By Falko Timme
Published: 2006-11-26 16:14

How To Compile A Kernel - The CentOS Way

Version 1.0
Author: Falko Timme <ft [at] falkotimme [dot] com>
Last edited 11/23/2006

Each distribution has some specific tools to build a custom kernel from the sources. This article is about compiling a kernel on CentOS 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 CentOS 4.4.

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

In this article I will describe two ways of building a kernel for CentOS systems. The first one will get you a kernel rpm package that you can install or share with others. The second way is the same for all Linux distributions, but you don't end up with an rpm package.

2 Building A Kernel rpm Package

This chapter shows how to build a kernel and end up with an rpm package that you can install and share with others.

2.1 Download The Kernel Sources

First we download our desired kernel to `/usr/src`. Go to **www.kernel.org** and select the kernel you want to install, e.g. `linux-2.6.18.3.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-2.6.18.3.tar.bz2
```

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

```
tar xjf linux-2.6.18.3.tar.bz2

ln -s linux-2.6.18.3 linux

cd /usr/src/linux
```

2.2 Modify /etc/modprobe.conf

Now we must comment out the mptscsi module in */etc/modprobe.conf* because otherwise we will get an error like this:

No module mptscsi found for kernel 2.6.18.3-default, aborting.

when we try to create a ramdisk for our new kernel.

```
vi /etc/modprobe.conf
```

```
alias eth0 pcnet32
alias scsi_hostadapter mptbase
# alias scsi_hostadapter1 mptscsi
alias scsi_hostadapter2 mptfc
alias scsi_hostadapter3 mptspi
alias scsi_hostadapter4 mptsas
alias scsi_hostadapter5 mptscsih
```

2.3 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!

If your patches are compressed with gzip (*.gz*) instead of bzip2 (*.bz2*), then you patch your kernel as follows:

```
gunzip -c /usr/src/patch.gz | patch -p1 --dry-run

gunzip -c /usr/src/patch.gz | patch -p1
```

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-rc6, but the full sources haven't been released yet for this kernel. Instead, a *patch-2.6.19-rc6.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 or 2.6.18.3, 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-rc6 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 2.6.18.3!

This is how you apply the 2.6.19-rc6 patch to kernel 2.6.18:

```
cd /usr/src

wget http://www.kernel.org/pub/linux/kernel/v2.6/testing/patch-2.6.19-rc6.bz2

cd /usr/src/linux

bzip2 -dc /usr/src/patch-2.6.19-rc6.bz2 | patch -p1 --dry-run

bzip2 -dc /usr/src/patch-2.6.19-rc6.bz2 | patch -p1
```

2.4 Configure The Kernel

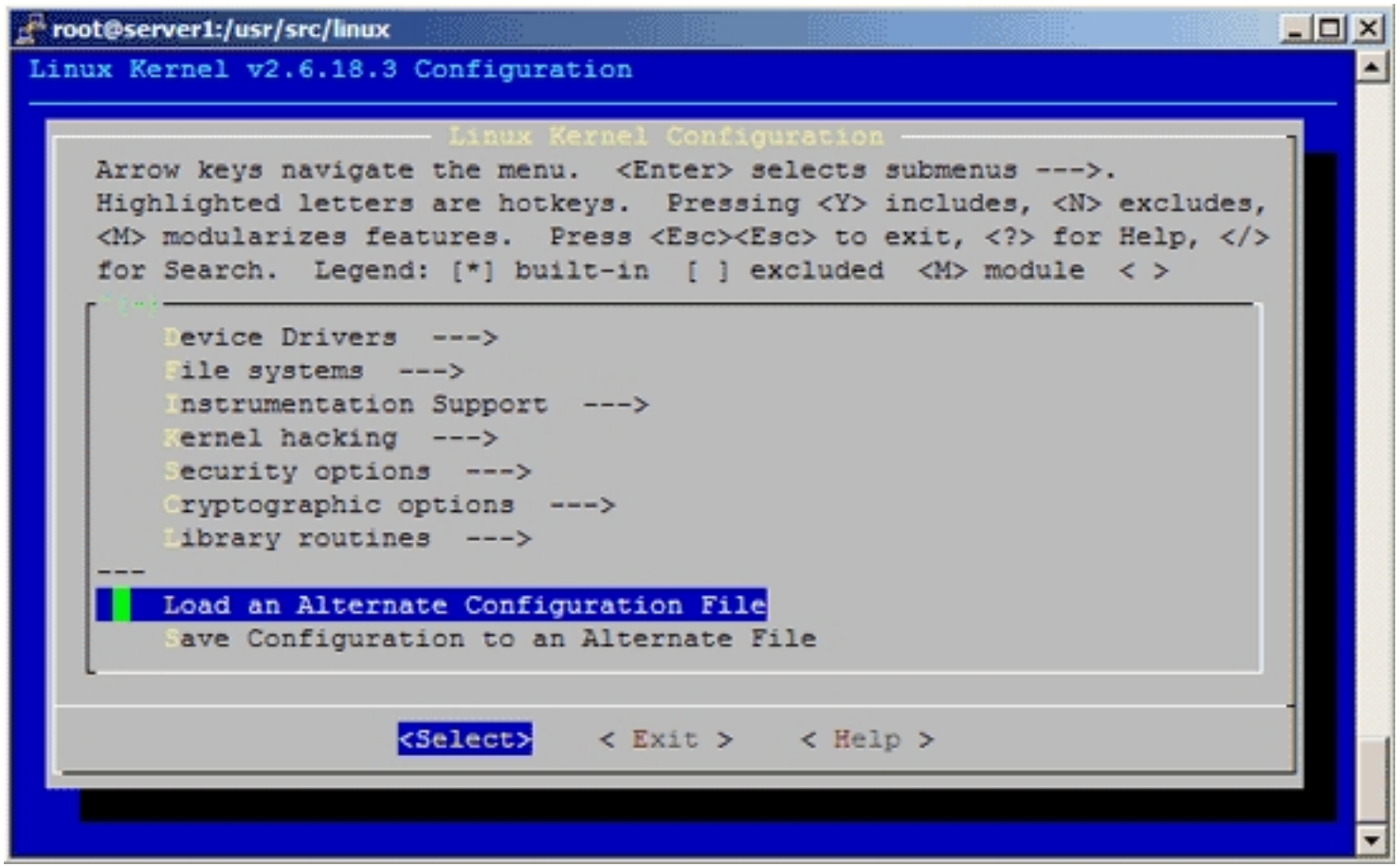
It's a good idea to use the configuration of your current working kernel as a basis for your new kernel. Therefore we copy the existing configuration to `/usr/src/linux`:

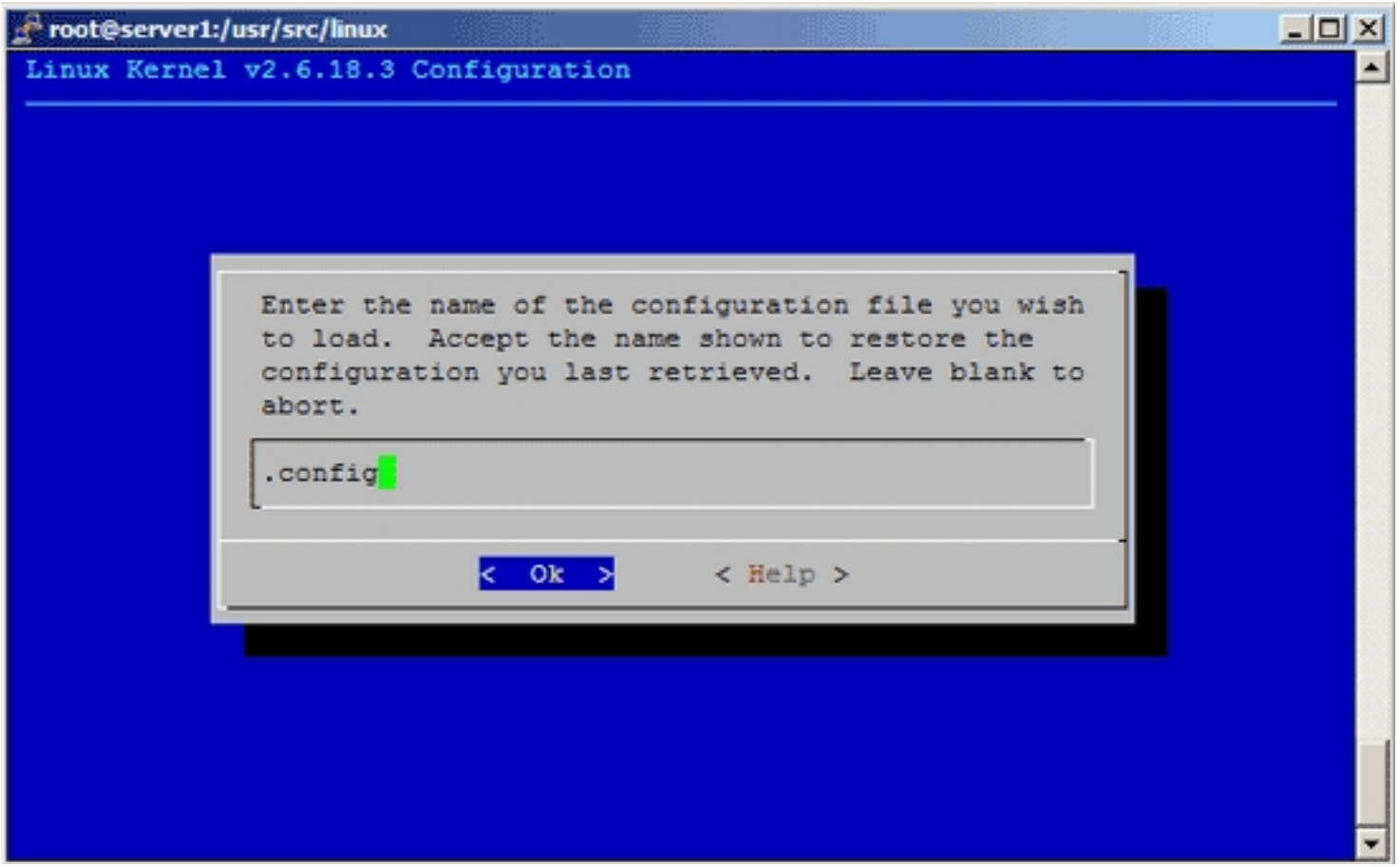
```
make clean && make mrproper

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

Then we run

```
make menuconfig
```

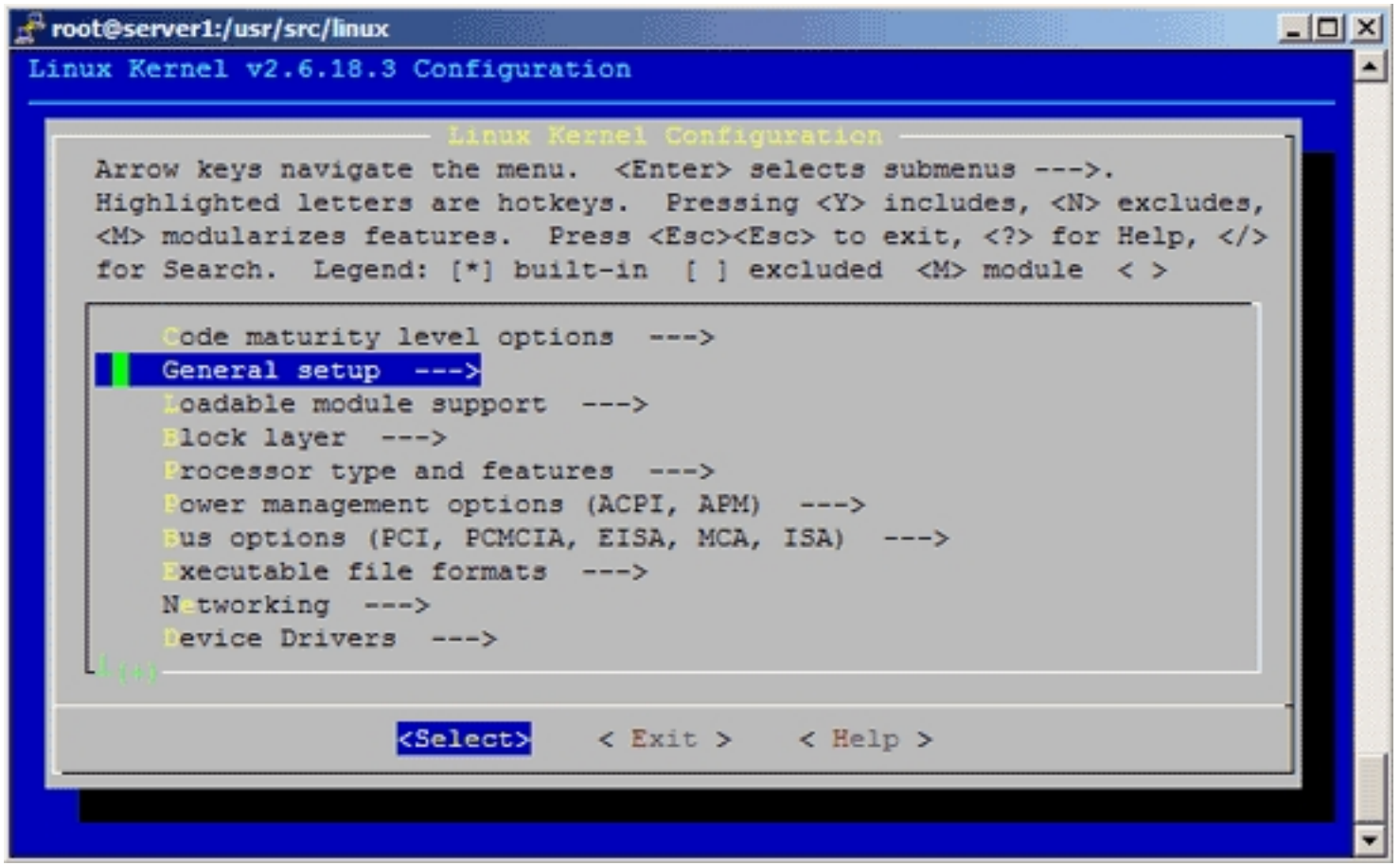
which brings up the kernel configuration menu. Go to `Load an Alternate Configuration File` and choose `.config` (which contains the configuration of your current working kernel) as the configuration file:

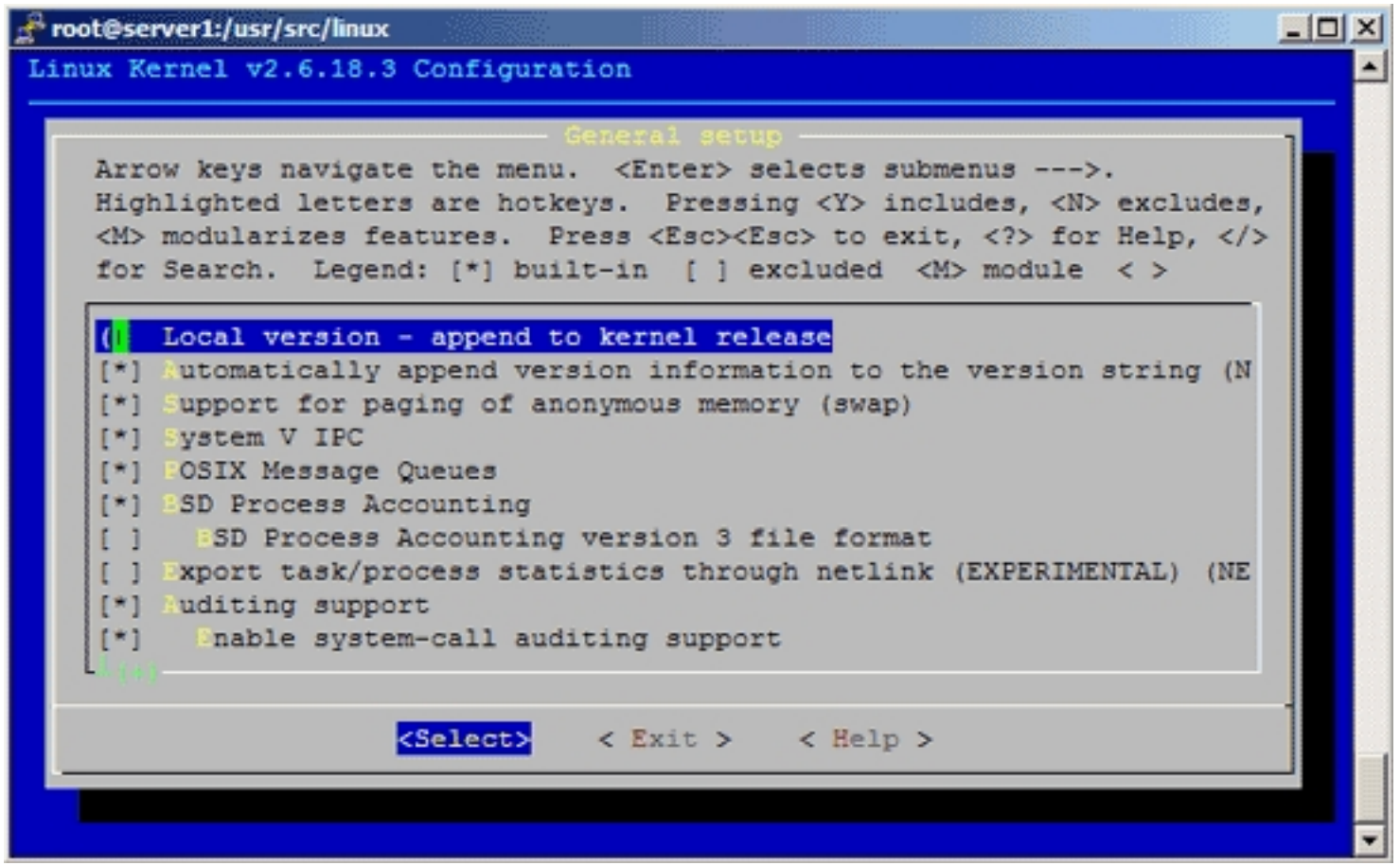


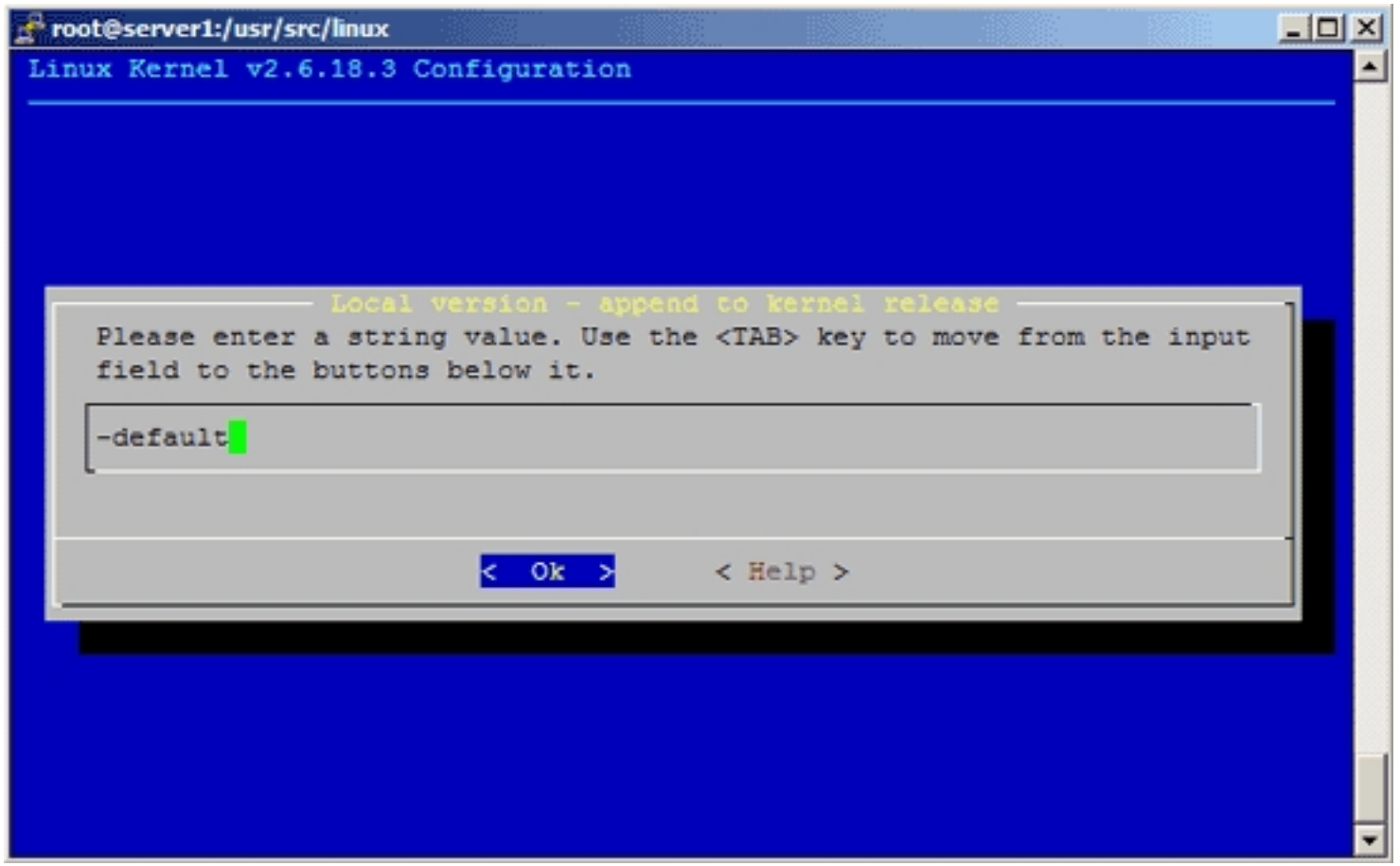


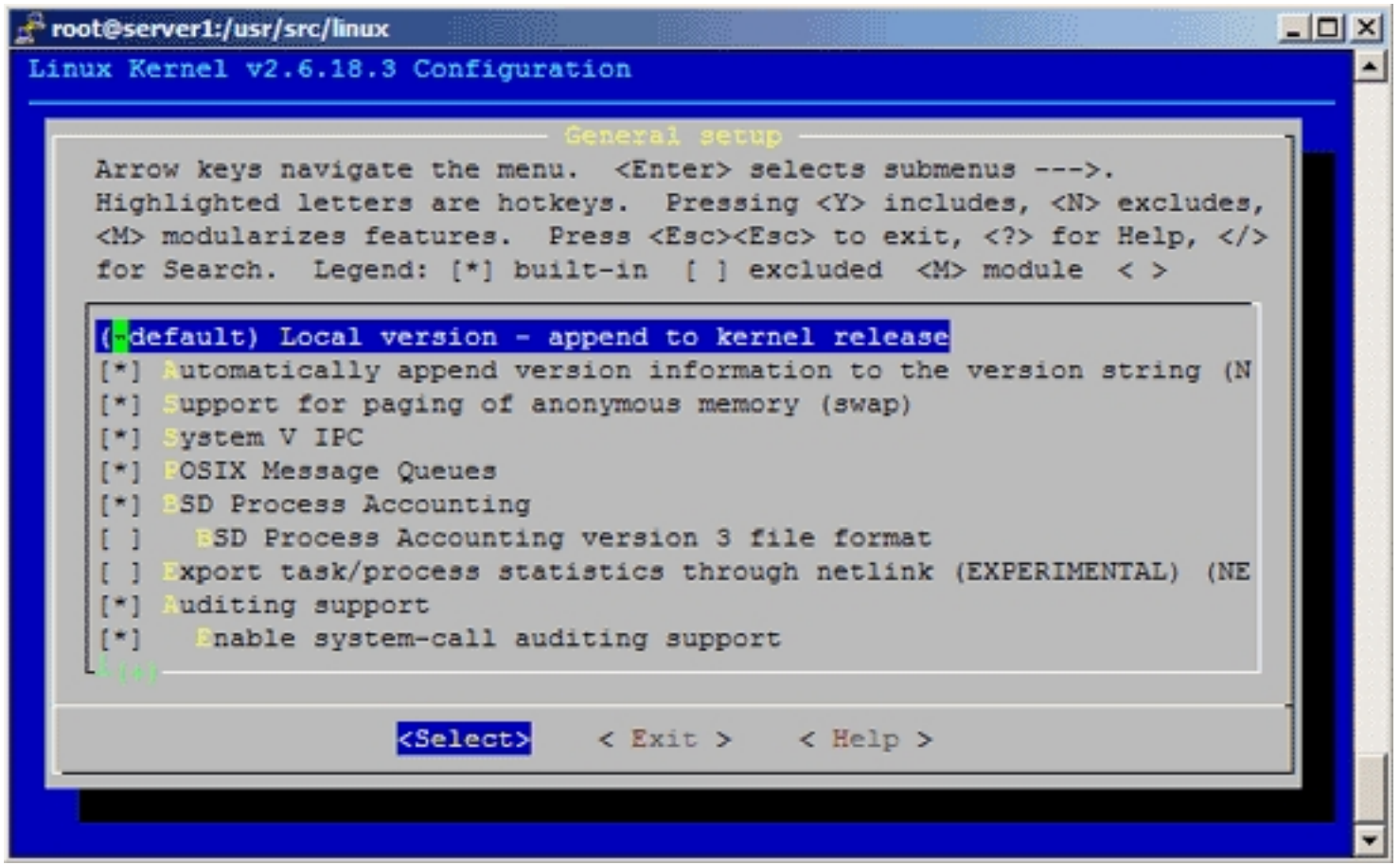
Then browse through the kernel configuration menu and make your choices. Make sure you specify a kernel version identification string under *General Setup ---> () Local version - append to kernel release*. I use *-default* so our kernel rpm package will be named *kernel-2.6.18.3default-1.i386.rpm*. You can leave the string empty or specify a different one which helps you identify the kernel (e.g. *-custom* or whatever you like).

After you have installed *kernel-2.6.18.3default-1.i386.rpm* and decide to compile another 2.6.18.3 kernel rpm package, it is important to use a different version string, e.g. *-default1*, *-default2*, etc., because otherwise you can't install your new kernel because rpm complains that *kernel-2.6.18.3default-1.i386.rpm* is already installed!

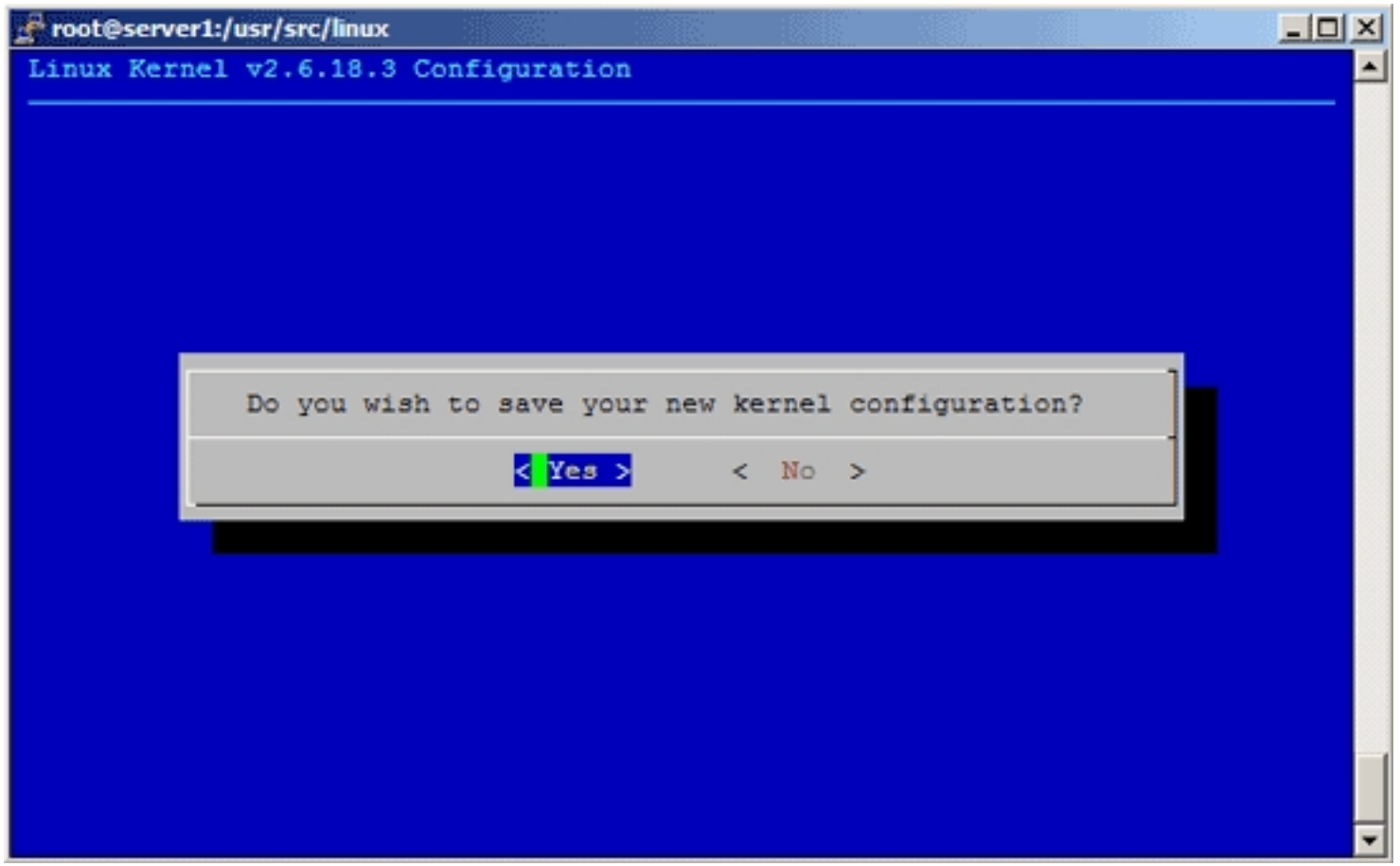








When you are finished and select *Exit*, answer the following question (*Do you wish to save your new kernel configuration?*) with *Yes*:



2.5 Build The Kernel

To build the kernel, simply execute this command:

```
make rpm
```

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

2.6 Install The New Kernel

After the successful kernel build, a *src.rpm* and an *rpm* package have been created. The *src.rpm* package can be found in the */usr/src/redhat/SRPMS/* directory, you can find out about its name by running

```
ls -l /usr/src/redhat/SRPMS/
```

On my system it was called *kernel-2.6.18.3default-1.src.rpm*.

The rpm package can be found in */usr/src/redhat/RPMS/i386/*, */usr/src/redhat/RPMS/i586/*, */usr/src/redhat/RPMS/i686/*, */usr/src/redhat/RPMS/x86_64/*, etc., depending on your architecture. On my system it was located in */usr/src/redhat/RPMS/i386/*, and by running

```
ls -l /usr/src/redhat/RPMS/i386/
```

I found out that its name was *kernel-2.6.18.3default-1.i386.rpm*.

Now we can install our kernel rpm package like this:

```
cd /usr/src/redhat/RPMS/i386/

rpm -ivh --nodeps kernel-2.6.18.3default-1.i386.rpm
```

Please note the *--nodeps* switch: if you don't use it, you will see an error like this:

```
error: Failed dependencies:
        kernel >= 2.6.10 conflicts with lksctp-tools-1.0.2-6.4E.1.i386
```

I found that ignoring this dependency didn't cause any problems on my system.

You can now even transfer the kernel rpm package to other CentOS systems and install it there exactly the same way, which means you don't have to

compile the kernel there again.

Next we create a ramdisk for our new kernel, because otherwise the system will most likely not boot our new kernel:

```
mkinitrd /boot/initrd-2.6.18.3-default.img 2.6.18.3-default
```

2.7 Configure The GRUB Boot Loader

Now we must configure our GRUB boot loader so that our new kernels gets booted when we restart the system.

Run

```
ls -l /boot
```

to find out about your new kernel (typically begins with *vmlinuz*, e.g. *vmlinuz-2.6.18.3-default*) and ramdisk (typically begins with *initrd*, e.g. *initrd-2.6.18.3-default.img*).

Then edit */boot/grub/menu.lst*. Have a look at your existing (working) kernel stanzas there and take one of them as a sample for your new stanza and replace the kernel and ramdisk, then add the stanza above all other stanzas.

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

For example, my *menu.lst* looks like this before I add the new stanza:

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
```

```
#         root (hd0,0)
#         kernel /vmlinuz-version ro root=/dev/VolGroup00/LogVol00
#         initrd /initrd-version.img
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.9-42.0.3.EL)
        root (hd0,0)
        kernel /vmlinuz-2.6.9-42.0.3.EL ro root=/dev/VolGroup00/LogVol00
        initrd /initrd-2.6.9-42.0.3.EL.img
title CentOS-4 i386 (2.6.9-42.EL)
        root (hd0,0)
        kernel /vmlinuz-2.6.9-42.EL ro root=/dev/VolGroup00/LogVol00
        initrd /initrd-2.6.9-42.EL.img
```

and like this afterwards:

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#         all kernel and initrd paths are relative to /boot/, eg.
#         root (hd0,0)
#         kernel /vmlinuz-version ro root=/dev/VolGroup00/LogVol00
#         initrd /initrd-version.img
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
```

```
hiddenmenu
title CentOS (2.6.18.3-default)
        root (hd0,0)
        kernel /vmlinuz-2.6.18.3-default ro root=/dev/VolGroup00/LogVol00
        initrd /initrd-2.6.18.3-default.img
title CentOS (2.6.9-42.0.3.EL)
        root (hd0,0)
        kernel /vmlinuz-2.6.9-42.0.3.EL ro root=/dev/VolGroup00/LogVol00
        initrd /initrd-2.6.9-42.0.3.EL.img
title CentOS-4 i386 (2.6.9-42.EL)
        root (hd0,0)
        kernel /vmlinuz-2.6.9-42.EL ro root=/dev/VolGroup00/LogVol00
        initrd /initrd-2.6.9-42.EL.img
```

Now reboot the system:

```
shutdown -r now
```

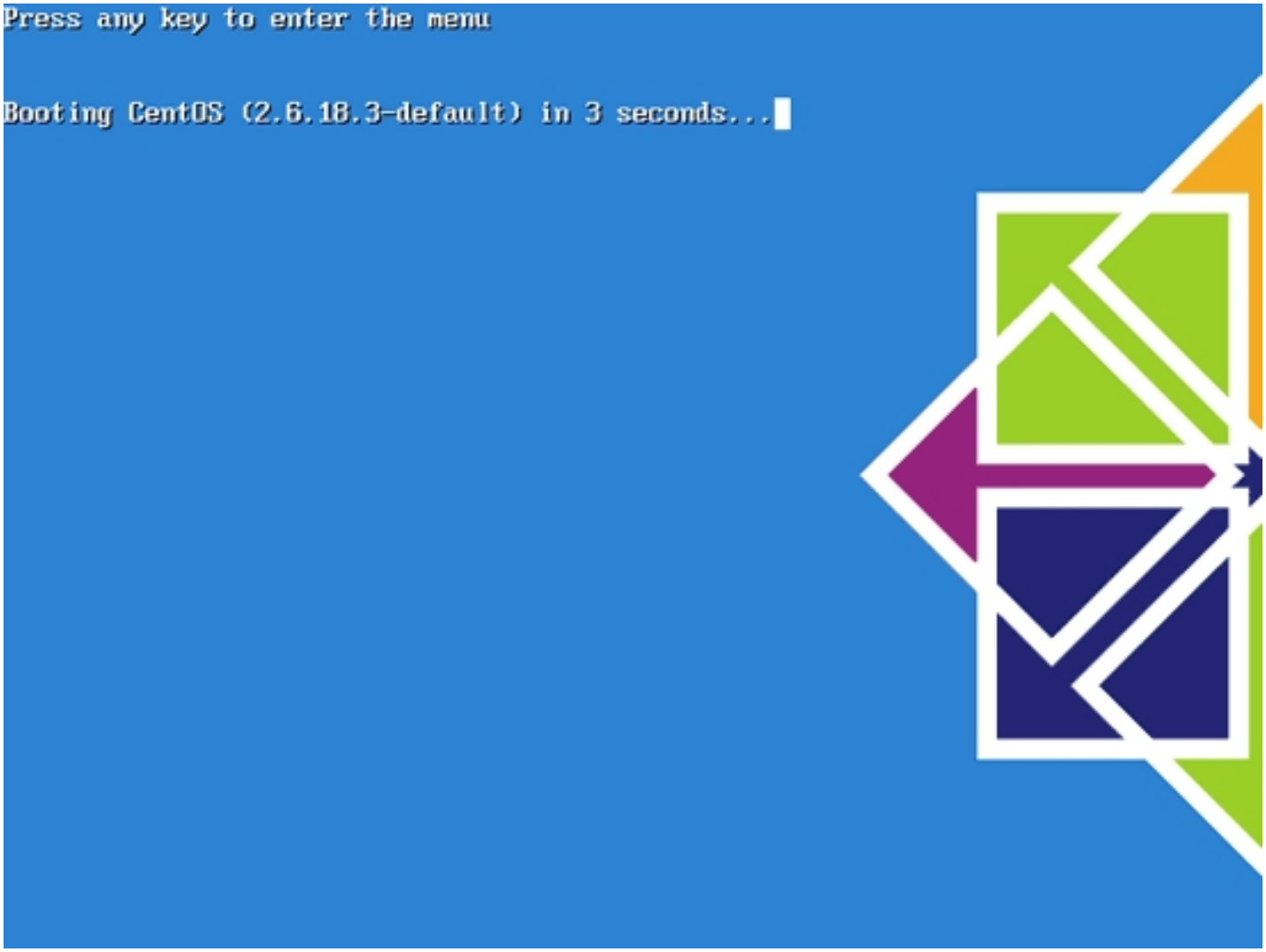
If everything goes well, it should come up with the new kernel. You can check if it's really using your new kernel by running

```
uname -r
```

This should display something like

```
2.6.18.3-default
```

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



press any key 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 stanza of the not-working kernel from `/boot/grub/menu.lst`.

3 Building A Kernel The Traditional Way

This chapter describes a different approach that can be used on any Linux system. As there's nothing CentOS-specific in this, of course you will not end up with a kernel rpm package.

3.1 Download The Kernel Sources

We download our desired kernel to `/usr/src`. Go to ***www.kernel.org*** and select the kernel you want to install, e.g. `linux-2.6.18.3.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-2.6.18.3.tar.bz2
```

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

```
tar xjf linux-2.6.18.3.tar.bz2

ln -s linux-2.6.18.3 linux

cd /usr/src/linux
```

3.2 Modify /etc/modprobe.conf

Now we must comment out the mptscsi module in `/etc/modprobe.conf` because otherwise we will get a warning like this:

```
WARNING: No module mptscsi found for kernel 2.6.18.3, continuing anyway
```

when we build our new kernel.

```
vi /etc/modprobe.conf
```

```
alias eth0 pcnet32
alias scsi_hostadapter mptbase
# alias scsi_hostadapter1 mptscsi
alias scsi_hostadapter2 mptfc
alias scsi_hostadapter3 mptspi
alias scsi_hostadapter4 mptsas
alias scsi_hostadapter5 mptscsih
```

3.3 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!

If your patches are compressed with gzip (*.gz*) instead of bzip2 (*.bz2*), then you patch your kernel as follows:

```
gunzip -c /usr/src/patch.gz | patch -p1 --dry-run

gunzip -c /usr/src/patch.gz | patch -p1
```

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-rc6, but the full sources haven't been released yet for this kernel. Instead, a *patch-2.6.19-rc6.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 or 2.6.18.3, 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-rc6 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 2.6.18.3!

This is how you apply the 2.6.19-rc6 patch to kernel 2.6.18:

```
cd /usr/src

wget http://www.kernel.org/pub/linux/kernel/v2.6/testing/patch-2.6.19-rc6.bz2

cd /usr/src/linux

bzip2 -dc /usr/src/patch-2.6.19-rc6.bz2 | patch -p1 --dry-run

bzip2 -dc /usr/src/patch-2.6.19-rc6.bz2 | patch -p1
```

3.4 Configure The Kernel

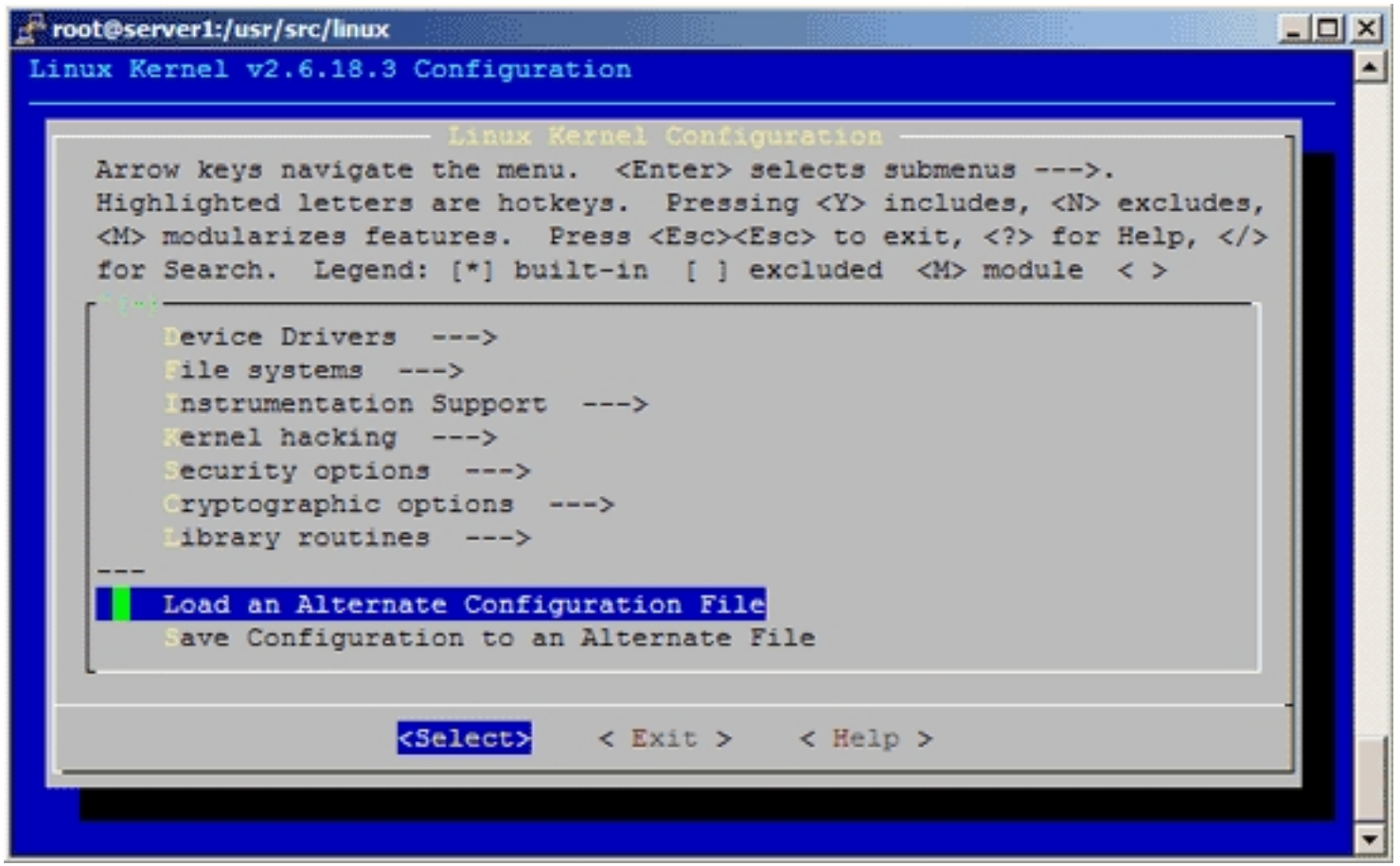
It's a good idea to use the configuration of your current working kernel as a basis for your new kernel. Therefore we copy the existing configuration to */usr/src/linux*:

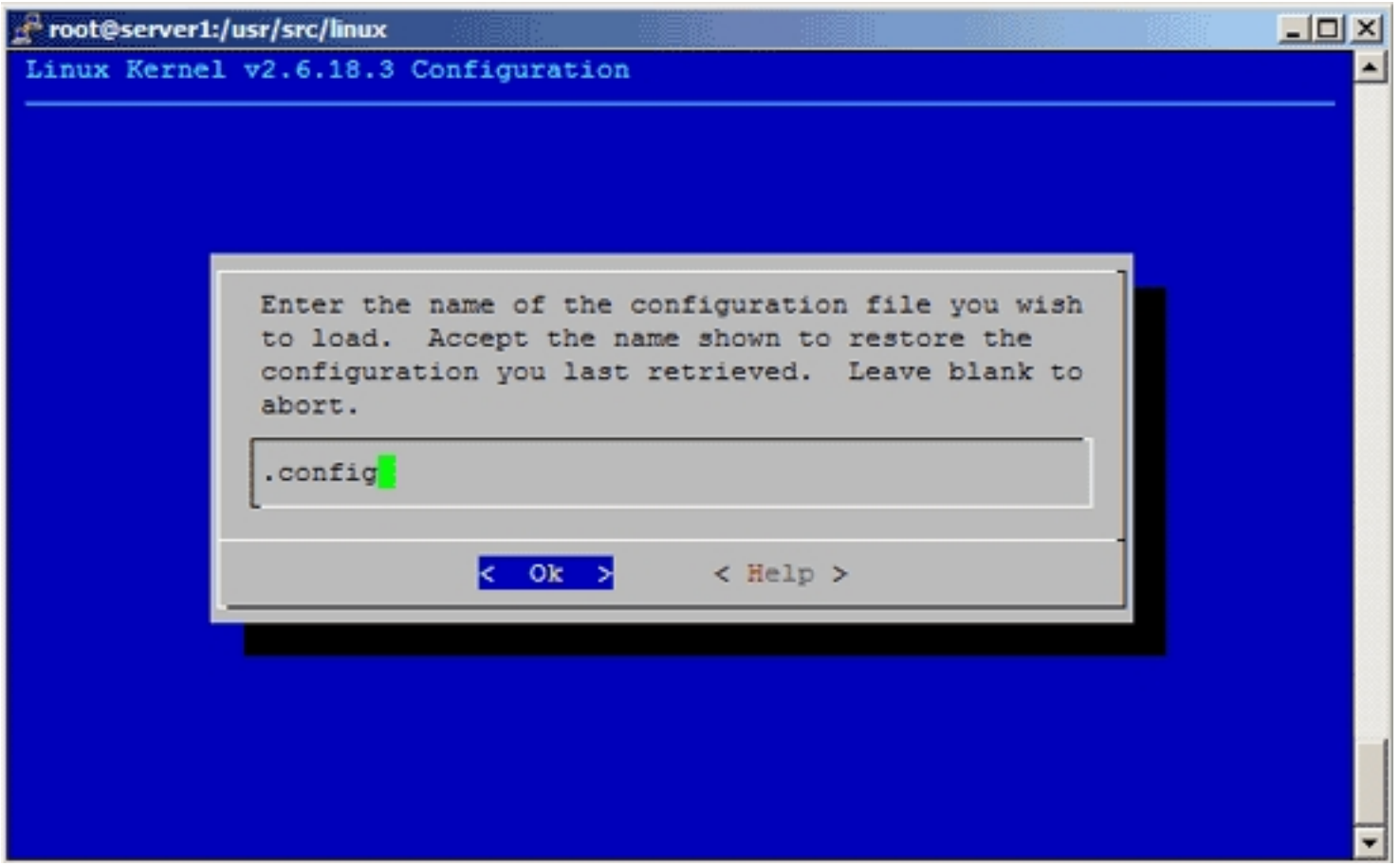
```
make clean && make mrproper

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

Then we run

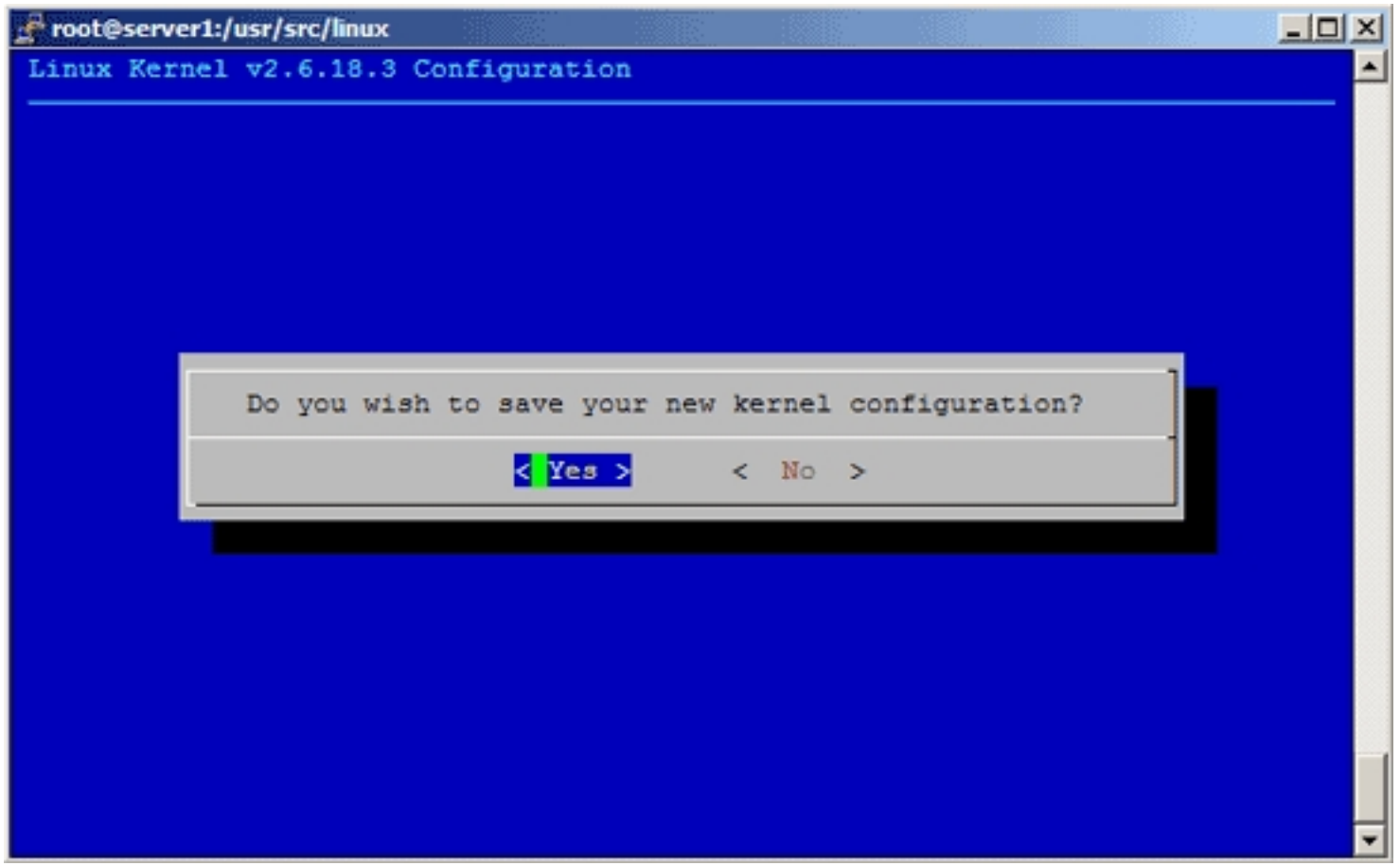
```
make menuconfig
```

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*:



3.5 Build And Install The Kernel

To build and install the kernel, execute these three commands:

```
make all

make modules_install
```

```
make install
```

Now be patient, the kernel compilation can take some hours, depending on your kernel configuration and your processor speed. The last command will also automatically create a ramdisk for you as well as configure */boot/grub/menu.lst*.

Now edit */boot/grub/menu.lst*. You should find a stanza for your new kernel at the top of the list, but to make sure that the new kernel gets booted instead of your old one, you must set the value of *default* to *0*.

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

My *menu.lst* looks like this:

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/VolGroup00/LogVol00
#          initrd /initrd-version.img
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.18.3)
        root (hd0,0)
        kernel /vmlinuz-2.6.18.3 ro root=/dev/VolGroup00/LogVol00
        initrd /initrd-2.6.18.3.img
title CentOS (2.6.9-42.0.3.EL)
```

```
        root (hd0,0)
        kernel /vmlinuz-2.6.9-42.0.3.EL ro root=/dev/VolGroup00/LogVol00
        initrd /initrd-2.6.9-42.0.3.EL.img
title CentOS-4 i386 (2.6.9-42.EL)
        root (hd0,0)
        kernel /vmlinuz-2.6.9-42.EL ro root=/dev/VolGroup00/LogVol00
        initrd /initrd-2.6.9-42.EL.img
```

Now reboot the system:

```
shutdown -r now
```

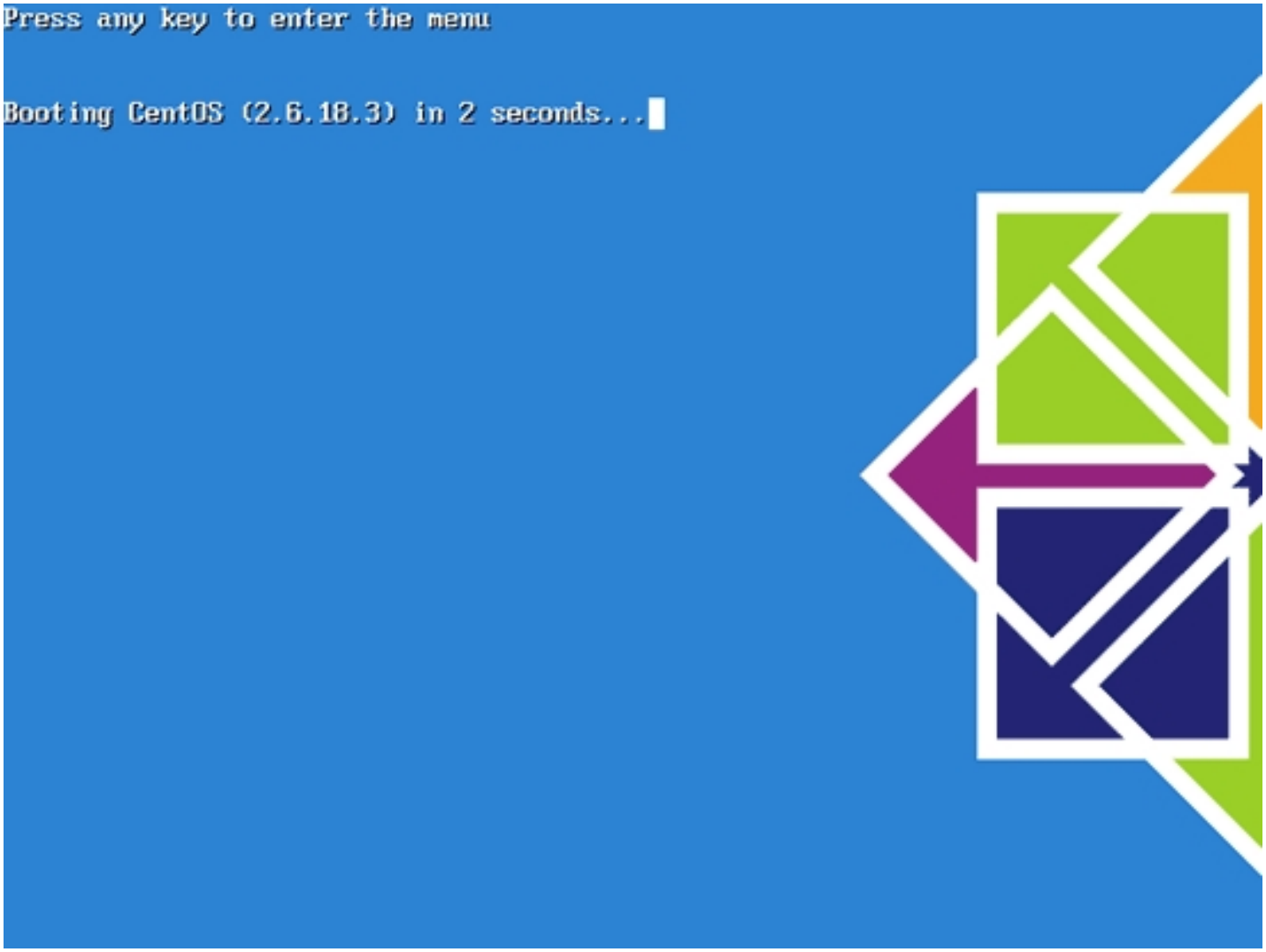
If everything goes well, it should come up with the new kernel. You can check if it's really using your new kernel by running

```
uname -r
```

This should display something like

```
2.6.18.3
```

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



press any key 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 stanza of the not-working kernel from `/boot/grub/menu.lst`.

4 Links

- CentOS: ***http://www.centos.org***
- The Linux Kernel Archives: ***http://www.kernel.org***