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

Содержание

- 8.1. Введение
- 8.2. Управление пакетами
- 8.5. Glibc-2.39
- 8.6. Zlib-1.3.1
- 8.12. M4-1.4.19
- 8.18. Pkgconf-2.1.1
- 8.19. Binutils-2.42
- 8.20. GMP-6.3.0
- 8.23. Attr-2.5.2
- 8.24. Acl-2.3.2
- 8.28. GCC-13.2.0
- 8.33. Bison-3.8.2
- 8.35. Bash-5.2.21
- 8.42. Perl-5.38.2
- 8.45. Autoconf-2.72
- 8.46. Automake-1.16.5
- 8.47. OpenSSL-3.2.1
- 8.51. Python-3.12.2
- 8.63. GRUB-2.12
- 8.64. Gzip-1.13
- 8.68. Make-4.4.1
- 8.69. Patch-2.7.6
- 8.71. Texinfo-7.1
- 8.72. Vim-9.1.0041
- 8.75. Systemd-255
- 8.77. Man-DB-2.12.0
- 8.79. Util-linux-2.39.3
- 8.81. Об отладочных символах
- 8.82. Удаление отладочных символов
- 8.83. Очистка

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