

Compilación del *kernel* Linux

Explicación de práctica 2

Sistemas Operativos

Facultad de Informática
Universidad Nacional de La Plata

2019



① Fundamentos

② Historia

③ Versionado

④ Compilación

⑤ Apéndice
Debian way



1 Fundamentos

2 Historia

3 Versionado

4 Compilación

5 Apéndice
Debian way

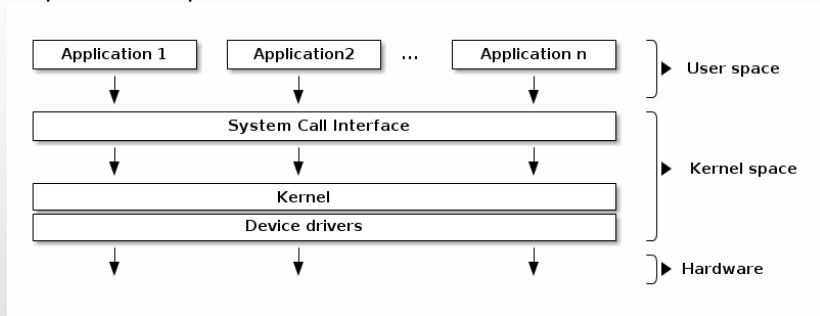


¿Qué es el kernel Linux?

- Programa que **ejecuta** programas y **gestiona** dispositivos de hardware
- Encargado de que el software y el hardware puedan trabajar juntos
- Principales funciones
 - Administración de **memoria principal**
 - Administración de **uso de la CPU**
- Es de código abierto a los usuarios (**kernel/sched.c**)
- En una misma estructura de código fuente se da soporte a **todas las arquitecturas**
- Liberado bajo licencia GPLv2
- En un sentido estricto es el Sistema Operativo



Arquitectura típica:



Fuente: <https://linux-kernel-labs.github.io>

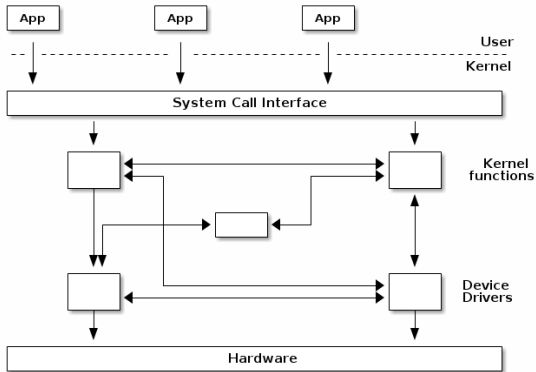


A monolithic kernel is one where there is no access protection between the various kernel subsystems and where public functions can be directly called between various subsystems.

Fuente: <https://linux-kernel-labs.github.io>



Tipos de kernel - Monolítico



Fuente: <https://linux-kernel-labs.github.io>



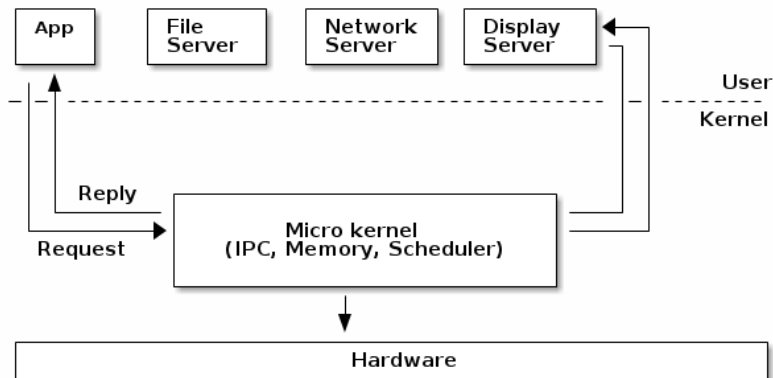
A micro-kernel is one where large parts of the kernel are protected from each-other, usually running as services in user space. Because significant parts of the kernel are now running in user mode, the remaining code that runs in kernel mode is significantly smaller, hence micro-kernel term.

In a micro-kernel architecture the kernel contains just enough code that allows for message passing between different running processes. Practically that means implement the scheduler and an IPC mechanism in the kernel, as well as basic memory management to setup the protection between applications and services.

Fuente: <https://linux-kernel-labs.github.io>



Tipos de kernel - Micro kernel



Fuente: <https://linux-kernel-labs.github.io>



¿Qué es el kernel Linux?

- Es un núcleo monolítico híbrido
 - Los drivers y el código del Kernel se ejecutan en modo privilegiado
 - Lo que lo hace híbrido es la posibilidad de cargar y descargar funcionalidad a través de módulos

Software libre

Se dispone de la **libertad** de:

- | | |
|------------|----------------------|
| 1 Usar | 3 Distribuir |
| 2 Estudiar | 4 Mejorar y publicar |



¿Qué es el kernel Linux?

- Es un núcleo monolítico híbrido
 - Los drivers y el código del Kernel se ejecutan en modo privilegiado
 - Lo que lo hace híbrido es la posibilidad de cargar y descargar funcionalidad a través de módulos

Software libre

Se dispone de la **libertad** de:

- | | |
|------------|----------------------|
| 1 Usar | 3 Distribuir |
| 2 Estudiar | 4 Mejorar y publicar |



1 Fundamentos

2 Historia

3 Versionado

4 Compilación

5 Apéndice
Debian way



En 1991 Linus Torvalds inicia la programación del *kernel* Linux basado en Minix[2] (Clon de Unix desarrollado por Tanenbaum en 1987 con el fin de crear un SO de uso didáctico).

Primer anuncio

(1991)

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones.

[...]

It is **NOT** protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-).

Linus Torvalds, 25/08/1991, comp.os.minix[3]



El 5 de octubre de 1991, se anuncia la primera versión “oficial” de Linux (0.02).

Primera *release*

(1991)

[...]

As I mentioned a month ago, I'm working on a free version of a minix-lookalike for AT-386 computers. It has finally reached the stage where it's even usable (though may not be depending on what you want), and I am willing to put out the sources for wider distribution. It is just version 0.02 (+1 (very small) patch already), but I've successfully run bash/gcc/gnu-make/gnu-sed/compress etc under it.

[...]

Linus Torvalds, 05/10/1991, `comp.os.minix`[3]



En 1992, con la *release* de la versión 0.12, se decide cambiar a una licencia GNU.

Licencia GNU

(1992)

[...]

The Linux copyright will change: I've had a couple of requests to make it compatible with the GNU copyleft, removing the "you may not distribute it for money" condition. I agree. I propose that the copyright be changed so that it confirms to GNU - pending approval of the persons who have helped write code. I assume this is going to be no problem for anybody: If you have grievances ("I wrote that code assuming the copyright would stay the same") mail me. Otherwise The GNU copyleft takes effect as of the first of February. If you do not know the gist of the GNU copyright - read it.

[...]

Linus Torvalds, 1992 [4]



Fechas relevantes:

- En marzo de 1994 Torvalds considera que todos los componentes del *kernel* estaban suficientemente maduros y lanza la versión 1.0.
- En el año 1995 Linux se porta a arquitecturas *DEC Alpha* y *Sun SPARC*. Con el correr de los años se portó a otra decena de arquitecturas.
- En mayo de 1996 se decide adoptar a Tux como mascota oficial de Linux.



Fechas relevantes:

- En julio de 1996 se lanza la versión 2.0 y se define un sistema de nomenclatura. Se desarrolló hasta febrero de 2004 y terminó con la versión 2.0.40. Esta versión comenzó a brindar soporte a sistemas multiprocesadores.
- En 2001 se lanza la versión 2.4 y se deja de desarrollar a fines del 2010 con la 2.4.37.11. La versión 2.4 fue la que catapultó a GNU/Linux como un sistema operativo estable y robusto.



Fechas relevantes:

- A fines del año 2003 se lanza la versión 2.6. Esta versión ha tenido muchas mejoras para el *kernel* dentro de las que se destacan soporte de *threads*, mejoras en la planificación y soporte de nuevo hardware.
- El 3 de agosto de 2011 se lanza la versión 2.6.39.4 anunciándose la misma desde meses previos como la última en su revisión.
- El 17 Julio de 2011 se lanza la versión 3.0
 - No agrega mayores cambios. La decisión del cambio son los 20 años del SO y no superar los 40 números de revisión
 - Totalmente compatible con Kernel 2.6



- El 17 Julio de 2011 se lanza la versión 3.0
 - Termina con la versión 3.8.30
 - Provee mejoras en Virtualización y FileSystems

Anuncio de Linux 3.0

(2011)

[...]

I decided to just bite the bullet, and call the next version 3.0. It will get released close enough to the 20-year mark, which is excuse enough for me, although honestly, the real reason is just that I can no longer comfortably count as high as 40.

[...]

Linus Torvalds, 2011 [1]



- El 12 de Abril de 2015 se lanza la versión 4.0
 - Una de sus principales mejoras es la posibilidad de aplicar parches y actualizaciones si necesidad de reiniciar el SO.
 - Soporte para nuevas CPU
 - La versión actual es 4.5

Anuncio de Linux 4.0

(2015)

[...]

.. Because the people have spoken, and while most of it was complete gibberish, numbers don't lie. People preferred 4.0, and 4.0 it shall be. Unless somebody can come up with a good argument against it.

[...]

Linus Torvalds, 2015



1 Fundamentos

2 Historia

3 Versionado

4 Compilación

5 Apéndice
Debian way



Anatomía de una versión

A.B.C.[D]

- A Denota Versión. Cambia con menor Frecuencia. en 1994 (versión 1.0), en 1996 (versión 2.0), en 2011 (versión 3.0) y en en 2015 (versión 4.0).
- B Denota revisión mayor. Antes de la versión 2.6, los números impares indicaban desarrollo, los pares producción.
- C Denota revisión menor. Solo cambia cuando hay nuevos *drivers* o características.
- D Se utiliza cuando se corrige un grave error sin agregar nueva funcionalidad.

En el año 2011, cuando se pasó de la versión 2.6.39 a la 3.0, se eliminó el número de revisión mayor (B)[1].



Anatomía de una versión

A.B.C.[D]

- A** Denota Versión. Cambia con menor Frecuencia. en 1994 (versión 1.0), en 1996 (versión 2.0), en 2011 (versión 3.0) y en en 2015 (versión 4.0).
- B** Denota revisión mayor. Antes de la versión 2.6, los números impares indicaban desarrollo, los pares producción.
- C** Denota revisión menor. Solo cambia cuando hay nuevos *drivers* o características.
- D** Se utiliza cuando se corrige un grave error sin agregar nueva funcionalidad.

En el año 2011, cuando se pasó de la versión 2.6.39 a la 3.0, se eliminó el número de revisión mayor (**B**)[1].



Anatomía de una versión

X.Y.Z

- X Indica serie principal. Cambia cuando su funcionalidad sufre un cambio muy importante.
 - Y Indica si es una versión de producción o desarrollo.
 - Z Nuevas versiones dentro de la actual. *Bugfixes*.
- Existían dos versiones del *kernel*:
 - Números Y pares indicaban versión en Producción (estable)
 - Números Y impares indicaban versión en Desarrollo



1 Fundamentos

2 Historia

3 Versionado

4 Compilación

5 Apéndice
Debian way



¿Por qué recompilarlo?

- Soportar **nuevos dispositivos** como, por ejemplo, una placa de video
- Agregar **mayor funcionalidad** (soporte para algún hardware específico)
- Optimizar funcionamiento de acuerdo al **sistema en el que corre**
- **Adaptarlo** al sistema donde corre (quitar soporte de hardware no utilizado)
- **Corrección** de *bugs* (problemas de seguridad o errores de programación)



- gcc: Compilador de C
- make: ejecuta las directivas definidas en los Makefiles
- binutils: *assembler*, *linker*
- libc6: Archivos de encabezados y bibliotecas de desarrollo
- ncurses: bibliotecas de menú de ventanas (solo si usamos menuconfig)
- initrd-tools: Herramientas para crear discos RAM

Tip

En Debian y distribuciones derivadas, todo este software se encuentra empaquetado. Por ejemplo, para instalar el software requerido para hacer la práctica:

```
# apt-get install build-essentials  
libncurses-dev
```



- gcc: Compilador de C
- make: ejecuta las directivas definidas en los Makefiles
- binutils: *assembler*, *linker*
- libc6: Archivos de encabezados y bibliotecas de desarrollo
- ncurses: bibliotecas de menú de ventanas (solo si usamos menuconfig)
- initrd-tools: Herramientas para crear discos RAM

Tip

En Debian y distribuciones derivadas, todo este software se encuentra empaquetado. Por ejemplo, para instalar el software requerido para hacer la práctica:

```
# apt-get install build-essentials  
libncurses-dev
```




- 1 Obtener el código fuente.
- 2 Preparar el árbol de archivos del código fuente.
- 3 Configurar el *kernel*
- 4 Construir el *kernel* a partir del código fuente e instalar los módulos.
- 5 Reubicar el *kernel*.
- 6 Creación del `initramfs`
- 7 Configurar y ejecutar el gestor de arranque (LILO, GRUB, etc).
- 8 Reiniciar el sistema y probar el nuevo kernel.



`http://www.kernel.org`

Protocol	Location
HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/

Latest Stable Kernel:
 **4.5**

mainline:	4.6-rc2	2016-04-03	[tar.xz]	[pgp]	[patch]	[view diff]	[browse]
mainline:	4.5	2016-03-14	[tar.xz]	[pgp]	[patch]	[view diff]	[browse]
stable:	4.4.6	2016-03-16	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	4.1.21	2016-04-03	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	3.18.30	2016-04-03	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	3.14.65	2016-03-16	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]



Para descargarlo desde la terminal:

```
$ cd /usr/src  
$ sudo wget https://kernel.org/pub/linux/kernel/v4.x/linux-4.15.tar.xz
```

Opcional: Por cuestiones de seguridad, opcionalmente se puede descargar un archivo con una firma criptográfica del *kernel* descargado. Esta firma nos permite comprobar que el archivo que descargamos es exactamente el mismo que fue subido, y que no hubo modificaciones maliciosas de por medio.

```
$ wget https://kernel.org/pub/linux/kernel/v4.x/linux-4.15.tar.sign  
$ xz -cd linux-4.15.tar.xz | gpg --verify linux-4.15.tar.sign -  
gpg: Signature made Thu 30 Mar 2014 11:35:06 AM ART using RSA key ID 4395693E  
gpg: Good signature from ``Greg Kroah-Hartman (Linux kernel stable release signing key) <greg@kroah.com>''  
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: 647F 2865 4894 E3BD 4571 99BE 38DB BDC8 6092 693E
```



Preparar el árbol de archivos

Por convención el código fuente del *kernel* se guarda en `/usr/src`. Sin embargo, como dicho directorio generalmente no tiene permisos de escritura para usuarios no privilegiados, el archivo se debe descomprimir en un directorio donde tengamos permisos, como el `$HOME` del usuario actual.

```
$ mkdir $HOME/kernel  
$ cd $HOME/kernel  
$ tar xvf /usr/src/linux-4.15.tar.xz
```

Generalmente se crea un enlace simbólico llamado `linux` apuntando al directorio del código fuente que actualmente se está configurando

```
$ ln -s /usr/src/linux-4.15 /usr/src/linux
```



El *kernel* Linux se configura mediante el archivo `.config`. Este archivo, que reside en la raíz del directorio del *kernel*, contiene las instrucciones de qué es lo que el *kernel* debe compilar.

```
$ cat /boot/config-$(uname -r) | tail -n5  
CONFIG_UCS2_STRING=y  
CONFIG_FONT_SUPPORT=y  
# CONFIG_FONTS is not set  
CONFIG_FONT_8x8=y  
CONFIG_FONT_8x16=y
```

Existen tres interfaces que permiten generar este archivo:

- `make config`: modo texto y secuencial. **Tedioso.**
- `make xconfig`: interfaz gráfica utilizando un sistema de ventanas. **No todos los sistemas tienen instalado X.**
- `make menuconfig`: este modo utiliza ncurses, una librería que permite generar una interfaz con paneles desde la terminal. **Generalmente el más utilizado.**



El *kernel* Linux se configura mediante el archivo `.config`. Este archivo, que reside en la raíz del directorio del *kernel*, contiene las instrucciones de qué es lo que el *kernel* debe compilar.

```
$ cat /boot/config-$(uname -r) | tail -n5
CONFIG_UCS2_STRING=y
CONFIG_FONT_SUPPORT=y
# CONFIG_FONTS is not set
CONFIG_FONT_8x8=y
CONFIG_FONT_8x16=y
```

Existen tres interfaces que permiten generar este archivo:

- `make config`: modo texto y secuencial. **Tedioso.**
- `make xconfig`: interfaz gráfica utilizando un sistema de ventanas. **No todos los sistemas tienen instalado X.**
- `make menuconfig`: este modo utiliza `ncurses`, una librería que permite generar una interfaz con paneles desde la terminal. **Generalmente el más utilizado.**



Las herramientas mencionadas permiten:

- Crear un archivo `.config` con las directivas de compilación
- Configurar un *kernel* desde cero es una tarea tediosa y propensa a errores (*kernels* que no arranquen). Estas herramientas automatizan el proceso por nosotros.

Consejos

- Lo ideal es ir manteniendo el `.config` para no tener que configurar todo de cero
- Cada nueva versión, puede valerse de un `.config` anterior.
- Por convención, es recomendable almacenar en el directorio `/boot` la imagen compilada del *kernel* junto con su `.config`.



Consideración: certificados de seguridad

- A partir de la versión 3.7 del kernel existe la posibilidad de firmar criptográficamente los módulos del kernel.
- Esto puede fallar al utilizar un archivo .config basado en el de distribuciones como Debian GNU/Linux que asumen que habrá un par de claves generadas en un directorio específico.
- Para evitar este problema deshabilitaremos las opciones:

```
--- Enable loadable module support  
[ ]   Module signature verification  
--- Cryptographic API  
    --- Certificates for signature checking  
    [ ] Provide system-wide ring of trusted keys
```



¿Qué es un modulo del Kernel?

- Un fragmento de código que puede cargarse/descargarse en el mapa de memoria del SO (Kernel) bajo demanda
- Permiten extender la funcionalidad del Kernel en “caliente” (sin necesidad de reiniciar el sistema)
- Todo su código se ejecuta en modo Kernel (privilegiado)
- Cualquier error en el módulo, puede colgar el SO
- Permiten que el kernel se desarrolle bajo un diseño mas modular
- Los modulos disponibles se ubican en */lib/modules/version del kernel*
- Con el comando *lsmod* es posible ver qué modulos están cargados
- **Vamos a ampliar en las próximas prácticas**



¿El soporte lo damos como módulo o *built-in*?

- Si es *built-in*, el kernel crece. Más ocupación en memoria, más lento el arranque
- Si es *built-in*, es mas eficiente su utilización. No hay que cargar un adicional en memoria, acceso directo.
- En el soporte como módulo:
 - Si se quiere dar soporte a algún dispositivo, se carga el módulo y no es necesario recompilar (el soporte debe estar en el *kernel*).
 - Si hay una modificación de un *driver*, solo se modifica el módulo y no todo el código del *kernel*.
 - Los módulos se cargan bajo demanda, con lo cual la utilización de memoria es menor.



¿El soporte lo damos como módulo o *built-in*?

- Si es *built-in*, el kernel crece. Más ocupación en memoria, más lento el arranque
- Si es *built-in*, es mas eficiente su utilización. No hay que cargar un adicional en memoria, acceso directo.
- En el soporte como módulo:
 - Si se quiere dar soporte a algún dispositivo, se carga el módulo y no es necesario recompilar (el soporte debe estar en el *kernel*).
 - Si hay una modificación de un *driver*, solo se modifica el módulo y no todo el código del *kernel*.
 - Los módulos se cargan bajo demanda, con lo cual la utilización de memoria es menor.



- Es un mecanismo que permite aplicar actualizaciones NO incrementales sobre la version base
- Se basa en archivos *diff* (archivos de diferencia), que indican qué agregar y qué quitar
- Se aplican sobre la versión base
- Permiten agregar funcionalidad (nuevos drivers, correcciones menores, etc.)
- A veces puede resultar más sencillo descargar el archivo de diferencia y aplicarlo en vez de descargar todo el código de la nueva versión

```
$ cd linux; zcat ../patch-4.15.15.gz | patch  
-p1
```

Tip

Parámetro útil: `--dry-run`



- Es un mecanismo que permite aplicar actualizaciones NO incrementales sobre la version base
- Se basa en archivos *diff* (archivos de diferencia), que indican qué agregar y qué quitar
- Se aplican sobre la versión base
- Permiten agregar funcionalidad (nuevos drivers, correcciones menores, etc.)
- A veces puede resultar más sencillo descargar el archivo de diferencia y aplicarlo en vez de descargar todo el código de la nueva versión

```
$ cd linux; zcat ../patch-4.15.15.gz | patch  
-p1
```

Tip

Parámetro útil: `--dry-run`



En el trabajo práctico debemos dar el siguiente soporte adicional:

- Soporte para **FS Minix**: En la opción *File Systems* → *Miscellaneous filesystems*, tendremos que seleccionar *Minix fs support*
- Soporte para **FS ext4**: En la opción *File Systems*, tendremos que seleccionar *The Extended 4 (ext4) filesystem*
- Soporte para dispositivos de **Loopback**: En la opción *Device Drivers* → *Block Devices*, tendremos que seleccionar *Loopback device support*.



```
$ make
```

El comando `make` busca el archivo `Makefile`, interpreta sus directivas y compila el *kernel*. Este proceso puede durar mucho tiempo dependiendo del procesador que tengamos.

Tip

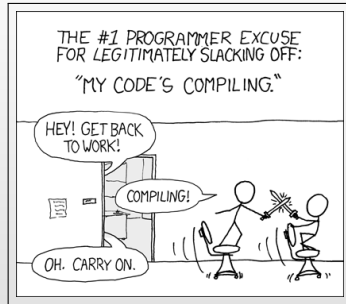
```
$ make -jX # X es el número de threads
```

Verificar que este proceso no arroje errores al concluir.



XKCD #303

<http://xkcd.com/303/>



Compilando

```
$ make modules
```

El comando `make modules` compila todos los módulos necesarios para satisfacer las opciones que hayan sido seleccionadas como módulo .

Tip

Generalmente la tarea anterior se encuentra incluida en la compilación del kernel con el comando `make`



Recién en este paso es necesario convertirse en root

Al terminar el proceso de compilación, la imagen del *kernel* quedará ubicada en

directorio-del-código/arch/arquitectura/boot/.

El próximo paso, entonces, es instalar el *kernel* y otros archivos en el directorio */boot*.

Ejemplo con arquitectura *i386*

```
$ cd $HOME/kernel/linux-4.15
$ sudo su
# cp arch/i386/boot/bzImage /boot/vmlinuz-4.15.15
# cp System.map /boot/System.map-4.15.15
# cp .config /boot/config-4.15.15
```

Por supuesto, también existe una regla en el Makefile que realiza esto mismo automáticamente.

```
$ sudo make install
```

System.map: <http://rlworkman.net/system.map/>



Recién en este paso es necesario convertirse en root

Al terminar el proceso de compilación, la imagen del *kernel* quedará ubicada en

directorio-del-código/arch/arquitectura/boot/.

El próximo paso, entonces, es instalar el *kernel* y otros archivos en el directorio */boot*.

Ejemplo con arquitectura *i386*

```
$ cd $HOME/kernel/linux-4.15
$ sudo su
# cp arch/i386/boot/bzImage /boot/vmlinuz-4.15.15
# cp System.map /boot/System.map-4.15.15
# cp .config /boot/config-4.15.15
```

Por supuesto, también existe una regla en el *Makefile* que realiza esto mismo automáticamente.

```
$ sudo make install
```

System.map: <http://rlworkman.net/system.map/>



Los módulos compilados deben residir en el directorio `/lib/modules/version-del-kernel`. Al igual que en el paso anterior, el archivo `Makefile` tiene una regla para instalar los módulos.

```
$ sudo make modules_install
```

El parámetro `modules_install` es una regla del `Makefile` que ubica los módulos del *kernel* recién compilado en el directorio correspondiente.



Un *initramfs* es un sistema de archivos temporal que se monta durante el arranque del sistema. Contiene ejecutables, *drivers* y módulos necesarios para lograr iniciar el sistema. Luego del proceso de arranque el disco se desmonta.

```
# mkinitramfs -o /boot/initrd.img-4.15.15  
4.15.15
```



Configuración del gestor de arranque

- LILO

```
/etc/lilo.conf
```

```
image=/boot/vmlinuz-4.15.15  
label=Linux-kernel-4.15.15  
read-only  
root=/dev/sda1
```

- GRUB *legacy*

```
/boot/grub/menu.lst
```

```
title Linux-kernel-4.15.15  
root (hd0,0) #Disco en el 1º IDE, 1º partición  
kernel /vmlinuz-4.15.15 root=/dev/sda1 ro  
initrd /initrd.img-4.15.15
```



En el trabajo práctico utilizaremos la versión 2 de GRUB. Luego de instalar el *kernel*, para que el gestor de arranque lo reconozca simplemente deberemos ejecutar, como usuario privilegiado, el siguiente comando:

```
# update-grub2
```



Verificar que el *kernel* esté instalado en `/boot`:

```
$ test -f /boot/vmlinuz-4.15.15 && echo ``Kernel instalado``
```

Verificar que los módulos estén instalados en `/lib/modules/4.15.15`:

```
$ test -d /lib/modules/4.15.15-arquitectura && echo ``Modulos aparentemente instalados``
```

Verificar que el gestor de arranque haya indexado el *kernel*. Si el gestor de arranque utilizado es GRUB 2, entonces el siguiente comando debería generar un par de líneas de salida.

```
$ cat /boot/grub/grub.cfg | grep --color 4.15
```

Paso final: ¡reiniciar y probar!



1 Fundamentos

2 Historia

3 Versionado

4 Compilación

5 Apéndice
Debian way



Instalar un *kernel* de forma empaquetada ofrece muchas ventajas:

- Integración con el sistema operativo
- *Bugfixes*
- Actualizaciones de **seguridad**
- LTS (*Long Term Support*)

Debian designa una versión de Linux a la cual dar soporte durante todo el ciclo de vida de su versión estable, actualmente Debian *jessie*. Esto implica portar actualizaciones críticas y *bugfixes*. La versión de Linux que está en la actual Debian estable es la 3.16.

```
# apt-get install linux-source-3.16
$ tar xjf /usr/src/linux-source-3.16.tar.bz2
$ cd linux-source-3.16
$ make menuconfig
$ scripts/config --disable DEBUG.INFO
$ make clean
$ make deb-pkg
# dpkg -i ../linux-image-3.16.*.deb
```





Linus Torvalds

Linux 3.0-rc1

<https://lkml.org/lkml/2011/5/29/204>, 2011



Andrew Tanenbaum

Minix

<http://www.minix3.org/>



Linus Torvalds

Linux History

<https://www.cs.cmu.edu/~awb/linux.history.html>, 1992



Linus Torvalds

Release notes for Linux v0.12

<http://ftp.funet.fi/pub/linux/historical/kernel/old-versions/RELNOTES-0.12>, 1992



Debian Linux Kernel Handbook

<http://kernel-handbook.alioth.debian.org/>



- Compilar el Kernel en la distribución que posean
- Una vez que el kernel haya sido compilado, bootear con ese kernel y ejecutar el siguiente comando:

```
# uname -r | md5sum
```

- Pegar el texto devuelto en la tarea que se publicará para tal fin en el sitio de la cátedra
- ¡Tienen tiempo hasta el Lunes 8 a las 15:00 horas!



¿Preguntas?

