

Linux From Scratch

Версия 5.0

Джерард Бикманс

[Copyright](#) © 1999-2003 Gerard Beekmans

Перевод: Виталий Катраев aka [Zawullon](#)

При переводе был также использован текст перевода Linux From Scratch v4.0 Дениса Каледина, Ника Фролова и Алекса Казанкова.

Эта книга описывает процесс создания Linux системы с нуля, используя лишь исходные коды необходимого программного обеспечения.

Посвящение

Эта книга посвящается моей любящей и заботливой жене *Беверли Бикманс*.

Пролог

Предисловие

Я перепробовал несколько различных дистрибутивов Linux, но я никогда не был полностью доволен ни одним из них. Мне не нравилось расположение загрузочных скриптов. Мне не нравилось, как некоторые программы были сконфигурированы по умолчанию. Это мне порядком надоело. В конце концов я понял, что только построив собственную систему из исходников, я буду доволен. Я решил отказаться от использования заранее скомпилированных пакетов и от загрузочного диска, который мог бы установить основу системы. Я буду использовать свою нынешнюю систему Linux для построения себе новой. Эта сумасбродная идея казалась очень сложной или вообще невыполнимой. Но после решения ряда проблем, таких как зависимости или ошибки на этапе компиляции, моя новая самосборная система оказалась полностью работоспособной. Я назвал свою систему Linux From Scratch или, для краткости, LFS. Я надеюсь, что вы не зря проведете время, работая над своей собственной LFS!

--

Джерард Бикманс

gerard@linuxfromscratch.org

Аудитория

Кому эта книга будет интересна

Существует ряд причин, по которым эта книга может представлять интерес. Главная причина - установка Linux системы из исходников. Многие люди спросят: "Зачем мучиться, вручную собирая Linux систему "с нуля", когда можно скачать и установить готовую?". Это хороший вопрос и мы постараемся ответить на него в этой части книги.

Важная причина существования LFS - помочь людям узнать работу системы Linux изнутри. Построение LFS-системы помогает показать, что заставляет Linux работать, как работают ее компоненты и как они друг от друга зависят. И наконец, вероятно важнее всего, оно учит как настроить Linux под себя, в соответствии со своими вкусами и потребностями.

Основное преимущество LFS состоит в том, что вы получаете больше контроля над системой, не полагаясь на чью-либо разработку. С LFS вы определяете структуру системы и диктуете ей свои условия, в частности структуру каталогов и загрузочные скрипты. Вы также указываете как, куда и зачем устанавливаются программы.

Другим преимуществом LFS является возможность создать наиболее компактную систему. При установке обычного дистрибутива, вместе с ним, как правило, устанавливается большое количество программ, которые вы никогда не будете использовать. Они просто будут занимать место на диске (или, возможно, время центрального процессора). В то же время не составит труда установить систему LFS на 100 Mb диска. Вам кажется, это много? Некоторые пользователи смогли создать весьма компактную LFS систему. Они собрали систему, позволяющую запускать веб-сервер Apache, которая занимала всего 8 Mb на диске.

Дальнейшее урезание ее позволит сократить используемое место до 5 Mb. Попробуйте сделать это, используя обычный дистрибутив.

Мы можем сравнить дистрибутив Linux с гамбургером из закусочной - вы не знаете точно, что едите. LFS же дает вам не гамбургер, а рецепт его приготовления. Внимательно изучив его, можно убрать ненужные ингредиенты и добавить свои по вкусу. Как только вы довольны рецептом, можете смело приступать к приготовлению гамбургера. Вы можете также выбрать способ приготовления: поджарить, сварить, запечь или съесть сырым.

Также LFS можно сравнить с домом. Мы даем вам только его план, а строить его вам. Разумеется, можно изменять план по ходу дела.

Немаловажным преимуществом построения своей Linux-системы является также безопасность. При компиляции из исходников вы получаете возможность проверять все компоненты системы и, в случае надобности, ставить патчи. Вам больше не придется ждать, пока кто-то выпустит бинарный пакет, устраниющий дырку в безопасности. Нет гарантии, что новый пакет адекватно собран и устраняет проблему, если вы самостоятельно не изучили и не собрали патч.

Есть еще много причин, по которым лучше собрать LFS, настолько много, что их все здесь просто не привести. Эта часть только верхушка айсберга. При установке LFS вы поймете, что все сила - в информации и знаниях.

Кому эта книга будет неинтересна

Здесь перечислены некоторые из причин, по которым вам, возможно, не следует читать эту книгу. Если у вас нет желания собирать Linux с нуля, то, наверное, эта книга не для вас. Нашей целью является построение законченной и готовой к системе основного уровня. Если вы просто хотите узнать, что происходит при загрузке системы, то мы рекомендуем почтить "From Power Up To Bash Prompt" HOWTO. Это руководство описывает процесс построения голой системы, подобной нашей, но ставит перед собой цель создание системы, способной загрузиться до командной оболочки BASH.

При выборе объекта для чтения, ясно определите свою цель. Если вы хотите построить свою Linux систему, изучая ее по ходу дела, то эта книга для вас. Если же ваша цель сугубо познавательная, и вы не планируете создать готовую систему, тогда вам лучше выбрать "From Power Up To Bash Prompt" HOWTO.

"From Power Up To Bash Prompt" HOWTO находится по адресу <http://axiom.anu.edu.au/~okeefe/p2b/> или на сайте The Linux Documentation Project - <http://www.tldp.org/HOWTO/From-PowerUp-To-Bash-Prompt-HOWTO.html>.

Предпосылки

Эта книга подразумевает, что у вас есть хорошие познания в области использования и установки программ для Linux. Перед началом построения вашей LFS системы, вам следует прочитать следующие HOWTO:

- Software-Building-HOWTO

Это исчерпывающее руководство по установке UNIX-ориентированных пакетов программ. Это HOWTO доступно по адресу <http://www.tldp.org/HOWTO/Software-Building-HOWTO.html>.

- The Linux Users' Guide

Это руководство описывает использование различных Linux-программ и доступно на <http://espc22.murdoch.edu.au/~stewart/guide/guide.html>.

- The Essential Pre-Reading Hint

Это Советы по LFS, написанные специально для новичков в мире Linux. В основном содержит ссылки на внешние источники информации на различные темы. Любой человек, собравшийся установить LFS, должен понимать многие темы из перечисленных в этом документе. Этот документ доступен по адресу http://www.linuxfromscratch.org/hints/downloads/files/essential_prereading.txt

Организация

Эта книга состоит из следующих двух частей:

Часть I - Вступление

Часть I содержит некоторые важные сведения о процессе инсталляции и некоторую дополнительную информацию о книге (версию, список изменений, благодарности, списки рассылки и т. п.).

Часть II - Подготовка к сборке

Часть II описывает приготовления к процессу сборки: создание раздела, скачивание пакетов и компиляцию временных средств.

Часть III - Сборка системы LFS

Часть III руководит сборкой системы LFS: компиляцией и установкой всех пакетов по порядку, созданием загрузочных скриптов и установкой ядра. В результате будет создана простая Linux-система в которую вы сможете установить дополнительные программы тем способом и такие, как захотите.

Часть IV - Приложения

Часть IV состоит из двух приложений. Первое является алфавитным списком всех устанавливаемых по ходу книги пакетов - для большинства из них с описанием и официальным ресурсом для скачивания, их содержанием и зависимостями установки. Второе приложение является списком всех устанавливаемых программ и библиотек в алфавитном порядке для того, чтобы вы смогли быстро найти пакет, содержащий программу или библиотеку.

(Большая часть первого приложения содержится в Частях II и III книги. Это просто расширение книги, но мы надеемся, что это облегчит чтение книги. Вам не обязательно обращаться к первому приложению в процессе установки. Беготня вперед и назад по страницам будет пустым занятием, особенно если вы читаете печатную версию этой книги)

I. Часть I - Вступление

Глава 1. Вступление

Как мы будем работать

Мы собираемся собрать вашу LFS-систему, используя ранее установленный дистрибутив Linux (такой как Debian, Mandrake, Red Hat или SuSE). Эта существующая Linux система (основа) будет использована как отправная точка, потому что вам будут необходимы некоторые программы, такие как компилятор, компоновщик (линкер) и командный интерпритатор (шелл) для сборки системы. Обычно все перечисленные средства доступны, если при установке дистрибутива вы отметили пункт "разработка" ("development").

В [Главе 3](#) вы сначала создадите новый раздел Linux native и файловую систему, место для компиляции и установки LFS-системы. Затем в [Главе 4](#) вы скачаете все необходимые пакеты и патчи для LFS-системы и сохраните их в новой файловой системе.

[Глава 5](#) описывает установку ряда пакетов, которые составят среду разработки (или инструментальные средства) с помощью которых мы соберем систему в [Главе 6](#). Некоторые из этих пакетов являются рекурсивно зависимыми - к примеру, компилятору для компиляции нужен компилятор.

Первым делом в [Главе 5](#) мы соберем инструментальные средства: Binutils и GCC. Программы из этих пакетов будут скомпонованы статически для того, чтобы обеспечить независимость при их использовании от основной системы. После этого мы соберем Glibc, библиотеку C. Glibc будет скомпилирована с использованием средств, собранных ранее. Затем мы соберем средства снова. На этот раз наши средства будут динамически скомпонованы с использованием только что собранной Glibc. Все остальные пакеты из [Главы 5](#) будут собраны с использованием новых средств и динамически скомпонованы с использованием новой независимой от основной системы Glibc. Когда мы все сделаем, дальнейший процесс установки LFS не будет более зависеть от основного дистрибутива и запущенного ядра.

Вы можете подумать: "это большая работа, почему нам надо обязательно не зависеть от основного дистрибутива?". Техническое описание ответа на этот вопрос содержится в самом начале [Главы 5](#), включая описание разницы между статически и динамически скомпонованными программами.

В [Главе 6](#) ваша система LFS будет собрана. С помощью программы chroot (change root) мы войдем в виртуальную среду и запустим интерпритатор shell с использованием в качестве корневой директории раздел LFS. Это намного проще перезагрузки и указаний для ядра использовать раздел LFS в качестве корневого. Помимо этого, чтобы перезагрузится, а не использовать chroot, надо создать полностью способную к загрузке систему, что мы еще не сделали к этому времени. Но главной причиной при использовании chroot, вы можете использовать основную систему во время сборки LFS. Пока идет компиляция, вы можете просто переключится на другую VC (Virtual Console) или на X-ы (графический сервер) и продолжить нормальное использование компьютера.

В заключение установки мы установим загрузочные скрипты в [Главе 7](#), ядро и загрузчик мы установим в [Главе 8](#), а [Глава 9](#) содержит некоторые моменты, которые могут вам помочь по окончании прочтения книги. В конце концов вы будете готовы перезагрузить компьютер в вашу новую LFS-систему.

Это общее описание процесса. Более детальная информация содержится в соответствующих главах и описаниях пакетов. Если что-то не совсем ясно сейчас, не волнуйтесь, со временем вы все поймете.

Пожалуйста, прочтите [Главу 2](#) внимательно, она содержит важную информацию, которую вы должны принять к сведению перед работой с [Главой 5](#) и последующими главами.

Соглашения, используемые в книге

Для облегчения понимания книги, ознакомьтесь с некоторыми соглашениями, которые будут в ней использованы:

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

Текст такого формата предназначен для набора в командной строке в точности как на экране, помимо случаев., когда даны другие указания. Также он используется в пояснительных разделах для пояснения команды, о которой идет речь.

```
install-info: unknown option `--dir-file=/mnt/lfs/usr/info/dir'
```

Текст такого формата (моноширный) показывает вывод на экран, как результат выполнения какой-либо команды. Также он используется для указания имени файла, например /etc/ld.so.conf.

Выделение

Текст такого формата имеет несколько назначений в книге. В основном, он используется для подчеркивания наиболее важных моментов и для примеров.

<http://www.linuxfromscratch.org/>

Текст такого формата используется для ссылок как внутри книги, так и на внешние ресурсы - руководства, сайты, места для скачивания.

```
cat > $LFS/etc/group << "EOF"
root:x:0:
bin:x:1:
.....
EOF
```

Такой раздел используется, в основном для создания файлов конфигурации. Первая команда говорит системе создать файл \$LFS/etc/group, наполняя его строчками до EOF. В командной строке эта секция печатается в точности как на экране.

Версия книги

Это версия 5.0 книги Linux From Scratch, датированная 5-м ноября 2003 г. Если этой книге больше двух месяцев, то более новая и улучшенная версия, возможно, уже есть на одном из зеркал из списка на <http://www.linuxfromscratch.org/>.

Ресурсы ЧАВО

Если в процессе построения вашей системы LFS вы обнаружили ошибку, у вас возникли вопросы, или вы считаете, что в книгу вкрадалась опечатка, то просмотрите в первую очередь наши ЧАВО (ЧАсто задаваемые ВОпросы) на <http://www.linuxfromscratch.org/faq/>.

IRC

Множество членов сообщества LFS используют IRC сервер сообщества. Перед тем как использовать этот вид поддержки, проверьте LFS ЧАВО и архивы списков рассылки. Возможно, на ваш вопрос уже есть ответ. Вы сможете найти наш IRC сервер на *irc.linuxfromscratch.org*, порт 6667. Канал поддержки называется #LFS-support.

Списки рассылки

На *linuxfromscratch.org* есть несколько списков рассылки, используемых разработчиками проекта LFS. Эти списки включают, помимо прочего, список разработчиков и список поддержки.

За информацией о доступных списках рассылки, подписки на них, просмотра их архивов и т.п. загляните на <http://www.linuxfromscratch.org/mail.html>.

Сервер новостей

Все списки рассылки с *linuxfromscratch.org* также доступны в виде NNTP сервера. Все сообщения отправленные в списки рассылки копируются в соответствующую группу новостей.

Сервер новостей доступен по адресу *news.linuxfromscratch.org*.

Зеркала сайта

Проект LFS имеет ряд зеркал в сети. Список зеркал можно увидеть на <http://www.linuxfromscratch.org/>.

Контактная информация

Пожалуйста, направляйте все ваши вопросы и комментарии в списки рассылки LFS (см. выше). Но если вы хотите обратиться лично к Джерарду Бикмансу, то пишите на gerard@linuxfromscratch.org.

Глава 2. Важная информация о \$LFS

Пожалуйста, внимательно прочтайте этот раздел. В этой книге будет часто использоваться переменная \$LFS. \$LFS должно будет всегда заменяться на директорию примонтированного раздела для будущей системы LFS. О том, как создать такой раздел, будет подробно написано в [Главе 3](#). К примеру, пусть раздел LFS будет монтируем на /mnt/lfs.

Когда вас попросят выполнить команду `./configure --prefix=$LFS/tools`, вам следует выполнить `./configure --prefix=/mnt/lfs/tools`.

Важно всегда использовать ее подобным образом: в командной строке или в создаваемых файлах. Но можно облегчить задачу, задав переменную окружения LFS. Таким образом \$LFS будет автоматически восприниматься системой как /mnt/lfs. Это достигается командой:

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

Теперь, когда вас просят выполнить команду `./configure --prefix=$LFS/tools`, вам надо набрать ее как есть. Ваш интерпритатор сам заменит "\$LFS" на "/mnt/lfs" когда вы нажмете Enter после ввода команды.

О SBU

Многие люди хотят заранее знать, сколько займет сборка и установка того или иного пакета. Но "Linux from Scratch" собирается на разных системах, и невозможно точно указать этот отрезок времени: самый большой пакет (Glibc) может собраться менее чем за 20 минут на быстрой системе, но может собираться и больше трех дней на медленной. В связи с этим мы решили использовать для указания времени сборки и установки пакета SBU (*Static Binutils Unit*) - время статической сборки и установки пакета Binutils.

Это используется следующим образом: первым пакетом, который нам надо будет собрать в этой книге, является статически скомпонованный Binutils в Главе 5, и время его компиляции берется за "Static Binutils Unit" или "SBU". Время компиляции всех других пакетов выражается через эту единицу.

К примеру, сборка статически скомпонованной версии GCC занимает 4.4 SBUs. Это значит, что если вашей системе понадобилось 10 минут на сборку статической версии Binutils, то сборка GCC займет 45 минут. Как вы увидите, время сборки большинства пакетов меньше чем у Binutils.

Примите к сведению, что если системный компилятор вашей основной системы базируется на GCC-2, то указанный SBU будет неверным. Это происходит по причине того, что SBU основан на самом первом из скомпилированных пакетов, который собран с помощью старого GCC, а при сборке остальной части системы будет использоваться GCC-3.3.1 который работает на 30% медленнее.

Также SBU нельзя будет использовать на SMP-based машинах. Если вы являетесь счастливым обладателем многопроцессорного компьютера, то эта единица будет неточной.

О тестировании

Многие из пакетов поддерживают тестирование. Запуск его для только что собранного пакета будет хорошей идеей, так как можно будет проверить корректность компиляции. Тестирование содержит несколько тестов для проверки функциональности пакета. Это в какой-то мере может гарантировать полное отсутствие ошибок (багов) в программе.

Некоторые тестирования более важны, чем другие. К примеру, тестирование пакетов с основными средствами -- GCC, Binutils и Glibc (библиотека C) -- которые играют решающую роль для общей функциональности системы. Но учтите, что тестирование GCC и Glibc может занять весьма продолжительное время, особенно на медленном оборудовании.

В процессе чтения книги вам встретятся команды запуска различных тестов. Вам решать, стоит ли их запускать или нет. Каждый раз будут приводиться аргументы за и против запуска этих команд и их важность для системы.

Примечание: Общей проблемой при запуске тестов для Binutils и GCC является запуск вне pseudo терминала (PTY, для краткоти). Симптомом ее является большое количество неудачных тестов. Это может происходить по многим причинам. Наиболее распространенная - ваша основная система не имеет корректно установленной файловой системы *devpts*. Мы более подробно опишем это в Главе 5.

Как просить о помощи

Если вы столкнетесь с проблемой при использовании этой книги, и ваша проблема не будет описана в ЧАВО (<http://www.linuxfromscratch.org/faq>), то множество людей в Internet Relay Chat (IRC) и в списках рассылки будут рады вам помочь. Обзор списков рассылки LFS вы можете найти в [Глава 1 - Списки рассылки](#). Для получения помощи в диагностике и устранении вашей проблемы вам надо будет дать наиболее полную информацию о ней в своем вопросе.

Что необходимо указать

Помимо краткого сообщения об ошибке, не забудьте привести следующую информацию:

- версия используемой книги (для этой - 5.0),
- основной дистрибутив его версию,
- пакет или раздел, с которыми у вас возникли проблемы,
- сообщение об ошибке или симптомы проблемы,
- отклонялись ли вы от шагов, описанных в книге.

(Имейте в виду, что если вы скажете, что отклонялись от действий, описанных в книге, то это не значит, что вам не помогут. Все таки LFS - это выбор. Просто это поможет решить вашу проблему.)

Проблемы конфигурации

Если что-то пошло не так при выполнении скрипта `configure`, то посмотрите в файле `config.log`. Этот файл содержит ошибки, которые не были выведены на экран. Включите информацию из этого файла в просьбу о помощи.

Проблемы при компиляции

Для того чтобы помочь нам найти причину проблемы, важен как вывод на экране, так и содержание некоторых файлов. Вывод на экране скрипта `./configure` и команды `make` содержит много полезной информации. Не стоит слепо включать в свое сообщение весь вывод, но и не следует включать слишком мало информации. Ниже приведен вывод на экран команды `make`:

```
gcc -DALIASPATH=\"/mnt/lfs/usr/share/locale..\\"  
-DLOCALEDIR=\"/mnt/lfs/usr/share/locale\" -DLIBDIR=\"/mnt/lfs/usr/lib\"  
-DINCLUDEDIR=\"/mnt/lfs/usr/include\" -DHAVE_CONFIG_H -I. -I.  
-g -O2 -c getopt1.c  
gcc -g -O2 -static -o make ar.o arscan.o commands.o dir.o expand.o file.o  
function.o getopt.o implicit.o job.o main.o misc.o read.o remake.o rule.o  
signature.o variable.o vpath.o default.o remote-stub.o version.o optl.o  
-lutil job.o: In function `load_too_high':  
/lfs/tmp/make-3.79.1/job.c:1565: undefined reference to `getloadavg'  
collect2: ld returned 1 exit status  
make[2]: *** [make] Error 1  
make[2]: Leaving directory '/lfs/tmp/make-3.79.1'  
make[1]: *** [all-recurse] Error 1  
make[1]: Leaving directory '/lfs/tmp/make-3.79.1'  
make: *** [all-recurse-am] Error 2
```

В этом случае большинство людей включает в сообщение строку:

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

и все что после нее. Но этого недостаточно для решения проблемы потому, что это только скажет что *что-то* не так, но не скажет что *именно* не так. В сообщение с просьбой помочь необходимо включить всю секцию вывода, т.к. она содержит выполненные команды и их сообщения об ошибках.

В Интернете есть отличная статья о том, как надо правильно задавать вопросы, написанная Эриком С. Раймондом. Ее можно найти на <http://catb.org/~esr/faqs/smart-questions.html>. Прочтите эту статью и примите к сведению при задавании вопросов, тогда ваши шансы получить полный и квалифицированный ответ значительно возрастут.

Проблемы при тестировании

Многие пакеты поддерживают тестирование, и мы, в зависимости от его важности, будем советовать вам запускать их. Иногда пакеты выдают ошибки. Если это случилось у вас, то посмотрите страничку LFS Wiki на <http://wiki.linuxfromscratch.org/>, возможно, мы уже нашли способ ее решения. Если мы уже знаем о ней, то вам не стоит беспокоиться.

II. Часть II - Подготовка к сборке

Глава 3. Подготовка нового раздела

Вступление

В этой главе мы подготовим раздел для нашей новой LFS-системы. Мы создадим раздел, установим файловую систему и примонтируем ее.

Создание нового раздела

Для сборки новой Linux системы нам будет нужно некоторое место: пустой раздел диска. Если у вас дефицит со свободным местом, то вы можете, в принципе, собрать LFS на том же разделе, где установлен ваш нынешний дистрибутив.. Это не рекомендуется для первой установки LFS, но если у вас мало свободного места на диске и вы очень смелый, то посмотрите справку на http://www.linuxfromscratch.org/hints/downloads/files/lfs_next_to_existing_systems.txt.

Для минимальной системы нам понадобится раздел около 1.2 GB. Это позволит сохранить все исходники и архивы и собрать все пакеты. Но если вы хотите установить LFS в качестве основной Linux системы, то вам, видимо, понадобятся дополнительные программы и надо будет больше места, возможно около 2 или 3 GB. Для того, чтобы вам всегда хватало памяти RAM, неплохо бы было использовать небольшой раздел диска для свопа (swap space) -- это место будет использоваться ядром для сохранения редко используемых данных из памяти для увеличения объема доступной памяти за счет этого раздела. Своп может быть одним для всех ваших систем: LFS и основной, так что вам не надо создавать второй своп-раздел, если у вас уже есть один. Программы для оперирования разделами, такие как cfdisk или fdisk, запускаются с именем диска, на котором будет производится изменение. в качестве параметра -- к примеру, /dev/hda для главного IDE диска. Создайте Linux native и своп разделы в случае надобности. Пожалуйста, изучите мануалы по cfdisk или fdisk, если вы не знаете, как использовать эти программы.

Запомните расположение вашего нового раздела -- что-то наподобие hda5. Эта книга ссылается на этот раздел как на раздел LFS. Если у вас есть своп-раздел, то запомните и его расположение. Оно нам понадобится для файла /etc/fstab.

Создание файловой системы на новом разделе

Теперь у нас есть пустой раздел и нам надо создать на нем файловую систему. Наиболее используемой системой в Linux является second extended file system (ext2), но современные высококачественные жесткие диски делают более популярными журналируемые файловые системы. Здесь мы создадим файловую системы ext2, но инструкции по установке других файловых систем можно найти на <http://www.linuxfromscratch.org/blfs/view/stable/postlfs/filesystems.html>.

Для создания файловой системы ext2 в разделе LFS необходимо выполнить:

```
mke2fs /dev/xxx
```

Замените xxx на имя вашего раздела LFS (что-то типа hda5).

Если вы создали новый своп-раздел, то вам необходимо инициализировать и его запуском:

```
mkswap /dev/yyy
```

Замените yyy на имя вашего своп-раздела.

Монтирование нового раздела

Теперь, когда мы создали файловую систему, нам надо получить доступ к ее разделу. Для этого нам необходимо примонтировать его и выбрать точку монтирования. В этой книге подразумевается, что файловая система монтирована на /mnt/lfs, но это не обязательно, поступите как вам больше нравится. Выберите точку монтирования и свяжите ее с переменной окружения LFS, для этого запустите:

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

Теперь нам надо создать точку монтирования для файловой системы LFS запуском:

```
mkdir -p $LFS  
mount /dev/xxx $LFS
```

Замените xxx на имя вашего раздела LFS.

Если вы решили использовать несколько разделов для LFS (одну для /, другую для /usr), примонтируйте их следующим образом:

```
mkdir -p $LFS  
mount /dev/xxx $LFS  
mkdir $LFS/usr  
mount /dev/yyy $LFS/usr
```

Само собой, вам надо заменить xxx и yyy на соответствующие имена разделов.

Вам также надо убедится, что новый раздел не примонтировался с ограничениями доступа (такие опции как nosuid, nodev или noatime). Вы можете запустить команду mount без параметров для того, чтобы увидеть с какими опциями примонтировался наш раздел. Если вы увидите nosuid, nodev или noatime, вам надо будет перемонтировать раздел снова.

Теперь у нас есть место для работы и мы готовы к закачке пакетов.

Глава 4. Материалы: Пакеты и патчи Вступление

Дальше будет приведен список пакетов которые вам надо будет скачать для сборки простейшей Linux системы. Указанные версии являются версиями, о которых *точно известно*, что они работают, и эта книга базируется на них. Если вы не являетесь опытным установщиком LFS, мы строго рекомендуем не

испытывать новые версии, так как команды сборки для них могут отличаться. Также не рекомендуется использовать последние версии пакетов из-за того, что еще неизвестно какие проблемы может вызвать их установка и как они будут работать.

Все ссылки, по возможности, ссылаются на страницу проекта на <http://www.freshmeat.net/>. Страницы с Freshmeat дают простой доступ к официальным сайтам для скачивания (сайтам проектов), спискам рассылки, ЧАБО, спискам изменений и др.

Мы не можем гарантировать, что эти адреса для скачивания будут все время доступны. В случае отсутствия доступа по ссылке в этой книге, попробуйте поискать в google. Если это не решит проблемы, вы можете проконсультироваться на странице errata этой книги по адресу <http://linuxfromscratch.org/lfs/print/> или, что лучше, попробовать скачать пакеты по ссылкам из списка на <http://linuxfromscratch.org/lfs/packages.html>. Вам надо сохранить скачанные пакеты и патчи так, чтобы они были доступны из точки сборки. Вам также будет нужна директория для распаковки исходников и их сборки. В книге мы используем директорию \$LFS/sources как место для записи тарболов и патчей и как рабочую директорию. Это позволяет сделать их доступными из раздела LFS на всем протяжении процесса сборки системы.

Таким образом вам надо выполнить (с правами *root*) следующую команду перед скачиванием:

```
mkdir $LFS/sources
```

И сделать эту директорию доступной для записи вашим нормальным пользователем (если вы не собираетесь скачивать из под *root*'а) командой:

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

Все пакеты

Скачайте или получите другим способом следующие пакеты:

Autoconf (2.57) - 792 KB:

<http://freshmeat.net/projects/autoconf/>

Automake (1.7.6) - 545 KB:

<http://freshmeat.net/projects/automake/>

Bash (2.05b) - 1,910 KB:

<http://freshmeat.net/projects/gnubash/>

Binutils (2.14) - 10,666 KB:

<http://freshmeat.net/projects/binutils/>

Bison (1.875) - 796 KB:

<http://freshmeat.net/projects/bison/>

Bzip2 (1.0.2) - 650 KB:

<http://freshmeat.net/projects/bzip2/>

Coreutils (5.0) - 3,860 KB:

<http://freshmeat.net/projects/coreutils/>

DejaGnu (1.4.3) - 1,775 KB:

<http://freshmeat.net/projects/dejagnu/>

Diffutils (2.8.1) - 762 KB:

<http://freshmeat.net/projects/diffutils/>

E2fsprogs (1.34) - 3,003 KB:

<http://freshmeat.net/projects/e2fsprogs/>

Ed (0.2) - 182 KB:

<http://freshmeat.net/projects/ed/>

Expect (5.39.0) - 508 KB:

<http://freshmeat.net/projects/expect/>

File (4.04) - 338 KB: (*) See Note Below

<http://freshmeat.net/projects/file/>

Findutils (4.1.20) - 760 KB:
<http://freshmeat.net/projects/findutils/>

Flex (2.5.4a) - 372 KB:
<ftp://ftp.gnu.org/gnu/non-gnu/flex/>

Gawk (3.1.3) - 1,596 KB:
<http://freshmeat.net/projects/gnuawk/>

GCC (2.95.3) - 9,618 KB:
<http://freshmeat.net/projects/gcc/>

GCC-core (3.3.1) - 10,969 KB:
<http://freshmeat.net/projects/gcc/>

GCC-g++ (3.3.1) - 2,017 KB:
<http://freshmeat.net/projects/gcc/>

GCC-testsuite (3.3.1) - 1,033 KB:
<http://freshmeat.net/projects/gcc/>

Gettext (0.12.1) - 5,593 KB:
<http://freshmeat.net/projects/gettext/>

Glibc (2.3.2) - 13,064 KB:
<http://freshmeat.net/projects/glibc/>

Glibc-linuxthreads (2.3.2) - 211 KB:
<http://freshmeat.net/projects/glibc/>

Grep (2.5.1) - 545 KB:
<http://freshmeat.net/projects/grep/>

Groff (1.19) - 2,360 KB:
<http://freshmeat.net/projects/groff/>

Grub (0.93) - 870 KB:
<ftp://alpha.gnu.org/pub/gnu/grub/>

Gzip (1.3.5) - 324 KB:
<ftp://alpha.gnu.org/gnu/gzip/>

Inetutils (1.4.2) - 1,019 KB:
<http://freshmeat.net/projects/inetutils/>

Kbd (1.08) - 801 KB:
<http://freshmeat.net/projects/kbd/>

Less (381) - 259 KB:
<http://freshmeat.net/projects/less/>

LFS-Bootscripts (1.12) - 25 KB:
<http://downloads.linuxfromscratch.org/lfs-bootscripts-1.12.tar.bz2>

Lfs-Utils (0.3) - 221 KB:
<http://www.linuxfromscratch.org/~winkie/downloads/lfs-utils/>

Libtool (1.5) - 2,751 KB:
<http://freshmeat.net/projects/libtool/>

Linux (2.4.22) - 28,837 KB:

<http://freshmeat.net/projects/linux/>

M4 (1.4) - 310 KB:
<http://freshmeat.net/projects/gnum4/>

Make (3.80) - 899 KB:
<http://freshmeat.net/projects/gnumake>

MAKEDEV (1.7) - 8 KB:
<http://downloads.linuxfromscratch.org/MAKEDEV-1.7.bz2>

Man (1.5m2) - 196 KB:
<http://freshmeat.net/projects/man/>

Man-pages (1.60) - 627 KB:
<http://freshmeat.net/projects/man-pages/>

Modutils (2.4.25) - 215 KB:
<http://freshmeat.net/projects/modutils/>

Ncurses (5.3) - 2,019 KB:
<http://freshmeat.net/projects/ncurses/>

Net-tools (1.60) - 194 KB:
<http://freshmeat.net/projects/net-tools/>

Patch (2.5.4) - 182 KB:
<http://freshmeat.net/projects/patch/>

Perl (5.8.0) - 10,765 KB:
<http://freshmeat.net/projects/perl/>

Procinfo (18) - 24 KB:
<http://freshmeat.net/projects/procinfo/>

Procps (3.1.11) - 242 KB:
<http://freshmeat.net/projects/procps/>

Psmisc (21.3) - 259 KB:
<http://freshmeat.net/projects/psmisc/>

Sed (4.0.7) - 678 KB:
<http://freshmeat.net/projects/sed/>

Shadow (4.0.3) - 760 KB:
<http://freshmeat.net/projects/shadow/>

Sysklogd (1.4.1) - 80 KB:
<http://freshmeat.net/projects/sysklogd/>

Sysvinit (2.85) - 91 KB:
<http://freshmeat.net/projects/sysvinit/>

Tar (1.13.25) - 1,281 KB:
<ftp://alpha.gnu.org/gnu/tar/>

Tcl (8.4.4) - 3,292 KB:
<http://freshmeat.net/projects/tcltk/>

Texinfo (4.6) - 1,317 KB:
<http://freshmeat.net/projects/texinfo/>

Util-linux (2.12) - 1,814 KB:

<http://freshmeat.net/projects/util-linux/>

Vim (6.2) - 3,193 KB:

<http://freshmeat.net/projects/vim/>

Zlib (1.1.4) - 144 KB:

<http://freshmeat.net/projects/zlib/>

Общий размер пакетов: 134 MB

Примечание: Файл (4.04) может быть недоступен когда вы это читаете. На главные местах для скачивания удаляют старые версии пакетов. Пожалуйста, посмотрите [Приложение А](#) для получения информации об альтернативных адресах.

Необходимые патчи

Помимо всех пакетов вам будут нужны некоторые патчи. Они корректируют некоторые ошибки в пакетах, которые были обнаружены, или улучшают функциональность. Вам будут нужны:

Bash Patch - 7 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/bash-2.05b-2.patch>

Bison Attribute Patch - 2 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/bison-1.875-attribute.patch>

Coreutils Hostname Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils-5.0-hostname-2.patch>

Coreutils Uname Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils-5.0-uname.patch>

Ed Mkstemp Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/ed-0.2-mkstemp.patch>

Expect Spawn Patch - 6 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/expect-5.39.0-spawn.patch>

Gawk Libexecdir Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/gawk-3.1.3-libexecdir.patch>

GCC No-Fixincludes Patch - 1 KB:

http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-3.3.1-no_fixincludes-2.patch

GCC Specs Patch - 10 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-3.3.1-specs-2.patch>

GCC Suppress-Liberty Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-3.3.1-suppress-liberty.patch>

GCC-2 Patch - 16 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-2.95.3-2.patch>

GCC-2 No-Fixincludes Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-2.95.3-no-fixinc.patch>

GCC-2 Return-Type Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-2.95.3-returntype-fix.patch>

Glibc Sscanf Patch - 2 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/glibc-2.3.2-sscanf-1.patch>

Grub Gcc33 Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/grub-0.93-gcc33-1.patch>

Kbd More-Programs Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/kbd-1.08-more-programs.patch>

Man 80-Columns Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/man-1.5m2-80cols.patch>

Man Manpath Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/man-1.5m2-manpath.patch>

Man Pager Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/man-1.5m2-pager.patch>

Ncurses Etip Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses-5.3-etip-2.patch>

Ncurses Vsscanf Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses-5.3-vsscanf.patch>

Net-tools Mii-Tool-Gcc33 Patch - 2 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/net-tools-1.60-miitool-gcc33-1.patch>

Perl Libc Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/perl-5.8.0-libc-3.patch>

Procps Locale Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/procps-3.1.11-locale-fix.patch>

Shadow Newgrp Patch - 1 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/shadow-4.0.3-newgrp-fix.patch>

Zlib Vsnprintf Patch - 10 KB:

<http://www.linuxfromscratch.org/patches/lfs/5.0/zlib-1.1.4-vsnprintf.patch>

Кроме необходимых патчей есть еще дополнительные, которые были созданы сообществом LFS. Большая часть из них устраняет мелкие проблемы или добавляет функциональности не доступной по умолчанию.

При желании вы можете их скачать спо ссылкам в базе данных патчей LFS на

<http://www.linuxfromscratch.org/patches/> и выбрать любые дополнительные патчи для использования.

Глава 5. Построение временной системы

Вступление

В этой главе мы скомпилируем и установим минимальную систему Linux. Эта система будет использоваться для построения конечной LFS системы в следующей главе.

Сборка этой минимальной системы будет проходить в два этапа: сначала мы соберем независящие от основной системы средства (компилятор, ассемблер, компоновщик и библиотеки), а потом используем их для сборки других средств.

Файлы, компилируемые в этой главе, устанавливаются в директорию \$LFS/tools для отделения их от файлов, устанавливаемых в следующей главе. Эти пакеты просто временные, мы не будем засорять ими нашу конечную LFS систему.

Ключевым для понимания работы Linux системы является знание того, какие пакеты используются и для чего они нужны системе. Из этих соображений мы даем краткие содержание и описание пакетов перед инструкциями по их установке. Для короткого описания программ, входящих в пакет, обратитесь к [Приложению А](#).

Инструкции по сборке подразумевают, что вы используете интерпритатор командной строки bash. Также считается, что вы уже распаковали исходники перешли с помощью команды cd в директорию с ними перед использованием команд.

Некоторые пакеты необходимо пропатчить перед компиляцией, но только когда это необходимо для решения тех или иных проблем. Часто патчи нужно применять в обеих главах (этой и следующей), но

некоторые необходимы только в одной из глав. Поэтому не беспокойтесь, если не найдете инструкций по нанесению некоторых скачанных патчей в этой главе.

В процессе установки многих пакетов вы увидите сообщения о предупреждениях (warning) на экране. Это нормально и вы можете не обращать на них внимания. Все что они говорят - внимание, есть неточность, но не ошибка, в коде C или C++. Это из-за того, что меняются стандарты на язык C, а некоторые пакеты написаны в соответствии со старыми стандартами, а это не представляет проблемы для компилятора.

Если не сказано обратное, то вы можете спокойно удалить директории с исходниками и файлами сборки пакетов - в целях экономии дискового пространства.

Перед тем как продолжить, убедитесь, что переменная окружения LFS задана корректно выполнением команды:

```
echo $LFS
```

Вывод должен указывать на точку монтирования раздела LFS, мы используем /mnt/lfs в качестве примера.

Технические моменты

Этот раздел предназначен для объяснения некоторых логических и технических моментов рационального метода сборки. Ничего страшного, если вы не поймете все, что написано здесь. Большая часть из них имеет значение при выполнении сборки конкретных пакетов. Вы можете вернуться сюда в любое время.

Основной целью [Главы 5](#) является подготовка окружения для входа через chroot для создания полноценной системы в [Главе 6](#). По ходу дела мы соберем, используя основную систему, самодостаточные средства. Это будет сделано для обеспечения минимального риска и максимальной независимости одновременно.

Другими словами, мы соберем инструменты для сборки системы.

Важно: Перед дальнейшей работой вы должны знать название вашей платформы, которое также называется *target triplet*. В некоторых случаях target triplet может быть, к примеру: *i686-pc-linux-gnu*. Простейшим способом определения вашего target triplet является запуск скрипта config.guess, который содержится во многих пакетах. Распакуйте архив с исходниками Binutils, запустите скрипт ./config.guess и запомните вывод.

Вам также необходимо знать имя *динамического компоновщика* для вашей платформы, его также называют *динамическим загрузчиком*, не спутайте его со стандартным компоновщиком *ld* из Binutils. Динамический компоновщик является частью Glibc и служит для поиска и загрузки библиотек, в которых нуждается программа, подготовки программы к запуску и ее запуска. Как правило, динамический компоновщик называется *ld-linux.so.2*. На некоторых не очень распространенных plataформах он называется *ld.so.1*, а на некоторых 64-битных plataформах - по другому. Вы можете определить имя динамического компоновщика для вашей платформы, заглянув в директорию */lib* вашей основной системы. Безошибочным способом проверки случайной библиотеки на вашей основной системе является запуск: 'readelf -l <name of binary> | grep interpreter' и просмотр вывода.

Некоторые технические моменты которые позволяют методам сборки из [Главы 5](#) работать:

- Принципиальное сходство перекрестной компиляции, посредством чего установленные средства работают с одним префиксом и таким образом используют маленько "волшебство" GNU.
- Осторожная манипуляция путями поиска библиотек стандартным компоновщиком для того, чтобы убедиться, что программы были скомпонованы с использованием выбранных нами библиотек.
- Осторожная манипуляция *specs*- файлом gcc для передачи компилятору информации об используемом динамическом компоновщике.

Binutils устанавливаются первыми потому, что GCC и Glibc выполняют тестирование ассемблера и компоновщика во время запуска ./configure для определения того, какие настройки программного обеспечения выбраны или отключены. Это очень важно при первой реализации. Неверно сконфигурированные GCC и Glibc могут испортить все сборку средств и дистрибутив будет с ошибками. Благодаря тестированию мы можем определить это своевременно.

Binutils устанавливают ассемблер и компоновщик в два места: /tools/bin и /tools/\$TARGET_TRIPLET/bin. Точнее, средства в одной из директорий являются шестками ссылками на другие. Очень важным для компоновщика является порядок поиска библиотек. Точную информацию о нем можно получить от ld указав параметр --verbose. К примеру: 'ld --verbose | grep SEARCH' покажет текущие пути поиска и их порядок. Вы можете увидеть, какие файлы скомпонованы с помощью ld при компиляции программы-пустышки и используя

переключатель `--verbose`. К примеру: '`gcc dummy.c -Wl,--verbose 2>&1 | grep succeeded'` какие файлы были открыты удачно в процессе компоновки.

Следующим пакетом мы установим GCC и в процессе работы `./configure` вы увидите, например:

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

Это важно по причинам, описанным выше. И это также показывает, что скрипт конфигурации GCC не ищет в директориях `$PATH` какие средства использовать. Тем не менее, текущий процесс `gcc`, не использует эти пути поиска. Вы можете определить, какой стандартный компоновщик `gcc` используется, запуском: '`gcc -print-prog-name=ld`'. Детальная информация получается от `gcc` добавлением параметра `-v` при компиляции. Например: '`gcc -v dummy.c`' покажет полную информацию о препроцессоре, компиляторе и ассемблере, включая пути поиска `gcc` и другую информацию.

Следующим устанавливаемым пакетом будет Glibc. Наиболее важными зависимостями сборки Glibc являются компилятор, средства для бинарных файлов и заголовки ядра. ТКомпилятор не представляет проблемы, Glibc всегда использует `gcc` из директории `$PATH`. Бинарные средства и заголовки ядра могут привести к некоторым проблемам. Поэтому мы не рискуем и используем переключатели конфигурации для правильного выбора. После запуска `./configure` вы можете проверить содержимое файла `config.make` в директории `glibc-build` для получения информации. Вы будете использовать некоторые интересные параметры, такие как `CC="gcc -B/tools/bin/"` которые определяют используемые средства, а также флаги `-nostdinc` и `-isystem` для контроля за путем поиска для компилятора. Эти параметры показывают важный аспект пакета Glibc: он не обязательно полагается на средства по умолчанию.

После установки Glibc, мы создадим некоторые установки для того, чтобы убедиться что пути для поиска содержат только директории в `/tools`. Мы установим откорректированный `ld`, в котором будет жестко указан путь поиска в `/tools/lib`. Задем мы исправим `specs`-файл `gcc` для указания на наш новый динамический компоновщик в `/tools/lib`. Этот последний шаг жизненно важен для нашего процесса. Как указано выше, путь к динамическому компоновщику будет жестко вшит в каждый исполняемый файл ELF. Вы можете убедиться в этом, запустив '`readelf -l <name of binary> | grep interpreter`'. Благодаря исправлению `specs`-файла `gcc`, мы убедимся, что все программы из оставшейся части [Главы 5](#) будут использовать новый динамический компоновщик из `/tools/lib`.

Необходимость использования нового динамического компоновщика также является причиной, по которой мы применяем Specs патч на втором этапе сборки GCC. Опускание этого патча приведет к тому, что программы GCC будут использовать системный динамический компоновщик из директории `/lib` основной системы, а это будет означать зависимость от основной системы.

В ходе второго шага сборки Binutils мы будем использовать переключатель `--with-lib-path` для контроля путя поиска библиотек для `ld`. С этого момента все наши средства будут самодостаточными. Остальные пакеты из [Главы 5](#) будут собраны с новым Glibc из `/tools`.

После входа в окружение `chroot` в [Главе 6](#), мы первым делом установим Glibc по вышеописанным причинам. Теперь Glibc будет установлен в `/usr`, и мы будем использовать средства по умолчанию для сборки остальных пакетов из [Главы 6](#) для LFS системы.

О статической компоновке

Многие программы выполняют, помимо своей основной задачи, и многие другие операции. Это включает распределение памяти, поиск директорий, чтение и запись файлов, манипуляции со строками, использование шаблонов, арифметические операции и многое другое. Вместо того, чтобы включать все это в программу, GNU система поддерживает простые функции в библиотеках. Самой главной из них является для любой Linux системы является *Glibc*.

Есть два варианта использования библиотечных функций: статически и динамически. Когда программа скомпонована статически, код используемых функций включается в программу и она становится более громоздкой. Когда же программа скомпонована динамически, в программу включаются только ссылка на динамический компоновщик, имя библиотеки и имя функции, в результате мы получаем более маленький исполняемый файл. (Есть еще третий путь - использование программного интерфейса для динамического компоновщика. Смотрите мануал по `dlopen` для более полной информации.)

Динамическая компоновка является компоновкой по умолчанию в Linux и имеет три главных преимущества перед статической компоновкой. Первое. вам достаточно иметь только одну копию исполняемого кода библиотеки на жестком диске, в отличие от нескольких копий в каждом исполняемом файле в противном случае - вы экономите место на диске. Второе, когда несколько программ используют одну и ту же библиотечную функцию одновременно, только одна копия этой функции находится в памяти - вы экономите оперативную память. Третье, если вы обнаружили ошибку в функции, достаточно перекомпилировать одну библиотеку, в отличие от необходимости перекомпилировать все программы, использующие эту функцию, в противном случае.

Если динамическая компоновка настолько лучше, то почему мы ее не используем, а статически компонуем первые два пакета в этой главе? Есть тройственная причина для этого: историческая, образовательная и

техническая. Историческая, потому что простейшая версия LFS статически компонует все программы из этой главы. Образовательная, потому что надо знать отличие между статической и динамической компоновкой. Техническая, потому что мы получаем элемент независимости от основной системы, наши программы не должны от нее зависеть, они должны иметь возможность работать самостоятельно. Таким образом, если мы хотим построить LFS систему, нам необходимо отказаться от динамической компоновки первых двух пакетов.

Создание директории \$LFS/tools

Все программы компилируемые в этой главе устанавливаются в директорию \$LFS/tools для обеспечения отсутствия проблем с программами, собираемыми в следующей главе. Программы, компилируемые здесь, являются только временными средствами, и нам надо получить возможность легко избавится от них в дальнейшем.

Помимо этого нам надо отделить имя этой директории от остальных. Для этого мы используем название "tools", при желании вы можете использовать другие названия, например "tools_for_lfs".

Создайте директорию:

```
mkdir $LFS/tools
```

Теперь нам надо создать ссылку /tools на новую директорию в основной системе:

```
ln -s $LFS/tools /
```

Эта ссылка нам нужна потому, что все средства из этой главы будут собираться с префиксом /tools, и они же нам понадобятся в следующей главе (когда мы с помощью chroot войдем в новый раздел).

Замечание: Изучите эти команды внимательно. Это может показаться странным на первый взгляд. Но команда ln имеет множество различных вариантов синтаксиса, и вы должны изучить man-страницу по ln прежде чем сообщать о том, что вы считаете, что обнаружили ошибку.

Добавление пользователя lfs

Если вы зарегистрировались в системе как *root*, малейшая ошибка может иметь фатальные последствия для вашей системы. Поэтому мы рекомендуем собирать пакеты из этой главы под непrivилегированным пользователем. Вы, конечно, можете использовать имя вашего текущего пользователя, но более простым шагом будет созданее нового пользователя *lfs* и использовать его в процессе установки. Под правами *root*, исполните следующие команды для добавления нового пользователя:

```
useradd -s /bin/bash -m lfs  
passwd lfs
```

Чтобы новый пользователь *lfs* получил полный доступ к директории \$LFS/tools изменим ее владельца:

```
chown lfs $LFS/tools
```

Если вы создали отдельную директорию для работы, смените также и ее владельца на *lfs*:

```
chown lfs $LFS/sources
```

Теперь, войдем в систему как пользователь *lfs*. Это можно сделать через виртуальную консоль, через менеджер экрана или через команду:

```
su - lfs
```

Инструкция "-" команды su запустит новый интерпретатор командной строки.

Настройка окружения

Когда вы залогинитесь как *lfs*, Введите следующие команды для настройки окружения:

```
cat > ~/.bash_profile << "EOF"  
set +h  
umask 022  
LFS=/mnt/lfs  
LC_ALL=POSIX  
PATH=/tools/bin:$PATH  
export LFS LC_ALL PATH  
unset CC CXX CPP LD_LIBRARY_PATH LD_PRELOAD  
EOF  
  
source ~/.bash_profile
```

Команда set +h отключает функцию запоминания bash. Обычно это используется так: bash использует hash-таблицу для запоминания полного пути к исполняемым файлам для сокращения времени поиска и

отсутствия необходимости запоминания путей этих файлов. Но мы собираемся использовать новые средства после установки. При отключении этой функции наши "интерактивные" команды (make, patch, sed, cp и другие) будут использовать наиболее новые из доступных версий программ в процессе сборки.

Установка маски для создания файлов пользователем в 022 позволит убедиться, что вновь созданные файлы и директории будут доступны для записи только владельцу, а для чтения и выполнения любому.

Переменная LFS указывает на точку монтирования, которую вы выбрали для раздела LFS.

Переменная LC_ALL контролирует локализацию некоторых программ, делает вывод их сообщений зависимым от страны. Если ваша система основана на Glibc старее версии 2.2.4, установка LC_ALL в что-то отличное от "POSIX" или "C" может создать проблемы при выходе в среду chroot. Установка LC_ALL в "POSIX" (или "C", что аналогично) мы страхуемся от ошибок при использовании chroot.

Мы добавляем /tools/bin к стандартной переменной PATH для того, чтобы на этапе сборки использовались средства, которые мы уже собрали.

Переменные CC, CXX, CPP, LD_LIBRARY_PATH и LD_PRELOAD несут потенциальную опасность для средств из Главы 5. Мы обнуляем их для того, чтобы было больше шансов собрать все корректно.

Теперь мы можем приступить к сборке временных средств, используемых в следующих главах.

Установка Binutils-2.14 - Шаг 1

Ожидаемое время сборки: 1.0 SBU
Ожидаемое место на диске: 194 MB

Описание Binutils

Binutils является коллекцией средств разработки программ, содержащих компоновщик, ассемблер и другие средства для работы с объектными файлами и архивами.

Устанавливаемые программы: addr2line, ar, as, c++filt, gprof, ld, nm, objcopy, objdump, ranlib, readelf, size, strings и strip

Устанавливаемые библиотеки: libiberty.a, libbfd.[a,so] и libopcodes.[a,so]

Зависимости установки Binutils

Binutils зависит от: Bash, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed, Texinfo.

Установка Binutils

Важно чтобы Binutils был первым из пакетов, которые мы установим, потому что Glibc и GCC проводят некоторые тесты на доступные компоновщик и ассемблер для определения доступных опций.

Замечание: Хотя Binutils является важным пакетом, мы не собираемся запускать тестирование на этом простом шаге. Во-первых тестирование пока неуместно, а во-вторых, программы из этого этапа будут заново установлены на втором.

Этот пакет известен своим нестабильным поведением при компиляции с измененными опциями оптимизации (включая опции -march и -mtcpu). Binutils рекомендуется компилировать с настройками по умолчанию. Следовательно, если вы задали переменные такие как CFLAGS или CXXFLAGS, изменяющие уровень оптимизации по умолчанию, рекомендуется убрать их при сборке пакета binutils. Изменяя оптимизации для binutils, вы действуете на свой страх и риск.

В документации по Binutils рекомендуется собирать Binutils вне директории с исходниками, в отдельной директории для сборки:

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

Note: Если вы хотите вычислить переменную SBU, которая используется в этой книге, вам надо засечь время, которое понадобится на сборку этого пакета. Это очень просто сделать чем-то похожим на такую команду: time { ./configure ... && ... && make install; }.

Теперь подготовим Binutils к компиляции:

```
..../binutils-2.14/configure \  
--prefix=/tools --disable-nls
```

Описание используемых опций:

- `--prefix=/tools`: Это скажет скрипту конфигурации, что программы Binutils следует устанавливать в директорию `/tools`.
- `--disable-nls`: Это отключит интернационализацию (сокращенно - i18n). Во-первых нам это пока не надо. а во-вторых `nls` может вызвать определенные проблемы при статической компиляции.

Вернемся к компиляции пакета:

```
make configure-host
make LDFLAGS="-all-static"
```

Описание параметров сборки:

- `configure-host`: Это правильно настроит все субдиректории. Статическая сборка без этого будет невозможна. Мы используем эту опцию для беспроблемной работы.
- `LDFLAGS="-all-static"`: Это скажет компоновщику, что все программы Binutils будут скомпонованы статически. Точнее, строго говоря, `"-all-static"` отсылаемое программе `libtool`, которая отсылает `"-static"` компоновщику.

И установим пакет:

```
make install
```

Теперь подготовим к компоновщик к последующему "встраиванию" в Glibc:

```
make -C ld clean
make -C ld LDFLAGS="-all-static" LIB_PATH=/tools/lib
```

Описание параметров сборки:

- `-C ld clean`: Это говорит программе сборки удалить все скомпилированные файлы, но только в субдиректории `ld`.
- `-C ld LDFLAGS="-all-static" LIB_PATH=/tools/lib`: Этот параметр пересоберет все в субдиректории `ld`. Указание переменной файла сборки `LIB_PATH` заменит установленное значение по умолчанию и установит на расположение наших временных средств. Содержимое этой переменной указывает путь по умолчанию для поиска библиотек компоновщиком. Вы увидите, как это будет использовано далее в главе.

Внимание

Не удаляйте сейчас директории для сборки и исходников Binutils. Вам они еще будут нужны в этой главе далее в их теперешнем состоянии.

Установка GCC-3.3.1 - Шаг 1

Ожидаемое время сборки:	4.4 SBU
Ожидаемое место на диске:	300 MB

Описание GCC

Пакет GCC содержит коллекцию компилятора GNU, включая компиляторы C и C++.

Устанавливаемые программы: c++, cc (link to gcc), cc1, cc1plus, collect2, cpp, g++, gcc, gccbug и gcov

Устанавливаемые библиотеки: libgcc.a, libgcc_eh.a, libgcc_s.so, libstdc++.a, libsupc++.a

Зависимости установки GCC

GCC зависит от: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed, Texinfo.

Установка GCC

Распакуйте только тарбол GCC-core, нам пока не нужен компилятор C++.

Замечание: Несмотря на то, что GCC является очень важным пакетом средств, мы не запускаем сейчас тестирование. Во-первых, тестирование сейчас не нужно, а во-вторых, программы из этого шага будут переписаны на следующем.

Этот пакет известен своим нестабильным поведением при компиляции с измененными опциями оптимизации (включая опции `-march` и `-mtcpu`). GCC рекомендуется компилировать с настройками по умолчанию. Следовательно, если вы задали переменные такие как `CFLAGS` или `CXXFLAGS`, изменяющие уровень оптимизации по умолчанию, рекомендуется убрать их при сборке пакета GCC. Изменяя оптимизации для GCC, вы действуете на свой страх и риск.

В документации по GCC рекомендуется собирать GCC вне директории с исходниками в отдельной директории для сборки:

```
mkdir ..\gcc-build  
cd ..\gcc-build
```

Подготовим GCC к компиляции:

```
..\gcc-3.3.1\configure --prefix=/tools \  
--with-local-prefix=/tools \  
--disable-nls --enable-shared \  
--enable-languages=c
```

Описание опций конфигурации:

- `--with-local-prefix=/tools`: Назначение этого переключателя исключить `/usr/local/include` из пути поиска включений gcc. Это не очень существенно, но мы собираемся минимизировать зависимость от основной системы, и мы считаем, что это нужно сделать.
- `--enable-shared`: Этот переключатель может показаться интуитивно понятным поначалу. Но его использование допускает сборку `libgcc_s.so.1` и `libgcc_eh.a`, а также делает `libgcc_eh.a` доступным для скрипта конфигурации Glibc (следующего компилируемого пакета), выдавая правильный результат. Заметьте, что бинарники gcc компонуются статически, если это задано значением `-static` для `BOOT_LDFLAGS` далее.
- `--enable-languages=c`: Эта опция позволяет быть уверенным, что будет собран только компилятор C. "Она нужна только в том случае, если вы скачали и распаковали полный тарбол.

Продолжим компиляцию пакета:

```
make BOOT_LDFLAGS="-static" bootstrap
```

Описание параметров сборки:

- `BOOT_LDFLAGS="-static"`: Это скажет GCC компоновать эти программы статически.
- `bootstrap`: Этот параметр заставляет не просто компилироваться GCC, а компилироваться несколько раз. Программы, скомпилированные сначала используются для компиляции программ еще раз, а те, в свою очередь, используются при компиляции в третий раз. Идентичность второй и третьей компиляции позволяет убедиться в корректности сборки.

И установим пакет:

```
make install
```

Под конец мы создадим ссылку `/tools/bin/cc`. Многие программы и скрипты используют cc вместо gcc для обеспечения переносимости программ на все Unix системы. Не у всех установлен именно компилятор GNU C. Запуск cc позволяет администратору выбирать, какой компилятор C устанавливать в систему, и мы создаем ссылку на него:

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

Установка заголовков Linux-2.4.22

Ожидаемое время сборки: 0.1 SBU
Ожидаемое место на диске: 186 MB

Описание Linux

Ядро Linux - это основа любой Linux системы. Это то, что делает Linux собой. Когда компьютер включается и загружает систему Linux, самым первым загружается ядро. Ядро инициализирует аппаратные компоненты: последовательные и параллельные порты, звуковые карты, сетевые карты, контроллеры IDE, контроллеры SCSI и много чего еще. Собственно, ядро делает доступным аппаратные элементы `cbscntvs` и позволяет запускаться программам.

Устанавливаемые файлы: ядро и заголовки ядра

Зависимости установки Linux

Linux зависит от: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

Установка заголовков ядра

Некоторые пакеты нуждаются в ссылках на заголовки ядра, и мы собираемся распаковать архив ядра, собрать и скопировать необходимые файлы туда, где их сможет найти gcc.

Подготовимся к установке заголовков:

```
make mrproper
```

Это гарантирует нам, что дерево с исходниками ядра будет абсолютно "чистым". Команда разработчиков ядра рекомендует выполнять эту команду перед каждой компиляцией ядра. Вы не можете быть абсолютно уверены в чистоте дерева исходников ядра даже после распаковки.

Создадим файл include/linux/version.h:

```
make include/linux/version.h
```

Создадим платформо-зависимую ссылку include/asm:

```
make symlinks
```

Установим платформо-зависимые файлы заголовков:

```
mkdir /tools/include/asm  
cp include/asm/* /tools/include/asm  
cp -R include/asm-generic /tools/include
```

Установим кросс-платформенные файлы заголовков:

```
cp -R include/linux /tools/include
```

Некоторые из заголовков ядра используют файл заголовков autoconf.h. Поскольку мы пока не сконфигурировали ядро, нам надо создать этот файл для того, чтобы компиляция следующих пакетов не закончилась ошибкой. Создадим пустой файл autoconf.h:

```
touch /tools/include/linux/autoconf.h
```

Установка Glibc-2.3.2

Ожидаемое время сборки: 11.8 SBU
Ожидаемое место на диске: 800 MB

Описание Glibc

Glibc является библиотекой С, которая обеспечивает системные вызовы и основные функции, такие как open, malloc, printf и т.д. Библиотека С используется для всех динамически скомпонованных программ.

Устанавливаемые программы: catchsegv, gencat, getconf, getent, glibcbug, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, mtrace, nsqd, nsqd_nischeck, pprofilerdump, pt_chown, rpcgen, rpcinfo, sln, sprof, tzselect, xtrace, zdump и zic

Устанавливаемые библиотеки: ld.so, libBrokenLocale.[a,so], libSegFault.so, libanl.[a,so], libbsd-compat.a, libc.[a,so], libc_nonshared.a, libcrypt.[a,so], libdl.[a,so], libg.a, libieee.a, libm.[a,so], libmcheck.a, libmemusage.so, libnsl.a, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libnss_nis.so, libnss_nisplus.so, libpcprofile.so, libpthread.[a,so], libresolv.[a,so], librpcsvc.a, librt.[a,so], libthread_db.so и libutil.[a,so]

Зависимости установки Glibc

Glibc зависит от: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Make, Perl, Sed, Texinfo.

Установка Glibc

Перед началом установки Glibc, вы должны перейти (с помощью команды cd, например) в директорию glibc-2.3.2 и распаковать Glibc-linuxthreads в этой директории, а не там где вы обычно распаковываете все исходники.

Замечание: Мы собираемся запустить тестирование для Glibc в этой главе. Это тестирование тут является менее важным, чем тестирование Glibc в [Главе 6](#).

Этот пакет известен своим нестабильным поведением при компиляции с измененными опциями оптимизации (включая опции -march и -mcrr). Glibc рекомендуется компилировать с настройками по

умолчанию. Следовательно, если вы задали переменные такие как `CFLAGS` или `CXXFLAGS`, изменяющие уровень оптимизации по умолчанию, рекомендуется убрать их при сборке пакета Glibc. Изменяя оптимизации для glibc, вы действуете на свой страх и риск.

Хотя это и безвредное сообщение, но при установке Glibc жалуется на отсутствие `/tools/etc/ld.so.conf`. Исправим это с помощью команд:

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

Также, Glibc имеет некоторые тонкие проблемы при компиляции с GCC 3.3.1. Применим следующий патч для их исправления:

```
patch -Np1 -i ./glibc-2.3.2-sscanf-1.patch
```

Документация по Glibc рекомендует собирать Glibc вне директории с исходниками, в отдельной директории для сборки:

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

Далее, подготовим Glibc к компиляции:

```
./glibc-2.3.2/configure --prefix=/tools \  
--disable-profile --enable-add-ons \  
--with-headers=/tools/include \  
--with-binutils=/tools/bin \  
--without-gd
```

Описание опций конфигурации:

- `--disable-profile`: Это отключает сборку библиотек с использованием профильной информации. Опустите эту опцию, если собираетесь использовать профили.
- `--enable-add-ons`: Это подключает любые дополнения при остановке Glibc, в нашем случае `Linuxthreads`.
- `--with-binutils=/tools/bin` and `--with-headers=/tools/include`: Стого говоря, эти опции необязательны. Но они позволяют нам удостовериться, что будут использованы нужные заголовки ядра и программы Binutils для сборки Glibc.
- `--without-gd`: Этот переключатель позволит нам быть уверенными, что не соберется программа `memusagetest`, которая будет пытаться подключить библиотеки из основной системы (`libgd`, `libpng`, `libz` и некоторые другие).

На этом шаге вы можете увидеть 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` безвредна, но может привести к определенным проблемам при тестировании.

Скомпилируем пакет:

```
make
```

Запустим тестирование:

```
make check
```

Тестирование Glibc сильно зависит от некоторых функций вашей основной системы, в частности ядра. Также, некоторые тесты в этой главе могут взаимодействовать с окружением вашей системы. Само собой, мы не должны получить таких проблем при запуске тестирования в [Главе 6](#). В общем, тестирование Glibc должно пройти удачно. Тем не менее, по причинам, перечисленным ниже, тестирование может закончиться неудачно. Вот список наиболее вероятных причин этого:

- Тест `math` иногда не проходит при его запуске на системе с процессорами, отличными от новых Intel-совместимых или оригинальных AMD. Также это может произойти при некоторых установках оптимизации.
- Тест `gettext` иногда не проходит из-за зависимостей от основной системы. Точная причина пока неясна.
- Тест `atime` иногда не проходит когда раздел LFS монтирован с опцией `noatime` или из-за других причин, связанных с файловой системой.
- Тест `shm` может не пройти, если в вашей системе запущена файловая система `devfs`, но нет файловой системы `tmpfs`, монтированной на `/dev/shm`, если отключена поддержка `tmpfs` в ядре.
- При запуске на старом и медленном оборудовании некоторые тесты могут не пройти по тайм-ауту.

В общем, вам не стоит беспокоится, если вы увидите, что тестирование Glibc не прошло. Glibc в [Главе 6](#) будет последним из устанавливаемых и его тестирование будет более важным. Но имейте в виду, что в [Главе 6](#) некоторые тесты могут также не пройти -- тест `math`, к примеру. Когда вы получите сообщение о непрохождении теста, запомните его, а затем продолжите тестирование дальше. Это можно сделать, так как

скрипт тестирования запоминает пройденные тесты для возможности его продолжения после выхода из-за ошибки. Вы можете использовать эту возможность "запуска-остановки" автоматически с помощью команды make -k check. Но если вы так сделаете, проверьте логи тестирования и посмотрите общее количество и причины проваленных тестов.

Теперь установим пакет:

```
make install
```

Разные страны и культуры имеют различные соглашения для коммуникаций. Эти соглашения состоят как из очень простых, таких как форматы даты и времени, так и из более сложных, таких как разговорный язык. "Интернационализация" программ GNU работает с помощью локалей (*locales*). Так что установим локали для Glibc:

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

Альтернативой запуску предыдущей команды является установка только определенных локалей, тех которые вам нужны. Это может быть достигнуто использованием команды localedef. Информацию об использовании этой команды можно получить из файла INSTALL в исходниках glibc-2.3.2. Тем не менее, список локалей может быть существенным для некоторых тестов, в частности, теста *libstdc++* из GCC. Следующие команды, используемые вместо вышеописанной *install-locales*, устанавливают минимальный набор локалей для успешного завершения тестирований:

```
mkdir -p /tools/lib/locale
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
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 es_MX -f ISO-8859-1 es_MX
localedef -i fr_FR -f ISO-8859-1 fr_FR
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i it_IT -f ISO-8859-15 it_IT
localedef -i ja_JP -f EUC-JP ja_JP
```

"Интеграция" Glibc

Теперь, когда временные библиотеки С были установлены, мы хотим чтобы все оставшиеся в этой главе пакеты компоновались с использованием уже установленных библиотек. Чтобы обеспечить это, мы должны настроить specs файл компилятора.

Для начала установим откорректированный компоновщик запуском следующей команды в директории binutils-build:

```
make -C ld install
```

Мы установили откорректированный компоновщик вместо того, который был установлен на первом шаге сборки Binutils. С этого момента все пакеты будут собраны *только* с использованием библиотек из /tools/lib.

Замечание: Если вы пропустили предупреждение о том, что не надо было удалять директории с исходниками и сборкой Binutils из первого шага или у вас нет доступа к ним, не беспокойтесь, не все потеряно. Просто проигнорируйте вышеприведенную команду. В результате появится небольшой шанс, что программы будут скомпонованы с использованием библиотек основной системы. Это не очень хорошо, но тем не менее не является такой уж большой проблемой. Ситуация будет исправлена позже на втором шаге установки Binutils.

Теперь, когда установлен откорректированный компоновщик, вы можете удалить директории с исходниками и сборкой Binutils.

Следующим нашим действием будет исправление specs-файла GCC для задания расположения нашего компоновщика. Просто выполните следующую команду:

```
SPECFILE=/tools/lib/gcc-lib/*/*/specs &&
sed -e 's@/lib/ld-linux.so.2@/tools/lib/ld-linux.so.2@g' \
$SPECFILE > tempspecfile &&
mv -f tempspecfile $SPECFILE &&
unset SPECFILE
```

Мы рекомендуем просто скопировать и вставить вышеуказанное вместо того, чтобы просто вводить. Или, при желании, вы можете просто отредактировать specs-файл вручную: замените встречающиеся строки "/lib/ld-linux.so.2" на "/tools/lib/ld-linux.so.2".

Важно: Если вы работаете на платформе, где имя динамического компоновщика отлично от ld-linux.so.2, вы *должны* заменить ld-linux.so.2 на имя динамического компоновщика вашей

платформы в вышеуказанных командах. При необходимости вернитесь к разделу ["Технические моменты"](#).

Есть возможность, что некоторые включаемые файлы из основной системы имеются внутри директорий для включаемых файлов GCC. Это могло произойти по причине того, что процесс "fixincludes" запускается как часть сборки GCC. Позже мы расскажем об этом подробнее в этой главе. А пока запустите следующую команду, чтобы исключить эту возможность:

```
rm -f /tools/lib/gcc-lib/*/*/include/{pthread.h,bits/sigthread.h}
```

Внимание

На этом месте необходимо остановиться и убедиться, что основные функции (компиляция и компоновка) новых средств работают корректно. Для этого есть простой тест:

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

Если все в порядке, то не будет ошибок и на выводе вы увидите:

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

Если эта надпись вообще не появилась или появилась другая, то чтото сильно не так. Вам надо исследовать и повторить все пройденные шаги, чтобы найти в чем проблема и устраниить ее. Точки для возврата после этого места уже не будет. Как правило, что-то не так бывает с вышеописаной правкой specs-фала. Убедитесь, что /tools/lib содержит префикс вашего динамического компоновщика. Само собой, если вы работаете на платформе с названием динамического компоновщика, отличным от ld-linux.so.2, вывод будет несколько иным.

Если все прошло нормально, удалим тестовые файлы:

```
rm dummy.c a.out
```

Теперь мы закончили установку самодостаточных средств, и они будут использованы для сборки оставшихся временных средств.

Установка Tcl-8.4.4

Ожидаемое время сборки: 0.9 SBU
Ожидаемое место на диске: 23 MB

Описание Tcl

Пакет Tcl содержит Tool Command Language (Язык Команд Средств).

Устанавливаемые программы: tclsh (link to tclsh8.4), tclsh8.4

Устанавливаемые библиотеки: libtcl8.4.so

Зависимости установки Tcl

Tcl зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Установка Tcl

Этот и следующие два устанавливаемых пакета нужны для работы тестирования GCC и Binutils. Установка этих трех пакетов нужна только для этого, и если вы не хотите тестировать устанавливаемые средства, то можно пропустить эти установки, но мы рекомендуем проверять работоспособность собираемых средств. Подготовим Tcl к компиляции:

```
cd unix  
./configure --prefix=/tools  
Соберем пакет:  
make
```

Этот пакет поддерживает тесты для определения корректности сборки. Тем не менее, тестирование Tcl в этой главе может не завершиться успешно из-за зависимостей от основной системы, которые полностью не понятны. Таким образом, неудачное завершение тестирования здесь не будет сюрпризом и не является критичным. Если вы решили протестировать Tcl, то запустите команду:

```
TZ=UTC make test
```

Описание параметров сборки:

- TZ=UTC: Это установит временную зону в Coordinated Universal Time (UTC) также известное как время по Гринвичу - Greenwich Mean Time (GMT), но только на время работы тестирования. Это позволит убедиться, что тестирование часов пройдет успешно. Больше информации о переменной окружения TZ можно получить в [Главе 7](#).

Иногда, тестирование пакета не проходит. Вы можете проконсультироваться на LFS Wiki по адресу <http://wiki.linuxfromscratch.org/> для проверки того, какие из ошибок нормальны. Это относится ко всем тестированиям в этой книге.

Установим пакет:

```
make install
```

Важно: Не удаляйте директорию с исходниками tcl8.4.4 сейчас, так как следующие пакеты будут нуждаться во внутренних заголовках из этой директории.

Создадим необходимую символьическую ссылку:

```
ln -s tclsh8.4 /tools/bin/tclsh
```

Установка Expect-5.39.0

Ожидаемое время установки: 0.1 SBU
Ожидаемое место на диске: 3.9 MB

Описание Expect

Пакет Expect содержит программы, обеспечивающую программируемый диалог с другими интерактивными программами.

Устанавливаемые программы: expect

Устанавливаемые библиотеки: libexpect5.39.a

Зависимости установки Expect

Expect зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed, Tcl.

Установка Expect

Для начала применим патч:

```
patch -Np1 -i ./expect-5.39.0-spawn.patch
```

Этот патч устранит неисправность в Expect который выдает неверный результат при тестировании GCC.

Теперь подготовим Expect к компиляции:

```
./configure --prefix=/tools --with-tcl=/tools/lib --with-x=no
```

Описание опций конфигурации:

- --with-tcl=/tools/lib: Это позволит убедиться, что скрипт конфигурации найдет Tcl в нашей временной директории для средств. Мы не хотим, чтобы использовался Tcl из основной системы.
- --with-x=no: Это скажет скрипту конфигурации не искать Tk (компонент Tcl GUI) или библиотеки X Window System, поскольку он найдет их на основной системе.

Соберем пакет:

```
make
```

Этот пакет поддерживает тестирование для проверки корректности сборки. Тем не менее, тестирование Expect здесь, в Главе Chapter 5, известно своими ошибками из-за влияния основной системы. Таким образом отрицательные результаты тестов не будут здесь сюрпризом и не являются критичными. Если вы захотите запустить тестирование, воспользуйтесь следующей командой:

```
make test
```

И установим:

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

Описание параметров сборки:

- SCRIPTS="": Этот параметр запускает установку без вспомогательных скриптов, которые не нужны.

Теперь вы можете удалить директории с исходниками Tcl и Expect.

Установка DejaGnu-1.4.3

Ожидаемое время сборки: 0.1 SBU
Ожидаемое место на диске: 8.6 MB

Описание DejaGnu

Пакет DejaGnu содержит основы для тестирования других программ.
Устанавливаемые программы: runtest

Зависимости установки DejaGnu

Dejagnu зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Установка DejaGnu

Подготовим DejaGnu к компиляции:

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

Соберем и установим пакет:

```
make install
```

Установка GCC-3.3.1 - Шаг 2

Ожидаемое время сборки: 11.0 SBU
Ожидаемое место на диске: 274 MB

Переустановка GCC

Средства, необходимые для тестирования GCC и Binutils, теперь установлены (Tcl, Expect и DejaGnu). Мы можем вернуться к пересборке GCC и Binutils, соединить их с новым Glibc и правильно их протестировать. Но есть одно замечание, тестирование сильно зависит от правильного функционирования терминалов pseudo (PTY) из основной системы. На данный момент, PTY осуществляется с помощью файловой системы *devpts*. Вы можете быстро проверить правильность установки системы командой:

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

Если вы получили ответ:

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

Ваш основной дистрибутив не поддерживает операции PTY. В этом случае здесь не место запуску тестирования для GCC и Binutils и вы можете пропустить его. Вы можете проконсультироваться в LFS Wiki на <http://wiki.linuxfromscratch.org/> для более подробной информации о работе с PTY.

Распакуйте все три тарбола GCC (-core, -g++ и -testsuite) в одной рабочей директории. Они распакуются в одну общую поддиректорию *gcc-3.3.1/*.

Для начала исправим одну проблему и создадим необходимую сборку:

```
patch -Np1 -i ./gcc-3.3.1-no_fixincludes-2.patch  
patch -Np1 -i ./gcc-3.3.1-specs-2.patch
```

Первый патч отключит скрипт GCC "fixincludes". Мы расскажем об этом вкратце, не вдаваясь в подробности. При нормальных обстоятельствах скрипт GCC fixincludes сканирует вашу систему на файлы заголовков Glibc, нуждающиеся в исправлении, исправляет их и переносит их в директорию включений для GCC. После чего, в [Главе 6](#), после установки нового Glibc, эта директория будет найдена, и в результате GCC найдет заголовки основной системы, и мы не сможем корректно использовать Glibc в системе LFS. Последний патч изменяет расположение по умолчанию для динамического компоновщика GCC (обычно *ld-linux.so.2*). Он также удаляет */usr/include* из пути для поиска GCC. Пропатчивание specs-файла сейчас позволит убедиться, что наш новый динамический компоновщик будет использоваться собираемым GCC. То есть, наши окончательные (и временные) бинарники будут скомпонованы с новым Glibc.

Важно: Эти патчи являются *критичными* для корректной сборки. Не забудьте применить их.

Создайте отдельную директорию для сборки и перейдите в нее:

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

Перед началом сборки GCC не забудьтеdezактивировать все переменные окружения, в которых были изменены флаги оптимизации.

Теперь подготовим GCC к компиляции:

```
./gcc-3.3.1/configure --prefix=/tools \
--with-local-prefix=/tools \
--enable-locale=gnu --enable-shared \
--enable-threads=posix --enable-__cxa_atexit \
--enable-languages=c,c++
```

Описание опций конфигурации:

- `--enable-threads=posix`: Это подключит расширение C++ для мультилинейного кода.
- `--enable-__cxa_atexit`: Эта опция разрешит использование `__cxa_atexit`, которое предпочтительнее использования `atexit`, для регистрации деструкторов C++ для локальных и глобальных объектов для обеспечения полного соответствия стандартам ссылок на деструкторы. Он также повлияет на C++ ABI и некоторые другие результаты в библиотеках C++ и программах C++, которые взаимодействуют с другими дистрибутивами Linux.
- `--enable-locale=gnu`: Эта опция позволяет убедиться в корректном выборе модели локали для библиотек C++ во всех обстоятельствах. Если скрипт конфигурации найдет установленную локаль `de_DE`, он выберет корректную модель `gnu`. Тем не менее, люди, которые не установили локаль `de_DE` рискуют собрать ABI-несовместимую библиотеку C++ из-за неверной модели локали в общем случае.
- `--enable-languages=c,c++`: Эта опция позволяет убедиться, что будут собраны компиляторы C и C++.

Скомпилируем пакет:

```
make
```

Здесь нет необходимости использовать цель `bootstrap`, так как мы компилируем этот GCC с помощью той же самой версии GCC, но установленной ранее.

Замечание: Запуск тестирования здесь не настолько важен, как в [Глава 6](#).

Протестируем результаты:

```
make -k check
```

Флаг `-k` используется для того, чтобы тестирование не прекратилось после первого отрицательного результата, а проводило дальнейшие тесты. Тестирование GCC довольно исчерпывающее, и поэтому мы практически гарантировано получим пару отрицательных результатов тестов. Для просмотра результатов тестирования выполните команду:

```
./gcc-3.3.1/contrib/test_summary | more
```

Вы можете сравнить ваши результаты с результатами из списка рассылки для того, чтобы найти аналогичные вашим. К примеру, посмотреть как результаты тестирования GCC-3.3.1 на i686-pc-linux-gnu можно на <http://gcc.gnu.org/ml/gcc-testresults/2003-08/msg01612.html>.

Проверите, чтобы в результатах были:

```
* 1 XPASS (unexpected pass) for g++
* 1 FAIL (unexpected failure) for g++
* 2 FAIL for gcc
* 26 XPASS's for libstdc++
```

Неожиданным прохождением тест для g++ обязан использованию `--enable-__cxa_atexit`. Но не все платформы поддерживающие GCC поддерживают также `"__cxa_atexit"` в своих библиотеках C, поэтому этот тест не всегда может пройти..

26 неожиданных успешных тестов libstdc++ обязаны использованию `--enable-locale=gnu`, которая корректирует выбор на Glibc-based системах версий 2.2.5 и выше. Поддержка основных локалей в библиотеке GNU C важнее, в противном случае выбирается "общая" модель (которая может быть преминима в случаях использования Newlib, Sun-libc или других libc). Тестирование libstdc++ в случае применения "основной" модели для тестов и не всегда может пройти успешно.

Неожиданные отрицательные результаты также не могут быть проигнорированы. Разработчики GCC обычно знают о них, но пока не могут исправить. В общем, проверьте результаты, как было описано выше. Если все впорядке, то можно продолжать.

Наконец установим пакет:

```
make install
```

Замечание: В этом месте рекомендуется повторить простой тест, описанный ранее. Вернитесь к [части "Интеграция" Glibc](#) и повторите проверку. Если она не удалась, видимо, вы забыли наложить вышеупомянутый патч GCC Specs.

Установка Binutils-2.14 - Шаг 2

Ожидаемое время сборки: 1.5 SBU
Ожидаемое место на диске: 108 MB

Переустановка Binutils

Снова создадим отдельную директорию для сборки:

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

Теперь подготовим Binutils к компиляции:

```
../binutils-2.14/configure --prefix=/tools \  
--enable-shared --with-lib-path=/tools/lib
```

Описание опций конфигурации:

- `--with-lib-path=/tools/lib`: Этот параметр передает скрипту конфигурации путь по умолчанию для поиска библиотек. Мы не хотим, чтобы в пути поиска оказались библиотеки из нашей основной системы.

Перед началом сборки Binutils не забудьте сбросить значения переменных окружения с флагами оптимизации по умолчанию.

Откомпилируем пакет:

```
make
```

Замечание: Здесь не обязательно запускать тестирование Binutils, так как это не настолько важно, как в [Главе 6](#).

Протестируем результаты:

```
make check
```

К сожалению, это нет простого пути увидеть результаты теста, как в предыдущем пакете GCC. Тем не менее, если тестирование не прошло, то это сразу будет видно. На выводе будет что-то наподобие:

```
make[1]: *** [check-binutils] Error 2
```

И установим пакет:

```
make install
```

Теперь подготовим Binutils к переопределению средств в следующей главе:

```
make -C ld clean  
make -C ld LIB_PATH=/usr/lib:/lib
```

Внимание

Не удаляйте сейчас директории с исходниками и сборкой Binutils. Они будут нужны нам в следующей главе в их нынешнем виде.

Установка Gawk-3.1.3

Ожидаемое время сборки: 0.2 SBU
Ожидаемое место на диске: 17 MB

Описание Gawk

Gawk является реализацией awk, который используется для манипуляций с текстовыми файлами.

Устанавливаемые программы: awk (link to gawk), gawk, gawk-3.1.3, grcat, igawk, pgawk, pgawk-3.1.3 и pwcat

Зависимости установки Gawk

Gawk зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Установка Gawk

Подготовим Gawk к компиляции:

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

Откомпилируем пакет:

```
make
```

Этот пакет поддерживает тестирование корректности сборки. Вы можете запустить его командой:

```
make check
```

И установим его:

```
make install
```

Установка Coreutils-5.0

Ожидаемое время сборки: 0.9 SBU

Ожидаемое место на диске: 69 MB

Описание Coreutils

Пакет Coreutils содержит некоторые простые системные утилиты.

Устанавливаемые программы: basename, cat, chgrp, chmod, chown, chroot, cksum, comm, cp, esplit, cut, date, dd, df, dir, dircolors, dirname, du, echo, env, expand, expr, factor, false, fmt, fold, groups, head, hostid, hostname, id, install, join, kill, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mv, nice, nl, nohup, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, rm, rmdir, seq, sha1sum, shred, sleep, sort, split, stat, stty, su, sum, sync, tac, tail, tee, test, touch, tr, true, tsort, tty, uname, unexpand, uniq, unlink, uptime, users, vdir, wc, who, whoami и yes

Зависимости установки Coreutils

Coreutils зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed.

Установка Coreutils

Подготовим Coreutils к компиляции:

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

Скомпилируем пакет:

```
make
```

Этот пакет поддерживает тестирование корректности сборки. Вы можете запустить его командой:

```
make RUN_EXPENSIVE_TESTS=yes check
```

Описание параметров make:

- RUN_EXPENSIVE_TESTS=yes: Это заставит запустить некоторые дополнительные тесты, которые требуют достаточно много времени на некоторых платформах. Тем не менее, это не проблема для Linux.

И установим пакет:

```
make install
```

Установка Bzip2-1.0.2

Ожидаемое время сборки: 0.1 SBU

Ожидаемое место на диске: 2.5 MB

Описание Bzip2

Bzip2 является блочным файловым архиватором (block-sorting file compressor), который сжимает, как правило, лучше традиционного gzip.

Устанавливаемые программы: bunzip2 (link to bzip2), bzcat (link to bzip2), bzcmp, bzdiff, bzegrep, bzfgrep, bzgrep, bzip2, bzip2recover, bzless и bzmore

Устанавливаемые библиотеки: libbz2.a, libbz2.so (link to libbz2.so.1.0), libbz2.so.1.0 (link to libbz2.so.1.0.2) и libbz2.so.1.0.2

Зависимости установки Bzip2

Bzip2 зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make.

Установка Bzip2

Пакет Bzip2 не содержит скрипта configure. Он компилируется и устанавливается командой:
make PREFIX=/tools install

Установка Gzip-1.3.5

Ожидаемое время сборки: 0.1 SBU
Ожидаемое место на диске: 2.6 MB

Описание Gzip

Пакет Gzip содержит программы для сжатия и распаковки файлов с использованием кодирования Lempel-Ziv (LZ77).

Устанавливаемые программы: gunzip (ссылка на gzip), gexe, gzip, uncompress (ссылка на gunzip), zcat (ссылка на gzip), zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore и znew

Зависимости установки Gzip

Gzip зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Установка Gzip

Подготовим Gzip к установке:

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

Скомпилируем пакет:

```
make
```

И установим его:

```
make install
```

Установка Diffutils-2.8.1

Ожидаемое время сборки: 0.1 SBU
Ожидаемое место на диске: 7.5 MB

Описание Diffutils

Программы из этого пакета могут показать вам различия между двумя файлами или директориями. Они обычно используются для создания различных патчей.

Устанавливаемые программы: cmp, diff, diff3 и sdiff

Зависимости установки Diffutils

Diffutils зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Установка Diffutils

Подготовим Diffutils к компиляции:

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

Скомпилируем пакет:

```
make
```

И установим его:

```
make install
```

Установка Findutils-4.1.20

Ожидаемое время сборки: 0.2 SBU
Ожидаемое место на диске: 7.6 MB

Описание Findutils

Пакет Findutils содержит программы для поиска файлов, в том числе "на лету" (путем рекурсивного поиска от директории и показывая только файлы удовлетворяющие параметрам поиска) или поиск через базу данных.

Устанавливаемые программы: bigram, code, find, frcode, locate, updatedb и xargs

Зависимости установки Findutils

Findutils зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Установка Findutils

Подготовим Findutils к компиляции:

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

Скомпилируем пакет:

```
make
```

Этот пакет поддерживает тестирование корректности сборки. Если вы хотите запустить его, то выполните следующую команду:

```
make check
```

И установим пакет:

```
make install
```

Установка Make-3.80

Ожидаемое время сборки: 0.2 SBU

Ожидаемое место на диске: 8.8 MB

Описание Make

Make автоматически определяет, какие части большой программы должны быть перекомпилированы и вызывает команды для их перекомпиляции.

Устанавливаемые программы: make

Зависимости установки Make

Make зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Sed.

Установка Make

Подготовим Make к компиляции:

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

Скомпилируем программу:

```
make
```

Этот пакет поддерживает тестирование корректности сборки. Если вы хотите запустить его, то выполните следующую команду:

```
make check
```

Теперь установим пакет и документацию к нему:

```
make install
```

Установка Grep-2.5.1

Ожидаемое время сборки: 0.1 SBU

Ожидаемое место на диске: 5.8 MB

Описание Grep

Grep является программой, используемой для печати строк из файла, которые совпадают с заданным шаблоном.

Устанавливаемые программы: egrep (ссылка на grep), fgrep (ссылка на grep) и grep

Зависимости установки Grep

Grep зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Make, Texinfo.

Установка Grep

Подготовим Grep к компиляции:

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

Описание параметров конфигурации:

- `--disable-perl-regexp`: Это позволит убедиться, что grep не будет скомпонован с использованием библиотеки PCRE, которая может присутствовать в основной системе, но не будет доступна из среды chroot.
- `--with-included-regex`: Это позволит убедиться, что Grep использует внутренние регулярные выражения. В противном случае, он будет использовать код из Glibc, который может содержать ошибки.

Скомпилируем программу:

```
make
```

Этот пакет поддерживает тестирование корректности сборки. Если вы хотите его использовать, выполните команду:

```
make check
```

Теперь установим пакет и документацию к нему:

```
make install
```

Установка Sed-4.0.7

Ожидаемое время сборки: 0.2 SBU

Ожидаемое место на диске: 5.2 MB

Описание Sed

Sed является редактором потоков. Редактор потоков используется для выполнения простых преобразований текста на входном потоке (файл или стандартный ввод).

Устанавливаемые программы: sed

Зависимости установки Sed

Sed зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Texinfo.

Установка Sed

Подготовим Sed к компиляции:

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

Скомпилируем программу:

```
make
```

Этот пакет поддерживает тестирование корректности сборки. Если вы хотите использовать его, то выполните команду:

```
make check
```

Теперь установим пакет и документацию:

```
make install
```

Установка Gettext-0.12.1

Ожидаемое время сборки: 7.2 SBU

Ожидаемое место на диске: 55 MB

Описание Gettext

Пакет Gettext используется для интернационализации и локализации. Программы могут быть скомпилированы с Поддержкой Родного Языка (Native Language Support - NLS) для получения возможности вывода сообщений на языке пользователя.

Устанавливаемые программы: autopoint, config.charset, config.rpath, gettext, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, project-id, team-address, trigger, urlget, user-email и xgettext
Устанавливаемые библиотеки: libasprintf[a,so], libgettextlib[a,so], libgettextpo[a,so] и libgettextsrc[a,so]

Зависимости установки Gettext

Gettext зависит от: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Установка Gettext

Подготовим Gettext к компиляции:

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

Скомпилируем пакет:

```
make
```

Этот пакет поддерживает тестирование корректности сборки. Тем не менее, тестирование Gettext в этой главе может не пройти успешно из-за зависимостей от основной системы - к примеру, если будет найден компилятор Java. Тестирование Gettext занимает много времени и не является критичным. Поэтому мы не рекомендуем запускать его здесь. Если же вы все-таки захотите использовать его, выполните команду:

```
make check
```

И установим пакет:

```
make install
```

Установка Ncurses-5.3

Ожидаемое время сборки: 0.7 SBU

Ожидаемое место на диске: 26 MB

Описание Ncurses

Пакет Ncurses содержит библиотеки для расширения возможностей текстового интерфейса, включая панели и меню.

Устанавливаемые программы: captoinfo (link to tic), clear, infocmp, infotocap (link to tic), reset (link to tset), tack, tic, toe, tput и tset

Устанавливаемые библиотеки: libcurses.[a,so] (link to libncurses.[a,so]), libform.[a,so], libmenu.[a,so], libncurses++.a, libncurses.[a,so], libpanel.[a,so]

Зависимости установки Ncurses

Ncurses зависит от: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Установка Ncurses

Сначала сделаем следующее:

```
patch -Np1 -i ./ncurses-5.3-etip-2.patch  
patch -Np1 -i ./ncurses-5.3-vsscanf.patch
```

Первый патч скорректирует файл заголовков etip.h, а второй - исправит некоторые предупреждения компилятора при присутствии конкурирующих библиотек.

Теперь подготовим Ncurses к компиляции:

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

Описание параметров конфигурации:

- **--without-ada:** Это заставит Ncurses не собираться с использованием Ada, если он установлен в основной системе. Эта нам нужно потому, что при работе в среде chroot, Ada не будет доступен.
- **--enable-overwrite:** Это скажет Ncurses устанавливать заголовки в /tools/include, а не в /tools/include/ncurses, нам надо убедиться, что другие пакеты смогут эти заголовки найти.

Скомпилируем программы и библиотеки:

```
make
```

Теперь установим пакет и документацию:
make install

Установка Patch-2.5.4

Ожидаемое время сборки: 0.1 SBU
Ожидаемое место на диске: 1.9 MB

Описание Patch

Программа patch модифицирует файлы в соответствии с файлом патча. Файл с патчем - это, как правило, список, сгенерированный программой diff, который содержит инструкции к тому, какие из оригинальных файлов ныряются в мозгификации.

Устанавливаемые программы: patch

Зависимости установки Patch

Patch зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Установка Patch

Подготовим Patch к компиляции:

CPPFLAGS=-D_GNU_SOURCE ./configure --prefix=/tools

Флаг препроцессора -D_GNU_SOURCE нужен только на платформах PowerPC. На других платформах вы можете его пропустить.

Скомпилируем программу:

make

Теперь установим пакет и документацию:

make install

Установка Tar-1.13.25

Ожидаемое время сборки: 0.2 SBU
Ожидаемое место на диске: 10 MB

Описание Tar

Tar является архиватором разработанным для сохранения и распаковки файлов в и из архивов, известных как файлы tar (тарболы, tarballs).

Устанавливаемые программы: rmt и tar

Зависимости установки Tar

Tar зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Установка Tar

Подготовим Tar к компиляции:

./configure --prefix=/tools

Скомпилируем программы:

make

Этот пакет поддерживает тестирование корректности сборки. Если вы хотите его использовать, то выполните команду:

make check

Теперь установим пакет и документацию:

make install

Установка Texinfo-4.6

Ожидаемое время сборки: 0.2 SBU
Ожидаемое место на диске: 16 MB

Описание Texinfo

Пакет Texinfo содержит программы, используемые для чтения, записи и конвертирования документов Info, которые содержат системную документацию.

Установляемые программы: info, infokey, install-info, makeinfo, texi2dvi и texindex

Зависимости установки Texinfo

Texinfo зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed.

Установка Texinfo

Подготовим Texinfo к компиляции:

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

Скомпилируем программы:

```
make
```

Этот пакет поддерживает тестирование корректности сборки. Если вы хотите использовать его, то выполните команду:

```
make check
```

Теперь установим пакет и документацию:

```
make install
```

Установка Bash-2.05b

Ожидаемое время сборки: 1.2 SBU

Ожидаемое место на диске: 27 MB

Описание Bash

Bash - это Bourne-Again SHell, который обычно используется как интерпритатор команд на Unix системах. Программа bash считывает данные со стандартного ввода (клавиатура). Пользователь вводит что-либо, и программа определяет, что именно пользователь ввел и что с этим делать, например, запустить другую программу.

Установляемые программы: bash, sh (ссылка на bash) и bashbug

Зависимости установки Bash

Bash зависит от: Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Sed.

Установка Bash

Bash содержит несколько известных ошибок. Исправим их, применив патч:

```
patch -Np1 -i ./bash-2.05b-2.patch
```

Теперь подготовим Bash к компиляции:

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

Скомпилируем программу:

```
make
```

Этот пакет поддерживает тестирование корректности сборки. Если вы хотите использовать эту возможность, то выполните команду:

```
make tests
```

Теперь установим пакет и документацию:

```
make install
```

И создадим ссылку для программ, которые используют sh в качестве шелла:

```
ln -s bash /tools/bin/sh
```

Установка Util-linux-2.12

Ожидаемое время сборки: 0.1 SBU

Ожидаемое место на диске: 8 MB

Описание Util-linux

Пакет Util-linux содержит ряд разнообразных утилит. Некоторые из них используются весьма часто для монтирования, размонтирования, форматирования и управления драйверами дисков, открытия портов tty и вывода сообщений ядра.

Устанавливаемые программы: agetty, arch, blockdev, cal, cfdisk, chkdupexe, col, colcrt, colrm, column, ctrlaltdel, cytune, ddate, dmesg, elvtune, fdformat, fdisk, fsck.cramfs, fsck.minix, getopt, hexdump, hwclock, iperm, ipcs, isosize, kill, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, parse.bash, parse.tcsh, pg, pivot_root, ramsize (ссылка на rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (ссылка на rdev), script, setfdprm, setsid, setterm, sdfdisk, swapoff (ссылка на swapon), swapon, test.bash, test.tcsh, tunelp, ul, umount, vidmode (ссылка на rdev), whereis и write

Зависимости установки Util-linux

Util-linux зависит от: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Zlib.

Установка Util-linux

Util-linux не может использовать заголовки и библиотеки из директории /tools. Это исправляется с помощью исправления скрипта конфигурации:

```
cp configure configure.backup  
sed "s@/usr/include@/tools/include@g" configure.backup > configure
```

Подготовим Util-linux к компиляции:

```
./configure
```

Скомпилируем поддержку некоторых шаблонов:

```
make -C lib
```

И, поскольку нам нужны только некоторые из утилит из этого пакета, соберем только их:

```
make -C mount mount umount  
make -C text-utils more
```

Теперь скопируем эти файлы в директорию временных средств:

```
cp mount/{,u}mount text-utils/more /tools/bin
```

Установка Perl-5.8.0

Ожидаемое время сборки: 0.8 SBU
Ожидаемое место на диске: 74 MB

Описание Perl

Пакет Perl содержит perl, Языка Практичной Обработки и Отчетов (Practical Extraction and Report Language). Perl собрал в себе некоторые лучшие свойства C, sed, awk и sh в одном мощном языке.

Устанавливаемые программы: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (ссылка на perl), perlbug, perlcc, perldoc, perlivp, piconv, pl2pm, pod2html, pod2latex, pod2man, pod2text, pod2usage, podchecker, podselect, psed (ссылка на s2p), pstruct (ссылка на c2ph), s2p, splain и xsubpp

Устанавливаемые библиотеки: (слишком много названий)

Зависимости установки Perl

Perl зависит от: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Установка Perl

Для начала применим патч для библиотеки C:

```
patch -Np1 -i ../perl-5.8.0-libc-3.patch
```

И убедимся, что некоторые статические расширения будут собраны:

```
chmod u+w hints/linux.sh  
echo 'static_ext="IO re Fcntl"' >> hints/linux.sh
```

Теперь подготовим Perl к компиляции:

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

Скомпилируем только необходимые средства:

```
make perl utilities
```

Теперь скопируем эти средства и их библиотеки:

```
cp perl pod/pod2man /tools/bin
```

```
mkdir -p /tools/lib/perl5/5.8.0  
cp -R lib/* /tools/lib/perl5/5.8.0
```

Сжатие

Описываемый здесь шаг не является обязательным. Если ваш раздел LFS очень мал, вы хотите просто попробовать или у вас есть другие причины, то вы можете выполнить инструкции этого раздела. Исполняемые файлы и библиотеки собираются с использованием ненужных символов отладки - это около 130 МБ. Вы можете удалить эти символы командами:

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

Первая команда пропустит около двадцати файлов, сообщив что они неподдерживаемого формата. Большая часть из них скрипты, я не бинарники.

Ни в коем случае *не используйте* параметр --strip-unneeded для библиотек -- они будут испорчены и вам придется собирать их все заново вместе с Glibc.

Для освобождения еще пары мегабайт, вы можете убрать всю документацию:

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

Вам понадобится около 850 МБ свободного места на вашей системе LFS для для сборки и установки Glibc на следующем шаге. Если вы сможете собрать и установить Glibc, то вы сможете собрать и установить все остальное.

III. Часть III - Сборка системы LFS

Глава 6. Установка основных системных программ

Вступление

В этой главе мы войдем в фазу сборки и начнем серьезно собирать нашу LFS систему. Для этого мы войдем в среду chroot внутри нашей мини Linux системы, проделаем некоторые дополнительные работы и начнем устанавливать пакеты один за одним.

Установка всех этих пакетов весьма проста, и вы можете подумать, что достаточно было бы указать общее описание установки для всех пакетов, а конкретизировать только если установка их отличается от стандартной. Несмотря на то, что мы согласны с этим, мы решили дать полные инструкции по установке каждого из пакетов просто для того, чтобы минимизировать возможность возникновения ошибок.

Если вы решите использовать оптимизацию компилятора в этой главе, то вам стоит посмотреть справку по оптимизации на <http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt>. Оптимизация компилятора может сделать программы быстрее, но могут возникнуть сложности при компиляции и некоторые проблемы при работе этих программ. Если пакет отказывается компилироваться при использовании оптимизации, то попробуйте скомпилировать пакет без нее, возможно проблемы исчезнут. Если пакет компилируется с использованием оптимизации, то есть возможность, что он скомпилируется некорректно, например, могут появиться проблемы между интерактивностью в коде и собраными средствами. В общем, использование оптимизации при сборке является рискованным. Новичкам в LFS рекомендуется собирать ее без оптимизации. В ваших силах сделать систему быстрой и стабильной одновременной.

Список пакетов, которые надо собирать, является строго последовательным, нам надо убедиться, что в них не будет прописан путь к /tools. По некоторым причинам, *не стоит* компилировать пакеты параллельно.

Параллельная сборка может показаться выгодной иногда (в особенности на многопроцессорных системах), но в результате у программ появится зависимость от средств из /tools и они не смогут работать, когда мы удалим эту директорию.

Про отладочные символы

Большинство программ и библиотек по умолчанию компилируются с символами отладки (с опцией -g для gcc).

Когда отлаживается программа или библиотека, которая была собрана с включением отладочной информации, то отладчик может передать вам не только адреса в памяти, но и имена переменных и функций.

Но исключение отладочных символов существенно уменьшает размер программы или библиотеки. Чтобы получить представление о том, сколько места занимает отладочная информация, посмотрите следующее:

- бинарник bash с отладочными символами: 1200 KB
- бинарник bash без отладочных символов: 480 KB
- файлы Glibc и GCC (*/lib* и */usr/lib*) с отладочными символами: 87 MB
- файлы Glibc и GCC без отладочных символов: 16 MB

Размер может немного варьироваться в зависимости от используемых компилятора и библиотеки C. Но размер между программами и библиотеками, собранными с отладочной информацией и без таковой может различаться в 2-5 раз.

Большинство людей никогда не используют отладчик, и довольно много места на диске освободится при удалении отладочных символов.

Для удаления отладочных символов из бинарников (которые должны быть бинарниками форматов a.out или ELF), запустите `strip --strip-debug filename`. Могут использоваться шаблоны для выбора нескольких файлов (например, `strip --strip-debug $LFS/tools/bin/*`).

Для вашего удобства, [Глава 9](#) включает одну простую команду для удаления всех отладочных символов изо всех программ и библиотек в вашей системе. Дополнительную информацию по оптимизации можно получить на <http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt>.

Вход в среду chroot

Теперь пришло время войти в среду chroot для установки необходимых пакетов. Перед тем, как войти в chroot, вам необходимо войти в систему как *root*, потому что только *root* может выполнить команду chroot. Убедитесь, что переменная окружения LFS установлена корректно запуском `echo $LFS`, и проверьте, что она содержит путь к точке монтирования раздела LFS, к примеру `/mnt/lfs`.

Из-под привилегий *root* запустите следующую команду для входа в среду chroot:

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

Параметр `-i`, передаваемые команде env, очистят все переменные окружения среды chroot. После этого мы устанавливаем только переменные HOME, TERM, PS1 и PATH. Конструкция `TERM=$TERM` означает, что переменная окружения TERM внутри среды chroot примет то же значение, что и вне среды; Эта переменная нужна таким программам как vim и less для правильной работы. Если вам нужны другие переменные, такие как CFLAGS или CXXFLAGS, то это хорошее место задать их.

С этого места нам уже не надо использовать переменную LFS, потому что директория, указанная в ней стала корневой при входе в среду chroot.

Мы указали `/tools/bin` последней в списке путей PATH. Благодаря этому пакеты из этой директории не будут использованы, если мы уже установили окончательную версию соответствующего пакета. Это действует в том случае, когда shell не запоминает расположение исполняемых файлов, именно поэтому мы отключили эту функцию в предыдущей главе.

Запомните, что все команды до конца этой главы и во всех последующих выполняются из среды chroot.

Если вы покинули эту среду по любой причине (например, после перезагрузки), то вы должны снова зайти в среду chroot и применить файловые системы proc и devpts (описываются позже) перед продолжением установки.

Заметьте, что bash выводит "I have no name!". Это нормально, поскольку `/etc/passwd` еще не создан.

Изменение владельца

На данный момент, права на директорию `/tools` принадлежат пользователю *lfs*, пользователю, который существует только на основной системе. Хотя мы собираемся удалить директорию `/tools` когда наша система будет собрана, вы можете захотеть не делать этого. Например, для построения другой системы LFS. Но если вы хотите оставить директорию `/tools`, то у нее будет владелец с определенным ID но без аккаунта. Это опасно потому, что впоследствии созданный пользовательский аккаунт может получить такой же ID и станет владельцем директории `/tools` и всех файлов внутри нее, и использовать это в своих целях.

Чтобы избежать этого, вы можете добавить пользователя *lfs* в вашу новую LFS при создании файла `/etc/passwd`, и подправить его таким образом, чтобы ID этого пользователя и группы были идентичны им же на основной системе. Помимо этого, вы можете (и в этой книге так и делается) связать содержимое директории `/tools` с пользователем *root* запуском команды:

```
chown -R 0:0 /tools
```

Команда использует "0:0" вместо "root:root" потому, что chown не может использовать имя "root" до того, как будет создан файл паролей.

Создание директорий

Теперь нам надо создать структуру файловой системы LFS. Создадим дерево директорий, используя следующие команды:

```
mkdir -p /{bin,boot,dev/{pts,shm},etc/opt/home,lib,mnt,proc}
mkdir -p /{root,sbin,tmp,usr/local,var,opt}
for dirname in /usr /usr/local
do
mkdir $dirname/{bin,etc,include,lib,sbin,share,src}
ln -s share/{man,doc,info} $dirname
mkdir $dirname/share/{dict,doc,info,locale,man}
mkdir $dirname/share/{nls,misc,terminfo,zoneinfo}
mkdir $dirname/share/man/man{1,2,3,4,5,6,7,8}
done
mkdir /var/{lock,log,mail,run,spool}
mkdir -p /var/{tmp,opt,cache,lib/misc,local}
mkdir /opt/{bin,doc,include,info}
mkdir -p /opt/{lib,man/man{1,2,3,4,5,6,7,8}}
```

По умолчанию, директории создаются с правами доступа 755, но это правильно не для всех из них. Мы сделаем два изменения: одно для домашней директории пользователя *root*, а другое для директории для временных файлов.

```
chmod 0750 /root
chmod 1777 /tmp /var/tmp
```

Первой командой мы запрещаем доступ в директорию */root* для всех, кроме владельца -- тоже самое нам надо будет потом сделать для домашних директорий остальных пользователей.. Второй командой мы позволяем всем пользователям записывать и читать файлы в и из директорий */tmp* и */var/tmp* directories, но они не смогут удалять оттуда файлы других пользователей. Последнее достигается установкой "sticky bit" -- старшего бита в битовой маске 1777.

Совместимость с FHS

Мы создаем наше дерево каталогов совместимым со стандартом FHS (доступен на <http://www.pathname.com/fhs/>). За исключением директорий */usr/local/games* и */usr/share/games*, поскольку они не нужны на минимальной системе. Тем не менее можно сделать вашу систему FHS-совместимой. Поскольку структура субдиректорий */usr/local/share* не описана в стандарте FHS, то мы создаем ее так, как нам надо.

Монтирование файловых систем *proc* и *devpts*

Для правильного функционирования основных программ файловые системы *proc* и *devpts* должны быть доступны из среды chroot. Файловые системы могут монтироваться несколько раз в различных точках, так что то, что они будут смонтированы и на вашей основной системе, не создаст проблем - особенно потому, что они являются виртуальными файловыми системами.

Файловая система *proc* является pseudo-системой с информацией о процессах, которая используется ядром для представления информации о статусе системы.

Файловая система *proc* монтируется в */proc* запуском команды:

```
mount proc /proc -t proc
```

Вы можете получить предупреждение от команды *mount* следующего вида:

```
warning: can't open /etc/fstab: No such file or directory
not enough memory
```

Проигнорируйте его, оно выводится только потому, что система еще не полностью установлена и отсутствуют некоторые файлы. Команда *mount* все равно прошла успешно и это все, о чем вам следует беспокоиться здесь.

Файловая система *devpts* описывается проще, она обеспечивает доступ к терминалам pseudo (PTY).

Файловая система *devpts* монтируется в */dev/pts* командой:

```
mount devpts /dev/pts -t devpts
```

Если эта команда не прошла, то появится сообщение вида:

```
filesystem devpts not supported by kernel
```

Как правило, это означает, что ядро вашей основной системы было скомпилировано без поддержки файловой системы *devpts*. Вы можете проверить какие файловые системы поддерживаются ядром запуском команды *cat /proc/filesystems*. Если файловая система с названием *devfs* будет присутствовать в этом списке, то вы можете пока работать без проблем смонтирував основную файловую систему *devfs* в корне структуры */dev*, которую мы создадим позднее в разделе "Создание устройств (Makedev)". Если *devfs* нет в списке, то не

беспокойтесь, потому что не обязательно использовать именно такую работу с PTY в среде chroot. Мы исправим это в последующем разделе Makedev. Запомните, если по любой причине вы прекратили работу с LFS и решили возобновить позже, важно убедиться, что эти системы будут смонтированы внутри среды chroot, иначе могут возникнуть определенные проблемы.

Создание необходимых ссылок

Некоторые программы имеют встроенные пути к программам, которые пока не установлены. Чтобы обеспечить доступ к этим программам, мы создадим ссылки на них которые заменят пока нам реальные файлы до тех пор, пока мы их не установим.

```
ln -s /tools/bin/{bash,cat,pwd,stty} /bin  
ln -s /tools/bin/perl /usr/bin  
ln -s /tools/lib/libgcc_s.so.1 /usr/lib  
ln -s bash /bin/sh
```

Создание файлов passwd и group

Для возможности регистрации в качестве пользователя *root* и для распознавания имени "toot", нам необходимо создать соответствующие элементы в файлах /etc/passwd и /etc/group.

Создадим файл /etc/passwd запуском команды:

```
cat > /etc/passwd << "EOF"  
root:x:0:root:/root:/bin/bash  
EOF
```

Пароль пользователя *root* (символ "x" заменяет его здесь) будет определен позднее.

Создадим файл /etc/group запуском следующей команды:

```
cat > /etc/group << "EOF"  
root:x:0:  
bin:x:1:  
sys:x:2:  
kmem:x:3:  
tty:x:4:  
tape:x:5:  
daemon:x:6:  
floppy:x:7:  
disk:x:8:  
lp:x:9:  
dialout:x:10:  
audio:x:11:  
EOF
```

Созданные группы не являются частью какого-то стандарта -- эти группы используются скриптом MAKEDEV в следующей главе. Помимо группы "root", LSB (<http://www.linuxbase.org>) рекомендует устанавливать GID равный 1 для группы "bin". Все другие имена групп и их GID могут свободно выбираться пользователем, но некоторые пакеты зависят от имени группы, хоть и не зависят от номера GID. Наконец, мы перезайдем в chroot. Имя пользователя и имя группы начнут работать правильно после создания файлов /etc/passwd и /etc/group потому, что мы установили полную Glibc в Главе 5. Это должно убрать вывод сообщения "I have no name!".

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

Замечание по использованию параметра +h. Это скажет bash не использовать внутренний кеш путей. Без этого bash будет запоминать пути к исполненным бинарникам. Поскольку мы собираемся использовать вновь скомпилированные пакеты по мере их установки, мы должны отключить эту функцию в этой главе.

Создание устройств (Makedev-1.7)

Ожидаемое время сборки: 0.1 SBU
Ожидаемое место на диске: 50 KB

Описание MAKEDEV

Скрипт MAKEDEV создает статические записи для устройств, которые обычно расположены в директории /dev. Детальную информацию об устройствах вы можете получить в файле Documentation/devices.txt в директории с исходными кодами ядра.

Устанавливаемый скрипт: MAKEDEV

Зависимости установки MAKEDEV

Make зависит от: Bash, Coreutils.

Создание устройств

Заметьте, что распаковка файла MAKEDEV-1.7.bz2 не создаст директорий и вам не надо будет переходить в нее с помощью cd, так как этот файл содержит только скрипт.

Установим скрипт MAKEDEV:

```
bzcat MAKEDEV-1.7.bz2 > /dev/MAKEDEV  
chmod 754 /dev/MAKEDEV
```

Запустим скрипт для создания файлов устройств:

```
cd /dev  
./MAKEDEV -v generic-pnpty
```

Описание аргументов:

- **-v**: Это заставит скрипт запуститься в verbose режиме.
- **generic-pnpty**: Это заставит MAKEDEV создать необходимые файлы устройств за исключением файлов ptyXX и ttyXX. Они нам не нужны потому, что мы используем Unix98 PTY через файловую систему devpts.

Если скрипт не выполниться из-за пропуска какого-либо специального устройства zzz, то попробуйте запустить ./MAKEDEV -v zzz. Помимо этого, вы можете создать устройства через программу mknod.

Пожалуйста, загляните на man или info страницы этой программы для более подробной информации.

Помимо этого, если вы не монтировали файловую систему devpts в главе "Монтирование файловых систем proc и devpts", сейчас самое время сделать это. Если ваше ядро поддерживает файловую систему devfs, запустите следующую команду для монтирования devfs:

```
mount -t devfs devfs /dev
```

Эта команда смонтирует файловую систему devfs внутри новой статической структуре /dev. Это не создаст проблем, так как если уже были файлы устройств, то они перекрылись новой файловой системой devfs.

Если у вас что-либо не сработало, то вам надо запустить скрипт MAKEDEV для создания файлов ptyXX и ttyXX, которые в противном случае нам не нужны. Убедитесь, что вы находились в директории /dev и запустите ./MAKEDEV -v pty. Последнее создаст более 512 специальных файлов устройств, которые не понадобятся на когда мы окончательно загрузим законченную систему LFS.

Установка заголовков Linux-2.4.22

Ожидаемое время сборки: 0.1 SBU
Ожидаемое место на диске: 186 MB

Описание Linux

Ядро Linux - это основа любой Linux системы. Это то, что делает Linux собой. Когда компьютер включается и загружает систему Linux, самым первым загружается ядро. Ядро инициализирует аппаратные компоненты: последовательные и параллельные порты, звуковые карты, сетевые карты, контролеры IDE, контролеры SCSI и много чего еще. Собственно, ядро делает доступным аппаратные элементы cbcntvs и позволяет запускаться программам.

Устанавливаемые файлы: ядро и заголовки ядра

Зависимости установки Linux

Linux зависит от: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

Установка заголовков ядра

На данном этапе мы не будем компилировать ядро -- мы сделаем это после установки всех пакетов. Однако, в связи с тем, что некоторые пакеты требуют заголовочных файлов ядра, мы распакуем ядро, настроим его и скопируем заголовки туда, где их будут искать пакеты при компиляции.

Важно отметить, что файлы в директории с исходниками ядра не являются файлами с владельцем *root*.

Когда вы распакуете пакет как пользователь *root* (как это происходит здесь, в среде chroot), конечные файлы

будут иметь ID владельца и группы как на компьютере, где создавался пакет. Это обычно не создает проблемы для любых других пакетов потому, что вы обычно удаляете директорию с исходниками после установки. Но директории с исходниками ядра Linux будут еще нужны очень долго, и может получиться, что какой-либо пользователь вашей системы получит доступ к исходникам ядра.

Чтобы исправить это, запустите команду chown -R 0:0 в директории linux-2.4.22 чтобы убедиться, что владельцем всех файлов оттуда будет *root*.

Подготовим заголовки к установке:

```
make mrproper
```

Это гарантирует нам, что дерево с исходниками ядра будет абсолютно "чистым". Команда разработчиков ядра рекомендует выполнять эту команду перед *каждой* компиляцией ядра. Вы не можете быть абсолютно уверены в чистоте дерева исходников ядра даже после распаковки.

Создадим файл include/linux/version.h:

```
make include/linux/version.h
```

Создадим платформо-зависимую ссылку include/asm:

```
make symlinks
```

Установим платформо-зависимые файлы заголовков:

```
cp -HR include/asm /usr/include
```

```
cp -R include/asm-generic /usr/include
```

Установим кросс-платформенные файлы заголовков:

```
cp -R include/linux /usr/include
```

Некоторые из заголовков ядра используют файл заголовков autoconf.h. Поскольку мы пока не сконфигурировали ядро, нам надо создать этот файл для того, чтобы компиляция следующих пакетов не закончилась ошибкой. Создадим пустой файл autoconf.h:

```
touch /usr/include/linux/autoconf.h
```

Почему мы копируем заголовки ядра, а не создаем для них символические ссылки?

Раньше было принято создавать символические ссылки директорий /usr/include/{linux,asm} на /usr/src/linux/include/{linux,asm}. Однако, это была плохая идея, как объясняет Линус Торвальдс в списке рассылки ядра Linux (Linux Kernel Mailing List, отрывок):

Тем, кто компилирует новые ядра, настоятельно рекомендую:

- не создавать символьических ссылок (кроме той, которую создает само ядро, "linux/include/asm", символьическая ссылка, которая используется при внутренней компиляции ядра.)

Да, именно так делаю я. В моем каталоге /usr/src/linux есть заголовки старого ядра 2.2.13, несмотря на то, что я не использовал ядро 2.2.13 уже очень давно. Однако именно с этими заголовками компилировался пакет glibc, соответственно именно эти заголовки соответствуют файлам объектов библиотеки.

Также именно такое положение вещей являлось рекомендуемым на протяжении последних пяти лет. Я не могу понять, почему вся эта заморочка с символьическими ссылками все еще существует, как зомби. Практически во всех дистрибутивах есть эта неработающая символьическая ссылка, и люди до сих пор помнят, что исходники ядра Linux должны находятся в "/usr/src/linux", несмотря на то, что это уже давно не так.

Наиболее важная часть сообщения Линуса заключается в том, что файлы заголовков должны быть именно теми, с которыми компилировался пакет glibc. Это те заголовки, которые следует использовать в будущем при компиляции других пакетов, т.к. именно они соответствуют файлам библиотек объектного кода.

Копируя заголовки, мы удостоверяемся, что они останутся доступными, если позже мы решим обновить ядро.

Заметьте, что не страшно если исходники ядра находятся в /usr/src/linux, главное, чтобы у вас не было ссылок на /usr/include/{linux,asm}.

Установка Man-pages-1.60

Ожидаемое время сборки: 0.1 SBU

Ожидаемое место на диске: 15 MB

Описание Man-pages

Пакет Man-pages содержит более 1200 страниц документации. Это детальная информация по функциям С и С++, описание наиболее важных файлов устройств и документация, которая обычно опущена в других пакетах.

Устанавливаемые файлы: страницы документации

Зависимости установки Man-pages

Man зависит от: Bash, Coreutils, Make.

Установка Man-pages

Установим Man-pages запуском:

```
make install
```

Установка Glibc-2.3.2

Ожидаемое время сборки:	12.3 SBU
Ожидаемое место на диске:	784 MB

Описание Glibc

Glibc является библиотекой С, которая обеспечивает системные вызовы и основные функции, такие как open, malloc, printf и другие. Библиотека С используется всеми динамически скомпонованными программами. *Устанавливаемые программы:* catchsegv, gencat, getconf, getent, glibcbug, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, mtrace, nscd, nsd_nischeck, pprofledump, pt_chown, rpcgen, rpcinfo, sln, sprof, tzselect, xtrace, zdump и zic

Устанавливаемые библиотеки: ld.so, libBrokenLocale.[a,so], libSegFault.so, libanl.[a,so], libbsd-compat.a, libc.[a,so], libc_nonshared.a, libcrypt.[a,so], libdl.[a,so], libg.a, libieee.a, libm.[a,so], libmcheck.a, libmemusage.so, libnsl.a, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libnss_nis.so, libnss_nisplus.so, libpcprofile.so, libpthread.[a,so], libresolv.[a,so], librpcsvc.a, librt.[a,so], libthread_db.so и libutil.[a,so]

Зависимости установки Glibc

Glibc зависит от: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Make, Perl, Sed, Texinfo.

Установка Glibc

Система сборки Glibc является весьма самодостаточной и должна установится корректно, если ваши specs файл и компоновщик находятся в /tools. Мы не можем установить spec и компоновщик перед установкой Glibc потому, что тест автоконфигурации Glibc даст фиктивные результаты и от этого зависит чистота сборки.

Замечание: Тестирование Glibc в этой главе является *критичным*. Ни в коем случае нельзя его пропускать.

Перед началом сборки Glibc не забудьте снова распаковать Glibc-linuxthreads в директории glibc-2.3.2 и отключить переменные окружения, отвечающие за флаги оптимизации сборки.

На стадии установки Glibc выдает ошибку при отсутствии файла /etc/ld.so.conf. Исправим это командой:
touch /etc/ld.so.conf

Применим ранее использовавшийся патч:

```
patch -Np1 -i ./glibc-2.3.2-sscanf-1.patch
```

Документация по Glibc рекомендует собирать Glibc вне директории с исходниками в отдельной директории для сборки:

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

Теперь подготовим Glibc к компиляции:

```
./glibc-2.3.2/configure --prefix=/usr \  
--disable-profile --enable-add-ons \  
--libexecdir=/usr/bin --with-headers=/usr/include
```

Описание параметров конфигурации:

- `--libexecdir=/usr/bin`: Это надо для установки программы `pt_chown` в директорию `/usr/bin`.
- `--with-headers=/usr/include`: Это позволит убедиться, что заголовки ядра из `/usr/include` будут использованы при сборке. Если вы не укажете этот параметр, то будут использованы заголовки из `/tools/include` что не идеально (несмотря на их идентичность). Использование этого переключателя позволит также убедиться что вы не забыли установить заголовки ядра в `/usr/include`.

Скомпилируем пакет:

```
make
```

Протестируем результаты:

```
make check
```

Тестирование, описаное в части [Установка Glibc-2.3.2 Главы 5](#), будет очень уместно здесь. Убедитесь перед продолжением установки, что все в порядке.

И установим пакет:

```
make install
```

Локали позволяют вашей системе работать на разных языках. Их установка не обеспечивается предыдущей командой, вам надо зделать это с помощью:

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

Альтернативой запуску предыдущей команды будет установка только необходимых вам локалей. Это обеспечивается запуском команды `localeddef`. Информацию о ней можно получить из файла `INSTALL file` из директории с `glibc-2.3.2`. Тем не менее, есть ряд локалей, которые необходимы для успешного прохождения некоторых тестов других пакетов. Следующие инструкции установят минимально необходимый набор локалей:

```
mkdir -p /usr/lib/locale
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
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 es_MX -f ISO-8859-1 es_MX
localedef -i fr_FR -f ISO-8859-1 fr_FR
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i ja_JP -f EUC-JP ja_JP
```

Наконец, соберем таран страницы для `linuxthreads`:

```
make -C ./glibc-2.3.2/linuxthreads/man
```

И установим их:

```
make -C ./glibc-2.3.2/linuxthreads/man install
```

Конфигурирование Glibc

Нам нужно создать файл `/etc/nsswitch.conf` потому, что по умолчанию Glibc не создает этот файл, а без него Glibc не работает с сетью. Также это надо для работы с вашей временной зоной.

Создадим новый файл `/etc/nsswitch.conf` запуском следующих команд:

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

```
# Begin /etc/nsswitch.conf
```

```
passwd: files
group: files
shadow: files
```

```
publickey: files
```

```
hosts: files dns
networks: files
```

```
protocols: db files
services: db files
ethers: db files
rpc: db files
```

```
netgroup: db files
```

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

Для определения вашей временной зоны, запустите скрипт:

```
tzselect
```

Когда вы ответите на некоторые вопросы о вашим местонахождении, скрипт выдаст вам вашу временную зону. Что-то наподобие `EST5EDT` или `Canada/Eastern`. Создадим файл `/etc/localtime` запуском:

```
cp --remove-destination /usr/share/zoneinfo/Canada/Eastern /etc/locatime
```

Описание параметров:

- `--remove-destination`: Это необходимо для удаления существующей ссылки. Мы используем копию файла вместо ссылки на случай, если `/usr` находится на другом разделе. Также это нужно, например, когда вы загружаетесь в режиме `single user` (одного пользователя).

Само собой, вам надо заменить *Canada/Eastern* на вашу временную зону, которую мы определили с помощью скрипта `tzselect`.

Настройка динамического загрузчика

По умолчанию, динамический загрузчик (`/lib/ld-linux.so.2`) ищет динамические библиотеки, необходимые для программ, в `/lib` и `/usr/lib`. Таким образом, если есть директории с библиотеками, отличные от `/lib` и `/usr/lib`, вам надо указать их в файле `/etc/ld.so.conf`. Две наиболее распространенные директории с дополнительными библиотеками находятся в `/usr/local/lib` и `/opt/lib`, и мы добавим их в путь поиска динамического компоновщика. Создадим новый файл `/etc/ld.so.conf` запуском команды:

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

```
/usr/local/lib
/opt/lib
```

```
# End /etc/ld.so.conf
EOF
```

Переустановка средств

Теперь, когда мы установили новую библиотку С, пришло время переустановить наши средства. Это надо для того, чтобы вновь скомпилированные программы использовали именно новую библиотеку С. Если говорить проще, то то, что мы сейчас сделаем, это реверсия того, что мы делали на этапе "интеграции" в прошлой главе.

Первым делом мы отрегулируем компоновщик. Для этого мы вернемся к директориям с исходниками и сборкой из второго шага установки Binutils. Установим отрегулированный компоновщик запуском следующей команды из директории `binutils-build`:

```
make -C ld INSTALL=/tools/bin/install install
```

Замечание: Если вы пропустили предупреждение о нежелательности удаления директорий с исходниками и сборкой Binutils из второго шага их установки в Главе 5 или по другим причинам удалили их или у вас нет доступа к ним, не беспокойтесь, не все потеряно.

Просто проигнорируйте вышеупомянутую команду. В результате следующий пакет, Binutils, будет скомпонован с использованием библиотек Glibc из `/tools` вместо `/usr`. Это не идеально, но наше тестирование показывает что в обоих случаях бинарники Binutils будут идентичными.

С этого момента все компилируемые программы будут собираться *только* с использованием библиотек из `/usr/lib` и `/lib`. Параметр `INSTALL=/tools/bin/install` необходим потому, что `Makefile` созданный на втором шаге содержит ссылки на `/usr/bin/install`, который пока не установлен. Некоторые дистрибутивы содержат ссылку `ginstall` которая имеет первенство в `Makefile` и это может создать здесь проблему. Вышеуказанная команда также решает эту проблему.

Теперь вы можете удалить обе директории Binutils.

Теперь нам необходимо исправить точки в `spec` файлах GCC, которые указывают на динамический компоновщик, так, чтобы они указывали на новый компоновщик. Самым простым будет следующее:

```
SPECFILE=/tools/lib/gcc-lib/*/*/specs &&
sed -e 's@/@/tools/lib/ld-linux.so.2@/lib/ld-linux.so.2@g' \
$SPECFILE > newspecfile &&
mv -f newspecfile $SPECFILE &&
unset SPECFILE
```

Желательно просто скопировать и вставить эту команду. После этого также неплохо бы было проверить `spec` файлы, чтобы убедиться в сделанных изменениях.

Важно: Если вы работаете на платформах, на которых имя динамического компоновщика отличается от ld-linux.so.2, вы должны заменить ld-linux.so.2 на имя компоновщика для вашей платформы в вишеуказанной команде. Вернитесь к [части Технические моменты в Главе 5](#), если у вас возникли вопросы.

Внимание

На этом месте необходимо остановиться и убедиться, что основные функции (компиляция и компоновка) новых средств работают корректно. Для этого есть простой тест:

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

Если все в порядке, то не будет ошибок и на выводе вы увидите:

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

Если эта надпись вообще не появилась или появилась другая, то чтото сильно не так. Вам надо исследовать и повторить все пройденные шаги, чтобы найти в чем проблема и устранить ее. Точки для возврата после этого места уже не будет. Как правило, что-то не так бывает с вышеописанной правкой specs-фала. Убедитесь, что /lib содержит префикс вашего динамического компоновщика. Само собой, если вы работаете на платформе с названием динамического компоновщика, отличным от ld-linux.so.2, вывод будет несколько иным.

Если все прошло нормально, удалим тестовые файлы:

```
rm dummy.c a.out
```

Установка Binutils-2.14

Ожидаемое время сборки: 1.4 SBU
Ожидаемое место на диске: 167 MB

Описание Binutils

Binutils является набором средств разработки, содержащим компоновщик, ассемблер и другие средства для работы с объектными файлами и архивами.

Устанавливаемые программы: addr2line, ar, as, c++filt, gprof, ld, nm, objcopy, objdump, ranlib, readelf, size, strings и strip

Устанавливаемые библиотеки: libiberty.a, libbfd.[a,so] and libopcodes.[a,so]

Зависимости установки Binutils

Binutils зависит от: Bash, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed, Texinfo.

Установка Binutils

Сейчас необходимо проверить правильность работы pseudo терминалов (PTY) в среде chroot. Мы проверим что все в порядке простым тестом:

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

Если вы получили сообщение вида:

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

Ваша среда chroot не может оперировать PTY. В этом случае вам не стоит запускать тестирование для Binutils и GCC до того, как вы исправите это. Пожалуйста, вернитесь к [части Монтирование файловых систем proc и devpts](#) и [части Создание устройств \(Makedev-1.7\)](#) и выполните необходимые действия по устранению проблемы.

Замечание: Тестирование Binutils здесь является *критичным*. Ни в коем случае не пропускайте его.

Этот пакет известен своим нестабильным поведением при компиляции с измененными опциями оптимизации (включая опции -march и -mcpu). Binutils рекомендуется компилировать с настройками по умолчанию. Следовательно, если вы задали переменные такие как CFLAGS или CXXFLAGS, изменяющие

уровень оптимизации по умолчанию, рекомендуется убрать их при сборке пакета Binutils. Изменяя оптимизации для binutils, вы действуете на свой страх и риск.

Документация по Binutils рекомендует собирать Binutils вне директории с исходниками в отдельной директории для сборки:

```
mkdir ..binutils-build  
cd ..binutils-build
```

Теперь подготовим Binutils к компиляции:

```
../binutils-2.14/configure \  
--prefix=/usr --enable-shared
```

Скомпилируем пакет:

```
make tooldir=/usr
```

Обычно *директория со средствами* (директория, где располагаются исполняемые файлы) устанавливается в `$(exec_prefix)/$(target_alias)`, куда они и располагаются, например, `/usr/i686-pc-linux-gnu`. Когда мы собираем средства только для своей системы, мы не нуждаемся в этом, нам достаточно установить все просто в директорию `/usr`. Такая установка нужна только в случае использования cross-компиляции (к примеру, компилируем пакет на машине Intel, но нам нужен код для работы на платформе PowerPC).

Проверим результаты:

```
make check
```

Тестирование, описаное в [части Установка Binutils-2.14 - Шаг 2 Главы 5](#) является весьма уместным здесь.

Убедитесь, что все впорядке перед продолжением установки.

Установим пакет:

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

Установим заголовки *libiberty*, которые нужны для некоторых пакетов:

```
cp ../binutils-2.14/include/libiberty.h /usr/include
```

Установка GCC-3.3.1

Ожидаемое время сборки: 11.7 SBU
Ожидаемое место на диске: 294 MB

Описание GCC

Пакет GCC содержит коллекцию компиляторов GNU, включая компиляторы C и C++.

Устанавливаемые программы: c++, cc (ссылка на gcc), cc1, cc1plus, collect2, cpp, g++, gcc, gccbug, и gcov

Устанавливаемые библиотеки: libgcc.a, libgcc_eh.a, libgcc_s.so, libstdc++.[a,so] и libsupc++.a

Зависимости установки GCC

GCC зависит от: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed, Texinfo.

Установка GCC

Замечание: Тестирование GCC в этой главе является *критичным*. Ни в коем случае не пропускайте его.

Этот пакет известен своим нестабильным поведением при компиляции с измененными опциями оптимизации (включая опции `-march` и `-mtcpu`). GCC рекомендуется компилировать с настройками по умолчанию. Следовательно, если вы задали переменные такие как `CFLAGS` или `CXXFLAGS`, изменяющие уровень оптимизации по умолчанию, рекомендуется убрать их при сборке пакета GCC. Изменяя оптимизации для GCC, вы действуете на свой страх и риск.

Сейчас мы собираемся установить компиляторы языков C и C++, таким образом вам надо распаковать архивы GCC-core и GCC-g++ -- они распакуются в одну и ту же директорию. Вы можете также распаковать пакет GCC-testsuite. Полный пакет GCC содержит намного больше компиляторов. Инструкции по их установке вы можете найти на <http://www.linuxfromscratch.org/blfs/view/stable/general/gcc.html>.

```
patch -Np1 -i ../gcc-3.3.1-no_fixincludes-2.patch  
patch -Np1 -i ../gcc-3.3.1-suppress-libiberty.patch
```

Второй патч отключает установку *libiberty* из GCC, которые мы установили вместе с binutils. Будьте внимательны, *не применяйте* патч GCC specs из Главы 5 здесь.

Документация GCC рекомендует собирать GCC вне директории с исходниками, в отдельной директории для сборки:

```
mkdir ..gcc-build  
cd ..gcc-build
```

Теперь подготовим GCC к компиляции:

```
./gcc-3.3.1/configure --prefix=/usr \
--enable-shared --enable-threads=posix \
--enable-_cxa_atexit --enable-clocale-gnu \
--enable-languages=c,c++
```

Скомпилируем пакет:

```
make
```

Протестируем результаты, но не будем останавливаться при ошибках (вы должны просто запомнить их):
make -k check

Тестирование из [части Установка GCC-3.3.1 - Шаг 2 Главы 5](#) является весьма желательным здесь.

Убедитесь, что все в порядке перед продолжением установки.

И установим пакет:

```
make install
```

Некоторые пакеты рассчитывают, что C PreProcessor будет установлен в директории /lib. Для удовлетворения нужд этих пакетов, создадим ссылку:

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

Многие пакеты используют имя cc для вызова компилятора С. Для этих пакетов мы тоже создадим ссылку:

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

Замечание: В этом месте рекомендуется повторить тест из [части Переустановка средств](#).

Если тест не прошел, то, скорее всего, вы применили патч GCC Specs из Главы 5.