# Ubuntu Packaging Guide

*Реліз 1.0.2 bzr685 ubuntu14.04.1*

Ubuntu Developers

April 27, 2019

Зміст

Welcome to the Ubuntu Packaging and Development Guide!

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 лише вільна операційна система з відкритим джерельним кодом, її платформа також є відкритою й забезпечує прозорість розробки. Можна легко отримати джерельний код для кожного окремого компонента, й кожну окрему зміну у платформі Ubuntu можна перевірити.

Це означає, що Ви можете прийняти активну участь у її покращенні, й спільнота розробників платформи Ubuntu завжди зацікавлена у залученні нових участників.

Ubuntu також є спільнотою чудових людей, що вірять у те, що проґрамне забезпечення має бути вільним та доступним для усіх. Участники спільноти вітають Вас й бажають, щоб Ви теж до них приєдналися. Ми бажаємо, щоб Ви приймали участь у нашій праці, задавали питання, робили Ubuntu ліпшою разом з нами.

Якщо в Вас виникнуть складнощі: не хвилюйтеся! Прочитайте *розділ про комунікацію*, й Ви дізнаєтеся, як легко зв'язатися з рештою розробників.

Цей посібник складається з двох розділів:

- Перелік статтей, заснованих на певних завданнях, які Вам можливо знадобиться виконати.
- Набір статтей бази знань, у яких детальніше розглядаються використовувані нами інструменти й робочі процеси.

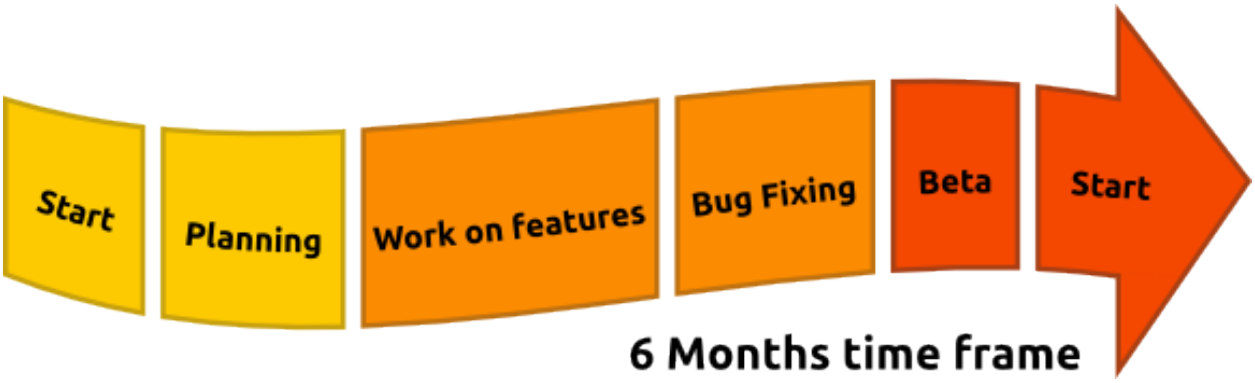
# Статті

## 1.1 Вступ у розробку Ubuntu

Ubuntu складається з тисяч різних компонентів, написаних великою кількістю мов програмування. Кожен компонент — бібліотека, засіб або графічний застосунок — доступний у вигляді пакунку джерельного коду. Пакунки джерельних кодів у більшості випадків складаються з двох частин: сам джерельний код і метадані. Метадані включають у себе залежності пакунку, інформацію про авторське право і ліцензію, а також інструкції зі збирання пакунку. Після того, як пакунок джерельних кодів скомпільований, у процесі збирання ми отримаємо двійкові пакунки, що є .deb файлами, які користувачі можуть встановити.

Кожного разу, коли виходить нова версія додатку, або коли хтось вносить зміни у джерельний код пакунку, що входить у склад Ubuntu, пакунок з джерельним кодом повинен бути завантажений на збірочні комп'ютери Launchpad для компілювання. Готові бінарні пакунки потім потрапляють у сховище ПЗ та його дзеркала у різних країнах. URL в `/etc/apt/sources.list` є посиланнями на сховище або його дзеркал. Кожного дня збираються образи для різних версій Ubuntu. Вони можуть бути використані у різних умовах. Є образи, які можна записати на USB-носії або на DVD-диски, образи, які можна використовувати для мережевого завантаження й є образи, призначені для встановлення на телефон або планшет. Ubuntu, серверна версія Ubuntu, Kubuntu й інші гілки мають специфічний перелік необхідних пакунків, які потрапляють в образ. Це образи, які потім використовуються для перевірки встановлення й забезпечують зворотній зв'язок для подальшого планування випуску.

Розробка Ubuntu дуже залежить від поточної фази циклу випуску. Ми випускаємо нову версію Ubuntu щопів-року, що можливо лише завдяки тому, що ми встановлюємо точні дати «заморожування». При досягненні кожної дати заморожування очікується, що розробники будуть вносити рідші та менш значущі зміни. Заморожування нових функцій (feature freeze) — це перша велика дата заморожування, що приходить після проходження першої половини циклу розробки. На цьому етапі нові функції повинні бути переважно зреалізовані. У залишкову частину циклу має бути зосередженя на виправленні вад. Після цього «заморожується» користувацький інтерфейс, потім документація, ядро й тощо, після чого випускається бета-версія (beta release), яка інтенсивно тестується. Після того, як випущена бета-версія, виправляються лише критичні помилки, й випускається release candidate, який, якщо він не містить серйозних помилок, стає у подальшому кінцевим випуском.




Тисячі пакунків джерельного коду, мільярди рядків коду, сотні розробників потребують більше спілкування й планування для підтримування високих стандартів якості. На початку й у середині кожного циклу випуску організовується Ubuntu Developer Summit, де розробники та участники збираються разом, щоб планувати нові функції у наступних випусках. Кожна функція обговорюється зацікавленими сторонами, і складається специфікація, що містить детальну інформацію про початкові припущення, реалізації, внесення необхідних змін у інші пакунки, про тестування тощо. Усе це робиться прозоро й відкрито, тож Ви можете взяти участь віддалено, подивитися видивотрансляцію, поспілкуватися з участниками та підписатися на специфікації, щоб завжди знати про те що відбувається.

Втім на нараді можуть бути обмірковані не усі зміни, оскільки Ubuntu залежить від змін, що вносяться у інші проєкти. Тому участники розробки Ubuntu залишаються постійно на зв'язку. Більшість команд або проєктів використовують окремі переліки розсилки, щоб уникнути більшого потоку не пов'язаних з їх спеціялізацією листів. Для більш безпосереднього координування розробники і добровільні участники використовують Internet Relay Chat (IRC). Усі оговорення є відкритими та публічними.

Ще одним важливим інструментом зв'язку є звіти про вади. Якщо у пакунку або частині інфраструктури виявлена помилка, звіт про неї відправляється на Launchpad. У цьому звіті зібрана уся інформація, й при необхідності відомості про важливість, статус помилки, та особа призначена для її усунення оновлюються. Це робить звіт про ваду ефективним засобом відстежування помилок у пакунках або проєктах й оптимізації робочого навантаження.

Велика частина доступних в Ubuntu проґрам створена не самими розробниками Ubuntu. Багато з проґрам написані розробниками з інших проєктів з відкритим джерельним кодом, а потім інтегровані в Ubuntu. Такі проєкти прийнято називати «апстрімом» (від англ. upstream — угору за течією), оскільки їх джерельний код вливається в Ubuntu, де ми «просто» інтегруємо його у систему. Зв'язок з апстрімом дуже важливий для Ubuntu. Не лише Ubuntu отримує проґрамний код з апстріму, але й апстрім отримує від користувачів Ubuntu(й інших дистрибутивів) звіти про вади та латки.

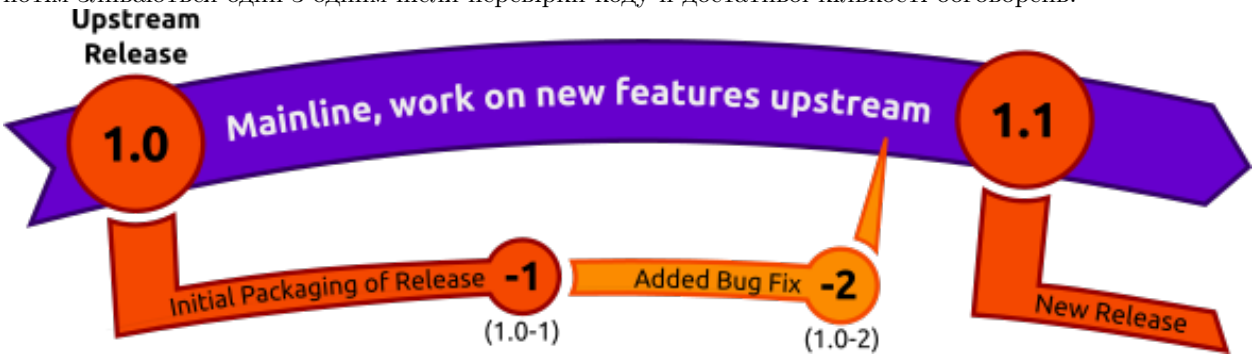
Найважливішим апстрімом для Ubuntu є Debian. Debian — це дистрибутив, на якому заснована Ubuntu, й саме там створені багато інженерних рішень, що стосуються інфраструктури пакунків. За традицією, в Debian для кожного окремого пакунку завжди є супроводжуючий (мейнтейнер) або окрема група супроводу. В Ubuntu теж є команди, зацікавлені у роботі над підмножиною пакунків й, зрозуміло, кожен розробник має власну область компетенції, але участь (і права завантаження змін) зазвичай доступні будь-кому, хто продемонструє здатність та бажання працювати.

Внести зміну в Ubuntu новому участнику не так складно, як здається, й це може бути вельми корисним досвідом. Це дозволяє не лише навчитися чомусь новому й захоплюючому, але й поділитися своїм вирішенням проблеми з мільйонами інших користувачів.

Розробка відкритого проґрамного забезпечення відбувається у розподіленому світі, у якому у розробників можуть бути різні цілі й різні області інтересів. Наприклад, може бути так, що окремий апстрім зацікавлений у роботі над великим нововведенням, в той час як Ubuntu, внаслідок тісного розкладу релізів, більше зацікавлена у випуску стабільної версії, у якій лише виправлені деякі вади. Тому ми використовуємо «Ubuntu Distributed Development», де робота над кодом ведеться у різних гілках, які

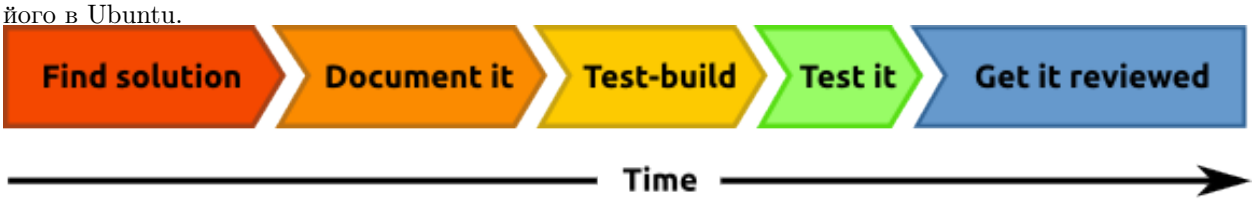
 

потім зливаються один з одним після перевірки коду й достатньої кількості обговорень.




У приведеному вище прикладі має сенс включити в Ubuntu існуючу версію проєкту, зробити виправлення вад, додати їх у апстрім для наступного випуску проєкту й включити його (якщо він до цього придатний) у наступний випуск Ubuntu. Це буде найкращим компромісом і ситуацією у якій виграють обидві сторони.

Щоб виправити помилку в Ubuntu, потрібно спочатку отримати джерельний код пакунку, попрацювати над його виправленням, забезпечити свою роботу документацією, щоб іншим розробникам та користувачам було легко зрозуміти, що саме Ви зробили, а потім зібрати пакунок й протестувати його. Після того, як Ви протестували змінений пакунок, можна запропонувати включити його у поточний розроблюваний реліз Ubuntu. Розробник з правом завантаження перевірить Ваш пакунок й потім інтегрує його в Ubuntu.




При спробі знайти вирішення корисно перевірити, чи відомо про проблему, над якою Ви працюєте в апстрімі, й чи не знайдено там вже її можливе вирішення. Якщо ні, зробіть усе можливе, щоб вирішити проблему спільними зусиллями.

Додаткові кроки, які Ви можете зробити, — це адаптування Вашої зміни для попередніх підтримуваних випусків Ubuntu й відправлення її в апстрім.

Найважливішими вимогами для успішної розробки в Ubuntu є: вміти «примушувати речі знову працювати», не боятися читати документацію й задавати питання, бути командним гравцем та мати певну схильність до роботи 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 Підготовка

Існує декілька речей, які потрібно зробити перед початком розробки в Ubuntu. Ця стаття допоможе Вам підготувати комп'ютер до роботи з пакунками й відправці Ваших пакунків на платформу хостингу Ubuntu — Launchpad. Ось що ми розглянемо:

- Встановлення проґрам для роботи з пакунками. Вони включають у себе:
  - специфічні для Ubuntu засоби створення пакунків

  - криптографічну проґраму, яка дозволить іншим переконатися, що робота виконана саме Вами
  - додаткові проґрами шифрування, що забезпечують безпечне передавання файлів
- Створення й налаштування облікового запису на Launchpad
- Налаштування Вашого середовища розробки для полегшення локальної збірки пакунків, взаємодії з іншими розробниками й відправки Ваших змін на Launchpad.

---

**Примітка:** Рекомендується виконувати роботу зі створення пакунків у поточній розроблюваній версії Ubuntu. Це дозволить Вам тестувати зміни у тому ж середовищі, у яке вони у дійсності потім будуть внесені.

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

---

### 1.2.1 Встановлення базового проґрамного забезпечення для створення пакунків

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
```

Ця команда встановить такі проґрами:

- `gnupg` – GNU Privacy Guard містить інструменти, які знадобляться для створення криптографічного ключа, за допомогою якого Ви будете підписувати файли, які бажаєте завантажити на Launchpad
- `pbuilder` – інструмент для створення готових до подальшого розповсюдження збірок пакунків у чистому та ізольованому середовищі.
- `ubuntu-dev-tools` (і його безпосередня залежність `devscripts`) – набір інструментів, що спрощують багато завдань зі створення пакунків.
- `apt-file` надає простий спосіб знайти двійковий пакунок, що містить заданий файл.

#### Створення ключа GPG

GPG — абревіятура для GNU Privacy Guard, реалізуючого стандарт OpenPGP, який дозволяє підписувати та шифрувати повідомлення й файли. Це корисно у ряді ситуацій. У нашому випадку важливо, що Ви можете використовувати свій ключ для підписування файлів, щоб можна було переконатися, що саме Ви з ними працювали. Якщо Ви завантажите пакунок джерельного коду на Launchpad, він буде прийнятий лише у тому випадку, якщо можна точно визначити, хто саме відправив пакунок.

Щоб згенерувати новий ключ GPG, наберіть:

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

GPG спочатку спитає, який тип ключа Ви бажаєте створити. Типовий вибір (RSA и DSA) цілком влаштує. Далі він спитає вказати розмір ключа. Типовий розмір (на цей час 2048) влаштує, але 4096 надійніше. Далі, проґрама спитає, чи бажаєте Ви, щоб термін дії ключа вийшов через якийся час. Безпечніше відповісти "0", що означає, що термін дії не сплине ніколи. Останнє питання буде про Ваше ім'я та адресу електронної пошти. Просто виберіть адресу, якою Ви користуєтеся при розробці Ubuntu, додаткові адреси можна буде додати потім. Додавати коментар не обов'язково. Після цього потрібно вказати надійне парольне гасло (парольне гасло — це просто пароль, у якому дозволяється використовувати пробіли).

Тепер GPG створить для Вас ключ, що може зайняти певний час. Для його створення знадобляться випадкові байти, тому буде просто чудово, якщо Ви задасте своїй системі якусь роботу. Порухайте вказівник миші, наберіть декілька абзаців будь-якого тексту, завантажте будь-яку веб-сторінку.

Коли процес буде завершено, Ви отримаєте повідомлення типу такого:

```
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
```

У даному випадку, `43CDE61D` — це ідентифікатор ключа (*key ID*).

Потім потрібно завантажити відкриту (public) частину Вашого ключа на сервер ключів, щоб усі могли ідентифікувати повідомлення і файли, як відправлені Вами. Для цього уведіть:

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

Ця команда відправить Ваш ключ на сервер ключів Ubuntu, а мережа серверів ключів автоматично синхронізує ключ між собою. Після того, як ця синхронізація завершиться, Ваш підписаний відкритий ключ буде готовий для засвідчення зробленого Вами вкладу в усьому світі.

### Створення ключа SSH

SSH або *Secure Shell* — це протокол, який дозволяє безпечно обмінюватися даними мережею. Звичайною практикою є використання SSH для доступу й відкриття командної оболонки на іншому комп'ютері й для безпечної пересилки файлів. З нашою метою ми переважно будемо використовувати SSH для безпечної відправки пакунків джерельного коду на Launchpad.

Щоб згенерувати ключ SSH, уведіть:

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

Типове ім'я файлу цілком влаштує, тож можете залишити його як є. З метою безпеки настійно рекомендується вказати парольне гасло.

### Налаштування pbuilder

`pbuilder` дозволяє локально збирати пакунки на Вашому комп'ютері. Він слугує декільком цілям:

- Збірка буде виконана у мінімальному й чистому середовищі. Це дасть можливість переконатися, що збірку вдастся успішно відтворити й на інших комп'ютерах, але при цьому допоможе уникнути змін у Вашій локальній системі
- Зникає необхідність у локальному встановленні усіх *збірочних залежностей*
- Можна налаштувати декілька екземплярів для різних випусків Ubuntu і Debian

Налаштувати `pbuilder` дуже просто. Наберіть

```
$ 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 Підготовка до роботи з Launchpad

Після того, як базова локальна конфігурація створена, наступним кроком буде налаштування системи для роботи з Launchpad. У цьому розділі ми сфокусуємося на таких питаннях:

- Що таке Launchpad й як створити обліковий запис на Launchpad
- Завантаження Ваших ключів GPG та SSH на Launchpad
- Configure your shell to recognize you (for putting your name in changelogs)

#### Відомості про Launchpad

Launchpad є центральним елементом інфраструктури, що використовується нами в Ubuntu. Він зберігає не лише наші пакунки й наш код, але й такі речі, як переклади, звіти про вади, а також інформацію про людей, що працюють над Ubuntu і їх приналежність до різних команд. Ви можете також використовувати Launchpad, щоб оголосити пропоновані Вами виправлення й попрохати інших розробників Ubuntu перевірити та підтримати їх.

Вам потрібно буде зареєструватися на Launchpad та надати певну мінімальну кількість інформації про себе. Це дозволить Вам стягувати або відправляти джерельний код, відправляти звіти про вади тощо.

Крім хостингу Ubuntu, Launchpad може надавати місце для будь-якого вільного проґрамного проєкту. Додаткову інформацію дивіться на Довідкових вікі-сторінках Launchpad

#### Створення облікового запису на Launchpad

Якщо в Вас ще немає облікового запису на Launchpad, Ви легко можете створити його. Якщо обліковий запис є, але Ви не пам'ятаєте свій Launchpad ID, його можна дізнатися, зайшовши на https://launchpad.net/~ і поглянувши на частину після ~ в URL.

При реєстрації на Launchpad Вас попрохають вибрати відображуване ім'я. Рекомендується вказати тут Ваше справжнє ім'я, щоб Ваші колеги - розробники Ubuntu могли краще з Вами познайомитися.

При реєстрації нового облікового запису Launchpad відправить Вам листа з посиланням, яке потрібно відкрити у оглядачі тенет, щоб підтвердити вказану Вами адресу електронної пошти. Якщо Ви не отримали листа, перевірте теку небажаної пошти (спаму).

Довідкова сторінка нового облікового запису на Launchpad містить додаткову інформацію про процес та додаткові налаштування, які можна зробити.

#### Завантаження Вашого ключа GPG на Launchpad

Спочатку потрібно отримати відбиток та ідентифікатор ключа.

Щоб взнати свій відбиток ключа GPG (fingerprint), наберіть:

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

й Ви побачите щось типу:

```
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
```

Потім виконайте цю команду для відправки Вашого ключа на сервер ключів Ubuntu:

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

де `43CDE61D` слід замінити на Ваш ідентифікатор ключа (він у першому рядку виводу попередньої команди). Тепер можна імпортувати свій ключ на Launchpad.

Зайдіть на https://launchpad.net/~/+editpgpkeys й скопіюйте дані з рядка «Key fingerprint» у текстове поле. У наведеному вище прикладі це буде `5C28 0144 FB08 91C0 2CF3 37AC 6F0B F90F 43CD E61D`. Потім клацніть «Import Key».

Launchpad скористається відбитком ключа для перевірки наявності Вашого ключа на сервері ключів Ubuntu й, у випадку успіху, відправить Вам зашифроване повідомлення електронної пошти, з пропозицією підтвердити імпорт ключа. Перевірте свою пошту й прочитайте листа, отриманого з Launchpad. *Якщо Ваш клієнт електронної пошти підтримує шифрування OpenPGP, він запропонує увести пароль, який Ви вибрали при створенні ключа GPG. Уведіть пароль, потім клацніть на посиланні, аби підтвердити, що цей ключ належить Вам.*

Launchpad шифрує пошту, використовуючи Ваш публічний ключ, тож Ви зможете переконатися, що ключ Ваш. Якщо Ви користуєтеся поштовим клієнтом Thunderbird, який використовується в Ubuntu типово, для дешифрування повідомлення можете встановити доповнення Enigmail. Якщо Ваша поштова програма не підтримує шифрування OpenPGP, скопіюйте зашифрований вміст листа у буфер обміну, наберіть у терміналі `gpg` й вставте вміст листа у вікно терміналу.

Повернувшись на сайт Launchpad, скористайтеся кнопкою «Confirm», щоб Launchpad завершив імпорт Вашого ключа OpenPGP.

Додаткову інформацію можна знайти на https://help.launchpad.net/YourAccount/ImportingYourPGPKey

### Завантаження Вашого ключа SSH на Launchpad

Відкрийте у оглядачі тенет https://launchpad.net/~/+editsshkeys, а у текстовому редакторі файл `~/.ssh/id_rsa.pub`. Це відкрита частина Вашого ключа SSH, тому можна без побоювань надати до неї спільний доступ на Launchpad. Скопіюйте вміст файлу й вставьте його у текстове поле на веб-сторінці з міткою «Add an SSH key». Потім клацніть «Import Public Key».

Для додаткової інформації про цей процес відвідайте сторінку про створення ключової пари SSH на Launchpad.

### Налаштування командної оболонки

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"
```

Потім збережіть файл й перезапустіть термінал або наберіть:

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

(Якщо Ви не користуєтеся стандартною командною оболонкою *bash*, відредагуйте конфігураційний файл тієї оболонки, яку Ви полюбляєте використовувати.)

# 1.3 Виправлення помилок в Ubuntu

## 1.3.1 Вступ

Якщо Ви дотримувалися інструкцій з *підготовки до розробки Ubuntu*, усе повинно бути вже готово до роботи.


Як Ви можете бачити на картинці вище, у процесі виправлення помилок в Ubuntu немає ніяких несподіванок: Ви знаходите проблему, отримуєте код, виправляєте його, тестуєте, відправляєте на Launchpad й прохаєте, щоб його перевірили та об'єднали з основним кодом. У цьому посібнику ми пройдемо через усі необхідні кроки, один за іншим.

## 1.3.2 Пошук проблеми

Існують різні способи знайти, над чим можна попрацювати. Це може бути помилка, яку Ви виявили самі (що дає Вам відмінну можливість перевірити своє виправлення) або проблема, яку Ви помітили десь ще, наприклад у звіті про ваду.

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 З'ясування того, що потрібно виправити

Якщо Ви не знаєте, у якому пакунку джерельного коду міститься помилка, але знаєте шлях до цього застосунку у Вашій системі, то Ви зможете знайти пакунок джерельного коду, над яким потрібно попрацювати.

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 виведе таку інформацію:

```
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` встановлена в Ubuntu типово.

### 1.3.4 Підтвердження проблеми

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

Припустимо, у описі пакунку `bumprace` відсутня інформація про його домашню сторінку. У якості першого кроку, слід перевірити, чи не виправлена вже ця помилка. Зробити це просто: подивіться у Центрі додатків або запустіть:

```
apt-cache show bumprace
```

Вивід команди повинен бути приблизно таким:

```
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
```

У якості контрприкладу можна привести `gedit`, де домашня сторінка вказана:

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

У деяких випадках Ви можете зіштовхнутися з тим, що проблема, вирішення якої Ви шукаєте, вже кимось усунена. Щоб уникнути даремного витрачення часу й праці, має сенс проробити деяку детективну роботу.

### 1.3.5 Вивчення ситуації з помилкою

Спочатку потрібно перевірити, чи не існує вже повідомлення про цю помилку в Ubuntu. Можливо, хтось вже працює над її виправленням, або ми можемо якось внести свій внесок у вирішення цієї проблеми. Для Ubuntu ми поглянемо на https://bugs.launchpad.net/ubuntu/+source/bumprace й побачимо, що відкритого звіту про нашу ваду там немає.

**Примітка:** Для Ubuntu URL `https://bugs.launchpad.net/ubuntu/+source/<пакунок>` завжди приводить на сторінку помилок у вказаномум пакунку джерельного коду.

У Debian, який є основним джерелом пакунків Ubuntu ми поглянемо на http://bugs.debian.org/src:bumprace й також не знайдемо там повідомлення про нашу ваду.

**Примітка:** Для Debian URL `http://bugs.debian.org/src:<пакет>` завжди приводить на сторінку помилок у вказаному пакунку джерельного коду.

Помилка, над якою ми працюємо, незвичайна у тому сенсі, що вона стосується лише пакункування `bumprace`. Якби це була вада у джерельному коді, корисно було б також перевірити систему відстеження помилок апстріму. Нажаль, ця процедура часто різниться для кожного окремого пакунку, але завжди можна скористатися пошуком в інтернеті, й у більшості випадків Ви з'ясуєте, що вона виявиться не такою вже й складною.

### 1.3.6 Пропозиція допомоги

Якщо Ви виявили відкриту ваду, яка ще нікому не призначена, й Ви готові взятися за її усунення, слід написати коментар з Вашим вирішенням. Включіть у нього щонайбільше інформації: При яких обставинах з'являється помилка? Як Ви її виправили? Чи тестували Ви свій спосіб усунення помилки?

Якщо повідомлення про помилку не було зареєстроване, Ви можете його створити. Подумайте над двома речами: Може бути, зміна настільки мала, що достатньо просто попрохати когось застосувати її? Може бути, в Вас вийшло лише частково виправити ваду, й Ви бажаєте поділитися Вашою часткою?

Буде чудово, якщо Ви можете запропонувати свою допомогу, й вона, без сумніву, буде з готовністю прийнята.

### 1.3.7 Отримання коду

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 Виправлення помилки

Є цілі книги про знаходження помилок, їх виправлення, тестування й так далі. Якщо Ви початківець у проґрамуванні, спробуйте виправляти нескладні помилки, такі як очевидні друкарські помилки. Намагайтеся робити Ваші зміни мінімальними й чітко документувати Ваші зміни та припущення.

Перед тим, як працювати над помилкою, переконайтеся, що вона не виправлена вже кимось іншим, й ніхто не займається на дану мить її виправленням. Не завадить перевірити наступні джерела:

- Система відстеження помилок апстріму (й Debian) — відкриті та закриті помилки,
- Історія версій в апстрімі (або у новій версії) може містити відомості про виправлення помилки,
- звіти про вади й нові версії пакунків в Debian та інших 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
```

Ця команда скопіює файли, необхідні для збірки пакунку, у тимчасову директорію. Ви можете змінювати ці файли у текстовому редакторі або застосовувати латки з upstream, наприклад:

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

Після редагування файлу наберіть `exit` або натисніть `control-d`, щоб вийти з тимчасової командної оболонки. Нова латка буде додана 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.

### Документування виправлення

Дуже важливо документувати свої зміни у достатній мірі, щоб розробникам потім не довелося здогадуватися, якими були причини й передумови зроблених Вами змін. Кожен пакунок джерельного коду Debian та Ubuntu включає у себе файл `debian/changelog`, у якому відстежуються вносювані у пакунок зміни.

Найпростіший спосіб оновити його — це виконати:

```
$ dch -i
```

Ця команда додасть у файл шаблон запису й запустить редактор, у якому Ви зможете додати інформацію якої бракує. Ось приклад цього запису:

```
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` повинна заповнити перший і останній рядок цього запису у файлі changelog. Перший рядок містить ім'я пакунку джерельного коду, номер його версії, у який реліз Ubuntu він завантажений, терміновість (майже завжди низька — 'low'). Останній рядок завжди містить ім'я, адресу електронної пошти та мітку часу змінювання (у форматі **RFC 5322**).

Тепер давайте сфокусуємося на тому що має міститися у самому запису файлу changelog. Дуже важливо задокументувати:

1. Where the change was done.

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

У нашому (досить поодинокому) прикладі останньому пункту відповідає `(LP: #123456)`, тобто посилання на помилку на Launchpad з номером 123456. Звіти про вади, теми переліків розсилки або специфікації зазвичай є добрими джерелами інформації для обґрунтування змін. У якості додаткового заохочення, якщо Ви використовуєте нотацію `LP: #<номер>` для помилок на Launchpad, то помилка автоматично отримає статус закритої при відправці пакунку 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 Тестування виправлення

Щоб зібрати тестовий пакунок з Вашими змінами, виконайте такі команди:

```
$ 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.

---

**Примітка:** 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.

---

**Примітка:** У більшості випадків Вам доведеться дійсно встановити пакунок, щоб перевірити правильність його роботи. Наш випадок на багато простіший. Якщо збірка завершилася з успіхом, готові двійкові пакунки можна буде знайти в `~/pbuilder/<випуск>_result`. Встановіть їх за допомогою `sudo dpkg -i <пакунок>.deb` або подвійного клацу на них у файловому менеджері.

---

### 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
```

Ця команда проведе Вас через декілька кроків, необхідних для оформлення звіту про ваду й відправки його у правильне місце. Обов'язково продивіться Ваші зміни, щоб переконатися, що там немає нічого зайвого.

Комунікація має дуже велике значення, тому при додаванні опису надайте добре та дружнє пояснення.

Якщо усе пройшло добре, то Ви повинні отримати поштове повідомлення від системи відстежування помилок Debian з додатковою інформацією. Інколи це може зайняти декілька хвилин.

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 hours 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 Додаткові зауваження

Якщо Ви можете внести у пакунок декілька тривіальних виправлень одразу, зробіть це. Це дозволить розробникам швидше розглянути й застосувати ці зміни.

Якщо Ви бажаєте внести декілька великих змін, краще посилати латки або запити на злиття окремо для кожної зміни. Це простіше, якщо вже створені індивідуяльні повідомлення про вади.

## 1.4 Створення пакунків для нових проґрам

Хоч у архіві Ubuntu є тисячі пакунків, вистачає проґрам, якими ніхто не займається. Якщо Ви знаєте про якусь чудову проґраму, про яку, на Вашу думку, варто взнати ширшому колу користувачів, Ви можете спробувати докласти свою руку до створення пакунку для Ubuntu або PPA. Цей посібник проведе Вас через етапи створення пакунку для нової проґрами.

Спочатку Вам слід прочитати статтю *Підготовка*, щоб підготувати своє середовище розробки.

### 1.4.1 Перевірка проґрами

Першим етапом створення пакунку є отримання tar-файлу з апстріму («апстрімом» ми звемо авторів застосунків) й перевірка того, що він нормально компілюється та запускається.

Цей посібник проведе Вас через процес створення пакунку для невеликого застосунку 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
```

Цей застосунок використовує систему збірки autoconf, тож нам потібно запустити `./configure` для підготовки до компілювання.

При цьому буде перевірено наявність необхідних для збірки залежностей. Оскільки `hello` — простий приклад, `build-essential` забезпечить нас усім, що потрібно. Для складніших проґрам, команда завершиться помилкою, якщо в нас немає необхідних бібліотек і файлів для розробки. Встановіть потрібні пакунки й повторіть процес, поки команда не завершиться з успіхом.:

```
$ ./configure
```

Тепер потрібно скомпілювати джерельний код:

```
$ make
```

Якщо компілювання завершилося успішно, можна встановити та запустити проґраму:

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

### 1.4.2 Створення пакунку

`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
```

Коли він спитає тип пакунку, оберіть `s`: одинарний бінарник. Це імпортує код у гілку й створить теку `debian/`. Погляньте на її вміст: більшість автоматично створених файлів потрібні лише для спеціялізованих пакунків (наприклад модулі Emacs), тож можна почати з вилучення зайвих файлів:

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

Тепер потрібно внести зміни у кожен з файлів.

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`.

Велика частина процесу компілювання пакунку здійснюється скриптами з `debhelper`. Оскільки поведінка `debhelper` змінюється при виході старшої версії, файл compat повідомляє `debhelper` яку саме версію використовувати. Має сенс використовувати найсвіжішу версію: `9`.

Файл `control` містить усі метадані пакунку. Перший абзац дає опис пакунку джерельних кодів. Другий та наступні абзаци дають опис двійкових пакунків, які повинні бути зібрані. Нам знадобиться додати пакунки, необхідні для компілювання додатку в `Build-Depends:`. Для `hello` він має включати, як мінімум:

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

Також потрібно заповнити опис проґрами у полі `Description:`.

`copyright` потрібно заповнити у відповідності з ліцензією на джерело з апстріму. Згідно файлу hello/COPYING, це ліцензія GNU GPL 3 або пізніша її версія.

`docs` повинен містити будь-які файли документації з апстріму, які, на Вашу думку, мають бути включені у готовий пакунок.

`README.source` і `README.Debian` необхідні, лише якщо Ваш пакунок має якісь нестандартні функції. В нас таких немає, тож можна вилучити ці файли.

`source/format` можна залишити як є, він дає опис формату версії пакунку джерельного коду, який повинен бути `3.0 (quilt)`.

`rules` — найскладніший файл. Це Makefile, який компілює код й перетворює його у двійковий пакунок. На щастя, основну частину роботи на сьогодні автоматично виконує `debhelper 7`, тож універсальна мета `%` просто запускає сценарій `dh`, який робить усе, що потрібно.

Детальніший опис усіх цих фалів у статті *огляд каталогу debian*.

Нарешті, закоммте код у гілку для пакунків:

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

### 1.4.3 Збірка пакунку

Тепер нам потрібно перевірити, що наші джерельні файли для пакунку успішно компілюються й збираються у двійковий .deb-пакунок:

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

`bzr builddeb` — це команда для збірки пакунку у його поточному місцезнаходженні. `-us -uc` повідомляє що підписувати пакунок за допомогою GPG не потрібно. Результат буде поміщений у каталог «..».

Продивитися вміст пакунку можна за допомогою:

```
$ lesspipe hello_2.10-0ubuntu1_amd64.deb
```

Встановіть пакунок й перевірте, що він працює (пізніше за бажання Ви зможете вилучити його командою `sudo apt-get remove hello`):

```
$ sudo dpkg --install hello_2.10-0ubuntu1_amd64.deb
```

Можете також встановити усі пакунки одразу за допомогою:

```
$ sudo debi
```

### 1.4.4 Наступні кроки

Навіть якщо двійковий .deb-пакунок успішно збирається, Ваші джерельні файли для пакунку можуть містити помилки. Багато помилок можуть автоматично виявлятися нашим інструментом `lintian`, який можна застосувати до файлу метаданих .dsc, двійкових пакунків .deb або файлу .changes:

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

Щоб побачити детальний опис вад, використовуйте прапорець lintian `--info` або команду `lintian-info`.

Для пакунків Python є також інструмент `lintian4python`, здійснюючий деякі додаткові перевірки lintian.

Після створення виправлення для файлів пакунку можна перезібрати його з параметром `-nc` (“no clean”), щоб не починати збірку з самого початку:

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

Переконавшись, що пакунок успішно збирається локально, Ви повинні перевірити, чи правильно його збирання буде відбуватися у чистій системі, за допомогою `pbuilder`. Оскільки незабаром ми збираємося завантажити його в PPA (персональний архів пакунків), це завантаження потрібно забезпечити *цифровим підписом*, щоб Launchpad міг переконатися, що завантаження зробили Ви (дізнатися про те, що завантаження буде підписане, можна за відсутністю передаваних `bzr builddeb` прапорців `-us` і `-uc`, які ми використовували раніше). Для підписування своєї роботи Вам знадобиться налаштувати GPG. Якщо Ви ще не налаштували `pbuilder-dist` або GPG, *зробіть це зараз*:

```
$ bzr builddeb -S
$ cd ../build-area
$ pbuilder-dist trusty build hello_2.10-0ubuntu1.dsc
```

Після того як Ви залишитеся задоволені отриманим пакунком, потрібно, щоб його перевірили інші люди. Ви можете вивантажити гілку на Launchpad для перевірки:

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

Вивантаження в PPA дозволить переконатися, що пакунок збирається нормально, а також дозволить Вам і решті тестувати бінарні пакунки. Вам знадобиться створити PPA на Launchpad, після чого вивантажити пакунок за допомогою `dput`:

```
$ dput ppa:<lp-username>/<ppa-name> hello_2.10-0ubuntu1.changes
```

Попрохати провести review можна на каналі IRC `#ubuntu-motu`, або у переліку розсилки MOTU. У деяких випадках може знадобитися участь конкретної команди: у подібних випадках команда GNU допоможе розібратися.

### 1.4.5 Відправка на включення

Існує декілька шляхів, якими пакунок може потрапити в Ubuntu. У більшості випадків кращим шляхом може бути проходження спочатку через Debian. Це дозволить Вашому пакунку стати доступним для щонайбільшої кількості користувачів, оскільки він буде доступний не лише в Debian та Ubuntu, але й в усіх дистрибутивах, створених на їх основі. Ось декілька корисних посилань з відправки нових пакунків в Debian:

- Debian Mentors FAQ - debian-mentors створений для менторства нових і перспективних розробників Debian. Це те місце, де можна знайти спонсора для завантаження Вашого пакунку в архів.
- Work-Needing and Prospective Packages - інформація про те як відправляти баги “Intent to Package” (Призначення пакунку) і “Request for Package” (Запит пакунку), а також переліки відкритих ITP та RFP.
- Посібник Розробника Debian, 5.1. Створення пакунків - безцінний документ для творців пакунків як під Ubuntu, так й під Debian. Даний розділ дає опис процесу відправки нових пакунків.

У деяких випадках має сенс відправлятися прямо в Ubuntu. Наприклад, якщо Debian знаходиться у стані “freeze”: тоді Ваш пакунок навряд встигне увійти в Ubuntu до найближчого релізу. Опис цього процесу на сторінці “Нові Пакунки” Ubuntu Wiki.

### 1.4.6 Знятки екрану

Завантаживши пакунок в Debian, Вам слід додати знятки екрану, аби майбутні користувачі змогли отримати уяву про те, як виглядає інтерфейс проґрами. Знятки потрібно відправляти на http://screenshots.debian.net/upload .

# 1.5 Оновлення безпеки й оновлення стабільних релізів

## 1.5.1 Виправлення помилок безпеки в Ubuntu

### Вступ

Виправлення дірок у безпеці в Ubuntu фактично не відрізняється від *виправлення звичайного багу*, і припускається, що Ви знайомі з виправленням звичайних багів. Для демонстрації відмінностей ми будемо додавати у пакунок dbus в Ubuntu 12.04 LTS (Precise Pangolin) оновлення для системи безпеки.

### Отримання джерельного коду

У даному прикладі ми вже знаємо, що бажаємо виправити пакунок dbus в Ubuntu 12.04 LTS (Precise Pangolin). Тому спочатку потрібно визначити версію пакунку, який бажаєте стягнути. Ми можемо використовувати `rmadison` у якості допомоги у даній ситуації.

```
$ 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
```

Зазвичай обирають найостаннішу версію для релізу, який Ви бажаєте залатати, який не в -proposed або -backports. Оскільки ми оновлюємо dbus Precise, Ви стягнете 1.4.18-1ubuntu1.4 з precise-updates:

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

### Створення латки

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.

Щоб створити латку за допомогою `edit-patch`:

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

Це застосує усі існуючі латки й помістить пакунок у тимчасовий каталог. Тепер відредагуйте файли для виправлення вразливостей. Зазвичай в апстрімі лежить і латка, тому Ви одразу можете її застосувати:

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

Після внесення необхідних змін просто натисніть Ctrl-D або наберіть exit, щоб залишити тимчасову командну оболонку.

### Форматування файлу changelog і латок

Після застосування латок Вам знадобиться внести зміни в лог. Команда `dch` використовується для редагування файлу `debian/changelog` і `edit-patch` автоматично запустить `dch` після відкату усіх латок. Якщо Ви не користуєтеся `edit-patch`, то можете запустити `dch -i` вручну. На відміну від звичайних латок, Вам слід використовувати наступний формат (зверніть увагу, що в імені дитрибутиву використовується precise-security, оскільки це оновлення безпеки для Precise) для оновлень безпеки:

```
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]
...
```

Оновіть свою латку для використання відповідних тегів. Ваша латка повинна містити як мінімум теги Origin, Description і Bug-Ubuntu. Наприклад, відредагуйте `debian/patches/99-fix-a-vulnerability.patch`, щоб він мав приблизно такі рядки:

```
## 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
...
```

Множинні вразливості можна виправити одним завантаженням безпеки, просто переконайтеся що використовуєте різні латки для різних вразливостей.

#### Перевірка й відправка Вашої праці

На цьому етапі процес такий самий, як при *виправленні звичайних вад в Ubuntu*. А саме, Вам потрібно:

1. Виконати збірку пакунку й перевірити, що він компілюється без помилок і компілювальник не видає ніяких додаткових застережень
2. Виконати оновлення з попередньої версії пакунку до нової версії
3. Переконатися, що новий пакунок латає вразливість й не вносить ніяких погіршень
4. Відправляйте свою працю через пропозицію про об'єднання Launchpad й відправляйте баг в Launchpad, переконавшись що позначили баг як помилку безпеки, й для підписки `ubuntu-security-sponsors`

Якщо це вразливість у безпеці, про яку ще не оголошено публічно, то не відправляйте пропозицію злиття й переконайтеся, що Ви позначили свою помилку, як приватну (private).

Відправлений баг повинен містити Тестовий Приклад, тобто коментар, який чітко показує як відтворити баг, запустивши стару версію, також показуючи як переконатися, що баг більше не існує у новій версії.

Звіт про баг також повинен підтверджувати, що вада виправлена у нових версіях Ubuntu за допомогою запропонованого фіксу (у вищевказаному прикладі вище ніж в Precise). Якщо проблема не виправлена у нових версіях Ubuntu, Ви повинні підготувати оновлення й для нових версій.

### 1.5.2 Оновлення стабільного релізу

Ми також дозволяємо вносити оновлення у випуски, у яких пакунок містить серйозну помилку, таку як значна регресія у порівнянні з попереднім випуском або вада, яка може призвести до втрати даних. З-за того, що такі зміни самі потенційно можуть призвести до появи додаткових помилок, ми дозволяємо робити це лише там, де зміни легко можна зрозуміти та перевірити.

Процес оновлень стабільного випуску (Stable Release Updates або SRU) такий самий, як і для виправлень помилок безпеки, за виключенням того, що потрібно підписати на звіт про ваду команду `ubuntu-sru`.

Оновлення потрапить в архів `proposed‘‘(наприклад ‘‘precise-proposed`), де його перевіряють на здатність виправити проблему й підтверджують, що воно не є наслідком нових проблем. Після тижню роботи без заявлених проблем, оновлення потрапляє у розділ `updates`.

Дивіться ‘Вікі сторінку Оновлень Стабільного Релізу <SRUWiki>‘_ для отримання додаткової інформації.

## 1.6 Латки для пакунків

Інколи розробникам пакунку Ubuntu потрібно змінити джерельний код апстріму, щоб примусити його працювати в Ubuntu належним чином. Приклади включають латки для апстрімів, які ще не потрапили у версію релізу, або зміни до систем білдів апстріму, необхідні лише для їх збірки на Ubuntu. Ми будемо міняти джерельний код апстріму напряму, але такий метод робить складнішим подальше вилучення латок, коли апстрім вже застосував їх, також ускладнюючи витягнення змін для їх відправки у проєкт апстріму. Замість цього, ми будемо зберігати ці зміни як окремі латки у формі diff файлів.

Існують різні способи роботи з латками для пакунків Debian. На щастя, ми зупинимося на одній системі, Quilt, яка наразі використовується більшістю пакунків.

Давайте візьмемо у якості прикладу пакунок `kamoso` в Trusty:

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

Латки зберігаються в `debian/patches`. Для цього пакунку є одна латка `kubuntu_01_fix_qmax_on_armel.diff` для виправлення вади компілювання на платформі ARM. Цій латці привласнено ім’я, що дає опис, того що вона робить, номер патчу за порядком (щоб уникнути плутанини, якщо дві латки мають однакове ім’я) й, у даному випадку, команда Kubuntu додала свій власний префікс, щоб показати, що латка походить від них, а не від Debian.

Порядок застосування латок зберігається в `debian/patches/series`.

### 1.6.1 Латки за допомогою Quilt

Перед роботою з Quilt потрібно вказати цій системі, де шукати латки. Додайте в `~/.bashrc` таке:

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

Й джерело файлу для застосування нового експорту:

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

Типово усі латки застосовуються вже з UDD витягнень або завантажуваних пакунків. Ви можете перевірити це за допомогою:

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

Якщо Ви бажаєте вилучити латку, потрібно виконати `pop`:

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

```
No patches applied
```

А щоб застосувати латку, використовуйте `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 Додавання нової латки

Для додавання нової латки потрібно вказати Quilt створити нову латку, повідомити йому, які файли ця латка повинна змінити, відредагувати файли, а потім оновити латку:

```
$ 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
```

Крок `quilt add` важливий: якщо Ви забудете його зробити, файли не потраплять у латку.

Тепер зміни будуть в `debian/patches/kubuntu_02_program_description.diff`, а у файл `series` буде додана інформація про нову латку. Ви повинні додати новий файл у джерельні файли для пакунку:

```
$ 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 містить свої мета-дані у директорії `.pc/`, тому зараз Вам потрібно додати у пакунок і її. Це має бути покращено у майбутньому.

Як правило, слід проявляти обережність при додаванні латок до програм, якщо вони походять не з апстріму. Часто є вагома причина, з якої зміни ще не були зроблені. У розглянутому вище прикладі змінюється рядок у користувацькому інтерфейсі, тож це може зламати усі переклади. Якщо маєте сумнів, спитайте автора з апстріму перед тим, як додати латку.

### 1.6.3 Заголовки латок

Ми рекомендуємо додавати до кожної латки заголовки DEP-3, поміщаючи їх у самому верху файлу латки. Ось деякі заголовки, які можна використовувати:

**Description** Опис того, що робить латка. Має формат, аналогічний полю `Description` в `debian/control`: перший рядок містить короткий опис, що починається з рядкової літери, решта рядків містять довший опис з пробілом у якості відступу.

**Author** Хто написав латку (наприклад, "Jane Doe <packager@example.com>").

**Origin** Звідки прийшла ця латка (наприклад, "upstream"), якщо заголовок *Author* відсутній.

**Bug-Ubuntu** Посилання на інформацію про помилку на Launchpad, бажано, у короткій формі (типу *https://bugs.launchpad.net/bugs/XXXXXXX*). Якщо також є звіти про вади

в апстрімі або системах відстежування помилок Debian, додайте заголовки *Bug* або *Bug-Debian*.

**Forwarded** Чи була латка передана в апстрим. Значення: “yes”, “no” або “not-needed”.

**Last-Update** Дата останньої ревізії (у формі “ГГГГ-ММ-ДД”).

### 1.6.4 Оновлення до нових версій з апстріму

Щоб виконати оновлення до останньої версії, Ви можете використовувати 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 запуску цієї команди відбудеться відткат усіх латок, оскільки вони можуть стати застарілими. Можливо знадобиться їх оновити для відповідності новому джерелу апстріму, або знадобиться вилучити їх усі разом. Для полегшення перевірки проблем застосовуйте латки по одній.

```
$ 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
```

Якщо для латки вказано `it can be reverse-applied`, значить латку вже було застосовано апстрімом, тож ми можемо вилучити цю латку:

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

Потім продовжуйте:

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

Непоганою думкою буде виконати refresh, це оновить латку відносно змін джерельного коду в апстрімі:

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

Потім виконайте фіксацію, як звичайно:

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

### 1.6.5 Створення пакунку з використанням Quilt

Сучасні пакунки використовують Quilt типово, це вбудовано у формат джерельних файлів пакунку. Перевірте, що в `debian/source/format` вказано `3.0 (quilt)`.

Для старіших пакунків, що використовують формат 1.0, необхідно використовувати Quilt явно, зазвичай за допомогою включення make-файлу у `debian/rules`.

### 1.6.6 Конфігурування Quilt

Ви можете скористатися файлом `~/.quiltrc` для конфігурування quilt. Ось декілька опцій, які можуть бути корисні при використанні quilt з пакунками 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 Інші системи керування латками

Інші системи латання, що використовуються у пакунках, включають `dpatch` і `cdbs simple-patchsys`, принцип роботи яких схожий на Quilt - латки зберігаються в `debian/patches`, але для їх застосування, скасування або створення потрібні інші команди. Ви можете дізнатися яка система латання використовується у пакунку за допомогою команди `what-patch‘‘(у пакунку ‘‘ubuntu-dev-tools`). Ви можете використовувати `edit-patch`, показаний в *попередніх розділах*, у якості надійного способу для роботи з усіма системами.

У старіших пакунках зміни будуть включені напряму у джерела і зберігатися у джерельному файлі `diff.gz`. Це робить складнішим процес оновлення до нових версій апстріму або відмінності між латками - ліпше уникати.

Не змінюйте систему латання, не обміркувавши це з супроводжуючим Debian або командою Ubuntu яка має відношення до справи. Якщо існуючої системи латання немає, можете додати Quilt.

## 1.7 Виправлення пакунків FTBFS (Fails To Build From Source)

Перед тим, як пакунок можна буде використовувати в Ubuntu, він повинен бути зібраний з джерельного коду. Якщо це не вдається, пакунок, ймовірно, буде очікувати в -proposed й не буде доступний у архівах Ubuntu. Повний перелік пакунків, які не вдалося зібрати з джерельного коду, можна знайти на http://qa.ubuntuwire.org/ftbfs/. На цій сторінці показано 5 основних категорій:

- Package failed to build (F): Щось справді пішло не так у процесі збірки.
- Скасована збірка (X): збірка була скасована з певної причини. Для початку, з ними ліпше не зв’язуватися.
- Package is waiting on another package (M): Цей пакунок очікує збірки або оновлення іншого пакунку, або (якщо це пакунок в main) одна з його залежностей знаходиться не у тій частині архіву.
- Проблема в chroot (C): Певна операція над chroot-оточенням зіштовхнулася з помилкою. Частіше за усе виправляється повторним збиранням. Попрохайте розробника запустити перезбірку.
- Помилка при завантаженні (U): Пакунок не може бути завантажений на сервер. Зазвичай у цьому випадку потрібно зробити перезбирання, але перед цим – перевірте логи збірки.

### 1.7.1 Перші кроки

Найперше необхідно повторити FTBFS самостійно. Стягніть код за допомогою `bzr branch lp:ubuntu/PACKAGE` й отримайте tar-архів, або запустіть `dget PACKAGE_DSC` над .dsc-файлом зі сторінки проєкту на Launchpad. Після цього, створіть chroot-оточення.

В Вас має вийти відтворити помилку FTBFS. Якщо ж ні – перевірте, чи не стягує збірка відсутню залежність: у такому випадку необхідно у файлі debian/control оголосити її як build-depends. Інший варіянт – спробувати зібрати пакунок локально, що дозволить перевірити відсутні або не вказані залежності (у такому випадку локальна збірка повинна бути успішна, а в schroot – ні)

### 1.7.2 Перевірка Debian

У випадку, якщо проблему вдалося відтворити – необхідно почати пошук вирішення. Якщо пакунок також знаходиться в Debian, – перевірте, можливо в них пакунок збирається нормально: http://packages.qa.debian.org/PACKAGE. Якщо в Debian є новіша версія пакунку, його потрібно об'єднати (merge). Якщо ж ні – перевірте логи збірки й посилання на відомі проблеми: там може бути додаткова інформація про FTBFS або латки. Debian також підтримує перелік команд різних FTBFS, у якому також є варіянти вирішення різноманітних проблем: https://wiki.debian.org/qa.debian.org/FTBFS.

### 1.7.3 Інші причини виникнення FTBFS

Якщо пакунок знаходиться в main, але для нього відсутня залежність не з main, то необхідно відправити MIR-баг: сторінка https://wiki.ubuntu.com/MainInclusionProcess дає опис цієї процедури.

### 1.7.4 Виправлення помилки

Якщо вдалося виправити проблему, дотримуйтеся такої ж процедури як й при будь-яких інших проблемах: створіть латку, додайте її у гілку або баг bzr, підпишіть ubuntu-sponsors, а потім спробуйте домогтися її додавання у джерельний пакунок і/або в Debian.

## 1.8 Спільні бібліотеки

Спільні бібліотеки — це скомпільований код, призначений для спільного використання декількома різними проґрамами. Вони розповсюджуються у вигляді файлів `.so` в `/usr/lib/`.

Бібліотеки експортують символи у скомпільованому вигляді: функції, кляси та змінні. В кожної бібліотеки також є назва SONAME, що включає номер її версії, але який не обов'язково збігається з офіційним номером релізу. Проґрами компілюються з конкретним SONAME бібліотеки. Так, якщо якийсь з символів бібліотеки був вилучений або змінений – небхідно змінити версію з тим, щоб усі залежні від бібліотеки пакунки були перекомпільовані з використанням нової версії. Зазвичай версії встановлюються у джерелі, й ми використовуємо ті ж номери версій для двійкових пакунків, що називаються "номер ABI", але у випадку, якщо джерело не використовує притомної версійності, творці пакунків можуть використовувати окрему, традиційнішу нумерацію.

Бібліотеки зазвичай розповсюджуються апстрімом у вигляді окремих випусків. Інколи вони розповсюджуються, як частина проґрами. У останньому випадку вони можуть бути включені у двійковий пакунок разом з проґрамою (це зветься bundling), якщо Ви не припускаєте використання цих бібліотек іншими проґрамами, але частіше їх усе ж слід виділяти у окремі двійкові пакунки.

Самі бібліотеки поміщаються у двійковий пакунок з іменем `libfoo1`, де `foo` — ім'я бібліотеки, а `1` — версія з SONAME. Файли розробки з пакунку, такі як заголовкові файли, необхідні для компілювання проґрам з бібліотекою, поміщаються у пакунок з іменем `libfoo-dev`.

### 1.8.1 Приклад

У якості прикладу ми використовуємо libnova:

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

Щоб знайти SONAME бібліотеки, виконайте:

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

SONAME у даному випадку `libnova-0.12.so.2`, що відповідає імені файлу (як правило, але не завжди). Тут апстрім помістив номер версії з апстріму, як частину SONAME, й задав ABI-версію 2. Імена бібліотекових пакунків повинні слідувати SONAME бібліотеки, яку вони містять. Двійковий бібліотековий пакунок зветься `libnova-0.12-2`, де `libnova-0.12` — ім'я бібліотеки, а 2 — наш ABI-номер.

Якщо автори з апстріму внесли несумісні зміни у свою бібліотеку, вони повинні змінити номер версії SONAME, а ми повинні перейменувати нашу бібліотеку. Будь-які інші пакунки, що використовують наш бібліотековий пакунок, потрібно буде перекомпілювати з новою версією, це зветься переходом (transition) й потребує певних зусиль. Потрібно сподіватися, наш ABI-номер продовжить відповідати SONAME апстріму, але інколи вони вносять несумісності без зміни їх номеру версії, а нам потрібно змінити наш.

Поглянувши на debian/libnova-0.12-2.instal, ми побачимо, що він включає у себе два файли:

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

Другий рядок — справжня бібліотека, з повним номером версії. Перше — символьне посилання, що вказує на справжню бібліотеку. Проґрами, що використовують бібліотеку, как правило, будуть користуватися символічним посиланням.

`libnova-dev.install` містить усі файли, необхідні для компілювання проґрами з даною бібліотекою. Заголовкові файли, бінарник конфігурації, файл libtool'а `.la`, а також `libnova.so` – ще один симлінк на бібліотеку, створюваний з тим, щоб проґрами могли компілюватися поза залежністю від старшого номеру версії (при цьому, скомпільваний застосунок все одно буде залежати від версії).

`.la`-файли libtool'у потрібні на деяких не-Linux системах з обмеженою підтримкою бібліотек, але на системах Debian часто створюють більше проблем, ніж вирішують. Мета Debian відмовитися від .la-файлів сьогодні вельми актуальна, й Ви можете допомогти з вирішенням цього завдання.

### 1.8.2 Статичні бібліотеки

Пакунок -dev також містить `usr/lib/libnova.a`. Це статична бібліотека, альтернатива спільній бібліотеці. Будь-яка проґрама, скомпільована зі статичною бібліотекою, містить її код безпосередньо у собі. Це дозволяє не бентежитися про двійкову сумісність бібліотеки. Втім це також значить, що будь-які помилки, у тому числі вразливості у безпеці, не будуть виправлені за рахунок оновлення бібліотеки, поки проґрама не буде перекомпільована. З цієї причини використовувати проґрами з статичними бібліотеками не радимо.

### 1.8.3 Символьні файли

Коли додаток компілюється з бібліотекою, механізм `shlibs` додасть до пакунку залежність від цієї бібліотеки. Саме тому більшість проґрам містять `Depends: ${shlibs:Depends}` у файлі `debian/control`. Це замінюється переліком залежних бібліотек при білді. Втім shlibs може лише вказувати залежність від старшої ABI-версії , 2 у нашому прикладі з libnova, тож якщо нові символи будуть додані у майбутній libnova 2.1 – застосунок буде запускатися й зі старішою версією libnova ABI 2.0, що призведе до аварійного завершення.

Щоб точніше вказувати залежності від бібліотек, ми створили файл `.symbols`, який перераховує усі символи бібліотеки й версії, у яких вони з'явилися.

libnova не має символьного файлу, тож ми можемо створити його. Почніть з компілювання пакунку:

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

Опція `-nc` вказує не вилучати збірочні файли після завершення компілювання. Перейдіть у каталог збірки й виконайте `dpkg-gensymbols` для пакунку бібліотеки:

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

Це створить diff-файл, який Ви зможете застосувати самостійно:

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

Це створить файл з іменем вигляду `dpkg-gensymbolsnY_WWI`, у якому будуть перераховані усі символи. Він також вказує поточну версію пакунку. Версію пакунку можна прибрати з файлу, оскільки нові символи зазвичай додаються не з новою версією пакунку, а розробниками початкової бібліотеки.

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

Тепер перемістіть файл туди, де він має знаходитися, зафіксуйте зміни та виконайте тестову збірку:

```
$ 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
```

Якщо компілювання виконується успішно, значить символьний файл не містить вад. З виходом наступної апстрім-версії libnova Вам доведеться знову запустити dpkg-gensymbols, щоб створити diff для оновлення символьного файлу.

### 1.8.4 Символьні файли бібліотек C++

В мови C++ більш суворі стандарти на двійкову сумісність, ніж у C. Команда Debian Qt/KDE підтримує деякі скрипти, які допоможуть подолати це: сторінка Робота з файлами symbols дає опис принципів їх використання.

### 1.8.5 Матеріяли для подальшого читання

Стаття Junichi Uekawa Пакунковання бібліотек для Debian розглядає це питання детальніше.

## 1.9 Бекпортування оновлень проґрам

Буває може знадобитися додати функційності у стабільный реліз, який не пов'язаний з виправленням критичних проблем. У подібних випадках, є два варіянти: або Ви завантажите його в PPA, або підготуєте бекпорт (backport).

### 1.9.1 Персональні архіви пакунків (PPA)

Використання PPA має ряд переваг. Це достатньо просто, Вам не знадобиться схвалення від кого б то не було, але недолік у тому, що користувачам доведеться вручну під'єднувати PPA. Це нестандартне джерело застосунків.

Документація до PPA на Launchpad має достатньо всеосяжний характер й допоможе Вам швидко почати роботу з ним.

### 1.9.2 Офіційні бекпорти Ubuntu

Метою проєкту Backports є надання користувачам нової функційності. З-за ризиків зменшення стабільності при портуванні новинок, бекпорти недоступні користувачам, поки вони не задіють їх. Тому бекпорти не є місцем для виправлення помилок. Якщо у пакунку Ubuntu виявлена помилка, вона повинна бути виправлена через *оновлення безпеки та стабільності*.

Коли Ви визначите, чи потрібно Вам адаптувати Ваші зміни для стабільного релізу, Вам буде необхідно зібрати та протестувати Ваш пакунок на даному релізі. Команда `pbuilder-dist` (з пакунку `ubuntu-dev-tools`) допоможе Вам зробити це.

Щоб подати заявку на бекпорт, можна використовувати засіб `requestbackport` (також з пакунку `ubuntu-dev-tools`). Він визначить усі проміжні випуски, для яких пакунок також доведеться бекпортувати, покаже, які пакунки залежать від даного, й створить заявку. Він також включить перелік потрібних тестів у заявку.

Розділ 2

# База знань

## 2.1 Комунікація при Розробці в Ubuntu

В проєкті, де піддаються змінам тисячі рядків коду, приймається багато рішень, й де сотні людей повинні взаємодіяти кожен день, важливо мати ефективний зв'язок.

### 2.1.1 Поштові розсилки

Поштові розсилки— це достатньо ефективний інструмент, якщо Ви бажаєте обміркувати ідеї в команді та переконатися що Ви сповістили усіх, не дивлячись на відмінності у часових поясах.

З точки зору розробки, це найважиливіші з розсилок:

- https://lists.ubuntu.com/mailman/listinfo/ubuntu-devel-announce (лише анонси, найважливіші об'-яви розробки потрапляють сюди)
- https://lists.ubuntu.com/mailman/listinfo/ubuntu-devel (головна дискусія розробників Ubuntu )
- https://lists.ubuntu.com/mailman/listinfo/ubuntu-motu (обговорення команди MOTU, отримання довідки зі створення пакунків)

### 2.1.2 Канали IRC

Для дискусії у мережі під'єднайтеся до irc.freenode.net і приєднайтеся до будь-якого з каналів:

- #ubuntu-devel (для головної дискусії розробників)
- #ubuntu-motu (для обговорень команди MOTU та отримання допомоги)

## 2.2 Загальний огляд каталогу `debian/`

Ця стаття дає короткі пояснення до різних файлів, важливих для створення пакунків Ubuntu, які містяться у каталозі `debian/`. Найважливішими з них є `changelog`, `control`, `copyright`, і `rules`. Вони потрібні для усіх пакунків. Багато додаткових файлів у `debian/` можуть використовуватися для налаштування та зміни поведінки пакунку. Деякі з цих файлів обговорюються у цій статті, але це далеко не повний перелік.

### 2.2.1 Файл changelog

Цей файл, як видко з його назви — це перелік змін, внесених у кожну версію. Він має особливий формат, який показує ім'я пакунку, версію, дистрибутив, зміни, й хто вносив зміни в даний час. Якщо в Вас є ключ GPG (дивіться: *Підготовка*), переконайтеся, що Ви використовуєте в `changelog` ті ж ім'я та адресу електронної пошти, що й в Вашому ключі. Нижче наведено шаблон `changelog`:

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

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

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

Формат (особливо дати) важливий. Дата повинна бути у форматі **RFC 5322**, який можна побачити при виконанні команди `date -R`. Для зручності, можна використовувати для редагування changelog команду `dch`. Вона оновить дату автоматично.

Пункти з незначними змінами позначаються тире «-», у той час як у важливих пунктах використовується зірочка «*».

Якщо Ви створюєте пакунок «з нуля», `dch --create` (`dch` знаходиться у пакунку `devscripts`) створить для Вас стандартний файл `debian/changelog`.

Ось приклад файлу `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
```

Зверніть увагу, що версія має `-0ubuntu1` доданий до нього, це - distro версія, яка використовується так, щоб упаковка могла бути оновлена (щоб виправити помилки, наприклад) з новими завантаженнями у тій же джерельній версії випуску.

Ubuntu і Debian використовують схеми нумерації версій пакунків, що трохи різняться, щоб уникнути конфлікту пакунків з однією й тою самою початковою версією. Якщо пакунок Debian був змінений в Ubuntu, до кінця Debian-версії додається `ubuntuX` (де `X` — номер редакції в Ubuntu). Таким чином, якщо пакунок Debian hello `2.6-1` був змінений в Ubuntu, номер версії буде `2.6-1ubuntu1`. Якщо пакунок додатку в Debian не існує, то редакція Debian дорівнює `0` (наприклад, `2.6-0ubuntu1`).

Детальнішу інформацію можна знайти на сторінці changelog (Розділ 4.4) документу Debian Policy Manual.

### 2.2.2 Файл control

Файл `control` містить інформацію, яку використовує менеджер пакунків (такий, як `apt-get`, `synaptic` або `adept`), збірочні залежності, інформацію мейнтейнера і багато іншого.

Для пакунку Ubuntu `hello` файл `control` виглядає таким чином:

```
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).
```

Перший абзац дає опис джерельного пакунку, включаючи перелік пакунків, що потрібні для збірки даного пакунку з джерельного коду, у полі `Build-Depends`. Він також містить деяку метаінформацію, таку як ім'я мейнтейнера, версію Debian Policy, з якою компілюється пакунок, місцерозташування репозиторію керування версіями та домашню сторінку апстріму.

Зауважте, що в Ubuntu ми вказуємо у полі `Maintainer` загальну адресу, оскільки будь-яка людина може змінити будь-який пакунок (на відміну від Debian, де правом змінювання пакунків володіють лиши окремі люди або команда). Пакунки в Ubuntu, як правило, повинні у полі `Maintainer` містити `Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>`. Якщо поле Maintainer змінено, старе значення повинне бути збережене у полі `XSBC-Original-Maintainer`. Це можна зробити автоматично сценарієм `update-maintainer` з пакунку `ubuntu-dev-tools`. Для подальшої інформації дивіться Debian Maintainer Field spec в Ubuntu wiki.

Кожен додатковий абзац дає опис бінарного пакунку, який буде створений.

Детальнішу інформацію можна знайти на сторінці секції control-файлу (Розділ 5) документу Debian Policy Manual.

### 2.2.3 Файл copyright

Файл містить інформацію про копірайти джерельних кодів і самого пакунку. Ubuntu і Debian Policy (Розділ 12.5) потребують, щоб кожен пакунок встановлював незмінну копію копірайту та інформації з ліцензування у теку `/usr/share/doc/$(ім'я_пакунку)/copyright`

Як правило, інформацію про авторскькі права можна знайти у файлі `COPYING` в каталозі з джерельним кодом програми. Цей файл повинен включати таку інформацію, як імена автора та пакувальника, URL, з якої отримано джерело, рядок зі значком копірайту з вказівкою року і власника авторських прав, а також сам текст авторського права. Шаблон для прикладу:

```
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’.
```

Приклад використовує машино-зрозумілий формат debian/copyright, й авторам пакунків також рекомендується використовувати цей формат.

### 2.2.4 Файл rules

Останній файл, який ми розглянемо, це `rules`. Він виконує усю роботу по збірці нашого пакунку. Це Makefile, у якому є функції компілювання проґрами, її встановлення та створення `.deb`-пакунку з встановлених файлів. У ньому є також функція очистки файлів збірки, яка вилучає усе, крім власне пакунку джерельного коду.

Ось спрощений приклад файлу rules, створеного `dh_make` (який можна знайти у пакунку `dh-make`):

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

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

%:
	dh $@
```

Давайте розглянемо цей файл уважніше. На кожному етапі збірки `debian/rules` викликається з аргументом, який передається `/usr/bin/dh`, який, у свою чергу, викликає необхідні команди `dh_*`.

`dh` запускає послідовність команд debhelper. Підтримувані послідовності відповідають призначенням файлу `debian/rules`: «build», «clean», «install», «binary-arch», «binary-indep» і «binary». Щоб побачити, які команди виконуються у кожному призначенні, наберіть:

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

Командам з послідовності `binary-indep` передається аргумент `-i`, щоб вони зачепали лише архітектурно-незалежні пакунки, а командам з послідовності `binary-arch` — аргумент `-a`, щоб вони зачепали лише архітектурно-залежні пакунки.

Кожна команда debhelper при її успішному виконанні робить запис у журналі `debian/package.debhelper.log` (який потім вилучає dh_clean.) Таким чином dh може визначи-

ти, які команди вже були виконані й для яких пакунків, що допомогає уникнути повторного виконання цих команд.

При кожному запуску dh він вивчає журнал, знаходить додані останніми команди, які відносяться до вказаної послідовності. Потім він продовжує виконання з наступної команди у цій послідовності. Опції --until, --before, --after і --remaining можуть змінити цю поведінку.

Якщо в debian/rules є функція з іменем, схожим на override_dh_команда, то замість даної команди dh виконає дану функцію. Ця функція може запустити ту ж команду з іншими аргументами, або ж геть іншу команду. (Зауваження: для використання цієї функційності при збірці потрібен пакунок debhelper версії не менше 7.0.50).

За додатковими прикладами зверніться до /usr/share/doc/debhelper/examples/ і man dh. Дивіться також розділ про файл rules (Розділ 4.9) в «Debian Policy Manual».

### 2.2.5 Додаткові файли

#### Файл install

Файл install використовується dh_install для встановлення файлів у двійковий пакунок. Він має два стандартних варіянти використання:

- Для встановлення у Ваш пакунок файлів, не встановлених оригінальною системою збірки.
- Розділення одного великого пакунку джерела на декілька бінарних пакунків.

У першому випадку файл install повинен містити один рядок для кожного встановлюваного файлу, що вказує як файл, так і встановлювальний каталог. Наприклад, наступний файл install встановить сценарій foo з кореневого каталогу пакунку джерельного коду в usr/bin і desktop-файл з каталогу debian в usr/share/applications:

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

Якщо пакунок джерельного коду продукує декілька двійкових пакунків, dh встановить файли в debian/tmp замість встановлення безпосередньо в debian/<пакунок>. Файли, встановлені в debian/tmp потім можна перемістити у окремі двійкові пакунки за допомогою декількох файлів $ім'я_пакунку.install. Це часто робиться, щоб розбити велику кількість незалежних від архітектури даних з залежних від архітектури пакунків у пакунки Architecture: all. У цьому випадку потрібно вказати лише імена встановлюваних файлів (або каталогів), без встановлювального каталогу. Наприклад, foo.install, що містить лише залежні від архітектури файли, може виглядати наподобі:

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

У той час як foo-common.install, що містить лише не залежні від архітектури файли, може виглядати так:

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

Будуть створені два двійкові пакунки: foo і foo-common. Для обох потрібен їх власний абзац в debian/control.

Для додаткових подробиць дивіться man dh_install і розділ про файл install (Розділ 5.11) в «Debian New Maintainers' Guide».

### Файл watch

Файл `debian/watch` дозволяє автоматично перевіряти наявність нових версій в апстрімі за допомогою інструменту `uscan` з пакунку `devscripts`. Першим рядком файлу watch повинна бути версія формату (3 на мить написання цього посібника), а наступні рядки містять будь-які URL для аналізу. Наприклад:

```
version=3

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

Запуск `uscan` у кореневому каталозі джерельних кодів порівняє номер апстрім-версії у `debian/changelog` з останньою доступною в апстрімі версією. Якщо в апстрімі знайдена нова версія, вона буде автоматично завантажена. Наприклад:

```
$ 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
```

Якщо Ваші tarball-файли перебувають на Launchpad, файл `debian/watch` має трохи складніший вигляд (про те, чому це так, дивіться Question 21146 і Bug 231797). У цьому випадку використовуйте щось типу:

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

Додаткові відомості дивіться в `man uscan` і у розділі про файл watch (Розділ 4.11) «Debian Policy Manual».

Перелік пакунків, для яких файл `watch` повідомляє про те, що вони не синхронізовані з апстрімом, дивіться Ubuntu External Health Status.

### Файл source/format

Цей файл вказує формат пакунку джерельного коду. Він повинен містити один рядок, що показує вибраний формат:

- `3.0 (native)` для «рідних» пакунків Debian (апстрім-версія відсутня)
- `3.0 (quilt)` для пакунків з окремим тарболом з апстріму
- `1.0` для пакунків, бажаючих явно вказати типовий формат

На цей час обирається типовий формат пакунку джерельного коду 1.0, якщо цей файл відсутній. У файлі source/format можна вказати його явно. Якщо Ви не використовуєте цей файл для вказування формату джерельного коду, Lintian видасть попередження про відсутність файлу. Це чисто інформаційне попередження і його можна без побоювань знехтувати.

Рекомендується використовувати більш новий формат 3.0. Він надає деякі нові можливості:

- Підтримка додаткових форматів стиснення: bzip2, lzma і xz
- Підтримка декількох архівів з оригінальним джерельним кодом
- Необов'язково перепаковувати архів з оригінальним джерельним кодом, щоб вилучити директорію `debian`.
- Специфічні для Debian зміни тепер зберігаються не в одному файлі .diff.gz, а у вигляді декількох латок, сумісних з quilt, у каталозі `debian/patches/`

https://wiki.debian.org/Projects/DebSrc3.0 містить додаткову інформацію стосовно переходу на версію 3.0 формату джерельних пакунків.

Додаткову інформацію можна знайти в `man dpkg-source` і у Розділі source/format (Розділ 5.21) посібника Debian New Maintainers.

### 2.2.6 Додаткові ресурси

Крім Debian Policy Manual, на який посилається стаття, посібник Debian New Maintainers' Guide містить більш детальний опис для кожного файлу. Розділ 4, "Необхідні файли у теці debian" дає опис файлів control, changelog, copyright і rules. Розділ 5, "Інші файли у теці debian" дає опис додаткових файлів, які можна використовувати.

## 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: Автоматичне тестування пакунків

Специфікація DEP 8 визначає, як можна легко інтегрувати автоматичне тестування у Ваші пакунки. Для цього, необхідно:

- додати файл `debian/tests/control`, який визначає вимоги до тестового оточення,
- додати тести в `debian/tests`.

### 2.4.1 Вимоги до тестового оточення

У файлі `debian/tests/control` Ви можете визначити вимоги до тестового оточення. Наприклад, якщо тести не проходять при збірці, або потрібні права `root` – Ви перераховуєте необхідні для тестів пакунки. У Специфікації DEP 8 Ви знайдете усі доступні опції.

Нижче ми розглянемо пакунок джерельного коду `glib2.0`. У дуже простому випадку він буде виглядати так:

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

Це буде означати, що для тесту `debian/tests/build` потрібні пакунки `libglib2.0-dev` і `build-essential`.

---

**Примітка:** У полі `Depends` можна вказати `@`, якщо Ви бажаєте встановлення усіх бінарних пакунків, зібраних з розглядуваного пакунку джерельного коду.

---

### 2.4.2 Поточні тести

Тест, що відповідає розглянутому вище прикладу, буде виглядати 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"
```

Тут невелика програма мовою C копіюється у тимчасову теку, потім компілюється з використанням системних бібліотек (з використанням прапорців та шляхів до бібліотек, визначених через `pkg-config`). Потім запускається скомпільований файл, який запускає декілька основних функцій glib.

Хоч цей тест дуже маленький і простий, він перевіряє багато: що Ваш -dev пакунок має усі необхідні залежності, що Ваш пакунок встановлює робочі файли pkg-config, заголовкові файли і бібліотеки поміщуються у потрібне місце, або що компілювальник та компонувальник працюють. Це допомогає виявити критичні помилки на початковій стадії.

### 2.4.3 Виконання тесту

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 безперервної інтеграції Ubuntu CI використовує емулятор QEMU й запускає тести з пакунків у архіві, з увімкненим прапорцем `-proposed`. Щоб вручну отримати те ж оточення, спочатку необхідно встановити наступні пакунки:

```
sudo apt install autopkgtest qemu-system qemu-utils autodep8
```

Тепер виконайте збірку тестового оточення, виконавши таке:

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

(Детальніше про вибір інших релізів, архітектур, цільових директорій, й про використання проксі – в manpage й у виводі опції `--help`). Ця команда виконає збірку, наприклад `adt-trusty-amd64-cloud.img`.

Тепер запустіть тести джерельного пакунку, наприклад `libpng`, у образі 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 Подальші приклади

Цей перелік не повний, але може допомогти Вам отримати уяву про те, як автоматичні тести реалізовані, й як вони використовуються в Ubuntu.

- Для бібліотеки libxml2 , тести дуже схожі. Вони також запускають тестову збірку простого коду на C й виконують його.
- Тести пакунку gtk+3.0 також компілюють/лінкують/запускають перевірку у тесті “build”. Також є додатковий тест “python3-gi”, який перевіряє що бібліотека GTK може бути використана під час тесту.
- У пакунку ubiquity tests використовується набір тестів батьківського пакунку
- Пакунок gvfs tests – дуже цікавий приклад: він тестує свій функціонал “на повну”, включаючи емулювання CD, Samba, DAV та інших компонентів.

### 2.4.5 Інфраструктура Ubuntu

Пакунки з увімкненим `autopkgtest` будуть тестуватися при вивантаженні, або якщо оновляться якісь залежності. Результат роботи автоматичний запуск тестів autopkgtest може бути переглянутий на сайті, й він регулярно оновлюється.

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 Додавання тесту в Ubuntu

Процес додавання й відправлення autopkgtest-тесту для пакунків дуже схожий на *процес виправлення помилок в Ubuntu*. Вистачить лише:

- виконайте `bzr branch ubuntu:<ім’я_пакунку>`,

- увімкніть тести в `debian/control`,
- створіть директорію `debian/tests`,
- створіть `debian/tests/control`, основуючись на Специфікації DEP 8,
- додайте Ваші тести в `debian/tests`,
- закомте Ваші зміни, відправте їх на Launchpad, запропонуйте merge й дочекайтеся його розгляду, – у точності як з будь-якими іншими покращеннями у джерельних пакунках.

### 2.4.7 Чи Ви можете допомогти

Команда Ubuntu Engineering зібрала перелік необхідних тестів, у якому пакунки, що потребують тестування, розподілені за категоріями. Там можна знайти приклади таких тестів й взяти їх на себе.

Якщо Ви зіштовхнетеся з проблемами, приєднуйтеся до IRC на каналу #ubuntu-quality IRC channel: розробники звичайно Вам допоможуть.

## 2.5 Використання chroot-оточень

Якщо Ви користуєтеся однією з версій Ubuntu, але працюєте над пакунками для іншої версії, Ви можете створити середовище іншої версії за допомогою `chroot`.

Використання `chroot` дозволить Вам мати у розпорядженні повну файлову систему іншого дистрибутиву для зручності роботи. Це дозволяє уникнути затрат, пов'язаних з встановленням віртуальної машини.

### 2.5.1 Створення chroot

Використовуйте команду `debootstrap`, щоб створити новий chroot:

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

Це створить теку `trusty` і встановить мінімальний образ trusty у неї.

Якщо Ваша версія `debootstrap` не визначить Trusty, спробуйте оновитися до версії в `backports`.

Після цього Ви можете працювати всередині chroot:

```
$ sudo chroot trusty
```

Де можна встановити або вилучити будь-який пакунок, який Ви бажаєте, без шкоди для основної системи.

Ви можете скопіювати свої ключі GPG і SSH, а також конфігурацію Bazaar в chroot, щоб отримувати доступ й підписувати пакунки безпосередньо звідти:

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

Щоб apt й інші програми не скаржилися на відсутні локалі, можна встановити відповідний мовний пакунок:

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

Якщо Вам потрібно запускати проґрами, що використовують X-сервер, Вам потрібно додати в chroot директорію /tmp, для цього ззовні chroot запустіть:

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

Для деяких проґрам, можливо, знадобиться прив'язати /dev або /proc.

На сторінці Debootstrap Chroot вікі Ви знайдете детальнішу інформацію про chroot-оточення.

### 2.5.2 Альтернативи

SBuild – система, схожа на PBuilder, що використовується для створення оточення, у якому виконуються тестові збірки пакунку. Вона близька до тієї, яку використовує Launchpad для збірки пакунків, але її встановлення дещо складніше, ніж PBuilder. Повнішу інформацію можна знайти на вікісторінці Система Збірки Security Team.

Повні віртуальні машини можуть бути корисні для створення пакунків й тестування проґрам. TestDrive — це проґрама, яка дозволяє автоматизувати синхронізацію і запуск щоденних ISO-образів. Детальніше дивіться wiki-сторінку TestDrive.

Можна також налаштувати pbuilder так, щоб він призупинявся при виявленні помилки збірки. Скопіюйте C10shell з /usr/share/doc/pbuilder/examples у каталог й використовуйте аргумент `--hookdir=`, щоб вказати на нього.

Хмарний сервіс Amazon EC2 дозволить Вам придбати комп'ютер у хмарі, ціна за який – усього декілька центів на годину. Там Ви можете встановити Ubuntu будь-якої підтримуваної версії і працювати з пакунками віддалено, що дуже зручно, якщо потрібна збірка багатьох пакунків одночасно, або якщо потрібно подолати повільну швидкість Інтернет-під'єднання.

## 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 Робота з пакунками KDE

Створенням пакунків програм KDE в Ubuntu займаються команди Kubuntu і MOTU. Зв'язатися з командою Kubuntu можна через поштову розсилку Kubuntu й канал Freenode IRC `#kubuntu-devel`. Додаткова інформація про розробку Kubuntu доступна на wiki-сторінці Kubuntu.

При створенні пакунків ми враховуємо практику команд Debian Qt/KDE і Debian KDE Extras. Більшість наших пакунків є похідними від пакунків, створених цими командами Debian.

### 2.7.1 Політика створення латок

Kubuntu додає латки у застосунки KDE, лише якщо вони походять від оригінальних авторів, або якщо вони були відправлені в upstream й є впевненість у їх швидкому прийнятті, або якщо ми обміркували проблеми з розробниками KDE.

Kubuntu не змінює фірмового оформлення пакунків, крім випадків, коли цього потребує апстрім (наприклад, логотип у горішньому лівому куту меню Kickoff) або для спрощення (наприклад, вилучення показуваних при запуску заставок).

### 2.7.2 debian/rules

Пакунки Debian включають деякі доповнення до звичайного використання Debhelper. Вони зберігаються у пакунку `pkg-kde-tools`.

Пакунки, що використовують Debhelper 7, повинні додати опцію `--with=kde`. Це дозволить переконатися у використанні правильних прапорців збірки і опцій, таких як обробка завдань kdeinit та файлів перекладу.

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

Деякі нові пакунки KDE використовують систему `dhmk`, альтернативу `dh`, створену командою Debian Qt/KDE. Прочитати про неї можна в /usr/share/pkg-kde-tools/qt-kde-team/2/README. Пакунки що її використовують будуть `включати /usr/share/pkg-kde-tools/qt-kde-team/2/debian-qt-kde.mk` замість запуску `dh`.

### 2.7.3 Переклади

Переклади пакунків з репозиторію main імпортуються на Launchpad і експортуються з Launchpad у мовні пакунки Ubuntu.

Отже, кожен KDE-пакунок в main повинен створювати шаблони перекладів, включати оригінальні переклади і опрацьовувати рядки що перекладаються у `.desktop`-файлах.

Для генерації шаблонів перекладів пакунок повинен містити файл `Messages.sh`; якщо його там немає, зверніться в апстрім. Перевірити його роботу можна, виконавши сценарій `extract-messages.sh`, який повинен створити один або декілька файлів `.pot` у каталозі `po/`. У процесі збирання це буде зроблено автоматично, якщо Ви використовуєте опцію `--with=kde` для `dh`.

Апстрім зазвичай також поміщає файли перекладів `.po` у каталог `po/`. Якщо їх там немає, перевірте, чи не виділені вони у один з окремих мовних пакунків апстріму, наприклад, у мовні пакунки KDE SC. Якщо вони у окремому мовному пакунку, на Launchpad необхідно збирати їх разом вручну. Проконсультуйтеся з цього приводу з Девідом Планеллою (David Planella).

Якщо пакунок переміщений з universe в main, його слід перевивантажити перед імпортом перекладів на Launchpad.

Файли `.desktop` також потребують перекладу. Ми додали латку до KDELibs для читання перекладів з `.po`-файлів, що вказують на рядок `X-Ubuntu-Gettext-Domain=`, що додається до файлів `.desktop` під час збірки пакунку. Файл .pot для кожного пакунку генерується під час збірки й .po-файли необхідно стягнути з апстріму і включити у пакунок або в наші мовні пакунки. Перелік .po-файлів, які потрібно стягнути зі сховищ KDE, знаходиться в `/usr/lib/kubuntu-desktop-i18n/desktop-template-list`.

### 2.7.4 Бібліотекові символи

Бібліотекові символи відстежуються за допомогою файлів `.symbols`, які дозволяють переконатися, що усе на місці. KDE використовує бібліотеки C++, які діють дещо інакше, ніж бібліотеки C. Для Debian команда Qt/KDE створила скрипти, які дозволяють подолати це. Документ Робота з файлами symbols дає опис, як створювати й оновлювати такі файли.

# Матеріяли для подальшого читання

Ви можете прочитати офлайн-версію цього посібника у різних форматах, якщо встановите один з двійкових пакунків.

Якщо Ви бажаєте дізнатися більше про збирання пакунків Debian, ось декілька ресурсів Debian, які можуть бути Вам корисними:

- Як створювати пакунки для Debian;
- Посібник з політики Debian;
- Посібник розробника-початківця Debian — доступний різними мовами;
- Посібник зі створення пакунків (також доступний у вигляді пакунку);
- Посібник зі створення пакунків для модулів Python.

Ми завжди намагаємося поліпшити цей посібник. Якщо Ви знайдете якусь помилку, або бажаєте щось запропонувати, будь ласка, створіть звіт про ваду на Launchpad. Якщо Ви бажали б допомогти у праці над посібником його джерельний код також доступний на Launchpad.