debian-from-scratch

Руководство по обучению пользователей Linux From Scratch созданию системы Debian.

Почему Debian с нуля?

Оригинальное руководство Linux from Scratch намеренно неопределенно в отношении того, какую технику следует использовать для управления зависимостями программного обеспечения. Предложения, которые оно дает, хотя, несомненно, являются интересными упражнениями по управлению пакетами, не обязательно являются сложными ответами для системного администратора, который намерен эффективно управлять своим временем.

Недостатком компиляции всего для создания полноценной системы является время. После того, как кто-то впервые построит систему LFS, он/она склонны понимать, что управление зависимостями может быть трудной задачей, мягко говоря. Прохождение невыносимых упражнений по поиску десятков, а возможно, и сотен пакетов, сопоставление зависимостей, настройка и установка этих зависимостей в правильном порядке, только для того, чтобы установить одну часть программного обеспечения, не является жизнеспособной альтернативой системному администратору, который ценит свое время.

Ответ на эту проблему, очевидно, заключается в использовании менеджера пакетов. Существует множество доступных менеджеров пакетов, наиболее популярными из которых являются менеджеры пакетов на основе Debian (dpkg и apt) и менеджеры пакетов на основе Red Hat (rpm и yum).

Это руководство научит вас собирать систему с использованием набора инструментов управления пакетами Debian, используя временную системную среду, созданную в Linux From Scratch.

Цель этого проекта

Я решил сделать это руководство, потому что я видел ужасно старые руководства в Интернете, обучающие других, как заставить dpkg и apt работать на их собственном Linux, и люди спрашивали на различных форумах, как установить dpkg и apt, но не получали необходимой помощи. Эти руководства устарели и больше не содержат актуальной информации, что я намерен исправить здесь, в этом руководстве.

Целью данного проекта является создание ресурса сообщества, призванного помочь тем, кто заинтересован в создании собственной системы с нуля, в полной мере используя возможности пакета управления пакетами Debian, dpkg и apt, для решения проблем установки и управления зависимостями пакетов.

Как пользоваться этим руководством?

Это руководство предназначено для использования после завершения всех инструкций до конца Главы 5 Linux From Scratch book, version 7.9. Сначала следует следовать инструкциям оригинальной книги LFS и построить временную систему, которая создается в Главе 5 LFS. Требуется иметь полностью функциональную временную систему, которая является результатом Главы 5.

После завершения предварительной подготовки следует обратиться к данному руководству и следовать ему шаг за шагом.

Как и в оригинальном руководстве LFS, при работе с пакетами, которые нужно скомпилировать, каждый раздел уже предполагает, что вы извлекли исходный код и изменили свой основной каталог на основную папку извлеченного контента. Однако при работе с файлами .deb такое извлечение не требуется. Нужно только следовать инструкциям, имея файл .deb в вашем текущем каталоге.

После завершения предварительной подготовки следует обратиться к данному руководству и следовать ему шаг за шагом.

Как и в оригинальном руководстве LFS, при работе с пакетами, которые нужно скомпилировать, каждый раздел уже предполагает, что вы извлекли исходный код и изменили свой основной каталог на основную папку извлеченного контента. Однако при работе с файлами .deb такое извлечение не требуется. Нужно только следовать инструкциям, имея файл .deb в вашем текущем каталоге..

Обзор нашего метода создания собственной системы Debian

В оригинальной книге Linux From Scratch мы создали кросс-цепочку инструментов, используя собственную цепочку инструментов нашей системы. Затем мы использовали эту кросс-цепочку инструментов для создания собственной цепочки инструментов, которая в итоге стала временной системной средой `/tools`. Это было целью Главы 5. Затем мы использовали эту временную систему для создания нашей окончательной системы, что было целью Главы 6.

В Debian From Scratch мы отталкиваемся от конца Главы 5. Вместо того чтобы использовать набор инструментов и другие утилиты, установленные в /toolsдля компиляции каждой отдельной части окончательной системы, мы вместо этого используем этот набор инструментов для компиляции и установки менеджера пакетов Debian, dpkg, в качестве первой части нашей окончательной системы.

Затем мы делаем некоторый взлом зависимостей, чтобы удовлетворить все оставшиеся зависимости, необходимые для установки apt. Это позволяет нам полагаться на apt для подавляющего большинства задач, связанных с установкой программного обеспечения на нашу новую систему, и позволяет нам избегать утомительного упражнения, которое представляет собой ручное управление зависимостями.

Затем мы используем apt для установки всех базовых пакетов, необходимых для корректной

работы системы, в правильном порядке, чтобы предотвратить возникновение проблем и поломку пакетов.

Получение всех необходимых пакетов

Давайте начнем с использования нашей хост-системы для загрузки необходимых нам пакетов и размещения их где-нибудь внутри нашего раздела $LFS.

исходные файлы

The only source file you will need to download is the source for dpkg:

dpkg

.deb files

Вам нужно будет загрузить следующие файлы .deb и поместить их все в один каталог внутри вашего $LFS раздела. Только эти файлы .deb должны занимать этот каталог. Они необходимы для установки всей цепочки зависимостей `apt`. Откройте ссылку и вручную загрузите файл .deb, соответствующий архитектуре вашей системы LFS:

apt

debian-archive-keyring

dpkg

gcc-4.9-base

gnupg

gpgv

libacl1

libapt-pkg4.12

libattr1

libbz2-1.0 [libc6](https://packages.debian.org/jessie/libc6)

[libgcc1](https://packages.debian.org/jessie/libgcc1)

[liblzma5](https://packages.debian.org/jessie/liblzma5)

[libpcre3](https://packages.debian.org/jessie/libpcre3)

[libreadline6](https://packages.debian.org/jessie/libreadline6)

[libselinux1](https://packages.debian.org/jessie/libselinux1)

[libstdc++6](https://packages.debian.org/jessie/libstdc++6)

[libtinfo5](https://packages.debian.org/jessie/libtinfo5)

[libusb-0.1-4](https://packages.debian.org/jessie/libusb-0.1-4)

[multiarch-support](https://packages.debian.org/jessie/multiarch-support)

[readline-common](https://packages.debian.org/jessie/readline-common)

[tar](https://packages.debian.org/jessie/tar)

[zlib1g](https://packages.debian.org/jessie/zlib1g)

Создание системы Debian From Scratch

Все следующие команды необходимо выполнять как пользователь `root`, то есть стать rootпользователем вашей хост-системы:

`su`

Подготовка точек монтирования файловой системы виртуального ядра

Сначала мы создаем каталоги, которые должны содержать виртуальные файловые системы ядра. Это файловые системы, которые находятся только в памяти и создаются динамически каждый раз при загрузке ядра.

Каждый тип имеет свое назначение. `devpts` Содержит файлы устройств для каждого псевдотерминала в вашей системе. `proc` Содержит информацию о каждом отдельном процессе. `sysfs` Содержит информацию о драйверах и устройствах. Это `tmpfs` свободно используемое пространство, которое программы могут использовать для хранения информации в памяти.

Поскольку мы еще не построили наше ядро, мы вынуждены использовать те, которые существуют в нашей хостовой системе, смонтировав их в соответствующих местах в нашей целевой системе. Когда наша система будет полностью построена, новое ядро автоматически смонтирует эти файловые системы в соответствующих местах.

```
mkdir -pv $LFS/{dev,proc,sys,run}
mknod -m 600 $LFS/dev/console c 5 1
mknod -m 666 $LFS/dev/null c 1 3
mount -v --bind /dev $LFS/dev
mount -vt devpts devpts $LFS/dev/pts -o gid=5,mode=620
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
mount -vt tmpfs tmpfs $LFS/run

if [ -h $LFS/dev/shm ]; then
```

```
  mkdir -pv $LFS/$(readlink $LFS/dev/shm)
fi
```

Вход в нашу среду chroot

Теперь мы должны, как `root` пользователь нашей хостовой системы, войти в нашу базовую среду, изменив наш корневой каталог на корневой каталог конечной системы, и использовать временную среду, которую мы ранее построили, чтобы построить нашу конечную систему. Используйте следующую команду после того, как вы стали `root` на своем хосте:

```
chroot "$LFS" /tools/bin/env -i \
HOME=/root                  \
TERM="$TERM"                \
PS1='\[\033[01m\][
\[\033[01;34m\]\u@\h\[\033[00m\]\[\033[01m\]]\[\033[01;32m\]\w\[\033[00m\]\n
\[\033[01;34m\]$\[\033[00m\]> ' \
PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin:/tools/sbin \
/tools/bin/bash --login +h
```

Установка dpkg

With our `/tools` environment completely set up, we are ready to directly compile and install `dpkg` into our target environment. Replace the `build` variable with the appropriate architecture if it isn't 64-bit (which I am assuming that it is):

```
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var --
build=x86_64-unknown-linux-gnu
make
make install
```

Создание базы данных dpkg

Нам нужно создать dpkg базу данных , которая представляет собой просто текстовый файл, расположенный в /var/lib/dpkg/status. dpkg хранит всю информацию о пакете в этом файле, включая версию пакета, архитектуру, зависимости и т. д. В настоящее время он еще не существует. Без этого файла dpkg не будет работать правильно, поэтому важно создать его, прежде чем двигаться дальше.

```
touch /var/lib/dpkg/status
```

Создание временных ссылок, ссылающихся на /tools/bin/bash

Для того, чтобы скрипты до и после установки, которые находятся внутри стандартного файла .deb, работали, они должны иметь доступ к оболочке. Обычно они указывают, используя /bin/shили /bin/bash. Без доступа к оболочке в точном месте, указанном в скрипте, скрипт установки завершится ошибкой, что приведет к ошибке установки самого пакета.

Мы должны решить эту проблему, убедившись, что /binкаталог уже существует, а затем создав символические ссылки из этих двух мест на bash, который находится в нашей временной /toolscреде.

Когда мы дойдем до момента установки пакетов Debian «priority:essential», которые включают обе эти оболочки, эти символические ссылки будут перезаписаны собственными копиями этих двоичных файлов.

```
mkdir /bin
ln /tools/bin/bash /bin/bash
ln /tools/bin/bash /bin/sh
```

Установка apt

Прежде чем мы сможем установить `apt` и использовать его для автоматической установки большей части остального системного программного обеспечения, нам сначала необходимо установить его непосредственные зависимости в нашей целевой системе.

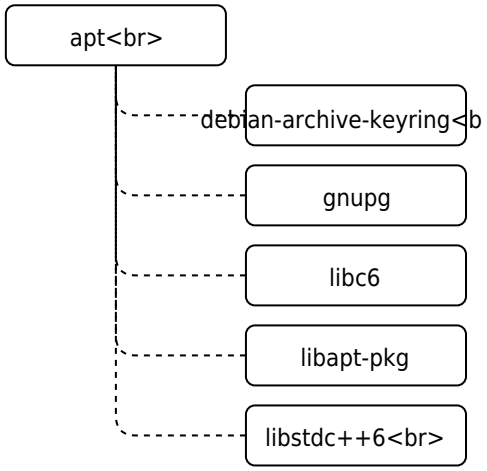


Рисунок 2. Дерево зависимостей apt, глубина в один уровень

Каждая из этих непосредственных зависимостей имеет свой собственный набор зависимостей для выполнения. Мы начнем с завершения дерева зависимостей для `debian-archive-keyring`. В отличие от процесса компиляции, необходимого для установки `dpkg`, процесс, который мы теперь используем для установки программного обеспечения, заключается в установке . `deb` файлов с использованием `dpkg`.

Установка debian-archive-keyring

Итак, мы начнем выполнять `dpkg` зависимости , сначала установив `debian-archive-keyring`. Однако здесь есть проблема - у нас есть циклическая зависимость, как показано ниже.

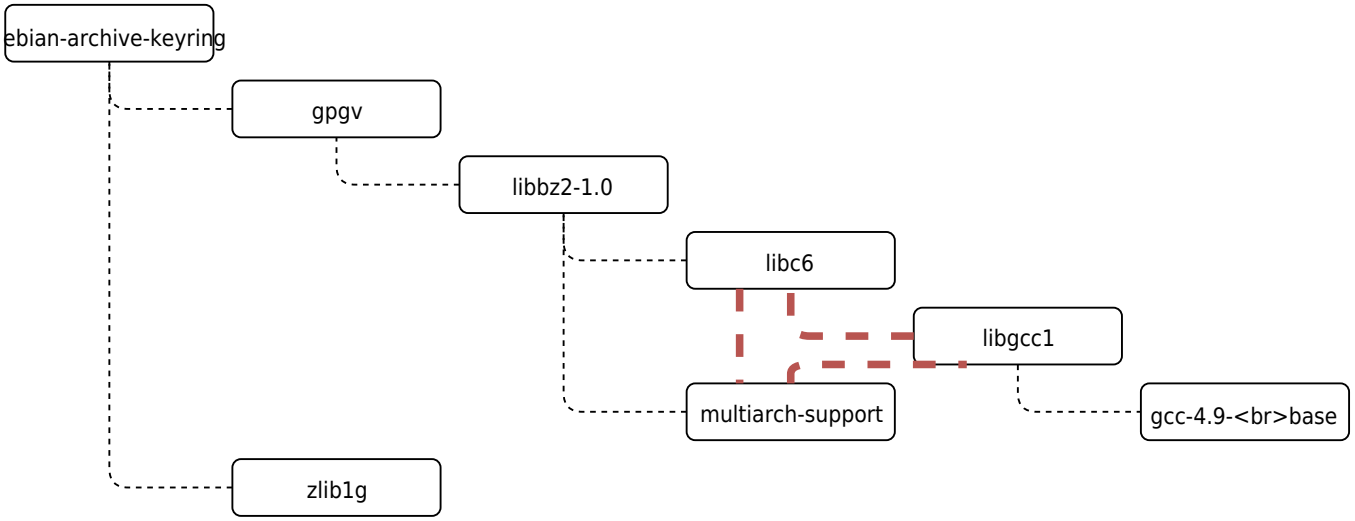


Рисунок 3. Дерево зависимостей Debian-Archive-Keyring, циклическая зависимость выделена красным

`libgcc1` зависит от `multiarch-support`, который зависит от `libc6`, который зависит от `libgcc1`. Это серьезная проблема, поскольку ни один из этих пакетов не будет полностью установлен без других.

Чтобы решить эту, казалось бы, неразрешимую проблему, нам придется немного отступить от правил.

Для начала нам необходимо установить `gcc-4.9-base`:

```
dpkg -i (location of gcc-4.9-base)
```

Затем нам нужно будет сначала установить один пакет частично:

```
dpkg - i (LOCATION_OF_libgcc1)
```

ПРИМЕЧАНИЕ: Здесь вы получите сообщение об ошибке, содержащее ошибки, очень похожие на следующие:

```
Unpacking libc6:amd64 (2.28-10) over (2.28-10) ...
dpkg: dependency problems prevent configuration of libc6:amd64:
 libc6:amd64 depends on libgcc1; however:
  Package libgcc1:amd64 is not configured yet.

dpkg: error processing package libc6:amd64 (--install):
 dependency problems - leaving unconfigured
Errors were encountered while processing:
 libc6:amd64
```

Это нормально - на этом этапе вы не можете полностью установить ни один из этих пакетов. Вы можете установить их только частично, что мы исправим позже.

После установки пакета настройте базу данных, чтобы убедиться, что она полностью установлена:

```
sed -ir 's/not-installed/installed/' /var/lib/dpkg/status
```

Теперь установите два других пакета по порядку:

```
dpkg -i (location_of_multiarch)
```

```
dpkg -i (location_of_libc6)
```

И переустановите libgcc1, чтобы скрыть наш маленький некрасивый хак и завершить полную установку каждого пакета:

```
dpkg -i –reinstall (location of libgcc1)
```

Установка остальных зависимостей apt

На этом этапе я предполагаю, что -все- файлы .deb, которые вам нужно установить, были помещены в один каталог. Дважды проверьте, что у вас есть все эти файлы. cdв этот каталог прямо сейчас.

Правда в том, что для всех оставшихся зависимостей дерево зависимостей слишком сложное, чтобы я мог его составить и предоставить вам подробную последовательность команд установки для выполнения того, что в противном случае было бы простой операцией.

Независимо от фактического дерева зависимостей, есть быстрый и грязный способ установить остальную часть всего дерева зависимостей apt. Просто выполните следующую команду и повторяйте ее столько раз, сколько вам нужно, пока dpkg не перестанет жаловаться:

```
dpkg -i *
```

Здесь происходит следующее: dpkg попытается установить все программные пакеты в каталоге. Это неизбежно приведет к сбою, но не волнуйтесь. Просто повторяйте команду, пока все не будет установлено.

Как это работает

Что происходит, когда мы запускаем команду выше, так это то, что dpkg пытается установить каждый пакет, фактически не принимая во внимание дерево зависимостей. Таким образом, с каждым выполнением команды выше будет выполняться еще один уровень дерева зависимостей, пока не будет успешно подтверждено, что все дерево было установлено. Как только вы больше не получаете никаких ошибок от dpkg, это означает, что все было установлено.

dpkg просто не так умен, как что-то вроде apt, которое автоматически создает карту зависимостей и устанавливает все необходимое программное обеспечение перед попыткой установки программного обеспечения, зависящего от указанного программного обеспечения.

Вы также можете заметить, что сам dpkg является частью файлов .deb, которые необходимо установить. Не волнуйтесь, это ничего не сломает. У нас уже есть dpkg, но мы просто устанавливаем официальный пакет для обновления его собственной базы данных, поскольку база данных никогда не содержала себя как часть списка установленных пакетов.

Дважды проверьте, все ли установлено

Чтобы быть абсолютно уверенным, вы можете выполнить следующую команду, которая подсчитывает, сколько пакетов dpkg считает установленными в своей базе данных:

`echo $`[1)]

Если возвращается значение 23, то можно с уверенностью предположить, что все установлено так, как задумано.

Создание файлов конфигурации сети

Прежде чем приступить к обновлению `apt's` кэша, нам необходимо определить как файлы конфигурации сети системы, так и список репозиториев программного обеспечения apt.

/etc/resolv.conf

`/etc/resolv.conf` это файл, необходимый для того, чтобы ваша система имела разрешение DNS. Без этого файла разрешение обычно не происходит, что затрудняет обновление aptкэша устанавливаемых пакетов через `/etc/apt/sources.list`.

```
cat > /etc/resolv.conf << "EOF"
nameserver 8.8.8.8
nameserver 8.8.4.4
EOF
```

/etc/apt/sources.list

sources.list — это файл, который `apt` используется для связи с репозиториями, содержащими ваше программное обеспечение. Вы хотите использовать репозитории из одного и только одного дистрибутива как можно чаще, в противном случае вы рискуете нарушить четкую цепочку зависимостей вашего Debian и создать безумный беспорядок в вашей системе.

```
cat > /etc/apt/sources.list << "EOF"
# Debian Jessie main repos
deb http://httpredir.debian.org/debian/ jessie main
deb-src http://httpredir.debian.org/debian/ jessie main

#Debian Jessie security repos
deb http://security.debian.org/ jessie/updates main
deb-src http://security.debian.org/ jessie/updates main

# non-free plugins
deb http://http.debian.net/debian/ jessie non-free contrib main

# jessie-updates, previously known as 'volatile'
```

```
deb http://httpredir.debian.org/debian/ jessie-updates main
deb-src http://httpredir.debian.org/debian/ jessie-updates main
EOF
```

/etc/hosts

`/etc/hosts` это файл, используемый для хранения сопоставления IP-адресов с именами хостов. Этот файл обычно проверяется перед DNS-запросами, по крайней мере, он должен содержать ваши `ipv4` и `ipv6` петлевые адреса.

```
cat > /etc/hosts << "EOF"
127.0.0.1       localhost

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
EOF
```

/etc/hostname

`/etc/hostname` используется для хранения имени DNS вашего хоста. Отредактируйте это, если хотите.

```
cat > /etc/hostname << "EOF"
debianfromscratch
EOF
```

Обновление списков пакетов apt

Теперь мы готовы обновить наш список пакетов и в полной мере воспользоваться apt. Для этого мы обновляем нашу локальную связку ключей действительных подписей разработчиков Debian gnu pgp с помощью `apt-key update`, а затем обновляем с помощью `apt-get update`.

```
apt-key update
apt-get update
```

Создание баз данных пользователей и групп

Прежде чем устанавливать больше программного обеспечения, мы должны убедиться, что все наши механизмы паролей, групп и аутентификации на месте. Это связано с тем, что некоторые пакеты потребуют добавления нового пользователя или группы в систему в рамках процесса установки. Без этих базовых функций установка указанных пакетов не удастся.

debianutils

Мы устанавливаем debianutils, чтобы предоставить `tempfile` команду, необходимую одному из `base-passwd` установочных скриптов. Без этой команды установка `base-passwd` не будет выполнена.

apt-get install debianutils

base-passwd

Мы устанавливаем `base-passwd`, чтобы предоставить стандартный минимальный стандарт `/etc/passwd` и `/etc/group` файлы, которые одинаковы во всех системах Debian. Это происходит путем запуска `update-passwd` двоичного файла при его установке.

```
apt-get install base-passwd
```

Создание /etc/shadow и /etc/gshadow

Нам придется вручную создать `/etc/shadow` и `/etc/gshadow`, так как `passwd` пакет не сможет настроиться, если не сможет найти эти файлы:

```
touch /etc/shadow /etc/gshadow
```

авторизоваться

Затем мы устанавливаем `login` пакет, который дает нам возможность устанавливать новые сеансы в системе с `login`, повышение привилегий с `su`, файлы подключаемого модуля аутентификации Linux (PAM) для обоих указанных двоичных файлов, поддельную оболочку `/bin/nologin` и `/etc/login.defs` файл, который необходим для создания группы. В этот пакет включено больше функций, но эти наиболее упоминаемые.

```
apt-get install login
```

пароль

Затем мы устанавливаем `passwd` пакет, который предоставляет львиную долю утилит и файлов конфигурации, используемых для создания и управления информацией об учетных записях пользователей и групп.

```
apt-get install passwd
```

adduser

Нам также необходимо установить `adduser` пакет, поскольку он предоставляет нам

`/etc/adduser.conf` файл по умолчанию, который понадобится для правильной установки новых пользователей.

```
apt-get install adduser
```

Установка пароля root и записей shadowfile

После установки всех вышеупомянутых утилит и пакетов наша система теперь способна выполнять все функции управления учетными записями пользователей и групп.

На этом этапе нам следует запустить `passwd`, чтобы изменить пароль `root`.

```
passwd root
```

Затем нам следует запустить pwconvпреобразование наших записей `/etc/passwd` в теневые записи в `/etc/shadow`:

```
pwconv
```

Исправление терминала и добавление утилит чтения/редактирования

Наш терминал пока не обладает полной функциональностью, которую можно было бы ожидать от терминала, и стандартные утилиты терминала могут по-прежнему не работать должным образом на этом этапе. Давайте установим соответствующие библиотеки и создадим конфигурации, необходимые для того, чтобы сделать их, прежде чем мы установим

Создание файла /etc/inputrc

`/etc/inputrc` это глобальный файл конфигурации для используемой библиотеки `libreadline6`, который большинство оболочек используют для понимания того, как обрабатывать многие особые ситуации с клавиатурой, например, какое поведение должно быть по умолчанию при нажатии клавиш HOME и END. Без этого файла многие особые клавиши и двухклавишные комбинации, такие как ctrl+left, не будут работать.

Поскольку этот файл не создается по умолчанию при установке `libreadline6` библиотеки, нам придется создать его самостоятельно.

```
cat > /etc/inputrc << "EOF"
# /etc/inputrc - global inputrc for libreadline
# See readline(3readline) and `info rluserman' for more information.

# Be 8 bit clean.
set input-meta on
set output-meta on

# To allow the use of 8bit-characters like the german umlauts, uncomment
# the line below. However this makes the meta key not work as a meta key,
# which is annoying to those which don't need to type in 8-bit characters.
```

```
# set convert-meta off

# try to enable the application keypad when it is called.  Some systems
# need this to enable the arrow keys.
# set enable-keypad on

# see /usr/share/doc/bash/inputrc.arrows for other codes of arrow keys

# do not bell on tab-completion
# set bell-style none
# set bell-style visible

# some defaults / modifications for the emacs mode
$if mode=emacs

# allow the use of the Home/End keys
"\e[1~": beginning-of-line
"\e[4~": end-of-line

# allow the use of the Delete/Insert keys
"\e[3~": delete-char
"\e[2~": quoted-insert

# mappings for "page up" and "page down" to step to the beginning/end
# of the history
# "\e[5~": beginning-of-history
# "\e[6~": end-of-history

# alternate mappings for "page up" and "page down" to search the history
# "\e[5~": history-search-backward
# "\e[6~": history-search-forward

# mappings for Ctrl-left-arrow and Ctrl-right-arrow for word moving
"\e[1;5C": forward-word
"\e[1;5D": backward-word
"\e[5C": forward-word
"\e[5D": backward-word
"\e\e[C": forward-word
"\e\e[D": backward-word

$if term=rxvt
"\e[7~": beginning-of-line
"\e[8~": end-of-line
"\eOc": forward-word
"\eOd": backward-word
$endif

# for non RH/Debian xterm, can't hurt for RH/Debian xterm
# "\eOH": beginning-of-line
# "\eOF": end-of-line
```

```
# for freebsd console
# "\e[H": beginning-of-line
# "\e[F": end-of-line

$endif
EOF
```

Библиотеки и двоичные файлы ncurses

Большое количество утилит командной строки полагаются на библиотеку ncurses для предоставления текстового интерфейса для взаимодействия пользователя через терминал. К ним относятся простые утилиты, такие как `less` и nano. Без этой библиотеки эти утилиты не будут отображаться должным образом. Мы должны установить полный набор библиотеки ncurses, чтобы предотвратить возникновение указанных ошибок:

```
apt-get install ncurses-base ncurses-bin ncurses-doc
```

dialog

Dialog — это модуль `perl`, который некоторые скрипты пытаются использовать для предоставления текстового интерфейса, используемого во время настройки или установки. Вы могли заметить, что некоторые пакеты предупреждали вас, что эта утилита отсутствует во время установки. Давайте исправим это:

```
apt-get install dialog
```

less, vim and nano

Теперь, когда мы установили большую часть библиотек и утилит, необходимых для терминальных утилит, давайте установим некоторые из самых основных и часто используемых из них:

```
apt-get install less vim nano
```

Создание стандартной иерархии файловой системы в системе Debian

Давайте создадим стандартную структуру папок для системы Debian. Это можно легко сделать, установив `base-files`. Сначала нам нужно удалить `/var/mail` каталог, иначе он будет жаловаться, что он уже существует.

```
rm -rf /var/mail
apt-get install base-files
```

Создание системы документации для человека

В любой системе Linux обычно есть база данных, полная страниц руководства, доступ к которой осуществляется с помощью man команды. Большинство программ, которые мы уже установили, уже добавили свои файлы документации в нужное место. Все, что нам нужно сделать сейчас, это установить их man, чтобы воспользоваться ими, и любыми другими, которые будут добавлены со временем.

```
apt-get install man
```

Установка всех оставшихся необходимых пакетов

Мы установили большую часть пакетов, осталось только установить остальные пакеты, отмеченные приоритетом essential со стороны сопровождающих Debian. Некоторые из них абсолютно необходимы для управления системой, а некоторые вообще вряд ли будут использоваться.

Чтобы соответствовать стандарту Debian, нам необходимо установить все это:

```
apt-get install bash bsdutils coreutils dash diffutils e2fsprogs findutils grep gzip hostname libc-bin init mount perl-base sed sysvinit-utils tar util-linux
```

Ниже приведено краткое описание каждого установленного пакета:

Описание	Что это дает
bash:	Оболочка GNU Bourne-again, которая является стандартной оболочкой Linux.
bsdutils:	Предоставляет несколько двоичных файлов, наиболее важный reniceиз которых необходим для изменения приоритетов процессов и loggerиспользуется для взаимодействия с системным модулем syslog.
coreutils:	Самая важная группа двоичных файлов, необходимая для того, чтобы сделать любую оболочку полезной.
dash:	Оболочка Debian Almquist, которая представляет собой более быструю версию sh и предназначена в основном для использования в скриптах.
diffutils:	Предоставляет утилиты для сравнения содержимого файлов между собой.
e2fsprogs:	Предоставляет утилиты для работы с семейством файловых систем ext.
findutils:	Предоставляет утилиту find для поиска файлов.
grep:	Предоставляет утилиту grep, используемую для поиска строк в файлах или выводимых в нее данных.
gzip:	Предоставляет утилиту gzip, используемую для работы с файлами, использующими кодировку LZ77.
hostname:	Предоставляет набор утилит для управления именем хоста системы.
libc-bin:	Предоставляет реализацию GNU стандартной библиотеки C. Необходим для создания и использования программ.
init:	Предоставляет стандартный набор средств инициализации системы для Debian.
mount:	Предоставляет стандартные системные утилиты для монтирования и размонтирования файловых систем, включая файлы подкачки.
perl-base:	Предоставляет язык программирования Perl.

sed:	Предоставляет язык программирования sed, обычно используемый для редактирования текста.
sysvinit-utils:	Предоставляет утилиты типа system-v.
tar:	Предоставляет программу tar, используемую для хранения и извлечения файлов из архива на ленте.
util-linux:	Предоставляет множество жизненно важных системных утилит.

Установка ядра

Здесь у вас есть два варианта: вы можете либо установить последний образ ядра для архитектуры вашей системы, предоставленный проектом Debian, либо скомпилировать свой собственный.

Установка ядра Debian

Если вы хотите установить стандартное ядро Debian, вам нужно будет найти доступные образы, соответствующие вашей архитектуре, а затем установить соответствующий образ.

```
apt-cache search linux-image
apt-get install (selected image)
```

Компиляция и установка собственного ядра

При желании вы также можете скомпилировать свое собственное ядро.

Для этого вам понадобится tar-архив исходного кода ядра Linux, который у вас, вероятно, уже есть.

Вам также потребуется установить пакет gcc (который, кстати, устанавливает большую часть другого программного обеспечения, необходимого для компиляции), пакет libncurses5-dev (который предоставляет библиотеки, необходимые для использования утилиты конфигурации командной строки для исходного кода ядра), пакет bc (язык, поддерживающий точные числа) и пакет make (утилита make, используемая для компиляции исходного кода в двоичный файл).

```
apt-get install gcc libncurses5-dev bc make
```

Теперь откройте извлеченный исходный код ядра. Убедитесь, что в дереве исходного кода нет устаревших файлов, оставленных разработчиками:

```
make mrproper
```

Установите файлы заголовков для этого конкретного ядра. Они вам понадобятся в будущем, если вы собираетесь компилировать программное обеспечение, которое будет использовать API этого ядра в будущем:

```
make INSTALL_HDR_PATH=dest headers_install
cp -rv dest/include/* /usr/include
```

Я рекомендую вам использовать конфигурацию по умолчанию в качестве основы для сборки ядра, так как это, по крайней мере, гарантирует, что ваша система сможет загрузиться с использованием образа, который мы создадим позже.

```
make defconfig
```

Затем настройте свое ядро по своему вкусу.

```
make menuconfig
```

Эта книга не поможет вам разобраться, какие именно модули следует добавить в ядро — вам нужно самостоятельно определить точное оборудование, которое есть в вашей системе, и провести исследование относительно того, какие модули сделают это оборудование функциональным. Google — ваш друг.

После настройки ядра скомпилируйте его.

`make`

Если вы создали модульное ядро, установите в него скомпилированные модули:

`make modules_install`

Скопируйте готовое ядро в каталог /boot/ и убедитесь, что его имя начинается с 'vmlinuz'. Хорошей идеей будет добавить идентификационную строку к имени этого файла. Номер версии этого ядра подойдет. Мы также копируем файл System.map и конфигурацию для этого ядра (чтобы мы могли легко проверить, как было собрано это ядро). Мы также добавляем к этому нашу идентификационную строку, потому что со временем мы можем захотеть установить больше ядер.

```
cp -v arch/x86/boot/bzImage /boot/vmlinuz-(identifier)
cp -v System.map /boot/System.map-4.4.2
cp -v .config /boot/config-(identifier)
```

Adding modular functionality to the system

Regardless of if you installed the standard Debian GNU/Linux kernel, or compiled your own, you are going to need to provide your system with the binaries needed to load, remove and manipulate kernel modules.

`apt-get install kmod `

The only exception to this case would be if you compiled your own monolithic kernel with no modules, and do not intend on adding any beyond the ones you've already compiled into the kernel (which would generally be the case if you're doing embedded development).

Making the system bootable

Reconfiguring a drive with a pre-installed GRUB2 bootloader If this Debian Linux system is on a

partition of an already bootable drive, all one needs to do to make this system bootable is to replace the configuration file of the drive's bootloader to include this system's corresponding partition in its list of bootable partitions.

If using GRUB2, then using the `grub2-mkconfig` utility makes it very simple to do so. Merely point it to the location of the already existing grub.cfg file:

`grub2-mkconfig -o /boot/(path to grub.cfg)`

Installing GRUB2 onto a drive with no pre-installed bootloader In the event that the partition is on a drive that does not yet have its own bootloader, you must install it and configure it yourself. To install GRUB2, you must first exit your chroot.

```
exit
grub2-install /dev/(location of drive that holds this partition)`
```

Since this GRUB2 install does not have a configuration file yet, let's create one via template:

```
cat > /boot/grub/grub.cfg << "EOF"
# Begin /boot/grub/grub.cfg
set default=0
set timeout=5

insmod ext2
set root=(hd0,2)

menuentry "Debian from Scratch GNU/Linux" {
        linux   /boot/(kernel-location) root=/dev/sda2 ro
}
EOF
```

Then, edit this file to replace the (kernel-location) string in the file with the actual location of said kernel inside the partition.

Afterwards, use the previously aforementioned `chroot` command provided in the 'Entering our chroot' section to return to the chroot.

Creating /etc/fstab

Before you boot into your system, it is absolutely -vital- for you to create and configure your /etc/fstab file.

If you do -not- have an /etc/fstab file or fail to specify what partition is to be mounted as the root filesystem, you will be stuck within a temporary, read-only filesystem created by the kernel, located only in RAM. It uses this to switch into the supposed root filesystem. You do not want to be stuck here.

So we create a base /etc/fstab:

```
cat > /etc/fstab << "EOF"
# Begin /etc/fstab
```

```
# file system  mount-point  type     options             dump  fsck
#                                                              order

/dev/<xxx>     /            <fff>    defaults            1     1
/dev/<yyy>     swap         swap     pri=1               0     0

# End /etc/fstab
EOF
```

Modify the first entry with the partition location of this system, and the type of filesystem. The second entry should contain the location of your swap partition. If you don't intend on using one, remove the line.

The End

Congratulations! You've finished building your Debian system, and now have the complete range of functionality expected of a Debian system on top of your regular LFS install.

Comments, suggestions, bugs

If you found this guide useful, or have suggestions, the author would love to hear from you. Email him at

scottwilliambeasley AT gmail .com . Replace the AT with @, and remove spaces.

Also, contributions or additions to this manual are very much welcomed. You may do so by using this repository:

- https://github.com/scottwilliambeasley/debian-from-scratch/
- https://git.wwoss.ru/lfs/debian-from-scratch

1)
$(dpkg -l | wc -l)-5

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

Permanent link:
https://wwoss.ru/doku.php?id=software:linux_server:dfs:dfs&rev=1743013429

Last update: **2025/03/26 21:23**