Construyendo un kernel linux a partir de su código fuente

Así es como construyo mis kernels 2.6
Lo mismo se aplicará casi completamente para los kernels 3.x en Slackware 14 y de ahí en adelante.

Comencemos, X y su

Los comandos se ejecutan desde una terminal X y, en algún punto, se inicia el configurador del kernel (basado en X).
El escritorio corre como “yo” (mi usuario) pero mis kernels se construyen como root. Para lograr lo anterior y hacer que root tenga acceso a mi servidor X, se hace lo siguiente (desde mi terminal): obtener privilegios de superusuario, unir mi propio archivo Xauthority con el de root, y asignar la variable DISPLAY. Después de hacer eso, se pueden ejecutar aplicaciones X desde la terminal “su”.

```
echo $DISPLAY                  #necesitará este valor 3 líneas más abajo
sudo -i                        #o “su -” en versiones más viejas de
Slackware
xauth merge ~alien/.Xauthority # use su propio nombre de usuario en vez de
“alien”
export DISPLAY=:0.0            # use el valor de DISPLAY obtenido 3 líneas
antes
```

Alternativamente puede usar el siguiente comando que dará el mismo resultado

```
sudo -s         # Un efecto secundario de '-s' es que permite a root
ejecutar programas basados en X
. /etc/profile  #Proveer el perfil global, asegura que root tiene los
directorios ''sbin'' en el $PATH
```

Descargando y configurando

Ahora que el entorno de compilación está configurado, continuemos con el siguiente paso: la obtención de las fuentes. Descargue un nuevo kernel de www.kernel.org, descomprímalo en /usr/src y cree el vínculo simbólico “linux” para que los comandos sean un poco más genéricos. Tomaremos una versión del kernel 2.6.27.7 como ejemplo. Si su versión es diferente, sabrá donde hacer los cambios de versión en las respectivas cadenas de texto subsecuentes. Si desea saber como verificar la integridad del código fuente usando la clave GPG del kernel, lea el último capítulo.

```
wget http://www.us.kernel.org/pub/linux/kernel/v2.6/linux-2.6.27.7.tar.bz2
tar -C /usr/src -jxvf linux-2.6.27.7.tar.bz2
cd /usr/src
rm linux                     # remueve el vínculo simbólico existente
ln -s linux-2.6.27.7 linux   # crea un vínculo simbólico apuntando a su
nueva fuente linux
```

Cambiar el vínculo simbólico “linux” es seguro. Las aplicaciones no fallarán si deja que apunte hacia un kernel que no sea el que Slackware instaló por Usted.

Probablemente notará más directorios linux-* en /usr/src. Es común dejar que el vínculo “linux” apunte al kernel con el cual trabaja actualmente, sin embargo no es un requerimiento contar con éste. El software moderno que necesite saber la localización del código fuente de un kernel instalado, buscará adonde el vínculo simbólico /lib/modules/<kernelversion>/build apunte

Hay un debate acerca de si se deberían compilar los kernels en el árbol /usr/src o en alguna otra parte completamente diferente.
La causa de la discusión, es un viejo post escrito por Linus Torvalds (julio de 2000) donde aconseja a la gente construir el kernel desde dentro del directorio del usuario (home). Yo creo que el consejo es irrelevante para Slackware por la forma en que tiene sus “kernel headers” y la configuración del paquete glibc. Así, mi consejo es ignorar este viejo post de Linus Torvalds e instalar las fuentes del kernel en /usr/src si así lo desea. La localización del entorno de construcción del kernel es sólamente un asunto de preferencia personal.

Ahora, obtenga un archivo de configuración del kernel de Slackware para tener cierta ventaja durante su propia configuración. Los archivos de configuración de Pat son bastante genéricos. Para cuando lea esto, pueden haber archivos disponibles para una nueva versión 2.6.x

```
wget
http://slackware/mirrors.tds.net/pub/slackware/slackware-12.2/source/k/confi
g-generic-smp-2.6.27.7-smp
cp config-generic-smp-2.6.27.7-smp /usr/src/linux/.config
```

De manera alternativa, puede obtener la configuración del kernel que está corriendo actualmente

```
zcat /proc/config.gz > /usr/src/linux/.config
```

Ejecute make oldconfig en el directorio donde estén las fuentes del kernel nuevo, así, las opciones predeterminadas son usadas desde el archivo .config que recién instaló. Dado que las fuentes de su kernel son más nuevas que su archivo .config, habrá nuevas opciones de configuración. Usted sólo deberá responder a éstas(presione Enter para aceptar los valores por defecto, o M para construir el controlador como un módulo)

```
cd /usr/src/linux
make oldconfig
```

El paso anterior es bastante importante para no contar con una configuración desactualizada, sobre todo si se cambia de rama del kernel.

Ahora ha configurado un kernel bastante genérico (esa es probablemente la razón por la que Pat lo llama “kernel-generic”) pero seguramente querrá cambiar algunas cosas para que se ajuste a sus necesidades. Para ello ejecute el configurador basado en X (si no tiene un servidor X corriendo y está en una consola, sólo ejecute “make menuconfig” para obtener el programa basado en curses)

```
make xconfig
```

De un paseo por el bosque de opciones. Lo que usualmente cambio son cosas como:

- Construir ext3 (necesita el controlador jbd) y los controladores de los sistemas de archivos reiser/xfs/jfs/ext4 dentro del kernel en vez de compilarlos como módulos (así no es necesario crear un "initrd" adicional [vea "Filesystems" en el configurador)])
- Habilitar 64GB de RAM. (dentro de "Processor type and features" > "High Memory Support (64GB)"). Use esto si tiene un sistema con 4 GB de ram o más.
- Habilite el kernel de baja latencia si usa un computador portátil o de escritorio (las aplicaciones multimedia correrán de manera más fluida [dentro de "Processor type and features" > "Preemption model" > "Preemptible kernel"]). Si tiene un sistema de escritorio con muchas aplicaciones multimedia, entonces ésta es una opción útil para Usted, ya que mantendrá a su sistema con mejores respuestas incluso ante una gran carga de procesos.
- Asignar un temporizador de 1000Hz (en "Processor type and features" > "Timer Frequency" > "1000 Hz"). Un conteo de tics más alto, puede ser beneficioso para sistemas de escritorio multimedia
- O asignar un temporizador sin tics (dynamic ticks - under "Processor type and features" > "Tickless System (Dynamic Ticks)").
- Si está (re)construyendo un kernel de Slackware, debería asegurarse que, instalar el nuevo kernel dejará los módulos originales del kernel intactos. Esto se hace cambiando la parte de *local-version* del número de versión del kernel a una cadena de caracteres única (en "General setup" > "Local version - append to kernel release"). Esta opción corresponde a **CONFIG_LOCALVERSION** en su archivo .config. Slackware coloca el valor a "-smp" para un kernel SMP, para que tenga una idea.
  El valor de versión del kernel resultante (como lo muestra "`umake -r`") para una versión "`2.6.37.6`" con una *local-version* "`-alien`" sería "`2.6.37.6-alien`"
- ... y más cosas que no se me ocurren ahora. Puede decidir deshabilitar muchos de los módulos que las configuración global construirá, para acortar el tiempo de arranque, (si no tiene el hardware en su computador). También puede revisar las opciones de software suspend y CPU frequency scaling (dentro de "Processor type and features") si posee un computador portátil

Finalmente guarde su configuración si está satisfecho con ella.

Hay que tener cuidado con los sistemas de archivos ya que si no se incluye correctamente la opción para aquel donde reside el sistema (opción Y) o bien se tiene como módulo sin un initrd, el sistema arrojará un kernel panic y no arrancará

Construyendo el kernel

Ahora inicie la construcción del kernel y de los módulos, e instálelos en los lugares adecuados

```
make bzImage modules                                    # compila el kernel
y los módulos
make modules_install                                    # instala los
módulos en /lib/modules/<kernelversion>
cp arch/x86/boot/bzImage /boot/vmlinuz-custom-2.6.27.7   # copia el nuevo
```

```
kernel
cp System.map /boot/System.map-custom-2.6.27.7           # copia System.map
(opcional)
cp .config /boot/config-custom-2.6.27.7                  # copia de respaldo
de su coniguración del kernel
cd /boot
rm System.map                                            # borra el vínculo
antiguo
ln -s System.map-custom-2.6.27.7 System.map              # crea un nuevo
vínculo
```

En los kernels 2.6.x y 3.x, ejecutar “make” o “make all” en vez de “make bzImage modules” debería ser suficiente, esto construirá los objetos predeterminados, siendo *vmlinux* el kernel sin comprimir, *bzImage* el kernel comprimido y *modules* los módulos del kernel. Ya que no necesitamos el kernel descomprimido, es mejor usar el comando “make bzImage modules”
Si desea saber más acerca de los objetos make disponibles, puede ejecutar “make help” y revisar lo que entrega. Los objetos predeterminados están marcados con un asterisco (*).

Modificando lilo.conf

Edite /etc/lilo.conf y agregue una nueva sección para su nuevo kernel. Recuerde, puede que su nuevo kernel ni siquiera arranque si cometió un error en alguna parte, por eso querrá dejar la(s) seccion(es) de su(s) kernel(s) intacta(s)(para poder cargar su sistema de los kernels que funcionen, en caso de cualquier problema). Su archivo /etc/lilo.conf actual, tendrá una sección similar a ésta al final del archivo

```
image = /boot/vmlinuz
root = /dev/hda1
label = linux
read-only # Non-UMSDOS filesystems should be mounted read-only for checking
```

Agregue otra sección debajo (añadiéndola debajo, garantizará que su kernel actual permanecerá como opción por defecto para arrancar)

```
image = /boot/vmlinuz-custom-2.6.27.7
root = /dev/hda1
label = newkernel
read-only # Non-UMSDOS filesystems should be mounted read-only for checking
```

Después de agregar un párrafo, para el nuevo kernel a /etc/lilo.conf, y guardar el archivo, ejecute lilo para activar los cambios

```
lilo
```

Ahora es tiempo de reiniciar, para probar los resultados. En la pantalla de lilo, seleccione la opción, “newkernel” en vez de “linux”. Si el nuevo kernel arranca bien, puede agregar esta línea en la parte superior de /etc/lilo.conf y re-ejecutar lilo

```
default = newkernel
```

<h3>El paquete Slackware kernel-headers </h3> Usted va a construir y usar un nuevo kernel. y puede preguntarse, ¿Qué debo hacer con el paquete Slackware kernel-headers? **La respuesta es: no lo remueva!**

Hay dos lugares donde encontrará kernel headers: uno es dentro del directorio del código fuente del kernel (en nuestro caso /usr/src/linux-2.6.27.7) y el otro lugar es /usr/include/linux. El paquete kernel-headers usualmente contiene las cabeceras tomadas del código fuente del kernel que Slackware tiene por defecto. Estas cabeceras particulares, son usadas cuando el paquete glibc es construido. El hecho que el paquete kernel-headers instala estos archivos a /usr/include/linux los hace independientes de los archivos de cabecera que encontrará en el directorio del código fuente.

Mientras no remueva glibc, no debería actualizar o remover el paquete kernel-headers

- ¿Cómo se relacionan los paquetes glibc y kernel-headers?

En algún punto, querrá actualizar (recompilar!) partes del software de su sistema. Si ese software está “vinculado” con glibc (como la mayoría del software lo hace), su compilación existosa depende de la presencia de las correctas cabeceras del kernel en /usr/include/linux. No importa si está ejecutando un sistema completamente distinto al kernel que Slackware trae por defecto. El paquete kernel-headers refleja el estado del sistema cuando glibc fue construido. Si elimina el paquete kernel-headers, su sistema no se verá afectado, pero no será capaz de (re-)compilar la mayoría del software

- ¿Todavía sirven las fuentes del kernel, una vez que éste se ha construido?

En el punto anterior, se dijo que la compilación de software del sistema usa los archivos de cabecera localizados en /usr/include/linux. Asimismo, el árbol del código fuente del kernel, es requerido cada vez que quiera compilar un módulo de terceros (madwifi, linux-uvc,ndiswrapper... la lista es interminable). Usted no está limitado a compilar un driver para el kernel que se esté ejecutando. Puede hacerlo para cualquier kernel, mientras el árbol de módulos(/lib/modules) y las fuentes estén presentes. Digamos que va a construir un módulo para un kernel cuya versión está específicada en una variable de entorno $KVER, por ejemplo ejecutando:

```
export KVER=2.6.38.2
```

Durante la compilación del driver necesitará tener disponibles las cabeceras del kernel en /lib/modules/$KVER/build/include/linux. El vínculo simbólico /lib/modules/$KVER/build es creado cuando instala su nuevo kernel y modulos. Si borra las fuentes del kernel después de que construye éste, no será capaz de construir ningún driver “fuera-del-kernel” (otra manera de decir drivers de terceros) más adelante.

Otros paquetes que contienen módulos del kernel

Lo más probable es que tenga instalados uno o más paquetes que contienen módulos del kernel pero que no son partes de éste. Por ejemplo, Slackware instala el paquete “svgalib-helper”, y si instala cualquier controlador inalámbrico, éstos también son básicamente módulos del kernel.
Sea consciente que instalando y arrancando su nuevo kernel, ya no tendrá disponibles estos módulos

fuera-del-kernel.
Debe recompilar sus fuentes para que los módulos resultantes coincidan con la versión de su nuevo kernel.
Puede tener una visión general de todos los paquetes que han instalado un módulo para su kernel actual, ejecutando éste comando (debe ejecutarlo mientras ejecute su viejo kernel)

```
cd /var/log/packages
grep -l "lib/modules/$(uname -r)" *
```

Todos los paquetes mencionados necesitarán recompilarse si también quiere tener los módulos disponibles para su nuevo kernel

Si reconstruye un paquete que contiene un módulo del kernel, use **installpkg** en vez de **upgradepkg** para instalarlo sin remover su versión original.
Si usa *upgradepkg*, esto removerá el módulo del kernel antiguo, y todavía podría necesitarlo si quiere usar versiones anteriores del kernel. Este truco funciona bajo el supuesto que la versión del kernel es parte de la *VERSION* del paquete como por ejemplo: svgalib_helper-1.9.25_2.6.37.6-i486-1.txz (ya sé que es un mal ejemplo ya que este paquete ya no existe)

El método descrito anteriormente, no tendrá consciencia de los módulos del kernel que pueda haber compilado manualmente en vez de crear un paquete para ellos. Típicamente los controladores gráficos propietarios como los de *Nvidia* o *Ati* le causarán algunos momentos preocupantes si olvida recompilarlos para el nuevo kernel antes de iniciar el servidor X, especialmente si su computador arranca con un *runlevel 4* de manera predeterminada.
En ese caso, reinicie en *runlevel 3*, descargue, compile e instale la última versión de los controladores gráficos. Esto le permitirá iniciar su sistema con login gráfico.
Para aquellos que olvidaron cómo iniciar en un *runlevel* distinto al predeterminado, es sencillo: cuando la pantalla de LILO aparezca, escriba la etiqueta de su kernel (*newkernel* en nuestro ejemplo anterior) seguido del número del *runlevel*: Newkernel 3

Creando un initrd

En caso que su kernel no incluya un controlador para su sistema de archivos raiz, o un controlador para su bus SATA, u otras cosas que son sólo construidas como módulos, su kernel “entrará en pánico”(kernel panic), si arranca y no puede acceder los discos, particiones y/o archivos necesarios. Esto, típicamente se ve así:

```
VFS: Cannot open root device "802" or unknown-block (8,2)
Please append a correct "root=" boot option
Kernel Panic-not syncing: VFS: unable to mount root fs on unknown block(8,2)
```

Esto significa que deberá construir un initrd o “Initial Ram Disk”, conteniendo los módulos requeridos. Su localización es luego agregada a la sección apropiada de /etc/lilo.conf, así el kernel puede

encontrarlo cuando arranca, y es capaz de cargar los controladores que necesita para acceder a los discos. Crear un initrd es bastante simple, y mostraré dos casos:uno en caso que posea un sistema de archivos Reiser en su partición raiz y el segundo caso, si tiene un sistema de archivos ext3. En los comandos ejecutados a continuación, se asume que tiene un kernel 2.6.27.7 . Si su nuevo kernel es de una versión distinta, cambie donde corresponda.

Ingrese al directorio /boot

```
cd /boot
```

Ejecute el comando “mkinitrd” para crear el archivo /boot/initrd, el cual contiene un sistema de archivos comprimido con los módulos que le indica que agregue.

```
mkinitrd -c -k 2.6.27.7 -m reiserfs
```

Para un sistema reiser o

```
mkinitrd -c -k 2.6.27.7 -m ext3
```

Para un sistema ext3

Agregue la línea “initrd = /boot/initrd.gz” a la sección “newkernel” en el archivo /etc/lilo.conf. Guarde los cambios y luego re-ejecute lilo; Usaré el ejemplo de lilo.conf que ya había usado en las secciones anteriores

```
image = /boot/vmlinuz-custom-2.6.27.7
  root = /dev/hda1
  initrd = /boot/initrd.gz
  label = newkernel
  read-only # Non-UMSDOS filesystems should be mounted read-only for
checking
```

Re-ejecute lilo

```
lilo
```

La próxima vez que inicie, su kernel no entrará en pánico.
Si ya está usando una imagen initrd con su kernel actual, puede elegir entre dos opciones:

Crear una segunda imagen initrd usando el comando que se muestra a continuación, pero con un nombre explícito para el archivo resultante (el cual debe ser diferente del nombre por defecto para no sobreescribir el antiguo)

```
mkinitrd -c -k 2.6.27.7 -m ext3 -o /boot/initrd-custom-2.6.27.7.gz
```

Luego modifique lilo.conf

```
image = /boot/vmlinuz-custom-2.6.27.7
  root = /dev/hda1
  initrd = /boot/initrd-custom-2.6.27.7.gz
  label = newkernel
```

```
  read-only # Non-UMSDOS filesystems should be mounted read-only for
checking
```

O agregue los módulos para su nuevo kernel en el archivo initrd existente. De esa manera, tendrá una sola imagen initrd conteniendo módulos para múltiples kernels. Todo lo que necesita hacer es dejar fuera la opcion “-c” la cual sirve para eliminar el directorio /boot/initrd-tree y empezar de cero:

```
mkinitrd -k 2.6.27.7 -m ext3
```

He escrito un script (mkinitrd_command_generator.sh) el cual examina su actual sistema Slackware y le muestra un ejemplo de un comando mkintrd adecuado para su configuración actual. Si lo ejecuta, producirá una imagen initrd que contiene todos los módulos del kernel y librerías de soporte para que su sistema pueda arrancar con el kernel genérico de Slackware. Aquí hay un ejemplo de como ejecutar el comando y su salida:

```
/usr/share/mkinitrd/mkinitrd_command_generator.sh /boot/vmlinuz-
generic-2.6.27.7

#
# $Id: mkinitrd_command_generator.sh,v 1.11 2008/04/10 23:56:18
root Exp root $
#
# This script will now make a recommendation about the command
to use
# in case you require an initrd image to boot a kernel that does
not
# have support for your storage or root filesystem built in
# (such as the Slackware 'generic' kernels').
# A suitable 'mkinitrd' command will be:\\
mkinitrd -c -k 2.6.27.7 -f ext3 -r cryptslack -m
mbcache:jbd:ext3 -C /dev/sda8 -u -o /boot/initrd.gz
# An entry in 'etc/lilo.conf' for kernel '/boot/vmlinuz-
generic-2.6.27.7' would look like this:
# Linux bootable partition config begins
# initrd created with 'mkinitrd -c -k 2.6.27.7 -f ext3 -r
cryptslack -m mbcache:jbd:ext3 -C /dev/sda8 -u -o
/boot/initrd.gz'
image = /boot/vmlinuz-generic-2.6.27.7
  initrd = /boot/initrd.gz
  root = /dev/mapper/cryptslack
  label = 2.6.27.7
  read-only
# Linux bootable partition config ends
```

Puede notar que detectó mi partición root LUKS-encriptada.
Este script está incluido en el paquete *mkinitrd* de los release de Slackware después de la versión 12.2

Cargando módulos en el arranque

Previo a Slackware 11.0, los módulos para su kernel eran cargados ya sea por el “hotplug subsystem” o por comandos modprobe explícitos en el archivo /etc/rc.d/rc.modules. Tener los mismos archivos rc.modules para los kernels 2.4.x y 2.6.x no era una situación

En Slackware 12.0 y versiones posteriores, el kernel 2.6.x es el único kernel que está disponible. La carga de los módulos está manejada por udev y explícitamente por los comandos modprobe: los módulos que no son cargados por udev pueden ser puestos todavía en un archivo rc.d. Slackware revisará por la existencia de los siguientes archivos (ejecutables) en éste orden:

- Si /etc/rc.d/rc.modules.local existe, se ejecutará
- Si no, si /etc/rc.d/rc.modules-$(uname -r) existe, se ejecutará
- Si no, si /etc/rc.d/rc.modules existe, se ejecutará

$(uname -r) es la actual versión del kernel. Entonces, por ejemplo, si su versión del kernel es 2.6.27.7-smp, entonces Slackware buscará un archivo /etc/rc.modules-2.6.27.7-smp para ejecutar. De esta manera, archivos rc específicos para diferentes kernels pueden estar presentes, permitiendo un “afinamiento” óptimo de su sistema

EL paquete de Slackware 13.37 /slackware/a/kernel-modules-smp-2.6.37.6_smp-i686-1.tgz instalará el archivo /etc/rc.d/rc.modules-2.6.37.6-smp. Puede usar éste como un ejemplo si quiere construir si propio kernel y necesita comandos modprobe explícitos para módulos específicos del kernel

Si decide construir su propio kernel 2.6 desde las fuentes, podría intrigarse por el hecho que no habrá un archivo llamado /etc/rc.d/rc.modules-$(uname -r)- deberá crearlo. rc.modules es usualmente un vínculo simbólico a rc.modules-2.6.27.7-smp. Un resultado típico de la ausencia de un archivo rc.modules para su kernel específico es que su mouse no responderá. Tome ese comportamiento como una pista para crear el archivo rc.modules. Puede basarse en una copia completa de cualquier archivo rc.modules-2.6.xx. Si su sistema no tiene ningún archivo, puede tomar uno del CD de Slackware como un ejemplo: /source/k/kernel-modules-smp/rc.modules.new. Aquí hay un ejemplo en caso que hubiera construido un kernel versión 2.6.28.8.alien teniendo ya instalado un kernel versión 2.6.27.7-smp

```
cp -a /etc/rc.d/rc.modules-2.6.27.7-smp
/etc/rc.d/rc.modules-2.6.28.8.alien
```

El archivo /etc/rc.d/rc.modules-2.6.28.8.alien será usado cuando su kernel 2.6.28.8.alien arranque

Firma gpg

Los archivos fuente del kernel de linux están firmados con OpenPGP “Linux Kernel Archives Verification Key”. Esto es un medio para verificar que el código fuente que descargó es el archivo original y no ha sido alterado. Los pasos para esta validación, son resumidos en este capítulo

Primero, importe la llave OpenPGP en su llavero GnuPG; ya sea copiándola desde la página de firmas o importándola desde un servidor. La clave ID del kernel es 0x517D0F0E. Un ejemplo sería algo como:

```
gpg --keyserver wwwkeys.pgp.net --recv-keys 0x517D0F0E
```

La salida resultante será algo como esto

```
gpg: key 517D0F0E: public key "Linux Kernel Archives Verification Key
<ftpadmin@kernel.org>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

A continuación obtenga el archivo de firmas para el kernel que acaba de descargar

```
wget
http://www.us.kernel.org/pub/linux/kernel/v2.6/linux-2.6.27.7.tar.bz2.sign
```

Y asegúrese que está en el mismo directorio del kernel.
El paso final es ejecutar gpg en este archivo de firma

```
gpg --verify linux-2.6.27.7.tar.bz2.sign linux-2.6.27.7.tar.bz2
```

Y chequear lo que tiene que reportar

```
gpg: Signature made Fri 21 Nov 2008 12:10:49 AM CET using DSA key ID
517D0F0E
gpg: Good signature from "Linux Kernel Archives Verification Key
<ftpadmin@kernel.org>"
gpg: checking the trustdb
gpg: checking at depth 0 signed=1 ot(-/q/n/m/f/u)=0/0/0/0/0/4
gpg: checking at depth 1 signed=0 ot(-/q/n/m/f/u)=0/0/0/0/1/0
gpg: next trustdb check due at 2012-12-21
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the
owner.
Primary key fingerprint: C75D C40A 11D7 AF88 9981  ED5B C86B A06A 517D 0F0E
```

Si le hubiese dicho a gnupg que confiase en esta llave, la última parte se hubiese visto distinta. Para mi el agregar un nivel de confianza a llave no tiene sentido a menos que conociese a algún desarrollador del kernel que tuviese su clave consigo y que pudiera presentar credenciales confiables. Sin embargo el resultado es que el archivo del código fuente, fue ciertamente firmado con la llave que importó. Así que son buenas noticias

Fuentes

- Fuente Original: http://alien.slackbook.org/dokuwiki/doku.php?id=linux:kernelbuilding
- Escrito originalmente por: Eric Hameleers

- Traducido originalmente por: Mandrake

También vea

- Si desea automatizar el proceso, es posible que desee extraer un script de shell siguiendo las instrucciones anteriores de aquí

howtos, software, kernel, author jcourbis