IV. Сборка системы LFS. Глава 8. Установка базового системного программного обеспечения

Содержание

8.1. Введение

В этой главе мы приступаем к сборке конечной системы LFS.

Установка программного обеспечения проста. Хотя во многих случаях инструкции по установке можно было бы сделать короче и универсальнее, мы решили предоставить полные инструкции для каждого пакета, чтобы свести к минимуму вероятность ошибок. Ключом к пониманию того, что заставляет систему Linux работать, является знание того, для чего используется каждый пакет и зачем он вам (или системе) может понадобиться.

Мы не рекомендуем использовать оптимизации. С ними программа может работать немного быстрее, но также они могут вызвать сложности при компиляции и проблемы при запуске программы. Если пакет не компилируется при использовании оптимизации, попробуйте скомпилировать его без оптимизации и посмотрите, решает ли это проблему. Даже если пакет компилируется при использовании оптимизации, существует риск, что он может быть скомпилирован неправильно из-за сложных взаимодействий между кодом и инструментами сборки. Также обратите внимание, что параметры -march и -mtune, не тестировались со значениями отличными от указанных в книге. Это может вызвать проблемы с пакетами набора инструментов (Binutils, GCC и Glibc). Небольшие потенциальные плюсы, достигаемые за счет оптимизации, часто перевешиваются рисками. Тем кто собирает LFS впервые рекомендуется делать это без пользовательских оптимизаций.

С другой стороны, мы сохраняем оптимизацию включенной в конфигурации пакетов по умолчанию. Кроме того, иногда мы явно включаем оптимизированную конфигурацию, предоставляемую пакетом, но не включенную по умолчанию. Сопровождающие пакета уже протестировали эти конфигурации и считают их безопасными, поэтому маловероятно, что они сломают сборку. Как правило, конфигурация по умолчанию уже включает параметры -O2 или -O3, поэтому результирующая система по-прежнему будет работать очень быстро без какой-либо пользовательской оптимизации и в то же время будет стабильной.

Перед инструкцией по установке на каждой странице представлена информация о пакете, включая краткое описание того, что он содержит, примерное время, необходимое для сборки, и сколько места на диске требуется в процессе сборки. После инструкции по установке идет список программ и библиотек (вместе с кратким описанием), которые устанавливает пакет.

Примечание
Для всех пакетов в Главе 8 значения SBU и требуемое дисковое пространство указано с учетом тестов. Значения SBU были рассчитаны с использованием четырех ядер ЦП (-j4) для всех операций, если не указано иное.

8.1.1. О библиотеках

Как правило, редакторы LFS не рекомендуют собирать и устанавливать статические библиотеки. Большинство статических библиотек устарели в современной системе Linux. Кроме того, линковка статической библиотеки с программой может быть вредна. Если для устранения проблемы безопасности требуется обновление библиотеки, все программы, использующие статическую библиотеку, необходимо будет повторно перелинковать с новой библиотекой. Поскольку использование статических библиотек не всегда очевидно, соответствующие программы (и процедуры, необходимые для линковки) могут быть даже неизвестны.

В инструкциях этой главы мы удаляем или отключаем установку большинства статических библиотек. Обычно это делается путем передачи параметра –disable-static при выполнении configure. Иногда необходимо использовать альтернативные методы. В некоторых случаях, в частности в пакетах Glibc и GCC, использование статических библиотек остается важным элементом процесса сборки пакетов.

Более подробное обсуждение библиотек смотрите Библиотеки: статические или общие? в книге BLFS.

8.2. Управление пакетами

Управление пакетами — часто спрашиваемое дополнение к книге LFS. Менеджер пакетов позволяет отслеживать установку файлов, упрощая удаление и обновление пакетов. Хороший менеджер пакетов также будет обрабатывать конфигурационные файлы, чтобы сохранить пользовательские настройки при переустановке или обновлении пакета. Прежде чем вы начнете задаваться вопросом, НЕТ—в этом разделе не будет ни говориться, ни рекомендоваться какой-либо конкретный менеджер пакетов. Что он действительно предоставляет, так это обзор наиболее популярных методов и того, как они работают. Идеальным менеджером пакетов для вас может быть один из этих методов или комбинация двух и более методов. В этом разделе кратко упоминаются проблемы, которые могут возникнуть при обновлении пакетов.

Некоторые причины, по которым менеджер пакетов не упоминается в LFS или BLFS представлены ниже:

- Рассмотрение управления пакетами отвлекает внимание от целей этих книг—обучения тому, как строится система Linux.
- Существует множество решений для управления пакетами, каждое из которых имеет свои сильные и слабые стороны. Трудно найти такое, которое удовлетворит всех.

Есть несколько советов, написанных на тему управления пакетами. Посетите проект Советы возможно вы найдете решение, которое соответствует вашим потребностям.

8.2.1. Проблемы с обновлением

Менеджер пакетов упрощает обновление до более новых версий после их выпуска. Как правило, инструкции в книгах LFS и BLFS можно использовать для обновления до более новых версий. Вот некоторые моменты, о которых следует помнить при обновлении пакетов, особенно в работающей системе.

- Если нужно обновить ядро Linux (например, с 5.10.17 до 5.10.18 или 5.11.1), дополнительно пересобирать ничего не нужно. Система продолжит нормально работать благодаря четко определенной границе между ядром и пользовательским пространством. В частности, заголовки Linux API не нужно обновлять вместе с ядром. Вам просто нужно перезагрузить систему, чтобы использовать обновленное ядро.
- Если необходимо обновить Glibc до более новой версии (например, с Glibc-2.36 до Glibc-2.39) необходимо выполнить некоторые дополнительные действия, чтобы избежать поломки системы. Подробности читайте в Разделе 8.5. «Glibc-2.39».
- Если пакет, содержащий общую библиотеку, обновляется и имя библиотеки изменилось, то любые пакеты, динамически связанные с библиотекой, необходимо перекомпилировать, чтобы связать с более новой библиотекой. (Обратите внимание, что между версией пакета и именем библиотеки нет никакой связи.) Например, рассмотрим пакет foo-1.2.3, который устанавливает общую библиотеку с именем libfoo.so.1.

Предположим, вы обновили пакет до более новой версии foo-1.2.4, которая устанавливает общую библиотеку с именем libfoo.so.2, все пакеты, которые динамически связаны с libfoo.so.1, должны быть перекомпилированы для связи с libfoo.so.2, чтобы использовать новую версию библиотеки. Вы не должны удалять старые библиотеки, пока все зависимые пакеты не будут перекомпилированы.

- Если пакет (прямо или косвенно) связан как со старым, так и с новым именем общей библиотеки (например, пакет ссылается как на libfoo.so.2, так и на libbar.so.1, в то время как последний ссылается на libfoo.so.3), пакет может работать неправильно, поскольку разные версии общей библиотеки содержат несовместимые определения для некоторых имен символов. Это может быть вызвано перекомпиляцией некоторых, но не всех, пакетов, связанных со старой общей библиотекой, после обновления пакета, предоставляющего общую библиотеку. Чтобы избежать этой проблемы, пользователям необходимо как можно скорее пересобрать каждый пакет, связанный с общей библиотекой, с обновленной версией (например, с libfoo.so.2 на libfoo.so.3).

- Если пакет, содержащий общую библиотеку, обновляется, а имя библиотеки не меняется, но уменьшается номер версии файла библиотеки (например, библиотека по-прежнему называется libfoo.so.1, но имя файла библиотеки изменилось с libfoo.so.1.25 на libfoo.so.1.24), следует удалить файл библиотеки ранее установленной версии (в данном случае libfoo.so.1.25). В противном случае, команда ldconfig (запущенная самостоятельно с помощью командной строки или при установке какого-либо пакета) приведёт к сбросу символической ссылки libfoo.so.1, которая будет указывать на старый файл библиотеки, потому что кажется, что она имеет «более новую» версию, поскольку её номер версии больше. Такая ситуация может произойти, если вам нужно понизить версию пакета или авторы изменили схему управления версиями файлов библиотеки.

- Если пакет, содержащий общую библиотеку, обновляется, а имя библиотеки не меняется, но устраняется серьезная проблема (особенно уязвимость в системе безопасности), необходимо перезапустить все работающие программы, связанные с общей библиотекой. Следующая команда, запущенная от имени пользователя root после завершения обновления, выведет список программ, которые использует старые версии этих библиотек (замените libfoo именем библиотеки):

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

Если для доступа к системе используется OpenSSH и он связан с обновленной библиотекой, вам необходимо перезапустить службу sshd, затем выйти из системы, снова войти в систему и повторно выполнить предыдущую команду, чтобы убедиться, что удаленные библиотеки более не используются.

Если демон **systemd** (работающий как PID 1) связан с обновленной библиотекой, вы можете перезапустить его без перезагрузки, запустив systemctl daemon-reexec от имени пользователя root.

- Если исполняемая программа или библиотека перезаписаны, процессы, использующие код или данные из них, могут завершиться сбоем. Правильный способ обновить программу или общую библиотеку, не вызывая сбоя процесса, - это сначала удалить его, а затем установить новую версию. Команда install, предоставляемая Coreutils, уже реализовала это, и большинство пакетов используют ее для установки двоичных файлов и библиотек. Это означает, что большую часть времени вас не будет беспокоить эта

проблема. Однако процесс установки некоторых пакетов (в частности, SpiderMonkey в BLFS) просто перезаписывает файл, если он существует, и вызывает сбой. Поэтому безопаснее сохранить свою работу и закрыть ненужные запущенные программы перед обновлением пакета.

8.2.2. Методы управления пакетами

Ниже приведены некоторые распространенные методы управления пакетами. Прежде чем принять решение о менеджере пакетов, проведите исследование различных методов, особенно недостатки каждой конкретной схемы.

8.2.2.1. Всё у меня в голове!

Да, это метод управления пакетами. Некоторым людям не нужен менеджер пакетов, потому что они хорошо знакомы с пакетами и знают, какие файлы устанавливаются каждым пакетом. Некоторым пользователям также не требуется какое-либо управление пакетами, поскольку они планируют пересобирать всю систему при каждом изменении пакета.

8.2.2.2. Установка в отдельные каталоги

Это упрощенный метод управления пакетами, для которого не требуется специальная программа управления. Каждый пакет устанавливается в отдельный каталог. Например, пакет **foo-1.1** устанавливается в **/opt/foo-1.1**, а символическая ссылка создается из **/opt/foo** в **/opt/foo-1.1**. Когда появляется новая версия **foo-1.2**, она устанавливается в **/opt/foo-1.2** и предыдущая символическая ссылка заменяется символической ссылкой на новую версию.

Переменные окружения, такие как **PATH, MANPATH, INFOPATH, PKG_CONFIG_PATH, CPPFLAGS, LDFLAGS** и файл конфигурации **/etc/ld.so.conf**, возможно, потребуется расширить, включив соответствующие подкаталоги в **/opt/foo-x.y**.

Этот подход используется в книге BLFS для установки некоторых очень больших пакетов, чтобы упростить их обновление. Если вы устанавливаете много таких пакетов, эта схема становится неуправляемой. Некоторые пакеты (например, заголовки Linux API и Glibc) могут плохо работать с такой структурой. **Никогда не используйте её в масштабах всей системы.**

8.2.2.3. Управление пакетами с использованием символических ссылок

Это разновидность предыдущей техники.Каждый пакет устанавливается аналогично, но вместо создания символической ссылки на общее имя пакета, каждому файлу создаётся символическая ссылка в иерархии каталогов **/usr**. Это исключает необходимость модификации значений переменных окружения. Хотя такие ссылки могут быть созданы пользователем, многие менеджеры пакетов используют именной такой подход. Наиболее популярные из них - **Stow, Epkg, Graft и Depot**.

Установку нужно сымитировать, чтобы пакет думал, что он установлен в **/usr**, хотя на самом

деле он установлен в иерархии **/usr/pkg**. Установка таким способом обычно является нетривиальной задачей. Например, предположим, что вы устанавливаете пакет **libfoo-1.1**. Следующие инструкции могут привести к неправильной установке пакета:

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

Установка будет выполнена, но зависимые пакеты не смогут ссылаться на libfoo. Если вы скомпилируете пакет, который ссылается на **libfoo**, вы заметите, что он связан с **/usr/pkg/libfoo/1.1/lib/libfoo.so.1** вместо **/usr/lib/libfoo.so.1**, как вы ожидаете. Правильный подход заключается в использовании переменной DESTDIR для управления установкой. Этот подход работает следующим образом:

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

8.2.2.4. На основе временной метки

В этом методе файлу присваивается временная метка перед установкой пакета. После установки простое использование команды **find** с соответствующими параметрами может создать журнал всех файлов, установленных после создания файла с временной метки. Менеджером пакетов, использующим этот подход, является **install-log**.

Хотя преимущество этой схемы в том, что она проста, у нее есть два недостатка. Если во время установки, файлы устанавливаются с отметкой времени, отличной от текущего времени, эти файлы не будут отслеживаться менеджером пакетов. Кроме того, эта схема может использоваться только при установке пакетов по одному. Журналы ненадежны, если два пакета устанавливаются одновременно на двух разных консолях.

8.2.2.5. Отслеживание сценариев установки

При таком подходе, записываются команды, выполняемые сценариями установки. Есть два метода, которые можно использовать:

Переменная среды **LD_PRELOAD** может быть установлена так, чтобы она указывала на библиотеку, которую нужно предварительно загрузить перед установкой. Во время установки эта библиотека отслеживает устанавливаемые пакеты, присоединяясь к различным исполняемым файлам, таким как **cp**, **install**, **mv**, и отслеживая системные вызовы, изменяющие файловую систему. Чтобы этот подход работал, все исполняемые файлы должны быть динамически связаны без битов **suid** или **sgid**. Предварительная загрузка библиотеки может вызвать некоторые нежелательные побочные эффекты во время установки. Поэтому рекомендуется выполнить некоторые тесты, чтобы убедиться, что менеджер пакетов ничего не сломает и что он регистрирует все соответствующие файлы.

Другой метод заключается в использовании **strace**, который регистрирует все системные

вызовы, сделанные во время выполнения сценариев установки.

8.2.2.6. Создание архивов пакетов

В этой схеме установка пакета имитируется в отдельном дереве, как описано ранее в разделе управление пакетами с использованием символических ссылок. После установки из установленных файлов создается архив пакета. Затем этот архив используется для установки пакета на локальный компьютер или даже на другие компьютеры.

Этот подход используется большинством менеджеров пакетов, имеющихся в коммерческих дистрибутивах. Примерами менеджеров пакетов, которые следуют этому подходу, являются RPM (который, кстати, требуется согласно спецификации Linux Standard Base Specification), pkg-utils, apt Debian и система Portage Gentoo. Описание того, как использовать этот стиль управления пакетами для систем LFS, находится по адресу https://mirror.linuxfromscratch.ru/hints/downloads/files/fakeroot.txt.

Создание файлов пакетов, содержащих информацию о зависимостях, является сложной задачей и выходит за рамки LFS.

Slackware использует систему на основе tar для архивов пакетов. Эта система намеренно не обрабатывает зависимости пакетов, как это делают более сложные менеджеры пакетов. Подробнее об управлении пакетами Slackware см. https://www.slackbook.org/html/package-management.html.

8.2.2.7. Пользовательское управление пакетами

Эта схема, уникальная для LFS, была разработана Маттиасом Бенкманом и доступна в проекте Hints. В этой схеме каждый пакет устанавливается отдельным пользователем в стандартные папки. Файлы, принадлежащие пакету, легко идентифицируются путем проверки идентификатора пользователя. Особенности и недостатки этого подхода слишком сложны, чтобы описывать их в этом разделе. Для получения более подробной информации, пожалуйста, ознакомьтесь с советами по адресу https://mirror.linuxfromscratch.ru/hints/downloads/files/more_control_and_pkg_man.txt.

8.2.3. Развертывание LFS на нескольких системах

Одним из преимуществ системы LFS является отсутствие файлов, зависящих от положения файлов на диске. Клонировать сборку LFS на другой компьютер с той же архитектурой, что и у базовой системы, так же просто, как использовать tar для архивации раздела LFS, содержащем корневой каталог (около 900 МБ в несжатом виде для базовой сборки LFS), скопировать этот файл по сети или с помощью CD / USB носителя в новую систему и распаковать его. После этого необходимо изменить несколько конфигурационных файлов. Файлы, которые, возможно, потребуется изменить представлены в списке ниже: **/etc/hosts, /etc/fstab, /etc/passwd, /etc/group, /etc/shadow,** и **/etc/ld.so.conf**.

Возможно, потребуется собрать собственное ядро для новой системы в зависимости от различий в системном оборудовании и исходной конфигурации ядра.

Примечание
Поступали некоторые сообщения о проблемах при копировании между похожими, но не идентичными архитектурами. Например, набор инструкций для Intel не идентичен набору инструкций для процессора AMD, и более поздние версии некоторых процессоров могут содержать инструкции, недоступные в более ранних версиях.

Наконец, новую систему необходимо сделать загрузочной так, как это описано в Разделе 10.4. «Использование GRUB для настройки процесса загрузки»

8.3. Man-pages-6.06

Пакет Man-pages содержит более 2400 справочных руководств.	
Приблизительное время сборки:	менее 0.1 SBU
Требуемое дисковое пространство:	33 MB

8.3.1. Установка пакета Man-pages

Удалите две справочные страницы для функций хэширования паролей. Libxcrypt предоставит улучшенную версию этих справочных страниц:

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

Установите пакет Man-pages выполнив команду:

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

8.3.2. Содержимое пакета Man-pages

Установленные файлы:	различные справочные страницы

Краткое описание

man pages	Описывают функции языка программирования C, важные файлы устройств и важные файлы конфигурации.

8.4. Iana-Etc-20240125

Пакет Iana-Etc предоставляет данные для сетевых служб и протоколов.	
Приблизительное время сборки:	менее 0.1 SBU
Требуемое дисковое пространство:	4.8 MB

8.4.1. Установка пакета Iana-Etc

Для этого пакета необходимо лишь скопировать нужные файлы:

```
cp services protocols /etc
```

8.4.2. Содержимое пакета Iana-Etc

Установленные файлы:	/etc/protocols и /etc/services

Краткое описание

/etc/protocols	Описывает различные интернет-протоколы DARPA, которые доступны из подсистемы TCP/IP
/etc/services	Обеспечивает сопоставление понятных текстовых имен для интернет-сервисов с назначенными им номерами портов и типами протоколов.

8.5. Glibc-2.39

Пакет Glibc содержит основную библиотеку C. Эта библиотека предоставляет основные процедуры для выделения памяти, поиска в каталогах, открытия и закрытия файлов, чтения и записи файлов, обработки строк, сопоставления с образцом, арифметики и так далее	
Приблизительное время сборки:	1.2 SBU
Требуемое дисковое пространство:	3.1 GB

8.5.1. Установка пакета Glibc

Некоторые программы Glibc используют не совместимый с FHS каталог /var/db для хранения своих данных во время выполнения. Примените следующий патч, чтобы эти программы хранили свои данные в каталогах, совместимых с FHS:

```
patch -Np1 -i ../glibc-2.39-fhs-1.patch
```

Документация Glibc рекомендует выполнять компиляцию в отдельном каталоге:

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

Убедитесь, что утилиты ldconfig и sln будут установлены в /usr/sbin:

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

Подготовьте Glibc к компиляции:

```
../configure --prefix=/usr                            \
             --disable-werror                         \
             --enable-kernel=4.19                     \
             --enable-stack-protector=strong          \
```

```
            --disable-nscd                          \
            libc_cv_slibdir=/usr/lib
```

Значение параметров настройки:

- **–disable-werror**

Отключает параметр -Werror, передаваемый GCC. Это необходимо для запуска набора тестов.

- **–enable-kernel=4.19**

Этот параметр сообщает системе сборки, что Glibc может использоваться с ядрами старше 4.19. Это значение используется для создание обходных путей на случай, если системный вызов, представленный в более поздней версии, нельзя будет использовать.

- **–enable-stack-protector=strong**

Этот параметр повышает безопасность системы за счет добавления дополнительного кода для проверки переполнения буфера. Обратите внимание, что Glibc всегда явно переопределяет параметры GCC по умолчанию, поэтому необходимо всегда указывать эту опцию, несмотря на то, что мы уже указали –enable-default-ssp для GCC.

- **–disable-nscd**

Параметр отключает сборку демона кэша службы имен, который больше не используется.

- **libc_cv_slibdir=/usr/lib**

Эта переменная устанавливает правильную библиотеку для всей системы. Мы не хотим, чтобы использовалась lib64

Скомпилируйте пакет:

```
make
```

Важно
В этом разделе набор тестов для Glibc считается критически важным. Ни в коем случае не пропускайте его.

Как правило, несколько тестов не проходят. Ошибки тестирования, перечисленные ниже, можно игнорировать.

```
make check
```

Вы можете увидеть, что ряд тестов завершились неудачей. Набор тестов Glibc в некоторой степени зависит от хост-системы. Несколько ошибок из более чем 5000 тестов можно игнорировать. Список наиболее распространенных проблем последних версий LFS:

- Известно, что **io/tst-lchmod** не работает в среде **chroot LFS**.
- Известно, что некоторые тесты, например **nss/tst-nss-files-hosts-multi и nptl/tst-thread-affinity** завершаются неудачей из-за тайм-аута (особенно когда система работает относительно медленно и/или набор тестов запущен в несколько потоков). Эти тесты могут быть идентифицированы с помощью следующей команды:

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

Можно повторно запустить отдельный тест, увеличив таймаут с помощью команды **TIMEOUTFACTOR=< class=""> make test t=<test name>**. Например, **TIMEOUTFACTOR=10 make test t=nss/tst-nss-files-hosts-multi** перезапустит **nss/tst-nss-files-hosts-multi**, увеличив начальный таймаут в 10 раз.

- Кроме того, некоторые тесты могут завершиться неудачно при использовании относительно старой модели процессора (например, **elf/tst-cpu-features-cpuinfo**) или версии ядра хоста (например, **stdlib/tst-arc4random-thread**).

На этапе установки **Glibc** будет жаловаться на отсутствие файла **/etc/ld.so.conf**, хотя это безобидное сообщение, предотвратить его появление можно с помощью команды:

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

Исправьте Makefile, чтобы пропустить устаревшую проверку работоспособности, которая завершается неудачей в современной конфигурации Glibc:

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

Важно
При обновлении Glibc до новой минорной версии (например, с Glibc-2.36 до Glibc-2.39) в работающей системе LFS вам необходимо принять некоторые дополнительные меры предосторожности, чтобы избежать поломки системы:

- Обновление Glibc в системе LFS до версии 11.0 не поддерживается. Пересоберите LFS, если вы используете такую старую систему, но вам нужна более новая Glibc.
- При обновлении системы LFS до версии 12.0 установите Libxcrypt следуя инструкции Раздел 8.26, «Libxcrypt-4.4.36.». В дополнение к обычной установке Libxcrypt, **вы ДОЛЖНЫ следовать примечанию со страницы Libxcrypt, чтобы установить libcrypt.so.1* (заменив libcrypt.so.1 из предыдущей установки Glibc).**
- При обновлении системы LFS до версии 12.1 удалите программу nscd:

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

Если система (до LFS 12.1) основана на Systemd, необходимо также отключить и остановить службу nscd прямо сейчас:

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

- Обновите ядро и перезагрузитесь, если оно старше 4.19 (проверьте текущую

версию с помощью uname -r) или, если вы хотите обновить имеющееся ядро, выполните действия из Раздел 10.3, «Linux-6.7.4.»
- Обновите заголовочные файлы API ядра, если они старше 4.19 (проверьте текущую версию с помощью cat /usr/include/linux/version.h) или, если вы просто хотите обновить их, следуйте Раздел 5.4, «Заголовочные файлы Linux-6.7.4 API» (но удалив $LFS из команды cp).
- Выполните установку DESTDIR и обновите общие библиотеки Glibc в системе с помощью одной команды install:

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

Крайне важно строго следовать описанным выше шагам, если вы не совсем понимаете, что делаете. **Любое неожиданное отклонение может сделать систему полностью непригодной для использования. ВЫ ПРЕДУПРЕЖДЕНЫ.** Затем продолжайте выполнять команды **make install, sed** для **/usr/bin/ldd** и команды для установки локалей. Как только они будут выполнены, перезагрузите систему.

Установите пакет:

```
make install
```

Исправьте жестко заданный путь к исполняемому загрузчику в скрипте ldd:

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

Затем установите локали, которые дадут возможность системе отвечать на разных языках. Ни одна из локалей не требуется системе, но если некоторые из них отсутствуют, то наборы тестов ряда пакетов будут пропускать важные тестовые сценарии.

Отдельные локали можно установить с помощью программы **localedef**. Например, вторая команда **localedef** приведенная ниже, объединяет определение независимой от набора символов локали **/usr/share/i18n/locales/cs_CZ** с набором символов **/usr/share/i18n/charmaps/UTF-8.gz** и добавляет результат в файл **/usr/lib/locale/locale-archive**. Следующие инструкции установят минимальный набор локалей, необходимый для оптимального охвата тестов

```
mkdir -pv /usr/lib/locale
localedef -i C -f UTF-8 C.UTF-8
localedef -i cs_CZ -f UTF-8 cs_CZ.UTF-8
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
localedef -i de_DE -f UTF-8 de_DE.UTF-8
localedef -i el_GR -f ISO-8859-7 el_GR
localedef -i en_GB -f ISO-8859-1 en_GB
localedef -i en_GB -f UTF-8 en_GB.UTF-8
localedef -i en_HK -f ISO-8859-1 en_HK
```

```
localedef -i en_PH -f ISO-8859-1 en_PH
localedef -i en_US -f ISO-8859-1 en_US
localedef -i en_US -f UTF-8 en_US.UTF-8
localedef -i es_ES -f ISO-8859-15 es_ES@euro
localedef -i es_MX -f ISO-8859-1 es_MX
localedef -i fa_IR -f UTF-8 fa_IR
localedef -i fr_FR -f ISO-8859-1 fr_FR
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i fr_FR -f UTF-8 fr_FR.UTF-8
localedef -i is_IS -f ISO-8859-1 is_IS
localedef -i is_IS -f UTF-8 is_IS.UTF-8
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i it_IT -f ISO-8859-15 it_IT@euro
localedef -i it_IT -f UTF-8 it_IT.UTF-8
localedef -i ja_JP -f EUC-JP ja_JP
localedef -i ja_JP -f SHIFT_JIS ja_JP.SJIS 2> /dev/null || true
localedef -i ja_JP -f UTF-8 ja_JP.UTF-8
localedef -i nl_NL@euro -f ISO-8859-15 nl_NL@euro
localedef -i ru_RU -f KOI8-R ru_RU.KOI8-R
localedef -i ru_RU -f UTF-8 ru_RU.UTF-8
localedef -i se_NO -f UTF-8 se_NO.UTF-8
localedef -i ta_IN -f UTF-8 ta_IN.UTF-8
localedef -i tr_TR -f UTF-8 tr_TR.UTF-8
localedef -i zh_CN -f GB18030 zh_CN.GB18030
localedef -i zh_HK -f BIG5-HKSCS zh_HK.BIG5-HKSCS
localedef -i zh_TW -f UTF-8 zh_TW.UTF-8
```

Кроме того, установите локаль для вашей страны, языка и набора символов.

В качестве альтернативы, установите сразу все локали перечисленные в файле **glibc-2.39/localedata/SUPPORTED** (он включает все локали из списка выше и многие другие), выполнив команду:

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

Затем используйте команду **localedef** для создания и установки локалей, не перечисленных в файле **glibc-2.39/localedata/SUPPORTED**, когда они вам понадобятся. Например, для некоторых тестов в этой главе потребуются следующие две локали:

```
localedef -i C -f UTF-8 C.UTF-8
localedef -i ja_JP -f SHIFT_JIS ja_JP.SJIS 2> /dev/null || true
```

Примечание
Glibc теперь использует libidn2 для разрешения интернационализированных доменных имен. Если такая функция необходима, то инструкцию по установке libidn2 можно найти на странице BLFS libidn2.

8.5.2. Настройка Glibc

8.5.2.1. Добавление nsswitch.conf

Необходимо создать файл **/etc/nsswitch.conf**, потому что настроенный по умолчанию Glibc плохо работает в сетевой среде.

Создайте новый фай **/etc/nsswitch.conf**, выполнив следующие действия:

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

passwd: files systemd
group: files systemd
shadow: files systemd

hosts: mymachines resolve [!UNAVAIL=return] files myhostname dns
networks: files

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

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

8.5.2.2. Добавление данных о часовом поясе

Установите и настройте часовой пояс следующим образом:

```
tar -xf ../../tzdata2024a.tar.gz

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

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

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

Значение команд zic:

- **zic -L /dev/null …**

Создаёт часовые пояса posix без секунд координации. Обычно их помещают как в zoneinfo так и в zoneinfo/posix. Часовые пояса POSIX должны быть прописаны в zoneinfo, иначе различные тесты будут сообщать об ошибках. На встраиваемых системах с небольшим диском, где часовые пояса никогда не будут обновляться, можно сэкономить примерно 1.9 MB не используя каталог posix, однако некоторые приложения или наборы тестов могут вызывать сбои.

- **zic -L leapseconds …**

Создаёт правильные часовые пояса с секундами координации. На встраиваемых системах с небольшим диском, где часовые пояса никогда не будут обновляться, а правильность времени неважна, можно выиграть примерно 1.9 MB, исключив каталог right.

- **zic … -p …**

Создаёт файл posixrules. Используется New York, потому что POSIX требует, чтобы правила перехода на летнее время соответствовали правилам США.

Один из способов определить местный часовой пояс — запустить следующий скрипт:

```
tzselect
```

После нескольких вопросов о местоположении скрипт выдаст наименование часового пояса (например **America/Edmonton**). В файле **/usr/share/zoneinfo** перечислены и другие возможные часовые пояса, такие как **Canada/Eastern** или **EST5EDT**, которые не распознаются скриптом, но могут быть использованы.

Создайте файл **/etc/localtime** выполнив:

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

Замените <xxx> на имя выбранного часового пояса (например, Europe/Moscow).

8.5.2.3. Настройка динамического загрузчика

По умолчанию, динамический загрузчик (**/lib/ld-linux.so.2**) ищет в каталоге **/usr/lib**, нужные для работы программ библиотеки. Однако, если библиотеки находятся в другом каталоге, то его необходимо указать в файле **/etc/ld.so.conf**, чтобы динамический загрузчик мог их найти. Два каталога - **/usr/local/lib** и **/opt/lib** часто используются для дополнительных библиотек, поэтому добавьте их в пути поиска для динамического загрузчика.

Создайте новый файл **/etc/ld.so.conf** выполнив:

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

```
/opt/lib

EOF
```

Динамический загрузчик может выполнить поиск в каталоге и включить содержимое найденных там файлов. Обычно такие файлы состоят из одной строки и содержат путь к библиотеке. Чтобы добавить эту возможность, выполните следующие команды:

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

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

8.5.3. Содержимое пакета Glibc

Установленные файлы:	gencat, getconf, getent, iconv, iconvconfig, ldconfig, ldd, lddlibc4, ld.so (symlink to ld-linux-x86-64.so.2 or ld-linux.so.2), locale, localedef, makedb, mtrace, pcprofiledump, pldd, sln, sotruss, sprof, tzselect, xtrace, zdump и zic
Установленные библиотеки:	ld-linux-x86-64.so.2, ld-linux.so.2, libBrokenLocale.{a,so}, libanl.{a,so}, libc.{a,so}, libc_nonshared.a, libc_malloc_debug.so, libdl.{a,so.2}, libg.a, libm.{a,so}, libmcheck.a, libmemusage.so, libmvec.{a,so}, libnsl.so.1, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libpcprofile.so, libpthread.{a,so.0}, libresolv.{a,so}, librt.{a,so.1}, libthread_db.so и libutil.{a,so.1}
Созданные каталоги:	/usr/include/arpa, /usr/include/bits, /usr/include/gnu, /usr/include/net, /usr/include/netash, /usr/include/netatalk, /usr/include/netax25, /usr/include/neteconet, /usr/include/netinet, /usr/include/netipx, /usr/include/netiucv, /usr/include/netpacket, /usr/include/netrom, /usr/include/netrose, /usr/include/nfs, /usr/include/protocols, /usr/include/rpc, /usr/include/sys, /usr/lib/audit, /usr/lib/gconv, /usr/lib/locale, /usr/libexec/getconf, /usr/share/i18n, /usr/share/zoneinfo и /var/lib/nss_db

Краткое описание

gencat	Создает каталоги сообщений
getconf	Отображает настройки системы для специфичных переменных файловой системы
getent	Получает записи из административной базы данных
iconv	Выполняет преобразование набора символов
iconvconfig	Создает быстрозагружаемые файлы настроек модуля iconv
ldconfig	Настраивает привязки времени выполнения динамического компоновщика
ldd	Сообщает, какие общие библиотеки требуются каждой программе или общей библиотеке
lddlibc4	Помогает ldd работать с объектными файлами. Он не существует на более новых архитектурах, таких как x86_64
locale	Выводит различную информацию о текущей локали
localedef	Компилирует спецификации локали

makedb	Создает простую базу данных на основе текстового ввода
mtrace	Читает и интерпретирует файл трассировки памяти; отображает сводку в удобочитаемом формате
pcprofiledump	Создает дамп информации, генерируемой при профилировании ПК
pldd	Перечисляет динамические общие объекты, используемые запущенными процессами.
sln	Статически скомпонованная программа ln
sotruss	Отслеживает вызовы процедур общей библиотеки указанной команды
sprof	Читает и отображает данные профилирования общих объектов.
tzselect	Запрашивает у пользователя информацию о текущем местоположении системы и выводит описание соответствующего часового пояса.
xtrace	Отслеживает выполнение программы, отображая выполняемую в данный момент функцию
zdump	Выдает дамп часового пояса
zic	Компилятор часовых поясов
ld-*.so	Вспомогательная программа для исполняемых файлов общей библиотеки
libBrokenLocale	Используется внутри Glibc как грубый хак для запуска сломанных программ (например, некоторые приложения Motif). Прочитайте комментарии в glibc-2.39/locale/broken_cur_max.c для получения дополнительной информации
libanl	Библиотека-заглушка, не содержащая функций. Ранее это была библиотека асинхронного поиска имен, функции которой теперь находятся в libc
libc	Основная библиотека C
libc_malloc_debug	Включает проверку выделения памяти при предварительной загрузке
libdl	Библиотека-заглушка, не содержащая функций. Ранее была библиотекой интерфейса динамической компоновки, функции которой теперь находятся в libc
libg	Библиотека-заглушка без функций. Раньше была библиотекой среды выполнения для g++
libm	Математическая библиотека
libmvec	Библиотека векторных математических вычислений, подключаемая по мере необходимости при использовании libm
libmcheck	Включает проверку выделения памяти при подключении к
libmemusage	Используется memusage для сбора информации об использовании памяти программой
libnsl	Библиотека сетевых служб, которая в настоящее время устарела
libnss_*	Модули Name Service Switch, содержащие функции для разрешения имен хостов, имен пользователей, имен групп, псевдонимов, служб, протоколов и т. д. Загружаются libc в соответствии с конфигурацией в /etc/nsswitch.conf
libpcprofile	Содержит функции профилирования, используемые для отслеживания времени, потраченного процессором в конкретных строках исходного кода
libpthread	Библиотека-заглушка, не содержащая функций. Ранее содержала функции, обеспечивающие большинство интерфейсов, заданных POSIX.1c Threads Extensions (расширения реализации потоков) и интерфейсы семафоров, указанных в POSIX.1b Real-time Extension (расширения реального времени), теперь эти функции находятся в libc
libresolv	Содержит функции создания, пересылки и интерпретации пакетов, используемых на серверах доменных имен в сети интернет

librt	Содержит функции, реализующие большую часть интерфейсов, определяемых в POSIX.1b Real-time Extension (расширения реального времени)
libthread_db	Содержит функции, полезные для сборки отладчиков для многопоточных программ
libutil	Библиотека-заглушка, не содержащая функций. Ранее содержал код для «стандартных» функций, используемых во многих утилитах Unix. Эти функции теперь находятся в libc

8.6. Zlib-1.3.1

Пакет Zlib содержит подпрограммы сжатия и распаковки, используемые некоторыми программами.	
Приблизительное время сборки:	менее 0.1 SBU
Требуемое дисковое пространство:	6.4 MB

8.6.1. Установка пакета Zlib

Подготовьте Zlib к компиляции:

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

Скомпилируйте пакет:

```
make
```

Чтобы протестировать пакет, выполните:

```
make check
```

Удалите бесполезную статическую библиотеку:

```
rm -fv /usr/lib/libz.a
```

8.6.2. Содержимое пакета Zlib

Установленные файлы:	libz.so

Краткое описание

libz	Содержит функции сжатия и распаковки, используемые некоторыми программами.

8.7. Bzip2-1.0.8

Пакет Bzip2 содержит программы для сжатия и распаковки файлов. Сжатие текстовых файлов с помощью bzip2 даёт больший процент сжатия, чем традиционный gzip.	
Приблизительное время сборки:	менее 0.1 SBU
Требуемое дисковое пространство:	7.2 MB

8.7.1. Установка пакета Bzip2

Примените патч, который установит документацию для этого пакета:

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

Следующая команда гарантирует установку символических ссылок с относительным путём:

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

Убедитесь, что справочные страницы установлены в правильном месте:

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

Подготовьте Bzip2 к компиляции:

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

Значение параметра make:

- **-f Makefile-libbz2_so**

Этот параметр позволяет выполнить сборку, с использованием другого **Makefile**, в данном случае **Makefile-libbz2_so**, который создает динамическую библиотеку **libbz2.so** и связывает с ней **Bzip2**.

Скомпилируйте и протестируйте пакет:

```
make
```

Установите пакет:

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

Установите библиотеку:

```
cp -av libbz2.so.* /usr/lib
ln -sv libbz2.so.1.0.8 /usr/lib/libbz2.so
```

Установите общий бинарный файл bzip2 в каталог /usr/bin, и замените две копии bzip2 символическими ссылками:

```
cp -v bzip2-shared /usr/bin/bzip2
for i in /usr/bin/{bzcat,bunzip2}; do
  ln -sfv bzip2 $i
done
```

Удалите ненужную статическую библиотеку:

```
rm -fv /usr/lib/libbz2.a
```

8.7.2. Содержимое пакета Bzip2

Установленные программы:	bunzip2 (ссылка на bzip2), bzcat (ссылка на bzip2), bzcmp (ссылка на bzdiff), bzdiff, bzegrep (ссылка на bzgrep), bzfgrep (ссылка на bzgrep), bzgrep, bzip2, bzip2recover, bzless (ссылка на bzmore) и bzmore
Установленные библиотеки:	libbz2.so
Созданные каталоги:	/usr/share/doc/bzip2-1.0.8

Краткое описание

bunzip2	Распаковывает bzip-файлы
bzcat	Распаковывает в поток стандартного вывода
bzcmp	Запускает программу cmp для bzip файлов
bzdiff	Запускает программу diff для bzip файлов
bzegrep	Запускает программу egrep для bzip файлов
bzfgrep	Запускает программу fgrep для bzip файлов
bzgrep	Запускает программу grep для bzip файлов
bzip2	Сжимает файлы, используя алгоритм сжатия текста с блочной сортировкой Барроуза — Уилера и кодирование Хафмана; степень сжатия лучше, чем у более традиционных архиваторов, использующих алгоритмы «Lempel-Ziv», например gzip
bzip2recover	Пытается восстанавливать данные из поврежденных архивов
bzless	Запускает программу less для bzip файлов
bzmore	Запускает программу more для bzip файлов
libbz2	Библиотека, реализующая сжатие данных без потерь с использованием алгоритма Барроуза-Уилера.

8.8. Xz-5.4.6

8.6.2. Содержимое пакета Zlib

Установленные файлы:	libz.so

Краткое описание

libz	Содержит функции сжатия и распаковки, используемые некоторыми программами.