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.

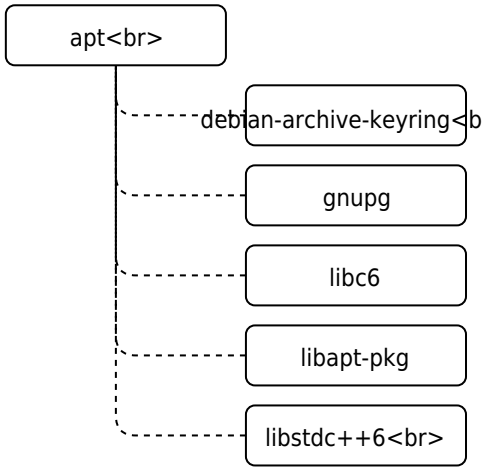


Рисунок 2. Дерево зависимостей apt, глубина в один уровень

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.

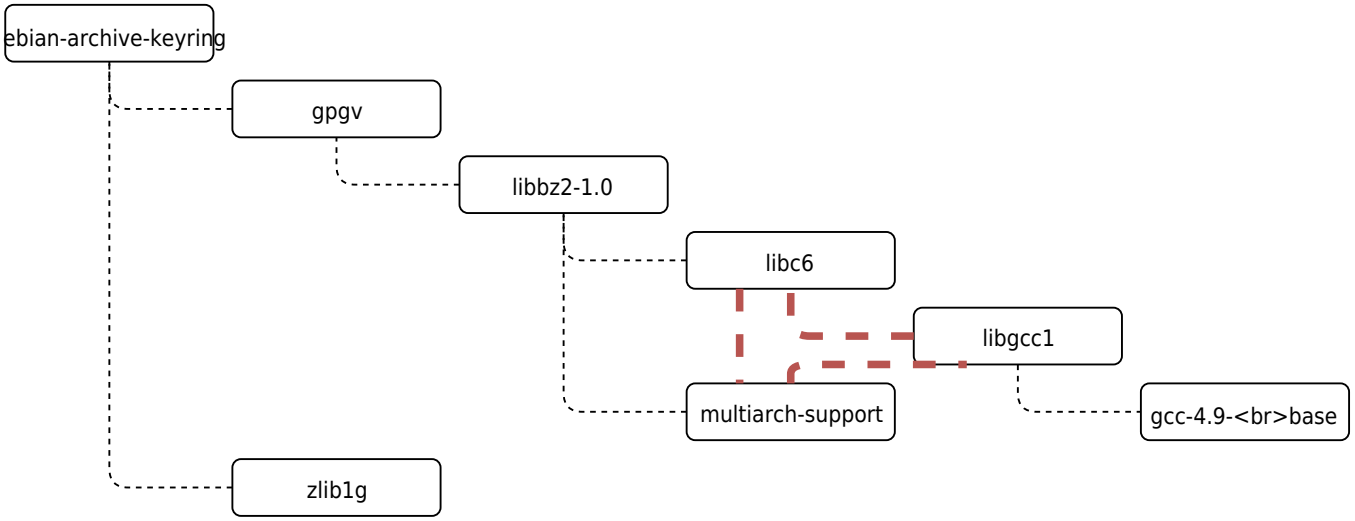


Рисунок 3. Дерево зависимостей Debian-Archive-Keyring, циклическая зависимость выделена красным

`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)
```

NOTE: You will receive an error here, containing errors very similar to the following:

<code bash> 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 <code>

This is normal - at this point you cannot fully install any of these packages. You can only partially install them, which we will remedy later on.

Once the package is installed, tweak the database to convince it that it -was- fully installed:

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

Now install the other two packages in order:

`dpkg -i (location_of_multiarch)`

`dpkg -i (location_of_libc6)`

And reinstall libgcc1 to cover over our ugly little hack and complete the full installation of each package:

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

Installing the rest of apt's dependencies

At this point, I am assuming that -all- of the .deb files you need to install have been placed in a single directory. Double check to see that you have all of these files. `cd` to this directory right now.

The truth is, for the remaining dependencies, the dependency tree for all of them is just too complicated for me to map out and give you a granular series of installation commands to execute what should otherwise be a straightforward operation.

Regardless of the actual dependency tree, there is a quick and dirty way to install the rest of the entire apt dependency tree. Simply execute the following command, and repeat it as many times as you need until dpkg no longer complains:

` dpkg -i *`

What happens here is that dpkg will attempt to install all software packages in the directory. It will inevitably fail, but do not fret. Simply repeat the command until everything is installed.

How this works

What happens when we run the command above is that dpkg is attempting to install each package without actually taking into mind the dependency tree. So with each execution of the above command, another level of the dependency tree will be fulfilled, until it's successfully able to confirm that the entire tree was installed. Once you no longer get any errors from dpkg, it means everything was installed.

dpkg is simply not as intelligent as something like apt, which would automatically build a dependency map, and install all prerequisite software before attempting to install software dependent on said software.

You might also notice that dpkg itself is part of the .deb files that needed to be installed. Don't worry, this doesn't break anything. We already have dpkg, but are just installing the official package to update its own database because the database never actually contained itself as part of the list of installed packages.

Double checking if everything is installed

To be absolutely certain, you can execute the following command, which counts how many packages dpkg counts as installed in its database:

`echo $[1)]`

If this returns the value 23, then you can correctly assume that everything was installed as intended.

Creating networking configuration files

Before we proceed with updating `apt's` cache, we need to define both the system's networking

configuration files, and apt's list of software repositories.

/etc/resolv.conf

`/etc/resolv.conf` is a file needed in order for your system to have DNS resolution. Without this file, no resolution typically happens, which makes updating `apt`'s cache of installable packages via `/etc/apt/sources.list` difficult.

<code bash> cat > /etc/resolv.conf « «EOF» nameserver 8.8.8.8 nameserver 8.8.4.4 EOF <code>

/etc/apt/sources.list

sources.list is a file which `apt` uses to contact the repositories that hold your software. You want to use the repositories from one and only one distribution as much as possible, otherwise you risk breaking your Debian's clear chain of dependencies and creating an insane mess of your system.

<code bash> 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 <code>

/etc/hosts

`/etc/hosts` is a file used to contain mapping of IP addresses to hostnames. This file is usually checked before DNS queries, at the very least, this should contain your `ipv4` and `ipv6` loopback addresses.

<code bash> 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 <code>

/etc/hostname

`/etc/hostname` is used to contain your host's DNS name. Edit this if you like.

<code bash> cat > /etc/hostname « «EOF» debianfromscratch EOF <code>

Updating apt's package lists

We are now ready to update our list of packages and take full advantage of `apt`. To do this, we update our local keyring of valid Debian developer gnu pgp signatures using `apt-key update`, and then update with `apt-get update`. <code bash> apt-key update apt-get update <code>

Creating user and group databases

Before we install more software, we must make sure that our password, group and authentication mechanisms are all in place. This is because some packages will require the adding of a new user or

group to the system as part of their installation process. Without these base functionalities already in place, installation of said packages will fail.

debianutils We install debianutils to provide the `tempfile` command needed by one of `base-passwd`'s installation scripts. Without this command, installation of `base-passwd` will fail.

`apt-get install debianutils`

base-passwd We install `base-passwd` to provide standard the standard minimal `/etc/passwd` and `/etc/group` files, which are the same across all debian systems. It does this by running the `update-passwd` binary upon its installation.

`apt-get install base-passwd`

Creating /etc/shadow and /etc/gshadow We have to manually create `/etc/shadow` and `/etc/gshadow`, as the `passwd` package will fail to configure if it cannot find these files:

`touch /etc/shadow /etc/gshadow`

login We then install the `login` package, which gives us the ability to establish new sessions on the system with `login`, privilege escalation with `su`, the linux pluggable authentication module (PAM) files for both said binaries, a fake shell `/bin/nologin`, and the `/etc/login.defs` file which is essential for group creation. There are more functionalities included with this package, but these are the most mentionable.

`apt-get install login`

passwd We then install `passwd` package, which provides the lion's share of utilities and configuration files used to create and manipulate user and group account information.

`apt-get install passwd`

adduser We must also install the `adduser` package, because this provides us with the default `/etc/adduser.conf` file which will be needed to install new users properly.

`apt-get install adduser`

Establishing root password and shadowfile entries With all the aforementioned utilities and packages installed, our system is now capable of the full functionality of user & group account manipulation.

At this point, we should run passwd to change our root password.

`passwd root`

We should then run `pwconv` to convert our /etc/passwd entries into shadow entries in `/etc/shadow`:

`pwconv`

Fixing the terminal and adding reading/editing utilities

Our terminal does not yet have the full functionality one would expect of a terminal, and standard terminal utilities may still fail to function properly at this point. Let's install the proper libraries and

create the configurations needed to make these , before we install

Creating /etc/inputrc file

`/etc/inputrc` is the global configuration file for the used by the `libreadline6` library, which most shells use in order to understand how to handle many special keyboard situations, such as what behavior should be default when hitting the HOME and END keys. Without this file, many special keys and two-key stroke combos such as ctrl+left will fail to work.

Since this file is not created by default when installing the `libreadline6` library, we must create it ourselves.

<code bash> 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 <code>

ncurses libraries and binaries A large number of command line utilites rely on the ncurses library to provide a text-interface for user interaction over the terminal. These include simple utilities such as `less` and `nano`. Without this library, these utilities will fail to display properly. We must install the complete suite of the ncurses library in order to prevent said errors from occurring:

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

dialog Dialog is a perl module which some scripts attempt to use to provide a text-interface used during configuration or installation. You may have noticed some packages warning you that this utility was non-existent during installation. Let's fix this:

`apt-get install dialog`

less, vim and nano Now that we've installed most of the libraries and utilities needed for terminal utilities, let's install some of the most basic and well-used ones:

`apt-get install less vim nano`

Creating the standard filesystem hierarchy on a Debian system Let's create the standard folder structure for a debian system. This can be done easily by installing `base-files`. First, we have to remove the `/var/mail` directory though or else it will complain that it already exists.

<code bash> rm -rf /var/mail apt-get install base-files <code>

Building the man documentation system Any Linux system typically has a database full of manual pages, accessed by the `man` command. Most programs we've installed already have already added their documentation files in the proper location. All we need to do now is actually install `man` to take advantage of them, and any more that are added as time passes by.

`apt-get install man`

Installing all remaining essential packages We've installed most of the packages, all that is left to do is install the rest of the packages marked with the priority `essential` by the Debian maintainers. Some of these are absolutely essential to system management, and some will barely be used at all.

To comply with the Debian standard, we must install all of these:

`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`

Here follows is a short description of each package installed:

Description	What it provides
———--	:—-:
bash:	The gnu bourne-again shell, which is your standard linux shell.
bsdutils:	Provides a few binaries, most notably `renice` which is needed for changing process priorities, and `logger` which is used for interacting with the syslog system module.
coreutils:	The absolute most essential group of binaries needed to make any shell useful.
dash:	The Debian Almquist shell, which is a faster version of sh intended mainly for use by scripts.

diffutils:	Provides utilities for comparing the contents of files between each other.
e2fsprogs:	Provides utilities for working with the ext family of filesystems.
findutils:	Provides the find utility for finding files.
grep:	Provides the grep utility, used for finding strings within files or output you pipe into it.
gzip:	Provides the gzip utility, used for working with files using LZ77 encoding.
hostname:	Provides the a set of utilities for manipulating the system's host name.
libc-bin:	Provides the GNU implementation of the standard C library. Essential for creating and using programs.
init:	Provides the standard system initialization suite for Debian.
mount:	Provides the standard system utilities for mounting and unmounting filesystems, including swapfiles.
perl-base:	Provides the perl programming language.
sed:	Provides the sed programming language, generally used for editing text.
sysvinit-utils:	Provides system-v like utilities.
tar:	Provides the tar program, used for storing and retrieving files from a taped archive.
util-linux:	Provides many vital system utilities.

Installing the kernel

You have two options here: either you can install the latest kernel image for your system architecture provided by the Debian project, or you can compile your own.

Installing Debian's kernel

If you want to install Debian's standard kernel, you will want to search for the available images provided for your architecture, and then install the approriate image.

<code bash> apt-cache search linux-image apt-get install (selected image) <code>

Compiling and installing your own custom kernel

You can also compile your own custom kernel if you feel like it.

To do this, you will need a tarball of the Linux kernel source, which you probably already have.

You will also need to install the gcc package (which incidentally installs most other software needed to compile), the libncurses5-dev package (which provides the libraries needed to use the command line configuration utility for the kernel source), the bc package (a language that supports precision numbers), and the make package (the make utility used for compiling the source into binary).

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

Now open up your extracted kernel source. Ensure that there are no stale files left behind from the developers in the source tree:

`make mrproper`

Install the header files for this particular kernel. You will need these in the future if you intend to compile software that will take advantage of this kernel's API in the future:

<code bash> make INSTALL_HDR_PATH=dest headers_install cp -rv dest/include/* /usr/include <code>

I recommend that you use the default configuration as the base for your kernel build, as it will at the very least ensure that your system will be able to boot using the image we create later.

`make defconfig`

Then, customize your kernel according to your heart's desire.

`make menuconfig`

This book cannot help you figure out what exact modules to add to the kernel - you need determine the exact hardware that exists on your system yourself, and do research as to what modules will make those pieces of hardware functional. Google is your friend.

After you've customized your kernel, compile it.

`make`

If you've created a modular kernel, install the compiled modules into it:

`make modules_install`

Copy the completed kernel into the /boot/ directory, and make sure that it's name starts with 'vmlinuz'. It's a good idea to append an identifying string to the name of this file. The version number of this kernel will do. We also copy the System.map file, and the config for this kernel (so we can easily examine how this kernel was built). We append our identifying string to this too, because as time passes by we may want to install more kernels.

<code bash> 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) <code>

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.

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

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

<code bash> 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 <code>

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:

<code bash> 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 <code>

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=1743011019

Last update: **2025/03/26 20:43**