Ubuntu Packaging Guide

Выпуск 1.0.0 bzr650 ubuntu14.04.1

Ubuntu Developers

19 January 2018

Оглавление

Welcome to the Ubuntu Packaging and Development Guide! We are currently developing codename Bionic Beaver, which is to be released in April 2018 as Ubuntu 18.04 LTS.

This is the official place for learning all about Ubuntu Development and packaging. After reading this guide you will have:

- Heard about the most important players, processes and tools in Ubuntu development,
- Your development environment set up correctly,
- A better idea of how to join our community,
- Fixed an actual Ubuntu bug as part of the tutorials.

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.

Глава 1

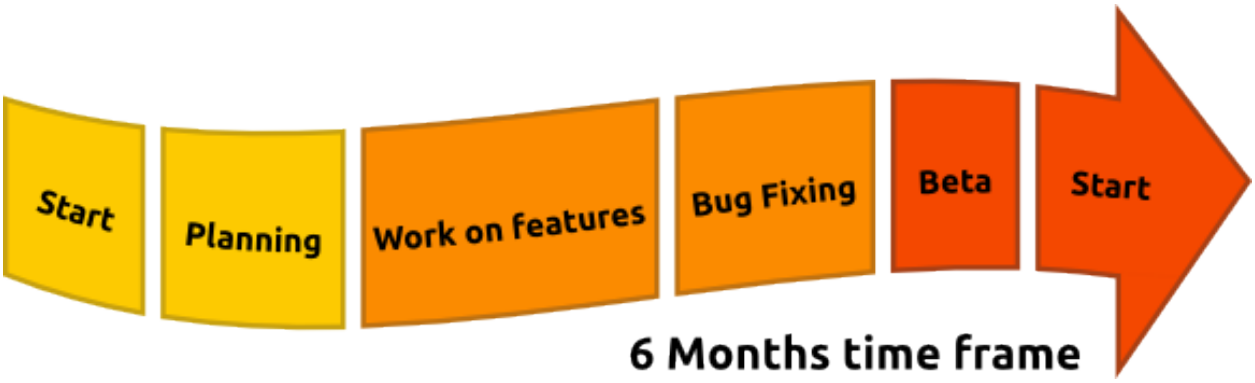
Статьи

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.

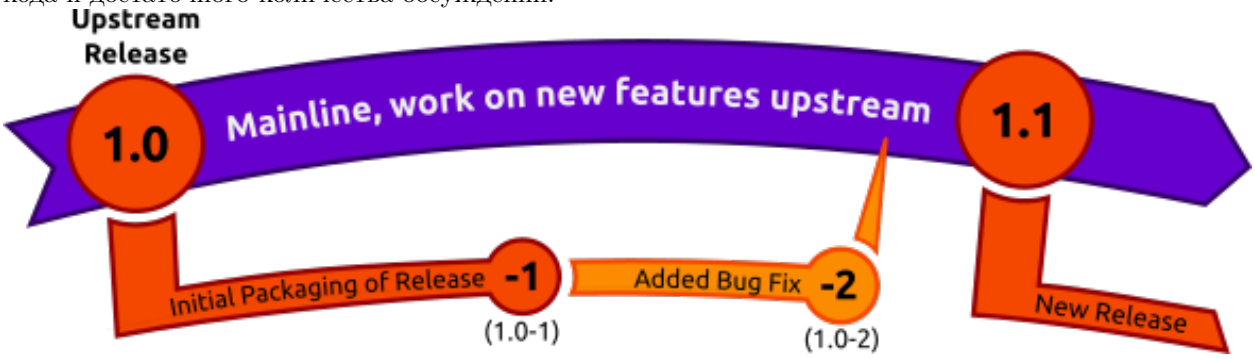
Большая часть доступных в Ubuntu программ создана не самими разработчиками Ubuntu. Многие из программ написаны разработчиками из других проектов с открытым исходным кодом, а затем интегрированы в Ubuntu. Такие проекты принято называть «апстримом» (от англ. upstream — вверх по течению), так как их исходный 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.

Good places to ask your questions are `ubuntu-motu@lists.ubuntu.com` and `#ubuntu-motu` on `freenode`.. You will easily find a lot of new friends and people with the same passion that you have: making the world a better place by making better Open Source software.

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.

Don't want to install the latest development version of Ubuntu? Spin up an LXD container.

1.2.1 Установка базового программного обеспечения для создания пакетов

There are a number of tools that will make your life as an Ubuntu developer much easier. You will encounter these tools later in this guide. To install most of the tools you will need run this command:

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

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.
- `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
```


where <release> is for example *xenial*, *zesty*, *artful* or in the case of Debian maybe *sid* or *buster*. This will take a while as it will download all the necessary packages for a "minimal installation". These will be cached though.

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
- Configure your shell to recognize you (for putting your name in changelogs)

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 командной оболочки

The Debian/Ubuntu packaging tools need to learn about you as well in order to properly credit you in the changelog. Simply open your *~/.bashrc* in a text editor and add something like this to the bottom of it:

```
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 Исправление ошибок в Ubuntu

1.3.1 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.

1.3.2 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.

Take a look at the bitesize bugs in Launchpad, and that might give you an idea of something to work on. It might also interest you to look at the bugs triaged by the Ubuntu One Hundred Papercuts team.

1.3.3 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.

Let's say you've found a bug in Bumprace, a racing game. The Bumprace application can be started by running `/usr/bin/bumprace` on the command line. To find the binary package containing this application, use this command:

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

Команда выведет следующую информацию:

```
bumprace: /usr/bin/bumprace
```

Note that the part preceding the colon is the binary package name. It's often the case that the source package and binary package will have different names. This is most common when a single source package is used to build multiple different binary packages. To find the source package for a particular binary package, type:

```
$ apt-cache showsrc bumprace | grep ^Package:
Package: bumprace
$ apt-cache showsrc tomboy | grep ^Package:
Package: tomboy
```


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

1.3.4 Подтверждение проблемы

Once you have figured out which package the problem is in, it's time to confirm that the problem exists.

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>
XNBC-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.

1.3.5 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.

1.3.6 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.

1.3.7 Получение кода

Once you know the source package to work on, you will want to get a copy of the code on your system, so that you can debug it. The ubuntu-dev-tools package has a tool called `pull-lp-source` that a developer can use to grab the source code for any package. For example, to grab the source code for the tomboy package in `xenial`, you can type this:

```
$ pull-lp-source bumprace xenial
```

If you do not specify a release such as `xenial`, it will automatically get the package from the development version.

Once you've got a local clone of the source package, you can investigate the bug, create a fix, generate a debdiff, and attach your debdiff to a bug report for other developers to review. We'll describe specifics in the next sections.

1.3.8 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.

You may want to create a patch which includes the fix. The command `edit-patch` is a simple way to add a patch to a package. Run:

```
$ 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`.

You must then add a header to your patch containing meta information so that other developers can know the purpose of the patch and where it came from. To get the template header that you can edit to reflect what the patch does, type this:

```
$ quilt header --dep3 -e
```

This will open the template in a text editor. Follow the template and make sure to be thorough so you get all the details necessary to describe the patch.

In this specific case, if you just want to edit `debian/control`, you do not need a patch. Put `Homepage: http://www.linux-games.com/bumprace/` at the end of the first section and the bug should be fixed.

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. Where the change was done.

2. What was changed.
3. Where the discussion of the change happened.

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.

In order to get it sponsored in the next section, you need to file a bug report in Launchpad (if there isn't one already, if there is, use that) and explain why your fix should be included in Ubuntu. For example, for tomboy, you would file a bug here (edit the URL to reflect the package you have a fix for). Once a bug is filed explaining your changes, put that bug number in the changelog.

1.3.9 Тестирование исправления

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

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

This will create a source package from the branch contents (`-us -uc` will just omit the step to sign the source package and `-d` will skip the step where it checks for build dependencies, pbuilder will take care of that) and `pbuilder-dist` will build the package from source for whatever `release` you choose.

Примечание: If `debuild` errors out with "Version number suggests Ubuntu changes, but Maintainer: does not have Ubuntu address" then run the `update-maintainer` command (from ubuntu-dev-tools) and it will automatically fix this for you. This happens because in Ubuntu, all Ubuntu Developers are responsible for all Ubuntu packages, while in Debian, packages have maintainers.

In this case with bumprace, run this to view the package information:

```
$ dpkg -I ~/pbuilder/*_result/bumprace_*.deb
```

As expected, there should now be a `Homepage:` field.

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.

1.3.10 Submitting the fix and getting it included

With the changelog entry written and saved, run `debuild` one more time:

```
$ debuild -S -d
```

and this time it will be signed and you are now ready to get your diff to submit to get sponsored.

In a lot of cases, Debian would probably like to have the patch as well (doing this is best practice to make sure a wider audience gets the fix). So, you should submit the patch to Debian, and you can do that by simply running this:


```
$ 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.

It might be beneficial to just get it included in Debian and have it flow down to Ubuntu, in which case you would not follow the below process. But, sometimes in the case of security updates and updates for stable releases, the fix is already in Debian (or ignored for some reason) and you would follow the below process. If you are doing such updates, please read our *Security and stable release updates* article. Other cases where it is acceptable to wait to submit patches to Debian are Ubuntu-only packages not building correctly, or Ubuntu-specific problems in general.

But if you're going to submit your fix to Ubuntu, now it's time to generate a "debdiff", which shows the difference between two Debian packages. The name of the command used to generate one is also `debdiff`. It is part of the `devscripts` package. See `man debdiff` for all the details. To compare two source packages, pass the two dsc files as arguments:

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

In this case, `debdiff` the dsc you downloaded with `pull-lp-source` and the new dsc file you generated. This will generate a patch that your sponsor can then apply locally (by using `patch -p1 < /path/to/debdiff`). In this case, pipe the output of the debdiff command to a file that you can then attach to the bug report:

```
$ debdiff <package_name>_1.0-1.dsc <package_name>_1.0-1ubuntu1.dsc > 1-1.0-1ubuntu1.debdiff
```

The format shown in `1-1.0-1ubuntu1.debdiff` shows:

1. `1-` tells the sponsor that this is the first revision of your patch. Nobody is perfect, and sometimes follow-up patches need to be provided. This makes sure that if your patch needs work, that you can keep a consistent naming scheme.
2. `1.0-1ubuntu1` shows the new version being used. This makes it easy to see what the new version is.
3. `.debdiff` is an extension that makes it clear that it is a debdiff.

While this format is optional, it works well and you can use this.

Next, go to the bug report, make sure you are logged into Launchpad, and click "Add attachment or patch" under where you would add a new comment. Attach the debdiff, and leave a comment telling your sponsor how this patch can be applied and the testing you have done. An example comment can be:

```
This is a debdiff for Artful applicable to 1.0-1. I built this in pbuilder
and it builds successfully, and I installed it, the patch works as intended.
```

Make sure you mark it as a patch (the Ubuntu Sponsors team will automatically be subscribed) and that you are subscribed to the bug report. You will then receive a review anywhere between several housr from submitting the patch to several weeks. If it takes longer than that, please join `#ubuntu-motu` on `freenode` and mention it there. Stick around until you get an answer from someone, and they can guide you as to what to do next.

Once you have received a review, your patch was either uploaded, your patch needs work, or is rejected for some other reason (possibly the fix is not fit for Ubuntu or should go to Debian instead). If your patch needs work, follow the same steps and submit a follow-up patch on the bug report, otherwise submit to Debian as shown above.

Remember: good places to ask your questions are `ubuntu-motu@lists.ubuntu.com` and `#ubuntu-motu` on freenode. You will easily find a lot of new friends and people with the same passion that you have: making the world a better place by making better Open Source software.

1.3.11 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.

1.4 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.

1.4.1 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.

Download GNU Hello:

```
$ wget -O hello-2.10.tar.gz "http://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz"
```

Now uncompress it:

```
$ tar xf hello-2.10.tar.gz
$ cd hello-2.10
```

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
```

1.4.2 Создание пакета

`bzr-builddeb` includes a plugin to create a new package from a template. The plugin is a wrapper around the `dh_make` command. Run the command providing the package name, version number, and path to the upstream tarball:

```
$ sudo apt-get install dh-make bzr-builddeb
$ cd ..
$ bzr dh-make hello 2.10 hello-2.10.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.

In `debian/changelog` change the version number to an Ubuntu version: `2.10-0ubuntu1` (upstream version 2.10, Debian version 0, Ubuntu version 1). Also change `unstable` to the current development Ubuntu release such as `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."
```

1.4.3 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_2.10-0ubuntu1_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_2.10-0ubuntu1_amd64.deb
```

Можете также установить все пакеты сразу с помощью:

```
$ sudo debi
```

1.4.4 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_2.10-0ubuntu1.dsc
$ lintian hello_2.10-0ubuntu1_amd64.deb
```

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

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_2.10-0ubuntu1.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_2.10-0ubuntu1.changes
```

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.

1.4.5 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.

1.4.6 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 .

1.5 Обновления безопасности и обновления стабильных релизов

1.5.1 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

Now that we have the source package, we need to patch it to fix the vulnerability. You may use whatever patch method that is appropriate for the package, but this example will use `edit-patch` (from the ubuntu-dev-tools package). `edit-patch` is the easiest way to patch packages and it is basically a wrapper around every other patch system you can imagine.

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.

1.5.2 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.

1.6 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`.

1.6.1 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
```

1.6.2 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.

1.6.3 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, начинающе-

ется 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").

1.6.4 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"
```


1.6.5 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`.

1.6.6 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"
```

1.6.7 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.

1.7 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.

1.7.1 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)

1.7.2 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.

1.7.3 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.

1.7.4 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.

1.8 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`.

1.8.1 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.

1.8.2 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.

1.8.3 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.

1.8.4 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.

1.8.5 Информация для дальнейшего чтения

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

1.9 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).

1.9.1 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.

1.9.2 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 ubuntu-dev-tools: Tools for Ubuntu developers

`ubuntu-dev-tools` package is a collection of 30 tools created for making packaging work much easier for Ubuntu developers. It's similar in scope to Debian `devscripts` package.

2.3.1 Setting up packaging environment

`setup-packaging-environment` command allows to interactively set up packaging environment, including setting environment variables, installing required packages and ensuring that required repositories are enabled.

2.3.2 Environment variables

Introducing yourself

`ubuntu-dev-tools` configurations can be set using environment variables. It's used for example in changelogs. For example, to set e-mail address (and full name), use `UBUMAIL` variable. It overrides the `DEBEMAIL` and `DEBFULLNAME` variables used by `devscripts`. To learn `ubuntu-dev-tools` about you, open *~/.bashrc* in text editor and add something like this:

```
export UBUMAIL="Marcin Miko␣lajczak <marcin@example.org>"
```

Now, save this file and restart your terminal or use `source ~/.bashrc`.

Changing preferred builder

Default builder is specified by `UBUNTUTOOLS_BUILDER` variable. To set between *pbuilder* (default), *pbuilder-dist*, and *sbuild*, change this variable. Example:

```
export UBUNTUTOOLS_BUILDER=sbuild
```


Save file and restart terminal.

You can also check whether to update the builder every time before building, by changing `UBUNTUTOOLS_UPDATE_BUILDER` from `no` (default) to `yes`.

2.3.3 Downloading source packages

`ubuntu-dev-tools` comes with `pull-lp-source` command, allowing to download source packages from Launchpad. Its usage is simple. To download latest source package for ubuntu-settings, use:

```
$ pull-lp-source ubuntu-settings
```

You can also specify release from which you want to download source or specify version of source package. `-d` option allows to download source package without extracting. A slightly more complex example would look like this:

```
$ pull-lp-source brisk-menu 0.5.0-1 -d
```

`pull-debian-source` package allows to do the same for Debian source packages. It has similar syntax.

2.3.4 Backporting packages

`ubuntu-dev-tools` provides `backportpackage` allowing us to backport a package from specified release of Ubuntu or Debian. For example, to backport `bzr` package from latest development release for your installed Ubuntu version, simply:

```
$ backportpackage -w . bzr
```

This command allows to use more options. To specify Ubuntu release for which you are going to backport a package, use `-d dest` or `--destination=DEST` parameter, where `DEST` is Ubuntu release, for example `xenial`. You can specify more than one destination. In turn, `-s SOURCE` and `--source=SOURCE` specifies the Ubuntu or Debian release from which you are going to backport a package. `-w DIR` and `--workdir=DIR` specifies directory, where package files will be downloaded, unpacked and built. By default, it will create temporary directory that will be automatically deleted. `-U` or `--update` allows to update build environment before building package. `-u` or `--upload` allows to upload package after building (for example to PPAs) using `dput`.

2.3.5 Requesting backports

`requestbackport` command makes creating backports through Launchpad bugs much easier. It creates testing checklist that will be included in the bug. For example, to request backporting libqt5webkit5 from latest development branch to current stable release (without optional parameters):

```
$ requestbackport libqt5webkit5
```

You should fulfill the checklist if you have already tested the backport.

Additional options allows to specify destination of backport and its source, by using `-d DEST` or `--destination=DEST` and `s SRC` or `--source=SRC`.

2.3.6 Other simple commands

`ubuntu-dev-tools` also includes small utilities allowing to do simple tasks like checking whether .iso file is an Ubuntu installation media.

ubuntu-iso

To do this, use `ubuntu-iso <pathtoiso>`, for example:

```
$ ubuntu-iso ~/Downloads/ubuntu.iso
```

bitesize

"Bitesize" tag is used on Launchpad to describe tasks that are suitable for begineers who want to contribute to one of the projects. `bitesize` command allows to add "bitesize" tag to Launchpad bug with just simple command, by providing its number, like:

```
$ bitesize 1735410
```

404main

`404main` allows to check whether all of package build dependencies are included in main repository of specified Ubuntu distribution. Example:

```
$ 404main libqt5webkit5 xenial
```

If any of the required packages isn't part of Ubuntu main repository, you can check whether the package fulfill Ubuntu main inclusion requirements and request it.

Further reading

`ubuntu-dev-tools` manpages are covering more about usage of this package.

2.4 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`.

2.4.1 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.

2.4.2 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.

2.4.3 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.

2.4.4 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.

2.4.5 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.

2.4.6 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.

2.4.7 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.

2.5 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.

2.5.1 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.

2.5.2 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.

2.6 Setting up sbuild

`sbuild` simplifies building Debian/Ubuntu binary package from source in clean environment. It allows to try debugging packages in environment similar (as opposed to `pbuild`) to builders used by Launchpad.

It works on different architectures and allows to build packages for other releases. It needs kernel supporting overlayfs.

2.6.1 Installing sbuild

To use sbuild, you need to install sbuild and other required packages and add yourself to the `sbuild` group:

```
$ sudo apt install debhelper sbuild schroot ubuntu-dev-tools
$ sudo adduser $USER sbuild
```

Create `.sbuildrc` in your home directory with following content:

```
# Name to use as override in .changes files for the Maintainer: field
# (mandatory, no default!).
$maintainer_name='Your Name <user@example.org>';
```



```
# Default distribution to build.
$distribution = "bionic";
# Build arch-all by default.
$build_arch_all = 1;

# When to purge the build directory afterwards; possible values are "never",
# "successful", and "always".  "always" is the default. It can be helpful
# to preserve failing builds for debugging purposes.  Switch these comments
# if you want to preserve even successful builds, and then use
# "schroot -e --all-sessions" to clean them up manually.
$purge_build_directory = 'successful';
$purge_session = 'successful';
$purge_build_deps = 'successful';
# $purge_build_directory = 'never';
# $purge_session = 'never';
# $purge_build_deps = 'never';

# Directory for writing build logs to
$log_dir=$ENV{HOME}."/ubuntu/logs";

# don't remove this, Perl needs it:
1;
```

Replace "Your Name <user@example.org>" with your name and e-mail address. Change default distribution if you want, but remember that you can specify target distribution when executing command.

If you haven't restarted your session after adding yourself to the `sbuild` group, enter:

```
$ sg sbuild
```

Generate GPG keypair for sbuild and create chroot for specified release:

```
$ sbuild-update --keygen
$ mk-sbuild bionic
```

This will create chroot for your current architecture. You might want to specify another architecture. For this, you can use `--arch` option. Example:

```
$ mk-sbuild xenial --arch=i386
```

2.6.2 Using schroot

Entering schroot

You can use `schroot -c <release>-<architecture> [-u <USER>]` to enter newly created chroot, but that's not exactly the reason why you are using sbuild:

```
$ schroot -c bionic-amd64 -u root
```

Using schroot for package building

To build package using sbuild chroot, we use (surprisingly) the `sbuild` command. For example, to build `hello` package from x86_64 chroot, after applying some changes:


```
apt source hello
cd hello-*
sed -i -- 's/Hello/Goodbye/g' src/hello.c   # some
sed -i -- 's/Hello/Goodbye/g' tests/hello-1 #
dpkg-source --commit
dch -i                                      #
update-maintainer                           # changes
sbuild -d bionic-amd64
```

To build package from source package (`.dsc`), use location of the source package as second parameter:

```
sbuild -d bionic-amd64 ~/packages/goodbye_*.dsc
```

To make use of all power of your CPU, you can specify number of threads used for building using standard `-j<threads>`:

```
sbuild -d bionic-amd64 -j8
```

2.6.3 Maintaining schroots

Listing chroots

To get list of all your sbuild chroots, use `schroot -l`. The `source:` chroots are used as base of new schroots. Changes here aren't recommended, but if you have specific reason, you can open it using something like:

```
$ schroot -c source:bionic-amd64
```

Updating schroots

To upgrade the whole schroot:

```
$ sbuild-update -ubc bionic-amd64
```

Expiring active schroots

If because of any reason, you haven't stopped your schroot, you can expire all active schroots using:

```
$ schroot -e --all-sessions
```

2.6.4 Further reading

There is Debian wiki page covering sbuild usage.

Ubuntu Wiki also has article about basics of sbuild.

`sbuild` manpages are covering details about sbuild usage and available features.

2.7 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.

2.7.1 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).

2.7.2 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`.

2.7.3 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`.

2.7.4 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.