.ext4dev, mklost+found, resize2fs, 和 tune2fs
安装的库:	libcom_err.so, libe2p.so, libext2fs.so, 和 libss.so
安装目录:	/usr/include/e2p, /usr/include/et, /usr/include/ext2fs, /usr/include/ss, /usr/share/et, 和 /usr/share/ss

简要介绍

badblocks	搜索设备（通常是一个磁盘分区）的坏块
chattr	更改 ext2 文件系统中文件的属性；它也可以更改 ext3 文件系统，这是 ext2 文件系统的日志版本
compile_et	错误表编译器；它将错误代码名称和信息对照表转换为适用于 com_err 库的 C 源文件。
debugfs	文件系统调试器；可用于检查和更改 ext2 文件系统的状态
dumpe2fs	对指定设备上的文件系统打印超级块和块组信息
e2freefrag	报告空闲空间的碎片信息
e2fsck	用于检查或者修复 ext2 文件系统和 ext3 文件系统
e2image	用于将重要 ext2 文件系统数据保存到文件中
e2label	显示或更改指定设备上的 ext2 文件系统的标签
e2undo	对设备上发现的 ext2/ext3/ext4 文件系统重做撤销日志 undo_log [这可用于取消一个 e2fsprogs 程序的失败操作。]
e4defrag	ext4 文件系统的在线碎片整理器
filefrag	报告一个文件可能的碎片化程度
fsck.ext2	默认检查 ext2 文件系统，是到 e2fsck 的硬链接
fsck.ext3	默认检查 ext3 文件系统，是到 e2fsck 的硬链接
fsck.ext4	默认检查 ext4 文件系统，是到 e2fsck 的硬链接
fsck.ext4dev	默认检查 ext4 开发版文件系统，是到 e2fsck 的硬链接
logsave	在日志文件中保存命令的输出
lsattr	列出二级扩展文件系统中一个文件的属性
mk_cmds	将命令名称和帮助信息的映射表转换为适用于 libss 子系统库的 C 源文件
mke2fs	在指定设备上创建 ext2 或 ext3 文件系统
mkfs.ext2	默认创建 ext2 文件系统，是到 mke2fs 的硬链接
mkfs.ext3	默认创建 ext3 文件系统，是到 mke2fs 的硬链接
mkfs.ext4	默认创建 ext4 文件系统，是到 mke2fs 的硬链接
mkfs.ext4dev	默认创建 ext4 开发版文件系统，是到 mke2fs 的硬链接
mklost+found	用于在 lost+found 文件系统上创建 ext2 目录，它可以预先为目录分配磁盘块，以减轻 e2fsck 任务
resize2fs	用于伸缩 ext2 文件系统的大小
tune2fs	调整 ext2 文件系统上的可调文件系统参数
libcom_err	常用错误显示例程
libe2p	用于 dumpe2fs，chattr，和 lsattr
libext2fs	包含使用户层程序可以操作 ext2 文件系统的例程
libss	用于 debugfs

6.77. 关于调试符号

默认情况下大多数程序和库的编译带有调试符号。（类似 gcc 的 *-g* 选项。）这意味着当你调试一个包含调试信息的已编译的程序或库时，调试程序不仅能提供内存地址，还能提供变量和实例的名字。

然而，包含这些调试符号明显的增大了程序或库。下面这个例子说明了这些符号有多么占地方：

- 有调试符号的二进制 bash：1200 KB
- 无调试符号的二进制 bash：480 KB
- 有调试符号的 Glibc 和 GCC 文件（`/lib` 和 `/usr/lib`）：87 MB
- 无调试符号的 Glibc 和 GCC 文件：16 MB

大小可能会因为所使用的编译器和 C 语言库的不同而改变，但是当比较有无调试符号的程序时，大小可能相差 2 到 5 倍。

因为大多数用户从来不会在他们的系统软件上使用调试器，没了这些调试符号可以省下很多磁盘空间。下一页将会告诉你如何剥离程序和库中所有的调试符号。

6.78. 再次清理无用内容

这个部分是可选的。如果预期的用户不是一个程序员或者不打算对系统软件进行任何调试，通过从二进制文件和库中删除调试符号能减少 90MB 的系统大小。除了不能完全调试软件，这不会导致任何不便。

大部分人使用下面提到的命令并不会感到任何困难。然而，很容易出现错误并导致新的系统不可用，因此在运行 strip 命令之前，对当前状态的 LFS 系统进行备份是个好主意。

首先将选定库的调试符号文件分开放置。如果要在后续的 BLFS 中用 valgrind 或 gdb 做回归测试，那么调试信息还有用武之地。

```
save_lib="ld-2.29.so libc-2.29.so libpthread-2.29.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.6.0.25
             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
```

在进行清理无用内容之前，格外注意确保要删除的二进制文件没有正在运行：

```
exec /tools/bin/bash
```

现在可以安心的清除二进制文件和库：

```
/tools/bin/find /usr/lib -type f -name \*.a \
   -exec /tools/bin/strip --strip-debug {} ';'

/tools/bin/find /lib /usr/lib -type f \( -name \*.so* -a ! -name \*dbg \) \
   -exec /tools/bin/strip --strip-unneeded {} ';'

/tools/bin/find /{bin,sbin} /usr/{bin,sbin,libexec} -type f \
    -exec /tools/bin/strip --strip-all {} ';'
```

该命令会报告有很大数目的文件不能识别它们的格式。你可以安全地忽略这些警告。这些警告表示这些文件是脚本而不是二进制文件。

6.79. 清理

最后，清除运行测试留下来的多余文件：

```
rm -rf /tmp/*
```

现在，登出后用以下新的 chroot 命令重新进入 chroot 环境。在此以后当需要进入 chroot 环境时，都是用这个新的 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
```

这样做的原因是不再需要 `/tools` 中的程序。因此你可以删除 `/tools` 目录。

> **Note**
>
> 移除 `/tools` 也会删除用于运行工具链测试的 Tcl、Expect和 DejaGNU 的临时复制。如果你在后面还需要这些程序，需要重新编译并安装它们。BLFS 手册有关于这些的指令（请查看http://www.linuxfromscratch.org/blfs/）。

如果通过手动或者重启卸载了虚拟内核文件系统，重新进入 chroot 的时候确保挂载了虚拟内核文件系统。在 Section#6.2.2, “挂载和激活 /dev”和 Section#6.2.3, “挂载虚拟文件系统”中介绍了该过程。

还有一些此章之前为了一些软件包的回归测试而留下的静态库。这些库来自 binutils、bzip2、e2fsprogs、flex、libtool 和 zlib。如果想删的话，现在就删：

```
rm -f /usr/lib/lib{bfd,opcodes}.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
```

还有几个安装在 /usr/lib 和 /usr/libexec 目录下的文件，文件的扩展名为 .la。这些是「libtool 归档」文件，在 Linux 系统中通常不需要它们。这些都是没有必要的东西。想要删除的话，运行：

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

关于 libtool 归档文件的更多信息，参考 BLFS 段落「关于 Libtool 归档（.la）文件」。

Chapter 7. 基本系统配置

7.1. 简介

这一章将会讲解配置文件和 systemd 服务。首先，给出了设置网络需要的通用配置文件。

- Section#7.2, “通用网络配置”
- Section#7.2.3, “配置系统主机名称”
- Section#7.2.4, “自定义 /etc/hosts 文件”

然后，讨论可能影响正确的设备配置的问题。

- Section#7.3, “设备与模块管理概述”
- Section#7.4, “设备管理”

第三步，配置系统时间和键盘布局。

- Section#7.5, “配置系统时间”
- Section#7.6, “配置 Linux 控制台”

第四步，简单介绍当用户登录系统时用到的脚本和配置文件。

- Section#7.7, “配置系统语言环境”
- Section#7.8, “创建 /etc/inputrc 文件”

最后，配置 systemd。

- Section#7.10, “systemd 的用法与配置”

7.2. 通用网络配置

本节仅在需要配置网卡时参考。

7.2.1. 网络接口配置文件

自版本 209 开始，systemd 提供了一个名为 systemd-networkd 的命令用于处理基本的网络配置。另外，自版本 213 开始，DNS 名称解析可用 systemd-resolved 代替静态的 `/etc/resolv.conf` 文件来解决。默认情况，两种服务都将被启用。

systemd-networkd（和 systemd-resolved 的配置文件可能在 `/usr/lib/systemd/network` 或 `/etc/systemd/network`中。`/etc/systemd/network` 中文件比 `/usr/lib/systemd/network`中的有更高的优先级。配置文件类型有三种：`.link`，`.netdev` 和 `.network` 文件。可以通过查阅 man 手册的 `systemd-link(5)`，`systemd-netdev(5)` 和 `systemd-network(5)` 获取更多关于这些配置文件的详细介绍。

7.2.1.1. 网络设备命名

udev 通常会根据系统物理特性分配接口名称。例如，enp2s1。如果你不确定你的接口名称，你可以在系统启动后运行 ip link 查看。

对于大多数系统，每种连接只会有一种网络接口。例如，传统有线连接的接口名 eth0。而无线连接的名称通常是 wifi0 或 wlan0。

如果你更青睐传统或是自定义的网络接口名称，有三种实现的方法：

- 为默认的策略隐蔽 udev 的 .link 文件：

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

- 创建手动命名规则，比方说，将接口命名成「internet0」，「dmz0」或「lan0」这样。为此，请在 /etc/systemd/network/ 中创建 .link 文件，为其中的一个，一些，或者说你全部的接口赋予明确的名字或是更妥善的命名规则。示例：

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

[Link]
Name=ether0
EOF
```

参考 man 手册 systemd.link(5) 获取更多信息。

- 在 /boot/grub/grub.cfg 中，给内核命令行传递 net.ifnames=0 选项。

7.2.1.2. 静态 IP 的配置

以下为设置静态 IP 而创建的基础配置文件（同时用到了 systemd-networkd 和 systemd-resolved）：

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

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

如果你拥有的 DNS 服务器超过了一个，可以为其添加多个 DNS 条目。然而，如果你打算用静态的 /etc/resolv.conf 文件，就不要添加 DNS 或域名条目。

7.2.1.3. DHCP 的配置

通过以下命令，你可以创建用于设置 IPv4 DHCP 的基础配置文件：

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

[Network]
DHCP=ipv4

[DHCP]
UseDomains=true
EOF
```

7.2.2. 创建 /etc/resolv.conf 文件

如果你的系统需要连接到互联网，它需要利用 DNS 服务将互联网域名解析为实际的 IP 地址，反之亦然。最好的方法是将从 ISP 或者是网络管理员那里取得的 DNS 服务器地址填入 /etc/resolv.conf。

7.2.2.1. systemd-resolved 的配置

> **Note**
>
> 如果你使用其他方式来配置你的网络接口（例如：ppp，network-manager，等），或是任何类型的本地解析器（例如：bind，dnsmasq，等），或是任何生成 /etc/resolv.conf 文件的软件（例如：resolvconf），也就别用 systemd-resolved 服务了。

用 systemd-resolved 配置 DNS 时，会创建 /run/systemd/resolve/resolv.conf 文件。并在 /etc 中创建一个指向生成文件的软链接：

```
ln -sfv /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

7.2.2.2. 静态 resolv.conf 的配置

如果需要静态的 /etc/resolv.conf 文件，请使用以下命令：

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

domain <Your Domain Name>
nameserver <IP address of your primary nameserver>
nameserver <IP address of your secondary nameserver>

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

domain 声明可以忽略或者以 search 声明替换。参考 man 手册的 resolv.conf 部分获得更多信息。

其中，*<IP address of the nameserver>* 替换为最合适的 DNS 的 IP 地址。通常会有多个条目（需要备选服务器具有相关兼容性）。如果你只需要一台 DNS 服务器，请不要输入第二行 nameserver 的内容。该 IP 地址也可以是本地网络中的一台路由。

> **Note**
>
> Google 公开的 DNS 解析服务器地址 IPv4 的为：*8.8.8.8* 和 *8.8.4.4*。IPv6 的为 *2001:4860:4860::8888* 和 *2001:4860:4860::8844*。
>
> 114 DNS：*114.114.114.114* 和 *114.114.115.115*。
>
> 阿里 DNS：*223.5.5.5* 和 *223.6.6.6*。
>
> 百度 DNS：*180.76.76.76*
>
> OpenDNS：*208.67.220.220*

7.2.3. 配置系统主机名称

在系统启动过程中，/etc/hostname 文件用于创建系统的主机名称。

通过以下命令创建 /etc/hostname 文件：

```
echo "<lfs>" > /etc/hostname
```

<lfs> 替换为你想要设置的名称。请不要输入完整域名（Fully Qualified Domain Name，FQDN），那应该是放在 /etc/hosts 文件中的信息。

7.2.4. 自定义 /etc/hosts 文件

决定完整域名（Fully-Qualified Domain Name，FQDN），和可在文件 /etc/hosts 中使用的别名。如果使用的是静态地址，你还需要决定 IP 地址。hosts 文件中条目的语法为：

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

除非电脑在互联网中可见（例如，拥有注册的域名且分配有有效的 IP 地址——大多数用户并没有这些），请确保 IP 地址位于有效的私有网络 IP 地址段。有效区间是：

```
Private Network Address Range      Normal Prefix
10.0.0.1 - 10.255.255.254           8
172.x.0.1 - 172.x.255.254           16
192.168.y.1 - 192.168.y.254         24
```

x 可以时 16-31 之间的任何数字。y 可以时 0-255 之间的任何数字。

有效的私有 IP 可以是 192.168.1.1。而与之相配的完整域名可以是 lfs.example.org。

即使没有网卡，一个有效的完整仍然有其必要。它的存在可以确保程序正常运行。

如果使用的是 DHCP，DHCPv6，IPv6 自动配置，或者说不配置网卡的话，通过以下命令可以创建 /etc/hosts 文件：

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

127.0.0.1 localhost
127.0.1.1 <FQDN> <HOSTNAME>
::1       localhost ip6-localhost ip6-loopback
ff02::1   ip6-allnodes
ff02::2   ip6-allrouters

# End /etc/hosts
EOF
```

::1 相对与 IPv4 的 127.0.0.1，是 IPv6 的回环地址。127.0.1.1 专门为 FQDN 保留的回环地址。

如果使用的是静态地址，可代替的使用以下命令创建 /etc/hosts 文件。

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

127.0.0.1 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>*，和 *<HOSTNAME>* 的值需要根据具体的用途或需求更改（如果网络/系统管理员会分配了 IP 地址，且机器将会接入现有的网络）。那可选的别名可以忽略。

7.3. 设备与模块管理概述

在 Chapter#6我们构建 systemd 时，已经安装了 udev 包。在我们详细说明 udev 系统的用法之前，我们先大致了解早先的设备控制方式。

传统的 Linux 不管硬件是否真实存在，都以创建静态设备的方法来处理硬件，因此需要在 /dev 目录下创建大量的设备节点文件（有时会有上千个）。这通常由 MAKEDEV 脚本完成，它通过大量调用 mknod 程序为这个世界上可能存在的每一个设备建立对应的主设备号和次设备号。

而使用 udev 方法，只有当内核检测到硬件接入，才会建立对应的节点文件。因为需要在系统启动的时候重新建立设备节点文件，所以将它存储在 devtmpfs 文件系统中（完全存在于内存中的虚拟文件系统）。设备节点文件无需太多的空间，所以占用的内存也很小。

7.3.1. 历史

2000 年 2 月，一种名叫 devfs 的文件系统被合并到 2.3.46 版本的内核之中，而在 2.4 系列的稳定内核中基本可用。尽管它存在于内核代码中，但是这种动态创建设备的方法却从来都没有到核心开发者的大力支持。

问题存在于它处理设备的检测、创建和命令的方式。其中最大的问题莫过于它对设备节点的命名方式。大部分开发者的观点是：设备的命名应该由系统的所有者决定，而不是开发者。而且 devfs 存在严重的竞争条件(race condition)问题，如不对内核做大量的修改就无法修正这一问题。最终，因为缺乏有效的维护，在 2006 年 6 月终被移出内核源代码。

后来，有一种新的虚拟文件系统 sysfs 在 2.5 系列测试版本内核中引入，并且加入了 2.6 系列的稳定版本内核之中。sysfs 系统的任务就是将系统中的硬件配置状态导出至用户空间，而这给开发一种运行于用户空间的新型 devfs 系统带来了可能。

7.3.2. udev 实现

7.3.2.1. Sysfs

上文简单的提及了 sysfs 文件系统。有些人可能会问，sysfs 到底是如何知道当前系统有哪些设备、这些设备又该使用什么设备号呢。对于那些已经编译进内核的设备，会在内核检测到时被直接注册为 sysfs 对象（由 devtmpfs 内建）。对于编译为内核模块的设备，将会在模块载入的时候注册。一旦 sysfs 文件系统挂载到 /sys，已经在 sysfs 注册的硬件数据就可以被用户空间的进程使用，随后也就可以被 udevd 处理了（包括对设备节点进行修改）。

7.3.2.2. 设备节点的创建

设备文件是通过内核中的 devtmpfs 文件系统创建的。任何想要注册的设备都需要通过 devtmpfs （经由驱动程序核心）实现。每当一个 devtmpfs 实例挂载到 /dev，就会建立一个设备节点文件，它拥有固定的名称、权限以及所有者。

很短的时间之后，内核将给 udevd 一个 uevent。基于 /etc/udev/rules.d、/lib/udev/rules.d 和 /run/udev/rules.d 目录内文件指定的规则，udevd 将会建立到设备节点文件的额外符号链接，这有可能更改其权限、所有者和所在组，或者是更改 udevd 内建接口（名称）。

这三个文件夹中的规则文件都应以数字编号，并会被一起处理。当发现一个新的设备时，若 udevd 无法找到对应的规则，将会使用 devtmpfs 中初始的权限以及所有者。

7.3.2.3. 加载模块

编译成模块的设备驱动可能会包含别名。别名可以通过 modinfo 命令查看，一般是模块支持的特定总线的设备描述符。举个例子，驱动 snd-fm801 支持厂商 ID 为 0x1319 以及设备 ID 为 0x0801 的设备，它包含一个「pci:v00001319d00000801sv*sd*bc04sc01i*」的别名，总线驱动导出该驱动别名并通过 `sysfs` 处理相关设备。例如，文件 `/sys/bus/pci/devices/0000:00:0d.0/modalias` 应该会包含字符串「pci:v00001319d00000801sv00001319sd00001319bc04sc01i00」。udev 采用的默认规则会让 udevd 根据 uevent 环境变量 `MODALIAS` 的内容（它应该和 sysfs 里的 `modalias` 文件内容一样）调用 /sbin/modprobe，这样就可以加载在通配符扩展后能和这个字符串一致的所有模块。

这个例子意味着，除了 snd-fm801 之外，一个已经废弃的（不是我们所希望的）驱动 forte 如果存在的话也会被加载。下面有几种可以避免加载多余驱动的方式。

内核本身也能够根据需要加载网络协议，文件系统以及 NLS 支持模块。

7.3.2.4. 处理热插拔/动态设备

在你插入一个设备时，例如一个通用串行总线（USB）MP3 播放器，内核检测到设备已连接就会生成一个 uevent。这个 uevent 随后会被上面所说的 udevd 处理。

7.3.3. 加载模块和创建设备时可能碰到的问题

在自动创建设备节点时可能会碰到一些问题。

7.3.3.1. 内核模块没有自动加载

udev 只会加载包含有特定总线别名而且已经被总线驱动导出到 `sysfs` 下的模块。在其它情况下，你应该考虑用其它方式加载模块。采用 Linux-4.20.12，udev 可以加载编写合适的 INPUT、IDE、PCI、USB、SCSI、SERIO 和 FireWire 设备驱动。

要确定你希望加载的驱动是否支持 udev，可以用模块名字作为参数运行 modinfo。然后查看 `/sys/bus` 下的设备目录里是否有个 `modalias` 文件。

如果在 `sysfs` 下能找到 `modalias` 文件，那么就能驱动这个设备并可以直接操作它，但是如果该文件里没有包含设备别名，那意味着这个驱动有问题。我们可以先尝试不依靠 udev 直接加载驱动，并寄希望于这个问题能在日后得到解决。

如果在 `/sys/bus` 下的相应目录里没有 `modalias` 的话，意味着内核开发人员还没有为这个总线类型增加 modalias 支持。使用 Linux-4.20.12 内核，应该是 ISA 总线的问题。希望这个可以在后面的内核版本里得到解决。

udev 根本不打算加载 snd-pcm-oss 这样的「封装」驱动程序和 loop 这样的非硬件设备驱动。

7.3.3.2. 内核驱动没有自动加载，udev 也没有尝试加载

如果「封装」模块只是强化其它模块的功能（比如，snd-pcm-oss 模块通过允许 OSS 应用直接访问声卡的方式加强了 snd-pcm 模块的功能），需要配置 modprobe 在 udev 加载硬件驱动模块后再加载相应的封装模块。为此，可以在对应的 `/etc/modprobe.d/<filename>.conf` 文件中增加「softdep」行。例如：

```
softdep snd-pcm post: snd-pcm-oss
```

请注意「softdep」也支持 `pre:` 的依赖方式，或者混合 `pre:` 和 `post:`。查看 `modprobe.d(5)` 手册了解更多关于「softdep」语法和功能的信息。

如果问题模块并非一个封装，其本身也是有用的话，配置 modules 开机脚本在引导系统的时候加载模块。这样需要把模块名字添加到 `/etc/sysconfig/modules` 文件里的单独一行。这也可以用于封装模块，但是仅作为备选方案。

7.3.3.3. udev 加载了一些无用模块

要么不要编译该模块，要么把它加入到模块黑名单 `/etc/modprobe.d/blacklist.conf` 里，如下的例子中屏蔽了 forte 模块：

```
blacklist forte
```

被屏蔽的模块仍然可以用 modprobe 强行加载。

7.3.3.4. udev 创建了错误的设备节点，或错误的软链接

这个情况通常是因为设备匹配错误。例如，一条写的不好的规则可能同时匹配到 SCSI 磁盘（希望加载的）和对应厂商的 SCSI 通用设备（错误的）。找出这条问题规则，并通过 udevadm info 命令的帮助改得更具体一些。

7.3.3.5. udev 规则不能可靠的工作

这可能是上个问题的另一种表现形式。若非如此，而且你的规则使用了 sysfs 特性，那可能是内核时序问题，希望在后面版本的内核中能得以解决。目前的话，你可以暂时建立一条规则等待使用的 sysfs 特性，并附加到 /etc/udev/rules.d/10-wait_for_sysfs.rules 文件里（如果没有这个文件就创建一个）。如果你是这样做的，并且起作用了，请务必通过 LFS 开发邮件列表通知我们。

7.3.3.6. udev 没有创建设备

后面的内容会假设驱动已经静态编译进内核或已经作为模块加载，而且你也已经确认 udev 没有创建相应的设备节点。

如果内核驱动没有将一个设备的信息导出至 sysfs 系统，则 udev 无法创建相应的设备结点。这种情况经常会在内核树之外的第三方驱动程序中出现。其解决方法是在文件 /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结点（相应的设备号可以在内核文档中的 devices.txt 文件或者由第三方驱动程序的文档中找到）。之后 udev 会根据这些信息在 /dev 中创建一个静态设备结点。

7.3.3.7. 设备名称顺序在重启后随机改变

这是因为 udev 被设计成并行处理 uevents 并加载模块，所以是不可预期的顺序。这个不会被「修复」。你不应该依赖稳定的内核模块名称。而是在检测到设备的稳定特征，比如序列号或 udev 安装的一些 *_id 应用的输出，来判断设备的稳定名称，之后创建自己的规则生成相应的软链接。可以参考 Section#7.4, “设备管理”和 Section#7.2, “通用网络配置”。

7.3.4. 有用的读物

其他的帮助文档可以参考下面的链接：

- devfs 的用户空间实现： http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf
- sysfs 文件系统： http://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf

7.4. 设备管理

7.4.1. 处理相似性质的设备

正如 Section#7.3, “设备与模块管理概述”中解释的，具有相同功能的设备出现在 /dev 目录下的顺序是随机的。假如你有一个 USB 摄像头和一个电视调谐器，/dev/video0 有可能是 USB 摄像头， /dev/video1 是电视调谐器，有时候又可能是反过来的。对于除声卡和网卡外的设备， 都可以通过创建自定义持久性符号链接的 udev 规则来固定。网卡如何设置请看 Section#7.2, “通用网络配置”, 网卡的相关设置请看 BLFS。

对于你所有的硬件，都有可能遇到此问题（尽管此问题可能在你当前的 Linux 发行版上不存在），在 /sys/class 或是 /sys/block 目录下找到对应目录，比如，显卡可能的路径为 /sys/class/video4linux/videoX。找到该设备的唯一设备标识（通常，厂商和产品 ID 以及/或 序列号会有用）：

```
udevadm info -a -p /sys/class/video4linux/video0
```

然后通过写入规则建立符号链接：

```
cat > /etc/udev/rules.d/83-duplicate_devs.rules << "EOF"

# Persistent symlinks for webcam and tuner
KERNEL=="video*", ATTRS{idProduct}=="1910", ATTRS{idVendor}=="0d81", \
    SYMLINK+="webcam"
KERNEL=="video*", ATTRS{device}=="0x036f", ATTRS{vendor}=="0x109e", \
    SYMLINK+="tvtuner"

EOF
```

最终，/dev/video0 和 /dev/video1 依旧会随机分配给 USB 摄像头和电视调谐器，但是 /dev/tvtuner 和 /dev/webcam 将会固定的分配给正确的设备。

7.5. 配置系统时间

本节将会讨论如何配置 systemd-timedated 系统服务，包括配置系统时间和时区。

如果你不确定是否将硬件时钟设置为 UTC，可以通过 **`hwclock --localtime --show`** 来查看。 这将根据硬件时钟显示当前的时间。如果显示的和手表的时间相同，应该是设置为本地时间了； 如果 hwclock 输出的时间不一致，应该是设置为 UTC 了。通过增减数小时，可以确定你所在时区。

systemd-timedated 读取 `/etc/adjtime`， 然后确定是本地时间还是 UTC 时间。

如果想要将硬件时钟设置为本地时间，使用以下命令建立 `/etc/adjtime` ：

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

如果第一次启动时 `/etc/adjtime` 文件不存在， systemd-timedated 会认为硬件时钟设置成 UTC 并且以此调整该文件。

你也可以使用 timedatectl 程序来告诉 systemd-timedated 你的硬件时钟是 UTC 还是本地时间：

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

timedatectl 也可以用来更改系统时间和时区。

要更改当前系统时间，使用下面的命令：

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

硬件时钟也会相应更新。

要更改当前时区，使用下面命令：

```
timedatectl set-timezone TIMEZONE
```

你可以通过运行下面命令查看可用时区列表：

```
timedatectl list-timezones
```

> **Note**
>
> timedatectl 仅可用于 systemd 环境。

7.5.1. 网络时间同步

从版本号 213 的 systemd 开始，包括了一个名为 systemd-timesyncd 的守护进程，将能够和远程的 NTP 服务器同步时间。

此守护进程并不是为了取代已有的 NTP 服务，而是作为 SNTP 协议的实现客户端，它可用于更高级的任务和资源有限的系统。

从版本号 216 的 systemd 开始，systemd-timesyncd 守护进程默认启用，如果需要禁用，执行以下命令：

```
systemctl disable systemd-timesyncd
```

systemd-timesyncd 也可以通过配置 `/etc/systemd/timesyncd.conf` 文件同步更改 NTP 服务。

需要注意的是，当使用本地时间作为系统时间时，systemd-timesyncd 无法更新硬件时钟。

7.6. 配置 Linux 控制台

（译者注：对于中文用户，因为大部分用户的键盘布局都是标准 us 键盘，所以大可忽略此文件的配置）

本节将会讨论如何配置 systemd-vconsole-setup 系统服务，包括配置控制台字体和控制台的键盘映射。

systemd-vconsole-setup 服务读取 `/etc/vconsole.conf` 中的配置信息， 确定使用的键盘类型和屏幕的字体。http://www.tldp.org/HOWTO/HOWTO-INDEX/other-lang.html 页面有很多其它语言的「HOWTO」内容可以给你很多帮助。 localectl list-keymaps 可以列举出所有可用的终端键盘布局。`/usr/share/consolefonts` 目录提供了所有可用的字体。

`/etc/vconsole.conf` 的每一行都应该形如：VARIABLE="value"。以下列举了可用的 VARIABLE：

KEYMAP
此变量指定了键盘的按键映射表。如未设置，默认为 us。
KEYMAP_TOGGLE
此变量指定配置第二个切换键盘映射，默认不设置。
FONT
此变量指定虚拟控制台的字体。
FONT_MAP
此变量指定要使用的控制台映射。
FONT_UNIMAP
此变量指定 Unicode 字体映射。

以下为德语键盘和和控制台的实例：

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

你可以使用 localectl 实用程序更改 KEYMAP 值：

```
localectl set-keymap MAP
```

> **Note**
>
> localectl 仅可用于 systemd 环境。

localectl 实用程序跟随相应的参数也可以更改 X11 的键盘布局、模型、变形和选项：

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

下面的 localectl 命令列出了 localectl set-x11-keymap 命令参数的可用值：

list-x11-keymap-models
显示所有已知的 X11 键盘映射模型。
list-x11-keymap-layouts
显示所有已知的 X11 键盘映射布局。
list-x11-keymap-variants
显示所有已知的 X11 键盘映射变形。
list-x11-keymap-options
显示所有已知的 X11 键盘映射选项。

> **Note**
>
> 想要使用以上的参数，你需要从 BLFS 中安装 Xkeyboard 软件包。

7.7. 配置系统语言环境

本地语言的支持依赖于 `/etc/locale.conf`，它包含不少和此相关的环境变量。更改此文件后，可能会出现以下的变化：

- 程序的输出将以本地语言展示
- 修正字符在字母、数字等类型的分类。对于非英语区域设置来说，只有这样，bash 才能正常显示非 ASCII 字符
- 国家顺序可以按照字母顺序正常排序（译者注：这里所谓的正常排序，应该是首先将国家名称转换成本地语言的国家名称，比如中国的开头字母是 Z，那么显示就很靠后了。但是假如显示为英文，就是 C 开头，那么将会靠前显示。）
- 默认纸张尺寸
- 货币、时间和日期值的格式

/etc/locale.conf 中 <*ll*> 使用语言代码代替（比如中文是「zh」，英文是「en」），<*CC*> 使用国家/地区代码代替（比如中国是「CN」，中国香港是「HK」，中国台湾是「TW」，美国是「US」），<*charmap*> 使用选定字符集指定的标准字符映射表替换。诸如「@euro」这样的可选修饰符也可使用。

运行以下命令可以获得当前 Glibc 支持的本地字符集。

```
locale -a
```

字符映射表可能存在很多的别名，比如「ISO-8859-1」可以写作「iso8859-1」或「iso88591」。 但是有一些程序不支持这些乱七八糟的写法（比如「UTF-8」只能写作「UTF-8」，「utf8」它就不认识了）。 所以，为了安全起见，在设置的时候还是尽量的使用特定区域设置的规范名称。可以通过以下命令查询在特定区域下的字符映射表标准名称，*<locale name>* 为运行 locale -a 输出的首选区域设置（这里以「zh_CN.utf8」为例）。

```
LC_ALL=<locale name> locale charmap
```

对于「zh_CN.utf8」以上命令将会如下输出：

```
UTF-8
```

根据以上输出，我们再次修改 `/etc/locale.conf`， 将字符映射表设置为标准形式（「zh_CN.utf8」变为「zh_CN.UTF-8」）。 同理，也可一并查询以下设置的标准命令，然后将其添加到 bash 的启动文件中 （译者注：对于 bash 来说，启动文件为 `.bashrc`）。

```
LC_ALL=<locale name> locale language
LC_ALL=<locale name> locale charmap
LC_ALL=<locale name> locale int_curr_symbol
LC_ALL=<locale name> locale int_prefix
```

以上的命令将会打印当前区域设置的语言、字符编码、本地货币单位以及电话国际编码。如果出现类似下文的错误输出， 可能是你没有严格按照第六章指导的方法操作或者是你当前所用的 Glibc 不支持。

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

如果这种情况真的发生，你应该使用 localedef 命令安装对应的系统区域， 或者是考虑更改为其它的区域。假如没有出现错误提示，我们就可以继续进行下一步操作了！

有一些 LFS 之外的包可能出现对你设置的区域支持很差劲的情况。比如 X 的库（X Windows System 的一部份）， 就可能在内部文件中输出以下消息：

```
Warning: locale not supported by Xlib, locale set to C
```

在若干情况下，Xlib 希望以带规范破折号的大写形式列出字符映射表。比如「ISO-8859-1」而不应该写作「iso88591」。不过， 也可以通过去除区域规范中的字符映射部分找到合适的规范。这可以通过运行 locale charmap 命令来检查。 例如，需要更改「de_DE.ISO-8859-15@euro」为「de_DE@euro」以便 Xlib 能识别区域。

即便如此，也可能遇到某些程序因为区域设置和它们预置的不同而导致功能异常（可能不会显示任何的错误消息）。 如果出现这样的情况，可以通过查看其它的发行版是如何进行设置区域，从而得到启发。

一旦确定了到底该使用哪个区域设置，就可以创建 `/etc/locale.conf` 文件了：

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

也可以通过 systemd 提供的实用程序 localectl 修改 `/etc/locale.conf`：

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

此处，也可以指定其它和语言相关的环境变量，例如 `LANG`，`LC_CTYPE`，`LC_NUMERIC` 或任何其它 locale 输出的环境变量，只需要用一个空格分开它们就可以了。以下示例中，`LANG` 设置为 en_US.UTF-8，但 `LC_CTYPE` 设置为 en_US：

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

> **Note**
>
> localectl 仅可用于 systemd 环境。

「C」（默认）和「en_US」（推荐美国英语用户使用）这两种区域设置有所不同。「C」使用 US-ASCII 7 位字符集，并把设置了最高位的字节作为无效字符。这就是为什么类似 ls 的命令本地化时会用疑问号代替。同样，如果你想要使用 Mutt 或 Pine 发送包含有类似字符的邮件，将会得到如下消息：非 RFC 兼容字符（发送邮件中的字符集为「unknown 8-bit」）。所以，如果你一定以及肯定一定不会用到 8 位的字符，那你可以仅使用「C」。

不少程序还不支持 UTF-8 区域设置。我们正在完善文档并修复类似问题，可以查看http://www.linuxfromscratch.org/blfs/view/8.4/introduction/locale-issues.html.

7.8. 创建 /etc/inputrc 文件

`inputrc` 文件的作用是告知系统应该以怎样的键盘布局处理键盘。此文件对于 readline —— 输入相关库，或者是一些 shell （例如 bash 等）来说十分重要。

对于大部分用户来说，都不使用那些奇奇怪怪的键盘映射，所以，可以通过以下的命令建立一个全局的 `/etc/inputrc` 以供所有用户使用。如果需要更改某一个用户的键盘映射，仅需要在那个用户的 HOME 目录下建立一个 `.inputrc` 文件，然后修改对应的键盘映射就可以了。

如果需要了解更多有关如何编辑 `inputrc` 文件的信息，可以查看 info bash 中 Readline Init File 一节。其实查看 info readline 也可以获得不少有用的东西。

下面显示的就是通用的 `inputrc` 文件，其中包含有 “#” 的都是注释行（需要注意的是，此文件不支持在设置后跟随注释）。使用以下命令创建此文件：

```
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
```

7.9. 创建 /etc/shells 文件

`shells` 文件是当前系统所有可用 shell 的列表文件。应用程序通过读取它可以知道需要使用的 shell 是否有效。每行指定一个 shell 的绝对路径，即从根目录（/）开始的路径。

例如，当非特权用户想要使用 chsh 命令更改自己登录所用的 shell 时。如果命令没有在 /etc/shell 中找到，那么将会拒绝更改。

这个文件对于某些程序来说是必需的，比如 GDM 在找不到 /etc/shells 时就不会启用面部浏览器，还有 FTP 守护进程通常会禁止使用不在这个文件里列出终端的用户登录。

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

/bin/sh
/bin/bash

# End /etc/shells
EOF
```

7.10. systemd 的用法与配置

7.10.1. 基础配置

/etc/systemd/system.conf 文件包含了大量的 systemd 控制命令。 假如未作任何的更改，文件中的所有行应该都是注释掉的，这代表了 systemd 正使用默认的运行方式。 这个文件中可以设置日志级别，可以修改日志的基本设置。所有设置项都可以在 man 手册的 systemd-system.conf(5) 中查看。

7.10.2. 禁用启动时清屏

默认情况下，systemd 将会在系统启动快要结束的时候清屏。如不需要，使用以下操作禁用：

```
mkdir -pv /etc/systemd/system/getty@tty1.service.d

cat > /etc/systemd/system/getty@tty1.service.d/noclear.conf << EOF
[Service]
TTYVTDisallocate=no
EOF
```

拥有 root 权限的账户可以通过 **journalctl -b** 查看启动信息。

7.10.3. 禁止 /tmp 使用 tmpfs

默认情况下，/tmp 使用 tmpfs 文件系统。如不需要，使用以下操作禁用：

```
ln -sfv /dev/null /etc/systemd/system/tmp.mount
```

或许，如果你需要一个单独的 /tmp 分区，那请在 /etc/fstab 条目中指定该分区。

> **Warning**
>
> 如果使用的是独立的 /tmp 分区，就不要创建上述链接。这将保护根（/）文件系统，使其在重新挂载后失去 r/w 权限，从而导致系统在启动后无法使用。

7.10.4. 配置自动创建和删除文件

有这样几个服务可以建立或删除文件/目录：

- systemd-tmpfiles-clean.service
- systemd-tmpfiles-setup-dev.service
- systemd-tmpfiles-setup.service

系统配置文件在 /usr/lib/tmpfiles.d/*.conf 中。 本地配置文件在 /etc/tmpfiles.d 中。 /etc/tmpfiles.d中的文件会覆盖 /usr/lib/tmpfiles.d 中相同名称的文件。 （译者注：首先读取系统范围配置文件，再读取用户范围配置文件，用户范围配置文件会覆盖系统范围配置文件的相同部分。）可以在 man 手册的 tmpfiles.d(5) 中获取文件格式详情。

注意，/usr/lib/tmpfiles.d/*.conf 文件的语法比较容易搞错。例如，默认对于 /tmp 目录下文件的删除，源自 /usr/lib/tmpfiles.d/tmp.conf 中的这一行：

```
q /tmp 1777 root root 10d
```

类型字段 q，用于创建带配额的子卷，仅适用于 btrfs 文件系统。它会引用类型 v，然后引用类型 d （目录）。然后创建指定的目录，如果它原先不存在的话，并为其调整权限，指定所有权。

如果默认参数不能达到要求，那就复制文件至 /etc/tmpfiles.d 按照自己的需求修改它。复制示例：

```
mkdir -p /etc/tempfiles.d
cp /usr/lib/tmpfiles.d/tmp.conf /etc/tempfiles.d
```

7.10.5. 覆盖默认服务的行为

可以通过在 /etc/systemd/system 中创建目录和配置文件来覆盖单元的参数。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
```

可以在 man 手册的 systemd.unit(5) 中查询更多信息。创建好文件之后，请运行 **systemctl daemon-reload** 和 **systemctl restart foobar** 激活所做更改。

7.10.6. 调试启动顺序

相较于 SysVinit 或 BSD 风格的初始化使用的是简单的 shell 脚本，而 systemd 对于不同类型的启动文件（或单元）使用的是统一的格式。systemctl 命令可用于用于，启用，禁用，控制状态，和获取单元文件的状态。以下是一些常用命令的示例：

- systemctl list-units -t ***<service>*** [--all]：列出加载的服务类型的单元文件。
- systemctl list-units -t ***<target>*** [--all]：列出加载的目标类型的单元文件。
- systemctl show -p Wants ***<multi-user.target>***：显示所有依赖多用户的目标。目标是与 SysVinit 运行级不同的特殊单元文件。
- systemctl status ***<servicename.service>***：显示服务名称服务的状态。扩展名 .service 在没有同名单元文件，例如 .socket（创建监听套接字功能与 inetd/xinetd 相似的）文件，的情况下可以省略。

7.10.7. 使用 systemd 日志

systemd 启动的系统上的日志记录（默认）由 systemd-journald 处理的，而不是传统的 unix syslog 守护程序。如果需要，你也可以添加 syslog 的守护进程与之并行运作。systemd-journald 程序使用二进制形式存储日志条目而非纯文本的日志文件。为了帮助解析文件，提供了 journalctl 命令。以下是一些常用命令的示例：

- journalctl -r：按时间逆序现实所有的内容。
- journalctl -u ***UNIT***：显示与指定 UNIT 相关的日志条目。
- journalctl -b[=ID] -r：按时间逆序显示自上次成功启动（或启动 ID）的日志条目。
- journalctl -f：提供类似与 tail -f (follow) 类似的功能。

7.10.8. 长时间运行进程

自 systemd-230 开始，用户的所有进程都会随着用户会话的结束而被杀死，即使你用了 nohup，或进程使用了 daemon() 或是 setsid() 功能。这是为了是让原先宽松的环境变得更为严格而故意为之的。如果你依赖长时间运行的程序（例如，screen 或是 tmux），需要在你结束用户会话后保持运作。有三个方法能让进程保持在用户会话结束后仍然逗留。

- 仅为需要的用户启用进程逗留：正常用户有权使用命令 loginctl enable-linger 来为其启用进程逗留。系统管理员可以使用相同的命令伴随着 *user* 参数，来帮某个用户实现。用户可以在之后使用命令 systemd-run 来启动长时间运行进程。举个例子：systemd-run --scope --user /usr/bin/screen。如果你为你的用户启用了逗留，user@.service 会在所有的登录会话被关闭后仍然保留，并在系统启动后自动启动。这样做的好处是明确地表明是否允许进程在用户会话结束后继续运行，但却打破了对于类似于 nohup 之类工具和使用 deamon() 的公用方法的向后兼容。
- 启用系统范围的进程逗留：你可以在 /etc/logind.conf 中设置 *KillUserProcesses=no* 来全局地为每个用户启用进程逗留。这在牺牲明确控制的同时，带来了将旧方法提供给所有用户的好处。
- 在构建时禁用：你可以通过在构建 systemd 时向命令 meson 添加 *-Ddefault-kill-user-processes=no*，以此来默认启用逗留。这会使 systemd 完全丧失在用户会话结束后杀死进程的能力。

Chapter 8. 让 LFS 系统可引导

8.1. 简介

是时候该让 LFS 系统可以启动了。本章节将讨论以下内容：创建 fstab 文件、为新的 LFS 系统编译内核、安装 GRUB 引导器。如此，就可以在电脑启动的时候选择启动 LFS 系统了。

8.2. 创建 /etc/fstab 文件

/etc/fstab 文件的作用是让其它程序确定存储设备的默认挂载点、挂载参数和检查信息（例如完整性检测）。仿照以下格式新建一个文件系统列表（file system table，简称 fstab）文件：

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

# file system  mount-point  type     options             dump  fsck
#                                                              order

/dev/<xxx>     /            <fff>    defaults            1     1
/dev/<yyy>     swap         swap     pri=1               0     0

# End /etc/fstab
EOF
```

其中，<*xxx*>，<*yyy*> 和 <*fff*> 请使用适当的值替换。例如 sda2，sda5 和 ext4。关于文件中六个字段的含义，请查看 man 5 fstab（译者注：fsck 列的数值来决定需要检查的文件系统的检查顺序。允许的数字是0, 1, 和2。根目录应当获得最高的优先权 1, 其它所有需要被检查的设备设置为 2。0 表示设备不会被 fsck 所检查）。

基于 MS-DOS 或者是来源于 Windows 的文件系统（例如：vfat，ntfs，smbfs，cifs，iso9660，udf）需要在挂载选项中添加「iocharset」，才能让非 ASCII 字符的文件名正确解析。此选项的值应该与语言区域设置的值相同，以便让内核能正确处理。此选项在相关字符集定义已为内核内建或是编译为模块时生效（在文件系统 -> 本地语言支持中查看）。此外，vfat 和 smbfs 还需启用「codepage」支持。例如，想要挂载 USB 闪存设备，zh-CN.GBK 用户需要在 /etc/fstab 中添加以下的挂载选项：

```
noauto,user,quiet,showexec,iocharset=gbk,codepage=936
```

对于 zh_CN.UTF-8 用户的对应选项是：

```
noauto,user,quiet,showexec,iocharset=utf8,codepage=936
```

需要注意的是，iocharset 默认值是 iso8859-1 （其保证文件系统大小写敏感），而 utf8 这个参数告知内核使用 UTF-8 转换文件名，以便可以在 UTF-8 语言环境中解释它们。

此外，还有可能在内核的配置过程中就指定一些文件系统的默认代码页和 iocharset 值。相关参数有「默认 NLS 选项 」（CONFIG_NLS_DEFAULT），「默认远程 NLS 选项」（CONFIG_SMB_NLS_DEFAULT），「FAT 默认代码页」（CONFIG_FAT_DEFAULT_CODEPAGE），和「FAT 默认 IO 字符集」（CONFIG_FAT_DEFAULT_IOCHARSET）。不过，无法在内核编译阶段指定 ntfs 文件系统的设置。

另外，一些硬盘类型在遇到电源故障时，假如在 /etc/fstab 中使用 barrier=1 这个挂载选项，则会让 ext3 文件系统的数据更加安全。如需检查磁盘是否支持此选项，请运行 hdparm。例如：

```
hdparm -I /dev/sda | grep NCQ
```

如果有输出内容，则代表选项可用。

注意：基于 逻辑卷管理（LVM）的分区不可使用 barrier 选项。

8.3. Linux-4.20.12

Linux 软件包包含 Linux 内核。

大致构建用时: 4.4 - 66.0 SBU （通常约为 6 SBU）
所需磁盘空间: 960 - 4250 MB （通常约为 1100 MB）

8.3.1. 安装内核

编译内核包括以下步骤——配置、编译和安装。阅读内核源码树中的 README 可以获得替代本手册配置的方法。

运行以下命令准备编译：

```
make mrproper
```

这将保证内核树的绝对干净。内核小组建议在每次编译之前都执行此命令，无用的代码将会在解压后删除。

通过菜单界面配置内核。配置内核的一般信息请查看：http://www.linuxfromscratch.org/hints/downloads/files/kernel-configuration.txt。 BLFS 包含有一些内核的特殊配置，可以查看：http://www.linuxfromscratch.org/blfs/view/8.4/longindex.html#kernel-config-index。 内核配置和编译的附加信息可查看：http://www.kroah.com/lkn/。

Note

配置内核的一个好的起点是运行 make defconfig。这样会参考你的机器架构生成一份基本能用的基础配置。

注意要确保启用/禁用/设置下面这些特性，否则系统也许不能正常工作甚至根本无法启动：

```
General setup -->
   [ ] Enable deprecated sysfs features to support old userspace tools [CONFIG_SYSFS_DEPRECATED]
   [ ] Enable deprecated sysfs features by default [CONFIG_SYSFS_DEPRECATED_V2]
   [*] open by fhandle syscalls [CONFIG_FHANDLE]
   [ ] Auditing support [CONFIG_AUDIT]
   [*] Control Group support [CONFIG_CGROUPS]
Processor type and features  --->
   [*] Enable seccomp to safely compute untrusted bytecode [CONFIG_SECCOMP]
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]
   [ ] Fallback user-helper invocation for firmware loading [CONFIG_FW_LOADER_USER_HELPER]
Firmware Drivers  --->
   [*] Export DMI identification via sysfs to userspace [CONFIG_DMIID]
File systems  --->
   [*] Inotify support for userspace [CONFIG_INOTIFY_USER]
   <*> Kernel automounter version 4 support (also supports v3) [CONFIG_AUTOFS4_FS]
  Pseudo filesystems  --->
   [*] Tmpfs POSIX Access Control Lists [CONFIG_TMPFS_POSIX_ACL]
   [*] Tmpfs extended attributes [CONFIG_TMPFS_XATTR]
Kernel hacking  --->
       Choose kernel unwinder (Frame pointer unwinder)  ---> [CONFIG_UNWINDER_FRAME_POINTER]
```

Note

尽管「The IPv6 Protocol」不是必须打开，但是 systemd 开发人员强烈推荐打开。

Note

如果你主机的硬件用的是 UEFI，那么上面的'make defconfig'应该会自动添加一些 EFI 相关的内核选项。

为了让你的 LFS 内核，在你的主机是 UEFI 引导环境的情况下，能够被引导，你的内核必须要有这项：

```
Processor type and features  --->
   [*]   EFI stub support  [CONFIG_EFI_STUB]
```

文件 lfs-uefi.txt 中包含了管理 UEFI 环境的完整描述，参见 http://www.linuxfromscratch.org/hints/downloads/files/lfs-uefi.txt。

上述配置项的一些原理说明：

Support for uevent helper
打开这个选项会影响 udev/Eudev 设备管理。

Maintain a devtmpfs
这个选项允许内核在 udev 运行之前就创建自动设备节点。之后 udev 在这个基础上运行，管理权限以及增加软链接。对于所有 udev/Eudev 用户，这个配置项是必须的。

```
make menuconfig
```

可选 make 环境变量的含义：

LANG=<host_LANG_value> LC_ALL=
建立与宿主系统相同的地域设定。在 UTF-8 linux 文本命令行上逐行绘制适宜的 menuconfig ncurses 接口时可能需要这项配置。
要使用的话，请我务必使用宿主系统中的变量 $LANG 去代替 *<host_LANG_value>*。你也可以用宿主系统中的 $LC_ALL 或 $LC_CTYPE 来代替。

另外，make oldconfig 在某些情况下可能更合适。查看 README 文件了解更多信息。

想偷懒的话，可以拷贝宿主系统的内核配置文件 .config（如果有的话）到解压后的 linux-4.20.12 目录下来跳过内核配置。不过，我们不建议这样做。最好是探索一下整个内核配置菜单，从最开始配置内核。

编译内核映像和模块：

```
make
```

如果使用内核模块，需要 /etc/modprobe.d 文件里的模块配置。关于模块和内核配置的信息可以查看 Section#7.3, “设备与模块管理概述”以及 linux-4.20.12/Documentation 目录下的内核文档。还有，modprobe.d(5) 也可以看一下。

如果内核配置里用到，需要安装模块：

```
make modules_install
```

在内核编译完成后，还需要一个额外步骤来完成安装。有些文件需要拷贝到 /boot 目录下。

Caution

如果宿主系统拥有单独的 /boot 分区，那么文件就应该复制到那里。简单的解决方法就是在执行前将 /boot 绑定到宿主的 /mnt/lfs/boot。以宿主系统中的 root 用户运行：

```
mount --bind /boot /mnt/lfs/boot
```

内核映像文件所在的实际目录根据主机系统架构可能会不一样。下面的文件名你也可以改成你喜欢的，不过开头最好是 vmlinuz 才可以兼容下一节要讲的配置引导过程的自动设定。下面的命令假设主机是 x86 架构：

```
cp -iv arch/x86/boot/bzImage /boot/vmlinuz-4.20.12-lfs-8.4-systemd
```

System.map 是内核的符号文件。它映射了每一个内核 API 函数的入口，以及内核运行时的数据结构地址。是调查内核问题时的资源。运行下面的命令安装映射文件：

```
cp -iv System.map /boot/System.map-4.20.12
```

在之前命令 make menuconfig 里生成的内核配置文件 .config 包含了当前编译的内核的所有配置。最好能保存下来留作参考：

```
cp -iv .config /boot/config-4.20.12
```

安装 Linux 内核文档：

```
install -d /usr/share/doc/linux-4.20.12
cp -r Documentation/* /usr/share/doc/linux-4.20.12
```

需要注意一下内核源代码目录下的文件属主并不是 root。在以 root 用户解压包的时候（我们在 chroot 环境里做的），解压出来的文件会拥有生成这个包的电脑里用户和组。在安装其他包的时候这并不是问题，因为它们的源代码在安装完后就删除了。不过，Linux 内核的源代码经常会保留比较长时间。这样的话，就有可能会把软件包作者的用户 ID 对应到本机的某个用户上。从而这个用户就会拥有内核源代码的写权限。

> **Note**
>
> 在很多情况下，内核的配置信息需要在稍后安装来自于 BLFS 的软件包后更新。这和其他的软件包不同，在安装完成编译好的内核后不需要将内核源码树删除。
>
> 如果想要保留内核的源码树，在 linux-4.20.12 下运行 chown -R 0:0 来确保所有文件的所有者都 root。

> **Warning**
>
> 一些内核文档里建议创建软链接 /usr/src/linux 指向内核源代码目录。这是 2.6 及以前版本内核的特定要求，而在 LFS 系统里一定不要创建这个链接，因为这样的话，在你的基础 LFS 系统完成后安装某些软件包时可能引起问题。

> **Warning**
>
> 系统 include 目录（/usr/include）下的头文件应该总是和编译 Glibc 时用到的头文件保持一致。就是在 Section#6.7, “Linux-4.20.12 API 头文件”里整理过的头文件。因此，它们不要替换成原始内核头文件或任何清理过的内核头文件。

8.3.2. 配置 Linux 模块加载顺序

虽然大多数情况下，Linux 模块会被自动加载，但是有时候需要特别指定加载顺序。modprobe 或 insmod 在加载模块时会读取 /etc/modprobe.d/usb.conf。如果将 USB 设备（ehci_hcd、ohci_hcd 和 uhci_hcd）编译为模块，则需要此文件，这样它们就会以正确的顺序加载。ehci_hcd 需要在 ohci_hcd 和 uhci_hcd 之前加载，否则在系统启动过程中将会输出警告。

运行以下命令建立 /etc/modprobe.d/usb.conf 文件：

```
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
```

8.3.3. Linux 的内容

安装的文件: config-4.20.12, vmlinuz-4.20.12-lfs-8.4-systemd, and System.map-4.20.12
安装的目录: /lib/modules, /usr/share/doc/linux-4.20.12

简要说明

config-4.20.12 包含内核的所有配置选项

`vmlinuz-4.20.12-lfs-8.4-systemd`	Linux 系统的引擎。当电脑启动时，内核作为整个系统的第一部分载入。它首先检测和初始化所有的电脑硬件，然后将这些硬件模块抽象成文件树让软件访问，并把单个 CPU 转换成多任务系统，可以看上去同时地运行多个程序
`System.map-4.20.12`	地址和符号列表；包含有入口点的映射以及所有函数和内核数据结构的地址

8.4. 使用 GRUB 设置启动过程

8.4.1. 简介

> **Warning**
>
> 对 GRUB 进行错误的配置可能会导致在没有外置启动设备（某些USB设备，某些 CD-ROM 等）的情况下无法正常的启动。你可能仅仅需要修改当前正在使用的引导器（比如：Grub-Legacy，GRUB2 或 LILO 等）的配置。

一定要确保你有一个可以引导的光盘或者是 U 盘以备不时之需，否则万一电脑真的不能启动谁都救不了你。如果你需要建立可引导的设备，可以按照以下方法操作（跳转到 BLFS 从 libisoburn 软件包安装 **xorriso**）。

```
cd /tmp
grub-mkrescue --output=grub-img.iso
xorriso -as cdrecord -v dev=/dev/cdrw blank=as_needed grub-img.iso
```

> **Note**
>
> 想要在启用 UEFI 的宿主系统上引导 LFS，需要遵循上一节的描述在内核中构建 CONFIG_EFI_STUB 功能。也可以使用 GRUB2 来引导 LFS，这样就无需实施以上的额外操作了。但需要在宿主系统的 BIOS 中关闭 UEFI 模式和安全引导（Secure Boot）功能。详细参见文件 lfs-uefi.txt。

8.4.2. GRUB 命名约定

GRUB 对于硬盘和分区自有一套命名规则 (hdn,m)，其中 n 是硬盘数，m 是分区号。硬盘数 N 从 0 开始计数，分区数需要区别对待——主分区从 1 开始计数而扩展分区从 5 开始计数。需要注意的是，和早期版本相比，计数方式都有所变化。例如，分区 `sda1` 是 (hd0,1)，`sdb3` 是 (hd1,3)。Linux 下，并不将 CD-ROM 设备假想为硬盘。例如，就算已有 CD 设备挂载为 `hdb`，第二块硬盘挂载为 `hdc`，GRUB 依旧将第二块硬盘称为 (hd1)。

8.4.3. 配置配置文件

GRUB 会将一些数据写入硬盘的第一个物理扇区。这一部分不属于任何一个操作系统，在启动时，该部分数据激活，然后寻找 Grub 的模块，Grub 模块的默认位置为 /boot/grub/。

一种建议是使用一个独立的小分区（建议大小 100MB）专用于引导信息。那样的话，每一个发行版，不论是 LFS 还是其他的商业发行版，都能访问相同的引导文件而且任何已经启动的系统都能访问它。如果你选择这么做，你需要挂载这个独立分区，移动所有的文件从当前的 `/boot` 目录（比如说你上一节刚编译的 Linux 内核）到新的分区。你然后要卸载这个新分区，重新挂载它为 `/boot`。如果你这么做，一定要更新 `/etc/fstab`。

使用当前的 lfs 分区也没有什么问题，但是在配置多系统启动的时候有些不同。

从以上信息可知，需要确定根分区的磁盘位置（如果使用单独的分区，则需要知道引导分区的磁盘位置），以下假定根分区（或者是磁盘分区）是 `sda2`。

将 GRUB 文件安装到 `/boot/grub` 然后设置启动扇区：

> **Warning**
>
> 以下命令将会覆盖已有的引导器。如无需要，请勿运行（比如已经有第三方引导器管理 MBR）。

```
grub-install /dev/sda
```

> **Note**
>
> 如果系统是通过 UEFI 引导的，grub-install 将会尝试将文件安装至 x86_64-efi，但是这些文件并未在第 6 章中安装。如果是在这种情况下，在上述命令后面追加 `--target i386-pc`。

8.4.4. 创建 GRUB 配置文件

创建 `/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 4.20.12-lfs-8.4-systemd" {
        linux   /boot/vmlinuz-4.20.12-lfs-8.4-systemd root=/dev/sda2 ro
}
EOF
```

> **Note**
>
> 从 GRUB 的角度看，内核文件相当于一个分区，所以假如你使用单独的 /boot 分区，请不要在 linux 行添加 /boot。此外还需要将 set root 行指向 /boot 所在的实际分区。

GRUB 功能十分强大，它提供了大量的用于从种类繁多的设备和操作系统、以及不同的分区类型启动的选项。此外还可以定制启动页、播放声音或者是鼠标等。很遗憾的是，这些功能超出本文的范畴，我们一概不予讨论。

> **Caution**
>
> grub-mkconfig 命令可以自动建立配置文件。它使用位于 /etc/grub.d/ 下的一组脚本且将会忽略用户的设置。这些命令主要用于那些非源码编译的发行版，所以不建议 LFS 用户使用此命令。如果你使用商业发行版，你可以试着运行这个命令（运行之前记得备份原来的 grub.cfg 文件）。

Chapter 9. 尾声

9.1. 最后的最后

干的很棒嘛！至此，全新的 LFS 系统就已经安装完成啦！我们衷心地祝愿，你亲手定制的崭新 Linux 系统能陪着你乘风破浪。

创建一个 systemd 所需的 `/etc/os-release` 文件：

```
cat > /etc/os-release << "EOF"
NAME="Linux From Scratch"
VERSION="8.4-systemd"
ID=lfs
PRETTY_NAME="Linux From Scratch 8.4-systemd"
VERSION_CODENAME="<your name here>"
EOF
```

建议和非 systemd 分支一样创建文件 `/etc/lfs-release`。该文件能帮助你（和我们，如果你需要我们帮助的话）确定你当前使用的 LFS 版本。运行以下命令以创建该文件：

```
echo 8.4-systemd > /etc/lfs-release
```

推荐遵守 Linux Standards Base (LSB)，建立文件以显示当前系统的完整信息。运行以下命令新建此文件：

```
cat > /etc/lsb-release << "EOF"
DISTRIB_ID="Linux From Scratch"
DISTRIB_RELEASE="8.4-systemd"
DISTRIB_CODENAME="<your name here>"
DISTRIB_DESCRIPTION="Linux From Scratch"
EOF
```

你可以在「DISTRIB_CODENAME」字段填写一些特别的字符来彰显「你的」系统的与众不同！

9.2. 为 LFS 用户数添砖加瓦

截至此刻，你已经读完了这本书。你想要为 LFS 用户数添砖加瓦吗？赶快点击鼠标访问 http://www.linuxfromscratch.org/cgi-bin/lfscounter.php 输入用户名和第一次使用的 LFS 版本注册成为 LFS 用户吧。

赶快重启到 LFS 吧！

9.3. 重启系统

至此，所有的软件都已安装完毕，是时候重启你的电脑了。然而，你也应该注意一些事情。通过学习本书建立起来的系统属于最小系统，这也就意味着可能会缺失一些你需要的功能。就是说你还需要做些事情。当重启进入你的新 LFS 中时，这是一个在当前的 chroot 环境中安装一些 BLFS 书中的额外软件包的好时机。以下给出了一些建议：

- 文本模式的浏览器，例如 Lynx，可以在虚拟终端中访问这本 BLFS 书，以进行后续的编译打包工作。
- GPM（GPM：一个支持控制台和 xterm 的鼠标服务） 软件包可以让你在虚拟终端中更方便的执行复制/粘贴工作。
- 如果静态 IP 配置不能很好的适用于你当前环境的网络配置，可以安装 dhcpcd 或者是 dhcp 的客户端部分来解决。
- 安装 sudo，以便在非 root 用户环境下编译软件包，且可以很轻松的在新系统中安装编译出来的软件。
- 如果你想要在舒适的 GUI 环境远程访问新系统，请安装 openssh 及其依赖包 openssl。
- 为了更加便利的从网络中下载文件，请安装 wget。
- 如果你有 GUID 分区表（GPT）类型的磁盘，你也许需要 gptfdisk 或是 parted 。
- 最后，检查以下的配置文件是不是都是正确的吧。
  - /etc/bashrc
  - /etc/dircolors
  - /etc/fstab
  - /etc/hosts
  - /etc/inputrc

- /etc/profile
- /etc/resolv.conf
- /etc/vimrc
- /root/.bash_profile
- /root/.bashrc

辛苦了那么久，是该初次启动我们崭新的 LFS 系统的时候了！首先，请退出 chroot 环境：

```
logout
```

然后卸载虚拟文件系统：

```
umount -v $LFS/dev/pts
umount -v $LFS/dev
umount -v $LFS/run
umount -v $LFS/proc
umount -v $LFS/sys
```

卸载 LFS 文件系统本身：

```
umount -v $LFS
```

如果还建立了其它的挂载点，请在卸载 LFS 文件系统之前先卸载它们：

```
umount -v $LFS/usr
umount -v $LFS/home
umount -v $LFS
```

至此，重启系统吧：

```
shutdown -r now
```

这里假设 GRUB 引导器已经如前文所述安装完毕且配置正确，启动项也已经自动设置为 LFS 8.4。

重启后，LFS 便已经可以使用了，你可以安装一些其它的软件以满足自己的需求。

9.4. 接下来做什么呢？

十分感谢你耐心的阅读这本 LFS 书，我们十分期待本书能够为你构建系统带来一点点的帮助。

我猜，你现在一定很开心——LFS系统已经安装完成。「但是，下面该作些什么呢？」不用担心，我们早已经帮你准备好以下资源！

- 维护

  定期检查软件的 bug 和安全公告。因为在从源码构建出 LFS 之后，你便应该养成经常去查看这些报告的好习惯。有关查询的去处，网上倒是有一些不错的资源，这里列举几个：

  - CERT（计算机应急响应小组）

    CERT 有一个邮件列表，专门公示各种操作系统和应用程序的安全警报。订阅信息请点击此链接查看：http://www.us-cert.gov/cas/signup.html.

  - Bugtraq

    Bugtraq 是一个专门公示计算机安全的邮件列表。它公示新发现的安全问题，偶尔还会尽可能的提出修补方案。订阅信息请点击此链接查看：http://www.securityfocus.com/archive.

- Beyond Linux From Scratch

  Beyond Linux From Scratch（BLFS）涵盖了比 LFS 书多得多的应用程序。BLFS 项目主页是： http://www.linuxfromscratch.org/blfs/.

- LFS Hints

  LFS Hints 是由 LFS 社区的志愿者提交的教育文集。有关信息访问以下网址取得： http://www.linuxfromscratch.org/hints/list.html.

- 邮件列表

  有几个 LFS 相关的邮件列表，在你需要的时候可以订阅，也可通过它获得最新的发展动态，对项目作出力所能及的贡献等等。查看 第 1 章 - 邮件列表 可以获得更多的信息。

- The Linux Documentation Project（TLDP，Linux 文档项目）

Linux 文档项目（TLDP）的目标是通过协作来完善 Linux 文档中的所有不足。TLDP 已经完成了大量的 HOWTO、指南和 man 帮助页。它的网站是：http://www.tldp.org/.

Part IV. 附录

Appendix A. 缩写和术语

ABI	Application Binary Interface（应用程序二进制接口）
ALFS	Automated Linux From Scratch （自动化 LFS）
API	Application Programming Interface （应用程序设计接口）
ASCII	American Standard Code for Information Interchange （美国信息交换标准代码）
BIOS	Basic Input/Output System （基本输入/输出系统）
BLFS	Beyond Linux From Scratch
BSD	Berkeley Software Distribution （伯克利软件发行版）
chroot	change root （更改根目录）
CMOS	Complementary Metal Oxide Semiconductor （互补金属氧化物半导体）
COS	Class Of Service （服务等级）
CPU	Central Processing Unit （中央处理单元）
CRC	Cyclic Redundancy Check （循环冗余码校验）
CVS	Concurrent Versions System （并发版本系统）
DHCP	Dynamic Host Configuration Protocol （动态主机配置协议）
DNS	Domain Name Service （域名服务）
EGA	Enhanced Graphics Adapter （增强型图形适配器）
ELF	Executable and Linkable Format （可执行和可链接格式）
EOF	End of File （文件或数据流结束标志）
EQN	equation （相等）
ext2	second extended file system （第二代可扩展文件系统)
ext3	third extended file system （第三代可扩展文件系统)
ext4	fourth extended file system （第四代可扩展文件系统)
FAQ	Frequently Asked Questions （常见问题）
FHS	Filesystem Hierarchy Standard （文件系统层次结构标准）
FIFO	First-In, First Out （先进先出）
FQDN	Fully Qualified Domain Name （完全合格的域名）
FTP	File Transfer Protocol （文件传输协议）
GB	Gigabytes
GCC	GNU Compiler Collection （GNU 编译器集合）
GID	Group Identifier （组标志符）
GMT	Greenwich Mean Time （格林威治标准时间）
HTML	Hypertext Markup Language （超文本标记语言）
IDE	Integrated Drive Electronics （智能磁盘设备，集成电路设备）
IEEE	Institute of Electrical and Electronic Engineers （电气与电子工程师学会）
IO	Input/Output （输入/输出）
IP	Internet Protocol （互联网协议）
IPC	Inter-Process Communication （进程间通信）
IRC	Internet Relay Chat （互联网中继聊天）
ISO	International Organization for Standardization （国际标准化组织）
ISP	Internet Service Provider （因特网服务提供者）
KB	Kilobytes
LED	Light Emitting Diode （发光二极管）
LFS	Linux From Scratch
LSB	Linux Standard Base

MB Megabytes
MBR Master Boot Record （主引导记录）
MD5 Message Digest 5 （信息摘要算法第五版）
NIC Network Interface Card （网络接口卡）
NLS Native Language Support （本地语言支持）
NNTP Network News Transport Protocol （网络新闻传输协议）
NPTL Native POSIX Threading Library （本地 POSIX 线程库）
OSS Open Sound System （开放声音系统）
PCH Pre-Compiled Headers （预编译头文件）
PCRE Perl Compatible Regular Expression （Perl 兼容正则表达式）
PID Process Identifier （进程标志符）
PTY pseudo terminal （伪终端）
QOS Quality Of Service （服务质量）
RAM Random Access Memory （随机存取存储器）
RPC Remote Procedure Call （远程程序调用）
RTC Real Time Clock （实时时钟）
SBU Standard Build Unit （标准编译单位）
SCO The Santa Cruz Operation （圣克鲁斯操作）
SHA1 Secure-Hash Algorithm 1 （安全哈希算法1）
TLDP The Linux Documentation Project （Linux 文档项目）
TFTP Trivial File Transfer Protocol （简单文件传输协议）
TLS Thread-Local Storage （线性本地存储）
UID User Identifier （用户标志符）
umask user file-creation mask （用户文件创建掩码）
USB Universal Serial Bus （通用串行接口）
UTC Coordinated Universal Time （通用协调时间）
UUID Universally Unique Identifier （通用唯一标识符）
VC Virtual Console （虚拟控制台）
VGA Video Graphics Array （视频图形阵列）
VT Virtual Terminal （虚拟终端）

Appendix B. 致谢

我们十分感谢下列为 Linux From Scratch 项目做出了贡献的贡献者和组织。

- Gerard Beekmans <gerard@linuxfromscratch.org> – LFS 创始人
- Bruce Dubbs <bdubbs@linuxfromscratch.org> – LFS 管理编辑
- Jim Gifford <jim@linuxfromscratch.org> – CLFS 项目共同负责人
- Pierre Labastie <pierre@linuxfromscratch.org> – BLFS 编辑和 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 的邮件列表里还有无数同志为本书的发布帮过忙，他们提供建议，帮忙测试，提交问题报告和指令，以及分享他们安装各种软件包的经验。

翻译

简体中文

- Linux 中国 - LCTT - LFS 翻译小组
  - wxy， ictlyh， dongfengweixiao， zpl1025， H-mudcup， Yuking-net， kevinSJ, martin2011qi， vizv， strugglingyouth， Athenacle， haoqixu， zytrenren
- Manuel Canales Esparcia <macana@macana-es.com> – 西班牙语翻译项目
- 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 年至今的 LFS 法语翻译项目
- Anderson Lizardo <lizardo@linuxfromscratch.org> – LFS 葡萄牙语翻译项目
- Thomas Reitelbach <tr@erdfunkstelle.de> – LFS 德语翻译项目
- Anton Maisak <info@linuxfromscratch.org.ru> – LFS 俄罗斯翻译项目
- Elena Shevcova <helen@linuxfromscratch.org.ru> – LFS 俄罗斯翻译项目

镜像维护者

北美镜像

- Scott Kveton <scott@osuosl.org> – lfs.oregonstate.edu 镜像
- William Astle <lost@l-w.net> – ca.linuxfromscratch.org 镜像
- Eujon Sellers <jpolen@rackspace.com> – lfs.introspeed.com 镜像
- Justin Knierim <tim@idge.net> – lfs-matrix.net 镜像

南美镜像

- Manuel Canales Esparcia <manuel@linuxfromscratch.org> – lfsmirror.lfs-es.info 镜像
- Luis Falcon <Luis Falcon> – torredehanoi.org 镜像

欧洲镜像

- Guido Passet <guido@primerelay.net> – nl.linuxfromscratch.org 镜像
- Bastiaan Jacques <baafie@planet.nl> – lfs.pagefault.net 镜像
- Sven Cranshoff <sven.cranshoff@lineo.be> – lfs.lineo.be 镜像
- Scarlet Belgium – lfs.scarlet.be 镜像
- Sebastian Faulborn <info@aliensoft.org> – lfs.aliensoft.org 镜像
- Stuart Fox <stuart@dontuse.ms> – lfs.dontuse.ms 镜像
- Ralf Uhlemann <admin@realhost.de> – lfs.oss-mirror.org 镜像
- Antonin Sprinzl <Antonin.Sprinzl@tuwien.ac.at> – at.linuxfromscratch.org 镜像

- Fredrik Danerklint <fredan-lfs@fredan.org> – se.linuxfromscratch.org 镜像
- Franck <franck@linuxpourtous.com> – lfs.linuxpourtous.com 镜像
- Philippe Baque <baque@cict.fr> – lfs.cict.fr 镜像
- Vitaly Chekasin <gyouja@pilgrims.ru> – lfs.pilgrims.ru 镜像
- Benjamin Heil <kontakt@wankoo.org> – lfs.wankoo.org 镜像
- Anton Maisak <info@linuxfromscratch.org.ru> – linuxfromscratch.org.ru 镜像

亚洲镜像

- Satit Phermsawang <satit@wbac.ac.th> – lfs.phayoune.org 镜像
- Shizunet Co.,Ltd. <info@shizu-net.jp> – lfs.mirror.shizu-net.jp 镜像
- Init World <http://www.initworld.com/> – lfs.initworld.com 镜像

澳大利亚镜像

- Jason Andrade <jason@dstc.edu.au> – au.linuxfromscratch.org 镜像

前项目组成员

- Christine Barczak <theladyskye@linuxfromscratch.org> – LFS 手册编辑
- Archaic <archaic@linuxfromscratch.org> – LFS 技术作家/编辑，HLFS 项目负责人，BLFS 编辑，Hints and Patches 项目维护者
- Matthew Burgess <matthew@linuxfromscratch.org> – LFS 项目负责人，LFS 技术作家/编辑
- Nathan Coulson <nathan@linuxfromscratch.org> – LFS-Bootscripts 版本维护者
- 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 版本维护者
- 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 Backend-Scripts 维护者
- Randy McMurchy <randy@linuxfromscratch.org> – BLFS 项目负责人，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 Gateway 维护者
- Douglas R. Reno <renodr@linuxfromscratch.org> – systemd 编辑
- Ryan Oliver <ryan@linuxfromscratch.org> – CLFS 项目联合负责人
- Greg Schafer <gschafer@zip.com.au> – LFS 技术作家，下一代 64 位模式构建模型架构师
- 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 and Patches 项目负责人
- 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

Appendix C. 依赖关系

LFS 中构建或安装的每一个包或多或少都会依赖于其它的包，甚至有些包甚至是循环依赖关系：第一个包依赖于第二个包，反过来，第二个包的生成或者是安装又需要第 一个包。因为存在这些依赖关系，所以在构建 LFS 系统的时候，软件包的编译或安装的顺序是十分重要的。本章的目的就是告知用户在构建 LFS 时每一个包的依赖项。

对于所有需要编译的软件包，我们列举出三种或者四种类型的依赖。第一种类型是「安装必选依赖」，否则在编译和安装的时候会出现问题。第二类是「测试套件依赖」，假如运行一些测试套件，除了第一类的必须安装外，第二类也需要安装。第三类是「必须之前安装」，此类需要在编译和安装之前就安装。在大多数情况下，是因为它们软件包中的脚本包含有到二进制文件的硬编码。如果不按照顺序编译它们，将会导致 /tools/bin/"binary" 路径变为真实路径写入脚本，然后安装到最终系统中去，这显然是不合适的。（译者注：这句话写的不是太清楚，大概解释一下：在真实系统中，可能会在某些脚本 中出现这样的路径信息：/usr/bin/vi，但是假如因为安装顺序的问题，这个路径就有可能变成 /tools/usr/bin/vi，最终导致脚本不能正常的运行，这显然是错误的。）

最后一类提及的「可供选择依赖」，安装方法在此文档类并没有提及，但是这些程序往往对用户来说又都是极其有用。这些软件包可能又有属于自己的「安装必选依赖」或者是「可用选择依赖」。对于这些依赖关系，推荐的做法是在学习完本书之后回到重建 LFS 包。在大多数情况下，重新安装的问题都记录在 BLFS 中。

Acl

安装必选依赖: Attr, Bash, Binutils, Coreutils, GCC, Gettext, Grep, M4, Make, Perl, Sed, 和 Texinfo
测试套件依赖: Automake, Diffutils, Findutils, 和 Libtool
必须预先安装: Coreutils, Sed, Tar, 和 Vim
可供选择依赖: None

Attr

安装必选依赖: Bash, Binutils, Coreutils, GCC, Gettext, Grep, M4, Make, Perl, Sed, 和 Texinfo
测试套件依赖: Automake, Diffutils, Findutils, 和 Libtool
必须预先安装: Acl 和 Libcap
可供选择依赖: None

Autoconf

安装必选依赖: Bash, Coreutils, Grep, M4, Make, Perl, Sed, 和 Texinfo
测试套件依赖: Automake, Diffutils, Findutils, GCC, 和 Libtool
必须预先安装: Automake
可供选择依赖: Emacs

Automake

安装必选依赖: Autoconf, Bash, Coreutils, Gettext, Grep, M4, Make, Perl, Sed, 和 Texinfo
测试套件依赖: Binutils, Bison, Bzip2, DejaGNU, Diffutils, Expect, Findutils, Flex, GCC, Gettext, Gzip, Libtool, 和 Tar
必须预先安装: None
可供选择依赖: None

Bash

安装必选依赖: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Patch, Readline, Sed, 和 Texinfo
测试套件依赖: Shadow
必须预先安装: None
可供选择依赖: Xorg

Bc

安装必选依赖: Bash, Binutils, Bison, Coreutils, GCC, Glibc, Grep, Make, 和 Readline
测试套件依赖: Gawk
必须预先安装: Linux Kernel
可供选择依赖: None

Binutils

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, File, Gawk, GCC, Glibc, Grep, Make, Perl, Sed, Texinfo 和 Zlib
测试套件依赖: DejaGNU 和 Expect
必须预先安装: None
可供选择依赖: None

Bison

安装必选依赖: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Perl, 和 Sed
测试套件依赖: Diffutils, Findutils, 和 Flex
必须预先安装: Kbd 和 Tar
可供选择依赖: Doxygen (test suite)

Bzip2

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make, 和 Patch
测试套件依赖: None
必须预先安装: None
可供选择依赖: None

Check

安装必选依赖: GCC, Grep, Make, Sed, 和 Texinfo
测试套件依赖: None
必须预先安装: None
可供选择依赖: None

Coreutils

安装必选依赖: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, GMP, Grep, Make, Patch, Perl, Sed, 和 Texinfo
测试套件依赖: Diffutils, E2fsprogs, Findutils, Shadow, 和 Util-linux
必须预先安装: Bash, Diffutils, Eudev, Findutils, 和 Man-DB
可供选择依赖: Perl Expect 和 IO:Tty modules (for test suite)

DejaGNU

安装必选依赖: Bash, Coreutils, Diffutils, GCC, Grep, Make, 和 Sed
测试套件依赖: None
必须预先安装: None
可供选择依赖: None

Diffutils

安装必选依赖: Bash, Binutils, Coreutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed, 和 Texinfo
测试套件依赖: Perl
必须预先安装: None
可供选择依赖: None

E2fsprogs

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Gzip, Make, Sed, Texinfo, 和 Util-linux
测试套件依赖: Procps-ng 和 Psmisc
必须预先安装: None
可供选择依赖: None

Eudev

安装必选依赖: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Gperf, Make, 和 Sed
测试套件依赖: None
必须预先安装: None
可供选择依赖: None

Expat

安装必选依赖: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, 和 Sed
测试套件依赖: None
必须预先安装: XML::Parser
可供选择依赖: None

Expect

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Patch, Sed, 和 Tcl
测试套件依赖: None
必须预先安装: None
可供选择依赖: None

File

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, 和 Zlib
测试套件依赖: None
必须预先安装: None
可供选择依赖: None

Findutils

安装必选依赖: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, 和 Texinfo
测试套件依赖: DejaGNU, Diffutils, 和 Expect
必须预先安装: None
可供选择依赖: None

Flex

安装必选依赖: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Patch, Sed, 和 Texinfo
测试套件依赖: Bison 和 Gawk
必须预先安装: IPRoute2, Kbd, 和 Man-DB
可供选择依赖: None

Gawk

安装必选依赖: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, GMP, Grep, Make, MPFR, Patch, Readline, Sed, 和 Texinfo
测试套件依赖: Diffutils
必须预先安装: None
可供选择依赖: None

Gcc

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, GMP, Grep, M4, Make, MPC, MPFR, Patch, Perl, Sed, Tar, 和 Texinfo
测试套件依赖: DejaGNU, Expect, 和 Shadow
必须预先安装: None
可供选择依赖: GNAT and ISL

GDBM

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, GCC, Grep, Make, 和 Sed
测试套件依赖: None
必须预先安装: None
可供选择依赖: None

Gettext

安装必选依赖: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed, 和 Texinfo
测试套件依赖: Diffutils, Perl, 和 Tcl
必须预先安装: Automake
可供选择依赖: None

Glibc

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Gzip, Linux API Headers, Make, Perl, Sed, 和 Texinfo
测试套件依赖: File
必须预先安装: None
可供选择依赖: None

GMP

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, M4, Make, Sed, 和 Texinfo
测试套件依赖: None
必须预先安装: MPFR 和 GCC
可供选择依赖: None

Gperf

安装必选依赖: Bash, Binutils, Coreutils, GCC, Glibc, 和 Make
测试套件依赖: Diffutils 和 Expect
必须预先安装: None
可供选择依赖: None

Grep

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Patch, Sed, 和 Texinfo
测试套件依赖: Gawk
必须预先安装: Man-DB
可供选择依赖: Pcre

Groff

安装必选依赖: Bash, Binutils, Bison, Coreutils, Gawk, GCC, Glibc, Grep, Make, Patch, Sed, 和 Texinfo
测试套件依赖: No test suite available
必须预先安装: Man-DB 和 Perl
可供选择依赖: GPL Ghostscript

GRUB

安装必选依赖: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Texinfo, 和 Xz
测试套件依赖: None
必须预先安装: None
可供选择依赖: None

Gzip

安装必选依赖: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, 和 Texinfo
测试套件依赖: Diffutils 和 Less
必须预先安装: Man-DB
可供选择依赖: None

Iana-Etc

安装必选依赖: Coreutils, Gawk, 和 Make
测试套件依赖: No test suite available
必须预先安装: Perl
可供选择依赖: None

Inetutils

安装必选依赖: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed, Texinfo, 和 Zlib
测试套件依赖: No test suite available
必须预先安装: Tar
可供选择依赖: None

Intltool

安装必选依赖: Bash, Gawk, Glibc, Make, Perl, Sed, 和 XML::Parser
测试套件依赖: Perl
必须预先安装: None
可供选择依赖: None

IProute2

安装必选依赖: Bash, Bison, Coreutils, Flex, GCC, Glibc, Make, 和 Linux API Headers
测试套件依赖: No test suite available
必须预先安装: None
可供选择依赖: None

Kbd

安装必选依赖: Bash, Binutils, Bison, Check, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, Patch, 和 Sed
测试套件依赖: No test suite available
必须预先安装: None
可供选择依赖: None

Kmod

安装必选依赖: Bash, Binutils, Bison, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, Sed, Xz-Utils, 和 Zlib
测试套件依赖: No test suite available
必须预先安装: Eudev
可供选择依赖: None

Less

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, 和 Sed
测试套件依赖: No test suite available
必须预先安装: Gzip
可供选择依赖: Pcre

Libcap

安装必选依赖: Attr, Bash, Binutils, Coreutils, GCC, Glibc, Perl, Make, 和 Sed
测试套件依赖: No test suite available
必须预先安装: None
可供选择依赖: Linux-PAM

Libelf

安装必选依赖: Bash, Binutils, Coreutils, GCC, Glibc, 和 Make
测试套件依赖: No test suite available
必须预先安装: Linux Kernel
可供选择依赖: None

Libffi

安装必选依赖: Bash, Binutils, Coreutils, GCC, Glibc, Make, 和 Sed
测试套件依赖: DejaGnu
必须预先安装: Python
可供选择依赖: None

Libpipeline

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, 和 Texinfo
测试套件依赖: Check
必须预先安装: Man-DB
可供选择依赖: None

Libtool

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, 和 Texinfo
测试套件依赖: Autoconf, Automake, 和 Findutils
必须预先安装: None
可供选择依赖: None

Linux Kernel

安装必选依赖: Bash, Bc, Binutils, Coreutils, Diffutils, Findutils, GCC, Glibc, Grep, Gzip, Kmod, Libelf, Make, Ncurses, OpenSSL, Perl, 和 Sed
测试套件依赖: No test suite available
必须预先安装: None
可供选择依赖: None

M4

安装必选依赖: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, and Texinfo
测试套件依赖: Diffutils
必须预先安装: Autoconf 和 Bison
可供选择依赖: libsigsegv

Make

安装必选依赖: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, 和 Texinfo
测试套件依赖: Perl 和 Procps-ng
必须预先安装: None
可供选择依赖: None

Man-DB

安装必选依赖: Bash, Binutils, Bzip2, Coreutils, Flex, GCC, GDBM, Gettext, Glibc, Grep, Groff, Gzip, Less, Libpipeline, Make, Sed, 和 Xz
测试套件依赖: Util-linux
必须预先安装: None
可供选择依赖: None

Man-Pages

安装必选依赖: Bash, Coreutils, 和 Make
测试套件依赖: No test suite available
必须预先安装: None
可供选择依赖: None

Meson

安装必选依赖: Ninja 和 Python
测试套件依赖: No test suite available
必须预先安装: None
可供选择依赖: None

MPC

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, GMP, Make, MPFR, Sed, 和 Texinfo
测试套件依赖: None
必须预先安装: GCC
可供选择依赖: None

MPFR

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, GMP, Make, Sed, 和 Texinfo
测试套件依赖: None
必须预先安装: Gawk 和 GCC
可供选择依赖: None

Ncurses

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Patch, 和 Sed
测试套件依赖: No test suite available
必须预先安装: Bash, GRUB, Inetutils, Less, Procps-ng, Psmisc, Readline, Texinfo, Util-linux, 和 Vim
可供选择依赖: None

Ninja

安装必选依赖: Binutils, Coreutils, Gcc, 和 Python
测试套件依赖: None
必须预先安装: Meson
可供选择依赖: Asciidoc, Doxygen, Emacs, 和 re2c

Openssl

安装必选依赖: Binutils, Coreutils, Gcc, Make, 和 Perl
测试套件依赖: None
必须预先安装: Linux
可供选择依赖: None

Patch

安装必选依赖: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, 和 Sed
测试套件依赖: Diffutils
必须预先安装: None
可供选择依赖: Ed

Perl

安装必选依赖: Bash, Binutils, Coreutils, Gawk, GCC, GDBM, Glibc, Grep, Groff, Make, Sed, 和 Zlib
测试套件依赖: Iana-Etc 和 Procps-ng
必须预先安装: Autoconf
可供选择依赖: None

Pkg-config

安装必选依赖: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Popt, 和 Sed
测试套件依赖: None
必须预先安装: Kmod
可供选择依赖: None

Popt

安装必选依赖: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, 和 Make
测试套件依赖: Diffutils 和 Sed
必须预先安装: Pkg-config
可供选择依赖: None

Procps-ng

安装必选依赖: Bash, Binutils, Coreutils, GCC, Glibc, Make, 和 Ncurses
测试套件依赖: DejaGNU
必须预先安装: None
可供选择依赖: None

Psmisc

安装必选依赖: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, 和 Sed
测试套件依赖: No test suite available
必须预先安装: None
可供选择依赖: None

Python

安装必选依赖: Bash, Binutils, Coreutils, GCC, Gdbm, Gettext, Glibc, Grep, Libffi, Make, Ncurses, 和 Sed
测试套件依赖: GDB 和 Valgrind
必须预先安装: Ninja
可供选择依赖: Berkeley DB, OpenSSL, SQLite, 和 Tk

Readline

安装必选依赖: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed, 和 Texinfo
测试套件依赖: No test suite available
必须预先安装: Bash 和 Gawk
可供选择依赖: None

Sed

安装必选依赖: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, 和 Texinfo
测试套件依赖: Diffutils 和 Gawk
必须预先安装: E2fsprogs, File, Libtool, 和 Shadow
可供选择依赖: None

Shadow

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Make, 和 Sed
测试套件依赖: No test suite available
必须预先安装: Coreutils
可供选择依赖: Acl, Attr, Cracklib, 和 PAM

Sysklogd

安装必选依赖: Binutils, Coreutils, GCC, Glibc, Make, 和 Patch
测试套件依赖: No test suite available
必须预先安装: None
可供选择依赖: None

systemd

安装必选依赖: Acl, Attr, Bash, Binutils, Coreutils, Diffutils, Expat, Gawk, GCC, Glibc, Gperf, Grep, Intltool, Libcap, Make, Sed, 和 Util-linux
测试套件依赖: None
必须预先安装: None
可供选择依赖: None

Sysvinit

安装必选依赖: Binutils, Coreutils, GCC, Glibc, Make, 和 Sed
测试套件依赖: No test suite available
必须预先安装: None
可供选择依赖: None

Tar

安装必选依赖: Acl, Attr, Bash, Binutils, Bison, Coreutils, GCC, Gettext, Glibc, Grep, Inetutils, Make, Sed, 和 Texinfo
测试套件依赖: Autoconf, Diffutils, Findutils, Gawk, 和 Gzip
必须预先安装: None
可供选择依赖: None

Tcl

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, 和 Sed
测试套件依赖: None
必须预先安装: None
可供选择依赖: None

Texinfo

安装必选依赖: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Patch, 和 Sed
测试套件依赖: None
必须预先安装: None
可供选择依赖: None

Util-linux

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, Eudev, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, 和 Zlib
测试套件依赖: None
必须预先安装: None
可供选择依赖: Libcap-ng

Vim

安装必选依赖: Acl, Attr, Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, 和 Sed
测试套件依赖: None
必须预先安装: None
可供选择依赖: Xorg, GTK+2, LessTif, Python, Tcl, Ruby, 和 GPM

XML::Parser

安装必选依赖: Bash, Binutils, Coreutils, Expat, GCC, Glibc, Make, 和 Perl
测试套件依赖: Perl
必须预先安装: Intltool
可供选择依赖: None

Xz

安装必选依赖: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, 和 Make
测试套件依赖: None
必须预先安装: Eudev, GRUB, Kmod, 和 Man-DB
可供选择依赖: None

Zlib

安装必选依赖: Bash, Binutils, Coreutils, GCC, Glibc, Make, 和 Sed
测试套件依赖: None
必须预先安装: File, Kmod, Perl, 和 Util-linux
可供选择依赖: None

Appendix D. LFS Licenses

This book is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 2.0 License.

Computer instructions may be extracted from the book under the MIT License.

D.1. Creative Commons License

Creative Commons Legal Code

Attribution-NonCommercial-ShareAlike 2.0

> **Important**
>
> 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.

> **Important**
>
> 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

Index

Packages

Programs

Libraries

Scripts

Others