By Falko Timme
Published: 2006-11-05 17:01

How To Compile A Kernel - The Ubuntu Way

Version 1.0
Author: Falko Timme <ft [at] falkotimme [dot] com>
Last edited 11/05/2006

Each distribution has some specific tools to build a custom kernel from the sources. This article is about compiling a kernel on Ubuntu systems. It describes how to build a custom kernel using the latest unmodified kernel sources from **www.kernel.org** (**vanilla kernel**) so that you are independent from the kernels supplied by your distribution. It also shows how to patch the kernel sources if you need features that are not in there.

I have tested this on Ubuntu 6.10 Server ("Edgy Eft") and Ubuntu 6.06 Desktop ("Dapper Drake").

I want to say first that this is not the only way of setting up such a system. There are many ways of achieving this goal but this is the way I take. I do not issue any guarantee that this will work for you!

1 Preliminary Note

I prefer to do all the steps here as the *root* user. So if you haven't already created a root login, you should do so now:

```
sudo passwd root
```

Afterwards, log in as root:

```
su
```

If you would like to work as a normal user instead of root, remember to put *sudo* in front of all the commands shown in this tutorial. So when I run

```
apt-get update
```

you should run

```
sudo apt-get update
```

instead, etc.

1.1 /bin/sh on Ubuntu 6.10 ("Edgy Eft")

On Ubuntu 6.10, `/bin/sh` is a symlink to `/bin/dash` by default. `/bin/dash` seems to make problems when you compile software from the sources, at least I had that impression. That's why I make `/bin/sh` a symlink to `/bin/bash` instead.

If you are on Ubuntu 6.10, you should do this now:

```
rm -f /bin/sh

ln -s /bin/bash /bin/sh
```

2 Install Required Packages For Kernel Compilation

First we update our package database:

```
apt-get update
```

Then we install all needed packages like this:

```
apt-get install kernel-package libncurses5-dev fakeroot wget bzip2
```

3 Download The Kernel Sources

Next we download our desired kernel to */usr/src*. Go to **www.kernel.org** and select the kernel you want to install, e.g. *linux-2.6.18.1.tar.bz2* (you can find all 2.6 kernels here: **http://www.kernel.org/pub/linux/kernel/v2.6/**). Then you can download it to */usr/src* like this:

```
cd /usr/src

wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.18.1.tar.bz2
```

Then we unpack the kernel sources and create a symlink linux to the kernel sources directory:

```
tar xjf linux-2.6.18.1.tar.bz2

ln -s linux-2.6.18.1 linux

cd /usr/src/linux
```

4 Apply Patches To The Kernel Sources (Optional)

Sometimes you need drivers for hardware that isn't supported by the new kernel by default, or you need support for virtualization techniques or some other bleeding-edge technology that hasn't made it to the kernel yet. In all these cases you have to patch the kernel sources (provided there is a patch available...).

Now let's assume you have downloaded the needed patch (I call it *patch.bz2* in this example) to */usr/src*. This is how you apply it to your kernel sources (you must still be in the */usr/src/linux* directory):

```
bzip2 -dc /usr/src/patch.bz2 | patch -p1 --dry-run

bzip2 -dc /usr/src/patch.bz2 | patch -p1
```

The first command is just a test, it does nothing to your sources. If it doesn't show errors, you can run the second command which actually applies the patch.

Don't do it if the first command shows errors!

You can also apply kernel prepatches to your kernel sources. For example, if you need a feature that is available only in kernel 2.6.19-rc4, but the full sources haven't been released yet for this kernel. Instead, a *patch-2.6.19-rc4.bz2* is available. You can apply that patch to the 2.6.18 kernel sources, but not to kernel 2.6.18.1 or 2.6.18.2, etc. This is explained on **http://kernel.org/patchtypes/pre.html**:

Prepatches are the equivalent to alpha releases for Linux; they live in the testing directories in the archives. They should be applied using the patch(1) utility to the source code of the previous full release with a 3-part version number (for example, the 2.6.12-rc4 prepatch should be applied to the 2.6.11 kernel sources, not, for example, 2.6.11.10.)

So if you want to compile a 2.6.19-rc4 kernel, you must download the 2.6.18 kernel sources (***http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.18.tar.bz2***) in step 3 instead of kernel 2.6.18.1!

This is how you apply the 2.6.19-rc4 patch to kernel 2.6.18:

```
cd /usr/src

wget http://www.kernel.org/pub/linux/kernel/v2.6/testing/patch-2.6.19-rc4.bz2

cd /usr/src/linux

bzip2 -dc /usr/src/patch-2.6.19-rc4.bz2 | patch -p1 --dry-run

bzip2 -dc /usr/src/patch-2.6.19-rc4.bz2 | patch -p1
```

5 Configure The Kernel

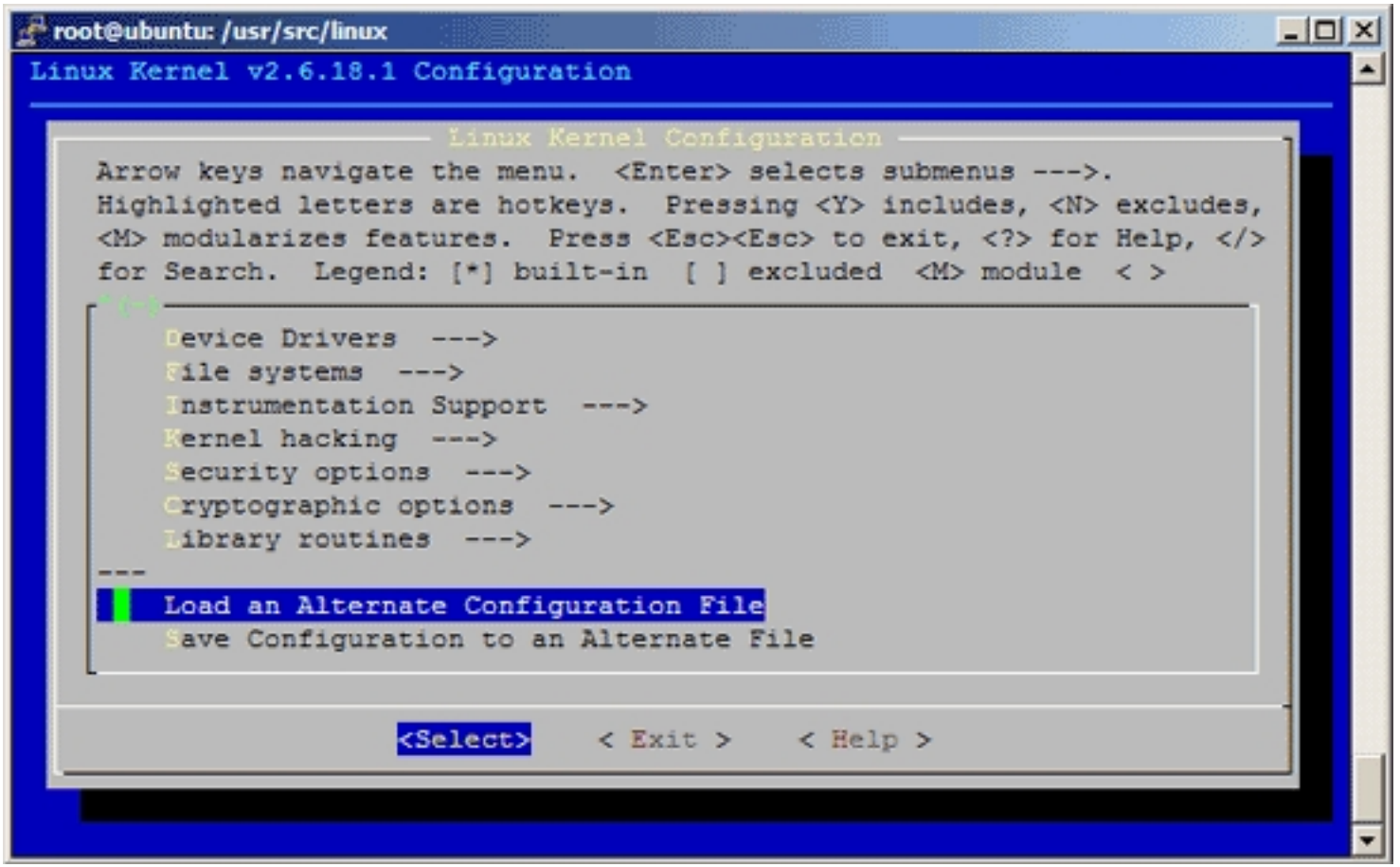
It's a good idea to use the configuration of your current working kernel as a basis for your new kernel. Therefore we copy the existing configuration to */usr/src/linux*:

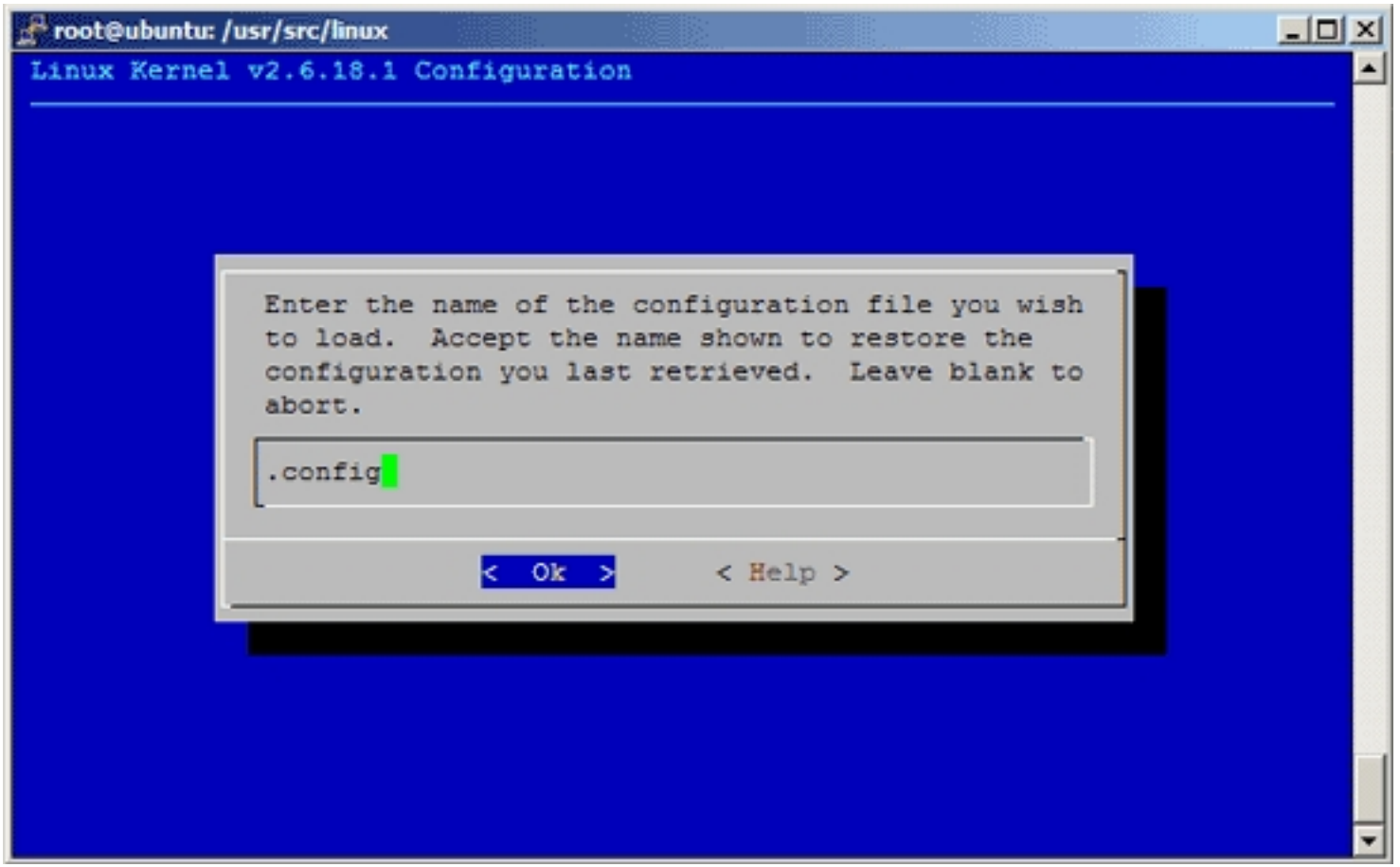
```
cp /boot/config-`uname -r` ./.config
```

Then we run

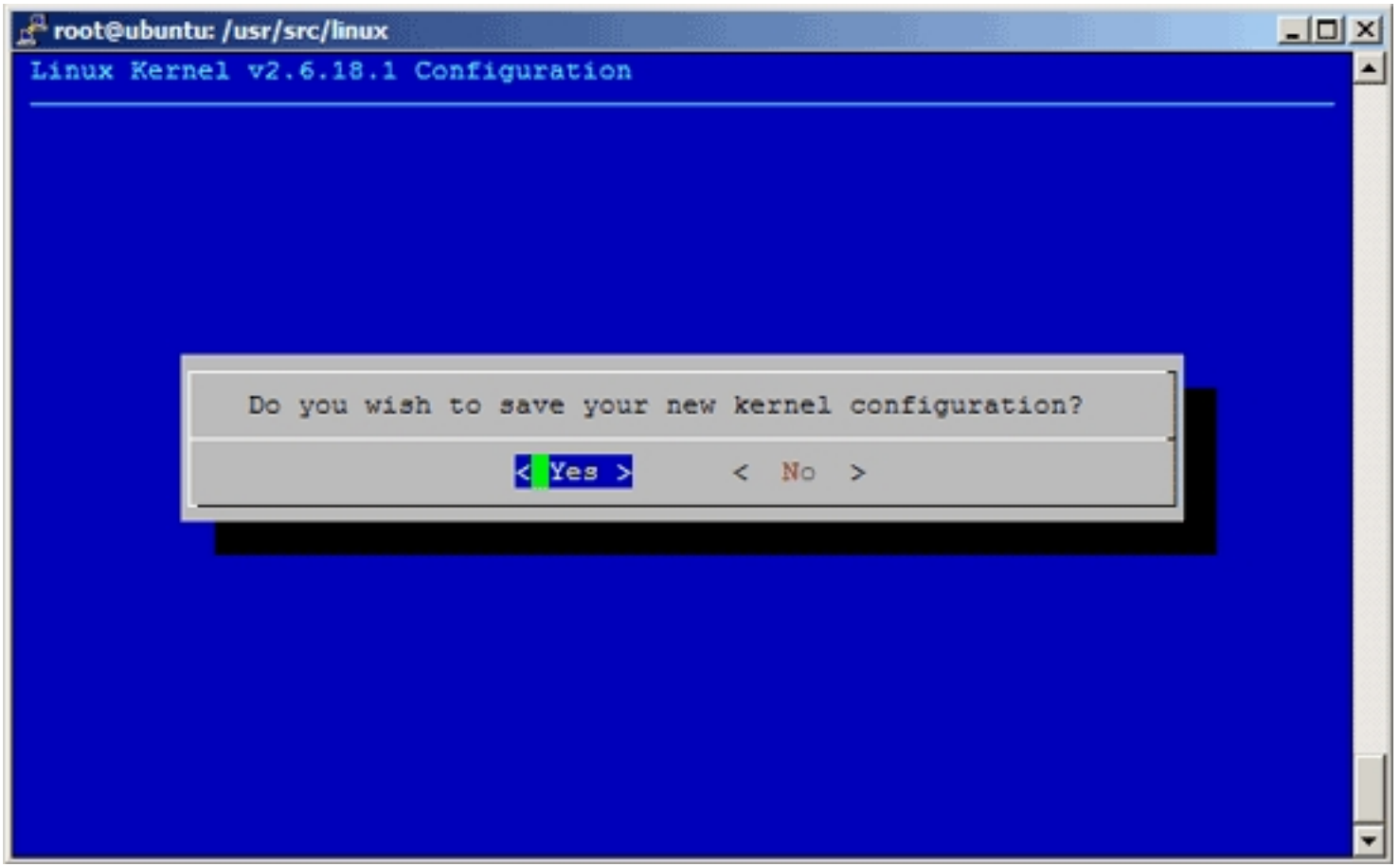
```
make menuconfig
```

which brings up the kernel configuration menu. Go to *Load an Alternate Configuration File* and choose *.config* (which contains the configuration of your current working kernel) as the configuration file:





Then browse through the kernel configuration menu and make your choices. When you are finished and select `Exit`, answer the following question (`Do you wish to save your new kernel configuration?`) with `Yes`:



6 Build The Kernel

To build the kernel, execute these two commands:

```
make-kpkg clean

fakeroot make-kpkg --initrd --append-to-version=-custom kernel_image kernel_headers
```

After *--append-to-version=* you can write any string that helps you identify the kernel, but it must begin with a minus (-) and must not contain whitespace.

Now be patient, the kernel compilation can take some hours, depending on your kernel configuration and your processor speed.

7 Install The New Kernel

After the successful kernel build, you can find two .deb packages in the */usr/src* directory.

```
cd /usr/src

ls -l
```

On my test system they were called *linux-image-2.6.18.1-custom_2.6.18.1-custom-10.00.Custom_i386.deb* (which contains the actual kernel) and *linux-headers-2.6.18.1-custom_2.6.18.1-custom-10.00.Custom_i386.deb* (which contains files needed if you want to compile additional kernel modules later on). I install them like this:

```
dpkg -i linux-image-2.6.18.1-custom_2.6.18.1-custom-10.00.Custom_i386.deb

dpkg -i linux-headers-2.6.18.1-custom_2.6.18.1-custom-10.00.Custom_i386.deb
```

(You can now even transfer the two .deb files to other Ubuntu systems and install them there exactly the same way, which means you don't have to compile the kernel there again.)

That's it. You can check */boot/grub/menu.lst* now, you should find two stanzas for your new kernel there:

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

The stanzas that were added on my test system look like these:

```
title          Ubuntu, kernel 2.6.18.1-custom
root           (hd0,0)
kernel         /boot/vmlinuz-2.6.18.1-custom root=/dev/sda1 ro quiet splash
initrd         /boot/initrd.img-2.6.18.1-custom
savedefault
boot

title          Ubuntu, kernel 2.6.18.1-custom (recovery mode)
root           (hd0,0)
kernel         /boot/vmlinuz-2.6.18.1-custom root=/dev/sda1 ro single
initrd         /boot/initrd.img-2.6.18.1-custom
boot
```

Now reboot the system:

```
shutdown -r now
```

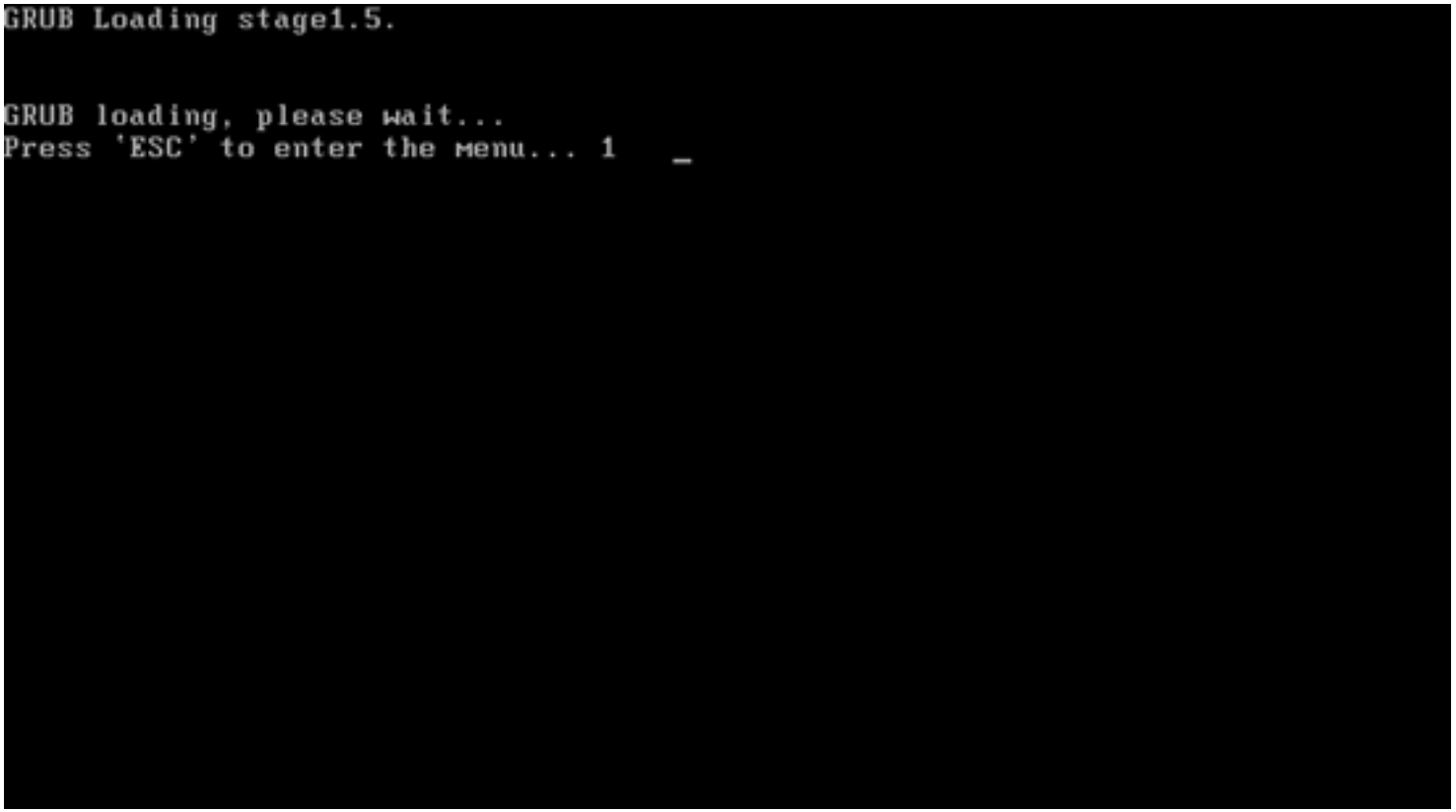
If everything goes well, it should come up with the new kernel. You can check if it's really using your new kernel by running

```
uname -r
```

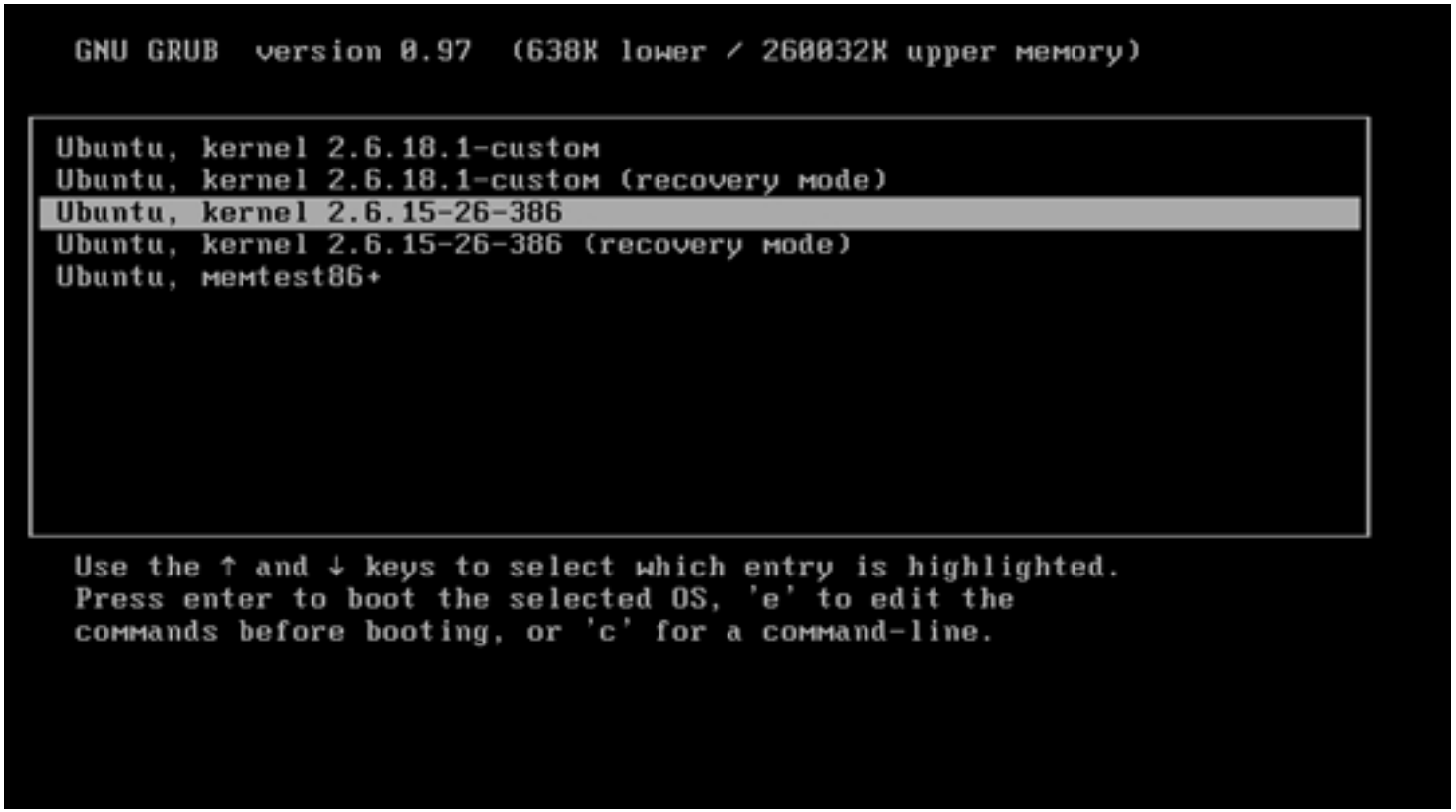
This should display something like

```
2.6.18.1-custom
```

If the system doesn't start, restart it, and when you see this:



press *ESC* to enter the GRUB menu:



Select your old kernel and start the system. You can now try again to compile a working kernel. Don't forget to remove the two stanzas of the not-working kernel from `/boot/grub/menu.lst`.

8 Links

- Ubuntu: ***http://www.ubuntu.com***
- The Linux Kernel Archives: ***http://www.kernel.org***