



debian

Руководство для сопровождающих Debian

Osamu Aoki and Лев Ламберов

November 10, 2023

Руководство для сопровождающих Debian

by Osamu Aoki and Лев Ламберов

Copyright © 2014-2021 Osamu Aoki

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Данное руководство было создано на основе информации, содержащейся в следующих документах:

- «Создание пакета Debian (руководство по debmake)», copyright © 1997 Джалдхар Виас
- «Практическое руководство нового сопровождающего по созданию пакетов Debian», copyright © 1997 Уилл Лоу
- «Руководство начинающего разработчика Debian», copyright © 1998—2002 Джосип Родин, 2005—2017 Осаму Аоки, 2010 Крэйг Смолл, а также 2010 Рафаэль Херцог

Последняя версия данного руководства доступна:

- в пакете [debmake-doc](#) и
- на [веб-сайте Документации Debian](#).

Contents

1	Обзор	1
2	Необходимые предварительные требования	3
2.1	Люди вокруг Debian	3
2.2	Как принять участие	3
2.3	Социальная динамика Debian	4
2.4	Техническая памятка	4
2.5	Документация Debian	5
2.6	Справочные ресурсы	5
2.7	Ситуация с архивом	6
2.8	Подходы к участию	7
2.9	Начинающий участник и сопровождающий	8
3	Настройка инструментов	10
3.1	Адрес электронной почты	10
3.2	mc	10
3.3	git	11
3.4	quilt	11
3.5	devscripts	12
3.6	sbuild	12
3.7	git-buildpackage	13
3.8	HTTP-прокси	14
3.9	Частный репозиторий Debian	14
3.10	Virtual machines	14
3.11	Local network with virtual machines	14
4	Простой пример	15
4.1	Packaging tarball	15
4.2	Общая картина	15
4.3	Что такое debmake?	16
4.4	Что такое debuild?	17
4.5	Шаг 1: получение исходного кода основной ветки разработки	17
4.6	Шаг 2: создание шаблонных файлов с помощью debmake	18
4.7	Шаг 3: изменение шаблонных файлов	22
4.8	Шаг 4: сборка пакета с помощью debuild	24
4.9	Шаг 3 (альтернативный): изменение исходного кода основной ветки разработки	27
4.9.1	Создание заплаты с помощью diff -u	28
4.9.2	Создание заплаты с помощью dquilt	28
4.9.3	Создание заплаты с помощью dpkg-source --commit	30
5	Основы	32
5.1	Работа по созданию пакета	32
5.1.1	Пакет debhelper	34
5.2	Historical perspective of Debian packaging practices	34
5.3	Future perspective on Debian packaging practices	35
5.4	Имя пакета и версия	36
5.5	Родной пакет Debian	37
5.6	debian/rules	37
5.6.1	dh	38
5.6.2	Простой сценарий debian/rules	38
5.6.3	Настроенный файл debian/rules	39
5.6.4	Переменные debian/rules	39
5.6.5	Повторяемая сборка	40
5.7	debian/control	40

5.7.1	Split of a Debian binary package	41
5.7.1.1	debmake -b	41
5.7.1.2	Сценарии и примеры разделения пакета	42
5.7.1.3	Имя пакета библиотеки	42
5.7.2	Переменные подстановки	43
5.7.3	Безопасная binNMU-загрузка	44
5.8	debian/changelog	44
5.9	debian/copyright	45
5.10	debian/patches/*	46
5.10.1	dpkg-source -x	47
5.10.2	dquilt и dpkg-source	47
5.11	debian/upstream/signing-key.asc	48
5.12	debian/watch и критерии Debian по определению свободного ПО (DFSG)	48
5.13	Other debian/* Files	49
5.14	Настройка пакета Debian	53
5.15	Запись в систему управления версиями (стандарт)	53
5.16	Запись в систему управления версиями (альтернатива)	54
5.17	Сборка пакета без постороннего содержимого	55
5.17.1	Исправление с помощью debian/rules clean	55
5.17.2	Исправление с помощью систем управления версиями	55
5.17.3	Исправление с помощью extend-diff-ignore	56
5.17.4	Исправление с помощью tar-ignore	56
5.18	Системы сборки основной ветки	57
5.18.1	Autotools	57
5.18.2	CMake	57
5.18.3	Meson	58
5.18.4	Python distutils	58
5.19	Отладочная информация	58
5.19.1	New -dbgsym package (Stretch 9.0 and after)	59
5.20	Пакет библиотеки	59
5.20.1	Библиотека символов	60
5.20.2	Смена библиотек	61
5.21	debconf	61
5.22	Multiarch	61
5.22.1	Путь мультиархитектурной библиотеки	62
5.22.2	Путь мультиархитектурных заголовочных файлов	63
5.22.3	Мультиархитектурный путь к файлу *.pc	63
5.23	Усиление безопасности компилятора	64
5.24	Непрерывная интеграция	64
5.25	Other new requirements	64
5.26	Предзагрузка	64
5.27	Bug reports	65
6	Опции debmake	66
6.1	Опции быстрых действий (-a, -i)	66
6.1.1	Модуль Python	66
6.2	Snapshot upstream tarball (-d, -t)	67
6.3	debmake -cc	67
6.4	debmake -k	67
6.5	debmake -j	68
6.6	debmake -x	69
6.7	debmake -P	69
6.8	debmake -T	69

7	Полезные советы	70
7.1	debdiff	70
7.2	dget	70
7.3	debc	70
7.4	piuparts	70
7.5	debsign	71
7.6	dput	71
7.7	bts	71
7.8	git-buildpackage	71
7.8.1	gbp import-dscs --debsnap	72
7.8.2	git-репозиторий основной ветки	72
7.9	dgit	73
7.9.1	Snapshot upstream tarball (alternative git deborig approach)	73
7.10	chroot	74
7.11	Новая редакция Debian	75
7.12	Новый выпуск основной ветки	76
7.12.1	uupdate + tarball	76
7.12.2	uscan	77
7.12.3	gbp	77
7.12.4	gbp + uscan	77
7.13	Формат исходного кода 3.0	78
7.14	CDBS	78
7.15	Сборка с использованием кодировки UTF-8	79
7.16	Преобразование в кодировку UTF-8	79
7.17	Загрузите orig.tar.gz	79
7.18	Пропущенные загрузки	80
7.19	Продвинутые темы работы над пакетом	80
7.20	Другие дистрибутивы	81
7.21	Отладка	81
8	Дополнительные примеры	83
8.1	Выборочное применение шаблонов	83
8.2	Без Makefile (командная оболочка, интерфейс командной оболочки)	85
8.3	Makefile (командная оболочка, интерфейс командной оболочки)	91
8.4	setup.py (Python3, интерфейс командной оболочки)	94
8.5	Makefile (командная оболочка, графический интерфейс пользователя)	98
8.6	setup.py (Python3, графический интерфейс пользователя)	101
8.7	Makefile (single-binary package)	104
8.8	Makefile.in + configure (single-binary package)	106
8.9	Autotools (single-binary package)	110
8.10	СMake (single-binary package)	112
8.11	Autotools (multi-binary package)	116
8.12	СMake (multi-binary package)	121
8.13	Интернационализация	125
8.14	Детали	131
A	Страница руководства debmake(1)	132
A.1	НАЗВАНИЕ	132
A.2	СИНТАКСИС	132
A.3	ОПИСАНИЕ	132
A.3.1	необязательные аргументы:	132
A.4	ПРИМЕРЫ	135
A.5	ВСПОМОГАТЕЛЬНЫЕ ПАКЕТЫ	135
A.6	ПРЕДОСТЕРЕЖЕНИЯ	136
A.7	ОТЛАДКА	136
A.8	АВТОР	136
A.9	ЛИЦЕНЗИЯ	137
A.10	СМОТРИТЕ ТАКЖЕ	137

Abstract

Данное учебное руководство описывает сборку пакета Debian с помощью команды **debmake** и предназначено для обычных пользователей Debian и будущих разработчиков.

Руководство сконцентрировано на современном стиле создания пакетов и содержит множество простых примеров:

- Создание пакета, содержащего сценарий командной оболочки POSIX
- Создание пакета, содержащего сценарий на языке Python3
- C и Makefile/Autotools/CMake
- Несколько двоичных пакетов с разделяемой библиотекой и т.д.

Данное «Руководство для сопровождающих Debian» может рассматриваться как замена «Руководства начинающего разработчика Debian».

Предисловие

Если вы уже в какой-то степени являетесь опытным пользователем Debian 1, то можете столкнуться со следующими ситуациями:

- Желание установить некоторый пакет ПО, который пока отсутствует в архиве Debian.
- Желание обновить пакет Debian до более свежего выпуска из основной ветки разработки.
- Желание исправить ошибки в пакете Debian с помощью заплат.

Если вы хотите создать пакет Debian для удовлетворения указанных желаний, а также чтобы поделиться вашей работой с сообществом, то вы входите в целевую аудиторию данного руководства как будущий сопровождающий Debian. 2 Добро пожаловать в сообщество Debian.

В Debian имеется множество социальных и технических правил и договорённостей, которым необходимо следовать, поскольку Debian представляет собой большую организацию добровольцев со своей историей. Также в Debian был разработан огромный массив инструментов для создания пакетов и инструментов для сопровождения архива, которые позволяют создавать согласованный набор двоичных пакетов, отвечающих множеству технических целей:

- packages build across many architectures (Section 5.6.4)
- reproducible build (Section 5.6.5)
- clean build under clearly specified package dependencies and patches (Section 5.7, Section 5.10, Section 7.10)
- optimal splits into multiple binary packages (Section 5.7.1)
- smooth library transitions (Section 5.20.2)
- interactive installation customization (Section 5.21)
- multiarch support (Section 5.22)
- security enhancement using specific compiler flags (Section 5.23)
- continuous integration (Section 5.24)
- boot strapping (Section 5.26)
- ...

Всё это ошеломляет многих новых будущих сопровождающих Debian и затрудняет их участие в Debian. В настоящем руководстве делается попытка предоставить отправные точки для того, чтобы начать работу. В руководстве описано следующее:

- Что следует знать до того, как быть вовлечённым в Debian в качестве будущего сопровождающего.
- Как создать простой пакет Debian.
- Какие существуют виды правил для создания пакета Debian.

¹Вам необходимо знать немного о программировании в Unix, но от вас определённо не требуется быть экспертом. Вы можете узнать об основах управления системой Debian из [Справочника Debian](#). Там же можно найти ссылки на ресурсы для изучения программирования в Unix.

²Если вы не желаете делиться пакетом Debian с другими, то вы, разумеется, можете скомпилировать ПО и установить пакет с исправленным исходным кодом из основной ветки разработки в каталог `/usr/local/`.

- Полезные советы по созданию пакета Debian.
- Примеры создания пакетов Debian для некоторых типовых сценариев.

Автор осознаёт возможные ограничения при обновлении изначального «Руководства начинающего разработчика Debian» и добавлении сведений по использованию пакета **dh-make** и принял решение создать альтернативный инструмент и соответствующую ему документацию для того, чтобы они соответствовали современным требованиям. Результатом является пакет **debmake** (версия 4.4.0) и данное обновлённое «Руководство для сопровождающих Debian» в пакете **debmake-doc** (версия 1.17-9).

В команду **debmake** было интегрировано большое количество полезных советов и рутинных действий, что значительно упрощает данное руководство. Кроме того, руководство содержит множество примеров создания пакетов.

Caution



На создание и сопровождение пакета Debian хорошего качества уходят многие часы. Для выполнения этой задачи сопровождающий Debian должен быть одновременно **и технически компетентным, и усердным**.

Некоторые важные темы объяснены подробно. Некоторые из них могут показаться вам незначительными, но, пожалуйста, проявите терпение. Некоторые специальные случаи пропускаются. Для некоторых тем приводятся только ссылки на внешние источники. Всё это сделано намеренно, чтобы данное руководство оставалось простым и удобным в сопровождении.

Chapter 1

Обзор

Создание пакета Debian из архива *package-1.0.tar.gz*, содержащего простой исходный код на языке C, соответствующий [Стандартам написания кода GNU](#) и [Стандарту иерархии файловой системы](#), может быть выполнено с помощью команды **debmake**, как показано ниже.

```
$ tar -xvzf package-1.0.tar.gz
$ cd package-1.0
$ debmake
... Make manual adjustments of generated configuration files
$ debuild
```

Если будет пропущена ручная правка созданных настроечных файлов, то в созданном двоичном пакете будет отсутствовать осмысленное описание, но он будет вполне работоспособным при использовании команды **dpkg** для его локального развёртывания.

Caution



Команда **debmake** предоставляет лишь хорошие файлы шаблонов. Если пакет предназначен для широкой аудитории, то эти шаблоны следует отредактировать вручную, чтобы довести их до совершенства и соответствия строгим требованиям качества, предъявляемым к архиву Debian.

Если вы только начинаете создавать пакеты Debian, то не беспокойтесь о деталях и переходите к рассмотрению общей картины.

Если у вас уже имеется опыт создания пакетов Debian, то предлагаемое в данном руководстве очень похоже на использование команды **dh_make**, так как команда **debmake** задумывалась в качестве замены для функций, исторически предоставляемых командой **dh_make**. [1](#)

Команда **debmake** имеет следующие возможности:

- современный стиль создания пакетов
 - **debian/copyright**: [DEP-5](#) compliant
 - **debian/control**: поддержка **переменных подстановки**, поддержка **мультиархитектурности**, несколько двоичных пакетов, ...
 - **debian/rules**: синтаксис **dh**, опции компилятора для улучшения безопасности, ...
- гибкость
 - множество опций (см. [Section 5.7.1.1](#) и [Chapter 6, Appendix A](#))
- разумные действия по умолчанию
 - выполнение без остановок с чистыми результатами

¹Команда **deb-make** была популярна ещё до команды **dh_make**. Версии текущего пакета **debmake** начинаются с **4.0**, чтобы исключить совпадения и конфликты версий с устаревшим пакетом **debmake**, предоставлявшим команду **deb-make**.

- создание мультиархитектурного пакета, если явно не указана опция **-m**.
- создание неродного пакета Debian в формате «**3.0 (quilt)**», если явно не указана опция **-n**.
- дополнительные обслуживающие программы
 - проверка **debian/copyright** с учётом текущего исходного кода (см. Section 6.4)

The **debmake** command delegates most of the heavy lifting to its back-end packages: **debhelper**, **dpkg-dev**, **devscripts**, **sbuild**, **schroot**, etc.

Tip



Обязательно защитите путём соответствующего включения в кавычки аргументы опций **-b**, **-f**, **-l** и **-w** от вмешательства командной оболочки.

Tip



Неродной пакет Debian — обычный пакет Debian.

Tip



Подробный журнал всех примеров сборки пакетов из данной документации можно получить, следуя инструкциям из Section 8.14.

Note



Создание файла **debian/copyright**, а также результаты работы опций **-c** (Section 6.3) и **-k** (Section 6.4) предполагают эвристические операции, производимые над информацией об авторском праве и лицензиях. Это может приводить к получению ошибочных результатов.

Chapter 2

Необходимые предварительные требования

Ниже приведены необходимые предварительные требования, которые вам нужно понять до того, как вы примете участие в Debian.

2.1 Люди вокруг Debian

Существует несколько типов людей, взаимодействующих с Debian в рамках разных ролей:

- **Автор основной ветки разработки:** тот, кто создал исходную программу.
- **Сопровождающий основной ветки разработки:** тот, кто в настоящее время сопровождает программу.
- **Сопровождающий:** тот, кто создаёт пакет Debian с программой.
- **Поручитель:** тот, кто помогает сопровождающим загружать пакеты в официальный архив пакетов Debian (после проверки содержимого пакетов).
- **Ментор:** тот, кто помогает начинающим сопровождающим создавать пакеты и проч.
- **разработчик Debian (DD):** член проекта Debian с полными правами на загрузку в официальный архив пакетов Debian.
- **сопровождающий Debian (DM):** тот, кто имеет ограниченные права на загрузку в официальный архив пакетов Debian.

Заметьте, что сразу же стать официальным **разработчиком Debian (DD)** нельзя, так как для этого требуется нечто большее, чем только технические навыки. Тем не менее, не отчаивайтесь из-за этого. Если ваша работа полезна для других, вы всё равно можете загрузить ваш пакет либо как **сопровождающий** через **поручителя**, либо как **сопровождающий Debian**.

Помните, для того чтобы стать разработчиком Debian не обязательно создавать какие-либо новые пакеты. Участие в уже существующих пакетах тоже может дать вам возможность к получению статуса разработчика Debian. Множество пакетов ждут своих хороших сопровождающих (см., Section 2.8).

2.2 Как принять участие

Чтобы узнать, как принять участие в Debian, обратите внимание на следующее:

- [Как вы можете помочь Debian?](#) (официальный источник)
- [The Debian GNU/Linux FAQ, Chapter 13 - "Contributing to the Debian Project"](#) (semi-official)
- [Debian Wiki, HelpDebian](#) (дополнительный источник)
- [Сайт новых участников Debian](#) (официальный источник)
- [ЧаВО для менторов Debian](#) (дополнительный источник)

2.3 Социальная динамика Debian

Для подготовки к взаимодействию с Debian следует понять социальную динамику Debian, которая состоит в следующем:

- Все мы являемся добровольцами.
 - Вы не можете диктовать остальным, что им делать.
 - У вас должна быть мотивация делать что-то самостоятельно.
- Движущей силой является дружеское сотрудничество.
 - Ваше участие не должно чрезмерно досаждать остальным.
 - Ваш вклад ценен только в том случае, если остальные вам за него признательны.
- Debian — это не школа, где вы автоматически получаете внимание учителей.
 - Вам следует быть готовым к самостоятельному обучению многим вещам.
 - Внимание других добровольцев является очень дефицитным ресурсом.
- Debian постоянно улучшается.
 - От вас ожидается, что вы будете создавать пакеты высокого качества.
 - Вы сами должны адаптироваться к изменениям.

Поскольку в оставшейся части настоящего руководства мы концентрируемся исключительно на технических аспектах создания пакетов, постольку чтобы понять социальную динамику Debian, рекомендуем обратиться к следующей документации:

- [Debian: 17 лет Свободного ПО, «делократия» и демократия](#) (вводные слайды одного из бывших Лидеров проекта)

2.4 Техническая памятка

Ниже приведена техническая памятка, целью которой является облегчение работы других сопровождающих над вашим пакетом и увеличение полезности для Debian в целом.

- Упростите отладку вашего пакета.
 - Делайте ваш пакет простым.
 - Не усложняйте ваш пакет.
- Хорошо документируйте ваш пакет.
 - Используйте читаемый стиль для исходного кода.
 - Оставляйте в коде комментарии.
 - Форматируйте свой код везде одинаковым образом.
 - Сопровождайте git-репозиторий [1](#) пакета.

Note



Отладка ПО чаще требует большего количества времени, чем написание изначально работающего ПО.

¹ Подавляющее большинство сопровождающих Debian используют **git**, а не другие системы управления версиями, такие как **hg**, **bzr** и т.д.

2.5 Документация Debian

Please make yourself ready to read the pertinent part of the latest Debian documentation to generate perfect Debian packages:

- «Руководство по политике Debian»
 - The official “must follow” rules (<https://www.debian.org/doc/devel-manuals#policy>)
- «Справочник разработчика Debian»
 - The official “best practice” document (<https://www.debian.org/doc/devel-manuals#devref>)
- “Guide for Debian Maintainers” — this guide
 - A “tutorial reference” document (<https://www.debian.org/doc/devel-manuals#debmake-doc>)

All these documents are published to <https://www.debian.org> using the **unstable** suite versions of corresponding Debian packages. If you wish to have local accesses to all these documents from your base system, please consider to use techniques such as [apt-pinning](#) and [chroot](#) (Section 7.10).²

Если данное руководство противоречит официальной документации Debian, то верной является последняя. В таком случае отправьте сообщение об ошибке в пакете **debmake-doc** с помощью команды **reportbug**.

Также существует следующая альтернативная вводная документация, которую вы можете прочитать вместе с настоящим руководством:

- «Руководство нового сопровождающего Debian» (более старое)
 - <https://www.debian.org/doc/devel-manuals#maint-guide>
 - <https://packages.qa.debian.org/m/maint-guide.html>
- «Введение в создание пакетов Debian»
 - <https://www.debian.org/doc/devel-manuals#packaging-tutorial>
 - <https://packages.qa.debian.org/p/packaging-tutorial.html>
- «Руководство по созданию пакетов Ubuntu» (дистрибутив Ubuntu основан на Debian.)
 - <http://packaging.ubuntu.com/html/>

Tip



When reading these tutorials, you should consider using the **debmake** command in place of the **dh_make** command.

2.6 Справочные ресурсы

Прежде чем публично задать вопрос, приложите усилия для его самостоятельного разрешения, например, прочтите хорошую документацию:

- Информацию о пакете, доступную с помощью команд **aptitude**, **apt-cache** и **dpkg**.
- Файлы в каталоге `/usr/share/doc/пакет` для всех релевантных пакетов.
- Содержимое **man** команда для всех релевантных команд.

²It is unwise to run your base system under the pure **unstable** suite even for the development system. Normally, most package development activities use **unstable** [chroot](#) as in Section 7.10. By using [virtualization](#), you can test demanding situation such as the full Desktop system, network daemons, and system installer in the **unstable** suite environment even from the base system running the **stable** suite.

- Содержимое **info** команда для всех релевантных команд.
- Содержимое архива списка рассылки debian-mentors@lists.debian.org.
- Содержимое архива списка рассылки debian-devel@lists.debian.org.

Нужную вам информацию можно оперативно найти с помощью хорошо сформированного поискового запроса, такого как «ключевое-слово **site:lists.debian.org**». Такой запрос ограничивает поиск указанным доменом.

Создание небольшого тестового пакета — хороший способ изучить детали создания пакетов. Исследование существующих хорошо сопровождаемых пакетов — лучший способ изучить то, как другие люди создают пакеты.

Если у вас всё ещё остались вопросы по поводу создания пакетов, вы можете задать их в следующих списках рассылки:

- Список рассылки debian-mentors@lists.debian.org (предназначен для новичков.)
- Список рассылки debian-devel@lists.debian.org (предназначен для экспертов)
- IRC, например, в #debian-mentors
- Команды концентрируют усилия на конкретном наборе пакетов. (Полный список команд доступен по адресу <https://wiki.debian.org/Teams>)
- Списки рассылки, в которых принято общаться на отличных от английского языках.
 - debian-devel-{french,italian,portuguese,spanish}@lists.debian.org
 - debian-chinese-gb@lists.debian.org (This mailing list is for general (Simplified) Chinese discussion.)
 - debian-devel@debian.or.jp

Более опытные разработчики Debian с радостью вам помогут, если вы правильно зададите вопрос после того, как уже самостоятельно попробовали разобраться.

Caution



Разработка Debian представляет собой движущуюся цель. Информация, которую можно найти в Сети, может оказаться устаревшей, неправильной, неприменимой. Используйте её с осмотрительно.

2.7 Ситуация с архивом

Пожалуйста, поймите ситуацию с архивом Debian.

- В Debian уже имеются пакеты для большинства видов программ.
- Число пакетов в архиве Debian уже в несколько раз превышает число активных сопровождающих.
- К сожалению, некоторые пакеты нуждаются в должном внимании сопровождающих.

Поэтому, участие в работе над уже добавленными в архив пакетами более чем ценно и желательно (и гораздо больше вероятность получить поручительство для загрузки) со стороны других сопровождающих.

Tip



Команда **wpp-alert** из пакета **devscripts** может проверять, если ли среди установленных пакетов пакеты открытые для усыновления или же осиротевшие пакеты.

Tip



The **how-can-i-help** package can show opportunities for contributing to Debian on packages installed locally.

2.8 Подходы к участию

Ниже приводится псевдокод на питоноподобном языке, описывающий в **программном** виде возможности вашего участия в Debian:

```

if exist_in_debian(program):
    if is_team_maintained(program):
        join_team(program)
    if is_orphaned(program): # maintainer: Debian QA Group
        adopt_it(program)
    elif is_RFA(program): # Request for Adoption
        adopt_it(program)
    else:
        if need_help(program):
            contact_maintainer(program)
            triaging_bugs(program)
            preparing_QA_or_NMU_uploads(program)
        else:
            leave_it(program)
else: # new packages
    if not is_good_program(program):
        give_up_packaging(program)
    elif not is_distributable(program):
        give_up_packaging(program)
    else: # worth packaging
        if is_ITPed_by_others(program):
            if need_help(program):
                contact_ITPer_for_collaboration(program)
            else:
                leave_it_to_ITPer(program)
        else: # really new
            if is_applicable_team(program):
                join_team(program)
            if is_DFSG(program) and is_DFSG(dependency(program)):
                file_ITP(program, area="main") # This is Debian
            elif is_DFSG(program):
                file_ITP(program, area="contrib") # This is not Debian
            else: # non-DFSG
                file_ITP(program, area="non-free") # This is not Debian
            package_it_and_close_ITP(program)

```

Где:

- Для функций `exist_in_debian()` и `is_team_maintained()` нужно проверить следующее:
 - команду **aptitude**
 - веб-страницу [пакеты Debian](#)
 - [команды участников](#)
- Для функций `is_orphaned()`, `is_RFA()` и `is_ITPed_by_others()` нужно проверить следующее:
 - вывод команды **wnpp-alert**

- пакеты требующие доработки и будущие
 - журналы отчётов об ошибках Debian: ошибки в псевдопакете wnpp в нестабильном выпуске
 - пакеты Debian, которым требуется внимание и забота
 - ошибки в пакете wnpp по меткам debtag
- Для функции `is_good_program()` нужно проверить следующее:
 - программа должна быть полезна
 - программа не усложняет поддержку безопасности и сопровождение системы Debian
 - программа хорошо документирована, а её код понятен (то есть, не обфусцирован)
 - авторы программы согласны с созданием пакета и дружелюбно относятся к Debian [3](#)
 - Для функций `is_it_DFSG()` и `is_its_dependency_DFSG()` нужно проверить следующее:
 - [Критерии Debian по определению Свободного ПО \(DFSG\)](#).
 - Для функции `is_it_distributable()` нужно проверить следующее:
 - ПО должно иметь лицензию и лицензия должна разрешать распространение ПО.

Вам необходимо либо отправить сообщение об ошибке **ИТР** или усыновить пакет, чтобы начать над ним работать. См. «Справочник разработчика Debian»:

- [5.1. Новые пакеты.](#)
- [5.9. Перемещение, удаление, переименование, придание статуса осиротевшего, усыновление и повторное введение пакетов.](#)

2.9 Начинаящий участник и сопровождающий

Начинаящий участник и сопровождающий могут недоумевать по поводу того, что же следует изучить, чтобы начать участвовать в Debian. Ниже приводятся некоторые предложения в зависимости от того, чем вы хотите заниматься.

- Создание пакетов
 - Основы **командной оболочки POSIX** и инструмента **make**.
 - Некоторое зачаточное знание **Perl** и **Python**.
- Перевод
 - Основы работы системы перевода PO.
- Документация
 - Основы различных языков разметки (XML, ReST, Wiki, ...).

Начинаящий участник и сопровождающий могут недоумевать по поводу того, где же начать участвовать в Debian. Ниже приводятся некоторые предложения в зависимости от ваших навыков.

- Навыки работы с **командной оболочкой POSIX, Perl и Python**:
 - Отправляйте заплатки для программы установки Debian.
 - Send patches to the Debian packaging helper scripts such as **devscripts**, **sbuild**, **schroot**, etc. mentioned in this document.
- Навыки **C** и **C++**:

Это не является абсолютным требованием. Тем не менее, враждебные разработчики основной ветки могут стать тем, что будет опустошать ресурсы всех нас. С дружелюбными разработчиками можно консультироваться в решении любых проблем с программой.

- Отправляйте заплаты для пакетов, имеющих приоритеты **required** и **important**.
- Навыки работы с отличными от английского языками:
 - Отправляйте заплаты для PO-файлов программы установки Debian.
 - Отправляйте заплаты для PO-файлов пакетов, имеющих приоритеты **required** и **important**.
- Навыки написания документации:
 - Обновляйте содержание [Debian Wiki](#).
 - Отправляйте заплаты к существующей [документации Debian](#).

Эта деятельность даст вам возможность познакомиться с другими участниками Debian и улучшить вашу репутацию.

Начинающему сопровождающему следует избегать работу над пакетами, содержащими программы с высокими рисками в плане безопасности:

- программы, имеющие флаги доступа **setuid** или **setgid**
- **службы**
- программы, устанавливаемые в каталоги **/sbin/** или **/usr/sbin/**

Когда вы получите больше опыта в работе над пакетами, вы сможете создавать пакеты и с такими программами.

Chapter 3

Настройка инструментов

В сборочном окружении должен быть установлен пакет **build-essential**.

В окружении сопровождающего должен быть установлен пакет **devscripts**.

Although this is not necessarily an absolute requirement, it is a good idea to install and set up all of the popular set of packages mentioned in this chapter in the maintainer environment. This enables us to share the common baseline working environment. . Please install the tools mentioned in the [Overview of Debian Maintainer Tools](#) in the “Debian Developer’s Reference”, as needed, too.

Caution



Настройки инструментов, представленные ниже, являются лишь примером и могут быть неактуальны при использовании самых свежих пакетов. Разработка Debian является движущейся целью. Обязательно прочтите соответствующую документацию и при необходимости обновите настройки.

3.1 Адрес электронной почты

Различные инструменты сопровождения Debian назначают ваш адрес электронной почты и ваше имя из переменных окружения **\$DEBEMAIL** и **\$DEBFULLNAME**.

Настроим эти пакеты, добавив в `~/.bashrc` [1](#) приведённые ниже строки.

Добавьте в файл `~/.bashrc`

```
DEBEMAIL="your.email.address@example.org"
DEBFULLNAME="Firstname Lastname"
export DEBEMAIL DEBFULLNAME
```

3.2 mc

Команда **mc** предлагает вам простой способ работы с файлами. Она может открывать двоичные **deb**-файлы для проверки их содержимого по простому нажатию клавиши «Ввод» при выборе соответствующего двоичного **deb**-файла. В качестве движка эта программа использует команду **dpkg-deb**. Настроим её на поддержку простой функции **chdir** следующим образом.

Добавьте в файл `~/.bashrc`

```
# mc related
if [ -f /usr/lib/mc/mc.sh ]; then
    . /usr/lib/mc/mc.sh
fi
```

¹Предполагается, что в качестве интерактивной командной оболочки с регистрацией вы используете Bash. Если вы используете какую-то другую командную оболочку, например, Zsh, то вместо `~/.bashrc` необходимо изменить соответствующие файлы настройки.

3.3 git

На сегодняшний день команда **git** является необходимым инструментом для работы с деревом исходного кода с историей.

Глобальные пользовательские настройки для команды **git**, такие как ваши имя и адрес электронной почты, можно установить в файле `~/.gitconfig` следующим образом.

```
$ git config --global user.name "Name Surname"
$ git config --global user.email yourname@example.com
```

Если вы привыкли использовать команды CVS или Subversion, то можете установить несколько указанных ниже псевдонимов команд.

```
$ git config --global alias.ci "commit -a"
$ git config --global alias.co checkout
```

Проверить ваши глобальные настройки можно следующим образом.

```
$ git config --global --list
```

Tip



Для эффективной работы с историей git-репозитория необходимо использовать какой-нибудь инструмент с графическим интерфейсом пользователя, например, **gitk** или **gitg**.

3.4 quilt

Команда **quilt** предлагает простой метод записи изменений. Для работы с пакетами Debian следует выполнить настройку так, чтобы изменения записывались в каталог **debian/patches/** вместо каталога **patches/** по умолчанию.

Чтобы не менять поведение самой команды **quilt**, создадим псевдоним **dquilt** для работы с пакетами Debian, добавив следующие строки в файл `~/.bashrc`. Вторая строка предоставляет команде **dquilt** ту же функциональность автодополнения, что и у команды **quilt**.

Добавьте в файл `~/.bashrc`

```
alias dquilt="quilt --quiltrc=${HOME}/.quiltrc-dpkg"
. /usr/share/bash-completion/completions/quilt
complete -F _quilt_completion $quilt_complete_opt dquilt
```

Теперь создадим файл `~/.quiltrc-dpkg` со следующим содержимым.

```
d=.
while [ ! -d $d/debian -a `readlink -e $d` != / ];
do d=$d/..; done
if [ -d $d/debian ] && [ -z $QUILT_PATCHES ]; then
# if in Debian packaging tree with unset $QUILT_PATCHES
QUILT_PATCHES="debian/patches"
QUILT_PATCH_OPTS="--reject-format=unified"
QUILT_DIFF_ARGS="-p ab --no-timestamps --no-index --color=auto"
QUILT_REFRESH_ARGS="-p ab --no-timestamps --no-index"
QUILT_COLORS="diff_hdr=1;32:diff_add=1;34:diff_rem=1;31:diff_hunk=1;33:"
QUILT_COLORS="${QUILT_COLORS}diff_ctx=35:diff_cctx=33"
if ! [ -d $d/debian/patches ]; then mkdir $d/debian/patches; fi
fi
```

О том, как использовать команду **quilt** см. в **quilt(1)** и [Как выжить, имея много заплат, или Введение в Quilt](#).

Для примеров использования см. Section 4.9.

3.5 devscripts

Для подписывания пакета Debian вашим закрытым GPG-ключом используется команда **debsign**, входящая в состав пакета **devscripts**.

Команда **debbuild**, входящая в состав пакета **devscripts**, собирает двоичный пакет и проверяет его с помощью команды **lintian**. Полезно иметь более подробный вывод команды **lintian**.

Вы можете настроить эти команды в файле `~/devscripts` следующим образом.

```
DEBUILD_DPKG_BUILDPACKAGE_OPTS="-i -I -us -uc"
DEBUILD_LINTIAN_OPTS="-i -I --show-overrides"
DEBSIGN_KEYID="Your_GPG_keyID"
```

Опции **-i** и **-I** в **DEBUILD_DPKG_BUILDPACKAGE_OPTS** для команды **dpkg-source** помогают повторно собирать пакеты Debian без какого-либо постороннего содержимого (см. Section 5.17).

В настоящее время хорошо иметь RSA-ключ длины 4096 бит, см. [Создание нового GPG-ключа](#).

3.6 sbuild

The **sbuild** package provides a clean room (**chroot**) build environment. It offers this efficiently with the help of **schroot** using the bind-mount feature of the modern Linux kernel.

Since it is the same build environment as the Debian's **build** infrastructure, it is always up to date and comes with full of useful features.

It can be customized to offer following features:

- The **schroot** package to boost the chroot creation speed.
- Пакет **lintian** предназначен для обнаружения ошибок в пакете.
- The **piuparts** package to find bugs in the package.
- The **autopkgtest** package to find bugs in the package.
- Пакет **ccache** предназначен для увеличения скорости работы **gcc** (необязательно).
- Пакет **libeatmydata1** предназначен для увеличения скорости работы **dpkg** (необязательно).
- Параллельный запуск **make** позволяет увеличить скорость сборки (необязательно).

Let's get started by following <https://wiki.debian.org/sbuild>.

```
$ sudo apt install sbuild piuparts autopkgtest lintian
$ sudo usermod -a -G <your_user_name> sbuild
```

Reboot your system or execute command such as "**kill -TERM -1**"². Then login and check you are a member of **sbuild** group using **id** command.

```
$ id
uid=1000(<yourname>) gid=1000(<yourname>) groups=...,132(sbuild)
```

Let's create the configuration file `~/sbuilderc` in line with recent Debian practice <https://wiki.debian.org/SourceOnlyUpload> as:

```
cat >~/sbuilderc << 'EOF'
#####
# PACKAGE BUILD RELATED (source-only-upload as default)
#####
# -d
$distribution = 'unstable';
# -A
$build_arch_all = 1;
# -s
$build_source = 1;
# --source-only-changes
$source_only_changes = 1;
```

²Logout from your system via GUI menu may not work.

```
# -v
$verbose = 1;

#####
# POST-BUILD RELATED (turn off functionality by setting variables to 0)
#####
$run_lintian = 1;
$lintian_opts = ['-i', '-I'];
$run_piuparts = 1;
$piuparts_opts = ['--schroot', 'unstable-amd64-sbuild'];
$run_autopkgtest = 1;
$autopkgtest_root_args = '';
$autopkgtest_opts = [ '--', 'schroot', '%r-%a-sbuild' ];

#####
# PERL MAGIC
#####
1;
EOF
```

Following document assumes that sbuild is configured this way.

Edit this to your needs. Post-build tests can be turned on and off by assigning 1 or 0 to the corresponding variables,

Warning



Необязательные настройки могут вызывать отрицательные последствия. Отключите их в случае сомнения.

Note



Параллельный запуск **make** может быть неудачным для некоторых уже имеющихся пакетов и может сделать журнал сборки сложным для прочтения.

3.7 git-buildpackage

Вам может потребоваться установить некоторые глобальные настройки в файле `~/gbp.conf`

```
# Configuration file for "gbp <command>"

[DEFAULT]
# the default build command:
builder = sbuild
# use pristine-tar:
pristine-tar = True
# Use color when on a terminal, alternatives: on/true, off/false or auto
color = auto
```

Tip



Команда **gbp** является псевдонимом команды **git-buildpackage**.

3.8 HTTP-прокси

Чтобы сохранить пропускную способность при обращении к репозиторию пакетов Debian вам следует настроить локальный кэширующий HTTP-прокси. Имеется несколько вариантов:

- Специализированный кэширующий HTTP-прокси, использующий пакет **apt-cacher-ng**.
- Generic HTTP caching proxy (**squid** package) configured by **squid-deb-proxy** package

In order to use this HTTP proxy without manual configuration adjustment, it's a good idea to install either **auto-apt-proxy** or **squid-deb-proxy-client** package to everywhere.

3.9 Частный репозиторий Debian

Вы можете настроить собственный репозиторий пакетов Debian с помощью пакета **reprepro**.

3.10 Virtual machines

For testing GUI application, it is good idea to have virtual machines. Install **virt-manager** and **qemu-kvm**.

Use of chroot and virtual machines allow us not to update the whole host PC to the latest unstable.

3.11 Local network with virtual machines

In order to access virtual machines easily over the local network, setting up multicast DNS service discovery infrastructure by installing **avahi-utils** is a good idea.

For all running virtual machines and the host PC, we can use each host name appended with **.local** for SSH to access each other.

Chapter 4

Простой пример

Есть старая латинская поговорка: «**Longum iter est per praecepta, breve et efficax per exempla**» («Долг путь поучений, короток и успешен путь примеров»).

4.1 Packaging tarball

Ниже приведён пример создания простого пакета Debian из простого исходного кода на языке C, использующего в качестве системы сборки **Makefile**.

Допустим, имя tar-архива из основной ветки разработки будет **debhello-0.0.tar.gz**.

Предполагается, что этот тип исходного кода будет установлен как несистемный файл:

```
$ tar -xzmf debhello-0.0.tar.gz
$ cd debhello-0.0
$ make
$ make install
```

При создании пакетов Debian требуется изменить процесс установки файлов с помощью «**make install**» так, чтобы установка производилось в системный каталог, а не в обычный каталог в **/usr/local**.

Note



Примеры создания пакета Debian из других более сложных систем сборки описаны в [Chapter 8](#).

4.2 Общая картина

Общая картина сборки простого неродного пакета Debian из tar-архива основной ветки разработки **debhello-0.0.tar.gz** может быть представлена следующим образом:

- Сопровождающий получает tar-архив **debhello-0.0.tar.gz** из основной ветки разработки и распаковывает его содержимое в каталог **debhello-0.0**.
- Команда **debmake** добавляет шаблонные файлы исключительно в каталог **debian**.
 - Создаётся символическая ссылка **debhello_0.0.orig.tar.gz**, указывающая на файл **debhello-0.0.tar.gz**.
 - Сопровождающий настраивает шаблонные файлы.
- Команда **debuild** собирает двоичный пакет из подготовленного дерева исходного кода.
 - Создаётся файл **debhello-0.0-1.debian.tar.xz**, содержащий каталог **debian**.

Общая картина сборки пакета


```
$ tar -xzmf debhello-0.0.tar.gz
$ cd debhello-0.0
$ debmake
... manual customization
$ debuild
...
```

Tip



The **debuild** command in this and following examples may be substituted by equivalent commands such as the **sbuild** command.

Tip



Если доступен tar-архив основной ветки разработки в формате **.tar.xz**, то используйте его вместо архивов в формате **.tar.gz** или **.tar.bz2**. Утилита **xz** предлагает более высокую степень сжатия, чем **gzip** и **bzip2**.

4.3 Что такое debmake?

Команда **debmake** является вспомогательным сценарием для создания и изменения пакетов Debian.

- Она всегда устанавливает большинство очевидных опций в разумные значения.
- Создаёт tar-архив основной ветки разработки и необходимую символическую ссылку в случае их отсутствия.
- Не переписывает существующие файлы настройки в каталоге **debian/**.
- Поддерживает **мультиархитектурные** пакеты.
- Создаёт хорошие шаблонные файлы. Например, файл **debian/copyright**, соответствующий **DEP-5**.

Эти возможности делают работу с пакетами Debian с помощью **debmake** простой и современной.

In retrospect, I created **debmake** to simplify this documentation. I consider **debmake** to be more-or-less a demonstration session generator for tutorial purpose.

Note



Many packages are packaged using only a text editor while imitating how other similar packages are packaged and consulting how the Debian policy requires us to do. This seems to me the most popular method for the real-life packaging activity.

The **debmake** command isn't the only helper script to make a Debian package. If you are interested alternative packaging helper tools, please see:

- [Debian wiki: AutomaticPackagingTools](#) — Extensive comparison of packaging helper scripts
- [Debian wiki: CopyrightReviewTools](#) — Extensive comparison of copyright review helper scripts

4.4 Что такое debuild?

Ниже приведён обзор команд, похожих на команду **debuild**.

- Файл **debian/rules** определяет то, как будет собран двоичный пакет Debian.
- **dpkg-buildpackage** — официальная команда для сборки двоичного пакета Debian. Для обычной двоичной сборки она, грубо говоря, выполняет следующую последовательность команд:
 - “**dpkg-source --before-build**” (apply Debian patches, unless they are already applied)
 - «**fakeroot debian/rules clean**»
 - «**dpkg-source --build**» (сборка пакета Debian с исходным кодом)
 - «**fakeroot debian/rules build**»
 - «**fakeroot debian/rules binary**»
 - «**dpkg-genbuildinfo**» (создание файла ***.buildinfo**)
 - «**dpkg-genchanges**» (создание файла ***.changes**)
 - «**fakeroot debian/rules clean**»
 - “**dpkg-source --after-build**” (unapply Debian patches, if they are applied during **--before-build**)
 - «**debsign**» (подписывание файлов ***.dsc** и ***.changes**)
 - * Если вы следовали инструкциям (см. Section 3.5) и передали программе сборки опции **-us** и **-uc**, то данный шаг будет пропущен, а для подписи требуется вручную запустить команду **debsign**.
- Команда **debuild** представляет собой обёртку для команды **dpkg-buildpackage**, которая собирает двоичный пакет Debian в окружении с подходящими значениями переменных окружения.
- The **sbuild** command is a wrapper script to build the Debian binary package under the proper chroot environment with the proper environment variables.

Note



Подробную информацию см. в **dpkg-buildpackage(1)**.

4.5 Шаг 1: получение исходного кода основной ветки разработки

Получим исходный код основной ветки разработки.

Скачаем файл **debhello-0.0.tar.gz**

```
$ wget http://www.example.org/download/debhello-0.0.tar.gz
...
$ tar -xzf debhello-0.0.tar.gz
$ tree
.
+-- debhello-0.0
|   +-- LICENSE
|   +-- Makefile
|   +-- src
|       +-- hello.c
+-- debhello-0.0.tar.gz

2 directories, 4 files
```

В нём содержится исходный код на языке C, **hello.c**, довольно простой.

hello.c

```
$ cat debhello-0.0/src/hello.c
#include <stdio.h>
int
main()
{
    printf("Hello, world!\n");
    return 0;
}
```

Итак, **Makefile** соответствует [Стандартам написания кода GNU](#) и [Стандарту иерархии файловой системы](#). А именно:

- сборку двоичных файлов с учётом значений **\$(CPPFLAGS)**, **\$(CFLAGS)**, **\$(LDFLAGS)** и т. д.
- установку файлов с учётом **\$(DESTDIR)** в качестве целевого системного образа
- установку файлов с **\$(prefix)**, который можно изменить на **/usr**

Makefile

```
$ cat debhello-0.0/Makefile
prefix = /usr/local

all: src/hello

src/hello: src/hello.c
    @echo "CFLAGS=$(CFLAGS)" | \
        fold -s -w 70 | \
        sed -e 's/^/# /'
    $(CC) $(CPPFLAGS) $(CFLAGS) $(LDCFLAGS) -o $@ $^

install: src/hello
    install -D src/hello \
        $(DESTDIR)$(prefix)/bin/hello

clean:
    -rm -f src/hello

distclean: clean

uninstall:
    -rm -f $(DESTDIR)$(prefix)/bin/hello

.PHONY: all install clean distclean uninstall
```

Note



В приведённом ниже примере применение команды **echo** к **\$(CFLAGS)** используется для проверки настройки сборочных флагов.

4.6 Шаг 2: создание шаблонных файлов с помощью debmake

Tip



Если команда **debmake** вызвана с опцией **-T**, то в шаблонные файлы будут добавлены более подробные комментарии.

Вывод команды **debmake** довольно подробен, в нём объяснены выполняемые действия, например, как это указано ниже.

```
$ cd debhello-0.0
$ debmake
I: set parameters
I: =====
I: package_dir      = /usr/lib/python3/dist-packages
I: base_path        = /usr
I: base_lib_path    = /usr/lib/debmake
I: base_share_path  = /usr/share/debmake
I: =====
I: sanity check of parameters
I: pkg="debhello", ver="0.0", rev="1"
I: *** start packaging in "debhello-0.0". ***
I: provide debhello_0.0.orig.tar.gz for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-0.0.tar.gz debhello_0.0.orig.tar.gz
I: pwd = "/path/to/debhello-0.0"
I: parse binary package settings:
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: analyze the source tree
I: build_type = make
I: scan source for copyright+license text and file extensions
I: 100 %, ext = c
I: check_all_licenses
I: ..
I: check_all_licenses completed for 2 files.
I: bunch_all_licenses
I: format_all_licenses
I: make debian/* template files
I: single binary package
I: debmake -x "1" ...
I: creating => debian/control
I: creating => debian/copyright
I: substituting => /usr/share/debmake/extra0/changelog
I: creating => debian/changelog
I: substituting => /usr/share/debmake/extra0/rules
I: creating => debian/rules
I: substituting => /usr/share/debmake/extra1/README.Debian
I: creating => debian/README.Debian
I: substituting => /usr/share/debmake/extra1/watch
I: creating => debian/watch
I: substituting => /usr/share/debmake/extra1source/format
I: creating => debian/source/format
I: substituting => /usr/share/debmake/extra1tests/control
I: creating => debian/source/control
I: substituting => /usr/share/debmake/extra1upstream/metadata
I: creating => debian/upstream/metadata
I: substituting => /usr/share/debmake/extra1tests/control
I: creating => debian/tests/control
I: substituting => /usr/share/debmake/extra1patches/series
I: creating => debian/patches/series
I: substituting => /usr/share/debmake/extra1sourcecx/local-options
I: creating => debian/source/local-options
I: substituting => /usr/share/debmake/extra1sourcecx/options
I: creating => debian/source/options
```

```
I: substituting => /usr/share/debmake/extra1sourcecx/patch-header
I: creating => debian/source/patch-header
I: run "debmake -x2" to get more template files
I: $ wrap-and-sort
```

Команда **debmake** создаёт все шаблонные файлы на основе опций командной строки. Поскольку никакие опции не были переданы, команда **debmake** выбирает для вас разумные значения по умолчанию:

- Имя пакета с исходным кодом: **debhello**
- Версия основной ветки разработки: **0.0**
- Имя двоичного пакета: **debhello**
- Номер редакции Debian: **1**
- Тип пакета: **bin** (пакет с двоичными исполняемыми файлами формата ELF)
- Опция **-x**: **-x1** (используется по умолчанию для случая с одним двоичным пакетом)

Проверим созданные шаблонные файлы.

Дерево исходного кода после простого выполнения debmake.

```
$ cd ..
$ tree
.
+-- debhello-0.0
|   +-- LICENSE
|   +-- Makefile
|   +-- debian
|       |   +-- README.Debian
|       |   +-- changelog
|       |   +-- control
|       |   +-- copyright
|       |   +-- patches
|       |       |   +-- series
|       |   +-- rules
|       |   +-- source
|       |       |   +-- control
|       |       |   +-- format
|       |       |   +-- local-options
|       |       |   +-- options
|       |       |   +-- patch-header
|       |   +-- tests
|       |       |   +-- control
|       |   +-- upstream
|       |       |   +-- metadata
|       |   +-- watch
|       +-- src
|           +-- hello.c
+-- debhello-0.0.tar.gz
+-- debhello_0.0.orig.tar.gz -> debhello-0.0.tar.gz

7 directories, 19 files
```

Файл **debian/rules** является сборочным сценарием, предоставляемым сопровождающим пакета. Ниже приводится его шаблонный файл, созданный командой **debmake**.

debian/rules (шаблонный файл):

```
$ cat debhello-0.0/debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed
```

```

%:
    dh $@

#override_dh_auto_install:
#    dh_auto_install -- prefix=/usr

#override_dh_install:
#    dh_install --list-missing -X.pyс -X.pyо

```

По сути, это стандартный файл **debian/rules** с командой **dh**. (Для удобства настройки в нём содержится несколько закомментированных строк.)

Файл **debian/control** предоставляет основные метаданные пакета Debian. Ниже приведён шаблонный файл, созданный командой **debmake**.

debian/control (шаблонный файл):

```

$ cat debhello-0.0/debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Firstname Lastname" <email.address@example.org>
Build-Depends: debhelper-compat (= 13)
Standards-Version: 4.5.1
Homepage: <insert the upstream URL, if relevant>
Rules-Requires-Root: no

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends: ${misc:Depends}, ${shlibs:Depends}
Description: auto-generated package by debmake
 This Debian binary package was auto-generated by the
 debmake(1) command provided by the debmake package.

```

Warning



Если вы оставите «**Section: unknown**» в шаблонном файле **debian/control** без изменений, то ошибка **lintian** может прервать сборку.

Поскольку это пакет с двоичными исполняемыми файлами в формате ELF, команда **debmake** устанавливает «**Architecture: any**» и «**Multi-Arch: foreign**». Кроме того, она устанавливает требуемые параметры **переменных подстановки** следующим образом: «**Depends: \${shlibs:Depends}, \${misc:Depends}**». Для объяснения см. Chapter 5.

Note



Please note this **debian/control** file uses the RFC-822 style as documented in [5.2 Source package control files — debian/control](#) of the “Debian Policy Manual”. The use of the empty line and the leading space are significant.

Файл **debian/copyright** предоставляет данные об авторском праве на пакет Debian. Ниже приведён шаблонный файл, созданный командой **debmake**.

debian/copyright (шаблонный файл):

```

$ cat debhello-0.0/debian/copyright
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: debhello
Upstream-Contact: <preferred name and address to reach the upstream project>

```

```

Source: <url://example.com>
#
# Please double check copyright with the licensecheck(1) command.

Files:      Makefile
           src/hello.c
Copyright:  __NO_COPYRIGHT_NOR_LICENSE__
License:    __NO_COPYRIGHT_NOR_LICENSE__

#-----
# Files marked as NO_LICENSE_TEXT_FOUND may be covered by the following
# license/copyright files.
#-----
# License file: LICENSE
License:
.
All files in this archive are licensed under the MIT License as below.
.
Copyright 2015 Osamu Aoki <osamu@debian.org>
.
Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the "Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following conditions:
.
The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.
.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```

4.7 Шаг 3: изменение шаблонных файлов

От сопровождающего требуется вручную внести некоторые изменения шаблонных файлов.

Для того, чтобы установить файлы в качестве части системных файлов, текущее значение **\$(prefix)**, **/usr/local**, в **Makefile** должно быть заменено на **/usr**. Это можно сделать в файле **debian/rules** с помощью цели **override_dh_auto_install**, передав значение **<prefix=/usr>**.

debian/rules (версия сопровождающего):

```

$ vim debhello-0.0/debian/rules
... hack, hack, hack, ...
$ cat debhello-0.0/debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
    dh $@

override_dh_auto_install:
    dh_auto_install -- prefix=/usr

```

Экспортирование переменных окружения **DH_VERBOSE** в файле **debian/rules**, как это сделано выше, приводит к тому, что инструмент **debhelper** создаёт более подробный отчёт о сборке.

Exporting **DEB_BUILD_MAINT_OPTION** as above sets the hardening options as described in the “FEATURE AREAS/ENVIRONMENT” in **dpkg-buildflags(1)**. ¹

Exporting **DEB_CFLAGS_MAINT_APPEND** as above forces the C compiler to emit all the warnings.

Exporting **DEB_LDFLAGS_MAINT_APPEND** as above forces the linker to link only when the library is actually needed. ²

The **dh_auto_install** command for the Makefile based build system essentially runs “**\$(MAKE) install DESTDIR=debian/debhello**”. The creation of this **override_dh_auto_install** target changes its behavior to “**\$(MAKE) install DESTDIR=debian/debhello prefix=/usr**”.

Here are the maintainer versions of the **debian/control** and **debian/copyright** files.

debian/control (версия сопровождающего):

```
$ vim debhello-0.0/debian/control
... hack, hack, hack, ...
$ cat debhello-0.0/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13)
Standards-Version: 4.5.1
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends: ${misc:Depends}, ${shlibs:Depends}
Description: Simple packaging example for debmake
 This Debian binary package is an example package.
 (This is an example only)
```

debian/copyright (версия сопровождающего):

```
$ vim debhello-0.0/debian/copyright
... hack, hack, hack, ...
$ cat debhello-0.0/debian/copyright
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: debhello
Upstream-Contact: Osamu Aoki <osamu@debian.org>
Source: https://salsa.debian.org/debian/debmake-doc

Files:      *
Copyright:  2015-2021 Osamu Aoki <osamu@debian.org>
License:    Expat
Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the "Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following conditions:
.
The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.
.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
```

¹This is a cliché to force a read-only relocation link for the hardening and to prevent the lintian warning “**W: debhello: hardening-no-relro usr/bin/hello**”. This is not really needed for this example but should be harmless. The lintian tool seems to produce a false positive warning for this case which has no linked library.

²This is a cliché to prevent overlinking for the complex library dependency case such as Gnome programs. This is not really needed for this simple example but should be harmless.

IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

В каталоге **debian/** имеются и другие шаблонные файлы. Их также следует обновить.
Шаблонные файлы в debian/. (v=0.0):

```
$ tree debhelloworld-0.0/debian
debhelloworld-0.0/debian
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- patches
|   +-- series
+-- rules
+-- source
|   +-- control
|   +-- format
|   +-- local-options
|   +-- options
|   +-- patch-header
+-- tests
|   +-- control
+-- upstream
|   +-- metadata
+-- watch

4 directories, 14 files
```

Tip



Configuration files used by the **dh_*** commands from the **debhelper** package usually treat **#** as the start of a comment line.

4.8 Шаг 4: сборка пакета с помощью debuild

В данном дереве исходного кода вы можете создать неродной пакет Debian с помощью команды **debuild** или эквивалентных ей команд (см. Section 4.4). Вывод команды очень подробен, выполняемые действия объясняются в нём следующим образом.

```
$ cd debhelloworld-0.0
$ debuild
dpkg-buildpackage -us -uc -ui -i -i
...
debian/rules clean
dh clean
...
debian/rules binary
dh binary
  dh_update_autotools_config
  dh_autoreconf
  dh_auto_configure
    install -d /path/to/debhelloworld-0.0/debian/.debhelper/generated/_source/...
  dh_auto_build
    make -j12 "INSTALL=install --strip-program=true"
make[1]: Entering directory '/path/to/debhelloworld-0.0'
# CFLAGS=-g -O2
```

```
# -ffile-prefix-map=/home/osamu/src/public/debmake-doc/debmake-doc/examp
...
Now running lintian -i -I --show-overrides debhello_0.0-1_amd64.changes ...
...
N:   Renamed from: binary-without-manpage
N:
W: debhello: readme-debian-contains-debmake-template
N:
...
```

You can verify that **CFLAGS** is updated properly with **-Wall** and **-pedantic** by the **DEB_CFLAGS_MAINT_APPEND** variable.

The manpage should be added to the package as reported by the **lintian** package, as shown in later examples (see Chapter 8). Let's move on for now.

Проверим результат сборки.

Файлы debhello версии 0.0, созданные с помощью команды debuild:

```
$ cd ..
$ tree -FL 1
.
+-- debhello-0.0/
+-- debhello-0.0.tar.gz
+-- debhello-dbgsym_0.0-1_amd64.deb
+-- debhello_0.0-1.debian.tar.xz
+-- debhello_0.0-1.dsc
+-- debhello_0.0-1_amd64.build
+-- debhello_0.0-1_amd64.buildinfo
+-- debhello_0.0-1_amd64.changes
+-- debhello_0.0-1_amd64.deb
+-- debhello_0.0.orig.tar.gz -> debhello-0.0.tar.gz

1 directory, 9 files
```

Вы видите все созданные файлы.

- **debhello_0.0.orig.tar.gz** представляет собой символическую ссылку на tar-архив основной ветки разработки.
- **debhello_0.0-1.debian.tar.xz** содержит файлы, созданные сопровождающим.
- **debhello_0.0-1.dsc** представляет собой файл с метаданными для пакета Debian с исходным кодом.
- **debhello_0.0-1_amd64.deb** — двоичный пакет Debian.
- The **debhello-dbgsym_0.0-1_amd64.deb** is the Debian debug symbol binary package. See Section 5.19.1.
- The **debhello_0.0-1_amd64.build** file is the build log file.
- The **debhello_0.0-1_amd64.buildinfo** file is the meta data file generated by **dpkg-genbuildinfo(1)**.
- **debhello_0.0-1_amd64.changes** — файл с метаданными для двоичного пакета Debian.

debhello_0.0-1.debian.tar.xz содержит изменения Debian, внесённые в исходный код основной ветки разработки. Содержимое этого файла приведено ниже.

Содержимое архива debhello_0.0-1.debian.tar.xz:

```
$ tar -tzf debhello-0.0.tar.gz
debhello-0.0/
debhello-0.0/src/
debhello-0.0/src/hello.c
debhello-0.0/LICENSE
debhello-0.0/Makefile
$ tar --xz -tf debhello_0.0-1.debian.tar.xz
debian/
debian/README.Debian
debian/changelog
debian/control
```

```

debian/copyright
debian/patches/
debian/patches/series
debian/rules
debian/source/
debian/source/control
debian/source/format
debian/source/options
debian/source/patch-header
debian/tests/
debian/tests/control
debian/upstream/
debian/upstream/metadata
debian/watch

```

The **debhello_0.0-1_amd64.deb** contains the binary files to be installed to the target system.

The **debhello-debsym_0.0-1_amd64.deb** contains the debug symbol files to be installed to the target system..

The binary package contents of all binary packages:

```

$ dpkg -c debhhello-dbgSYM_0.0-1_amd64.deb
drwxr-xr-x root/root ... ./
drwxr-xr-x root/root ... ./usr/
drwxr-xr-x root/root ... ./usr/lib/
drwxr-xr-x root/root ... ./usr/lib/debug/
drwxr-xr-x root/root ... ./usr/lib/debug/.build-id/
drwxr-xr-x root/root ... ./usr/lib/debug/.build-id/be/
-rw-r--r-- root/root ... ./usr/lib/debug/.build-id/be/11292eded3fc22396a0b62...
drwxr-xr-x root/root ... ./usr/share/
drwxr-xr-x root/root ... ./usr/share/doc/
lrwxrwxrwx root/root ... ./usr/share/doc/debhhello-dbgSYM -> debhhello
$ dpkg -c debhhello_0.0-1_amd64.deb
drwxr-xr-x root/root ... ./
drwxr-xr-x root/root ... ./usr/
drwxr-xr-x root/root ... ./usr/bin/
-rwxr-xr-x root/root ... ./usr/bin/hello
drwxr-xr-x root/root ... ./usr/share/
drwxr-xr-x root/root ... ./usr/share/doc/
drwxr-xr-x root/root ... ./usr/share/doc/debhhello/
-rw-r--r-- root/root ... ./usr/share/doc/debhhello/README.Debian
-rw-r--r-- root/root ... ./usr/share/doc/debhhello/changelog.Debian.gz
-rw-r--r-- root/root ... ./usr/share/doc/debhhello/copyright

```

The generated dependency list of all binary packages.

The generated dependency list of all binary packages (v=0.0):

```

$ dpkg -f debhhello-dbgSYM_0.0-1_amd64.deb pre-depends \
depends recommends conflicts breaks
Depends: debhhello (= 0.0-1)
$ dpkg -f debhhello_0.0-1_amd64.deb pre-depends \
depends recommends conflicts breaks
Depends: libc6 (>= 2.2.5)

```

Caution



Many more details need to be addressed before uploading the package to the Debian archive.

Note



Если вы пропустили ручную настройку автоматически созданных командой **debmake** файлов настройки, то у созданного двоичного пакета может отсутствовать понятное другим описание пакета, а также пакет может несоответствовать некоторым требованиям политики. Такой сырой пакет вполне хорошо работает, если передать его команде **dpkg**, и может оказаться вполне достаточным для его локального развёртывания.

4.9 Шаг 3 (альтернативный): изменение исходного кода основной ветки разработки

В примере выше при создании пакета Debian исходный код основной ветки разработки не был изменён.

Альтернативный подход состоит в том, что сопровождающий изменяет исходный код основной ветки, меняя файл **Makefile** так, чтобы значением $\$(prefix)$ было **/usr**.

Фактически процесс создания пакета в этом случае совпадает с тем, что описан выше в Section 4.7 за исключением двух моментов:

- Store the maintainer modifications to the upstream source as the corresponding patch files in the **debian/patches/** directory and list their filenames in the **debian/patches/series** file as indicated in Section 5.10. There are several ways to generate patch files. A few examples are given in these sections:
 - Section 4.9.1
 - Section 4.9.2
 - Section 4.9.3
- В изменениях, внесённых сопровождающим в файл **debian/rules**, отсутствует цель **override_dh_auto_install**, то есть:

debian/rules (альтернативная версия сопровождающего):

```
$ cd debhello-0.0
$ vim debian/rules
... hack, hack, hack, ...
$ cat debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
    dh $@
```

This alternative approach to Debian packaging using a series of patch files may be less robust for future upstream changes but more flexible coping with the difficult upstream source. (See Section 7.13.)

Note



Для этого конкретного случая приведённый выше Section 4.7, использующий файл **debian/rules**, является более подходящим. Тем не менее, продолжим работу с текущим подходом с целью обучения.

Tip



В случае более сложных пакетов требуется использовать оба подхода, и Section 4.7, и Section 4.9.

4.9.1 Создание заплата с помощью diff -u

Ниже приводится пример создания **000-prefix-usr.patch** с помощью команды **diff**.

```
$ cp -a debhello-0.0 debhello-0.0.orig
$ vim debhello-0.0/Makefile
... hack, hack, hack, ...
$ diff -Nru debhello-0.0.orig debhello-0.0 >000-prefix-usr.patch
$ cat 000-prefix-usr.patch
diff -Nru debhello-0.0.orig/Makefile debhello-0.0/Makefile
--- debhello-0.0.orig/Makefile    2021-07-02 16:26:38.734722687 +0900
+++ debhello-0.0/Makefile        2021-07-02 16:26:38.802723496 +0900
@@ -1,4 +1,4 @@
-prefix = /usr/local
+prefix = /usr

all: src/hello

$ rm -rf debhello-0.0
$ mv -f debhello-0.0.orig debhello-0.0
```

Заметьте, что дерево исходного кода основной ветки разработки восстанавливается к изначальному состоянию, а файл заплата доступен как **000-prefix-usr.patch**.

Этот **000-prefix-usr.patch** редактируется так, чтобы он соответствовал [DEP-3](#), а также перемещается в соответствующий каталог, как это указано ниже.

```
$ cd debhello-0.0
$ echo '000-prefix-usr.patch' >debian/patches/series
$ vim ../000-prefix-usr.patch
... hack, hack, hack, ...
$ mv -f ../000-prefix-usr.patch debian/patches/000-prefix-usr.patch
$ cat debian/patches/000-prefix-usr.patch
From: Osamu Aoki <osamu@debian.org>
Description: set prefix=/usr patch
diff -Nru debhello-0.0.orig/Makefile debhello-0.0/Makefile
--- debhello-0.0.orig/Makefile
+++ debhello-0.0/Makefile
@@ -1,4 +1,4 @@
-prefix = /usr/local
+prefix = /usr

all: src/hello
```

4.9.2 Создание заплата с помощью dquilt

Ниже приводится пример создания **000-prefix-usr.patch** с помощью команды **dquilt**, которая является простой обёрткой для программы **quilt**. Синтаксис и функции команды **dquilt** совпадают с синтаксисом и функциями команды **quilt(1)** за тем лишь исключением, что заплата сохраняется в каталог **debian/patches/**.

```
$ cd debhello-0.0
$ dquilt new 000-prefix-usr.patch
Patch debian/patches/000-prefix-usr.patch is now on top
$ dquilt add Makefile
File Makefile added to patch debian/patches/000-prefix-usr.patch
... hack, hack, hack, ...
```

```

$ head -1 Makefile
prefix = /usr
$ dquilt refresh
Refreshed patch debian/patches/000-prefix-usr.patch
$ dquilt header -e --dep3
... edit the DEP-3 patch header with editor
$ tree -a
.
+-- .pc
|   +-- .quilt_patches
|   +-- .quilt_series
|   +-- .version
|   +-- 000-prefix-usr.patch
|       |   +-- .timestamp
|       |   +-- Makefile
|   +-- applied-patches
+-- LICENSE
+-- Makefile
+-- debian
|   +-- README.Debian
|   +-- changelog
|   +-- control
|   +-- copyright
|   +-- patches
|       |   +-- 000-prefix-usr.patch
|       |   +-- series
|   +-- rules
|   +-- source
|       |   +-- control
|       |   +-- format
|       |   +-- local-options
|       |   +-- options
|       |   +-- patch-header
|   +-- tests
|       |   +-- control
|   +-- upstream
|       |   +-- metadata
|   +-- watch
+-- src
    +-- hello.c

8 directories, 24 files
$ cat debian/patches/series
000-prefix-usr.patch
$ cat debian/patches/000-prefix-usr.patch
Description: set prefix=/usr patch
Author: Osamu Aoki <osamu@debian.org>
Index: debhello-0.0/Makefile
=====
--- debhello-0.0.orig/Makefile
+++ debhello-0.0/Makefile
@@ -1,4 +1,4 @@
-prefix = /usr/local
+prefix = /usr

all: src/hello

```

Here, **Makefile** in the upstream source tree doesn't need to be restored to the original state. The **dpkg-source** command invoked by the Debian packaging procedure in Section 4.8, understands the patch application state recorded by the **dquilt** program in the **.pc/** directory. As long as all the changes are committed by the **dquilt** command, the Debian source package can be built from the modified source tree.

Note



If the `.pc/` directory is missing, the `dpkg-source` command assumes that no patch was applied. That's why the more primitive patch generation methods like in Section 4.9.1 without generating the `.pc/` directory require the upstream source tree to be restored.

4.9.3 Создание заплаты с помощью `dpkg-source --commit`

Ниже приводится пример создания `000-prefix-usr.patch` с помощью команды «`dpkg-source --commit`».

Отредактируем исходный код основной ветки разработки.

```
$ cd debhello-0.0
$ vim Makefile
... hack, hack, hack, ...
$ head -n1 Makefile
prefix = /usr
```

Сохраним изменения в файл заплаты.

```
$ dpkg-source --commit . 000-prefix-usr.patch
... editor to edit the DEP-3 patch header
...
```

Посмотрим результат.

```
$ cat debian/patches/series
000-prefix-usr.patch
$ cat debian/patches/000-prefix-usr.patch
Description: set prefix=/usr patch
Author: Osamu Aoki <osamu@debian.org>
Index: debhello-0.0/Makefile

--- debhello-0.0.orig/Makefile
+++ debhello-0.0/Makefile
@@ -1,4 +1,4 @@
-prefix = /usr/local
+prefix = /usr

all: src/hello

$ tree -a
.
+-- .pc
|   +-- .quilt_patches
|   +-- .quilt_series
|   +-- .version
|   +-- 000-prefix-usr.patch
|       |   +-- .timestamp
|       |   +-- Makefile
|   +-- applied-patches
+-- LICENSE
+-- Makefile
+-- debian
|   +-- README.Debian
|   +-- changelog
|   +-- control
|   +-- copyright
|   +-- patches
|       |   +-- 000-prefix-usr.patch
|       |   +-- series
|   +-- rules
+-- source
```

```
| | +-- control
| | +-- format
| | +-- local-options
| | +-- options
| | +-- patch-header
| +-- tests
| | +-- control
| +-- upstream
| | +-- metadata
| +-- watch
+-- src
  +-- hello.c
```

```
8 directories, 24 files
```

Итак, команда **dpkg-source** выполняет в точности то же, что и было выполнено командой **dquilt** в Section [4.9.2](#).

Chapter 5

ОСНОВЫ

A broad overview is presented here for the basic rules of Debian packaging focusing on the non-native Debian package in the “3.0 (quilt)” format.

Note



Для ясности в дальнейшем были умышленно опущены некоторые детали. Ознакомьтесь со страницами руководства **dpkg-source(1)**, **dpkg-buildpackage(1)**, **dpkg(1)**, **dpkg-deb(1)**, **deb(5)** и др.

Пакет Debian с исходным кодом является набором входных файлов, используемых для сборки двоичного пакета Debian, и не представляет собой только один файл.

The Debian binary package is a special archive file which holds a set of installable binary data with its associated information.

Один пакет Debian с исходным кодом может использоваться для создания нескольких двоичных пакетов Debian, определяемых в файле **debian/control**.

Перодной пакет Debian в формате «3.0 (quilt)» является наиболее обычным форматом пакетов Debian с исходным кодом.

Note



Существует множество обёрточных сценариев. Используйте их для упрощения вашей работы, но обязательно разберитесь с основами их внутреннего устройства.

5.1 Работа по созданию пакета

The Debian packaging workflow to create a Debian binary package involves generating several specifically named files (see Section 5.4) as defined in the “Debian Policy Manual”.

The oversimplified method for the Debian packaging workflow can be summarized in 10 steps as follows.

1. Загружается tar-архив основной ветки разработки в виде файла *пакет-версия.tar.gz*.
2. Этот архив распаковывается, создаётся множество файлов в каталоге *пакет-версия/*.
3. Архив основной ветки разработки копируется (или создаётся символическая ссылка на него) в файл со специальным именем *имяпакета_версия.orig.tar.gz*.
 - символ, разделяющий *пакет* и *версию*, заменяется с - (дефиса) на _ (подчёркивание)
 - к расширению добавляется **.orig**.
4. К исходному коду основной ветки разработки в каталог *пакет-версия/debian/* добавляются файлы спецификации пакета Debian.

- Обязательные файлы спецификации в каталоге **debian/***:
 - debian/rules** Исполняемый сценарий для сборки пакета Debian (см. Section 5.6)
 - debian/control** The package configuration file containing the source package name, the source build dependencies, the binary package name, the binary dependencies, etc. (see Section 5.7)
 - debian/changelog** Файл с историей пакета Debian, определяющий в первой строке версию пакета из основной ветки разработки и номер редакции Debian (см. Section 5.8)
 - debian/copyright** Информация об авторских правах и лицензии (см. Section 5.9)
 - Необязательные файлы спецификации в каталоге **debian/*** (see Section 5.13):
 - Команда **debmake**, запущенная в каталоге *пакет-версия/*, создаёт изначальный набор шаблонных файлов настройки.
 - Обязательные файлы спецификации создаются даже при использовании опции **-x0**.
 - Команда **debmake** не перезаписывает какие-либо существующие файлы настройки.
 - Эти файлы должны быть отредактированы вручную с целью доведения их до совершенства в соответствии с «Руководством по политике Debian» и «Справочником разработчика Debian».
5. The **dpkg-buildpackage** command (usually from its wrapper **debuild** or **sbuild**) is invoked in the *package-version/* directory to make the Debian source and binary packages by invoking the **debian/rules** script.
- Текущий каталог устанавливается следующим образом: **\$(CURDIR)=/путь/до/пакет-версия/**
 - Создание пакета Debian с исходным кодом в формате «3.0 (quilt)» с помощью **dpkg-source(1)**
 - *пакет_версия.orig.tar.gz* (копия или символическая ссылка на *пакет-версия.tar.gz*)
 - *пакет_версия-редакция.debian.tar.xz* (tar-архив каталога *пакет-версия/debian/**)
 - *пакет_версия-редакция.dsc*
 - Сборка исходного кода с помощью «**debian/rules build**» в **\$(DESTDIR)**
 - **DESTDIR=debian/binarypackage/** (один двоичный пакет)
 - **DESTDIR=debian/tmp/** (несколько двоичных пакетов)
 - Создание двоичного пакета Debian с помощью **dpkg-deb(1)**, **dpkg-genbuildinfo(1)** и **dpkg-genchanges(1)**.
 - *двоичныйпакет_версия-редакция_архитектура.deb*
 - ... (Могут существовать несколько файлов с двоичными пакетами Debian.)
 - *пакет_версия-редакция_архитектура.changes*
 - *package_version-revision_arch.buildinfo*
6. Проверка качества пакета Debian с помощью команды **lintian**. (рекомендуется)
- Follow the rejection guidelines from [ftp-master](#).
 - [REJECT-FAQ](#)
 - [Лист проверок для пакетов из NEW](#)
 - [Автоматические отклонения пакетов Lintian \(список тегов lintian\)](#)
7. Test the goodness of the generated Debian binary package manually by installing it and running its programs.
8. After confirming the goodness, prepare files for the normal source-only upload to the Debian archive.
9. Sign the *package_version-revision.dsc* and 'package_version-revision'_*source.changes* files with the **deb-sign** command using your private GPG key.
10. Upload the set of the Debian source package files with the **dput** command to the Debian archive.
- Test building and confirming of the binary package goodness as above is the moral obligation as a diligent Debian developer but there is no physical barrier for people to skip such operations at this moment for the source-only upload.
- Under some exceptional situation such as NEW uploads, uploads to the Debian archive may need to include Debian binary package files. For such situation, sign *package_version-revision_arch.changes* instead of 'package_version-revision'_*source.changes* in the step 9, and upload the set of the Debian source and binary package files in the step 10.
- Теперь замените каждую часть имени файла.

- часть *пакет* на имя пакета Debian с исходным кодом
- часть *двоичный пакет* на имя двоичного пакета Debian
- часть *версия* на версию основной ветки разработки
- часть *редакция* на номер редакции Debian
- часть *архитектура* на архитектуру пакета

See also [Source-only uploads](#).

Tip



Используется множество различных стратегий по управлению заплатками и использованию систем управления версиями. Вам не следует использовать все из них.

Tip



There is very extensive documentation in [Chapter 6. Best Packaging Practices](#) in the “Debian Developer’s Reference”. Please read it.

5.1.1 Пакет `debhelper`

Although a Debian package can be made by writing a `debian/rules` script without using the `debhelper` package, it is impractical to do so. There are too many modern “Policy” required features to be addressed, such as application of the proper file permissions, use of the proper architecture dependent library installation path, insertion of the installation hook scripts, generation of the debug symbol package, generation of package dependency information, generation of the package information files, application of the proper timestamp for reproducible build, etc.

`Debhelper` package provides a set of useful scripts in order to simplify Debian’s packaging workflow and reduce the burden of package maintainers. When properly used, they will help packagers handle and implement “Policy” required features automatically.

Процедура создания пакета Debian в современном стиле может быть организована в виде набора простых модульных действий:

- using the `dh` command to invoke many utility scripts automatically from the `debhelper` package, and
- настройка их поведения с помощью декларативных файлов настройки в каталоге `debian/`.

You should almost always use `debhelper` as your package’s build dependency. This document also assumes that you are using a fairly contemporary version of `debhelper` to handle packaging works in the following contents.

5.2 Historical perspective of Debian packaging practices

Let me oversimplify historical perspective of Debian packaging practices.

[Debian was started in 1990s](#) when upstream packages were available from public FTP sites such as [Sunsite](#). In those early days, Debian packaging used `dpkg-source` currently known as “Format : 1.0”:

- Upstream released packages in `tar.gz` format. They were obtained from some FTP sites.
- Debian applied “one big patch” as its source distribution and made its binary distribution form it.
- Many different approaches were adopted by different Debian developers to manage topic patches with custom scripts within the “one big patch”. — Chaos!

In order to address issues of old `dpkg-source` “Format : 1.0”, new `dpkg-source` “Format : 3.0 (quilt)” was invented around 2008:

- Upstream released packages in tar.gz/tar.bz2/tar.xz format optionally with signature files. They were obtained from some HTTP sites.
- Most Debian developers adopted patch queue mechanism of "Format: 3.0 (quilt)" to manage topic patches and made Debian packages while keeping files outside of `debian/` directory untouched.
- VCS, especially git, became popular and most Debian developers and started to record this packaging practice in `gbp-buildpackage(1)` style.
- The `gbp-buildpackage(1)` workflow records the exact same content of the upstream tarball to VCS for source files outside of `debian/` directory (= patch-unapplied).

The use of Git repositories to distribute upstream packages with signed tags (supported feature since 2011) became very popular.

- The tarball centric `gbp-buildpackage(1)` workflow to record changes to VCS was cumbersome for some Debian developers and `dgit(1)` was invented in 2013.
- The Git centric `dgit-maint-debrebase(7)` and `dgit-maint-merge(7)` workflows to record changes to VCS are gaining popularity among these Debian developers.
- Source files recorded to VCS by `dgit-maint-debrebase(7)` and `git-maint-merge(7)` are modified upstream source files (= patch-applied).
- Uploaded source packages by `dgit-maint-debrebase(7)` and `git-maint-merge(7)` workflows still use `dpkg-source "Format: 3.0 (quilt)"`.

Debian also enforced the source-only upload when developing Debian/11 Bullseye (released in 2021).

In this tutorial, mostly simple tarball based `dpkg-source "Format: 3.0 (quilt)"` examples are presented as an introductory purpose.

Please assess these VCS usage approaches by yourself later to decide which one to deploy as your preferred one.

Please also read [Notes on Debian](#) by Russ Allbery which have best practices such as [Using Git for Debian Packaging](#).

5.3 Future perspective on Debian packaging practices

Please look around to understand how Debian packaging practices are evolving and follow the current general trends if possible.

- [DEP - Debian Enhancement Proposals](#)
 - Debian Enhancement Proposals (or DEPs, for short) offer a device to organize discussions about various kinds of enhancements in the Debian project, reflect their current status and, in particular, archive their outcomes.
- [Debian Trends](#)
 - This page provides some historical perspective about how those practices evolved.
- [Debian git packaging maintainer branch formats and workflows](#)
 - Nice packaging summary by the latest `dgit` tool providers.

You can also search entire Debian source code by yourself, too.

- [Debian Sources](#) — code search tool
 - [Debian Code Search](#) — wiki page describing its usage
- [Debian Code Search](#) — another code search tool

5.4 Имя пакета и версия

Если исходный код основной ветки разработки поставляется в виде архива **hello-0.9.12.tar.gz**, можно использовать **hello** в качестве имени пакета с исходным кодом основной ветки разработки, а **0.9.12** — в качестве версии основной ветки.

Утилита **debmake** предназначена для создания шаблонных файлов в помощь сопровождающему пакету. Строки с комментариями начинаются с символа **#** и содержат обучающий текст. Вам следует удалить или отредактировать строки с комментариями до выполнения загрузки пакета в архив Debian.

The license extraction and assignment process involves a lot of heuristics; it may fail in some cases. It is highly recommended to use other tools such as **licensecheck** from the **devscripts** package in conjunction with **debmake**.

There are some limitations for what characters may be used as a part of the Debian package. The most notable limitation is the prohibition of uppercase letters in the package name. Here is a summary as a set of regular expressions:

- Имя пакета основной ветки разработки (**-p**): `[-+.a-z0-9]{2,}`
- Имя двоичного пакета (**-b**): `[-+.a-z0-9]{2,}`
- Версия основной ветки разработки (**-u**): `[0-9][-+.:~a-z0-9A-Z]*`
- Редакция Debian (**-r**): `[0-9][+~a-z0-9A-Z]*`

See the exact definition in [Chapter 5 - Control files and their fields](#) in the “Debian Policy Manual”.

debmake предполагает относительно простые случаи создания пакетов. Поэтому все программы, относящиеся к интерпретатору, считаются «**Architecture: all**». Тем не менее, это не всегда так.

Вам следует соответствующим образом изменить имя пакета и версию основной ветки разработки для создания пакета Debian.

Для того, чтобы информация об имени пакета и номере версии эффективно обрабатывались такими популярными инструментами как команда **aptitude**, рекомендуется, чтобы длина имени пакета была равна 30 символам или была меньше; а общая длина версии и редакции была равна 14 символам или меньше. ¹

Для того, чтобы не возникали конфликты, видимое пользователю имя двоичного пакета не следует выбирать из числа распространённых слов.

If upstream does not use a normal versioning scheme such as **2.30.32** but uses some kind of date such as **11Apr29**, a random codename string, or a VCS hash value as part of the version, make sure to remove them from the upstream version. Such information can be recorded in the **debian/changelog** file. If you need to invent a version string, use the **YYYYMMDD** format such as **20110429** as upstream version. This ensures that the **dpkg** command interprets later versions correctly as upgrades. If you need to ensure a smooth transition to a normal version scheme such as **0.1** in the future, use the **0~YYMMDD** format such as **0~110429** as upstream version, instead.

Строки версий можно сравнивать друг с другом с помощью команды **dpkg** следующим образом.

```
$ dpkg --compare-versions ver1 op ver2
```

Правило сравнения версий может быть представлены следующим образом:

- Строки сравниваются в порядке с начала до конца.
- Буквы больше чисел.
- Числа сравниваются как целые числа.
- Буквы сравниваются в порядке таблицы кодов ASCII.

Также имеются специальные правила для символов точки (**.**), плюса (**+**) и тильды (**~**). Они показаны ниже.

```
0.0 < 0.5 < 0.10 < 0.99 < 1 < 1.0~rc1 < 1.0 < 1.0+b1 < 1.0+nmu1 < 1.1 < 2.0
```

Один сложный случай возникает тогда, когда разработчики основной ветки выпускают **hello-0.9.12-ReleaseCandidate-99.tar.gz** как предварительный выпуск для **hello-0.9.12.tar.gz**. Вам следует гарантировать, что обновление пакета Debian будет происходить правильно, переименовав для этого архив с исходным кодом основной ветки в **hello-0.9.12~rc99.tar.gz**.

¹Для более чем 90% пакетов длина имени пакета равна 24 символам или меньше этого числа; длина версии основной ветки равна 10 символам или меньше, а длина номера редакции Debian равна 3 символам или меньше.

5.5 Родной пакет Debian

Неродной пакет Debian в формате «**3.0 (quilt)**» является самым обычным форматом пакетов Debian с исходным кодом. Файл `debian/source/format` должен содержать строку «**3.0 (quilt)**», что описывается в `dpkg-source(1)`. Описанные выше процедуры создания пакетов и примеры используют этот формат.

A native Debian package is the rare Debian binary package format. It may be used only when the package is useful and valuable only for Debian. Thus, its use is generally discouraged.

Caution



A native Debian package is often accidentally built when its upstream tarball is not accessible from the `dpkg-buildpackage` command with its correct name `package_version.orig.tar.gz`. This is a typical newbie mistake caused by making a symlink name with “-” instead of the correct one with “_”.

A native Debian package has no separation between the **upstream code** and the **Debian changes** and consists only of the following:

- `пакет_версия.tar.gz` (копия или символическая ссылка на `пакет-версия.tar.gz` с файлами `debian/*`.)
- `пакет_версия.dsc`

If you need to create a native Debian package, create it in the “**3.0 (native)**” format using `dpkg-source(1)`.

Tip



Some people promote packaging even programs that have been written only for Debian in the non-native package format. The required tarball without `debian/*` files needs to be manually generated in advance before the standard workflow in Section 5.1. ^a They claim that the use of non-native package format eases communication with the downstream distributions.

^aUse of the “`debmake -t ...`” command or “`git deborig -f HEAD`” can help this workflow. See Section 6.2 and `dgint-maint-merge(7)`.

Tip



При использовании формата родных пакетов нужно заранее создавать tar-архив нет. Родной пакет может быть создан в том случае, если файл `debian/source/format` содержит строку «**3.0 (native)**», в файле `debian/changelog` указана версия без редакции Debian (**1.0** вместо **1.0-1**), а также в дереве исходного кода вызывается команда «`dpkg-source -b .`». Тогда создаётся tar-архив, содержащий исходный код.

5.6 debian/rules

Сценарий `debian/rules` представляет собой исполняемый сценарий для сборки пакета Debian.

- Сценарий `debian/rules` настраивает сборочную систему основной ветки (см. Section 5.18) не установку файлов в `$(DESTDIR)` и формирует архив созданных файлов в виде файла `deb`.
 - Файл `deb` используется для распространения в виде двоичного кода и установки в систему с помощью команды `dpkg`.
- Обычно команда `dh` используется в качестве интерфейса сборочной системы в сценарии `debian/rules`.

- Путь **\$(DESTDIR)** зависит от типа сборки.
 - **\$(DESTDIR)=debian/двоичныйпакет** (один двоичный пакет)
 - **\$(DESTDIR)=debian/tmp** (несколько двоичных пакетов)

5.6.1 dh

The **dh** command from the **debhelper** package with help from its associated packages functions as the wrapper to the typical upstream build systems and offers us uniform access to them by supporting all the Debian policy stipulated targets of the **debian/rules** file.

- **dh clean** : вычищает файлы в дереве исходного кода.
- **dh build** : сборка дерева исходного кода
- **dh build-arch** : сборка зависящих от архитектуры пакетов из дерева исходного кода
- **dh build-indep** : сборка независящих от архитектуры пакетов из дерева исходного кода
- **dh install** : установка двоичных файлов в **\$(DESTDIR)**
- **dh install-arch** : установка двоичных файлов в **\$(DESTDIR)** для зависящих от архитектуры пакетов
- **dh install-indep** : установка двоичных файлов в **\$(DESTDIR)** для независящих от архитектуры пакетов
- **dh binary** : создание файла **deb**
- **dh binary-arch** : создание файла **deb** для зависящих от архитектуры пакетов
- **dh binary-indep** : создание файла **deb** для независящих от архитектуры пакетов

Note



Для пакетов, использующих **debhelper** и имеющих «compat >= 9», команда **dh** экспортирует флаги компилятора (**CFLAGS**, **CXXFLAGS**, **FFLAGS**, **CPPFLAGS** и **LDFLAGS**) со значениями, возвращаемыми командой **dpkg-buildflags** в том случае, если они не были установлены ранее. (Команда **dh** вызывает функцию **set_buildflags**, определённую в модуле **Debian::Debhelper::Dh_Lib**.)

5.6.2 Простой сценарий debian/rules

Благодаря абстракциям команды **dh** ² соответствующий политике Debian файл **debian/rules**, поддерживающий все требуемые цели, можно написать так же просто как **3**:

Простой файл debian/rules:

```
#!/usr/bin/make -f
#export DH_VERBOSE = 1

%:
    dh $@
```

По сути, эта команда **dh** работает как планировщик для вызовов всех требуемых команд **dh_*** в нужный момент.

²This simplicity is available since version 7 of the **debhelper** package. This guide assumes the use of **debhelper** version 13 or newer.
³Команда **debmake** создаёт несколько более сложный файл **debian/rules**. Тем не менее, это базовая часть.

Tip



Установка «**export DH_VERBOSE = 1**» приводит к тому, что в выводе отражаются все команды, изменяющие файлы в сборочной системе. Кроме того, для некоторых систем сборки это включает режим создания более подробных журналов.

5.6.3 Настроенный файл `debian/rules`

Гибкая настройка сценария `debian/rules` осуществляется путём добавления соответствующих целей `override_dh_*` и их правил.

Всякий раз как требуется выполнить какую-то специальную операцию для определённой команды `dh_foo`, вызываемой командой `dh`, всякое автоматическое выполнение её может быть отменено путём добавления Makefile-цели `override_dh_foo` в файл `debian/rules`.

The build process may be customized via the upstream provided interface such as arguments to the standard source build system commands, such as:

- `configure`,
- `Makefile`,
- `setup.py` или
- `Build.PL`.

If this is the case, you should add the `override_dh_auto_build` target and executing the “`dh_auto_build --arguments`” command. This ensures passing *arguments* to the such build system after the default parameters that `dh_auto_build` usually passes.

Tip



Please try not to execute the above build system commands directly if they are supported by the `dh_auto_build` command.

The `debmake` command creates the initial template file taking advantage of the above simple `debian/rules` file example while adding some extra customizations for package hardening, etc. You need to know how underlying build systems work under the hood (see Section 5.18) to address their irregularities using package customization.

- See Section 4.7 for basic customization of the template `debian/rules` file generated by the `debmake` command.
- See Section 5.22 for multiarch customization.
- Информацию о настройке улучшений безопасности см. в Section 5.23.

5.6.4 Переменные `debian/rules`

Некоторые определения переменных, которые могут оказаться полезными для `debian/rules`, можно найти в файлах в каталоге `/usr/share/dpkg/`. В частности:

`pkg-info.mk` `DEB_SOURCE`, `DEB_VERSION`, `DEB_VERSION_EPOCH_UPSTREAM`, `DEB_VERSION_UPSTREAM`, `DEB_VERSION_UPSTREAM`, and `DEB_DISTRIBUTION` variables. These are useful for backport support etc..

`vendor.mk` `DEB_VENDOR` and `DEB_PARENT_VENDOR` variables; and `dpkg_vendor_derives_from` macro. These are useful for vendor support (Debian, Ubuntu, ...).

architecture.mk Set **DEB_HOST_*** and **DEB_BUILD_*** variables. An alternative method of retrieving those variables is to invoke **dpkg-architecture** directly and query the value of a single variable. With explicit invocation of **dpkg-architecture** to retrieve necessary variables, there is no need to include **architecture.mk** in **debian/rules**, which would import all architecture-related variables.

buildflags.mk Установите флаги сборки **CFLAGS**, **CPPFLAGS**, **CXXFLAGS**, **OBJCFLAGS**, **OBJCXXFLAGS**, **GCJFLAGS**, **FFLAGS**, **FCFLAGS** и **LDFLAGS**.

If you wish to use some of these useful variables in **debian/rules**, copy relevant code to **debian/rules** or write a simpler alternative in it. Please keep **debian/rules** simple.

Например, можно добавить дополнительную опцию в **CONFIGURE_FLAGS** для архитектур **linux-any**, добавляя следующее в файл **debian/rules**:

```
DEB_HOST_ARCH_OS ?= $(shell dpkg-architecture -qDEB_HOST_ARCH_OS)
...
ifeq ($(DEB_HOST_ARCH_OS), linux)
CONFIGURE_FLAGS += --enable-wayland
endif
```

Tip



It was useful to include **buildflags.mk** in **debian/rules** to set the build flags such as **CPPFLAGS**, **CFLAGS**, **LDFLAGS**, etc. properly while honoring **DEB_CFLAGS_MAINT_APPEND**, **DEB_BUILD_MAINT_OPTIONS**, etc. for the **debhelper** “compat <= 8”. Now you should use the **debhelper** “compat >= 9”, should not include **buildflags.mk** without specific reasons, and should let the **dh** command set these build flags.

См. Section 5.22, **dpkg-architecture(1)** и **dpkg-buildflags(1)**.

5.6.5 Повторяемая сборка

Here are some recommendations to attain a reproducible build result.

- Не включайте в результат временную метку на основе системного времени.
- Используйте «**dh \$@**» в **debian/rules** для обращения к самым новым возможностям **debhelper**.
- Экпортируйте окружение сборки как «**LC_ALL=C.UTF-8**» (см. Section 7.15).
- Set the timestamp used in the upstream source from the value of the debhelper-provided environment variable **\$\$SOURCE_DATE_EPOCH**.
- Подробности можно найти на вики-странице [ReproducibleBuilds](#).
 - [Руководство ReproducibleBuilds](#).
 - [ReproducibleBuilds TimestampsProposal](#).

Управляющий файл *имя-исходного-кода_версия-исходного-кода_архитектура.buildinfo*, создаваемый **dpkg-genbuildinfo(1)**, содержит информацию о сборочном окружении. См. **deb-buildinfo(5)**

5.7 debian/control

The **debian/control** file consists of blocks of meta data separated by a blank line. Each block of meta data defines the following in this order:

- метаданных пакета Debian с исходным кодом
- метаданные двоичных пакетов Debian

See [Chapter 5 - Control files and their fields](#) of the “Debian Policy Manual” for the definition of each meta data.

Note



The **debmake** command sets the **debian/control** file with “**Build-Depends: debhelper-compat (= 13)**” to set the **debhelper** compatibility level.

Tip



If an existing package has lower than **debhelper** compatibility level 9, probably it’s time to update its packaging.

5.7.1 Split of a Debian binary package

For well behaving build systems, the split of a Debian binary package into small ones can be realized as follows.

- Создайте записи с определениями метаданных двоичных пакетах в файле **debian/control** для всех двоичных пакетов.
- Укажите все пути к файлам (относительно каталга **debian/tmp**) в соответствующих файлах **debian/двоичныйпакет.i**

С примерами можно ознакомиться в настоящем руководстве:

- Section 8.11 (на основе Autotools)
- Section 8.12 (на основе CMake)

5.7.1.1 debmake -b

The **debmake** command with the **-b** option provides an intuitive and flexible method to create the initial template **debian/control** file defining the split of the Debian binary packages with following stanzas:

- **Package:**
- **Architecture:**
- **Multi-Arch:**
- **Depends:**
- **Pre-Depends:**

The **debmake** command also sets an appropriate set of substvars used in each pertinent dependency stanza. Ниже приводится цитата соответствующей части страницы руководства **debmake**.

-b “*двоичныйпакет[:mun],...*”, **--binaryspec** “*двоичныйпакет[:mun],...*” set the binary package specs by a comma separated list of *binarypackage:type* pairs, e.g., in the full form “**foo:bin,foo-doc:doc,libfoo1:lib,libfoo-dev:dev**” or in the short form, “**-doc,libfoo1,libfoo-dev**”.

Here, *binarypackage* is the binary package name, and the optional *type* is chosen from the following *type* values:

- **bin**: скомпилированный двоичный пакет с исполняемыми файлами формата ELF на языке C/C++ (any, foreign) (по умолчанию, псевдоним: “”, то есть *пустая-строка*)
- **data**: пакет с данными (шрифты, графика, ...) (all, foreign) (псевдоним: **da**)
- **dev**: пакет с библиотекой разработки (any, same) (псевдоним: **de**)
- **doc**: пакет документации (all, foreign) (псевдоним: **do**)

- **lib**: пакет с библиотекой (any, same) (псевдоним: **l**)
- **perl**: пакет со сценарием на языке Perl (all, foreign) (псевдоним: **pl**)
- **python3**: Python (version 3) script package (all, foreign) (alias: **py3**)
- **ruby**: пакет со сценарием на языке Ruby (all, foreign) (псевдоним: **rb**)
- **nodejs**: Node.js based JavaScript package (all, foreign) (alias: **js**)
- **script**: пакет со сценарием командной оболочки (all, foreign) (псевдоним: **sh**)

Пары значений в скобках, такие как (any, foreign), представляют собой значения служебных строк **Architecture** и **Multi-Arch**, устанавливаемые в файле **debian/control**.

Во многих случаях команда **debmake** довольно хорошо предсказывает значение поля *mun*, исходя из значения поля *двоичныйпакет*. Если *mun* не очевиден, то значением поля *mun* становится **bin**. Например, исходя из **libfoo** значением поля *mun* становится **lib**, а исходя из **font-bar** значением поля *mun* становится **data**, ...

Если содержимое дерева исходного кода не совпадает с настройками поля *mun*, то команда **debmake** выводит предупреждение.

5.7.1.2 Сценарии и примеры разделения пакета

Ниже приводится несколько типичных сценариев разделения мультиархитектурного пакета для следующих примеров исходного кода основной ветки разработки, в которых используется команда **debmake**:

- исходный код библиотеки *libfoo-1.0.tar.gz*
- исходный код утилиты *bar-1.0.tar.gz*, написанный на компилируемом языке
- исходный код утилиты *baz-1.0.tar.gz*, написанный на интерпретируемом языке

<i>двоичныйпакет</i>	<i>mun</i>	Architecture:	Multi-Arch:	Содержимое пакета
libfoo1	lib*	any	same	разделяемая библиотечка, возможна совместная установка
libfoo-dev	dev*	any	same	заголовочные файлы разделяемой библиотеки и проч., возможна совместная установка
libfoo-tools	bin*	any	foreign	программы с поддержкой времени исполнения, совместная установка невозможна
libfoo-doc	doc*	all	foreign	файлы документации разделяемой библиотеки
<i>bar</i>	bin*	any	foreign	скомпилированные файлы программы, совместная установка невозможна
<i>bar-doc</i>	doc*	all	foreign	файлы документации программы
<i>baz</i>	script	all	foreign	файлы интерпретируемой программы

5.7.1.3 Имя пакета библиотеки

Let's consider that the upstream source tarball of the **libfoo** library is updated from **libfoo-7.0.tar.gz** to **libfoo-8.0.tar.gz** with a new SONAME major version which affects other packages.

Двоичный пакет библиотеки следует переименовать с **libfoo7** в **libfoo8**, чтобы после загрузки пакета, созданного из на осно, в **unstable** все зависимые пакеты остались в рабочем состоянии.

Warning



If the binary library package isn't renamed, many dependent packages in the **unstable** suite become broken just after the library upload even if a binNMU upload is requested. The binNMU may not happen immediately after the upload due to several reasons.

Пакет **-dev** должен соответствовать следующим правилам именования:

- Используйте имя пакета **-dev без номера версии: libfoo-dev**
 - This is the typical one for leaf library packages.
 - В архиве может находиться только одна версия пакета с исходным кодом библиотеки.
 - * The associated library package needs to be renamed from **libfoo7** to **libfoo8** to prevent dependency breakage in the **unstable** archive during the library transition.
 - Данный подход следует использовать в том случае, если простая binNMU-загрузка быстро разрешает проблемы с зависимостями от такой библиотеки для всех использующих её пакетов. (Изменение [ABI](#) путём прекращения поддержки устаревшего [API](#) с сохранением активного [API](#) без изменений.)
 - Этот подход всё равно может быть хорошей идеей в случае том случае, если ручное обновление кода и проч. можно координировать и осуществить для ограниченного числа пакетов. (Изменение [API](#))
- Используйте имена пакетов **-dev с указанием версии: libfoo7-dev и libfoo8-dev**
 - This is typical for many major library packages.
 - В архиве могут находиться две версии пакетов с исходным кодом библиотеки.
 - * Все зависимые пакет должны зависеть от **libfoo-dev**.
 - * Пусть **libfoo7-dev**, и **libfoo8-dev** предоставляют **libfoo-dev**.
 - * Пакет с исходным кодом следует переименовать в **libfoo7-7.0.tar.gz** и **libfoo8-8.0.tar.gz**, соответственно, из **libfoo-?.0.tar.gz**.
 - * В зависимости от пакета путь установки файлов, включающий **libfoo7** и **libfoo8**, соответственно, для заголовочных файлов и проч., следует выбирать так, чтобы их можно было установить одновременно.
 - По возможности не используйте слишком жёсткий подход.
 - This approach should be used if the update of multiple dependent packages require manual code updates, etc. ([API](#) change) Otherwise, the affected packages become RC buggy with FTBFS.

Tip



If the data encoding scheme changes (e.g., latin1 to utf-8), the same care as the [API](#) change needs to be taken.

См. Section [5.20](#).

5.7.2 Переменные подстановки

Кроме того, файл **debian/control** определяет зависимости пакета, в которых может использоваться [механизм подстановки переменных](#) (substvar), который освобождает сопровождающих пакета от рутинной работы по отслеживанию большинства простых зависимостей пакета. См. **deb-substvars(5)**.

The **debmake** command supports the following substvars:

- **\${misc:Depends}** для всех двоичных пакетов
- **\${misc:Pre-Depends}** для всех мультиархитектурных пакетов
- **\${shlibs:Depends}** для всех двоичных пакетов с исполняемыми файлами и пакетов библиотек
- **\${python:Depends}** для всех пакетов с кодом на языке Python
- **\${python3:Depends}** для всех пакетов с кодом на языке Python3
- **\${perl:Depends}** для всех пакетов с кодом на языке Perl

- `#{ruby:Depends}` для всех пакетов с кодом на языке Ruby

Для разделяемых библиотек необходимые библиотеки обнаруживаются с помощью просто команды «`objdump -p /путь/к/программе | grep NEEDED`» и обрабатываются переменной подстановки `shlib`.

For Python and other interpreters, required modules found simply looking for lines with “`import`”, “`use`”, “`require`”, etc., are covered by the corresponding substvars.

Для остальных программ, не использующих собственные переменные подстановки, зависимости обрабатываются переменной `misc`.

Для программ командной оболочки POSIX нет простого способа определения зависимостей, поэтому их зависимости не обрабатываются никакой переменной.

Для библиотек и модулей, требующихся через механизм динамической загрузки, включая механизм [GObject-интроспекция](#), нет простого способа определения зависимостей, поэтому их зависимости не обрабатываются никакой переменной.

5.7.3 Безопасная binNMU-загрузка

A [binNMU](#) is a binary-only non-maintainer upload performed for library transitions etc. In a binNMU upload, only the “**Architecture: any**” packages are rebuilt with a suffixed version number (e.g. version 2.3.4-3 will become 2.3.4-3+b1). The “**Architecture: all**” packages are not built.

The dependency defined in the `debian/control` file among binary packages from the same source package should be safe for the binNMU. This needs attention if there are both “**Architecture: any**” and “**Architecture: all**” packages involved in it.

- “**Architecture: any**” package: depends on “**Architecture: any**” *foo* package
 - `Depends: foo (= #{binary:Version})`
- “**Architecture: any**” package: depends on “**Architecture: all**” *bar* package
 - `Depends: bar (= #{source:Version})`
- “**Architecture: all**” package: depends on “**Architecture: any**” *baz* package
 - `Depends: baz (>= #{source:Version}), baz (<< #{source:Version}.0~)`

5.8 debian/changelog

The `debian/changelog` file records the Debian package history and defines the upstream package version and the Debian revision in its first line. The changes need to be documented in the specific, formal, and concise style.

- Even if you are uploading your package by yourself, you must document all non-trivial user-visible changes such as:
 - the security related bug fixes.
 - the user interface changes.
- If you are asking your sponsor to upload it, you should document changes more comprehensively, including all packaging related ones, to help reviewing your package.
 - The sponsor shouldn’t second guess your thought behind your package.
 - The sponsor’s time is more valuable than yours.

Команда `debmake` создаёт изначальный шаблонный файл с версией основной ветки и редакцией Debian. С целью предотвращения случайной загрузки в архив Debian выбирается выпуск **UNRELEASED**.

The `debchange` command (alias `dch`) is commonly used to edit this.

Tip



You can edit the `debian/changelog` file manually with any text editor as long as you follow the formatting convention used by the `debchange` command.

Tip



Строка с датой, используемая в файле **debian/changelog**, может быть создана вручную с помощью команды «**LC_ALL=C date -R**».

Этот файл устанавливается командой **dh_installchangelogs** в каталог `/usr/share/doc/двоичныйпакет` под именем **changelog.Debian.gz**.

Журнал изменений основной ветки устанавливается в каталог `/usr/share/doc/двоичныйпакет` под именем **changelog.gz**.

The upstream changelog is automatically found by the **dh_installchangelogs** using the case insensitive match of its file name to **changelog**, **changes**, **changelog.txt**, **changes.txt**, **history**, **history.txt**, or **changelog.md** and searched in the `./ doc/` or `docs/` directories.

After finishing your packaging and verifying its quality, please execute the “**dch -r**” command and save the finalized **debian/changelog** file with the distribution normally set to **unstable**.⁴ If you are packaging for backports, security updates, LTS, etc., please use the appropriate distribution names instead.

5.9 debian/copyright

Debian takes the copyright and license matters very seriously. The “Debian Policy Manual” enforces having a summary of them in the **debian/copyright** file in the package.

You should format it as a [machine-readable debian/copyright file](#) (DEP-5).

Caution



Файл **debian/copyright** должен быть отсортирован таким образом, что наиболее общие шаблоны файлов были размещены в начале списка. См. Section 6.4.

Команда **debmake** создаёт изначальный совместимый с DEP-5 шаблонный файл, сканируя все дерево исходного кода. Она использует специальную внутреннюю команду для проверки и классификации лицензий.⁵

Если команде **debmake** не была передана опция **-P**, то команда пропускает создаваемые автоматически файлы под разрешительными лицензиями.

Note



Если при проверке лицензионной информации вы обнаружите какие-либо проблемы, то отправьте сообщение об ошибке в пакете **debmake** с проблемной частью текста, содержащего информацию об авторском праве и лицензии.

⁴If you are using the **vim** editor, make sure to save this with the “**:wq**” command.

⁵Ранее для выполнения такой внутренней проверки использовалась команда **licensecheck** из пакета **devscripts**. Теперь же команда **licensecheck** предоставляется в виде отдельного пакета **licensecheck** и содержит множество улучшений.

Note



The **debmake** command focuses on bunching up same copyright and license claims in detail to create template for **debian/copyright**. In order to do this within reasonable time, it only picks the first section which looks like copyright and license claims. So its license assignment may not be optimal. Please also use other tools such as **licensecheck**.

Tip



You are highly encouraged to check the license status with the **licensecheck(1)** command and, as needed, with your manual code review.

5.10 debian/patches/*

Заплаты **-p1** в каталоге **debian/patches/** применяются в последовательности, определяемой в файле **debian/patches/series**, к дереву исходного кода основной ветки до запуска процесса **сборки**.

Note



В родных пакетах Debian (см. Section 5.5) эти файлы не используются.

Имеется несколько способов подготовки серии заплат **-p1**.

- Команда **diff**
 - См. Section 4.9.1
 - Примитивный, но универсальный метод
 - * Заплаты могут поступать из других дистрибутивов, из сообщений в списках рассылки или выборки заплат из **git**-репозитория основной ветки разработки с помощью команды «**git format-patches**»
 - Отсутствие каталога **.pc/**
 - Неизменённое дерево исходного кода основной ветки
 - Обновите файл **debian/patches/series** вручную
- Команда **dquilt**
 - См. Section 3.4
 - Простой и удобный метод
 - Правильное создание данных в каталоге **.pc/**
 - Изменённое дерево исходного кода основной ветки
- Команда «**dpkg-source --commit**»
 - См. Section 4.9.3
 - Более новый и простой метод
 - Правильное создание данных в каталоге **.pc/**
 - Изменённое дерево исходного кода основной ветки
- Автоматическое создание заплат с помощью **dpkg-buildpackage**

- См. Section 5.16
- Добавьте **single-debian-patch** в файл **debian/source/local-options**
- Настройте файл **debian/source/local-patch-header**
- Отсутствие каталога **.pc/**
- Изменённое дерево исходного кода основной ветки в Debian-ветке (**master**)
- The **gbp pq** command
 - Простая работа в стиле **git** с помощью пакета **git-buildpackage**
 - Отсутствие каталога **.pc/**
 - Modified upstream source tree in the throw-away branch (**patch-queue/master**)
 - Неизменённый исходный код основной ветки в Debian-ветке (**master**)
- The **git-dpm** command
 - Более продуманный подход в стиле **git** с помощью пакета **git-dpm**
 - Отсутствие каталога **.pc/**
 - Изменённый исходный код основной ветки в ветке **patched** (**patched/что угодно**)
 - Неизменённый исходный код основной ветки в Debian-ветке (**master/что угодно**)

Wherever these patches come from, it is a good idea to tag them with a [DEP-3](#) compatible header.

Tip



The **dggit** package offers an alternative git integration tool with the Debian package archive.

5.10.1 dpkg-source -x

Команда «**dpkg-source -x**» распаковывает пакет Debian с исходным кодом.

Обычно она же применяет заплатки из каталога **debian/patches/** к дереву исходного кода и записывает состояние заплат в каталог **.pc/**.

Если вы хотите сохранить исходный код в неизменном состоянии (например, для использования в Section 5.15), то используйте опцию **--skip-patches**.

5.10.2 dquilt и dpkg-source

До появления опции **--commit** у команды **dpkg-source** в версии 1.16.1 для работы с заплатками **-p1** из каталога **debian/patches/** требовалась команда **quilt** (или её обёртка **dquilt**).

The patches should apply cleanly when using the **dpkg-source** command. Thus you can't just copy the patches to the new packaging of the new upstream release if there are patch offsets, etc.

The **dquilt** command (see Section 3.4) is more forgiving. You can normalize the patches by the **dquilt** command.

```
$ while dquilt push; do dquilt refresh ; done
$ dquilt pop -a
```

There is one advantage of using the **dpkg-source** command over the **dquilt** command. While the **dquilt** command cannot handle modified binary files automatically, the **dpkg-source** command detects modified binary files and lists them in the **debian/source/include-binaries** file to include them in the Debian tarball.

5.11 debian/upstream/signing-key.asc

Some packages are signed by a GPG key.

Например, пакет [GNU hello](https://ftp.gnu.org/gnu/hello/) можно загрузить через HTTP по адресу <https://ftp.gnu.org/gnu/hello/>. Там содержится несколько файлов:

- **hello-версия.tar.gz** (исходный код основной ветки)
- **hello-версия.tar.gz.sig** (отделённая подпись)

Выберем самую последнюю версию.

```
$ wget https://ftp.gnu.org/gnu/hello/hello-2.9.tar.gz
...
$ wget https://ftp.gnu.org/gnu/hello/hello-2.9.tar.gz.sig
...
$ gpg --verify hello-2.9.tar.gz.sig
gpg: Signature made Thu 10 Oct 2013 08:49:23 AM JST using DSA key ID 80EE4A00
gpg: Can't check signature: public key not found
```

If you know the public GPG key of the upstream maintainer from the mailing list, use it as the **debian/upstream/signing-key.asc** file. Otherwise, use the hkp keyserver and check it via your [web of trust](#).

```
$ gpg --keyserver hkp://keys.gnupg.net --recv-key 80EE4A00
gpg: requesting key 80EE4A00 from hkp server keys.gnupg.net
gpg: key 80EE4A00: public key "Reuben Thomas <rtr@sc3d.org>" imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:         imported: 1
$ gpg --verify hello-2.9.tar.gz.sig
gpg: Signature made Thu 10 Oct 2013 08:49:23 AM JST using DSA key ID 80EE4A00
gpg: Good signature from "Reuben Thomas <rtr@sc3d.org>"
...
Primary key fingerprint: 9297 8852 A62F A5E2 85B2 A174 6808 9F73 80EE 4A00
```

Tip



Если ваше сетевое окружение блокирует доступ к НКР-порту **11371**, используйте «**hkp://keyserver.ubuntu.com:80**».

After confirming the key ID **80EE4A00** is a trustworthy one, download its public key into the **debian/upstream/signing-key.asc** file.

```
$ gpg --armor --export 80EE4A00 >debian/upstream/signing-key.asc
```

Затем настройте файл **debian/watch** соответствующим образом.

```
version=4
pgpsigurlmangle=s/$/.sig/ https://ftp.gnu.org/gnu/hello/ hello-(\d[\d.]*)\.tar ↵
\.(?:gz|bz2|xz)
```

Теперь команда **uscan** будет выполнять проверку подлинности пакета с помощью его GPG-подписи.

5.12 debian/watch и критерии Debian по определению свободного ПО (DFSG)

Debian очень серьёзно относится к вопросу свободы ПО и руководствуется в этом [DFSG](#).

Несоответствующие [DFSG](#) компоненты в tar-архиве исходного кода основной ветки можно легко удалить в том случае, когда для обновления пакета Debian используется команда **uscan**.

- Укажите список файлов для удаления в строке **Files-Excluded** файла **debian/copyright**.
- Укажите URL для загрузки tar-архива основной ветки в файле **debian/watch**.
- Запустите команду **uscan** для загрузки нового tar-архива основной ветки.
 - Либо используйте команду «**gbp import-orig --uscan --pristine-tar**».
- Получившийся tar-архив будет иметь версию с дополнительным суффиксом **+dfsg**.

5.13 Other debian/* Files

В каталог **debian/** можно добавить дополнительные файлы настройки. Большинство из них используются для управления командами **dh_***, предоставляемыми пакетом **debhelper**, но также имеются дополнительные файлы для команд **dpkg-source**, **lintian** и **gbp**.

Tip



Наиболее свежий список доступных команд **dh_*** см. в **debhelper(7)**.

Файлы **debian/двоичныйпакет.*** предоставляют очень мощные средства для выбора путей установки файлов. С помощью этих файлов можно создать пакет даже из исходный кода основной ветки, неиспользующего какую-либо систему сборки. Пример см. в Section 8.2.

The “^x” superscript notation that appears in the following list indicates the minimum value for the **debmake -x** option that will generate the associated template file. See Section 6.6 or **debmake(1)** for details.

Ниже приводится отсортированный по алфавиту список наиболее существенных файлов настройки.

двоичныйпакет.bug-control^{-x3} устанавливается как **usr/share/bug/двоичныйпакет/control** в **двоичныйпакет**. См. Section 5.27.

двоичныйпакет.bug-presubj^{-x3} устанавливается как **usr/share/bug/двоичныйпакет/presubj** в **binarypackage**. См. Section 5.27.

двоичныйпакет.bug-script^{-x3} устанавливается как **usr/share/bug/двоичныйпакет** или **usr/share/bug/двоичныйпакет/sc** в **двоичныйпакет**. См. Section 5.27.

двоичныйпакет.bash-completion Список устанавливаемых автодополнений **bash**.

The **bash-completion** package is required for both build and user environments.

См. **dh_bash-completion(1)**.

clean^{-x2} List files that should be removed but are not cleaned by the **dh_auto_clean** command.

См. **dh_auto_clean(1)** и **dh_clean(1)**.

compat^{-x3} Previously, this set the **debhelper** compatibility level.

Now, use **Build-Depends: debhelper-compat (= 13)** in **debian/control** to specify the compatibility level.

См. «COMPATIBILITY LEVELS» в **debhelper(8)**.

двоичныйпакет.conf No need for this file now since all files in the **etc/** directory are conffiles for recent “compat >= 3”.

If the program you’re packaging requires every user to modify the configuration files in the **/etc** directory, there are two popular ways to arrange for them not to be conffiles, keeping the **dpkg** command happy and quiet.

- Создайте символическую ссылку в каталоге **/etc**, указывающую на файл в каталоге **/var**, создаваемый сценариями сопровождающего.
- Создайте файл с помощью сценариев сопровождающего в каталоге **/etc**.

См. **dh_installdeb(1)**.

двоичныйпакет.config Это **config**-сценарий **debconf**, используемый для того, чтобы задавать пользователю любые необходимые для настройки пакета вопросы. См. Section 5.21.

двоичныйпакет.cron.hourly ^{-x3} Устанавливается в файл **etc/cron/hourly/двоичныйпакет** в **двоичныйпакет**.
См. **dh_installcron(1)** и **cron(8)**.

двоичныйпакет.cron.daily ^{-x3} Устанавливается в файл **etc/cron/daily/двоичныйпакет** в **двоичныйпакет**.
См. **dh_installcron(1)** и **cron(8)**.

двоичныйпакет.cron.weekly ^{-x3} Устанавливается в файл **etc/cron/weekly/двоичныйпакет** в **двоичныйпакет**.
См. **dh_installcron(1)** и **cron(8)**.

двоичныйпакет.cron.monthly ^{-x3} Устанавливается в файл **etc/cron/monthly/двоичныйпакет** в **двоичныйпакет**.
См. **dh_installcron(1)** и **cron(8)**.

двоичныйпакет.cron.d ^{-x3} Устанавливается в файл **etc/cron.d/двоичныйпакет** в **двоичныйпакет**.
См. **dh_installcron(1)**, **cron(8)** и **crontab(5)**.

двоичныйпакет.default ^{-x3} Если такой файл существует, то он устанавливается в **etc/default/двоичныйпакет** в **двоичныйпакет**.
См. **dh_installinit(1)**.

двоичныйпакет.dirs ^{-x3} Содержит список каталогов, которые должны быть созданы в **двоичныйпакет**.
См. **dh_installdirs(1)**.

Это не требуется, поскольку все команды **dh_install*** автоматически создают необходимые каталоги. Используйте этот файл только в том случае, если у вас возникают какие-либо затруднения.

двоичныйпакет.doc-base ^{-x2} Устанавливается как управляющий файл **doc-base** в **двоичныйпакет**.
См. **dh_installdocs(1)** и [Руководство Debian по doc-base](#), предоставляемое пакетом **doc-base**.

двоичныйпакет.docs ^{-x2} Содержит список файлов документации для их установки в **двоичныйпакет**.
См. **dh_installdocs(1)**.

двоичныйпакет.emacsen-compat ^{-x3} Устанавливается в **usr/lib/emacsen-common/packages/compat/двоичныйпакет** в **binarypackage**.
См. **dh_installemacsen(1)**.

двоичныйпакет.emacsen-install ^{-x3} Устанавливается в **usr/lib/emacsen-common/packages/install/двоичныйпакет** в **двоичныйпакет**.
См. **dh_installemacsen(1)**.

двоичныйпакет.emacsen-remove ^{-x3} Устанавливается в **usr/lib/emacsen-common/packages/remove/двоичныйпакет** в **двоичныйпакет**.
См. **dh_installemacsen(1)**.

двоичныйпакет.emacsen-startup ^{-x3} Устанавливается в **usr/lib/emacsen-common/packages/startup/двоичныйпакет** в **двоичныйпакет**.
См. **dh_installemacsen(1)**.

двоичныйпакет.examples ^{-x2} Содержит список файлов или каталогов с примерами для их установки в **usr/share/doc/двоичныйпакет/examples/** в **двоичныйпакет**.
См. **dh_installexamples(1)**.

gbp.conf Если этот файл существует, то он используется как файл настройки для команды **gbp**.
См. **gbp.conf(5)**, **gbp(1)** и **git-buildpackage(1)**.

двоичныйпакет.info ^{-x2} Содержит список info-файлов для их установки в **двоичныйпакет**.
См. **dh_installinfo(1)**.

двоичныйпакет.init ^{-x3} Устанавливается в **etc/init.d/двоичныйпакет** в **двоичныйпакет**.

См. **dh_installinit**(1).

двоичныйпакет.install ^{-x2} Содержит список файлов, которые должны быть установлены, но не устанавливаются командой **dh_auto_install**.

См. **dh_install**(1) и **dh_auto_install**(1).

license-examples/* ^{-x4} These are copyright file examples generated by the **debmake** command. Use these as the reference for making the **copyright** file.

Обязательно удалите эти файлы.

двоичныйпакет.links ^{-x2} List pairs of source and destination files to be symlinked. Each pair should be put on its own line, with the source and destination separated by whitespace.

См. **dh_link**(1).

двоичныйпакет.lintian-overrides ^{-x3} Устанавливается в **usr/share/lintian/overrides/двоичныйпакет** в каталоге сборки пакета. Этот файл используется для блокировки ошибочных диагностических процедур **lintian**.

См. **dh_lintian**(1), **lintian**(1) и [Руководство пользователя Lintian](#).

manpage.* ^{-x3} Команда **debmake** создаёт шаблонные файлы страниц руководства. Переименуйте эти файлы соответствующим образом и обновите их содержимое.

Политика Debian требует, чтобы у каждой программы, утилиты и функции была связанная с ними страница руководства, входящая в состав того же пакета. Страницы руководства пишутся в формате **nroff**(1).

If you are new to making a manpage, use **manpage.asciidoc** or **manpage.1** as the starting point.

двоичныйпакет.manpages ^{-x2} Содержит список страниц руководства для их установки.

См. **dh_installman**(1).

двоичныйпакет.menu (устарел, более не устанавливается) [Технический комитет в #741573](#) решил «В Debian следует соответствующим образом использовать файлы **.desktop**».

Файл меню Debian устанавливается в **usr/share/menu/двоичныйпакет** в **двоичныйпакет**.

Информацию о формате см. в **menufile**(5). См. **dh_installmenu**(1).

NEWS Устанавливается в **usr/share/doc/двоичныйпакет/NEWS.Debian**.

См. **dh_installchangelogs**(1).

patches/* Набор файлов заплат **-p1**, которые применяются к исходному коду основной ветки до запуска процесса сборки исходного кода.

См. **dpkg-source**(1), Section 3.4 и Section 4.9.

Команда **debmake** не создаёт файлы заплат.

patches/series ^{-x1} Последовательность применения файлов заплат **patches/***.

двоичныйпакет.preinst ^{-x2}, **двоичныйпакет.postinst** ^{-x2}, **двоичныйпакет.prerm** ^{-x2}, **двоичныйпакет.postrm** ^{-x2}
Указанные сценарии сопровождающего устанавливаются в каталог **DEBIAN**.

В самих этих сценариях токен **#DEBHELPER#** заменяется на фрагменты кода, создаваемые другими командами **debhelper**.

See **dh_installdeb**(1) and [Chapter 6 - Package maintainer scripts and installation procedure](#) in the “Debian Policy Manual”.

See also **debconf-devel**(7) and [3.9.1 Prompting in maintainer scripts](#) in the “Debian Policy Manual”.

README.Debian ^{-x1} Устанавливается в первый двоичный пакет, указанный в файле **debian/control** как **usr/share/doc/двоичныйпакет/README.Debian**.

См. **dh_installdocs**(1).

Этот файл содержит специальную информацию о пакете Debian.

двоичныйпакет.service ^{-x3} Если этот файл существует, то он устанавливается в `lib/systemd/system/двоичныйпакет.serv` в `binarypackage`.

См. `dh_systemd_enable(1)`, `dh_systemd_start(1)` и `dh_installinit(1)`.

source/format ^{-x1} Формат пакета Debian.

- Используйте «**3.0 (quilt)**» для создания неродных пакетов (рекомендуется)
- Используйте «**3.0 (native)**» для создания родного пакета

См. «SOURCE PACKAGE FORMATS» в `dpkg-source(1)`.

source/lintian-overrides или **source.lintian-overrides** ^{-x3} Эти файлы не устанавливаются, но загружаются командой `lintian` и предоставляют отмены проверок для пакета с исходным кодом.

См. `dh_lintian(1)` и `lintian(1)`.

source/local-options ^{-x1} The `dpkg-source` command uses this content as its options. Notable options are:

- **unapply-patches**
- **abort-on-upstream-changes**
- **auto-commit**
- **single-debian-patch**

Этот файл не добавляется в создаваемый пакет с исходным кодом и предназначен скорее для добавления в систему управления версиями, используемую сопровождающим.

См. «FILE FORMATS» в `dpkg-source(1)`.

source/local-patch-header Свободная текстовая форма, размещаемая в верхней части автоматически созданной заплаты.

Этот файл не добавляется в создаваемый пакет с исходным кодом и предназначен скорее для добавления в систему управления версиями, используемую сопровождающим.

+ См. «FILE FORMATS» в `dpkg-source(1)`.

двоичныйпакет.symbols ^{-x2} Файлы символов. Если эти файлы существуют, то они будут переданы для обработки и установки команде `dpkg-gensymbols`.

См. `dh_makeshlibs(1)` и Section 5.20.1..

двоичныйпакет.templates Это файл **шаблонов** для `debconf`. Он используется для вывода вопросов, необходимых для настройки пакета. См. Section 5.21.

tests/control This is the RFC822-style test meta data file defined in [DEP-8](#). See `autopkgtest(1)` and Section 5.24.

TODO Устанавливается в первый двоичный пакет, указанный в файле `debian/control` как `usr/share/doc/двоичныйпакет/` См. `dh_installdocs(1)`.

двоичныйпакет.tmpfile ^{-x3} Если этот файл существует, то он устанавливается в `usr/lib/tmpfiles.d/двоичныйпакет.conf` в `двоичныйпакет`.

См. `dh_systemd_enable(1)`, `dh_systemd_start(1)` и `dh_installinit(1)`.

двоичныйпакет.upstart ^{-x3} Если этот файл существует, то он устанавливается в `etc/init/package.conf` в каталог сборки пакета. (устарел)

См. `dh_installinit(1)` и Section 8.1.

watch ^{-x1} The control file for the `uscan` command to download the latest upstream version.

Этот управляющий файл можно настроить так, чтобы выполнялась проверка подлинности tar-архива с помощью его GPG-подписи (см. Section 5.11).

См. Section 5.12 и `uscan(1)`.

Here are a few reminders for the above list.

- For a single binary package, the `binarypackage.` part of the filename in the list may be removed.

- For a multi binary package, a configuration file missing the *binarypackage.* part of the filename is applied to the first binary package listed in the **debian/control**.
- Если создаётся много двоичных пакетов, то их настройки можно указать отдельно, добавив их имя к их файлам настройки, например, таким как *пакет-1.install*, *пакет-2.install* и т. д.
- Некоторые шаблонные файлы настроек могут не быть созданы командой **debmake**. В таких случаях вам следует создать их с помощью редактора.
- Unusual configuration template files generated by the **debmake** command with an extra **.ex** suffix need to be activated by removing that suffix.
- Неиспользуемые шаблонные файлы настроек, созданные командой **debmake**, следует удалить.
- Копируйте шаблонные файлы настроек по необходимости в файлы с соответствующими именами двоичных пакетов.

5.14 Настройка пакета Debian

Кратко резюмируем настройку пакета Debian.

All customization data for the Debian package resides in the **debian/** directory. A simple example is given in Section 4.7. Normally, this customization involves a combination of the following:

- Систему сборки пакета Debian можно настроить с помощью файла **debian/rules** file (см. Section 5.6.3).
- The Debian package installation path etc. can be customized through the addition of configuration files such as *package.install* and *package.docs* in the **debian/** directory for the **dh_*** commands from the **debhelper** package (see Section 5.13).

When these are not sufficient to make a good Debian package, modifications to the upstream source recorded as the **-p1** patches in the **debian/patches/** directory is deployed. These patches are applied in the sequence defined in the **debian/patches/series** file before building the package (see Section 5.10). Simple examples are given in Section 4.9.

You should address the root cause of the Debian packaging problem by the least invasive way. The generated package shall be more robust for future upgrades in this way.

Note



Send the patch addressing the root cause to the upstream maintainer if it is useful to the upstream.

5.15 Запись в систему управления версиями (стандарт)

Typically, **Git** is used as the **VCS** to record the Debian packaging activity with the following branches.

- ветка **master**
 - Записывает дерево исходного кода, используемое для пакета Debian.
 - Часть, относящаяся к дереву исходного кода основной ветки разработки, записывается в неизменённом виде.
 - Изменения основной ветки, производимые в пакете Debian, записываются в каталог **debian/patches/** в виде **-p1** заплат.
- ветка **upstream**
 - Record the upstream source tree untarred from the released upstream tarball.

Tip



It's a good idea to add to the `.gitignore` file the listing `.pc`.

Tip



Add `unapply-patches` and `abort-on-upstream-changes` lines to the `debian/source/local-options` file to keep the upstream portion unmodified.

Tip



Кроме того, вы можете отслеживать данные системы управления версиями основной ветки в отличной от `upstream` ветке, что облегчит выборочное использование заплат.

5.16 Запись в систему управления версиями (альтернатива)

Можно не сохранять заплаты `-p1` для всех изменений основной ветки, свою работу над пакетом Debian можно хранить в следующих ветках.

- ветка `master`
 - Записывает дерево исходного кода, используемое для пакета Debian.
 - Исходный код основной ветки сохраняется вместе с изменениями, внесёнными в ходе подготовки пакета Debian.
- ветка `upstream`
 - Record the upstream source tree untarred from the released upstream tarball.

Adding a few extra files in the `debian/` directory enables you to do this.

```
$ tar -xvzf <package-version>.tar.gz
$ ln -sf <package_version>.orig.tar.gz
$ cd <package-version>/
... hack...hack...
$ echo "single-debian-patch" >> debian/source/local-options
$ cat >debian/source/local-patch-header <<END
This patch contains all the Debian-specific changes mixed
together. To review them separately, please inspect the VCS
history at https://git.debian.org/?=collab-maint/foo.git.
```

Let the `dpkg-source` command invoked by the Debian package build process (`dpkg-buildpackage`, `debuild`, ...) generate the `-p1` patch file `debian/patches/debian-changes` automatically.

Tip



Этот подход может быть адаптирован для инструментов любых систем управления версиями. Поскольку в этом подходе происходит слияние всех изменений в объединённую заплату, постольку желательно сохранять данные системы управления версиями в открытом для всех виде.

Tip



Файлы **debian/source/local-options** и **debian/source/local-patch-header** следует сохранять в системе управления версиями. Они не входят в пакет Debian с исходным кодом.

5.17 Сборка пакета без постороннего содержимого

There are a few cases which cause the inclusion of undesirable contents in the generated Debian source package.

- Дерево исходного кода основной ветки может быть размещено в системе управления версиями. При пересборке пакета из дерева исходного кода создаваемый пакет Debian с исходным кодом содержит постороннее содержимое, источником которого является система управления содержимым.
- Дерево исходного кода основной ветки может содержать автоматически создаваемые файлы. При пересборке пакета из дерева исходного кода создаваемый пакет Debian с исходным кодом содержит постороннее содержимое, источником которого являются автоматически создаваемые файлы.

Обычно в Section 3.5 для команды **dpkg-source** устанавливаются опции **-i** и **-I**, что позволяет этого избежать. Опция **-i** предназначена для неродных пакетов, а опция **-I** — для родных пакетов. См. **dpkg-source(1)** и вывод команды «**dpkg-source --help**».

There are several methods to avoid inclusion of undesirable contents.

5.17.1 Исправление с помощью **debian/rules clean**

Проблема постороннего содержимого может быть исправлена путём удаления таких файлов в цели «**debian/rules clean**». Также это полезно и для файлов, создаваемых автоматически.

Note



Цель «**debian/rules clean**» вызывается командой **dpkg-buildpackage** до выполнения команды «**dpkg-source --build**», поэтому команда «**dpkg-source --build**» игнорирует удалённые файлы.

5.17.2 Исправление с помощью систем управления версиями

Проблема постороннего содержимого может быть исправлена путём восстановления дерева исходного кода путём внесения его в систему управления версиями до выполнения первой сборки.

Вы можете восстановить дерево исходного кода до выполнения второй сборки. Например:

```
$ git reset --hard
$ git clean -dfx
$ debuild
```

This works because the **dpkg-source** command ignores the contents of the typical VCS files in the source tree with the **DEBUILD_DPKG_BUILDPACKAGE_OPTS** setting in Section 3.5.

Tip



If the source tree is not managed by a VCS, you should run **“git init; git add -A .; git commit”** before the first build.

5.17.3 Исправление с помощью `extend-diff-ignore`

This is for a non-native package.

Проблема посторонних изменений может быть исправлена путём игнорирования изменений, внесённых в часто дерева исходного кода, путём добавления в файл `debian/source/options` строки **«extend-diff-ignore=...»**.

Для того, чтобы исключить файлы `config.sub`, `config.guess` и `Makefile`, добавьте следующее:

```
# Don't store changes on autogenerated files
extend-diff-ignore = "(^|/)(config\.sub|config\.guess|Makefile)$"
```

Note



Этот подход работает всегда, даже если вы не можете удалить файл. Поэтому он позволяет вам не делать резервную копию неизменённого файла только для того, чтобы восстановить его для выполнения следующей сборки.

Tip



Если используется файл `debian/source/local-options`, то можно скрыть эти настройки из создаваемого пакета с исходным кодом. Это может быть полезно в том случае, когда локальные нестандартные файлы системы управления версиями вмешиваются в процесс создания вашего пакета.

5.17.4 Исправление с помощью `tar-ignore`

This is for a native package.

Можно исключить некоторые файлы в дереве исходного кода из создаваемого tar-архива с помощью настройки файлового шаблона, добавив строки вида **«tar-ignore=...»** в файлы `debian/source/options` или `debian/source/local-options`.

Note



Если, например, пакету с исходным кодом для родного пакета требуются в качестве тестовых данных файлы с расширением `.o`, то использование Section 3.5 будет слишком агрессивным. Можно обойти эту проблему, удалив опцию `-I` из `DEB_BUILD_DPKG_BUILDPACKAGE_OPTS` в Section 3.5 и добавив строки вида **«tar-ignore=...»** в файл `debian/source/local-options` для каждого пакета.

5.18 Системы сборки основной ветки

Системы сборки основной ветки предназначены для выполнения нескольких шагов, необходимых для установки созданных из набора файлов исходного кода двоичных файлов в систему.

Three typical build systems are described here. The situation of other build systems are very similar to these since `debhelper(7)` does most of the work and helps you build a Debian package.

Tip



Before attempting to make a Debian package, you should become familiar with the upstream build system of the upstream source code and try to build it.

5.18.1 Autotools

Autotools (**autoconf** + **automake**) имеет 4 шага.

1. set up the build system (“**vim configure.ac Makefile.am**” and “**autoreconf -ivf**”)
2. настройка системы сборки («./**configure**»)
3. сборка дерева исходного кода («**make**»)
4. установка двоичных файлов («**make install**»)

The upstream maintainer usually performs step 1 and builds the upstream tarball for distribution using the “**make dist**” command. (The generated tarball contains not only the pristine upstream VCS contents but also other generated files.)

The package maintainer needs to take care of steps 2 to 4 at least. This is realized by the “**dh \$@ --with autotools-dev**” command used in the **debian/rules** file.

The package maintainer may wish to take care all steps 1 to 4. This is realized by the “**dh \$@ --with autoreconf**” command used in the **debian/rules** file. This rebuilds all auto-generated files to the latest version and provides better support for porting to the newer architectures.

For **compat** level **10** or newer, the simple “**dh \$@**” command without “**--with autoreconf**” option can take care all steps 1 to 4, too.

If you wish to learn more on Autotools, please see:

- [Документация по GNU Automake](#)
- [Документация по GNU Autoconf](#)
- [Вводное руководство Autotools](#)
- [Введение в autotools \(autoconf, automake, and libtool\)](#)
- [Разрушитель мифов Autotools](#)

5.18.2 CMake

CMake имеет 4 шага.

1. set up the build system (“**vim CMakeLists.txt config.h.in**”)
2. настройка системы сборки («**cmake**»)
3. сборка дерева исходного кода («**make**»)
4. установка двоичных файлов («**make install**»)

The upstream tarball contains no auto-generated files and is generated by the **tar** command after step 1.

The package maintainer needs to take care of steps 2 to 4.

Если вы хотите изучить CMake, то обратитесь к следующей документации:

- [CMake](#)
- [Вводное руководство CMake](#)

5.18.3 Meson

Meson has 4 steps.

1. set up the **Meson** build system (“**vim meson.build**”)
2. set up the **Ninja** build system and configure it (“**meson setup builddir && cd builddir**” and “**meson configure**”)
3. build the source tree (“**meson compile**”)
4. install the binary files (“**meson install**”)

The upstream tarball contains no auto-generated files and is generated by the **tar** command after step 1.

The package maintainer needs to take care of steps 2 to 4.

If you wish to learn more on the Meson, please see:

- [The Meson Build system](#)
- [Ninja](#)

5.18.4 Python distutils

Python distutils имеет 3 шага.

1. set up and configure the build system (“**vim setup.py**”)
2. build the source tree (“**python3 setup.py build**”)
3. install the binary files (“**python3 setup.py install**”)

The upstream maintainer usually performs step 1 and builds the upstream tarball for distribution using the “**python3 setup.py sdist**” command.

The package maintainer needs to take care of step 2. This is realized simply by the “**dh \$@**” command used in the **debian/rules** file.

These days, most upstream maintainers of Python packages use **setuptools** with **wheel**. Since **setuptools** is an extension of **distutils**, this step 2 works as expected even if **setup.py** doesn't explicitly uses **distutils**.

If you wish to learn more on Python3, **distutils**, and **setuptools**, please see:

- [Python3](#)
- [distutils](#)
- [setuptools](#)

5.19 Отладочная информация

The Debian package is built with the debugging information but packaged into the binary package after stripping the debugging information as required by [Chapter 10 - Files](#) of the “Debian Policy Manual”.

См.

- [6.7.9. Лучшие практики для отладочных пакетов](#) «Справочника разработчика Debian».
- [18.2 Отладочная информация в отдельных файлах](#) «Отладка с помощью gdb»
- **dh_strip(1)**
- **strip(1)**
- **readelf(1)**
- **objcopy(1)**
- Debian вики [DebugPackage](#)
- Debian вики [AutomaticDebugPackages](#)
- Сообщение в списке рассылки **debian-devel**: [Информация о статусе автоматических отладочных пакетов \(2015-08-15\)](#)

5.19.1 New `-dbgsym` package (Stretch 9.0 and after)

The debugging information is automatically packaged separately as the debug package using the `dh_strip` command with its default behavior. The name of such a debug package normally has the `-dbgsym` suffix.

If there were no `-dbg` packages defined in the `debian/control` file, no special care is needed for updating the package after the Stretch 9.0 release.

- The `debian/rules` file shouldn't explicitly contain `dh_strip`.
- Remove `debian/compat`.
- Set the **Build-Depends** to `debhelper-compat (>=13)` while removing **Build-Depends** to `debhelper` in `debian/control`.

If there were `-dbg` packages defined in the `debian/control` file, following care is needed for updating the old package after the Stretch 9.0 release.

- Drop definition entries of such `-dbg` packages in the `debian/control` file.
- Replace “`dh_strip --dbg-package=package`” with “`dh_strip --dbgsym-migration=package`” in the `debian/rules` file to avoid file conflicts with the (now obsolete) `-dbg` package. See `dh_strip(1)`.
- Remove `debian/compat`.
- Set the **Build-Depends** to `debhelper-compat (>=13)` while removing **Build-Depends** to `debhelper` in `debian/control`.

5.20 Пакет библиотеки

Packaging library software requires you to perform much more work than usual. Here are some reminders for packaging library software:

- The library binary package must be named as in Section 5.7.1.3.
- Debian ships shared libraries such as `/usr/lib/<triplet>/libfoo-0.1.so.1.0.0` (see Section 5.22).
- Debian encourages using versioned symbols in the shared library (see Section 5.20.1).
- Debian не поставляет `libtool`-архивы библиотек `*.la`.
- Debian discourages using and shipping `*.a` static library files.

Before packaging shared library software, see:

- [Chapter 8 - Shared libraries](#) of the “Debian Policy Manual”
- [10.2 Libraries](#) of the “Debian Policy Manual”
- [6.7.2. Библиотеки](#) «Справочника разработчика Debian»

Для получения исторических сведений обратитесь к следующей документации:

- [Спасение из ада зависимостей](#)⁶
 - This encourages having versioned symbols in the shared library.
- [Руководство по созданию пакетов Debian с библиотеками](#)⁷
 - Кроме того, прочтите ветку обсуждения, последовавшего за [сообщением](#) об этом.

⁶Этот документ был написан до появления файла `symbols`.

⁷The strong preference is to use the SONAME versioned `-dev` package names over the single `-dev` package name in [Chapter 6. Development \(-DEV\) packages](#), which does not seem to be shared by the former ftp-master (Steve Langasek). This document was written before the introduction of the `multiarch` system and the `symbols` file.

5.20.1 Библиотека символов

The symbols support in **dpkg** introduced in Debian **lenny** (5.0, May 2009) helps us to manage the backward ABI compatibility of the library package with the same package name. The **DEBIAN/symbols** file in the binary package provides the minimal version associated with each symbol.

An oversimplified method for the library packaging is as follows.

- Распакуйте старый файл **DEBIAN/symbols** ближайшего предыдущего двоичного пакета с помощью команды «**dpkg-deb -e**».
 - Либо можно использовать команду **mc** для распаковки файла **DEBIAN/symbols**.
- Скопируйте его в файл **debian/двоичныйпакет.symbols**.
 - Если это первый пакет, то используйте пустой файл.
- Соберите двоичный пакет.
 - If the **dpkg-gensymbols** command warns about some new symbols:
 - * Распакуйте обновлённый файл **DEBIAN/symbols** с помощью команды «**dpkg-deb -e**».
 - * Удалите номер редакции версии Debian, например, **-1**, из файла.
 - * Скопируйте его в файл **debian/двоичныйпакет.symbols**.
 - * Повторно соберите двоичный пакет.
 - If the **dpkg-gensymbols** command does not warn about new symbols:
 - * Работа с библиотекой завершена.

Подробные сведения можно получить, обратившись к следующей справочной информации:

- [8.6.3 The symbols system](#) of the “Debian Policy Manual”
- **dh_makeshlibs(1)**
- **dpkg-gensymbols(1)**
- **dpkg-shlibdeps(1)**
- **deb-symbols(5)**

Также следует ознакомиться со следующей документацией:

- Debian вики [UsingSymbolsFiles](#)
- Debian вики [Projects/ImprovedDpkgShlibdeps](#)
- Команда Debian KDE [Работа с файлами символов](#)
- Section [8.11](#)
- Section [8.12](#)

Tip



For C++ libraries and other cases where the tracking of symbols is problematic, follow [8.6.4 The shlibs system](#) of the “Debian Policy Manual”, instead. Please make sure to erase the empty **debian/binarypackage.symbols** file generated by the **debmake** command. For this case, the **DEBIAN/shlibs** file is used.

5.20.2 Смена библиотек

Когда вы создаёте новую версию пакета библиотеки, изменение которой влияет на другие пакеты, вам следует с помощью команды **reportbug** отправить сообщение об ошибке в псевдопакете **release.debian.org** с приложением **ben-файла** и подождать подтверждение загрузки от команды подготовки выпуска.

У команды подготовки выпуска имеется [система отслеживания переходов](#). См. [Transitions](#).

Caution



Please make sure to rename binary packages as in Section [5.7.1.3](#).

5.21 debconf

Пакет **debconf** позволяет нам настраивать пакеты в ходе их установки двумя основными способами:

- неинтерактивно из предпосевных настроек **программы установки Debian**.
- интерактивно из меню-ориентированного интерфейса (**dialog**, **gnome**, **kde**, ...)
- установка пакета: вызывается командой **dpkg**
- установленный пакет: вызывается командой **dpkg-reconfigure**

Всё взаимодействие с пользователем в ходе установки пакета должны обрабатываться системой **debconf** с помощью следующих файлов.

- **debian/binarypackage.config**
 - Этот **config**-сценарий **debconf** используется для того, чтобы задавать любые вопросы, необходимые для настройки пакета.
- **debian/двоичныйпакет.template**
 - Этот **templates**-файл **debconf** используется для того, чтобы задавать любые вопросы, необходимые для настройки пакета.
- сценарии настройка пакета
 - **debian/двоичныйпакет.preinst**
 - **debian/двоичныйпакет.prerm**
 - **debian/двоичныйпакет.postinst**
 - **debian/двоичныйпакет.postrm**

See **dh_installdebconf(1)**, **debconf(7)**, **debconf-devel(7)** and [3.9.1 Prompting in maintainer scripts](#) in the “Debian Policy Manual”.

5.22 Multiarch

Multiarch support for cross-architecture installation of binary packages (particularly **i386** and **amd64**, but also other combinations) in the **dpkg** and **apt** packages introduced in Debian **wheezy** (7.0, May 2013), demands that we pay extra attention to packaging.

Вам следует подробно ознакомиться со следующей справочной документацией:

- Ubuntu вики (основная ветка разработки)
 - [MultiarchSpec](#)
- Debian вики (ситуация в Debian)

- [Поддержка мультиархитектурности в Debian](#)
- [Multiarch/Implementation](#)

Мультиархитектурность включается с помощью значения **<тройки>** вида **i386-linux-gnu** или **x86_64-linux-gnu** в пути установки разделяемых библиотек вида **/usr/lib/<тройка>/** и т. д.

- Значение **<тройки>**, внутренне необходимое для сценариев **debhelper**, устанавливается самими сценариями неявно. Сопровождающему не нужно об этом беспокоиться.
- The **<triplet>** value used in **override_dh_*** target scripts must be explicitly set in the **debian/rules** file by the maintainer. The **<triplet>** value is stored in the **\$(DEB_HOST_MULTIARCH)** variable in the following **debian/rules** snippet example:

```
DEB_HOST_MULTIARCH = $(shell dpkg-architecture -qDEB_HOST_MULTIARCH)
...
override_dh_install:
    mkdir -p package1/lib/$(DEB_HOST_MULTIARCH)
    cp -dR tmp/lib/. package1/lib/$(DEB_HOST_MULTIARCH)
```

См.:

- [Section 5.6.4](#)
- [dpkg-architecture\(1\)](#)
- [Section 5.7.1.1](#)
- [Section 5.7.1.2](#)

5.22.1 Путь мультиархитектурной библиотеки

Debian policy requires following [Filesystem Hierarchy Standard](#). Its [/usr/lib : Libraries for programming and packages](#) states **”/usr/lib** includes object files, libraries, and internal binaries that are not intended to be executed directly by users or shell scripts.”

Debian policy makes an exception to the [Filesystem Hierarchy Standard](#) to use **/usr/lib/<triplet>/** instead of **/usr/lib<qual>/** (e.g., **/lib32/** and **/lib64/**) to support a multiarch library.

Table 5.1 Опции пути мультиархитектурных библиотек

Классический путь	Мультиархитектурный путь для i386	Мультиархитектурный путь для amd64
/lib/	/lib/i386-linux-gnu/	/lib/x86_64-linux-gnu/
/usr/lib/	/usr/lib/i386-linux-gnu/	/usr/lib/x86_64-linux-gnu/

Для пакетов на основе Autotools, в которых используется пакет **debhelper** с (**compat**>=9), установка этого пути выполняется автоматически с помощью команды **dh_auto_configure**.

При работе с другими пакетами, использующими неподдерживаемые системы сборки, вам следует вручную изменить путь установки указанным ниже способом.

- If **./configure** is used in the **override_dh_auto_configure** target in **debian/rules**, make sure to replace it with **“dh_auto_configure --”** while re-targeting the install path from **/usr/lib/** to **/usr/lib/\$(DEB_HOST_MULTIARCH)/**.
- Замените все пути с **/usr/lib/** на **/usr/lib/*/** в файлах **debian/foo.install**.

All files installed simultaneously as the multiarch package to the same file path should have exactly the same file content. You must be careful with differences generated by the data byte order and by the compression algorithm.

Note



The `--libexecdir` option of the `.configure` command specifies the default path to install executable programs run by other programs rather than by users. Its Autotools default is `/usr/libexec/` but its Debian non-multi-arch default is `/usr/lib/`. If such executables are a part of a "Multi-arch: foreign" package, a path such as `/usr/lib/` or `/usr/lib/package-name` may be more desirable than `/usr/lib/<triplet>/`, which `dh_auto_configure` uses. The [GNU Coding Standards: 7.2.5 Variables for Installation Directories](#) has a description for `libexecdir` as "The definition of `libexecdir` is the same for all packages, so you should install your data in a subdirectory thereof. Most packages install their data under `$(libexecdir)/package-name/ ...`". (It is always a good idea to follow GNU unless it conflicts with the Debian policy.)

Файлы разделяемых библиотек, расположенные в каталогах по умолчанию, `/usr/lib/` и `/usr/lib/<тройка>/`, загружаются автоматически.

For shared library files in another path, the GCC option `-I` must be set by the `pkg-config` command to make them load properly.

5.22.2 Путь мультиархитектурных заголовочных файлов

В мультиархитектурной системе Debian GCC по умолчанию включает и `/usr/include/`, и `/usr/include/<тройка>/`.

If the header file is not in those paths, the GCC option `-I` must be set by the `pkg-config` command to make `"#include <foo.h>"` work properly.

Table 5.2 Опции пути мультиархитектурного заголовочного файла

Классический путь	Мультиархитектурный путь для i386	Мультиархитектурный путь для amd64
<code>/usr/include/</code>	<code>/usr/include/i386-linux-gnu/</code>	<code>/usr/include/x86_64-linux-gnu/</code>
<code>/usr/include/имяпакета/</code>	<code>/usr/include/i386-linux-gnu/имяпакета/</code>	<code>/usr/include/x86_64-linux-gnu/имяпакета/</code>
	<code>/usr/lib/i386-linux-gnu/имяпакета/</code>	<code>/usr/lib/x86_64-linux-gnu/имяпакета/</code>

The use of the `/usr/lib/<triplet>/package-name/` path for the library files allows the upstream maintainer to use the same install script for the multiarch system with `/usr/lib/<triplet>` and the biarch system with `/usr/lib<qual>/`.⁸

The use of the file path containing `package-name` enables having more than 2 development libraries simultaneously installed on a system.

5.22.3 Мультиархитектурный путь к файлу *.pc

Программа `pkg-config` используется для получения информации об установленных в системе библиотеках. Она сохраняет свои параметры настройки в файле `*.pc` и используется для установки опций `-I` и `-L` для GCC.

Table 5.3 Опции пути к файлу *.pc

Классический путь	Мультиархитектурный путь для i386	Мультиархитектурный путь для amd64
<code>/usr/lib/pkgconfig/</code>	<code>/usr/lib/i386-linux-gnu/pkgconfig/</code>	<code>/usr/lib/x86_64-linux-gnu/pkgconfig/</code>

⁸This path is compliant with the FHS. [Filesystem Hierarchy Standard: /usr/lib : Libraries for programming and packages](#) states "Applications may use a single subdirectory under `/usr/lib`. If an application uses a subdirectory, all architecture-dependent data exclusively used by the application must be placed within that subdirectory."

5.23 Усиление безопасности компилятора

The compiler hardening support spreading for Debian **jessie** (8.0, TBA) demands that we pay extra attention to the packaging.

Вам следует подробно ознакомиться со следующей справочной документацией:

- Debian вики [Усиление безопасности](#)
- Debian вики [Пошаговое руководство по усилению безопасности](#)

Команда **debmake** добавляет шаблонные комментарии в файл **debian/rules**, требующиеся для **DEB_BUILD_MAINT_**, **DEB_CFLAGS_MAINT_APPEND** и **DEB_LDFLAGS_MAINT_APPEND** (см. Chapter 4 и **dpkg-buildflags(1)**).

5.24 Непрерывная интеграция

DEP-8 defines the **debian/tests/control** file as the RFC822-style test metadata file for [continuous integration](#) (CI) of the Debian package.

It is used after building the binary packages from the source package containing this **debian/tests/control** file. When the **autopkgtest** command is run, the generated binary packages are installed and tested in the virtual environment according to this file.

See documents in the **/usr/share/doc/autopkgtest/** directory and [4. autopkgtest: Automatic testing for packages](#) of the “Ubuntu Packaging Guide”.

Note



Testing of the binary packages during their building time can be accommodated by **dh_auto_test**.

Кроме того, в Debian существует ещё несколько других инструментов непрерывной интеграции.

- The **debci** package: CI platform on top of the **autopkgtest** package
- Пакет **jenkins**: платформа непрерывной интеграции общего назначения

5.25 Other new requirements

Debian packaging practices are moving target. Please keep your eyes on [DEP - Debian Enhancement Proposals](#).

5.26 Предзагрузка

Debian cares about supporting new ports or flavours. The new ports or flavours require [bootstrapping](#) operation for the cross-build of the initial minimal native-building system. In order to avoid build-dependency loops during bootstrapping, the build-dependency needs to be reduced using the [profile](#) builds feature.

Tip



Если сборка базового пакета **foo** зависит от пакета **bar**, имеющего глубокую цепочку сборочных зависимостей, но **bar** используется только в цели **test** в **foo**, вы можете пометить **bar** меткой **<!nocheck>** в поле **Build-depends** пакета **foo**, что позволит избежать циклов при сборке.

5.27 Bug reports

Поведение команды **reportbug**, используемой для отправки отчётов об ошибке в *двоичном пакете*, может быть настроено с помощью файлов в каталоге **usr/share/bug/двоичныйпакет/**.

Команда **dh_bugfiles** устанавливает эти файлы из шаблонных файлов в каталоге **debian/**.

- **debian/двоичныйпакет.bug-control** → **usr/share/bug/двоичныйпакет/control**
 - Этот файл содержит некоторые указания, такие как перенаправления отчёта об ошибке другому пакету.
- **debian/двоичныйпакет.bug-presubj** → **usr/share/bug/двоичныйпакет/presubj**
 - Этот файл отображается пользователю с помощью команды **reportbug**.
- **debian/двоичныйпакет.bug-script** → **usr/share/bug/двоичныйпакет** или **usr/share/bug/двоичныйпакет/script**
 - Команда **reportbug** запускает этот сценарий для создания шаблонного файла для отчёта об ошибке.

См. **dh_bugfiles(1)** и [Возможности reportbug для разработчиков](#)

Tip



If you always remind the bug reporter of something or ask them about their situation, use these files to automate it.

Chapter 6

Опции debmake

Ниже приводятся некоторые наиболее важные опции команды **debmake**.

6.1 Опции быстрых действий (-a, -i)

Команда **debmake** предлагает 2 опции для выполнения быстрых действий.

- **-a** : открыть tar-архив основной ветки
- **-i** : выполнить сценарий для сборки двоичного пакета

Действия из примера, приведённого выше в Chapter 4, можно выполнить с помощью следующей простой команды.

```
$ debmake -a package-1.0.tar.gz -i debuild
```

Tip



A URL such as “<https://www.example.org/DL/package-1.0.tar.gz>” may be used for the **-a** option.

Tip



A URL such as “<https://arm.koji.fedoraproject.org/packages/ibus/1.5.7/3.fc21/src/ibus-1.5.7-3.fc21.src.rpm>” may be used for the **-a** option, too.

6.1.1 Модуль Python

You can generate a functioning single binary Debian package with a reasonable package description directly from the Python module package offered as a tarball, *pythonmodule-1.0.tar.gz*. The **-b** option specifying the package type **python** and the **-s** option to copy the package description from the upstream package need to be specified.

```
$ debmake -s -b:'python' -a pythonmodule-1.0.tar.gz -i debuild
```

For other interpreted languages that support the **-b** option, specify the pertinent *type* for the **-b** option.

For interpreted languages without the **-b** option support, specify the **script** type instead and add the interpreter package as a dependency of the resulting binary package by adjusting the **debian/control** file.

6.2 Snapshot upstream tarball (-d, -t)

This test building scheme is good for the git repository organized as described in **gbp-buildpackage(7)** which uses the master, upstream, and pristine-tar branches.

The upstream snapshot from the upstream source tree in the VCS can be made with the **-d** option if the upstream package supports the “**make dist**” equivalence.

```
$ cd /path/to/upstream-vcs
$ debmake -d -i debuild
```

С другой стороны, то же самое можно сделать с помощью опции **-t** в том случае, если с помощью команды **tar** можно создать tar-архив основной ветки.

```
$ cd /path/to/upstream-vcs
$ debmake -p package -t -i debuild
```

Если вы не укажете версию основной ветки с помощью опции **-u** или в файле **debian/changelog**, то версия среза основной ветки будет создана в формате **0~%y%m%d%H%M** из текущих даты и времени по UTC, напр., *0~1403012359*.

Если система управления версиями основной ветки размещена в каталоге *пакет/*, а не в каталоге *upstream-vcs/*, то «**-p пакет**» можно пропустить.

If the upstream source tree in the VCS contains the **debian/*** files, the **debmake** command with either the **-d** option or the **-t** option combined with the **-i** option automates the making of a non-native Debian package from the VCS snapshot while using these **debian/*** files.

```
$ cp -r /path/to/package-0~1403012359/debian/. /path/to/upstream-vcs/debian
$ dch
... update debian/changelog
$ git add -A .; git commit -m "vcs with debian/*"
$ debmake -t -p package -i debuild
```

This **non-native** Debian binary package building scheme without the real upstream tarball using the “**debmake -t -i debuild**” command may be considered as the **quasi-native** Debian package scheme since the packaging situation resembles the **native** Debian binary package building case using the **debuild** command without the upstream tarball.

Use of a **non-native** Debian package building scheme helps to ease communication with the downstream distros such as Ubuntu for bug fixes etc.

6.3 debmake -cc

The **debmake** command with the **-cc** option can make a summary of the copyright and license for the entire source tree to standard output.

```
$ tar -xvzf package-1.0.tar.gz
$ cd package-1.0
$ debmake -cc | less
```

Опция **-c** позволяет получить более краткий отчёт.

6.4 debmake -k

При обновлении пакета до нового выпуска основной ветки команда **debmake** может проверить содержимое существующего файла **debian/copyright** и сравнить его с информацией об авторских правах и лицензировании для всего обновлённого дерева исходного кода целиком.

```
$ cd package-vcs
$ gbp import-orig --uscan --pristine-tar
... update source with the new upstream release
$ debmake -k | less
```

The “**debmake -k**” command parses the **debian/copyright** file from the top to the bottom and compares the license of all the non-binary files in the current package with the license described in the last matching file pattern entry of the **debian/copyright** file.

При редактировании автоматически созданного файла **debian/copyright** убедитесь, что наиболее общие шаблоны файлов помещены в верхней части списка.

Tip



Для всех новых выпусков основной ветки запускайте команду «**debmake -k**», чтобы файл **debian/copyright** оставался актуальным.

6.5 debmake -j

The generation of a functioning multi-binary package always requires more manual work than that of a functioning single binary package. The test build of the source package is the essential part of it.

Например, создадим пакет из того же архива *package-1.0.tar.gz* (см. Chapter 4) с поддержкой набора из нескольких двоичных пакетов.

- Запустите команду **debmake** с опцией **-j** для выполнения тестовой сборки и создания отчёта.

```
$ debmake -j -a package-1.0.tar.gz
```

- Check the last lines of the *package.build-dep.log* file to judge build dependencies for **Build-Depends**. (You do not need to list packages used by **debhelper**, **perl**, or **fakeroot** explicitly in **Build-Depends**. This technique is useful for the generation of a single binary package, too.)
- Проверьте содержимое файла *пакет.install.log* для определения путей установки файлов, чтобы решить, как разделить эти файлы на несколько пакетов.
- Начните работу над пакетом с помощью команды **debmake**.

```
$ rm -rf package-1.0
$ tar -xvzf package-1.0.tar.gz
$ cd package-1.0
$ debmake -b"package1:type1, ..."
```

- Обновите файлы **debian/control** и **debian/двоичныйпакет.install**, используя полученную выше информацию.
- При необходимости обновите другие файлы **debian/***.
- Build the Debian package with the **debuild** command or its equivalent.

```
$ debuild
```

- All binary package entries specified in the **debian/binarypackage.install** file are generated as *binarypackage_version-revision_arch.deb*.

Note



Опция **-j** для команды **debmake** вызывает **dpkg-depcheck(1)** для запуска **debian/rules** под управлением **strace(1)** с целью получения зависимостей от библиотек. К сожалению, это выполняется очень медленно. Если вам известны зависимости от библиотек из других источников, например, файла SPEC, то можно просто запустить команду «**debmake ...**» без опции **-j** и запустить команду «**debian/rules install**», чтобы проверить пути установки созданных файлов.

6.6 debmake -x

Количество шаблонных файлов, создаваемых командой **debmake** зависит от опции **-x[01234]**.

- See Section 8.1 for cherry-picking of the template files.

Note



Команда **debmake** не меняет ни один из существующих файлов настройки.

6.7 debmake -P

Команда **debmake**, запущенная с опцией **-P**, педантично проверяет создаваемые автоматически файлы на предмет наличия текста об авторских правах и лицензировании, даже если они подпадают под действие разрешительной лицензии.

This option affects not only the content of the **debian/copyright** file generated by normal execution, but also the output by the execution with the **-k**, **-c**, **-cc**, and **-ccc** options.

6.8 debmake -T

The **debmake** command invoked with the **-T** option additionally prints verbose tutorial comment lines. The lines marked with **###** in the template files are part of the verbose tutorial comment lines.

Chapter 7

Полезные советы

Here are some notable tips about Debian packaging.

Debian now enforces the source-only upload when developing packages. So there may be 2 different *.changes files:

- 'package_version-revision'_*source.changes* for the normal source-only upload
- *package_version-revision_arch.changes* for the binary upload

7.1 debdiff

Можно сравнивать содержимое файлов в двух пакетах Debian с исходным кодом с помощью команды **debdiff**.

```
$ debdiff old-package.dsc new-package.dsc
```

Также можно сравнивать списки файлов в двух наборах двоичных пакетов Debian с помощью команды **debdiff**.

```
$ debdiff old-package.changes new-package.changes
```

Это полезно для определения изменений в пакетах с исходным кодом и для проверки на предмет нечаянных изменений, принесённых при обновлении двоичных пакетов, таких как непреднамеренное ошибочное размещение или удаление файлов.

7.2 dget

Можно скачать набор файлов для пакета Debian с исходным кодом с помощью команды **dget**.

```
$ dget https://www.example.org/path/to/package_version-rev.dsc
```

7.3 debc

Созданные пакеты следует установить с помощью команды **debc** для их локальной проверки.

```
$ debc package_version-rev_arch.changes
```

7.4 piuparts

Созданные пакеты следует установить с помощью команды **piuparts** для их автоматической проверки.

```
$ sudo piuparts package_version-rev_arch.changes
```

Note



This is a very slow process with remote APT package repository access.

7.5 debsign

После завершения тестирования пакета можно его подписать с помощью команды **debsign**.

For uploading source-only package (normal):

```
$ debsign package_version-rev_source.changes
```

For uploading binary package:

```
$ debsign package_version-rev_arch.changes
```

7.6 dput

After signing the package with the **debsign** command, you can upload the set of files for the Debian source and binary packages with the **dput** command.

For uploading source-only package (normal):

```
$ dput package_version-rev_source.changes
```

For uploading binary package:

```
$ dput package_version-rev_arch.changes
```

7.7 bts

После загрузки пакета вы начнёте получать сообщения об ошибках. Соответствующая работа с ошибками является важным обязательством сопровождающего, что описано в разделе 5.8. [Работа с ошибками](#) «Справочника разработчика Debian».

The **bts** command is a handy tool to manage bugs on the [Debian Bug Tracking System](#).

```
$ bts severity 123123 wishlist , tags -1 pending
```

7.8 git-buildpackage

Пакет **git-buildpackage** предлагает множество команд для автоматизации деятельности по работе над пакетом с помощью репозитория git.

- **gbp import-dsc**: импортировать предыдущий пакет Debian с исходным кодом в git-репозиторий.
- **gbp import-orig**: импортировать новый tar-архив основной ветки в git-репозиторий.
 - The **--pristine-tar** option for the **git import-orig** command enables storing the upstream tarball in the same git repository.
 - The **--uscan** option as the last argument of the **gbp import-orig** command enables downloading and committing the new upstream tarball into the git repository.
- **gbp dch**: создание записи журнала изменений Debian из сообщений о коммитах.
- **gbp buildpackage**: сборка двоичного пакета Debian из git-репозитория.

- The **sbuild** package can be used as its clean chroot build backend either by configuration or adding `--git-builder='sbuild -A -s --source-only-changes -v -d unstable'`
- **gbp pull**: безопасно обновить ветки **debian**, **upstream** и **pristine-tar** из удалённого репозитория.
- Команды **gbp pq**, **git-dpm** или **quilt** (или её псевдоним **dquilt**) используются для работы с заплатами quilt.
 - The **dquilt** command is the simplest to learn and requires you to commit the resulting files manually with the **git** command to the **master** branch.
 - The “**gbp pq**” command provides the equivalent functionality of patch set management without using **dquilt** and eases including upstream git repository changes by cherry-picking.
 - The “**git dpm**” command provides more enhanced functionality than that of the ‘**gbp pq**’ command.

Package history management with the **git-buildpackage** package is becoming the standard practice for most Debian maintainers.

См.:

- [Сборка пакетов Debian с помощью git-buildpackage](#)
- <https://wiki.debian.org/GitPackagingWorkflow>
- <https://wiki.debian.org/GitPackagingWorkflow/DebConf11BOF>
- <https://raphaelhertzog.com/2010/11/18/4-tips-to-maintain-a-3-0-quilt-debian-source-package-in-a-vcs/>
- Практическая документация по работе над пакетом **systemd** в [Сборка из исходного кода](#).

Tip



Расслабьтесь. Вам не нужно использовать все обёртки и инструменты. Используйте только те, которые подходят под ваши нужды.

7.8.1 gbp import-dscs --debsnap

Для пакета Debian с исходным кодом с именем `<source-package>`, сохранённого в архиве snapshot.debian.org начальный git-репозиторий со всей историей версий Debian можно получить следующим образом.

```
$ gbp import-dscs --debsnap --pristine-tar '<source-package>'
```

7.8.2 git-репозиторий основной ветки

For Debian packaging with the **git-buildpackage** package, the **upstream** branch on the remote repository **origin** is normally used to track the content of the released upstream tarball.

git-репозиторий основной ветки также может отслеживаться, его удалённый репозиторий следует назвать **upstream**, а не **origin** как по умолчанию. Теперь вы легко сможете избирательно применять наиболее свежие изменения основной ветки к редакции Debian с помощью команды **gitk** и используя команду **gbp-pq**.

Tip



The “**gbp import-orig --upstream-vcs-tag**” command can create a nice packaging history by making a merge commit into the **upstream** branch from the specified tag on the upstream git repository.

Caution



Содержимое выпущенного разработчиками основной ветки tar-архива может не совпадать в точности с соответствующим содержимым git-репозитория основной ветки. Оно может содержать созданные автоматически файлы, либо какие-то файлы могут отсутствовать. (Autotools, distutils, ...)

7.9 dgit

The new **dgit** package offers commands to automate packaging activities using the git repository as an alternative to still popular **gbp-buildpackage**. Please read their guide:

- **dgit-maint-merge(7)** — for dpkg-source "Format: 3.0 (quilt)" package with its changes flowing both ways between the upstream Git repository and the Debian Git repository which are tightly coupled.
- **dgit-maint-debbase(7)** — for dpkg-source "Format: 3.0 (quilt)" package with its changes flowing mostly one way from the upstream Git repository to the Debian Git repository.
- **dgit-maint-gbp(7)** — for dpkg-source "Format: 3.0 (quilt)" package with its Debian Git repository which is kept usable also for people using **gbp-buildpackage(1)**.
- **dgit-maint-native(7)** — for dpkg-source "Format: 3.0 (native)" package in the Debian Git repository.

The **dgit(1)** command can push the easy-to-trace change history to the <https://browse.dgit.debian.org/> site and can upload Debian package to the Debian repository properly without using **dput(1)**.

- **dgit push-source** — for uploading source-only package (normal)
- **dgit push** — for uploading binary package

If you already use `main` and `upstream` branches in the Debian packaging repository (`salsa.debian.org`), you can use your local `upstream-vcs-main` branch to track the `main` branch of the `upstream-vcs` repository as:

```
$ git fetch upstream-vcs main:upstream-vcs-main
```

Cherry picking bug fix commits from latest upstream commits are intuitive operation with **dgit-maint-merge(7)** and **dgit-maint-debbase(7)**. Just right-click those commits interactively on the `upstream-vcs-main` branch of the **gitk(1)** GUI dialog.

7.9.1 Snapshot upstream tarball (alternative git deborig approach)

For **quasi-native** Debian binary package scheme situation described in Section 6.2 can be addressed using alternative `git deborig` approach adopting the **dgit-maint-merge(7)** scheme when `debian/change log` contains the non-native version number with revision like `0.16-1`.

```
$ cd /path/to/upstream-git
$ git rm -rf debian
$ git tag -s upstream/0.16
$ git reset --hard HEAD^
$ git deborig
$ sbuild
```

Here, for source format `3.0 (quilt)`, removal of files under `debian/` directory in the upstream tarball is non-essential operation to quiet false positive warning from lintian.

For `-1` revision, this use of **git-deborig(1)** as above is how this `debmake-doc` package generates the upstream tarball.

For `-2`, `-3`, ... revisions, you need to fetch and use the uploaded upstream tarball instead. For this, **origtargz(1)** may be handy.

7.10 chroot

The **chroot** for a clean package build environment can be created and managed using the tools described in Chapter 3. 1

Ниже приводится краткий обзор доступных команд для сборки пакетов. Существует множество способов сделать одно и то же.

- **dpkg-buildpackage** = ядро инструмента для сборки пакета
- **debuild** = **dpkg-buildpackage** + **lintian** (сборка с очищенными переменными окружения)
- **schroot** = core of the Debian chroot environment tool
- **sbuild** = **schroot** + **dpkg-buildpackage** (build in the chroot)
- **gbp** = manage the Debian source under git
- **gbp buildpackage** = **sbuild** + **dpkg-buildpackage** + **gbp** (if configured with **sbuild**)

A clean **unstable** distribution chroot environment can be used as follows.

- The chroot filesystem creation command for the **unstable** distribution
 - **sudo sbuild-createtchroot --include=auto-apt-proxy unstable /srv/chroot/unstable-amd64-sbuild <http://deb.debian.org/debian>**
- The master chroot filesystem path for the **unstable** distribution chroot filesystem
 - path: **/srv/chroot/unstable-amd64-sbuild**
- The package build command for the **unstable** distribution chroot
 - **sbuild**
 - **gbp buildpackage**
- The command to update the **unstable** chroot
 - **sudo sbuild-update -udcar u**
- The command to login to the **unstable** chroot filesystem to modify it
 - **sudo sbuild-shell u**

Here, single "u" is predefined short alias for "unstable".

Произвольное окружение выпуска *выпуск* может быть использовано следующим образом.

- Команда создания файловой системы chroot-окружения для выпуска *выпуск*
 - **sudo sbuild-createtchroot --include=auto-apt-proxy dist /srv/chroot/dist-amd64-sbuild <http://deb.debian.org/debian>**
- Главный путь файловой системы chroot-окружения для chroot-окружения выпуска *выпуск*
 - path: **/srv/chroot/dist-amd64-sbuild**
- Команда сборки пакета для chroot-окружения выпуска *выпуск*
 - **sbuild -d dist**
 - **gbp buildpackage --git-dist=выпуск**
- Команда для обновления chroot-окружения *выпуск*
 - **sudo sbuild-update -udcar dist**

¹The **sbuild** style chroot organization is deployed here. See <https://wiki.debian.org/sbuild>. Be careful since many HOWTOs use different chroot setups.

- The command to login to the master chroot (**source:dist**) of the *dist* distribution environment to modify it persistently

– **sudo sbuild-shell dist**

Tip



For building new experimental packages or for debugging buggy packages without using slow network access, this “**sudo sbuild-shell dist**” command can be used to pre-load some packages. Make sure to retain the backup of the original minimal chroot since changes are persistent and interfere with the normal test package building with the minimal chroot.

Tip



If your old chroot filesystem is missing packages such as **libeatmydata1**, **ccache**, and **lintian**, you may want to install these with the “**sudo sbuild-shell dist**” command.

Tip



When the **orig.tar.gz** file needs to be uploaded for a Debian revision other than **0** or **1** (e.g., for a security upload), add the **-sa** option to the end of **dpkg-buildpackage**, **debuild**, and **sbuild** commands. For the “**gbp buildpackage**” command, temporarily modify the **builder** setting of **~/l.gbp.conf**.

Note



Описание в этом разделе слишком кратко, чтобы оно оказалось полезным для большинства будущих сопровождающих. Это — намеренный выбор автора. Настоятельно рекомендуется поискать и прочесть всю соответствующую документация, связанную с используемыми командами.

Note



When an essential package becomes a non-essential one (e.g., **adduser**), you need to remove it manually from the existing chroot environment for its use by **piuparts**.

7.11 Новая редакция Debian

Let’s assume that a bug report **#bug_number** was filed against your package, and it describes a problem that you can solve by editing the **buggy** file in the upstream source. Here’s what you need to do to create a new Debian revision of the package with the **bugname.patch** file recording the fix.

Новая редакция Debian с помощью команды dquilt

```
$ dquilt push -a
$ dquilt new bugname.patch
$ dquilt add buggy
$ vim buggy
...
$ dquilt refresh
$ dquilt header -e
$ dquilt pop -a
$ dch -i
```

Alternatively if the package is managed in the git repository using the **git-buildpackage** command with its default configuration:

Новая редакция Debian с помощью команды gbp-pq

```
$ git checkout master
$ gbp pq import
$ vim buggy
$ git add buggy
$ git commit
$ git tag pq/<newrev>
$ gbp pq export
$ gbp pq drop
$ git add debian/patches/*
$ dch -i
$ git commit -a -m "Closes: #<bug_number>"
```

Проверьте, что вы лаконично описали изменения, исправляющие ошибку, и закрыли сообщение об ошибке, добавив «**Closes: #<номер_ошибки>**» в файл **debian/changelog**.

Tip

Когда вы экспериментируете, то используйте запись **debian/changelog** со строкой версии вида **1.0.1-1~rc1**. Затем сведите такие записи **журнала изменений** в одну запись для официального пакета.

7.12 Новый выпуск основной ветки

Если пакет **foo** создан в одном из современных форматов «**3.0 (native)**» или «**3.0 (quilt)**», то работа над пакетом с новым выпуском основной ветки будет представлять собой по сути перемещение старого каталога **debian/** в новый исходный код. Это можно сделать, запустив команду «**tar -xvzf /путь/к/foo_стараяверсия.debian.tar.gz**» в каталоге с распакованным новым исходным кодом. ² Конечно, вам придётся сделать некоторую рутинную работу.

Для такой ситуации существует несколько инструментов. После обновления до нового выпуска основной ветки с помощью этих инструментов убедитесь, что вы лаконично описали изменения в новом выпуске основной ветки, исправляющие известные ошибки, и закрыли эти сообщения об ошибках, добавив «**Closes: #номер_ошибки**» в файл **debian/changelog**.

7.12.1 uupdate + tarball

You can automatically update to the new upstream source with the **uupdate** command from the **devscripts** package. It requires having the old Debian source package and the new upstream tarball.

```
$ wget https://example.org/foo/foo-newversion.tar.gz
$ cd foo-oldversion
$ uupdate -v newversion ../foo-newversion.tar.gz
...
```

²If a package **foo** is packaged in the old **1.0** format, this can be done by running the “**zcat /path/to/foo_oldversion.diff.gz|patch -p1**” command in the new extracted source, instead.

```
$ cd ../foo-newversion
$ while dquilt push; do dquilt refresh; done
$ dch
```

7.12.2 uscan

You can automatically update to the new upstream source with the **uscan** command from the **devscripts** package. It requires having the old Debian source package and the **debian/watch** file in it.

```
$ cd foo-oldversion
$ uscan
...
$ while dquilt push; do dquilt refresh; done
$ dch
```

7.12.3 gbp

You can automatically update to the new upstream source with the “**gbp import-orig --pristine-tar**” command from the **git-buildpackage** package. It requires having the old Debian source in the git repository and the new upstream tarball.

```
$ ln -sf foo-newversion.tar.gz foo_newversion.orig.tar.gz
$ cd foo-vcs
$ git checkout master
$ gbp pq import
$ git checkout master
$ gbp import-orig --pristine-tar ../foo_newversion.orig.tar.gz
...
$ gbp pq rebase
$ git checkout master
$ gbp pq export
$ gbp pq drop
$ git add debian/patches
$ dch -v <newversion>
$ git commit -a -m "Refresh patches"
```

Tip



If upstream uses a git repository, please also use the **--upstream-vcs-tag** option for the **gbp import-orig** command.

7.12.4 gbp + uscan

You can automatically update to the new upstream source with the “**gbp import-orig --pristine-tar --uscan**” command from the **git-buildpackage** package. It requires having the old Debian source in the git repository and the **debian/watch** file in it.

```
$ cd foo-vcs
$ git checkout master
$ gbp pq import
$ git checkout master
$ gbp import-orig --pristine-tar --uscan
...
$ gbp pq rebase
$ git checkout master
$ gbp pq export
$ gbp pq drop
```

```
$ git add debian/patches
$ dch -v <newversion>
$ git commit -a -m "Refresh patches"
```

Tip



If upstream uses a git repository, please also use the `--upstream-vcs-tag` option for the `gbp import-orig` command.

7.13 Формат исходного кода 3.0

Обновление стиля пакета не является требуемой деятельностью до обновления пакета. Тем не менее, такое обновление позволяет использовать все возможности современной системы `debhelper` и формат исходного кода **3.0**.

- Если вам требуется заново создать удалённые по какой-либо причине шаблонные файлы, вы можете снова запустить `debmake` в том же дереве исходного кода пакета Debian. Замет эти файлы можно отредактировать.
- Если пакет не был должным образом обновлён для использования команды `dh` в файле `debian/rules`, то обновите его (см. Section 5.6.2). Также соответствующим образом обновите файл `debian/control`.
- Если у вас имеется пакет с исходным кодом в формате **1.0** с файлом `foo.diff.gz`, то вы можете обновить его до более нового формате исходного кода «**3.0 (quilt)**», создав файл `debian/source/format` со строкой «**3.0 (quilt)**». Остальные файлы `debian/*` можно просто скопировать. При необходимости импортируйте файл `большой.diff`, созданный командой «`filterdiff -z -x /debian/ foo.diff.gz > big.diff`» в вашу систему quilt. 3
- Если в пакете используется другая система заплат, например, `dpatch`, `db` или `cdbs` с `-p0`, `-p1` или `-p2`, то преобразуйте её в формат заплат quilt с помощью сценария `deb3` из пакета `quilt`.
- Если в пакете используется команда `dh` с опцией «`--with quilt`» или с командами `dh_quilt_patch` и `dh_quilt_unpatch`, то удалите соответствующие вызовы и перейдите на использование более нового формата исходного кода «**3.0 (quilt)**».
- Если у вас имеется пакет с исходным кодом в формате **1.0** без файла `foo.diff.gz`, то вы можете обновить такой пакет до более нового формата исходного кода «**3.0 (native)**», создав файл `debian/source/format` со строкой «**3.0 (native)**». Остальные файлы `debian/*` можно просто скопировать.

Вам следует просмотреть [DEP — Предложения по улучшению Debian](#) и использовать принятые (ACCEPTED) предложения.

Информацию о статусе поддержки новых форматов исходного кода Debian в наборе инструментальных средств Debian см. в [ProjectsDebSrc3.0](#).

7.14 CDBS

Общая система сборки Debian (**CDBS**) является обёрткой над пакетом `debhelper`. **CDBS** построена на основе механизма включения Makefile и настраивается с помощью установки переменных настройки `DEB_*` в файле `debian/rules`.

До добавления команды `dh` в пакет `debhelper` (начиная с версии 7) **CDBS** являлась единственным подходом к созданию простого и ясного файла `debian/rules`.

For many simple packages, the `dh` command alone allows us to make a simple and clean `debian/rules` file now. It is desirable to keep the build system simple and clean by not using the superfluous **CDBS**.

Зможно разделить файл `большой.diff` на несколько небольших инкрементальных заплат с помощью команды `splittediff`.

Note



Neither “the **CDBS** magically does the job for me with less typing” nor “I don’t understand the new **dh** syntax” can be an excuse to keep using the **CDBS** system.

For some complicated packages such as GNOME related ones, the **CDBS** is leveraged to automate their uniform packaging by the current maintainers with justification. If this is the case, please do not bother converting from the **CDBS** to the **dh** syntax.

Note



Если вы работаете вместе с командой сопровождающих, то следуйте установленным традициям и практикам этой команды.

При переходе с использования системы **CDBS** на синтаксис **dh** в качестве справочника используйте следующую документацию:

- [Документация по CDBS](#)
- [Общая система сборки Debian \(CDBS\), FOSDEM 2009](#)

7.15 Сборка с использованием кодировки UTF-8

Локалью по умолчанию в сборочном окружении является **C**.

Некоторые программы, такие как функции **read** из Python3, изменяют своё поведение в зависимости от текущей локали.

Adding the following code to the **debian/rules** file ensures building the program under the **C.UTF-8** locale.

```
LC_ALL := C.UTF-8
export LC_ALL
```

7.16 Преобразование в кодировку UTF-8

Если документация основной ветки разработки закодирована с помощью устаревших кодировок, то лучше будет преобразовать её в **UTF-8**.

Используйте команду **iconv** из пакета **libc-bin** для преобразования кодировки в обычных текстовых файлах.

```
$ iconv -f latin1 -t utf8 foo_in.txt > foo_out.txt
```

Используйте **w3m(1)** для преобразования HTML-файлов в обычные текстовые файлы в кодировке **UTF-8**. При выполнении преобразования убедитесь, что у вас используется локаль **UTF-8**.

```
$ LC_ALL=C.UTF-8 w3m -o display_charset=UTF-8 \
  -cols 70 -dump -no-graph -T text/html \
  < foo_in.html > foo_out.txt
```

Запустите эти сценарии в цели **override_dh_*** файла **debian/rules**.

7.17 Загрузите orig.tar.gz

При первой загрузке пакета в архив вам следует включить в загрузку также и архив с оригинальным исходным кодом, **orig.tar.gz**.

Если номер редакции Debian вашего пакета не является **1** или **0**, то это происходит по умолчанию. В противном случае, вам следует передать опцию **-sa** команде **dpkg-buildpackage**.

- **dpkg-buildpackage -sa**
- **debuild -sa**
- ***sbuid**
- При использовании **gbp buildpackage** отредактируйте файл **~/gbp.conf**.

Tip



С другой стороны использование опции **-sd** приведёт к тому, что архив с оригинальным исходным кодом, **orig.tar.gz**, будет включён в загрузку.

Tip



Security uploads require including the **orig.tar.gz** file.

7.18 Пропущенные загрузки

Если вы создаёте несколько записей в файле **debian/changelog** и пропускаете загрузки, то вам следует создать соответствующий файл ***_changes**, включающий все изменения с последней загрузки. Это можно сделать, передав **dpkg-buildpackage** опцию **-v** с указанием последней загруженной версии, например, **1.2**.

- **dpkg-buildpackage -v1.2**
- **debuild -v1.2**
- **sbuid --debbuildopts -v1.2**
- При использовании **gbp buildpackage** отредактируйте файл **~/gbp.conf**.

7.19 Продвинутые темы работы над пакетом

Полезные советы по приведённым ниже вопросам можно найти в странице руководства **debhelper(7)**:

- отличия поведения инструмента **debhelper** при использовании «**compat <= 8**»
- сборка нескольких двоичных пакетов в нескольких разных условиях сборки
 - создание нескольких копий исходного кода основной ветки
 - вызов нескольких команд «**dh_auto_configure -S ...**» в цели **override_dh_auto_configure**
 - Вызов нескольких команд «**dh_auto_build -S ...**» в цели **override_dh_auto_build**
 - вызов нескольких команд «**dh_auto_install -S ...**» в цели **override_dh_auto_install**
- building **udeb** packages with “**Package-Type: udeb**” in **debian/control** (see [Package-Type](#))
- excluding some packages for the bootstrapping process (see also [BuildProfileSpec](#))
 - добавление полей **Build-Profiles** в записи двоичных пакетов в файле **debian/control**
 - сборка пакетов с переменной окружения **DEB_BUILD_PROFILES**, имеющей в качестве значения имя соответствующего профиля

Полезные советы по приведённым ниже вопросам можно найти в странице руководства **dpkg-source(1)**:

- naming convention for multiple upstream source tarballs
 - *имяпакета_версия.orig.tar.gz*
 - *имяпакета_версия.orig-имякомпонента.tar.gz*
- запись изменений Debian в пакет исходного кода основной ветки
 - `dpkg-source --commit`

7.20 Другие дистрибутивы

Несмотря на то, что в tar-архиве основной ветки имеется вся информация для сборки пакета Debian, не всегда легко понять, какую комбинацию опций следует использовать.

Also, the upstream package may be more focused on feature enhancements and may be less eager about backward compatibilities etc., which are an important aspect of Debian packaging practice.

The leveraging of information from other distributions is an option to address the above issues.

If the other distribution of interest is a Debian derivative one, it is trivial to reuse it.

If the other distribution of interest is an RPM based distribution, see [Repackage src.rpm](#).

Скачивание и открытие файла `src.rpm` может быть выполнено с помощью команды `rget`. (Разместите сценарий `rget` в одном из каталогов, указанных в вашей переменной `PATH`.)

Сценарий `rget`

```
#!/bin/sh
FCSRPM=$(basename $1)
mkdir ${FCSRPM}; cd ${FCSRPM}/
wget $1
rpm2cpio ${FCSRPM} | cpio -diun
```

Во многих tar-архивах основной ветки содержится файл SPEC, который имеет имя *packagename.spec* или *packagename.spec.in* и используется системой RPM. Также он может использоваться в качестве основы для пакета Debian.

7.21 Отладка

Когда вы сталкиваетесь с проблемами сборки или дампом памяти созданных двоичных программ, вам необходимо разрешить их самостоятельно. Это называется **отладкой**.

Это слишком обширная тема, чтобы обсуждать её в настоящем руководстве. Поэтому позвольте просто привести несколько ссылок и полезных советов по использованию типичных инструментов отладки.

- **дамп памяти**
 - “`man core`”
 - Обновите файл «`/etc/security/limits.conf`» так, чтобы он включал в себя следующее:


```
* soft core unlimited
```
 - «`ulimit -c unlimited`» в `~/.bashrc`
 - «`ulimit -a`» для проверки
 - Нажмите `Ctrl-\` или «`kill -ABRT PID`» для того, чтобы создать файл с дампом памяти
- **`gdb`** — отладчик GNU
 - “`info gdb`”
 - «Отладка с помощью GDB» в `/usr/share/doc/gdb-doc/html/gdb/index.html`
- **`strace`** — трассировка системных вызовов и сигналов
 - Используйте сценарий `strace-graph` из каталога `/usr/share/doc/strace/examples/`, чтобы иметь удобную визуализацию в виде дерева

- “**man strace**”
- **ltrace** - трассировка библиотечных вызовов
 - «**man ltrace**»
- «**sh -n script.sh**» — проверка синтаксиса сценариев командной оболочки
- «**sh -x script.sh**» — трассировка сценария командной оболочки
- «**python -m py_compile script.py**» — проверка синтаксиса сценария на языке Python
- «**python -mtrace --trace script.py**» — трассировка сценария на языке Python
- «**perl -I ../libpath -c script.pl**» — проверка синтаксиса сценария на языке Perl
- «**perl -d:Trace script.pl**» — трассировка сценария на языке Perl
 - Install the **libterm-readline-gnu-perl** package or its equivalent to add input line editing capability with history support.
- **lsuf** — вывод списка файлов, открытых процессами
 - “**man lsuf**”

Tip



The **script** command records console outputs.

Tip



The **screen** and **tmux** commands used with the **ssh** command offer secure and robust remote connection terminals.

Tip



A Python- and Shell-like REPL (=READ + EVAL + PRINT + LOOP) environment for Perl is offered by the **reply** command from the **libreply-perl** (new) package and the **re.pl** command from the **libdevel-repl-perl** (old) package.

Tip



The **rlwrap** and **rlfe** commands add input line editing capability with history support to any interactive commands. E.g. “**rlwrap dash -i**”.

Chapter 8

Дополнительные примеры

Есть старая латинская поговорка: «**fabricando fit faber**» («мастер создаётся трудом»).

It is highly recommended to practice and experiment with all the steps of Debian packaging with simple packages. This chapter provides you with many upstream cases for your practice.

Кроме того, это должно служить в качестве вводных примеров для множества тем по программированию.

- Программирование в командной оболочке POSIX, на языках Python3 и C.
- Method to create a desktop GUI program launcher with icon graphics.
- Conversion of a command from [CLI](#) to [GUI](#).
- Conversion of a program to use **gettext** for [internationalization and localization](#): POSIX shell, Python3, and C sources.
- Обзор множества систем сборки: Makefile, Python distutils, Autotools и CMake.

Please note that Debian takes a few things seriously:

- Свободное ПО
- Stability and security of OS
- Универсальная операционная система реализуется через
 - свободный выбор источников и исходных кодов основной ветки разработки,
 - свободный выбор архитектур ЦП, а также
 - свободный выбор языка пользовательского интерфейса.

Знакомство с типичным примером работы над пакетом, представленным в [Chapter 4](#), является предварительным условием для чтения данной главы.

Some details are intentionally left vague in the following sections. Please try to read the pertinent documentation and practice yourself to find them out.

Tip



The best source of a packaging example is the current Debian archive itself. Please use the “[Debian Code Search](#)” service to find pertinent examples.

8.1 Выборочное применение шаблонов

Ниже приводится пример создания простого пакета Debian из исходного кода без какого-либо содержимого в пустом каталоге.

This is a good platform to get all the template files without making a mess in the upstream source tree you are working on.

Допустим, пустым каталогом будет **debhello-0.1**.

```
$ mkdir debhello-0.1
$ tree
.
+-- debhello-0.1

1 directory, 0 files
```

Создадим максимальное число шаблонных файлов, указав опцию **-x4**.

Let's also use the "**-p debhello -t -u 0.1 -r 1**" options to make the missing upstream tarball.

```
$ debmake -t -p debhello -u 0.1 -r 1 -x4
I: set parameters
I: =====
...
I: sanity check of parameters
W: -p "debhello" != auto set value ""
W: -u "0.1" != auto set value "0~2107020726"
W: -r "1" != auto set value ""
...
I: debmake -x "4" ...
I: creating => debian/control
I: creating => debian/copyright
I: substituting => /usr/share/debmake/extra0/changelog
...
I: creating => debian/license-examples/GPL-3.0+
I: substituting => /usr/share/debmake/extra4/LGPL-2.1+
I: creating => debian/license-examples/LGPL-2.1+
I: substituting => /usr/share/debmake/extra4/LGPL-3.0+
I: creating => debian/license-examples/LGPL-3.0+
I: $ wrap-and-sort
```

Проверим созданные шаблонные файлы.

```
$ cd ..
$ tree
.
+-- debhello-0.1
|   +-- debian
|   |   +-- README.Debian
|   |   +-- changelog
|   |   +-- clean
|   |   +-- compat.ex
|   |   +-- control
|   |   +-- copyright
|   |   +-- debhello.bug-control.ex
|   |   +-- debhello.bug-presubj.ex
|   |   +-- debhello.bug-script.ex
|   |   +-- debhello.conffiles.ex
|   |   +-- debhello.cron.d.ex
|   |   +-- debhello.cron.daily.ex
|   |   +-- debhello.cron.hourly.ex
|   |   +-- debhello.cron.monthly.ex
|   |   +-- debhello.cron.weekly.ex
|   |   +-- debhello.default.ex
|   |   +-- debhello.dirs.ex
|   |   +-- debhello.emacsen-install.ex
|   |   +-- debhello.emacsen-remove.ex
|   |   +-- debhello.emacsen-startup.ex
|   |   +-- debhello.init.ex
|   |   +-- debhello.lintian-overrides.ex
|   |   +-- debhello.service.ex
|   |   +-- debhello.tmpfile.ex
|   |   +-- debhello.upstart.ex
|   |   +-- install
```

```

|   +-- license-examples
|   |   +-- Apache-2.0
|   |   +-- Artistic-1.0
|   |   +-- BSD-3-Clause
|   |   +-- Expat
|   |   +-- GPL-2.0+
|   |   +-- GPL-3.0+
|   |   +-- LGPL-2.1+
|   |   +-- LGPL-3.0+
|   +-- links
|   +-- manpage.1.ex
|   +-- manpage.asciidoc.ex
|   +-- manpage.sgml.ex
|   +-- manpage.xml.ex
|   +-- patches
|   |   +-- series
|   +-- postinst
|   +-- postrm
|   +-- preinst
|   +-- prerm
|   +-- rules
|   +-- source
|   |   +-- control
|   |   +-- format
|   |   +-- local-options
|   |   +-- options
|   |   +-- patch-header
|   +-- source.lintian-overrides.ex
|   +-- tests
|   |   +-- control
|   +-- upstream
|   |   +-- metadata
|   +-- watch
+-- debhello-0.1.tar.gz
+-- debhello_0.1.orig.tar.gz -> debhello-0.1.tar.gz

7 directories, 56 files

```

Теперь вы можете скопировать любой из созданных в каталоге *debhello-0.1/debian/* шаблонных файлов в ваш пакет, при необходимости их переименовав.

Tip



The generated template files can be made more verbose by invoking the **debmake** command with the **-T** option (tutorial mode).

8.2 Без Makefile (командная оболочка, интерфейс командной оболочки)

Ниже приводится пример создания простого пакета Debian из программы с интерфейсом командной оболочки, написанной для командной оболочки POSIX и не имеющей системы сборки.

Допустим tar-архив основной ветки имеет имя **debhello-0.2.tar.gz**.

Этот тип исходного кода не имеет средств автоматизации, и файлы должны быть установлены вручную.

For example:

```

$ tar -xzf debhello-0.2.tar.gz
$ cd debhello-0.2
$ sudo cp scripts/hello /bin/hello
...

```

Let's get this source as tar file from a remote site and make it the Debian package.

Загрузим debhello-0.2.tar.gz

```
$ wget http://www.example.org/download/debhello-0.2.tar.gz
...
$ tar -xzf debhello-0.2.tar.gz
$ tree
.
+-- debhello-0.2
|   +-- LICENSE
|   +-- data
|       |   +-- hello.desktop
|       |   +-- hello.png
|   +-- man
|       |   +-- hello.1
|   +-- scripts
|       +-- hello
+-- debhello-0.2.tar.gz

4 directories, 6 files
```

Итак, сценарий командной оболочки POSIX **hello** является очень простым.

hello (v=0.2)

```
$ cat debhello-0.2/scripts/hello
#!/bin/sh -e
echo "Hello from the shell!"
echo ""
echo -n "Type Enter to exit this program: "
read X
```

Here, **hello.desktop** supports the [Desktop Entry Specification](#).

hello.desktop (v=0.2)

```
$ cat debhello-0.2/data/hello.desktop
[Desktop Entry]
Name=Hello
Name[fr]=Bonjour
Comment=Greetings
Comment[fr]=Salutations
Type=Application
Keywords=hello
Exec=hello
Terminal=true
Icon=hello.png
Categories=Utility;
```

Here, **hello.png** is the icon graphics file.

Let's package this with the **debmake** command. Here, the **-b':sh'** option is used to specify that the generated binary package is a shell script.

```
$ cd debhello-0.2
$ debmake -b':sh'
I: set parameters
...
I: sanity check of parameters
I: pkg="debhello", ver="0.2", rev="1"
I: *** start packaging in "debhello-0.2". ***
I: provide debhello_0.2.orig.tar.gz for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-0.2.tar.gz debhello_0.2.orig.tar.gz
I: pwd = "/path/to/debhello-0.2"
I: parse binary package settings: :sh
I: binary package=debhello Type=script / Arch=all M-A=foreign
I: analyze the source tree
I: build_type = Unknown
I: scan source for copyright+license text and file extensions
```

```
I: 33 %, ext = desktop
...
```

Проверим созданные шаблонные файлы.

Дерево исходного кода после простого выполнения debmake. (v=0.2)

```
$ cd ..
$ tree
.
+-- debhello-0.2
|   +-- LICENSE
|   +-- data
|       | +-- hello.desktop
|       | +-- hello.png
|   +-- debian
|       | +-- README.Debian
|       | +-- changelog
|       | +-- control
|       | +-- copyright
|       | +-- patches
|       |     | +-- series
|       | +-- rules
|       | +-- source
|       |     | +-- control
|       |     | +-- format
|       |     | +-- local-options
|       |     | +-- options
|       |     | +-- patch-header
|       | +-- tests
|       |     | +-- control
|       | +-- upstream
|       |     | +-- metadata
|       | +-- watch
|   +-- man
|       | +-- hello.1
|   +-- scripts
|       | +-- hello
+-- debhello-0.2.tar.gz
+-- debhello_0.2.orig.tar.gz -> debhello-0.2.tar.gz

9 directories, 21 files
```

debian/rules (шаблонный файл, v=0.2):

```
$ cat debhello-0.2/debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1

%:
    dh $@
```

По сути, это стандартный файл **debian/rules**, использующий команду **dh**. Поскольку это пакет со сценарием, этот шаблонный файл **debian/rules** не имеет содержимого, связанного с флагом сборки.

debian/control (шаблонный файл, v=0.2):

```
$ cat debhello-0.2/debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Firstname Lastname" <email.address@example.org>
Build-Depends: debhelper-compat (= 13)
Standards-Version: 4.5.1
Homepage: <insert the upstream URL, if relevant>
Rules-Requires-Root: no
```



```
Package: debhello
Architecture: all
Multi-Arch: foreign
Depends: ${misc:Depends}
Description: auto-generated package by debmake
 This Debian binary package was auto-generated by the
 debmake(1) command provided by the debmake package.
```

Поскольку это пакет со сценарием командной оболочки, команда **debmake** устанавливает «**Architecture: all**» и «**Multi-Arch: foreign**». Кроме того, она устанавливает требуемые параметры **переменных подстановки**, такие как «**Depends: \${misc:Depends}**». Всё это объясняется в Chapter 5.

Since this upstream source lacks the upstream **Makefile**, that functionality needs to be provided by the maintainer. This upstream source contains only a script file and data files and no C source files; the **build** process can be skipped but the **install** process needs to be implemented. For this case, this is achieved cleanly by adding the **debian/install** and **debian/manpages** files without complicating the **debian/rules** file.

Сделаем этот пакет Debian лучше.

debian/rules (версия сопровождающего, v=0.2):

```
$ vim debhello-0.2/debian/rules
... hack, hack, hack, ...
$ cat debhello-0.2/debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1

%:
    dh $@
```

debian/control (версия сопровождающего, v=0.2):

```
$ vim debhello-0.2/debian/control
... hack, hack, hack, ...
$ cat debhello-0.2/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13)
Standards-Version: 4.5.1
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: all
Multi-Arch: foreign
Depends: ${misc:Depends}
Description: Simple packaging example for debmake
 This Debian binary package is an example package.
 (This is an example only)
```

Warning



If you leave “**Section: unknown**” in the template **debian/control** file unchanged, the **lintian** error may cause a build failure.

debian/install (версия сопровождающего, v=0.2):

```
$ vim debhello-0.2/debian/install
... hack, hack, hack, ...
$ cat debhello-0.2/debian/install
```

```
data/hello.desktop usr/share/applications
data/hello.png usr/share/pixmaps
scripts/hello usr/bin
```

debian/manpages (версия сопровождающего, v=0.2):

```
$ vim debhello-0.2/debian/manpages
... hack, hack, hack, ...
$ cat debhello-0.2/debian/manpages
man/hello.1
```

В каталоге **debian/** имеются и другие шаблонные файлы. Их также следует обновить.

Шаблонные файлы в каталоге debian/. (v=0.2):

```
$ tree debhello-0.2/debian
debhello-0.2/debian
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- install
+-- manpages
+-- patches
|  +-- series
+-- rules
+-- source
|  +-- control
|  +-- format
|  +-- local-options
|  +-- options
|  +-- patch-header
+-- tests
|  +-- control
+-- upstream
|  +-- metadata
+-- watch
```

4 directories, 16 files

В данном дереве исходного кода вы можете создать неродной пакет Debian с помощью команды **debuild** (или её эквивалента). Вывод этой команды очень подробен, в нём объясняется, что происходит, и выглядит это следующим образом.

```
$ cd debhello-0.2
$ debuild
dpkg-buildpackage -us -uc -ui -i -i
...
debian/rules clean
dh clean
...
debian/rules binary
dh binary
dh_update_autotools_config
dh_autoreconf
create-stamp debian/debhelper-build-stamp
dh_prep
  rm -f -- debian/debhello.substvars
  rm -fr -- debian/.debhelper/generated/debhello/ debian/debhello/ debi...
dh_install
  install -d debian/debhello/usr/bin
  cp --reflink=auto -a ./scripts/hello debian/debhello/usr/bin/
...
```

Проверим результат сборки.

Командой debuild были созданы следующие файлы debhello версии 0.2:

```

$ cd ..
$ tree -FL 1
.
+-- debhello-0.2/
+-- debhello-0.2.tar.gz
+-- debhello_0.2-1.debian.tar.xz
+-- debhello_0.2-1.dsc
+-- debhello_0.2-1_all.deb
+-- debhello_0.2-1_amd64.build
+-- debhello_0.2-1_amd64.buildinfo
+-- debhello_0.2-1_amd64.changes
+-- debhello_0.2.orig.tar.gz -> debhello-0.2.tar.gz

1 directory, 8 files

```

Вы видите все созданные файлы.

- The **debhello_0.2.orig.tar.gz** file is a symlink to the upstream tarball.
- The **debhello_0.2-1.debian.tar.xz** file contains the maintainer generated contents.
- The **debhello_0.2-1.dsc** file is the meta data file for the Debian source package.
- The **debhello_0.2-1_all.deb** file is the Debian binary package.
- The **debhello_0.2-1_amd64.build** file is the build log file.
- The **debhello_0.2-1_amd64.buildinfo** file is the meta data file generated by **dpkg-genbuildinfo(1)**.
- The **debhello_0.2-1_amd64.changes** file is the meta data file for the Debian binary package.

The **debhello_0.2-1.debian.tar.xz** file contains the Debian changes to the upstream source as follows.

Сжатое содержимое архива debhello_0.2-1.debian.tar.xz:

```

$ tar -tzf debhello-0.2.tar.gz
debhello-0.2/
debhello-0.2/data/
debhello-0.2/data/hello.desktop
debhello-0.2/data/hello.png
debhello-0.2/man/
debhello-0.2/man/hello.1
debhello-0.2/scripts/
debhello-0.2/scripts/hello
debhello-0.2/LICENSE
$ tar --xz -tf debhello_0.2-1.debian.tar.xz
debian/
debian/README.Debian
debian/changelog
debian/control
debian/copyright
debian/install
debian/manpages
debian/patches/
debian/patches/series
debian/rules
debian/source/
debian/source/control
debian/source/format
debian/source/options
debian/source/patch-header
debian/tests/
debian/tests/control
debian/upstream/
debian/upstream/metadata
debian/watch

```

The **debhello_0.2-1_amd64.deb** file contains the files to be installed as follows.

The binary package contents of debhello_0.2-1_all.deb:

```
$ dpkg -c debhello_0.2-1_all.deb
drwxr-xr-x root/root ... ./
drwxr-xr-x root/root ... ./usr/
drwxr-xr-x root/root ... ./usr/bin/
-rwxr-xr-x root/root ... ./usr/bin/hello
drwxr-xr-x root/root ... ./usr/share/
drwxr-xr-x root/root ... ./usr/share/applications/
-rw-r--r-- root/root ... ./usr/share/applications/hello.desktop
drwxr-xr-x root/root ... ./usr/share/doc/
drwxr-xr-x root/root ... ./usr/share/doc/debhello/
-rw-r--r-- root/root ... ./usr/share/doc/debhello/README.Debian
-rw-r--r-- root/root ... ./usr/share/doc/debhello/changelog.Debian.gz
-rw-r--r-- root/root ... ./usr/share/doc/debhello/copyright
drwxr-xr-x root/root ... ./usr/share/man/
drwxr-xr-x root/root ... ./usr/share/man/man1/
-rw-r--r-- root/root ... ./usr/share/man/man1/hello.1.gz
drwxr-xr-x root/root ... ./usr/share/pixmaps/
-rw-r--r-- root/root ... ./usr/share/pixmaps/hello.png
```

Here is the generated dependency list of **debhello_0.2-1_all.deb**.

The generated dependency list of debhello_0.2-1_all.deb:

```
$ dpkg -f debhello_0.2-1_all.deb pre-depends depends recommends conflicts br...
```

(No extra dependency packages required since this is a POSIX shell program.)

Note



If you wish to replace upstream provided PNG file **data/hello.png** with maintainer provided one **debian/hello.png**, editing **debian/install** isn't enough. When you add **debian/hello.png**, you need to add a line "include-binaries" to **debian/source/options** since PNG is a binary file. See `dpkg-source(1)`.

8.3 Makefile (командная оболочка, интерфейс командной оболочки)

Ниже приводится пример создания простого пакета Debian из программы с интерфейсом командной оболочки, написанной для командной оболочки POSIX и использующей в качестве системы сборки **Makefile**.

Допустим tar-архив основной ветки имеет имя **debhello-1.0.tar.gz**.

Предполагается, что этот тип исходного кода будет установлен как несистемный файл:

```
$ tar -xzf debhello-1.0.tar.gz
$ cd debhello-1.0
$ make install
```

При создании пакетов Debian требуется изменить процесс установки файлов с помощью «**make install**» так, чтобы установка производилось в системный каталог, а не в обычный каталог в **/usr/local**.

Получим исходный код и создадим пакет Debian.

Загрузим debhello-1.0.tar.gz

```
$ wget http://www.example.org/download/debhello-1.0.tar.gz
...
$ tar -xzf debhello-1.0.tar.gz
$ tree
.
+-- debhello-1.0
|   +-- LICENSE
|   +-- Makefile
|   +-- data
```

```
| | +-- hello.desktop
| | +-- hello.png
| +-- man
| | +-- hello.1
| +-- scripts
| +-- hello
+-- debhello-1.0.tar.gz
```

4 directories, 7 files

Here, the **Makefile** uses **\$(DESTDIR)** and **\$(prefix)** properly. All other files are the same as in Section 8.2 and most of the packaging activities are the same.

Makefile (v=1.0)

```
$ cat debhello-1.0/Makefile
prefix = /usr/local

all:
    : # do nothing

install:
    install -D scripts/hello \
        $(DESTDIR)$(prefix)/bin/hello
    install -m 644 -D data/hello.desktop \
        $(DESTDIR)$(prefix)/share/applications/hello.desktop
    install -m 644 -D data/hello.png \
        $(DESTDIR)$(prefix)/share/pixmaps/hello.png
    install -m 644 -D man/hello.1 \
        $(DESTDIR)$(prefix)/share/man/man1/hello.1

clean:
    : # do nothing

distclean: clean

uninstall:
    -rm -f $(DESTDIR)$(prefix)/bin/hello
    -rm -f $(DESTDIR)$(prefix)/share/applications/hello.desktop
    -rm -f $(DESTDIR)$(prefix)/share/pixmaps/hello.png
    -rm -f $(DESTDIR)$(prefix)/share/man/man1/hello.1

.PHONY: all install clean distclean uninstall
```

Let's package this with the **debmake** command. Here, the **-b':sh'** option is used to specify that the generated binary package is a shell script.

```
$ cd debhello-1.0
$ debmake -b':sh' -e osamu@debian.org -f 'Osamu Aoki'
I: set parameters
...
I: sanity check of parameters
I: pkg="debhello", ver="1.0", rev="1"
I: *** start packaging in "debhello-1.0". ***
I: provide debhello_1.0.orig.tar.gz for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.0.tar.gz debhello_1.0.orig.tar.gz
I: pwd = "/path/to/debhello-1.0"
I: parse binary package settings: :sh
I: binary package=debhello Type=script / Arch=all M-A=foreign
I: analyze the source tree
I: build_type = make
I: scan source for copyright+license text and file extensions
I: 33 %, ext = desktop
...
```

Let's inspect the notable template files generated.

debian/rules (шаблонный файл, v=1.0):

```
$ cat debhello-1.0/debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1

%:
    dh $@

#override_dh_auto_install:
#    dh_auto_install -- prefix=/usr

#override_dh_install:
#    dh_install --list-missing -X.pyc -X.pyo
```

Сделаем этот пакет Debian лучше.

debian/rules (версия сопровождающего, v=1.0):

```
$ vim debhello-1.0/debian/rules
... hack, hack, hack, ...
$ cat debhello-1.0/debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1

%:
    dh $@

override_dh_auto_install:
    dh_auto_install -- prefix=/usr
```

Since this upstream source has the proper upstream **Makefile**, there is no need to create **debian/install** and **debian/manpages** files.

Файл **debian/control** в точности совпадает с тем же файлом из случая Section 8.2.

В каталоге **debian/** имеются и другие шаблонные файлы. Их также следует обновить.

Шаблонные файлы в каталоге debian/. (v=1.0):

```
$ tree debhello-1.0/debian
debhello-1.0/debian
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- patches
|   +-- series
+-- rules
+-- source
|   +-- control
|   +-- format
|   +-- local-options
|   +-- options
|   +-- patch-header
+-- tests
|   +-- control
+-- upstream
|   +-- metadata
+-- watch

4 directories, 14 files
```

The rest of the packaging activities are practically the same as the ones in Section 8.2.

8.4 setup.py (Python3, интерфейс командной оболочки)

Many interesting upstream Python program packages are packaged using the **setuptools** based **setup.py**. Usually, these source packages are built into Python specific binary distribution format **wheel** and **wheel** packages are distributed to end users. Then, end users find these from package listing sites such as [the Python Package Index \(PyPI\)](#) and install these **wheel** packages using **pip** to the user's environment.

In order to make Debian package of these Python programs, you must start with their source packages. The current Debian packaging practice of such Python programs uses `dh_python3(1)` and `pybuild(1)` from `dh_python` package. Although **setuptools** isn't mentioned in `pybuild(1)`, you can treat these as if these are packaged using the **distutils** based **setup.py** since **setuptools** is an enhanced extension of **distutils**. [1](#)

Let's assume the upstream tarball to be **debhello-1.1.tar.gz**.

"[Installing Python Modules \(Legacy version\)](#)" explains how to install source to the system using **distutils**. Essentially, the source is installed using **distutils** as:

```
$ tar -xzf debhello-1.1.tar.gz
$ cd debhello-1.1
$ python3 setup.py install
```

Debian packaging requires changing the last line to "**python3 setup.py install --install-layout=deb**" to install files into the target system image location. This is automatically addressed when using the **dh** command for Debian packaging.

Here is an example of creating a simple Debian package from a Python3 CLI program using the **setuptools** based **setup.py** as its build system.

Получим исходный код и создадим пакет Debian.

Загрузим debhello-1.1.tar.gz

```
$ wget http://www.example.org/download/debhello-1.1.tar.gz
...
$ tar -xzf debhello-1.1.tar.gz
$ tree
.
+-- debhello-1.1
|   +-- LICENSE
|   +-- MANIFEST.in
|   +-- README.md
|   +-- data
|       | +-- hello.desktop
|       | +-- hello.png
|   +-- manpages
|       | +-- hello.1
|   +-- pyproject.toml
|   +-- setup.cfg
|   +-- setup.py
|   +-- src
|       +-- debhello
|           +-- __init__.py
|           +-- __main__.py
+-- debhello-1.1.tar.gz

5 directories, 12 files
```

Here, the content of this **debhello** source tree as follows.

setup.py (v=1.1) — a boiler plate setup.py for the backward compatibility

```
$ cat debhello-1.1/setup.py
# coding: utf-8
import setuptools
setuptools.setup()
```

setup.cfg (v=1.1) — modern static metadata

If the source package is lacks **setup.py** but still based on **setuptools**, you need to add a boiler plate **setup.py**. If the source package is based on other build systems such as **flit**, you need to use respective plugins.

```

$ cat debhello-1.1/setup.cfg
[metadata]
name = debhello
version = attr: debhello.__version__
description = Hello Python (CLI)
long_description = file: README.md
classifiers =
    Development Status :: 5 - Production/Stable
    Intended Audience :: Developers
    Topic :: System :: Archiving :: Packaging
    License :: OSI Approved :: MIT License
    Programming Language :: Python
    Programming Language :: Python :: 3
    Programming Language :: Python :: 3.5
    Programming Language :: Python :: 3.6
    Programming Language :: Python :: 3.7
    Programming Language :: Python :: 3.8
    Programming Language :: Python :: 3.9
author = Osamu Aoki
author_email = osamu@debian.org
maintainer = Osamu Aoki
maintainer_email = osamu@debian.org
url = https://salsa.debian.org/debian/debmake-doc
project_urls =
    Documentation = https://salsa.debian.org/debian/debmake-doc
keywords = debhello
license = MIT

[options]
package_dir =
    = src
packages = find:
python_requires = >=3.4
setup_requires = setuptools >= 42

[options.packages.find]
where = src

[options.entry_points]
console_scripts =
    hello = debhello.__main__:main

```

pyproject.toml (v=1.1) — PEP 517 configuration

```

$ cat debhello-1.1/pyproject.toml
[build-system]
requires = [
    "setuptools>=42",
    "wheel"
]
build-backend = "setuptools.build_meta"

```

MANIFEST.in (v=1.1) — for distutils to make sdist

```

$ cat debhello-1.1/MANIFEST.in
include data/*
include manpages/*

```

src/debhello/__init__.py (v=1.1) — program version definition

```

$ cat debhello-1.1/src/debhello/__init__.py
__version__ = '1.1.0'

```

src/debhello/__main__.py (v=1.1) — command entry point


```
$ cat debhello-1.1/src/debhello/__main__.py
"""
debhello command line entry (enable python -m debhello syntax)
"""

import sys
import debhello

def main(): # needed for console script
    print(' ===== Hello Python3 =====')
    print('argv = {}'.format(sys.argv))
    print('version = {}'.format(debhello.__version__))
    return

if __name__ == "__main__":
    sys.exit(main())
```

Let's package this with the **debmake** command. Here, the **-b':py3'** option is used to specify the generated binary package containing Python3 script and module files.

```
$ cd debhello-1.1
$ debmake -b':py3' -e osamu@debian.org -f 'Osamu Aoki'
I: set parameters
...
I: sanity check of parameters
I: pkg="debhello", ver="1.1", rev="1"
I: *** start packaging in "debhello-1.1". ***
I: provide debhello_1.1.orig.tar.gz for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.1.tar.gz debhello_1.1.orig.tar.gz
I: pwd = "/path/to/debhello-1.1"
I: parse binary package settings: :py3
I: binary package=debhello Type=python3 / Arch=all M-A=foreign
I: analyze the source tree
W: unknown python version. check setup.py.
I: build_type =
I: scan source for copyright+license text and file extensions
...
```

Let's inspect the notable template files generated.

debian/rules (шаблонный файл, v=1.1):

```
$ cat debhello-1.1/debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1

%:
    dh $@ --with python3
```

По сути, это стандартный файл **debian/rules**, использующий команду **dh**.

The use of the **"--with python3"** option invokes **dh_python3** to calculate Python dependencies, add maintainer scripts to byte compiled files, etc. See **dh_python3(1)**.

The use of the **"--buildsystem=pybuild"** option invokes various build systems for requested Python versions in order to build modules and extensions. See **pybuild(1)**.

debian/control (шаблонный файл, v=1.1):

```
$ cat debhello-1.1/debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13), dh-python, python3-all
```

```
Standards-Version: 4.5.1
Homepage: <insert the upstream URL, if relevant>
Rules-Requires-Root: no

Package: debhello
Architecture: all
Multi-Arch: foreign
Depends: ${misc:Depends}, ${python3:Depends}
Description: auto-generated package by debmake
 This Debian binary package was auto-generated by the
 debmake(1) command provided by the debmake package.
```

Поскольку это пакет на языке Python3, команда **debmake** устанавливает «**Architecture: all**» и «**Multi-Arch: foreign**». Кроме того, она устанавливает требуемые параметры **подстановочных переменных** в виде «**Depends: \${python3:Depends}, \${misc:Depends}**». Всё это объясняется в Chapter 5.

Сделаем этот пакет Debian лучше.

debian/rules (версия сопровождающего, v=1.1):

```
$ vim debhello-1.1/debian/rules
... hack, hack, hack, ...
$ cat debhello-1.1/debian/rules
#!/usr/bin/make -f
export PYBUILD_NAME=debhello
export PYBUILD_VERBOSE=1
export DH_VERBOSE=1

%:
    dh @$ --with python3 --buildsystem=pybuild
```

debian/control (версия сопровождающего, v=1.1):

```
$ vim debhello-1.1/debian/control
... hack, hack, hack, ...
$ cat debhello-1.1/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13),
               dh-python,
               python3-all,
               python3-setuptools
Standards-Version: 4.5.1
Rules-Requires-Root: no
Vcs-Browser: https://salsa.debian.org/debian/debmake-doc
Vcs-Git: https://salsa.debian.org/debian/debmake-doc.git
Homepage: https://salsa.debian.org/debian/debmake-doc

Package: debhello
Architecture: all
Depends: ${misc:Depends}, ${python3:Depends}
Description: Simple packaging example for debmake
 This is an example package to demonstrate Debian packaging using
 the debmake command.
.
The generated Debian package uses the dh command offered by the
 debhelper package and the dpkg source format `3.0 (quilt)'.
```

В каталоге **debian/** имеются и другие шаблонные файлы. Их также следует обновить.

This **debhello** command comes with the upstream-provided manpage and desktop file but the upstream **setup.cfg** doesn't install them. So you need to update **debian/install** and **debian/manpages** as follows:

debian/install (maintainer version, v=1.1):

```
$ vim debhello-1.1/debian/install
... hack, hack, hack, ...
```

```
$ cat debhello-1.1/debian/install
data/hello.desktop usr/share/applications
data/hello.png usr/share/pixmaps
```

debian/manpages (maintainer version, v=1.1):

```
$ vim debhello-1.1/debian/manpages
... hack, hack, hack, ...
$ cat debhello-1.1/debian/manpages
manpages/hello.1
```

The rest of the packaging activities are practically the same as the ones in Section 8.3.

Шаблонные файл в каталоге debian/. (v=1.1):

```
$ tree debhello-1.1/debian
debhello-1.1/debian
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- install
+-- manpages
+-- patches
|   +-- series
+-- rules
+-- source
|   +-- control
|   +-- format
|   +-- local-options
|   +-- options
|   +-- patch-header
+-- tests
|   +-- control
+-- upstream
|   +-- metadata
+-- watch

4 directories, 16 files
```

Here is the generated dependency list of **debhello_1.1-1_all.deb**.

The generated dependency list of debhello_1.1-1_all.deb:

```
$ dpkg -f debhello_1.1-1_all.deb pre-depends depends recommends conflicts br...
Depends: python3:any
```

8.5 Makefile (командная оболочка, графический интерфейс пользователя)

Ниже приводится пример создания простого пакета Debian из программы с графическим интерфейсом пользователя, написанной для командной оболочки POSIX и использующей в качестве системы сборки **Makefile**.

This upstream is based on Section 8.3 with enhanced GUI support.

Допустим, tar-архив основной ветки имеет имя **debhello-1.2.tar.gz**.

Получим исходный код и создадим пакет Debian.

Загрузим debhello-1.2.tar.gz

```
$ wget http://www.example.org/download/debhello-1.2.tar.gz
...
$ tar -xzmf debhello-1.2.tar.gz
$ tree
.
+-- debhello-1.2
|   +-- LICENSE
|   +-- Makefile
```

```
| +-- data
| | +-- hello.desktop
| | +-- hello.png
| +-- man
| | +-- hello.1
| +-- scripts
|   +-- hello
+-- debhello-1.2.tar.gz
```

4 directories, 7 files

Итак, сценарий **hello** был переписан таким образом, чтобы для создания графического интерфейса пользователя на основе GTK+ использовалась команда **zenity**.

hello (v=1.2)

```
$ cat debhello-1.2/scripts/hello
#!/bin/sh -e
zenity --info --title "hello" --text "Hello from the shell!"
```

Файл `desktop` должен быть обновлён и должен содержать строку **Terminal=false**, поскольку эта программа имеет графический интерфейс.

hello.desktop (v=1.2)

```
$ cat debhello-1.2/data/hello.desktop
[Desktop Entry]
Name=Hello
Name[fr]=Bonjour
Comment=Greetings
Comment[fr]=Salutations
Type=Application
Keywords=hello
Exec=hello
Terminal=false
Icon=hello.png
Categories=Utility;
```

All other files are the same as in Section 8.3.

Let's package this with the **debmake** command. Here, the **-b':sh'** option is used to specify that the generated binary package is a shell script.

```
$ cd debhello-1.2
$ debmake -b':sh' -e osamu@debian.org -f 'Osamu Aoki'
I: set parameters
...
I: sanity check of parameters
I: pkg="debhello", ver="1.2", rev="1"
I: *** start packaging in "debhello-1.2". ***
I: provide debhello_1.2.orig.tar.gz for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.2.tar.gz debhello_1.2.orig.tar.gz
I: pwd = "/path/to/debhello-1.2"
I: parse binary package settings: :sh
I: binary package=debhello Type=script / Arch=all M-A=foreign
I: analyze the source tree
I: build_type = make
I: scan source for copyright+license text and file extensions
I: 33 %, ext = desktop
...
```

Let's inspect the notable template files generated.

debian/control (шаблонный файл, v=1.2):

```
$ cat debhello-1.2/debian/control
Source: debhello
Section: unknown
```

```

Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13)
Standards-Version: 4.5.1
Homepage: <insert the upstream URL, if relevant>
Rules-Requires-Root: no

Package: debhello
Architecture: all
Multi-Arch: foreign
Depends: ${misc:Depends}
Description: auto-generated package by debmake
 This Debian binary package was auto-generated by the
 debmake(1) command provided by the debmake package.

```

Сделаем этот пакет Debian лучше.

debian/control (версия сопровождающего, v=1.2):

```

$ vim debhello-1.2/debian/control
... hack, hack, hack, ...
$ cat debhello-1.2/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13)
Standards-Version: 4.5.1
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: all
Multi-Arch: foreign
Depends: zenity, ${misc:Depends}
Description: Simple packaging example for debmake
 This Debian binary package is an example package.
 (This is an example only)

```

Please note the manually added **zenity** dependency.

Файл **debian/rules** полностью совпадает с тем же файлом из Section 8.3.

В каталоге **debian/** имеются и другие шаблонные файлы. Их также следует обновить.

Шаблонные файлы в каталоге debian/. (v=1.2):

```

$ tree debhello-1.2/debian
debhello-1.2/debian
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- patches
|  +-- series
+-- rules
+-- source
|  +-- control
|  +-- format
|  +-- local-options
|  +-- options
|  +-- patch-header
+-- tests
|  +-- control
+-- upstream
|  +-- metadata
+-- watch

4 directories, 14 files

```

The rest of the packaging activities are practically the same as in Section 8.3.

Here is the generated dependency list of **debhello_1.2-1_all.deb**.

The generated dependency list of debhello_1.2-1_all.deb:

```
$ dpkg -f debhello_1.2-1_all.deb pre-depends depends recommends conflicts br...
Depends: zenity
```

8.6 setup.py (Python3, графический интерфейс пользователя)

Here is an example of creating a simple Debian package from a Python3 GUI program using the **setuptools** based **setup.py** as its build system.

This upstream is based on Section 8.4 with enhanced GUI, desktop icon, and manpage support.

Допустим tar-архив основной ветки имеет имя **debhello-1.3.tar.gz**.

Получим исходный код и создадим пакет Debian.

Загрузим debhello-1.3.tar.gz

```
$ wget http://www.example.org/download/debhello-1.3.tar.gz
...
$ tar -xzmf debhello-1.3.tar.gz
$ tree
.
+-- debhello-1.3
|   +-- LICENSE
|   +-- MANIFEST.in
|   +-- README.md
|   +-- data
|       | +-- hello.desktop
|       | +-- hello.png
|   +-- manpages
|       | +-- hello.1
|   +-- pyproject.toml
|   +-- setup.cfg
|   +-- setup.py
|   +-- src
|       +-- debhello
|           +-- __init__.py
|           +-- __main__.py
+-- debhello-1.3.tar.gz

5 directories, 12 files
```

Here, the content of this **debhello** source tree as follows.

setup.py (v=1.3) — a boiler plate setup.py for the backward compatibility

```
$ cat debhello-1.3/setup.py
import setuptools
setuptools.setup()
```

setup.cfg (v=1.3) — modern static metadata

```
$ cat debhello-1.3/setup.cfg
[metadata]
name = debhello
version = attr: debhello.__version__
description = Hello Python (GUI)
long_description = file: README.md
classifiers =
    Development Status :: 5 - Production/Stable
    Intended Audience :: Developers
    Topic :: System :: Archiving :: Packaging
    License :: OSI Approved :: MIT License
    Programming Language :: Python
    Programming Language :: Python :: 3
```

```

Programming Language :: Python :: 3.5
Programming Language :: Python :: 3.6
Programming Language :: Python :: 3.7
Programming Language :: Python :: 3.8
Programming Language :: Python :: 3.9
author = Osamu Aoki
author_email = osamu@debian.org
maintainer = Osamu Aoki
maintainer_email = osamu@debian.org
url = https://salsa.debian.org/debian/debmake-doc
project_urls =
    Documentation = https://salsa.debian.org/debian/debmake-doc
keywords = debhello
license = MIT

[options]
package_dir =
    = src
packages = find:
python_requires = >=3.4
setup_requires = setuptools >= 42

[options.packages.find]
where = src

[options.entry_points]
console_scripts =
    hello = debhello.__main__:main

```

pyproject.toml (v=1.3) — PEP 517 configuration

```

$ cat debhello-1.3/pyproject.toml
[build-system]
requires = [
    "setuptools>=42",
    "wheel"
]
build-backend = "setuptools.build_meta"

```

MANIFEST.in (v=1.3) — for distutils to make sdist

```

$ cat debhello-1.3/MANIFEST.in
include data/*
include manpages/*

```

src/debhello/__init__.py (v=1.3) — program version definition

```

$ cat debhello-1.3/src/debhello/__init__.py
__version__ = '1.3.0'

```

src/debhello/__main__.py (v=1.3) — command entry point

```

$ cat debhello-1.3/src/debhello/__main__.py
#!/usr/bin/python3
from gi.repository import Gtk

class TopWindow(Gtk.Window):

    def __init__(self):
        Gtk.Window.__init__(self)
        self.title = "Hello World!"
        self.counter = 0
        self.border_width = 10
        self.set_default_size(400, 100)
        self.set_position(Gtk.WindowPosition.CENTER)
        self.button = Gtk.Button(label="Click me!")

```

```

self.button.connect("clicked", self.on_button_clicked)
self.add(self.button)
self.connect("delete-event", self.on_window_destroy)

def on_window_destroy(self, *args):
    Gtk.main_quit(*args)

def on_button_clicked(self, widget):
    self.counter += 1
    widget.set_label("Hello, World!\nClick count = %i" % self.counter)

def main():
    window = TopWindow()
    window.show_all()
    Gtk.main()

if __name__ == '__main__':
    main()

```

Let's package this with the **debmake** command. Here, the **-b:py3** option is used to specify that the generated binary package contains Python3 script and module files.

```

$ cd debhello-1.3
$ debmake -b:py3 -e osamu@debian.org -f 'Osamu Aoki'
I: set parameters
...
I: sanity check of parameters
I: pkg="debhello", ver="1.3", rev="1"
I: *** start packaging in "debhello-1.3". ***
I: provide debhello_1.3.orig.tar.gz for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.3.tar.gz debhello_1.3.orig.tar.gz
I: pwd = "/path/to/debhello-1.3"
I: parse binary package settings: :py3
I: binary package=debhello Type=python3 / Arch=all M-A=foreign
I: analyze the source tree
W: unknown python version. check setup.py.
I: build_type =
I: scan source for copyright+license text and file extensions
...

```

The result is practically the same as in Section 8.4.

Сделаем этот пакет Debian лучше.

debian/rules (версия сопровождающего, v=1.3):

```

$ vim debhello-1.3/debian/rules
... hack, hack, hack, ...
$ cat debhello-1.3/debian/rules
#!/usr/bin/make -f
export PYBUILD_NAME=debhello
export PYBUILD_VERBOSE=1
export DH_VERBOSE=1

%:
    dh $@ --with python3 --buildsystem=pybuild

```

debian/control (версия сопровождающего, v=1.3):

```

$ vim debhello-1.3/debian/control
... hack, hack, hack, ...
$ cat debhello-1.3/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>

```



```
Build-Depends: debhelper-compat (= 13), dh-python, python3-all
Standards-Version: 4.5.1
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no
```

```
Package: debhello
Architecture: all
Multi-Arch: foreign
Depends: gir1.2-gtk-3.0, python3-gi, ${misc:Depends}, ${python3:Depends}
Description: Simple packaging example for debmake
 This Debian binary package is an example package.
 (This is an example only)
```

Please note the manually added **python3-gi** and **gir1.2-gtk-3.0** dependencies.

The rest of the packaging activities are practically the same as in Section 8.4.

Here is the generated dependency list of **debhello_1.3-1_all.deb**.

The generated dependency list of debhello_1.3-1_all.deb:

```
$ dpkg -f debhello_1.3-1_all.deb pre-depends depends recommends conflicts br...
Depends: gir1.2-gtk-3.0, python3-gi, python3:any
```

8.7 Makefile (single-binary package)

Here is an example of creating a simple Debian package from a simple C source program using the **Makefile** as its build system.

Это — пример улучшенного исходного кода основной ветки из Chapter 4. Он содержит страницу руководства, файл `desktop`, а также иконку рабочего стола. Кроме того, чтобы этот пример имел большую практическую ценность, исходный кодкомпонуется с внешней библиотекой **libm**.

Допустим `tar`-архив основной ветки имеет имя **debhello-1.4.tar.gz**.

Предполагается, что этот тип исходного кода будет установлен как несистемный файл:

```
$ tar -xzmf debhello-1.4.tar.gz
$ cd debhello-1.4
$ make
$ make install
```

Debian packaging requires changing this “**make install**” process to install files into the target system image location instead of the normal location under **/usr/local**.

Получим исходный код и создадим пакет Debian.

Загрузим debhello-1.4.tar.gz

```
$ wget http://www.example.org/download/debhello-1.4.tar.gz
...
$ tar -xzmf debhello-1.4.tar.gz
$ tree
.
+-- debhello-1.4
|   +-- LICENSE
|   +-- Makefile
|   +-- data
|       | +-- hello.desktop
|       | +-- hello.png
|   +-- man
|       | +-- hello.1
|   +-- src
|       +-- config.h
|       +-- hello.c
+-- debhello-1.4.tar.gz

4 directories, 8 files
```

Ниже приводится содержимое этого архива с исходным кодом.

src/hello.c (v=1.4):

```
$ cat debhello-1.4/src/hello.c
#include "config.h"
#include <math.h>
#include <stdio.h>
int
main()
{
    printf("Hello, I am " PACKAGE_AUTHOR "!\\n");
    printf("4.0 * atan(1.0) = %10f8\\n", 4.0*atan(1.0));
    return 0;
}
```

src/config.h (v=1.4):

```
$ cat debhello-1.4/src/config.h
#define PACKAGE_AUTHOR "Osamu Aoki"
```

Makefile (v=1.4):

```
$ cat debhello-1.4/Makefile
prefix = /usr/local

all: src/hello

src/hello: src/hello.c
    $(CC) $(CPPFLAGS) $(CFLAGS) $(LDFLAGS) -o $@ $^ -lm

install: src/hello
    install -D src/hello \
        $(DESTDIR)$(prefix)/bin/hello
    install -m 644 -D data/hello.desktop \
        $(DESTDIR)$(prefix)/share/applications/hello.desktop
    install -m 644 -D data/hello.png \
        $(DESTDIR)$(prefix)/share/pixmaps/hello.png
    install -m 644 -D man/hello.1 \
        $(DESTDIR)$(prefix)/share/man/man1/hello.1

clean:
    -rm -f src/hello

distclean: clean

uninstall:
    -rm -f $(DESTDIR)$(prefix)/bin/hello
    -rm -f $(DESTDIR)$(prefix)/share/applications/hello.desktop
    -rm -f $(DESTDIR)$(prefix)/share/pixmaps/hello.png
    -rm -f $(DESTDIR)$(prefix)/share/man/man1/hello.1

.PHONY: all install clean distclean uninstall
```

Заметьте, что этот файл **Makefile** имеет соответствующую цель **install** для страницы руководства, файла **desktop** и иконки рабочего стола.

Создадим пакет из этого исходного кода с помощью команды **debmake**.

```
$ cd debhello-1.4
$ debmake -e osamu@debian.org -f 'Osamu Aoki'
I: set parameters
...
I: sanity check of parameters
I: pkg="debhello", ver="1.4", rev="1"
I: *** start packaging in "debhello-1.4". ***
I: provide debhello_1.4.orig.tar.gz for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.4.tar.gz debhello_1.4.orig.tar.gz
I: pwd = "/path/to/debhello-1.4"
```

```
I: parse binary package settings:
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: analyze the source tree
I: build_type = make
I: scan source for copyright+license text and file extensions
I: 40 %, ext = c
...

```

The result is practically the same as in Section 4.6.

Let's make this Debian package, which is practically the same as in Section 4.7, better as the maintainer.

If the `DEB_BUILD_MAINT_OPTIONS` environment variable is not exported in `debian/rules`, lintian warns "W: debhello: hardening-no-relro usr/bin/hello" for the linking of `libm`.

The `debian/control` file makes it exactly the same as the one in Section 4.7, since the `libm` library is always available as a part of `libc6` (Priority: required).

В каталоге `debian/` имеются и другие шаблонные файлы. Их также следует обновить.

Шаблонные файлы в каталоге `debian/`. (v=1.4):

```
$ tree debhello-1.4/debian
debhello-1.4/debian
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- patches
|   +-- series
+-- rules
+-- source
|   +-- control
|   +-- format
|   +-- local-options
|   +-- options
|   +-- patch-header
+-- tests
|   +-- control
+-- upstream
|   +-- metadata
+-- watch

4 directories, 14 files

```

Остальные работы по подготовке пакета практически полностью совпадают с описанными в Section 4.8.

Here is the generated dependency list of all binary packages.

The generated dependency list of all binary packages (v=1.4):

```
$ dpkg -f debhello-dbgSYM_1.4-1_amd64.deb pre-depends depends recommends con...
Depends: debhello (= 1.4-1)
$ dpkg -f debhello_1.4-1_amd64.deb pre-depends depends recommends conflicts ...
Depends: libc6 (>= 2.3.4)

```

8.8 Makefile.in + configure (single-binary package)

Here is an example of creating a simple Debian package from a simple C source program using `Makefile.in` and `configure` as its build system.

This is an enhanced upstream source example for Section 8.7. This also links to an external library, `libm`, and this source is configurable using arguments to the `configure` script, which generates the `Makefile` and `src/config.h` files.

Допустим tar-архив основной ветки имеет имя `debhello-1.5.tar.gz`.

Этот тип исходного кода предполагает установку в виде несистемного файла, например, как

```
$ tar -xzmf debhello-1.5.tar.gz
$ cd debhello-1.5
$ ./configure --with-math

```

```
$ make
$ make install
```

Получим исходный код и создадим пакет Debian.

Загрузим debhello-1.5.tar.gz

```
$ wget http://www.example.org/download/debhello-1.5.tar.gz
...
$ tar -xzmf debhello-1.5.tar.gz
$ tree
.
+-- debhello-1.5
|   +-- LICENSE
|   +-- Makefile.in
|   +-- configure
|   +-- data
|       |   +-- hello.desktop
|       |   +-- hello.png
|   +-- man
|       |   +-- hello.1
|   +-- src
|       +-- hello.c
+-- debhello-1.5.tar.gz

4 directories, 8 files
```

Ниже приводится содержимое этого архива с исходным кодом.

src/hello.c (v=1.5):

```
$ cat debhello-1.5/src/hello.c
#include "config.h"
#ifdef WITH_MATH
# include <math.h>
#endif
#include <stdio.h>
int
main()
{
    printf("Hello, I am " PACKAGE_AUTHOR "!\n");
#ifdef WITH_MATH
    printf("4.0 * atan(1.0) = %10f8\n", 4.0*atan(1.0));
#else
    printf("I can't do MATH!\n");
#endif
    return 0;
}
```

Makefile.in (v=1.5):

```
$ cat debhello-1.5/Makefile.in
prefix = @prefix@

all: src/hello

src/hello: src/hello.c
    $(CC) @VERBOSE@ \
        $(CPPFLAGS) \
        $(CFLAGS) \
        $(LDFLAGS) \
        -o $@ $^ \
        @LINKLIB@

install: src/hello
    install -D src/hello \
        $(DESTDIR)$(prefix)/bin/hello
```

```

install -m 644 -D data/hello.desktop \
        $(DESTDIR)$(prefix)/share/applications/hello.desktop
install -m 644 -D data/hello.png \
        $(DESTDIR)$(prefix)/share/pixmaps/hello.png
install -m 644 -D man/hello.1 \
        $(DESTDIR)$(prefix)/share/man/man1/hello.1

clean:
    -rm -f src/hello

distclean: clean

uninstall:
    -rm -f $(DESTDIR)$(prefix)/bin/hello
    -rm -f $(DESTDIR)$(prefix)/share/applications/hello.desktop
    -rm -f $(DESTDIR)$(prefix)/share/pixmaps/hello.png
    -rm -f $(DESTDIR)$(prefix)/share/man/man1/hello.1

.PHONY: all install clean distclean uninstall

```

configure (v=1.5):

```

$ cat debhello-1.5/configure
#!/bin/sh -e
# default values
PREFIX="/usr/local"
VERBOSE=""
WITH_MATH="0"
LINKLIB=""
PACKAGE_AUTHOR="John Doe"

# parse arguments
while [ "${1}" != "" ]; do
    VAR="${1%=*}" # Drop suffix =*
    VAL="${1#*=}" # Drop prefix *=
    case "${VAR}" in
        --prefix)
            PREFIX="${VAL}"
            ;;
        --verbose|-v)
            VERBOSE="-v"
            ;;
        --with-math)
            WITH_MATH="1"
            LINKLIB="-lm"
            ;;
        --author)
            PACKAGE_AUTHOR="${VAL}"
            ;;
        *)
            echo "W: Unknown argument: ${1}"
            esac
            shift
    done

# setup configured Makefile and src/config.h
sed -e "s,@prefix@,${PREFIX}," \
    -e "s,@VERBOSE@,${VERBOSE}," \
    -e "s,@LINKLIB@,${LINKLIB}," \
    <Makefile.in >Makefile
if [ "${WITH_MATH}" = 1 ]; then
echo "#define WITH_MATH" >src/config.h
else
echo "/* not defined: WITH_MATH */" >src/config.h
fi

```

```
echo "#define PACKAGE_AUTHOR \"${PACKAGE_AUTHOR}\"" >>src/config.h
```

Please note that the **configure** command replaces strings with **@...@** in **Makefile.in** to produce **Makefile** and creates **src/config.h**.

Создадим пакет из этого исходного кода с помощью команды **debmake**.

```
$ cd debhello-1.5
$ debmake -e osamu@debian.org -f 'Osamu Aoki'
I: set parameters
...
I: sanity check of parameters
I: pkg="debhello", ver="1.5", rev="1"
I: *** start packaging in "debhello-1.5". ***
I: provide debhello_1.5.orig.tar.gz for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.5.tar.gz debhello_1.5.orig.tar.gz
I: pwd = "/path/to/debhello-1.5"
I: parse binary package settings:
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: analyze the source tree
I: build_type = configure
I: scan source for copyright+license text and file extensions
I: 20 %, ext = in
...
```

Полученный результат похож на то, что описано в Section 4.6, но полностью они не совпадают. Let's inspect the notable template files generated.

debian/rules (шаблонный файл, v=1.5):

```
$ cat debhello-1.5/debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
    dh $@
```

Сделаем этот пакет Debian лучше.

debian/rules (версия сопровождающего, v=1.5):

```
$ vim debhello-1.5/debian/rules
... hack, hack, hack, ...
$ cat debhello-1.5/debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
    dh $@

override_dh_auto_configure:
    dh_auto_configure -- \
        --with-math \
        --author="Osamu Aoki"
```

В каталоге **debian/** имеются и другие шаблонные файлы. Их также следует обновить.

Остальные работы по подготовке пакета практически полностью совпадают с описанными в Section 4.8.

8.9 Autotools (single-binary package)

Here is an example of creating a simple Debian package from a simple C source program using Autotools = Autoconf and Automake (**Makefile.am** and **configure.ac**) as its build system. See Section 5.18.1.

This source usually comes with the upstream auto-generated **Makefile.in** and **configure** files, too. This source can be packaged using these files as in Section 8.8 with the help of the **autotools-dev** package.

The better alternative is to regenerate these files using the latest Autoconf and Automake packages if the upstream provided **Makefile.am** and **configure.ac** are compatible with the latest version. This is advantageous for porting to new CPU architectures, etc. This can be automated by using the “**--with autoreconf**” option for the **dh** command.

Допустим tar-архив основной ветки имеет имя **debhello-1.6.tar.gz**.

Этот тип исходного кода предполагает установку в виде несистемного файла, например, как

```
$ tar -xzmf debhello-1.6.tar.gz
$ cd debhello-1.6
$ autoreconf -ivf # optional
$ ./configure --with-math
$ make
$ make install
```

Получим исходный код и создадим пакет Debian.

Загрузим debhello-1.6.tar.gz

```
$ wget http://www.example.org/download/debhello-1.6.tar.gz
...
$ tar -xzmf debhello-1.6.tar.gz
$ tree
.
+-- debhello-1.6
|   +-- LICENSE
|   +-- Makefile.am
|   +-- configure.ac
|   +-- data
|       | +-- hello.desktop
|       | +-- hello.png
|   +-- man
|       | +-- Makefile.am
|       | +-- hello.1
|   +-- src
|       +-- Makefile.am
|       +-- hello.c
+-- debhello-1.6.tar.gz

4 directories, 10 files
```

Ниже приводится содержимое этого архива с исходным кодом.

src/hello.c (v=1.6):

```
$ cat debhello-1.6/src/hello.c
#include "config.h"
#ifdef WITH_MATH
# include <math.h>
#endif
#include <stdio.h>
int
main()
{
    printf("Hello, I am " PACKAGE_AUTHOR "!\n");
#ifdef WITH_MATH
    printf("4.0 * atan(1.0) = %10f8\n", 4.0*atan(1.0));
#else
    printf("I can't do MATH!\n");
#endif
    return 0;
}
```

Makefile.am (v=1.6):

```
$ cat debhello-1.6/Makefile.am
SUBDIRS = src man
$ cat debhello-1.6/man/Makefile.am
dist_man_MANS = hello.1
$ cat debhello-1.6/src/Makefile.am
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```

configure.ac (v=1.6):

```
$ cat debhello-1.6/configure.ac
#                                     -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.
AC_PREREQ([2.69])
AC_INIT([debhello],[2.1],[foo@example.org])
AC_CONFIG_SRCDIR([src/hello.c])
AC_CONFIG_HEADERS([config.h])
echo "Standard customization chores"
AC_CONFIG_AUX_DIR([build-aux])
AM_INIT_AUTOMAKE([foreign])
# Add #define PACKAGE_AUTHOR ... in config.h with a comment
AC_DEFINE(PACKAGE_AUTHOR, ["Osamu Aoki"], [Define PACKAGE_AUTHOR])
echo "Add --with-math option functionality to ./configure"
AC_ARG_WITH([math],
  [AS_HELP_STRING([--with-math],
    [compile with math library @<:@default=yes@:>@]),
  [],
  [with_math="yes"]
)
echo "==== withval := \"\$withval\""
echo "==== with_math := \"\$with_math\""
# m4sh if-else construct
AS_IF([test "x$with_math" != "xno"],[
  echo "==== Check include: math.h"
  AC_CHECK_HEADER(math.h,[],[
    AC_MSG_ERROR([Couldn't find math.h.] )
  ])
  echo "==== Check library: libm"
  AC_SEARCH_LIBS(atan, [m])
  #AC_CHECK_LIB(m, atan)
  echo "==== Build with LIBS := \"\$LIBS\""
  AC_DEFINE(WITH_MATH, [1], [Build with the math library])
],[
  echo "==== Skip building with math.h."
  AH_TEMPLATE(WITH_MATH, [Build without the math library])
])
# Checks for programs.
AC_PROG_CC
AC_CONFIG_FILES([Makefile
                 man/Makefile
                 src/Makefile])
AC_OUTPUT
```

Tip

Without “**foreign**” strictness level specified in **AM_INIT_AUTOMAKE()** as above, **automake** defaults to “**gnu**” strictness level requiring several files in the top-level directory. See “3.2 Strictness” in the **automake** document.

Создадим пакет из этого исходного кода с помощью команды **debmake**.

```
$ cd debhello-1.6
$ debmake -e osamu@debian.org -f 'Osamu Aoki'
I: set parameters
...
I: sanity check of parameters
I: pkg="debhello", ver="1.6", rev="1"
I: *** start packaging in "debhello-1.6". ***
I: provide debhello_1.6.orig.tar.gz for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.6.tar.gz debhello_1.6.orig.tar.gz
I: pwd = "/path/to/debhello-1.6"
I: parse binary package settings:
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: analyze the source tree
I: build_type = Autotools with autoreconf
I: scan source for copyright+license text and file extensions
I: 38 %, ext = am
...
```

Получившийся результат похож на то, что было описано в Section 8.8, но не совпадает с ним в точности. Let's inspect the notable template files generated.

debian/rules (шаблонный файл, v=1.6):

```
$ cat debhello-1.6/debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
    dh $@ --with autoreconf

#override_dh_install:
#    dh_install --list-missing -X.la -X.pyc -X.pyo
```

Сделаем этот пакет Debian лучше.

debian/rules (версия сопровождающего, v=1.6):

```
$ vim debhello-1.6/debian/rules
... hack, hack, hack, ...
$ cat debhello-1.6/debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
    dh $@ --with autoreconf

override_dh_auto_configure:
    dh_auto_configure -- \
        --with-math
```

В каталоге **debian/** имеются и другие шаблонные файлы. Их также следует обновить.

Остальные работы по подготовке пакета практически полностью совпадают с описанными в Section 4.8.

8.10 CMake (single-binary package)

Here is an example of creating a simple Debian package from a simple C source program using CMake (**CMakeLists.txt** and some files such as **config.h.in**) as its build system. See Section 5.18.2.

The **cmake** command generates the **Makefile** file based on the **CMakeLists.txt** file and its **-D** option. It also configures the file as specified in its **configure_file(...)** by replacing strings with **@...@** and changing the **#cmakedefine ...** line.

Допустим tar-архив основной ветки имеет имя **debhello-1.7.tar.gz**.

Этот тип исходного кода предполагает установку в виде несистемного файла, например, как

```
$ tar -xzf debhello-1.7.tar.gz
$ cd debhello-1.7
$ mkdir obj-x86_64-linux-gnu # for out-of-tree build
$ cd obj-x86_64-linux-gnu
$ cmake ..
$ make
$ make install
```

Получим исходный код и создадим пакет Debian.

Загрузим debhello-1.7.tar.gz

```
$ wget http://www.example.org/download/debhello-1.7.tar.gz
...
$ tar -xzf debhello-1.7.tar.gz
$ tree
.
+-- debhello-1.7
|   +-- CMakeLists.txt
|   +-- LICENSE
|   +-- data
|       | +-- hello.desktop
|       | +-- hello.png
|   +-- man
|       | +-- CMakeLists.txt
|       | +-- hello.1
|   +-- src
|       +-- CMakeLists.txt
|       +-- config.h.in
|       +-- hello.c
+-- debhello-1.7.tar.gz

4 directories, 10 files
```

Ниже приводится содержимое этого архива с исходным кодом.

src/hello.c (v=1.7):

```
$ cat debhello-1.7/src/hello.c
#include "config.h"
#ifdef WITH_MATH
# include <math.h>
#endif
#include <stdio.h>
int
main()
{
    printf("Hello, I am " PACKAGE_AUTHOR "!\n");
#ifdef WITH_MATH
    printf("4.0 * atan(1.0) = %10f8\n", 4.0*atan(1.0));
#else
    printf("I can't do MATH!\n");
#endif
    return 0;
}
```

src/config.h.in (v=1.7):

```
$ cat debhello-1.7/src/config.h.in
/* name of the package author */
#define PACKAGE_AUTHOR "@PACKAGE_AUTHOR@"
```

```
/* math library support */
#cmakedefine WITH_MATH
```

CMakeLists.txt (v=1.7):

```
$ cat debhello-1.7/CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(debhello)
set(PACKAGE_AUTHOR "Osamu Aoki")
add_subdirectory(src)
add_subdirectory(man)
$ cat debhello-1.7/man/CMakeLists.txt
install(
  FILES ${CMAKE_CURRENT_SOURCE_DIR}/hello.1
  DESTINATION share/man/man1
)
$ cat debhello-1.7/src/CMakeLists.txt
# Always define HAVE_CONFIG_H
add_definitions(-DHAVE_CONFIG_H)
# Interactively define WITH_MATH
option(WITH_MATH "Build with math support" OFF)
#variable_watch(WITH_MATH)
# Generate config.h from config.h.in
configure_file(
  "${CMAKE_CURRENT_SOURCE_DIR}/config.h.in"
  "${CMAKE_CURRENT_BINARY_DIR}/config.h"
)
include_directories("${CMAKE_CURRENT_BINARY_DIR}")
add_executable(hello hello.c)
install(TARGETS hello
  RUNTIME DESTINATION bin
)
```

Создадим пакет из этого исходного кода с помощью команды **debmake**.

```
$ cd debhello-1.7
$ debmake -e osamu@debian.org -f 'Osamu Aoki'
I: set parameters
...
I: sanity check of parameters
I: pkg="debhello", ver="1.7", rev="1"
I: *** start packaging in "debhello-1.7". ***
I: provide debhello_1.7.orig.tar.gz for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.7.tar.gz debhello_1.7.orig.tar.gz
I: pwd = "/path/to/debhello-1.7"
I: parse binary package settings:
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: analyze the source tree
I: build_type = Cmake
I: scan source for copyright+license text and file extensions
I: 38 %, ext = text
...
```

Получившийся результат похож на то, что было описано в Section 8.8, но не совпадает с ним в точности. Let's inspect the notable template files generated.

debian/rules (шаблонный файл, v=1.7):

```
$ cat debhello-1.7/debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed
```

```

%:
    dh $@

#override_dh_auto_configure:
#    dh_auto_configure -- \
#        -DCMAKE_LIBRARY_ARCHITECTURE="$(DEB_TARGET_MULTIARCH)"

```

debian/control (шаблонный файл, v=1.7):

```

$ cat debhello-1.7/debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: cmake, debhelper-compat (= 13)
Standards-Version: 4.5.1
Homepage: <insert the upstream URL, if relevant>
Rules-Requires-Root: no

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends: ${misc:Depends}, ${shlibs:Depends}
Description: auto-generated package by debmake
 This Debian binary package was auto-generated by the
 debmake(1) command provided by the debmake package.

```

Сделаем этот пакет Debian лучше.

debian/rules (версия сопровождающего, v=1.7):

```

$ vim debhello-1.7/debian/rules
... hack, hack, hack, ...
$ cat debhello-1.7/debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

```

```

%:
    dh $@

override_dh_auto_configure:
    dh_auto_configure -- -DWITH-MATH=1

```

debian/control (версия сопровождающего, v=1.7):

```

$ vim debhello-1.7/debian/control
... hack, hack, hack, ...
$ cat debhello-1.7/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: cmake, debhelper-compat (= 13)
Standards-Version: 4.5.1
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends: ${misc:Depends}, ${shlibs:Depends}
Description: Simple packaging example for debmake
 This Debian binary package is an example package.

```

(This is an example only)

В каталоге **debian/** имеются и другие шаблонные файлы. Их также следует обновить. Остальные работы по подготовке пакета практически полностью совпадают с описанными в Section 8.8.

8.11 Autotools (multi-binary package)

Here is an example of creating a set of Debian binary packages including the executable package, the shared library package, the development file package, and the debug symbol package from a simple C source program using Autotools = Autoconf and Automake (which use **Makefile.am** and **configure.ac** as their input files) as its build system. See Section 5.18.1.

Let's package this in the same way as in Section 8.9.

Допустим tar-архив основной ветки имеет имя **debhello-2.0.tar.gz**.

Этот тип исходного кода предполагает установку в виде несистемного файла, например, как

```
$ tar -xzmf debhello-2.0.tar.gz
$ cd debhello-2.0
$ autoreconf -ivf # optional
$ ./configure --with-math
$ make
$ make install
```

Получим исходный код и создадим пакет Debian.

Загрузим debhello-2.0.tar.gz

```
$ wget http://www.example.org/download/debhello-2.0.tar.gz
...
$ tar -xzmf debhello-2.0.tar.gz
$ tree
```

```
.
+-- debhello-2.0
|   +-- LICENSE
|   +-- Makefile.am
|   +-- configure.ac
|   +-- data
|       | +-- hello.desktop
|       | +-- hello.png
|   +-- lib
|       | +-- Makefile.am
|       | +-- sharedlib.c
|       | +-- sharedlib.h
|   +-- man
|       | +-- Makefile.am
|       | +-- hello.1
|   +-- src
|       +-- Makefile.am
|       +-- hello.c
+-- debhello-2.0.tar.gz
```

5 directories, 13 files

Ниже приводится содержимое этого архива с исходным кодом.

src/hello.c (v=2.0):

```
$ cat debhello-2.0/src/hello.c
#include "config.h"
#include <stdio.h>
#include <sharedlib.h>
int
main()
{
    printf("Hello, I am " PACKAGE_AUTHOR "!\n");
    sharedlib();
    return 0;
}
```

}

lib/sharedlib.h и lib/sharedlib.c (v=1.6):

```
$ cat debhello-2.0/lib/sharedlib.h
int sharedlib();
$ cat debhello-2.0/lib/sharedlib.c
#include <stdio.h>
int
sharedlib()
{
    printf("This is a shared library!\n");
    return 0;
}
```

Makefile.am (v=2.0):

```
$ cat debhello-2.0/Makefile.am
# recursively process `Makefile.am` in SUBDIRS
SUBDIRS = lib src man
$ cat debhello-2.0/man/Makefile.am
# manpages (distributed in the source package)
dist_man_MANS = hello.1
$ cat debhello-2.0/lib/Makefile.am
# libtool librares to be produced
lib_LTLIBRARIES = libsharedlib.la

# source files used for lib_LTLIBRARIES
libsharedlib_la_SOURCES = sharedlib.c

# C pre-processor flags used for lib_LTLIBRARIES
#libsharedlib_la_CPPFLAGS =

# Headers files to be installed in <prefix>/include
include_HEADERS = sharedlib.h

# Versioning Libtool Libraries with version triplets
libsharedlib_la_LDFLAGS = -version-info 1:0:0
$ cat debhello-2.0/src/Makefile.am
# program executables to be produced
bin_PROGRAMS = hello

# source files used for bin_PROGRAMS
hello_SOURCES = hello.c

# C pre-processor flags used for bin_PROGRAMS
AM_CPPFLAGS = -I$(srcdir) -I$(top_srcdir)/lib

# Extra options for the linker for hello
# hello_LDFLAGS =

# Libraries the `hello` binary to be linked
hello_LDADD = $(top_srcdir)/lib/libsharedlib.la
```

configure.ac (v=2.0):

```
$ cat debhello-2.0/configure.ac
#
# Process this file with autoconf to produce a configure script.
AC_PREREQ([2.69])
AC_INIT([debhello], [2.2], [foo@example.org])
AC_CONFIG_SRCDIR([src/hello.c])
AC_CONFIG_HEADERS([config.h])
echo "Standard customization chores"
AC_CONFIG_AUX_DIR([build-aux])
#
#-*- Autoconf -*-
```

```

AM_INIT_AUTOMAKE([foreign])

# Set default to --enable-shared --disable-static
LT_INIT([shared disable-static])

# find the libltdl sources in the libltdl sub-directory
LT_CONFIG_LTDL_DIR([libltdl])

# choose one
LTDL_INIT([recursive])
#LTDL_INIT([subproject])
#LTDL_INIT([nonrecursive])

# Add #define PACKAGE_AUTHOR ... in config.h with a comment
AC_DEFINE(PACKAGE_AUTHOR, ["Osamu Aoki"], [Define PACKAGE_AUTHOR])
# Checks for programs.
AC_PROG_CC

# only for the recursive case
AC_CONFIG_FILES([Makefile
                 lib/Makefile
                 man/Makefile
                 src/Makefile])
AC_OUTPUT

```

Создадим пакет с помощью команды **debmake** в виде нескольких пакетов:

- **debhello**: type = **bin**
- **libsharedlib1**: type = **lib**
- **libsharedlib-dev**: type = **dev**

Here, the **-b',libsharedlib1,libsharedlib-dev'** option is used to specify the generated binary packages.

```

$ cd debhello-2.0
$ debmake -b',libsharedlib1,libsharedlib-dev' -e osamu@debian.org -f 'Osamu ...
I: set parameters
...
I: sanity check of parameters
I: pkg="debhello", ver="2.0", rev="1"
I: *** start packaging in "debhello-2.0". ***
I: provide debhello_2.0.orig.tar.gz for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-2.0.tar.gz debhello_2.0.orig.tar.gz
I: pwd = "/path/to/debhello-2.0"
I: parse binary package settings: ,libsharedlib1,libsharedlib-dev
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: binary package=libsharedlib1 Type=lib / Arch=any M-A=same
I: binary package=libsharedlib-dev Type=dev / Arch=any M-A=same
I: analyze the source tree
I: build_type = Autotools with autoreconf
...

```

Получившийся результат похож на то, что было описано в Section 8.8, но имеет большее количество шаблонных файлов.

Let's inspect the notable template files generated.

debian/rules (шаблонный файл, v=2.0):

```

$ cat debhello-2.0/debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

```

```

%:
    dh $@ --with autoreconf

#override_dh_install:
#    dh_install --list-missing -X.la -X.pyс -X.pyo

```

Сделаем этот пакет Debian лучше.

debian/rules (версия сопровождающего, v=2.0):

```

$ vim debhello-2.0/debian/rules
... hack, hack, hack, ...
$ cat debhello-2.0/debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

```

```

%:
    dh $@ --with autoreconf

override_dh_missing:
    dh_missing -X.la

```

debian/control (версия сопровождающего, v=2.0):

```

$ vim debhello-2.0/debian/control
... hack, hack, hack, ...
$ cat debhello-2.0/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13), dh-autoreconf
Standards-Version: 4.5.1
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends: libsharedlib1 (= ${binary:Version}),
        ${misc:Depends},
        ${shlibs:Depends}
Description: Simple packaging example for debmake
 This package contains the compiled binary executable.
.
 This Debian binary package is an example package.
 (This is an example only)

Package: libsharedlib1
Section: libs
Architecture: any
Multi-Arch: same
Pre-Depends: ${misc:Pre-Depends}
Depends: ${misc:Depends}, ${shlibs:Depends}
Description: Simple packaging example for debmake
 This package contains the shared library.

Package: libsharedlib-dev
Section: libdevel
Architecture: any
Multi-Arch: same
Depends: libsharedlib1 (= ${binary:Version}), ${misc:Depends}

```


Description: Simple packaging example for debmake
This package contains the development files.

debian/*.install (версия сопровождающего, v=2.0):

```
$ vim debhello-2.0/debian/debhello.install
... hack, hack, hack, ...
$ cat debhello-2.0/debian/debhello.install
usr/bin/*
usr/share/*
$ vim debhello-2.0/debian/libsharedlib1.install
... hack, hack, hack, ...
$ cat debhello-2.0/debian/libsharedlib1.install
usr/lib/*/*.so.*
$ vim debhello-2.0/debian/libsharedlib-dev.install
... hack, hack, hack, ...
$ cat debhello-2.0/debian/libsharedlib-dev.install
###usr/lib/*/pkgconfig/*.pc
usr/include
usr/lib/*/*.so
```

Since this upstream source creates the proper auto-generated **Makefile**, there is no need to create **debian/install** and **debian/manpages** files.

В каталоге **debian/** имеются и другие шаблонные файлы. Их также следует обновить.

Шаблонные файлы в каталоге debian/. (v=2.0):

```
$ tree debhello-2.0/debian
debhello-2.0/debian
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- debhello.install
+-- libsharedlib-dev.install
+-- libsharedlib1.install
+-- libsharedlib1.symbols
+-- patches
|   +-- series
+-- rules
+-- source
|   +-- control
|   +-- format
|   +-- local-options
|   +-- options
|   +-- patch-header
+-- tests
|   +-- control
+-- upstream
|   +-- metadata
+-- watch
```

4 directories, 18 files

Остальные работы по подготовке пакета практически полностью совпадают с описанными в Section 8.8. Here are the generated dependency list of all binary packages.

The generated dependency list of all binary packages (v=2.0):

```
$ dpkg -f debhello-dbgSYM_2.0-1_amd64.deb pre-depends depends recommends con...
Depends: debhello (= 2.0-1)
$ dpkg -f debhello_2.0-1_amd64.deb pre-depends depends recommends conflicts ...
Depends: libsharedlib1 (= 2.0-1), libc6 (>= 2.2.5)
$ dpkg -f libsharedlib-dev_2.0-1_amd64.deb pre-depends depends recommends co...
Depends: libsharedlib1 (= 2.0-1)
$ dpkg -f libsharedlib1-dbgSYM_2.0-1_amd64.deb pre-depends depends recommend...
Depends: libsharedlib1 (= 2.0-1)
```

```
$ dpkg -f libsharedlib1_2.0-1_amd64.deb pre-depends depends recommends confl...
Depends: libc6 (>= 2.2.5)
```

8.12 CMake (multi-binary package)

Here is an example of creating a set of Debian binary packages including the executable package, the shared library package, the development file package, and the debug symbol package from a simple C source program using CMake (**CMakeLists.txt** and some files such as **config.h.in**) as its build system. See Section 5.18.2.

Допустим tar-архив основной ветки имеет имя **debhello-2.1.tar.gz**.

Этот тип исходного кода предполагает установку в виде несистемного файла, например, как

```
$ tar -xzf debhello-2.1.tar.gz
$ cd debhello-2.1
$ mkdir obj-x86_64-linux-gnu
$ cd obj-x86_64-linux-gnu
$ cmake ..
$ make
$ make install
```

Получим исходный код и создадим пакет Debian.

Загрузим debhello-2.1.tar.gz

```
$ wget http://www.example.org/download/debhello-2.1.tar.gz
...
$ tar -xzf debhello-2.1.tar.gz
$ tree
```

```
.
+-- debhello-2.1
|   +-- CMakeLists.txt
|   +-- LICENSE
|   +-- data
|       |   +-- hello.desktop
|       |   +-- hello.png
|   +-- lib
|       |   +-- CMakeLists.txt
|       |   +-- sharedlib.c
|       |   +-- sharedlib.h
|   +-- man
|       |   +-- CMakeLists.txt
|       |   +-- hello.1
|   +-- src
|       +-- CMakeLists.txt
|       +-- config.h.in
|       +-- hello.c
+-- debhello-2.1.tar.gz
```

5 directories, 13 files

Ниже приводится содержимое этого архива с исходным кодом.

src/hello.c (v=2.1):

```
$ cat debhello-2.1/src/hello.c
#include "config.h"
#include <stdio.h>
#include <sharedlib.h>
int
main()
{
    printf("Hello, I am " PACKAGE_AUTHOR "!\n");
    sharedlib();
    return 0;
}
```

src/config.h.in (v=2.1):

```
$ cat debhello-2.1/src/config.h.in
/* name of the package author */
#define PACKAGE_AUTHOR "@PACKAGE_AUTHOR@"
```

lib/sharedlib.c и lib/sharedlib.h (v=2.1):

```
$ cat debhello-2.1/lib/sharedlib.h
int sharedlib();
$ cat debhello-2.1/lib/sharedlib.c
#include <stdio.h>
int
sharedlib()
{
    printf("This is a shared library!\n");
    return 0;
}
```

CMakeLists.txt (v=2.1):

```
$ cat debhello-2.1/CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(debhello)
set(PACKAGE_AUTHOR "Osamu Aoki")
add_subdirectory(lib)
add_subdirectory(src)
add_subdirectory(man)
$ cat debhello-2.1/man/CMakeLists.txt
install(
    FILES ${CMAKE_CURRENT_SOURCE_DIR}/hello.1
    DESTINATION share/man/man1
)
$ cat debhello-2.1/src/CMakeLists.txt
# Always define HAVE_CONFIG_H
add_definitions(-DHAVE_CONFIG_H)
# Generate config.h from config.h.in
configure_file(
    "${CMAKE_CURRENT_SOURCE_DIR}/config.h.in"
    "${CMAKE_CURRENT_BINARY_DIR}/config.h"
)
include_directories("${CMAKE_CURRENT_BINARY_DIR}")
include_directories("${CMAKE_SOURCE_DIR}/lib")

add_executable(hello hello.c)
target_link_libraries(hello sharedlib)
install(TARGETS hello
    RUNTIME DESTINATION bin
)
```

Создадим пакет из этого исходного кода с помощью команды **debmake**.

```
$ cd debhello-2.1
$ debmake -b',libsharedlib1,libsharedlib-dev' -e osamu@debian.org -f 'Osamu ...
I: set parameters
...
I: sanity check of parameters
I: pkg="debhello", ver="2.1", rev="1"
I: *** start packaging in "debhello-2.1". ***
I: provide debhello_2.1.orig.tar.gz for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-2.1.tar.gz debhello_2.1.orig.tar.gz
I: pwd = "/path/to/debhello-2.1"
I: parse binary package settings: ,libsharedlib1,libsharedlib-dev
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: binary package=libsharedlib1 Type=lib / Arch=any M-A=same
```

```
I: binary package=libsharedlib-dev Type=dev / Arch=any M-A=same
I: analyze the source tree
I: build_type = Cmake
...
```

Получившийся результат похож на то, что было описано в Section 8.8, но не совпадает с ним в точности. Let's inspect the notable template files generated.

debian/rules (шаблонный файл, v=2.1):

```
$ cat debhello-2.1/debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
    dh $@

#override_dh_auto_configure:
#    dh_auto_configure -- \
#        -DCMAKE_LIBRARY_ARCHITECTURE="$(DEB_TARGET_MULTIARCH)"
```

Сделаем этот пакет Debian лучше.

debian/rules (версия сопровождающего, v=2.1):

```
$ vim debhello-2.1/debian/rules
... hack, hack, hack, ...
$ cat debhello-2.1/debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed
DEB_HOST_MULTIARCH ?= $(shell dpkg-architecture -qDEB_HOST_MULTIARCH)

%:
    dh $@

override_dh_auto_configure:
    dh_auto_configure -- \
        -DCMAKE_LIBRARY_ARCHITECTURE="$(DEB_HOST_MULTIARCH)"
```

debian/control (версия сопровождающего, v=2.1):

```
$ vim debhello-2.1/debian/control
... hack, hack, hack, ...
$ cat debhello-2.1/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: cmake, debhelper-compat (= 13)
Standards-Version: 4.5.1
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends: libsharedlib1 (= ${binary:Version}),
        ${misc:Depends},
        ${shlibs:Depends}
Description: Simple packaging example for debmake
```

```
This package contains the compiled binary executable.
.
This Debian binary package is an example package.
(This is an example only)
```

```
Package: libsharedlib1
Section: libs
Architecture: any
Multi-Arch: same
Pre-Depends: ${misc:Pre-Depends}
Depends: ${misc:Depends}, ${shlibs:Depends}
Description: Simple packaging example for debmake
 This package contains the shared library.

Package: libsharedlib-dev
Section: libdevel
Architecture: any
Multi-Arch: same
Depends: libsharedlib1 (= ${binary:Version}), ${misc:Depends}
Description: Simple packaging example for debmake
 This package contains the development files.
```

debian/*.install (версия сопровождающего, v=2.1):

```
$ vim debhello-2.1/debian/debhello.install
... hack, hack, hack, ...
$ cat debhello-2.1/debian/debhello.install
usr/bin/*
usr/share/*
$ vim debhello-2.1/debian/libsharedlib1.install
... hack, hack, hack, ...
$ cat debhello-2.1/debian/libsharedlib1.install
usr/lib/*/*.so.*
$ vim debhello-2.1/debian/libsharedlib-dev.install
... hack, hack, hack, ...
$ cat debhello-2.1/debian/libsharedlib-dev.install
###usr/lib/*/pkgconfig/*.pc
usr/include
usr/lib/*/*.so
```

Файл CMakeList.txt из основной ветки необходимо изменить тк, чтобы в нём была обеспечена поддержка мультиархитектурных путей.

debian/patches/* (версия сопровождающего, v=2.1):

```
... hack, hack, hack, ...
$ cat debhello-2.1/debian/libsharedlib1.symbols
libsharedlib.so.1 libsharedlib1 #MINVER#
sharedlib@Base 2.1
```

Since this upstream source creates the proper auto-generated **Makefile**, there is no need to create **debian/install** and **debian/manpages** files.

В каталоге **debian/** имеются и другие шаблонные файлы. Их также следует обновить.

Шаблонные файлы в каталоге debian/. (v=2.1):

```
$ tree debhello-2.1/debian
debhello-2.1/debian
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- debhello.install
+-- libsharedlib-dev.install
+-- libsharedlib1.install
+-- libsharedlib1.symbols
+-- patches
```

```

|   +-- 000-cmake-multiarch.patch
|   +-- series
+-- rules
+-- source
|   +-- control
|   +-- format
|   +-- local-options
|   +-- options
|   +-- patch-header
+-- tests
|   +-- control
+-- upstream
|   +-- metadata
+-- watch

4 directories, 19 files

```

Остальные работы по подготовке пакета практически полностью совпадают с описанными в Section 8.8. Here are the generated dependency list of all binary packages.

The generated dependency list of all binary packages (v=2.1):

```

$ dpkg -f debhello-dbgSYM_2.1-1_amd64.deb pre-depends depends recommends con...
Depends: debhello (= 2.1-1)
$ dpkg -f debhello_2.1-1_amd64.deb pre-depends depends recommends conflicts ...
Depends: libsharedlib1 (= 2.1-1), libc6 (>= 2.2.5)
$ dpkg -f libsharedlib-dev_2.1-1_amd64.deb pre-depends depends recommends co...
Depends: libsharedlib1 (= 2.1-1)
$ dpkg -f libsharedlib1-dbgSYM_2.1-1_amd64.deb pre-depends depends recommend...
Depends: libsharedlib1 (= 2.1-1)
$ dpkg -f libsharedlib1_2.1-1_amd64.deb pre-depends depends recommends confl...
Depends: libc6 (>= 2.2.5)

```

8.13 Интернационализация

Here is an example of updating the simple upstream C source **debhello-2.0.tar.gz** presented in Section 8.11 for internationalization (i18n) and creating the updated upstream C source **debhello-2.0.tar.gz**.

In the real situation, the package should already be internationalized. So this example is educational for you to understand how this internationalization is implemented.

Tip



The routine maintainer activity for the i18n is simply to add translation po files reported to you via the Bug Tracking System (BTS) to the **po/** directory and to update the language list in the **po/LINGUAS** file.

Получим исходный код и создадим пакет Debian.

Загрузим debhello-2.0.tar.gz (i18n)

```

$ wget http://www.example.org/download/debhello-2.0.tar.gz
...
$ tar -xzmf debhello-2.0.tar.gz
$ tree
.
+-- debhello-2.0
|   +-- LICENSE
|   +-- Makefile.am
|   +-- configure.ac
|   +-- data
|       |   +-- hello.desktop
|       |   +-- hello.png

```

```

|   +-- lib
|   |   +-- Makefile.am
|   |   +-- sharedlib.c
|   |   +-- sharedlib.h
|   +-- man
|   |   +-- Makefile.am
|   |   +-- hello.1
|   +-- src
|       +-- Makefile.am
|       +-- hello.c
+-- debhello-2.0.tar.gz

5 directories, 13 files

```

Internationalize this source tree with the **gettextize** command and remove files auto-generated by Autotools.
запуск gettextize (i18n):

```

$ cd debhello-2.0
$ gettextize
Creating po/ subdirectory
Creating build-aux/ subdirectory
Copying file ABOUT-NLS
Copying file build-aux/config.rpath
Not copying intl/ directory.
Copying file po/Makefile.in.in
Copying file po/Makevars.template
Copying file po/Rules-quot
Copying file po/boldquot.sed
Copying file po/en@boldquot.header
Copying file po/en@quot.header
Copying file po/insert-header.sin
Copying file po/quot.sed
Copying file po/remove-potcdate.sin
Creating initial po/POTFILES.in
Creating po/ChangeLog
Creating directory m4
Copying file m4/gettext.m4
Copying file m4/iconv.m4
Copying file m4/lib-ld.m4
Copying file m4/lib-link.m4
Copying file m4/lib-prefix.m4
Copying file m4/nls.m4
Copying file m4/po.m4
Copying file m4/progtest.m4
Creating m4/ChangeLog
Updating Makefile.am (backup is in Makefile.am~)
Updating configure.ac (backup is in configure.ac~)
Creating ChangeLog

Please use AM_GNU_GETTEXT([external]) in order to cause autoconfiguration
to look for an external libintl.

Please create po/Makevars from the template in po/Makevars.template.
You can then remove po/Makevars.template.

Please fill po/POTFILES.in as described in the documentation.

Please run 'aclocal' to regenerate the aclocal.m4 file.
You need aclocal from GNU automake 1.9 (or newer) to do this.
Then run 'autoconf' to regenerate the configure file.

You will also need config.guess and config.sub, which you can get from the CV...
of the 'config' project at http://savannah.gnu.org/. The commands to fetch th...
are
$ wget 'http://savannah.gnu.org/cgi-bin/viewcvs/*checkout*/config/config/conf...

```

```
$ wget 'http://savannah.gnu.org/cgi-bin/viewcvs/*checkout*/config/config/conf...
```

You might also want to copy the convenience header file `gettext.h` from the `/usr/share/gettext` directory into your package. It is a wrapper around `<libintl.h>` that implements the `configure --disable-nl...` option.

Press Return to acknowledge the previous 6 paragraphs.

```
$ rm -rf m4 build-aux *~
```

Проверим созданные файлы в каталоге `po/`.
файлы в каталоге `po` (i18n):

```
$ ls -l po
total 60
-rw-rw-r-- 1 osamu osamu  494 Jul  2 16:28 ChangeLog
-rw-rw-r-- 1 osamu osamu 17577 Jul  2 16:28 Makefile.in.in
-rw-rw-r-- 1 osamu osamu  3376 Jul  2 16:28 Makevars.template
-rw-rw-r-- 1 osamu osamu   59 Jul  2 16:28 POTFILES.in
-rw-rw-r-- 1 osamu osamu  2203 Jul  2 16:28 Rules-quot
-rw-rw-r-- 1 osamu osamu   217 Jul  2 16:28 boldquot.sed
-rw-rw-r-- 1 osamu osamu  1337 Jul  2 16:28 en@boldquot.header
-rw-rw-r-- 1 osamu osamu  1203 Jul  2 16:28 en@quot.header
-rw-rw-r-- 1 osamu osamu   672 Jul  2 16:28 insert-header.sin
-rw-rw-r-- 1 osamu osamu   153 Jul  2 16:28 quot.sed
-rw-rw-r-- 1 osamu osamu   432 Jul  2 16:28 remove-potcdate.sin
```

Обновим файл `configure.ac`, добавив «`AM_GNU_GETTEXT([external])`» и т. д.
`configure.ac` (i18n):

```
$ vim configure.ac
... hack, hack, hack, ...
$ cat configure.ac
#                                     -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.
AC_PREREQ([2.69])
AC_INIT([dehello], [2.2], [foo@example.org])
AC_CONFIG_SRCDIR([src/hello.c])
AC_CONFIG_HEADERS([config.h])
echo "Standard customization chores"
AC_CONFIG_AUX_DIR([build-aux])

AM_INIT_AUTOMAKE([foreign])

# Set default to --enable-shared --disable-static
LT_INIT([shared disable-static])

# find the libltdl sources in the libltdl sub-directory
LT_CONFIG_LTDL_DIR([libltdl])

# choose one
LTDL_INIT([recursive])
#LTDL_INIT([subproject])
#LTDL_INIT([nonrecursive])

# Add #define PACKAGE_AUTHOR ... in config.h with a comment
AC_DEFINE(PACKAGE_AUTHOR, ["Osamu Aoki"], [Define PACKAGE_AUTHOR])
# Checks for programs.
AC_PROG_CC

# desktop file support required
AM_GNU_GETTEXT_VERSION([0.19.3])
AM_GNU_GETTEXT([external])

# only for the recursive case
```



```
AC_CONFIG_FILES([Makefile
                 po/Makefile.in
                 lib/Makefile
                 man/Makefile
                 src/Makefile])
AC_OUTPUT
```

Let's create the **po/Makevars** file from the **po/Makevars.template** file.

po/Makevars (i18n):

```
... hack, hack, hack, ...
$ diff -u po/Makevars.template po/Makevars
--- po/Makevars.template      2021-07-02 16:28:02.711724581 +0900
+++ po/Makevars 2021-07-02 16:28:02.767725251 +0900
@@ -18,14 +18,14 @@
# or entity, or to disclaim their copyright. The empty string stands for
# the public domain; in this case the translators are expected to disclaim
# their copyright.
-COPYRIGHT HOLDER = Free Software Foundation, Inc.
+COPYRIGHT HOLDER = Osamu Aoki <osamu@debian.org>

# This tells whether or not to prepend "GNU " prefix to the package
# name that gets inserted into the header of the $(DOMAIN).pot file.
# Possible values are "yes", "no", or empty. If it is empty, try to
# detect it automatically by scanning the files in $(top_srcdir) for
# "GNU packagename" string.
-PACKAGE_GNU =
+PACKAGE_GNU = no

# This is the email address or URL to which the translators shall report
# bugs in the untranslated strings:
$ rm po/Makevars.template
```

Let's update C sources for the i18n version by wrapping strings with **_(...)**.

src/hello.c (i18n):

```
... hack, hack, hack, ...
$ cat src/hello.c
#include "config.h"
#include <stdio.h>
#include <sharedlib.h>
#define _(string) gettext (string)
int
main()
{
    printf(_("Hello, I am " PACKAGE_AUTHOR "!\n"));
    sharedlib();
    return 0;
}
```

lib/sharedlib.c (i18n):

```
... hack, hack, hack, ...
$ cat lib/sharedlib.c
#include <stdio.h>
#define _(string) gettext (string)
int
sharedlib()
{
    printf(_("This is a shared library!\n"));
    return 0;
}
```

The new **gettext** (v=0.19) can handle the i18n version of the desktop file directly.

data/hello.desktop.in (i18n):

```
$ fgrep -v '[ja]=' data/hello.desktop > data/hello.desktop.in
$ rm data/hello.desktop
$ cat data/hello.desktop.in
[Desktop Entry]
Name=Hello
Comment=Greetings
Type=Application
Keywords=hello
Exec=hello
Terminal=true
Icon=hello.png
Categories=Utility;
```

Приведём список входных файлов для извлечения переводных строк в **po/POTFILES.in**.

```
... hack, hack, hack, ...
$ cat po/POTFILES.in
src/hello.c
lib/sharedlib.c
data/hello.desktop.in
```

Here is the updated root **Makefile.am** with **po** added to the **SUBDIRS** environment variable.

Makefile.am (i18n):

```
$ cat Makefile.am
# recursively process `Makefile.am` in SUBDIRS
SUBDIRS = po lib src man

ACLOCAL_AMFLAGS = -I m4

EXTRA_DIST = build-aux/config.rpath m4/ChangeLog
```

Let's make a translation template file, **debhello.pot**.

po/debhello.pot (i18n):

```
$ xgettext -f po/POTFILES.in -d debhello -o po/debhello.pot -k_
$ cat po/debhello.pot
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2021-07-02 16:28+0900\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"Language: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"

#: src/hello.c:8
#, c-format
msgid "Hello, I am "
msgstr ""

#: lib/sharedlib.c:6
#, c-format
msgid "This is a shared library!\n"
```

```
msgstr ""

#: data/hello.desktop.in:3
msgid "Hello"
msgstr ""

#: data/hello.desktop.in:4
msgid "Greetings"
msgstr ""

#: data/hello.desktop.in:6
msgid "hello"
msgstr ""
```

Let's add a translation for French.

po/LINGUAS и po/fr.po (i18n):

```
$ echo 'fr' > po/LINGUAS
$ cp po/debhello.pot po/fr.po
$ vim po/fr.po
... hack, hack, hack, ...
$ cat po/fr.po
# SOME DESCRIPTIVE TITLE.
# This file is put in the public domain.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
msgid ""
msgstr ""
"Project-Id-Version: debhello 2.2\n"
"Report-Msgid-Bugs-To: foo@example.org\n"
"POT-Creation-Date: 2015-03-01 20:22+0900\n"
"PO-Revision-Date: 2015-02-21 23:18+0900\n"
"Last-Translator: Osamu Aoki <osamu@debian.org>\n"
"Language-Team: French <LL@li.org>\n"
"Language: ja\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"

#: src/hello.c:34
#, c-format
msgid "Hello, my name is %s!\n"
msgstr "Bonjour, je m'appelle %s!\n"

#: lib/sharedlib.c:29
#, c-format
msgid "This is a shared library!\n"
msgstr "Ceci est une bibliothèque partagée!\n"

#: data/hello.desktop.in:3
msgid "Hello"
msgstr ""

#: data/hello.desktop.in:4
msgid "Greetings"
msgstr "Salutations"

#: data/hello.desktop.in:6
msgid "hello"
msgstr ""

#: data/hello.desktop.in:9
msgid "hello.png"
msgstr ""
```

Работа над подготовкой пакета практически полностью совпадает с тем, что описывается в Section 8.11. You can find more i18n examples by following Section 8.14.

8.14 Детали

Детали представленных в документации примеров, а также их вариантов, можно получить следующим образом.

Как получить детали

```
$ apt-get source debmake-doc
$ cd debmake-doc*
$ cd examples
$ view examples/README.md
```

Follow the exact instruction in `examples/README.md`.

```
$ cd examples
$ sudo apt-get install devscripts build-essentials ...
$ make
```

Now, each directory named as `examples/debhello-?._?_build-?` contains the Debian packaging example.

- эмулированный журнал активности командной строки консоли: файл **.log**
- эмулированный журнал активности командной строки консоли (короткий): файл **.slog**
- срез образа дерева исходного кода после выполнения команды **debmake**: каталог **debmake**
- snapshot source tree image after proper packaging: the **packge** directory
- срез образа дерева исходного кода после выполнения команды **debuild**: каталог **test**

Notable examples are:

- the Python3 script with bare distutils (v=1.8),
- the POSIX shell script with Makefile with i18n (v=3.0),
- the Python3 script with setuptools with i18n (v=3.1),
- the C source with Makefile.in + configure with i18n (v=3.2),
- the C source with Autotools with i18n with i18n (v=3.3), and
- the C source with CMake with i18n (v=3.4).

Appendix A

Страница руководства debmake(1)

A.1 НАЗВАНИЕ

debmake - program to make a Debian source package

A.2 СИНТАКСИС

debmake [-h] [-c | -k] [-n | -a пакет-версия.orig.tar.gz | -d | -t] [-p пакет] [-u версия] [-r редакция] [-z расширение] [-b "двоичныйпакет, ...]" [-e foo@example.org] [-f "имя фамилия"] [-i "инструментсборки" | -j] [-l файл_лицензии] [-m] [-o файл] [-q] [-s] [-v] [-w "дополнение, ...]" [-x [01234]] [-y] [-L] [-P] [-T]

A.3 ОПИСАНИЕ

debmake помогает собрать пакет Debian из исходного кода основной ветки разработки. Обычно это делается следующим образом:

- Загружается tar-архив основной ветки разработки в виде файла *пакет-версия.tar.gz*.
- Исходный код распаковывается, создаются файлы в каталоге *пакет-версия/*.
- Вызывается **debmake** в каталоге *пакет-версия/*, возможно, без аргументов.
- Файлы в каталоге *package-version/debian/* настраиваются вручную.
- **dpkg-buildpackage** (usually from its wrapper **debuild** or **sbuild**) is invoked in the *package-version/* directory to make Debian packages.

Обязательно защитите путём соответствующего включения в кавычки аргументы опций **-b**, **-f**, **-l** и **-w** от вмешательства командной оболочки.

A.3.1 необязательные аргументы:

-h, --help показать справочное сообщение и выйти.

-c, --copyright сканировать исходный код на предмет текста об авторском праве и лицензировании и выйти.

- **-c**: простой стиль вывода
- **-cc**: обычный стиль вывода (схож с файлом **debian/copyright**)
- **-ccc**: отладочный стиль вывода

-k, --kludge сравнить файл **debian/copyright** с исходным кодом и выйти.

Файл **debian/copyright** должен быть организован таким образом, что наиболее общие файловые шаблоны размещаются раньше конкретных исключений.

- **-k**: простой стиль вывода

- **-kk**: подробный стиль вывода

-n, --native make a native Debian source package without **.orig.tar.gz**. This makes a “**3.0 (native)**” format package.

If you are thinking of packaging a Debian-specific source tree with **debian/*** in it into a native Debian package, please think otherwise. You can use the “**debmake -d -i debuild**” or “**debmake -t -i debuild**” commands to make a “**3.0 (quilt)**” format non-native Debian package. The only difference is that the **debian/changelog** file must use the non-native version scheme: *version-revision*. The non-native package is more friendly to downstream distributions.

-a пакет-версия.tar.gz, --archive пакет-версия.tar.gz использовать непосредственно tar-архив с исходным кодом основной ветки. (отменяются опции **-p, -u, -z**)

The upstream tarball may be specified as *package_version.orig.tar.gz* and **tar.gz**. For other cases, it may be **tar.bz2**, or **tar.xz**.

Если в имени указанного tar-архива основной ветки содержатся буквы в верхнем регистре, то в имени пакета Debian они будут преобразованы в буквы нижнего регистра.

Если в качестве аргумента указан URL (<http://>, <https://> или <ftp://>) tar-архива основной ветки, то этот архив скачивается с помощью **wget** или **curl**.

-d, --dist сначала запустить эквиваленты команды «make dist» для создания tar-архива основной ветки, затем использовать его.

Команда «**debmake -d**» предназначена для запуска в каталоге *пакет/*, в котором расположена система управления версиями и система сборки, поддерживающая эквиваленты команды «**make dist**». (autotake/autoconf, Python distutils, ...)

-t, --tar запустить команду «**tar**» для создания tar-архива основной ветки, затем использовать его.

The “**debmake -t**” command is designed to run in the *package/* directory hosting the upstream VCS. Unless you provide the upstream version with the **-u** option or with the **debian/changelog** file, a snapshot upstream version is generated in the **0~%y%m%d%H%M** format, e.g., *0~1403012359*, from the UTC date and time. The generated tarball excludes the **debian/** directory found in the upstream VCS. (It also excludes typical VCS directories: **.git/ .hg/ .svn/ .CVS/**.)

-p пакет, --package пакет установить имя пакета Debian.

-u версия, --upstreamversion версия установить версию пакета основной ветки.

-r редакция, --revision редакция установить номер редакции пакета Debian.

-z расширение, --targz расширение set the tarball type, *extension*=(**tar.gz|tar.bz2|tar.xz**). (alias: **z, b, x**)

-b "двоичныйпакет[:тип],...", **--binaryspec "двоичныйпакет[:тип],..."** set the binary package specs by a comma separated list of *binarypackage:type* pairs, e.g., in the full form “**foo:bin,foo-doc:doc,libfoo1:lib,libfoo-dev:dev**” or in the short form, “**-doc,libfoo1,libfoo-dev**”.

Here, *binarypackage* is the binary package name, and the optional *type* is chosen from the following *type* values:

- **bin**: скомпилированный двоичный пакет с исполняемыми файлами формата ELF на языке C/C++ (any, foreign) (по умолчанию, псевдоним: **”**, то есть *пустая-строка*)
- **data**: пакет с данными (шрифты, графика, ...) (all, foreign) (псевдоним: **da**)
- **dev**: пакет с библиотекой разработки (any, same) (псевдоним: **de**)
- **doc**: пакет документации (all, foreign) (псевдоним: **do**)
- **lib**: пакет с библиотекой (any, same) (псевдоним: **l**)
- **perl**: пакет со сценарием на языке Perl (all, foreign) (псевдоним: **pl**)
- **python3**: Python (version 3) script package (all, foreign) (alias: **py3**)
- **ruby**: пакет со сценарием на языке Ruby (all, foreign) (псевдоним: **rb**)
- **nodejs**: Node.js based JavaScript package (all, foreign) (alias: **js**)
- **script**: пакет со сценарием командной оболочки (all, foreign) (псевдоним: **sh**)

Пары значений в скобках, такие как (any, foreign), представляют собой значения служебных строк **Architecture** и **Multi-Arch**, устанавливаемые в файле **debian/control**.

Во многих случаях команда **debmake** довольно хорошо предсказывает значение поля *тип*, исходя из значения поля *двоичный пакет*. Если *тип* не очевиден, то значением поля *тип* становится **bin**. Например, исходя из **libfoo** значением поля *тип* становится **lib**, а исходя из **font-bar** значением поля *тип* становится **data**, ...

Если содержимое дерева исходного кода не совпадает с настройками поля *тип*, то команда **debmake** выводит предупреждение.

-e foo@example.org, --email foo@example.org установить адрес электронной почты.

По умолчанию берётся значение переменной окружения **\$DEBEMAIL**.

-f "имя фамилия", --fullname "имя фамилия" установить имя и фамилию.

По умолчанию берётся значение переменной окружения **\$DEBFULLNAME**.

-i "инструмент сборки", --invoke "инструмент сборки" invoke "buildtool" at the end of execution. *build-tool* may be "**dpkg-buildpackage**", "**debuild**", "**sbuild**", etc.

По умолчанию никакая программа не выполняется.

Передача этой опции автоматически приводит к передаче опции **--local**.

-j, --judge запустить **dpkg-depcheck** для выявления сборочных зависимостей и определения путей файлов. Файлы журнала располагаются в родительском каталоге.

- **package.build-dep.log**: файл журнала **dpkg-depcheck**.
- **package.install.log**: файл журнала, в который записываются файлы из каталога **debian/tmp**.

-l "файл_лицензии,...", --license "файл_лицензии,..." add formatted license text to the end of the **debian/copyright** file holding license scan results.

The default is to add **COPYING** and **LICENSE**, and *license_file* needs to list only the additional file names all separated by ",".

-m, --monoarch подготовить пакеты без поддержки мультиархитектурности.

-o файл, --option файл read optional parameters from *file*. (This is not for everyday use.)

The content of *file* is sourced as the Python code at the end of **para.py**. For example, the package description can be specified by the following file.

```
para['desc'] = 'program short description'
para['desc_long'] = '''\
program long description which you wish to include.
.
Empty line is space + .
You keep going on ...
'''
```

-q, --quitearly выйти до создания файлов в каталоге **debian/**.

-s, --spec использовать spec-файло основной ветки (в случае исходного кода на языке Python используется **setup.py** и т. д.) в качестве источника описания пакета.

-v, --version показать информацию о версии.

-w "дополнение,...", --with "дополнение,..." добавить дополнительные аргументы опции **--with** команды **dh(1)** в качестве дополнений в файл **debian/rules**.

The *addon* values are listed all separated by " , ", e.g., "**-w python3,autoreconf**".

For Autotools based packages, **autoreconf** as *addon* to run "**autoreconf -i -v -f**" for every package building is default behavior of the **dh(1)** command.

For Autotools based packages, if they install Python (version 3) programs, setting **python3** as *addon* to the **debmake** command argument is needed since this is non-obvious. But for **setup.py** based packages, setting **python3** as *addon* to the **debmake** command argument is not needed since this is obvious and the **debmake** command automatically set it to the **dh(1)** command.

-x n, --extra n generate configuration files as templates. (Please note **debian/changelog**, **debian/control**, **debian/copyright**, and **debian/rules** are bare minimum configuration files to build a Debian binary package.)

The number *n* determines which configuration templates are generated.

- **-x0**: bare minimum configuration files. (default option if any of bare minimum configuration files already exist)
- **-x1**: all **-x0** files + desirable configuration files for the single binary package. (default option for the single binary package if none of bare minimum configuration files exist)
- **-x2**: all **-x1** files + desirable configuration files for the multi binary package. (default option for the multi binary package if none of bare minimum configuration files exist)
- **-x3**: all **-x2** files + unusual configuration template files. Unusual configuration template files are generated with the extra **.ex** suffix to ease their removal. To use these as configuration files, rename their file names to ones without the **.ex** suffix.
- **-x4**: all **-x3** files + copyright file examples.

-y, --yes «отвечать утвердительно» на все вопросы. (без опции: «спрашивать [Y/n]»; двочная опция: «отвечать отрицательно»)

-L, --local создать файлы настройки для локального пакета, чтобы перехитрить проверки **lintian(1)**.

-P, --pedantic педантично проверять автоматически создаваемые файлы.

-T, --tutorial вывести в шаблонных файлах строки с обучающими комментариями.

A.4 ПРИМЕРЫ

For a well behaving source, you can build a good-for-local-use installable single Debian binary package easily with one command. Test install of such a package generated in this way offers a good alternative to the traditional “**make install**” command installing into the **/usr/local** directory since the Debian package can be removed cleanly by the “**dpkg -P ...**” command. Here are some examples of how to build such test packages. (These should work in most cases. If the **-d** option does not work, try the **-t** option instead.)

Для дерева исходного кода обычной программы на языке C с autoconf/automake:

- **debmake -d -i debuild**

For a typical Python (version 3) module source tree:

- **debmake -s -d -b”:python3” -i debuild**

For a typical Python (version 3) module in the *package-version.tar.gz* archive:

- **debmake -s -a package-version.tar.gz -b”:python3” -i debuild**

Для обычного модуля языка Perl в виде архива *пакет-версия.tar.gz*:

- **debmake -a пакет-версия.tar.gz -b”:perl” -i debuild**

A.5 ВСПОМОГАТЕЛЬНЫЕ ПАКЕТЫ

Для работы над пакетами может потребоваться установка некоторых дополнительных специализированных вспомогательных пакетов.

- Python (version 3) programs may require the **dh-python** package.
- The Autotools (Autoconf + Automake) build system may require **autotools-dev** or **dh-autoreconf** package.
- Ruby programs may require the **gem2deb** package.
- Node.js based JavaScript programs may require the **pkg-js-tools** package.
- Java programs may require the **javahelper** package.
- Для программ для окружения Gnome может потребоваться пакет **gobject-introspection**.
- и т. д.

A.6 ПРЕДОСТЕРЕЖЕНИЯ

Утилита **debmake** предназначена для создания шаблонных файлов в помощь сопровождающему пакету. Строки с комментариями начинаются с символа **#** и содержат обучающий текст. Вам следует удалить или отредактировать строки с комментариями до выполнения загрузки пакета в архив Debian.

The license extraction and assignment process involves a lot of heuristics; it may fail in some cases. It is highly recommended to use other tools such as **licensecheck** from the **devscripts** package in conjunction with **debmake**.

There are some limitations for what characters may be used as a part of the Debian package. The most notable limitation is the prohibition of uppercase letters in the package name. Here is a summary as a set of regular expressions:

- Имя пакета основной ветки разработки (**-p**): `[-+a-z0-9]{2,}`
- Имя двоичного пакета (**-b**): `[-+a-z0-9]{2,}`
- Версия основной ветки разработки (**-u**): `[0-9][-+~a-z0-9A-Z]*`
- Редакция Debian (**-r**): `[0-9][+~a-z0-9A-Z]*`

See the exact definition in [Chapter 5 - Control files and their fields](#) in the “Debian Policy Manual”.

debmake предполагает относительно простые случаи создания пакетов. Поэтому все программы, относящиеся к интерпретатору, считаются «**Architecture: all**». Тем не менее, это не всегда так.

A.7 ОТЛАДКА

Сообщения об ошибках отправляйте с помощью команды **reportbug** для пакета **debmake**.

Набор символов в переменной окружения **\$DEBUG** определяет уровень вывода журнала.

- **i**: печать информации
- **p**: вывод всех глобальных параметров
- **d**: вывод всех грамматически разобранных параметров для всех двоичных пакетов
- **f**: ввод имени файла для сканирования copyright
- **y**: разделение год/имя для строки об авторском праве
- **s**: сканер строки для format_state
- **b**: цикл сканирования content_state: начало-конец
- **m**: цикл сканирования content_state: после совпадения с регулярным выражением
- **e**: цикл сканирования content_state: конец-цикл
- **c**: печать текста из раздела об авторских правах
- **l**: печать текста из раздела о лицензиях
- **a**: печать текста из раздела автор/переводчик
- **k**: сортировать ключи для строк debian/copyright
- **n**: сканировать результат debian/copyright («**debmake -k**»)

Используйте эту переменную следующим образом:

```
$ DEBUG=pdfbmc lak debmake ...
```

Дополнительную информацию см. в файле README.developer.

A.8 АВТОР

Copyright © 2014-2021 Osamu Aoki <osamu@debian.org>

А.9 ЛИЦЕНЗИЯ

Лицензия Ехрат

А.10 СМОТРИТЕ ТАКЖЕ

The **debmake-doc** package provides the “Guide for Debian Maintainers” in plain text, HTML and PDF formats under the `/usr/share/doc/debmake-doc/` directory.

See also **dpkg-source**(1), **deb-control**(5), **debhelper**(7), **dh**(1), **dpkg-buildpackage**(1), **debuild**(1), **quilt**(1), **dpkg-depcheck**(1), **sbuid**(1), **gbp-buildpackage**(1), and **gbp-pq**(1) manpages.