

02.09

ADMINISTRADOR LINUX

COMPILACIÓN DE KERNEL

EXPERTO EN ADMINISTRACIÓN
DE REDES LINUX

- . MÓDULO 02 • CAPÍTULO 09 . -



Capítulo 09: COMPILACIÓN DE KERNEL



ÍNDICE

| | |
|--|----|
| Compilación del kernel | 3 |
| ¿Para qué compilar el kernel? | 3 |
| Paquetes necesarios para la compilación | 4 |
| ¿De dónde podemos bajar el núcleo de GNU/Linux? | 5 |
| Parches del kernel | 6 |
| ¿Qué es la compilación? | 7 |
| Comandos para la compilación del kernel | 7 |
| Comando make | 7 |
| Comando gcc | 8 |
| Comando lspci | 8 |
| Comando lsmod | 9 |
| ¿Qué debemos conocer antes de empezar? | 10 |
| Pasos a seguir para compilar el kernel | 11 |
| Configuración de grub para iniciar con el nuevo kernel | 14 |
| Resumen de pasos de compilación | 18 |
| Compilación del kernel en formato Debian | 19 |



Suscribite a nuestro Twitter:
twitter.com/CarreraLinuxAr



Suscribite a nuestro Blog:
blog.carreralinux.com.ar





Capítulo 09: **COMPILACIÓN DE KERNEL**

Compilación del kernel

Debemos comenzar por respondernos esta pregunta.



El **kernel** es el corazón o **núcleo del Sistema Operativo**. En nuestro caso, el kernel se llama Linux, y el Sistema Operativo en sí lo compone el kernel junto con una serie de programas y aplicaciones.

¿Para qué compilar el kernel?

Una pregunta que a veces los usuarios nos hacemos es justamente:

¿Para qué compilar el kernel si ya tenemos nuestra máquina funcionando?

La respuesta es simple, y es que entre una versión y otra de los distintos kernel se agregan características nuevas al mismo, además de una serie de mejoras. Por otro lado, el kernel que acompaña a las distribuciones es un kernel genérico, esto quiere decir que no tiene ningún tipo de optimización para nuestro hardware específico, viene por defecto para procesadores i386, y lo más probable es que tenga soporte para una gran cantidad de dispositivos que es innecesaria puesto que no los poseemos.



Por último, el compilar el kernel es un paso altamente educativo y didáctico para comprender en mayor profundidad el funcionamiento del sistema, y siempre es bueno saberlo.



Paquetes necesarios para la compilación

DESARROLLO:

- gcc
- libc5-dev
- libc6-dev
- binutils
- make
- (para x86)bin86

INTERPRETES:

- mawk
- gawk

BASE:

- gzip
- shellutils
- grep

OTROS:

- bzip2
- libncurses5-dev
- patch

ENSAMBLADORES:

- nasm

Una vez que confirmemos la existencia de todas estas herramientas en nuestro sistema, es necesario contar, obviamente, con las fuentes del kernel.



Los parches son archivos que contienen las diferencias entre un árbol de fuentes y otro. Es mucho más conveniente actualizar nuestro kernel a través de este método.



Suscribite a nuestro Facebook:
www.facebook.com/carreralinuxar



Suscribite a nuestro Twitter:
twitter.com/CarreraLinuxAr





¿De dónde podemos bajar el núcleo de GNU/Linux?



Podemos obtener el código fuente del kernel en el sitio www.kernel.org.

Una vez que ingresemos al sitio veremos una gran variedad de versiones para descargar.



Lo que se recomienda es bajar “The latest stable versión of the Linux kernel is: (versión) (fecha) F V VI Changelog”, Si deseamos descargar la versión completa debemos clickear en “F” (full).

Los otros son parches, que sirven para aplicar las últimas modificaciones sobre la versión anterior de las fuentes si ya las habíamos descargado.

Para **descargar el código fuente** también podemos usar el siguiente comando:

```
$ wget ftp://ftp.kernel.org/pub/linux/kernel/vx.x/linux-version.tar.gz
```

Veamos qué significa cada número de la versión:

- El **primer número** de los que van separados por puntos corresponde a la **versión del kernel**, actualmente es la 2 y varía con muy poca frecuencia (la primera fue la 0 en 1991).
- La **segunda cifra** corresponde al **patchlevel** o **número de versión menor**, actualmente es la 6 y varía cada dos o tres años aproximadamente dependiendo de la estabilidad de los parches aplicados a las distintas versiones en las cifras pares.



Un “patchlevel” **impar** indica una **versión del kernel “beta”** o en pruebas, que hace de paso intermedio para que los desarrolladores puedan probar nuevos componentes para la siguiente versión par.



Las versiones impares normalmente no están pensadas para usuarios finales, y solo deberían usarse si la versión estable actual no contiene drivers para un dispositivo específico y no ha encontrado ningún driver por separado salvo en el kernel “beta”.

- Por último la **tercera cifra**, indica el **sublevel**, varía cada pocos meses y normalmente consiste en pequeñas correcciones de fallos que tenga la versión inicial del “patchlevel” correspondiente.

En las versiones estables va más despacio porque solo se corrigen pequeños fallos y van en proporción logarítmica. Recién liberada una versión estable hay más fallos que encontrar. En las versiones beta este número varía rápidamente, porque normalmente se van añadiendo nuevos drivers además de ir corrigiendo fallos.

Parches del kernel

Los parches **se aplican en forma consecutiva**. Esto quiere decir que, si por ejemplo, la versión del kernel es la 2.6.31 y deseamos actualizarlo a la 2.6.35, necesitamos los parches 2.6.32, 2.6.33 y 2.6.34.

Estos deben ser aplicados en ese orden, uno después del otro, y **no es necesario realizar una compilación en cada paso**.



Es necesario remarcar que no es posible parchar una versión mayor del kernel, por ejemplo del 2.4.29 no se puede parchar al 2.6.

Los parches los vamos a encontrar en el sitio **www.kernel.org** en el mismo directorio donde se encontraba el kernel completo pero el nombre del archivo que guarda el parche es: **patch-2.6.32.gz**.



Suscribite a nuestro Blog:
blog.carreralinux.com.ar





¿Qué es la compilación?



La compilación es el proceso que se le aplica a los archivos de texto que escribió un programador (con una sintaxis específica llamada “lenguaje de programación”) para que los mismos se conviertan en archivos “binarios” (formato que entiende la computadora).

Los comandos que se usan durante la compilación son los que a partir de este momento empezaremos a mostrar.

Comandos para la compilación del kernel

COMANDO MAKE

Sintaxis del comando:

```
make [opciones] [seccion-Makefile]
```



Este comando se utiliza para compilar, linkeditar e instalar un programa que requiera de la compilación de varias fuentes (archivos .c).

Generalmente, un programa medianamente grande, se integra de una biblioteca de funciones propias y varias fuentes. **El comando make conoce en qué orden de secuencia debe compilar todas estas fuentes y dónde debe instalarlas** para que el usuario utilice el programa recién compilado y cómo debe relacionarlos entre sí para que puedan ejecutarse en el orden especificado.



COMANDO GCC

Sintaxis del comando:

```
gcc [opciones] <archivo.c>
```

Este **comando compila y linkedita los programas**.

Opciones:

- **gcc -I:** directorio que utilizará para buscar los archivos .h (headers).
- **gcc -L:** directorio donde buscará las bibliotecas para linkeditar.
- **gcc -l:** nombre de la biblioteca a utilizar (sin las primeras 3 letras “lib”).

COMANDO LSPCI

Este comando nos **mostrará la información de los chipset** que tenemos en nuestro equipo:

```
equipo1:~# lspci
0000:00:00.0 Host bridge: Silicon Integrated Systems [SiS] 530 Host (rev 03)
0000:00:00.1 IDE interface: Silicon Integrated Systems [SiS] 5513
[IDE] (rev d0)
0000:00:01.0 ISA bridge: Silicon Integrated Systems [SiS]
SiS85C503/5513 (LPC Bridge) (rev b1)
0000:00:01.1 ff00: Silicon Integrated Systems [SiS] ACPI
0000:00:01.2 USB Controller: Silicon Integrated Systems [SiS] USB
1.0 Controller (rev 11)
0000:00:02.0 PCI bridge: Silicon Integrated Systems [SiS] Virtual
PCI-to-PCI bridge (AGP)
0000:00:0b.0 Ethernet controller: Digital Equipment Corporation
DECchip 21140 [FasterNet] (rev 22)
0000:00:0f.0 Multimedia audio controller: ESS Technology ES1969
Solo-1 Audiodrive (rev 01)
0000:01:00.0 VGA compatible controller: Silicon Integrated Sys-
tems [SiS] 530/620 PCI/AGP VGA Display Adapter (rev a3)
```




COMANDO LSMOD

Este comando nos va a permitir **verificar que módulos está incluyendo nuestro kernel actual:**

```
equipo1:~# lsmod
Module                Size  Used by
binfmt_misc           6599   1
nvidia                7088432 24
snd_via82xx           20140   2
gameport              9327   1 snd_via82xx
snd_ac97_codec        99227   1 snd_via82xx
snd_usb_audio         86480   1
ac97_bus              1014   1 snd_ac97_codec
snd_pcm               71475   3 snd_via82xx,snd_ac97_codec,s-
nd_usb_audio
snd_hwdep             5040   1 snd_usb_audio
snd_usbmidi_lib       17413   1 snd_usb_audio
snd_page_alloc        7120   2 snd_via82xx,snd_pcm
snd_mpu401_uart       5661   1 snd_via82xx
snd_seq_midi          4588    0
snd_seq_midi_event    6047   1 snd_seq_midi
```



Suscribite a nuestro Facebook:

www.facebook.com/carreralinuxar



Suscribite a nuestro Twitter:

twitter.com/CarreraLinuxAr



Suscribite a nuestro Blog:

blog.carreralinux.com.ar





¿Qué debemos conocer antes de empezar?

Antes de empezar a compilar el kernel debemos tener en cuenta el tipo de hardware que tenemos en el equipo:

- 1.- Tipo de procesador que poseemos.
- 2.- Si nuestra computadora es PCI.
- 3.- El tipo de controlador IDE de nuestro sistema, si es que éste tiene uno.
- 4.- El tipo de controlador SCSI de nuestro sistema, si lo posee.
- 5.- El tipo de interfaz que utilice el CD.
- 6.- El tipo de placa de red de nuestro sistema.
- 7.- El número de fabricación y modelo del chipset de la placa de red.
- 8.- Los números de fabricación, de modelo y de chip de la placa de sonido.

Una buena forma de obtener toda esta información es tipiar en la consola el siguiente comando:

```
equipo1:~# dmesg
```



El mismo nos mostrará todo lo que detectó el kernel en la máquina. **No olviden anotar toda la información del hardware.** La necesitaremos después.

Otra forma de **detectar el hardware del equipo** es ejecutar el **comando lspci** que vimos anteriormente:

```
equipo1:~# lspci
```

Algo que no podemos dejar de mencionar es que el kernel también tiene algo llamado módulos. **Los módulos son controladores que se cargan al Kernel cuando éste está operando**, o sea, **en tiempo de ejecución**. Están ubicados dentro del directorio **/lib/modules/version**. Son “.o” porque puede unirse con más código.



Suscribite a nuestro Facebook:
www.facebook.com/carreralinuxar





Pasos a seguir para compilar el kernel

- 01 Bajamos el kernel de **www.kernel.org** y lo guardamos en **/tmp**.
- 02 Descomprimos el núcleo en el directorio **/usr/src**. Recordemos que para cambiarnos a este directorio debemos tipear **cd /usr/src**. Para descomprimir tipeamos **tar xzvf /tmp/linux-version.tar.gz**. Una vez que se ha descomprimido tendremos el siguiente directorio **/usr/src/linux-version**:

```
equipo1:~# cd /usr/src  
equipo1:~# tar xzvf /tmp/linux-version.tar.gz
```

- 03 Si tuviéramos en nuestra máquina el kernel una versión anterior a la disponible podríamos bajar únicamente el parche de kernel.org en vez de la versión completa del kernel. Si realizamos esta operación deberemos hacer lo siguiente:

```
equipo1:~# zcat /tmp/patch-version.gz | patch -p0.
```

- Nota: si el kernel fuera versión 2.6 lo que cambia es el parámetro que le pasamos al **comando patch**:
- ```
equipo1:~# zcat /tmp/patch-2.6-32.gz | patch -p1.
```

De esta forma estaremos utilizando el **zcat** para ver el archivo y redireccionando esta salida al **comando patch**. Una vez que finalicemos este proceso se verá una lista de los archivos modificados por el parche.

- 04 Procedemos ahora a **renombrar el archivo parcheado** para evitar confusiones en un futuro:

```
equipo1:~# mv linux-version-vieja linux-version-nueva
```

- A partir de aquí **comienza el proceso de configuración**.



Suscribite a nuestro Blog:  
**blog.carreralinux.com.ar**



- 05 Si queremos borrar cualquier configuración que haya en el núcleo y dejar un kernel genérico ejecutamos lo siguiente:

```
equipo1:~# make mrproper
```

**IMPORTANTE:**

Este comando va a borrar cualquier configuración de kernel anterior así que sólo debe usarse la primera vez que compilamos el kernel, si queremos recompilar el kernel actual no vamos a usar este comando.

- 06 Si deseamos setear el núcleo definiendo paso a paso las configuraciones, deberemos elegir alguna de estas **tres opciones**:

- **MODO TEXTO:** tipiamos en la consola el **comando make config** y nos harán preguntas acerca de cómo queremos setear cada cosa. (Esta operación **puede demorar muchísimo tiempo**, ya que el Sistema realizará pregunta por pregunta y esperará que contestemos por sí o por no cada una de ellas para continuar).
- **MODO TEXTO CON MENÚ DE OPCIONES:** tipiamos en la consola el **comando make menuconfig** iniciará una aplicación funcional para configurar el kernel. Observemos que su uso es bastante intuitivo: las opciones con un < > indican la posibilidad de modularizar. Si en ella presionamos **M** la seleccionaremos como módulo. Si presionamos ESPACIO aparecerá un \* indicando que será incluido en el kernel. Finalmente, si presionamos **N** lo deseleccionaremos. Las opciones marcadas con **[ ]** no son modularizables. Es importante aclarar en este momento que existen dos formas de setear el núcleo: **monolítica** ó **modular**. En la **monolítica** todo lo que necesitamos está incluido en el núcleo y **una vez compilado no podremos cambiar nada**. Con la forma **modular** podremos **instalar y desinstalar los módulos por separado** sin tener que reiniciar la PC. Vamos a seleccionar los módulos en la configuración **n** y se guardarán en **/lib/modules**.



• **MODO GRÁFICO:** para acceder a este modo tipiamos **make xconfig**. En este modo se lanzará una aplicación sumamente amigable en la que no tendremos ningún problema. Una vez que elegimos la opción tendremos **configurar los siguientes puntos** (es muy importante que no nos olvidemos ¡ninguno!):

- **Procesador:** procesor type and features.
- **Dispositivo de placa de red:** network device support, recordemos habilitar el protocolo ppp, que nos permitirá configurar la conexión a internet una vez que finalice la compilación.
- **Firewall:** networking packet filtering, seleccionamos networking filters y luego Iptables suport.
- **File Systems:** seleccionamos Ext3, NFS, NTFS, Netware, DOS FAT, MSDOS, Vfat, Virtual Memory, ISO 9660 (cdrom).
- **Networking Options:** seleccionamos Packet Socket, Packet Filtering, IP: Netfilter Configuration.
- **Sound**
- **Kernel Hacking**

**07** Cuando se termina de configurar se genera el archivo **.config** y nos instará a ejecutar **make dep** y **clean**:

- **Make dep:** compila lo que se necesita para que el kernel se compile correctamente. Llama a gcc y compila las dependencias.
- **Make clean:** borra los archivos temporales que se generaron al compilar las librerías.

```
equipo1:~# make dep
equipo1:~# make clean
```

**08** En este momento deberemos ejecutar el **comando make bzImage** que generará el kernel propiamente dicho con el cual trabajaremos luego. Los archivos creados se guardan en **/usr/src/linux/arch/i386/boot/bzImage**.

```
equipo1:~# make bzImage
```

**09** Tipiamos en la consola **make modules**, para compilarlos.

```
equipo1:~# make modules
```





- 10 Tipiamos **make modules\_install** para acoplarlos al núcleo.
- ```
equipo1:~#make modules_install
```

- 11 En este punto el kernel ya ha sido compilado y debe ser ubicado en el sector de booteo para estar accesible cuando iniciamos la PC. Para esto deberemos tipiar:
- ```
equipo1:~# cp /usr/src/linux/arch/i386/boot/bzImage /boot/vmlinuz-version
```

Ahora vamos a escribir el sector de boot para arrancar con nuestro nuevo kernel debemos tener en cuenta el gestor de arranque que está usando el sistema operativo en este momento las opciones son **lilo** o **grub**.

## Configuración de grub para iniciar con el nuevo kernel

Ahora llegó el momento de que el kernel sea visible por el gestor de arranque y que este pueda bootearlo. Para esto vamos a usar GRUB.

- 01 Primero crearemos la versión reducida del kernel:
- ```
mkinitramfs -v -k -o /boot/initramfs-<version>-<arch> <version>
```

- ! Donde version corresponde a la version de kernel que estamos instalando. Es importante que sea la version que descargamos.

- 02 Luego debemos actualizar el archivo grub.cfg que es el archivo que utiliza grub para saber donde encontrar el kernel:
- ```
grub-mkconfig -o /boot/grub/grub.cfg
```



Suscribite a nuestro Twitter:  
**[twitter.com/CarreraLinuxAr](https://twitter.com/CarreraLinuxAr)**



- 03 Debemos modificar la sección que hace referencia a nuestro kernel. Para esto abrimos el reciente archivo `grub.cfg`, y buscamos la sección `menuentry` que haga referencia a nuestro nuevo kernel:

```
menuentry 'Debian GNU/Linux, with Linux <version>-amd64' --class
debian --class gnu-linux --class gnu --class os {
 insmod part_msdos
 insmod ext2
 set root='(hd0,msdos1)'
 search --no-floppy --fs-uuid --set f54bf91d-...-7fcfb1549db2
 echo 'Loading Linux <version>-amd64 ...'
 linux /vmlinuz-<version>-amd64 root=/dev/sda2 ro quiet
}
menuentry 'Debian GNU/Linux, with Linux <version>-amd64 (reco-
very mode)' --class debian --class gnu-linux --class gnu --class
os {
 insmod part_msdos
 insmod ext2
 set root='(hd0,msdos1)'
 search --no-floppy --fs-uuid --set f54bf91d-...-7fcfb1549db2
 echo 'Loading Linux <version>-amd64 ...'
 linux /vmlinuz-<version>-amd64 root=/dev/sda2 ro single
}
```

- ! Como pueden ver, hay una línea que dice `/vmlinuz-<version>` seguida de otra que dice `root=/dev/sda2`. Esto indica la partición en la cual se encuentra `vmlinuz.<version>`.



Suscribite a nuestro Facebook:

[www.facebook.com/carreralinuxar](https://www.facebook.com/carreralinuxar)



- 04 Debemos modificar esto por el UUID para que en caso que cambie de sda a sdb aún encuentre el kernel.

```
blkid /dev/sda2
```

```
/dev/sda2: UUID="55f17a20-7216-436a-97ae-537bfc6e60c7" TYPE="ext4"
```



Algo importante de comprender es que `/dev/sda2` se encuentra montado en `/boot` en mi ejemplo, es por ese motivo que busco esa partición.

Aparte la línea de Linux dice `/vmlinuz-<version>`. Y no `/boot/vmlinuz-<version>`. Esto es porque primero se para en la partición `/dev/sda2` y luego busca el kernel dentro. Si el `/` estaría en la misma partición que `/boot`, entonces la línea será `linux /boot/vmlinuz-<version>-`



Noten también que vamos a agregar debajo de cada sección **menuentry** las líneas correspondientes al kernel reducido. Estas son:

```
echo 'Loading initial ramdisk ...'
initrd /initramfs-<version>-amd64
```



Suscribite a nuestro Facebook:  
**[www.facebook.com/carreralinuxar](https://www.facebook.com/carreralinuxar)**



Suscribite a nuestro Twitter:  
**[twitter.com/CarreraLinuxAr](https://twitter.com/CarreraLinuxAr)**



Suscribite a nuestro Blog:  
**[blog.carreralinux.com.ar](https://blog.carreralinux.com.ar)**



Entonces, ese fragmento nos queda:

```
menuentry 'Debian GNU/Linux, with Linux <version>-amd64' --class
debian --class gnu-linux --class gnu --class os {
 insmod part_msdos
 insmod ext2
 set root='(hd0,msdos1)'
 search --no-floppy --fs-uuid --set f54bf91d-...-7fcfb1549db2
 echo 'Loading Linux -amd64 ...'
 linux /vmlinuz-<version>-amd64 root=UUID=-
55f17a20-7216-436a-97ae-537bfc6e60c7 ro quiet
 echo 'Loading initial ramdisk ...'
 initrd /initramfs-<version>-amd64
}
menuentry 'Debian GNU/Linux, with Linux <version>-amd64 (reco-
very mode)' --class debian --class gnu-linux --class gnu --class
os {
 insmod part_msdos
 insmod ext2
 set root='(hd0,msdos1)'
 search --no-floppy --fs-uuid --set f54bf91d-...-7fcfb1549db2
 echo 'Loading Linux <version>-amd64 ...'
 linux /vmlinuz-<version>-amd64 root=UUID=-
55f17a20-7216-436a-97ae-537bfc6e60c7 ro single
 echo 'Loading initial ramdisk ...'
 initrd /initramfs-<version>-amd64
}
```



Suscribite a nuestro Facebook:

**[www.facebook.com/carreralinuxar](https://www.facebook.com/carreralinuxar)**



Suscribite a nuestro Twitter:

**[twitter.com/CarreraLinuxAr](https://twitter.com/CarreraLinuxAr)**





## Resumen de pasos de compilación

Lista rápida de **pasos a seguir** para compilar nuestro kernel:

- make mrproper
- make menuconfig
- make dep
- make clean
- make bzImage
- # (ir a tomar café)
- make modules
- make modules\_install
- cp /usr/src/linux-version/arch/i386/boot/bzImage /boot/vmlinuz-version



OJO, en la línea anterior es **vmlinuz** (con z al final)

- Ramdisk inicial
- mkinitrd -o /boot/initrd-version.img version



OPCIONAL, crear un nuevo **rescue disk** (¡RECOMENDADO!).

- mkboot version

Tenemos que escribir el siguiente comando:

```
equipo1:~# dmesg | less
```



Este comando nos permitirá ver que está detectando nuestro kernel durante el arranque.



Suscribite a nuestro Blog:  
**[blog.carreralinux.com.ar](http://blog.carreralinux.com.ar)**







## Compilación del Kernel en formato Debian

Por último veremos cómo compilar el kernel según Debian. Para hacerlo tenemos que generar los archivos **.deb** del kernel y los módulos.

Veamos el proceso **paso a paso**:

- 01 Seleccionamos de esta lista lo que queremos compilar:  
`make-kpkg kernel-image modulo-image kernel-headers`



El primer y segundo parámetro generarán el archivo **.deb** del kernel y los módulos. El tercer parámetro genera los encabezados de las fuentes del kernel.

- 02 Instalación del nuevo kernel. El comando que utilizaremos será el siguiente:  
`dpkg -i kernel-version`

- 03 Para generar la imagen de boot tenemos que tipiar el siguiente comando:  
`mkinitrd -o /boot/initrd-img-version versionkernel`

Una vez hecho este último paso ya **estamos listos para rebootear**, ya que genera las líneas necesarias en el archivo **/boot/grub/menu.lst** así que sólo nos resta reiniciar la máquina y testear nuestro nuevo kernel.



Suscribite a nuestro Facebook:  
[www.facebook.com/carreralinuxar](https://www.facebook.com/carreralinuxar)



Suscribite a nuestro Twitter:  
[twitter.com/CarreraLinuxAr](https://twitter.com/CarreraLinuxAr)

