4. Заключительный этап подготовки

Содержание

- 4.1. Введение
- 4.2. Создание ограниченной иерархии папок в файловой системе LFS
- 4.3. Создание пользователя LFS
- 4.4. Настройка окружения
- 4.5. О SBU (Стандартная единица времени сборки)
- 4.6. О наборах тестов

4.1. Введение

В этой главе мы выполним несколько дополнительных настроек для подготовки к сборке временной системы. Мы создадим несколько каталогов в $LFS (в котором установим временные инструменты), добавим непривилегированного пользователя и настроим окружение для этого пользователя. Кроме этого, будут даны пояснения по стандартной единице времени сборки, или «SBU», которую мы используем для измерения времени необходимого для сборки пакетов LFS, и предоставим некоторую информацию о наборах тестов.

4.2. Создание ограниченной иерархии папок в файловой системе LFS

В этом разделе мы начинаем заполнять файловую систему LFS элементами, которые будут основой конечной системы Linux. Первым шагом является создание ограниченной иерархии каталогов, чтобы программы, скомпилированные в Главе 6 (а также glibc и libstdc++ в Главе 5), могли быть установлены в их конечном расположении. Это необходимо для того, чтобы эти временные программы были перезаписаны при сборке окончательных версий в Главе 8.

Создайте необходимую иерархию каталогов, выполнив следующую команду от имени root:

```
mkdir -pv $LFS/{etc,var} $LFS/usr/{bin,lib,sbin}

for i in bin lib sbin; do
  ln -sv usr/$i $LFS/$i
done

case $(uname -m) in
  x86_64) mkdir -pv $LFS/lib64 ;;
esac
```

Программы в Главе 6 будут скомпилированы с помощью кросс-компилятора (более подробная информация приведена в разделе Технические примечания по сборочным инструментам). Чтобы отделить кросс-компилятор от других программ, он будет установлен в специальный

каталог. Создайте этот каталог с помощью следующей команды:

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



Примечание
Редакторы LFS намеренно решили не использовать каталог /usr/lib64. В процессе сборки предпринимается ряд шагов, чтобы убедиться, что набор инструментов не будет его использовать. Если по какой-либо причине этот каталог появится (это может произойти, если вы допустили ошибку, следуя инструкциям, или потому что вы установили бинарный пакет, создавший его после сборки LFS), это может привести к поломке вашей системы. Вы должны быть уверены, что этого каталога не существует.

4.3. Создание пользователя LFS

При входе в систему под учетной записью root допущение одной ошибки может привести к повреждению или разрушению системы. Поэтому пакеты в следующих двух главах собираются из-под учетной записи непривилегированного пользователя. Вы можете использовать свое собственное имя пользователя, но чтобы упростить настройку рабочей среды, создайте нового пользователя с именем *lfs*, который является членом одноименной группы и выполняйте команды из-под этой учетной записи в процессе установки. От имени пользователя root выполните следующие команды, чтобы добавить нового пользователя:

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

Значение параметров командной строки:

-s /bin/bash
Устанавливает *bash* оболочкой по умолчанию для пользователя *lfs*.

-g lfs
Эта опция добавляет пользователя *lfs* в группу *lfs*.

-m
Создает домашний каталог для пользователя *lfs*.

-k /dev/null
Этот параметр предотвращает возможное копирование файлов из предустановленного набора каталогов (по умолчанию /etc/skel) путем изменения местоположения ввода на специальное null-устройство.

lfs
Это имя нового пользователя.

Если вы хотите войти в систему как *lfs* или переключиться на *lfs* из учетной записи

непривилегированного пользователя (в отличие от переключения на пользователя *lfs* при входе в систему как root, для которого не требуется пароль пользователя *lfs*), вам необходимо установить пароль для *lfs*. Выполните следующую команду от имени пользователя *root*, чтобы установить пароль:

```
passwd lfs
```

Предоставьте пользователю lfs полный доступ ко всем каталогам в папке $LFS, назначив lfs владельцем:

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



Примечание
В некоторых хост-системах следующая команда не выполняется должным образом и приостанавливает вход пользователя lfs в фоновом режиме. Если подсказка «lfs:~$» не появляется сразу, ввод команды fg устранит проблему.

Затем запустите оболочку, работающую от имени пользователя *lfs*. Это можно сделать, войдя в систему как *lfs* на виртуальной консоли или с помощью следующей команды замены/переключения пользователя:

```
su - lfs
```

Аргумент «-» передает значение команде su для запуска оболочки входа в систему, а не обычной оболочки. Разница между этими двумя типами оболочек подробно описана в bash(1) и info bash.

4.4. Настройка окружения

Настроим хорошо работающее окружение, создав два новых файла запуска для оболочки bash. Войдя в систему как пользователь lfs, введите следующую команду, чтобы создать новый .bash_profile:

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

При входе в систему под учетной записью пользователя lfs или при переключении на lfs, используя команду su с опцией «-», начальная оболочка представляет собой оболочку login, которая читает данные из /etc/profile хоста (который, вероятно, содержит некоторые настройки и переменные среды), а затем .bash_profile. Команда exec env -i.../bin/bash в файле .bash_profile заменяет запущенную оболочку новой, не содержащей переменные среды, за исключением переменных HOME, TERM, и PS1. Это гарантирует, что никакие нежелательные и потенциально

опасные переменные среды из хост-системы не попадут в среду сборки.

Новый экземпляр оболочки представляет собой non-login оболочку, которая не считывает и не выполняет содержимое файлов /etc/profile и .bash_profile, а вместо этого выполняет чтение из файла .bashrc. Создайте файл .bashrc:

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

Значение настроек в .bashrc

- **set +h**

Команда set +h отключает хэш-функцию bash. Хеширование является полезной функцией —bash использует хеш-таблицу для запоминания полного пути к исполняемому файлу, чтобы избежать многократного поиска одного и того же исполняемого файла в переменной окружения PATH. Однако новые инструменты требуется использовать сразу же после их установки. Отключение хэш-функции, заставляет оболочку искать переменную окружения PATH, всякий раз, когда программу необходимо запустить. Таким образом, оболочка найдет вновь скомпилированные инструменты в $LFS/tools/bin, как только они станут доступны, не запоминая предыдущую версию той же программы, предоставленную хост-дистрибутивом, в /usr/bin или /bin.

- **umask 022**

Установка значения пользовательской маски создания файлов (umask) 022 гарантирует, что вновь созданные файлы и каталоги доступны для записи только их владельцу, но будут доступны для чтения и выполнения остальным пользователям (при условии, что системный вызов open(2) использует режимы по умолчанию, новые файлы получат разрешения 644, а каталоги 755).

- **LFS=/mnt/lfs**

Переменная окружения LFS должна указывать на выбранную точку монтирования.

- **LC_ALL=POSIX**

Переменная LC_ALL управляет локализацией определенных программ, и формирует сообщения в соответствии с локализацией указанной страны. Установка в LC_ALL значения «POSIX» или «C» (они эквивалентны) гарантирует, что все будет работать должным образом в среде кросс-компиляции.

- **LFS_TGT=$(uname -m)-lfs-linux-gnu**

Переменная LFS_TGT устанавливает нестандартное, но совместимое описание компьютера для использования при создании кросс-компилятора и компоновщика, а также при кросс-компиляции временного набора инструментов. Дополнительная информация об этом представлена в Технические примечания по сборочным инструментам.

- **PATH=/usr/bin**

Многие современные дистрибутивы Linux объединили /bin и /usr/bin. В этом случае стандартной переменной PATH необходимо установить значение /usr/bin/ для окружения из Глава 6. Когда это не так, следующая строка добавит /bin к пути.

- **if [! -L /bin]; then PATH=/bin:$PATH; fi**

Если /bin не является символической ссылкой, то его необходимо добавить в переменную PATH.

- **PATH=$LFS/tools/bin:$PATH**

Поместив $LFS/tools/bin перед стандартным PATH, кросс-компилятор, установленный в начале Главы 5, будет обнаружен оболочкой сразу после его установки. Это, в сочетании с отключением хеширования, ограничивает риск использования компилятора хоста вместо кросс-компилятора.

- **CONFIG_SITE=$LFS/usr/share/config.site**

В Главе 5 и Главе 6, если эта переменная не задана, сценарии configure могут попытаться загрузить элементы конфигурации, специфичные для некоторых дистрибутивов, из /usr/share/config.site в хост-системе. Переопределите её, чтобы предотвратить потенциальное влияние хоста.

- **export ...**

Приведенные выше команды установили некоторые переменные, чтобы сделать их видимыми в любых вложенных оболочках, мы экспортируем их.



Важно
Некоторые коммерческие дистрибутивы добавляют недокументированный экземпляр /etc/bash.bashrc для инициализации bash. Этот файл потенциально может изменить среду пользователя lfs таким образом, что это может повлиять на сборку важных пакетов LFS. Чтобы убедиться, что пользовательская среда lfs чиста, проверьте наличие файла /etc/bash.bashrc и, если он есть, переименуйте его. От имени пользователя root, запустите:

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

Когда пользователь lfs больше не нужен (в начале Главы 7) вы можете безопасно восстановить /etc/bash.bashrc (по желанию).

Обратите внимание, что пакет LFS Bash, который мы создадим в Разделе 8.35,



«Bash-5.2.21», не настроен на загрузку или выполнение /etc/bash.bashrc, поэтому этот файл бесполезен в готовой системе LFS.

Для многих современных систем с несколькими процессорами (или ядрами) время компиляции пакета можно сократить, выполнив «параллельную сборку», либо установив переменную среды, либо сообщив программе make, сколько ядер задействовать для сборки. Например, процессор Intel Core i9-13900K имеет 8 ядер P (производительность) и 16 ядер E (энергоэффективность), ядро P может одновременно запускать два потока, поэтому каждое ядро P моделируется ядром Linux как два логических ядра. В результате получается 32 логических ядра. Очевидный способ задействовать все эти логические ядра - разрешить make создавать до 32 заданий сборки. Это можно сделать, передав параметр -j32 команде make:

```
make -j32
```

Или установите переменную окружения MAKEFLAGS, и ее содержимое будет автоматически использоваться make в качестве параметров командной строки:

```
export MAKEFLAGS=-j32
```



Важно
Никогда не передавайте параметр **-j** без номера в **make** и не устанавливайте такой параметр в **MAKEFLAGS**. Иначе **make** будет создавать бесконечные задания сборки, что вызовет проблемы со стабильностью системы.

Чтобы использовать все логические ядра, доступные для сборки пакетов в Главе 5 и Главе 6, укажите параметр MAKEFLAGS в .bashrc сейчас:

```
cat >> ~/.bashrc << "EOF"
export MAKEFLAGS=-j$(nproc)
EOF
```

Замените **$(nproc)** количеством логических ядер, которые вы хотите использовать, если вы планируете использовать не все логические ядра.

Наконец, чтобы убедиться, что среда полностью подготовлена для сборки временных инструментов, перечитайте только что созданный профиль пользователя:

```
source ~/.bash_profile
```

4.5. О SBU (Стандартная единица времени сборки)

Многие люди хотели бы знать заранее, сколько примерно времени потребуется для компиляции и установки каждого пакета. Поскольку Linux From Scratch может быть собран на различных системах, невозможно дать точную оценку времени. Сборка самого большого

пакета (gcc) займет около 5 минут на быстрых системах, но может занять несколько дней на более медленных компьютерах! Вместо фактического времени в книге используется показатель «стандартная единица времени сборки» (SBU).

Показатель SBU рассчитывается следующим образом. Первым пакетом, который нужно скомпилировать, является binutils в Главе 5. Время, необходимое для компиляции этого пакета с использованием одного ядра, будет называться стандартной единицей времени сборки или SBU. Время компиляции остальных пакетов будет рассчитано относительно этого времени.

Например, рассмотрим пакет, время компиляции которого составляет 4,5 SBU. Это означает, что если вашей системе потребовалось 10 минут для компиляции и сборки первого прохода binutils, то для сборки этого пакета потребуется примерно 45 минут. К счастью, в большинстве случаев, время сборки меньше, чем у binutils.

В целом, величина SBU не совсем точна, поскольку она зависит от многих факторов, включая версию GCC хост-системы. Она приведены здесь, чтобы дать оценку того, сколько времени может потребоваться для сборки пакета, но в некоторых случаях цифры могут отличаться на десятки минут.



Примечание
Когда используется несколько ядер, единицы измерения SBU будут различаться еще больше, чем обычно. В некоторых случаях make просто завершится ошибкой. Анализ выходных данных процесса сборки также будет более сложным, поскольку строки разных потоков будут чередоваться. Если вы столкнулись с проблемой на этапе сборки, вернитесь к сборке на одном ядре, чтобы проанализировать сообщения об ошибках.

Представленные здесь значения времени основаны на замерах при использовании четырех ядер (-j4). Время, указанное в главе 8, также включает время выполнения регрессионных тестов для пакета, если не указано иное.

4.6. О наборах тестов

Большинство пакетов предоставляют набор тестов. Запуск набора тестов для только что собранного пакета — хорошая идея, потому что он может обеспечить «проверку работоспособности», указывающую, что все скомпилировано правильно. Набор тестов, который проходит свой набор проверок, обычно доказывает, что пакет работает так, как задумал разработчик. Однако это не гарантирует, что пакет полностью без ошибок.

Некоторые наборы тестов более важны, чем другие. Например, наборы тестов для основных инструментов — GCC, binutils и glibc — имеют первостепенное значение из-за их центральной роли в правильно функционирующей системе. Выполнение наборов тестов для GCC и glibc может занять очень много времени, особенно на медленном оборудовании, но их выполнение настоятельно рекомендуется.





Примечание
Запуск наборов тестов, описанных в главе 5 и главе 6, не имеет смысла, поскольку программы компилируются с помощью кросс-компилятора, они, вероятно, не могут работать на хосте сборки.

Распространенной проблемой при запуске наборов тестов для binutils и GCC является нехватка псевдотерминалов (PTY). Это может привести к большому количеству неудачных тестов. Причин может быть несколько, но наиболее вероятная причина заключается в том, что в хост-системе неправильно настроена файловая система devpts. Этот вопрос более подробно обсуждается на странице https://mirror.linuxfromscratch.ru/lfs/faq.html#no-ptys.

Иногда наборы тестов не работают, по причинам, о которых знают разработчики и которые они считают некритичными. Просмотрите журналы, расположенные по адресу https://mirror.linuxfromscratch.ru/lfs/build-logs/12.1/, чтобы проверить, ожидаются ли сбои. Этот сайт актуален для всех наборов тестов, описанных в книге.

From:
https://wwoss.ru/ - **worldwide open-source software**

Permanent link:
https://wwoss.ru/doku.php?id=software:linux_server:lfs:chapter04&rev=1719866377

Last update: **2024/07/01 23:39**

