[Kernel Compilation](http://shabbathster.blogspot.com/2013/09/kernel-compilation.html)

**Analyzing What the Kernel Is Doing**

To help analyze what the kernel is doing, some tools are provided by the Linux

operating systems:

■ The **dmesg** utility

■ The /proc file system

■ The uname utility

The first utility to consider whether detailed information about the kernel activity

is required is dmesg. This utility shows the contents of the kernel ring buffer, an

area of memory where the Linux kernel keeps its recent log messages. An alternative

method to get access to the same information in the kernel ring buffer is by using

the **journalctl --dmesg** command, which is equivalent to **journalctl -k** . In Listing

16.2 , you can see a part of the result of the **dmesg** command.

In the dmesg output, all kernel-related messages are shown. Each message starts

with a time indicator that shows at which specific second the event was logged. This

time indicator is relative to the start of the kernel, which allows you to see exactly

how many seconds have passed between the start of the kernel and a particular

event. (Notice that the **journalctl -k / --dmesg** commands show clock time, instead

of time that is relative to the start of the kernel.) This time indicator gives a clear

indication of what has been happening and at which time it has happened.

Another valuable source of information is the /proc file system. The /proc file

system is an interface to the Linux kernel, and it contains files with detailed actual

status information on what is happening on your server. Many of the performancerelated

tools mine the /proc file system for more information.

A last useful source of information that should be mentioned here is the **uname**

command. This command gives different kinds of information about your operating

system. Type, for instance, **uname -a** for an overview of all relevant parameters of

**uname -r** to see which kernel version currently is used. This information also shows

when using the **hostnamectl status** command.

A modular kernel consists of a

relatively small core kernel and provides driver support through modules that are

loaded when required. Modular kernels are very efficient, as only those modules that

really are needed are included.

**TIP** A kernel module implements specific kernel functionality. Kernel modules

are used to load drivers that allow proper communications with hardware devices,

but are not limited to loading hardware drivers alone. Other kernel features can be

loaded as modules as well.

**Understanding Hardware Initialization**

The loading of drivers is an automated process that roughly goes like this:

**1.** During boot, the kernel probes available hardware.

**2.** Upon detection of a hardware component, the **systemd-udevd** process

takes care of loading the appropriate driver and making the hardware device

available.

**3.** To decide how the devices are initialized, systemd-udevd reads rules files in /

usr/lib/udev/rules.d. These are system provided udev rules files that should

not be modified.

**4.** After processing the system provided udev rules files, systemd-udevd goes to

the /etc/udev/rules.d directory to read any custom rules if these are available.

**5.** As a result, required kernel modules are loaded automatically and status about

the kernel modules and associated hardware is written to the sysfs file system

which is mounted on the /sys directory.

The systemd-udevd process is not a one-time only process; it continuously monitors

plugging and unplugging of new hardware devices. To get an impression of how this

works, as root you can type the command **udevadm monitor** . This is all events that

are processed while activating new hardware devices. Use Ctrl+C to close the udevadm

monitor output.

the **udevadm monitor** command. In this command,

you can see how features that are offered by the hardware are discovered automatically

by the kernel and udev working together. Each phase of the hardware probing

is concluded by the creation of a file in the /sys file system. Once the hardware has

been fully initialized, you can also see that some kernel modules are loaded.

**Managing Kernel Modules**

Linux kernel modules normally are loaded automatically for the devices that need

them, but you will sometimes have to load the appropriate kernel modules manually.

**Command Use**

**lsmod** Lists currently loaded kernel modules

**modinfo** Displays information about kernel modules

**modprobe** Loads kernel modules, including all of their dependencies

**modprobe -** r Unloads kernel modules, considering kernel module dependencies

**Managing Kernel Modules from the Command Line**

In this exercise, you work with the basic commands that are used for managing Linux

kernel modules from the command line.

**1.** Open a root shell and type **lsmod | head** . This shows all kernel modules currently

loaded.

**2.** Type **modprobe ext4** to load the ext4 kernel module. Verify that it is loaded,

using the **lsmod** command again.

**3.** Type **modinfo ext4** to get information about the ext4 kernel module. Notice

that it does not have any parameters.

**4.** Type **modprobe -r ext4** to unload the ext4 kernel module again.

**5.** Type **modprobe -r xfs** to try to unload the xfs kernel module. Notice that you

get an error message as the kernel module currently is in use.

Use of Kernel Compilation  
-----------------------------

**How to: Compile Linux kernel 2.6**

by [nixCraft](http://www.cyberciti.biz/tips/about-us) on September 29, 2005 · [146 comments](http://www.cyberciti.biz/tips/compiling-linux-kernel-26.html#comments)· LAST UPDATED May 5, 2012

in [CentOS](http://www.cyberciti.biz/tips/category/centos), [Debian Linux](http://www.cyberciti.biz/tips/category/debian-linux), [GNU/Open source](http://www.cyberciti.biz/tips/category/gnuopen-source)

Compiling custom kernel has its own advantages and disadvantages. However, new Linux user / admin find it difficult to compile Linux kernel. Compiling kernel needs to understand few things and then just type couple of commands. This step by step howto covers compiling Linux kernel version 2.6.xx under Debian GNU Linux. However, instructions remains the same for any other distribution except for apt-get command.

**Step # 1 Get Latest Linux kernel code**

Visit <http://kernel.org/> and download the latest source code. File name would be linux-x.y.z.tar.bz2, where x.y.z is actual version number. For example file inux-2.6.25.tar.bz2 represents 2.6.25 kernel version. Use wget command to download kernel source code:  
$ cd /tmp  
$ wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-x.y.z.tar.bz2  
Note: Replace x.y.z with actual version number.

**Step # 2 Extract tar (.tar.bz3) file**

Type the following command:  
# tar -xjvf linux-2.6.25.tar.bz2 -C /usr/src  
# cd /usr/src

**Step # 3 Configure kernel**

Before you configure kernel make sure you have development tools (gcc compilers and related tools) are installed on your system. If gcc compiler and tools are not installed then use apt-get command under Debian Linux to install development tools.  
# apt-get install gcc   
Now you can start kernel configuration by typing any one of the command:

* **$ make menuconfig** - Text based color menus, radiolists & dialogs. This option also useful on remote server if you wanna compile kernel remotely.
* **$ make xconfig** - X windows (Qt) based configuration tool, works best under KDE desktop
* **$ make gconfig** - X windows (Gtk) based configuration tool, works best under Gnome Dekstop.

For example make menuconfig command launches following screen:  
$ make menuconfig  
You have to select different options as per your need. Each configuration option has HELP button associated with it so select help button to get help.

**Step # 4 Compile kernel**

Start compiling to create a compressed kernel image, enter:  
$ make  
Start compiling to kernel modules:  
$ make modules   
Install kernel modules (become a root user, use su command):  
$ su -  
# make modules\_install

**Step # 5 Install kernel**

So far we have compiled kernel and installed kernel modules. It is time to install kernel itself.  
# make install   
It will install three files into /boot directory as well as modification to your kernel grub configuration file:

* System.map-2.6.25
* config-2.6.25
* vmlinuz-2.6.25

**Step # 6: Create an initrd image**

Type the following command at a shell prompt:  
# cd /boot  
# mkinitrd -o initrd.img-2.6.25 2.6.25  
initrd images contains device driver which needed to load rest of the operating system later on. Not all computer requires initrd, but it is safe to create one.

**Step # 7 Modify Grub configuration file - /boot/grub/menu.lst**

Open file using vi:  
# vi /boot/grub/menu.lst

title Debian GNU/Linux, kernel 2.6.25 Default

root (hd0,0)

kernel /boot/vmlinuz root=/dev/hdb1 ro

initrd /boot/initrd.img-2.6.25

savedefault

boot

Remember to setup correct root=/dev/hdXX device. Save and close the file. If you think editing and writing all lines by hand is too much for you, try out update-grub command to update the lines for each kernel in /boot/grub/menu.lst file. Just type the command:  
# update-grub  
Neat. Huh?

**Step # 8 : Reboot computer and boot into your new kernel**

Just issue reboot command:  
# reboot

##############################################################################  
To remove the unwanted modules we are using kernel compilation  
  
---------------------------------------------------------------------------------------------------  
  
  
  
-:Steps to compile kernel 2.6 :-  
------------------------------------------------  
  
  
1> copy kernel tarball file ino /usr/src/kernels/ location & untar into that location  
     cp linux-2.6.29.3.tar.bz2 /usr/src/kernels/  
      
     cd /usr/src/kernels/  
    
2>tar -jxvf /usr/src/kernels/linux-2.6.29.3.tar.bz2  
  
3> rpm -qa | grep gcc\*  
    
   cd linux-2.6.29.3  
  
4>make gconfig (graphical)  
  make menuconfig (text)    ( To select the modules and all)  
  
#5> make clean  
6> make bzImage  ( It will map the image off kernel after refering/reading from the config file)  
  
Note:- bzImaze is located in /usr/src/kernels/linux-2.6.29.3/arch/x86/boot/bzImage  
  
7> make modules ( used to create modules of yhe above written kernel image)  
  
8> make modules\_install  ( To insatll modules)  
  
9>cp arch/x86/boot/bzImage /boot/vmlinuz-2.6.29.3  
  
10>cp /usr/src/kernels/linux-2.6.29.3/System.map /boot/System.map-2.6.29.3  
  
11>ln -s /boot/System.map-2.6.29.3 /boot/System.map  
  
12> Create initrd :--  
  
#first check into /lib/modules/2.6.29.3 --> this is created or not  
then execute next command  
  
cd /lib/modules  
  
mkinitrd /boot/initrd-2.6.29.3.img 2.6.29.3  
  
Final Steps  
  
#vi /etc/grub.conf  
  
default=0  
timeout=77  
splashimage=(hd0,0)/grub/splash.xpm.gz  
title Red Hat Enterprise Linux AS (2.6.9-34.EL)  
        root (hd0,0)  
        kernel /vmlinuz-2.6.9-34.EL ro root=LABEL=/ rhgb quiet  
        initrd /initrd-2.6.9-34.EL.img  
  
title Red Hat Enterprise Linux AS (2.6.29.3)  
        root (hd0,0)  
        kernel /vmlinuz-2.6.29.3 ro root=LABEL=/ rhgb quiet  
        initrd /initrd-2.6.29.3.img  
  
reboot ur m/c