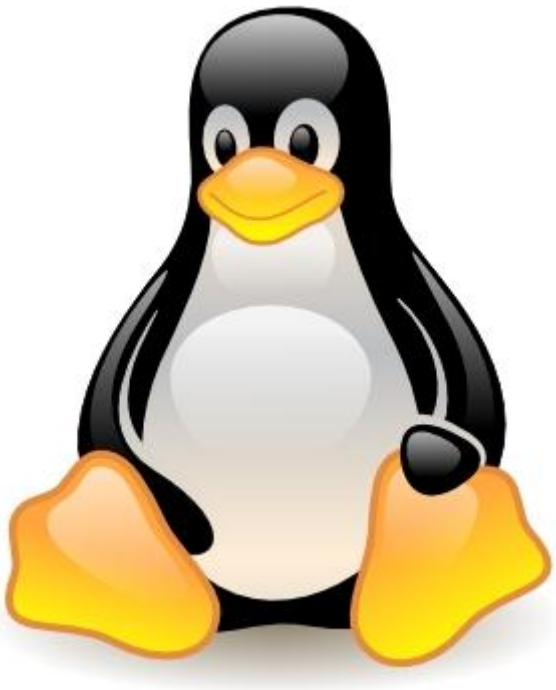
从头开始搭建 Linux 系统

Linux From Scratch

版本 7.0



Gerard Beekmans 著

Matthew Burgess ， Bruce Dubbs 编辑

屈剑平 译

qjpcpu@gmail.com

目录

第一部分： 绪论

第一章 简介

1.1 怎样搭建一个 LFS 系统

要搭建一个 LFS 系统必须使用一个已经安装好的 Linux 发行版（比如 Debian, Mandriva, RedHat 或者 SUSE）。这个已存在的 Linux 系统（宿主系统）充当一个搭建新系统的起点，提供必要的程序，包括编译器，链接器和 shell。在发行版安装时选择“开发（development）”选项可以安装这些工具。

作为安装一个分离的发行版到你的计算机上的替代选择，你可能希望使用一张来自商业发行版的 LiveCD。

本书的第二章讲述了如何创建一个新的本地 Linux 分区和文件系统。我们的新 LFS 系统将在这上面编译安装。

第三章解释了哪些包和补丁需要下载和怎样将这些包存储到新的文件系统上。第四章讨论了怎样合理地设置工作环境。

请务必仔细阅读第四章，按照解释的几个重要问题，在开始第五章的工作之前充分了解工作方法。

第五章解释了形成基本开发套件（或者说工具链）的大量软件包的安装，这些软件包用来在第六章搭建实际的系统。其中的一些包需要解决循环依赖——例如，为了编译一个编译器，你必须要有一个编译器。第五章也展示了怎样制作第一条工具链，包括 Binutils 和 Gcc（第一条工具链意味着这两个核心的软件包需要被重新安装）。下一步是编译 Glibc，即 C 库。Glibc 是使用第一条工具链的程序来编译得到的。然后，再一次制作工具链。这次的工具链会被动态链接到新的 Glibc。第五章剩下部分的软件包就是用这个第二遍工具链来编译。当完成后，LFS 的制作将不再依赖宿主系统。

为了将新系统和宿主系统分离而花费如此多的精力看似有点过分。但在 5.2 节“工具链技术说明”中将会有一个全面的技术解释。

在第六章会搭建完整的 LFS 系统。**chroot**（改变根目录）程序用来进入一个虚拟的环境并启动一个新 shell，其根目录被设置到 LFS 分区。这非常类似于重启系统并让内核挂载 LFS 分区作为根分区。系统实际上并没有重启，只是 **chroot** 了，因为创建一个可启动的系统要求许多不必要的额外工作。在等待软件包完成编译时，你可以继续正常地使用你的计算机。

为了完成安装，将在第七章设置 LFS-Bootscripts，内核和 boot loader 将在第八章安装。第九章包含了完成本书后继续 LFS 之旅的一些信息。在这些步骤完成后，可以重启计算机进入新的 LFS 系统。

概括地说这就是搭建 LFS 的流程。详细信息将在接下来的章节和软件包的描述信息中阐述。复杂的事都会被理清，但在 LFS 的旅途中什么事都可能发生。

1.2 比上个版本的更新

相较于本书上个版本，下面的列表给出了包的更新。

Note

本书这个版本的某个主要变化是增加了一个顶级目录，/run。tmpfs 文件系统挂载在该目录下，udev 等程序会在该目录下存储运行时信息。目录/var/run 和/var/lock 也链接到该目录。启动脚本为了适应该变化也做了相应的更新。

更新列表：

- Binutils 2.21.1a
- Bison 2.5
- Coreutils 8.14
- DejaGNU 1.5
- Diffutils 3.2
- File 5.09
- Gawk 4.0.0
- GCC 4.6.1
- GDBM 1.9.1
- Glibc 2.14.1
- GMP 5.0.2
- Grep 2.9
- GRUB 1.99
- IPRoute2 2.6.39
- Less 444
- LFS-Bootscripts 20111017
- Linux 3.1
- M4 1.4.16
- Man-DB 2.6.0.2
- Module-Init-Tools 3.16
- MPC 0.9
- MPFR 3.1.0
- Ncurses 5.9
- Perl 5.14.2
- Psmisc 22.14
- Tar 1.26
- TCL 8.5.10
- Udev 173
- Util-Linux 2.20
- XZ-Utils 5.0.3

新增：

- bash-4.2-fixes-3.patch
- coreutils-8.14-i18n-1.patch

- gcc-4.6.1-cross_compile-1.patch
- gcc-4.6.1-startfiles_fix-1.patch
- gcc-4.6.1-locale-1.patch
- glibc-2.14.1-fixes-1.patch
- glibc-2.14.1-gcc_fix-1.patch
- glibc-2.14.1-cpuid-1.patch
- libpipeline-1.2.0
- module-init-tools-3.16-man_pages-1.patch
- perl-5.14.1-libc-1.patch
- readline-6.2-fixes-1.patch

移除：

- coreutils-8.10-i18n-1.patch
- dejagnu-1.4.4-consolidated-1.patch
- gcc-4.6.0-cross_compile-1.patch
- gcc-4.6.0-startfiles_fix-1.patch
- glibc-2.13-gcc_fix-1.patch
- perl-5.12.3-libc-1.patch
- Pkg-Config-0.25

1.3 更改日志

略。

1.4 资源

1.4.1 FAQ

如果在搭建LFS系统中遇到任何错误，有任何问题，或者认为书中有错字，请咨询Frequently Asked Questions(FAQ)位于*http://www.linuxfromscratch.org/faq/*。

1.4.2 邮件列表

在linuxfromscratch.org服务器上有许多邮件列表用于LFS项目的开发这些列表包含了主要的开发和支持列表。如果FAQ不能解决你的问题，下一步就应该是在 *http://www.linuxfromscratch.org/search.html* 搜索邮件列表。对于列表的附加信息，请访问*http://www.linuxfromscratch.org/mail.html*。

1.4.3 IRC

LFS 社区的许多成员都在我们的社区 Internet Relay Chat(IRC)网络上提供帮助。在利用这些支持前，请确定你的问题在 LFS FAQ 或邮件列表里还没有被回答。你可以在 irc.linuxfromscratch 找到 IRC 网络。该支持通道被命名为#LFS-support。

1.4.4 镜像站点

为了使访问 LFS 网站和下载必要的软件包更加方便，LFS 在全世界有许多镜像站点。请访问 LFS 网站 *http://www.linuxfromscratch.org/mirrors.html* 获取当前镜像站点列表。

1.4.5 联系信息

请直接将你所有的问题和评论提交到上述 LFS 邮件列表。

1.5 帮助

当你在阅读本书时遇到问题时，请查阅 FAQ*http://www.linuxfromscratch.org/faq/#generalfaq*。常见的问题很可能在那里已经被回答了。如果你没有在网页上找到答案，请试着找到问题的源头。下面的提示可能对解决你的问题有些知道作用：*http://www.linuxfromscratch.org/hints/downloads/files/errors.txt*。

如果你在 FAQ 的列表中找不到你的问题，请搜索邮件列表 *http://www.linuxfromscratch.org/search.html* 。

我们也有一个优秀的 LFS 社区愿意通过邮件列表和 IRC（见 1.4 节“资源”部分）来提供帮助。但是，每天我们接到许多支持问题，其中有许多都是可以先去查询 FAQ 和搜索邮件列表就可以简单地找到答案的。所以，为了我们能更好地提供帮助，你需要自己先自己做些研究。这样便于我们集中精力在更不平常的支持上。如果你的搜索并不能得到一个解决方案，请在你请求帮助中囊括所有相关信息（上面提到的）。

1.5.1 提醒

略

1.5.2 配置脚本问题

当运行 configure 脚本出错时，查看 config.log 文件。该文件可能包含配置时发生的错误，这些错是不会打印到屏幕的。如果你请求帮助请包含相关行。

1.5.3 编译问题

在判断编译问题时，屏幕输出和不同文件的内容都非常重要。从 configure 脚本在屏幕的输出和 make 运行后屏幕输出都很有用。不必要包含全部的输出，但要包含足够的相关信息。下面是一个 make 屏幕输出的信息的例子：

```
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
```

在这种情况下，许多人仅仅会包含最下面一部分信息：

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

这对于正常地诊断问题来说信息量太小了，因为这仅仅只是表示某些地方出错了，而不知道到底哪儿出错了。如上面例子中全部都应该保存，因为它包含了运行相关的命令和相关的错误信息。

在 *http://catb.org/~esr/faqs/smartquestions.html* 上可以找到一篇优秀的怎样请求帮助的文章。请阅读并按照该文章的提示做，以提高得到帮助的可能性。

第二部分： 准备搭建系统

第二章 准备一个新分区

2.1 简介

在本章中，将要准备 LFS 系统所在分区。我们要创建该分区，并在上面创建文件系统然后挂载该分区。

2.2 创建新分区

和大多数其他操作系统一样，LFS 常常也被安装在一个专门的分区上。值得推荐的办法是在一个可用的空分区上搭建 LFS 系统，或者如果你有足够的未分配的磁盘分区你也可以创建一个新分区。

一个最小系统至少需要分区大小为 2.8GB。这才足够存放所有的源文件压缩包和编译这些包。然而，如果打算把 LFS 系统做成主要的 Linux 系统，那么就需要额外的空间来安装额外的软件。10GB 是一个合适的大小来满足系统的增长。LFS 系统本身并不会占据如此大的空间。其中很大一部分是为了提供足够的暂存空间。编译软件包需要很大磁盘空间且在编译完成后会释放。

由于一般没有足够的随机访问存储（RAM）供编译进程使用，所以最好是使用一块小的磁盘分区充当交换空间（swap）。交换空间是供内核存储不常用的数据，以便于腾出内存供激活的进程使用。LFS 系统的 swap 分区可以使用宿主系统的 swap，不必要创建一个新的。

运行磁盘分区程序如 cfdisk 或 fdisk，用命令行选项命名新创建的分区，例如用/dev/had 命名电子集成驱动(IDE)磁盘。创建一个 Linux 本地分区，如果需要可创建一个 swap 分区。如果你不知道怎么使用这两个程序，请查阅 cfdisk(8)和 fdisk(8)用户手册。

请记住新分区的命名（如 hda5）。本书将引用它作为 LFS 分区。同样请记住 swap 分区的命名。这些名称以后将在/etc/fstab 文件中用到。

2.2.1 其他分区问题

咨询分区的请求经常在 LFS 的邮件列表中提及。这是一个非常个人的话题。大部分发行版都会使用除了交换分区外的所有此片空间。由于某些原因，对 LFS 来说这并不是最佳的方式。它降低了灵活性，使不同发行版或者 LFS 之间分享数据变得更困难，使备份数据更耗时，而且由于低效的分配文件系统结构浪费了大量磁盘空间。

2.2.1.1 根分区

LFS 系统的根分区（不要和/root 目录混淆）定为 10GB 对大多数系统都是一个不错的选择。它提供了足够的空间搭建 LFS 系统和大多数 BLFS 系统，但是却是非常小巧，有利于实验创建多个分区。

2.2.1.2 swap 分区

大多数 Linux 发行版会自动创建一个 swap 分区。一般 swap 分区的推荐大小是物理 RAM 的两倍，然而它其实是极少被使用的。如果磁盘空间有限，保持 swap 分区为 2GB 然后监视磁盘交换的数量。

交换并不是很好的。一般来说可以听磁盘活动和监视系统对命令的反应来判别系统是否

在交换数据。交换数据的第一反应是检查例如编辑一个 5GB 的文件的不合理的命令。如果交换变得经常发生，最好的解决办法就是为你的计算机购买更多的 RAM。

2.2.1.3 便利地分区

下面是一些没有被提及的分区，但是在磁盘布局时应该考虑它们。下面的列表不容易理解，但是这只是为了做一个指导。

- /boot——极力推荐。使用该分区来存储内核和其他启动信息。在使用大磁盘的情况下为了最小化潜在的启动问题，将该分区作为磁盘的第一个物理分区。只需要 100MB 大小就非常足够了。
- /home——极力推荐。在不同发行版或者 LFS 系统间共享你的家目录和用户自定义信息。一般该分区都相当大，但以你可用的磁盘空间为准。
- /usr——如果为一个瘦用户端建造一个服务器或者无盘工作站，常常会使用分离的/usr 分区。但对 LFS 一般是不必要的。5GB 的大小就足够满足大多数安装要求了。
- /opt——对 BLFS 来说该目录很有用，BLFS 需要安装诸如 Gnome 或 KDE 等大型软件而不用放置嵌套文件到/usr 目录下。如果使用该分区，那么 5 到 10GB 大小就足够了。
- /tmp——分离的/tmp 目录是很少见的，不过在配置一个瘦客户端时是很有用的。如果使用该分区，不超过 2GB 即可。
- /usr/src——对于存储 BLFS 的源文件和在不同 LFS 间共享，该目录是非常有用的。它也可以充当搭建 BLFS 包的一个位置。分配 30-50GB 是合理的大小。

如果你希望在启动时自动挂载任何分离的分区，需要在/etc/fstab 文件中配置好。详细配置分区将在 8.2 节“创建/etc/fstab“讨论。

2.3 为分区创建文件系统

我们已经创建好了一个空白分区，现在可以在上面创建文件系统了。在Linux世界里最广泛使用的文件系统是ext2文件系统，但对于更新的大容量硬盘，日志文件系统变得越来越流行。作为ext2的性能提高版ext3文件系统被广泛使用，它增加了记录日志能力而且和E2fsprogs工具兼容。我们将创建ext3文件系统。关于创建其他文件系统的指导可以查阅*http://www.linuxfromscratch.org/blfs/view/svn/postlfs/filesystems.html*。

运行下面的命令，可以在 LFS 分区上创建一个 ext3 文件系统：

```
mke2fs -jv /dev/<xxx>
```

请用 LFS 分区的名称（在我们前面的例子中是 hda5）代替<xxx>。

Note

一些 Linux 发行版在它们的文件系统创建工具（E2fsprogs）中使用了自定义的特性。因为这些特性不被 LFS 安装的 E2fsprogs 所支持，所以在第九章中启动 LFS 系统时可能引起问题；你可能会遇到诸如“不支持的文件系统特性，请升级你的 e2fsprogs“之类的错误。如果你的宿主系统使用了自定义性能提高，请用下面的命令检查：

```
debugfs -R feature /dev/<xxx>
```

如果你的输出不只包括：has_journal, ext_attr, resize_inode, dir_index,filetype, sparse_super, large_file 或 needs_recovery，那么你的宿主系统很可能使用了自定义提高。在这种情况下，为了避免将来的错误，你应该编译 E2fsprogs 包，然后用创建的二进制文件在 LFS 分区上重新创建文件系统：

```
cd /tmp
tar -xzvf /path/to/sources/e2fsprogs-1.41.14.tar.gz
cd e2fsprogs-1.41.14
mkdir -v build
cd build
../configure

make #注意，这里我们故意没有使用"make install" !

./misc/mke2fs -jv /dev/<xxx>
cd /tmp
rm -rfv e2fsprogs-1.41.14
```

如果你在使用一个已经存在的 swap 分区，就没有必要去格式化了。如果创建了一个新 swap 分区，可以用下面的命令初始化：

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

请用 swap 分区名代替<yyy>。

2.4 挂载新分区

现在我们已经创建了文件系统，那么就需要使分区可访问。所以，LFS 分区需要在选定的挂载点挂载。本书中是假设该文件系统是挂载在/mnt/lfs 下，但你也可以挂载在别的目录下。

运行以下命令，选定挂载点并添加到 LFS 环境变量：

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

运行下面的命令，创建挂载点并挂载 LFS 文件系统：

```
mkdir -pv $LFS
mount -v -t ext3 /dev/<xxx> $LFS
```

请用 LFS 分区名代替<xxx>。

若为 LFS 使用了多个分区（如，一个“/”分区一个“/usr”分区），使用下面的命令分别挂载他们：

```
mkdir -pv $LFS
mount -v -t ext3 /dev/<xxx> $LFS
mkdir -v $LFS/usr
mount -v -t ext3 /dev/<yyy> $LFS/usr
```

请合适的分区名代替<xxx>和<yyy>。

请确保新分区没有使用过于严格的权限（如, nosuid, nodev 或 notime 选项）来挂载。运行无参的 **mount** 命令查看挂载 LFS 分区时使用了哪些选项。如果设置了 nosuid, nodev 和/或 noatime，那么就要重新挂载分区。

如果使用了 swap 分区，确保使用了 swapon 命令来启用:

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

使用 swap 分区名代替<zzz>。

既然已经建立好了工作基地，就可以开始下载软件包了。

第三章 软件包和补丁

3.1 简介

本章列举了构建一个基本的 Linux 系统需要下载的软件包列表。列表中列举的软件响应版本是已知可以运行的，本书也是基于这些软件的。我们不推荐使用新版本的软件，因为搭建命令对某版本有效但对更新的版本可能无法正常工作。最新版本的软件可能存在亟待解决的问题。这些问题的解决办法会在本书的新版本中解决。

下载地址不总是可用。在本书出版后若下载地址有所改变，那么利用Google（*http://www.google.com/*）有力的搜索引擎可以找到大部分的软件包。如果这样仍然没有成功，请尝试另一个替代方法，在 *http://www.linuxfromscratch.org/lfs/packages.html#packages* 取得下载。

下载的软件包需要存储在一个合适的地方，以便于在整个搭建过程中易于访问。我们需要一个工作目录里解压缩软件包并使用它们。$LFS/sources 目录既可以作为压缩包和补丁的存储目录也可以作为工作目录。使用该目录后，所以需要的元素可以在 LFS 分区定位到，也保证了整个搭建过程中随时可访问。

在开始下载之前，请以 root 用户身份运行以下命令，来创建该目录：

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

使该目录可写并设置粘连属性。“粘连”属性意为尽管多个用户都对目录有写权限，但对于一个设置了粘连属性的目录，只有文件的拥有者才能够删除该文件。下面的命令可以启用写和粘连权限：

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

将 wget-list 作为 wget 的输入是下载所有软件包和补丁的简便方式。如示例：

```
wget -i wget-list -P $LFS/sources
```

此外，为了开始 LFS-7.0，这里有一个分离的文件，md5sums，用于验证所有的软件包是否正确可用。放置该文件到$LFS/sources 下并运行：

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

3.2 所有软件包

请下载或用别的方式获得下列软件包：

• Autoconf (2.68) - 1,350 KB:
Home page: *http://www.gnu.org/software/autoconf/*
Download: *http://ftp.gnu.org/gnu/autoconf/autoconf-2.68.tar.bz2*
MD5 sum: `864d785215aa60d627c91fcb21b05b07`

• Automake (1.11.1) - 1,042 KB:
Home page: *http://www.gnu.org/software/automake/*
Download: *http://ftp.gnu.org/gnu/automake/automake-1.11.1.tar.bz2*
MD5 sum: `c2972c4d9b3e29c03d5f2af86249876f`

• Bash (4.2) - 6,845 KB:
Home page: *http://www.gnu.org/software/bash/*
Download: *http://ftp.gnu.org/gnu/bash/bash-4.2.tar.gz*
MD5 sum: `3fb927c7c33022f1c327f14a81c0d4b0`
• Binutils (2.21.1a) - 18,553 KB:
Home page: *http://www.gnu.org/software/binutils/*
Download: *http://ftp.gnu.org/gnu/binutils/binutils-2.21.1a.tar.bz2*
MD5 sum: `bde820eac53fa3a8d8696667418557ad`
• Bison (2.5) - 1,983 KB:
Home page: *http://www.gnu.org/software/bison/*
Download: *http://ftp.gnu.org/gnu/bison/bison-2.5.tar.bz2*
MD5 sum: `9dba20116b13fc61a0846b0058fbe004`
• Bzip2 (1.0.6) - 764 KB:
Home page: *http://www.bzip.org/*
Download: *http://www.bzip.org/1.0.6/bzip2-1.0.6.tar.gz*
MD5 sum: `00b516f4704d4a7cb50a1d97e6e8e15b`
• Check (0.9.8) - 546 KB:
Home page: *http://check.sourceforge.net/*
Download: *http://sourceforge.net/projects/check/files/check/0.9.8/check-0.9.8.tar.gz*
MD5 sum: `5d75e9a6027cde79d2c339ef261e7470`
• Coreutils (8.14) - 4,842 KB:
Home page: *http://www.gnu.org/software/coreutils/*
Download: *http://ftp.gnu.org/gnu/coreutils/coreutils-8.14.tar.xz*
MD5 sum: `bcb135ce553493a45aba01b39eb3920a`
• DejaGNU (1.5) - 563 KB:
Home page: *http://www.gnu.org/software/dejagnu/*
Download: *http://ftp.gnu.org/gnu/dejagnu/dejagnu-1.5.tar.gz*
MD5 sum: `3df1cbca885e751e22d3ebd1ac64dc3c`
• Diffutils (3.2) - 1,976 KB:
Home page: *http://www.gnu.org/software/diffutils/*
Download: *http://ftp.gnu.org/gnu/diffutils/diffutils-3.2.tar.gz*
MD5 sum: `22e4deef5d8949a727b159d6bc65c1cc`
• E2fsprogs (1.41.14) - 4,406 KB:
Home page: *http://e2fsprogs.sourceforge.net/*
Download: *http://prdownloads.sourceforge.net/e2fsprogs/e2fsprogs-1.41.14.tar.gz*
MD5 sum: `05f70470aea2ef7efbb0845b2b116720`
• Expect (5.45) - 614 KB:
Home page: *http://expect.sourceforge.net/*
Download: *http://prdownloads.sourceforge.net/expect/expect5.45.tar.gz*
MD5 sum: `44e1a4f4c877e9ddc5a542dfa7ecc92b`
• File (5.09) - 593 KB:
Home page: *http://www.darwinsys.com/file/*
Download: *ftp://ftp.astron.com/pub/file/file-5.09.tar.gz*

MD5 sum: 6fd7cd6c4281e68fe9ec6644ce0fac6f

Note

文件（5.09）可能不在列表中的位置了。当新版本发布时该站点的管理员偶然移除了旧版的下载位置。下载正确版本的可用的位置是：*http://www.linuxfromscratch.org/lfs/download.html#ftp*

• Findutils (4.4.2) - 2,100 KB:
Home page: *http://www.gnu.org/software/findutils/*
Download: *http://ftp.gnu.org/gnu/findutils/findutils-4.4.2.tar.gz*
MD5 sum: 351cc4adb07d54877fa15f75fb77d39f
• Flex (2.5.35) - 1,227 KB:
Home page: *http://flex.sourceforge.net*
Download: *http://prdownloads.sourceforge.net/flex/flex-2.5.35.tar.bz2*
MD5 sum: 10714e50cea54dc7a227e3eddcd44d57
• Gawk (4.0.0) - 2,016 KB:
Home page: *http://www.gnu.org/software/gawk/*
Download: *http://ftp.gnu.org/gnu/gawk/gawk-4.0.0.tar.bz2*
MD5 sum: 7cdc48e99b885a4bbe0e98dcf1706b22
• GCC (4.6.1) - 70,009 KB:
Home page: *http://gcc.gnu.org/*
Download: *http://ftp.gnu.org/gnu/gcc/gcc-4.6.1/gcc-4.6.1.tar.bz2*
MD5 sum: c57a9170c677bf795bdc04ed796ca491
• GDBM (1.9.1) - 542 KB:
Home page: *http://www.gnu.org/software/gdbm/*
Download: *http://ftp.gnu.org/gnu/gdbm/gdbm-1.9.1.tar.gz*
MD5 sum: 59f6e4c4193cb875964ffbe8aa384b58
• Gettext (0.18.1.1) - 14,785 KB:
Home page: *http://www.gnu.org/software/gettext/*
Download: *http://ftp.gnu.org/gnu/gettext/gettext-0.18.1.1.tar.gz*
MD5 sum: 3dd55b952826d2b32f51308f2f91aa89
• Glibc (2.14.1) - 15,284 KB:
Home page: *http://www.gnu.org/software/libc/*
Download: *http://ftp.gnu.org/gnu/glibc/glibc-2.14.1.tar.bz2*
MD5 sum: 5869a2620c6917dd392289864c6ce595
• GMP (5.0.2) - 1,977 KB:
Home page: *http://www.gnu.org/software/gmp/*
Download: *http://ftp.gnu.org/gnu/gmp/gmp-5.0.2.tar.bz2*
MD5 sum: 0bbaedc82fb30315b06b1588b9077cd3
• Grep (2.9) - 1,749 KB:
Home page: *http://www.gnu.org/software/grep/*
Download: *http://ftp.gnu.org/gnu/grep/grep-2.9.tar.gz*
MD5 sum: 03e3451a38b0d615cb113cbeaf252dc0
• Groff (1.21) - 3,774 KB:
Home page: *http://www.gnu.org/software/groff/*

Download: *http://ftp.gnu.org/gnu/groff/groff-1.21.tar.gz*
MD5 sum: 8b8cd29385b97616a0f0d96d0951c5bf

• GRUB (1.99) - 4,544 KB:
Home page: *http://www.gnu.org/software/grub/*
Download: *http://ftp.gnu.org/gnu/grub/grub-1.99.tar.gz*
MD5 sum: ca9f2a2d571b57fc5c53212d1d22e2b5

• Gzip (1.4) - 886 KB:
Home page: *http://www.gnu.org/software/gzip/*
Download: *http://ftp.gnu.org/gnu/gzip/gzip-1.4.tar.gz*
MD5 sum: e381b8506210c794278f5527cba0e765

• Iana-Etc (2.30) - 201 KB:
Home page: *http://freshmeat.net/projects/iana-etc/*
Download:
http://anduin.linuxfromscratch.org/sources/LFS/lfs-packages/conglomeration//iana-etc/iana-etc-2.30.
tar.bz2
MD5 sum: 3ba3afb1d1b261383d247f46cb135ee8

• Inetutils (1.8) - 1,810 KB:
Home page: *http://www.gnu.org/software/inetutils/*
Download: *http://ftp.gnu.org/gnu/inetutils/inetutils-1.8.tar.gz*
MD5 sum: ad8fdcdf1797b9ca258264a6b04e48fd

• IPRoute2 (2.6.39) - 465 KB:
Home page:
http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2
Download:
http://devresources.linuxfoundation.org/dev/iproute2/download/iproute2-2.6.39.tar.gz
MD5 sum: 8a3b6bc77c2ecf752284aa4a6fc630a6

• Kbd (1.15.2) - 1,520 KB:
Download:
http://anduin.linuxfromscratch.org/sources/LFS/lfs-packages/conglomeration/kbd/kbd-1.15.2.tar.gz
MD5 sum: 77d0b51454522bc6c170bbdc6e31202a

• Less (444) - 301 KB:
Home page: *http://www.greenwoodsoftware.com/less/*
Download: *http://www.greenwoodsoftware.com/less/less-444.tar.gz*
MD5 sum: 56f9f76ffe13f70155f47f6b3c87d421

• LFS-Bootscripts (20111017) - 32 KB:
Download:
http://www.linuxfromscratch.org/lfs/downloads/7.0/lfs-bootscripts-20111017.tar.bz2
MD5 sum: 7a229a3f297afac2f53dec64be37c6df

• Libpipeline (1.2.0) - 670 KB:
Home page: *http://libpipeline.nongnu.org/*
Download:
http://download.savannah.gnu.org/releases/libpipeline/libpipeline-1.2.0.tar.gz

MD5 sum: dd3a987a0d2b594716baee2f73d61ae3

• Libtool (2.4) - 2,520 KB:

Home page: *http://www.gnu.org/software/libtool/*

Download: *http://ftp.gnu.org/gnu/libtool/libtool-2.4.tar.gz*

MD5 sum: b32b04148ecdd7344abc6fe8bd1bb021

• Linux (3.1) - 75,381 KB:

Home page: *http://www.kernel.org/*

Download: *http://www.kernel.org/pub/linux/kernel/v3.x/linux-3.1.tar.bz2*

MD5 sum: 8d43453f8159b2332ad410b19d86a931

Note

Linux 内核经常在更新，许多都是由于发现了安全漏洞。最近的 Linux3.1.x 内核是可用的，除非勘误表页面声明不再安全。对于有限网速或昂贵贷款的用户，如果希望更新 Linux 内核，可以分开下载基本版本的软件包和补丁。这样能节约时间或者只花费较少从小版本升级。

• M4 (1.4.16) - 1,229 KB:

Home page: *http://www.gnu.org/software/m4/*

Download: *http://ftp.gnu.org/gnu/m4/m4-1.4.16.tar.bz2*

MD5 sum: 8a7cef47fecab6272eb86a6be6363b2f

• Make (3.82) - 1,213 KB:

Home page: *http://www.gnu.org/software/make/*

Download: *http://ftp.gnu.org/gnu/make/make-3.82.tar.bz2*

MD5 sum: 1a11100f3c63fcf5753818e59d63088f

• Man-DB (2.6.0.2) - 2,322 KB:

Home page: *http://www.nongnu.org/man-db/*

Download: *http://download.savannah.gnu.org/releases/man-db/man-db-2.6.0.2.tar.gz*

MD5 sum: 2b41c96efec032d2b74ccbf2e401f93e

• Man-pages (3.35) - 1,650 KB:

Home page: *http://man7.org/linux/man-pages/index.html*

Download: *http://man7.org/linux/man-pages/download/man-pages-3.35.tar.gz*

MD5 sum: e41432ee35a49036bbaf8d4598506e9c

• Module-Init-Tools (3.16) - 224 KB:

Home page: *https://modules.wiki.kernel.org/index.php/Module_init_tools_3_12*

Download:
http://anduin.linuxfromscratch.org/sources/LFS/lfs-packages/conglomeration/module-init-tools/
module-init-tools-3.16.tar.bz2

MD5 sum: bc44832c6e41707b8447e2847d2019f5

• MPC (0.9) - 553 KB:

Home page: *http://www.multiprecision.org/*

Download: *http://www.multiprecision.org/mpc/download/mpc-0.9.tar.gz*

MD5 sum: 0d6acab8d214bd7d1fbbc593e83dd00d

• MPFR (3.1.0) - 1,176 KB:

Home page: *http://www.mpfr.org/*

Download: *http://www.mpfr.org/mpfr-3.1.0/mpfr-3.1.0.tar.bz2*
MD5 sum: 238ae4a15cc3a5049b723daef5d17938

• Ncurses (5.9) - 2,760 KB:
Home page: *http://www.gnu.org/software/ncurses/*
Download: *ftp://ftp.gnu.org/gnu/ncurses/ncurses-5.9.tar.gz*
MD5 sum: 8cb9c412e5f2d96bc6f459aa8c6282a1

• Patch (2.6.1) - 248 KB:
Home page: *http://savannah.gnu.org/projects/patch/*
Download: *http://ftp.gnu.org/gnu/patch/patch-2.6.1.tar.bz2*
MD5 sum: 0818d1763ae0c4281bcdc63cdac0b2c0

• Perl (5.14.2) - 12,917 KB:
Home page: *http://www.perl.org/*
Download: *http://www.cpan.org/src/5.0/perl-5.14.2.tar.bz2*
MD5 sum: 04a4c5d3c1f9f19d77daff8e8cd19a26

• Procps (3.2.8) - 279 KB:
Home page: *http://procps.sourceforge.net/*
Download: *http://procps.sourceforge.net/procps-3.2.8.tar.gz*
MD5 sum: 9532714b6846013ca9898984ba4cd7e0

• Psmisc (22.14) - 374 KB:
Home page: *http://psmisc.sourceforge.net/*
Download: *http://prdownloads.sourceforge.net/psmisc/psmisc-22.14.tar.gz*
MD5 sum: ba3f4e971895c92bba7770d81c981503

• Readline (6.2) - 2,225 KB:
Home page: *http://cnswww.cns.cwru.edu/php/chet/readline/rltop.html*
Download: *http://ftp.gnu.org/gnu/readline/readline-6.2.tar.gz*
MD5 sum: 67948acb2ca081f23359d0256e9a271c

• Sed (4.2.1) - 878 KB:
Home page: *http://www.gnu.org/software/sed/*
Download: *http://ftp.gnu.org/gnu/sed/sed-4.2.1.tar.bz2*
MD5 sum: 7d310fbd76e01a01115075c1fd3f455a

• Shadow (4.1.4.3) - 1,762 KB:
Home page: *http://pkg-shadow.alioth.debian.org/*
Download: *http://pkg-shadow.alioth.debian.org/releases/shadow-4.1.4.3.tar.bz2*
MD5 sum: b8608d8294ac88974f27b20f991c0e79

• Sysklogd (1.5) - 85 KB:
Home page: *http://www.infodrom.org/projects/sysklogd/*
Download: *http://www.infodrom.org/projects/sysklogd/download/sysklogd-1.5.tar.gz*
MD5 sum: e053094e8103165f98ddafe828f6ae4b

• Sysvinit (2.88dsf) - 108 KB:
Home page: *http://savannah.nongnu.org/projects/sysvinit*
Download:
http://download.savannah.gnu.org/releases/sysvinit/sysvinit-2.88dsf.tar.bz2
MD5 sum: 6eda8a97b86e0a6f59dabbf25202aa6f

• Tar (1.26) - 2,285 KB:

Home page: *http://www.gnu.org/software/tar/*
Download: *http://ftp.gnu.org/gnu/tar/tar-1.26.tar.bz2*
MD5 sum: 2cee42a2ff4f1cd4f9298eeeb2264519

• Tcl (8.5.10) - 4,393 KB:
Home page: *http://tcl.sourceforge.net/*
Download: *http://prdownloads.sourceforge.net/tcl/tcl8.5.10-src.tar.gz*
MD5 sum: a08eaf8467c0631937067c1948dd326b

• Texinfo (4.13a) - 2,687 KB:
Home page: *http://www.gnu.org/software/texinfo/*
Download: *http://ftp.gnu.org/gnu/texinfo/texinfo-4.13a.tar.gz*
MD5 sum: 71ba711519209b5fb583fed2b3d86fcb

• Udev (173) - 594 KB:
Home page: *http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html*
Download:
http://anduin.linuxfromscratch.org/sources/LFS/lfs-packages/conglomeration/udev/udev-173.tar.bz2
MD5 sum: 91a88a359b60bbd074b024883cc0dbde

• Udev Test Tarball (173) - 152 KB:
Download:
http://anduin.linuxfromscratch.org/sources/other/udev-173-testfiles.tar.bz2
MD5 sum: d97f80f6a70cd97f0519b14f15e3e195

• Udev Configuration Tarball - 7 KB:
Download:
http://www.linuxfromscratch.org/lfs/downloads/7.0/udev-config-20100128.tar.bz2
MD5 sum: caef7ea7331ab4f1d498e16b637a40c7

• Util-linux (2.20) - 4,507 KB:
Home page: *http://userweb.kernel.org/~kzak/util-linux/*
Download:
http://anduin.linuxfromscratch.org/sources/LFS/lfs-packages/conglomeration/util-linux/util-linux-2.20.
tar.bz2
MD5 sum: 4dcacdbdafa116635e52b977d9d0e879

• Vim (7.3) - 8,675 KB:
Home page: *http://www.vim.org*
Download: *ftp://ftp.vim.org/pub/vim/unix/vim-7.3.tar.bz2*
MD5 sum: 5b9510a17074e2b37d8bb38ae09edbf2

• Xz Utils (5.0.3) - 1,002 KB:
Home page: *http://tukaani.org/xz*
Download: *http://tukaani.org/xz/xz-5.0.3.tar.bz2*
MD5 sum: 8d900b742b94fa9e708ca4f5a4b29003

• Zlib (1.2.5) - 532 KB:
Home page: *http://www.zlib.net/*
Download: *http://www.zlib.net/zlib-1.2.5.tar.bz2*
MD5 sum: be1e89810e66150f5b0327984d8625a0

这些包的总大小大约为：294MB。

3.3 必要的补丁

除了这些软件包，还需要一些补丁文件。这些补丁修正了本应该被软件维护者修正的错误。这些补丁同时也做了一些修改以便于软件包能更好地工作。以下是搭建 LFS 系统必要的补丁：

• Bash Upstream Fixes Patch - 14 KB:
Download: *http://www.linuxfromscratch.org/patches/lfs/7.0/bash-4.2-fixes-3.patch*
MD5 sum: 16ef261d87673ffaa6e838423d1cc4d1

• Bzip2 Documentation Patch - 1.6 KB:
Download:
http://www.linuxfromscratch.org/patches/lfs/7.0/bzip2-1.0.6-install_docs-1.patch
MD5 sum: 6a5ac7e89b791aae556de0f745916f7f

• Coreutils Internationalization Fixes Patch - 122 KB:
Download:
http://www.linuxfromscratch.org/patches/lfs/7.0/coreutils-8.14-i18n-1.patch
MD5 sum: fd08f2a15c14a9bd2b675fd7f8d54d08

• Coreutils Uname Patch - 1.6 KB:
Download:
http://www.linuxfromscratch.org/patches/lfs/7.0/coreutils-8.14-uname-1.patch
MD5 sum: 500481b75892e5c07e19e9953a690e54

• Flex GCC-4.4.x Patch - 1 KB:
Download: *http://www.linuxfromscratch.org/patches/lfs/7.0/flex-2.5.35-gcc44-1.patch*
MD5 sum: ad9109820534278c6dd0898178c0788f

• GCC Cross Compile Patch - 1.8 KB:
Download:
http://www.linuxfromscratch.org/patches/lfs/7.0/gcc-4.6.1-cross_compile-1.patch
MD5 sum: 1b7886a7a4df3a48617e88a481862264

• GCC Startfiles Fix Patch - 1.5 KB:
Download:
http://www.linuxfromscratch.org/patches/lfs/7.0/gcc-4.6.1-startfiles_fix-1.patch
MD5 sum: 799ef1971350d2e3c794f2123f247cc6

• Glibc Bug Fixes Patch - 5.5 KB:
Download: *http://www.linuxfromscratch.org/patches/lfs/7.0/glibc-2.14.1-fixes-1.patch*
MD5 sum: 13bdfb7db1654d9c3d7934d24479a6c4

• Glibc GCC Build Fix Patch - 2.5 KB:
Download:
http://www.linuxfromscratch.org/patches/lfs/7.0/glibc-2.14.1-gcc_fix-1.patch
MD5 sum: d1f28cb98acb9417fe52596908bbb9fd

• Glibc GCC CPUID Patch - 0.8 KB:
Download:
http://www.linuxfromscratch.org/patches/lfs/7.0/glibc-2.14.1-cpuid-1.patch
MD5 sum: 4f110dc9c8d4754fbda841492ce796b4

• GCC Locale Patch - 4 KB:

Download: *http://www.linuxfromscratch.org/patches/lfs/7.0/gcc-4.6.1-locale-1.patch*
MD5 sum: 406572f979f480be1450eb88eea08caa

• Kbd Backspace/Delete Fix Patch - 12 KB:
Download:
http://www.linuxfromscratch.org/patches/lfs/7.0/kbd-1.15.2-backspace-1.patch
MD5 sum: f75cca16a38da6caa7d52151f7136895

• Module Init Tools - 44 KB:
Download:
http://www.linuxfromscratch.org/patches/lfs/7.0/module-init-tools-3.16-man_pages-1.patch
MD5 sum: e90aa105293df7b8691fd8ac697cefc9

• Patch Testsuite Fix Patch - 1 KB:
Download:
http://www.linuxfromscratch.org/patches/lfs/7.0/patch-2.6.1-test_fix-1.patch
MD5 sum: c51e1a95bfc5310635d05081472c3534

• Perl Libc Patch - 1 KB:
Download: *http://www.linuxfromscratch.org/patches/lfs/7.0/perl-5.14.2-libc-1.patch*
MD5 sum: 23682f20b6785e97f99d33be7719c9d6

• Procps HZ Errors Patch - 2.3 KB:
Download:
http://www.linuxfromscratch.org/patches/lfs/7.0/procps-3.2.8-fix_HZ_errors-1.patch
MD5 sum: 2ea4c8e9a2c2a5a291ec63c92d7c6e3b

• Procps Watch Patch - 3.5 KB:
Download:
http://www.linuxfromscratch.org/patches/lfs/7.0/procps-3.2.8-watch_unicode-1.patch
MD5 sum: cd1a757e532d93662a7ed71da80e6b58

• Readline Upstream Fixes Patch - 1.3 KB:
Download: *http://www.linuxfromscratch.org/patches/lfs/7.0/readline-6.2-fixes-1.patch*
MD5 sum: 3c185f7b76001d3d0af614f6f2cd5dfa

这些补丁总大小约为：221.4KB。

除了上述必要的补丁，这里还有一些 LFS 社区创建的可选的补丁。这些可选的补丁要么解决了更小的问题要么使能了默认情况下禁用的功能。请自愿使用补丁数据库 *http://www.linuxfromscratch.org/patches/downloads/* ，获取任何附加补丁来满足系统需要。

第四章 最后准备

4.1 关于$LFS

纵览全书，会使用 LFS 这个环境变量。它是一个定义好后一直存在的变量。它要被设置到 LFS 分区选定的挂载点上。检查 LFS 变量是否正确设置：

```
echo $LFS
```

确保输出显示了到 LFS 分区挂载点的路径，根据前面的例子应该是/mnt/lfs。如果输出不正确，可设置该变量为：

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

设定好该变量后有利于以后键入命令如 mkdir $LFS/tools 时简化输入。在命令行运行命令时，Shell 会自动用“/mnt/lfs”（或者任意设置值）代替“$LFS“。

在你离开然后重新进入当前工作环境（如使用 su 切换到 root 用户或者其他用户）时一定记得检查$LFS 是否设置。

4.2 创建$LFS/tools 目录

在第五章编译的所有程序都会被安装到$LFS/tools 目录下以便于和第六章编译的程序相分离。在该目录下编译的程序是临时工具，他们不是最终 LFS 系统的一部分。通过保持他们在一个独立的目录，在使用完后可以方便地丢弃。这样同时也可以放置这些程序在宿主系统产生目录（在第五章中很容易实现）。

以 root 用户运行下面命令可以创建需要的目录：

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

下一步是在宿主系统里创建一个/tools 的符号链接。该链接指向 LFS 分区中新建的目录。以 root 用户运行命令：

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

Note

上面的命令是正确的。**ln** 命令有一些句法变化，所以在你提出你认为可能有错时请先查阅 **info coreutils ln** 和 ln(1)手册。

该符号链接总是指向/tools，保证了工具链的编译，意味着编译器，汇编器和链接器能在第五章（我们仍然会使用宿主机的部分工具）及以后（当我们”chroot”到 LFS 分区）都能工作。

4.3 添加 LFS 用户

当以 root 用户登录时，一个小小的错误就可能破坏或摧毁系统。因而，在本章中我们推荐以非特权用户来编译软件包。你可以使用你自己的用户名，但是为了设置一个干净的工作环境，应创建一个叫做 lfs 的新用户，并让它成为一个新用户组（也叫 lfs）的成员，然后在搭建过程中使用该用户。作为 root 用户，键入下面命令来新建用户：

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

命令行参数的含义是：

-s /bin/bash

让 lfs 用户的默认 shell 是 **bash**。

-g lfs

将 lfs 用户添加到 lfs 用户组。

-m

为 lfs 用户创建家目录。

-k /dev/null

通过改变输入位置到特殊的 null 设备，达到阻止从框架目录（默认是/etc/skel）的可能的文件复制。

lfs

这是新创建的用户和用户组的实际名称。

为了能以 lfs 用户登录（root 用户登录后切换到 lfs 用户不需要 lfs 用户的密码，而这和以 lfs 用户登录时不同的），要为 lfs 用户设置密码：

```
passwd lfs
```

让 lfs 成为$LFS/tools 目录的拥有者，确保 lfs 用户对该目录的完全访问权限：

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

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

如果你按照建议创建了分离的工作目录，请将该目录的拥有者改成 lfs：

下一步，以 lfs 用户登录。可用通过基于现实管理器的虚拟终端登录，或如下的替代命令：

```
su - lfs
```

减号“-”表示 **su** 启动的是一个登录 shell 而不是非登录 shell。这两种 shell 的不同可以在 bash(1)和 **info bash** 手册中找到详细信息。

4.4 设立工作环境

通过创建两个新的 **bash** shell 启动文件来设立一个较好的工作环境。当以 lfs 用户登录时，按照下面的命令创建新的 .bash_profile 文件：

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

当我们以 lfs 登录时，初始 shell 通常是登录 shell，它会读取主机的/etc/profile 文件（很可能包含了一些设置和环境变量），然后读取 .bash_profile。除了 HOME, TERM 和 PS1 外，在.bash_profile 文件中的 **exec env –i…/bin/bash** 命令会以一个完全空的环境来代替正在运行的 shell。这样确保没有不需要的或有害的环境变量从宿主系统遗留到搭建中的环境中。这里使用的技术就可以保证一个干净的环境。

这个 shell 新实例是一个非登录 shell，它不会读取/etc/profile 或 .bash_profile 文件，但是会读取.bashrc 文件。下面我们创建 .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
```

命令 **set +h** 关闭 **bash** 的 hash 功能。按理说 Hash 是一个有用的特性——**bash** 使用 hash 表来记住可执行文件的全路径，以避免重复搜索 PATH 来定位同一个文件的时间。然而，新工具一旦被安装就希望尽快被使用。所以，一旦编译好的新工具在$LFS/tools 中可用就希望 shell 能马上找到这个新版本，而不是记住这个程序的之前的版本的位置。

设置用户的文件创建掩码（umask）为 022，保证新创建的文件和目录仅对拥有者可写，但对任何人（若系统的 open（2）系统调用使用默认模式，新文件的权限模式为 644，新目录为 755）都可读可执行。

变量 LFS 需要被设定到选取的挂载点。

变量 LC_ALL 控制某些程序的本地化，让它们的信息依据特定国家的约定。如果宿主系统使用了非 2.2.4 的 Glibc，且 LC_ALL 变量也设定为非”POSIX”或”C”（在本章中），那么在退出 chroot 环境后希望再回去就可能出问题。设置 LC_ALL 为”POSIX”或”C”（这两者是等效的）可确保在 chroot 环境中一切都能正常工作。

变量 LFS_TGT 没有设置为默认值，但考虑到当编译我们的临时工具链时制造交叉编译连接器的使用设为兼容机器描述了。更多信息在 5.2 节“工具链技术注解”。

将/tools/bin 放在了在标准 PATH 环境变量的最前面，保证在第五章中所有程序一旦安装好后就可以被 shell 立刻获取。这和 hash 的关闭联系起来，就可以保证在第五章环境中安装的同样程序新版本后，限制了从宿主系统调用旧程序的风险。

最后，为了完全准备好搭建临时工具的环境，使刚创建的用户配置生效：

```
source ~/.bash_profile
```

4.5 关于 SBU

许多人想在编译和安装软件包之前了解大概需要多少时间。因为 Linux From Scratch 可以在许多不同的系统上搭建，所以不可能提供准确的时间。在最快的系统上，最大的包（Glibc）大约会花 20 分钟，但在慢的系统上却可能花费长达 3 天时间！相比于提供实际时间，会以标准搭建时间元（SBU）测量方法作为替代。

SBU 的工作原理如下。在第五章中第一个被编译的包是 Binutils。编译它花费的时间被被当作标准搭建时间元或 SBU。所有其他编译时间会以和该时间相关方式表达。

例如，若某个包的编译时间是 4.5SBU。这意味着，如果系统花费 10 分钟来完成 Binutils 的第一遍编译安装，就会花费大约 45 分钟来完成这个包。幸运的是，大多数要比 Binutils 花的时间短。

一般地，考虑到宿主系统的 GCC 版本等诸多因素，SBU 时间并不是完全准确。它的作用只是为了对软件安装给出一个估计时间，但这个数值可能在某些情况下会相差数十分钟。

如果要查阅一系列特定机器的实际时间，我们推荐 Linux From Scratch 的 SBU 主页 http://www.linuxfromscratch.org/~sbu/ 。

对大多数拥有多处理器（或核心）的现代系统，可以通过设置环境变量或指定 **make** 程序有多少处理器可用来实现并行 **make**，这样能减少编译时间。例如，Core2Duo 能够同时支持两个进程：

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

或用下面的命令：

```
make -j2
```

当以这种方式使用了多处理器，本书中的 SBU 会比正常情况下相差更大。由于不同处理器的多线交错，分析搭建进程会变得更加困难。如果你在搭建步骤中遇到问题，就还原到单处理器搭建来分析错误信息。

4.6 关于测试套件

许多包都提供了一个测试套件。对新安装的软件包运行测试套件是一个不错的选择，因为它提供了一个“合理的检查”来判断是否一切都编译正确。一个测试套件的通过集通常证明了软件的功能是否如开发者希望的一样工作。然而，这并不能保证软件是完全没有 bug 的。

一些测试套件比其他的更重要。举例来说，核心工具链包——GCC，Binutils 和 Glibc，由于它们在正常功能系统的中心位置，所以它们的测试套件是最重要的。GCC, Glibc 的测试套件会花费很长的时间才能结束，尤其是在慢的硬件上，但还是强烈推荐运行这几个测试套件。

经验显示，在第五章没有必要运行测试套件。因为在第五章中，宿主系统会不可避免地对测试施加影响，这往往会引起莫名其妙的失败。因为第五章中的工具是临时的最后会被抛弃，所以对普通读者我们不推荐在第五章运行测试套件。运行这些套件的指导是为了给测试者和开发者提供方便，但它们都是严格可选的。

为GCC和Binutils运行测试套件的一个常见的问题是在伪终端(PTYs)之外运行。这会导致大量的测试错误。这个发生的原因有很多，但最可能的诱因是宿主系统并没有正确设置devpts文件系统。这个问题在http://www.linuxfromscratch.org//lfs/faq.html#no-ptys有非常详细的讨论。

有时包的测试套件会失败，但开发者已经意识到且将其视为非关键原因。咨询站点 http://www.linuxfromscratch.org/lfs/build-logs/7.0/ 的日志来验证这些错误是否是已预期到的。对全书的所有测试这个站点都是正确的。

第五章 搭建临时系统

5.1 简介

本章介绍怎样搭建一个最小 Linux 系统。该系统仅包含搭建第六章的最终 LFS 系统必要的工具，允许一个最小的工作环境给予用户便利。

搭建最小系统需要两步。第一步是制作一个新的独立于宿主系统的工具链（包括编译器，汇编器，连接器，库和一些有用的工具）。第二步是使用这条工具链构建其他必须的工具。

本章中编译的文件会被安装到$LFS/tools 目录下，以保证和下一章安装的文件与宿主产生的文件夹分离。因为这里编译的包是临时的，我们不希望它们污染即将搭建的 LFS 系统。

Important

在继续之前，要知道工作平台的名称，通常被引用为三个目标。一个最简单确定目标平台名称的办法就是运行源软件包自带的 **config.guess** 脚本。解压缩 Binutils 源文件然后运行脚本：**./config.guess** 然后留意输出。举个例子，对于一个现代 32-bit 的 Intel 处理器，输出很可能是 *i686-pc-linux-gnu*。

同样还要知道平台的动态链接器的名称，通常被引用为动态加载器（不要和标准链接器——Binutils 的 **ld** 相混淆）。动态链接器由 Glibc 提供，用于查找和加载程序需要的共享库，为程序运行做准备，并运行程序。一个 32-bit 的 Intel 机器的动态链接器的名称是 ld-linux.so。确定动态链接器的一个万无一失的方法是用后面的命令检查宿主系统的任一二进制文件：**readelf -l <name of binary> | grep interpreter** 然后留意输出。包含所有平台的权威引用在 Glibc 源码树的根下的 shlib-versions 文件里。

5.2 工具链技术注解

本节解释了在搭建方法后的一些理由和技术细节。不需要在本节就立刻理解全部的内容。在进行实际的搭建时大多数信息会逐渐明朗。在搭建过程中可以随时查阅本节。

第五章的全部目标是建立一个临时区域，包含可以从宿主系统分离的全部已知工具集。通过使用 **chroot**，余下章节的命令会被包含在 chroot 环境中，确保一个纯净，无错的目标 LFS 系统的搭建。对于新读者，搭建过程已经被设计成能最小化出错的风险，且同时能提供最大的学习价值。

下面是第五章搭建方法工作的一些技术要点：

- 通过 LFS_TGT 变量的方式改变目标平台的“供应商”域，略微调整一下工作平台的名称，确保第一次编译 Binutils 和 GCC 能产生一个兼容的交叉链接器和交叉编译器。它们会产生和当前硬件兼容的二进制文件，避免了专门为另一个架构产生二进制文件。
- 临时库也是交叉编译的。由于交叉编译器天生不会依赖于宿主系统，该方法通过减少宿主系统的头文件和函数库被带进新工具，消除了目标系统的潜在污染。交叉编译也满足了在 64-bit 硬件上同时构建 32-bit 和 64-bit 函数库的可能性。
- 谨慎操作 **gcc** 的 specs 文件，告诉编译器应该使用哪个目标动态链接器。

Binutils 必须最先安装，因为 GCC 和 Glibc 运行 configure 脚本会对汇编器和链接器执行不同的特性测试，以便于决定某些软件特性是否启用。这是比首次意识到更重要的。GCC 和 Glibc 的不正确配置会导致工具链微妙的错误，而这些错误可能知道整个发行版搭建结束才会被发现。在做更多的额外工作前一个测试套件错误通常会凸显这个错误。

Binutils 将其汇编器和链接器安装在两个位置，/tools/bin 和/tools/$LFS_TGT/bin。其中一个是另一个的硬链接。链接器的一个重要方面是它的库搜索顺序。详细信息可以通过给 **ld** 传递*--verbose* 标志来获得。例如，**ld --verbose | grep SEARCH** 会输出当前的查找路径及其顺序。通过编译一个哑程序并传递*--verbose* 开关给链接器可以看到哪个文件被 **ld** 链接到了。例如，**gcc dummy.c –Wl,--verbose 2>&1 |grep succeeded** 会输出在链接过程中所有成功打开的文件。

第二个要安装的包是 GCC。在运行 configure 脚本时可以看到：

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

由于上面提到的原因，这是非常重要的。这同样也演示了 GCC 的配置脚本不会去 PATH 目录寻找使用哪个工具。然而，在 **gcc** 自己实际的操作中，不必要使用同样的搜索路径。为了找到 **gcc** 使用的是哪个标准链接器，运行：**gcc –print-prog-name=ld**。

在编译哑程序时，可以给 **gcc** 传递命令行选项-v 获得详细信息。例如，gcc –v dummy.c 会输出关于预处理器，编译器，汇编平台包括 **gcc** 内置搜索路径及顺序等等详细信息。

然后再安装 Glibc。构建 Glibc 最需要考虑的是编译器，二进制工具和内核头文件。由于 Glibc 总是使用 configure 脚本传递的*--host* 参数决定的编译器，所以编译器不是需要考虑的。例如在我们的是，**i686-lfs-linux-gnu-gcc**。二进制工具和内核头文件就要麻烦一点了。因而，不要冒险，直接使用可用的配置开关强制正确的选择。在运行 configure 后，检查 glibc-build 目录中的 config.make 文件的内容获得全部重要信息。注意 CC="i686-lfs-gnu-gcc"是控制使用哪个二进制文件，-nostdinc 和-isystem 标志是控制编译器的搜索路径。这些项凸显了 Glibc 的一个重要方面——基于它的建造机理是自给自足的，一般不依赖于默认工具链。

在安装好 Glibc 之后，改变 **gcc** 的 specs 文件指向/tools/lib 中新的的动态链接器。这最后一步是至关重要的，确保仅在/tools 为前缀情况下搜索和链接。一条到动态链接器的硬线路径被嵌入到每个可执行的 ELF(Executable and Link Format)文件中。可以通过运行：**readelf –l <name of binary> | grep interpreter** 来检验。修改 **gcc** 的 specs 文件后能保证从现在开始到本章结束时编译的每个程序都会链接到/tools/lib 中的新动态链接器。

第二遍编译 GCC 时，仍然需要修改 GCC 的源代码让它使用新的动态链接器。如果修改失败，将会导致 GCC 将宿主系统的/lib 目录下的动态链接器的名称嵌入其中，这将会导致和宿主系统分离的目标落空。

在第二遍编译 Binutils 时，我们能利用配置开关*--with-lib-path* 控制 **ld** 的库搜索路径。从这一刻开始，核心工具链能够自满足和自管理。第五章的其余包都是基于/tools 目录中的新 Glibc 来构建的。

当在第六章进入 chroot 环境后，由于 Glibc 的天然自给自足的特性，第一个重要的要安装的包就是 Glibc。一旦 Glibc 被安装到/usr 后，我们会进行一个快速的工具链转换，然后继续搭建目标 LFS 系统的其余部分。

5.3 通用编译说明

在编译软件包时有许多假设：

- 一些软件包在编译前需要打补丁，但仅在补丁文件能规避存在的问题的情况下。在这里和下一章常常需要打补丁，但有时只需要其中一个。因而，如果丢失了一个下载的补丁不要过于纠结。当打补丁时也会常常遇到关于 *offset* 或者 *fuzz* 的警告信息。不要担心那些警告，因为补丁仍然成功被应用了。
- 在编译大多数软件包时，会有许多警告信息在屏幕上滚动。这都是正常的，可以安全地

被忽略。那些警告就像它们显示的——警告某些用法已过时，但并非不正确的 C 或 C++ 的语法。标准的 C 时刻在变化，但一些包仍然使用的较旧的标准。这不是什么问题，但的确会弹出警告。

- 最后再检查一次 LFS 环境变量是否正确设置：

```
echo $LFS
```

确定输出显示路径是 LFS 分区的挂载点，在我们的例子中是/mnt/lfs。

- 最后，必须强调最后两个要点：

Important

搭建说明假设正在使用的是 **bash** shell。

Important

再次强调搭建流程：

1. 请将所有的源文件和补丁放置在 chroot 环境可以访问的目录，比如/mnt/lfs/sources。不要放置源文件在/mnt/lfs/tools/下。
2. 进入源文件目录。
3. 对每个软件包：
   a) 使用 **tar** 程序将软件包解压缩。在第五章，当你解压缩包时确保你是 *lfs* 用户。
   b) 解压缩后进入解压缩产生的目录。
   c) 根据书中的指示编译软件包。
   d) 回到源目录。
   e) 除非教材指出不能删除外，删除解压缩后的源文件夹和任何构建编译时产生的目录。

5.4 Binutils-2.21.1a——第一遍

Binutils 包含一个链接器，一个汇编器，和其他处理二进制文件的工具。

估计构建时间：1SBU

需要磁盘空间：350MB

5.4.1 安装交叉 Binutils

Note

请再复习一下前面部分。深刻理解标记为 Important 的部分会在以后大量节省你的时间。

Binutils 必须最先安装，因为 GCC 和 Glibc 运行 configure 脚本会对汇编器和链接器执行不同的特性测试，以便于决定某些软件特性是否启用。

Binutils 的文档推荐最好在源文件目录外的一个专用目录里编译：

```
mkdir -v ../binutils-build
cd ../binutils-build
```

Note

为了在本书后面的那些 SBU 值能派上用处，请记录从配置开始到安装结束的时间。为了方便地完成，包裹这 3 个命令在 **time** 命令中：**time { ./configure … && make && make install; }** 。

Note

在第五章中，估计构建 SBU 值和需要的磁盘空间都不包含测试套件数据。

现在准备编译 Binutils，配置：

```
../binutils-2.21.1/configure \
  --target=$LFS_TGT --prefix=/tools \
  --disable-nls --disable-werror
```

配置参数的含义：

--target=$LFS_TGT

由于 LFS_TGT 变量的机器描述和 **config.guess** 脚本返回的值有所不同，该参数会让 **configure** 脚本调整 Binutils 的搭建系统构建一个交叉链接器。

--prefix=/tools

该参数使 configure 脚本准备在/tools 目录下安装 Binutils。

--disable-nls

该参数禁用国际化因为 i18n 对临时工具来说是不必要的。

--disable-werror

该参数防止因为宿主编译器弹出警告而终止编译。

继续编译：

```
make
```

现在编译结束。按理说我们应该运行测试套件了，但现在测试套件框架（Tcl,Expect 和 DejaGNU）还没有就绪。第一遍编译就运行测试套件获益是最小的，因为我们很快就会用第二次编译的程序替代它。

如果是在 x86_64 位机上搭建，创建一个符号链接保证工具链的合理性：

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

安装软件包：

```
make install
```

Binutils 包的详细信息位于 6.13.2 节“Binutils 的内容”。

5.5 GCC-4.6.1——第一遍

GCC 包含了 GNU 编译器集合，其中就包含 C 和 C++编译器。

```
make install
```

估计构建时间：5.0SBU
需要磁盘空间：1.2GB

5.5.1 安装交叉 GCC

现在 GCC 依赖于 GMP, MPFR 和 MPC 包。由于你的宿主系统可能没有这些包，它们需要

```
tar -jxf ../mpfr-3.1.0.tar.bz2
mv -v mpfr-3.1.0 mpfr
tar -jxf ../gmp-5.0.2.tar.bz2
mv -v gmp-5.0.2 gmp
tar -zxf ../mpc-0.9.tar.gz
mv -v mpc-0.9 mpc
```

和 GCC 一起编译。把他们都解压到 GCC 的源代码目录并重命名这些解压缩后的目录，这样编译 GCC 时就能自动使用它们：

```
patch -Np1 -i ../gcc-4.6.1-cross_compile-1.patch
```

应用下面的补丁，允许禁用 libiberty 和 zlib 的目标库的构建，因为在交叉编译环境中它们不能正确编译：

GCC 的文档推荐在 GCC 源代码目录外的专用目录编译：

```
mkdir -v ../gcc-build
cd ../gcc-build
```

准备编译：

```
../gcc-4.6.1/configure \
--target=$LFS_TGT --prefix=/tools \
--disable-nls --disable-shared --disable-multilib \
--disable-decimal-float --disable-threads \
--disable-libmudflap --disable-libssp \
--disable-libgomp --disable-libquadmath \
--disable-target-libiberty --disable-target-zlib \
--enable-languages=c --without-ppl --without-cloog \
--with-mpfr-include=$(pwd)/../gcc-4.6.1/mpfr/src \
--with-mpfr-lib=$(pwd)/mpfr/src/.libs
```

配置选项的含义：

--disable-shared

该开关强制 GCC 静态链接到自身内部函数库。这样可以避免和宿主系统可能的联系。

--disable-decimal-float, --disable-threads, --disable-libmudflap, --disablelibssp,
--disable-libgomp, --disable-libquadmath --disable-target-libiberty
--disable-target-zlib

这些开关禁用了以下支持：浮点小数扩展，线程，libmudflap，libssp 和 libgomp，libquadmath，libiberty 和 zlib。在交叉编译时这些特性会导致编译失败，同时对交叉编译临时 libc 的任务也是不必要的。

--disable-multilib

在 x86_64 位机上，LFS 暂不支持多个库配置。该开关对 x86 是无害的。

--enable-languages=c

该选项保证仅构建 C 编译器。因为现在只需要 C 语言。

--without-ppl, --without-cloog

这些开关防止 GCC 根据 PPL 和 CLooG 库来构建，这些库目前在宿主系统，但在 chroot 环境中无法使用。

编译 GCC：

```
make
```

现在编译结束。此刻，测试套件可以正常运行，但是如前面提到的，测试套件框架还未就绪。现在运行测试获益有限，因为第一遍编译的程序很快会被替换。

安装软件：

使用*--disable-shared* 意味着 libgcc_eh.a 文件还没有被创建安装。Glibc 依赖于这个库，在它的构建系统中使用了-lgcc_eh。该依赖可以通过创建一个到 libgcc.a 的链接来解决，该文件会包含正常应该包含在 libgcc_eh.a 中的对象：

```
ln -vs libgcc.a `$LFS_TGT-gcc -print-libgcc-file-name | \
sed 's/libgcc/&_eh/'`
```

关于该包的详细信息位于 6.17.2 节“GCC 内容”中。

5.6 Linux-3.1 API 头文件

Linux API 头文件（在 linux-3.1.tar.gz 中）暴露出内核的 API 供 Glibc 使用。

估计构建时间：0.1SBU

需要磁盘空间：511MB

5.6.1 安装 Linux API 头文件

Linux 内核需要暴露应用程序接口（API）供系统 C 库（LFS 的 Glibc）调用。通过从 Linux 源代码包获得不同 C 头文件并进行除污，就可以完成了。

确保在之前的行为中没有产生陈旧的文件和依赖：

```
make mrproper
```

现在从源代码中测试和提取用户可见的内核头文件。它们需要被放置在一个中间本地目录然后被复制到需要的位置，因为提取过程会删除目标目录中所有已存在的文件。

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

该包的详细信息位于 6.7.2 节“Linux API 头文件的内容”。

5.7 Glibc-2.14.1

Glibc 包含了主要的 C 库。该库提供了基本的例程如：分配内存，搜索目录，打开和关闭文件，读写文件，字符串操作，模式匹配，算术运算等等。

估计构建时间：5.5SBU

需要磁盘空间：501MB

5.7.1 安装 Glibc

修复之前 Glibc 构建 GCC-4.6.1 时的 bug：

```
patch -Np1 -i ../glibc-2.14.1-gcc_fix-1.patch
```

由于现在的编译环境还不完整，需要定位一个因此而失败的头文件检查：

```
patch -Np1 -i ../glibc-2.14.1-cpuid-1.patch
```

Glibc 的文档推荐在源代码目录外的一个专用目录编译：

```
mkdir -v ../glibc-build
cd ../glibc-build
```

由于 Glibc 不再支持 i386，其开发者建议使用编译标记*-march=i486* 为 x86 机器编译。有很多种方法可以达到目的，但测试显示该标记最好是被放在配置变量“CFLAGS”里。为了避免完全覆盖 Glibc 的内部构建系统中的 CFLAGS，我们利用了一个特殊的文件 configparms，将这个新编译标记追加到 CFLAGS 内容的尾部。当设置-march 时会改变-mtune 的值，所以*-mtune=native* 标记也是必要的，它的作用是将-mtune 复位为一个合理的值。

```
case `uname -m` in
  i?86) echo "CFLAGS += -march=i486 -mtune=native" > \
  configparms ;;
esac
```

接着，准备编译 Glibc：

```
../glibc-2.14.1/configure --prefix=/tools \
--host=$LFS_TGT
--build=$(../glibc-2.14.1/scripts/config.guess) \
--disable-profile --enable-add-ons \
--enable-kernel=2.6.25 --with-headers=/tools/include \
libc_cv_forced_unwind=yes libc_cv_c_cleanup=yes
```

配置选项的含义如下：

--host=$LFS_TGT, --build=$(../glibc-2.14.1/scripts/config.guess)

这些开关的作用就是将 Glibc 的构建系统配置为交叉编译，使用的是/tools 下的交叉链接器和交叉编译器。

--disable-profile

构建不含程序性能分析相关文件的库。如果在临时工具中需要程序分析信息就忽略该参数。

--enable-add-ons

使 Glibc 使用附加 NPTL 作为它的线程库。

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

--enable-kernel=2.6.25

使 Glibc 编译的库支持版本 2.6.25 及其以后的 Linux 内核。不支持更旧的内核。

--with-headers=/tools/include

使 Glibc 使用我们最新安装到/tools/include 中头文件来编译，Glibc 会根据内核的准确特性对自身做相应的优化。

libc_cv_forced_unwind=yes

我们在 5.4 节“Binutils-2.21.1a——第一遍”中安装的链接器是交叉编译的，所以只有在安装好了 Glibc 后才能使用。因为依赖于一个工作中的链接器，该选项意味着对 force_unwind 支持的配置测试会失败。变量 libc_cv_forced_unwind=yes 通知 **configure** 脚本可用 force-unwind 支持而不要进行测试。

libc_cv_c_cleanup=yes

同样的，传递 libc_cv_cleanup=yes 参数给 **configure** 脚本以跳过测试，而且 C 清理处理支持已配置。

在控制台可能出现下面的警告：

这个缺少或不兼容的 **msgfmt** 程序通常是无害的。**msgfmt** 程序是宿主系统应该提供的 Gettext 包的一部分。

编译软件包：

```
make
```

该软件包的确包含一个测试套件，但是，由于我们现在没有 C++编译器，所以现在不是时候运行测试套件。


Note

要成功地运行测试套件还需要安装语言环境数据。语言环境数据提供给系统以下信息：系统工具可接受的和可输出的数据，时间和货币格式。如果在本章中没有运行测试套件（如前的建议），则也没有必要安装语言环境。在下一章安装合适的语言环境才是合适的。如果实在要安装 Glibc 的语言环境，请依据 6.9 节”Glibc-2.14.1”的指导。

安装软件：

```
make install
```

关于 Glibc 软件包的详细信息位于 6.9.4 节“Glibc 的内容”。

5.8 调整工具链

现在我们已经安装好了临时 C 函数库，那么本章其余的工具都应该被编译链接到该库。要做到这点，交叉编译器的 specs 文件就必须要调整指向位于/tools 目录的新的动态链接器。

我们需要将编译器的“specs”文件倾倒在编译器默认会查找的位置。一个简单的 **sed** 命令替换就可以改变 GCC 使用的动态链接器。这里的原则是找到所有引用到/lib 目录或/lib64 目录（如果宿主是 64 位）下的链接文件，然后调整它们指向/tools 目录下新的位置。

考虑到准确性，当键入下面的命令时建议使用复制-粘贴方法。请确定亲眼检查 specs 文件，以验证确实正确地调整所有引用指向新动态链接器的位置。如果需要知道动态链接器的默认名称，请查阅 5.2 节“工具链技术注解”。

```
SPECS=`dirname $($LFS_TGT-gcc -print-libgcc-file-name)`/specs
$LFS_TGT-gcc -dumpspecs | sed \
-e 's@/lib\(64\)\?/ld@/tools&@g' \
-e "/^\*cpp:$/{n;s,$, -isystem /tools/include,}" > $SPECS
echo "New specs file is: $SPECS"
unset SPECS
```

Caution

再次，迫切需要停下来检查一下新工具链是否如我们预期地那样能正常工作（主要是编译器和链接器）。运行下面的命令进行正确性检查：

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

如果一切都正常工作，应该是没有错误的，最后一条命令输出应该是下面的形式：

```
[Requesting program interpreter:
/tools/lib/ld-linux.so.2]
```

注意动态链接器的前缀是/tools/lib，在 64 位机上是/tools/lib64。

如果没有输出或输出和上面不一样，说明什么地方出错了。折回去调查问题出在哪儿然后纠正。在继续下去之前务必解决错误的问题。可能在上面修改 specs 文件时出错了。如果是这样，重做 specs 文件的修改，修改时小心复制-粘贴命令。

一旦一切正常，删除测试文件：

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

Note

在下一节编译 Binutils 会作为对工具链构建正确性的一个额外的检查。如果 Binutils 编译安装失败，则表明在之前安装 Binutils，GCC，或 Glibc 时某些地方出问题了。

5.9 Binutils-2.21.1a——第二遍

Binutils 包包含了一个链接器，一个汇编器和其他处理目标文件的工具。

估计构建时间：1.1SBU

需要磁盘空间：363MB

5.9.1 安装 Binutils

再次创建一个分离的目录：

```
mkdir -v ../binutils-build
cd ../binutils-build
```

准备编译 Binutils：

```
CC="$LFS_TGT-gcc -B/tools/lib/" \
  AR=$LFS_TGT-ar RANLIB=$LFS_TGT-ranlib \
  ../binutils-2.21.1/configure --prefix=/tools \
  --disable-nls --with-lib-path=/tools/lib
```

下面是新的配置选项的含义：

CC="$LFS_TGT-gcc –B/tools/lib/" AR=$LFS_TGT-ar RANLIB=$LFS_TGT-ranlib

现在是编译一个真正的本地 Binutils，设置这些变量保证编译系统使用交叉编译及相关工具而不是用宿主系统的工具。

--with-lib-path=/tools/lib

该参数告诉 configure 脚本，在编译 Binutils 时指定函数库搜索路径，此处将/tools/lib 传递给链接器。这样就防止链接器在宿主系统库目录里搜索函数库。

编译软件：

```
make
```

安装软件：

```
make install
```

现在为下一章的“再次调整工具链”准备链接器：

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

下面是 make 参数的含义：

-C ld clean

该参数让 make 程序删除 ld 子目录下所有已编译的文件。

-C ld LIB_PATH=/usr/lib:/lib

该参数重新编译 ld 子目录下所有内容。在命令行指定 Makefile 的 LIB_PATH 变量，覆盖临时工具中的默认值并使它指向正确的路径。该变量的值指定了链接器的默认函数库搜索路径。该准备工作是为下一章做准备。

关于 Binutils 包的详细信息位于 6.13.2 节“Binutils 的内容”。

5.10 GCC-4.6.1——第二次

GCC 软件包包含了 GNU 编译器集，C 和 C++编译器也在其中。

估计构建时间：7.0SBU

需要磁盘空间：1.5GB

5.10.1 安装 GCC

GCC-4.3 之后的版本会将本次安装作为新位置的额外编译器，并且不允许从*--prefix* 指定的位置搜索头文件和库文件。实际上本次编译的并不是额外编译器，而且/tools 目录下的库文件对构建我们的工作编译器并将其链接到该目录下的库是非常关键的。应用下面的补丁可以将 GCC 恢复到正确的行为：

```
patch -Np1 -i ../gcc-4.6.1-startfiles_fix-1.patch
```

在正常情况下，运行 GCC 的 fixincludes 脚本是为了修正头文件潜在的错误。但因为现在已经安装了 GCC-4.6.1 和 Glibc-2.14.1，而且它们的头文件是确定无须修改的，所以不再需要 fixincludes 脚本。实际上，运行该脚本会将宿主系统的修正头文件安装到 GCC 的私有 include 目录，这会导致搭建环境的污染。键入下面的命令，消除运行 fixincludes 脚本造成的影响：

```
cp -v gcc/Makefile.in{,.orig}
sed 's@\./fixinc\.sh@-c true@' gcc/Makefile.in.orig > \
gcc/Makefile.in
```

对于 x86 机器，使用参数-fomit-frame-pointer 使 GCC 进行引导式编译。默认情况下忽略该参数进行非引导编译，而目标是产生一个和引导编译完全一样的编译器。使用 **sed** 命令强制使用该参数：

```
cp -v gcc/Makefile.in{,.tmp}
sed 's/^T_CFLAGS =$/& -fomit-frame-pointer/' \
gcc/Makefile.in.tmp > gcc/Makefile.in
```

键入下面的命令，将 GCC 的默认动态链接器的位置改为指向/tools。同时从 GCC 的 include 的搜索路径里移除/usr/include。现在完成这些工作而不是等到安装完成后去修改 specs 文件的好处是，在实际编译 GCC 时保证使用的是新动态链接器。换句话说，在编译过程中创建的所有二进制文件将会链接到新的 Glibc。

```
for file in \
$(find gcc/config -name linux64.h -o -name linux.h \
-o -name sysv4.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_INCLUDE_DIR
#define STANDARD_INCLUDE_DIR 0
#define STANDARD_STARTFILE_PREFIX_1 ""
#define STANDARD_STARTFILE_PREFIX_2 ""' >> $file
touch $file.orig
done
```

如果上面的命令看起来很难理解，那么我们就逐句解释一下。首先，找到 gcc/config 目录下名为 linux.h，linux64，或者 sysv4.h 的所有文件。对于找到的每一个文件，我们将其复制为新文件并添加".orig"后缀。第一个 sed 表达式为每个"/lib/ld"，"/lib64/ld"或"/lib32/ld"添加前缀"/tools"，第二个 sed 表达式将硬编码的"/usr/"修改为"/tools"。然后添加宏定义声明，这些宏会修改 include 的搜索路径和默认的头文件前缀到文件尾部。最后，使用 **touch** 命令更新新文件的时间戳。使用 cp –u 命令，防止因为不经意间运行两次 cp 命令而对原文件造成意想不到的改变。

在 x86_64 位机上，取消 GCC 的多库 spec，防止 GCC 尝试链接到宿主的库：

```
case $(uname -m) in
   x86_64)
   for file in $(find gcc/config -name t-linux64) ; do \
      cp -v $file{,.orig}
      sed '/MULTILIB_OSDIRNAMES/d' $file.orig > $file
   done
   ;;
esac
```

和第一次编译 GCC 一样，需要 GMP，MPFR 和 MPC 包。解压这几个压缩包，然后将其移到要求的目录名下：

```
tar -jxf ../mpfr-3.1.0.tar.bz2
mv -v mpfr-3.1.0 mpfr
tar -jxf ../gmp-5.0.2.tar.bz2
mv -v gmp-5.0.2 gmp
tar -zxf ../mpc-0.9.tar.gz
mv -v mpc-0.9 mpc
```

创建分离的编译目录：

```
mkdir -v ../gcc-build
cd ../gcc-build
```

在开始编译 GCC 之前，请记得清除所有覆盖默认优化标记的环境变量。

现在开始准备编译 GCC：

新的配置选项的含义如下：

--enable-clocale=gnu

该选项保证在任何情况下，都为 C++库选择了正确的语言环境。如果配置脚本发现安装了 de_DE 语言环境，就会选择正确的 gnu 语言环境模块。然而，如果没有安装 de_DE 环境，就可能错误地选择通用语言环境因而导致搭建 C++库不兼容的二进制应用程序接口（ABI）。

--enable-__cxa_atexit

该选项表示使用__cxa_atexit 而不是 atexit 来为本地静态和全局对象注册 C++析构函数。对于完全标准兼容的析构函数处理这事非常必要的。它也会影响 C++的 ABI，因而影响 C++共享库和 C++程序和其他 Linux 版本的互操作性。

--enable-languages=c, c++

该选项表示支持 C 和 C++编译器。

--disable-libstdcxx-pch

禁止为 libstdc++编译预编译头文件（PCH），它会占据大量的空间但我们却没有用到它。

--disable-bootstrap

对于本地编译 GCC，默认是进行“bootstrap”编译。Bootstrap 会多次编译 GCC。它使用自己第一次编译出的程序对自己进行第二次编译，然后进行第三次。然后会比较第二次和第三次，确保重新编译的结果的正确性。但是，LFS 搭建方法应该提供一个固定的编译器，所以不需要 bootstrap 编译。

```
CC="$LFS_TGT-gcc -B/tools/lib/" \
  AR=$LFS_TGT-ar RANLIB=$LFS_TGT-ranlib \
  ../gcc-4.6.1/configure --prefix=/tools \
  --with-local-prefix=/tools --enable-clocale=gnu \
  --enable-shared --enable-threads=posix \
  --enable-__cxa_atexit --enable-languages=c,c++ \
  --disable-libstdcxx-pch --disable-multilib \
  --disable-bootstrap --disable-libgomp \
  --without-ppl --without-cloog \
  --with-mpfr-include=$(pwd)/../gcc-4.6.1/mpfr/src \
  --with-mpfr-lib=$(pwd)/mpfr/src/.libs
```

编译软件：

```
make
```

安装软件：

```
make install
```

作为一个结束操作，创建一个符号链接。由于许多 UNIX 系统上并没有安装 GNU C 编译器，而且许多程序和脚本运行 cc 而不是 gcc，为了在所有 UNIX 系统保持程序的通用性和可用性，运行 cc 命令使系统管理员可以自由决定使用哪个 C 编译器：

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

Caution

现在，迫切需要停下来检查新工具链的基本功能（编译器和链接器）是否如我们期望的正常工作。为进行正确性检查，运行下面的命令：

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

如果一切正常，没有错误输出，而且最后一条命令的输出为如下形式：

```
[Requesting program interpreter:
/tools/lib/ld-linux.so.2]
```

注意动态链接器的前缀是/tools/lib，64 位机上是/tools/lib64。

如果输出和上面的不同或者根本没有输出，那么肯定是哪里出错了。重新检查前面的步骤找出问题所在并改正。必须在继续下去前解决问题。首先，再次执行正确性检查，使用 gcc 命令而不是 cc。如果这样有效，表示/tools/bin/cc 符号链接丢失。按照上面的指导建立该链接。然后，确保 PATH 路径正确。这可以运行 echo $PATH 命令检查，验证输出列表的头部是/tools/bin。如果 PATH 错误，意味着你并不是以 lfs 用户登录，或者在 4.4 节“设置环境”中有错。

若一切正常，清除测试文件：

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

关于 GCC 包的详细信息请查阅 6.17.2 节“GCC 内容”。

5.11. Tcl-8.5.10

Tcl 软件包包含工具命令语言。

估计构建时间：0.3SBU

需要磁盘空间：33MB

5.11.1. 安装 Tcl

安装 Tcl 软件包和接下来的三个软件包（Expect，DejaGNU 和 Check）为运行 GCC 和 Binutils 等包的测试套件提供支持。仅仅为了测试目的而安装四个软件包看似有些多余，但实际上是为保证最重要的工具能够正常工作。尽管这些测试套件不在本章运行（不在本章强制要求），但在第 6 章必须要运行它们。

准备编译 Tcl：

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

编译软件：

```
make
```

编译结束。如之前讨论的那样，对于现在的临时工具链，我们并不强制要求运行测试套件。如果需要运行，键入下面的命令：

```
TZ=UTC make test
```

在某些宿主条件下，Tcl 测试套件可能会出错。因此，在这里测试失败是不值得惊讶的，这些错误并不关键。参数 TZ=UTC 设置时区为协调世界时间（UTC），也称为格林威治时间（GMT），但仅仅影响测试套件。它能保证时钟测试正确完成。关于 TZ 环境变量的详细信息在第 7 章提供。

安装软件：

```
make install
```

下面的命令使安装目录可写，便于以后清除调试信息：

```
chomd -v u+w /tools/lib/libtcl8.5.so
```

安装 Tcl 头文件。下一个要安装的软件包 Expect 在编译时需要它们。

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

创建一个符号链接：

```
ln -sv tclsh8.5 /tools/bin/tclsh
```

5.11.2. Tcl 包的内容

安装的程序： tclsh(link to tclsh8.5)和 tclsh8.5

安装的库： libtcl8.5, libtclstub8.5.a

简述

tclsh8.5 Tcl 命令 shell

tclsh 指向 tclsh8.5 的链接

libtcl8.5.so Tcl 库
libtclstub8.5.a Tcl 的 Stub 库

5.12. Expect-5.45

Expect 软件包处理和其他交互性程序的脚本化的对话。

估计搭建时间： 0.1SBU

需要磁盘空间： 4.1MB

5.12.1.安装 Expect

首先，强制 Expect 配置脚本使用/bin/stty，避免从宿主系统上寻找/usr/local/bin/stty。这就保证我们的测试套件对于最后构建的工具链保持正确：

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

配置 Expect 脚本：

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

配置参数的含义：

--with-tcl=/tools/lib

使配置脚本到临时工具所在位置寻找 Tcl，避免到宿主系统找到可能存在的 Tcl。

--with-tclinclude=/tools/include

显式地让 Expect 到/tools/include 目录下寻找 Tcl 头文件。由于 Expect 不能自动找到 Tcl 头文件，使用该参数避免 configure 脚本出错导致配置失败。

编译软件：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make test
```

在某些宿主条件下，Expect 测试套件可能会出错。因此，在这里测试失败是不值得惊讶的，这些错误并不关键。

安装软件：

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

make 命令的参数的含义：

SCRIPTS=""

禁止安装 Expect 不必要的附加脚本。

5.12.2. Expect 的内容

安装的程序： expect

安装的库： libexpect-5.45.a

简述

expect 按照脚本和其他交互式程序通信

libexpect-5.45.a 包含一些库函数，他们允许 Expect 作为 Tcl 扩展部分使用，或者直接从 C 或 C++调用（不需经过 Tcl）

5.13. DejaGNU-1.5

DejaGNU 提供其他程序的一个测试框架。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 6.1MB

5.13.1. 安装 DejaGNU

配置 DejaGNU：

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

编译安装该软件：

```
make install
```

测试安装结果，键入：

```
make check
```

5.13.2.DejaGNU 内容

安装的程序： runtest

简述

runtest　runtest 是一个包装脚本，用于定位正确的 expect shell 并运行 DejaGNU

5.14. Check-0.9.8

Check 是 C 语言的一个单元测试框架。

估计搭建时间： 0.1SBU

需要磁盘空间： 4.8MB

5.14.1. 安装 Check

配置 Check：

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

编译：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

注意，Check 的测试套件大概需要较长（长达 4SBU）时间。

安装软件包：

```
make install
```

5.14.2. Check 的内容

安装的库： libcheck.{a,so}

简述

libcheck.{a,so} 该函数库允许从测试程序调用 Check

5.15. Ncurses-5.9

Ncurses 的库对字符界面独立终端的处理提供了支持。

估计搭建时间： 0.7SBU

需要磁盘空间： 30MB

5.15.1. 安装 Ncurses

配置 Ncurses：

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

配置参数的含义：

--without-ada

该参数告诉 Ncurses 即使宿主系统存在 Ada 编译器也不要为编译 Ada 支持的库，因为在我们进入 chroot 环境后没有 Ada 编译器。

--enable-overwrite

该参数告诉 Ncurses 将头文件安装到/tools/include 中而不是/tools/include/ncurses，以便于其他程序能够找到 Ncurses 头文件。

编译软件：

```
make
```

该软件包含有一个测试套件，但只能在软件在安装后才可以运行。该套件位于 test/目录。查看其中的 README 文件获取详细信息。

安装软件：

```
make install
```

该软件包的详细信息位于 6.20.2 节“Ncurses 内容”。

5.16. Bash-4.2

Bash 软件包含了 Bourne-Again Shell。

估计搭建时间： 0.5SBU

需要磁盘空间： 35MB

5.16.1 安装 Bash

首先，应用下面的补丁修复已知的一些 bug：

```
patch -Np1 -i ../bash-4.2-fixes-3.patch
```

准备编译：

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

配置选项的含义如下：

--without-bash-malloc

该参数使得 Bash 不编译产生内存分配函数 malloc，该 malloc 函数会引发分段错误。禁用 bash 的 malloc 后，Bash 使用 Glibc 的 malloc 函数，它更为稳定。

编译：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make tests
```

安装：

```
make install
```

许多程序使用 sh 调用 shell，故添加一个链接：

```
ln –vs bash /tools/bin/sh
```

关于该软件包的详细信息位于 6.30.2 节“Bash 包含的内容”。

5.17. Bzip2-1.0.6

Bzip2 软件包含了用于压缩和解压缩的程序。使用 bzip2 压缩文本文件比传统的 gzip 压缩的压缩率要高得多。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 4.8MB

5.17.1 安装 Bzip2

Bzip2 不包含 configure 脚本。直接使用 make 命令编译：

```
make
```

安装软件：

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

Bzip2 的详细信息位于 6.19.2 节“Bzip2 包含的内容”。

5.18. Coreutils-8.14

Coreutils 包含了显示和设置基本系统特性的工具集。

估计搭建时间： 0.7SBU

需要磁盘空间： 88MB

5.18.1 安装 Coreutils

配置 Coreutils：

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

配置参数的含义：

--enable-install-program=hostname

该参数允许 hostname 编译安装（默认情况下是不编译它的），Perl 的测试套件会用到它。

编译：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make RUN_EXPENSIVE_TESTS=yes check
```

RUN_EXPENSIVE_TESTS=yes 参数使测试套件运行一些额外的测试，考虑到 CPU 能耗和内存占用，这些测试在某些平台上代价较高，但通常在 Linux 系统上是可以接受的。

安装软件包：

```
make install
```

上面的命令不会安装 su 程序，因为 su 程序不能被安装为 setuid 的 root 用户是非特权用户。所以，我们手动将其安装且重命名，这样就可以在最终系统上以非特权用户运行测试，而且我们保持了来自宿主机的可用的 su。安装：

```
cp -v src/su /tools/bin/su-tools
```

关于软件包的详细信息位于 6.23.2 节“Coreutils 的内容”。

5.19. Diffutils-3.2

Diffutils 包含了用于显示文件或目录之间的区别的程序。

估计搭建时间： 0.1SBU

需要磁盘空间： 6.1MB

5.19.1 安装 Diffutils

配置：

```
./configure –prefix=/tools
```

编译：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

安装：

```
make install
```

该软件包的详细信息位于 6.37.2 节“Diffutils 的内容”。

5.20. File-5.09

File 软件包含判断文件类型的工具。

估计搭建时间： 0.2SBU

需要磁盘空间： 9.5MB

5.20.1 安装 File

配置：

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

编译：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

安装：

```
make install
```

该软件包的详细信息位于 6.12.2 节“File 的内容”。

5.21. Findutils-4.4.2

Findutils 软件包含了查找文件的程序。该程序允许沿文件树的递归查找，还能创建，管理和搜索一个文件数据库（通常比递归查找快，但当数据库变动时查找不可靠）。

估计搭建时间： 0.3SBU

需要磁盘空间： 20MB

5.21.1 安装 Findutils

配置：

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

编译：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

安装：

```
make install
```

软件包的详细信息位于 6.39.2 节“Findutils 的内容”。

5.22. Gawk-4.0.0

Gawk 软件包含了操作文本文档的程序。

估计搭建时间： 0.2SBU

需要磁盘空间： 28MB

5.22.1. 安装 Gawk

配置软件：

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

编译软件：

```
make
```

```
make check
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

安装：

```
make install
```

软件包的详细信息位于 6.38.2 节“Gawk 的内容”。

5.23. Gettext-0.18.1.1

Gettext 软件包含了国际化和本地化的工具。这些工具支持程序编译为支持本地语言支持（NLS, Native Language Support），使得这些程序的输出信息具有本地语言的格式。

估计搭建时间： 0.8SBU

需要磁盘空间： 82MB

5.23.1. 安装 Gettext

考虑到我们是为临时工具链而编译的，所以现在仅需要编译安装 Gettext 的一个二进制文件。

配置 Gettext：

```
cd gettext-tools
./configure --prefix=/tools --disable-shared
```

配置参数的含义：

--disable-shared

现在我们不需要 Gettext 的共享库，所以没有必要编译他们。

编译：

```
make -C gnulib-lib
make -C src msgfmt
```

因为现在仅仅编译了一个二进制文件，所以不能运行测试套件。即，现在强烈不推荐尝试去运行测试套件。

安装 msgfmt 二进制文件：

```
cp -v src/msgfmt /tools/bin
```

关于该软件包的详细信息位于 6.41.2 节“Gettext 的内容”。

5.24. Grep-2.9

Grep 包含了用于文件内部搜索的程序。

估计搭建时间： 0.2SBU

需要磁盘空间： 18MB

5.24.1. 安装 Grep

配置：

```
./configure --prefix=/tools \
--disable-perl-regexp
```

配置参数的含义为：

--disable-perl-regexp

该参数保证 **grep** 程序不会链接到 Perl 兼容的正则表达式库，该库可能存在于宿主系统，然而当我们进入 chroot 环境后是不能使用这个库的。

编译软件包：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

安装：

```
make install
```

关于该软件的详细信息位于 6.28.2 节“Grep 的内容”。

5.25. Gzip-1.4

Gzip 包含了压缩和解压缩文件的程序。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 3.3MB

5.25.1. 安装 Gzip

配置：

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

编译：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

安装软件：

```
make install
```

关于该软件的详细信息位于 6.44.2 节“Gzip 的内容”。

5.26. M4-1.4.16

M4 软件包含了一个宏处理器。

估计搭建时间： 0.2SBU

需要磁盘空间： 11.6MB

5.26.1. 安装 M4

配置：

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

编译：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

安装软件：

```
make install
```

关于该软件的详细信息位于 6.25.2 节“M4 的内容”。

5.27. Make-3.82

Make 程序用于编译软件。

估计搭建时间： 0.1SBU

需要磁盘空间： 9.6MB

5.27.1. 安装 Make

配置：

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

编译软件：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

安装：

```
make install
```

关于该软件的详细信息位于 6.49.2 节“Make 的内容”。

5.28. Patch-2.6.1

Patch 软件利用 **diff** 程序创建的“patch”文件来修改文件。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 1.9MB

5.28.1. 安装 Patch

配置：

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

编译：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

安装软件：

```
make install
```

关于该软件的详细信息位于 6.53.2 节“Patch 的内容”。

5.29. Perl-5.14.2

Perl 程序包含了抽取测试和报告语言。

估计搭建时间： 1.8SBU

需要磁盘空间： 223MB

5.29.1. 安装 Perl

首先，应用下面的补丁修改硬性链接到 C 库的路径：

```
patch -Np1 -i ../perl-5.14.2-libc-1.patch
```

配置 Perl：

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

编译：

```
make
```

尽管 Perl 自带了测试套件，但建议最好还是在下一章运行它。

现在，仅仅需要安装一部分工具和库：

```
cp -v perl cpan/podlators/pod2man /tools/bin
mkdir -pv /tools/lib/perl5/5.14.2
cp -Rv lib/* /tools/lib/perl5/5.14.2
```

关于该软件的详细信息位于 6.34.2 节“Perl 的内容”。

5.30. Sed-4.2.1

Sed 是一个流编辑器。

估计搭建时间： 1.8SBU

需要磁盘空间： 223MB

5.30.1. 安装 Sed

配置 Sed：

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

编译软件：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

安装软件包：

```
make install
```

关于该软件的详细信息位于 6.18.2 节“Sed 的内容”。

5.31. Tar-1.26

Tar 程序用于文件归档。

估计搭建时间： 0.3SBU

需要磁盘空间： 20.9MB

5.31.1. 安装 Tar

配置：

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

编译软件：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

安装：

```
make install
```

关于该软件的详细信息位于 6.58.2 节“Tar 的内容”。

5.32. Texinfo-4.13a

Texinfo 程序用于读，写以及转换 info 页面。

估计搭建时间： 0.3SBU

需要磁盘空间： 20.9MB

5.32.1. 安装 Texinfo

配置 Texinfo：

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

编译软件：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

安装软件：

```
make install
```

关于该软件的详细信息位于 6.59.2 节“Texinfo 的内容”。

5.33. Xz-5.0.3

Xz 程序用于压缩和解压缩文件。Xz 提供了对 lzma 和新的 xz 压缩格式的支持。用 xz 压缩文本文件能产生比 gzip 或 bzip2 更好的压缩比率。

估计搭建时间： 0.3SBU

需要磁盘空间： 14MB

5.33.1. 安装 Xz-Utils

配置 Xz：

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

编译：

```
make
```

现在编译结束。如同前面讨论的那样，现在不强制要求运行测试套件。若仍要运行，键入以下命令：

```
make check
```

安装软件：

```
make install
```

关于该软件的详细信息位于 6.50.2 节“Xz 的内容”。

5.34. 清理环境

首先要说明的是，本节的操作不是必须的。但是，如果你的 LFS 分区太小，那么清除一些不必要的东西是很有好处的。到目前为止，可执行文件和库文件大约包含了 70MB 左右不必要的调试符号。可以用下面的命令清除它们：

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

运行这些命令时会跳过一些文件，并报告不能识别这些文件的格式。这些文件大多数是脚本文件而不是二进制文件。

注意，不要对库文件使用*--strip-unneeded*，否则会损坏静态库文件，导致需要重头制作工具链。

如果需要节省更多空间，可以删除一些文档：

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

此时，在$LFS 上至少有 850MB 空闲空间，这些空间足够下一阶段安装 Glibc 了。如果足够编译安装 Glibc，那么也就有足够空间安装其他软件了。

5.35. 改变文件拥有主

Note

当以 root 用户而不是 lfs 用户登陆时，必须执行本书提醒部分中的命令。而且，反复确认在 root 用户环境中已设置了$LFS 变量。

现在，$LFS/tools 目录的拥有主是 lfs 用户，而 lfs 用户仅存在于宿主系统。如果不改变$LFS/tools 目录的拥有主，该目录下的文件会被表示为一个不存在的用户 ID。由于以后创建的用户可能拥有相同的 ID，导致该新建的用户错误地拥有了$LFS/tools 目录及其中的文件，由此可能导致恶意的文件操作，所以这是非常危险的。

为了避免这种情况发生，在新的 LFS 系统中创建/etc/passwd 文件时，我们需要添加 lfs 用户，在添加时注意使 lfs 用户的 ID 和组 ID 和宿主系统保持一致。当然，最好是将$LFS/tools 目录的拥有主改为 root 用户，键入下面的命令可达到目的：

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

尽管当 LFS 系统完成时就可以删除$LFS/tools 目录，但是还是可以保留它以便于搭建和本书同版本号的其他 LFS 系统。

Caution

如果你想要保持临时工具以便于将来搭建其他 LFS 系统，现在是最好的备份的时候了。第六章的命令会改变现在的工具，使得这些工具不再适合将来的搭建工作。

第三部分：搭建 LFS 系统

第六章. 安装基本系统软件

6.1. 概述

在本章中，我们开始正式搭建 LFS 系统。首先，我们 chroot 到临时的迷你 Linux 系统做一些最后的准备，然后开始安装软件。

Note

第六章中的 SBU 值和需要的磁盘空间包含了测试套件的数据。

这些软件的安装都是很简单的。尽管在大多数情况下安装指导会简短而类似，但是我们还是选择给出了完整的指导以便于尽可能地减少错误的发生。了解 Linux 系统怎样工作的关键就是要知道每一个软件是用来干什么的且为什么你（或系统）需要它！

我们并不推荐使用优化参数。尽管它能使程序运行得更快一点，但是它会增加编译的难度甚至引发运行时的问题。如果某个软件包因为使用了优化参数而停止编译，请尝试取消优化参数，然后再次编译看看问题是否仍然存在。即便使用优化参数可以通过编译，但由于在代码和编译工具间的复杂交互，还是有编译错误的风险。还需要注意的是如果-march 和 -mtune 参数使用非本书的其他值是不能保证正确性，它们是没有经过作者测试的。所以使用其他值可能导致工具链(Bintuils,GCC 和 Glibc)的错误。使用编译优化选项增加的潜在风险可能比我叙述的还要大。第一遍搭建 LFS 系统的读者最好不要使用自定义优化参数。这样编译出的系统同样会运行得很快且稳定。

软件包的安装顺序必须严格遵循本章中指定的顺序，以保证没有程序偶然依赖于一个指向/tools 的路径且硬链接到该路径。同理，不要并行编译分离的软件包。尽管并行编译可能会节省时间（尤其在双核的 CPU 中），但这却可能导致程序包含一条硬链接到/tools 的路径，如果以后移除该目录该程序就无法继续工作了。

在安装指导前，每个安装页都提供了关于该软件的详细信息，其中包括了该软件的内容，大约需要的编译时间，会占用多大的磁盘空间等明确的描述。按照这些安装指导信息，你会看到需要安装的程序和库（同样有简单的描述）的列表。

6.2. 准备虚拟内核文件系统

内核导出的不同文件系统的作用是和内核自身进行通信。这些文件系统是虚拟的，所以不占用磁盘空间。他们的内容都被放置在内存中。

为虚拟文件系统创建挂载点：

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

6.2.1.创建初始设备节点

当内核启动系统时，需要使用一些设备节点，尤其是 console 和 null 设备。这些设备节点必须已在硬盘上建立，保证 udevd 进程启动时它们已经可用，除此之外，Linux 用 init=/bin/bash 启动时也需要它们。键入下面的命令创建设备节点：

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

6.2.2. 挂载并填充/dev

为/dev 目录填充设备的推荐方法是将某虚拟文件系统（如 tmpfs）挂载到/dev 目录下，当虚拟文件系统检测到设备或访问设备时能动态创建这些设备。设备的创建通常是在系统启动时由 Udev 创建。因为我们的新系统还没有 Udev 而且也还没有启动，手动挂载并填充/dev 就显得很必要了。这通过绑定地挂载宿主系统的/dev 目录可以完成。绑定式的挂载是一类比较特殊的挂载方法，它为某个目录或挂载点创建了指向其他位置的镜像。键入下面的命令完成挂载填充/dev：

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

6.2.3. 挂载虚拟内核文件系统

现在挂载虚拟内核文件系统：

```
mount -vt devpts devpts $LFS/dev/pts
mount -vt tmpfs shm $LFS/dev/shm
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
```

6.3. 包管理器

将包管理器是添加到 LFS 书中是一个被频繁提及的请求。包管理器能够跟踪安装的文件，这样在移除包和升级包时就非常简单。对于二进制文件和库文件，包管理器也能记录其安装的配置文件。在你开始憧憬选用一个包管理器时，我们的回答是“不”，本节既不会讨论也不会推荐某个特定的包管理器。本节提供的是更流行的技术和它们怎么工作的一个综述。对你来说最好的包管理器可能存在于这些技术中或者是一或多个技术的结合。本节将简短提及升级包时将会发生的事情。

在 LFS 或 BLFS 中是没有包管理器的，下面是一些原因：

- 使用包管理器偏离了本书的目的——教你如何搭建一个 Linux 系统。
- 现在有很多包管理器，每个都有各自的优点和缺点。寻找一个满足所有人的包管理器是不可能的。

关于包管理器有很多提示，访问 *Hints Project* 查找是否有满足你需要的包管理器。

6.3.1. 升级软件遇到的问题

当软件发布新版本时，使用包管理器可以很容易就升级到新版本。通常，在 LFS 和 BLFS 书中的指导可用于升级到新版本。下面是你在升级软件时必须要意识到的问题，尤其时在运行的系统上升级时。

- 如果工具链（Glibc, GCC 或 Binutils）的某个软件包需要被升级到一个新的镜像版本，更安全的方式是重新搭建 LFS。尽管你也许能够按照软件所有的依赖顺序重建系统，但我们还是不建议你这样做。举个例子，如果 glibc-2.2.x 需要更新到 glibc-2.3x，你需要重新搭建系统。如果是小版本升级，简单地重新安装该软件即可，但这也不敢保证成功。例如，将 glibc 从 2.3.4 升级到 glibc-2.3.5 通常不会引发任何问题。
- 如果一个包含共享库的软件包被更新了，而且该库的名称也改变了，那么所有动态链接到该库的软件包都需要重新编译链接到这个新的库。（注意，包的版本和库的名称是不相关的）。例如，软件包 foo-1.2.3 可能安装了一个名为 libfoo.so.1 的库。你升级该软件包到 foo-1.2.4 后安装的库可能名为 libfoo.so.2。注意，你需要先重新编译所有依赖的包后再移除之前的库。

6.3.2. 包管理技术

下面是一些常见的包管理技术。在决定选择某个包管理器前，请先研究一下不同的技术，尤其是某个包管理架构的缺点。

6.3.2.1. 天才和白痴！

是的，这就是包管理技术。一些人并没有寻找包管理器的需求，因为他们非常了解软件包而且知道每个软件包安装了哪些文件。另外一些人也不需要包管理，他们正在计划重建整个系统，原因仅仅是某一个软件包改变了。

6.3.2.2. 在分离的目录安装软件

有一个简化的包管理方法，不需要任何额外的软件包管理软件的安装，将每个包在分离的目录里安装。举个例子，包 foo-1.1 安装在/usr/pkg/foo-1.1，然后创建一个符号链接从 /usr/pkg/foo 指向/usr/pkg/foo-1.1。当安装一个新版本 foo-1.2，则将其安装在/usr/pkg/foo-1.1 然后将符号链接替换为指向新版本的符号链接。

一些环境变量如 PATH, LD_LIBRARY_PATH, MANPATH, INFOPATH 和 CPPFLAGS 需要扩展为包含/usr/pkg/foo。但对于大量的包管理，这中架构就变得不可管理了。

6.3.2.3. 符号链接风格式的包管理

符号链接风格式的包管理是上节包管理技术的变体。每个包的安装都和上节的架构类似。不同是制作符号链接，每个文件都被链接到/usr 层次中。这就免去了扩展环境变量的需要。尽管用户可以用自动匹配创建符号链接，许多包管理都是用这种方式写的。这些流行的包管理器有 Stow, Epkg, Graft 和 Depot。

这种安装需要伪造一下，使得软件包好像是安装在/usr 下而实际上被安装到/usr/pkg 层次中。以这种方式安装通常并不是小事情。例如，若你安装软件包 libfoo-1.1。下面的指导可能无法正确地安装软件包：

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

这样安装时可以成功的，但依赖它的包可能不会像你期望的那样链接到 libfoo。如果你编译一个包要链接到 libfoo，你会发现它链接到/usr/pkg/libfoo/1.1/lib/libfoo.so.1 而不是/usr/lib/libfoo.so.1。正确的方法是使用 DESTDIR 策略来欺骗安装包。命令如下：

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

大多数软件包支持这种方法，但也有例外。对于那些不兼容的包，你要么手动安装这些包，要么将有问题的包安装到/opt 目录下。

6.3.2.4. 基于时间戳

在软件包安装前，每个文件都有一个时间戳。在安装完成后，简单地使用具有合适参数的 find 命令就能在时间戳文件创建后生成一个文件日志。使用这种技术的包管理器有 install-log。

尽管这种技术优点是简单，但它也具有两个缺点。如果在安装时文件的时间戳不是当前时间，那么这些文件就不能被包管理器跟踪到。所以，这种架构只能用于一个软件包能在同一时刻安装。如果两个软件包在两个不同的控制台安装，那么该技术也不可靠。

6.3.2.5. 跟踪安装脚本

跟踪安装脚本的方法是，安装脚本执行的命令会被记录下来。记录的方法有两种：

安装前设置环境变量 LD_PRELOAD 指向预加载的一个库。在安装时，该库会将自己附加不同的可执行文件如 cp， install， mv，而且追踪那么会修改文件系统的系统调用，达到跟踪安装包的目的。若要这种技术真正有效，所有的可执行文件都需要被动态链接且要去除 suid 和 sgid 位。在安装时预加载库可能会导致不希望的影响。因而，强烈建议你做一些测试确保包管理器不会破坏任何文件而且能将所有需要的文件录入日志。

第二种技术是使用 strace 命令，它能记录安装脚本执行期间所有的系统调用。

6.3.2.6. 创建软件包归档

使用这种技术，软件包被欺骗安装到一个分离的树中，就像符号链接风格式包管理一样。

安装完成后，使用安装的文件创建一个包的归档。在以后该归档可以用来在本地机器上安装软件包也可以在别的机器上安装软件包。

目前大多数商业linux发行版都使用了这种技术。例如RPM包管理器（顺便说一下，这是Linux基本标准规范的要求），pkg-utils，Debian的apt和Gentoo的Portage系统。关于怎样在LFS系统中适应这种风格的包管理器的指导在*http://www.linuxfromscratch.org/hints/downloads/files/fakeroot.txt*。

创建包含了依赖关系的包文件太过于复杂，它超出了 LFS 讨论的范畴。

Slackware 系统使用基于 tar 的系统进行包归档。该系统故意不像其他复杂包管理器那样处理包依赖关系。关于 Slackware 包管理器的详细信息在 *http://www.slackbook.org/html/package-management.html*。

6.3.2.7. 基于用户的包管理

该架构是 LFS 所特有的，它是 Matthias Benkmann 设计的，可以从 Hints Project 获取。该架构中，每个包都被分离的用户安装在标准的位置。属于某个包的文件可以简单地用用户 ID 来标识。该架构的特性和缺点太复杂了，所有本节中我们不打算讨论。详细信息请查阅 *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 系统的优点之一是没有文件依赖于其在磁盘上的位置。如果要克隆一个 LFS 到另一台计算机并且保持它的构架基本不变，就像在包含根目录（基本 LFS 上未压缩的大小约 250MB）的 LFS 分区上使用 tar 命令一样简单，你可以通过网络传输或 CD-ROM 复制文件到新系统然后释放它。从这点来说，一些配置文件会发生改变。需要更新的配置文件包括：/etc/hosts, /etc/fatab, /etc/passwd, /etc/group, /etc/shadow, /etc/ld,so.conf, /etc/scsi_id.config, /etc/sysconfig/network 和/etc/sysconfig/network-devices/ifconfig.eth0/ipv4。

由于新系统的硬件区别和原始的内核配置的不同，自定义的内核可能需要为新系统重新编译。

最后，新系统需要可启动，参阅 8.4 节“使用 GRUB 配置系统启动进程”。

6.4. 进入 Chroot 环境

现在是时候进入 chroot 环境开始搭建最后的 LFS 系统了。作为 root 用户，运行下面的命令进入该环境，此时该环境仅仅有一条临时工具链：

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

带-i 参数的 env 命令会清除 chroot 环境的所有环境变量。之后，仅仅设置了 HOME, TERM, PS1 和 PATH 变量。TERM=”$TERM”设置 chroot 环境内的 TERM 变量和外部环境的 TERM 的值一致。诸如 vim 和 less 程序会使用该变量才能正常工作。如果还需要其他变量，比如 CFLAGS 或 CXXFLAGS，现在也是设置它们的时候。

从现在起，没有必要再使用 LFS 变量，因为所有的工作都天然被限制在 LFS 文件系统内。这是因为 Bash Shell 现在将$LFS 视为根（/）目录。

注意，/tools/bin 位于 PATH 的最后。这意味着一旦临时工具的最终版本安装好后临时工具就不再被用到了。这只有 shell 不记录可执行二进制文件的位置才可能发生——由此，给 bash 传递+h 参数关闭 bash 的 hash 查找。

另外，bash 会提示“I have no name! “，这是正常的，因为我们还没有创建/etc/passwd 文件。

Note

需要着重强调的是，本章和接下来章节的命令都必须在 chroot 后的环境中执行。如果因为某种原因离开了该环境（比如重启电脑），请确保按照 6.2.2 节“挂载并填充/dev”挂载上虚拟内核文件系统，同时按 6.2.3 节“挂载虚拟内核文件系统”，然后再次 chroot，最后才能继续进行安装。

6.5. 创建文件夹

是时候在 LFS 文件系统中创建一些结构了。键入下列命令创建标准目录树：

```
mkdir -pv \
/{bin,boot,etc/{opt,sysconfig},home,lib,mnt,opt,run}
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/{doc,info,locale,man}
mkdir -v /usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -pv /usr/{,local/}share/man/man{1..8}
for dir in /usr /usr/local; do
   ln -sv share/{man,doc,info} $dir
done
case $(uname -m) in
   x86_64) ln -sv lib /lib64 && ln -sv lib /usr/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/{misc,locate},local}
```

文件夹创建后的默认权限是 755，但我们并不希望所有文件夹权限都是 755。在上面的命令中有两个不同——root 用户的 home 目录和临时文件目录。

第一个模式改变保证不是任何用户都能进入/root 目录——和普通用户的家目录一样。第二个模式改变保证任何用户都可以向/tmp 目录和/var/tmp 目录执行写操作，但是却不能删除别的用户在其中创建的文件。后者是用所谓的“粘连位”（在 1777 位屏蔽码的最高位）实现的。

6.5.1. FHS 兼容

目录树是基于文件系统标准层次规范（Filesystem Hierarchy Standard）（查看 *http://www.pathname.com/fhs/*）创建的。除了FHS，我们为man，doc，和info目录创建兼容的符号链接，因为许多包仍然会安装他们的文档到/usr/<directory>或/usr/local/<directory>而不是/usr/share/<directory>或/usr/local/share/<directory>。FHS还规定了/usr/local/games和/usr/share/games的存在。对于/usr/local/share的子目录结构FHS规定得并不精确，所有我们仅创建必须的目录。然而，若想严格按照FHS则去创建其他文件夹。

6.6. 创建必要的文件和符号链接

一些程序使用硬链接的路径链接到一些已不存在的程序。为了满足那些程序，创建一些符号链接，这些符号链接会在软件被安装好后被真实的文件代替：

```
ln -sv /tools/bin/{bash,cat,echo,pwd,stty} /bin
ln -sv /tools/bin/perl /usr/bin
ln -sv /tools/lib/libgcc_s.so{,.1} /usr/lib
ln -sv /tools/lib/libstdc++.so{,.6} /usr/lib
ln -sv bash /bin/sh
```

通常的Linux系统维持了一个所有已挂载文件系统的列表，该表存在/etc/mtab文件中。我们可以在挂载新文件系统时创建该文件。虽然我们不会挂载任何文件系统到chroot环境，但有的工具需要/etc/mtab存在，故而就创建一个空的文件：

要使 root 用户能够登陆系统且名称“root”能被识别，必须在/etc/passwd 和/etc/group 文件中存在相应的项。

创建/etc/passwd 文件：

root 用户的实际密码将在随后设置（“x”仅仅是作占位符）。

创建/etc/group 文件：

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
tty:x:4:
tape: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:
mail:x:34:
nogroup:x:99:
EOF
```

这些创建的组并不是标准的，它们是本章中 Udev 配置要求的部分需要，也是现今 Linux 发行版通用的公约。Linux 基础标准（LSB，位于 *http://www.linuxbase.org*）仅仅建议组 ID（GID）为 0 的 root 组和组 ID 为 1 的 bin 组。除此之外，系统管理员可以自由选择其他组名和组 ID，这是因为编写良好的程序并不依赖于 GID 号，而是使用组名。

要清除“I have no name!”提示，就需要启动一个新 shell。由于在第 5 章中已经安装好了完整的 Glibc，而且/etc/passwd 和/etc/group 文件也建立好了，用户和用户组名的解析可以生效了：

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

注意这里使用了+h 指令，它使得 bash 不使用内部的 hash 路径查找功能。如果不使用

该指令，bash 会记忆它运行过的二进制文件的路径，而我们希望新编译安装好了的程序可以立即被使用，所以本章中都使用了+h 指令。

login, agetty 和 init（还有其他一些程序）程序使用了大量日志文件记录登陆信息，如谁登陆了系统及何时登陆。但是，若日志文件不存在的话，这些程序就不会写日志文件。下面就创建这些日志文件并赋予相应的权限：

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

/var/run/utmp 文件记录当前登陆的用户。/var/log/wtmp 文件记录所有的登陆和登出。/var/log/lastlog 文件记录最后登陆的时间。/var/log/btmp 文件记录失败的登陆尝试。

6.7. Linux-3.1 API Headers

Linux API 头文件（在 linux-3.1.tar.gz 文件中）暴露了内核的 API 供 Glibc 调用。

估计搭建时间： 0.1SBU

需要磁盘空间： 515MB

6.7.1. 安装 Linux API 头文件

Linux 内核必须暴露应用程序编程接口（Application Programming Interface，API）供系统 C 库（LFS 中即 Glibc）使用。所以，我们要从 Linux 内核源代码文件中提取不同的 C 头文件。

清除之前可能产生的旧的文件或依赖：

```
make mrproper
```

现在开始测试并从源代码释放用户可见的内核头文件。它们被放置在一个中间的本地目录，再被复制到需要的位置，这样做是因为释放过程会删除目标目录的所有其他文件。另外，还有一些内核开发者用到的隐藏文件，但 LFS 是不会用到的，所以也在中间目录中将其删除。

```
make headers_check
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 头文件的内容

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

安装的目录： /usr/include/asm, /usr/include/asm-generic, /usr/include/drm, /usr/include/linux, /usr/include/mtd, /usr/include/rdma, /usr/include/scsi, /usr/include/sound, /usr/include/video, /usr/include/xen

简要描述：

/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 Sound 头文件

/usr/include/video/*.h Linux API Video 头文件

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

6.8. Man-pages-3.35

Man-pages 软件包包含了超过 1900 页 man 页面。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 21MB

6.8.1. 安装 Man-pages

安装 Man-pages:

```
make install
```

6.8.2. Man-pages 的内容

安装的文件： 大量不同的 man 页面

简要描述：

man pages 描述 C 语言功能，重要设备文件和有重要意义的配置文件。

6.9. Glibc-2.14.1

Glibc 包含了主要的 C 库。该库提供了一些基本例程：分配内存，搜索目录，打开和关闭文件，读写文件，操作字符串，模式匹配，算术运算等等。

估计搭建时间：14.2SBU

需要磁盘空间： 856MB

6.9.1. 安装 Glibc

LFS 之外的某些包建议安装 GNU libiconv 将数据从一种编码翻译成另一种。该工程的主页(*http://www.gnu.org/software/libiconv/*)告诉我们：“该库为 iconv()提供了实现，它可以被用在没有其实现的系统上，或者其实现无法完成 Unicode 的转换。”而 Glibc 库提供了 iconv（）的实现，而且可以完成 Unicode 的转换，故而在 LFS 系统上无须安装 libiconv。

由于 Glibc 是自满足的系统，所以尽管编译器的 specs 文件和连接器此刻仍然指向/toos 目录，但还是不妨碍 Glibc 的安装。在安装好 Glibc 之前，specs 文件和连接器是不能被调整的，这是因为 Glibc 的自动配置测试会得出错误结果从而无法搭建一个干净系统。

当运行 make install 时，脚本 test-installation.pl 会为刚安装的 Glibc 执行一个小的正确性测试。然而，因为我们的工具链仍指向/tools 目录，该测试的目标将是错误的 Glibc。那么，我们需要强制该脚本检查我们刚安装的 Glibc：

```
DL=$(readelf -l /bin/sh | sed -n \
   's@.*interpret.*/tools\(.*\)]$@\1@p')
sed -i "s|libs -o|libs -L/usr/lib -Wl,\
   -dynamic-linker=$DL -o|"  scripts/test-installation.pl
unset DL
```

此外，在 test-installation.pl 脚本中存在一个 bug，该 bug 在执行 make install 时会链接一个未安装的测试程序到函数库，我们利用 sed 命令来修正它：

```
sed -i -e 's/"db1"/& \&\& $name ne "nss_test1"/' \
   scripts/test-installation.pl
```

shell 脚本 ldd 包含了特定 Bash 的语法。将它的默认程序翻译器改为/bin/bash，以防安装了 BLFS 书中 Shells 一章中的/bin/sh：

```
sed -i 's|@BASH@|/bin/bash|' elf/ldd.bash.in
```

修正 Glibc 的一对 bug，他们会导致崩溃和核心泄漏：

```
patch -Np1 -i ../glibc-2.14.1-fixes-1.patch
```

修复 Glibc 阻碍安装 GCC-4.6.1 的 bug：

```
patch -Np1 -i ../glibc-2.14.1-gcc_fix-1.patch
```

修复在某些条件下会发生的栈失衡：

```
sed -i '195,213 s/PRIVATE_FUTEX/FUTEX_CLOCK_REALTIME/' \
nptl/sysdeps/unix/sysv/linux/x86_64/pthread_rwlock_timed{rd,
wr}lock.S
```

根据 Glibc 文档推荐，最好在 Glibc 的源目录之外的专用目录进行编译：

```
mkdir -v ../glibc-build
cd ../glibc-build
```

和第 5 章一样，对于 x86 机器要添加必须的编译标志到 CFLAGS。这里也设置了 gcc 编译器的库优化参数，以提高编译速度(-pipe)和包性能(-O3)。

```
case `uname -m` in
   i?86)  echo "CFLAGS += -march=i486 -mtune=native -O3 -pipe" >
configparms ;;
esac
```

配置 Glibc：

```
../glibc-2.14.1/configure --prefix=/usr \
   --disable-profile --enable-add-ons \
   --enable-kernel=2.6.25 --libexecdir=/usr/lib/glibc
```

新配置参数的含义：

--libexecdir=/usr/lib/glibc

该选项将 pt_chown 程序的位置从默认的/usr/libexec 改为/usr/lib/glibc。

编译 Glibc：

```
make
```

Important

在本节，Glibc 的测试套件是非常重要的，切记不可跳过。

在运行测试之前，从源码树复制一个文件到编译目标树，这是为了防止一些测试错误。最后开始测试：

```
cp -v ../glibc-2.14.1/iconvdata/gconv-modules iconvdata
make -k check 2>&1 | tee glibc-check-log
grep Error glibc-check-log
```

在 posix/annexc 测试中，你很可能会看到预料到的（被忽略）的错误。此外，Glibc 测试套件可能会依赖宿主系统。下面是最常发生的错误列表：

- nptl/tst-clock2，nptl/tst-attr3 和 rt/tst-cpuclock2 已知会失败。失败原因还不完全清楚，但推测是由于镜像时序触发的失败。
- 如果你的系统 CPU 不是相关的新 genuine Intel 或算数 AMD 处理器，那么某些数学测试也可能失败。
- 如果你以 noatime 选项挂载 LFS 分区，那么 atime 测试会失败。如 2.4 节提到的：“挂载新分区”，在搭建 LFS 时不要使用 noatime 选项。

- 当运行在较老或较旧的硬件或系统过载时，一些测试由于响应超时也可能失败。修改 make check 命令，设置 TIMEOUTFACTOR 来清除这些错误（例如：TIMEOUTFACTOR=16 make –k check）。
- 其他已知的错误可能发生在这些架构上：posix/bug-regex32，misc/tst-writev，elf/check-textrel，nptl/tst-getpid2 和 stdio-common/bug22。

安装 Glibc 时会报告缺失/etc/ld.so.conf 文件，但这是无害的信息。下面的命令可阻止该警告：

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

安装软件包：

```
make install
```

上面的命令并没有安装语言环境，故系统不能在不同的语言环境下响应。这些语言环境都不是必要的，但如果缺失了一部分，将来测试套件可能会跳过一些重要的测试用例。

各语言环境可用 localedef 程序安装，即下面第一条 localedef 命令和 /usr/share/i18n/locales/cs_CZ 独立字符集语言环境定义，/usr/share/i18n/charmaps/UTF-8.gz 字符映射定义和附加到/usr/lib/locale/locale-archive 文件的结果。下面的指示会安装最少的语言环境集达到对测试的最佳覆盖：

```
mkdir -pv /usr/lib/locale
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 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 ja_JP -f EUC-JP ja_JP
localedef -i tr_TR -f UTF-8 tr_TR.UTF-8
localedef -i zh_CN -f GB18030 zh_CN.GB18030
```

此外，安装你的语言环境，语言和字符集。

另一种方法是，一次性安装 glibc-2.14.1/localedata/SUPPORTED 文件列出的所有语言环境，但该安装命令会非常耗时：

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

在最不可能的情况下，使用 localedef 命令创建安装 glibc-2.14.1/localedata/SUPPORTED 文件中没有列出的语言环境。

6.9.2. 配置 Glibc

我们需要创建/etc/nsswitch.conf 文件，尽管当该文件丢失或损坏时 Glibc 会提供一个默认文件，但是该默认文件在网络环境下工作不佳。另外，我们还需要配置时区。

运行命令创建新的/etc/nsswitch.conf 文件：

```
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
```

判断当地时区的方法，运行下面的脚本：

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

```
tzselect
```

在回答了一些关于当地的信息后，该脚本会输出时区名称（例如：America/Edmonton）。其它可能的时区列在/usr/share/zoneinfo 中，比如 Canada/Eastern 或 EST5EDT（该脚本不能识别但仍可以使用）。

然后创建/etc/localtime 文件：

```
cp -v --remove-destination /usr/share/zoneinfo/<xxx> \
   /etc/localtime
```

用选择的时区（如 Canada/Eastern）代替<xxx>。

cp 命令参数的含义：

--remove-destination

该参数强制移除已存在的符号链接。使用复制而不使用符号链接的原因是包含/usr 在一个分离的分区的情形。当 boot 到单用户模式时这是非常重要的。

6.9.3. 配置动态加载器

默认情况下，动态加载器（/lib/ld-linux.so.2）在/lib 和/usr/lib 下搜索动态库，在程序运行时需要使用这些库。然而，如果库文件不再/lib 或/usr/lib 下，这些库文件就需要被添加到/etc/ld.so.conf 文件供动态加载器查找。两个常见的包含额外库文件的目录是/usr/local/lib 和

/opt/lib，所以要将它们添加进动态加载器的搜索路径。

创建新的/etc/ld.so.conf 文件：

如果需要，动态加载器也可以搜索目录及该目录下文件的内容。通常，include 目录下的文件作为一行指定要求的库路径。运行命令添加该功能：

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

6.9.4. Glibc 的内容

安装的程序：catchsegv, gencat, getconf, getent, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale,localedef, mtrace, nscd, pcprofiledump, pt_chown, rpcgen, rpcinfo, sln, sprof, tzselect,xtrace, zdump, zic

安装的库文件：ld.so, libBrokenLocale.{a,so}, libSegFault.so, libanl.{a,so}, libbsd-compat.a,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,libpcprofile.so, libpthread.{a,so}, libpthread_nonshared.a, libresolv.{a,so},librpcsvc.a, librt.{a,so}, libthread_db.so, and libutil.{a,so}

安装的目录：/usr/include/arpa, /usr/include/bits, /usr/include/gnu, /usr/include/net, /usr/include/netash, /usr/include/netatalk, /usr/include/netax25, /usr/include/neteconet, /usr/include/netinet, /usr/include/netipx, /usr/include/netiucv, /usr/include/netpacket, /usr/include/netrom, /usr/include/netrose, /usr/include/nfs, /usr/include/protocols, /usr/include/rpc, /usr/include/rpcsvc, /usr/include/sys, /usr/lib/gconv, /usr/lib/glibc, /usr/lib/locale, /usr/share/i18n, /usr/share/zoneinfo

简要描述

catchsegv	当程序因为段错误终止时可以创建栈追踪
gencat	生成消息目录
getconf	为文件系统特定变量显示系统配置值
getent	从管理的数据库获取入口
iconv	进行字符集转换
iconvconfig	创建快速加载 iconv 模块配置文件
ldconfig	配置动态链接运行时绑定
ldd	指出给定程序或共享库需要的共享库
lddlibc4	以对象文件辅助 ldd
locale	打印关于当前语言环境的变量信息
localedef	编译特定语言环境
mtrace	读取并翻译内存追踪文件并以人类可读形式显示概要
nscd	为大多数常见名称服务请求提供缓存的守护进程
pcprofiledump	填充由 PC 剖析产生的信息
pt_chown	一个 grantpt 的帮助程序，用于设置拥有主，拥有组和从伪终端的访问权限
rpcgen	产生 C 代码实现远程过程调用(Remonte Procedure Call,RPC)协议

sln 静态链接的 ln 程序
sprof 读取并显示共享对象剖析数据
tzselect 请求用户关于系统的位置信息并报告相应的时区描述
xtrace 通过打印当前执行函数跟踪可执行程序
zdump 时区填充器
zic 时区编译器
ld.so 共享库可执行文件的帮助程序
libBrokenLocale 供 Glibc 内部使用，使破坏的程序（如，一些主题程序）重新运行。更多信息请查看 glibc-2.14.1/locale/broken_cur_max.c 的注释
libSegFault 段错误信号句柄，供 catchsegv 使用
libanl 异步名称查询库
libbsd-compat 为在 Linux 下运行某个伯克利软件发行版(Berkeley Software Distribution, BSD)的程序提供需要的移植性
libc 主要 C 库
libcidn 供 Glibc 内部使用，处理 getaddrinfo()函数的国际化域名
libcrypt 密码库
libdl 动态链接接口库
libg 填充不含函数的库。以前是 g++的运行时库
libieee 链接到该库会为数学函数强制执行 IEEE(Institute of Electrical and Eletronic Engineers)定义的错误处理规则。默认为 POSIX.1 错误处理
libm 数学库
libmcheck 当连接到该库时开启内存分配检查
libmemusage 供 memusage 使用，帮助收集程序内存使用情况的信息
libsnl 网络服务库
libnss 命名服务转换库，包含了解析主机名，用户名，组名，别名，服务，协议等的函数
libpcprofile 包含了跟踪 CPU 花费在某个源代码行时间的函数
libpthread POSIX 线程库
libresolv 包含了 Internet 域名服务的包的创建，发送和解释的函数
librpcsvc 包含了提供 RPC 杂项服务的函数
librt 包含了 POSIX.1b 实时扩展定义的大多数接口的函数
libthread_db 包含了为多线程程序搭建调试器的函数
libutil 包含了许多 Unix 工具集使用的标准函数的代码

6.10. 重新调整工具链

既然最终的 C 库已经安装好了，现在就需要再次调整工具链了，这样所有新编译的程序才能连接到新的库，这次和第 5 章开头的“调整”是类似的，但调整的顺序是相反的。在第 5 章，工具链的指向从宿主机的/{,usr/}lib 目录改为新的/tools/lib 目录。现在，工具链指向从/tools/lib 改为 LFS/{,usr/}lib 目录。

```
mv -v /tools/bin/{ld,ld-old}
mv -v /tools/$(gcc -dumpmachine)/bin/{ld,ld-old}
mv -v /tools/bin/{ld-new,ld}
ln -sv /tools/bin/ld /tools/$(gcc -dumpmachine)/bin/ld
```

首先，备份/tools 链接，然后用我们在第 5 章创建的调整链接替代它。我们也要创建一个指向/tools/$(gcc –dumpmachine)/bin 中副本的链接：

然后，修改 GCC 的 specs 文件指向新的动态连接器。简单地删除“/tools”下所有的实例应该能使路径正确地指向动态连接器。调整 specs 文件也能使 GCC 知道到哪儿搜索正确的头文件和 Glibc 启动文件。用 sed 命令就能达到目的：

```
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
```

亲自查看 specs 文件是否按照我们的想法被修改无疑是个好主意。

此刻最迫切是的确认调整工具链后工具链的基本功能（编译链接）如我们预期的那样工作。运行下面的命令进行正确性检查：

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

如果一切正常是没有错误的，而且输出命令的最后一行是（不同平台动态连接器名称可能不同）：

```
[Requesting program interpreter: /lib/ld-linux.so.2]
```

可以看到，现在动态连接器的前缀是/lib。

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

现在检查我们配置使用了正确的启动文件：

如果一切正确，也是没有错误的，而且最后命令的输出是：

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

验证编译器在搜索正确的头文件：

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

该命令成功执行后输出：

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

下一步，验证新的链接器使用了正确的搜索路径：

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

若一切正确没有错误，命令的最后输出是（不同平台三元目标不同）：

```
SEARCH_DIR("/tools/i686-pc-linux-gnu/lib")
SEARCH_DIR("/usr/lib")
SEARCH_DIR("/lib");
```

下一步验证我们使用了正确的 libc：

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

若一切正确没有错误，命令的最后输出是（64 位机上可能为 lib64）：

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

最后，验证 GCC 是否使用正确的动态连接器：

```
grep found dummy.log
```

若一切正确没有错误，命令的最后输出是（特定平台动态链接器名称可能不同，64 位机上可能为 lib64）：

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

如果输出和上面显示的不同，或者根本没有输出，则表示某些地方出错了。检查跟踪之前的步骤找出问题的源头并改正。最可能的原因是 specs 文件调整时出错了。在继续下去前必须先解决所有的问题。

若一切正常，清除测试文件：

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

6.11. Zlib-1.2.5

Zlib 包含了压缩和解压缩的例程。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 2.8MB

6.11.1. 安装 Zlib

首先，修复头文件里的一个错字：

```
sed -i 's/ifdef _LARGEFILE64_SOURCE/ifndef \
_LARGEFILE64_SOURCE/' zlib.h
```

配置 Zlib：

```
CFLAGS='-mstackrealign -fPIC -O3' ./configure --prefix=/usr
```

新的环境配置变量的含义：

CFLAGS='-mstackrealign –fPIC –O3'

设置 CFLAGS 变量，覆盖软件包中默认的优化参数防止运行时出错。注意参数 -mstackrealign 在非 Intel 架构的系统上可能引起编译错误。

编译：

```
make
```

测试结果：

```
make check
```

安装软件：

```
make install
```

共享库需要移到/lib 下，所以/usr/lib 下的.so 文件需要重建。

```
mv -v /usr/lib/libz.so.* /lib
ln -sfv ../../lib/libz.so.1.2.5 /usr/lib/libz.so
```

6.11.2. Zlib 的内容

安装的库： libz.{a,so}

简要描述

libz	包含了某些程序使用的压缩和解压缩的函数

6.12. File-5.09

File 包含了判断给定文件的类型的一组工具。

估计搭建时间： 0.2SBU

需要磁盘空间： 9.5MB

6.12.1. 安装 File

配置：

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

编译：

```
make
```

测试编译结果：

```
make check
```

安装：

```
make install
```

6.12.2. File 的内容

安装的程序： file

安装的库： libmagic.{a.so}

简要描述

file 用于分类每个给定文件；这是通过执行一些测试完成的：文件系统测试，魔数测试，语言测试

libmagic 供 file 程序使用，包含了魔数识别的例程

6.13. Binutils-2.21.1a

Binutils 包含了一个链接器，一个汇编器和其他处理对象文件的工具。

估计搭建时间： 1.9SBU

需要磁盘空间： 307MB

6.13.1. 安装 Binutils

下面执行一个简单的测试，验证 chroot 环境中的 PTY 是否正常工作：

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

该命令的输出应该为：

```
spawn ls
```

然而，若输出像下面这样，说明 PTY 操作的环境配置不正确。在运行 Binutils 和 GCC 测试前，必须先解决这个问题：

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

取消安装过时的 standards.info 文件，因为以后在 Autoconf 安装时会安装一个较新的版本：

```
rm -fv etc/standards.info
sed -i.bak '/^INFO/s/standards.info //' etc/Makefile.in
```

修复一些测试，否则在使用 GCC-4.6.1 时会失败：

```
sed -i "/exception_defines.h/d" ld/testsuite/ld-elf/new.cc
sed -i "s/-fvtable-gc //" \
   ld/testsuite/ld-selective/selective.exp
```

根据 Binutils 文档的建议，最好在其源码目录外的专用目录内编译：

```
mkdir -v ../binutils-build
cd ../binutils-build
```

配置 Binutils：

```
../binutils-2.21.1/configure --prefix=/usr --enable-shared
```

编译：

```
make tooldir=/usr
```

make 的参数的含义：

tooldir=/usr

通常，tooldir(可执行文件最终放置的位置)设置为$(exec_prefix)/$(target_alias)。例如，x86_64 机器会将该目录扩展为/usr/x86_64-unknown-linux-gnu。因为这是自定义系统，在/usr 下特定目标的目录不是必要的。如果系统是用于交叉编译（例如，在 Intel 机器上编译的代

码可以在 PowerPC 上运行）那么$(exec_prefix)/$(target_alias)就会被用到。

Important

本节中的 Binutils 的测试套件是非常重要的，所以千万不要跳过。

测试编译结果：

```
make -k check
```

安装：

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

安装 libiberty 头文件，一些软件会使用到它：

```
cp -v ../binutils-2.21.1/include/libiberty.h /usr/include
```

6.13.2. Binutils 的内容

安装的程序：addr2line, ar, as, c++filt, gprof, ld, nm, objcopy, objdump, ranlib, readelf, size, strings, strip

安装的库：libiberty.a, libbfd.{a,so}, and libopcodes.{a,so}

安装的目录：/usr/lib/ldscripts

简要描述

addr2line　将程序地址翻译为文件名和行号；给出可执行文件的地址和名称，该程序利用可执行文件的调试信息判断哪个源文件和行号和该地址关联

ar　创建和修改归档文件，从归档中释放文件

as　汇编器，用于将 gcc 的输出汇编为对象文件

c++filt　供链接器使用，去除 C++和 Java 的破碎符号，保证从冲突中重新载入

gprof　显示调用图标分析数据

ld　链接器，用于将大量对象和归档文件组合为单个文件，重新放置他们的数据并占用符号引用

nm　列出给定对象文件的符号

objcopy　转换对象文件的类型

objdump　显示给定对象文件的信息，可使用参数控制显示特定信息；对于使用编译工具的开发人员这些信息是非常有用的

ranlib　生成归档内容的序号并存入归档；该序号列出了归档文件成员的所有符号，归档文件的成员即重组织的对象文件

readelf　显示 ELF 类型二进制文件的信息

size　列出给定对象文件的块大小和文件整体大小

strings　对于给定文件，以特定最小长度（默认为 4）输出可打印字符序列；对于对象文件，默认打印初始化和加载时的字符串，而对于其他类型的文件，它会扫描整个文件

strip　从对象文件清除符号

libiberty　包含了供不同 GNU 程序使用的例程，包括 getopt，obstack，strerror，strtol 和 strtoul

libbfd　二进制文件描述符库

libopcodes　处理操作符的库，操作符即对处理器为可读文本形式的指令；用于编译类似 objdump 的工具

6.14. GMP-5.0.2

GMP 包含数学函数库，其中有大量任意精度的算术函数。

估计搭建时间： 1.7SBU

需要磁盘空间： 39MB

6.14.1. 安装 GMP

Note

如果你在为 32 位 x86 机器编译，但你的 CPU 能够运行 64 位代码而且已经在环境中设置了 CFLAGS 变量，则配置脚本会尝试为 64 位配置最终会失败。要避免这种情况就需要运行命令：

```
ABI=32 ./configure ...
```

修复已上传到网站的一个镜像错误：

```
sed -i 's/np + dn, qn/& - dn/' mpn/generic/dcpi1_bdiv_q.c
```

配置：

```
./configure --prefix=/usr --enable-cxx --enable-mpbsd
```

配置参数的含义：

--enable-cxx

支持 c++；

--enable-mpbsd

该参数允许编译 Berkeley MP 兼容库；

编译：

```
make
```

Important

本节中 GMP 的测试套件是非常关键的。千万不要跳过。

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

测试编译结果：

确保全部共 162 个测试通过。键入下面的命令检查结果：

```
awk '/tests passed/{total+=$2} ; END{print total}' \
  gmp-check-log
```

安装：

```
make install
```

如果需要，安装 GMP 的文档：

```
mkdir -v /usr/share/doc/gmp-5.0.2
cp -v doc/{isa_abi_headache,configuration} doc/*.html \
   /usr/share/doc/gmp-5.0.2
```

6.14.2. GMP 的内容

安装的库： libgmp.{a,so}, libgmpxx.{a,so}, and libmp.{a,so}

安装的目录： /usr/share/doc/gmp-5.0.2

简要描述

libgmp	包含数学精度函数
libgmpxx	包含 C++数学精度函数
libmp	包含 Berkeley MP 数学函数

6.15. MPFR-3.1.0

MPFR 包含多精度数学函数。

估计搭建时间： 1.7SBU

需要磁盘空间： 39MB

6.15.1. 安装 MPFR

配置：

```
./configure --prefix=/usr --enable-thread-safe \
    --docdir=/usr/share/doc/mpfr-3.1.0
```

编译：

```
make
```

Important

本节中 MPFR 的测试套件是非常重要的。千万不能跳过。

测试编译结果：

```
make check
```

安装：

```
make install
```

安装 MPFR 文档：

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

6.15.2. MPFR 的内容

安装的库： libmpfr.{a,so}

安装的目录： /usr/share/doc/mpfr-3.1.0

简要描述

libmpfr	包含多精度数学函数

6.16. MPC-0.9

MPC 包含了用于复杂数字的算术运算库，可以处理任意高精度数字，保证结果正确的四舍五入。

估计搭建时间： 0.3SBU

需要磁盘空间： 10.5MB

6.16.1. 安装 MPC

配置：

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

编译：

```
make
```

测试结果：

```
make check
```

安装：

```
make install
```

6.16.2. MPC 的内容

安装的库：libmpc.{a,so}

简要描述

libmpc　包含了复杂数学函数

6.17. GCC-4.6.1

GCC 软件包是 GNU 编译器的集合，包含了 C 和 C++编译器。

估计搭建时间： 47SBU

需要磁盘空间： 1.7GB

6.17.1. 安装 GCC

使用 sed 命令取消安装 libiberty.a，因为我们使用的是 Binutils 提供的 libiberty.a：

```
sed -i 's/install_to_$(INSTALL_DEST) //' \
   libiberty/Makefile.in
```

和 5.10 节“GCC-4.6.1——第二遍“一样，使用 sed 命令强制编译器使用 -fomit-frame-pointer 编译标志保证编译一致性：

```
case `uname -m` in
   i?86) sed -i 's/^T_CFLAGS =$/& -fomit-frame-pointer/' \
      gcc/Makefile.in ;;
esac
```

脚本 fixincludes 有时会错误地尝试修改已安装的系统头文件。然而，到目前为止的头文件都是不需要修改的，键入下面的命令防止 fixincludes 脚本运行：

```
sed -i 's@\./fixinc\.sh@-c true@' gcc/Makefile.in
```

最后，应用补丁修复测试代码，这些代码用于 glibc-2.14 及以后的语言环境变化。

```
patch -Np1 -i ../gcc-4.6.1-locale-1.patch
```

根据 GCC 的文档，最好是在 GCC 源码目录外的专用目录进行编译：

```
mkdir -v ../gcc-build
cd ../gcc-build
```

配置：

```
../gcc-4.6.1/configure --prefix=/usr \
--libexecdir=/usr/lib --enable-shared \
--enable-threads=posix --enable-__cxa_atexit \
--enable-clocale=gnu --enable-languages=c,c++ \
--disable-multilib --disable-bootstrap --with-system-zlib
```

注意，对于其他语言，这里有些先决条件是无法满足的。请查看 BLFS 一书关于编译所有 GCC 支持的语言的指导。

新的配置参数的含义：

--with-system-zlib

该参数使 GCC 链接到系统安装的 zlib 库而不是自带的 zlib 版本。

编译：

```
make
```

Important

在本节中，GCC 的测试套件是非常重要的，无论如何请不要跳过。

由于 GCC 测试套件的一个测试集的堆栈会溢出，所以先增加栈的大小：

```
ulimit -s 16384
```

测试编译结果，但遇到错误并不停止检查：

```
make -k check
```

运行下面命令，查看测试结果的概述：

```
../gcc-4.6.1/contrib/test_summary
```

若仅需要概述，使用管道命令输出到 **grep –A7 Summ**。

请将检查结果和*http://www.linuxfromscratch.org/lfs/build-logs/7.0/* 及 *http://gcc.gnu.org/ml/gcc-testresults/* 作对比。

出现少量的错误可能无法避免。GCC 的开发者也知道这些，然而目前还没有解决。特别的，libmudflap 测试作为 GCC 的 bug 是典型的问题（*http://gcc.gnu.org/bugzilla/show_bug.cgi?id=20003* ）。除非测试结果和该 URL 的结果有大量不同，否则可以放心继续下去。

安装：

```
make install
```

一些软件希望将 C 预处理安装到/lib 目录，为满足这些软件的要求，创建一个符号链接：

```
ln -sv ../usr/bin/cpp /lib
```

许多软件使用 cc 调用 C 编译器，故也创建一个符号链接：

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

现在我们的最终工具链已就位，现在亟待确认工具链是否像我们期望地那样工作。进行和前面章节一样的正确性测试：

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

若一切正常，则没有错误输出，而且命令输出的最后是（考虑到特定平台动态链接器的名称的不同）：

```
[Requesting program interpreter: /lib/ld-linux.so.2]
```

现在确认我们配置使用了正确的启动文件：

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

若一切正常，则没有错误输出，而且命令输出的最后是：

```
/usr/lib/gcc/i686-pc-linux-gnu/4.6.1/../../../crt1.o succeeded
/usr/lib/gcc/i686-pc-linux-gnu/4.6.1/../../../crti.o succeeded
/usr/lib/gcc/i686-pc-linux-gnu/4.6.1/../../../crtn.o succeeded
```

根据你自己机器架构的区别，上述输出可能有微小不同，这些不同可能是/usr/lib/gcc后的目录名称的不同。如果是 64 位系统，该字符串的末端很可能是 lib64。这里，最重要的是 gcc 找到了位于/usr/lib 目录下所有三个 crt*.o 文件。

验证编译器能找到正确的头文件：

```
grep -B4 '^ /usr/include' dummy.log
```

该命令应该成功运行且输出为：

```
#include <...> search starts here:
/usr/local/include
/usr/lib/gcc/i686-pc-linux-gnu/4.6.1/include
/usr/lib/gcc/i686-pc-linux-gnu/4.6.1/include-fixed
/usr/include
```

再次强调，由于你机器架构不同，在你目标三元组后的目录名可能和上面不同。

Note

自 GCC4.3.0 版本起，GCC 无条件将 limits.h 文件安装入私有的 include-fixed 目录，故该目录需要已经存在。

下一步，验证新的链接器使用了正确的搜索路径：

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

若一切正常，则没有错误输出，而且命令输出的最后是（特定平台目标三元组会有不同）：

```
SEARCH_DIR("/usr/i686-pc-linux-gnu/lib")
SEARCH_DIR("/usr/local/lib")
SEARCH_DIR("/lib")
SEARCH_DIR("/usr/lib");
```

在 64 位机上可能看到更多目录，例如，下面是来自 x86_64 位机的输出：

```
SEARCH_DIR("/usr/x86_64-unknown-linux-gnu/lib64")
SEARCH_DIR("/usr/local/lib64")
SEARCH_DIR("/lib64")
SEARCH_DIR("/usr/lib64")
SEARCH_DIR("/usr/x86_64-unknown-linux-gnu/lib")
SEARCH_DIR("/usr/local/lib")
SEARCH_DIR("/lib")
SEARCH_DIR("/usr/lib");
```

现在验证我们使用了正确的 libc：

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

若一切正常，则没有错误输出，而且命令输出的最后是（64 位机上可能是 lib64）：

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

最后，确认 GCC 使用了正确的动态链接器：

```
grep found dummy.log
```

若一切正常，则没有错误输出，而且命令输出的最后是（特定平台的动态链接器名称可能不同，在 64 位机上目录名为 lib64）：

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

如果你的输出和上面不同或者根本没有输出，则说明出现了严重错误。检查跟踪前面的步骤查处错误的根源并纠正它。最肯能出错的地方是 specs 文件的调整。在进行下一步前请务必解决所有问题。

若一切工作正常，清除测试文件：

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

6.17.2. GCC 的内容

安装的程序：c++, cc (link to gcc), cpp, g++, gcc, gccbug, and gcov

安装的库：libgcc.a, libgcc_eh.a, libgcc_s.so, libgcov.a, libgomp.{a,so}, libmudflap.{a,so}, libmudflapth.{a,so}, libssp.{a,so}, libssp_nonshared.a, libstdc++.{a,so} and libsupc++.a

安装的目录：/usr/include/c++, /usr/lib/gcc, /usr/share/gcc-4.6.1

简要描述

c++ C++编译器

cc C 编译器

cpp C 预处理器；编译器用它来扩展源文件中#include, #define 等类似的声明

g++ C++编译器

gcc C 编译器

gccbug 一个用于帮助创建 bug 报告的 shell 脚本

gcov 一个全面的测试工具；用于分析程序决断哪里的优化最为有效

libgcc 为 gcc 提供运行时的支持

libgcov 当 GCC 允许分析时该库被链接到程序中

libgomp 一个 OpenMP API 的 GNU 实现，供 C/C++和 Fortran 等多平台共享内存并行编程使用

libmudflap 包含支持 GCC 边界检查功能的例程

libssp 包含支持 GCC 堆栈溢出保护功能的例程

libstdc++ 标准 C++库

libsupc++ 对 C++语言提供支持

6.18. Sed-4.2.1

Sed 是一个流编辑器。

估计搭建时间： 0.2SBU

需要磁盘空间： 8.3MB

6.18.1. 安装 Sed

配置 sed：

```
./configure --prefix=/usr --bindir=/bin \
--htmldir=/usr/share/doc/sed-4.2.1
```

新的配置参数的含义：

--htmldir

指定 HTML 文档将安装到的目录。

编译：

```
make
```

产生 HTML 文档：

```
make html
```

测试结果：

```
make check
```

安装：

```
make install
```

安装 HTML 文档：

```
make -C doc install-html
```

6.18.2. Sed 的内容

安装的程序：sed

安装的目录：/usr/share/doc/sed-4.2.1

简要描述

sed 以行为单位过滤和转换文本文件

6.19. Bzip2-1.0.6

Bzip2 用于压缩和解压缩文件。用 bzip2 压缩文本文件产生的压缩比率要比 gzip 的高得多。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 6.4MB

6.19.1. 安装 Bzip2

应用补丁使 Bzip2 能安装文档：

```
patch -Np1 -i ../bzip2-1.0.6-install_docs-1.patch
```

下面的命令确保安装相关的符号链接：

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

准备编译 Bzip2：

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

make 参数的含义：

-f Makefile-libbz2_so

该参数使 Bzip2 使用不同的 Makefile 文件编译，本例中是 Makefile-libbz2_so，它会创建一个动态 libbz2.so 库并将 Bzip2 工具集链接至它。

编译并测试：

```
make
```

安装程序：

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

将 bzip2 共享文件安装到/bin 目录，添加必要的符号链接并清理：

```
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
```

6.19.2. Bzip2 的内容

安装的程序：bunzip2 (链接至 bzip2), bzcat (链接至 bzip2), bzcmp (链接至 bzdiff), bzdiff, bzegrep(链接至 bzgrep), bzfgrep (链接至 bzgrep), bzgrep, bzip2, bzip2recover, bzless (链接至 bzmore), 和 bzmore

安装的库：libbz2.{a,so}

安装的目录：/usr/share/doc/bzip2-1.0.6

简要描述

bunzip2	解压缩 bzip 压缩的文件
bzcat	解压缩到标准输出

bzcmp 在 bzip 压缩的文件上运行 cmp 命令
bzdiff 在 bzip 压缩的文件上执行 diff 命令
bzegrep 在 bzip 压缩的文件上执行 egrep 命令
bzfgrep 在 bzip 压缩的文件上执行 fgrep 命令
bzgrep 在 bzip 压缩的文件上执行 grep 命令
bzip2 使用 Burrows-Wheeler 块排序文本压缩算法和 Huffman 代码压缩文件；该压缩比率比更方便的压缩算法，如 gzip 的”Lempel-Ziv”的压缩比率更好
bzip2recover 从 bzip 压缩的文件恢复数据
bzless 在 bzip 压缩的文件上执行 less 命令
bzmore 在 bzip 压缩的文件上执行 more 命令
libbz2* 使用 Burrows-Wheeler 算法，该库实现了低损失，块排序数据压缩

6.20. Ncurses-5.9

Ncurse 是用于独立终端处理字符界面的库。

估计搭建时间： 0.8SBU

需要磁盘空间： 35MB

6.20.1. 安装 Ncurse

配置：

```
./configure --prefix=/usr --with-shared --without-debug \
--enable-widec
```

配置参数的含义：

--enable-widec

该参数会使编译建立普通字符库（即 libncurses.so.5.9）而不是宽字符库（即.libncursesw.so.5.9）。宽字符库用于多字节或传统 8bit 语言环境，而普通字符库仅能正常地工作于 8bit 语言环境。宽字符库和普通字符库在源代码级别兼容，但在二进制级别不兼容。

编译：

```
make
```

本软件包含了一个测试套件，但是需要在软件安装完成后才能运行。该测试套件被放置于 test/目录。查看该目录下的 README 文件获取详细信息。

安装：

```
make install
```

将共享库移动到/lib 目录：

```
mv -v /usr/lib/libncursesw.so.5* /lib
```

由于函数库被移动了，而符号链接却指向一个不存在的文件，所以需要重建它：

```
ln -sfv ../../lib/libncursesw.so.5 /usr/lib/libncursesw.so
```

许多程序仍然希望链接器能够找到非宽字符的 Ncurses 库，利用符号链接和链接器文件将这些程序仍链接到宽字符集：

```
for lib in ncurses form panel menu ; do \
rm -vf /usr/lib/lib${lib}.so ; \
echo "INPUT(-l${lib}w)" >/usr/lib/lib${lib}.so ; \
ln -sfv lib${lib}w.a /usr/lib/lib${lib}.a ; \
done
ln -sfv libncurses++w.a /usr/lib/libncurses++.a
```

最后，验证所有需要-lcurses 的老程序在编译时仍有效：

```
rm -vf /usr/lib/libcursesw.so
echo "INPUT(-lncursesw)" >/usr/lib/libcursesw.so
ln -sfv libncurses.so /usr/lib/libcurses.so
ln -sfv libncursesw.a /usr/lib/libcursesw.a
ln -sfv libncurses.a /usr/lib/libcurses.a
```

如果需要，安装 Ncurses 文档：

```
mkdir -v /usr/share/doc/ncurses-5.9
cp -v -R doc/* /usr/share/doc/ncurses-5.9
```

Note

上面的指示并没有创建非宽字符集的 Ncurses 库，这是因为从源代码编译安装的软件包在运行时都不会链接到这些库。如果一些二进制包程序需要或你希望满足 LSB 标准而一定需要非宽字符库，使用下面的命令再次创建软件包：

```
make distclean
./configure --prefix=/usr --with-shared --without-normal \
--without-debug --without-cxx-binding
make sources libs
cp -av lib/lib*.so.5* /usr/lib
```

6.20.2. Ncurses 的内容

安装的程序：captoinfo (link to tic), clear, infocmp, infotocap (link to tic), ncursesw5-config, reset (link to tset), tic, toe, tput, tset

安装的库：libcursesw.{a,so} (指向 libncursesw.{a,so}的链接或脚本), libformw.{a,so}, libmenuw.{a,so}, libncurses++w.a, libncursesw.{a,so}, libpanelw.{a,so}和其库名无'w'的非宽字符集

安装的目录：/usr/share/tabset, /usr/share/terminfo

简要描述

captoinfo	将 termcap 描述转换为 terminfo 描述
clear	如果可能就请清屏
infocmp	比较或打印 terminfo 描述
infotocap	将 terminfo 转换成 termcap 描述
ncursesw5-config	为 ncurses 提供配置信息
reset	重新初始化终端为默认值
tic	一个 terminfo 入口描述编译器，用于为 ncurses 库例程将 terminfo 文件从源格式翻译为二进制格式。terminfo 文件包含了某个终端的容量信息
toe	列出所有可用的终端类型，并为每个类型给出主要名称和描述
tput	使终端相关的容量变量可供 shell 使用；它也可用于重置或初始化终端或报告终端的长名称
tset	可以被用于初始化终端
libcurses	指向 libncurses 的链接
libncurses	提供在终端屏幕上以各种复杂方式显示文本的功能；这些功能的一个极好的例子就是内核的 make menuconfig 显示的菜单

libform	包含实现表单的功能
libmenu	包含实现菜单的功能
libpanel	包含实现面板的功能

6.21. Util-linux-2.20

Util-linux 包含了一些杂项工具：处理文件系统，控制台，分区和信息等等程序。

估计搭建时间： 0.7SBU

需要磁盘空间： 69MB

6.21.1 与 FHS 的兼容性

FHS 建议将 adjtime 文件放置在/var/lib 目录下而不是/etc 目录下。若要使 hwclock 程序和 FHS 兼容，则运行下面的命令：

```
sed -e 's@etc/adjtime@var/lib/hwclock/adjtime@g' \
   -i $(grep -rl '/etc/adjtime' .)
mkdir -pv /var/lib/hwclock
```

6.21.2 安装 Util-linux

```
./configure --enable-arch --enable-partx --enable-write
```

配置参数的含义如下：

--enable-arch

编译 arch 程序。

--enable-partx

编译 addpart，delpart 和 partx 程序

--enable-write

编译 write 程序

编译软件：

```
make
```

本软件不包含测试套件。

安装软件：

```
make install
```

6.21.3 Util-linux 的内容

安装的程序：addpart, agetty, arch, blkid, blockdev, cal, cfdisk, chkdupexe, chrt, col, colcrt, colrm, column, ctrlaltdel, cytune, ddate, delpart, dmesg, fallocate, fdformat, fdisk, findfs,findmnt, flock, fsck, fsck.cramfs, fsck.minix, fsfreeze, fstrim, getopt, hexdump,hwclock, i386, ionice, ipcmk, ipcrm, ipcs, isosize, ldattach, line, linux32, linux64,logger, look, losetup, lsblk, lscpu, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix,mkswap, more, mount, mountpoint, namei, partx, pg, pivot_root, readprofile, rename,renice, rev, rtcwake, script, scriptreplay, setarch, setsid, setterm, sfdisk, swaplabel,swapoff (link to swapon), swapon, switch_root, tailf, taskset, tunelp, ul, umount, unshare, uuidd, uuidgen, wall, whereis, wipefs, write

安装的目录：libblkid.{a,so}, libmount.{a,so}, libuuid.{a,so}

安装的库：/usr/share/getopt, /var/lib/hwclock

简要描述

addpart　　监测到新分区时向 linux 内核发出通知

agetty 打开一个 tty 端口，弹出窗口要求输入登录名，然后调用 login 程序
arch 报告计算机的架构
blkid 命令行工具，用于定位和打印设备属性
blockdev 用于从命令行为块设备调用 ioctls 操作
cal 显示一个简单日历
cfdisk 管理给定设备的分区表
chkdupexe 搜索复制的可执行文件
chrt 处理进程的实时属性
col 过滤控制字符
colcrt 对于某些缺乏加粗或半行的终端过滤 nroff 的输出
colrm 过滤给定的列
column 将指定的文件格式化为多栏形式
ctrlaltdel 将 Ctrl+Alt+Del 组合键设置为一个硬或软重置
cytune 调节 Cyclades 卡串行驱动的参数
ddate 给出 Discordian 日期或将给定的 Gregorian 日期转换为 Discrodian 日期
delpart 请求 Linux 内核移除一个分区
dmesg 填充内核启动信息
fallocate 为文件预分配空间
fdformat 为软盘执行低级格式化
fdisk 操作给定设备的分区表
findfs 利用标签或通用唯一标识符(Universally Unique Identifier, UUID)查找文件系统
findmnt 使用 libmount 库的命令行接口，和 mountinfo，fstab 和 mtab 文件协同工作
flock 获取并保持一个文件锁然后执行命令
fsck 用于检查和优化修复文件系统
fsck.cramfs 在给定设备的 Cramfs 文件系统上执行连续检查
fsck.minix 在给定设备的 Minix 文件系统上执行连续检查
fsfreeze 包装在 FIFREEZE/FITHAW ioctl 内核驱动操作周围的简单外壳
fstrim 丢弃已挂载文件系统上无用的块
getopt 在给定命令行上的转换选项
hexdump 用十六进制或其他给定格式填充给定文件
hwclock 读取或设置系统的硬件时钟，也称为实时时钟(Real-Time Clock, RTC)或基本输入输出系统(Basic Input-Output System, BIOS)时钟
i386 设置构架的符号链接
ionice 设置或读取某个程序的 io 规程类和优先级
ipcmk 创建多种内部进程通信(Inter-Process Communication,IPC)资源
ipcrm 移除给定的 IPC 资源
ipcs 提供 IPC 状态信息
isosize 获取 iso9660 文件系统的大小
ldattach 为串行附加一个行规则
line 复制一行
linux32 到架构设置的符号链接
linux64 到架构设置的符号链接
logger 将给定信息键入系统日志
look 显示以给定字符串开头的行

losetup	启动并控制 loop 设备
lsblk	以树形结构列出所有或选定的块设备信息
lscpu	打印 cpu 架构信息
mcookie	为 xauth 产生魔法 cookies(128bit 十六进制随机数)
mkfs	在设备(通常是硬盘分区)创建文件系统
mkfs.bfs	创建 Santa Cruz Operations(SCO) bifs 文件系统
mkfs.cramfs	创建 cramfs 文件系统
mkfs.minix	创建 minix 文件系统
mkswap	初始化将要用做 swap 分区的设备或文件
more	一个文本页过滤器，每次在屏幕显示一页
mount	将给定设备上的文件系统挂载到文件系统树的指定目录下
mountpoint	检查某个目录是否为一个挂载点
namei	在给定路径名中显示符号链接
partx	将磁盘分区的就位和编号通知内核
pg	在屏幕上一次全屏显示一个文本文件
pivot_root	使指定文件系统成为当前进程的根文件系统
readprofile	读取内核配置信息
rename	重命名给定文件
renice	改变运行进程的优先级
rev	逆序指定文件行
rtcwake	用于进入系统睡眠状态，直到指定时间醒来
script	使终端会话成为类型脚本
scriptreplay	使用时间信息回放类型脚本
setarch	在新程序环境中改变架构信息并设置个人标记
setsid	在新会话中运行程序
setterm	设置终端属性
sfdisk	磁盘分区表管理器
swaplabel	改变交换分区 UUID 和标签
swapoff	禁用设备和文件的分页及交换功能
swapon	启用设备和文件的分页及交换功能，并累出当前正在使用的设备和文件
switch_root	转换到另一个文件系统并作为挂载树的根
tailf	跟踪 log 的增长，显示日志文件的最后 10 行，然后当日志文件增长时继续显示新的项
taskset	恢复或设置进程的 cpu 关联
tunelp	调节行打印的参数
ul	一个过滤器，将当前终端下划线部分翻译为转义序列
umount	从文件数卸载文件系统
unshare	运行进程并使部分命名空间不和父进程共享
uuidd	一个 UUID 库使用的守护进程，用安全可靠的方法产生基于时间的 UUID
uuidgen	创建新的 UUID。每个 UUID 在本地系统和其他系统中都可以看作唯一存在
wall	在登录用户的终端中默认在标准输出中显示文件内容
whereis	获取给定命令的本地二进制文件，源，man 页面
wipefs	清除设备的文件系统签名
write	若用户没有禁用信息收取，则发送信息到给定用户

libblkid	包含设备标识和段抽取的例程
libuuid	包含例程用于产生唯一的不局限于本地系统的标识符

6.22. E2fsprogs-1.41.14

E2fsprogs 是处理 ext2 文件系统的一组工具。它也支持 ext3 和 ext4 日志文件系统。

估计搭建时间： 0.5SBU

需要磁盘空间： 45MB

6.22.1. 安装 E2fsprogs

根据 e2fsprogs 文档，最好在源文件树的子文件夹下进行编译：

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

配置 e2fsprogs：

```
PKG_CONFIG=/tools/bin/true LDFLAGS=-lblkid \
../configure --prefix=/usr --with-root-prefix="" \
--enable-elf-shlibs --disable-libblkid --disable-libuuid \
--disable-uuidd --disable-fsck
```

配置参数的含义如下：

PKG_CONFIG…

该参数使得 E2fsprogs 可以在没有预先安装 Pkg-config 的情况下编译。

--with-root-prefix=””

某些关键程序（如 e2fsck 程序）在/usr 未挂载时仍然需要它们。他们属于像/lib 和/sbin 这样的目录。如果本参数没有传递给 E2fsprogs 的配置文件，这些程序会被安装到/usr 目录下。

--enable-elf-shlibs

创建本软件包内一些程序会用到的共享库。

*--disable-**

该参数禁止 e2fsprogs 编译安装 libuuid 和 libblkid 库，uuidd 守护进程和 fsck 外壳已经包含在之前安装的 Util-Linux 中。

编译软件：

```
make
```

若要测试编译结果，键入命令：

```
make check
```

E2fsprogs 的一个测试会尝试分配 256MB 内存，如果你的内存明显不够，那么我们建议为该测试启用足够的交换空间。请查阅 2.3 节“在分区上创建文件系统”和 2.4 节“挂载新分区”获得创建和启用交换分区的足够信息。

安装二进制包，文档和共享库：

```
make install
```

安装静态库和头文件：

```
make install-libs
```

为了以后可以移除 debug 符号，需要使安装的静态库可写：

```
chmod -v u+w /usr/lib/{libcom_err,libe2p,libext2fs,libss}.a
```

本软件包安装了一个 gzip 压缩的.info 文件但没有更新整个系统的 dir 文件。键入下面的命令解压缩该文件然后更新系统 dir 文件：

```
gunzip -v /usr/share/info/libext2fs.info.gz
install-info --dir-file=/usr/share/info/dir \
   /usr/share/info/libext2fs.info
```

如果需要，创建并安装一些额外的文档：

```
makeinfo -o doc/com_err.info ../lib/et/com_err.texinfo
install -v -m644 doc/com_err.info /usr/share/info
install-info --dir-file=/usr/share/info/dir \
   /usr/share/info/com_err.info
```

6.22.2. E2fsprogs 的内容

安装的程序：badblocks, chattr, compile_et, debugfs, dumpe2fs, e2freefrag, e2fsck, e2image, e2initrd_helper, e2label, e2undo, 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, tune2fs

安装的库文件：libcom_err.{a,so}, libe2p.{a,so}, libext2fs.{a,so} ， libss.{a,so}

安装的目录：/usr/include/e2p, /usr/include/et, /usr/include/ext2fs, /usr/include/ss, /usr/share/et, /usr/share/ss

简要描述

badlocks	搜索设备（通常是磁盘分区）的坏块
chattr	改变 ext2 文件系统文件的属性，也可用于 ext3 文件系统（ext2 文件系统的日志版本）
compile_et	错误表编译器，将错误代码名表和消息表转换为 com_err 库适合的 C 源文件
debugfs	文件系统调试器，用于检查和改变 ext2 文件系统的状态
dumpe2fs	为给定设备上的文件系统打印超级块和块组信息
e2freefrag	报告空闲空间碎片信息
e2fsck	主要用于检查 ext2 和 ext3 文件系统，也可用于修复 ext2,ext3 文件系统
e2image	用于保存关键 ext2 文件系统数据到文件
e2initrd_helper	打印给定文件系统的 FS 类型，打印结果为设备名或标签
e2label	显示或改变给定设备上 ext2 文件系统的文件系统标签
e2undo	重做在设备上找到的 ext2/ext3/ext4 文件系统 undo_log 日志，这可以用来撤销一个 e2fsprogs 程序的失败操作
filefrag	报告某个特定文件的碎片程度
fsck.ext2	默认检查 ext2 文件系统，它是指向 e2fsck 的硬链接
fsck.ext3	默认检查 ext3 文件系统，它是指向 e2fsck 的硬链接
fsck.ext4	默认检查 ext4 文件系统，它是指向 e2fsck 的硬链接
fsck.ext4dev	默认检查 ext4 开发文件系统，它是指向 e2fsck 的硬链接
logsave	将命令行输出保存为日志文件
lsattr	列出位于第二个扩展文件系统的文件的属性
mk_cmds	将命令名和帮助消息表转换为 libss 库适合的 C 源文件

mke2fs 在指定设备上创建 ext2 或 ext3 文件系统
mkefs.ext2 默认创建 ext2 文件系统，它是指向 mke2fs 的硬链接
mkefs.ext3 默认创建 ext3 文件系统，它是指向 mke2fs 的硬链接
mkefs.ext4 默认创建 ext4 文件系统，它是指向 mke2fs 的硬链接
mkefs.ext2dev 默认创建 ext4 开发文件系统，它是指向 mke2fs 的硬链接
mklost+found 用于在 ext2 文件系统上创建一个 lost+found 目录，它会为该目录预分配磁盘块来降低 e2fsck 的任务
resize2fs 用于增大或缩小 ext2 文件系统
tune2fs 在 ext2 文件系统上调节可调文件系统参数
libcom_err 通用错误显示例程
libe2p 供 dumpe2fs, chattr 和 lsattr 使用
libext2fs 提供例程供用户级程序操作 ext2 文件系统
libss 供 debugfs 使用

6.23. Coreutils-8.14

Coreutils 包含了一系列显示和设置系统基本特性的工具。

估计搭建时间： 3.2SBU

需要磁盘空间： 99MB

6.23.1. 安装 Coreutils

该软件包中的 uname 程序使用参数-p 时总是返回“未知参数”，使用补丁修复 Intel 架构上该软件包的 bug：

```
case `uname -m` in
   i?86 | x86_64) patch -Np1 -i \
   ../coreutils-8.14-uname-1.patch ;;
esac
```

按照 POSIX 标准，Coreutils 的程序必须能在包括多字节语言环境下正确识别字符边界。使用下面的补丁修复该兼容性和其他国家化相关的 bug：

```
patch -Np1 -i ../coreutils-8.14-i18n-1.patch
```

Note

在该补丁中曾发现很多 bug，当提交新 bug 到 Coreutils 维护人员时，请确认他们是否会再产生。

现在配置 Coreutils：

```
./configure --prefix=/usr \
   --enable-no-install-program=kill,uptime
```

配置参数的含义：

--enable-no-install-program=kill,uptime

该参数的目的是取消安装 kill，uptime 程序，以后会用别的程序包安装它们。

编译软件：

```
make
```

如果不想运行测试套件请直接跳到“安装软件”。

首先，以 root 用户运行测试套件：

```
make NON_ROOT_USERNAME=nobody check-root
```

我们要以 nobody 用户运行测试提醒者。然而，某些测试要求用户属于多个组，所以我们建立一个临时组并将 nobody 用户添加到该组：

```
echo "dummy:x:1000:nobody" >> /etc/group
```

改变某些权限以便于非 root 用户可以编译运行测试套件：

```
chown -Rv nobody .
```

现在运行测试：

```
su-tools nobody -s /bin/bash -c  \
   "make RUN_EXPENSIVE_TESTS=yes check"
```

移除临时组：

```
sed -i '/dummy/d' /etc/group
```

安装软件：

```
make install
```

将程序移动到 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
```

LFS-Bootsript 软件包的某些脚本依赖于 head, sleep 和 nice 程序。由于在启动早期/usr 不可用，故这些二进制文件需要放置在根分区：

```
mv -v /usr/bin/{head,sleep,nice} /bin
```

6.23.2. Coreutils 的内容

安装的程序：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, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, 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, yes

安装的库：libstdbuf.so

安装的目录：/usr/lib/coreutils

简要描述

base64	根据 base64（RFC3548）规范编码或解码数据
basename	从文件名截除路径和给定前缀
cat	输出文件内容到标准输出
chcon	为文件和目录改变安全上下文
chgrp	改变文件和目录的拥有组
chmod	将文件的权限修改为指定模式，该模式可以为一个代表改变权限的符号或一个代表新权限的 10 进制数
chroot	改变文件和目录的拥有者和拥有组
cksum	打印给定文件的循环冗余校验（Cyclic Redumdancy Check, CRC）和与字节数
comm	比较两个有序文件，以三栏的形式输出唯一的行和相同的行
cp	复制文件

csplit 将给定文件分割为几个新文件，分割的标准是依据给定模式或行号，输出每个新文件的字节数
cut 根据给定域或位置打印某些行或选定的部分
date 以给定形式显示当前时间或设置系统时间
dd 使用给定块大小和数目复制文件，同时也可以执行某些转换
df 报告所有挂载文件系统上可用的磁盘空间（和已使用的空间），或者仅在选定的文件的文件系统上报告空间
dir 列出给定目录的内容（和 ls 命令相同）
dircolors 输出命令设置 LS_COLOR 环境变量来改变 ls 命令使用的颜色架构
dirname 从文件名中截除非目录部分
du 报告当前目录使用的磁盘空间，可以报告给定目录（包括所有子目录）或给定文件
echo 显示给定字符串
env 在修改的环境变量中运行命令
expand 将 tab 转换为空格
expr 计算表达式
factor 打印所有特定整数的素因子
false 不做任何事，总是退出并给出表示失败的错误码
fmt 重新格式化给定文件的段落
fold 包裹给定文件的某些行
groups 报告用户的组成员
head 打印给定文件的首 10 行（或给定行数）
hostid 报告主机的数字标识符（16 进制）
id 报告当前用户或特定用户的有效用户 ID，组 ID 和组成员
install 复制文件同时设置它们的权限，如果可能并设置它们的拥有主和拥有组
join 按相同标识合并两个文件的行
link 以给定名称创建文件的硬链接
ln 创建文件的硬链接或软链接（符号链接）
logname 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）总和
mkdir 创建目录
mkfifo 以给定名称创建先进先出管道（Fisrt-In First-Out, FIFO），在 UNIX 用语中称为命名管道
mknod 用给定名称创建设备节点，设备节点可能是特殊字符文件或特殊块文件或一个 FIFO
mktemp 以安全的方式创建临时文件，主要用在脚本中
mv 移动或重命名文件或目录
nice 以修改的优先级运行程序
nl 标识给定文件的行号
nohup 运行不会被挂起的命令，将其输出重定向到日志文件
nproc 打印进程执行单元的数目
od 以 10 进制或其他形式填充文件
paste 合并给定文件，根据行依次合并，并以 tab 分隔

pathchk 检查文件名是否合法或可移植
pinky 轻量级的指纹客户端，提供了给定用户的一些信息
pr 为打印文件分页分栏文件
printenv 打印环境变量
printf 根据给定格式打印给定参数，类似 C 语言的 printf 函数
ptx 产生给定文件内容的交换索引，并包含了每个关键字
pwd 报告当前工作目录
readlink 报告给定符号链接的名称
rm 移除文件或目录
rmdir 移除空目录
runcon 在特定安全上下文中运行命令
seq 在给定范围内按一定增长值打印一组数
sha1sum 打印或检查 160-bit 安全哈希算法 1（Secure Hash Algorithm 1, SHA1）检验和
sha224sum 打印或检查 224-bit 安全哈希算法（Secure Hash Algorithm, SHA）检验和
sha256sum 打印或检查 256-bit 安全哈希算法（Secure Hash Algorithm, SHA）检验和
sha384sum 打印或检查 384-bit 安全哈希算法（Secure Hash Algorithm, SHA）检验和
sha512sum 打印或检查 512-bit 安全哈希算法（Secure Hash Algorithm, SHA）检验和
shred 以复杂的模式反复覆盖给定文件，使它的数据无法被恢复
shuf 搅乱文本行
sleep 暂停指定时间
sort 将制定文件按大小或行数分割为小块
stat 显示文件或文件系统的状态
stdbuf 对标准流使用改变的缓冲操作运行命令
stty 设置或报告终端行设置
sum 对给定文件打印检验和与块数目
sync 刷新文件系统缓冲，强制改变的磁盘块和超级块写入磁盘
tac 逆序输出文件内容
tail 打印给定文件最后 10 行（或给定数目的行）
tee 将输出到标准输出的内容同时输出到指定文件
test 比较值并检查文件类型
timeout 运行命令并制定超时值
touch 改变文件时间戳，设置访问时间和修改时间为当前世界，若不存在文件则创建长度为 0 的文件
tr 翻译，压缩和删除标准输出的给定字符
true 不做任何事，总是退出并返回一个表示成功的返回码
truncate 将文件压缩或扩展到指定大小
tsort 执行拓扑排序，在指定文件中根据有序部分输出完整有序列表
tty 报告连接到标准输出的终端的文件名
uname 报告系统信息
unexpand 将空格转换为 tab
uniq 除了唯一行外丢弃所有其他行
unlink 移除指定文件
users 报告当前所有登录的用户名
vdir 和 ls -l 相同

wc	报告给定文件的行数，词数，字节数，当给出多个文件时给出总行数
who	报告谁登录了系统
whoami	报告和当前有效用户 ID 关联的用户名
yes	在进程被杀死前重复重复“y”或给定字符串
libstdbuf	stdbuf 程序使用的库

6.24. lana-Etc-2.30

lana-Etc 软件为网络服务和协议提供数据。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 2.3MB

6.24.1. 安装 lana-Etc

下面的命令将为/etc/protocols 和/etc/services 文件把 IANA 提供的原始数据转换为正确的格式：

```
make
```

本软件包不含测试套件。

安装软件：

```
make install
```

6.24.2. lana-Etc 的内容

安装的文件： /etc/protocols 和/etc/services

简要描述

/etc/protocols	描述 TCP/IP 子系统下可用的多种 DARPA 网络协议
/etc/services	将网络服务及其端口号，协议类型映射为友好的文本名

6.25. M4-1.4.16

M4 是一个宏处理器。

估计搭建时间： 0.4SBU

需要磁盘空间： 14.2MB

6.25.1. 安装 M4

配置软件：

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

编译：

```
make
```

要测试编译结果，键入命令：

```
make check
```

安装软件：

```
make install
```

6.25.2. M4 的内容

安装的程序： M4

简要描述

M4 扩展宏时复制所在文件。这些宏要么是内嵌宏要么是用户定义的宏且这些宏可带任意数量参数。除了执行宏扩展，M4 的内建函数可以包含命名的文件，运行 Unix 命令，执行整数运算，处理文本，递归等等。M4 程序也可以被用做编译器的边界结束程序或仅作为宏处理器。

6.26. Bison-2.5

Bison 是一个语法解析器。

估计搭建时间： 1.1SBU

需要磁盘空间： 19.2MB

6.26.1. 安装 Bison

配置：

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

如果按照 Bison 的默认配置编译，当$PATH 中没有 bison 程序时其错误信息将不支持国际化。使用下面的命令修正该错误：

```
echo '#define YYENABLE_NLS 1' >> lib/config.h
```

编译 Bison：

```
make
```

测试编译结果（大约需要 0.5SBU）：

```
make check
```

安装软件：

```
make install
```

6.26.2. Bison 包含的内容

安装的程序： bison, yacc

安装的库： liby.a

安装的目录： /usr/share/bison

简要描述

bison Bison 根据一系列规则对文本文件的结构进行分析，bison 是 yacc（另一个语法分析器）的替代选择

yacc yacc 是 bison 的外壳，这意味着应用程序仍然调用的是 yacc 而不是 bison，它使用带参数-y 的 bison 命令来调用 bison

liby.a yacc 库包含了兼容 yacc 的 yyerror 和 main 函数的实现，该库并不是非常有用，但按照 POSIX 标准要求保留它

6.27. Procps-3.2.8

Procps 是一个监视进程的程序。

估计搭建时间： 0.1SBU

需要磁盘空间： 2.3MB

6.27.1. 安装 Procps

应用下面的补丁禁止判定内核时钟滴答时显示错误信息：

```
patch -Np1 -i ../procps-3.2.8-fix_HZ_errors-1.patch
```

应用下面的补丁修复 watch 程序中 unicode 相关问题：

```
patch -Np1 -i ../procps-3.2.8-watch_unicode-1.patch
```

由于 makefile 中的 bug，导致使用 make-3.82 无法编译 procps，下面的命令将修复该 bug：

```
sed -i -e 's@\*/module.mk@proc/module.mk ps/module.mk@'
Makefile
```

编译软件包：

```
make
```

procps 软件包中不含测试套件。

直接安装软件：

```
make install
```

6.27.2. Procps 包含的内容

安装的程序：	free, kill, pgrep, pkill, pmap, ps, pwdx, skill, slabtop, snice, sysctl, tload, top, uptime, vmstat, w, and watch
安装的库：	libproc.so
简要描述	
free	报告系统中空闲内存和已使用的内存（包括物理内存和交换空间）
kill	向进程发送信号
pgrep	基于名称和其他属性查询进程
pkill	基于名称和其他属性向进程发送信号
pmap	报告指定进程的内存分布图
ps	列出当前正在运行的进程
pwdx	报告当前进程的工作目录
skill	按照过滤器向匹配的进程发送信号
slabtop	实时显示内核 slap 缓冲的详情
snice	根据过滤器改变匹配到的进程的计划优先级
sysctl	在运行时修改内核参数
tload	打印系统当前负荷图表
top	显示最密集使用 CPU 的进程列表，该程序实时提供处理器活动的查询
uptime	报告系统运行了多长时间，有多少用户登录过以及系统平均负荷

vmstat 报告虚拟内存统计值，包含的信息有处理器，内存，分页，块输入输出，门调用和 CPU 活动

w 显示当前登录的用户，登录的位置，登录的时间

watch 重复运行给定命令，全屏输出其结果，该命令使用户可以不断查看输出

libproc 包含本软件包内大多数程序使用的函数

6.28. Grep-2.9

Grep 是一个文件过滤程序。

估计搭建时间： 0.1SBU

需要磁盘空间： 2.3MB

6.28.1. 安装 Grep

首先，修复测试脚本中的问题：

```
sed -i 's/cp/#&/' tests/unibyte-bracket-expr
```

准备编译 Grep：

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

编译软件：

```
make
```

测试编译结果：

```
make check
```

安装软件：

```
make install
```

6.28.2. Grep 包含的内容

安装的程序：	egrep, fgrep, grep
简要描述	
egrep	打印满足扩展正则表达式的行
fgrep	打印满足一系列固定字符串的行
grep	打印满足基本正则表达式的行

6.29. Readline-6.2

Readline 软件包含了一系列函数库，它们提供了命令行编辑和命令行历史查询。

估计搭建时间： 0.2SBU

需要磁盘空间： 13.8MB

6.29.1. 安装 Readline

重新安装 Readline 会导致旧的库被重命名为<libraryname>.old。通常这并不算什么问题，但在某些情况下会引发 ldconfig 中的链接 bug。要避免出错就需要下面 sed 命令来修复：

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

应用补丁修复一个已知的问题：

```
patch -Np1 -i ../readline-6.2-fixes-1.patch
```

配置 Readline：

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

编译软件：

```
make SHLIB_LIBS=-lncurses
```

make 的参数的含义：

SHLIB_LIBS=-lncurses

该参数强制 Readline 链接到 libncurses（实际上是 libncursesw）库。

该软件包也没有测试套件。

安装软件：

```
make install
```

现在，将静态库移动到合适的位置：

```
mv -v /lib/lib{readline,history}.a /usr/lib
```

下面，删除/lib 目录下的.so 文件并把其他文件重新链接进/usr/lib：

```
rm -v /lib/lib{readline,history}.so
ln -sfv ../../lib/libreadline.so.6 /usr/lib/libreadline.so
ln -sfv ../../lib/libhistory.so.6 /usr/lib/libhistory.so
```

如果需要，安装文档：

```
mkdir -v /usr/share/doc/readline-6.2
install -v -m644 doc/*.{ps,pdf,html,dvi} \
/usr/share/doc/readline-6.2
```

6.29.2. Readline 的内容

安装的库： libhistory.{a,so}, and libreadline.{a,so}

安装的目录： /usr/include/readline, /usr/share/readline, /usr/share/doc/readline-6.2

简要描述

libhistory	提供一个用于回忆历史行的稳固用户接口
libreadline	为分散的程序提供一个稳固的命令行接口

6.30. Bash-4.2

Bash 程序包含了 Bourne-Again Shell。

估计搭建时间： 1.4SBU

需要磁盘空间： 35MB

6.30.1. 安装 Bash

首先，应用下列补丁修复已知的一些 bug：

```
patch -Np1 -i ../bash-4.2-fixes-3.patch
```

配置 Bash 的编译：

```
./configure --prefix=/usr --bindir=/bin \
--htmldir=/usr/share/doc/bash-4.2 --without-bash-malloc \
--with-installed-readline
```

配置参数的含义：

--htmldir

该参数指出 HTML 格式化文档应安装的目录。

--with-installed-readline

该参数使 Bash 使用已安装 Readline 函数库而不是 Bash 自身的 Readline。

编译：

```
make
```

如果不想运行测试套件请直接跳到“安装软件”。

若要运行测试套件，先确保 nobody 用户对源代码树有写权限：

```
chown -Rv nobody .
```

现在可以以 nobody 用户运行测试套件了：

```
su-tools nobody -s /bin/bash -c "make tests"
```

安装软件：

```
make install
```

运行新编译好的 bash 程序（代替当前正在被使用的 bash）：

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

Note

该参数使 bash 执行一个交互式的 shell，它禁用 hash 搜索，这样一旦安装好新程序就立即可用。

6.30.2. Bash 的内容

安装的程序： bash, bashbug, and sh (link to bash)

安装的目录： /usr/share/doc/bash-4.2

简要描述

bash bash 是一个广泛使用的命令行解释器，在执行命令前它会进行多种扩展和替换，因此 bash 是一个强大的解释工具

bashbug bashbug 是一个 shell 脚本，当使用 bash 遇到 bug 时它能组织并发送格式化的 bug 信息

sh sh 是指向 bash 的符号链接，当调用 sh 时，bash 尽量模仿 sh 历史版本的行为，保持和 POSIX 标准的统一性

6.31. Libtool-2.4

Libtool 包含了支持 GNU 函数库的脚本，它将复杂的共享库包裹成稳固轻便的接口。

估计搭建时间： 3.7SBU

需要磁盘空间： 35MB

6.31.1. 安装 Libtool

配置 Libtool：

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

编译软件：

```
make
```

若要测试编译结果（大约需要 3.0SBU），键入命令：

```
make check
```

安装软件：

```
make install
```

6.31.2. Libtool 包含的内容

安装的程序： libtool, libtoolize

安装的库： libltdl.{a,so}

安装的目录： /usr/include/libltdl, /usr/share/libtool

简要描述

libtool	为搭建通用类库提供支持
libtoolize	为软件包提供添加 libtool 支持的标准方式
libltdl	隐藏打开库文件时遇到的各种阻碍

6.32. GDBM-1.9.1

GDBM 是一个 GNU 数据库管理器。它是一个磁盘文件形式的数据库，在单独的文件中存储了键-数据对。而记录实际的数据使用该唯一键来索引，将键单独存储使得数据获取时间较短。

估计搭建时间： 0.1SBU

需要磁盘空间： 2.7MB

6.32.1. 安装 GDBM

配置 GDBM：

```
./configure --prefix=/usr --enable-libgdbm-compat
```

配置参数的含义如下：

--enable-libgdbm-compat

该参数将编译产生 libgdbm 兼容库，使得 LFS 外的某些程序可以使用老版本的 DBM 程序。

执行编译：

```
make
```

测试编译结果：

```
make check
```

安装软件：

```
make install
```

6.32.2. GDBM 包含的内容

安装的库： libgdbm.{so,a} and libgdbm_compat.{so,a}

简要描述

libgdbm 提供管理 hash 数据库的函数库

6.33. Inetutils-1.8

Inetutils 包含了基本网络通信需要的软件。

估计搭建时间： 0.4SBU

需要磁盘空间： 17MB

6.33.1. 安装 Inetutils

```
./configure --prefix=/usr --libexecdir=/usr/sbin \
   --localstatedir=/var --disable-ifconfig \
   --disable-logger --disable-syslogd --disable-whois \
   --disable-servers
```

配置参数的含义：

--disable-ifconfig

该参数禁止 Inetutils 安装 ifconfig 程序，ifconfig 用于配置网络接口，而 LFS 系统将使用 IPRoute2 软件包的 ip 程序完成该工作；

--disable-logger

该参数禁止 Inetutils 安装 logger 程序，某些脚本调用 logger 程序向系统日志守护进程传递消息。由于之前 Util-linux 安装了一个较新版本因而不需要再安装它；

--disable-syslogd

该参数禁止 Inetutils 安装系统日志守护进程，由以后安装的 Sysklogd 软件包安装它；

--disable-whois

该参数禁止编译 whois 客户端，该程序已过时。更好的 whois 客户端将在 BLFS 中介绍；

--disable-servers

该参数禁止安装Inetutils软件包的一些网络服务。将这些服务在基本LFS系统并不合适，其中一些服务是不安全的它们仅在可信任的网络上被认为是安全的。更多信息请查阅 *http://www.linuxfromscratch.org/blfs/view/svn/basicnet/inetutils.html*，那里有更多合适的替代选择。

执行编译：

```
make
```

测试编译结果：

```
make check
```

安装软件包：

```
make install
make -C doc html
make -C doc install-html docdir=/usr/share/doc/inetutils-1.8
```

将一些程序移动到 FHS 兼容的位置：

```
mv -v /usr/bin/{hostname,ping,ping6} /bin
mv -v /usr/bin/traceroute /sbin
```

6.33.2. Inetutils 包含的内容

安装的程序： ftp, hostname, ping, ping6, rcp, rexec, rlogin, rsh, talk, telnet, tftp, traceroute

简要描述

ftp	文件传输协议程序
hostname	报告或设置主机名称
ping	发送回送请求包并报告回复花费的时间
ping6	ping 的 IPv6 版本
rcp	执行远程文件复制的程序
rexec	在远程主机上执行命令
rlogin	执行远程登录
rsh	运行一个远程 shell
talk	用于和另一位用户聊天
telnet	Telnet 协议的接口程序
tftp	短小文件传输程序
traceroute	跟踪报文从主机到达另一网络经过的路由，并显示沿途所有的媒介网关

6.34. Perl-5.14.2

Perl 是一种直译式的编程语言。

估计搭建时间： 7.6SBU

需要磁盘空间： 235MB

6.34.1. 安装 Perl

首先创建基本的/etc/hosts 文件，Perl 配置文件和测试套件中需要用到它：

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

该版本的 Perl 会编译 Compress::Raw::Zlib 模块。默认情况下 Perl 使用的是内部 Zlib 源，键入下面的命令使 Perl 使用 Zlib 使用系统上已安装的 Zlib：

```
sed -i -e "s|BUILD_ZLIB\s*= True|BUILD_ZLIB = False|" \
   -e "s|INCLUDE\s*= ./zlib-src|INCLUDE = /usr/include|" \
   -e "s|LIB\s*= ./zlib-src|LIB = /usr/lib|" \
   cpan/Compress-Raw-Zlib/config.in
```

如果要仔细控制 Perl 的配置，就移除下面命令中的”-des”参数再精选参数控制软件包的编译方式。如果不需要精细控制，使用下面的命令进行默认配置就足够了：

```
sh Configure -des -Dprefix=/usr \
  -Dvendorprefix=/usr \
  -Dman1dir=/usr/share/man/man1 \
  -Dman3dir=/usr/share/man/man3 \
  -Dpager="/usr/bin/less -isR" \
  -Duseshrplib
```

配置选项的含义：

-Dvendorprefix=/usr

该参数决定 perl 的模块安装路径；

-Dpager="/usr/bin/less –isR"

该参数修正 perldoc 调用 less 程序时的一个错误；

-Dman1dir=/usr/share/man/man1 –Dman3dir=/usr/share/man/man3

由于没有安装 Groff，配置脚本就不再为 Perl 安装 man 帮助页面。键入这些参数使 Perl 仍旧安装 man 页面；

-Duseshrplib

该参数使 perl 编译所需的共享库。

编译软件：

```
make
```

若要测试编译结果（大约需 2.5SBU），键入命令：

```
make test
```

6.34.2. Perl 包含的内容

安装的程序： a2p, c2ph, config_data, corelist, cpan, cpan2dist, cpanp, cpanp-run-perl, dprofpp, enc2xs, find2perl, h2ph, h2xs, instmodsh, libnetcfg, perl, perl5.14.2 (指向Perl的链接), perlbug, perldoc, perlivp, perlthanks (link to perlbug), piconv, pl2pm, pod2html, pod2latex, pod2man, pod2text, pod2usage, podchecker, podselect, prove, psed (链接到s2p), pstruct (链接到c2ph), ptar, ptardiff, s2p, shasum, splain, xsubpp

安装的库： 安装的库较多，不在此处列举

安装的目录： /usr/lib/perl5

简要描述

a2p	将 awk 翻译为 perl
c2ph	模拟 cc –g –S 命令填充 C 结构
config_data	查询或改变 Perl 模块配置
corelist	CoreList 模块的命令行前缀
cpan	综合 Perl 的归档网络(CPAN, Comprehensive Perl Archive Network)的命令行交互工具
cpan2dist	CPANPLUS 发行版创建器
cpanp	启动 CPANPLUS
cpanp-run-perl	在进程写输出缓冲后，perl 脚本刷新缓冲区
dprofpp	显示 perl 配置数据
enc2xs	从 Unicode 字符集或 Tcl 编码文件的编码模块编译出的 Perl 扩展
find2perl	将 find 命令翻译为 Perl 语言
h2ph	将.h 头文件转换为.ph 的 Perl 头文件
h2xs	将.h 的 C 头文件转换为 Perl 扩展
instmodsh	用于检查安装的 Perl 模块的 Shell 脚本，它也能从一个安装的模块创建 tar 压缩包
libnetcfg	用于配置 libnet Perl 模块
perl	结合 C, sed, awk, sh 语言的最好的特性融合成 Perl 语言

perl5.14.2	指向 perl 的硬链接
perlbug	用于产生 perl 或其模块的 bug 报告，并以邮件发出
perldoc	以 pod 格式显示文档，该格式内嵌于 perl 安装树或某个 perl 脚本中
perlivp	Perl 的安装验证程序，它用来验证 Perl 和 perl 函数库是否正确安装
perlthanks	用于生成感谢消息并发送给 Perl 开发者
piconv	iconv 程序的 Perl 版本
pl2pm	将 Perl4 的.pl 文件转化为 Perl5 的.pm 模块
pod2html	将 pod 格式文件转化为 html 格式
pod2latex	将 pod 格式文件转化为 LaTeX 格式
pod2man	将 pod 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	从嵌入的 pod 文档中打印使用信息
podchecker	检查 pod 格式文档文件的语法
podselect	显示 pod 文档的选中的部分
prove	按照 Test::Harness 模块运行测试的命令行工具
psed	sed 程序的 Perl 版本
pstruct	模拟 cc –g –S 命令填充 C 结构
ptar	Perl 中的类 tar 程序
ptardiff	比较释放的归档文件和未释放的文件的不同
s2p	将 sed 脚本翻译为 Perl
shasum	打印或检查 SHA 检验和
splain	强制显示 Perl 中的冗长的警告诊断
xsubpp	将 Perl 的 XS 代码转换为 C 代码

6.36. Automake-1.11.1

Automake 程序可以为 Autoconf 生成 Makefile 文件。

估计搭建时间： 18.3SBU

需要磁盘空间： 28.8MB

6.36.1. 安装 Automake

配置 Automake：

```
./configure --prefix=/usr \
   --docdir=/usr/share/doc/automake-1.11.1
```

编译软件：

```
make
```

测试编译结果：

```
make check
```

测试过程较长，大约需要花费 10SBU。

测试完成后安装软件：

```
make install
```

6.36.2. Automake 包含的内容

安装的程序 acinstall, aclocal, aclocal-1.11.1, automake, automake-1.11.1, compile, config.guess, config.sub, depcomp, elisp-comp, install-sh, mdate-sh, missing, mkinstalldirs, pycompile, symlink-tree, ylwrap

安装的目录 /usr/share/aclocal-1.11, /usr/share/automake-1.11, /usr/share/doc/automake-1.11.1

简要描述

acinstall	一个安装 aclocal 风格 M4 文件的脚本
aclocal	基于 configure.in 文件产生 aclocal.m4 文件
aclocal-1.11.1	指向 aclocal 的硬链接
automake	一个由 Makefile.am 文件自动产生 Makefile.in 文件的工具。若要创建软件包内所有 Makefile.in 文件，就在顶级目录运行该程序。通过扫描 configure.in 文件，它自动搜索每个适合的 Makefile.am 文件并生成相应的 Makefile.in 文件
automake-1.11.1	指向 automake 的硬链接
compile	编译器的包装程序
config.guess	一个尝试猜测给定编译或主机或目标架构的目标三元组的脚本
config.sub	配置验证子程序的脚本
depcomp	该脚本使编译程序时除了产生必要的输出还输出依赖信息
elisp-comp	字节编译的 Emacs Lisp 代码
install-sh	该脚本用于安装程序，脚本或数据文件
mdate-sh	该脚本打印文件或目录的修改时间
missing	该脚本在安装时扮演 GNU 程序的通用存根的角色

mkinstalldirs	该脚本创建一颗目录树
py-compile	编译 Python 程序
symlink-tree	该脚本用于创建目录树的符号链接树
ylwrap	lex 和 yacc 程序包装

6.37. Diffutils-3.2

Diffutils 软件包内包含了比较文件和目录区别的程序。

估计搭建时间： 0.1SBU

需要磁盘空间： 6.3MB

6.37.1. 安装 Diffutils

配置 Diffutils：

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

执行编译：

```
make
```

测试编译结果：

```
make check
```

安装软件：

```
make install
```

6.37.2. Diffutils 包含的内容

安装的程序：	cmp, diff, diff3, sdiff
简要描述	
cmp	比较两个文件并报告字节级别的不同
diff	比较两个文件或目录并报告二者不同的行
diff3	按行比较三个文件
sdiff	合并两个文件并交互地输出结果

6.38. Gawk-4.0.0

Gawk 是一个处理文本文件的程序。

估计搭建时间： 0.2SBU

需要磁盘空间： 28MB

6.38.1. 安装 Gawk

配置 Gawk：

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

执行编译：

```
make
```

测试编译结果：

```
make check
```

安装软件包：

```
make install
```

如果需要，安装 Gawk 文档：

```
mkdir -v /usr/share/doc/gawk-4.0.0
cp -v doc/{awkforai.txt,*.{eps,pdf,jpg}} \
   /usr/share/doc/gawk-4.0.0
```

6.38.2. Gawk 包含的内容

安装的程序： awk (link to gawk), gawk, gawk-4.0.0, grcat, igawk, pgawk, pgawk-4.0.0, pwcat

安装的目录： /usr/lib/awk, /usr/share/awk

简要描述

awk	指向 gawk 的链接
gawk	处理文本文件的程序，是 awk 的 GNU 实现
gawk-4.0.0	指向 gawk 的硬链接
grcat	填充用户组数据库/etc/group
igawk	赋予 gawk 包含文件的能力
pgawk	gawk 的精简版本
pgawk-4.0.0	指向 pgawk 的硬链接
pwcat	填充密码数据库/etc/passwd

6.39. Findutils-4.4.2

Findutils 软件包包含了一些查找文件的程序。这些程序可以递归地搜索目录树，创建维护并搜索数据库（通常比递归查找更快，但如果数据库最近被更新过则查找结果不可靠）。

估计搭建时间： 0.5SBU

需要磁盘空间： 22MB

6.39.1. 安装 Findutils

配置 Findutils：

```
./configure --prefix=/usr --libexecdir=/usr/lib/findutils \
--localstatedir=/var/lib/locate
```

配置参数的含义：

--localstatedir

该参数将 locate 数据库的位置改为/var/lib/locate，以保持和 FHS 的兼容性。

开始编译：

```
make
```

若要测试编译结果，键入命令：

```
make check
```

安装软件包：

```
make install
```

LFS-Bootscripts 软件包中的某些脚本依赖于 find 程序。由于系统启动的早期/usr 目录可能并不存在，就要求该程序位于根分区。另外，还需要修改 updatedb 脚本中的路径设置：

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

6.39.2. Findutils 包含的内容

安装的程序： bigram, code, find, frcode, locate, oldfind, updatedb, xargs

安装的目录： /usr/lib/findutils

简要描述

bigram 以前用来生成 locate 数据库文件

code 以前用来生成 locate 数据库文件，它是 frcode 的老版本

find 搜索目录树中符合给定条件的文件

frcode updatedb 程序调用它来压缩文件名列表，它使用前段压缩控制参数为 4~5 减小数据库的大小

locate 搜索文件名数据库并报告包含给定字符串或符合给定模式的文件名

oldfind find 程序的较旧版本，它使用的算法和 find 不同

updatedb 更新 locate 数据库，它扫描整个文件系统（除了显示指出例外文件系统，默认包括当前挂载的其他文件系统）并将搜索到的每个文件名放入数据库

xargs	将给定命令应用于一系列文件

6.40. Flex-2.5.35

Flex 包含一个工具集，用于产生能识别文本中的模式的程序。

估计搭建时间： 0.7SBU

需要磁盘空间： 28MB

6.40.1. 安装 Flex

向 C++扫描产生器应用补丁修复一个 bug，该 bug 会导致扫描器使用 GCC-4.6.1 编译失败：

```
patch -Np1 -i ../flex-2.5.35-gcc44-1.patch
```

配置 Flex:

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

执行编译：

```
make
```

测试编译结果（大约需要 0.5SBU）：

```
make check
```

安装软件包：

```
make install
```

由于某些软件需要/usr/lib 中的 lex 库，所以需要创建一个符号链接：

```
ln -sv libfl.a /usr/lib/libl.a
```

一些程序不会调用 flex 而使用的是它的老版本 lex。为了支持这些程序，创建一个包裹脚本 lex，在该脚本中调用 flex：

```
cat > /usr/bin/lex << "EOF"
#!/bin/sh
# Begin /usr/bin/lex
exec /usr/bin/flex -l "$@"
# End /usr/bin/lex
EOF
chmod -v 755 /usr/bin/lex
```

如果需要，安装 flex.pdf 文档：

```
mkdir -v /usr/share/doc/flex-2.5.35
cp -v doc/flex.pdf \
/usr/share/doc/flex-2.5.35
```

6.40.2. Flex 包含的内容

安装的程序： flex, lex

安装的库： libfl.a , libfl_pic.a

简要描述

flex 用于产生能识别文本中模式的程序，它允许指定多种查找模式规则，避免了为满足查找模式而开发特定程序

lex 以 lex 模拟模式运行 flex 的脚本

libfl.a flex 函数库

6.41. Gettext-0.18.1.1

Gettext 包含了一些国际化和本地化的工具。这些工具使得程序可以编译为支持本地语言(Native Language Support, NLS)，并让程序以本地语言的形式输出。

估计搭建时间： 5.8SBU

需要磁盘空间： 125MB

6.41.1. 安装 Gettext

配置 Gettext：

```
./configure --prefix=/usr \
   --docdir=/usr/share/doc/gettext-0.18.1.1
```

执行编译：

```
make
```

如果要测试编译结果（大概需要花费 3SBU），键入命令：

```
make check
```

安装软件包：

```
make install
```

6.41.2. Gettext 包含的内容

安装的程序： autopoint, config.charset, config.rpath, envsubst, gettext, gettext.sh, gettextize,
hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec,
msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, recodesr-latin, xgettext

安装的库： libasprintf.{a,so}, libgettextlib.so, libgettextpo.{a,so}, libgettextsrc.so, preloadable_libintl.so

安装的目录： /usr/lib/gettext, /usr/share/doc/gettext-0.18.1.1, /usr/share/gettext

简要描述

autopoint	将标准 Gettext 结构的文件复制到源码包
config.charset	输出字符编码别名的系统依赖表
config.rpath	输出系统依赖的变量集，在可执行文件头中描述怎样设置运行时共享库的搜索路径
envsubst	替换 shell 格式字符串的环境变量
gettext	查询消息目录，将自然语言的消息翻译成用户语言
gettext.sh	充当 gettext 的 shell 功能库
gettextize	将所有标准 Gettext 文件复制到软件包的顶级目录以便于开始进行国际化
hostname	以不同形式显示网络主机名
msgattrib	根据消息的属性过滤翻译目录的消息，并对属性进行管理
msgcat	连接合并给定.po 文件

msgcmp	比较两个.po 文件检查二者是否包含同样的 msgid 字符串集合
msgcomm	查找给定.po 文件的通用消息
msgconv	将一个翻译目录转换为不同的字符编码
msgen	创建英语翻译目录
msgexec	将命令应用到翻译目录的全部翻译
msgfilter	将筛选条件应用到翻译目录的全部翻译
msgfmt	从翻译目录生成二进制消息目录
msggrep	释放匹配给定模式或属于同一给定源文件的翻译目录的全部消息
msginit	创建新的.po 文件，以用户环境中的值初始化元数据
msgmerge	将两个原始翻译组合到同一个文件
msgunfmt	将二进制消息目录反编译为原始翻译文本
msguniq	统一同一翻译目录的重复翻译
ngettext	将文本消息显示为本地化语言，该文本消息语法形式依赖于一个数字
recode-sr-latin	将 Serbian 文本从 Cyrillic 改为 Latin 脚本再记录
xgettext	释放源文件的可翻译的消息行作为第一个翻译模版
libasprintf	定义 autosprintf 类使用<string>中的字符串和<iostream>中的流，使得 C 格式化的输出例程能工作在 C++程序中
libgettextlib	一个私有库，它包含了不同 Gettext 程序使用的通用的例程，而原本它们不是通用的
libgettextpo	用于书写处理.po 文件的特定程序，当标准应用自带的 Gettext(如 msgcomm, msgcmp, msgattrib, msgen)不满足要求时才使用该库
libgettextsrc	一个私有库，它包含了不同 Gettext 程序使用的通用的例程，而原本它们不是通用的
preloadable_libintl	该库主要用于支持 LD_PRELOAD，而 LD_PRELAOD 用于在登录时未翻译消息时支持 libintl

6.42. Groff-1.21

Groff 用于处理并格式化文本。

估计搭建时间： 0.4SBU

需要磁盘空间： 78MB

6.42.1. 安装 Groff

Groff 程序会从环境变量 PAGE 中获取默认的页大小。对于美国地区的用户而言，PAGE=letter 是合理的。但对其他地方 PAGE=A4 可能才是合理的值。尽管默认页面大小是在配置编译时确定的，但可以在以后修改/etc/papersize 文件将值修改为“A4”或“letter”。

配置：

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

执行编译：

```
make
```

Groff 软件包不含测试套件。

安装软件包：

```
make install
```

若要使 xman 之类的文档程序正常工作，就需要建立下列符号链接：

```
ln -sv eqn /usr/bin/geqn
ln -sv tbl /usr/bin/gtbl
```

6.42.2. Groff 包含的内容

安装的程序： addftinfo, afmtodit, chem, eqn, eqn2graph, gdiffmk, geqn (link to eqn), grap2graph, grn, grodvi, groff, groffer, grog, grolbp, grolj4, grops, grotty, gtbl (link to tbl), hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pdfroff, pfbtops, pic, pic2graph, postgrohtml, preconv, pre-grohtml, refer, roff2dvi, roff2html, roff2pdf, roff2ps, roff2text, roff2x, soelim, tbl, tfmtodit, troff

安装的目录： /usr/lib/groff, /usr/share/doc/groff-1.21, /usr/share/groff

简要描述

addftinfo	读取 troff 字体文件并增加 groff 系统需要的额外的字体尺寸信息
afmtodit	创建 groff 和 grops 程序需要使用的字体文件
chem	生成化学结构图表的 groff 预处理器
eqn	将 troff 输入文件中嵌入等式的描述编译为 troff 程序能够理解的命令
eqn2graph	将 troff 等式转换为裁切的图片
gdiffmk	标记 groff/nroff/troff 文件的不同
geqn	指向 eqn 的链接
grap2gragh	将方格图转换为裁切的位图
grn	小文件的 groff 预处理器
grodvi	生成 TeX dvi 格式文件的 groff 驱动
groff	groff 文档格式系统的前端，为选定设备执行 troff 程序和一个后处理

器

groffer 在 X 和 tty 终端显示 groff 文件和 man 页面

grog 读取文件并为打印文件猜测选择合适的 groff 参数：-e, -man, -me, -mm, -ms, -p, -s, -t，最后给出包含这些参数的 groff 命令

grolbp 佳能 CAPSL 打印机（LBP-4 和 LBP-8 系列激光打印机）的 groff 驱动

grolj4 groff 的驱动程序，为 HP LaserJet4 打印机生成合适的 PLC5 格式输出

grops 将 GNU troff 的输出翻译为 PostScript

grotty 将 GNU troff 的输出翻译为适合类似打字机等设备的形式

gtbl 指向 tbl 的链接

hpftodit 为了满足 groff-Tlj4 程序的要求，使用 HP 标记字体大小的文件创建新的字体文件

indxbib 利用文献数据库的特定文件创建逆序索引，满足 refer， lookbib，lkbib 等程序的要求

lkbib 在标准错误输出中打印提示（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，搜索文献数据库文件是否包含这些关键字，并在标准输出打印符合的引用，然后重复这个过程直到输入末端

mmroff groff 的简单预处理器

neqn 格式化方程组为 ASCII 输出

nroff 一个使用 groff 模拟 nroff 命令的脚本

pdfroff 使用 groff 创建 pdf 文档

pfbtops 将.pfb 格式的 PostScript 翻译为 ASCII 码

pic 将 troff 或 TeX 文件内嵌入的图片描述编译为 T 额 X 或 troff 能够理解的命令

pic2graph 将 PIC 图表转换为裁切的图片

post-grohtml 将 GNU troff 输出翻译为 HTML

preconv 将输入文件编码转换成 GNU troff 能理解的内容

pre-grohtml 将 GNU troff 输出转换为 HTML

refer 将文件内容复制到标准输出，复制时不包含".["与".]"之间的行，它们表示引用，也不复制在".R1"和".R2"之间的行，它们表示引用内容是如何执行的

roff2dvi 将 roff 文件转换为 DVI 格式

roff2html 将 roff 文件转换为 HTML 格式

roff2pdf 将 roff 文件转换为 PDF 格式

roff2ps 将 roff 文件转换为 ps 文件

roff2text 将 roff 文件转换为文本文件

roff2x 将 roff 文件转换为其他格式

soelim 读取文件并将.*sofile* 形式的行替换成已知的 *file* 的内容

tbl 将 troff 输入文件中嵌入的表格描述编译成 troff 能理解的命令

tfmtodit 为 groff-Tdvi 程序创建字体文件

troff 和 Unix troff 程序高度兼容，通常使用 groff 命令调用它，在调用时还会携带合适的参数以合适的顺序运行预处理和后处理器

6.43. GRUB-1.99

估计搭建时间： 0.6SBU
需要磁盘空间： 76MB

6.43.1. 安装 GRUB

配置 GRUB：

```
./configure --prefix=/usr \
  --sysconfdir=/etc \
  --disable-grub-emu-usb \
  --disable-efiemu \
  --disable-werror
```

参数—disable 将在编译时关闭某些 LFS 不需要的特性和测试程序，从而最小化编译出的程序。

执行编译：

```
make
```

GRUB 软件包没有测试套件。

安装软件：

```
make install
```

怎样使用 GRUB 令 LFS 系统可启动呢？我们将在 8.4 节“使用 GRUB 配置启动进程”里具体讲述。

6.43.2. GRUB 的内容

安装的程序： grub-bin2h, grub-editenv, grub-install, grub-mkconfig, grub-mkdevicemap, grubmkelfimage, grub-mkimage, grub-mkisofs, grub-mkpasswd-pbkdf2, grub-mkrelpath, grub-mkrescue, grub-probe, grub-reboot, grub-script-check, grub-set-default, grubsetup

安装的目录： /usr/lib/grub, /etc/grub.d, /usr/share/grub

简要描述

grub-bin2h	将二进制文件转换为 C 头文件
grub-editenv	编辑环境块的工具
grub-install	将 GRUB 安装到驱动器
grub-mkconfig	创建 grub 配置文件
grub-mkdevicemap	自动生成设备信息图文件
grub-mkelfimage	生成 GRUB 的可启动映像文件
grub-mkimage	生成 GRUB 的可启动映像文件
grub-mkisofs	常见可启动的 ISO 映像文件
grub-mkpasswd-pbkdf2	在启动菜单中创建一份 PBKDF2 加密的密码
grub-mkrelpath	将系统路径名和它的根相关联
grub-mkrescue	使 GRUB 的可启动映像能匹配软盘或 CDROM/DVD
grub-probe	探测给定路径或设备的设备信息

grub-reboot	仅为下一次启动设置默认的 GRUB 启动入口
grub-script-check	检查 GRUB 配置脚本的语法错误
grub-set-default	设置 GRUB 的默认启动入口
grub-setup	设置来自设备的映像可启动

6.44. Gzip-1.4

Gzip 是一个压缩和解压缩的程序。

估计搭建时间： 小于 0.1SBU

需要磁盘空间： 3.3MB

6.44.1. 安装 Gzip

配置 Gzip:

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

执行编译：

```
make
```

若要测试编译结果，键入命令：

```
make check
```

安装软件包：

```
make install
```

部分程序无须位于根文件系统，故将这些程序移动到其他位置：

```
mv -v /bin/{gzexe,uncompress,zcmp,zdiff,zegrep} /usr/bin
mv -v /bin/{zfgrep,zforce,zgrep,zless,zmore,znew} /usr/bin
```

6.44.2. Gzip 包含的内容

安装的程序： gunzip, gzexe, gzip, uncompress, zcat, zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore, znew

简要描述

gunzip	解压缩 gzip 文件
gzexe	创建自解压的可执行文件
gzip	使用 Lempel-Ziv(LZ77)编码压缩给定文件
uncompress	解压缩文件
zcat	将给定 gzip 文件解压缩至标准输出
zcmp	对 gzip 文件执行 cmp 命令
zdiff	对 gzip 文件执行 diff 命令
zegrep	对 gzip 文件执行 egrep 命令
zfgrep	对 gzip 文件执行 fgrep 命令
zforce	为所有 gzip 压缩的文件强制使用.gz 扩展名，这样 gzip 文件就不会重复压缩这些文件，在文件传输中文件名被截断时该程序就很有用了
zgrep	对 gzip 文件执行 grep 命令
zless	对 gzip 文件执行 less 命令
zmore	对 gzip 文件执行 more 命令
znew	将 compress 压缩格式的文件重新压缩为 gzip 格式，即.Z->.gz

6.45. IPRoute2-2.6.39

IPRoute2 软件包含了 IPv4 的基本和高级网络工具。

估计搭建时间： 0.1SBU

需要磁盘空间： 6.6MB

6.45.1. 安装 IPRoute2

该软件包中的arpd程序依赖于Berkeley DB。而对于基本Linux系统，arpd程序并不是必须的，故而使用下面的sed命令移除对Berkeley DB的依赖。如果确实需要arpd，可以查阅BLFS（*http://www.linuxfromscratch.org/blfs/view/svn/server/databases.html#db*）获得编译Berkeley DB的详细指导。

```
sed -i '/^TARGETS/s@arpd@@g' misc/Makefile
```

编译软件：

```
make DESTDIR=
```

make 参数的含义：

DESTDIR=

该参数使 IPRoute2 能够被安装到正确的目录。若不使用该参数，默认 DESTDIR 值为/usr。

该软件包含有一个测试套件，但由于我们修改了编译过程，所以在非 chroot 环境中测试结果并不可靠。如果要在进入新 LFS 系统后运行测试套件，请确保选择/proc/config.gz CONFIG_IKCONFIG_PROC(“General setup”->”Enable access to .config through /proc/config.gz”)支持所选的内核，然后从 testsuite/子目录中运行”make alltests”。

安装软件：

```
make DESTDIR= SBINDIR=/sbin MANDIR=/usr/share/man \
   DOCDIR=/usr/share/doc/iproute2-2.6.39 install
```

6.45.2. IPRounte2 包含的内容

安装的程序： ctstat (链接到lnstat), genl, ifcfg, ifstat, ip, lnstat, nstat, routef, routel, rtacct, rtmon, rtpr, rtstat (链接到lnstat), ss, tc

安装的目录： /etc/iproute2, /lib/tc, /usr/share/doc/iproute2-2.6.39, /usr/lib/tc

简要描述

ctstate 连接状态工具

genl

ifcfg 包裹命令 ip 的一个 shell 脚本。注意它依赖于 ip 工具包（*http://www.skbuff.net/iputils/*）中的 arping 和 rdisk 程序

ifstat 显示接口统计值，包括通过接口发送和接收的包的数量

ip 主程序。它包含多个功能：

ip link <device> 查看设备状态并允许修改

ip addr 查看地址和属性，新增地址删除旧地址

ip neighbor 查看绑定的网上邻居及其属性，新增邻居，删除旧邻居

ip rule 查看并修改路由策略

ip route 查看路由表并修改路由表规则

ip tunnel 查看并修改 ip 隧道及其属性
ip maddr 查看并修改多播地址及属性
ip mroute 设置，修改或删除多播路由
ip monitor 不断监视设备状态，地址和路由

lnstat 提供 Linux 网络统计数据。它是 rtstat 程序的更通用更优异的替代版本

nstat 显示网络统计数据

routef ip route 的组件，用于刷新路由表

routel ip route 的组件，用于罗列路由表

rtacct 显示/proc/net/rt_acct 的内容

rtmon 路由监视工具

rtpr 将 ip –o 的输出转换为可读的形式

rtstat 路由状态工具

ss 类似 netstat 命令，显示激活的连接

tc 交通控制程序，它是质量服务(Quality Of Service, QOS)和类服务(Class Of Service, COS)的实现：

tc qdisc 设置队列规则
tc class 设置基于队列规则任务的类
tc estimator 预估进入网络的流量
tc filter 设置 QOS/COS 包过滤
tc policy 设置 QOS/COS 策略

6.46. Kbd-1.15.2

Kbd 软件包含了一系列键码表文件和键盘工具。

估计搭建时间： 小于 0.1SBU

需要磁盘空间： 16.0MB

6.46.1. 安装 Kbd

Kbd 包中的 Backspace 键和 Delete 键行为不可靠，使用下面的补丁修复 i386 键位图的问题：

```
patch -Np1 -i ../kbd-1.15.2-backspace-1.patch
```

打完补丁后，Backspace 键产生的字符码值为 127，Delete 键是反斜杠序列。

配置 Kbd：

```
./configure --prefix=/usr --datadir=/lib/kbd
```

配置参数的含义：

--datadir=/lib/kbd

该参数使得键盘布局数据被放置在根分区而不是默认的/usr/share/kbd。

编译：

```
make
```

该软件包不含测试套件。

安装软件包：

```
make install
```

Note

对某些语言（如白俄罗斯语），Kbd 没有提供有用的键码图，并假定使用 ISO-8859-5 编码，使用 CP1251 键码图。使用这些语言的用户需要自己下载合适的键码图。

LFS-Bootscripts 中的某些脚本依赖于 kbd_mode, loadkeys, openvt 和 setfont。由于系统启动早期/usr 可能并未就绪，故而这些程序都需要被放在根分区：

```
mv -v /usr/bin/{kbd_mode,loadkeys,openvt,setfont} /bin
```

如果需要，安装 kbd 文档：

```
mkdir -v /usr/share/doc/kbd-1.15.2
cp -R -v doc/* \
   /usr/share/doc/kbd-1.15.2
```

6.46.2. Kbd 包含的内容

安装的程序： chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (link to psfxtable), psfgettable (链接到 psfxtable), psfstriptable (链接到psfxtable), psfxtable, resizecons, setfont, setkeycodes, setleds, setmetamode, showconsolefont, showkey, unicode_start, unicode_stop

安装的目录： /lib/kbd

简要描述

chvt	改变虚拟终端前景色
deallocvt	释放未使用的虚拟终端
dumpkeys	填充键盘转换表
fgconsole	打印激活的虚拟终端数
getkeycodes	打印内核扫描码-键码映射表
kbd_mode	报告或设置键盘模式
kbdrate	设置键盘重复和延迟率
loadkeys	加载键盘转换表
loadunimap	加载内核 unicode-字体映射表
mapscrn	已过时，用于加载用户定义的输出字符映射表到控制台驱动，现在由 setfont 实现
openvt	在虚拟终端运行某个程序
psfaddtable	链接到 psfxtable
psfgettable	链接到 psfxtable
psfstriptable	链接到 psfxtable
psfxtable	为控制台字体处理 Unicode 字符表
resizecons	改变控制台大小
setfont	改变高级图形适配器(EGA)和图像图形队列(VGA)在控制台的字体
setkeycodes	加载内核扫描码—键码映射表入口，这在键盘上含有未使用的键时很有用
setleds	设置键盘标志和 LED 灯
setmetamode	定义键盘元键控制
showconsolefont	显示当前 EGA/VGA 控制屏字体
showkey	报告键盘当前按下键的扫描码，键码，ASCII 码
unicode_start	将键盘和控制台设为 UNICODE 模式。除非键盘映射文件是 ISO-8859-1 编码否则不予使用该程序。对其他编码，该程序将产生错误结果
unicode_stop	将键盘和控制台设为非 UNICODE 模式

6.47. Less-444

Less 是文本查看程序。

估计搭建时间： 小于 0.1SBU

需要磁盘空间： 3.5MB

6.47.1. 安装 Less

配置：

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

配置参数的含义：

--sysconfdir=/etc

该参数使得程序在/etc 中查找配置文件。

执行编译：

```
make
```

本软件不含测试套件。

安装：

```
make install
```

6.47.2. Less 包含的内容

安装的程序：	less, lessecho, lesskey
简要描述	
less	文件浏览器或分页器，显示给定文件，运行用户滚动，查找字符串或调转到指定标记
lessecho	用于展开元字符，比如*, ?
lesskey	用于执行 less 的绑定键

6.48. Libpipeline-1.2.0

Libepipeline 包含了一个函数库，该库以灵活方便的方式维护子进程的管道行。

估计搭建时间： 0.1SBU

需要磁盘空间： 8.0MB

6.48.1. 安装 Libpipeline

配置：

```
./configure CHECK_CFLAGS=-I/tools/include \
   CHECK_LIBS="-L/tools/lib -lcheck" --prefix=/usr
```

配置参数的含义：

CHECK-CFLAGS=, CHECK_LIBS=

这些环境变量指定了 5.14 节“Check-0.98”编译的测试库的位置。

执行编译：

```
make
```

若要运行测试套件，执行命令：

```
make check
```

安装软件：

```
make install
```

6.48.2. Libpipeline 的包含的内容

安装的库： libpipeline.so

简要描述

libpipeline 该库用于在子进程间安全地创建管道

6.49. Make-3.82

Make 程序用于编译软件。

估计搭建时间： 0.3SBU

需要磁盘空间： 9.7MB

6.49.1. 安装 Make

开始配置：

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

执行编译：

```
make
```

若要测试编译结果，键入命令：

```
make check
```

安装软件：

```
make install
```

6.49.2. Make 包含的内容

安装的程序： make

简要描述

make 自动判断哪些软件需要（重新）编译并可执行相关命令

6.50. xz-5.0.3

Xz 用于压缩和解压缩文件。它对 lzma 和新的 xz 压缩格式提供支持。xz 压缩的文本文件比 gzip 或 bzip2 有更好的压缩比率。

估计搭建时间： 0.4SBU

需要磁盘空间： 13MB

6.50.1. 安装 Xz

开始配置：

```
./configure --prefix=/usr --docdir=/usr/share/doc/xz-5.0.3
```

执行编译：

```
make
```

若要测试编译结果，运行：

```
make check
```

安装软件：

```
make install
```

6.50.2. Xz 包含的内容

安装的程序： lzcat (链接到xz), lzcmp (链接到xzdiff), lzdiff (链接到xzdiff), bzdiff, lzegrep (链接到xzgrep), lzfgrep 链接到xzgrep), lz (链接到 xz), lzmadec, lzmainfo, lzmore (链接到xzmore), unlzma (链接到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, xzmore

安装的库： liblzma.{a,so}

安装的目录： /usr/include/lzma and /usr/share/doc/xz-5.0.3

lzcat	解压缩到标准输出
lzcmp	在 LZMA 压缩的文件上执行 cmp 命令
lzdiff	在 LZMA 压缩的文件上执行 diff 命令
lzegrep	在 LZMA 压缩的文件上执行 egrep 命令
lzfgrep	在 LZMA 压缩的文件上执行 grep 命令
lzgrep	在 LZMA 压缩的文件上执行 grep 命令
lzless	在 LZMA 压缩的文件上执行 less 命令
lzma	使用 LZMA 格式压缩或解压缩文件
lzmadec	LZMA 压缩文件的小型快速解码器
lzmainfo	显示存储在 LZMA 压缩文件头的信息
lzmore	在 LZMA 压缩的文件上执行 more 命令
unlzma	使用 LZMA 格式解压缩文件
unxz	使用 XZ 格式解压缩文件
xz	使用 XZ 格式压缩或解压缩文件
xzcat	解压缩到标准输出
xzcmp	在 XZ 压缩的文件上执行 cmp 命令

xzdec	XZ 压缩文件的小型快速的解码器
xzdiff	在 XZ 压缩的文件上执行 diff 命令
xzegrep	在 XZ 压缩的文件上执行 egrep 命令
xzfgrep	在 XZ 压缩的文件上执行 fgrep 命令
xzgrep	在 XZ 压缩的文件上执行 grep 命令
xzless	在 XZ 压缩的文件上执行 less 命令
xzmore	在 XZ 压缩的文件上执行 more 命令
liblzma*	该库使用 Lempel-Ziv-Markov 链式算法，实现了低损失率，块排序数据压缩

6.51. Man-DB-2.6.0.2

Man-DB 软件包内的程序用于查询和查看 man 页面。

估计搭建时间： 0.4SBU

需要磁盘空间： 22MB

6.51.1. 安装 Man-DB

开始配置：

```
PKG_CONFIG=/tools/bin/true \
libpipeline_CFLAGS='' \
libpipeline_LIBS='-lpipeline' \
./configure --prefix=/usr --libexecdir=/usr/lib \
--docdir=/usr/share/doc/man-db-2.6.0.2 --sysconfdir=/etc \
--disable-setuid --with-browser=/usr/bin/lynx \
--with-vgrind=/usr/bin/vgrind --with-grap=/usr/bin/grap
```

配置参数的含义如下：

PKG_CONFIG=, libpipeline_...

使用这些环境变量能够使配置进程能在不包含外部 pkg-config 程序的情况下完成配置。

--disable-setuid

该参数将 man 程序的设置用户 id（setuid）设为 man 用户。

--with-...

这三个参数用于设置一些默认程序。lynx 是一个基于文本的 web 浏览器（请查阅 BLFS 的安装指导），vgrind 将程序源转换为 Groff 输入，grap 在 Groff 文档的图形排版时很有用。通常 vgrind 和 grap 程序在查看 man 手册时不是必要的。它们不是 LFS 或 BLFS 的必要部分，但如果你愿意可以在完成整个 LFS 系统后再安装。

执行编译：

```
make
```

若要测试编译结果，键入命令：

```
make check
```

安装软件：

```
make install
```

6.51.2. LFS 中的非英语 man 手册

表 6.1 展示了 Man-DB 手册假定安装在/usr/share/man/<ll>目录下的编码字符集。除此之外，如果在该目录下是 UTF-8 编码则 Man-DB 可以正确地识别。

表 6.1 传统 8 位 man 页面认可字符集

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
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

Man 手册不支持表格外的语言。

6.51.3. Man-DB 包含的内容

安装的程序： accessdb, apropos (链接到whatis), catman, lexgrog, man, mandb, manpath, whatis, zsoelim

安装的目录： /usr/lib/man-db, /usr/share/doc/man-db

简要描述

accessdb	以人类可读的形式填充 whatis 数据库
apropos	搜索 whatis 数据库并显示包含给定字符串的系统命令
catman	创建或更新预格式化 man 手册页面
lexgrog	显示给定 man 手册页面的在线摘要信息
man	格式化显示请求的 man 页面
mandb	创建或更新 whatis 数据库

manpath	基于 man.conf 和用户环境的设置显示$MANPATH 或（若未设置 $MANPATH）合适的搜索路径的内容
whatis	搜索 whatis 数据库并显示包含独立关键字的系统命令的简要描述
zsoelim	读取文件并以之前 *file* 的内容替换.*so file* 形式的行

6.52. Module-Init-Tools-3.16

Module-Init-Tools 用于处理内核中大于等于 2.5.47 的模块。

估计搭建时间： 0.1SBU

需要磁盘空间： 8.6MB

6.52.1. 安装 Module-Init-Tools

应用补丁补充发布源码包中缺失的 man 手册页面：

```
patch -Np1 -i ../module-init-tools-3.16-man_pages-1.patch
```

本软件的测试套件由它的维护器在需要时启动。make check 命令会创建一个特殊的 modprobe 包装版本，但正常的操作一般是用不到该程序的。若要此时运行（需要约 0.2SBU），键入下述命令（注意，在重新编译或正常使用前需要使用 make clean 命令清理源码树）：

```
DOCBOOKTOMAN=/bin/true ./configure
make check
sed -i -e 's@../../configure@DOCBOOKTOMAN=/bin/true &@'
tests/runtests
./tests/runtests
make clean
```

开始配置：

```
DOCBOOKTOMAN=/bin/true ./configure --prefix=/ \
   --enable-zlib-dynamic --mandir=/usr/share/man
```

执行编译：

```
make
```

安装软件：

```
make INSTALL=install install
```

make 参数的含义如下：

INSTALL=install

通常，make install 命令不会安装已存在的二进制文件，该参数覆盖默认的脚本调用 install 强制安装已有二进制文件。

6.52.2. Module-Init-Tools 包含的内容

安装的程序： depmod, insmod, insmod.static, lsmod, modinfo, modprobe, rmmod

简要描述

depmod	基于找到的已有模块集的符号创建依赖文件，modprobe 程序会使用该依赖文件自动加载需要的模块
insmod	为运行中的内核安装可加载模块
insmod.static	insmod 的静态编译版本
lsmod	列出当前加载的模块
modinfo	检查和内核模块相关的对象文件，并显示它收集的全部信息

modprobe	利用 depmod 创建的依赖文件自动加载相关模块
rmmod	从运行中的内核中卸载模块

6.53. Path-2.6.1

Path 程序用于为文件应用尤其是 diff 程序创建的补丁文件。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 1.9MB

6.53.1. 安装 Path

由于测试套件中某个测试依赖于 ed，而我们尚未安装该编辑器，应用补丁禁止该测试：

```
patch -Np1 -i ../patch-2.6.1-test_fix-1.patch
```

开始配置：

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

执行编译：

```
make
```

若要运行测试套件，键入命令：

```
make check
```

安装软件：

```
make install
```

6.53.2. Path 包含的内容

安装的程序： path

简要描述

path 根据补丁文件修改文件。补丁文件通常是用 diff 程序比较文件区别时创建的，通过为原文件应用这些不同，path 程序就创建了打过补丁的版本

6.54. Psmisc-22.14

Psmisc 软件包中的程序用于显示运行进程的信息。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 3.6MB

6.54.1. 安装 Psmisc

开始配置：

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

执行编译：

```
make
```

该软件包不含测试套件。

安装软件：

```
make install
```

最后，按照 FHS 规范将 killall 和 fuser 程序移动到指定位置：

```
mv -v /usr/bin/fuser /bin
mv -v /usr/bin/killall /bin
```

6.54.2. Psmisc 包含的内容

安装的程序：	fuser, killall, peekfd, prtstat, pstree, and pstree.x11 (链接到 pstree)
简要描述	
fuser	报告正在使用给定文件或文件系统的进程 ID
killall	终止进程，它向运行给定命令的全部进程发送信号
peekfd	按进程 ID 查看进程文件描述符
prtstat	打印进程的信息
pstree	以树形结构显示运行的进程
pstree.x11	类似 pstree，但它会等待确认再退出

6.55. Shadow-4.1.4.3

Shadow 以一种安全的方式处理密码。

估计搭建时间： 0.3SBU

需要磁盘空间： 30MB

6.55.1. 安装 Shadow

若需要强制使用强密码，在安装Shadow前请查阅 *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/cracklib.html*并按照CrackLib，而且在下面的配置命令中增加参数*--with-libcrack*。

由于 Coreutils 包提供了更好的 groups 程序，故这里不再安装 groups 及其 man 手册：

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

修复安装俄语 man 手册的一个 bug：

```
sed -i 's/man_MANS = $(man_nopam) /man_MANS = /' \
   man/ru/Makefile.in
```

加密时不使用默认的 *crypt* 方法，而使用更安全的 *SHA-512* 方法，该方法还支持多于 8 字符的密码。本软件还使用了用户邮箱目录，故需要将过时的位置/var/spool/mail 修改为/var/mail：

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

如果你编译前安装了 Cracklib，则需要执行不同的修改命令：

```
sed -i 's@DICTPATH.*@DICTPATH\t/lib/cracklib/pw_dict@' \
etc/login.defs
```

开始配置：

```
./configure --sysconfdir=/etc
```

执行编译：

```
make
```

该软件包不含测试套件。

安装软件：

```
make install
```

调整 passwd 程序到正确的位置：

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

6.55.2. 配置 Shadow

Shadow 软件包中包含了一系列工具用于增加，修改和删除用户和用户组，设置和更改其密码，以及执行其他一些管理任务。若要深入了解什么叫阴影密码机制，请查阅未打包的源码树的文档 doc/HOWTO。如果使用了阴影机制，要确保需要验证密码（显示管理者，FTP 程序，pop3 守护进程等）的程序必须兼容阴影机制。即，它们需要能够识别阴影密码。

若要使用阴影密码，执行命令：

```
pwconv
```

若要支持阴影组密码，执行命令：

```
grpconv
```

Shadow 对 useradd 工具的默认配置有几点需要说明。首先，useradd 的默认操作是创建用户和用户组，且用户组使用和用户相同的名称，另外默认情况下用户 ID(UID)和用户组 ID(GID)从 1000 开始。这就意味着如果使用无参数的 useradd，每个用户会属于一个唯一组。如果不希望发生这种情况，就需要给 useradd 命令床底-g 参数。useradd 的默认参数存放在 /etc/default/useradd 文件中。如果需要就修改改文件中的这两个参数。

/etc/default/useradd 的参数解释

GROUP=1000

该参数设置/etc/group 文件中的用户组号起始值。你可以将该参数修改为任意值。注意 useradd 绝不会重用 UID 或 GID。如果该参数中的标识符数字被使用了，系统会自动向后取用未使用的数字。另外，如果系统中不存在组 ID 为 1000 的组，第一次使用 useradd 命令且不带-g 参数时，会在终端上显示：*useradd: unknown GID 1000*。此时忽略该信息即可，系统会使用 1000 作为组 ID。

CREATE_MAIL_SPOOL=yes

该参数使 useradd 命令创建用户时为该新用户创建一个邮箱文件。并且 useradd 会将该文件的拥有组设为 mail 组（组权限为 0660）。如果不想 useradd 创建这些邮箱文件，使用下面的命令来禁止该行为：

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

6.55.3. 设置 root 密码

设置 root 用户的密码：

```
passwd root
```

6.55.4. Shadow 包含的内容

安装的程序： chage, chfn, chgpasswd, chpasswd, chsh, expiry, faillog, gpasswd, groupadd, groupdel, groupmems, groupmod, grpck, grpconv, grpunconv, lastlog, login, logoutd, newgrp, newusers, nologin, passwd, pwck, pwconv, pwunconv, sg (链接到newgrp), su, useradd, userdel, usermod, vigr (链接到 vipw), vipw

安装的目录： /etc/default

简要描述

chage	用于改变强制修改密码间隔天数的最大值
chfn	用于改变用户全名和其他信息
chgpasswd	以批处理模式改变用户组密码
chpasswd	以批处理模式改变用户密码
chsh	更改用户默认登录 shell
expiry	检查并强制执行当前密码过期策略
faillog	用于检查登录失败日志，设置账户冻结前最大失败登录次数或者重置失败次数
gpasswd	用于添加或删除组的成员和管理员
groupadd	以给定名称创建组
groupdel	删除指定名称的组
groupmems	允许用户管理其组成员而不必拥有超级用户权限
groupmod	用于修改给定用户组名或组 ID
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更新阴影组文件
grpunconv	从/etc/gshadow 文件更新/etc/group，然后删除前者
lastlog	报告所有用户中最近登陆的用户或某用户的最近登陆
login	用户登录
logoutd	是一个守护进程，用于强制登陆时间和端口的限制
newgrp	在一个登陆会话中改变当前 GID
newusers	用于创建或更新用户账户的全部序列
nologin	显示不可用的账户信息。该程序被当作禁用账户的默认登陆 shell
passwd	用于改变用户或用户组的密码
pwck	验证密码文件/etc/passwd 和/etc/shadow 的完整性
pwconv	从普通密码文件创建或更新阴影密码文件
pwunconv	从/etc/shadow 更新/etc/passwd 并删除/etc/shadow
sg	当用户的 GID 被设定为指定组并运行给定命令
su	以替代用户和组 ID 运行 shell
useradd	以指定名称创建新用户或更新新用户的默认信息
userdel	用于修改给定用户的登陆名，用户 ID(UID)，shell，初始化组，家目录等
vigr	编辑/etc/group 或/etc/gshadow 文件
vipw	编辑/etc/passwd 或/etc/shadow 文件

6.56. Sysklogd-1.5

Sysklogd 软件包内的程序用于记录系统信息，比如发生非正常事件后内核发出的信息。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 0.5MB

6.56.1. 安装 Sysklogd

执行编译：

```
make
```

该软件包不含测试套件。

安装软件包：

```
make BINDIR=/sbin install
```

6.56.2. 配置 Sysklogd

执行下面命令创建新的/etc/syslog.conf 文件：

```
cat > /etc/syslog.conf << "EOF"
# Begin /etc/syslog.conf
auth,authpriv.* -/var/log/auth.log
*.*;auth,authpriv.none -/var/log/sys.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
*.emerg *
# End /etc/syslog.conf
EOF
```

6.56.3. Sysklogd 包含的内容

安装的程序： klogd 和 syslogd

简要描述

klogd 系统守护进程，用于截断和记录内核信息

syslogd 记录系统程序提供的信息。每条记录至少包含一个时间戳和一个主机名，还有程序的名称，但这些信息的记录都依赖于记录守护进程被赋予的任务

6.57. Sysvinit-2.88dsf

Sysvinit 软件包内的程序用于控制系统的启动，运行和关闭。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 1MB

6.57.1. 安装 Sysvinit

当运行级别发生改变时（例如关闭系统），init 向那些由它启动的进程或不应该在新运行级别中运行的进程发送终止信号。同时，init 输出类似“向进程发送 TERM 信号”，该信息给人的错觉是好像向当前所有进出发送了信号。若要消除这个误导，可以将该信息修改为“按/etc/inittab 文件的配置向进程发送 TERM 信号”：

```
sed -i 's@Sending processes@& configured via /etc/inittab@g' \
src/init.c
```

之前的 Util-linux 已经安装了 wall 程序和 mountpoint 程序的可维护版本。所以现在不需要再安装 Sysvinit 包含的这些程序及其 man 手册：

```
sed -i -e 's/utmpdump wall/utmpdump/' \
  -e '/= mountpoint/d' \
  -e 's/mountpoint.1 wall.1//' src/Makefile
```

执行编译：

```
make –C src
```

该软件包不含测试套件。

安装软件：

```
make -C src install
```

6.57.2. Sysvinit 包含的内容

安装的程序： bootlogd, fstab-decode, halt, init, killall5, last, lastb (链接到last), mesg, mountpoint, pidof (链接到killall5), poweroff (链接到halt), reboot (链接到halt), runlevel, shutdown, sulogin, telinit (链接到init), utmpdump

简要描述

bootlogd	将启动信息记录到日志文件
fstab-decode	以 fstab 编码的参数运行命令
halt	除了在运行级别 0 外，通常以-h 参数调用 shutdown 命令，并通知内核关闭系统。然后再/var/log/wtmp 文件中注明系统正在关闭
init	内核初始化硬件后启动的第一个进程，它接管了启动进程并启动要求的全部进程
killall5	除了位于自身会话中的进程，killall5 向其他所有进程发送信号，不向自身会话进程发送信号是防止杀死运行它调用脚本的 shell
last	搜索/var/log/wtmp 文件，显示最后登入（登出）的用户；它也显示系统启动，关闭和运行级别的改变
lastb	显示/var/log/btmp 中记录的失败登录尝试

mesg	控制其他用户是否可以向当前用户终端发送信息
pidof	报告给定程序的进程 ID
poweroff	通知内核终止系统并关闭计算机（请看 halt 的描述）
reboot	通知内核重启系统（请看 halt 的描述）
runlevel	根据/var/run/utmp 文件记录的上次运行级别记录，报告之前和当前运行级别
shutdown	以安全的方式关闭系统，并向所有进程发送信号同时通知全部登入的用户
sulogin	允许 root 用户登录；当系统进入单用户模式时 init 进程会调用该程序
telinit	通知 init 将要跳转到的运行级别
utmpdump	以更友好的格式显示给定登录文件

6.58. Tar-1.26

Tar 是一个归档处理程序。

估计搭建时间： 1.9SBU

需要磁盘空间： 21.2MB

6.58.1. 安装 Tar

准备编译 Tar：

```
FORCE_UNSAFE_CONFIGURE=1 ./configure --prefix=/usr \
   --bindir=/bin --libexecdir=/usr/sbin
```

配置参数的含义如下：

FORCE_UNSAFE_CONFIGURE=1

该参数强制 mknod 的测试以 root 用户身份执行。尽管以 root 用户执行该测试是很危险的，但考虑到仅仅在半成品的系统中执行，这样做还是可以接受的。

执行编译：

```
make
```

若要运行测试套件，执行命令：

```
make check
```

安装软件：

```
make install
make -C doc install-html docdir=/usr/share/doc/tar-1.26
```

6.58.2. Tar 包含的内容

安装的程序：	rmt 和 tar
简要描述	
rmt	通过进程间通信连接远程操作磁带机设备
tar	创建，释放归档文件，列出归档文件内容

6.59. Texinfo-4.13a

Texinfo 软件包内的程序用于读，写和转换 info 手册。

估计搭建时间： 0.3SBU

需要磁盘空间： 21MB

6.59.1. 安装 Texinfo

开始配置 Texinfo：

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

执行编译：

```
make
```

若要测试编译结果，执行命令：

```
make check
```

安装软件：

```
make install
```

安装 TeX 的组件，该安装操作是可选的：

```
make TEXMF=/usr/share/texmf install-tex
```

make 参数的含义如下：

TEXMF=/usr/share/texmf

makefile 变量 TEXMF 定义了 TeX 树的根目录，如果以后要安装 TeX 包，就会安装到该目录

Info 文档系统使用普通文本文件保持菜单项列表。该文件位于/usr/share/info/dir。不幸的是，由于各种软件包 Makefile 文件的偶然问题，系统安装的 info 手册有时会失去同步。如果需要重建/usr/share/info/dir 文件，下面的可选命令可以完成该任务：

```
cd /usr/share/info
rm -v dir
for f in *
do install-info $f dir 2>/dev/null
done
```

6.59.2. Texinfo 包含的内容

安装的程序： info, infokey, install-info, makeinfo, pdftexi2dvi, texi2dvi, texi2pdf, texindex

安装的目录： /usr/share/texinfo

简要描述

info 读取 info 手册，info 手册类似于 man 手册，man 手册仅解释命令行参数，但 info 手册介绍得更为深入。例如，可以比较一下 man bison 和 info bison 显示的内容

infokey	将包含自定义 info 的源文件编译为二进制格式
install-info	安装 info 手册，更新 info 索引文件的项目
makeinfo	将给定 Texinfo 源文档翻译为 info 手册页面，普通文本或 HTML
pdftexi2dvi	将给定的 Texinfo 文档格式化为 PDF 文件
texi2dvi	将给定的 Texinfo 文档格式化为特定设备的文件，使文件可打印
texi2pdf	将给定的 Texinfo 文档格式化为 PDF 文件
texindex	排序 Texinfo 索引文件

6.60. Udev-173

Udev 用于动态创建设备节点。

估计搭建时间： 0.2SBU

需要磁盘空间： 9.3MB 外加 37MB 测试文件

6.60.1. 安装 Udev

在系统启动时会显示 udev 的版本，若要去除该警告信息，执行命令：

```
sed -i -e '/deprecated/d' udev/udevadm-trigger.c
```

udev-config 压缩包包含了配置 udev 的特定 LFS 文件。将它解压缩到 Udev 源码目录：

```
tar -xvf ../udev-config-20100128.tar.bz2
```

udev-testfiles 压缩包内是测试 udev 的必要文件。这些文件大约 37MB 但实际只占用了 7MB 的磁盘空间。

```
tar -xvf ../udev-173-testfiles.tar.bz2 --strip-components=1
```

在系统启动初期 Udev 还无权创建某些设备和目录，故手动创建它们：

```
install -dv /lib/{firmware,udev/devices/pts}
mknod -m0666 /lib/udev/devices/null c 1 3
```

开始配置 Udev：

```
./configure --prefix=/usr \
  --sysconfdir=/etc --sbindir=/sbin \
  --with-rootlibdir=/lib --libexecdir=/lib/udev \
  --disable-hwdb --disable-introspection \
  --disable-keymap --disable-gudev
```

新配置参数的含义：

--with-rootlibdir=/lib

该参数决定 libudev 库的安装位置。在系统启动的早期就需要使用该库，而默认-rootlibdir 是/usr/lib，由于启动早期/usr 可能不可用，故需要修改安装位置。

--libexecdir=/lib/udev

该参数设置 Udev 内部规则和帮助器程序的安装位置。

*--disable-**

该参数禁止 Udev 安装帮助器等等程序，它们依赖于其他外部库，而这些库不包含在 LFS 系统中，请查阅 Udev 的 README 文件获得详细信息。

执行编译：

```
make
```

测试软件编译结果：

```
make check
```

安装软件包：

```
make install
```

移除空文档目录：

```
rmdir -v /usr/share/doc/udev
```

现在安装特定 LFS 的自定义规则文件：

```
cd udev-config-20100128
make install
```

安装特定 LFS 规则文件的说明文档：

```
make install-doc
```

6.60.2. Udev 包含的内容

安装的程序： ata_id, cdrom_id, collect, create_floppy_devices, edd_id, firmware.sh, fstab_import, path_id, scsi_id, udevadm, udevd, usb_id, write_cd_rules, write_net_rules

安装的库： libudev.{a,so}

安装的目录： /etc/udev, /lib/udev, /lib/firmware

简要描述

ata_id	对于 ATA 设备，ata_id 为 Udev 提供唯一字符串或附加信息（uuid，标签）
cdrom_id	为 Udev 提供管理 CD-ROM 或 DVD-ROM 驱动的能力
collect	为当前 udev 事件提供 ID 并给出全部 udev 事件 ID 列表，为当前 ID 注册并判断是否需要为全部 ID 注册
create_floppy_devices	基于给定 CMOS 类型创建全部可能的软驱设备
edd_id	向 Udev 程序提供 BIOS 磁盘驱动的 EDD ID
firmware.sh	将固件上载到设备
fstab_import	在/etc/fstab 中搜索符合当前设备的项，并将信息提供给 Udev
path_id	将最短可能的唯一硬件提供给设备
scsi_id	发送 SCSI INQUIRY 命令道特定设备，并根据返回数据向 Udev 提供唯一 SCSI 标识符
udevadm	常用 udev 管理工具：控制 udevd 守护进程，从 Udev 数据库获取信息，监视 uevent 事件，等待 uevent 事件完成，测试 udev 配置，为给定设备触发 uevent 事件
udevd	监听并响应网络链接套接字上 uevent 事件的一个守护进程，它可以创建设备并执行预配置的附加程序
usb_id	将 USB 设备信息提供给 Udev
write_cd_rules	该脚本文件为可选设备产生 Udev 规则生成稳定的名称（查看 7.5 节“为设备创建自定义符号链接”）
write_net_rules	该脚本文件为网络接口产生 Udev 规则生成稳定的名称（查看 7.2 节“通用网络配置”）
libudev	udev 设备信息的函数接口库
/etc/udev	包含了 Udev 配置文件，设备权限和设备命名的规则

6.61. Vim-7.3

Vim 是一款非常强大的文本编辑器。

估计搭建时间： 1.0SBU

需要磁盘空间： 87MB

Vim 替代品

如果不喜欢vim而习惯于使用Emacs, Joe或Nano，请按照 *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html*的安装说明执行安装

6.61.1. 安装 Vim

首先，将默认配置文件 vimrc 的位置改为/etc：

```
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
```

开始配置 Vim：

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

配置参数的含义：

--enable-multibyte

该参数使 vim 支持多字节字符编码的文件编辑。当使用的语言环境是多字节字符集时这是很必要的，尤其是在编辑有类似 Fedora 等 Linux 发行版时更为有用，这些 linux 版本默认使用了 UTF-8 字符集。

执行编译：

```
make
```

若要测试编译结果，执行命令：

```
make test
```

然而，该测试套件会输出大量二进制数据到屏幕，可能导致当前终端设置出错。这可以通过将输出重定向到日志文件来避免。若测试成功结尾会输出“ALL DONE”。

安装软件：

```
make install
```

许多人喜欢使用 vi 命令而不是 vim。若用户喜欢键入 vi 来运行 vim，则需要为二进制文件和 man 手册创建符号链接：

```
ln -sv vim /usr/bin/vi
for L in /usr/share/man/{,*/}man1/vim.1; do
   ln -sv vim.1 $(dirname $L)/vi.1
done
```

默认情况下，Vim 的文档安装在/usr/share/vim 目录下。创建下面的符号链接使得可以用/usr/share/doc/vim-7.3 来访问，并和其他软件包的文档位置保持一致：

```
ln -sv ../vim/vim73/doc /usr/share/doc/vim-7.3
```

若要在 LFS 系统上安装 X Window 系统，可能需要在安装 X 后重新编译 vim。GUI 版本的 Vim 依赖于 X 和其他一些额外的函数库。要了解详情，请查阅 Vim 文档和 BLFS 中 Vim 的安装手册 *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html#postlfs-editors-vim*。

6.61.2. 配置 Vim

vim 默认运行在 vi 不兼容模式。这对于以前使用其他编辑器的用户可能很陌生。这个“nocompatible”设置在下面，高亮强调使用了新特性。同时提醒那些改为“compatible”的用户它应该在配置文件中首先被设置，其他设定需要紧随其后。执行下面的命令创建默认 vim 配置文件：

```
cat > /etc/vimrc << "EOF"
" Begin /etc/vimrc
set nocompatible
set backspace=2
syntax on
if (&term == "iterm") || (&term == "putty")
set background=dark
endif
" End /etc/vimrc
EOF
```

设定语句“*set nocompatible*”使 vim 工作在比 vi 兼容模式更有用的模式下。移除“*no*”可以维持旧的 vi 的行为。语句“*set backspace=2*”允许回退键能在断行处回退，并在行首插入。参数“*syntax on*”使 vim 支持语法高亮。最后，*if* 语句块中的“*set background=dark*”集中 vim 推测的终端模拟器的背景色。这能够在某些程序的黑色背景色中石油更好的色彩高亮主题。

运行下面的命令，查看文档中其他可选命令：

```
vim -c ':options'
```

Note

默认情况下，vim 仅为英语安装拼写文件。若要为你习惯的语言安装拼写文件，从 *ftp://ftp.vim.org/pub/vim/runtime/spell/*下载改语言的*.spl 文件（可选），和*.sug 文件以及字符编码文件，并将它们保存到/usr/share/vim/vim73/spell/。

修改/etc/vimrc 中的配置以便于使用这些拼写文件，例如：

```
set spelllang=en,ru
set spell
```

更多详细信息请查看上面 URL 中的 README 文件。

6.61.3. Vim 包含的内容

安装的程序： 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, xxd

安装的目录： /usr/share/vim

简要描述

ex	在 ex 模式中启动 vim
rview	view 的受限版本；不能执行 shell 命令并且 view 不能被挂起
rvim	vim 的受限版本；不能执行 shell 命令并且 vim 不能被挂起
vi	链接到 vim
view	以只读模式启动 vim
vim	vim 编辑器
vimdiff	用 vim 编辑两到三个版本的文件并显示它们的区别
vimtutor	教授 vim 的使用关键
xxd	以 16 进制查看指定文件，也可以从 16 进制查看转为普通查看，所以该程序可以用做二进制打包

6.62. 关于调试符号

大多数程序和库默认执行带调试符号地编译（gcc 命令带-g 参数）。这样的好处是在调试程序或库时，调试器不但能提供内存地址，而且能给出子程序和变量的名称。

然而，这些调试符号显著增大了程序或库的体积。下面的示例展示了调试符号占用的空间：

- 带调试符号的 bash 程序：1200KB
- 不带调试符号的 bash 程序：480KB
- 带调试符号的 Glibc 和 GCC 文件（/lib 和/usr/lib）：87MB
- 不带调试符号的 Glibc 和 GCC 文件（/lib 和/usr/lib）：16MB

不同的编译器和 C 库会导致程序大小的有所区别，但带调试符号的程序一般比不带的程序大 2~5 倍。

由于大多数用户不会在系统中使用调试器，所以移除调试符号将节省大量空间。下一节将介绍怎样清理程序或库的调试符号。

6.63. 清除调试符号

如果系统的用户不是程序员或者不打算对系统软件进行任何调试，那么移除程序和库的调试符号将节省大约 90MB 的磁盘空间。除了不能再完全调试这些软件外，移除操作不会引起任何不便。

大多数人使用下面的命令清理系统都不存在问题。但是，不小心的输入错误可能导致新系统损坏，所以在执行 strip 命令前最好为当前 LFS 系统做一次备份。

在执行清理操作前，确保没有任何需要被清理的软件正在运行。如果不确定是否有用户按照 6.4 节“进入 chroot 环境”的命令 chroot 进入了系统，那么最好先退出 chroot 环境：

```
logout
```

然后再次进入 chroot 环境：

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

现在可以安全地清理程序和库的调试符号了：

```
/tools/bin/find /{,usr/}{bin,lib,sbin} -type f \
   -exec /tools/bin/strip --strip-debug '{}' ';'
```

此时会有大量提示无法识别某些文件格式，这些警告都可以被安全地忽略，它们仅说明这些文件是脚本文件而不是二进制文件。

6.64. 清理系统

现在，先退出 chroot 环境，在用下面新的 chroot 命令进入系统：

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

使用新命令的原因是不再需要/tools 目录下的程序了，同理现在甚至可以删除/tools 目录。

Note

删除/tools 同时会删除 Tcl，Expect 和 DejaGNU 的临时版本，它们之前被用来运行工具链测试。如果你以后还需要这些程序，就需要重新编译安装。关于该操作 BLFS 书中有详细指导（*http://www.linuxfromscratch.org/blfs/*）。

如果手动卸载了虚拟内核文件系统或者重启过系统，请先确保进入 chroot 环境前挂载好虚拟内核文件系统。挂载过程在 6.2.2 节“挂载和填充/dev”和 6.2.3 节“挂载虚拟内核文件系统”有详细讲解。

第七章 配置系统启动脚本

7.1. 概述

本章讨论了一些配置文件和启动脚本。首先，要创建网络连接设置的配置文件。

- 7.2 节“一般网络配置”
- 7.3 节“自定义/etc/hosts 文件”

然后，设置设备启动。

- 7.4 节“LFS 系统的设备和模块处理”
- 7.5 节“创建指向设备的自定义链接”

下一节详细讨论了怎样安装和配置 LFS 系统启动脚本。大多数脚本可以直接使用无须修改，但有部分依赖于硬件信息的脚本需要额外的进行配置。

本书中使用的是System-V风格的init脚本，它们使用广泛而且非常简单。若不想使用System-V风格脚本，可以按照*http://www.linuxfromscratch.org/hints/downloads/files/bsd-init.txt*的提示安装BSD风格的init脚本。此外，在LFS邮件列表中搜索“depinit”，“upstart”，“systemd”将为你提供更多选择。

如果你决定使用其他风格的脚本，则跳过下面几节。

在附录D中是启动脚本的列表。

- 7.6节“LFS-Bootscripts-20111017”
- 7.7节“启动脚本的工作原理”
- 7.8节“配置系统主机名”
- 7.9节“配置setclock脚本”
- 7.10节“配置Linux控制台”
- 7.11节“配置sysklogd脚本”

最后是一些脚本和配置文件的简要介绍，在用户登录系统时会涉及这些文件。

- 7.13节“Bash Shell启动文件”
- 7.14节“创建/etc/inputrc文件”

7.2. 一般网络配置

本节讲述了怎样配置网卡。如果现在不使用网卡，就没必要创建网卡配置文件，此时可以从所有运行级别目录（/etc/rc.d/rc*.d）下删除*network*符号链接。

7.2.1. 为网络接口创建固定名称

如果系统中仅有一个网络接口需要配置，那么也可以跳过本节，但按本节照做也不会出错。大多数情况下（例如：同时拥有有线和无线接口的笔记本），完成本节的配置是必要的

由于Udev和模块化的网络驱动会并行加载驱动，使得网络接口编号是随机的，所以默认情况下系统重启后网络接口编号并不一定固定。例如，某计算机拥有一块Intel和一块Realtek网卡，Intel的网卡可能命名为eth0而Realtek网卡被命名为eth1。但在某些情况下，系统重启后者两块网卡的编号可能发生互换。要避免发生变动，就要配置Udev的脚本的规则，使其按照MAC地址产生固定命令。

使用下面的命令，确保网卡按照设备硬件地址命名，使得每次重启后网卡名称不变：

```
for NIC in /sys/class/net/* ; do
   INTERFACE=${NIC##*/} udevadm test --action=add $NIC
done
```

现在检查/etc/udev/rules.d/70-persistent-net.rules文件，查看网络设备分配到的名称：

```
cat /etc/udev/rules.d/70-persistent-net.rules
```

该文件中，每块网卡，以后简称NIC(Network Interface Card)以命令块开始，后接两行。第一行注释描述了硬件ID（若为PCI卡则为PCI供应商和设备ID）和能找到的硬件驱动。硬件ID和驱动都不是命名网卡的依据，它们仅仅作为参考信息放在这里。第二行是匹配NIC的Udev规则并实际命名。

所有的Udev规则都由几部分组成，各部分间以逗号或空格分隔。规则组成部分和个部分说明如下所示：

- SUBSYSTEM==”net”——忽略非网卡设备；
- ACTION==”add”——使Udev忽略非add的uevent事件规则（如“删除”和“更改”的uevent发生时不需要重命名网卡）
- DRIVERS==”?*”——Udev会忽略VLAN或桥接子接口（因为子接口不含驱动）。为避免和父设备冲突，Udev会跳过这些子接口
- ATTR{address}——该部分的值是NIC的硬件地址
- ATTR{type}==”1”——在存在某些无线驱动创建了多块虚拟网卡的情况下，仅将规则应用于主要网卡。第二块网卡将被跳过，原因仍然是为了避免命名冲突
- KERNEL==”eth*”——这部分作用于Udev规则生成器，使它支持具有相同MAC地址的多网卡计算机（PS3即是这样的机器）。这部分对多大叔LFS用户是没必要的，但留着也无害
- NAME——该部分即是Udev附加到网卡的名称

NAME的值是最重要的部分。在继续之前确保你记住了每块网卡的名称，下面将使用NAME的值创建配置文件。

7.2.2. 创建网卡配置文件

当前网络使用或禁用哪块网卡依赖于/etc/sysconfig/下的文件。该目录下包含了可配置

的每块网卡，例如"ifconfig.xyz"，其中"xyz"是设备名称（如eth0）。文件内容即为该接口的属性，比如IP地址，子网掩码等等。该文件名必须以*ifconfig*开头。

下面的命令演示了为静态IP的eth0设备创建样例文件：

```
cd /etc/sysconfig/
cat > ifconfig.eth0 << "EOF"
ONBOOT=yes
IFACE=eth0
SERVICE=ipv4-static
IP=192.168.1.1
GATEWAY=192.168.1.2
PREFIX=24
BROADCAST=192.168.1.255
EOF
```

每个配置文件中的变量的值必须和正确的设置相匹配。

如果 ONBOOT 变量设为“yes”，那么在系统启动时该网络配置脚本就会自动启用 NIC。如果被设为非“yes“的其他值，那么 NIC 就不会自动启用。但可以手动使用 ifup 和 ifdown 命令来启用或禁用网卡。

变量 IFACE 定义网卡名称，例如 eth0。在网卡配置文件中必须设置该变量。

变量 SERVICE 定义了获取 IP 地址的方法。LFS-Bootscripts 软件包中含有一个模块化的 IP 分配格式，它能在/lib/services/目录下创建额外的文件允许其他 IP 分配方法。BLFS 书中介绍的动态主机配置协议(DHCP)常常会使用该变量。

如果当前存在默认网关，那么变量 GATEWAY 定义了默认网关的 IP 地址，如果没有则注释掉该变量。

变量 PREFIX 包含了子网中使用的比特位。每个字节代表 IP 地址的 8 位。如果子网掩码是 255.255.255.0，那么那么使用头三个字节（24 位）指定网络号。如果子网掩码是 255.255.255.240，则使用了头 28 比特。DSL 和基于线缆的网络服务提供商(ISP)通常使用长于 24 比特位的前缀。在本例中（PREFIX=24），子网掩码是 255.255.255.0。请根据特定子网调整 PREFIX 变量。

7.2.3. 创建/etc/resolv.conf 文件

如果系统尝试连接到网络，就需要域名解析服务（DNS）将网络域名翻译成 IP 地址或将 IP 地址翻译为域名。最简单的方法是将 DNS 服务器的 IP 地址放置在解析文件/etc/resolv.conf 中，方便 ISP 或网络管理员访问。使用下面的命令创建该文件：

```
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
```

可以忽略 *domain* 语句或用 *search* 语句替代。请查阅 man 手册获取 resolv.conf 的详细说明。

将*<IP address of your secondary nameserver>*替换为最合适的 DNS 的 IP 地址。通常可能有多项存在（考虑可靠性可能需要备选服务器）。如果仅需要一个 DNS 服务器，则删除第二个 *nameserver* 行。另外，IP 地址也可能是本地网络的一个路由器。

Note

Google 公共 IPv4 DNS 服务器地址为 8.8.8.8 和 8.8.8.4。

7.3. 自定义/etc/hosts 文件

在/etc/hosts 文件中可以将网卡配置为使用 IP 地址和正式域名（Full Qualified Domain Name, FQDN）及可以使用别名，配置的语法为：

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

除非本计算机需要在因特网中可见（即存在已注册的域名和合法分配的 IP 地址——大多数用户无法拥有这些），否则就将 IP 地址设置为私有网络 IP 地地址范围，合法的范围为：

```
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 的范围是 16-31，y 的范围是 0-255。

合法的私有网络 IP 地址可能是 192.168.1.1。合法的 FQDN 可能是 lfs.example.org。

即便不使用网卡也需要一个合法的 FQDN，这是因为某些程序需要它才能正常工作。

创建/etc/hosts 文件：

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (network card version)
127.0.0.1 localhost
<192.168.1.1> <HOSTNAME.example.org> [alias1] [alias2 ...]
# End /etc/hosts (network card version)
EOF
```

请按照具体需要（比如网络管理员或系统管理员分配了一个 IP 地址以便于计算机连接到现有网络）设置*<192.168.1.1>*和*<HOSTNAME.example.org>*的值。另外，别名的设置是可选的。

如果暂时不需要配置网卡，则按下面的方式创建/etc/hosts 文件：

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (no network card version)
127.0.0.1 <HOSTNAME.example.org> <HOSTNAME> localhost
# End /etc/hosts (no network card version)
EOF
```

7.4. LFS 系统的设备和模块管理

在第六章中我们安装了 Udev 软件包。在开始讨论它的工作原理前，我们先来回顾一下以前管理设备的的方法。

传统 Linux 系统使用静态设备创建办法，因此在/dev 目录下创建了大量的设备节点（有时可能创建了成百上千个），而不管某些设备可能根本不存在。执行这项工作的典型脚本是 MAKEDEV，它使用了大量对 mknod 程序的调用来创建现存的所有主设备和从设备。

如果使用 Udev 方法，则仅创建内核探测到的节点对应的设备节点。由于每次系统启动时都需要创建这些设备节点，故而将他们存储在 tmpfs 文件系统中（一种虚拟文件系统，完全存在于内存）。由于设备节点仅占用很小的空间，所以对内存的使用是微不足道的。

7.4.1. 历史

在 2000 年 2 月，一种名叫 devfs 的新文件系统被合并到 2.3.46 的内核，并正式使用到稳定版本的内核 2.4 中。尽管它存在于内核源代码中，但这种动态创建设备节点的方法一直没有得到内核开发人员的过多关注。

主要的问题在于 devfs 探测，创建，命名设备时采用的方式。尤其是命名设备的方式是最关键的问题。如果设备命名允许更改配置，而且设备的命名策略决定于系统管理员而不是由特定开发人员强制决定，那么 devfs 可能会被人接受。由于 devfs 的设计缺陷，该文件系统还会产生竞争条件，而修正方法是将内核整个回复到旧版本。因而它被弃用了很长时间，所以缺乏维护，最终在 2006 年六月从内核移除。

随着内核不稳定版 2.5 以及后来稳定的 2.6 版本的发布，一种新的虚拟文件系统 sysfs 出现了。sysfs 的功能是生成系统硬件配置的快照并导出到用户空间进程。由于用户空间可见表述，查看 devfs 的用户空间更替的可能逐渐变得现实起来。

7.4.2. Udev 实现

7.4.2.1. Sysfs

上面简要提到了 sysfs 文件系统。你可能想了解 sysfs 怎样发现系统当前设备以及怎样为它们分配设备号的。由于驱动程序已经被编译进内核，当探测到可识别的设备时就向 sysfs 注册这些对象。如果驱动被编译成模块，那么这个注册行为将在模块加载后发生。一旦 sysfs 文件系统挂载到/sys 上，sysfs 中注册的内置驱动数据就开放给用户空间进程，此时 udevd 守护进程也可以利用这些信息创建设备节点了。

7.4.2.2. Udev 启动脚本

当 Linux 系统启动时，/etc/rc.d/init.d/udev 初始化脚本负责创建设备节点。该脚本按照默认的/sbin/hotplug 重置 uevent 句柄。因为此时内核不再需要调用外部程序，所以这是可行的。首先，udevd 守护进程会监听网络链接套接字上的 uevent 事件。然后，启动脚本将/lib/udev/devices 下的静态设备节点复制到/dev 下。为什么要执行这一步呢？这是因为在系统启动的早期，动态设备管理进程尚未准备完成，此时就需要这些设备，目录和符号链接，此外 udevd 自身也需要这些东西。在/lib/udev/devices 创建静态设备节点同样也可以为设备提供易用的工作环境，但动态设备管理架构却不支持该目录。在复制完成后，启动脚本就启动 udev 守护进程 udevd，该进程会相应收到的 uevent 事件。最后，启动脚本强制内核为已注册设备重播这些 uevent，然后 udevd 就好处理这些事件。

在 mountfs 运行前（/usr 和/var 会启动它），/etc/rc.d/init.d/udev_retry 初始化脚本会为

那些依赖于挂载的文件系统的子系统重新触发事件。Udev 启动脚本在 mountfs 之后运行，所以子系统的规则必须在第二遍成功满足。在/etc/sysconfig/udev_retry 文件中定义了规则；该文件中除注释外的所有语句都是子系统名称，在第二遍触发时将使用它们。（若要获得设备的子系统，使用 udevadm info—attribute-walk）。

7.4.2.3. 创建设备节点

为了获得设备正确的主，次设备号，Udev 需要/sys 下 sysfs 提供的信息。例如，/sys/class/tty/vcs/dev 包含字符串“7:0”，udevd 使用该字符串创建设备节点主设备号为 7 次设备号为 0。/dev 目录下创建的节点名称和权限依据了/etc/udev/rules.d/目录下的文件定义的规则。这些命令规则类似于 LFS-Bootscripts 软件包中的规则。如果 udevd 无法为设备找到对应的规则，就会为设备使用默认的权限 660 以及默认的拥有主 root:root。在 /usr/share/doc/udev-173/writing_udev_rules/index.html 中可以找到关于 Udev 规则配置文件的语法说明的文档。

7.4.2.4. 加载模块

将设备驱动编译成模块时可能将别名也编译进去了。设备别名可以在 modinfo 程序的输出中看到，它通常和模块设备支持的特定总线标识符有关。比如，*snd-fm801* 驱动支持供应商 ID 为 0x1319 和设备 ID 为 0x0801 的 PCI 设备，且它的别名是“*pci:v00001319d00000801sv*sd*bc04sc01i**”。对更多的设备，总线驱动会导出驱动的别名便于 sysfs 管理设备。即，/sys/bus/pci/devices/0000:00:0d.0/modalias 可能包含字符串“*pci:v00001319d00000801sv00001319sd00001319bc04sc01i00*”。Udev 提供的默认规则使 udevd 利用 uevent 环境变量 MODALIAS 的内容（必须和 sysfs 的 modalias 文件内容相同）调用/sbin/modprobe 程序，在扩展通配符后所有别名匹配该字符串的模块就就被加载起来。

在本例中，除了 *snd-fm801* 外，甚至于已过时的 *forte* 驱动都会被加载(如果它存在)。接下去我们将讲述怎样阻止加载不需要的驱动。

同时，内核本身也能够按需加载网络协议模块，文件系统模块和 NLS 支持模块。

7.4.2.5. 管理热插拔、动态设备

当你插入某设备，如通用串行总线(Universal Serial Bus, USB)MP3 播放器时，内核监测到有设备连接并且产生一个 uevent 事件。然后 udevd 守护进程就会处理该事件。

7.4.3. 加载模块和创建设备可能遇到的问题

当自动创建设备节点时可能遇到一些奇怪的问题。

7.4.3.1. 内核模块无法自动加载

仅当模块拥有特定总线别名且总线驱动正确将必要的别名导出到 sysfs 时，udev 才会加载该模块。否则，你必须用其他方式安排模块加载。在 Linux-3.1 中，Udev 可以正确加载 INPUT，IDE，PCI，USB，SCSI，SERIO，FireWire 设备驱动。

若要判断 udev 是否支持你的设备驱动，将模块名称作参数运行 modinfo 程序，然后再 /sys/bus 目录下设备目录中查找是否存在 modalias 文件。

如果在 sysfs 中存在 modalias 文件，那么支持设备的驱动就可以直接使用它，如果不存在则说明驱动程序存在 bug。此时尝试不使用 Udev 加载驱动，看看问题是否解决。

如果在/sys/bus 相关目录下没有 modalias 文件，说明内核开发人员尚未增加支持该总线类型的别名。在 Linux-3.1 中，ISA 总线就属于这种情况。此时只好等待后续内核版本能解决

问题。

Udev 不会尝试加载“外壳”驱动如 *snd-pcm-oss* 和非硬件驱动如 *loop*。

7.4.3.2. 内核模块没有自动加载，而且 Udev 也没有尝试加载它

如果“外壳”模块仅仅为了提高其他模块的功能（例如，*snd-pcm-oss* 提高了 *snd-pcm* 的功能，使得 OSS 应用程序能够适应声卡），在 udev 加载外壳模块后需要配置 modprobe 加载这些外壳。所以，需要在*/etc/modprobe.d/<filename>.conf* 文件中添加一行“install”：

```
install snd-pcm /sbin/modprobe -i snd-pcm ; \
/sbin/modprobe snd-pcm-oss ; true
```

如果问题模块不是外壳模块而是仅供自身使用，则要配置 modules 启动脚本使得系统启动时自动加载它。因此，需要在/etc/sysconfig/modules 文件中添加独立行加入模块名称。这种方法也可以用来解决外壳模块的问题，但并不是最优方案。

7.4.3.3. Udev 加载了不必要的模块

若要避免加载不必要的模块，要么不编译该模块，要么将它加入黑名单文件 /etc/modprobe.d/balcklist.conf，以 *forte* 模块为例：

```
blacklist forte
```

注意，黑名单中的模块仍然可以使用 modprobe 命令手动加载。

7.4.3.4. Udev 创建了错误设备或者错误了符号链接

通常在设备和规则发生错误匹配时会发生这种情况。例如，一条书写拙劣的规则可以同时匹配 SCSI 磁盘（正确的）和相应提供商的 SCSI 普通设备（错误）。解决方法是利用 **udevadm info** 命令找到该规则并改正。

7.4.3.5. Udev 规则工作不稳定

这也可能是上一个问题的另一种表现。如果不是，或者规则使用了 sysfs 属性，那么也可能是内核的时序问题，这会在后续内核版本修复。而现在，可以创建一条规则等待用过的 sysfs 属性并将之追加到/etc/udev/rules.d/10-wait_for_sysfs.rules 文件中（若无此文件则创建它）。然后将它通知 LFS 开发列表。

7.4.3.6. Udev 无法创建设备

进一步假设设备驱动已经被静态编译进内核，或者驱动已经以模块的形式加载，而且确认 Udev 没有创建错误名称的设备。

如果内核设备无法导出数据到 sysfs，那么 Udev 不能得到必要的信息创建设备节点。当使用的是独立于源码树外的第三方驱动时，更容易发生这中情况。这时需要手动在 /lib/udev/devices 下安装合适的主、从设备号创建静态设备节点（请查阅内核文档的 devices.txt 或第三方驱动提供商提供的文档）。udev 启动脚本会将静态设备节点复制到/dev 下。

7.4.3.7. 系统重启后设备命名顺序随机改变

这是 Udev 的设计原因造成的，Udev 被设计为并行处理 uevent 事件并加载模块，因此会产生不可预料的顺序。这绝不可能修正，所以绝不应该依赖于内核的设备名称。所以，应

该使用设备的固定属性比如序列号或 Udev 安装工具不同的变量的*_id 输出创建规则并创建符号链接。查阅 7.5 节“创建设备的自定义符号链接”和 7.2 节“一般网络配置”。

7.4.4. 有用的参考资料

下面的站点是额外的帮助文档：

- *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
- 深入阅读 *http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html*

7.5. 创建设备的自定义符号链接

7.5.1. CD-ROM 符号链接

以后可能会安装一些软件（比如各种媒体播放器），它们可能需要/dev/cdrom（指向 CD-ROM）和/dev/dvd（指向 DVD-ROM）符号链接。而且，如果在/etc/fstab 中放置这些符号链接的引用也会更加方便。Udev 的脚步会根据每个设备的功能，为你产生规则文件并创建符号链接，但你需要决定脚本使用的两种操作模式。

首先，脚本可以工作在“路径依赖”模式（USB 和 FireWire 设备默认模式），在该模式下是根据 CD 或 DVD 设备的物理路径创建规则。其次，它可以工作在“id 依赖”的模式（IDE 和 SCSI 设备的默认模式），此时是根据存储在 CD 或 DVD 设备中的标识字符串创建规则。Udev 使用 path_id 脚本判断路径，而 ata_id 或 scsi_id 程序用于获得硬件的标识字符串，具体使用哪个程序依赖于你的具体设备类型。

上述两种模式各有千秋，具体使用哪种模式由具体设备变动的类型决定。如果你希望设备的物理路径（即设备插入的端口或插槽）可以更改，如常常需要将设备移动到另一个 IDE 端口或 USB 插口，此时就需要使用“id 依赖”模式。反过来，如果希望设备的标识符可以更改，如设备损坏时需要更换新的同类设备，并接入到同一插口，此时就需要使用“路径依赖”模式。

如果你的设备变量属于这两种之一，那么请按需选择合适的模式。

Important

外部设备（如 USB 连接的 CD 设备）最好不要使用路径依赖模式，因为每次设备都可能连接到新的尾部端口，它的物理路径可能是不同的。如果你使用路径依赖模式书写了 Udev 的规则，那么所有外部设备都会存在该问题而不仅限于 CD 或 DVD 设备。

如果想要查看 Udev 脚本为 CD-ROM 设备使用的值，找到/sys 下相应的目录（例如，这里是/sys/block/hdd），然后运行类似下面的命令：

```
udevadm test /sys/block/hdd
```

查看包含不同*_id 程序输出的行。如果存在“id 依赖”模式则它会使用 ID_SERIAL 值，否则会使用 ID_MODEL 和 ID_REVISION。而“路径依赖“会使用 ID_PATH 的值。

如果默认模式不适合你的情况，那么可以按下面命令修改 /lib/udev/rules.d/75-cd-aliases-generator.rules 文件：

```
sed -i -e 's/"write_cd_rules"/"write_cd_rules mode"/' \
   /lib/udev/rules.d/75-cd-aliases-generator.rules
```

注意，此时并不强制要求创建规则文件或符号链接，这时因为我们之前吧宿主系统的/dev 目录绑定地挂载到 LFS 系统，并且假定符号链接存在于宿主系统。在你第一次启动 LFS 系统后会创建规则和符号链接。

然而，如果有多个 CD-ROM，并且产生的符号链接指向了主机之外的设备，这时因为设备检测的顺序不可预知。当第一次启动 LFS 系统时分配创建是稳定的，所以问题在于你是否需要两个系统的符号链接都指向同一设备。如果是，则需要在启动后检查（执行可能的编辑）产生的/etc/udev/rules.d/70-persistent-cd.rules 文件，确保分配的符号链接符合需要。

7.5.2. 管理重复的设备

正如 7.4 节“LFS 系统中的设备和模块管理“解释的那样，相同功能的设备在/dev 出现的顺序是随机的。即，有一个 USB 网络摄像机和 TV 调谐器，有时/dev/video0 是摄像机 /dev/video1 是调谐器，但有时重启后它们指代的设备正好相反。对于除了声卡和网卡外的各级硬件设备，可以通过为自定义固定符号链接创建 udev 规则文件来解决问题。网卡的解决方法在 7.2 节”一般网络配置“专门讨论，而声卡的配置在 BLFS 中。

对每个可能发生该问题的设备(尽管问题可能不存在当前 Linux 发行版中)，在/sys/class 或/sys/block 下找到相应目录。对视频设备，可能是/sys/class/video4linux/videoX。找出能唯一标识设备的属性（通常是提供商和产品 ID 和/或序列号）：

```
udevadm info -a -p /sys/class/video4linux/video0
```

然后写入规则并创建符号链接，例如：

```
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
```

写入后的结果是/dev/video0 和/dev/video1 仍然随机指代调谐器和网络摄像机（所以绝不能直接使用），但符号/dev/tvtuner 和/dev/webcam 始终指向正确的设备。

7.6. LFS-Bootscripts-20111017

LFS-Bootscripts 软件包包含一系列脚本用于管理 LFS 系统启动、关闭。

估计搭建时间： 少于 0.1SBU

需要磁盘空间： 260KB

7.6.1. 安装 LFS-Bootscripts

安装软件包：

```
make install
```

7.6.2. LFS-Bootscripts 包含的内容

安装的脚本： checkfs, cleanfs, console, consolelog, functions, halt, ifdown, ifup, localnet, modules, mountfs, mountkernfs, network, rc, reboot, sendsignals, setclock, static, swap, sysctl, sysklogd, template, udev, udev_retry

安装的目录： /etc/rc.d, /etc/init.d (符号链接), /etc/sysconfig, /lib/services, /lib/lsb (符号链接)

简要描述

checkfs	在挂载文件系统前检查完整性（日志文件系统和网络文件系统除外）
cleanfs	清除那些两次重启间不需要持久存在的文件，比如/var/run/和/var/lock 中的文件；它会重建/var/run/utmp 并删除现在的/etc/nologin，/fastboot 和/forcefsck 文件
console	为指定的键盘布局加载正确的键码图，它也能设置屏幕字体
consolelog	通过设置内核日志级别控制到达控制台的信息
functions	包含了常用函数，比如错误和状态检查，许多启动脚本会使用它们
halt	终止系统
ifdown	禁用网卡设备
ifup	初始化网络设备
localnet	设置系统主机名和本地回环设备
modules	加载/etc/sysconfig/modules 列表中的内核模块，加载时使用的参数也在该文件中
mountfs	除了被标记为 *noauto* 或基于网络的文件系统外，挂载所有文件系统
mountkernfs	挂载虚拟文件系统，如 proc
network	配置网络接口，比如网卡，并设置默认网关（如果有合适的网关）
rc	主要的运行级控制脚本；负责逐个运行其他启动脚本，运行的顺序决定于正在处理的符号链接名称
reboot	重启系统
sendsignals	在系统重启或关闭前确保每个进程都已终止
setclock	将内核时钟重置为本地时间，防止硬件时钟不是 UTC 时间
static	当为网络接口分配静态 IP 地址时提供必要的功能
swap	启用或禁用交换分区和交换文件
sysctl	如果/etc/sysctl.conf 文件存在则从中加载系统配置值，并将这些值通知运行中的内核
sysklogd	启动或停止系统和内核日志守护进程

template	该模版用于为其他守护进程创建自定义启动脚本
udev	准备/dev 目录并启动 Udev
udev_retry	重试失败的 udev 事件，如果需要就从/etc/udev/rules.d 复制规则文件

7.7. 怎样让启动脚本工作起来？

Linux 使用了一种特殊的启动策略叫做 SysVinit，它是基于运行级的一种概念。在不同系统间差异很大，所以无法假设在特定 Linux 发行版中的运行情况和 LFS 中的工作情况一样。LFS 有自己的工作方式，但它同时也遵守广泛接受的标准。

SysVint（以后用“init“称呼）使用了运行级别的框架。一共有 7（从 0 到 6）种运行级别（实际上，可能有更多的运行级别，但更多运行级别仅用于特殊情形并不常用。请查看 man 手册 init(8)获取详情），计算机启动时会执行相应的工作。默认的运行级别是 3.下面是不同运行级别的描述：

```
0: 关闭计算机
1: 单用户模式
2: 无网络的多用户模式
3: 有网络的多用户模式
4: 为以后保留
5: 同4，通常用于图形界面登录（比如X的xdm或KDE的kdm）
6:重启计算机
```

7.7.1. 配置 Sysvinit

在内涵初始化时，第一个运行的程序要么由命令行指定，要么就执行默认的 **init**。该程序读取初始化文件/etc/inittab。用下述命令创建它：

```
cat > /etc/inittab << "EOF"
# Begin /etc/inittab
id:3:initdefault:
si::sysinit:/etc/rc.d/init.d/rc S
l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
su:S016:once:/sbin/sulogin
1:2345:respawn:/sbin/agetty --noclear tty1 9600
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600
# End /etc/inittab
EOF
```

关于该文件的解释清查看 inittab 的 man 手册。对 LFS 而言，关键的命令是 rc。上面的

初始化文件会指导 rc 逐个执行/etc/rc.d/rcsysinit.d 目录中所有以 S 开头的脚本，然后是 /etc/rc.d/rc?.d 目录中以 S 开头的脚本，“？”是 initdefault 的值。

为了方便起见，rc 脚本会读取/lib/lsb/init-functions 中的函数库。该库也会读取一个可选的配置文件，/etc/sysconfig/init_params。后续部分讨论的任何系统配置文件参数都可以放在在文件中，这样所有的系统参数都固定存在一个文件中。

考虑到便于调试，函数脚本也会在/run/var/bootlog 中记录日志。因为/run 目录是 tmpfs 文件系统，所以该文件不会持久化。

7.7.2. 改变运行级别

使用命令 **init *<runlevel>***可以改变运行级别，其中***<runlevel>***是目标运行级别。例如，若要重启计算机，用户可以键入 **init 6** 命令，它也是 **reboot** 命令的别名。反过来，**init 0** 也是 **halt** 命令的别名。

在/etc/rc.d 下有很多目录形似 rc?.d（？是运行级别）和 rcsysinit.d，它们都包含了大量符号链接。有些以 K 开头，有些以 S 开头，在首字母后都跟两个数字。K 意为(Kill)停止服务，S 意为启动(Start)服务。而数字决定了脚本运行的顺序，从 00 到 99，数字越低运行得越早。当 **init** 转换到另一个运行级别，根据运行级别就会选择合适的服务启动或停止。

/etc/rc.d/init.d 中的脚本才是执行真正工作的脚本，所有的符号链接都指向它们。K 和 S 都链接到/etc/rc.d/init.d 中同一脚本。这是因为不同的脚本可以使用不同的参数来调用，比如 *start*，*stop*，*restart*，*reload* 和 *status*。当遇到 K，合适的脚本会以 *stop* 参数执行，当遇到 S，就会以 *start* 为参数调用脚本。

然而，还有一个例外。在 rc0.d 和 rc6.d 目录下以 S 开头的链接不会启动任何服务。它们仅使用 *stop* 参数来停止服务。这时因为用户需要重启或关闭系统时不需要启动任何服务，系统唯一要做的事就是停止。

下面是脚本参数的描述：

start

启动服务。

stop

停止服务。

restart

重启服务。

reload

更新服务的配置。当修改服务的配置后，需要改参数更新服务而不必重启服务。

status

获取服务运行状态或获取 PID。

可以自由修改启动进程的执行方式（毕竟，这是你的 LFS 系统）。这里给出的文件仅仅是一个参考而已。

7.8. 配置系统主机名

localnet 脚本的部分功能就是设置系统的主机名。而这需要在/etc/sysconfig/network 文件中做一些配置。

创建/etc/sysconfig/network 文件并键入主机名：

```
echo "HOSTNAME=<lfs>" > /etc/sysconfig/network
```

请将*<lfs>*换成你想要的主机名。不要在这里输入正式域名 FQDN，它应该位于/etc/hosts 文件中。

7.9. 配置 setclock 脚本

setclock 脚本能从硬件时钟获取时间，如 BIOS 或 CMOS 时钟。如果硬件时钟是 UTC 时间，该脚本会使用/etc/localtime 文件（该文件帮助 hwclock 程序判断用户所在时区）将硬件时钟转换为本地时间。由于没有方法探测硬件时钟是否 UTC 时间，故而需要手动配置。

当内核在启动时探测到硬件时 udev 会允许 setclock。也可以 stop 参数手动停止该脚本来存储系统时间到 CMOS 时钟。

如果你不知道硬件时钟是否被设置为 UTC，可以用 **hwclock --localtime --show** 命令判别。该命令会根据硬件时钟输出当前时间。如果输出的时间和你的手表匹配，说明硬件时钟被设为本地时间。如果 hwclock 输出不是本地时间，很可能是 UTC 时间。这可以通过在 hwclock 显示的时间上加减正确的时区差量来判断。例如，如果你在 MST 时区，也即 GMT-0700，则在当地时间上加 7 个小时。

如果硬件时钟不是 UTC 时间，则将下面的 UTC 变量设为 0.

执行命令创建新的/etc/sysconfig/clock 文件：

```
cat > /etc/sysconfig/clock << "EOF"
# Begin /etc/sysconfig/clock
UTC=1
# Set this to any options you might need to give to hwclock,
# such as machine hardware clock type for Alphas.
CLOCKPARAMS=
# End /etc/sysconfig/clock
EOF
```

在*http://www.linuxfromscratch.org/hints/downloads/files/time.txt*上讲述了怎样处理LFS的时间。它解释了时区，UTC和TZ环境变量等等。

Note

在/etc/sysconfig/rc.site 文件中只能选择设置 CLOCKPARAMS 或 UTC 之一。

7.10. 配置 Linux 控制台

这部分讨论怎样配置 console 和 consolelog 启动脚本，设置键盘映射，控制台字体和控制台内核日志级别。如果仅使用 ASCII 字符集（即不使用版权符号、英镑和欧元符号），而且使用 U.S.键盘，那么可以跳过本节大部分内容。如果没有配置文件，那么 console 启动脚本什么也不会做。

console 和 consolelog 脚本会读取/etc/sysconfig/console 文件获得配置信息。决定使用哪种键盘映射以及字体。不同语言的“怎样做”*http://www.tldp.org/HOWTO/HOWTO-INDEX/other-lang.html* 也可以获得一些帮助。如果仍有疑惑，查看/lib/kbd 目录下合法的键盘映射和屏幕字体。阅读 man 手册 loadkeys(1)和 setfont(8)决定正确的参数。

/etc/sysconfig/console 文件应该包含以下形式的行 VARIABLE=”value”。可识别的变量如下：

LOGLEVEL

该变量指定了 dmesg 程序发送到控制台的内核信息的日志级别。合法的级别从“1“（无信息）到”8“。默认值是”7“。

KEYMAP

该变量指定了 loadkeys 程序的参数，典型的有要加载的键盘的名称如“es“。如果没有设置改变量，启动脚本不会运行 loadkeys 程序，并使用默认内核键盘图。

KEYMAP_CORRECTIONS

该变量（极少使用）为二次调用 laodkeys 程序指定参数。当预置键盘映射无法完全满足需求并需要小调整时才会需要改参数。如，将欧元符号加入不含它的映射中，将该变量设为“euro2“。

FONT

该变量为 setfont 程序指定参数。典型地，它包含了字体名，“-m“，和要加载的应用程序字符集映射名称。如，为了加载”lat-16”字体和”8859-1“应用程序字符集（适用于 USA），则将该变量设为”lat-16 –m 8859-1“。在 UTF-8 模式下，内核使用应用程序字符映射来转换 8-bit 键码到 UTF-8 键码，此时”-m“参数就需要在键盘映射时设置编码。

UNICODE

按顺序将变量设置为“1“，”yes“或”true“，可以将控制台置为 UTF-8 模式。在 UTF-8 的语言环境中很有用。

LEGACY_CHARSET

对很多键盘布局来说，在 Kbd 软件包中都没有相应的 Unicode 键盘映射。如果该变量被设置非 UTF-8 编码的键盘映射，那么 console 启动脚本会将可用的键盘映射转换 为 UTF-8 编码。

下面是一些例子：

对于非 Unicode 设置，仅仅需要设置 KEYMAP 和 FONT 变量，例如，对于波兰的设置可能是：

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console
KEYMAP="pl2"
FONT="lat2a-16 -m 8859-2"
# End /etc/sysconfig/console
EOF
```

如前面提到的，有时需要稍微调整键盘映射。下例就是将欧元符号添加到德语键盘映射的配置：

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console
KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
FONT="lat0-16 -m 8859-15"
# End /etc/sysconfig/console
EOF
```

下面是为保加利亚语允许 Unicode 编码，使用已存在的 UTF-8 键盘映射：

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console
UNICODE="1"
KEYMAP="bg_bds-utf8"
FONT="LatArCyrHeb-16"
# End /etc/sysconfig/console
EOF
```

由于在前面的例子中使用了 512-glyph LatArCyrHeb-16 字体，除非在 Linux 控制台使用帧缓存否则无法使用高亮颜色。如果不使用帧缓冲就像要高亮色，可能就不能使用自己语言的字符，然而还是可以使用特定 256 色的语言，如下所示：

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console
UNICODE="1"
KEYMAP="bg_bds-utf8"
FONT="cyr-sun16"
# End /etc/sysconfig/console
EOF
```

下面的例子展示了将 ISO-8859-15 键盘映射自动转换为 UTF-8 编码，并且在 Unicode 模式下使用了某些死键：

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console
UNICODE="1"
KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
LEGACY_CHARSET="iso-8859-15"
FONT="LatArCyrHeb-16 -m 8859-15"
# End /etc/sysconfig/console
EOF
```

一些键盘映射存在一些死键（即，这些键无法产生字符，但却会在下一个键前产生一个重音）或定义了合成规则（比如：“按下 Ctrl+.AE 获得 Æ “）。Linux-3.1 解释了死键和合成规则，但进度源字符合成后不是多字节才行。该缺陷不会影响欧洲语言的键盘映射，因为它们的重音是加载未重读的 ASCII 字符，或两个 ASCII 字符被组合在一起。然而，在 UTF-8 模式下这就是一个问题了，如，对希腊语，有时需要在”alpha“字母上加重音。解决方案是要么避免使用 UTF-8，要么安装 X window 系统，X window 中没有输入的限制。

对于中文，日文，韩文等语言，Linux 控制台无法配置显示这些语言。如果用户需要这些语言就需要安装 X window 系统，字体和正确的输入法（如，SCIM，支持大量语言）。

Note

/etc/sysconfig/console 文件仅仅包含 Linux 文本控制台的本地化。它不包含设置正常键盘布局、X window 系统终端字体、和 ssh 会话或者串行控制台的东西。在这种情形下，最后提到的两个限制并不适用。

7.11. 配置 sysklogd 脚本

sysklogd 脚本以-m　0 为参数调用 syslogd。syslogd 默认每隔 20 分钟写入日志文件一次时间戳标记，该参数会关闭该周期性动作。如果你需要开启，编辑/etc/sysconfig/rc.site 并将 SYSKLOGD_PARMS 变量设置为需要的值。例如，若要移除所有参数，就将变量设为 null 值：

```
SYSKLOGD_PARMS=
```

查阅 **man syslogd** 获取更多参数。

7.12. rc.site 文件

可选的/etc/sysconfig/rc.site 文件包含了每个启动脚本的自动设置信息。可以将它作为一种替代方式设置/etc/sysconfig 目录下的 hostname，console 和 clock 文件中的变量。如果同时在这些分离的文件和 rc.site 文件中设置了变量值，那么在独立文件中的变量值有更高优先级。

rc.site 还包含了自定义启动进程的其他参数。设置 IPROMPT 变量可以允许选择运行的启动脚本。其他参数在文件注释中描述。该文件的默认版本如下：

```
# rc.site
# Optional parameters for boot scripts.
# Distro Information
DISTRO="Linux From Scratch" # The distro name
DISTRO_CONTACT="lfs-dev@linuxfromscratch.org" # Bug report
address
DISTRO_MINI="LFS" # Short name used in filenames for distro config
# Define custom colors used in messages printed to the screen
# These values, if specified here, override the defaults
#BRACKET="\\033[1;34m" # Blue
#FAILURE="\\033[1;31m" # Red
#INFO="\\033[1;36m" # Cyan
#NORMAL="\\033[0;39m" # Grey
#SUCCESS="\\033[1;32m" # Green
#WARNING="\\033[1;33m" # Yellow
# Interactive startup
#IPROMPT="yes" # Whether to display the interactive boot promp
itime="10" # The ammount of time (in seconds) to display the prompt
# The total length of the distro welcome string, without escape
codes
wlen=$(echo "Welcome to ${DISTRO}" | wc -c )
welcome_message="Welcome to ${INFO}${DISTRO}${NORMAL}"
# The total length of the interactive string, without escape codes
ilen=$(echo "Press 'I' to enter interactive startup" | wc -c )
i_message="Press '${FAILURE}I${NORMAL}' to enter interactive
startup"
# Set scripts to skip the file system check on reboot
#FASTBOOT=yes
# Skip reading from the console
#HEADLESS=yes
# Skip cleaning /tmp
#SKIPTMPCLEAN=yes
# For setclock
#UTC=1
#CLOCKPARAMS=
# For consolelog
#LOGLEVEL=5
# For network
#HOSTNAME=mylfs
```

```
# Delay between TERM and KILL signals at shutdown
#KILLDELAY=3
# Optional sysklogd parameters
#SYSKLOGD_PARMS="-m 0"
# Console parameters
#UNICODE=1
#KEYMAP="de-latin1"
#KEYMAP_CORRECTIONS="euro2"
#FONT="lat0-16 -m 8859-15"
#LEGACY_CHARSET=
```

7.13. Bash Shell 启动文件

/bin/bash shell 使用了一系列启动文件帮助它创建一个运行环境。每个文件都有它的用途，其中许多会影响登录和交互环境的差异。/etc 目录下的文件直接提供了全局的设置。如果等价的文件存在于家目录下，则会覆盖全局设置。

在使用/bin/login 程序和/etc/passwd 文件成功登陆后，会启动一个交互的登录 shell。一个非登录的交互 shell 可以在命令行启动（如，[prompt]$/bin/bash）。shell 脚本通常在非交互 shell 中运行。它是非交互的因为在脚本运行时不需要获取用户输入。

要获取更多信息，请查看 bash 启动文件和交互 shell 部分下的 **info bash**。

当 shell 以交互登录 shell 形式被调用时会读取/etc/profile 和!/.bash_profile 文件。

/etc/profile 文件设置了支持本地语言的一些环境变量，正确地设置它们可以达到：

- 程序的输出能翻译成本地语言
- 正确地将字符分类为字母，数字和其他类。在非英语环境中，这能帮助 bash 正确地在命令行接受非 ASCII 字符集
- 正确的字母排序
- 合适的页面大小
- 正确的货币，时间，日期格式

将下面的*<ll>*替换为指定语言的两字母代号（如，“en “），将*<CC>*替换为正确国家的两字母代码（如，”GB“），*<charmap>*替换为选择的语言环境的正确字符映射。其他如”@euro“是可选的修饰符。

执行下面的命令可以获得 Glibc 支持的所有语言环境：

```
locale -a
```

字符集映射可能有许多别名，如“ISO-8859-1“也可以叫”iso8859-1“和”iso88591“。然而某些程序可能无法正确识别这些同义词（如，需要”UTF-8“写作”UTF-8“而不是”utf-8“），所以最保险的办法是按照特定语言环境选择正确的名称。运行下面的命令来获取正确名称，其中<locale name>是你选择语言环境（”en_GB.iso88591“）的 locale -a 命令输出：

```
LC_ALL=<locale name> locale charmap
```

若是“en_GB.iso88591“语言环境，那么命令的输出是：

```
ISO-8859-1
```

说明最终语言环境的设置是“en_GB.ISO-8859-1“。使用上述试探的方式获得语言设置是很重要的，下面将它们添加到 bash 启动文件：

```
LC_ALL=<locale name> locale language
LC_ALL=<locale name> locale charmap
LC_ALL=<locale name> locale int_curr_symbol
LC_ALL=<locale name> locale int_prefix
```

上述命令能够打印当前语言环境的语言名称，字符编码，本地货币和该国家的电话号码的拨号前缀。如果上面命令输入和下面类似的失败信息，就意味着你没有在第六章安装你的语言环境或者不被默认 Glibc 所支持：

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

如果发生这种情况，要么需要使用 localedef 命令安装语言环境需要的语言，要么考虑选择不同的语言环境。下面的内容将假设没有错误信息。

LFS 外的一些软件包可能对你选择的语言环境缺乏支持。比如 X 库（X window 系统的部分），如果语言环境无法和它内置的文件的键盘映射完全匹配，它会输出下面的错误信息：

```
Warning: locale not supported by Xlib, locale set to C
```

很多情况下，Xlib 希望字符集以大写符号和减号的列表形象出现。比如，“ISO-8859-1“而不是”iso88591“。通过移除本地语言环境规范的字符集映射部分来获得合适的规范。在两个语言环境中执行命令 locale charmap 可来检查。例如，你可能需要将”de_DE.ISO-8859-15@euro”更改为”de_DE@euro”以便于 Xlib 来识别。

如果语言环境名称不符合某些软件的期望，它们可能也无法正常工作（但也许不会显示错误信息）。在这种情况下，调查其他 Linux 发行版在该语言环境下的运行情况可能对你有所帮助。

一旦设置了正确的语言环境，就可以创建/etc/profile 文件：

```
cat > /etc/profile << "EOF"
# Begin /etc/profile
export LANG=<ll>_<CC>.<charmap><@modifiers>
# End /etc/profile
EOF
```

“C“（默认）和“en_US”（美式英语用户的推荐设置）语言环境是不同的。“C”使用了 US-ASCII7-bit 字符集，并将字节中的高 bit 视为非法字符。这就是为什么 ls 命令在该语言环境中以问号替代某些字符的原因。以及，使用 Mutt 或 Pine 以这些字符尝试发送非 RFC 格式的信息邮件时出现正在发送（已发送邮件的字符集被认为“未知 8bit”）字符集。所以仅当你确认不会使用 8bit 字符时才去使用“C”。

许多程序都无法很好地支持基于 UTF-8 的语言环境。请查阅 *http://www.linuxfromscratch.org/blfs/view/svn/introduction/locale-issues.html* 获悉修复信息。

7.14. 创建/etc/inputrc 文件

/etc/inputrc 文件为特定情形处理键盘映射问题。该文件是 Readline（输入相关的库）的启动文件，它也是 bash 和其他大多数 shell 需要使用的文件。

大多数用户不需要用户特定的键盘映射，所以下面的命令创建了一个全局/etc/inputrc 文件供登入的所有用户使用。如果你需要覆盖该配置，可以在用户家目录下使用修改后的映射文件创建一个新的.*inputrc* 文件。

关于编辑 inputrc 文件的详情，请查阅 **info bash** 的 Readline Init File 部分。**info readline** 也是一个好的信息源。

下面是 inputrc 的一个通用全局配置，注释部分解释了各变量的用途。请注意注释不能和命令位于同一行。

```
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
```

第八章. 使 LFS 系统可启动

8.1. 简介

现在是时候让 LFS 系统可启动了。本章讨论怎样创建 fstab 文件，搭建新 LFS 系统的内核，并安装 GRUB 启动加载器使得在启动时可以选择 LFS 系统。

8.2. 创建/etc/fatab 文件

某些程序使用/etc/fstab 文件判断默认挂载的文件系统，决定挂载顺序，是否需要在挂载前做检查（完整性错误）。按下面的例子创建新的文件系统表：

```
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
proc /proc proc defaults 0 0
sysfs /sys sysfs defaults 0 0
devpts /dev/pts devpts gid=4,mode=620 0 0
tmpfs /run tmpfs defaults 0 0
# End /etc/fstab
EOF
```

将<xxx>，<yyy>，<fff>替换为系统合适的值，如 hda2，hda5,ext3。关于该文件第 6 个字段的详细信息请查阅 **man 5 fstab**。

MS-DOS或Windows（即vfat，ntfs,smbfs,cifs,iso9660,udf）需要挂载参数“iocharset”，否则文件名中的非ASCII字符无法正确解读。该参数的值应该和你选择的语言环境的字符集相同，请做适当调整以便于内核能够识别。如果相应的字符集定义（在File systems -> Native Language Support）被编译到内核或编译为模块时就可以识别。对于vfat和smbfs文件系统还需要”codepage”参数。它的值应该设置为MS-DOS在你所在国家的codepage数。例如，为了挂载USB 闪存设备，一个ru_RU.KOI8-R用户可能需要在/etc/fstab中的挂载行中使用这些参数：

```
noauto,user,quiet,showexec,iocharset=koi8r,codepage=866
```

而对应ru_RU.UTF-8用户的选项可能是：

```
noauto,user,quiet,showexec,iocharset=utf8,codepage=866
```

在后一个例子中，内核可能会发出下列信息：

FAT: utf8 不是 FAT 文件系统推荐的 IO 字符集，文件系统是大小写敏感的！

可以忽略该负面推荐，因为“iocharset”的所有其他值都会导致 UTF-8 语言环境中文件名的错误显示。

在内核配置时也可以指定默认的codepage和iocharset值。相关的参数是“Default NLS

Option" (CONFIG_NLS_DEFAULT), "Default Remote NLS Option" (CONFIG_SMB_NLS_DEFAULT), "Default codepage for FAT" (CONFIG_FAT_DEFAULT_CODEPAGE)，和 "Default iocharset for FAT" (CONFIG_FAT_DEFAULT_IOCHARSET)。在内核编译时不必为ntfs文件系统指定这些参数。

在发生对某些硬盘类型的电源故障时，可以保障ext3文件系统的可靠性。在/etc/fstab合适的位置添加挂载参数barrier=1。若要检查磁盘驱动是否支持该选项，在磁盘驱动上执行hdparm命令。例如，如果：

```
hdparm -I /dev/sda | grep NCQ
```

运行结果没有输出则说明该选项是被支持的。

注意：基于分区的逻辑卷管理（Logical Volume Management，LVM）不能使用barrier参数。

8.3. Linux-3.1

Linux 内核。

估计搭建时间： 1.0-5.0 SBU

需要磁盘空间： 540-800 MB

8.3.1. 安装内核

内核的编译包含了一系列步骤——配置，编译和安装。你可以请阅读内核源码树的 README 文件寻找和本书中不同的方法配置内核。

运行下面的命令准备编译内核：

```
make mrproper
```

该命令将清扫内核源码树获得一个干净内核。内核开发团队建议在内核编译前先执行该命令。不要以为刚解包的内核源码树是干净的。

利用菜单驱动的接口配置内核。要获取内核配置的详情请查看 *http://www.linuxfromscratch.org/hints/downloads/files/kernel-configuration.txt*。关于LFS外的特定内核配置需求请查看BLFS的 *http://www.linuxfromscratch.org/blfs/view/svn/longindex.html#kernel-config-index*。

```
make LANG=<host_LANG_value> LC_ALL= menuconfig
```

make 参数的含义如下：

LANG=<host_LANG_value> LC_ALL=

该参数将按照宿主系统的配置建立语言环境的配置。当 menuconfig 使用 ncurses 接口绘制 UTF-8 的 linux 控制台时需要改配置。请确定用宿主系统的$LANG 的值替换 <host_LANG_VALUE>。如果没有，可以使用$LC_ALL 或$LC_CTYPE 的值替换。

在某些情况下，**make oldconfig** 可能更适合，请查阅 README 获得更多信息。

如果需要，可以从宿主系统复制.config 文件到解压的 linux-3.1 目录，从而跳过内核配置。但我们并不推荐这样做，最好是探索全部的配置菜单并从头创建内核配置。

编译内核映像和模块：

```
make
```

如果使用了内核模块，那么就需要/etc/modprobe.d 的模块配置。关于模块和内核配置的信息位于 7.4 节“LFS 系统的设备和模块管理“以及 linux-3.1/Documentation 目录下的内核文档。甚至 modprobe.conf(5)也是不错的选择。

如果内核需要使用模块就安装模块：

```
make modules_install
```

在完成内核编译后，还需要额外的步骤完成安装。需要复制一些文件到/boot 目录。

根据不同平台内核映像的路径可能不同。另外文件名可以根据你的喜好更改，但文件名的主干必须是 vmlinuz，这样才能和下一部分描述的启动进程的自动配置保持兼容。下面的命令假定是 x86 架构：

```
cp -v arch/x86/boot/bzImage /boot/vmlinuz-3.1-lfs-7.0
```

System.map 是内核的符号文件。它记录了内核 API 的每个函数入口，以及运行中的内核

的数据结构地址。当检查内核 bug 是非常有用。使用下面的命令安装 map 文件：

```
cp -v System.map /boot/System.map-3.1
```

make menuconfig 命令产生的内核配置文件.config 包含了内核配置的所有选择。为了将来考虑最好保留该文件的引用：

```
cp -v .config /boot/config-3.1
```

为 Linux 内核安装文档：

```
install -d /usr/share/doc/linux-3.1
cp -r Documentation/* /usr/share/doc/linux-3.1
```

必须注意到内核源码目录中的文件的拥有主并不是 root。无论何时软件包以 root 用户解压缩（就像我们在 chroot 中做的那样），该文件都拥有了你在软件包的计算机上的用户和组 ID。这对于其他软件包通常都不是问题，因为他们在安装完成后都删除了其源码树。然而，linux 源码树通常会保留很长时间。因此，很可能软件包的用户 ID 被分配给机器上的某个用户。因此该用户就拥有了对源码树的写权限。

如果要保留内核源码树，在 linux-3.1 目录上执行命令 **chown –R 0:0** 确保全部文件的拥有主被改为 root。

Warning

一些内核文档建议创建一个符号链接，从/usr/src/linux 指向内核源码目录。这仅适用于 2.6 之前的系列，不能在 LFS 系统上使用，一旦 LFS 系统完成后会引起软件包编译问题。

Warning

系统头文件目录 include 应该始终和编译 Glibc 时不同，即，解压自 Linux 内核压缩包的内核头文件应消毒。因而，它们绝不能被原内核头文件或其他内核消毒头文件替换。

8.3.2. 配置 Linux 模块加载顺序

如果USB驱动（ehci_hcd, ohci_hcd和uhci_hcd）被编译为模块，就需要创建/etc/modprobe.d/usb.conf文件，它们会以正确的顺序加载；为了避免在启动时输出警告信息，ehci_hcd需要先于ohci_hcd和uhci_hcd加载。

运行下面命令创建/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 包含的内容

安装的文件： config-3.1, vmlinux-3.1-lfs-7.0-3.1, System.map-3.1

安装的目录： /lib/modules, /usr/share/doc/linux-3.1

简要描述

config-3.1 包含了内核的所有配置选项

vmlinux-3.1-lfs-7.0 Linux 系统的引擎。当启动系统时，内核是操作系统首先加载的部分。它探测并初始化所有计算机硬件组件，然后将组件构成文件树的形式以便访问，并将单 CPU 的计算机配置为拥有多任务能力的机器，能够同时运行多个程序

System.map-3.1 地址和符号的列表；它定位内核的函数和数据结构的入口地址

8.4. 使用 GRUB 配置启动进程

8.4.1. 简介

Warning

如果在没有其他启动设备如 CD-ROM 的情况下，错误地配置 GRUB 会导致系统无法继续操作。本部分不要求启动 LFS 系统。你可以仅仅修改当前的 boot loader，即 Grub-Legacy, GRUB2 或 LILO。

确保你拥有一张紧急系统启动盘，当计算机无法启动时可以用来恢复系统。如果没有就制作一张。如果要使用下面的命令制作，就需要先跳到 BLFS 安装 xorriso 程序。

```
cd /tmp &&
grub-mkrescue --output=grub-img.iso &&
xorriso -as cdrecord -v dev=/dev/cdrw blank=as_needed \
grub-img.iso
```

8.4.2. GRUB 的命名约定

GRUB 会在硬盘的第一物理磁道写入数据。该区域不属于任何文件系统。那里的程序会访问启动分区的 GRUB 模块，GRUB 的默认位置为/boot/grub/。

启动分区的位置是用户影响配置的一种选择。推荐的选择是为启动信息专门分一个小区（建议 100MB 大小）。无论是 LFS 或其他商业发行版都可以访问同样的启动文件，而且任何启动的系统都可以访问该分区。如果你这样做了，就需要挂载该独立分区，将当前/boot 目录下（即刚刚编译的 linux 内核）所有文件移动到该分区。然后再卸载该分区并重新挂载到/boot。最后，还需要更新/etc/fstab 文件。

直接使用当前 lfs 分区也是可以的，但为多系统进行配置就麻烦多了。

利用前面的信息，判定根分区的标识符（如果使用分离的启动分区则为该分区）。下面的例子，假定根分区（或分离启动分区）是 sda2。

安装 GRUB 文件到/boot/grub 并设置启动磁道：

Warning

下面的命令会覆盖当前 boot loader。如果不必要请不要执行该命令，如使用第三方启动管理器管理主引导记录（MBR）。

```
grub-install /dev/sda
```

Note

grub-install是一个脚本，它会调用其他程序，grub-probe，它可能会失败并输出信息“cannot stat `/dev/root'”。如果这样，创建一个符号链接从根分区指向/dev/root：

```
ln -sv /dev/sda2 /dev/root
```

该符号链接会存在知道下次重启。它仅在安装程序时需要。

8.4.4. 创建配置文件

生成/boot/grub/grub.cfg：

```
cat > /boot/grub/grub.cfg << "EOF"
# Begin /boot/grub/grub.cfg
set default=0
set timeout=5
insmod ext2
set root=(hd0,2)
menuentry "GNU/Linux, Linux 3.1-lfs-7.0" {
linux /boot/vmlinuz-3.1-lfs-7.0 root=/dev/sda2 ro
}
EOF
```

GRUB 是一款非常强大的程序，它为不同设备，操作系统和分区类型的启动提供了大量选项。同时也提供了大量自定义的选项如，图片闪屏，声音播放，鼠标输入等等。这些参数的详细介绍不在本书范围内。

Note

grub-mkconfig命令可以自动写配置文件。它使用了/etc/grub.d/中的一系列脚本并会破坏你的自定义配置。这些脚本主要设计用于非源码发行版，所以在LFS中并不推荐使用。如果你安装了商业Linux发行版，很有可能该程序可以运行。确保已对gub.cfg文件进行备份。

第九章 结尾

9.1. 结尾

干得漂亮！新的 LFS 系统已经安装完成！我们祝贺你成功编译出这个精美的自定义 Linux 系统。

创建/etc/lfs-release 文件是个好主意。利用该文件，你会很容易（如果你需要向我们的帮助也很容易提供版本号）判断出系统上安装的 LFS 版本。执行下面的命令创建该文件：

```
echo 7.0 > /etc/lfs-release
```

9.2. 计入 LFS 用户

现在你已完成了本书，向成为一名LFS用户吗？前往 *http://www.linuxfromscratch.org/cgi-bin/lfscounter.cgi*并注册成为一名LFS用户，整个过程只需要你输入姓名和你完成的LFS的版本即可。

现在，让我们重启进入 LFS 系统吧！

9.3. 重启系统

既然已经安装好了所有的软件，那么现在可以重启系统了。但是，首先应该注意到一些事情。你按本书制作的系统非常迷你，很可能不具备你需要的功能。通过继续在 chroot 环境中安装 BLFS 中的额外软件包将是更好的选择。安装基于文本模式的网络浏览器，比如 Lynx，你可以在一个虚拟终端中方便地查阅 BLFS，而在另一个终端继续编译软件包。GPM 软件包还能允许你在虚拟终端中执行复制、粘贴操作。最后，如果静态 IP 无法满足你的要求，此时安装譬如 Dhcpcd 或 PPP 之类的软件包会很有帮助。

说了那么多，让我们开始启动我们的精巧 LFS 系统吧！首先推出 chroot 环境：

```
logout
```

然后卸载虚拟文件系统：

```
umount -v $LFS/dev/pts
umount -v $LFS/dev/shm
umount -v $LFS/dev
umount -v $LFS/proc
umount -v $LFS/sys
```

卸载 LFS 文件系统：

```
umount -v $LFS
```

如果创建了多个分区，在卸载注意分区前先卸载其他分区：

```
umount -v $LFS/usr
umount -v $LFS/home
umount -v $LFS
```

现在，重启系统：

```
shutdown -r now
```

假设 GRUB 的 boot loader 像前面讲述的那样配置，并且菜单被设置为自动启动 LFS7.0。当完成重启后，就可以使用 LFS 系统并按你的需要安装更多的软件了。

9.4. 下一步做什么？

感谢您阅读本书。希望您觉得本书有点用并且从中学到了系统创建的知识。

现在 LFS 系统已经安装完成，你可能想知道“接下来该干什么？”，为回答这个问题，我们为此准备了一个列表：

- 维护

  所有软件都有 bug 和安全警告报告。由于 LFS 是从源代码编译得来，当然应该由你负责这份报告。下面是跟踪这种报告的在线资源，下面是其中一些：

  - Freshmeat.net(*http://freshmeat.net/*)

    Freshmeat 会以邮件通知您在您系统上的软件包的新版本。
  - CERT（计算机紧急相应团队）

    CERT 有一个邮件列表，专门用于发布操作系统和程序的安全警报。

    http://www.us-cert.gov/cas/signup.html 上是它的订阅信息。
  - Bugtraq

    Bugtraq是一个完全开放的计算机安全邮件列表。它发布新发现的安全漏洞，和偶然修复的方法。*http://www.securityfocus.com/archive*上是他的订阅信息。
- Beyond Linux From Scratch

  Beyond Linux From Scratch 是 LFS 之后的书籍，包含了大量 LFS 外的软件的安装说明。BLFS 项目位于 *http://www.linuxfromscratch.org/hints/list.html*。
- LFS 提示

  LFS 提示是一系列教育文档，它们由 LFS 社区的志愿者提供。

  http://www.linuxfromscratch.org/hints/list.html 上是这些提示。
- 邮件列表

  如果你需要帮助，和最新的开放近距离接触，为该项目做贡献等等，这里有许多 LFS 邮件列表。查看第一章——在邮件列表中获得更多信息。
- Linux 文档项目

  Linux文档项目（The Linux Documentation Project, TLDP）是为了所有Linux文档事宜的合作。TLDP集合了大量了“怎么做”，指导和man手册。它位于*http://www.tldp.org/*。

第五部分　附录

略