Ubuntu Packaging Guide

Version 1.0.3 bzr695 ubuntu14.04.1

Ubuntu Developers

25 February 2020

Table des matières

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 n'est pas seulement un système d'exploitation libre et open source, sa plate-forme est également ouverte et développée de manière transparente. Le code source de chaque composant s'obtient aisément et chaque modification de la plate-forme Ubuntu peut être examinée.

Cela signifie que vous pouvez activement participer à son amélioration et que la communauté des développeurs de la plate-forme Ubuntu se sent toujours concernée pour aider ses pairs à démarrer.

Ubuntu est également une communauté de personnes formidables qui croient au logiciel libre afin qu'il reste accessible au plus grand nombre. Ses membres sont accueillants et souhaitent également votre participation. Nous souhaitons que vous vous impliquiez, posiez des questions, rendiez Ubuntu meilleur avec nous.

Si vous rencontrez des problèmes : pas de panique ! Vérifiez l'*article de communication* et vous découvrirez comment rentrer plus facilement en contact avec d'autres développeurs.

Le guide est divisé en deux sections :

— Une liste d'articles basés sur les tâches ou les choses que vous souhaiteriez voir traitées.
— Un ensemble d'articles composant une base de connaissances plus approfondies sur des parties spécifiques de nos outils et nos flux de travail.

CHAPITRE 1

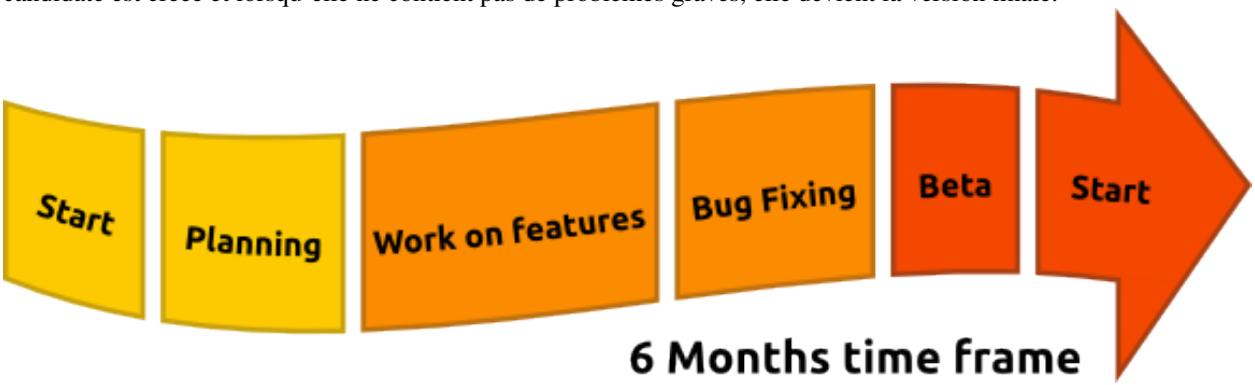
Articles

1.1 Introduction au Développement d'Ubuntu

Ubuntu est constitué de milliers de composants différents, écrits dans de multiples langages de programmation. Chaque composant - pouvant être une bibliothèque logicielle, un outil ou une application graphique - est disponible sous forme d'un paquet source. Dans la plupart des cas, les paquets sources se composent de deux parties : le code source réel et les métadonnées. Ces dernières comprennent les dépendances du paquet, les droits d'auteur et les informations de licence, ainsi que des instructions sur la façon de construire le paquet. Une fois ce paquet source compilé, le processus de construction fournit les paquets binaires, en d'autres termes, les fichiers .deb que les utilisateurs peuvent installer.

Every time a new version of an application is released, or when someone makes a change to the source code that goes into Ubuntu, the source package must be uploaded to Launchpad's build machines to be compiled. The resulting binary packages then are distributed to the archive and its mirrors in different countries. The URLs in `/etc/apt/sources.list` point to an archive or mirror. Every day images are built for a selection of different Ubuntu flavours. They can be used in various circumstances. There are images you can put on a USB key, you can burn them on DVDs, you can use netboot images and there are images suitable for your phone and tablet. Ubuntu Desktop, Ubuntu Server, Kubuntu and others specify a list of required packages that get on the image. These images are then used for installation tests and provide the feedback for further release planning.

Le développement d'Ubuntu est très dépendant de la phase actuelle du cycle de sortie. Nous publions une nouvelle version d'Ubuntu tous les six mois, ce qui est rendu possible par l'établissement dates strictement figées. À chaque échéance atteinte, les développeurs doivent faire moins de modifications, ou les moins intrusives possibles. Le gel des fonctionnalités est la première grosse échéance après la première moitié du cycle. À ce stade, les nouvelles fonctionnalités doivent être largement appliquées. Le reste du cycle est censé se concentrer sur la correction des bogues. Ensuite, l'interface utilisateur, puis la documentation, le noyau, etc. sont gelés, puis la version bêta est mise en ligne pour recevoir un maximum de tests. À partir de la version bêta, seuls les bogues critiques sont corrigés, une version candidate est créée et lorsqu'elle ne contient pas de problèmes graves, elle devient la version finale.



Des milliers de paquets sources, des milliards de lignes de code, des centaines de contributeurs exigent beaucoup de

communication et de planification pour maintenir des normes élevées de qualité. Au début et au milieu de chaque cycle de version, nous avons le Sommet des Développeurs Ubuntu, où les développeurs et les contributeurs sont réunis pour planifier les fonctionnalités des prochaines versions. Chaque fonctionnalité est décrite par ses intervenants et un cahier des charges est écrit, contenant des informations détaillées sur ses hypothèses, sa mise en œuvre, les modifications nécessaires à d'autres endroits, la façon de la tester, etc. Tout cela est fait de manière ouverte et transparente, afin que vous puissiez participer à distance et regarder un flux vidéo, discuter avec les participants et adhérer aux modifications de spécifications, de sorte que vous soyez toujours informé.

Chaque modification ne peut malgré tout pas être abordée lors d'une réunion, en particulier parce qu'Ubuntu repose sur des modifications effectuées dans d'autres projets. C'est pourquoi les contributeurs à Ubuntu restent constamment en contact. La plupart des équipes ou des projets utilisent des listes de diffusion dédiées en vue d'éviter trop de perturbations inutiles. Pour la coordination plus immédiate, les développeurs et les contributeurs utilisent Internet Relay Chat (IRC). Toutes les discussions y sont ouvertes et publiques.

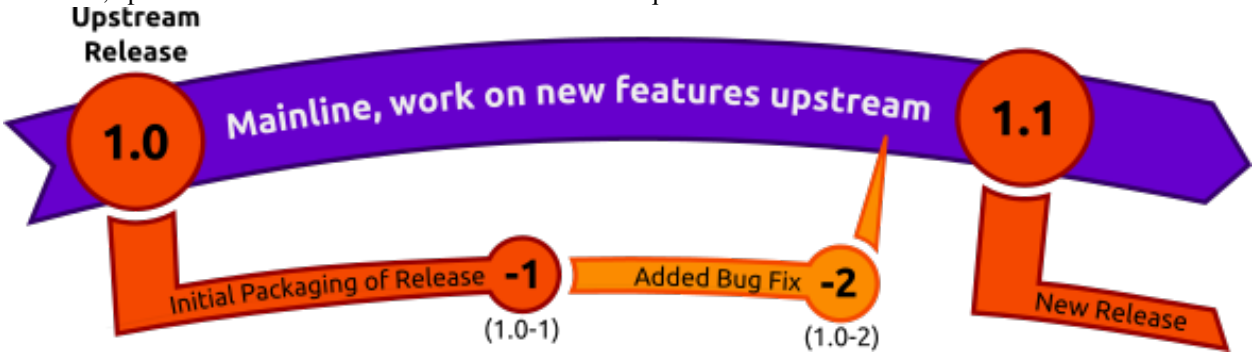
Un autre outil important en matière de communication est le rapport de bogue. Chaque fois qu'un défaut est décelé dans un paquet ou une partie de l'infrastructure, un rapport de bogue est déposé dans Launchpad. Toutes les informations sont recueillies dans ce rapport et l'importance, le statut et l'affectation du bogue sont mis à jour si nécessaire. Cela en fait un outil efficace pour surmonter les bogues dans un paquet ou un projet et pour organiser la charge de travail.

La plupart des logiciels disponibles dans Ubuntu ne sont pas écrits par les développeurs d'Ubuntu eux-mêmes. Une grande partie est écrite par les développeurs d'autres projets Open Source et ensuite intégrés dans Ubuntu. Ces projets sont appelés "amonts", parce que leurs codes sources se déversent dans Ubuntu, où nous faisons "juste" leur intégration. La relation avec les projets en amonts est extrêmement importante pour Ubuntu. Ce n'est pas uniquement le code qu'Ubuntu reçoit des projets en amont, ceux-ci profitent également des utilisateurs, des rapports de bogues et des correctifs de la part d'Ubuntu (et des autres distributions).

L'amont le plus important pour Ubuntu est Debian. Debian est la distribution sur laquelle Ubuntu est basée et la plupart des décisions de conception relatives à l'infrastructure d'empaquetage sont prises là. Traditionnellement, Debian a toujours eu des responsables dédiés pour chaque paquet ou des équipes de maintenance dédiées. Dans Ubuntu, ce sont également des équipes qui s'intéressent à un sous-ensemble de paquets et, naturellement, chaque développeur possède un domaine d'expertise, mais la participation (et les droits de téléchargement) est généralement ouverte à tous ceux qui prouvent leur talent et leur volonté.

Obtenir une modification dans Ubuntu en tant que nouveau collaborateur n'est pas aussi intimidant qu'il n'y paraît et peut même s'avérer une expérience très enrichissante. Il ne s'agit pas seulement d'apprendre quelque chose de nouveau et d'excitant, mais également de partager une solution et de résoudre un problème pour des millions d'utilisateurs.

Le développement Open Source intervient dans un monde partagé selon différents objectifs et différents domaines d'intérêt. Par exemple, cela pourrait être le cas d'un Amont particulier intéressé par le travail sur une nouvelle fonctionnalité importante, alors qu'Ubuntu, en raison de son échéancier serré de sortie, est intéressé par la fourniture d'une version robuste avec juste quelques corrections supplémentaires de bogues. C'est pourquoi nous utilisons le "Développement Distribué", où le code est en cours d'élaboration dans différentes branches qui sont fusionnées les unes avec les autres, après des révisions du code et des discussions adéquates.



Dans l'exemple mentionné ci-dessus, il serait logique de fournir Ubuntu avec la version existante du projet, d'ajouter le correctif, de l'inclure à l'amont pour leur prochaine version et de fournir de nouveau le projet (s'il est convenable)

avec la prochaine version d'Ubuntu. Ce serait le meilleur compromis possible et une situation gagnant-gagnant.

Pour corriger un bogue dans Ubuntu, vous devrez d'abord obtenir le code source pour le paquet, puis travailler sur le correctif, le documenter pour qu'il soit simple à comprendre pour les autres développeurs et utilisateurs, puis construire le paquet pour le tester. Après l'avoir testé, vous pouvez aisément proposer que la modification soit incluse dans la version d'Ubuntu actuellement en développement. Un développeur ayant les droits de téléchargement l'examinera pour vous puis l'intégrera dans Ubuntu.

En cherchant une solution, il est généralement intelligent de vérifier avec l'amont et de voir si le problème (ou une solution possible) est connu et, dans le cas contraire, de faire de votre mieux pour que la solution soit un effort concerté.

Des étapes supplémentaires pourraient induire le rétroportage des modifications vers une ancienne version d'Ubuntu encore prise en charge et leur transmission vers l'amont.

Les pré-requis les plus importants pour réussir dans le développement d'Ubuntu sont les suivants : avoir un talent pour "faire que les choses fonctionnent de nouveau", ne pas avoir peur de lire la documentation et de poser des questions, jouer collectif et apprécier le travail de détective.

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 Mise en route

Un certain nombre de choses doivent être accomplies pour commencer à développer pour Ubuntu. Cet article est conçu pour configurer votre ordinateur de sorte que vous puissiez commencer à travailler avec des paquets, puis télécharger vos paquets vers la plate-forme d'hébergement d'Ubuntu, Launchpad. Voici ce que nous allons aborder :

- Installation des logiciels relatifs à l'empaquetage. Cela comprend :
  - Les utilitaires d'empaquetage spécifiques à Ubuntu
  - Logiciel de chiffrement afin que votre travail soit reconnu comme étant bien réalisé par vous
  - Logiciel de chiffrement supplémentaire afin de transférer des fichiers en toute sécurité
- Création et configuration de votre compte sur Launchpad
- Configuration de votre environnement de développement pour vous aider à construire des paquets localement, à interagir avec d'autres développeurs, et à proposer vos modifications sur Launchpad.

Note : Il est conseillé de travailler sur l'empaquetage en utilisant la version de développement actuelle d'Ubuntu. Procéder ainsi vous permettra de tester les modifications dans le même environnement où elles seront effectivement appliquées et utilisées.

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

1.2.1 Installer les logiciels d'empaquetage de base

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
```

Cette commande va installer les logiciels suivants :

— `gnupg` – GNU Privacy Guard contains tools you will need to create a cryptographic key with which you will sign files you want to upload to Launchpad.
— `pbuilder` – un outil pour réaliser des constructions reproductibles d'un paquet dans un environnement propre et isolé.
— `ubuntu-dev-tools` (et `devscripts`, sa dépendance directe) – une collection d'outils simplifiant les nombreuses tâches d'empaquetage.
— `apt-file` donne un moyen facile de trouver le paquet binaire contenant un fichier donné.

Créez votre clé GPG

GPG stands for GNU Privacy Guard and it implements the OpenPGP standard which allows you to sign and encrypt messages and files. This is useful for a number of purposes. In our case it is important that you can sign files with your key so they can be identified as something that you worked on. If you upload a source package to Launchpad, it will only accept the package if it can absolutely determine who uploaded the package.

Pour générer une nouvelle clé GPG, exécutez :

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

GPG vous demandera d'abord le type de clé que vous souhaitez générer. Le choix par défaut (RSA et DSA) convient. Ensuite, il vous demandera la taille de clé. La valeur par défaut (actuellement 2048) est correcte, mais 4096 est plus sécurisé. Par la suite, il vous sera demandé si vous souhaitez faire expirer la clé à un certain stade. Vous pouvez sans danger indiquer "0", ce qui signifie que la clé n'expirera jamais. Les dernières questions seront à propos de votre nom et de votre adresse de courriel. Il suffit ici de choisir ceux que vous allez utiliser pour le développement d'Ubuntu, vous pourrez ajouter des adresses de courriel supplémentaires plus tard. Ajouter un commentaire n'est pas nécessaire. Ensuite, vous devrez définir une phrase secrète, choisissez-en une sûre (une phrase secrète est juste un mot de passe qui autorise les espaces).

GPG va maintenant vous créer une clé, ce qui peut prendre un peu de temps ; comme GPG a besoin d'octets aléatoires, il serait souhaitable que vous donniez quelques tâches à réaliser au système. Déplacez le curseur, tapez quelques paragraphes de texte aléatoire, chargez quelques pages dans votre navigateur internet, etc.

Une fois cela fait, vous obtiendrez un message similaire à celui-ci :

```
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
```

Dans ce cas `43CDE61D` est l'*identifiant clé*.

Ensuite, vous devez envoyer la partie publique de votre clé à un serveur de sorte que tout le monde puisse identifier vos messages et fichiers. Pour ce faire, entrez :

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

Cela enverra votre clé au serveur de trousseau d'Ubuntu, inclus dans un réseau d'autres serveurs de trousseaux qui se synchroniseront entre eux. Une fois cette synchronisation terminée, votre clé publique signée sera prête à vérifier vos contributions partout dans le monde.

Créez votre clé SSH

SSH signifie *Secure Shell* , le protocole permettant d'échanger des données de façon sécurisée sur un réseau. Il est courant d'utiliser SSH pour accéder à un environnement, l'ouvrir sur un autre ordinateur et l'utiliser pour transférer des fichiers en toute sécurité. Dans notre cas, nous allons principalement utiliser SSH pour télécharger en toute sécurité nos paquets source vers Launchpad.

Pour générer une clé SSH, saisissez :

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

Le nom de fichier par défaut est généralement logique, vous pouvez donc simplement le laisser tel quel. Pour des raisons de sécurité, il est fortement recommandé d'utiliser une phrase secrète.

Configurer pbuilder

`pbuilder` vous permet de construire des paquets en local sur votre machine. Il dispose de différentes fonctions :

- La construction se fera dans un environnement minimal et propre. Cela vous aide à vous assurer que vos constructions réussiront de façon reproductible, mais sans modifier votre système local
- Il est inutile d'installer toutes les *dépendances de construction* nécessaires en local
- Vous pouvez configurer plusieurs instances pour différentes versions d'Ubuntu et de Debian

Configurer `pbuilder` est très simple, lancez :

```
$ 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 Mettez vous en place pour travailler avec Launchpad

Avec l'installation d'une configuration de base en local, la prochaine étape sera de configurer votre système pour travailler avec Launchpad. Cette section portera sur les sujets suivants :

- Description de Launchpad et création d'un compte Launchpad
- Téléversement de vos clés GPG et SSH vers Launchpad
- Configure your shell to recognize you (for putting your name in changelogs)

À propos de Launchpad

Launchpad est la pièce maîtresse de l'infrastructure que nous utilisons dans Ubuntu. Il stocke non seulement nos paquets et notre code, mais également d'autres choses comme les traductions, les rapports de bogues et les informations sur les personnes qui travaillent sur Ubuntu et leurs équipiers. Vous pourrez également utiliser Launchpad pour publier vos propositions de corrections et obtenir leur examen et leur parrainage par d'autres développeurs d'Ubuntu.

Vous devez vous inscrire à Launchpad et fournir un minimum d'informations. Cela vous permettra de télécharger du code depuis et vers Launchpad, de soumettre des rapports de bogues et plus encore.

Outre l'hébergement d'Ubuntu, Launchpad peut accueillir tout projet de logiciel libre. Pour plus d'informations, voir le wiki Aide de Launchpad.

Créer un compte Launchpad

If you don't already have a Launchpad account, you can easily create one. If you have a Launchpad account but cannot remember your Launchpad id, you can find this out by going to https://launchpad.net/~ and looking for the part after the ~ in the URL.

Le processus d'inscription à Launchpad vous demandera de choisir un nom à afficher. Il vous est recommandé d'utiliser votre vrai nom, afin que vos collègues développeurs sur Ubuntu soient en mesure de mieux vous connaître.

Lorsque vous enregistrez un nouveau compte, Launchpad envoie un courriel avec un lien que vous devez ouvrir dans votre navigateur afin de faire vérifier votre adresse de courriel. Si vous ne le recevez pas, vérifiez dans votre dossier de pourriel.

The new account help page on Launchpad has more information about the process and additional settings you can change.

Téléversez votre clé GPG sur Launchpad

First, you will need to get your fingerprint and key ID.

Pour connaître votre empreinte GPG, exécutez :

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

et cela affichera quelque chose comme :

```
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
```

Then run this command to submit your key to Ubuntu keyserver :

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

where `43CDE61D` should be replaced by your key ID (which is in the first line of output of the previous command). Now you can import your key to Launchpad.

Dirigez-vous vers https://launchpad.net/~/+editpgpkeys et copiez le fichier "Key fingerprint" dans la zone de texte. Dans le cas ci-dessus ce serait `5C28 0144 FB08 91C0 2CF3 37AC 6F0B F90F 43CD E61D`. Ensuite, cliquez sur "Importer la clé".

Launchpad utilise l'empreinte pour vérifier votre clé sur le serveur de trousseau d'Ubuntu et, en cas de succès, vous envoie un courriel crypté vous demandant de confirmer l'import de la clé. Vérifiez votre compte de messagerie et lisez le courriel envoyé par Launchpad. *Si votre client de messagerie prend en charge le chiffrement OpenPGP, il vous demandera d'entrer le mot de passe choisi pour la clé lors de la génération GPG. Entrez le mot de passe, puis cliquez sur le lien pour confirmer que la clé est la vôtre.*

Launchpad encrypts the email, using your public key, so that it can be sure that the key is yours. If you are using Thunderbird, the default Ubuntu email client, you can install the Enigmail plugin to easily decrypt the message. If your email software does not support OpenPGP encryption, copy the encrypted email's contents, type `gpg` in your terminal, then paste the email contents into your terminal window.

De retour sur le site Launchpad, utilisez le bouton "Confirmer" et Launchpad terminera l'import de votre clé OpenPGP.

Vous trouverez plus d'informations sur https://help.launchpad.net/YourAccount/ImportingYourPGPKey

Téléchargez votre clé SSH vers Launchpad

Ouvrez https://launchpad.net/~/+editsshkeys dans un navigateur internet, ainsi que `~/.ssh/id_rsa.pub` dans un éditeur de texte. Il s'agit de la partie publique de votre clé SSH, il est donc sûr de la partager avec Launchpad. Copiez le contenu du fichier et collez-le dans la zone de texte sur la page Web affichant "Ajouter une clé SSH". Ensuite, cliquez sur "Importer la clé publique".

For more information on this process, visit the creating an SSH keypair page on Launchpad.

Configurez votre shell

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"
```

Maintenant, sauvegardez le fichier et redémarrez votre terminal ou exécutez :

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

(Si vous n'utilisez pas l'environnement par défaut, *bash*, veuillez modifier en conséquence le fichier de configuration pour cet environnement.)

1.3 Correction d'un bogue dans Ubuntu

1.3.1 Introduction

Si vous avez suivi les instructions pour *obtenir la configuration de Ubuntu Développement*, vous devriez être configuré et prêt à commencer.

Comme vous pouvez le constater dans l'image ci-dessus, il n'y a pas de surprises dans le processus de correction des bogues dans Ubuntu : vous découvrez un problème, vous obtenez le code, vous travaillez sur le correctif, le testez, soumettez vos modifications dans Launchpad et demandez à ce qu'elles soient examinées puis fusionnées. Dans ce guide, nous allons passer toutes les étapes nécessaires une par une.

1.3.2 Trouver le problème

Il existe de nombreuses façons de trouver des choses sur lesquelles travailler. Ce sera peut-être un rapport de bogue que vous-même pouvez rencontrer (ce qui vous donne une bonne occasion de tester le correctif), ou un problème relevé par ailleurs, éventuellement dans un rapport de bogue.

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 Déterminer ce qu'il faut corriger

Si vous ne connaissez pas le paquet source contenant le code problématique, mais que vous connaissez le chemin du programme concerné sur votre système, vous pouvez découvrir sur quel paquet source vous pourrez travailler.

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
```

Cela affichera :

```
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
```

`apt-cache` fait partie de l'installation standard d'Ubuntu.

1.3.4 Confirmer le problème

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

Supposons que le paquet `bumprace` n'a pas de page d'accueil dans sa description de paquet. Dans un premier temps, vous vérifiez si le problème n'est pas déjà résolu. C'est facile à vérifier, soit en jetant un coup d'œil à la Logithèque Ubuntu, soit en exécutant :

```
apt-cache show bumprace
```

La sortie devrait être similaire à ceci :

```
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
```


Un contre-exemple serait `gedit`, qui dispose d'une page d'accueil :

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

Parfois, vous constaterez qu'un problème particulier sur lequel vous vous penchez est déjà corrigé. Pour éviter le gaspillage des efforts et le travail en doublon, il est logique de commencer par un petit travail de détective.

1.3.5 Localisation de bogue

En premier lieu, nous devons vérifier l'existence préalable d'un bogue pour ce problème dans Ubuntu. Peut-être que quelqu'un travaille déjà sur un correctif, ou nous pouvons contribuer à la solution d'une manière ou d'une autre. Pour Ubuntu, nous jetons un coup d'œil à https://bugs.launchpad.net/ubuntu/+source/bumprace et constatons qu'aucun bogue n'y est ouvert avec notre problème.

Note : Pour Ubuntu, l'URL `https://bugs.launchpad.net/ubuntu/+source/<paquet>` devrait toujours afficher la page concernant les bogues du paquet source en question.

Pour Debian, qui est la principale source des paquets d'Ubuntu, nous jetons un œil sur http://bugs.debian.org/src:bumprace et nous ne pouvons de nouveau pas trouver de rapport de bogue pour notre problème.

Note : Pour Debian, l'URL `http://bugs.debian.org/src:<paquet>` devrait toujours afficher la page concernant les bogues du paquet source en question.

Le problème sur lequel nous travaillons est spécifique, car il concerne uniquement les bits d'empaquetage liés à `bumprace`. Si le problème se trouvait dans le code source, il serait utile de vérifier également le traceur de bogues en amont. Malheureusement, il arrive souvent que cela soit différent à chaque paquet examiné ; mais si vous effectuez une recherche sur internet, vous devriez le trouver facilement dans la plupart des cas.

1.3.6 Offrir de l'aide

Si vous avez trouvé un bogue ouvert, qu'il n'est affecté à personne et que vous êtes en mesure de régler le problème, vous pouvez le commenter avec votre solution. N'oubliez pas d'inclure autant d'informations que possible : dans quelles circonstances se produit le bogue ? Comment avez-vous résolu le problème ? Avez-vous testé votre solution ?

Si aucun rapport de bogue n'a été déposé, vous pouvez en déposer un. Une chose que vous devriez garder à l'esprit est : Le problème est-il si petit qu'une simple demande de réalisation serait suffisante ? Avez-vous réussi à partiellement résoudre le problème et vous voulez au moins partager votre partie de la solution ?

C'est très bien si vous pouvez offrir votre aide, et elle sera assurément appréciée.

1.3.7 Obtenir le code

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 Travailler sur un correctif

Il existe des livres entiers sur la façon de trouver des bogues, de les corriger, de les tester, etc. Si vous êtes complètement novice en programmation, essayez en premier lieu de corriger les bogues simples comme les fautes de frappe évidentes. Essayez de minimiser les changements autant que possible et documentez clairement vos modifications et hypothèses.

Avant de travailler sur un correctif vous-même, assurez-vous de vérifier que personne d'autre ne l'a déjà corrigé ou est en train de le faire. Les bonnes sources à vérifier sont :

— Traceur de bogues en amont (et dans Debian) (bogues ouverts et fermés),
— Historique des révisions en amont (ou version plus récente) pourrait avoir résolu le problème,
— des bogues ou des ajouts de paquets de distributions Debian ou autres.

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
```

Cela va copier l'empaquetage dans un répertoire temporaire. Vous pouvez maintenant éditer les fichiers avec un éditeur de texte ou appliquer des correctifs en amont, par exemple :

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

Après avoir modifié le fichier, saisissez `exit` ou appuyez sur `ctrl-d` pour quitter l'environnement temporaire. Le nouveau correctif a été ajouté dans `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.

Documenter le correctif

Il est très important de documenter abondamment vos modifications afin que les développeurs qui reliront votre code dans le futur n'aient pas à deviner ce qu'étaient votre raisonnement et vos hypothèses. Chaque paquet source Debian et Ubuntu inclut `debian/changelog`, où les modifications de chaque paquet téléchargé sont suivies.

Le moyen le plus simple de mettre à jour est d'exécuter :

```
$ dch -i
```

Cela vous ajoute une entrée passe-partout du changelog et lance un éditeur dans lequel vous pourrez remplir les blancs. Un exemple de ceci pourrait être :


```
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
```

`dch` doit déjà remplir pour vous la première et la dernière ligne d'une telle entrée du changelog. La ligne 1 se compose du nom du paquet source, du numéro de version, de la version d'Ubuntu vers laquelle le paquet est transféré, du niveau d'urgence (qui est presque toujours «{nbsp]faible »). La dernière ligne contient toujours le nom, l'adresse électronique et l'horodatage du changement (au format **RFC 5322**).

Ces préoccupations en moins, concentrons-nous sur l'entrée effective du changelog elle-même : il est très important de renseigner :

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

Dans notre (très rare) exemple, le dernier point est couvert par `(LP: #123456)` se référant au bogue Launchpad 123456. Les rapports de bogues ou des fils de listes de diffusion ou les spécifications sont généralement de bonnes informations à fournir comme justification d'un changement. En prime, si vous utilisez la notation `LP: #<numéro>` pour les bogues sur Launchpad, le bogue sera automatiquement fermé quand le paquet correctif sera téléchargé vers 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 Tester le correctif

Pour construire un paquet de test avec vos modifications, exécutez ces commandes :

```
$ 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.

Note : 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.

Note : Dans de nombreux cas vous devrez réellement installer le paquet pour vous assurer qu'il fonctionne comme prévu. Notre cas est largement plus simple. Si la construction réussit, vous trouverez les paquets binaires dans `~/pbuilder/<release>_result`. Installez-les à l'aide de la commande `sudo dpkg -i <package>.deb` ou en double-cliquant dessus dans votre gestionnaire de fichiers.

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
```

Cela vous mènera à travers une série de mesures pour vous assurer que le bogue se retrouve au bon endroit. Assurez-vous de contrôler de nouveau la différence pour être certain qu'elle ne comprenne aucune modification antérieure non désirée.

La communication est importante, lorsque vous ajoutez une description à votre demande d'inclusion, soyez gentil de bien l'expliquer.

Si tout s'est bien déroulé, vous obtenez un message du système de suivi des bogues Debian avec plus d'informations. Cela peut parfois prendre quelques minutes.

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 Considérations supplémentaires

Si vous trouvez qu'il y a un certain nombre de choses triviales possibles à corriger en même temps dans un paquet, faites-le. Cela permettra d'accélérer l'examen et l'inclusion.

S'il y a plusieurs choses importantes que vous souhaitez corriger, il pourrait être plus judicieux d'envoyer des correctifs individuels ou de fusionner des propositions. S'il y a déjà des bogues individuels déposés sur le sujet, cela rend la chose encore plus aisée.

1.4 L'empaquetage de nouveaux logiciels

Bien qu'il existe des milliers de paquets dans l'archive d'Ubuntu, il y en reste encore beaucoup que personne n'a pu obtenir jusqu'à maintenant. S'il existe une nouvelle et passionnante partie de logiciel pour laquelle vous sentez le besoin d'une exposition plus large, peut-être voudriez-vous vous essayer à la création d'un paquet pour Ubuntu ou d'un PPA. Ce guide vous mènera à travers les étapes d'empaquetage de nouveaux logiciels.

Vous aurez envie en premier lieu de lire l'article *Mise en route* afin de préparer votre environnement de développement.

1.4.1 Vérification du programme

La première étape de l'empaquetage est d'obtenir le fichier .tar issu de l'amont (nous appelons les auteurs d'applications « l'amont ») et de vérifier qu'il se compile et s'exécute.

Ce guide vous mènera à travers l'empaquetage d'une application simple, appelée GNU Bonjour, postée sur 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
```

Cette application utilise le système de construction autoconf, nous exécuterons donc `./configure` pour préparer la compilation.

Cela vérifiera les dépendances de construction nécessaires. Comme `bonjour` est un exemple simple, `build-essential` doit fournir tout ce dont nous avons besoin. Pour les programmes plus complexes, la commande échouera si vous n'avez pas les bibliothèques et les fichiers de développement nécessaires. Installez les paquets nécessaires et recommencez jusqu'à ce que la commande s'exécute avec succès. :


```
$ ./configure
```

Maintenant vous pouvez compiler la source :

```
$ make
```

Si la compilation se termine avec succès, vous pouvez installer et exécuter le programme :

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

1.4.2 Commencer un paquet

`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
```

Lorsqu'il vous est demandé le type de paquet, entrez `s` pour binaire unique. Ceci importera le code dans une branche et ajoutera le répertoire d'empaquetage `debian/`. Jetez un œil sur son contenu. La plupart des fichiers ajoutés ne sont nécessaires que pour les paquets spécialisés (tels que les modules d'Emacs) de sorte que vous pouvez commencer par supprimer les fichiers optionnels d'exemple :

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

Vous devriez maintenant personnaliser chacun des fichiers.

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`.

Le gros du travail de construction de paquet est réalisé par une série de scripts appelée `debhelper`. Le comportement exact de `debhelper` change avec les nouvelles versions majeures, le fichier compat indique à `debhelper` à quelle version se conformer. Vous souhaiterez généralement régler ce paramètre à la version la plus récente qui est `9`.

`control` contient toutes les métadonnées du paquet. Le premier paragraphe décrit le paquet source. Les paragraphes suivants décrivent les paquets binaires à construire. Nous aurons besoin d'ajouter à `Build-Depends:` les paquets nécessaires pour compiler l'application. Pour `bonjour`, assurez-vous qu'il comprend au moins :

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

Vous devrez également remplir une description du programme dans le `champ Description:`.

Le `copyright` doit être rempli pour suivre la licence de la source en amont. Selon le fichier bonjour/COPYING, il s'agit de la licence GNU GPL 3 ou ultérieure.

`docs` contient les fichiers de documentation de l'amont qui, selon vous, devraient être inclus dans le paquet final.

`README.source` et `README.Debian` ne sont nécessaires que si votre paquet possède une caractéristique non standard, ce qui n'est pas le cas donc vous pouvez les supprimer.

`source/format` peut être laissé tel quel, il décrit le format de version du paquet source et devrait être `3.0 (quilt)`.

`rules` est le fichier le plus complexe. Il s'agit d'un Makefile qui compile le code et le transforme en un paquet binaire. Heureusement, le gros du travail se fait de nos jours automatiquement à l'aide de `debhelper 7` de telle sorte que la cible universelle `%` du Makefile lance uniquement le script `dh` qui exécute toutes les opérations nécessaires.

Tous ces fichiers sont expliqués plus en détail dans l'article *aperçu du répertoire Debian*.

Enfin, soumettez le code à votre branche d'empaquetage :

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

1.4.3 Construisez le paquet

Maintenant, nous devons vérifier que notre empaquetage compile le paquet correctement et construit le paquet binaire .deb :

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

`bzr builddeb` est une commande pour construire le paquet dans son emplacement actuel. Le `-us -uc` indique que GPG n'a pas besoin de signer le paquet. Le résultat sera placé dans le dossier `...`

Vous pouvez afficher le contenu du paquet avec :

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

Install the package and check it works (later you will be able to uninstall it using `sudo apt-get remove hello` if you want) :

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

You can also install all packages at once using :

```
$ sudo debi
```

1.4.4 Étapes suivantes

Even if it builds the .deb binary package, your packaging may have bugs. Many errors can be automatically detected by our tool `lintian` which can be run on the source .dsc metadata file, .deb binary packages or .changes file :

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

To see verbose description of the problems use `--info` lintian flag or `lintian-info` command.

For Python packages, there is also a `lintian4python` tool that provides some additional lintian checks.

Après avoir établi un correctif pour l'empaquetage, vous pouvez le reconstruire en utilisant `-nc` pour « no clean » afin d'éviter d'avoir à le reconstruire à partir de zéro :

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


Après avoir vérifié que le paquet est construit localement, vous devez vous assurer qu'il peut se compiler sur un système propre à l'aide de `pbuilder`. Puisque nous allons bientôt l'ajouter à un PPA (Personal Package Archives), ce téléchargement doit être *signé* pour permettre à Launchpad de vérifier que le téléchargement émane de vous (vous pouvez dire que le téléchargement sera signé car les options `-us` et `-uc` ne sont pas transmises à `bzr builddeb` comme auparavant). Pour que la signature fonctionne, vous devez avoir configuré GPG. Si vous n'avez pas encore configuré `pbuilder-dist` ou GPG, *faites le maintenant* :

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

Lorsque vous serez satisfait de votre paquet, vous souhaiterez en obtenir la relecture par d'autres. Vous pouvez télécharger la branche vers Launchpad pour la relecture :

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

Le télécharger vers un PPA assurera qu'il se construit et donnera un moyen aisé pour vous et les autres de tester les paquets binaires. Vous avez besoin de configurer un PPA dans Launchpad puis de télécharger avec `dput` :

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

You can ask for reviews in `#ubuntu-motu` IRC channel, or on the MOTU mailing list. There might also be a more specific team you could ask such as the GNU team for more specific questions.

1.4.5 Soumettez pour inclusion

Il existe nombre de chemins que peut emprunter un paquet pour entrer dans Ubuntu. Dans la plupart des cas, passer par Debian en premier peut s'avérer la meilleure voie. Cette façon vous assure que votre paquet atteindra le plus grand nombre d'utilisateurs, car il sera disponible non seulement dans Debian et Ubuntu, mais également dans l'ensemble de leurs dérivés. Voici quelques liens utiles pour soumettre de nouveaux paquets à Debian :

- Debian Mentors FAQ - debian-mentors is for the mentoring of new and prospective Debian Developers. It is where you can find a sponsor to upload your package to the archive.
- Work-Needing and Prospective Packages - Information on how to file "Intent to Package" and "Request for Package" bugs as well as list of open ITPs and RFPs.
- Debian Developer's Reference, 5.1. New packages - The entire document is invaluable for both Ubuntu and Debian packagers. This section documents processes for submitting new packages.

In some cases, it might make sense to go directly into Ubuntu first. For instance, Debian might be in a freeze making it unlikely that your package will make it into Ubuntu in time for the next release. This process is documented on the "New Packages" section of the Ubuntu wiki.

1.4.6 Captures d'écran

Une fois que vous avez téléversé un paquet vers debian, vous devez ajouter des captures d'écran pour permettre aux utilisateurs potentiels de voir à quoi le programme ressemble. Elles doivent être téléversées sur http://screenshots.debian.net/upload.

1.5 Mises à jour de sécurité et de version stable

1.5.1 Correction d'un bogue de sécurité dans Ubuntu

Introduction

La résolution de bogues de sécurité dans Ubuntu n'est pas vraiment différente de :doc :' la résolution d'un bogue normal dans Ubuntu<./fixing-a-bug>', et nous présumons que vous êtes familier avec la correction de bogues normaux. Pour montrer où les choses diffèrent, nous mettrons à jour le paquet dbus dans Ubuntu 12.04 LTS (Precise Pangolin) pour une mise à jour de sécurité.

L'obtention de la source

Dans cet exemple, nous savons déjà que nous souhaitons solutionner le paquet dbus dans Ubuntu 12.04 LTS (Precise Pangolin). Ainsi en premier lieu, vous devez déterminer la version du paquet que vous souhaiter télécharger. Nous pouvons utiliser le `rmadison` pour y parvenir :

```
$ 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
```

Typiquement, vous souhaiterez choisir la plus récente des versions pour la corriger qui n'est ni dans -proposed ni dans -backports. Puisque nous sommes en train de mettre à jour le dbus de Precise, vous téléchargerez 1.4.18-1ubuntu1.4 depuis precise-updates :

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

Correction de la source

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.

Pour créer votre correctif en utilisant `edit-patch` :

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

Cela appliquera les correctifs existants, et placera l'empaquetage dans un répertoire temporaire. Ensuite, éditez les fichiers nécessaires pour corriger la vulnérabilité. Souvent, l'amont a fourni un correctif afin que vous puissiez l'appliquer :

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

Après avoir effectué les modifications nécessaires, vous appuyez simplement sur Ctrl-D ou tapez exit pour quitter l'environnement temporaire.

Formatage du changelog et des correctifs

Après avoir appliqué vos correctifs vous devrez mettre à jour le changelog. La commande `dch` est utilisée pour modifier le fichier `debian/changelog` et `edit-patch` lancera automatiquement `dch` après ne pas avoir appliqué tous les correctifs. Si vous n'utilisez pas `edit-patch`, vous pouvez lancer manuellement `dch -i`. Contrairement aux correctifs normaux, vous devrez utiliser le format suivant (notez que le nom de distribution utilise precise-security puisqu'il s'agit d'un mise à jour de sécurité pour Precise) pour les mises à jour de sécurité :

```
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]
...
```

Mettez à jour votre correctif pour utiliser les balises de correctifs appropriées. Votre correctif devrait avoir au minimum les balises Origine, Description et Bug-Ubuntu. Par exemple, éditer debian/patches/99-fix-a-vulnerability.patch pour obtenir quelque chose comme :

```
## 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
...
```

De multiples vulnérabilités peuvent être corrigées dans le même téléchargement de sécurité ; il suffit de s'assurer d'utiliser des correctifs différents pour les différentes vulnérabilités.

Testez et soumettez votre travail

À ce stade, le processus est le même que pour *corriger un bogue normal dans Ubuntu*. Plus précisément, vous devrez :

1. Créer votre paquet et vérifier qu'il se compile sans erreur et sans aucun avertissement additionnel du compilateur
2. Mettre à niveau vers la nouvelle version du paquet à partir de la version précédente
3. Vérifier que le nouveau paquet corrige la vulnérabilité et n'introduit pas de régression
4. Soumettre votre travail par le biais d'une proposition de fusion sur Launchpad et déposer un rapport de bogue Launchpad en vous assurant tout d'abord de marquer le bogue comme étant un bogue de sécurité et ensuite de vous inscrire à `ubuntu-security-sponsors`

Si le problème de sécurité n'est pas encore public alors ne déposez pas une proposition de fusion mais assurez-vous de marquer le bogue comme privé.

Le bogue déposé doit inclure un cas de test, c'est à dire un commentaire qui explique clairement comment recréer le bogue en exécutant l'ancienne version, puis comment s'assurer que le bogue n'existe plus dans la nouvelle version.

Le rapport de bogue doit également confirmer que le problème est corrigé dans les versions Ubuntu ultérieures à celle proposée à la correction (dans l'exemple ci-dessus, plus récentes que Précise). Si le problème n'est pas corrigé dans les nouvelles versions, vous devrez préparer les mises à jour pour ces versions également.

1.5.2 Mises à jour vers Version Stable

Nous permettons aussi les mises à jour de versions où un paquet bogué a un lourd impact, comme une régression sévère depuis une version antérieure, ou un bogue qui cause des pertes de données. En raison de la capacité de telles mises à jour à introduire elles-mêmes des bogues, nous ne les autorisons que lorsque les modifications peuvent être facilement comprises et vérifiées.

Le processus des Mises à jour de version stable est exactement le même que celui des bogues de sécurité, hormis que vous devez vous inscrire à `ubuntu-sru` pour le bogue.

La mise à jour ira dans l'archive `proposed` (par exemple `precise-proposed`) où il sera vérifié qu'elle corrige le problème et n'introduit pas de nouveaux problèmes. Après une semaine sans problème rapporté, elle peut être déplacée vers `updates`.

See the Stable Release Updates wiki page for more information.

1.6 Correctifs aux paquets

Parfois, les mainteneurs de paquets d'Ubuntu doivent modifier le code source en amont afin de le faire fonctionner correctement avec Ubuntu. Les exemples incluent les correctifs vers l'amont qui n'ont pas encore été repris dans une version finale, ou les modifications du système de compilation de l'amont uniquement nécessaires pour la compilation avec Ubuntu. Nous pourrions modifier le code source en amont directement (mais en faisant cela il devient plus difficile d'enlever les correctifs plus tard, lorsque l'amont les a intégré) ou extraire les modifications pour les soumettre au projet de l'amont. Au lieu de ça, nous gardons ces modifications comme des correctifs séparés, sous la forme de fichiers diff.

Il existe de nombreuses façons différentes de gérer les correctifs dans les paquets Debian. Heureusement, nous sommes en cours de standardisation sur un système, Quilt, qui est désormais utilisé par la majorité des paquets.

Let's look at an example package, `kamoso` in Trusty :

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

Les correctifs sont conservés dans `debian/patches`. Ce paquet a un correctif `kubuntu_01_fix_qmax_on_armel.diff` pour résoudre un échec de compilation sur ARM. Le correctif a un nom explicite pour décrire ce qu'il fait, un numéro pour garder les correctifs dans l'ordre (deux correctifs peuvent se chevaucher s'ils modifient le même fichier) et dans ce cas l'équipe de Kubuntu ajoute son propre préfixe pour montrer que le correctif provient d'eux plutôt que de Debian.

L'ordre des correctifs à appliquer est conservé dans `debian/patches/series`.

1.6.1 Les correctifs avec Quilt

Avant de travailler avec Quilt, vous devrez lui dire où trouver les correctifs. Ajoutez ceci à votre `~/.bashrc` :

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

Et renseignez sur quel fichier appliquer le nouvel export :

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

Par défaut, tous les correctifs sont déjà appliqués sur les vérifications UDD ou les paquets téléchargés. Vous pouvez le vérifier avec :

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


Si vous voulez supprimer le correctif, vous lancerez `pop` :

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

No patches applied
```

Et pour appliquer un correctif, vous utiliserez `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 Ajout d'un nouveau correctif

Pour ajouter un nouveau correctif, vous devez dire à Quilt de créer un nouveau correctif, lui dire quels sont les fichiers que le correctif doit modifier, éditer les fichiers et ensuite actualiser le correctif :

```
$ 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
```

L'étape `quilt add` est importante, si vous l'oubliez les fichiers ne se retrouveront pas dans le correctif.

La modification sera désormais dans `debian/patches/kubuntu_02_program_description.diff` et le dossier `series` sera complété par le nouveau correctif. Vous devez ajouter le nouveau fichier à l'empaquetage :

```
$ 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 conserve ses métadonnées dans le répertoire `.pc/`, vous devez donc actuellement l'ajouter également à l'empaquetage. Cela devrait être amélioré à l'avenir.

Comme règle générale, vous devez rester prudent dans l'ajout de correctifs aux programmes à moins qu'ils ne proviennent de l'amont, car il existe souvent une bonne raison pour que la modification n'ait pas encore été faite. L'exemple ci-dessus modifie une chaîne de l'interface par exemple, elle pourrait donc briser toutes les traductions. En cas de doute, n'hésitez pas à consulter l'auteur de l'amont avant d'ajouter un patch.

1.6.3 En-têtes de correctifs

Nous vous recommandons de marquer chaque correctif avec des en-têtes DEP-3, en les plaçant au début du fichier de correctif. Voici quelques en-têtes que vous pouvez utiliser :

Description Description of what the patch does. It is formatted like `Description` field in `debian/control` : first line is short description, starting with lowercase letter, the next lines are long description, indented with a space.

Author Qui a écrit le correctif (par exemple, « Jane Doe <packager@example.com> »).

Origin D'où provient ce correctif (par exemple « de l'amont »), lorsque *Auteur* n'est pas renseigné.

Bug-Ubuntu Un lien vers le traceur de bogues Launchpad, un format court étant préférable (comme *https ://bugs.launchpad.net/bugs/XXXXXXX*). S'il existe également des bogues dans les traceurs de bogues de l'amont ou de Debian, ajoutez les en-têtes *Bug* ou *Bug-Debian*.

Forwarded Si le correctif a été transmis à l'amont. Soit « yes », « no » ou « not-needed ».

Last-Update Date de la dernière révision (au format "AAAA-MM-JJ").

1.6.4 Mise à niveau vers les nouvelles versions de l'amont

Pour mettre à niveau vers la nouvelle version, vous pouvez utiliser la commande `bzr merge-upstream` :

```
$ bzr merge-upstream --version 2.0.2 https://launchpad.net/ubuntu/+archive/primary/+files/kamoso_2.0
```

Lorsque vous lancez cette commande, tous les correctifs ne seront pas appliqués, car ils peuvent devenir obsolètes. Ils pourraient avoir besoin d'être actualisés, pour correspondre à la nouvelle source de l'amont, ou entièrement supprimés. Pour vérifier les problèmes, appliquez les correctifs un par un :

```
$ 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
```

S'il peut être rétro-appliqué, cela signifie que le correctif a déjà été appliqué par l'amont, nous pouvons donc le supprimer :

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

Ensuite, continuons :

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

C'est une bonne idée de lancer l'actualisation, cela va mettre à jour le correctif relatif à la source modifiée par l'amont :

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

Ensuite, soumettez comme d'habitude :

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

1.6.5 Réaliser un Paquet avec Quilt

Les paquets modernes utilisent Quilt par défaut, il est intégré dans le format d'empaquetage. Vérifiez dans `debian/source/format` que ce dernier est `3.0 (quilt)`.

Les paquets plus anciens utilisant le format source 1.0 vous obligent à utiliser Quilt explicitement, généralement en incluant un fichier makefile dans `debian/rules`.

1.6.6 Configurer Quilt

Vous pouvez utiliser le fichier `~/.quiltrc` pour configurer quilt. Voici quelques options pouvant s'avérer utiles pour l'utilisation de quilt avec debian/packages :

```
# 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 Autres systèmes de correctifs

Les autres systèmes de correctifs utilisés par les paquets incluent `dpatch` et `cdbs simple-patchsys`, qui travaillent de manière similaire à Quilt en conservant les correctifs dans `debian/patches`, mais possèdent des commandes différentes pour rendre applicables, non-applicables ou créer des correctifs. Vous pouvez savoir quel système de correctif est utilisé par un paquet en utilisant la commande `what-patch` (issue du paquet `ubuntu-dev-tools`). Vous pouvez utiliser `edit-patch`, illustré dans un *chapitre précédent*, comme un moyen fiable pour travailler avec tous les systèmes de correctifs.

Dans des paquets encore plus anciens, les modifications sont incluses directement aux sources et conservées dans le fichier source `diff.gz`. Cela rend difficile leur mise à niveau vers les nouvelles versions de l'amont ou la différenciation des correctifs. Il vaut mieux l'éviter.

Ne modifiez pas un système de correctif d'un paquet sans en discuter avec le responsable Debian ou l'équipe Ubuntu adéquate. S'il n'existe pas de système de correctif existant, alors n'hésitez pas à ajouter Quilt.

1.7 Réparation de paquets FTBFS

Avant qu'un paquet puisse être utilisé dans Ubuntu, il doit être compilé depuis le source. S'il échoue (on le qualifie en anglais de FTBFS « Fails To Build From Source »), il attendra probablement dans -proposed et ne sera pas disponible dans les archives Ubuntu. Vous trouverez une liste complète des paquets qui échouent à compiler depuis le source sur http://qa.ubuntuwire.org/ftbfs/. Il y a 5 catégories principales affichées sur la page :

- Package failed to build (F) : (Compilation du paquet a échoué) Quelque chose s'est réellement mal passé pendant la compilation.
- Cancelled build (X) : (Compilation annulée) Le processus de compilation a été annulé pour une raison ou une autre. Il faudrait probablement éviter de débuter avec.
- Package is waiting on another package (M) : (Le paquet attend un autre paquet) Ce paquet attend un autre paquet pour compiler, être mis à jour ou (si le paquet est dans main) une de ses dépendances est dans une mauvaise partie de l'archive.
- Failure in the chroot (C) : (Échec dans le chroot) Une partie du chroot a échoué, cela a de bonnes chances d'être réparé par une recompilation. Demandez à un développeur de recompiler le paquet et il devrait être réparé.
- Failed to upload (U) : (Échec du téléversement) Le paquet n'a pas pu être téléversé. Habituellement, il suffit juste de recompiler, mais vérifiez d'abord le journal de compilation.

1.7.1 Premiers pas

La première chose que vous devrez faire est de voir si vous pouvez reproduire vous-même le FTBFS. Récupérez le code soit en exécutant `bzr branch lp:ubuntu/PACKAGE` et en récupérant le tarball, soit en exécutant ‘‘dget PACKAGE_DSC‘‘sur le fichier .dsc depuis la page launchpad. Une fois que vous l'avez, compilez-le dans un schroot.

Vous devriez pouvoir reproduire le FTBFS. Sinon, vérifiez si la compilation télécharge une dépendance manquante, ce qui signifie que vous avez juste besoin d'en faire une dépendance de compilation dans debian/control. Compiler le paquet localement peut aussi aider à trouver si le problème est causé par une dépendance manquante, non listée (compile localement mais échoue sur un schroot).

1.7.2 Vérification de Debian

Une fois que vous avez reproduit le problème, il est temps d'essayer de trouver une solution. Si le paquet est aussi dans Debian, vous pouvez vérifier s'il y compile en allant à http://packages.qa.debian.org/PACKAGE. Si Debian a une version plus récente, vous devriez la fusionner. Sinon, recherchez dans les journaux de compilation et les bogues répertoriés sur cette page s'il y a des informations supplémentaires sur les ftbfs ou les correctifs. Debian tient aussi à jour une liste de commandes FTBFS (commandes qui font échouer la compilation) et ce qu'il convient de faire pour les corriger que vous trouverez à https://wiki.debian.org/qa.debian.org/FTBFS, vous devez aussi y rechercher des solutions.

1.7.3 Autres causes de paquets FTBFS

Si un paquet se trouvant dans main a une dépendance manquante qui n'est pas dans main, vous devrez déposer un rapport de bogue MIR. https://wiki.ubuntu.com/MainInclusionProcess explique la procédure.

1.7.4 Résoudre le problème

Une fois que vous avez trouvé comment corriger le problème, suivez la même démarche que pour tout autre bogue. Créez un correctif, ajoutez-le à une branche bzr ou à un bogue, abonnez-y ubuntu-sponsors, puis essayez de le faire inclure en amont et/ou dans Debian.

1.8 Bibliothèques partagées

Les bibliothèques partagées sont du code compilé destiné à être partagé entre plusieurs différents programmes. Ils sont distribués en tant que fichiers `.so` dans `/usr/lib/`.

Une bibliothèque exporte des symboles qui sont les versions compilées de fonctions, de classes et de variables. Une bibliothèque possède ce qui s'appelle un SONAME comprenant un numéro de version. Cette version de SONAME ne correspond pas nécessairement au numéro de version publique. Un programme est compilé avec une version SONAME donnée de la bibliothèque. Si l'un des symboles est retiré ou modifié, alors le numéro de version doit être changé, ce qui oblige la recompilation de tous les paquets utilisant cette bibliothèque avec la nouvelle version. Les numéros de version sont généralement fixés par l'amont et nous les suivons dans nos noms de paquets binaires à l'aide d'un nombre ABI, mais parfois les amonts n'utilisent pas de numéros logiques de version et les empaqueteurs doivent conserver des numéros de version séparés.

Les bibliothèques sont généralement distribuées par l'amont sous forme de versions autonomes. Parfois, elles sont distribuées comme partie d'un programme. Dans ce cas, elles peuvent être incluses dans le paquet binaire avec le programme (c'est ce qu'on appelle le regroupement) lorsqu'il n'est pas prévu que d'autres programmes utilisent la bibliothèque. Le plus souvent, elles sont divisées en plusieurs paquets binaires séparés.

Les bibliothèques elles-mêmes sont mises dans un paquet binaire nommé `libfoo1` où `foo` est le nom de la bibliothèque et `1` la version de SONAME. Les fichiers de développement issus du paquet, tels que les fichiers d'en-tête, nécessaires pour compiler des programmes avec la bibliothèque sont placés dans un paquet appelé `libfoo-dev`.

1.8.1 Un exemple

Nous allons utiliser libnova comme exemple :

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

Pour trouver le SONAME de la bibliothèque, lancez :

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

Le SONAME est `libnova-0.12.so.2`, qui correspond au nom du fichier (généralement c'est le cas, mais pas toujours). Ici, l'amont a mis le numéro de version comme partie du SONAME et lui a donné `2` comme version d'ABI. Les noms de paquets de bibliothèque devraient suivre le SONAME de la bibliothèque les contenant. Le paquet binaire de bibliothèque s'appelle `libnova-0.12-2`, où `libnova-0.12` est le nom de la bibliothèque et `2` est notre ABI.

Si l'amont apporte des modifications incompatibles avec leur bibliothèque, il devra revoir la version de SONAME et nous devrons renommer notre bibliothèque. Tous les autres paquets utilisant notre paquet de bibliothèque devront être recompilés à la nouvelle version, c'est ce qu'on appelle une transition et peut demander un certain travail. Heureusement, notre nombre ABI continuera à correspondre au SONAME des amonts, mais parfois ils introduisent des incompatibilités sans changer leurs numéros de version et nous devrons changer les nôtres.

En regardant dans debian/libnova-0.12-2.install, nous voyons qu'il comprend deux fichiers :

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

Le dernier est la bibliothèque réelle, se terminant par un numéro de version mineur et un point. Le premier est un lien symbolique pointant vers la bibliothèque réelle. Le lien symbolique est ce que les programmes utilisant la bibliothèque rechercheront, les programmes en cours d'exécution ne se souciant pas du numéro de version mineur.

`Libnova-dev.install` inclut tous les fichiers nécessaires pour compiler un programme utilisant cette bibliothèque. Les fichiers d'en-tête, un binaire de configuration, le fichier `.la` d'utilitaires bibliothèque et `libnova.so` qui est un autre lien symbolique pointant vers la bibliothèque, les programmes compilant avec cette bibliothèque ne se souciant pas du numéro de version majeur (même si le binaire qu'ils compilent le feront).

`.la` libtool files are needed on some non-Linux systems with poor library support but usually cause more problems than they solve on Debian systems. It is a current Debian goal to remove .la files and we should help with this.

1.8.2 Les bibliothèques statiques

Les paquets -dev délivrent aussi `usr/lib/libnova.a`. Il s'agit d'une bibliothèque statique, une alternative à la bibliothèque partagée. Tout programme compilé avec la bibliothèque statique comprendra le répertoire du code en lui-même. Cela contourne le souci de la compatibilité binaire de la bibliothèque. Mais cela signifie aussi que tous les bogues, y compris les problèmes de sécurité, ne seront pas mis à jour avec la bibliothèque jusqu'à ce que le programme soit recompilé. Pour cette raison, l'usage de programmes utilisant des bibliothèques statiques est déconseillé.

1.8.3 Les fichiers de symboles

Quand un paquet est construit à partir d'une bibliothèque, le mécanisme `shlibs` ajoute une dépendance de paquet sur cette bibliothèque. C'est pourquoi de nombreux programmes auront `Depends: ${shlibs:Depends}` dans `debian/control`. Ceci est remplacé par les dépendances de bibliothèque à la compilation. Cependant shlibs peut seulement le faire dépendre du numéro de version majeure ABI, `2` dans notre exemple libnova. Si de nouveaux symboles sont ajoutés à libnova 2.1, un programme utilisant ces symboles pourrait toujours être installé avec libnova ABI 2.0, ce qui provoquerait son plantage.

Pour rendre les dépendances de bibliothèques plus précises, nous gardons les fichiers `.symbols` qui répertorient tous les symboles d'une bibliothèque et la version à laquelle ils sont apparus.

libnova n'a pas de fichier de symboles, nous pouvons donc en créer un. Commencez par la compilation du paquet :

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

Le `-nc` terminera la compilation sans supprimer les fichiers de construction. Modifiez le fichier compilé et exécutez `dpkg-gensymbols` pour le paquet de bibliothèque :

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

Cela crée un fichier diff que vous pouvez rendre automatiquement applicable :

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

Ce qui va créer un fichier nommé de manière similaire à `dpkg-gensymbolsnY_WWI` listant tous les symboles. Il répertorie également la version actuelle du paquet. Nous pouvons supprimer la version d'empaquetage de celles indiquées dans le fichier de symboles car de nouveaux symboles ne sont généralement pas ajoutés par de nouvelles versions d'empaquetage, mais par les développeurs de l'amont :

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

Maintenant, déplacez le fichier à sa place, soumettez et faites un test de compilation :

```
$ 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
```

Si la compilation s'achève avec succès, le fichier de symboles est correct. Avec la prochaine version amont de libnova, vous devrez exécuter dpkg-gensymbols à nouveau et cela vous donnera un fichier diff pour mettre à jour le fichier de symboles.

1.8.4 Les fichiers C++ de la bibliothèque de symboles

C++ has even more exacting standards of binary compatibility than C. The Debian Qt/KDE Team maintain some scripts to handle this, see their Working with symbols files page for how to use them.

1.8.5 Lectures complémentaires

Junichi Uekawa's Debian Library Packaging Guide goes into this topic in more detail.

1.9 Rétroportage de mises à jour logicielles

Sometimes you might want to make new functionality available in a stable release which is not connected to a critical bug fix. For these scenarios you have two options : either you upload to a PPA or prepare a backport.

1.9.1 Personal Package Archive (PPA)

L'utilisation d'un PPA a un certain nombre d'avantages. C'est assez simple, vous n'avez besoin d'aucune approbation, mais l'inconvénient, c'est que vos utilisateurs devront l'activer manuellement. C'est une source de logiciels non standard.

The PPA documentation on Launchpad is fairly comprehensive and should get you up and running in no time.

1.9.2 Rétroportages officiels d'Ubuntu

Le projet Backports est un moyen d'offrir de nouvelles fonctionnalités aux utilisateurs. En raison des risques inhérents à la stabilité du rétroportage des paquets, les utilisateurs n'obtiennent pas les paquets rétroportés sans action explicite de leur part. Cela rend généralement le rétroportage inapproprié pour corriger les bogues. Si un paquet dans une mise à jour de Ubuntu comporte un bogue, il doit être réparé par la *Mise à jour de Sécurité ou le Processus de Mise à jour en Version Stable*, selon le cas.

Une fois que vous avez déterminé un paquet à rétroporter vers une version stable, vous aurez besoin de le tester y compris en construction sur la version stable donnée. `pbuilder-dist` (dans le paquet `ubuntu-dev-tools`) est un outil très pratique pour y arriver facilement.

Pour signaler la demande de rétroportage et la faire traiter par l'équipe Backporters, vous pouvez utiliser l'outil `requestbackport` (également dans le paquet `ubuntu-dev-tools`). Il déterminera les versions intermédiaires dans lesquelles le paquet doit être rétroporté, énumérera toutes les dépendances inverses, et déposera la demande de rétroportage. Il inclura également une liste de vérification des tests dans le bogue.

CHAPITRE 2

Base de connaissances

2.1 Communication dans Ubuntu développement

Dans un projet où des milliers de lignes de code sont modifiées, de nombreuses décisions sont prises et des centaines de personnes interagissent chaque jour, il est important de communiquer efficacement.

2.1.1 Listes de diffusion

Les listes de diffusion sont un outil très important si vous voulez communiquer des idées à une plus large équipe et vous assurer d'atteindre tout le monde, quels que soient les fuseaux horaires.

En termes de développement, ce sont les plus importants :

— https://lists.ubuntu.com/mailman/listinfo/ubuntu-devel-announce (annonces uniquement, les annonces de développement les plus importantes vont ici)
— https://lists.ubuntu.com/mailman/listinfo/ubuntu-devel (discussion générale sur le développement Ubuntu)
— https://lists.ubuntu.com/mailman/listinfo/ubuntu-motu (discussion de l'équipe MOTU, obtenir de l'aide avec l'empaquetage)

2.1.2 Canaux IRC

Pour discuter en temps réel, connectez-vous à irc.freenode.net et rejoignez un ou plusieurs de ces canaux :

— #ubuntu-devel (pour une discussion générale sur le développement)
— #ubuntu-motu (pour une discussion avec l'équipe MOTU et généralement obtenir de l'aide)

2.2 Aperçu élémentaire du dossier `debian/`

Cet article va vous expliquer brièvement les différents fichiers importants pour l'empaquetage des paquets Ubuntu, contenus dans le répertoire `debian/`. Les plus importants d'entre eux sont `changelog`, `control`, `copyright` et `rules`. Ceux-ci sont nécessaires pour tous les paquets. Certains fichiers supplémentaires dans `debian/` peuvent être utilisés afin de personnaliser et de configurer le comportement du paquet. Certains de ces fichiers sont décrits dans cet article, mais il n'est pas censé constituer une liste exhaustive.

2.2.1 Le changelog

Ce fichier est, comme son nom l'indique, une liste des modifications apportées à chaque version. Il possède un format spécifique donnant le nom du paquet, la version, la distribution, les modifications, et qui les a apportées à un moment

donné. Si vous avez une clé GPG (voir : *Obtenir configuration*), assurez-vous d'utiliser les mêmes nom et adresse email dans le changelog que dans votre clé. Ce qui suit est un modèle de changelog :

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

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

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

Le format (particulièrement celui de la date) est important. La date doit être au format **RFC 5322**, obtenu en utilisant la commande date -R. Pour plus de commodité, la commande dch est utilisée pour éditer le changelog. Elle mettra automatiquement la date à jour.

Les points mineurs sont indiquées par un tiret "-", tandis que les points majeurs utilisent un astérisque "*".

Si vous êtes parti de zéro pour l'empaquetage, dch --create (dch est dans le paquet devscripts) vous créera un debian /changelog standard.

Voici un exemple de fichier changelog pour bonjour :

```
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
```

Notez que la version comporte un -0ubuntu1 annexé, c'est la révision de la distribution, utilisée de telle sorte que l'empaquetage peut être mis à jour (pour par exemple corriger des bogues) avec de nouveaux ajouts dans la même version de la source.

Ubuntu et Debian ont des systèmes légèrement différents d'incrémentation de version de paquets pour éviter que des paquets issus d'une source de même version entrent en conflit. Lorsqu'un paquet Debian a été modifié dans Ubuntu, il présente un ubuntuX (où X est le numéro de révision Ubuntu) annexé à la fin de la version Debian. Donc, si le paquet Debian hello 2.6-1 a été modifié par Ubuntu, la chaîne de version indiquera 2.6-1ubuntu1. Si ce paquet pour l'application n'existe pas dans Debian, la révision Debian est 0 (par exemple, hello 2.6-0ubuntu1).

For further information, see the changelog section (Section 4.4) of the Debian Policy Manual.

2.2.2 Le fichier de contrôle.

Le fichier control contient les informations que les gestionnaires de paquets (comme apt-get, synaptic ou adept) utilisent, les moments de construction des dépendances, les informations concernant les mainteneurs et plus encore.

Pour le paquet hello de Ubuntu, le fichier control ressemble à ceci :

```
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).
```

Le premier paragraphe décrit le paquet source, y compris la liste des paquets nécessaires pour construire le paquet depuis son source dans le champ `Build-Depends`. Il contient également des méta-informations comme le nom du mainteneur, la version de la Charte Debian à laquelle le paquet se conforme, l'emplacement du dépôt du contrôle de version de l'empaquetage et la page d'accueil en amont.

Note that in Ubuntu, we set the `Maintainer` field to a general address because anyone can change any package (this differs from Debian where changing packages is usually restricted to an individual or a team). Packages in Ubuntu should generally have the `Maintainer` field set to `Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>`. If the Maintainer field is modified, the old value should be saved in the `XSBC-Original-Maintainer` field. This can be done automatically with the `update-maintainer` script available in the `ubuntu-dev-tools` package. For further information, see the Debian Maintainer Field spec on the Ubuntu wiki.

Chaque paragraphe supplémentaire décrit un paquet binaire à compiler.

For further information, see the control file section (Chapter 5) of the Debian Policy Manual.

2.2.3 Le fichier de copyright

This file gives the copyright information for both the upstream source and the packaging. Ubuntu and Debian Policy (Section 12.5) require that each package installs a verbatim copy of its copyright and license information to `/usr/share/doc/$(package_name)/copyright`.

En règle générale, les informations de copyright se trouvent dans le fichier `COPYING` du répertoire source du programme. Ce fichier doit contenir des informations telles que les noms des auteurs et de l'empaqueteur, l'URL de provenance de la source, une ligne Copyright indiquant l'année et le titulaire du copyright, et le texte du copyright lui-même. Un exemple pouvant servir de modèle serait :

```
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'.
```

This example follows the Machine-readable debian/copyright format. You are encouraged to use this format as well.

2.2.4 Le fichier des règles

Le dernier fichier que nous devons examiner est `rules`. Il complète tout le travail de création de notre paquet. Il s'agit d'un Makefile ayant pour objectifs de compiler et d'installer l'application, puis de créer le fichier `.deb` à partir des fichiers installés. Il a également pour objectif de nettoyer tous les fichiers de construction de telle sorte que vous vous retrouvez au final uniquement avec un paquet source.

Voici une version simplifiée du fichier de règles créé par `dh_make` (qui peut être trouvé dans le paquet `dh-make`) :

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

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

%:
	dh  $@
```

Passons ce fichier en revue de détails. Tout ce qu'il fait est de passer chaque cible construite qu'appelle ce `debian/rules` comme argument à `/usr/bin/dh`, qui appelle lui-même toutes les commandes `dh_*` nécessaires.

`dh` exécute une séquence de commandes de debhelper. Les séquences prises en charge correspondent aux objectifs du fichier « debian/rules » : « build », « clean », « install », « binary-arch », « binary-indep » et « binary ». Afin de voir quelles commandes sont lancées dans chaque cible, exécutez :

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

Les commandes de la séquence binary-indep sont passées avec l'option « -i » pour s'assurer qu'elles ne fonctionnent que sur des paquets binaires indépendants, et les commandes de la séquence binary-arch sont passées avec l'option « -a » pour s'assurer qu'elles ne fonctionnent que sur les paquets dépendants de l'architecture.

Chaque commande debhelper s'enregistrera après une exécution réussie dans `debian/package.debhelper.log`. (Que dh_clean efface.) Ainsi, dh peut révéler quelles commandes ont déjà été exécutées, pour quels paquets, et ignorera une nouvelle instance d'exécution de ces commandes.

A chaque exécution de `dh`, il examine le journal, et retrouve les dernières commandes historisées dans une séquence spécifiée. Il continue ensuite avec la commande suivante de la séquence. Les options `--until`, `--before`, `--after`, et `--remaining` modifient ce comportement.

Si debian/rules contient une cible nommée override_dh_command, alors quand il arrive à cette commande de la séquence, dh traitera cette cible à partir du fichier de règles, plutôt que de lancer la commande réelle. La cible modifiée peut ensuite exécuter la commande avec des options supplémentaires, ou exécuter des commandes complètement différentes à la place. (Notez que pour utiliser cette fonctionnalité, vous devez construire les dépendances à partir de debhelper 7.0.50 ou supérieur.)

Jetez un œil à usr/share/doc/debhelper/examples/ et man dh pour plus d'exemples. Voir également la section rules (Section 4.9) de la Charte Debian.

2.2.5 Les fichiers additionnels

Le fichier d'installation

Le fichier install est utilisé par dh_install pour installer les fichiers dans le paquet binaire. Il s'utilise classiquement dans deux cas :

— Pour installer des fichiers dans votre paquet lorsqu'ils ne sont pas gérés par le système de construction en amont.
— Fractionner un important et unique paquet source en plusieurs paquets binaires.

Dans le premier cas, le fichier install doit comporter une ligne par fichier installé, en précisant à la fois le fichier et le répertoire d'installation. Par exemple, le fichier install suivant installera le script foo dans le répertoire racine du paquet source de usr/bin et un fichier desktop dans le répertoire debian de usr/share/applications :

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

Lorsqu'un paquet source produit plusieurs paquets binaires dh va installer les fichiers dans debian/tmp plutôt que directement dans debian/<paquet>. Les fichiers installés dans debian/tmp peuvent alors être déplacés en paquets binaires séparés en utilisant plusieurs fichiers $package_name.install. Cela est couramment utilisé pour sortir de grandes quantités de données indépendantes de l'architecture hors de paquets dépendants de l'architecture ainsi qu'à l'intérieur de paquets Architecture : all. Dans ce cas, seul le nom des fichiers (ou des répertoires) à installer sont nécessaires, en omettant le répertoire d'installation. Par exemple, foo.install contenant uniquement des fichiers dépendants de l'architecture pourrait ressembler à :

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

Alors que le foo-common.install contenant uniquement des fichiers indépendants de l'architecture pourrait ressembler à :

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

Cela créera deux paquets binaires, ‘‘ foo‘‘ et foo-common. Les deux exigeraient leur propre paragraphe dans debian/control.

Voir man dh_install et la section du fichier install (Section 5.11) du Nouveau Guide des Mainteneurs Debian pour plus de détails.

Le fichier watch

Le fichier debian/watch nous permet de vérifier automatiquement les nouvelles versions amont en utilisant l'outil uscan se trouvant dans le paquet devscripts. La première ligne du fichier doit être la version du format (3, au moment d'écrire ces lignes), tandis que les lignes suivantes contiennent des URL à analyser. Par exemple :


```
version=3

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

L'exécution de `uscan` dans le répertoire racine source ira comparer le numéro de version amont dans `debian/changelog` avec celui de la dernière version amont disponible. Si une nouvelle version amont est trouvée, elle sera automatiquement téléchargée. Par exemple :

```
$ 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
```

If your tarballs live on Launchpad, the `debian/watch` file is a little more complicated (see Question 21146 and Bug 231797 for why this is). In that case, use something like :

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

Pour plus d'informations, consultez `man uscan` et la section fichier watch (Section 4.11) de la Charte Debian.

Pour une liste de paquets où les fichiers de rapport `watch` ne sont pas synchronisés avec l'amont, voir Etat de santé externe Ubuntu.

Le fichier source/format

Ce fichier indique le format du paquet source. Il doit contenir une seule ligne indiquant le format désiré :

- `3.0 (native)` pour les paquets Debian natifs (pas de version amont)
- `3.0 (quilt)` pour les paquets avec une archive séparée en amont
- `1.0` pour les paquets souhaitant déclarer explicitement le format par défaut

Actuellement, le format de paquet source sera 1.0 par défaut si ce fichier n'existe pas. Vous pouvez rendre cela explicite dans le fichier source/format. Si vous choisissez de ne pas utiliser ce fichier pour définir le format de la source, Lintian vous avertira de ce fichier manquant. Cet avertissement est purement informatif et peut être ignoré en toute sécurité.

Nous vous encourageons à utiliser le nouveau format de source 3.0. Il fournit un certain nombre de nouvelles fonctionnalités :

- Prise en charge de formats de compression supplémentaires : bzip2, lzma et xz
- Prise en charge de plusieurs archives amont
- Il n'est pas nécessaire de rempaqueter l'archive amont pour dépouiller le répertoire debian
- Les changements spécifiques à Debian ne sont plus stockés dans un seul fichier .diff.gz, mais dans plusieurs correctifs compatibles avec quilt dans `debian/patches/`

https://wiki.debian.org/Projects/DebSrc3.0 summarizes additional information concerning the switch to the 3.0 source package formats.

See `man dpkg-source` and the source/format section (Section 5.21) of the Debian New Maintainers' Guide for additional details.

2.2.6 Ressources supplémentaires

In addition to the links to the Debian Policy Manual in each section above, the Debian New Maintainers' Guide has more detailed descriptions of each file. Chapter 4, "Required files under the debian directory" further discusses the control, changelog, copyright and rules files. Chapter 5, "Other files under the debian directory" discusses additional files that may be used.

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 Mikołajczak <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 : tests automatiques pour les paquets

The DEP 8 specification defines how automatic testing can very easily be integrated into packages. To integrate a test into a package, all you need to do is :

— ajoutez un fichier appelé `debian/tests/control` qui spécifie les exigences pour le banc d'essai,
— ajoutez les tests dans `debian/tests/`.

2.4.1 Exigences du banc d'essai

In `debian/tests/control` you specify what to expect from the testbed. So for example you list all the required packages for the tests, if the testbed gets broken during the build or if `root` permissions are required. The DEP 8 specification lists all available options.

Ci-dessous, nous voyons un aperçu du paquet source `glib2.0`. Dans un cas très simple, le fichier devrait ressembler à ceci :

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

Pour le test dans `debian/tests/build`, cela permettrait de s'assurer que les paquets `libglib2.0-dev` et `build-essential` sont installés.

Note : Vous pouvez utiliser `@` de la ligne `Depends` pour indiquer que vous souhaitez tous les paquets installés construits par la paquet source en question.

2.4.2 Les tests réels

Le test accompagnant l'exemple ci-dessus pourrait être :

```
#!/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"
```

Ici, un morceau très simple de code C est écrit dans un répertoire temporaire. C'est ensuite compilé avec les bibliothèques système (en utilisant les drapeaux et chemins des bibliothèques fournis par *pkg-config*). Ensuite, le binaire compilé, qui fait juste appel à quelques fonctionnalités glib de parties du noyau, est exécuté.

While this test is very small and simple, it covers quite a lot : that your -dev package has all necessary dependencies, that your package installs working pkg-config files, headers and libraries are put into the right place, or that the compiler and linker work. This helps to uncover critical issues early on.

2.4.3 Exécution du test

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 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.

The Ubuntu CI system uses the QEMU runner and runs the tests from the packages in the archive, with `-proposed` enabled. To reproduce the exact same environment, first install the necessary packages :

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

Now build a testbed with :

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

(Please see its manpage and `--help` output for selecting different releases, architectures, output directory, or using proxies). This will build e. g. `adt-trusty-amd64-cloud.img`.

Then run the tests of a source package like `libpng` in that QEMU image :

```
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 Exemples complémentaires

Cette liste n'est pas exhaustive mais elle peut vous aider à mieux comprendre comment les tests automatiques sont implémentés et utilisés dans Ubuntu.

- The libxml2 tests are very similar. They also run a test-build of a simple piece of C code and execute it.
- The gtk+3.0 tests also do a compile/link/run check in the "build" test. There is an additional "python3-gi" test which verifies that the GTK library can also be used through introspection.
- In the ubiquity tests the upstream test-suite is executed.
- The gvfs tests have comprehensive testing of their functionality and are very interesting because they emulate usage of CDs, Samba, DAV and other bits.

2.4.5 infrastructure Ubuntu

Packages which have `autopkgtest` enabled will have their tests run whenever they get uploaded or any of their dependencies change. The output of automatically run autopkgtest tests can be viewed on the web and is regularly updated.

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 Faire passer le test dans Ubuntu

Le processus de soumission d'un autopkgtest pour un paquet est très similaire à *corriger un bogue dans Ubuntu*. Essentiellement, il vous suffit de :

- exécuter `bzr branch ubuntu:<nomdupaquet>`,
- modifier `debian/control` pour activer les tests,
- ajouter le répertoire `debian/tests`,
- write the `debian/tests/control` based on the DEP 8 Specification,
- ajouter votre cas de test(s) à `debian/tests`,
- valider vos modifications, les téléverser vers Launchpad, proposer une fusion et obtenir son examen, exactement comme pour toute autre amélioration dans un paquet source.

2.4.7 Ce que vous pouvez faire

The Ubuntu Engineering team put together a list of required test-cases, where packages which need tests are put into different categories. Here you can find examples of these tests and easily assign them to yourself.

If you should run into any problems, you can join the #ubuntu-quality IRC channel to get in touch with developers who can help you.

2.5 Utilisation des environnements Chroots

Si vous utilisez une version d'Ubuntu, mais que vous travaillez sur les paquets pour une autre version, vous pouvez créer l'environnement de l'autre version avec un `chroot`.

Un environnement `chroot` vous permet d'avoir un système de fichiers complet d'une autre distribution avec lequel vous pouvez travailler tout à fait normalement. Il évite la surcharge engendrée par l'exécution d'une machine virtuelle complète.

2.5.1 Créer un environnement Chroot

Utilisez la commande `debootstrap` pour créer un nouvel environnement chroot :

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

This will create a directory `trusty` and install a minimal trusty system into it.

If your version of `debootstrap` does not know about Trusty you can try upgrading to the version in `backports`.

Vous pouvez alors travailler dans l'environnement chroot :

```
$ sudo chroot trusty
```

Où vous pouvez installer ou désinstaller le paquet que vous souhaitez sans affecter votre système principal.

Vous devriez copier vos clés GPG/ssh et votre configuration Bazaar dans l'environnement chroot de manière à accéder aux paquets et les signer directement :

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

Pour empêcher apt et d'autres programmes de se plaindre de l'absence d'informations linguistiques, vous pouvez installer les paquets linguistiques adéquats :

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

Si vous voulez exécuter des programmes X, vous devez lier le répertoire /tmp dans le chroot, en dehors de chroot, exécutez :

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

Certains programmes peuvent nécessiter que vous liez /dev ou /proc.

For more information on chroots see our Debootstrap Chroot wiki page.

2.5.2 Alternatives

SBuild is a system similar to PBuilder for creating an environment to run test package builds in. It closer matches that used by Launchpad for building packages but takes some more setup compared to PBuilder. See the Security Team Build Environment wiki page for a full explanation.

Full virtual machines can be useful for packaging and testing programs. TestDrive is a program to automate syncing and running daily ISO images, see the TestDrive wiki page for more information.

Vous pouvez également configurer pbuilder pour marquer un arrêt lorsqu'il rencontre un échec de compilation. Copiez C10shell depuis /usr/share/doc/pbuilder/examples dans un répertoire et utilisez l'argument `--hookdir=` pour désigner ce dernier.

Amazon's EC2 cloud computers allow you to hire a computer paying a few US cents per hour, you can set up Ubuntu machines of any supported version and package on those. This is useful when you want to compile many packages at the same time or to overcome bandwidth restraints.

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 Empaquetage pour KDE

Packaging of KDE programs in Ubuntu is managed by the Kubuntu and MOTU teams. You can contact the Kubuntu team on the Kubuntu mailing list and `#kubuntu-devel` Freenode IRC channel. More information about Kubuntu development is on the Kubuntu wiki page.

Our packaging follows the practices of the Debian Qt/KDE Team and Debian KDE Extras Team. Most of our packages are derived from the packaging of these Debian teams.

2.7.1 Politique de correction

Kubuntu n'ajoute aucun correctif aux programmes KDE, sauf s'ils proviennent des auteurs de l'amont ou soumis à l'amont dans l'espoir qu'ils soient fusionnés au plus tôt, ou si nous avons consulté le problème avec les auteurs de l'amont.

Kubuntu ne modifie pas l'image de marque des paquets sauf si l'amont y compte (comme le logo en haut à gauche du menu Kickoff) ou si cela simplifie (comme la suppression des écrans de démarrage).

2.7.2 debian/rules

Les paquets Debian comprennent des ajouts à l'utilisation basique de Debhelper. Ceux-ci sont conservés dans le paquet `pkg-kde-tools`.

Les paquets utilisant Debhelper 7 doivent ajouter l'option `--with=kde`. Cela permettra de s'assurer que les options correctes de compilation sont utilisées et ajoutera des options telles que la gestion des souches kdeinit et traductions :

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


Quelques récents paquets KDE utilisent le système `dhmk`, une alternative à `dh` créé par l'équipe Debian Qt /KDE. Vous pouvez lire à ce sujet /usr/share/pkg-kde-tools/qt-kde-team/2/README. Les paquets utilisant ce système `inclueront /usr/share/pkg-kde-tools/qt-kde-team/2/debian-qt-kde.mk` au lieu d'éxecuter `dh`.

2.7.3 Traductions

Les traductions des paquets principaux sont importées dans Launchpad et exportés de Launchpad vers Ubuntu dans les paquets de langues.

Ainsi, tout paquet KDE principal doit générer des modèles de traduction, inclure ou mettre à disposition les traductions de l'amont et gérer les traductions du fichier `.desktop`.

Pour générer des modèles de traduction le paquet doit comprendre un fichier `Messages.sh` ; demandez à l'amont, s'il n'existe pas. Vous pouvez vérifier qu'il fonctionne en exécutant `extract-messages.sh` qui devrait produire un ou plusieurs fichiers `.pot``dans le dossier ``po/`. Cela se fera automatiquement lors de la compilation si vous utilisez l'option `--with=kde` de `dh`.

Upstream will usually have also put the translation `.po` files into the `po/` directory. If they do not, check if they are in separate upstream language packs such as the KDE SC language packs. If they are in separate language packs Launchpad will need to associate these together manually, contact David Planella to do this.

Si un paquet est déplacé du dépôt « universe » au dépôt « main », il devra être téléchargé de nouveau avant que les traductions soient importées vers Launchpad.

Les fichiers `.desktop` ont également besoin de traductions. Nous corrigeons KDElibs pour lire les traductions depuis les fichiers `.po` désignés par une ligne `X-Ubuntu-Gettext-Domain=` ajoutée aux fichiers `.desktop` au moment de la compilation du paquet. Un fichier .pot est généré pour chaque paquet lors de la compilation et les fichiers .po sont téléchargés depuis l'amont et inclus dans le paquet ou dans nos paquets de langue. La liste des fichiers .po à télécharger depuis les dépôts KDE se trouve dans `/usr/lib/kubuntu-desktop-i18n/desktop-template-list`.

2.7.4 Bibliothèque de symboles

Library symbols are tracked in `.symbols` files to ensure none go missing for new releases. KDE uses C++ libraries which act a little differently compared to C libraries. Debian's Qt/KDE Team have scripts to handle this. See Working with symbols files for how to create and keep these files up to date.

CHAPITRE 3

Lectures complémentaires

You can read this guide offline in different formats, if you install one of the binary packages.

Si vous souhaitez en savoir plus à propos de la construction de paquets Debian, voici quelques ressources Debian que vous trouverez utiles :

- How to package for Debian ;
- Debian Policy Manual ;
- Debian New Maintainers' Guide — available in many languages ;
- Packaging tutorial (also available as a package) ;
- Guide for Packaging Python Modules.

We are always looking to improve this guide. If you find any problems or have some suggestions, please report a bug on Launchpad. If you'd like to help work on the guide, grab the source there as well.