

Часть III. Глава 5. Сборка кросс-тулчейна

- [Часть II. Глава 4. Заключительный этап подготовки](#)

Содержание

- [5.1. Введение](#)
- [5.2. Binutils-2.42 - Проход 1](#)
- [5.3. GCC-13.2.0 - Проход 1](#)
- [5.4. Заголовочные файлы Linux-6.7.4 API](#)
- [5.5. Glibc-2.39](#)
- [5.6. Libstdc++ из GCC-13.2.0](#)

5.1. Введение

В этой главе дано описание, как создать кросс-компилятор и связанные с ним инструменты. Несмотря на то, что на данном этапе кросс-компиляция имитируется, принципы его работы те же, что и для настоящего кросс-тулчейна.

Программы, скомпилированные в этой главе, будут установлены в каталог **\$LFS/tools**, чтобы они были отделены от файлов, установленных в следующих главах. Библиотеки, же, устанавливаются на свое постоянное место, поскольку они относятся к системе, которую мы хотим создать.

5.2. Binutils-2.42 - Проход 1

Пакет Binutils содержит компоновщик, ассемблер и другие инструменты для работы с объектными файлами.

Приблизительное время сборки:	1 SBU
Требуемое дисковое пространство:	663 MB



Примечание

Вернитесь назад и перечитайте примечания в разделе Общие инструкции по компиляции. Понимание информации, помеченной как важная, может впоследствии избавить вас от многих проблем.

Очень важно, чтобы Binutils был скомпилированным первым, потому что и Glibc, и GCC выполняют различные тесты на доступных компоновщике и ассемблере, чтобы определить, какие из их функций следует включить. Переходим в библиотеку /sources

```
cd $LFS/sources
```

```
lfs:~$ cd $LFS/sources
lfs:/mnt/lfs/sources$
```

Распаковываем архив и переходим в каталог с его содержимым

```
tar -pxf binutils-2.42.tar.xz
cd binutils-2.42
```

```
lfs:/mnt/lfs/sources$ tar -pxf binutils-2.42.tar.xz
cd binutils-2.42
lfs:/mnt/lfs/sources/binutils-2.42$
```

В документации пакета Binutils рекомендуется выполнять сборку в отдельном каталоге, создадим его:

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

```
lfs:~$ mkdir -v build
cd build
mkdir: created directory 'build'
lfs:~/build$
```

Примечание



Для того, чтобы значения SBU, перечисленные в остальной части книги, были вам полезны, измерьте время, необходимое для сборки этого пакета, начиная с настройки и заканчивая установкой. Чтобы добиться этого, оберните команды сборки командой `time { ../configure ... && make && make install; }`.

Теперь подготовьте Binutils к компиляции:

```
../configure --prefix=$LFS/tools \
              --with-sysroot=$LFS \
              --target=$LFS_TGT \
              --disable-nls \
              --enable-gprofng=no \
              --disable-werror \
              --enable-default-hash-style=gnu
```

«Значение параметров настройки:»

- **-prefix=\$LFS/tools**

Указывает сценарию `configure` подготовить к установке пакет Binutils в каталог `$LFS/tools`.

- **-with-sysroot=\$LFS**

Для кросс-компиляции указывает системе сборки искать в `$LFS` библиотеки целевой системы, если необходимо.

- **-target=\$LFS_TGT**

Поскольку название машины в значении переменной `LFS_TGT` может отличаться от значения, которое возвращает сценарий `config.guess`, этот аргумент укажет сценарию `configure` как настроить систему сборки пакета Binutils для создания кросс-компоновщика.

- **-disable-nls**

Этот параметр отключает интернационализацию, так как `i18n` не требуется для временных инструментов.

- **-enable-gprofng=no**

Этот параметр отключает сборку `gprofng`, который не нужен для временного инструментария.

- **-disable-werror**

Этот параметр предотвращает остановку сборки в случае появления предупреждений от компилятора хоста.

- **-enable-default-hash-style=gnu**

По умолчанию компоновщик генерирует как хеш-таблицу в стиле GNU, так и классическую хеш-таблицу ELF для общих библиотек и динамически связанных исполняемых файлов. Хеш-таблицы необходимы только для динамического компоновщика, выполняющего поиск символов. В LFS динамический компоновщик (предоставляемый пакетом `Glibc`) всегда будет использовать хеш-таблицу в стиле GNU, к которой запросы выполняются быстрее. Так что классическая хеш-таблица ELF совершенно бесполезна. Этот параметр указывает компоновщику по умолчанию генерировать только хеш-таблицу в стиле GNU, поэтому мы можем избежать траты времени на создание классической хеш-таблицы ELF при сборке пакетов или не тратить дисковое пространство для ее хранения.

подготовим Binutils к компиляции:

```
lfs:/mnt/lfs/sources/binutils-2.42/build$ ../configure --prefix=$LFS/tools \
--with-sysroot=$LFS \
--target=$LFS_TGT \
--disable-nls \
--enable-gprofng=no \
--disable-werror \
--enable-default-hash-style=gnu
checking where to find the target windmc... just compiled
checking whether to enable maintainer-specific portions of Makefiles... no
configure: creating ./config.status
config.status: creating Makefile
lfs:/mnt/lfs/sources/binutils-2.42/build$
```

Скомпилируйте пакет:

```
make
```

```
make[4]: Leaving directory '/mnt/lfs/sources/binutils-2.42/build/ld'
make[3]: Leaving directory '/mnt/lfs/sources/binutils-2.42/build/ld'
make[2]: Leaving directory '/mnt/lfs/sources/binutils-2.42/build/ld'
make[1]: Nothing to be done for 'all-target'.
make[1]: Leaving directory '/mnt/lfs/sources/binutils-2.42/build'
lfs:/mnt/lfs/sources/binutils-2.42/build$
```

Установите пакет:

```
make install
```

```
checking where to find the target windmc... just compiled
checking whether to enable maintainer-specific portions of Makefiles... no
configure: creating ./config.status
config.status: creating Makefile
lfs:/mnt/lfs/sources/binutils-2.42/build$
```

Можем воспользоваться примечанием и обернуть перечисленные команды сборки командой `time: time { ../configure ... && make && make install; }`. Это объединит команды подготовки,

КОМПИЛЯЦИЮ И УСТАНОВКУ

```
time { ../configure --prefix=$LFS/tools \  
    --with-sysroot=$LFS \  
    --target=$LFS_TGT \  
    --disable-nls \  
    --enable-gprofng=no \  
    --disable-werror \  
    --enable-default-hash-style=gnu && make && make install; }
```

```
lfs:/mnt/lfs/sources/binutils-2.42/build$ time { ../configure --prefix=$LFS/tools \  
    --with-sysroot=$LFS \  
    --target=$LFS_TGT \  
    --disable-nls \  
    --enable-gprofng=no \  
    --disable-werror \  
    --enable-default-hash-style=gnu && make && make install; }
```

Итоговый отчет с указанием примерного времени сборки

```
ay-tree.h ../../libiberty/./include/timeval-utils.h; do \  
    /usr/bin/install -c -m 644 $h ${thd}; \  
done; \  
fi  
make[3]: Entering directory '/mnt/lfs/sources/binutils-2.42/build/libiberty/testsuite'  
make[3]: Nothing to be done for 'install'.  
make[3]: Leaving directory '/mnt/lfs/sources/binutils-2.42/build/libiberty/testsuite'  
make[2]: Leaving directory '/mnt/lfs/sources/binutils-2.42/build/libiberty'  
make[1]: Nothing to be done for 'install-target'.  
make[1]: Leaving directory '/mnt/lfs/sources/binutils-2.42/build'  
  
real    2m6.422s  
user    1m41.992s  
sys     0m27.623s  
lfs:/mnt/lfs/sources/binutils-2.42/build$
```

Перейдем в каталог sources и удалим более не нужный разархивированный каталог binutils-2.42

```
cd ../../  
rm -Rf binutils-2.42
```

```
lfs:/mnt/lfs/sources/binutils-2.42/build$ cd ../../  
rm -Rf binutils-2.42  
lfs:/mnt/lfs/sources$
```

Подробная информация об этом пакете находится в [Разделе 8.19.2, «Содержимое пакета Binutils.»](#)

5.3. GCC-13.2.0 - Проход 1

Пакет GCC содержит коллекцию компиляторов GNU, которая включает компиляторы C и C++.	
Приблизительное время сборки:	3.8 SBU
Требуемое дисковое пространство:	4.1 GB



Примечание

В этой главе часто возникают недоразумения, хотя применяются те же процедуры, что и в любой другой главе, следуйте инструкции которую получили ранее ([Инструкции по сборке пакетов](#)). Сначала распакуйте пакет gcc-13.2.0 из архива, а затем перейдите в созданный каталог. Только после этого следует приступить к приведенным ниже инструкциям.

распакуем пакет gcc-13.2.0 и перейдем в распакованный каталог

```
tar -pxf gcc-13.2.0.tar.xz
cd gcc-13.2.0
```

```
lfs:/mnt/lfs/sources$ tar -pxf gcc-13.2.0.tar.xz
cd gcc-13.2.0
lfs:/mnt/lfs/sources/gcc-13.2.0$
```

Для успешной компиляции нам потребуются исходники ещё трех пакетов: GMP, MPFR и MPC. Распакуем их в каталог исходников компилятора и переименуем каталоги так, как на них ссылаются в исходниках gcc

```
tar -xf ../mpfr-4.2.1.tar.xz
mv -v mpfr-4.2.1 mpfr
tar -xf ../gmp-6.3.0.tar.xz
mv -v gmp-6.3.0 gmp
tar -xf ../mpc-1.3.1.tar.gz
mv -v mpc-1.3.1 mpc
```

```
lfs:/mnt/lfs/sources/gcc-13.2.0$ tar -xf ../mpfr-4.2.1.tar.xz
mv -v mpfr-4.2.1 mpfr
tar -xf ../gmp-6.3.0.tar.xz
mv -v gmp-6.3.0 gmp
tar -xf ../mpc-1.3.1.tar.gz
mv -v mpc-1.3.1 mpc
renamed 'mpfr-4.2.1' -> 'mpfr'
renamed 'gmp-6.3.0' -> 'gmp'
renamed 'mpc-1.3.1' -> 'mpc'
lfs:/mnt/lfs/sources/gcc-13.2.0$
```

На хостах x86_64 измените имя каталога по умолчанию для 64-битных библиотек на «lib»:

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

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

В документации к GCC рекомендуется собирать GCC в отдельном каталоге:

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

```
lfs:/mnt/lfs/sources/gcc-13.2.0$ mkdir -v build
cd      build
mkdir: created directory 'build'
lfs:/mnt/lfs/sources/gcc-13.2.0/build$
```

Подготовьте GCC к компиляции:

```
../configure \
  --target=$LFS_TGT \
  --prefix=$LFS/tools \
  --with-glibc-version=2.39 \
```

```
--with-sysroot=$LFS      \  
--with-newlib            \  
--without-headers       \  
--enable-default-pie    \  
--enable-default-ssp    \  
--disable-nls           \  
--disable-shared        \  
--disable-multilib      \  
--disable-threads       \  
--disable-libatomic     \  
--disable-libgomp       \  
--disable-libquadmath   \  
--disable-libssp        \  
--disable-libvtv        \  
--disable-libstdcxx     \  
--enable-languages=c,c++
```

«Значение параметров настройки:»

- **-with-glibc-version=2.39**

Этот параметр указывает версию Glibc, которая будет использоваться на целевой системе. Он не имеет отношения к libc хост-дистрибутива, потому что все, скомпилированное в этом разделе, будет выполняться в среде chroot, которая изолирована от libc хост-дистрибутива.

- **-with-newlib**

Поскольку работающая библиотека C еще недоступна, это гарантирует, что константа `inhibit_libc` будет определена при сборке `libgcc`. Это предотвращает компиляцию любого кода, требующего поддержки `libc`.

- **-without-headers**

При создании полного кросс-компилятора GCC требует наличия стандартных заголовков, совместимых с целевой системой. Для наших целей эти заголовки не понадобятся. Этот параметр предотвращает их поиск GCC.

- **-enable-default-pie и -enable-default-ssp**

Эти параметры позволяют GCC по умолчанию компилировать программы с некоторыми функциями усиливающими безопасность (более подробная информация о них приведена в примечание о PIE и SSP в Главе 8). На данном этапе это не является строго обязательным, поскольку компилятор будет создавать только временные исполняемые файлы. Но лучше, чтобы временные пакеты были максимально приближены к тем, что будут в готовой системе LFS.

- **-disable-shared**

Этот параметр заставляет GCC статически связывать свои внутренние библиотеки. Он необходим потому что общие библиотеки требуют Glibc, который еще не установлен в целевой системе.

- **-disable-multilib**

На x86_64, LFS не поддерживает конфигурацию multilib. Этот аргумент никак не влияет на работу с архитектурой x86.

- **-disable-threads, -disable-libatomic, -disable-libgomp, -disable-libquadmath, -disable-libssp, -disable-libvtv, -disable-libstdcxx**

Эти аргументы отключают поддержку расширений для работы с многопоточностью, libatomic, libgomp, libquadmath, libssp, libvtv и стандартной библиотеки C++ соответственно. Эти функции могут не скомпилироваться при сборке кросс-компилятора и не нужны для задач кросс-компиляции временной libc

- **-enable-languages=c,c++**

Этот параметр обеспечивает сборку только компиляторов C и C++. Это единственные языки, которые нужны сейчас.

если увидели ошибки, повторите подготовку

```
lfs:/mnt/lfs/sources/gcc-13.2.0/build$ ../configure \
--target=$LFS_TGT \
--prefix=$LFS/tools \
--with-glibc-version=2.39 \
--with-sysroot=$LFS \
--with-newlib \
--without-headers \
--enable-default-pie \
--enable-default-ssp \
--disable-nls \
--disable-shared \
--disable-multilib \
--disable-threads \
--disable-libatomic \
--disable-libgomp \
--disable-libquadmath \
--disable-libssp \
--disable-libvtv \
--disable-libstdcxx \
--enable-languages=c,c++
</code>
```

Скомпилируйте GCC, выполнив:

```
time make
```

```
checking whether to enable maintainer-specific portions of Makefiles... no
configure: creating ./config.status
config.status: creating Makefile
lfs:/mnt/lfs/sources/gcc-13.2.0/build$ time make
```

ВЫОД ОКОНЧАНИЯ КОМПИЛЯЦИИ

```
make[3]: Leaving directory '/mnt/lfs/sources/gcc-13.2.0/build/x86_64-lfs-linux-gnu/libgcc'
make[2]: Leaving directory '/mnt/lfs/sources/gcc-13.2.0/build/x86_64-lfs-linux-gnu/libgcc'
make[1]: Leaving directory '/mnt/lfs/sources/gcc-13.2.0/build'

real    23m33.755s
user    20m47.674s
sys     2m52.960s
lfs:/mnt/lfs/sources/gcc-13.2.0/build$
```

Установите пакет:

```
lfs:/mnt/lfs/sources/gcc-13.2.0/build$ make install
```

```
make install
```

```
make[2]: Leaving directory '/mnt/lfs/sources/gcc-13.2.0/build/x86_64-lfs-linux-gnu/libgcc'
make[1]: Leaving directory '/mnt/lfs/sources/gcc-13.2.0/build'
lfs:/mnt/lfs/sources/gcc-13.2.0/build$
```

Во время сборки GCC установил пару внутренних системных заголовочных файлов. Обычно один из файлов `limits.h`, включает соответствующие системные ограничения **limits.h**, в данном случае **\$LFS/usr/include/limits.h**. Однако во время сборки **GCC \$LFS/usr/include/limits.h** не существует, поэтому только что установленный внутренний заголовочный файл является частичным, автономным файлом и не включает расширенные функции системного файла. Этого достаточно для сборки Glibc, но полный внутренний заголовочный файл понадобится позже. Создайте полную версию внутреннего заголовочного файла с помощью команды, идентичной той, что система сборки GCC использует обычно:

Примечание



В приведенной ниже команде показан пример подстановки вложенных команд, используя два метода: обратные кавычки и конструкцию `$()`. Его можно было бы переписать, используя один и тот же метод для обеих замен, но сделано так, чтобы продемонстрировать, как их можно использовать одновременно. В целом метод `$()` предпочтительнее.

Выйдем из каталога `build`

```
cd ..
```

```
lfs:/mnt/lfs/sources/gcc-13.2.0/build$ cd ..  
lfs:/mnt/lfs/sources/gcc-13.2.0$
```

Создадим полную версию внутреннего заголовка с помощью команды

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

```
lfs:/mnt/lfs/sources/gcc-13.2.0$ cat gcc/limitx.h gcc/glimits.h gcc/limity.h > \  
`dirname $($LFS_TGT-gcc -print-libgcc-file-name)'/include/limits.h  
lfs:/mnt/lfs/sources/gcc-13.2.0$
```

Подробная информация об этом пакете находится в [Разделе 8.28.2. «Содержимое пакета GCC.»](#)

5.4. Заголовочные файлы Linux-6.7.4 API

Заголовочные файлы Linux API (в `linux-6.7.4.tar.xz`) предоставляют API ядра для использования Glibc.

Приблизительное время сборки:	менее 0.1 SBU
Требуемое дисковое пространство:	1.5 GB

5.4.1. Установка заголовочных файлов

Ядро Linux должно предоставлять интерфейс прикладного программирования (API) для использования системной библиотекой C (Glibc в LFS). Это делается путем установки заголовочных файлов C, которые поставляются в архиве с исходным кодом ядра Linux.

Убедитесь, что в пакете нет устаревших файлов:


```
make mrproper
```

Теперь извлеките видимые пользователю заголовочные файлы ядра из исходного кода. Рекомендуемый способ `make «headers_install»` использовать нельзя, так как для этого требуется `rsync`, который может быть недоступен. Заголовочные файлы сначала помещаются в `/usr`, а затем копируются в нужное место.

```
make headers
find usr/include -type f ! -name '*.h' -delete
cp -rv usr/include $LFS/usr
```

5.4.2. Содержимое заголовочных файлов Linux API

Установленные заголовочные файлы:	<code>/usr/include/asm/*.h</code> , <code>/usr/include/asm-generic/*.h</code> , <code>/usr/include/drm/*.h</code> , <code>/usr/include/linux/*.h</code> , <code>/usr/include/misc/*.h</code> , <code>/usr/include/mtd/*.h</code> , <code>/usr/include/rdma/*.h</code> , <code>/usr/include/scsi/*.h</code> , <code>/usr/include/sound/*.h</code> , <code>/usr/include/video/*.h</code> , and <code>/usr/include/xen/*.h</code>
Созданные каталоги:	<code>/usr/include/asm</code> , <code>/usr/include/asm-generic</code> , <code>/usr/include/drm</code> , <code>/usr/include/linux</code> , <code>/usr/include/misc</code> , <code>/usr/include/mtd</code> , <code>/usr/include/rdma</code> , <code>/usr/include/scsi</code> , <code>/usr/include/sound</code> , <code>/usr/include/video</code> , and <code>/usr/include/xen</code>

Краткое описание

<code>/usr/include/asm/*.h</code>	Заголовочные файлы Linux API ASM
<code>/usr/include/asm-generic/*.h</code>	Заголовочные файлы Linux API ASM Generic
<code>/usr/include/drm/*.h</code>	Заголовочные файлы Linux API DRM
<code>/usr/include/linux/*.h</code>	Заголовочные файлы Linux API Linux
<code>/usr/include/misc/*.h</code>	Заголовочные файлы Linux API Miscellaneous
<code>/usr/include/mtd/*.h</code>	Заголовочные файлы API MTD
<code>/usr/include/rdma/*.h</code>	Заголовочные файлы Linux API RDMA
<code>/usr/include/scsi/*.h</code>	Заголовочные файлы Linux API SCSI
<code>/usr/include/sound/*.h</code>	Заголовочные файлы Linux API Sound
<code>/usr/include/video/*.h</code>	Заголовочные файлы Linux API Video
<code>/usr/include/xen/*.h</code>	Заголовочные файлы Linux API Xen

5.5. 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:	1.5 SBU
Требуемое дисковое пространство:	846 MB

5.5.1. Установка пакета Glibc

Во-первых, создайте символическую ссылку для соответствия требованиям LSB. Кроме того, для совместимости с `x86_64` создайте символическую ссылку, необходимую для правильной работы загрузчика динамической библиотеки:

```
case $(uname -m) in
  i?86) ln -sfv ld-linux.so.2 $LFS/lib/ld-lsb.so.3
  ;;
  x86_64) ln -sfv ../lib/ld-linux-x86-64.so.2 $LFS/lib64
          ln -sfv ../lib/ld-linux-x86-64.so.2 $LFS/lib64/ld-lsb-
x86-64.so.3
  ;;
esac
```



Примечание

Приведенная выше команда верна. Команда `ln` имеет несколько вариантов синтаксиса, поэтому обязательно ознакомьтесь с `info coreutils ln` и [ln\(1\)](#), прежде чем сообщать об ошибке.

Некоторые программы, использующие Glibc, применяют несовместимый с FHS каталог `/var/db` для хранения своих данных времени выполнения. Установите следующий патч, чтобы такие программы хранили свои данные в местах, совместимых с FHS:

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

В документации к Glibc рекомендуется собирать Glibc в отдельном каталоге:

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

Убедитесь, что утилиты `ldconfig` and `sln` установлены в `/usr/sbin`:

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

Затем подготовьте Glibc к компиляции:

```
../configure \
  --prefix=/usr \
  --host=$LFS_TGT \
  --build=$(../scripts/config.guess) \
  --enable-kernel=4.19 \
  --with-headers=$LFS/usr/include \
  --disable-nscd \
  libc_cv_slibdir=/usr/lib
```

Значение параметров настройки:

- **`-host=$LFS_TGT`, `-build=$(../scripts/config.guess)`**

Комбинация этих опций указывает на то, что система сборки Glibc настраивается на кросс-

компиляцию с использованием кросс-компоновщика и кросс-компилятора в `$LFS/tools`.

- **`-enable-kernel=4.19`**

Этот параметр позволяет Glibc выполнять компиляцию библиотеки с поддержкой ядра 4.19 и более поздних версий. Поддержка более старых ядер не включена.

- **`-with-headers=$LFS/usr/include`**

Этот аргумент позволяет скомпилировать библиотеку с заголовочными файлами, недавно установленными в каталоге `$LFS/usr/include`, таким образом, пакету будет известно, какие функции есть у ядра, чтобы оптимизировать себя.

- **`libc_cv_slibdir=/usr/lib`**

Этот аргумент гарантирует, что библиотека будет установлена в `/usr/lib` вместо стандартного `/lib64` на 64-битных машинах.

- **`-disable-nscd`**

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

На этом этапе может появиться следующее предупреждение:

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

Отсутствующая или несовместимая программа `msgfmt`, как правило, безвредна. `msgfmt` является частью пакета `Gettext`, который должен предоставлять хост-дистрибутив.



Примечание

Поступали сообщения о том, что этот пакет может не компилироваться при «параллельной сборке». Если это произойдет, повторно запустите команду `make` с параметром `-j1`.

Скомпилируйте пакет:

```
make
```

Установите пакет:



Важно

Если переменная `LFS` настроена неправильно, и, несмотря на рекомендации, вы выполняете сборку от имени пользователя `root`, следующая команда установит только что собранный Glibc в вашу хост-систему, что, скорее всего, сделает её



непригодной для использования. Поэтому дважды проверьте, правильность настройки среды и что вы вошли в систему не под учетной записью root, прежде чем запускать следующую команду.

```
make DESTDIR=$LFS install
```

Значение опции make install:

- **DESTDIR=\$LFS**

Переменная make DESTDIR используется почти всеми пакетами для определения места установки пакета. Если она не задана, по умолчанию для установки используется корневой каталог (/). Здесь мы указываем, что пакет должен быть установлен в \$LFS, который станет корневым каталогом в [Разделе 7.4. «Вход в окружение Chroot»](#) .

Исправьте жестко запрограммированный путь к исполняемому загрузчику в ldd:

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

Внимание

На этом этапе необходимо остановиться и убедиться, что основные функции (компиляция и компоновка) нового кросс-тулчейна работают должным образом. Чтобы выполнить проверку работоспособности, выполните следующие команды:

```
echo 'int main(){}' | $LFS_TGT-gcc -xc -  
readelf -l a.out | grep ld-linux
```

Если все работает правильно, ошибок быть не должно и вывод последней команды будет иметь вид:



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

Обратите внимание, что для 32-разрядных машин имя интерпретатора будет /lib/ld-linux.so.2.

Если выходные данные отображаются не так, как указано выше, или их вообще нет, значит, что-то сделано неправильно. Разберитесь с проблемой и повторите шаги выше, чтобы исправить ее. Эта проблема должна быть решена, прежде чем вы продолжите.

Как только все будет хорошо, удалите тестовый файл:

```
rm -v a.out
```

**Примечание**

Сборка пакетов в следующей главе послужит дополнительной проверкой правильности сборки временного кросс-тулчейна. Если какой-либо пакет, особенно Binutils или GCC, не удастся собрать, это указывает на то, что что-то пошло не так с установленными ранее Binutils, GCC, или Glibc.

Подробная информация об этом пакете находится в [Раздел 8.5.3. «Содержимое пакета Glibc.»](#)

5.6. Libstdc++ из GCC-13.2.0

Libstdc++ — это стандартная библиотека C++. Она нужна для компиляции кода C++ (часть GCC написана на C++), когда мы собирали [GCC-Пролог 1](#), нам пришлось отложить её установку, потому что она зависит от библиотеки Glibc, которой еще не было в целевом каталоге.

Приблизительное время сборки:	0.2 SBU
Требуемое дисковое пространство:	1.1 GB

5.6.1. Установка библиотеки Libstdc++

**Примечание**

Libstdc++ является частью исходников GCC. Сначала вы должны распаковать архив GCC и перейти в каталог gcc-13.2.0.

Создайте отдельный каталог сборки для libstdc++ и перейдите в него:

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

Подготовьте libstdc++ к компиляции:

```
../libstdc++-v3/configure \
  --host=$LFS_TGT          \
  --build=$(../config.guess) \
  --prefix=/usr            \
  --disable-multilib       \
  --disable-nls            \
  --disable-libstdcxx-pch  \
  --with-gxx-include-dir=/tools/$LFS_TGT/include/c++/13.2.0
```

Значение параметров настройки:

- **-host=...**

Указывает, что должен использоваться кросс-компилятор, который мы только что собрали, вместо того, который находится в `/usr/bin`.

- **-disable-libstdcxx-pch**

Этот аргумент предотвращает установку предварительно скомпилированных include-файлов, которые на данном этапе не нужны.

- **-with-gxx-include-dir=/tools/\$LFS_TGT/include/c++/13.2.0**

Указывает каталог установки для include-файлов. Поскольку `libstdc++` является стандартной библиотекой C++ для LFS, этот каталог должен соответствовать местоположению, в котором компилятор C++ (`$LFS_TGT-g++`) будет искать стандартные включаемые файлы C++. При обычной сборке эта информация автоматически передается в `Libstdc++` при выполнении `configure` из каталога верхнего уровня. В нашем случае эта информация должна быть указана явно. Компилятор C++ добавит путь `sysroot $LFS` (указанный при сборке GCC Проход 1) к пути поиска include-файлов, поэтому фактически он будет искать в `$LFS/tools/$LFS_TGT/include/c++/13.2.0`. Комбинация переменной `DESTDIR` (в приведенной ниже команде `make install`) и этого аргумента обеспечивает установку заголовочных файлов туда.

Скомпилируйте `Libstdc++`, выполнив:

```
make
```

Установите библиотеку:

```
make DESTDIR=$LFS install
```

Удалите архивные файлы `libtool`, поскольку они потенциально опасны при кросс-компиляции:

```
rm -v $LFS/usr/lib/lib{stdc++{,exp,fs},supc++}.la
```

Подробная информация об этом пакете находится в [Разделе 8.28.2. «Содержимое пакета GCC.»](#)

From:
<http://vladpolskiy.ru/> - **book51.ru**

Permanent link:
http://vladpolskiy.ru/doku.php?id=software:linux_server:ifs-example:chapter05&rev=1720978761

Last update: **2024/07/14 20:39**

