```
Non c'entra niente ma sfrutto l'effetto sorpresa che vi fa bene:
      ricordatevi di indentare i vostri sorgenti C
      potete farlo utilizzando il comando indent e chiedendo di utilizzare
      l'indentazione tipica di kernigham & Ritchie
      e delle tabulazioni costituite da 8 bianchi (forse meglio 4)
            indent -kr -i8
```

Compile Linux Kernel 4.2.6 on Light Ubuntu 15.04

This tutorial will outline the process to compile your own kernel for Ubuntu. It will demonstrate both the traditional process using 'make' and 'make install' as well as the Debian method, using 'make-dpkg'. This is the updated version, tested with the GNU Linux distribution LUbuntu 15.04, kernel version 4.2.6 (starting with kernel version 3.19.0-39), of the excellent tutorial (Compile Linux Kernel on Ubuntu 12.04 LTS).

In any case, we begin by installing some dependencies:

sudo apt-get update

sudo apt-get install git-core libncurses5 libncurses5-dev libelf-dev asciidoc binutils-dev linux-source fakeroot build-essential crash kexec-tools makedumpfile kernel-wedge kernel-package

Note: qt3-dev-tools and libqt3-mt-dev is necessary if you plan to use 'make xconfig' and libncurses5 and libncurses5-dev if you plan to use 'make menuconfig'. Next, copy the kernel sources with wget:

cd /usr/src

sudo wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.2.6.tar.xz

Extract the archive and change into the kernel directory:

sudo tar -xJvf linux-4.2.6.tar.xz && cd linux-4.2.6/

Now you are in the top directory of a kernel source tree. The kernel comes in a default configuration, determined by the people who put together the kernel source code distribution. It will include support for nearly everything, since it is intended for general use, and is huge. In this form it will take a very long time to compile and a long time to load. So, before building the kernel, you must configure it. If you wish to re-use the configuration of your currently-running kernel, start by copying the current config contained in /boot:

sudo cp -vi /boot/config-`uname -r` .config

Note that only debian and ubuntu linux distributions save the .config file in the /boot directory. The other distributions do not. In any distribution, we suggest to "Enable access to .config through /proc/config.gz" by enabling the "Kernel ,config support" **feature.**

Parse the .config file using make with the oldconfig flag. If there are new options available in the downloaded kernel tree, you may be prompted to make a selection to include them or not. If unsure, press enter to accept the defaults.

```
sudo make oldconfig
```

~~Since the 2.6.32 kernel, a new feature allows you to update the configuration to only compile modules that are actually used in your system. As above, make selections if prompted, otherwise hit enter for the defaults.~~

```
~~make localmodconfig~~
```

The next step is to configure the kernel to your needs. You can configure the build with ncurses using the 'menuconfig' flag:

```
sudo make menuconfig
```

~~or, using a GUI with the 'xconfig' flag:~~

```
~~make xconfig~~
```

In either case, you will be presented with a series of menus, from which you will choose the options you want to include. For most options you have three choices: (blank) leave it out; (M) compile it as a module, which will only be loaded if the feature is needed; (*) compile it into monolithically into the kernel, so it will always be there from the time the kernel first loads.

There are several things you might want to accomplish with your reconfiguration:

- Reduce the size of the kernel, by leaving out unnecessary components. This is helpful for kernel development. A small kernel will take a lot less time to compile and less time to load. It will also leave more memory for you to use, resulting in less page swapping and faster compilations.
- Retain the modules necessary to use the hardware installed on your system. To do this without including just about everything conceivable, you need figure out what hardware is installed on your system. You can find out about that in several ways.

Before you go too far, use the "General Setup" menu and the "Local version" and "Automatically append version info" options to add a suffix to the name of your kernel, so that you can distinguish it from the "vanilla" one. You may want to vary the local version string, for different configurations that you try, to distinguish them also.

Moreover, use the “General Setup” menu and select static * “Kernel ,config support” and select static * "Enable access to .config through /proc/config.gz" to allow the complete Linux kernel ".config" file will be available from a running kernel by reading the /proc/config.gz

Moreover, be sure to allow "NETLINK: mmaped IO" in the "Networking Options" of the "Networking support" menu.

Moreover, in "File systems" enter in "Pseudo filesystems" and select "/proc file system support".

Assuming you have a running Linux system with a working kernel, there are several places you can look for information about what devices you have, and what drivers are running.

- Look at the system log file, /var/log/messages or use the command dmesg to see the messages printed out by the device drivers as they came up.
- Use the command lspci -vv to list out the hardware devices that use the PCI bus.
- Use the command lsub -vv to list out the hardware devices that use the USB.
- Use the command lsmod to see which kernel modules are in use.
- Look at /proc/modules to see another view of the modules that are in use.
- Look at /proc/devices to see devices the system has recognized.
- Look at /proc/cpuinfo to see what kind of CPU you have.
- Open up the computer’s case and read the labels on the components.
- Check the hardware documentation for your system. If you know the motherboard, you should be able to look up the manual, which will tell you about the on-board devices.

Using the available information and common sense, select a reasonable set of kernel configuration options. Along the way, read through the on-line help descriptions (for at least all the top-level menu options) so that you become familiar with the range of drivers and software components in the Linux kernel.

Before exiting the final menu level and saving the configuration, it is a good idea to save it to a named file, using the “Save Configuration to an Alternate File” option. By saving different configurations under different names you can reload a configuration without going through all the menu options again. Alternatively, you can backup the file (which is named .config manually, by making a copy with an appropriate name.

One way to reduce frustration in the kernel trimming process (which involves quite a bit of guesswork, trial, and error) is to start with a kernel that works, trim just a little at a time, and test at each stage, saving copies of the .config file along the way so that you can back up when you run into trouble. However, the first few steps of this process will take a long time since you will be compiling a kernel with huge number of modules, nearly all of which you do not need. So, you may be tempted to try eliminating a large number of options from the start

Now we are ready to start the build. You can speed up the compilation process by enabling parallel make with the -j flag. The recommended use is ‘processor cores + 1′, e.g. 5 if you have a quad core processor:

sudo make -j5

NB: this step includes compiling of multiple targets, such as

make depend
make bzImage
make modules

This will compile the kernel and create a compressed binary image of the kernel. After the first step, the kernel image can be found at arch/i386/boot/bzImage (for a x86 based processor). Once the initial compilation has completed, install the dynamically loadable kernel modules:

sudo make modules_install

The modules are installed in a subdirectory of "**/lib/modules**", named after the kernel version. The resulting modules have the suffix ".ko". For example, if you chose to compile the network device driver for the Realtek 8139 card as a module, there will be a kernel module name 8139too.ko. The third command is OS specific and will copy the new kernel into the directory "/boot" and update the Grub bootstrap loader configuration file "/boot/grub/grub.cfg" to include a line for the new kernel.

Finally, install the kernel:

sudo make install

This command performs many operations behind the scenes. Examine the /etc/grub.d/ directory structure before and after you run the above commands to see the changes. Also look in the /boot/grub/grub.cfg file for your kernel entry.

The file /boot/grub/grub.cfg contains the configuration of the boot loader grub. It is automatically generated by 'grub-mkconfig', using templates from the directory /etc/grub.d and settings from the file /etc/default/grub. If you change the file /etc/default/grub you need to run 'update-grub' afterwards to update /boot/grub/grub.cfg.

In particular, in order to allow the user may select manually the desired operating system to boot, it is necessary to configure the following options in /etc/default/grub

GRUB_TIMEOUT_STYLE=menu	allows the user may select the o.s
# GRUB_HIDDEN_TIMEOUT=0	starts immediately the default o.s
# GRUB_HIDDEN_TIMEOUT_QUIET=false	
GRUB_TIMEOUT=10	after 10 secs starts the default o.s.

The OS specific make install, Ubuntu in this case, also creates an initrd image in the /boot directory. If you compiled the needed drives into the kernel then you will not need this ramdisk

file to aid in booting. For extra credit remove the created initrd from the /boot/ directory as well as the references in /etc/grub.d/*.

If there are error messages from any of the make stages, you may be able to solve them by going back and playing with the configuration options. some options require other options or cannot be used in conjunction with some other options. These dependencies and conflicts may not all be accounted-for in the configuration script. If you run into this sort of problem, you are reduced to guesswork based on the compilation or linkage error messages. For example, if the linker complains about a missing definition of some symbol in some module, you might either turn on an option that seems likely to provide a definition for the missing symbol, or turn off the option that made reference to the symbol.

Reboot the system, selecting your new kernel from the boot loader menu. Watch the messages. See if it works. If it does not, reboot with the old kernel, try to fix what went wrong, and repeat until you have a working new kernel

> NOTE that, in order to remove a kernel installed with “make install”, you need to manually remove the following entries:
> ```
> /boot/vmlinuz*KERNEL-VERSION*
> /boot/initrd*KERNEL-VERSION*
> /boot/System-map*KERNEL-VERSION*
> /boot/config-*KERNEL-VERSION*
> /lib/modules/*KERNEL-VERSION*/
> ```
> Then update the grub configuration:
> ```
> sudo update-grub2
> ```