Ubuntu Packaging Guide

Выпуск 0.3.8 bzr603 ubuntu14.04.1

Ubuntu Developers

12 April 2017

Оглавление

Добро пожаловать в руководство разработчика Ubuntu! Это официальная документация по всем темам, связанных с разработкой Ubuntu и сборкой пакетов для этой операционной системы. После прочтения этого руководства вы:

- будете знать о самых важных утилитах, процессах и командах в разработке Ubuntu,
- сможете правильно настроить вашу среду разработки,
- узнаете, как присоединиться к нашему сообществу,
- исправите настоящую ошибку в Ubuntu в процессе изучения руководства.

Ubuntu — не только свободная операционная система с открытым исходным кодом, её платформа также является открытой и обеспечивает прозрачность разработки. Можно легко получить исходный код для каждого отдельного компонента, и каждое отдельное изменение в платформе Ubuntu можно проверить.

Это означает, что вы можете принять активное участие в её улучшении, и сообщество разработчиков платформы Ubuntu всегда заинтересовано в привлечении новых участников.

Ubuntu также является сообществом замечательных людей, верящих в то, что программное обеспечение должно быть свободным и доступным для всех. Участники сообщества приветствуют вас и хотят, чтобы вы тоже к ним присоединились. Мы хотим, чтобы вы принимали участие в нашей работе, задавали вопросы, делали Ubuntu лучше вместе с нами.

Если у вас возникнут трудности: не волнуйтесь! Прочтите *раздел о коммуникации*, и вы узнаете, как легко связаться с остальными разработчиками.

Это руководство состоит из двух разделов:

- Список статей, основанных на определённых задачах, которые вам может понадобиться выполнить.
- Набор статей базы знаний, в которых подробнее рассматриваются используемые нами инструменты и рабочие процессы.

Это руководство фокусируется на методе создания пакетов Ubuntu Distributed Development. Это новый способ работы с пакетами, который использует ветки распределённой системы управления версиями. В настоящее время он имеет некоторые ограничения, поэтому многие команды Ubuntu по-прежнему пользуются *традиционными* методами создания пакетов. Чтобы узнать о различиях, смотрите страницу *Введение в UDD*.

Глава 1

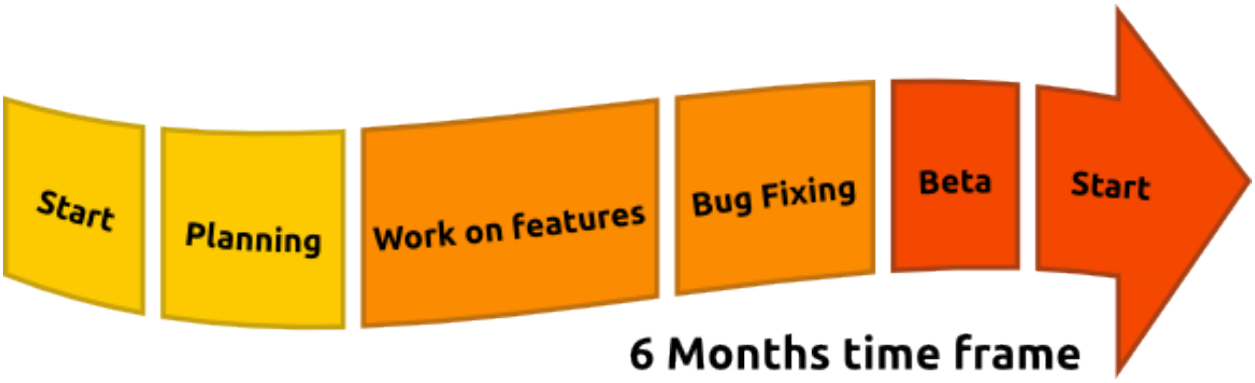
Статьи

1.1 Введение в разработку Ubuntu

Ubuntu состоит из тысяч различных компонентов, написанных на множестве языков программирования. Каждый компонент — библиотека, утилита или графическое приложение — доступен в виде пакета исходного кода. Пакты исходных кодов в большинстве случаев состоят из двух частей: сам исходный код и метаданные. Метаданные включают в себя зависимости пакета, информацию об авторском праве и лицензии, а также инструкции по сборке пакета. После того, как пакет исходных кодов скомпилирован, в процессе сборки мы получим двоичные пакеты, являющиеся .deb файлами, которые пользователи могут установить.

Каждый раз, когда выходит новая версия приложения, или когда кто-то вносит изменения в исходный код пакета, входящего в состав Ubuntu, пакет с исходным кодом должен быть загружен на сборочные компьютеры Launchpad для компиляции. Готовые бинарные пакеты затем попадают в репозиторий и его зеркала в разных странах. URL в `/etc/apt/sources.list` являются ссылками на репозиторий или его зеркал. Каждый день собираются образы для различных версий Ubuntu. Они могут быть использованы в различных условиях. Есть образы, которые можно записать на USB-носители или на DVD-диски, образы, которые можно использовать для сетевой загрузки и есть образы, предназначенные для установки на телефон или планшет. Ubuntu, серверная версия Ubuntu, Kubuntu и другие ветки имеют специфичный список необходимых пакетов, которые попадают в образ. Эти образы, которые затем используются для проверки установки и обеспечивают обратную связь для дальнейшего планирования выпуска.

Разработка Ubuntu сильно зависит от текущей фазы цикла выпуска. Мы выпускаем новую версию Ubuntu каждые шесть месяцев, что возможно только благодаря тому, что мы устанавливаем точные даты «заморозки». По достижении каждой даты заморозки ожидается, что разработчики будут вносить более редкие и менее значительные изменения. Заморозка новых функций (feature freeze) — это первая крупная дата заморозки, наступающая после прохождения первой половины цикла разработки. На этом этапе новые функции должны в основном быть реализованы. В оставшуюся часть цикла предполагается сфокусироваться на исправлении ошибок. После этого «замораживается» пользовательский интерфейс, затем документация, ядро и т.п., после чего выпускается бета-версия (beta release), которая подвергается интенсивному тестированию. После того, как выпущена бета-версия, исправляются только критические ошибки, и выпускается release candidate, который, если он не содержит серьёзных ошибок, становится в дальнейшем финальным выпуском.



Тысячи пакетов исходного кода, миллиарды строк кода, сотни разработчиков требуют больше общения и планирования для поддержания высоких стандартов качества. В начале и в середине каждого цикла выпуска организуется Ubuntu Developer Summit, где разработчики и участники собираются вместе, чтобы планировать новые функции в следующих выпусках. Каждая функция обсуждается заинтересованными сторонами, и составляется спецификация, содержащая подробную информацию об исходных предположениях, реализации, внесении необходимых изменений в другие пакеты, о тестировании и так далее. Всё это делается прозрачным и открытым образом, так что вы можете поучаствовать удалённо, посмотреть видеотрансляцию, пообщаться с участниками и подписаться на спецификации, чтобы всегда быть в курсе происходящего.

Однако на совещании могут быть обсуждены не все изменения, так как Ubuntu зависит от изменений, вносимых в другие проекты. Поэтому участники разработки Ubuntu остаются постоянно на связи. Большинство команд или проектов используют отдельные списки рассылки, чтобы избежать большого потока не связанных с их специализацией писем. Для более непосредственной координации разработчики и добровольные участники используют Internet Relay Chat (IRC). Все обсуждения являются открытыми и публичными.

Ещё одним важным инструментом связи являются отчёты об ошибках. Если в пакете или части инфраструктуры обнаружена ошибка, отчёт о ней отправляется на Launchpad. В этом отчёте собрана вся информация, и при необходимости сведения о важности, статусе ошибки и лице, назначенном для её устранения, обновляются. Это делает отчёт об ошибке эффективным средством отслеживания ошибок в пакетах или проектах и оптимизации рабочей нагрузки.

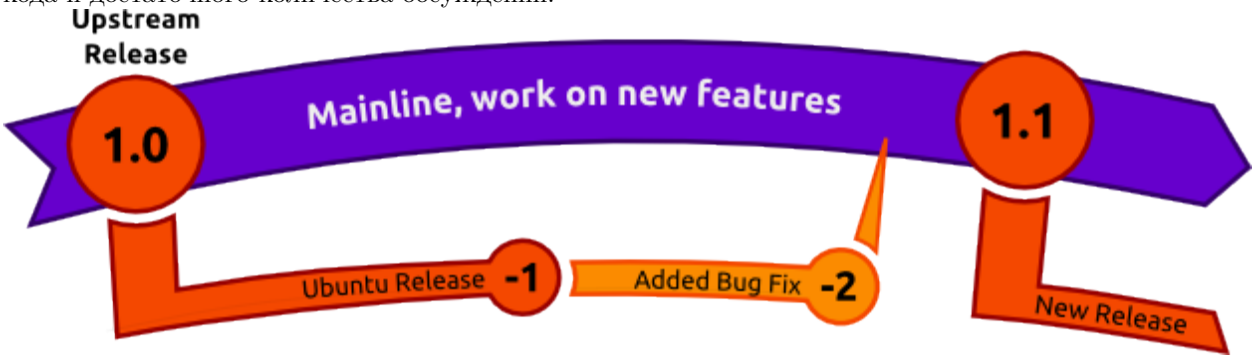
Большая часть доступных в Ubuntu программ создана не самими разработчиками Ubuntu. Многие из программ написаны разработчиками из других проектов с открытым исходным кодом, а затем интегрированы в Ubuntu. Такие проекты принято называть «апстримом» (от англ. upstream — вверх по течению), так как их исходный код вливается в Ubuntu, где мы «просто» интегрируем его в систему. Связь с апстримом очень важна для Ubuntu. Не только Ubuntu получает программный код из апстрима, но и апстрим получает от пользователей отчёты об ошибках и патчи от Ubuntu (и других дистрибутивов).

Наиболее важным апстримом для Ubuntu является Debian. Debian — это дистрибутив, на котором основана Ubuntu, и именно там созданы многие инженерные решения, касающиеся инфраструктуры пакетов. По традиции, в Debian для каждого отдельного пакета всегда имеется сопровождающий (мэйнтейнер) или отдельная группа сопровождения. В Ubuntu тоже есть команды, заинтересованные в работе над подмножеством пакетов и, разумеется, каждый разработчик имеет собственную область компетенции, но участие (и права загрузки изменений) обычно доступны любому, кто продемонстрирует способность и желание работать.

Внести изменение в Ubuntu новому участнику не так трудно, как кажется, и это может быть весьма полезным опытом. Это позволяет не только научиться чему-то новому и увлекательному, но и поделиться своим решением проблемы с миллионами других пользователей.

Разработка открытого программного обеспечения происходит в распределённом мире, в котором у

разработчиков могут быть различные цели и различные области интересов. Например, может быть так, что отдельный апстрим заинтересован в работе над крупным нововведением, в то время как Ubuntu, вследствие тесного расписания релизов, более заинтересована в выпуске стабильной версии, в которой лишь исправлены некоторые ошибки. Поэтому мы используем «Ubuntu Distributed Development», где работа над кодом ведётся в различных ветках, которые затем сливаются друг с другом после проверки кода и достаточного количества обсуждений.



В приведённом выше примере имеет смысл включить в Ubuntu существующую версию проекта, сделать исправления ошибок, добавить их в апстрим для следующего выпуска проекта и включить его (если он к этому пригоден) в следующий выпуск Ubuntu. Это будет наилучшим компромиссом и ситуацией, в которой выигрывают обе стороны.

Чтобы исправить ошибку в Ubuntu, нужно сначала получить исходный код пакета, поработать над его исправлением, снабдить свою работу документацией, чтобы другим разработчикам и пользователям было легко понять, что именно вы сделали, а затем собрать пакет и протестировать его. После того, как вы протестировали изменённый пакет, можно предложить включить его в текущий разрабатываемый релиз Ubuntu. Разработчик с правом загрузки проверит ваш пакет и затем интегрирует его в Ubuntu.

При попытке найти решение полезно проверить, известно ли о проблеме, над которой вы работаете, в апстриме и не найдено ли там уже её возможное решение. Если нет, сделайте всё возможное, чтобы решить проблему совместными усилиями.

Дополнительные шаги, которые вы можете предпринять, — это адаптирование вашего изменения для предыдущих поддерживаемых выпусков Ubuntu и отправление его в апстрим.

Наиболее важные требования для успешной разработки в Ubuntu: уметь «заставлять вещи снова работать», не бояться читать документацию и задавать вопросы, быть командным игроком и иметь некоторую склонность к работе детектива.

Подходящими местами для получения ответов на ваши вопросы являются `ubuntu-motu@lists.ubuntu.com` и `#ubuntu-motu` на `irc.freenode.net`. Там вы легко найдёте множество новых друзей и людей, разделяющих вашу страсть: делать мир лучше, улучшая открытое программное обеспечение.

1.2 Подготовка

Существует несколько вещей, которые нужно сделать перед началом разработки в Ubuntu. Эта статья поможет вам подготовить компьютер к работе с пакетами и отправке ваших пакетов на платформу

хостинга Ubuntu — Launchpad. Вот что здесь описано:

- Установка программ для работы с пакетами. Они включают в себя:
  - специфичные для Ubuntu утилиты создания пакетов
  - криптографическую программу, которая позволит другим удостовериться, что работа выполнена именно вами
  - дополнительные программы шифрования, обеспечивающие безопасную передачу файлов
- Создание и настройка учётной записи на Launchpad
- Настройка вашей среды разработки для облегчения локальной сборки пакетов, взаимодействия с другими разработчиками и отправки ваших изменений на Launchpad.

Примечание: Рекомендуется выполнять работу по созданию пакетов в текущей разрабатываемой версии Ubuntu. Это позволит вам тестировать изменения в той же среде, в которую они в действительности затем будут внесены.

Не беспокойтесь, вы можете воспользоваться Testdrive или *chroots* для безопасного использования разрабатываемой версии (релиза)

1.2.1 Установка базового программного обеспечения для создания пакетов

Существует множество инструментов, которые могут значительно упростить жизнь разработчику Ubuntu. Вы познакомитесь с ними далее в этом руководстве. Чтобы установить большинство инструментов, нужно выполнить следующую команду:

```
$ sudo apt-get install gnupg pbuilder ubuntu-dev-tools bzr-builddeb apt-file
```

Примечание: Начиная с Ubuntu 11.10 «Oneiric Ocelot» (или если включен репозиторий Backports в текущем поддерживаемом выпуске), следующая команда устанавливает всё вышеупомянутое и другие инструменты, часто используемые в разработке Ubuntu:

```
$ sudo apt-get install packaging-dev
```

Эта команда установит следующие программы:

- `gnupg` – GNU Privacy Guard содержит инструменты, которые понадобятся для создания криптографического ключа, с помощью которого вы будете подписывать файлы, которые хотите загрузить на Launchpad
- `pbuilder` – инструмент для создания готовых к дальнейшему распространению сборок пакетов в чистой и изолированной среде.
- `ubuntu-dev-tools` (и его непосредственная зависимость `devscripts`) – набор инструментов, упрощающих многие задачи по созданию пакетов.
- `bzr-builddeb` (и его зависимость – `bzr`) – управление распределёнными версиями с помощью Bazaar (новый способ работы с пакетами для Ubuntu), упрощающий совместную работу многих людей над одним и тем же кодом и позволяющий с лёгкостью объединять результаты их труда друг с другом.
- `apt-file` предоставляет простой способ найти двоичный пакет, содержащий заданный файл.

Создание ключа GPG

GPG — аббревиатура для GNU Privacy Guard, реализующего стандарт OpenPGP, который позволяет подписывать и шифровать сообщения и файлы. Это полезно в ряде ситуаций. В нашем случае важно, что вы можете использовать свой ключ для подписывания файлов, чтобы можно было удостовериться, что именно вы с ними работали. Если вы загрузите пакет исходного кода на Launchpad, он будет принят только в том случае, если можно точно определить, кто именно отправил пакет.

Чтобы сгенерировать новый ключ GPG, наберите:

```
$ gpg --gen-key
```

GPG сначала спросит, какой тип ключа вы хотите создать. Выбор по умолчанию (RSA и DSA) вполне подойдёт. Далее он попросит указать размер ключа. Размер по умолчанию (в настоящее время 2048) подойдёт, но 4096 надёжнее. Далее, программа спросит, хотите ли вы, чтобы срок действия ключа истёк через какое-то время. Безопаснее ответить "0", что означает, что срок действия не истечёт никогда. Последний вопрос будет о вашем имени и адресе электронной почты. Просто выберите адрес, которым вы пользуетесь при разработке Ubuntu, дополнительные адреса можно будет добавить позже. Добавлять комментарий не обязательно. После этого нужно указать надёжную парольную фразу (парольная фраза — это просто пароль, в котором допускается использовать пробелы).

Теперь GPG создаст для вас ключ, что может занять некоторое время. Для его создания понадобятся случайные байты, поэтому будет просто замечательно, если вы зададите своей системе какую-нибудь работу. Подвигайте указатель мыши, наберите несколько абзацев случайного текста, загрузите любую веб-страницу.

Когда процесс будет завершён, вы получите сообщение наподобие следующего:

```
pub   4096R/43CDE61D 2010-12-06
      Key fingerprint = 5C28 0144 FB08 91C0 2CF3  37AC 6F0B F90F 43CD E61D
uid                  Daniel Holbach <dh@mailempfang.de>
sub   4096R/51FBE68C 2010-12-06
```

В данном случае, `43CDE61D` — это идентификатор ключа (*key ID*).

Затем нужно загрузить открытую (public) часть вашего ключа на сервер ключей, чтобы все могли идентифицировать сообщения и файлы, как отправленные вами. Для этого введите:

```
$ gpg --send-keys --keyserver keyserver.ubuntu.com <KEY ID>
```

Эта команда отправит ваш ключ на сервер ключей Ubuntu, а сеть серверов ключей автоматически синхронизирует ключ между собой. После того, как эта синхронизация завершится, ваш подписанный открытый ключ будет готов для удостоверения сделанного вами вклада во всём мире.

Создание ключа SSH

SSH или *Secure Shell* — это протокол, позволяющий безопасно обмениваться данными по сети. Обычной практикой является использование SSH для доступа и открытия командной оболочки на другом компьютере и для безопасной пересылки файлов. В наших целях мы в основном будем использовать SSH для безопасной отправки пакетов исходного кода на Launchpad.

Чтобы сгенерировать ключ SSH, введите:

```
$ ssh-keygen -t rsa
```

Имя файла по умолчанию обычно вполне годится, так что можете оставить его как есть. Для целей безопасности настоятельно рекомендуется задать парольную фразу.

Настройка pbuilder

`pbuilder` позволяет локально собирать пакеты на вашем компьютере. Он служит нескольким целям:

- Сборка будет выполнена в минимальной и чистой среде. Это даст возможность убедиться, что сборку удастся успешно воспроизвести и на других компьютерах, но при этом поможет избежать изменений в вашей локальной системе
- Отпадает необходимость в локальной установке всех *сборочных зависимостей*
- Можно настроить несколько экземпляров для различных выпусков Ubuntu и Debian

Настроить `pbuilder` очень просто. Наберите

```
$ pbuilder-dist <release> create
```

где <release> — это, например *raring*, *saucy*, *trusty* или, в случае Debian, *sid*. Выполнение команды займёт некоторое время, поскольку будут загружены все пакеты, необходимые для "минимальной установки". Впрочем, при этом используется кэширование.

1.2.2 Подготовка к работе с Launchpad

После того, как базовая локальная конфигурация создана, следующим шагом будет настройка системы для работы с Launchpad. В этом разделе мы сфокусируемся на следующих вопросах:

- Что такое Launchpad и как создать учётную запись на Launchpad
- Загрузка ваших ключей GPG и SSH на Launchpad
- Настройка Bazaar для работы с Launchpad
- Настройка Bash для работы с Bazaar

Сведения о Launchpad

Launchpad является центральным элементом инфраструктуры, используемой нами в Ubuntu. Он хранит не только наши пакеты и наш код, но и такие вещи, как переводы, отчёты об ошибках, а также информацию о людях, работающих над Ubuntu и их принадлежности к различным командам. Вы можете также использовать Launchpad, чтобы опубликовать предлагаемые вами исправления и попросить других разработчиков Ubuntu проверить и поддержать их.

Вам понадобится зарегистрироваться на Launchpad и предоставить некоторое минимальное количество информации о себе. Это позволит вам скачивать или отправлять исходный код, отправлять отчёты об ошибках и т.п.

Кроме хостинга Ubuntu, Launchpad может предоставлять место для любого свободного программного проекта. Дополнительную информацию смотрите на Справочных вики-страницах Launchpad.

Создание учётной записи на Launchpad

Если у вас ещё нет учётной записи на Launchpad, вы легко можете создать её. Если учётная запись есть, но вы не помните свой Launchpad ID, его можно узнать, зайдя на https://launchpad.net/~ и взглянув на часть после ~ в URL.

При регистрации на Launchpad вас попросят выбрать отображаемое имя. Рекомендуется указать здесь ваше настоящее имя, чтобы ваши коллеги - разработчики Ubuntu могли лучше с вами познакомиться.

При регистрации новой учётной записи Launchpad отправит вам письмо со ссылкой, которую нужно открыть в веб-браузере, чтобы подтвердить указанный вами адрес электронной почты. Если вы не получили письмо, проверьте папку нежелательной почты (спама).

Справочная страница новой учётной записи на Launchpad содержит дополнительную информацию о процессе и дополнительных настройках, которые можно сделать.

Загрузка вашего ключа GPG на Launchpad

Сначала нужно получить отпечаток и идентификатор ключа.

Чтобы узнать свой отпечаток ключа GPG (fingerprint), наберите:

```
$ gpg --fingerprint email@address.com
```

и вы увидите что-то наподобие:

```
pub   4096R/43CDE61D 2010-12-06
      Key fingerprint = 5C28 0144 FB08 91C0 2CF3  37AC 6F0B F90F 43CD E61D
uid                  Daniel Holbach <dh@mailempfang.de>
sub   4096R/51FBE68C 2010-12-06
```

Затем выполните эту команду для отправки вашего ключа на сервер ключей Ubuntu:

```
$ gpg --keyserver keyserver.ubuntu.com --send-keys 43CDE61D
```

где `43CDE61D` следует заменить на ваш идентификатор ключа (он в первой строке вывода предыдущей команды). Теперь можно импортировать свой ключ на Launchpad.

Зайдите на https://launchpad.net/~/+editpgpkeys и скопируйте данные из строки «Key fingerprint» в текстовое поле. В приведённом выше примере это будет `5C28 0144 FB08 91C0 2CF3 37AC 6F0B F90F 43CD E61D`. Затем щёлкните «Import Key».

Launchpad воспользуется отпечатком ключа для проверки наличия вашего ключа на сервере ключей Ubuntu и, в случае успеха, отправит вам зашифрованное сообщение электронной почты, предлагающее подтвердить импорт ключа. Проверьте свою почту и прочтите письмо, полученное с Launchpad. *Если ваш клиент электронной почты поддерживает шифрование OpenPGP, он предложит ввести пароль, который вы выбрали при создании ключа GPG. Введите пароль, затем щёлкните на ссылке, чтобы подтвердить, что этот ключ принадлежит вам.*

Launchpad шифрует почту, используя ваш публичный ключ, так что вы сможете убедиться, что ключ ваш. Если вы пользуетесь почтовым клиентом Thunderbird, использующимся в Ubuntu по умолчанию, для дешифровки сообщения можете установить дополнение Enigmail. Если ваша почтовая программа не поддерживает шифрование OpenPGP, скопируйте зашифрованное содержимое письма в буфер обмена, наберите в терминале `gpg` и вставьте содержимое письма в окно терминала.

Вернувшись на сайт Launchpad, воспользуйтесь кнопкой «Confirm», чтобы Launchpad завершил импорт вашего ключа OpenPGP.

Дополнительную информацию можно найти на https://help.launchpad.net/YourAccount/ImportingYourPGPKey

Загрузка вашего ключа SSH на Launchpad

Откройте в веб-браузере https://launchpad.net/~/+editsshkeys, а в текстовом редакторе файл `~/.ssh/id_rsa.pub`. Это открытая часть вашего ключа SSH, поэтому можно без опасений предоставить к ней общий доступ на Launchpad. Скопируйте содержимое файла и вставьте его в текстовое поле на веб-странице с меткой «Add an SSH key». Затем щёлкните «Import Public Key».

Для дополнительной информации об этом процессе посетите страницу о создании ключевой пары SSH на Launchpad.

Настройка Bazaar

Bazaar — это инструмент, который мы используем для хранения изменений в коде логичным и предсказуемым способом, а также обмена предлагаемыми изменениями и их слияния, даже в том случае, если разработка ведётся параллельно несколькими людьми. Он используется в новом методе работы с пакетами Ubuntu — Ubuntu Distributed Development.

Что сообщить Bazaar о том, кто вы, просто наберите:

```
$ bzr whoami "Bob Dobbs <subgenius@example.com>"
$ bzr launchpad-login subgenius
```

whoami сообщит Bazaar, какое имя и адрес электронной почты он должен использовать для ваших коммитов. С помощью *launchpad-login* вы указываете свой Launchpad ID. Это код, который идентифицирует вашу учётную запись на Launchpad.

Примечание: если вы не помните свой идентификатор, перейдите на https://launchpad.net/~ и посмотрите, куда вас перенаправляет эта страница. Часть URL после символа «~» — это и есть ваш Launchpad ID.

Настройка командной оболочки

Как и Bazaar, инструментам создания пакетов Debian/Ubuntu понадобится некоторая информация о вас. Просто откройте *~/.bashrc* в текстовом редакторе и добавьте внизу что-то вроде этого:

```
export DEBFULLNAME="Bob Dobbs"
export DEBEMAIL="subgenius@example.com"
```

Затем сохраните файл и перезапустите терминал или наберите:

```
$ source ~/.bashrc
```

(Если вы не пользуетесь стандартной командной оболочкой *bash*, отредактируйте конфигурационный файл той оболочки, которую вы предпочитаете использовать.)

1.3 Распределенная разработка Ubuntu — введение

Это руководство фокусируется на работе с пакетами с использованием метода *Ubuntu Distributed Development* (UDD).

Ubuntu Distributed Development (UDD) — это новая технология разработки пакетов Ubuntu, использующая инструменты, процессы и последовательности действий, характерные для типичной схемы разработки программ, основанной на распределённой системе управления версиями (DVCS). DVCS, используемая в UDD — это Bazaar.

1.3.1 Ограничения традиционных методов создания пакетов

Традиционно пакеты Ubuntu хранятся в архивных tar-файлах. Традиционный пакет исходного кода состоит из tar-файла с исходным кодом из апстрима, “debian” tar-файла (или сжатого diff-файл

для более старых пакетов), содержащего набор входных файлов для создания пакета, и файла .dsc с метаданными. Чтобы посмотреть на традиционный пакет, выполните команду:

```
$ apt-get source kdetoys
```

Она загрузит исходники из апстрима `kdetoys_4.6.5.orig.tar.bz2`, набор входных файлов `kdetoys_4.6.5-0ubuntu1.debian.tar.gz` и метаданные `kdetoys_4.6.5-0ubuntu1~ppa1.dsc`. Если у вас установлен dpkg-dev, она извлечёт их содержимое и предоставит вам пакет исходного кода.

Традиционные методы создания пакетов отредактируют и загрузят эти файлы. Однако это дает ограниченные возможности для сотрудничества с другими разработчиками, изменения должны передаваться через файлы diff без центрального способ отслеживания них, плюс два разработчики не могут вносить изменения одновременно. Поэтому большинство команд разработчиков перешли от традиционного метода создания пакетов к системе контроля версий. Это позволяет нескольким разработчикам работать над пакетом сообща. Однако нет прямой связи между системы контроля версий и архивом пакетов, поэтому их необходимо будет вручную синхронизировать. Поскольку каждая команда работает в собственной системе контроля версий перспективный разработчик должен сначала разобраться где, что и как получить пакет, прежде чем они смогут работать с этим пакетом.

1.3.2 Распределенная разработка Ubuntu

С Ubuntu Distributed Development все пакеты в архиве Ubuntu (и Debian) автоматически импортируются в ветки Bazaar на нашем сайте хостинга кода Launchpad. Изменения можно вносить напрямую в эти ветки шагами увеличения любым пользователем, у которого есть доступ. Изменения также можно вносить в раздвоенные ветки и соединять из обратно при помощи Предложений и Слиянии (Merge Proposals), когда они станут достаточно большими для рассмотрения, либо если созданы кем-либо без прямого доступа.

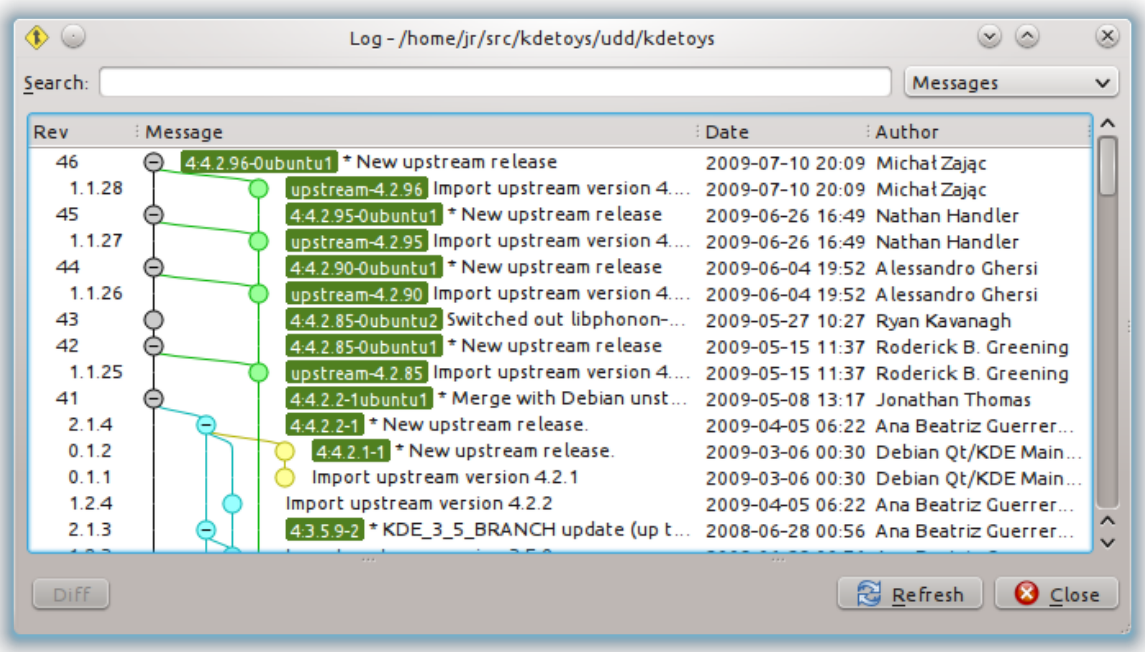
UDD ветки все находятся в стандартном местоположении, поэтому отладка будет легкой:

```
$ bzr branch ubuntu:kdetoys
```

История объединений включает две отдельные ветки, одну для источника апстрима и другую, которая добавит директорию пакета `debian/`:

```
$ cd kdetoys
$ bzr qlog
```

(Эта команда использует в качестве графического интерфейса *qbzr*. Для вывода в консоль, запустите `log` вместо `qlog`.)



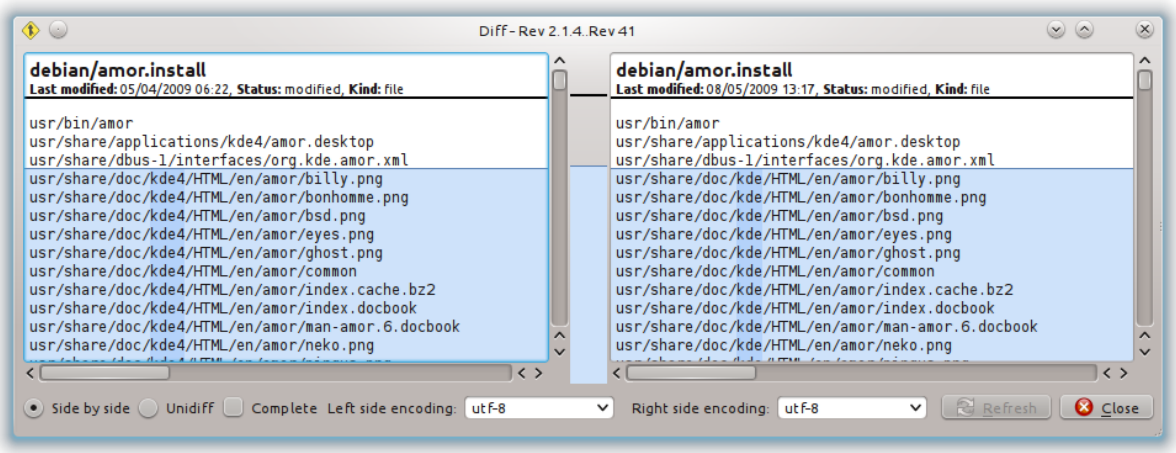
UDD ветка *kdetoys* помечает полный пакет для каждой версии, загруженной в Ubuntu серыми кругами и версии источника апстрима - зелеными. Версии помечаются либо версиями Ubuntu (например, `4:4.2.29-0ubuntu1`) или для веток апстрима - версией апстрима (`upstream-4.2.96`).

Многие пакеты Ubuntu основаны на пакетах в Debian, UDD также импортирует и пакет Debian в наши ветки. В ветке *kdetoys* выше версии Debian из *unstable* после объединения помечены синими кружочками, в то время как из *Debian experimental* после объединения помечены желтыми. Релизы Debian помечены номерами своих версий, например, `4:4.2.2-1`.

Таким образом из UDD-ветки вы можете увидеть полную историю изменений пакета и сравнить любые две версии. Например, чтобы увидеть различия между версией 4.2.2 в Debian и 4.2.2 в Ubuntu, используйте:

```
$ bzr qdiff -r tag:4:4.2.2-1..tag:4:4.2.2-1ubuntu1
```

(Эта команда использует графический интерфейс *qbzr*. Запустите `diff` вместо `qdiff` для вывода в консоль.)



Здесь мы можем ясно увидеть, что было изменено в Ubuntu по сравнению с Debian-версией. Очень удобно.

1.3.3 Bazaar

Ветки UDD используют Bazaar — распределённую систему управления версиями, которая проста в использовании для тех, кто знаком с такими популярными системами, как Subversion, и в то же время предоставляет всю мощь Git.

Чтобы сделать пакет с UDD Вам нужно знать основы использования Bazaar для управления файлами. Для получения базовых навыков работы с Bazaar смотрите Пятиминутное обучение Bazaar и Руководство по использованию Bazaar.

1.3.4 Ограничения UDD

Ubuntu Distributed Development — новый метод работы с пакетами Ubuntu. В настоящее время он имеет некоторые существенных ограничения:

- Создание полной ветки с историей может отнять много времени и сетевых ресурсов. Возможно Вам быстрее будет сделать легкую отладку `bzr checkout --lightweight ubuntu:kdetoys`, но тогда потребуется сетевой доступ для любых дальнейших операций bzr.
- Работа над патчами очень кропотливая. Патчи можно рассматривать как разветвленную переработку системы управления, таки образом мы получаем RCS поверх RCS.
- Нет способа создавать билды напрямую из веток. Нужно создавать исходный пакет и загружать его.
- Некоторые пакеты не были успешно импортированы в ветки UDD. Последние версии Bazaar автоматически будут уведомлять вас в случае возникновения подобной ситуации. Перед началом работы с веткой Вы можете вручную поставить отметку status of the package importer.

Над всем вышеперечисленным сейчас ведется работа и ожидается, что UDD вскоре станет основным способом работы над пакетами Ubuntu. Тем не менее, сейчас большинство команд Ubuntu еще не работали с ветками UDD. Но так как UDD ветки являются тем же самым, как и пакеты в архиве, любой команде не составит труда с ними работать.

1.4 Исправление ошибок в Ubuntu

1.4.1 Вступление

Если вы следовали инструкциям по *подготовке к разработке Ubuntu*, всё должно быть уже готово к работе.

Как вы можете видеть на картинке выше, в процессе исправления ошибок в Ubuntu нет никаких сюрпризов: вы находите проблему, получаете код, исправляете его, тестируете, отправляете на Launchpad и просите, чтобы его проверили и объединили с основным кодом. В этом руководстве мы пройдем через все необходимые шаги, один за другим.

1.4.2 Поиск проблемы

Существуют различные способы найти, над чем можно поработать. Это может быть ошибка, которую вы обнаружили сами (что даёт вам отличную возможность проверить своё исправление) или проблема, которую вы заметили где-то ещё, например в отчёте об ошибке.

Harvest хранит различные списки TODO, касающиеся разработки Ubuntu. Это списки ошибок, которые были уже исправлены в апстриме или в Debian, списки небольших ошибок (мы называем их «bitesize») и так далее. Просмотрите их и найдите ошибку, исправлением которой вы хотите заняться.

1.4.3 Выяснение того, что нужно исправить

Если вы не знаете, в каком пакете исходного кода содержится ошибка, но знаете путь к этому приложению в вашей системе, то вы сможете найти пакет исходного кода, над которым требуется поработать.

Предположим, вы обнаружили ошибку в Tomboy, приложении для создания заметок на рабочем столе. Приложение Tomboy можно запустить, выполнив `/usr/bin/tomboy` в командной строке. Чтобы найти двоичный пакет, содержащий это приложение, используйте следующую команду:

```
$ apt-file find /usr/bin/tomboy
```

Команда выведет следующую информацию:

```
tomboy: /usr/bin/tomboy
```

Обратите внимание на то, что часть, предшествующая двоеточию, является именем двоичного пакета. Часто бывает так, что у пакета исходного кода и двоичного пакета различные имена. Чаще всего это происходит, когда один пакет исходного кода используется, чтобы создать несколько различных двоичных пакетов. Чтобы найти исходный пакет для определенного двоичного пакета, введите:


```
$ apt-cache showsrc tomboy | grep ^Package:
Package: tomboy
$ apt-cache showsrc python-vigra | grep ^Package:
Package: libvigraimpex
```

Команда `apt-cache` установлена в Ubuntu по умолчанию.

1.4.4 Получение кода

Когда вы знаете, над каким пакетом исходного кода работать, вы можете загрузить копию этого пакета, и заняться отладкой. При распределённой разработке Ubuntu это делается при помощи *клонирования bzr-репозитория* этого пакета. Launchpad поддерживает Bazaar-ветки для всех пакетов в Ubuntu.

Как только вы получили локальную копию исходного кода, вы можете исследовать ошибку, исправить её, и отправить своё исправление на Launchpad, в виде Bazaar-ветки. Как только вы станете достаточно уверены в своём исправлении, вы можете отправить *заявку на слияние*, то есть попросить других разработчиков просмотреть и одобрить ваше изменение. В случае согласия с вашими изменениями они загрузят ваши изменения в репозиторий. От вашего изменения получать пользу все — в том числе и вы, чьё имя будет стоять в списке изменений. Вы теперь на пути становления разработчиком Ubuntu!

Мы опишем специфику загрузки кода, отправки изменений, и создания заявки на просмотр в следующимх разделах.

1.4.5 Исправление ошибки

Есть целые книги о нахождении ошибок, их исправлении, тестировании и так далее. Если вы новичок в программировании, попробуйте исправлять лёгкие ошибки, такие как опечатки. Пытайтесь делать ваши изменения минимальными и чётко документировать ваши изменения и предположения.

Перед тем, как работать над ошибкой, убедитесь, что она не исправлена уже кем-то другим, и никто не занимается в данный момент её исправлением. Не помешает проверить следующие источники:

- Система отслеживания ошибок апстрима (и Debian) — открытые и закрытые ошибки,
- История версий в апстриме (или в новой версии) может содержать сведения об исправлении ошибки,
- отчёты об ошибках и новые версии пакетов в Debian и других дистрибутивах.

Теперь можно создать патч, содержащий исправление ошибки. Команда `edit-patch` — самый простой способ добавить патч к пакету. Выполните:

```
$ edit-patch 99-new-patch
```

Эта команда скопирует файлы, необходимые для сборки пакета, во временную директорию. Вы можете изменять эти файлы в текстовом редакторе или применять патчи из upstream, например:

```
$ patch -p1 < ../bugfix.patch
```

После редактирования файла наберите `exit` или нажмите `control-d`, чтобы выйти из временной командной оболочки. Новый патч будет добавлен в `debian/patches`.

1.4.6 Тестирование исправления

Чтобы собрать тестовый пакет с вашими изменениями, выполните следующие команды:


```
$ bzr builddeb -- -S -us -uc
$ pbuilder-dist <release> build ../<package>_<version>.dsc
```

Это создаст пакет исходного кода из содержимого ветки (`-us -uc` просто позволяет пропустить этап подписывания пакета исходного кода), а `pbuilder-dist` соберёт пакет из исходного кода для любого выбранного вами `релиза`.

После успешного завершения сборки установите пакет из `~/pbuilder/<release>_result/` (с помощью `sudo dpkg -i <пакет>_<версия>.deb`). Затем проверьте, удалось ли устранить ошибку.

Документирование исправления

Очень важно документировать свои изменения в достаточной степени, чтобы разработчикам впоследствии не пришлось гадать, какими были основания и предпосылки сделанных вами изменений. Каждый пакет исходного кода Debian и Ubuntu включает в себя файл `debian/changelog`, в котором отслеживаются вносимые в пакет изменения.

Самый простой способ обновить его — это выполнить:

```
$ dch -i
```

Эта команда добавит в файл шаблон записи и запустит редактор, в котором вы сможете добавить недостающую информацию. Вот пример этой записи:

```
specialpackage (1.2-3ubuntu4) trusty; urgency=low

  * debian/control: updated description to include frobnicator (LP: #123456)

 -- Emma Adams <emma.adams@isp.com>  Sat, 17 Jul 2010 02:53:39 +0200
```

Команда `dch` должна заполнить первую и последнюю строку этой записи в файле changelog. Первая строка содержит имя пакета исходного кода, номер его версии, в какой релиз Ubuntu он загружен, срочность (почти всегда низкая — 'low'). Последняя строка всегда содержит имя, адрес электронной почты и метку времени изменения (в формате **RFC 5322**).

Теперь давайте сфокусируемся на том, что должно содержаться в самой записи файла changelog. Очень важно задокументировать:

1. где сделано изменение
2. что было изменено
3. где происходило обсуждение этого изменения

В нашем (довольно редком) примере последнему пункту соответствует `(LP: #123456)`, то есть сслыка на ошибку на Launchpad с номером 123456. Отчёты об ошибках, темы списков рассылки или спецификации обычно являются хорошими источниками информации для обоснования изменений. В качестве дополнительного приза, если вы используете нотацию `LP: #<номер>` для ошибок на Launchpad, то ошибка автоматически получит статус закрытой при отправке пакета в Ubuntu.

Закрепление изменения

Написав и сохранив запись в changelog, мы можем просто запустить:

```
debcommit
```

и изменение будет залито (локально) с вашей записью changelog в качестве сообщения коммита.

Для Launchpad-репозитория, в который отправлять ваши изменения, используйте следующее имя:


```
lp:~<yourlpid>/ubuntu/<release>/<package>/<branchname>
```

Это может быть, например:

```
lp:~emmaadams/ubuntu/trusty/specialpackage/fix-for-123456
```

Так что, если вы просто выполните:

```
bzr push lp:~emmaadams/ubuntu/trusty/specialpackage/fix-for-123456
bzr lp-propose
```

всё должно быть готово. Команда push выполнит отправку на Launchpad, а вторая команда откроет страницу удалённой ветки на Launchpad в вашем браузере. Найдите там ссылку «(+) Propose for merging» и щёлкните на ней, чтобы кто-нибудь проверил изменение и включил его в Ubuntu.

Наша статья о *поиске поручительства* предоставляет дополнительные подробности о получении обратной связи для предложенных вами изменений.

Если ваша ветка исправляет ошибку в стабильном выпуске или это исправление безопасности, прочтите нашу статью *Обновления безопасности и стабильных выпусков*.

1.5 Демонстрация: исправление ошибки в Ubuntu

Хотя техника *исправления ошибки* одинакова для любых ошибок, каждая ошибка всё же в чём то отличается от других. Данный пример исправления конкретной ошибки может помочь вам понять, что обычно следует принять во внимание.

Примечание: Во время написания данной статьи эта ошибка ещё не была исправлена. Возможно, у тому времени, когда вы будете читать статью, ошибку уже исправят. Считайте это просто примером и постарайте адаптировать его к конкретной проблеме, с которой вы столкнётесь.

1.5.1 Подтверждение проблемы

Предположим, в описании пакета `bumprace` отсутствует информация о его домашней странице. В качестве первого шага, следует проверить, не исправлена ли уже эта ошибка. Сделать это просто: посмотрите в Центре приложений или запустите:

```
apt-cache show bumprace
```

Вывод команды должен быть примерно следующим:

```
Package: bumprace
Priority: optional
Section: universe/games
Installed-Size: 136
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Original-Maintainer: Christian T. Steigies <cts@debian.org>
Architecture: amd64
Version: 1.5.4-1
Depends: bumprace-data, libc6 (>= 2.4), libsdl-image1.2 (>= 1.2.10),
         libsdl-mixer1.2, libsdl1.2debian (>= 1.2.10-1)
Filename: pool/universe/b/bumprace/bumprace_1.5.4-1_amd64.deb
Size: 38122
MD5sum: 48c943863b4207930d4a2228cedc4a5b
```



```
SHA1: 73bad0892be471bbc471c7a99d0b72f0d0a4babc
SHA256: 64ef9a45b75651f57dc76aff5b05dd7069db0c942b479c8ab09494e762ae69fc
Description-en: 1 or 2 players race through a multi-level maze
 In BumpRacer, 1 player or 2 players (team or competitive) choose among 4
 vehicles and race through a multi-level maze. The players must acquire
 bonuses and avoid traps and enemy fire in a race against the clock.
 For more info, see the homepage at http://www.linux-games.com/bumprace/
Description-md5: 3225199d614fba85ba2bc66d5578ff15
Bugs: https://bugs.launchpad.net/ubuntu/+filebug
Origin: Ubuntu
```

В качестве контрпримера можно привести `gedit`, где домашняя страница указана:

```
$ apt-cache show gedit | grep ^Homepage
Homepage: http://www.gnome.org/projects/gedit/
$
```

В некоторых случаях вы можете столкнуться с тем, что проблема, решение которой вы ищете, уже кем-то устранена. Чтобы избежать напрасной траты времени и труда, имеет смысл проделать кое-какую детективную работу.

1.5.2 Изучение ситуации с ошибкой

Сначала нужно проверить, не существует ли уже сообщения об этой ошибке в Ubuntu. Возможно, кто-то уже работает над её исправлением, или мы можем как-то внести свой вклад в решение этой проблемы. Для Ubuntu мы взглянем на https://bugs.launchpad.net/ubuntu/+source/bumprace и увидим, что открытого отчёта о нашей ошибке там нет.

Примечание: Для Ubuntu URL `https://bugs.launchpad.net/ubuntu/+source/<пакет>` всегда приводит на страницу ошибок в указанном пакете исходного кода.

В Debian, который является основным источником пакетов Ubuntu мы взглянем на http://bugs.debian.org/src:bumprace и также не найдём там сообщения о нашей ошибке.

Примечание: Для Debian URL `http://bugs.debian.org/src:<пакет>` всегда приводит на страницу ошибок в указанном пакете исходного кода.

Ошибка, над которой мы работаем, необычна в том смысле, что она касается только пакетирования `bumprace`. Если бы это была ошибка в исходном коде, полезно было бы также проверить систему отслеживания ошибок апстрима. К сожалению, эта процедура часто различается для каждого отдельного пакета, но всегда можно воспользоваться поиском в интернете, и в большинстве случаев вы обнаружите, что она окажется не такой уж сложной.

1.5.3 Предложение помощи

Если вы обнаружили открытую ошибку, которая ещё никому не назначена, и вы готовы взяться за её устранение, следует написать комментарий с вашим решением. Включите в него как можно больше информации: При каких обстоятельствах появляется ошибка? Как вы её исправили? Тестировали ли вы свой способ устранения ошибки?

Если сообщение об ошибке не было зарегистрировано, вы можете его создать. Подумайте над двумя вещами: Может быть, изменение настолько мало, что достаточно просто попросить кого-нибудь применить его? Может быть, у вас получилось только частично исправить ошибку, и вы хотите поделиться вашей частью?

Будет замечательно, если вы можете предложить свою помощь, и она, несомненно, будет с готовностью принята.

1.5.4 Исправление ошибки

В данном конкретном примере недостаточно просто найти веб-сайт `bumprace` и определить адрес его домашней страницы. Нужно убедиться, что сайт работает, и он не является просто каталогом различных программ. http://www.linux-games.com/bumprace/ выглядит подходящим местом.

Чтобы взяться за ошибку в пакете исходного кода, нам понадобится этот исходный код, и мы легко можем получить его, набрав:

```
bzr branch ubuntu:bumprace
```

Если вы прочитали *обзор каталога Debian*, то помните, вероятно, что домашняя страница указывается в первой части `debian/control`, в секции, начинающейся с `Source:`.

Так что теперь мы должны выполнить команду:

```
cd bumprace
```

и отредактировать `debian/control`, добавив `Homepage: http://www.linux-games.com/bumprace/`. В конце первой секции должно быть подходящее место для этого. После внесения изменений сохраните файл.

Если теперь вы выполните:

```
bzr diff
```

вы должны увидеть что-то вроде этого:

```
=== modified file 'debian/control'
--- debian/control      2012-05-14 23:38:14 +0000
+++ debian/control      2012-09-03 15:45:30 +0000
@@ -12,6 +12,7 @@
                libtool,
                zlib1g-dev
 Standards-Version: 3.9.3
+Homepage: http://www.linux-games.com/bumprace/

 Package: bumprace
 Architecture: any
```

Синтаксис diff очень прост для понимания. `+` указывает строку, которая была добавлена. В нашем случае она была добавлена непосредственно перед второй секцией, начинающейся с `Package`, которая указывает на готовый двоичный пакет.

1.5.5 Документирование исправления

Важно пояснить своим коллегам - разработчикам, что именно вы сделали. Если вы наберёте:

```
dch -i
```

это запустит редактор, с шаблоном записи в changelog, которую вам остаётся лишь дозаполнить. В нашем случае должно подойти что-то наподобие `debian/control: Added project's homepage.` Затем сохраните файл. Чтобы ещё раз проверить, что всё работает, наберите:


```
bzr diff debian/changelog
```

и вы увидите что-то вроде этого:

```
=== modified file 'debian/changelog'
--- debian/changelog    2012-05-14 23:38:14 +0000
+++ debian/changelog    2012-09-03 15:53:52 +0000
@@ -1,3 +1,9 @@
+bumprace (1.5.4-1ubuntu1) UNRELEASED; urgency=low
+
+  * debian/control: Added project's homepage.
+
+ -- Peggy Sue <peggy.sue@example.com>  Mon, 03 Sep 2012 17:53:12 +0200
+
 bumprace (1.5.4-1) unstable; urgency=low

   * new upstream version, sound and music have been removed (closes: #613344)
```

Несколько дополнительных соображений:

- Если у вас есть ссылка на ошибку на Launchpad, которую исправляет ваше изменение, добавьте (`LP: #<номер ошибки>`) в запись changelog, например: `(LP: #123456)`.
- Если вы хотите, чтобы ваше исправление было включено в Debian, синтаксис для ошибки в Debian будет `(Closes: #<номер ошибки>)`, например: `(Closes: #123456)`.
- Если это ссылка на сообщение об ошибке в апстриме или в Debian, или на обсуждение в почтовой рассылке, также укажите её.
- Старайтесь делать перенос строк после 80 символов.
- Старайтесь излагать подробно: не стоит писать целое эссе, но укажите достаточно информации, чтобы понять мог любой человек (даже если он не вникал глубоко в эту проблему).
- Укажите, как и где вы исправили ошибку.

1.5.6 Тестирование исправления

Чтобы проверить исправление, вам понадобится *настроить свою среду разработки*, затем собрать пакет, установить его и проверить, что проблема устранена. В нашем случае это будут следующие действия:

```
bzr bd -- -S
pbuilder-dist <current Ubuntu release> build ../bumprace_*.dsc
dpkg -I ~/pbuilder/*_result/bumprace_*.deb
```

Первой командой мы создаём пакет исходного кода из ветки, затем собираем его с помощью `pbuilder`, после чего проверяем, правильно ли добавлено поле домашней страницы в получившемся пакете.

Примечание: В большинстве случаев вам придётся действительно установить пакет, чтобы проверить правильность его работы. Наш случай намного проще. Если сборка завершилась успешно, готовые двоичные пакеты можно будет найти в `~/pbuilder/<выпуск>_result`. Установите их с помощью `sudo dpkg -i <пакет>.deb` или двойного щелчка на них в файловом менеджере.

После того, как мы убедились, что проблема решена, следующим шагом будет поделиться своим решением со всем миром.

1.5.7 Применение исправления

It makes sense to get the fix included as Upstream as possible. Doing that you can guarantee that everybody can take the Upstream source as-is and don't need to have local modifications to fix it.

В нашем случае мы установили, что ошибка относится только к пакетам Ubuntu и Debian. Так как Ubuntu основана на Debian, мы должны отправить исправление в Debian. После того, как в Debian появится исправленный пакет, он попадёт также и в Ubuntu. Наша ошибка не критична, поэтому можно так сделать. Если же важно применить исправление как можно скорее, то необходимо отправить исправление в несколько баг-трекеров (если оно затрагивает соотвествующие проекты).

Чтобы отправить патч в Debian, просто наберите:

```
submittodebian
```

Эта команда проведёт вас через несколько шагов, необходимых для оформления отчёта об ошибке и отправки его в правильное место. Обязательно просмотрите ваши изменения, чтобы убедиться, что там нет ничего постороннего.

Коммуникация имеет очень большое значение, поэтому при добавлении описания предоставьте хорошее и дружественное объяснение.

Если всё прошло нормально, то вы должны получить почтовое сообщение от системы отслеживания ошибок Debian с дополнительной информацией. Иногда это может занять несколько минут.

Примечание: Если проблема наблюдается только в Ubuntu, вы можете воспользоваться *инструкцией по поиску спонсора*.

1.5.8 Дополнительные замечания

Если вы можете внести в пакет несколько тривиальных исправлений сразу, сделайте это. Это позволит разработчикам быстрее рассмотреть и применить эти изменения.

Если вы хотите внести несколько больших изменений, лучше посылать патчи или запросы на слияние отдельно для каждого изменения. Это проще, если уже созданы индивидуальные сообщения об ошибках.

1.6 Создание пакетов для новых программ

Хотя в архиве Ubuntu имеются тысячи пакетов, есть ещё много программ, которыми никто не занимается. Если вы знаете о какой-то замечательной программе, о которой, по вашему мнению, стоит узнать более широкому кругу пользователей, вы можете попробовать приложить свою руку к созданию пакета для Ubuntu или PPA. Это руководство проведёт вас через этапы создания пакета для новой программы.

Сначала вам следует прочитать статью *Подготовка*, чтобы подготовить свою среду разработки.

1.6.1 Проверка программы

Первым этапом создания пакета является получение tar-файла из апстрима («апстримом» мы называем авторов приложений) и проверка того, что он нормально компилируется и запускается.

Это руководство проведёт вас через процесс создания пакета для небольшого приложения GNU Hello, доступного на GNU.org.

Если у вас ещё нет инструментов сборки, установите их. Также установите все необходимые зависимости.

Установим инструменты сборки:

```
$ sudo apt-get install build-essential
```

Скачаем основной пакет:

```
$ wget -O hello-2.7.tar.gz "http://ftp.gnu.org/gnu/hello/hello-2.7.tar.gz"
```

Теперь распакуем основной пакет:

```
$ tar xf hello-2.7.tar.gz
$ cd hello-2.7
```

Это приложение использует систему сборки autoconf, так что нам нужно запустить `./configure` для подготовки к компиляции.

При этом будет проверено наличие необходимых для сборки зависимостей. Поскольку `hello` — простой пример, `build-essential` обеспечит нас всем, что нужно. Для более сложных программ, команда завершится ошибкой, если у нас нет необходимых библиотек и файлов для разработки. Установите требуемые пакеты и повторите процесс, пока команда не завершится успешно.:

```
$ ./configure
```

Теперь нужно скомпилировать исходный код:

```
$ make
```

Если компиляция завершилась успешно, можно установить и запустить программу:

```
$ sudo make install
$ hello
```

1.6.2 Создание пакета

`bzr-builddeb` содержит модуль для создания нового пакета из шаблона. Этот модуль является обёрткой вокруг команды `dh_make` . Он уже должен быть у вас, если вы установили `packaging-dev`. Запустите команду, указав имя пакета, номер версии и путь к tar-архиву из апстрима:

```
$ sudo apt-get install dh-make bzr-builddeb
$ cd ..
$ bzr dh-make hello 2.7 hello-2.7.tar.gz
```

Когда он спросит тип пакета, выберите `s`: одиночный бинарник. Это импортирует код в ветку и создаст папку `debian/`. Взгляните на её содержимое: большинство автоматически созданных файлов требуются лишь для специализированных пакетов (например, модули Emacs), так что можно начать с удаления лишних файлов:

```
$ cd hello/debian
$ rm *ex *EX
```

Теперь нужно внести изменения в каждый из файлов.

В `debian/changelog` измените номер версии на версию Ubuntu: `2.7-0ubuntu1` (апстрим-версия 2.7, версия в Debian 0, версия в Ubuntu 1). Также смените `unstable` на текущий разрабатываемый выпуск Ubuntu, например, `trusty`.

Большая часть процесса компиляции пакета совершается скриптами из `debhelper`. Так как поведение `debhelper` меняется при выходе старшей версии, файл compat сообщает `debhelper`'у какую именно версию использовать. Имеет смысл использовать наиболее свежую версию: `9`.

Файл `control` содержит все метаданные пакета. Первый абзац описывает пакет исходных кодов. Второй и следующие абзацы описывают двоичные пакеты, которые должны быть собраны. Нам понадобится добавить пакеты, необходимые для компиляции приложения в `Build-Depends:`. Для `hello` он должен включать, как минимум:

```
Build-Depends: debhelper (>= 9)
```

Также нужно заполнить описание программы в поле `Description:`.

`copyright` нужно заполнить в соответствии с лицензией на источник из апстрима. Согласно файлу hello/COPYING, это лицензия GNU GPL 3 или более поздняя её версия.

`docs` должен содержать любые файлы документации из апстрима, которые, по вашему мнению, должны быть включены в готовый пакет.

`README.source` и `README.Debian` необходимы, лишь если ваш пакет имеет какие-то нестандартные функции. У нас таких нет, так что можно удалить эти файлы.

`source/format` можно оставить как есть, он описывает формат версии пакета исходного кода, который должен быть `3.0 (quilt)`.

`rules` — самый сложный файл. Это Makefile, который компилирует код и превращает его в двоичный пакет. К счастью, основную часть работы в наши дни автоматически выполняет `debhelper 7`, так что универсальная цель `%` просто запускает сценарий `dh`, который делает всё, что необходимо.

Все эти файлы подробнее описаны в статье *обзор каталога debian*.

Наконец, закоммитьте код в ветку для пакетов:

```
$ bzr add debian/source/format
$ bzr commit -m "Initial commit of Debian packaging."
```

1.6.3 Сборка пакета

Теперь нам нужно проверить, что наши исходные файлы для пакета успешно компилируются и собираются в двоичный .deb-пакет:

```
$ bzr builddeb -- -us -uc
$ cd ../../
```

`bzr builddeb` — это команда для сборки пакета в его текущем местоположении. `-us -uc` сообщает что подписывать пакет с помощью GPG не требуется. Результат будет помещён в каталог «..».

Просмотреть содержимое пакета можно с помощью:

```
$ lesspipe hello_2.7-0ubuntu1_amd64.deb
```

Установите пакет и проверьте, что он работает (позднее при желании вы сможете удалить его командой `sudo apt-get remove hello`):

```
$ sudo dpkg --install hello_2.7-0ubuntu1_amd64.deb
```

Можете также установить все пакеты сразу с помощью:

```
$ sudo debi
```


1.6.4 Дальнейшие шаги

Даже если двоичный .deb-пакет успешно собирается, ваши исходные файлы для пакета могут содержать ошибки. Многие ошибки могут автоматически обнаруживаться нашим инструментом `lintian`, который можно применить к файлу метаданных .dsc, двоичным пакетам .deb или файлу .changes:

```
$ lintian hello_2.7-0ubuntu1.dsc
$ lintian hello_2.7-0ubuntu1_amd64.deb
```

Чтобы увидеть подробное описание ошибок, используйте флаг lintian `--info` или команду `lintian-info`.

Результаты проверок архива Ubuntu можно найти в Интернете на http://lintian.ubuntuwire.org.

Для пакетов Python имеется также инструмент `lintian4python`, осуществляющий некоторые дополнительные проверки lintian.

После создания исправления для файлов пакета можно пересобрать его с параметром `-nc` (“no clean”), чтобы не начинать сборку с самого начала:

```
$ bzr builddeb -- -nc -us -uc
```

Убедившись, что пакет успешно собирается локально, вы должны проверить, правильно ли его сборка будет проходить в чистой системе, с помощью `pbuilder`. Поскольку вскоре мы собираемся загрузить его в PPA (персональный архив пакетов), эту загрузку нужно снабдить *цифровой подписью*, чтобы Launchpad мог удостовериться, что загрузку сделали вы (узнать о том, что загрузка будет подписана, можно по отсутствию передаваемых `bzr builddeb` флагов `-us` и `-uc`, которые мы использовали ранее). Для подписывания своей работы вам понадобится настроить GPG. Если вы ещё не настроили `pbuilder-dist` или GPG, *сделайте это сейчас*:

```
$ bzr builddeb -S
$ cd ../build-area
$ pbuilder-dist trusty build hello_2.7-0ubuntu1.dsc
```

После того как вы останетесь довольны получившимся пакетом, нужно, чтобы его проверили другие люди. Вы можете выгрузить ветку на Launchpad для проверки:

```
$ bzr push lp:~<lp-username>/+junk/hello-package
```

Выгрузка в PPA позволит удостовериться, что пакет собирается нормально, а также позволит вам и остальным тестировать бинарные пакеты. Вам потребуется создать PPA на Launchpad, после чего выгрузить пакет с помощью `dput`:

```
$ dput ppa:<lp-username>/<ppa-name> hello_2.7-0ubuntu1.changes
```

Смотрите раздел «*Загрузка*» для дополнительной информации.

Попросить провести review можно на канале IRC `#ubuntu-motu`, или в списке рассылки MOTU. В некоторых случаях может потребоваться участие конкретной команды: в подобных случаях команда GNU поможет разобраться.

1.6.5 Отправка на включение

Существует несколько путей, которыми пакет может попасть в Ubuntu. В большинстве случаев лучшим путём может быть прохождение сначала через Debian. Это позволит вашему пакету стать доступным для наибольшего количества пользователей, так как он будет доступен не только в Debian и Ubuntu, но и во всех дистрибутивах, созданных на их основе. Вот несколько полезных ссылок по отправке новых пакетов в Debian:

- Debian Mentors FAQ - debian-mentors создан для менторства новых и перспективных разработчиков Debian. Это то место, где можно найти спонсора для загрузки Вашего пакета в архив.
- Work-Needing and Prospective Packages - информация о том как отправлять баги "Intent to Package" (Назначение пакету) и "Request for Package" (Запрос пакета), а также списки открытых ITP и RFP.
- Руководство Разработчика Debian, 5.1. Создание пакетов - бесценный документ для создателей пакетов как под Ubuntu, так и под Debian. Данная глава описывает процесс отправки новых пакетов.

В некоторых случаях имеет смысл отправляться прямо в Ubuntu. Например, если Debian находится в состоянии "freeze": тогда ваш пакет врядли успеет войти в Ubuntu к ближайшему релизу. Этот процесс описан на странице "Новые Пакеты" Ubuntu Wiki.

1.6.6 Снимки экрана

Загрузив пакет в Debian, вам следует добавить снимки экрана, чтобы будущие пользователи смогли получить представление о том, как выглядит интерфейс программы. Снимки нужно отправлять на http://screenshots.debian.net/upload .

1.7 Обновления безопасности и обновления стабильных релизов

1.7.1 Исправление ошибок безопасности в Ubuntu

Вступление

Исправление дыр в безопасности в Ubuntu фактически не отличается от *исправления обычного бага*, и предполагается, что Вы знакомы с исправлением обычных багов. Для демонстрации отличий мы будем добавлять в пакет dbus в Ubuntu 12.04 LTS (Precise Pangolin) обновления для системы безопасности.

Получение исходного кода

В данном примере мы уже знаем, что хотим исправить пакет dbus в Ubuntu 12.04 LTS (Precise Pangolin). Поэтому сначала нужно определить версию пакета, который хотите скачать. Мы можем использовать `rmadison` в качестве помощи в данной ситуации.

```
$ rmadison dbus | grep precise
dbus | 1.4.18-1ubuntu1   | precise          | source, amd64, armel, armhf, i386, powerpc
dbus | 1.4.18-1ubuntu1.4 | precise-security | source, amd64, armel, armhf, i386, powerpc
dbus | 1.4.18-1ubuntu1.4 | precise-updates  | source, amd64, armel, armhf, i386, powerpc
```

Обычно выбирают самую последнюю версию для релиза, который Вы хотите пропатчить, который не в -proposed или -backports. Так как мы обновляем dbus Precise, Вы скачаете 1.4.18-1ubuntu1.4 с precise-updates:

```
$ bzr branch ubuntu:precise-updates/dbus
```

Создание патча

Теперь, когда мы имеем исходный пакет, мы должны сделать патч для исправления уязвимости. Вы можете использовать любой метод, подходящий для данного пакета, в том числе *методы UDD*, но в

этом примере будем использовать `edit-patch` (из пакета ubuntu-dev-tools). `edit-patch` — это самый простой способ для исправления пакетов, работающий с любой системой патчей, которую вы можете себе представить.

Чтобы создать патч с помощью `edit-patch`:

```
$ cd dbus
$ edit-patch 99-fix-a-vulnerability
```

Это применит все существующие патчи и поместит пакет во временный каталог. Теперь отредактируйте файлы для исправления уязвимостей. Обычно в апстриме лежит и патч, поэтому Вы сразу можете его применить:

```
$ patch -p1 < /home/user/dbus-vulnerability.diff
```

После внесения необходимых изменений просто нажмите Ctrl-D или наберите exit, чтобы покинуть временную командную оболочку.

Форматирование файла changelog и патчей

После применения патчей вам потребуется внести изменения в лог. Команда `dch` используется для редактирования файла `debian/changelog` и `edit-patch` автоматически запустит `dch` после отката всех патчей. Если Вы не пользуетесь `edit-patch`, то можете запустить `dch -i` вручную. В отличии от обычных патчей, Вам следует использовать следующий формат (обратите внимание, что в имени дитрибутива используется precise-security, так как это обновление безопасности для Precise) для обновлений безопасности:

```
dbus (1.4.18-2ubuntu1.5) precise-security; urgency=low

  * SECURITY UPDATE: [DESCRIBE VULNERABILITY HERE]
    - debian/patches/99-fix-a-vulnerability.patch: [DESCRIBE CHANGES HERE]
    - [CVE IDENTIFIER]
    - [LINK TO UPSTREAM BUG OR SECURITY NOTICE]
    - LP: #[BUG NUMBER]
...
```

Обновите свой патч для использования соответствующих тегов. Ваш патч должен содержать как минимум теги Origin, Description и Bug-Ubuntu. Например, отредактируйте `debian/patches/99-fix-a-vulnerability.patch`, чтобы он имел приблизительно следующие строки:

```
## Description: [DESCRIBE VULNERABILITY HERE]
## Origin/Author: [COMMIT ID, URL OR EMAIL ADDRESS OF AUTHOR]
## Bug: [UPSTREAM BUG URL]
## Bug-Ubuntu: https://launchpad.net/bugs/[BUG NUMBER]
Index: dbus-1.4.18/dbus/dbus-marshal-validate.c
...
```

Множественные уязвимости можно исправить одной загрузкой безопасности, просто убедитесь что используете разные патчи для разных уязвимостей.

Проверка и отправка вашей работы

На этом этапе процесс такой же, как при *исправлении обычных ошибок в Ubuntu*. А именно, вам нужно:

1. Выполнить сборку пакета и проверить, что он компилируется без ошибок и компилятор не выдаёт никаких дополнительных предупреждений
2. Выполнить обновление с предыдущей версии пакета до новой версии

3. Убедиться, что новый пакет закрывает уязвимость и не вносит никаких ухудшений
4. Отправляйте свою работу через предложение об объединении Launchpad и отправляйте баг в Launchpad, убедившись что пометили баг как ошибку безопасности, и для подписки `ubuntu-security-sponsors`

Если это уязвимость в безопасности, о которой ещё не объявлено публично, то не отправляйте предложение слияния и убедитесь, что вы пометили свою ошибку, как приватную (private).

Отправленный баг должен содержать Тестовый Пример, т.е. комментарий, который четко показывает как воссоздать баг, запустив старую версию, также показывая как убедиться, что баг больше не существует в новой версии.

Отчет по багу также должен подтверждать, что ошибка исправлена в новых версиях Ubuntu при помощи предложенного фикса (в вышеуказанном примере выше чем в Precise). Если проблема не исправлена в новых версиях Ubuntu, вы должны подготовить обновлениях и для новых версий.

1.7.2 Обновления стабильного релиза

Мы также разрешаем вносить обновления в выпуски, в которых пакет содержит серьёзную ошибку, такую как значительная регрессия по сравнению с предыдущим выпуском или ошибка, которая может привести к потере данных. Из-за того, что такие изменения сами потенциально могут привести к появлению дополнительных ошибок, мы позволяем делать это только там, где изменения легко можно понять и проверить.

Процесс обновлений стабильного выпуска (Stable Release Updates или SRU) такой же, как и для исправлений ошибок безопасности, за исключением того, что нужно подписать на отчёт об ошибке команду `ubuntu-sru`.

Обновление попадет в архив `proposed‘‘(к примеру ‘‘precise-proposed`), где его проверяют на способность исправить проблему и подтверждают, что оно не является следствием новых проблем. После недели работы без заявленных проблем, обновление попадает в раздел `updates`.

Смотрите ‘Вики страницу Обновлений Стабильного Релиза <SRUWiki>‘_ для получения дополнительной информации.

1.8 Патчи для пакетов

Иногда разработчикам пакета Ubuntu надо изменить исходный код апстрима, чтобы заставить его работать в Ubuntu должным образом. Примеры включают патчи для апстримов, которые еще не попали в версию релиза, либо изменения к системам билдов апстрима, необходимые только для их сборки на Ubuntu. Мы будем менять исходный код апстрима напрямую, но такой метод делает более сложным дальнейшее удаление патчей, когда апстрим уже применил их, также усложняя извлечение изменений для их отправки в проект апстрима. Вместо этого, мы будем хранить эти изменения как отдельные патчи в форме diff файлов.

Существуют различные способы работы с патчами для пакетов Debian. К счастью, мы остановимся на одной системе, Quilt, которая сейчас используется большинством пакетов.

Давайте возьмём в качестве примера пакет `kamoso` в Trusty:

```
$ bzr branch ubuntu:trusty/kamoso
```

Патчи хранятся в `debian/patches`. Для этого пакета имеется один патч `kubuntu_01_fix_qmax_on_armel.diff` для исправления ошибки компиляции на платформе ARM. Этому патчу присвоено имя, описывающее, что он делает, номер патча по порядку (чтобы избежать

путаницы, если два патча имеют одинаковое имя) и, в данном случае, команда Kubuntu добавила свой собственный префикс, чтобы показать, что патч исходит от них, а не от Debian.

Порядок применения патчей хранится в `debian/patches/series`.

1.8.1 Патчи с помощью Quilt

Перед работой с Quilt нужно указать этой системе, где искать патчи. Добавьте в `~/.bashrc` следующее:

```
export QUILT_PATCHES=debian/patches
```

И источник файла для применения нового экспорта:

```
$ . ~/.bashrc
```

По умолчанию все патчи применяются уже с UDD извлечений или загружаемых пакетов. Вы можете проверить это с помощью:

```
$ quilt applied
kubuntu_01_fix_qmax_on_armel.diff
```

Если вы хотите удалить патч, нужно выполнить `pop`:

```
$ quilt pop
Removing patch kubuntu_01_fix_qmax_on_armel.diff
Restoring src/kamoso.cpp

No patches applied
```

А чтобы применить патч, используйте `push`:

```
$ quilt push
Applying patch kubuntu_01_fix_qmax_on_armel.diff
patching file src/kamoso.cpp

Now at patch kubuntu_01_fix_qmax_on_armel.diff
```

1.8.2 Добавление нового патча

Для добавления нового патча нужно указать Quilt создать новый патч, сообщить ему, какие файлы этот патч должен изменить, отредактировать файлы, а затем обновить патч:

```
$ quilt new kubuntu_02_program_description.diff
Patch kubuntu_02_program_description.diff is now on top
$ quilt add src/main.cpp
File src/main.cpp added to patch kubuntu_02_program_description.diff
$ sed -i "s,Webcam picture retriever,Webcam snapshot program,"
src/main.cpp
$ quilt refresh
Refreshed patch kubuntu_02_program_description.diff
```

Шаг `quilt add` важен: если вы забудете его сделать, файлы не попадут в патч.

Теперь изменения будут в `debian/patches/kubuntu_02_program_description.diff`, а в файл `series` будет добавлена информация о новом патче. Вы должны добавить новый файл в исходные файлы для пакета:


```
$ bzr add debian/patches/kubuntu_02_program_description.diff
$ bzr add .pc/*
$ dch -i "Add patch kubuntu_02_program_description.diff to improve the program description"
$ bzr commit
```

Quilt содержит свои мета-данные в директории `.pc/`, поэтому сейчас вам нужно добавить в пакет и её. Это должно быть улучшено в будущем.

Как правило, следует проявлять осторожность при добавлении патчей к программам, если они исходят не из апстрима. Часто имеется веская причина, по которой изменения ещё не были сделаны. В рассмотренном выше примере изменяется строка в пользовательском интерфейсе, так что это может поломать все переводы. Если сомневаетесь, спросите автора из апстрима перед тем, как добавить патч.

1.8.3 Заголовки патчей

Мы рекомендуем добавлять к каждому патчу заголовки DEP-3, помещая их в самом верху файла патча. Вот некоторые заголовки, которые можно использовать:

Description Описание того, что делает патч. Имеет формат, аналогичный полю `Description` в `debian/control`: первая строка содержит краткое описание, начинающееся со строчной буквы, остальные строки содержат более длинное описание с пробелом в качестве отступа.

Author Кто написал патч (например, "Jane Doe <packager@example.com>").

Origin Откуда пришёл этот патч (например, "upstream"), если заголовок *Author* отсутствует.

Bug-Ubuntu Ссылка на информацию об ошибке на Launchpad, предпочтительно, в краткой форме (наподобие *https://bugs.launchpad.net/bugs/XXXXXXX*). Если также имеются отчёты об ошибках в апстриме или системах отслеживания ошибок Debian, добавьте заголовки *Bug* или *Bug-Debian*.

Forwarded Был ли патч передан в апстрим. Значения: "yes", "no" или "not-needed".

Last-Update Дата последней ревизии (в форме "ГГГГ-ММ-ДД").

1.8.4 Обновление до новых версий из апстрима

Чтобы выполнить обновление до последней версии, вы можете использовать команду `bzr merge-upstream`:

```
$ bzr merge-upstream --version 2.0.2 https://launchpad.net/ubuntu/+archive/primary/+files/kamoso_2.0.2.orig.tar.bz2
```

При запуске этой команды произойдет откат всех патчей, так как они могут стать устаревшими. Возможно потребуется их обновить для соответствия новому источнику апстрима, либо потребуется удалить их все вместе. Для облегчения проверки проблем применяйте патчи по одному.

```
$ quilt push
Applying patch kubuntu_01_fix_qmax_on_armel.diff
patching file src/kamoso.cpp
Hunk #1 FAILED at 398.
1 out of 1 hunk FAILED -- rejects in file src/kamoso.cpp
Patch kubuntu_01_fix_qmax_on_armel.diff can be reverse-applied
```

Если для патча указано `it can be reverse-applied`, значит патч уже был применён апстримом, так что мы можем удалить этот патч:


```
$ quilt delete kubuntu_01_fix_qmax_on_armel
Removed patch kubuntu_01_fix_qmax_on_armel.diff
```

Затем продолжайте:

```
$ quilt push
Applied kubuntu_02_program_description.diff
```

Неплохой мыслью будет выполнить refresh, это обновит патч относительно изменений исходного кода в апстриме:

```
$ quilt refresh
Refreshed patch kubuntu_02_program_description.diff
```

Затем выполните фиксацию, как обычно:

```
$ bzr commit -m "new upstream version"
```

1.8.5 Создание пакета с использованием Quilt

Современные пакеты используют Quilt по умолчанию, это встроено в формат исходных файлов пакета. Проверьте, что в `debian/source/format` указано `3.0 (quilt)`.

Для более старых пакетов, использующих формат 1.0, необходимо использовать Quilt явно, обычно с помощью включения make-фала в `debian/rules`.

1.8.6 Конфигурирование Quilt

Вы можете воспользоваться файлом `~/.quiltrc` для конфигурирования quilt. Вот несколько опций, которые могут быть полезны при использовании quilt с пакетами Debian:

```
# Set the patches directory
QUILT_PATCHES="debian/patches"
# Remove all useless formatting from the patches
QUILT_REFRESH_ARGS="-p ab --no-timestamps --no-index"
# The same for quilt diff command, and use colored output
QUILT_DIFF_ARGS="-p ab --no-timestamps --no-index --color=auto"
```

1.8.7 Другие системы управления патчами

Другие системы патчинга, используемые в пакетах, включают `dpatch` и `cdbs simple-patchsys`, принцип работы которых похож на Quilt - патчи хранятся в `debian/patches`, но для их применения, отмены или создания требуются другие команды. Вы можете узнать какая система патчинга используется в пакете при помощи команды `what-patch‘‘(в пакете ‘‘ubuntu-dev-tools`). Вы можете использовать `edit-patch`, показанный в *предыдущих главах*, в качестве надежного способа для работы со всеми системами.

В более старых пакетах изменения будут включены напрямую в источники и храниться в исходном файле `diff.gz`. Это делает сложнее процесс обновления до новых версий апстрима или различия между патчами - лучше избегать.

Не изменяйте систему патчей, не обсудив это с сопровождающим Debian или имеющей отношение к делу командой Ubuntu. Если существующей системы патчей нет, можете добавить Quilt.

1.9 Исправление пакетов FTBFS (Fails To Build From Source)

Перед тем, как пакет можно будет использовать в Ubuntu, он должен быть собран из исходного кода. Если это не удаётся, пакет, вероятно, будет ожидать в -proposed и не будет доступен в архивах Ubuntu. Полный список пакетов, которые не удалось собрать из исходного кода, можно найти на http://qa.ubuntuwire.org/ftbfs/. На этой странице показано 5 основных категорий:

- Package failed to build (F): Что-то действительно пошло не так в процессе сборки.
- Отменённая сборка (X): сборка была отменена по некоторой причине. Для начала, с ними лучше не связываться.
- Package is waiting on another package (M): Этот пакет ожидает сборки или обновления другого пакета, или (если это пакет в main) одна из его зависимостей находится не в той части архива.
- Проблема в chroot (C): Некоторая операция над chroot-окружением столкнулась с ошибкой. Чаще всего исправляется повторной сборкой. Попросите разработчика запустить пересборку.
- Ошибка при загрузке (U): Пакет не может быть загружен на сервер. Обычно в этом случае нужно сделать пересборку, но перед этим – проверьте логи сборки.

1.9.1 Первые шаги

Первым делом необходимо повторить FTBFS самостоятельно. Скачайте код с помощью `bzr branch lp:ubuntu/PACKAGE` и получите tar-архив, или запустите `dget PACKAGE_DSC` над .dsc-файлом со страницы проекта на Launchpad. После этого, создайте chroot-окружение.

У вас должно получиться воспроизвести ошибку FTBFS. Если же нет – проверьте, не качает ли сборка отсутствующую зависимость: в таком случае необходимо в файле debian/control объявить её как build-depends. Другой вариант – попробовать собрать пакет локально, что позволит проверить отсутствующие или не указанные зависимости (в таком случае локальная сборка должна быть успешна, а в schroot – нет)

1.9.2 Проверка Debian

В случае, если проблему удалось воспроизвести – необходимо приступить к поиску решения. Если пакет также находится в Debian, – проверьте, возможно у них пакет собирается нормально: http://packages.qa.debian.org/PACKAGE. Если в Debian есть более новая версия пакета, его нужно объединить (merge). Если же нет – проверьте логи сборки и ссылки на известные проблемы: там можеет быть дополнительная информация о FTBFS или патчи. Debian также поддерживает список команд разных FTBFS, в котором также есть варианты решения разнообразных проблем: https://wiki.debian.org/qa.debian.org/FTBFS.

1.9.3 Другие причины возникновения FTBFS

Если пакет находится в main, но для него отсутствует зависимость не из main, то необходимо отправить MIR-баг: страница https://wiki.ubuntu.com/MainInclusionProcess описывает эту процедуру.

1.9.4 Исправление ошибки

Если удалось исправить проблему, следуйте такой же процедуре как и при любых других проблемах: создайте патч, добавьте его в ветку или баг bzr, подпишите ubuntu-sponsors, а потом попробуйте добиться его добавления в исходный пакет и/или в Debian.

1.10 Общие библиотеки

Общие библиотеки — это скомпилированный код, предназначенный для совместного использования несколькими различными программами. Они распространяются в виде файлов `.so` в `/usr/lib/`.

Библиотеки экспортируют символы в скомпилированном виде: функции, классы и переменные. У каждой библиотеки также есть название SONAME, включающее номер её версии, но который не обязательно совпадает с официальным номером релиза. Программы компилируются с конкретным SONAME библиотеки. Так, если какой-либо из символов библиотеки был удалён или изменён – небходимо изменить версию с тем, чтобы все зависящие от библиотеки пакеты были перекомпилированы с использованием новой версии. Обычно версии устанавливаются в источнике, и мы используем те же номера версий для двоичных пакетов, называемые "номер ABI", но в случае, если источник не использует вменяемой версионности, создатели пакетов могут использовать отличную, более традиционную нумерацию.

Библиотеки обычно распространяются апстримом в виде отдельных выпусков. Иногда они распространяются, как часть программы. В последнем случае они могут быть включены в двоичный пакет вместе с программой (это называется bundling), если вы не предполагаете использование этих библиотек другими программами, но чаще их всё же следует выделять в отдельные двоичные пакеты.

Сами библиотеки помещаются в двоичный пакет с именем `libfoo1`, где `foo` — имя библиотеки, а `1` — версия из SONAME. Файлы разработки из пакета, такие как заголовочные файлы, необходимые для компиляции программ с библиотекой, помещаются в пакет с именем `libfoo-dev`.

1.10.1 Пример

В качестве примера мы используем libnova:

```
$ bzr branch ubuntu:trusty/libnova
$ sudo apt-get install libnova-dev
```

Чтобы найти SONAME библиотеки, выполните:

```
$ readelf -a /usr/lib/libnova-0.12.so.2 | grep SONAME
```

SONAME в данном случае `libnova-0.12.so.2`, что соответствует имени файла (как правило, но не всегда). Здесь апстрим поместил номер версии из апстрима, как часть SONAME, и задал ABI-версию 2. Имена библиотечных пакетов должны следовать SONAME библиотеки, которую они содержат. Двоичный библиотечный пакет называется `libnova-0.12-2`, где `libnova-0.12` — имя библиотеки, а `2` — наш ABI-номер.

Если авторы из апстрима внесли несовместимые изменения в свою библиотеку, они должны изменить номер версии SONAME, а мы должны переименовать нашу библиотеку. Любые другие пакеты, использующие наш библиотечный пакет, нужно будет перекомпилировать с новой весрией, это называется переходом (transition) и требует некоторых усилий. Надо надеяться, наш ABI-номер продолжит соответствовать SONAME апстрима, но иногда они вносят несовместимости без изменения их номера версии, а нам нужно изменить наш.

Взглянув на debian/libnova-0.12-2.instal, мы увидим, что он включает в себя два файла:

```
usr/lib/libnova-0.12.so.2
usr/lib/libnova-0.12.so.2.0.0
```

Вторая строчка — настоящая библиотека, с полным номером версии. Первая — символическая ссылка, указывающая на настоящую библиотеку. Программы, использующие библиотеку, как правило, будут пользоваться символической ссылкой.

`libnova-dev.install` содержит все файлы, необходимые для компиляции программы с данной библиотекой. Заголовочные файлы, бинарник конфигурации, файл libtool'а `.la`, а также `libnova.so` – ещё один симлинк на библиотеку, создаваемый с тем, чтобы программы могли компилироваться вне зависимости от старшего номера версии (при этом, скомпилированное приложение всё равно будет зависеть от версии).

`.la`-файлы libtool'а требуются на некоторых не-Linux системах с ограниченной поддержкой библиотек, но на системах Debian зачастую создают больше проблем, чем решают. Цель Debian отказаться от .la-файлов сегодня весьма актуальна, и вы можете помочь с решением этой задачи.

1.10.2 Статические библиотеки

Пакет -dev также содержит `usr/lib/libnova.a`. Это статическая библиотека, альтернатива общей библиотеке. Любая программа, скомпилированная со статической библиотекой, содержит её код непосредственно в себе. Это позволяет не беспокоиться о двоичной совместимости библиотеки. Однако это также означает, что любые ошибки, в том числе уязвимости в безопасности, не будут исправлены за счёт обновления библиотеки, пока программа не будет перекомпилирована. По этой причине использовать программы со статическими библиотеками не рекомендуется.

1.10.3 Символьные файлы

Когда приложение компилируется с библиотекой, механизм `shlibs` добавит к пакету зависимость от этой библиотеки. Именно поэтому большинство программ содержат `Depends: ${shlibs:Depends}` в файле `debian/control`. Этот максов заменяется списком зависимых библиотек при билде. Однако shlibs может только указывать зависимость от старшей ABI-версии , 2 в нашем примере с libnova, так что если новые символы будут добавлены в будущей libnova 2.1 – приложение будет запускаться и с более старой версией libnova ABI 2.0, что приведёт к аварианому завершению.

Чтобы точнее указывать зависимости от библиотек, мы создали файл `.symbols`, который перечисляет все символы библиотеки и версии, в которых они появились.

libnova не имеет символьного файла, так что мы можем создать его. Начните с компиляции пакета:

```
$ bzr builddeb -- -nc
```

Опция `-nc` указывает не удалять сборочные файлы после завершения компиляции. Перейдите в каталог сборки и выполните `dpkg-gensymbols` для пакета библиотеки:

```
$ cd ../build-area/libnova-0.12.2/
$ dpkg-gensymbols -plibnova-0.12-2 > symbols.diff
```

Это создаст diff-файл, который вы сможете применить самостоятельно:

```
$ patch -p0 < symbols.diff
```

Это создаст файл с именем вида `dpkg-gensymbolsnY_WWI`, в котором будут перечислены все символы. Он также указывает текущую версию пакета. Версию пакета можно убрать из файла, так как новые символы обычно добавляются не с новой версией пакета, а разработчиками исходной библиотеки.

```
$ sed -i s,-0ubuntu2,, dpkg-gensymbolsnY_WWI
```

Теперь переместите файл туда, где он должен находиться, зафиксируйте изменения и выполните тестовую сборку:


```
$ mv dpkg-gensymbolsnY_WWI ../../libnova/debian/libnova-0.12-2.symbols
$ cd ../../libnova
$ bzr add debian/libnova-0.12-2.symbols
$ bzr commit -m "add symbols file"
$ bzr builddeb
```

Если компиляция выполняется успешно, значит символьный файл не содержит ошибок. С выходом следующей апстрим-версии libnova вам придётся снова запустить dpkg-gensymbols, чтобы создать diff для обновления символьного файла.

1.10.4 Символьные файлы библиотек C++

У языка C++ более строгие стандарты на двоичную совместимость, чем у C. Команда Debian Qt/KDE поддерживает некоторые скрипты, которые помогут справиться с этим: страница Работа с файлами symbols описывает принципы их использования.

1.10.5 Информация для дальнейшего чтения

Статья Junichi Uekawa Пакетирование библиотек для Debian рассматривает этот вопрос более детально.

1.11 Бэкпортирование обновлений программ

Порой может потребоваться добавить функционал в стабильный релиз, который не связан с исправлением критических проблем. В подобных случаях, есть два варианта: либо вы загрузите его в PPA, или подготовите бэкпорт (backport).

1.11.1 Персональные архивы пакетов (PPA)

Использование PPA имеет ряд преимуществ. Это достаточно просто, вам не понадобится одобрение от кого бы то ни было, но недостаток в том, что пользователям придётся вручную подключать PPA. Это нестандартный источник приложений.

Документация к PPA на Launchpad достаточно исчерпывающая и поможет вам быстро начать работу с ним.

1.11.2 Официальные бэкпорты Ubuntu

Целью проекта Backports является предоставление пользователям новой функциональности. Из-за рисков уменьшения стабильности при портировании новшеств, бэкпорты недоступны пользователям, пока они не включат их. Поэтому бэкпорты не являются местом для исправления ошибок. Если в пакете Ubuntu обнаружена ошибка, она должна быть исправлена через *обновления безопасности и стабильности*.

Когда вы определите, требуется ли вам адаптировать ваши изменения для стабильного релиза, вам будет необходимо собрать и протестировать ваш пакет на данном релизе. Команда `pbuilder-dist` (из пакета `ubuntu-dev-tools`) поможет вам сделать это.

Чтобы подать заявку на бэкпорт, можно использовать утилиту `requestbackport` (также из пакета `ubuntu-dev-tools`). Она определит все промежуточные выпуски, для которых пакет также придётся бэкпортировать, покажет, какие пакеты зависят от данного, и создаст заявку. Она также включит список требуемых тестов в заявку.

Глава 2

База знаний

2.1 Коммуникация при Разработке в Ubuntu

В проекте, где подвергаются изменениям тысячи строк кода, принимается множество решений, и где сотни людей должны взаимодействовать каждый день, важно иметь эффективную связь.

2.1.1 Почтовые рассылки

Почтовые рассылки — это достаточно эффективный инструмент, если вы хотите обсудить идеи в команде и убедиться что вы оповестили всех, несмотря на различия в часовых поясах.

С точки зрения разработки, это наиболее важные из рассылок:

- https://lists.ubuntu.com/mailman/listinfo/ubuntu-devel-announce (только анонсы, самые важные объявления разработки попадают сюда)
- https://lists.ubuntu.com/mailman/listinfo/ubuntu-devel (главная дискуссия разработчиков Ubuntu)
- https://lists.ubuntu.com/mailman/listinfo/ubuntu-motu (обсуждения команды MOTU, получение справки по созданию пакетов)

2.1.2 Каналы IRC

Для онлайн-дискуссии подключитесь к irc.freenode.net и присоединяйтесь к любому из каналов:

- #ubuntu-devel (для главной дискуссии разработчиков)
- #ubuntu-motu (для обсуждений команды MOTU и получения помощи)

2.2 Общий обзор каталога `debian/`

Эта статья даёт краткие пояснения к различным файлам, важным для создания пакетов Ubuntu, которые содержатся в каталоге `debian/`. Самыми важными из них являются `changelog`, `control`, `copyright`, and `rules`. Они требуются для всех пакетов. Многие дополнительные файлы в `debian/` могут использоваться для настройки и изменения поведения пакета. Некоторые из этих файлов обсуждаются в этой статье, но это далеко не полный список.

2.2.1 Файл changelog

Этот файл, как следует из его названия — это список изменений, внесённых в каждую версию. Он имеет особый формат, который показывает имя пакета, версию, дистрибутив, изменения, и кто вносил изменения в данное время. Если у вас есть ключ GPG (смотрите: *Подготовка*), убедитесь, что вы используете в `changelog` те же имя и адрес электронной почты, что и в вашем ключе. Ниже приведен шаблон `changelog`:

```
package (version) distribution; urgency=urgency

 * change details
  - more change details
 * even more change details

-- maintainer name <email address>[two spaces]  date
```

Формат (особенно даты) важен. Дата должна быть в формате **RFC 5322**, который можно увидеть при выполнении команды `date -R`. Для удобства, можно использовать для редактирования changelog команду `dch`. Она обновит дату автоматически.

Пункты с незначительными изменениями отмечаются дефисом «-», в то время как в важных пунктах используется звездочка «*».

Если вы создаёте пакет «с нуля», `dch --create` (`dch` находится в пакете `devscripts`) создаст для вас стандартный файл `debian/changelog`.

Вот пример файла `changelog` для hello:

```
hello (2.8-0ubuntu1) trusty; urgency=low

  * New upstream release with lots of bug fixes and feature improvements.

-- Jane Doe <packager@example.com>  Thu, 21 Oct 2013 11:12:00 -0400
```

Обратите внимание, что версия имеет `-0ubuntu1` добавленный к нему, это - distro версия, используемая так, чтобы упаковка могла быть обновлена (чтобы исправить ошибки, например) с новыми закачками в той же исходной версии выпуска.

Ubuntu и Debian используют немного различающиеся схемы нумерации версий пакетов, чтобы избежать конфликта пакетов с одной и той же исходной версией. Если пакет Debian был изменён в Ubuntu, к концу Debian-версии добавляется `ubuntuX` (где `X` — номер редакции в Ubuntu). Таким образом, если пакет Debian hello `2.6-1` был изменён в Ubuntu, номер версии будет `2.6-1ubuntu1`. Если пакет приложения в Debian не существует, то редакция Debian равна `0` (например, `2.6-0ubuntu1`).

Более подробную информацию можно найти на странице changelog (Глава 4.4) документа Debian Policy Manual.

2.2.2 Файл control

Файл `control` содержит информацию, которую использует менеджер пакетов (такой, как `apt-get`, `synaptic` или `adept`), сборочные зависимости, информацию мэйнтейнера и многое другое.

Для пакета Ubuntu `hello` файл `control` выглядит следующим образом:

```
Source: hello
Section: devel
Priority: optional
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
```



```
XSBC-Original-Maintainer: Jane Doe <packager@example.com>
Standards-Version: 3.9.5
Build-Depends: debhelper (>= 7)
Vcs-Bzr: lp:ubuntu/hello
Homepage: http://www.gnu.org/software/hello/

Package: hello
Architecture: any
Depends: ${shlibs:Depends}
Description: The classic greeting, and a good example
 The GNU hello program produces a familiar, friendly greeting. It
 allows non-programmers to use a classic computer science tool which
 would otherwise be unavailable to them. Seriously, though: this is
 an example of how to do a Debian package. It is the Debian version of
 the GNU Project's 'hello world' program (which is itself an example
 for the GNU Project).
```

Первый абзац описывает исходный пакет, включая список пакетов, требуемых для сборки данного пакета из исходного кода, в поле `Build-Depends`. Он также содержит некоторую метаинформацию, такую как имя мэйнтейнера, версию Debian Policy, с которой компилируется пакет, местоположение репозитория управления версиями и домашнюю страницу апстрима.

Заметьте, что в Ubuntu мы указываем в поле `Maintainer` общий адрес, так как любой человек может изменить любой пакет (в отличие от Debian, где правом изменения пакетов обладают лишь отдельные люди или команда). Пакеты в Ubuntu, как правило, должны в поле `Maintainer` содержать `Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>`. Если поле Maintainer изменено, старое значение должно быть сохранено в поле `XSBC-Original-Maintainer`. Это можно сделать автоматически сценарием `update-maintainer` из пакета `ubuntu-dev-tools`. Для дальнейшей информации смотрите Debian Maintainer Field spec в Ubuntu wiki.

Каждый дополнительный абзац описывает бинарный пакет, который будет создан.

Более подробную информацию можно найти на странице секции control-файла (Глава 5) документа Debian Policy Manual.

2.2.3 Файл copyright

Файл содержит информацию о копирайтах исходных кодов и самого пакета. Ubuntu и Debian Policy (Глава 12.5) требуют, чтобы каждый пакет устанавливал неизменную копию копирайта и информации по лицензированию в папку `/usr/share/doc/$(имя_пакета)/copyright`

Как правило, информацию об авторских правах можно найти в файле `COPYING` в каталоге с исходным кодом программы. Этот файл должен включать такую информацию, как имена автора и упаковщика, URL, из которого получен источник, строку со значком копирайта с указанием года и владельца авторских прав, а также сам текст авторского права. Шаблон для примера:

```
Format: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: Hello
Source: ftp://ftp.example.com/pub/games

Files: *
Copyright: Copyright 1998 John Doe <jdoe@example.com>
License: GPL-2+

Files: debian/*
Copyright: Copyright 1998 Jane Doe <packager@example.com>
License: GPL-2+
```

```
License: GPL-2+
 This program is free software; you can redistribute it
 and/or modify it under the terms of the GNU General Public
 License as published by the Free Software Foundation; either
 version 2 of the License, or (at your option) any later
 version.
 .
 This program is distributed in the hope that it will be
 useful, but WITHOUT ANY WARRANTY; without even the implied
 warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
 PURPOSE.  See the GNU General Public License for more
 details.
 .
 You should have received a copy of the GNU General Public
 License along with this package; if not, write to the Free
 Software Foundation, Inc., 51 Franklin St, Fifth Floor,
 Boston, MA  02110-1301 USA
 .
 On Debian systems, the full text of the GNU General Public
 License version 2 can be found in the file
 '/usr/share/common-licenses/GPL-2'.
```

Пример испольует машино-понятный формат debian/copyright, и авторам пакетов также рекомендуется следовать этому формату.

2.2.4 Файл rules

Последний файл, который мы рассмотрим, это `rules`. Он выполняет всю работу по сборке нашего пакета. Это Makefile, в котором есть функции компиляции программы, её установки и создания `.deb`-пакета из установленных файлов. В нём есть также функция очистки файлов сборки, которая удаляет всё, кроме собственно пакета исходного кода.

Вот упрощённый пример файла rules, созданного `dh_make` (который можно найти в пакете `dh-make`):

```
#!/usr/bin/make -f
# -*- makefile -*-

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1

%:
	dh  $@
```

Давайте рассмотрим этот файл внимательнее. На каждом этапе сборки `debian/rules` вызывается с аргументом, который передаётся `/usr/bin/dh`, который, в свою очередь, вызывает необходимые команды `dh_*`.

`dh` запускает последовательность команд debhelper. Поддерживаемые последовательности соответствуют целям файла `debian/rules`: «build», «clean», «install», «binary-arch», «binary-indep» и «binary». Чтобы увидеть, какие команды выполняются в каждой цели, наберите:

```
$ dh binary-arch --no-act
```

Командам из последовательности `binary-indep` передаётся аргумент `-i`, чтобы они затрагивали только архитектурно-независимые пакеты, а командам из последовательности `binary-arch` — аргумент `-a`, чтобы онм затрагивали только архитектурно-зависимые пакеты.

Каждая команда debhelper при её успешном выполненении делает запись в журнале `debian/package.debhelper.log` (который затем удаляет dh_clean.) Таким образом dh может определить, какие команды уже были выполнены и для каких пакетов, что помогает избежать повторного выполнения этих команд.

При каждом запуске `dh` он изучает журнал, находит добавленные последними команды, которые относятся к указанной последовательности. Затем он продолжает выполнение со следующей команды в этой последовательности. Опции `--until`, `--before`, `--after` и `--remaining` могут изменить это поведение.

Если в `debian/rules` есть функция с именем, похожим на `override_dh_команда`, то вместо данной команды `dh` выполнит данную функцию. Эта функция может запустить ту же команду с другими аргументами, или же совершенно другую команду. (Замечание: для использования этой функциональности при сборке требуется пакет debhelper версии не менее 7.0.50).

За дополнительными примерами обратитесь к `/usr/share/doc/debhelper/examples/` и `man dh`. Смотрите также раздел о файле rules (Раздел 4.9) в «Debian Policy Manual».

2.2.5 Дополнительные файлы

Файл install

Файл `install` используется `dh_install` для установки файлов в двоичный пакет. Он имеет два стандартных варианта использования:

- Для установки в ваш пакет файлов, не установленных оригинальной системой сборки.
- Разделение одного большого пакета источника на несколько бинарных пакетов.

В первом случае файл `install` должен содержать одну строку для каждого устанавливаемого файла, указывающую как файл, так и установочный каталог. Например, следующий файл `install` установит сценарий `foo` из корневого каталога пакета исходного кода в `usr/bin` и desktop-файл из каталога `debian` в `usr/share/applications`:

```
foo usr/bin
debian/bar.desktop usr/share/applications
```

Если пакет исходного кода производит несколько двоичных пакетов, `dh` установит файлы в `debian/tmp` вместо установки непосредственно в `debian/<пакет>`. Файлы, установленные в `debian/tmp` затем можно переместить в отдельные двоичные пакеты с помощью нескольких файлов `$имя_пакета.install`. Это часто делается, чтобы разбить большое количество не зависящих от архитектуры данных из зависящих от архитектуры пакетов в пакеты `Architecture: all`. В этом случае нужно указать только имена устанавливаемых файлов (или каталогов), без установочного каталога. Например, `foo.install`, содерщащий только зависящие от архитектуры файлы, может выглядеть наподобие:

```
usr/bin/
usr/lib/foo/*.so
```

В то время как `foo-common.install`, содержащий только не зависящие от архитектуры файлы, может выглядеть так:

```
/usr/share/doc/
/usr/share/icons/
/usr/share/foo/
/usr/share/locale/
```

Будут созданы два двоичных пакета: `foo` и `foo-common`. Для обоих требуется их собственный абзац в `debian/control`.

Для дополнительных подробностей смотрите man dh_install и раздел о файле install (Раздел 5.11) в «Debian New Maintainers' Guide».

Файл watch

Файл debian/watch позволяет автоматически проверять наличие новых версий в апстриме с помощью инструмента uscan из пакета devscripts. Первой строкой файла watch должна быть версия формата (3 на момент написания этого руководства), а следующий строки содержат любые URL для анализа. Например:

```
version=3

http://ftp.gnu.org/gnu/hello/hello-(.*).tar.gz
```

Запуск uscan в корневом каталоге исходников сравнит номер апстрим-версии в debian/changelog с последней доступной в апстриме версией. Если в апстриме найдена новая версия, она будет автоматически загружена. Например:

```
$ uscan
hello: Newer version (2.7) available on remote site:
  http://ftp.gnu.org/gnu/hello/hello-2.7.tar.gz
  (local version is 2.6)
hello: Successfully downloaded updated package hello-2.7.tar.gz
    and symlinked hello_2.7.orig.tar.gz to it
```

Если ваши tarball-файлы обитают на Launchpad, файл debian/watch имеет немного более сложный вид (о том, почему это так, смотрите Question 21146 и Bug 231797). В этому случае используйте нечто вроде:

```
version=3
https://launchpad.net/flufl.enum/+download http://launchpad.net/flufl.enum/.*/flufl.enum-(.+).tar.gz
```

Дополнительные сведения смотрите в man uscan и в разделе о файле watch (Раздел 4.11) «Debian Policy Manual».

Список пакетов, для которых файл watch сообщает о том, что они не синхронизированы с апстримом, смотрите Ubuntu External Health Status.

Файл source/format

Этот файл указывает формат пакета исходного кода. Он должен содержать одну строку, показывающую выбранный формат:

- 3.0 (native) для «родных» пакетов Debian (апстрим-версия отсутствует)
- 3.0 (quilt) для пакетов с отдельным тарболом из апстрима
- 1.0 для пакетов, желающих явно указать формат по умолчанию

В настоящее время по умолчанию выбирается формат пакета исходного кода 1.0, если этот файл отсутствует. В файле source/format можно указать его явно. Если вы не используете этот файл для указания формата исходного кода, Lintian выдаст предупреждение об отстутствии файла. Это чисто информационное предупреждение и его можно без опасений проигнорировать.

Рекомендуется использовать более новый формат 3.0. Он предоставляет некоторые новые возможности:

- Поддержка дополнительных форматов сжатия: bzip2, lzma и xz
- Поддержка нескольких архивов с оригинальным исходным кодом

- Необязательно перепаковывать архив с оригинальным исходным кодом, чтобы удалить директорию `debian`.
- Специфичные для Debian изменения теперь хранятся не в одном файле .diff.gz, а в виде нескольких патчей, совместимых с quilt, в каталоге `debian/patches/`

https://wiki.debian.org/Projects/DebSrc3.0 содержит дополнительную информацию касательно перехода на версию 3.0 формата исходных пакетов.

Дополнительную информацию можно найти в `man dpkg-source` и в Главе source/format (Глава 5.21) руководства Debian New Maintainers.

2.2.6 Дополнительные ресурсы

Кроме Debian Policy Manual, на который ссылается статья, руководство Debian New Maintainers' Guide содержит более детальное описание для каждого файла. Глава 4, "Необходимые файлы в папке debian" описывает файлы control, changelog, copyright и rules. Глава 5, "Прочие файлы в папке debian" описывает дополнительные файлы, которые можно использовать.

2.3 autopkgtest: Автоматическое тестирование пакетов

Спецификация DEP 8 определяет, как можно легко интегрировать автоматическое тестирование в ваши пакеты. Для этого, необходимо:

- добавить файл `debian/tests/control`, который определяет требования к тестовому окружению,
- добавить тесты в `debian/tests`.

2.3.1 Требования к тестовому окружению

В файле `debian/tests/control` вы можете определить требования к тестовому окружению. Например, если тесты не проходят при сборке, или требуются права `root` – вы перечисляете необходимые для тестов пакеты. В Спецификации DEP 8 вы найдёте все доступные опции.

Ниже мы рассмотрим пакет исходного кода `glib2.0`. В очень простом случае он будет выглядеть так:

```
Tests: build
Depends: libglib2.0-dev, build-essential
```

Это будет означать, что для теста `debian/tests/build` требуются пакеты `libglib2.0-dev` и `build-essential`.

Примечание: В поле `Depends` можно указать `@`, если вы хотите установки всех бинарных пакетов, собранных из рассматриваемого пакета исходного кода.

2.3.2 Настоящие тесты

Тест, соответствующий рассмотренному выше примеру, будет выглядеть так:

```
#!/bin/sh
# autopkgtest check: Build and run a program against glib, to verify that the
# headers and pkg-config file are installed correctly
# (C) 2012 Canonical Ltd.
```



```
# Author: Martin Pitt <martin.pitt@ubuntu.com>

set -e

WORKDIR=$(mktemp -d)
trap "rm -rf $WORKDIR" 0 INT QUIT ABRT PIPE TERM
cd $WORKDIR
cat <<EOF > glibtest.c
#include <glib.h>

int main()
{
    g_assert_cmpint (g_strcmp0 (NULL, "hello"), ==, -1);
    g_assert_cmpstr (g_find_program_in_path ("bash"), ==, "/bin/bash");
    return 0;
}
EOF

gcc -o glibtest glibtest.c `pkg-config --cflags --libs glib-2.0`
echo "build: OK"
[ -x glibtest ]
./glibtest
echo "run: OK"
```

Здесь небольшая программа на языке C копируется во временную директорию, затем компилируется с использованием системных библиотек (с использованием флагов и путей к библиотекам, определённых через `pkg-config`). Затем запускается скомпилированный файл, который запускает несколько основных функций glib.

Хотя этот тест очень маленький и простой, он проверяет многое: что ваш -dev пакет имеет все необходимые зависимости, что ваш пакет устанавливает рабочие файлы pkg-config, заголовочные файлы и библиотеки помещаются в нужное место, или что компилятор и компоновщик работают. Это помогает обнаружить критические ошибки на ранней стадии.

2.3.3 Выполнение теста

While the test script can be easily executed on its own, it is strongly recommended to actually use `autopkgtest` from the `autopkgtest` package for verifying that your test works; otherwise, if it fails in the Ubuntu Continuous Integration (CI) system, it will not land in Ubuntu. This also avoids cluttering your workstation with test packages or test configuration if the test does something more intrusive than the simple example above.

The README.running-tests (online version) documentation explains all available testbeds (schroot, LXD, QEMU, etc.) and the most common scenarios how to run your tests with `autopkgtest`, e. g. with locally built binaries, locally modified tests, etc.

Система непрерывной интеграции Ubuntu CI использует эмулятор QEMU и запускает тесты из пакетов в архиве, с включённом флагом `-proposed`. Чтобы вручную получить то же окружение, сначала необходимо установить следующие пакеты:

```
sudo apt-get install autopkgtest qemu-system qemu-utils
```

Теперь выполните сборку тестового окружения, выполнив следующее:

```
autopkgtest-buildvm-ubuntu-cloud -v
```

(Более подробно о выборе других релизов, архитектур, целевых директорий, и об использовании прокси – в manpage и в выводе опции `--help`). Эта команда выполнит сборку, например

`adt-trusty-amd64-cloud.img`.

Теперь запустите тесты исходного пакета, например `libpng`, в образе QEMU:

```
autopkgtest libpng --- qemu adt-trusty-amd64-cloud.img
```

The Ubuntu CI system runs packages with only selected packages from `-proposed` available (the package which caused the test to be run); to enable that, run:

```
autopkgtest libpng -U --apt-pocket=proposed=src:foo --- qemu adt-release-amd64-cloud.img
```

or to run with all packages from `-proposed`:

```
autopkgtest libpng -U --apt-pocket=proposed --- qemu adt-release-amd64-cloud.img
```

The `autopkgtest` manpage has a lot more valuable information on other testing options.

2.3.4 Дальнейшие примеры

Этот список не полон, но может помочь вам получить представление о том, как автоматические тесты реализованы, и как они используются в Ubuntu.

- Для библиотеки libxml2 , тесты очень похожи. Они так же запускают тестовую сборку простого кода на C и выполняют его.
- Тесты пакета gtk+3.0 также компилируют/линкуют/запускают проверку в тесте “build”. Также есть дополнительный тест “python3-gi”, проверяющий что библиотека GTK может быть использована из языка Python.
- В пакете ubiquity tests используется набор тестов родительского пакета
- Пакет gvfs tests – очень интересный пример: он тестирует свой функционал “по полной”, включая эмуляцию CD, Samba, DAV и прочих компонентов.

2.3.5 Инфраструктура Ubuntu

Пакеты с включённым `autopkgtest` будут тестироваться при выгрузке, или если обновятся какие-либо зависимости. Результат работы автоматического запуска тестов autopkgtest может быть просмотрен на сайте, и он регулярно обновляется.

Debian also uses `autopkgtest` to run package tests, although currently only in schroots, so results may vary a bit. Results and logs can be seen on http://ci.debian.net. So please submit any test fixes or new tests to Debian as well.

2.3.6 Добавление теста в Ubuntu

Процесс добавления и отправки autopkgtest-теста для пакетов очень похож на *процесс исправления ошибок в Ubuntu*. Достаточно лишь:

- выполните `bzr branch ubuntu:<имя_пакета>`,
- включите тесты в `debian/control`,
- создайте директорию `debian/tests`,
- создайте `debian/tests/control`, основываясь на Спецификации DEP 8,
- добавьте ваши тесты в `debian/tests`,

- закоммитьте ваши изменения, отправьте их на Launchpad, предложите merge и дождитесь его рассмотрения и дождаться, – в точности как с любыми другими улучшениями в исходных пакетах.

2.3.7 Чем вы можете помочь

Команда Ubuntu Engineering собрала список необходимых тестов, в котором пакеты, нуждающиеся в тестировании, распределены по категориям. Там можно найти примеры таких тестов и взять их на себя.

Если вы столкнётесь с проблемами, присоединяйтесь к IRC на канал #ubuntu-quality IRC channel: разработчики наверняка вам помогут.

2.4 Получение исходного кода

2.4.1 URL пакетов исходного кода

Bazaar предоставляет несколько очень удобных сокращений для доступа к веткам исходного кода с Launchpad для пакетов как Ubuntu, так и Debian.

Чтобы сослаться на ветки исходного кода, используйте:

```
ubuntu:package
```

где *package* — имя пакета, который вам нужен. Этот URL ссылается на пакеты в текущей разрабатываемой версии Ubuntu. Чтобы сослаться на ветку Tomboy в разрабатываемой версии, нужно использовать:

```
ubuntu:tomboy
```

Чтобы сделать отсылку к версии исходного пакета в более старом релизе Ubuntu просто добавьте пакету префикс с кодовым именем релиза. Например, для отсылки к исходному пакету Tomboy в Saucy используйте:

```
ubuntu:saucy/tomboy
```

Поскольку первые буквы кодовых имён не повторяются, можно сократить имя выпуска:

```
ubuntu:s/tomboy
```

Похожую схему можно использовать для доступа к веткам исходного кода в Debian, хотя здесь нет сокращений для имён выпусков Debian. Чтобы получить доступ к ветке Tomboy в текущем разрабатываемом выпуске Debian, используйте:

```
debianlp:tomboy
```

также для доступа к Tomboy в Debian Wheezy используйте:

```
debianlp:wheezy/tomboy
```

2.4.2 Получение исходного кода

Каждый пакет исходного кода в Ubuntu связан с веткой исходного кода на Launchpad. Launchpad автоматически обновляет эти ветки исходного кода, хотя процесс не полностью «защищён от дурака».

Есть несколько вещей, которые мы сделаем в первую очередь, чтобы сделать рабочий процесс более эффективным впоследствии. После того, как вы освоите процесс, вы узнаете, когда имеет смысл пропускать эти этапы.

Создание общедоступного хранилища

К примеру, Вы хотите работать над пакетом Tomboy, и Вы уже проверили, что исходный пакет называется `tomboy`. Перед фактическим ветвлением кода для Tomboy создайте общедоступный репозиторий для хранения ветвей этого пакета. Общедоступный репозиторий сделает будущую работу более эффективной.

Воспользуйтесь для этого командой *bzr init-repo*, передав ей имя каталога, который вы хотите использовать:

```
$ bzr init-repo tomboy
```

Вы увидите, что в вашей текущей рабочей области создан каталог *tomboy*. Перейдите в него для продолжения работы:

```
$ cd tomboy
```

Получение ветки trunk

Мы используем команду *bzr branch* для создания локальной ветки пакета. Целевой каталог назовём *tomboy.dev*, просто потому, что так легче запомнить:

```
$ bzr branch ubuntu:tomboy tomboy.dev
```

Каталог tomboy.dev представляет собой версию Tomboy в разрабатываемой версии Ubuntu, и вы всегда можете перейти в этот каталог и выполнить *bzr pull* для получения любых будущих обновлений.

Проверка актуальности версии

Когда Вы делаете свою `bzr branch`, то получите сообщение о том является ли ветвь пакетов актуальной. К примеру:

```
$ bzr branch ubuntu:tomboy
Most recent Ubuntu version: 1.8.0-1ubuntu1.2
Packaging branch status: CURRENT
Branched 86 revisions.
```

Иногда импорт не проходит успешно и ветви пакета не совпадают с теми, что находятся в архиве. Сообщение:

```
Packaging branch status: OUT-OF-DATE
```

означает, что импорт не удался. Вы можете узнать причину по ссылке: http://package-import.ubuntu.com/status/ и отправить баг в UDD для разрешение проблемы.

Tar-файл из апстрима

Получить tar из апстрима можно с помощью:


```
bzr get-orig-source
```

Таким образом пробуются несколько методов для попадания в tar апстрима, сначала воссоздавая его из тега `upstream-x.y` в архиве bzr, затем скачивая из архива Ubuntu, а потом запуская `debian/rules get-orig-source`. Tar апстрима также будет воссоздан при использовании bzr для построения пакета:

```
bzr builddeb
```

У плагина *builddeb* есть несколько опций конфигурации.

Получение ветки для определённого выпуска

Если Вы хотите сделать что-то вроде обновления стабильного релиза (SRU), либо просто хотите изучить код в старом релизе, Вам нужно выбрать ветвь, соответствующую определенному релизу Ubuntu. К примеру, чтобы получить пакет Tomboy для Quantal:

```
$ bzr branch ubuntu:m/tomboy quantal
```

Импорт пакета исходного кода Debian

Если пакет, над которым Вы хотите работать, доступен в Debian, но не в Ubuntu - код легко импортировать в локальную ветку bzr для разработки. К примеру, Вы хотите импортировать исходный пакет *newpackage*. Мы начнем с создания общедоступного репозитория в качестве обычного, но нам также надо создать рабочее дерево, в которое будет импортирован исходный пакет (не забудьте выполнить cd out директории *tomboy*, созданной выше):

```
$ bzr init-repo newpackage
$ cd newpackage
$ bzr init debian
$ cd debian
$ bzr import-dsc http://ftp.de.debian.org/debian/pool/main/n/newpackage/newpackage_1.0-1.dsc
```

Как Вы видите - нужно просто указать удаленное расположение файла dsc, а Bazaar сделает все остальное. Теперь у Вас есть исходная ветка Bazaar.

2.5 Работа с пакетом

Как только у Вас есть ветка с исходным пакетом в общедоступном репозитории, Вы захотите создать дополнительные ветки для фиксов или другой запланированной работы. Вы захотите, чтобы Ваша ветка основывалась на пакете исходной ветки релиза distro, куда Вы планируете загружать. Обычно это текущий релиз разработки, но это могут быть и более старые релизы, если Вы, к примеру, выполняете обратный порт на SRU.

2.5.1 Ветвление для изменений

Первым делом убедитесь, что ветка исходного пакета актуальна. Если Вы ее только что проверили, то она будет актуальной, если же нет, то сделайте следующее:

```
$ cd tomboy.dev
$ bzr pull
```


Будут показаны любые обновления касательно пакета, которые были загружены с момента отладки. Вы не хотите вносить изменения в эту ветку. Вместо этого создайте ветку, которая будет содержать только те изменения, которые Вы собираетесь внести. Предположим, Вы хотите исправить баг 12345 для проекта Tomboy. Когда Вы находитесь в общедоступном репозитории, ранее созданном для Tomboy, Вы можете создать ветку для исправления багов следующим образом:

```
$ bzr branch tomboy.dev bug-12345
$ cd bug-12345
```

Теперь Вы можете выполнять всю работу в директории `bug-12345`. Делайте там все необходимые изменения, не забывая по ходу дела отправлять свою работу. Этот процесс схож с разработкой любых приложений при помощи Bazaar. Можно делать промежуточные отправки так часто, как захотите, а когда Вы закончили работу над изменениями, используйте стандартную команду `dch` (из пакета `devscripts`):

```
$ dch -i
```

Эта команда откроет редактор и добавит запись в файл *debian/changelog*. При добавлении записи в `debian/changelog` вы должны включить тег исправления ошибки, который указывает, для какого сообщения об ошибке на Launchpad вы создали исправление. Этот текстовый тег имеет вполне определённый формат: `LP: #12345`. Пробел между `:` и `#` обязателен и, разумеется, вы должны указать номер реально существующей ошибке, которую вы исправляете. Ваш `debian/changelog` может выглядеть примерно так:

```
tomboy (1.12.0-1ubuntu3) trusty; urgency=low

  * Don't fubar the frobnicator. (LP: #12345)

 -- Bob Dobbs <subgenius@example.com>  Mon, 10 Sep 2013 16:10:01 -0500
```

Подтвердить с нормальным:

```
bzr commit
```

Хук в bzr-builddeb будет использовать текст debian/changelog как сообщение о завершении отправки и установит тег для отметки бага #12345 в качестве фиксированного.

Это работает только с bzr-builddeb 2.7.5 и bzr 2.4, для более старых версий используйте `debcommit`.

2.5.2 Сборка пакета

По ходу дела Вы захотите создать свою ветку, чтобы Вы могли проверить ее и убедиться, что она действительно исправляет баг.

Чтобы создать пакет, Вы можете использовать команду `bzr builddeb` из пакета `bzr-builddeb`. Вы можете создать исходный пакет при помощи:

```
$ bzr builddeb -S
```

(`bd` — это псевдоним для `builddeb`.) Вы можете оставить пакет неподписанным, добавив к команде `--uc -us`.

Вы можете также использовать свои обычные инструменты, если они способны убирать каталоги `.bzr` из пакета, например:

```
$ debuild -i -I
```

Если Вы когда-либо столкнетесь с ошибкой, связанной с попыткой составить родной пакет без tar-архива, убедитесь что там есть файл `.bzr-builddeb/default.conf`, ошибочно выдающий пакет за

родной. Если в версии лога изменений есть тире, то это не родной пакет, поэтому файл конфигурации следует убрать. Обратите внимание, если у `bzr builddeb` есть оператор `--native`, то у него нет оператора `--no-native`.

После того, как вы получили пакет исходного кода, можно собрать его как обычно, с помощью `pbuilder-dist` (или `pbuilder`, или sbuild).

2.6 Поиск Обзоров и Спонсорства

Один из больших плюсов использования рабочего процесса UDD - улучшение качества путем отзывов об изменениях от Ваших друзей и коллег. Это работает вне зависимости от того есть ли у Вас права на загрузку. Конечно, если у Вас нет прав на загрузку, то Вам нужно найти спонсора.

Как только Вы довольны своим фиксом и Ваша ветка готова к работе, следует предпринять следующие шаги для публикации своей ветки на Launchpad, привязки ее к багу, а затем создать предложение о слиянии (*merge proposal*) для рассмотрения другими пользователями и возможностью загрузки его спонсорами.

2.6.1 Выгрузка на Launchpad

Ранее мы показали вам, как *связать вашу ветку с ошибкой*, используя команды `dch` и `bzr commit`. Однако ветка и ошибка в действительности не будут связаны, пока вы не выгрузите ветку на Launchpad командой push.

Создавать ссылку на ошибку для каждого вашего изменения не обязательно, но если вы исправляете ошибки, для которых имеется отчёт об ошибке, то ссылки на этот отчёт могут быть полезны.

Общий формат URL, на который вы должны выгрузить свою ветку, имеет следующий вид:

```
lp:~<user-id>/ubuntu/<distroseries>/<package>/<branch-name>
```

Например, чтобы продвинуть свой фикс для бага 12345 в пакете Tomboy для Trusty, используйте:

```
$ bzr push lp:~subgenius/ubuntu/trusty/tomboy/bug-12345
```

Последний компонент пути в этом примере выбран произвольно, укажите вместо него реально существующую ошибку.

Но обычно этого недостаточно, чтобы разработчики Ubuntu проверили ваше изменение и взяли на себя поручительство над ним. Вам нужно отправить *предложение слияния* (merge proposal).

Для этого откройте страницу ошибки в браузере, например:

```
$ bzr lp-open
```

Если сделать это не удалось, вы можете использовать:

```
$ xdg-open https://code.launchpad.net/~subgenius/ubuntu/trusty/tomboy/bug-12345
```

где большая часть URL совпадает с тем URL, который вы указывали в команде *bzr push*. На этой странице вы увидите ссылку *Propose for merging into another branch*. Наберите описание вашего изменения в поле *Initial Comment*. Затем щёлкните *Propose Merge*, чтобы завершить процесс.

Предложения о слиянии для исходных веток автоматически подпишут команду *~ubuntu-branches*, что должно быть достаточно для того, чтобы привлечь внимание разработчика Ubuntu, который сможет дать отзыв Вашим изменениям и стать спонсором.

2.6.2 Создание debdiff

Как было указано выше, некоторые поручители предпочитают проверять *debdiff*, прикреплённый к отчёту об ошибке, вместо предложения слияния. Если вас попросили включить debdiff, вы можете создать его следующим образом (из вашей ветки *bug-12345*):

```
$ bzr diff -rbranch:../tomboy.dev
```

Другой способ — открыть предложение слияния и загрузить diff.

Вы должны убедиться, что diff-файл содержит ожидаемые вами изменения, не больше и не меньше. Дайте ему подходящее имя, например, `foobar-12345.debdiff` и прикрепите к отчёту об ошибке.

2.6.3 Работа с обратной связью от поручителей

Если поручитель проверяет вашу ветку и просит вас что-то изменить, то сделать это очень легко. Просто перейдите в ветку, над которой вы работали, сделайте требуемые изменения и выполните фиксацию:

```
$ bzr commit
```

Теперь когда Вы продвинули свою ветку в Launchpad, Bazaar обновит ветку на Launchpad Вашими последними изменениями. Все, что Вам нужно сделать:

```
$ bzr push
```

Вы можете ответить на email с обзором предложения о слиянии объяснение того, какие изменения Вы внесли, а также можете попросить провести повторный обзор; или же Вы можете отправить свой ответ через Launchpad.

Обратите внимание - Вас спонсируют через debdiff, прикрепленный к отчету о баге - Вам следует обновлять его вручную, генерируя новый diff и прикрепляя его у отчету о баге.

2.6.4 Ожидание

Разработчики Ubuntu составили расписание людей (так называемых “patch pilots”), которые регулярно проверяют очередь поручительства и предоставляют обратную связь по вопросам работы с ветками и патчами. Но, даже несмотря на это, может пройти несколько дней, пока вы получите ответ. Это зависит от их занятости, от состояния заморозки текущей версии и других факторов.

Если вы не получаете ответа в течение долгого времени, подключитесь к каналу *#ubuntu-devel* на *irc.freenode.net* и найдите кого-нибудь, кто может вам помочь.

Чтобы узнать больше о процессе поручительства, прочтите также нашу вики-документацию: https://wiki.ubuntu.com/SponsorshipProcess

2.7 Загрузка пакета

Как только Ваше предложение о слиянии рассмотрено и подтверждено, Вы захотите загрузить свой пакет либо в архив (если у Вас есть права), либо в свой Персональный Архив Пакетов Personal Package Archive (PPA). Также возможно Вы захотите выполнить загрузку, если спонсируете изменения, внесенные другим пользователем.

2.7.1 Выгрузка изменений, сделанных вами

Когда у Вас есть ветка с изменениями, которые Вы хотите загрузить, то нужно отправить это изменение обратно на исходную ветку, создать исходный пакет, а затем загрузить его.

Сначала Вам нужно убедиться, что у Вас самая последняя версия пакета в отладке древа пакета разработки:

```
$ cd tomboy/tomboy.dev
$ bzr pull
```

Это применит любые изменения, которые были внесены за время Вашей работы над фиксом. Начиная с этого момента у Вас есть несколько вариантов. Если Ваши изменения большие и Вы чувствуете, что их следует протестировать вместе с Вашими изменениями - то можно объединить их в ветке исправления багов и провести тестирование там. Если нет, то Вы можете продолжить процесс слияния в Вашей ветке исправления бага. Касательно bzr 2.5 и bzr-builddeb 2.8.1, это работает так же как и стандартная команда `merge`:

```
$ bzr merge ../bug-12345
```

Для более старых версий bzr можно использовать взамен команду `merge-package`:

```
$ bzr merge-package ../bug-12345
```

Эта команда сольёт два дерева и, возможно, сообщит о конфликтах, которые вам надо будет разрешить вручную.

Далее следует убедиться в правильности содержимого `debian/changelog`, то есть, что там правильно указан дистрибутив, номер версии и т.п.

Как только это сделано, Вы должны еще раз перепроверить изменения, которые хотите отправить, при помощи `bzr diff`. Это должно показать те же изменения, как показал бы debdiff до загрузки исходного пакета.

Следующий шаг — собрать и протестировать изменённый пакет исходного кода, как вы это обычно делаете:

```
$ bzr builddeb -S
```

Когда Вы наконец довольны своей веткой, убедитесь что отправили все изменения, а затем пометьте ветку номером версии лога изменений. Команда `bzr tag` сделает это автоматически, если не указано ни одного аргумента:

```
$ bzr tag
```

Этот тег сообщит импортирующему пакет, что содержимое ветка Bazaar идентично содержимому архива.

Теперь вы можете выгрузить командой push изменения обратно на Launchpad:

```
$ bzr push ubuntu:tomboy
```

(Измените место назначения, если вы выгружаете SRU или что-то подобное.)

Вам нужен один последний шаг, чтобы отправить свои изменения в Ubuntu или ваш PPA: вам нужно загрузить с помощью `dput` пакет исходного кода в соответствующее место. Например, чтобы загрузить изменения в PPA, сделайте следующее:

```
$ dput ppa:imasponsor/myppa tomboy_1.5.2-1ubuntu5_source.changes
```


или, если у вас есть права загрузки в основной архив:

```
$ dput tomboy_1.5.2-1ubuntu5_source.changes
```

Теперь Вы можете спокойно удалить ветвь, так как она уже объединена, и при необходимости ее можно заново скачать с Launchpad.

2.7.2 Поручительство над изменением

Поручительство над чьим-нибудь изменением похоже на вышеописанную процедуру, но вместо слияния из созданной вами ветки вы выполняете слияние из ветки, имеющейся в предложении слияния:

```
$ bzr merge lp:~subgenius/ubuntu/trusty/tomboy/bug-12345
```

Если при слиянии возникает множество конфликтов, возможно Вы захотите попросить разработчика их исправить. Смотрите в следующем разделе как отменить запланированное слияние.

Но если с изменениями все в порядке - подтвердите, а затем пройдите оставшуюся часть процесса загрузки:

```
$ bzr commit --author "Bob Dobbs <subgenius@example.com>"
```

2.7.3 Отмена выгрузки

В любое время до выполнения действия *dput* с исходным пакетом Вы можете отменить загрузку и откатить все изменения:

```
$ bzr revert
```

Вы можете сделать это если заметите, что требуется еще немного работы, либо если хотите попросить разработчика исправить конфликты (если выступаете его спонсором).

2.7.4 Поручительство над чем-нибудь и внесение своих собственных изменений

Если вы являетесь поручителем над чьей-нибудь работой, но хотите дополнить её несколькими собственными изменениями, то вы можете сначала выполнить слияние их работы в отдельную ветку.

Если у Вас уже есть ветка, в которой Вы работаете над пакетом и Вы хотите включить все изменения - просто запустите `bzr merge` из этой ветки, вместо отладки пакета разработки. Затем Вы можете внести изменения и отправить, а затем продолжить работу с изменениями к пакету.

Если у Вас нет существующей ветки, но Вы знаете, что хотели бы внести изменения, основываюсь на данных разработчика, то Вы должны первым делом должны спарсить их ветку:

```
$ bzr branch lp:~subgenius/ubuntu/trusty/tomboy/bug-12345
```

затем работайте в этой новой ветке, а после этого выполните ее слияние с главной и загрузите таким образом, как если бы загружали собственную работу. Разработчик-участник все еще будет упомянут в логах изменений, и Bazaar должным образом назначит им внесенные ими изменения.

2.8 Получение последних изменений

Если кто-нибудь ещё внёс изменения в пакет, вам может понадобиться получить эти изменения в ваши копии веток пакета.

2.8.1 Обновление вашей основной ветки

Обновить вашу копию ветки, соответствующей пакету в определённом выпуске, очень просто: выполните *bzr pull* из соответствующего каталога:

```
$ cd tomboy/tomboy.dev
$ bzr pull
```

Это работает если у Вас есть отладка ветки, поэтому его можно применить для таких вещей как ветки *saucy*, *trusty-proposed*, итп.

2.8.2 Получение последних изменений в ваши рабочие ветки

Как только Вы обновили свою копию ветки distroseries, то возможно захотите также объединить ее со своими рабочими ветками, чтобы они работали на самом последнем коде.

Тем не менее, Вам не нужно делать это каждый раз. Вы можете без проблем работать и со слегка старым кодом. Недостатки могут всплыть если Вы работали над кодом, который изменил кто-то еще. Если Вы работаете не с самой последней версией, Ваши изменения могут быть некорректными, и даже могут стать причиной конфликта.

Слияние следует выполнять в определенный момент. Чем дольше Вы работаете - тем сложнее может быть процесс в дальнейшем. Выполняйте слияние регулярно, чтобы максимально упростить процесс. Если даже слияний много, в итоге нужно применять меньше общих усилий.

Чтобы выполнить слияние изменений, Вам нужно использовать `bzr merge`, но сначала Вы должны отправить свою текущую работу:

```
$ cd tomboy/bug-12345
$ bzr merge ../tomboy.dev
```

О любых конфликтах будет сообщаться. так что Вы сможете их исправить в дальнейшем. Чтобы рассмотреть изменения, используйте `bzr diff`. Как только Вы закончили работу - используйте `bzr commit`.

2.8.3 Относительно версий пакета

Вы часто будете думать относительно версий пакета, а не просто о цифрах предыдущих поправок в Bazaar. *bzr-builddeb* для удобства предоставляет спецификатор поправок. Любая команда, использующая аргумент `-r` для указаний поправки или диапазона поправок. будет работать с этим спецификатором, к примеру, `bzr log`, `bzr diff‘‘итд. Чтобы просмотреть версии пакета, используйте спецификатор ‘‘package::`

```
$ bzr diff -r package:0.1-1..package:0.1-2
```

Эта команда покажет вам различия между версиями пакета 0.1-1 и 0.1-2.

2.9 Слияние — обновления из Debian и апстрима

Слияние — это одна из сильнейших сторон Bazaar, и мы часто выполняем его в процессе разработки Ubuntu. Слияние может быть выполнено с обновлениями из Debian, из нового выпуска в апстриме и от

других разработчиков Ubuntu. Сделать это в Bazaar очень просто, всё основано на команде `bzr merge` [1].

Находясь в рабочем каталоге любой ветки, вы можете выполнить слияние изменений из ветки в другом месте. Сначала проверьте, что у вас нет незафиксированных изменений:

```
$ bzr status
```

Если появится отчет, то Вам нужно либо применить изменения, откатить их, либо отложить решения (и возвратиться к их решению позже).

2.9.1 Слияние из Debian

Затем запустите `bzr merge`, передавай URL ветки для слияния. К примеру, чтобы выполнить слияние версий пакета в Debian Unstable запустите [2]:

```
$ bzr merge lp:debian/tomboy
```

Это добавит изменения, внесенные с момента последнего слияния и даст Вам все изменения для обзора. Это может повлечь конфликты. Вы можете увидеть все действия, выполненные командой `merge`, запустив:

```
$ bzr status
$ bzr diff
```

Если появится отчет о конфликтах, Вам нужно отредактировать соответствующие файлы, чтобы привести их к должному виду, убрав конфликтующие маркеры (*conflict markers*). Как только Вы это сделали, выполните:

```
$ bzr resolve
$ bzr conflicts
```

Это решит проблему с конфликтующими файлами, которые Вы исправляли, и сообщит что еще Вам нужно сделать.

После того, как все конфликты разрешены, и вы внесли все другие необходимые изменения, нужно добавить новую запись в changelog и выполнить фиксацию:

```
$ dch -i
$ bzr commit
```

как было описано выше.

Однако перед фиксацией всегда желательно проверить все изменения, сделанные в Ubuntu, выполнив:

```
$ bzr diff -r tag:0.6.10-5
```

Эта команда покажет различия между версиями в Debian (0.6.10-5) и Ubuntu (0.6.10-5ubuntu1). Подобным же способом вы можете выполнить сравнение с любыми другими версиями. Чтобы увидеть все доступные версии, выполните:

```
$ bzr tags
```

[1] Для работы с командой `merge` вам понадобятся более новые версии `bzr` и `bzr-builddeb`. Используйте версии из Ubuntu 12.04 (Precise) или разрабатываемые версии из PPA `bzr`. Точнее говоря, вам понадобится `bzr` версии 2.5 beta 5 или более новой, а также `bzr-builddeb` версии 2.8.1 или более новой. Для старых версий используйте взамен команду `bzr merge-package`.

[2] Чтобы проверить наличие других веток пакета в Debian, см. страницу кода пакета. Например: https://code.launchpad.net/debian/+source/tomboy

После проверки и фиксации слияния, вам нужно найти попечителя или выгрузить пакет в архив обычным способом.

Если Вы собираетесь создать исходный пакет из этой объединенной ветки, то нужно использовать опцию `-S` команды `bd`. Также Вам захочется рассмотреть использование опции `--package-merge`. Это добавит соответствующие опции `-v` и `-sa` к исходному пакету, чтобы все записи в логе изменений после последних изменений в Ubuntu были включены в Ваш файл `_source.changes`. Например:

```
$ bzr builddeb -S --package-merge
```

2.9.2 Слияние с новой версией из апстрима

Когда в апстриме выпускается новая версия (или Вы хотите запаковать скриншот), Вам нужно выполнить слияние tar-архива и Вашей ветки.

Это делается командой `bzr merge-upstream`. Если в вашем пакете имеется файл `debian/watch` с правильным содержимым, то из каталога ветки, в которую вы собираетесь слить изменения, просто наберите:

```
$ bzr merge-upstream
```

Команда загрузит тарбол и сольёт его в вашу ветку, автоматически добавив запись в `debian/changelog`. `bzr-builddeb` просматривает файл `debian/watch`, чтобы определить местоположение тарбола из апстрима.

Если у вас *отсутствует* файл `debian/watch`, то вам нужно вручную указать местоположение тарбола из апстрима и версию:

```
$ bzr merge-upstream --version 1.2 http://example.org/releases/foo-1.2.tar.gz
```

Опция `--version` используется для указания версии апстрима, из которой выполняется слияние, поскольку команда не способна (пока) узнать её самостоятельно.

Последний параметр — это местоположение тарбола, на который выполняется обновление: это может быть путь в локальной файловой системе, http, ftp, sftp или другой URI, как показано в примере. Команда автоматически загрузит тарбол для вас, переименует его соответствующим образом и, если требуется, преобразует в `.gz`.

Команда `merge-upstream` сообщит либо о своём удачном завершении, либо о том, что имеются конфликты. В любом случае у вас будет возможность проверить изменения перед их фиксацией обычным способом.

Если Вы объединяете релиз апстрима с существующей веткой Bazaar, в которой еще не использовалась разметка UDD, `bzr merge-upstream` пройдет неудачно и с ошибкой, что тег предыдущих версий апстрима недоступен. Слияние нельзя выполнить без знания базовой версии для слияния. Для работы с этим, создайте тег в своем существующем репозитории для последней имеющейся там версии апстрима; например, если последний релиз Ubuntu был *1.1-0ubuntu3*, создайте тег *upstream-1.1*, указывая на изменение bzr, которое Вы хотите использовать как подсказку для ветки апстрима.

2.10 Использование chroot-окружений

Если вы пользуетесь одной из версий Ubuntu, но работаете над пакетами для другой версии, вы можете создать среду другой версии с помощью `chroot`.

Использование `chroot` позволит вам иметь в распоряжении полную файловую систему другого дистрибутива для удобства работы. Это позволяет избежать затрат, связанных с установкой виртуальной машины.

2.10.1 Создание chroot

Используйте команду `debootstrap`, чтобы создать новый chroot:

```
$ sudo debootstrap trusty trusty/
```

Это создаст папку `trusty` и установит минимальный образ trusty в неё.

Если ваша версия `debootstrap` не определит Trusty, попробуйте обновиться до версии в `backports`.

После этого вы можете работать внутри chroot:

```
$ sudo chroot trusty
```

Где можно установить или удалить любой пакет, который вы хотите, без ущерба для основной системы.

Вы можете скопировать свои ключи GPG и SSH, а также конфигурацию Bazaar в chroot, чтобы получать доступ и подписывать пакеты непосредственно оттуда:

```
$ sudo mkdir trusty/home/<username>
$ sudo cp -r ~/.gnupg ~/.ssh ~/.bazaar trusty/home/<username>
```

Чтобы apt и другие программы не жаловались на отсутствующие локали, можно установить соответствующий языковой пакет:

```
$ apt-get install language-pack-en
```

Если вам нужно запускать программы, использующие X-сервер, вам нужно добавить в chroot директорию /tmp, для этого снаружи chroot запустите:

```
$ sudo mount -t none -o bind /tmp trusty/tmp
$ xhost +
```

Для некоторых программ, возможно, понадобится привязать /dev или /proc.

На странице Debootstrap Chroot вики вы найдёте более подробную информацию о chroot-окружении.

2.10.2 Альтернативы

SBuild – система, похожая на PBuilder, используемая для создания окружения, в котором выполняются тестовые сборки пакета. Она близка к той, которую использует Launchpad для сборки пакетов, но её установка несколько сложнее, чем PBuilder. Более полную информацию можно найти на викистранице Система Сборки Security Team.

Полные виртуальные машины могут быть полезны для создания пакетов и тестирования программ. TestDrive — это программа, позволяющая автоматизировать синхронизацию и запуск ежедневных ISO-образов. Подробнее смотрите wiki-страницу TestDrive.

Можно также настроить pbuilder так, чтобы он приостанавливался при обнаружении ошибки сборки. Скопируйте C10shell из /usr/share/doc/pbuilder/examples в каталог и используйте аргумент `--hookdir=`, чтобы указать на него.

Облачный сервис Amazon EC2 позволит вам приобрести компьютер в облаке, цена за который – всего несколько центов в час. Там вы можете установить Ubuntu любой поддерживаемой версии и работать

с пакетами удалённо, что очень удобно, если требуется сборка множества пакетов одновременно, или если нужно преодолеть медленную скорость Интернет-подключения.

2.11 Традиционные методы создания пакетов

Большая часть данного руководства относится к *Ubuntu Distributed Development* (UDD), которое использует распространяемую версию системы управления (DVCS) Bazaar для *получения источников пакетов* и отправки фиксов через *предложения о слияниях.* В этой статье мы обсудим так называемые "традиционные" методы создания пакетов. До того как Bazaar стали применять в разработках Ubuntu, помочь Ubuntu можно было стандартными способами.

В некоторых случаях вам может понадобиться использовать эти инструменты вместо UDD. Поэтому не помешает познакомиться с ними. Перед тем, как продолжить, вам следует прочитать статью *Подготовка.*

2.11.1 Получение исходного кода

Чтобы получить пакет исходного кода, можно просто набрать:

```
$ apt-get source <package_name>
```

Но у этого метода есть некоторые недостатки. Он скачивает версию исходного кода, которая доступна в **вашей системе.** Скорее всего, у вас установлен текущий стабильный выпуск, а вы собираетесь внести изменения в пакет в разрабатываемом выпуске. К счастью, пакет `ubuntu-dev-tools` предоставляет вспомогательный сценарий:

```
$ pull-lp-source <package_name>
```

По умолчанию будет загружена самая свежая версия из разрабатываемого выпуска. Можно также указать версию выпуска Ubuntu следующим образом:

```
$ pull-lp-source <package_name> trusty
```

чтобы вытащить источник из релиза `trusty`, либо:

```
$ pull-lp-source <package_name> 1.0-1ubuntu1
```

чтобы скачать версию пакета `1.0-1ubuntu1`. Для получения дополнительной информации о команде воспользуйтесь `man pull-lp-source`.

Для примера, давайте представим, что мы получили отчёт об ошибке, в котором говорится, что вместо "colour" в описании `xicc` должно быть "color," и мы хотим это исправить. *(Примечание: это просто пример чего-то, что можно изменить, а не реальная ошибка.)* Чтобы получить исходный код, введите:

```
$ pull-lp-source xicc 0.2-3
```

2.11.2 Создание Debdiff

Файл `debdiff` показывает различия между двумя пакетами Debian. Имя команды, используемой для его создания, тоже `debdiff`. Она является частью пакета `devscripts`. Смотрите `man debdiff` для подробной информации о ней. Чтобы сравнить два пакета исходного кода, передайте команде в качестве аргумента два файла `dsc`:


```
$ debdiff <package_name>_1.0-1.dsc <package_name>_1.0-1ubuntu1.dsc
```

Чтобы продолжить наш пример, давайте отредактируем `debian/control` и «исправим» нашу «ошибку»:

```
$ cd xicc-0.2
$ sed -i 's/colour/color/g' debian/control
```

Мы также должны соблюдать Спецификации Обслуживания Debian<MaintFieldSpec_> и редактировать `debian/control` для замены:

```
Maintainer: Ross Burton <ross@debian.org>
```

на:

```
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
XSBC-Original-Maintainer: Ross Burton <ross@debian.org>
```

Для этого можно воспользоваться инструментом `update-maintainer` (из пакета `ubuntu-dev-tools`).

Не забудьте задокументировать ваши изменения в `debian/changelog` с помощью `dch -i`, после чего мы можем сгенерировать новый пакет исходного кода:

```
$ debuild -S
```

Теперь можно проверить наши изменения с помощью `debdiff`:

```
$ cd ..
$ debdiff xicc_0.2-3.dsc xicc_0.2-3ubuntu1.dsc | less
```

Чтобы создать файл патча, который можно отправить другим или приложить к отчёту об ошибке для одобрения, выполните:

```
$ debdiff xicc_0.2-3.dsc xicc_0.2-3ubuntu1.dsc > xicc_0.2-3ubuntu1.debdiff
```

2.11.3 Применение Debdiff

Чтобы применить debdiff, нужно иметь исходный код версии, на основе которой он был создан:

```
$ pull-lp-source xicc 0.2-3
```

Затем в терминале измените на директорию, куда был распакован исходник:

```
$ cd xicc-0.2
```

Фактически, debdiff похож на обычный файл патча. Примените его, как обычно, с помощью:

```
$ patch -p1 < ../xicc_0.2.2ubuntu1.debdiff
```

2.12 Работа с пакетами KDE

Созданием пакетов программ KDE в Ubuntu занимаются команды Kubuntu и MOTU. Связаться с командой Kubuntu можно через почтовую рассылку Kubuntu и канал Freenode IRC `#kubuntu-devel`. Дополнительная информация о разработке Kubuntu доступна на wiki-странице Kubuntu.

При создании пакетов мы учитываем практику команд Debian Qt/KDE и Debian KDE Extras. Большинство наших пакетов являются производными от пакетов, созданных этими командами Debian.

2.12.1 Политика создания патчей

Kubuntu добавляет патчи в приложения KDE, только если они исходят от оригинальных авторов, либо если они были отправлены в upstream и есть уверенность в их скором принятии, либо если мы обсудили проблемы с разработчиками KDE.

Kubuntu не меняет фирменного оформления пакетов, кроме случаев, когда этого требует апстрим (например, логотип в верхнем левом углу меню Kickoff) или для упрощения (например, удаление показываемых при запуске заставок).

2.12.2 debian/rules

Пакеты Debian включают некоторые дополнения к обычному использованию Debhelper. Они хранятся в пакете `pkg-kde-tools`.

Пакеты, использующие Debhelper 7, должны добавить опцию `--with=kde`. Это позволит убедиться в использовании правильных флагов сборки и опций, таких как обработка заданий kdeinit и файлов перевода.

```
%:
    dh $@ --with=kde
```

Некоторые новые пакеты KDE используют систему `dhmk`, альтернативу `dh`, созданную командой Debian Qt/KDE. Прочесть о ней можно в /usr/share/pkg-kde-tools/qt-kde-team/2/README. Использующие её пакеты будут `включать /usr/share/pkg-kde-tools/qt-kde-team/2/debian-qt-kde.mk` вместо запуска `dh`.

2.12.3 Переводы

Переводы пакетов из репозитория main импортируются на Launchpad и экспортируются с Launchpad в языковые пакеты Ubuntu.

Итак, каждый KDE-пакет в main должен создавать шаблоны переводов, включать оригинальные переводы и обрабатывать переводимые строки в `.desktop`-файлах.

Для генерации шаблонов переводов пакет должен содержать файл `Messages.sh`; если его там нет, обратитесь в апстрим. Проверить его работу можно, выполнив сценарий `extract-messages.sh`, который должен создать один или несколько файлов `.pot` в каталоге `po/`. В процессе сборки это будет сделано автоматически, если вы используете опцию `--with=kde` для `dh`.

Апстрим обычно также помещает файлы переводов `.po` в каталог `po/`. Если их там нет, проверьте, не выделены ли они в один из отдельных языковых пакетов апстрима, например, в языковые пакеты KDE SC. Если они в отдельном языковом пакете, на Launchpad необходимо собирать их вместе вручную. Проконсультируйтесь по этому поводу с Дэвидом Планеллой (David Planella).

Если пакет перемещён из universe в main, его следует перевыгрузить перед импортом переводов на Launchpad.

Файлы `.desktop` также требуют перевода. Мы добавили патч к KDELibs для чтения переводов из `.po`-файлов, указывающих на строку `X-Ubuntu-Gettext-Domain=`, добавляемую к файлам `.desktop` во время сборки пакета. Файл .pot для каждого пакета генерируется во время сборки и .po-файлы необходимо скачать из апстрима и включить в пакет или в наши языковые пакеты. Список .po-файлов, которые нужно скачать из репозиториев KDE, находится в `/usr/lib/kubuntu-desktop-i18n/desktop-template-list`.

2.12.4 Библиотечные символы

Библиотечные символы отслеживаются при помощи файлов `.symbols`, которые позволяют убедиться, что всё на месте. KDE использует библиотеки C++, которые действуют несколько иначе, чем библиотеки C. Для Debian команда Qt/KDE создала скрипты, которые позволяют справиться с этим. Документ Работа с файлами symbols описывает, как создавать и обновлять такие файлы.

Глава 3

Информация для дальнейшего чтения

Вы можете прочитать оффлайн-версию этого руководства в различных форматах, если установите один из двоичных пакетов.

Если вы желаете узнать больше о сборке пакетов Debian, вот несколько ресурсов Debian, которые могут быть вам полезны:

- Как создавать пакеты для Debian;
- Руководство по политике Debian;
- Руководство начинающего разработчика Debian — доступно на различных языках;
- Руководство по созданию пакетов (также доступно в виде пакета);
- Руководство по созданию пакетов для модулей Python.

Мы всегда стремимся улучшить это руководство. Если вы найдёте какую-либо ошибку, или хотите что-либо предложить, пожалуйста, создайте отчёт об ошибке на Launchpad. Если вы хотели бы помочь в работе над руководством, его исходный код также доступен на Launchpad.