Compiler un kernel pour debian

- Objet : Compiler d'une source externe.
- Commentaires : *Tout est dans le titre* 🙂
- Créé, Rédigé et Testé par naguam
- Discussion liée à ce tutoriel : Lien vers le forum concernant ce tuto [1)]
- Niveau requis : Tout niveau

1 Notes avant de commencer

- Ce tutoriel est destiné à toute personne, qui si elle le veut, devrait pouvoir presque tout faire au copier-collé excepté certaines choses comme `le passage en root` ou `le repassage en user` que nous considérons comme acquis, ou encore réfléchir un petit peu à la partie sécurité
- Les plus expérimentés ou ceux qui veulent aller plus loin, trouveront en bas de la page des liens pour une exploitation plus poussée de la technique (par exemple pour patcher.).
- Dans ce tutoriel, l'explication sera faite d'une manière `Nous allons faire etc` et `vous`, comme si c'était une personne réelle qui vous assistait pour que vous enregistriez au mieux les informations basiques qui seront expliquées au plus simple pour que vous n'ayez ensuite plus besoin du tutoriel pour le faire.
- Nous allons aborder la partie sécurité au plus simple, mais elle mérite une attention certaine.
- Si vous êtes sur ce tutoriel, c'est que vous voulez compiler un kernel : nous n'allons pas répéter les raisons de le faire, mais sachez que ce tutoriel est à appliquer à vos risques et périls ! Votre responsabilité.
- Lisez le tutoriel **dans sa totalité** et respectez bien les `codes en` **`user`** ou **`root`** car il est préférable d'utiliser le root le moins possible : normalement le root n'est nécessaire que pour l'installation des dépendances et l'installation du kernel à la fin.
- **ATTENTION LE FORMAT DES SOURCES COMPRESSÉES PEUVENT CHANGER SUR LES SITES DE TÉLÉCHARGEMENT,** si c'est le cas changez les `.xz` par `.gz` ou autres extentions, et si ce ne sont pas des archives TAR, tapez les commandes de décompression adaptées au format.

2 C'est parti! Prérequis!

2.1 Les Dépendances

Alors déjà, soyez sûr avant de commencer que vous avez suffisamment d'espace libre pour les dépendances mais aussi pour la compilation qui créera des paquets (10GB ou plus conseillés).

Il nous faudra donc installer des dépendances nécessaires à la compilation (en root)

```
apt install build-essential fakeroot dpkg-dev perl libssl-dev bc gnupg
dirmngr libncurses-dev libelf-dev flex bison lsb-release rsync dwarves
```

dwarves est nécessaire avec la configuration debian à partir de debian bullseye, et uniquement si vous laissez la partie debug activée dans la suite du tutoriel.

Si vous avez les dépôts src dans votre sources.list et que, n'avoir que le minimum de dépendances ne vous importe pas, vous pouvez également faire (en root)

```
apt build-dep linux
```

Cela installera les dépendances exactes du kernel packagé dans debian; dépendances choisies et utilisées par les mainteneurs.

Normalement ça s'accorde assez bien avec une configuration faite avec `make olddefconfig`, que nous verrons plus tard dans le tutoriel.

2.2 Répertoire de compilation

Nous allons créer un répertoire de compilation, c'est très fortement conseillé, puis nous allons aller dedans (en user!!)

```
mkdir ~/compilation && cd ~/compilation
```

2.3 Les sources

Bien sûr, il nous faut télécharger :

- les sources du kernel *(lien “tarball” sur kernel.org)* et
- la signature pour vérification de l'intégrité du kernel *(lien “pgp” sur kernel.org)* .

Nous pouvons par exemple, les télécharger chez kernel.org, le site officiel du kernel linux. (conseillé pour les moins expérimentés)
Vous devez avoir deux fichiers qui se terminent respectivement par `.tar.xz` (ou`.tar.gz`) et `.tar.sign`

Téléchargez vos sources dans votre répertoire de compilation créé juste avant!

Par simplicité dans les commandes suivantes, nous allons taper une petite commande dans un terminal (cette fois en user)
Attention! les X sont à remplacer par la version du kernel que vous avez téléchargé sur `kernel.org`

Si vous utilisez d'autres sources que le kernel de kernel.org, vous devez passer cette étape, mais dans des commandes futures, **vous devrez remplacer** `linux-$kversion.tar.xz`, `linux-$kversion.tar.sign` et `linux-$kversion` par les équivalent que vous avez téléchargés auparavant.

```
kversion=X.X.X
```

2.4 Vérification de l'intégrité du kernel /!\ Important pour la sécurité!

Maintenant nous allons passer à la vérification des sources, pour prouver l'intégrité des fichiers. C'est une étape assez fastidieuse mais importante car si il y a un problème de mauvaise signature **c'est mauvais signe**, pour les moins expérimentés, en cas de problème vous pouvez faire un post sur la page du forum reliée à ce tutoriel (lien en haut de page).

Cette procédure est bien expliquée sur le site officiel du noyau linux donc je conseille de la suivre en adaptant les commandes avec votre version du noyau.

Lien explicatif

Vous pouvez utiliser un outil de traduction pour le lire en français si l'anglais est compliqué pour vous. Pour plus d'aide, n'hésitez pas à demander sur la page du forum reliée à ce tutoriel (lien en haut de page).

3 Décompression et préparation des sources pour la compilation du kernel!

3.1 Décompression

Nous avons fini les prérequis, nous allons maintenant préparer nos sources pour la compilation. Nous allons commencer par décompresser nos sources (en user) puis nous allons aller dans nos sources décompressées pour l'étape suivante.

```
tar -xaf linux-$kversion.tar.xz && cd linux-$kversion
```

N'oubliez pas ce que nous avons déjà dit avant en 2.3 pour l'histoire du **kversion** !

3.2 Configuration et préparation des sources

Maintenant, nous allons configurer notre kernel avec nos options voulues (ici comme presque toujours, les commandes seront appliquées en user).

Pour les moins expérimentés ou ceux qui n'ont pas envie de se casser la tête vous pouvez faire la commande suivante :

```
make olddefconfig
```

Cette commande va chercher la configuration de votre kernel actuel et met toutes les nouvelles options en “par default”.

Pour les plus expérimentés, ou les personnes bidouilleuse, ou même encore les personnes qui ont besoin d'options activées en particulier, vous pouvez faire cette commande :

```
make menuconfig
```

Pour plus d'infos sur les options de configuration, les informations seront en bas de page.

Ensuite nous devons désactiver les clefs du kernel car sinon nous aurons droit à une belle erreur de compilation

En effet le kernel debian est signé avec les clefs debian. Reprendre leur configuration c'est tenter de réutiliser leurs clefs, sauf que comme c'est pas le même kernel avec les même sources, ça va poser problème.

Nous voulons donc générer de nouvelles clefs. La commande suivante permet donc de désactiver les actuelles, et au début de la compilation, deux validations (entrée pour une valeur par default) vous seront demandé et les nouvelles clef générées.

```
./scripts/config -d CONFIG_MODULE_SIG_ALL -d CONFIG_MODULE_SIG_KEY -d
CONFIG_SYSTEM_TRUSTED_KEYS
```

Maintenant **si vous êtes bourrin et que vous voulez tout créer passez en étape 4.1**, sinon pour optimiser le temps de compilation continuez de lire :)

Nous allons enlever le paquet debug dbg car il ne sert en général qu'aux développeurs, cela nous permettra de compiler en moins de temps

Pour ce faire nous allons faire la commande suivante :

```
./scripts/config -d CONFIG_DEBUG_INFO
```

Voilà, nous pouvons maintenant passer à la fameuse compilation. D'autres options pour optimiser le temps de compilation seront indiquées.
Vous pouvez passer en 4.2. Nous avons commencé à optimiser le temps, nous allons continuer 😀

4 C'est parti pour la compilation! La fameuse! (toujours en user et oui!)

Il est fort possible que vous ayez des réponses à donner au moment de la compilation, tapez **entrée**

Maintenant, **fakeroot doit être installé mais se fait automatiquement** (testé 4.16 et 4.17-rc1) donc si en compilant **une version ancienne**, vous avez une erreur, ajoutez fakeroot avant la commande de compilation

4.1 Pour la compilation bourin (indiquée soulignée et gras dans la partie

précédente)

Nous pouvons donc nous mettre à compiler :

```
make deb-pkg -j"$(nproc)" LOCALVERSION=-"$(dpkg --print-architecture)"
KDEB_PKGVERSION="$(make kernelversion)-1"
```

Maintenant, suivant la puissance de votre processeur, cela va prendre un peu de temps (10 min si vous avez un processeur très puissant genre intel core i7 overclocké ou amd ryzen, et une semaine pour un pIImmx, en tout cas dans la plupart des cas des machines actuelles, c'est environ entre 1 et 4h de compilation). Vous avez donc le temps de boire un thé ou un café, manger des Chocos-DF et d'écouter du hard-rock!!

Ensuite, passez à **l'étape 4.3** si vous voulez des explications simples pour juste certaines parties de la commande
Sinon passez directement à **l'étape 5**

4.2 Toujours plus d'optimisation de temps en ne créant que les paquets nécessaires

Nous allons créer les paquets avec `bindeb-pkg`, cela ne va créer que les binaires les plus utiles (certains indispensables) si vous voulez les nouvelles sources reconstruites, il vous faut faire **l'étape 4.1**

Nous pouvons nous mettre à compiler

```
make bindeb-pkg -j"$(nproc)" LOCALVERSION=-"$(dpkg --print-architecture)"
KDEB_PKGVERSION="$(make kernelversion)-1"
```

Maintenant, suivant la puissance de votre processeur, cela va prendre un peu de temps (10 min si vous avez un processeur très puissant genre intel core i7 overclocké ou amd ryzen, et une semaine pour un pIImmx, en tout cas dans la plupart des cas des machines actuelles, c'est environ entre 1 et 4h de compilation). Vous avez donc le temps de boire un thé ou un café, manger des Chocos-DF et d'écouter du hard-rock!!

Ensuite, passez à **l'étape suivante (4.3)** si vous voulez des explications simples pour juste certaines parties de la commande
Sinon passez directement à **l'étape 5**

4.3 Explications basiques

- **`-j"$(nproc)"`** prend le nombre de threads total de votre processeur (la compilation sera la plus rapide possible), pour les connaisseurs, vous pouvez mettre **`-jX`** en remplaçant X par le nombre de cores que vous voulez allouer dans la limite du nombre de threads maximums, **c'est**

optionnel sans cet argument, cela n'utilisera qu'un seul thread

- **LOCALVERSION=-"$(dpkg –print-architecture)"** cela donne le nom custom de votre architecture, vous pouvez aussi mettre un nom custom sans majuscules derrière **LOCALVERSION=-**, c'est **un argument optionnel**
- **KDEB_PKGVERSION="$(make kernelversion)-1"** cet argument sert à versionner le kernel **1** (soit version 1) **c'est encore optionnel mais il est conseillé**

5 Installation ou désinstallation des paquets

Tout d'abord nous devons aller dans le répertoire précédent

```
cd ..
```

Ensuite, pour installer le kernel, nous devons au minimum installer l'image et les headers **mais je conseille d'installer tous les .deb créés si vous ne manquez pas d'espace de stockage.** (commandes suivantes faites en root)

```
dpkg -i *.deb
```

Pour désinstaller:

```
dpkg -P *.deb
```

/!\ À bien être dans le bon répertoire /!

Vous pouvez aussi désinstaller avec apt.

Moi, sur la plupart des machines sur lesquelles j'ai appliqué le tutoriel, j'avais ensuite un message au boot de `pcspkr is already registered`. Du coup, j'ai dû le blacklister :

```
echo "blacklist pcspkr" > /etc/modprobe.d/blacklist-pcspkr.conf
```

(de plus pcspkr est un module kernel qui sert seulement au beep système en tty par exemple quand on fait retour trop loin pour supprimer du texte)

Avertissement

Dans `/etc/kernel/postinst.d/apt-auto-removal` :

```
# Mark as not-for-autoremoval those kernel packages that are:
#
# - the currently booted version
# - the kernel version we've been called for
# - the latest kernel version (as determined by debian version number)
```

```
# - the second-latest kernel version
#
# In the common case this results in two kernels saved (booted into the
# second-latest kernel, we install the latest kernel in an upgrade), but
# can save up to four. Kernel refers here to a distinct release, which can
# potentially be installed in multiple flavours counting as one kernel.
```

Ce script n'est peut-être présent que sur Stretch, pas sur Jessie Fix Me!

Il faut que je teste un peu plus, mais généralement, il ne reste que 2 noyaux installés.
N'installez jamais plusieurs versions de noyau sans vérifier que l'avant-dernier fonctionne, sinon...

- désinstallez un noyau qui ne convient pas auparavant
- ou an*nul*ez ce script

Conclusion et Sources

Bravo! vous avez une machine avec le kernel de votre choix!

À voir aussi ce lien en section 8.10.4, Toute la page est intéressante pour tous les Fous-faciles!
La doc officielle de kernel.org
En particulier, les dépendances minimums pour ceux qui tenteraient sur n'importe quelle distribution
Et enfin pour patcher
Doc officielle debian

1)
N'hésitez pas à y faire part de vos remarques, succès, améliorations ou échecs ! Tant qu'elles sont correctement justifiées (raisons valables) | Vous pouvez aussi y poser des questions!