Noyau linux

- Objet : composants, personnalisation, compilation, installation, gestion des modules
- Niveau requis :
  avisé
- Commentaires : *Contexte d'utilisation du sujet du tuto.*
- Débutant, à savoir : Utiliser GNU/Linux en ligne de commande, tout commence là !.

Introduction

Récupération des sources du noyau

Archives téléchargeables

Les sources du noyau sont archivées à cette adresse : The Linux Kernel Archives.

- **Trois sortes d'archives** :

1. stable [EOL] : versions qui ne sont plus supportées
2. longterm : support étalé dans le temps
3. stable : pour récupérer le noyau téléchargé [tar.xz]
4. mainline (...-rc) : version testée

- **Pour récupérer la version stable** :

```
cd /usr/src
```

Puis après avoir récupérer l'adresse du lien de l'archive “tar.xz” par exemple.

```
wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.16.1.tar.xz
```

```
tar -xvJf linux-3.16.1.tar.xz
```

```
ls
```

```
linux-3.16.1  linux-3.16.1.tar.xz
```

- **Le fichier “linux-3.16.1” est désarchivé et décompressé** :

```
ls linux-3.16.1
```

```
arch     Documentation    init     lib          README          sound
block    drivers    ipc     MAINTAINERS  REPORTING-BUGS  tools
COPYING  firmware   Kbuild     Makefile     samples          usr
CREDITS  fs     Kconfig  mm          scripts          virt
crypto   include    kernel     net          security
```

Remarques sur Décompression/Compression

- **Plusieurs programmes de compression**
  compression/décompression d'un seul fichier à la fois:
- **gunzip** ; pour le format gz
- **bzip2** ; pour le format bz2.
- **xz** : pour le format xz

xz –decompress nom-fichier.xz

tar archivage et compression

- **Archiver** :

Depuis le répertoire parent des fichiers ou dossiers à archiver.

```
tar -cf fichier.tar dossier-à-archiver/
```

ou

```
tar -cf fichier.tar fichier1 fichier2 ...
```

> -c pour archiver
> -f pour créer un fichier “filezilla.tar”

- **Pour désarchiver**:

Depuis le répertoire parent du fichier à désarchiver:

```
tar -xf fichier.tar
```

- **Compression et archivage** :

tar permet aussi de compresser :

> -cj : pour compresser du bz2
> -cz : pour compresser du gz
> -cJ : pour compresser du xz
> -cv[j[z][J]] : verbose
> -f : pour créer un fichier

```
tar -cJf fichier.tar.xz fichier1 fichier2
```

ou

```
tar -cf fichier.tar dossier/
```

- **Pour désarchiver et décompresser en même temps** :

```
tar -xvJf fichier.tar.xz
```

> -xvJf : pour décompresser du xz
> -xvjf : pour décompresser du bz2
> -xvzf : pour décompresser du gz

- **Exemple** :

```
touch essai1 essai2
```

Archivage et compression :

```
tar -cJf essai.tar.xz essai1 essai2
```

Désarchivage et décompression :

```
tar -xvJf essai.tar.xz
```

```
essai1
essai2
```

```
rm essai1 essai2
```

Structure du noyau

- **Le noyau linux :**

1. noyau monobloc (noyau monolithique) : un seul fichier binaire
2. noyau modulaire : modules (fonctionnalités dynamiques) qu'on charge ou qu'on décharge
3. en compilant le noyau on obtient un fichier principal et si l'on veut des modules
4. dans le dossier principal :
   on met les fonctionnalités critiques

- **différents types de monobloc** (dans /boot):

1. vmlinux : noyau non compressé, intermédiaire de compilation ; il n'est pas bootable.
2. vmlinuz : compressé avec fonctionnalité pour qu'il soit bootable (proposé par les distributions).
3. zImage : idem vmlinuz, permet compilation local mais obsolète (ancien format de compression, limité à 512 octets !)
4. BzImage : idem vmlinuz, ok pour compilation locale (bon algorithme de compression)
5. kernel : nom générique pour toutes les versions, compressées ou non (nommé ainsi sur les systèmes utilisant grub2)

Par exemple, sur mon système debian:

```
ls /boot/ | grep ".*linu*"
```

```
vmlinuz-3.2.0-4-amd64
```

Fichiers critiques des sources

1. `./drivers` : contient des dossiers pour les éléments matériels
2. `./fs` : contient des dossiers pour système de fichiers supportés
3. `./net` : contient les dossiers pour les différents supports réseaux

Lors de la préparation de la compilation on peut choisir les fonctionnalités que l'on souhaite, et si on les souhaite de façon modulaire ou dans le monobloc.
Voir plus bas

Documentation des sources

```
cd /usr/src/linux-source-3.2 && ls
```

```
arch     crypto        include  Kconfig      Makefile    REPORTING-BUGS
sound
block    Documentation    init     kernel       mm    samples        tools
COPYING  drivers    ipc     lib          net    scripts        usr
CREDITS  fs     Kbuild     MAINTAINERS  README    security    virt
```

1. README : instructions pour la compilation et information sur la documentation
2. `Documentation` : sur les fonctionnalités du kernel, très technique : dans cette documentation commencer par lire les fichiers “INDEX” qui sont des résumés de chaque fichier de “technique” de la doc.

Patch

Un patch permet de modifier un logiciel existant.
Cela permet corriger un noyau manuellement, mais aussi de “passer à une version supérieure”.
Par exemple on a une version du noyau 3.8 et on se sert du patch pour la faire passer à la version 3.9.
Il y a aussi des patch pour modifier le comportement du kernel par exemple rt (temps réel).

Patch correctif

Préparation d'une ancienne version à patcher

aller sur https://www.kernel.org/pub/ → “linux” puis v3.x → cliquer sur “Last modified”.

- **télécharger la version “linux-3.8.tar.xz”** :

```
cd /usr/src/ && wget
https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.8.tar.xz
```

- **La décompresser** :

```
tar -xvJf linux-3.8.tar.xz
```

```
linux-3.8          linux-3.8.tar.xz
```

Télécharger le dernier patch pour le noyau 3.8 :

Pour télécharger un patch : https://www.kernel.org/pub/
→ linux → kernel → les v... sont les différentes versions.
Par exemple, **v3.x** : il y a les “ChangeLogs”, les “kernels” (entiers, “linux...”), et les “patch”.

> Il existe le patch “patch-3.8.13.xz”

- Télécharger le patch :

```
cd /usr/src/ && wget
https://www.kernel.org/pub/linux/kernel/v3.0/patch-3.8.13.xz
```

- Décompresser le patch :

```
xz -k -d patch-3.8.13.xz
```

```
patch-3.8.13  patch-3.8.13.xz
```

man xz:
-z, –compress
-d, –decompress, –uncompress
-k, –keep (Don't delete the input files.)

ou **`unxz fichier.xz`**

Application d'un patch

- Se déplacer dans les sources “linux-3.8.1”

```
cd linux-3.8
```

- Vérifier la version de ce noyau:

```
make kernelversion
```

```
3.8.0
```

- Lancer le correctif :

```
patch -p1 < ../patch-3.8.13
```

- Et voilà !

```
make kernelversion
```

```
3.8.13
```

Application d'un patch :

- **`gzip -cd patch-version.gz | patch -p0`**

- **bzipp2 -dc patch-version.bz2 | patch -p0**
- **patch -p0 < version**
  quand la “source” à corriger et le “patch” sont dans le même dossier et qu'on lance la commande “patch” depuis ce dossier.

Normalement on devrait lancer “patch” depuis le dossier parent du dossier source, par exemple /usr/src/, ou /Téléchargement dans lesquels il y aurait le fichier “linux-3.8”.
Si on avait lancé “patch” depuis le dossier parent au dossier des sources (“linux-3.8”)

```
patch -p0 < ../patch-3.8.13
```

Si on lance patch depuis le dossier des sources (“linux-3.8”)

```
patch -p1 < ../patch-3.8.13
```

(../patch-3.8.13 car on n'a pas mis “patch-3.8.13” dans le dossier des sources, on l'a laisser dans le dossier “/urs/src”)

Patcher avec un patch de version supérieure :

S'il est possible à partir d'une version .0 (par exemple 3.8.0)[1] à un correctif éloigné (par exemple 3.8.13), pour une version déjà corrigée (par exemple 3.8.1) il faut passer par tous les intermédiaires (par exemple 3.8.2 ; 3.8.3 ; ... 3.8.13). Pour passer à une version supérieure (“upgrader” de 3.8 à 3.9), il faut par exemple appliquer le patch 3.9 à la version 3.8, mais appliquer un patch 3.9 à une version 3.8.4).

Par exemple, pour passer de la version 3.8 à la 3.9 avec le patch “patch-3.9.xz” :

```
cd /usr/src/
```

```
wget https://www.kernel.org/pub/linux/kernel/v3.x/patch-3.9.xz
```

```
unxz patch-3.9.xz
```

Décompresser “linux-3.8.tar.xz” (ce qui supprime l'ancienne “linux-3.8”, qui était une 3.13, du fait qu'on avait déjà appliqué le patch)

```
tar -xvJf linux-3.8.tar.xz
```

```
cd linux-3.8
```

```
make kernelversion
```

```
3.8.0
```

```
patch -p1 < ../patch-3.9
```

```
make kernelversion
```

```
3.9.0
```

Lancer un test avant d'appliquer un patch

```
patch -p1 --dry-run < fichier-du-patch
```

Par exemple :

```
cd linux-3.8 && patch -p1 --dry-run < ../patch-3.8.12
```

Il est demandé “Reversed (or previous applied) patch detected)! Assume -R? [n]“
Il faut taper **y** pour avancer jusqu'au bout. Ce message apparaît quand on applique un patch d'une version supérieure ou qu'on a déjà appliqué un patch, ou autre erreur. Cette version avait déjà été patchée. On supprime le dossier source “linux-3.8” et on décompresse “linux-3.8.tar.xz”.

On essaie à nouveau :

```
cd linux-3.8 && patch -p1 --dry-run < ../patch-3.8.12
```

C'est ok donc on lance la commande :

```
cd linux-3.8 && patch -p1 --dry-run < ../patch-3.8.12
```

Revenir à une version antérieure ("dé-patcher")

```
patch -p0 -R < ./patch-version-à-retirer/code>

Par exemple, à la version "linux-3.8",
<code root>cd linux-3.8
```

```
make kernelversion
```

```
3.8.0
```

- le patch-3.8.13[2)]

Pour passer de la version linux-3.8.13 à la version linux-3.8.0, on fait :

```
/usr/src/linux-3.8# patch -p1 -R < ../patch-3.8.13
```

Et voilà :

```
make kernelversion
```

```
3.8.0
```

Effectuer une sauvegarde avant d'appliquer un patch

Création d'une sauvegarde des sources à patcher

```
patch -p0 -B fichier-sauvegarde/ -b < ./patch-version
```

Par exemple, avant d'appliquer un patch,

```
cd linux-3.8 && mkdir save-linux-3.8
```

- Puis on applique le patch ainsi :

```
patch -p1 -B save-linux-3.8/ -b < ../patch-3.8.13
```

On vérifie la nouvelle version :

```
make kernelversion
```

```
3.8.13
```

Et on a une sauvegarde des sources de “linux-3.8.0”:

```
ls save-linux-3.8/
```

```
arch    Documentation  include    lib      net        sound
block   drivers        ipc    Makefile  scripts   tools
crypto  fs           kernel    mm      security  virt
```

Revenir à la version "linux-3.8.0" avec la sauvegarde :

- créer un dossier de sauvegarde des modifications :

```
diff -ur linux-version oldfiles/linux-version > fichier-de-récupération-du-patch
```

> “fichier-de-récupération-du-patch” est un fichier contenant une comparaison entre la sauvegarde de linux-3.8.0 et le dossier actuel (les sources patchée en linux-3.8.13).
> -r : indiquer des répertoires
> -u :

Par exemple :

```
cd /usr/src/
```

```
diff -ur . ./save-linux-3.8/ > patch-save
```

- Se servir de ce fichier de comparaison pour récupérer l'ancienne version du noyau :

```
patch -p0 < ./patch-save
```

Vérifier :

```
make kernelversion
```

```
3.8.0
```

Autre méthode pour créer une sauvegarde lors de l'application d'un patch

Il s'agit d'appliquer le patch en demandant la création d'un dossier “Makefile.orig” dans le dossier le nouveau dossier des sources patché.

Par exemple, application à la version linux-3.8.0

```
make kernelversion
```

```
3.8.0
```

du patch-3.8.13 :

```
patch -p1 -b < ../patch-3.8.13
```

Vérification de la nouvelle version du noyau:

```
 make kernelversion
```

```
3.8.13
```

Le fichier “Makefile.orig” a été créé dans le dossier des sources :

```
ls
```

```
arch           drivers     Kbuild       Makefile.orig   samples          usr
block          firmware  Kconfig      mm          save-linux-3.8  virt
COPYING        fs    kernel       net          scripts
CREDITS        include   lib          patch-save      security
crypto         init     MAINTAINERS  README          sound
Documentation  ipc   Makefile     REPORTING-BUGS  tools
```

Récupération de la version originale à partir de ce dossier "Makefile.orig"

Il faut utiliser un script shell.

```
for i in $(find linux-3.8 | grep "orig")
do
DOSSIER=$(echo $i | sed 's/\.orgig//')
mv -f $i $DOSSIER
done
```

Personnalisation du noyau

La commande make

Elle permet le nettoyage ; la configuration ; la compilation ; l'installation.

```
make [ -f makefile] [options ] ... [cibles]
```

La cible définit l'action.
Par exemple pour un nettoyage :

```
make mrproper
```

Choisir les composant en fonction du matériel

Pour connaître son matériel voir : lspci ; lsmod

Préparer son kernel

Avoir décompresser : “linux-3.16.1.xz”

```
cd /usr/src/linux-3.16.1/
```

- Le nettoyer :

```
make mrproper
```

```
make mrproper
```

> supprime tous, fichiers de configuration et fichiers générés, il ne conserve que la source. Il faut l'utiliser avant la compilation.

```
make clean
```

> Supprime les fichiers .o issus d'une éventuelle compilation précédente et d'autres fichiers inutiles, ne laisse que ceux nécessaires à la construction des modules.

```
make distclean
```

> Supprime les fichiers de sauvegarde des éditeurs, les patchs.

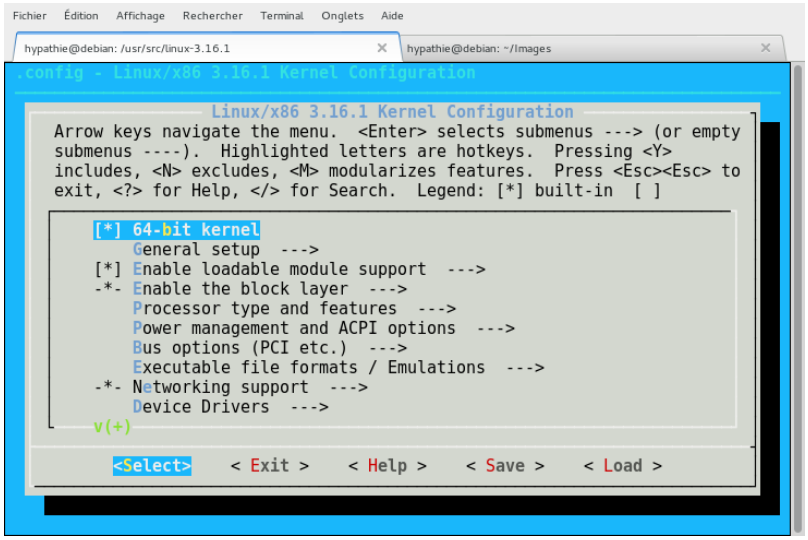
Configurer

La configuration permet de créer un fichier .config

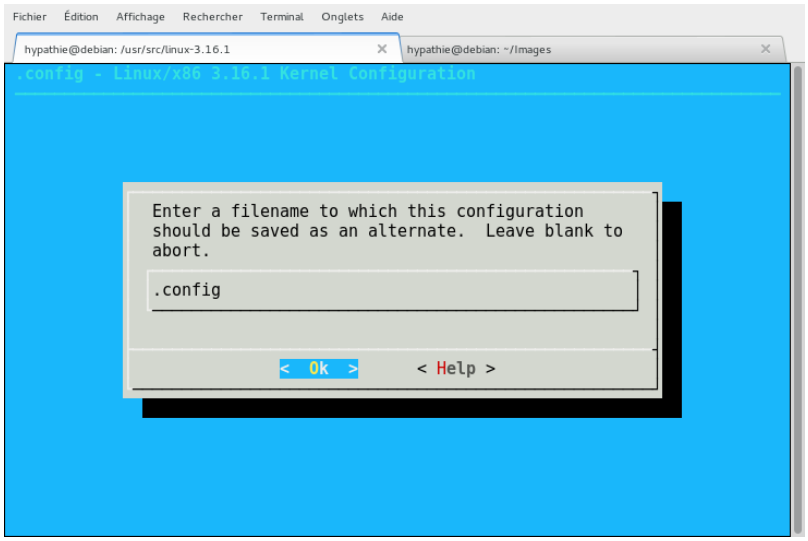
```
apt-get install ncurses-dev
```

Puis :

```
/usr/src/linux-3.16.1# make menuconfig
```



→ choisir **Save**



→ choisir **Ok**
→ puis “Configuration written to .config” : choisir “Exit” pour l'instant.
→ Retour au premier écran : “Exit”

```
ls -a
```

```
.    COPYING    firmware    ipc      .mailmap      README      sound
..   CREDITS    fs          Kbuild   MAINTAINERS  REPORTING-BUGS  tools
arch     crypto        .gitignore  Kconfig  Makefile      samples      usr
block    Documentation    include     kernel   mm          scripts      virt
.config  drivers    init        lib      net      security
```

→ On a un fichier **.config** : composé d'un ensemble de clés=valeurs.

La clé indique la fonctionnalité, et la valeur “y” pour activer la fonctionnalité; valeur “m” pour mettre la fonctionnalité en module.

Définir .config avec make config

On peut soit l'éditer, soit utiliser **`make config`**.
Avec make config, il demande fonctionnalité par fonctionnalité si on veut l'ajouter.
y pour l'ajouter
N pour ne pas l'activer
? pour avoir des infos sur la fonctionnalité.

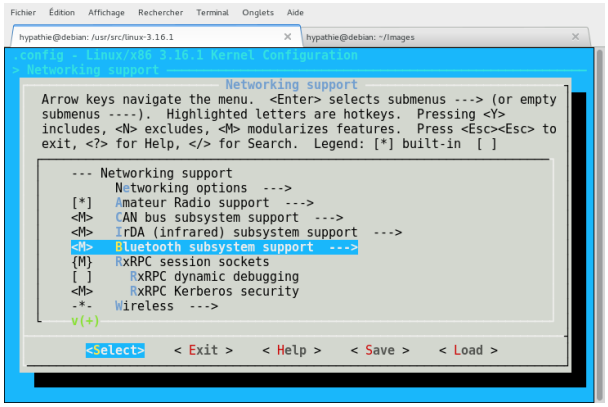
Pour sortir :

```
Stop
```

Définir avec menuconfig

```
make menuconfig
```

On sélectionne la catégorie qu'on souhaite modifier.
Par exemple on sélectionne “Network” puis <entrée>



On ajoute ce qu'on veut avec “shift+y” (ça met un “*” ou “shift+m” (ça met un “M”) ou “shift+n” (ça met un emplacement vide).

Ou **?** pour avoir la doc de la fonctionnalité.

On peut le faire avec make xconfig

Idem avec interface graphique.

Avec make gconfig

Pour les librairies GTK.

Compilation du noyau

Résumé :

1. vérifier les dépendances **make dep** (pour les noyau 2.2 et 2.4 seulement)
2. nettoyer (voir ci-dessus)
3. compiler le monobloc
4. compiler les modules
5. installer les modules

compiler le monobloc

Dans le fichier décompresser des sources qui a été nettoyé :

```
/usr/src/linux-3.16.1# make
```

ou

```
/usr/src/linux-3.16.1# make all
```

→ il construit le monobloc et les modules

afficher que les erreurs issues de la compilation

```
/usr/src/linux-3.16.1# make all | grep -iw "error"
```

C'est long ! Mais la compilation terminée, on a un “vmlinux” ; un monobloc qui peut être lancé par le bootloader.

Pour compiler le monobloc sans les modules

```
make [bzImage [zImage] ]
```

→ bzImage ou zImage selon le type de monobloc

Compiler les modules

Après avoir fait un **make bzImage**;
On fait un :

```
make modules
```

Créer un package (plutôt que de compiler le noyau)

- Pour CentOs **`make rpm`**
  une fois créé **`rpm fichier-créé-avec-make-rpm`**
- Pour Debian **`make-kpkg kernel_image`**

Avec debian

- Installer kpkg :

```
apt-get install kernel-package
```

Puis dans le dossier des sources, par exemple "/usr/src/linux-3.16.1"

- Créer le paquet :

```
make-kpkg kernel_image
```

Installer son noyau et ses modules

```
make install
```

Dans le fichier de ses sources (après `make all` ou `make`) → Cela copie le monobloc dans /boot
→ copie du fichier "system.map"
→ création et mise en place d'un initrd (si on ne l'a pas fait)
→ modifie grub

le fichier system.map

Il a été créé lors du `make all`. Il fait une table de correspondance entre nom symbolique et leur adresse en mémoire.

Il est bien de le copier :
(en root)

→ **`cp System.map /boot/system.map-xxx`**

→ **`rm /boot/System.map`**

→ **`ln -s /boot/System.map-version-de-son-kernel /bootSystem.map`**

Si on a voulu séparer les différentes opération et qu'on a pas fait de `make install` dans le fichier de source, on ajoute enfin les modules.

```
make modules_install
```

Cela installe tous les modules dans /lib/modules/n°kernel-version.

Opération indispensable avant de créer un initrd car pour le faire c'est dans /lib/modules/n°kernel-version que mkinitrd **ou** mkinitramfs **il trouve les infos nécessaires pour créer un initrd.**

INITial Ram Disk

Grub2 lance le kernel linux “vmlinux” mais peut avoir besoin de module pour lancer le système de fichiers.
Le INTial Ram Disk (ou **initrd** système de fichiers virtuel) se charge en mémoire vive (cramfs ou squashfs) qui sert de racine temporaire pour le kernel, qui peut alors aller chercher dans le système de fichiers du noyau ce dont il a besoin.

Mais si depuis la compilation le matos a changé, on ajoute les modules nécessaires dans un INITial Ram Disk qu'on crée pour cela, plutôt que de re-compiler un noyau.

Quelques outils pour créer un initrd :

- mkinitrd (distrib à base de Redhat)
- mkinitramfs (db)

mkinitrd

```
mkinitrd /boot/initrd-XXXX.img n°du-kernel-version
```

→ On donne le nom qu'on veut pour son initrd
→ Par exemple : `mkinitrd ./mon-initrd.img $(uname -r)`
→ `$(uname -r)` : pour le kernel actuel.
→ il se crée un initrd pour la version instalée de /boot : `vmlinuz-3.2.0-4-amd64`
→ Depuis /boot : ./mon-initrd.img 3.2.0-4-amd64
→ faire un uname -r
pour voir celle installée
→ Pour faire un initrd pour une autre, faire un make install avant (voir plus bas).

`-f` pour écraser l'ancien.
Mais mieux veut faire `mv initramfs-xxx.img ~/`

`–preload <module>` : définir un module qui sera chargé avant le module SCSI au démarrage (utile pour certains drivers du disque dur, ou certains système de fichier.

`image-version` : ajoute le nom du kernel au chemin de l'initrd ce qui permet d'avoir par exemple la création de initramfs-xxxx.img

`–with=<module>` : pour personnalisé, ajouter des modules après le SCSI

mkinitramfs

```
mkinittramfs -o /boot/initramfs-xxxx.img n°kernel-version
```

-o : pour indiquer le fichier de sortie
-d fichier de config : permet de détecter les drivers des systèmes de fichier dont il a besoin, par défaut c'est initramfs.
-k : ne pas supprimer le fichier temporaire qui a servi pendant la création.
-v : verbose
-r root : pour définir la partition racine (on peut définir ici ce qu'on a définit dans grub2)
–supported-host-version=version-spécifiée : avant de lancer mkinitramfs, vérifier si le programme peut créer une image pour la version spécifiée.
–supported-target-version=version-target : vérifier si une target est supportée

Quand on fait `mkinitramfs -o /boot/initrd.img n°version`, ce n° de version signifie “va chercher dans /lib/modules/n°version, créé lors de l'installation des modules avec `make modules_install`.

Par exemple :

```
mkintramfs -o mon-initrd.img $(uname -r)
```

Configuration de Grub

Legacy

Grub2

1)

qu'on trouve sous le nom de 3.9 sur le site “Index of/pub/kernel”

2)

`cd linux-3.8` puis `patch -p1 < ../patch-3.8.13` puis `make kernelversion` on a bien `3.8.13`