[doc.ubuntu-fr.org](https://doc.ubuntu-fr.org/tutoriel/creer_un_paquet)

**tutoriel:creer\_un\_paquet [Wiki ubuntu-fr]**

21-27 minutes

Dans une version récente d'un système [GNU](https://doc.ubuntu-fr.org/gnu)/[Linux](https://doc.ubuntu-fr.org/linux) comme peut l'être Ubuntu, l'installation de logiciels est devenue extrêmement facile grâce à [La logithèque](https://doc.ubuntu-fr.org/software-center) ou aux outils [apt-get](https://doc.ubuntu-fr.org/apt-get) (en ligne de commande) ou [Synaptic](https://doc.ubuntu-fr.org/synaptic) (Gnome) ou [Muon](https://doc.ubuntu-fr.org/muon) (KDE). Ces outils donnent accès à des [dépôts](https://doc.ubuntu-fr.org/depots) contenant plus de 49000 paquets [1)](about:reader?url=https%3A%2F%2Fdoc.ubuntu-fr.org%2Ftutoriel%2Fcreer_un_paquet#fn__1). Pourtant, la richesse de l'univers des [logiciels libres](https://fr.wikipedia.org/wiki/logiciels%20libres) fait que, malgré ce nombre important, il existe encore des applications qui ne sont pas empaquetées. Nous allons donc nous intéresser ici à la création de paquets Debian (les fameux fichiers « .deb », utilisés aussi dans Ubuntu) à partir des sources d'une application existante.

Il est aussi possible de créer un paquet d'une façon très simple avec [checkinstall](https://doc.ubuntu-fr.org/checkinstall). Cette est décrit la procédure détaillée pour une validation sur des dépôts officiels que ne permet pas checkinstall.

Pour pouvoir suivre cette documentation un certain nombre de pré-requis sont nécessaires :

**Paquet binaire et paquet source**

Commençons par une petite précision technique :

- Un paquet binaire est un paquet (.deb) qui contient les fichiers nécessaires à l'application pour pouvoir fonctionner sur votre ordinateur. C'est ce que nous allons nous efforcer de créer. Tous les paquets binaires disponibles dans les dépôts Ubuntu ont été construits à partir de paquets sources.

- Un paquet source est un ensemble de fichiers (attention le terme de paquet est utilisé par analogie. Il n'y a pas ici de fichier conteneur comme peut l'être le .deb pour les binaires) contenant les sources originelles de l'application, ainsi que les indications des modifications nécessaires à la création du paquet debian.

**Étude d'un exemple de paquet source**

Pour illustrer cela, nous allons examiner le contenu d'un paquet source. Pour cela il faut le télécharger d'un des dépôts source configuré pour apt.

Commençons par créer le dossier dans lequel nous allons travailler.

mkdir ~/packaging && cd ~/packaging

Et téléchargeons un paquet source :

apt-get source xfce4-xkb-plugin

ls

La commande « ls » nous donne comme contenu :

Vous obtiendrez exactement ces fichiers si vous êtes sous Dapper, peut-être une autre version sous d'autres versions d'Ubuntu.

Le fichier **xfce4-xkb-plugin\_0.4.1.orig.tar.gz** est en fait l'archive contenant les fichiers sources tels que l'on peut les obtenir sur le site du projet (typiquement ceux que l'on installe avec « ./configure && make && make install »). Le répertoire est d'ailleurs issu de la décompression de cette archive. Seul le nom du fichier a été changé pour des raisons que nous préciserons plus tard.

Le fichier **xfce4-xkb-plugin\_0.4.1-0ubuntu5.dsc** contient la description du paquet source (et des deux autres fichiers, avec leur taille et leur somme MD5).

Le fichier **xfce4-xkb-plugin\_0.4.1-0ubuntu5.diff.gz** contient toutes les modifications nécessaires pour construire le paquet Debian. Constituer un paquet source nécessite d'ajouter aux sources existantes un répertoire debian/ avec quelques fichiers. C'est ce répertoire que contient le fichier « .diff.gz ».

Maintenant que l'on a une idée de ce que sont les paquets sources, nous allons créer notre premier paquet.

**Création du paquet source**

Pour notre exemple nous allons construire le paquet de l'application [ePDFView](https://doc.ubuntu-fr.org/epdfview), petite application de consultation de documents PDF.

Après avoir supprimé le contenu de notre dossier « ~/packaging » pour repartir d'une situation propre, on télécharge les sources du dit logiciel.

Donc dans **~/packaging** :

Faites attention à bien être dans **~/packaging** : rm -rf efface tout sans confirmation !

cd ~/packaging && rm -rf \* # on nettoie :-)

mkdir epdfview && cd epdfview

wget http://gauvain.pocentek.net/u-classroom/2006-08-31/epdfview-0.1.5.tar.gz

On utilise ici le site de Gloubiboulga en lieu et place du site officiel, car ce dernier est particulièrement lent. Une fois les sources récupérées, la **première** chose à faire est de renommer le tarball, afin de pouvoir créer un paquet source correct par la suite. Attention le format de ce nom est important. Une erreur ici n'est pas particulièrement désastreuse, mais met en péril la qualité du paquet source. La syntaxe est *paquet\_version.orig.tar.gz* (notez le « \_ » à la place du « - » d'origine). Il faut donc exécuter dans le terminal :

mv epdfview-0.1.5.tar.gz epdfview\_0.1.5.orig.tar.gz

Notez le nom similaire au fichier correspondant que nous avons trouvé dans le chapitre précédent.

Puis on décompresse cette archive et on entre dans le répertoire contenant les sources :

tar zxvf epdfview\_0.1.5.orig.tar.gz

cd epdfview-0.1.5

Une règle d'or dans la création de paquets est de **ne pas toucher au tarball d'origine**. La seule chose autorisée est de le renommer (c'est même nécessaire). Ne pas modifier le tarball permet de bien distinguer le travail de l'auteur de celui du packageur (empaqueteur). Maintenant que les sources sont là, ainsi que le « .orig.tar.gz », on va pouvoir créer le paquet. Pour cela il faut créer le dossier « debian/ » dans le dossier des sources (je vous rappelle que c'est ce dossier « debian » qui va donner les différents fichiers contenus dans notre paquet source (cf. [chap1](about:reader?url=https%3A%2F%2Fdoc.ubuntu-fr.org%2Ftutoriel%2Fcreer_un_paquet#etude_du_contenu_d_un_paquet_debian_source))). Pour créer un squelette de dossier « debian/ », utilisez dh\_make installé disponible dans le paquet dh-make (cf. [pré-requis](about:reader?url=https%3A%2F%2Fdoc.ubuntu-fr.org%2Ftutoriel%2Fcreer_un_paquet#pre-requis)).

La commande :

dh\_make -e votreaddresse@email.tld

doit normalement vous retourner ceci :

Type of package: single binary, multiple binary, library, kernel module or cdbs?

[s/m/l/k/b]

Ceci vous permet de préciser quel type de package vous construisez. Ici, choisissez le type de paquet "single" : répondez donc \*s\* à la question.

Pour les curieux, quelques commentaires sur les autres choix :

Attention, il faut saisir cette commande dans le dossier contenant les sources. Dans notre cas :

$ ~/packaging/epdfview/epdfview-0.1.5

Hit <enter> to confirm :

On confirme avec la touche « Entrée ».

On doit maintenant avoir un dossier « debian/ » dans le dossier contenant les sources de notre application.

On entre dans ce dossier « debian/ » :

cd debian && ls

Et là, horreur, plein de fichiers à éditer ! Certains éléments de ce squelette ne nous sont pas utiles ici. On va supprimer le fichier « README » et les fichiers « \*.ex » et « \*.EX » (qui sont des fichiers exemples) avec la commande suivante :

rm -rf \*ex \*EX README\* && ls

Il ne nous reste plus que 7 fichiers :

Le fichier « docs » n'est pas indispensable mais les 6 autres sont absolument essentiels et il va falloir les éditer avec votre éditeur de texte préféré.

**Le fichier « changelog »**

Il contient le détail de toute l'évolution du paquet. Pour chaque entrée on a :

Attention à la syntaxe, les espaces et ponctuations ne sont pas anodins. La compilation du paquet source étant très automatisée, les scripts doivent avoir des points de repère solides.

Quelques mots sur les versions de paquets… Une même version d'un logiciel peut être empaquetée plusieurs fois. De même, Ubuntu étant basée sur Debian, il peut exister plusieurs modifications faites dans Ubuntu à partir du même paquet venant de chez Debian. Les versions sont donc de la forme \*<upstream\_version>-<debian\_version>ubuntu<ubuntu\_version>\*. Ce qui donne pour nous (ePDFView n'existe pas chez Debian, donc debian\_version = 0) : *0.1.5-0ubuntu1*.

Debian est « upstream » pour Ubuntu (« flux venant d'en haut »), de même que les sources du logiciel sont « upstream » pour Debian et Ubuntu.

Il faut donc mettre à jour ce fichier avec :

Le « Closes: #nnnn » n'est pas utilisé chez Ubuntu (ça permet de clore automatiquement un bug ITP chez Debian). On peut le supprimer ici.

Les dates et heures sont à renseigner au format UTC + décalage horaire.

Pour vérifier, voici un exemple de [fichier changelog](http://gauvain.pocentek.net/u-classroom/2006-08-31/debian/changelog).

Il est notamment possible d'éditer le fichier changelog avec la commande **dch**, provenant du paquet devscripts (debchange - Outil pour la maintenance du fichier debian/changelog d’un paquet source).

**Le fichier « copyright »**

Le squelette de ce fichier est plutôt clair, pas vraiment besoin de s'étendre dessus. Mais le contenu n'est pas toujours évident à déterminer. Il y a un fichier « COPYING » dans le répertoire source, qui contient dans notre exemple la GPL en entier. Notez que ce n'est pas ce fichier qui sert de référence pour définir la licence, mais les fichiers source (qui contiennent le code, souvent présents dans un dossier « src/ »).

Regardez par exemple le fichier « DocumentRectangle.cxx » dans les sources :

less ../src/DocumentRectangle.cxx

Les quatres premiers paragraphes sont ceux à recopier bêtement (et oui) dans « debian/copyright ». Ce sont ces paragraphes qui désignent le nom et l'année du Copyright, ainsi que la licence (pas le fichier « COPYING »). Attention, vérifiez bien tous les fichiers sources ! Plusieurs copyrights ou licences peuvent être utilisés ! Dans notre cas, le fichier « debian/copyright » ressemblera à : <http://gauvain.pocentek.net/u-classroom/2006-08-31/debian/copyright>

Il est important de faire attention à ce fichier car s'il est mal renseigné, le détenteur du copyright peut très bien aller jusqu'à vous intenter un procès (la majorité se contenteront d'un mail vous signalant l'erreur et, si vous êtes courtois, n'hésiterons pas à vous aider). Mais soyez prudent(e) malgré tout, certains deviennent susceptibles dès que l'on écorne leur copyright.

D'ailleurs, lors de la réalisation d'un package n'hésitez pas à contacter l'auteur. C'est en général assez apprécié et cela peut vous permettre de faire valider ce fichier directement auprès de la personne concernée.

**Le fichier « control »**

C'est le fichier de description du paquet source et de son paquet binaire résultant.

Le premier paragraphe décrit le paquet source. Le(s) paragraphe(s) suivant(s) décrit(décrivent) le(s) paquet(s) binaire(s), car il peut y avoir plusieurs paquets binaires générés pour un seul paquet source. Les champs présents dans ce fichier squelette **doivent** être présents.

Jetons d'abord un coup d'oeil aux champs concernant le paquet source :

La debian policy peut faire peur… mais c'est la référence pour l'empaquetage debian/ubuntu.

Jetons ensuite un coup d'oeil aux champs concernant le paquet binaire :

Notez que l'on peut indiquer ici les paquets supplémentaires suggérés par l'application à l'aide d'une ligne "Suggests: foo1, foo2, foo3" ou "Recommends: bar1, bar2, bar3".

Voici le fichier « control » de notre exemple : <http://gauvain.pocentek.net/u-classroom/2006-08-31/debian/control>

La difficulté principale de cette étape est d'indiquer correctement les Build-Depends. Elles peuvent être déterminées en fouillant les sources ou en compilant à la main l'application (grâce aux messages d'erreurs du ./configure notamment). Elles sont quelquefois indiquées sur le site du projet. Attention lors des tests sur votre machine. Il se peut qu'une dépendance nécessaire soit déjà installée et que la compilation fonctionne (à priori) parfaitement. Cependant tous les paquets ubuntu sont compilés dans un environnement vierge. Si la ligne "Build-Depends:" n'est pas exacte, la compilation risque alors d'échouer (pbuilder est un outil indispensable pour tester les Build Depends, nous en reparlerons plus tard). Vous pouvez obtenir la liste des dépendances de epdview en exécutant cette commande:

apt-cache show epdfview

**Le paquet « debhelper »**

Dans la ligne Build-Depends figure la mention d'un paquet particulier : le paquet debhelper.

debhelper est un outil qui facilite la vie des empaqueteurs, avec tout un tas de petits scripts dont les noms commencent par 'dh\_'. Chacun a un rôle particulier et fonctionne différemment. La commande `man dh\_truc` vous apprendra tout ce qu'il faut savoir ;). On verra quelques exemples en regardant le fichier debian/rules. La version de référence pour Ubuntu Dapper et Edgy est la version 5. La "compatibilité debhelper" (cf paragraphe suivant) correspond à la version majeure de debhelper (c'est donc 5). `dh\_make` utilise par défaut une compatibilité debhelper 4 sous dapper. Veillez à mettre à jour les informations (bien que ce ne soit à priori pas vital pour la compilation de votre paquet).

**Le fichier « compat »**

Ce fichier sert à indiquer la compatibilité debhelper.

**Le fichier « rules »**

Le fichier « rules » est un fichier exécutable. C'est lui qui va servir à piloter la compilation lors de la création du paquet binaire.

C'est ce fichier qui contient les règles nécessaires à la compilation du paquet. Le principe est simple. Il s'agit tout simplement de compiler le logiciel exactement comme on compile avec `./configure && make`(c'est la règle "build"), et de l'installer (c'est la règle "install") non pas sur le système, mais dans un dossier ("$(CURDIR)/debian/<paquet>", avec `make install`), avant de pouvoir en faire une archive .deb (c'est la règle "binary-install").

Le fichier rules est un makefile dont chaque cible est appelée lors de la construction du paquet. Chaque partie correspond en fait à une certaine étape de la compilation :

Ici l'installation se fait dans "$(CURDIR)/debian/epdfview", donc dans le dossier « debian/ » créé tout à l'heure. Le fichier « rules » que vous avez n'a pas à être modifié pour notre exemple. En général il n'est pas nécessaire de le modifier, c'est uniquement dans le cas où la compilation classique (./configure && make && make install) échoue que cela devient nécessaire (dans le cas d'un logiciel utilisant cette méthode de compilation, bien sûr). Par contre si l'on veut ajouter des éléments particuliers à l'installation (script shell supplémentaire ou page man par exemple), c'est dans ce fichier qu'on le précise. On va d'ailleurs ajouter une page man à notre exemple :

wget http://gauvain.pocentek.net/u-classroom/2006-08-31/debian/epdfview.1

Cette page est à installer dans « ~/packaging/epdvfiew/epdfview-0.1.5/debian ». Si vous ajoutez un fichier dans les sources c'est **toujours et uniquement** dans le dossier **debian**. Les sources de l'auteur ne doivent en aucun cas être directement modifiées (vous pouvez utiliser un système de patch tel que dpatch ou simple-patchsys, ce que nous n'aborderons pas ici).

Pour indiquer que l'on veut ajouter cette page man on va utiliser l'outil dh\_installman. Cela va se faire via le fichier debian/rules. Cherchez dans votre fichier rules la ligne qui contient *dh\_installman* (normalement ligne 92 ou 93). On va préciser quelle page man on veut installer en la changeant en :

dh\_installman debian/epdfview.1

(vous aurez remarqué que dh\_installman fait partie des outils du paquet debhelper).

Pour avoir des informations sur tous les outils dh\_\* que vous voyez listés dans ce fichier n'hésitez pas à consulter les pages man. Elles ont toutes été traduites en français (et en espagnol). Ces traductions font partie intégrante de debhelper et s'installent donc automatiquement. Si votre "locale" est «\*.fr» alors « man dh\_\* » affiche directement la version française, sinon il faut le spécifier avec « man -L fr dh\_\* ».

Une fois arrivé ici notre paquet source est prêt, il ne nous reste plus qu'à le créer vraiment pour pouvoir retrouver les trois fichiers que nous avons vus plus tôt, dans notre premier exemple.

**Le fichier « dirs »**

Le fichier « dirs » permet de créer des répertoires qui ne sont pas créés automatiquement à l'installation du programme.

**Construction des fichiers de compilation**

Pour construire ce paquet :

cd .. # on se place à la racine du dossier source : ~/packaging/epdfview/epdfview-0.1.5

debuild -S -sa --lintian-opts -i

Le '-S' permet de construire un paquet source, le '–lintian-opts -i' donne une information sur les messages d'erreur de lintian, le '-sa' permet d'inclure le .orig.tar.gz dans l'upload vers une archive (à retenir lorsque vous uploaderez votre premier paquet sur [REVU](http://revu.tauware.de/), le système de QA d'Ubuntu, permettant aux contributeurs de faire rentrer des paquets et surtout, comme son nom l'indique, de les faire vérifier par des MOTUs). La commande utilise automatiquement fakeroot (commande simulant les privilèges super-utilisateur) si rien est précisé (voir le man de debuild).

*ATTENTION* : le paquet source est à reconstruire avec cette commande **à chaque fois que vous modifiez quelque chose** dans le dossier sources.

À ce stade, vous risquez d'être confronté à l'erreur suivante :

debuild: fatal error at line 791:

running debsign failed

Pas de panique contrairement, à ce qui est indiqué ce n'est pas une erreur si fatale que cela. Elle signifie simplement que debuild n'a pas pu signer le paquet avec une clef gpg.  
Pour créer une clef gpg voir le lien suivant : [gerer\_les\_clefs\_avec\_gpg](https://doc.ubuntu-fr.org/gnupg#gerer_les_clefs_avec_gpg)  
Si vous avez une clef gpg associée à l'adresse mail que vous avez indiqué au début, vous pouvez utiliser **-k0x12345678** comme paramètre supplémentaire de debuild où 12345678 est remplacé par l’identifiant de 8 caractères de votre clé gpg **publique**.

Pour obtenir cette clef, il vous suffit de la chercher.

gpg --list-keys une\_partie\_de\_son\_uid

(la dernière option n'est pas obligatoire et permet simplement un tri)

Cela renvoie :

/home/utilisateur/.gnupg/pubring.gpg

----------------------------------

pub 2048R/clé\_publique\_de\_8\_caractères 2007-06-08

uid partie\_de\_son\_uid (commentaire) <adresse@nom\_de\_domaine.fr>

sub 2048/clé\_privé\_de\_8\_caractères 2007-06-08

Votre clef hexadécimale de 8 caractères récupérée , il vous suffit de rajouter un 0x devant afin de l'exploiter dans debuild sous forme :

debuild -k0x12345678 -S -sa --lintian-opts -i

Une autre solution est d'ajouter dans votre fichier **.devscripts** (dans votre $HOME : le créer si besoin) export DEBSIGN\_KEYID=<clé\_publique\_de\_8\_caractères> pour qu'elle soit prise en compte automatiquement. Pour une prise en compte immédiate, relancez le shell, ou bien entrez la commande :

source ~/.devscripts

*Note :* si votre clef nécessite une phrase de passe, que debuild ne vous le demande et que le message d'erreur suivant apparait :

gpg: [stdin]: clearsign failed: mauvaise phrase de passe

debsign: gpg error occurred! Aborting...

Taper les lignes suivantes de commande suivante et lancer debuild à nouveau :

export GPG\_TTY=$(tty)

Pour voir le résultat :

cd .. && ls

Et, ô miracle ! Nous avons bien nos trois fichiers !

**Compilation et création du paquet binaire**

**Préparation de pbuilder**

On va utiliser l'outil [pbuilder](https://doc.ubuntu-fr.org/pbuilder), qui crée un chroot d'un système Ubuntu de base, puis le compresse dans un fichier base.tgz.  
Lors de l'utilisation de pbuilder pour construire le paquet binaire, le chroot va être décompressé, les Build-Deps installées, et le paquet compilé. S'il manque une Build-Dep, la compilation échouera.  
Le principal intérêt de pbuilder est de vérifier que les Build Deps sont bonnes (cf. notre fichier control).

Pour supporter les dépôts Universe et Multiverse

echo "COMPONENTS=\"main restricted universe multiverse\"" | sudo tee -a /etc/pbuilderrc

sudo pbuilder update

Pour initialiser pbuilder, tapez dans votre terminal :

sudo pbuilder create

La première ligne permet d'activer l'utilisation des dépôts universe et multiverse sous dapper (désactivés par défaut). La seconde crée le chroot et le compresse.

Pour faire un chroot d'une autre version d'ubuntu que celle que vous utilisez actuellement (pour créer un paquet compatible), vous pouvez ajouter à la fin de la deuxième commande –distribution feisty:

sudo pbuilder create --distribution feisty

(feisty peut être remplacé par le nom de code d'une autre version d'Ubuntu).

Ces commandes peuvent être lancées en début de travail dans un autre terminal, car cette étape peut prendre un certain temps. N'étant utilisé qu'à la fin vous pouvez très bien commencer à préparer vos paquets sources pendant ce temps.

**Création du paquet binaire**

Une fois l'étape précédente terminée il ne vous reste plus qu'à taper la commande suivante pour compiler votre paquet source (dans « ~/packaging/epdfview ») :

sudo pbuilder build \*.dsc

Et à laisser mariner… en cas d'erreur de dépendance (eh oui cela peut encore arriver pendant la compilation) reprendre à l'étape **Création du paquet binaire** en ajoutant le paquet manquant avant lancer le *build*, il est possible d'ajouter plusieurs paquets en les séparant avec un espace :

sudo pbuilder update --extrapackages nomdupaquetmanquant (ex: libqt3-dev)

Une fois cette étape terminée, votre .deb est dans **/var/cache/pbuilder/result**

En dernière étape, nous allons vérifier quelques points :

Cet avertissement de lintian :

W: epdfview: old-fsf-address-in-copyright-file

n'est pas un bloqueur.

Linda fonctionne de la même manière que lintian.

Il faut remplacer l'adresse de la FSF (qui a changé depuis juillet 2005) dans le fichier copyright.  
L'ancienne adresse était : 59 Temple Place - Suite 330, Boston, MA 02111-1307 USA qu'il faut remplacer par : 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Vous avez donc maintenant votre paquet debian compilé et prêt à être diffusé (à vous de trouver le moyen de le faire, le plus simple étant de vous joindre à une équipe gérant déjà un dépôt existant. Cela vous permettra d'être en contact avec des gens ayant déjà de l'expérience qui pourront vous en faire profiter).

**Est-ce que je dois passer en root pour créer mon paquet ?**

Ce n'est pas nécessaire, et absolument pas recommandé. Seule l'installation d'un .deb et la compilation avec [pbuilder](https://doc.ubuntu-fr.org/pbuilder) nécessitent d'être super utilisateur.

`sudo pbuilder login` vous amènera dans l'environnement chrooté, donc sans paquets installés. Utilisez le `./configure && make` ici pour savoir quels paquets sont nécessaires à la compilation.

**Après avoir exécuté `debuild -S -sa`, je n'ai pas de .diff.gz !**

C'est que la syntaxe de votre .orig.tar.gz n'est pas correcte. Vérifiez bien qu'il est de la forme : nomDuPaquet**\_**x.y.z.orig.tar.gz

Notez le '\_' (pas de '-'), qui est souvent source d'erreur.

Lors de l'exécution de `debuild -S -sa`, vous pouvez vérifier qu'il a bien créé le .diff.gz et tenu compte de votre .orig.tar.gz en cherchant les lignes suivantes :

dpkg-source: building foo using existing foo\_x.y.z.orig.tar.gz

dpkg-source: building foo in foo\_x.y.z-0ubuntu1.diff.gz

**Le programme que je veux empaqueter est fourni dans un .tar.bz2. Qu'est-ce que j'en fait ?**

Il faut absolument avoir une archive compressée avec gzip. Décompressez ce que vous avez téléchargé avec :

bunzip2 foo-x.y.z.tar.bz2

puis recompressez avec gzip (compression maximale) :

gzip -9 foo-x.y.z.tar

Renommez enfin votre archive pour avoir un .orig.tar.gz correct.

Cela nécessite deux étapes :

L'intérêt de ce procédé est de n'avoir qu'un seul paquet -common sur les dépôts, utilisable sur toutes les architectures (des images, fichiers .xml, etc).

**Quand on installe un paquet qui va dans « /usr/include », il y reste ?**

Oui, jusqu'au moment où on l'enlève. On peut enlever après la compilation.

FIXME

Une des solutions pour tester est d'utiliser un système virtualisé (à l'aide de [VirtualBox](https://doc.ubuntu-fr.org/virtualbox) par exemple). En utilisant correctement les *snapshots*, on peut facilement faire un rollback du système virtualisé avant l'installation du paquet.

Une autre solution est d'utiliser Docker.