The Linux Kernel HOWTO

Al Dev (Alavoor Vasudevan)

< alavoor[AT]yahoo.com >

v5.6, 26 April 2003

This is a detailed guide to kernel configuration, compilation, upgrades, and troubleshooting for ix86−based systems. Can be useful for other architectures as well. This document is kept small & simple, so that even non−technical "home computer users" will be able to compile and run the Linux Kernel.

Table of Contents

Table of Contents

Table of Contents

1. Introduction

You compile Linux kernel for one of following reasons:

- You are doing kernel development
- You are adding a new hardware to machine
- You want to customize the kernel and do not want the default kernel shipped out to you.
- For *Defence Industries* or *Military applications* , you must read the kernel source code and compile with your own hands. No exceptions!! (U.S Dept of Defence compiles the Linux kernel before distributing the computers).
- Every country and every Government in the world compiles the kernel on site for security and integrity. Every Government/Corporation audits and verifies each and every line of the OS kernel source code before using the computer.
- Military Intelligence agencies around the world reads and compiles the Linux kernel source code. They know what each and every line of Linux kernel source code is doing!!
- If you compile the Linux kernel with your own hands, then it is *as good as reading and verifying* all the kernel source code!
- Each and every University in the world compiles the OS kernel before using any computer!
- For your education and knowledge of Linux kernel and ofcourse, just for fun!
- For very advanced scientific applications − you may need to do kernel compile
- It is an International Law (the U.N. laws) − "You cannot use a computer WITHOUT compiling the OS kernel with your own hands". If you disobey this law you will be "punished" with lot of computer problems!! You must compile the kernel with your own hands and not rely on someone else to do it for you!!
- It is Illegal, Unlawful, Felony and Fraud to use a computer without compiling the OS Kernel with your VERY OWN hands!
- In USA, all the corporations mandate compilation of OS kernel before using the computer and hence there is Linux, Linux & Linux everywhere in United States!
- And for many hundreds of reasons − too numerous to list!

Note: This document is kept small & simple, so that even non−technical "home computer users" will be able to compile and run the Linux Kernel!

---

2. Quick Steps − Kernel Compile

This section is written by Al Dev (alavoor[AT]yahoo.com) (The *latest version* of this document is at "http://www.milkywaygalaxy.freeservers.com" . You may want to check there for changes). Mirror sites are at − angelfire , geocities . These sites have lot of linux goodies and tips.

Kernel re−compile is required in order to make the kernel very lean and which will result in FASTER operating system . It is also required to support any new devices.

2.1. Precautionary Preparations

Before you build kernel, it is a good idea to do a backup of the system. If you had not backed up your system recently then you can do it now. You can use commercial backup tools like BRS Backup−Recovery−Software (also in this page you can find open−source/freeware backup tools listed under 'Backup and Restore Utility'). Backup is just a suggestion and it is not mandatory to do backup before building the Linux kernel.

2.2. Minor Upgrading of Kernel

If you had already built the kernel and you want to upgrade to next patch release, then you can simply copy the existing config file and reuse it. (For example you have built kernel 2.4.19 and want to upgrade to 2.4.20).

For minor upgrades : This step may save you time, if you want to reuse the old settings. Whenever you install the kernel, generally you put the config file in /boot. So, you can use the existing version of config file:

```
bash# mv /usr/src/linux/.config  /usr/src/linux/.config.save
bash# cp /boot/config−2.4.18−19.8.0  /usr/src/linux/.config
```

Or another method is − you can copy the .config file from your old linux kernel source tree to new kernel tree.

```
bash# ls −l /usr/src/lin*  # You can see that /usr/src/linux is a soft link
bash# cd /usr/src/linux
bash# cp ../linux−old−tree/.config .  # Example cp ../linux−2.4.19/.config .
```

or one other method is − you can use "make oldconfig" which default all questions based on the contents of your existing ./.config file.

NOTE: If you do not have lot of disk space in /usr/src then you can unpack the kernel source package on any partition where you have free disk space (like /home). Because kernel compile needs lot of disk space for object files like *.o. For this reason the /usr/src/linux MUST be a soft link pointing to your source directory.

After this, look in the next section to do make and install.

2.3. For the Impatient

1. Unpack the sources
2. Optional − Copy config file : You can copy the config file from your old linux kernel source tree to new kernel tree (may save time, if you want to reuse the old settings).
3. make clean; make mrproper

4. make xconfig
5. make dep
6. Give a unique name to your new Kernel − Edit /usr/src/linux/Makefile and change EXTRAVERSION
7. nohup make bzImage
8. 'make modules' and 'make modules_install'
9. And you can go to lunch or go to bed (have nice Linux dreams in sleep) and when you come back the system is ready! And see the log with 'less nohup.out'.
10. make install [num] But NOT recommended − use cp /usr/src/linux/arch/i386/boot/bzImage /boot/bzImage.myker
11. Configure GRUB or LILO.
12. Reboot and check new kernel is booting
13. Create emergency boot disk − bzdisk or mkbootdisk
14. Optional − make rpm [num] To build rpm packages
15. Optional − make clean (If you want to free up disk space)

See details of above steps in the following sections....

2.4. Building New Kernel − Explanation of Steps

Details of the steps mentioned in the previous section:

Note: Below 'bash[num]' denotes the bash prompt, you should type the commands that appear after the 'bash[num]' prompt. Below are commands tested on Redhat Linux Kernel 2.4.7−10, but it should work for other distributions with very minor changes. It should also work for older kernel versions like 2.2, 2.0 and 1.3. It should also work for future or newer versions of kernel (with little changes − let me know).

- *Note:* You can have many kernel images on your system. By following the steps below you do not overwrite or damage your existing kernel. These steps are *very safe* and your current kernel will be intact and will not be touched.

1. *Unpack the sources:* Login in as 'root' throughout all these steps. Mount Redhat linux cdrom and install the linux kernel source rpm

```
bash$ su − root
bash# cd /mnt/cdrom/RedHat/RPMS
bash# rpm −i kernel−headers*.rpm
bash# rpm −i kernel−source*.rpm
bash# rpm −i dev86*.rpm
bash# rpm −i bin86*.rpm
```

(The bin86*.rpm and 'as86' is required only for *OLDER Linux* systems like Redhat 5.x. Get Intel assembler 'as86' command from dev86*.rpm on cdrom or from bin86−mandrake , bin86−kondara). Also make sure that /usr/src/linux is soft link pointing to proper unpacked source.

```
bash# cd /usr/src
bash# ls −l    # You should see that /usr/src/linux is soft link pointing to sourc
lrwxrwxrwx    1 root     root           19 Jan 26 11:01 linux −> linux−2.4.18−19.8
drwxr−xr−x   17 root     root         4096 Jan 25 21:08 linux−2.4.18−14
drwxr−xr−x   17 root     root         4096 Mar 26 12:50 linux−2.4.18−19.8.0
drwxr−xr−x    7 root     root         4096 Jan 14 16:32 redhat
```

If it is not a soft link then do rename /usr/src/linux to /usr/src/linux−2.4.yy and create a soft link.

NOTE: If you do not have lot of disk space in /usr/src then you can unpack the kernel source package on any partition where you have free disk space (like /home). Because kernel compile needs lot of disk space for object files like *.o. For this reason the /usr/src/linux MUST be a soft link pointing to your source directory.

2. *Optional − Copy config file :* This step may save you time, if you want to reuse the old settings. Whenever you install the kernel, generally you put the config file in /boot. So, you can use the existing version of config file:

```
bash# mv /usr/src/linux/.config  /usr/src/linux/.config.save
bash# cp /boot/config−2.4.18−19.8.0  /usr/src/linux/.config
```

Or another method is − you can copy the .config file from your old linux kernel source tree to new kernel tree

```
bash# ls −l /usr/src/lin*  # You can see that /usr/src/linux is a soft link
bash# cd /usr/src/linux
bash# cp ../linux−old−tree/.config .  # Example cp ../linux−2.4.19/.config .
```

or one other method is − you can use "make oldconfig" which default all questions based on the contents of your existing ./.config file.

3. *Clean :* Before doing mrproper below, you may want to backup the .config file.

```
bash# cd /usr/src/linux
bash# cp .config .config.save
bash# make clean
bash# make mrproper  # Must do this if want to start clean slate or if you face lo
```

4. *Configure:*

   - Start X−windows with 'startx'. If you are not able to start X−window then see next step below.

     ```
     bash# man startx
     bash# startx
     bash# cd /usr/src/linux
     bash# make xconfig
     ```

   - If you are not able to start X−window above then try −

     ```
             bash# export TERM=xterm
             bash# make menuconfig
     If you find scrambled display, then use different terminal emulators like v
     vt102, vt220 or ansi. The display will be scrambled and will have garbage
     characters in cases where you use telnet to login to remote linux. In such
     cases you should use the terminal emulators like vt100, vt220.
     For example:
             bash# export TERM=vt220
             bash# export TERM=ansi
     At a lower level of VT, use:
             bash# export TERM=vt100
             bash# make menuconfig
     If the menuconfig command fails then try −
             bash# make config
     ```

   The *"make xconfig" or "make menuconfig"* brings up a user friendly GUI interface. And *"make config"* brings up command−line console mode interface. You can load the configuration file from */usr/src/linux/.config* (dot config file. Note the dot before config). Click on button "Load Configuration from File". Within 'make xconfig' you must do these (to avoid problems) −

   - *VERY IMPORTANT !!! :* Select proper CPU type − Pentium 3, AMD K6, Cyrix, Pentium 4,

Intel 386, DEC Alpha, PowerPC otherwise kernel compile will fail and even if it compiles, it will not boot!!

- Select SMP support − whether single CPU or multiple CPUs
- Filesystems − Select Windows95 Vfat, MSDOS, NTFS as part of kernel and not as loadable modules. (My personal preference, but you are free to pick your own option).
- Enable the Loadable kernel modules support! With this option you can load/unload the device drivers dynamically on running linux system on the fly. See the Modules chapter at Section 3 .

Save and Exit "make xconfig". All the options which you selected is now saved into configuration file at */usr/src/linux/.config* (dot config file).

5. *Dep :* And now, do −

```
bash# make dep
```

6. *Give a unique name to your new Kernel:* You can give a name to your kernel, so that it is unique and does not interfere with others.

```
bash# cd /usr/src/linux
bash# vi Makefile
```

Here look for EXTRAVERSION = −19.8.0_Blah_Blah_Blah and change to something like EXTRAVERSION = −19.8.0MyKernel.26Jan2003

7. *Do make:* Read the following file (to gain some knowledge about kernel building. Tip: Use the color editor gvim for better readability.

```
bash# gvim −R   /usr/src/linux/arch/i386/config.in
bash# man less
bash# less   /usr/src/linux/arch/i386/config.in
Type 'h' for help and to navigate press i, j, k, l, h or arrow, page up/down keys.
```

Now, give the make command −

```
        bash# cd /usr/src/linux
        bash# man nohup
        bash# nohup make bzImage &
        bash# man tail
        bash# tail −f nohup.out     (.... to monitor the progress)
This will put the kernel in /usr/src/linux/arch/i386/boot/bzImage
```

8. *LOADABLE MODULES:* Now, while the 'make' is cranking along in the previous step "Do make", you should bring up another new xterm shell window and follow these steps: This step is required *ONLY if* you had enabled Loadable module support in step "Configure Step" above. Loadable module are located in /lib/modules. You MUST do this step if you enabled or disabled any modules, otherwise you will get 'unresolved symbols' errors during or after kernel boot.

```
# Bring up a new Xterm shell window and ...
bash# cd /usr/src/linux
# Redirect outputs such that you do not overwrite the nohup.out which is still run
bash# nohup make modules 1> modules.out 2> modules.err  &
bash# make modules_install   # Do this, only after the above make command is succe
```

This will copy the modules to /lib/modules directory. See the Modules chapter at Section 3 .

9. *Now go to Lunch or Bed :* Since both the make windows are cranking along, and now, you can go to lunch (chitchat, have nap) or go to bed (have nice Linux dreams in sleep) and when you wake up and come back the system is ready! You can check with command 'less nohup.out' to see the log of output.

```
bash# cd /usr/src/linux
bash# less nohup.out
bash# less modules.err
```

```
bash# less modules.out
If no errors then do:
bash# make modules_install
```

10. *bzImage:* After bzImage is successful, copy the kernel image to /boot directory. You must copy the new kernel image to /boot directory, otherwise the new kernel *MAY NOT* boot. You must also copy the config file to /boot area to reflect the kernel image, for documentation purpose.
```
bash# cp /usr/src/linux/arch/i386/boot/bzImage     /boot/bzImage.myker.26mar2001
# You MUST copy the config file to reflect the corresponding kernel image,
# for documentation purpose.
bash# cp /usr/src/linux/.config /boot/config-<your_kernelversion_date>
# Example: cp /usr/src/linux/.config /boot/config-2.4.18-19.8.0-26mar2001
```
NOTE : If you are planning to use the initrd in LILO or GRUB then you may want to build initrd and place it in /boot/initrd*.img. See the Appendix A at Section 13 .
11. *Configure GRUB or LILO :* There are two options for boot loading under Redhat Linux − GRUB and LILO. *Configure GRUB:* GRUB is recent and much better tool than LILO and it is my first preference to use GRUB. LILO is an older technology. GRUB differs from bootloaders such as LILO in that *"it can lie to MS Windows and make MS Windows believe that it's installed on the first partition even if it's not!!"* . So you can keep your current Linux system where it is and install Windows on the side. See the Section 15 file. *Configure LILO:* LILO is older tool and see the Section 14 to configure LILO. (see also "http://www.linuxdoc.org/HOWTO/LILO-crash-rescue-HOWTO.html")
12. Reboot the machine and at lilo press tab key and type 'myker' If it boots then you did a good job! Otherwise at lilo select your old kernel, boot and re−try all over again. Your old kernel *is still INTACT and SAFE* at say */boot/vmlinuz−2.0.34−0.6*
13. If your new kernel 'myker' boots and works properly, you can create the boot disk. Insert a blank floppy into floppy drive and −
```
bash# cd /usr/src/linux
bash# make bzdisk
See also mkbootdisk −
bash# rpm −i mkbootdisk*.rpm
bash# man mkbootdisk
```
14. Build RPMs Optional − You can also build RPM packages of kernel, in case you want to install the new image on several machines.
```
make rpm   # To build rpm packages
```
15. *Clean:* Optional − make clean (If you want to free up disk space)

2.5. Troubleshooting

Having any problems? See the Section 17 .

2.6. Post Kernel Building

See the Section 16 .

3. Loadable Modules

Loadable kernel modules can save memory and ease configuration. The scope of modules has grown to include filesystems, ethernet card drivers, tape drivers, printer drivers, and more.

Loadable modules are pieces of kernel code which are not linked (included) directly in the kernel. One compiles them separately, and can insert and remove them into the running kernel at almost any time. Due to its flexibility, this is now the preferred way to code certain kernel features. Many popular device drivers, such as the PCMCIA drivers and the QIC−80/40 tape driver, are loadable modules.

See the Module−HOWTO at "http://www.tldp.org/HOWTO/Module−HOWTO" .

And see these man pages

```
bash# rpm −i /mnt/cdrom/Redhat/RPMS/modutils*.rpm
bash# man lsmod
bash# man insmod
bash# man rmmod
bash# man depmod
bash# man modprobe
```

For example to load the module `/lib/modules/2.4.2−2/kernel/drivers/block/loop.o` , you would do :

```
bash# man insmod
bash# modprobe loop
bash# insmod loop
bash# lsmod
```

You can set the PATH which the insmod searches in /etc/modules.conf.

3.1. Installing the module utilities

You can install the Module Utilities RPM with:

```
bash# rpm −i /mnt/cdrom/Redhat/RPMS/modutils*.rpm
```

`insmod` inserts a module into the running kernel. Modules usually have a `.o` extension; the example driver mentioned above is called `drv_hello.o` , so to insert this, one would say ` `insmod drv_hello.o` '. To see the modules that the kernel is currently using, use `lsmod` . The output looks like this: blah# lsmod Module: #pages: Used by: drv_hello 1 ` `drv_hello` ' is the name of the module, it uses one page (4k) of memory, and no other kernel modules depend on it at the moment. To remove this module, use ` `rmmod drv_hello` '. Note that `rmmod` wants a *module name*, not a filename; you get this from `lsmod` 's listing. The other module utilities' purposes are documented in their manual pages.

3.2. Modules distributed with the kernel

As of version 2.0.30, most of everything is available as a loadable modules. To use them, first make sure that you don't configure them into the regular kernel; that is, don't say `y` to it during `` `make config' ``. Compile a new kernel and reboot with it. Then, `cd` to `/usr/src/linux` again, and do a `` `make modules' ``. This compiles all of the modules which you did not specify in the kernel configuration, and places links to them in `/usr/src/linux/modules`. You can use them straight from that directory or execute `` `make modules_install' ``, which installs them in `/lib/modules/x.y.z`, where `x.y.z` is the kernel release.

This can be especially handy with filesystems. You may not use the minix or msdos filesystems frequently. For example, if I encountered an msdos (shudder) floppy, I would `insmod /usr/src/linux/modules/msdos.o`, and then `rmmod msdos` when finished. This procedure saves about 50k of RAM in the kernel during normal operation. A small note is in order for the minix filesystem: you should *always* configure it directly into the kernel for use in ``rescue'' disks.

3.3. Howto Install Just A Single Module ?

Let us assume that you already did 'make modules' and 'make modules_install'. And later you did 'make clean' to free up disk space. And now, you want to change a "C" file in one of the modules and want to rebuild just that module and copy the module file to /lib/modules. How do you do it?

You can compile just a single module file (say like foo.o) and install it. For this simply edit the Makefile and change the SUBDIRS to add only those directories you are interested.

For an example, if I am interested in installing only fs/autofs module, then I do the following :

```
cd /usr/src/linux
cp Makefile Makefile.my
vi Makefile.my
# And comment out the line having 'SUBDIRS' and add the
# directory you are interested, for example like fs/autofs as below :
        #SUBDIRS        =kernel drivers mm fs net ipc lib abi crypto
        SUBDIRS         =fs/autofs
# Save the file Makefile.my and give −
make −f Makefile.my modules
# This will create module autofs.o
# Now, copy the module object file to destination /lib/modules
make −f Makefile.my modules_install
# And this will do 'cp autofs.o /lib/modules/2.4.18−19.8.0/kernel/fs/autofs'
```

Learn more about Makefile and make. See the manual for GNU make at

- "http://www.gnu.org/manual/make" .
- University of Utah Makefile "http://www.math.utah.edu/docs/info/make−stds_toc.html"
- University of Hawaii Makefile "http://www.eng.hawaii.edu/Tutor/Make"
- In Linux − man make
- In Linux − info make

Get familiar with the Makefile which makes the modules. The Makefile has module line like

```
        modules: $(patsubst %, _mod_%, $(SUBDIRS))
```

The patsubst function has the syntax $(patsubst pattern,replacement,text). It uses the percent symbol ([percnt]) the same way pattern rules do − as a string which matches in both the pattern and the replacement text. It searches the text for whitespace−separated words that match the pattern and substitutes the replacement for them.

This makefile includes shell functions as well as standard make functions. The syntax for a shell function is $(shell command). This returns the output of the shell function (stripping new lines).

---

4. Cloning of Linux Kernels

You may want to build a Linux kernel on a system and then you may want to mass deploy to many identical hardware PCs. To make it easy to install your newly built kernel on hundreds of other systems, you may want to package it in RPMs (Redhat) or DEB package (Debian) or just tar.gz files.

1. Build a kernel rpm package with rpmbuild −ba kernel*.spec
2. Check that the kernel*.rpm generated has all the files in /lib/modules/2.x.x−y directory. Otherwise you may want to tar gzip the directory /lib/modules/2.x.x−y and take it to destination machines.
3. Check that your kernel package has /boot/initrd−2.x.x−y.img file, otherwise you may want to tar gzip and take it to destination machines.
4. And other files in /boot which are not in the kernel*.rpm package.

---

5. Important questions and their answers

5.1. What does the kernel do, anyway?

The Unix kernel acts as a mediator for your programs and your hardware. First, it does (or arranges for) the memory management for all of the running programs (processes), and makes sure that they all get a fair (or unfair, if you please) share of the processor's cycles. In addition, it provides a nice, fairly portable interface for programs to talk to your hardware.

There is certainly more to the kernel's operation than this, but these basic functions are the most important to know.

5.2. Why would I want to upgrade my kernel?

Newer kernels generally offer the ability to talk to more types of hardware (that is, they have more device drivers), they can have better process management, they can run faster than the older versions, they could be more stable than the older versions, and they fix silly bugs in the older versions. Most people upgrade kernels because they want the device drivers and the bug fixes.

5.3. What kind of hardware do the newer kernels support?

See the Hardware−HOWTO . Alternatively, you can look at the ` `config.in' file in the linux source, or just find out when you try ` `make config'. This shows you all hardware supported by the standard kernel distribution, but not everything that linux supports; many common device drivers (such as the PCMCIA drivers and some tape drivers) are loadable modules maintained and distributed separately.

5.4. What version of gcc and libc do I need?

Linus recommends a version of gcc in the `README` file included with the linux source. If you don't have this version, the documentation in the recommended version of gcc should tell you if you need to upgrade your libc. This is not a difficult procedure, but it is important to follow the instructions.

5.5. What's a loadable module?

See the Modules chapter at Section 3 .

5.6. How much disk space do I need?

It depends on your particular system configuration. First, the compressed linux source is nearly 14 megabytes large at version 2.2.9. Many sites keep this even after unpacking. Uncompressed and built with a moderate configuration, it takes up another 67 MB.

5.7. How long does it take?

With newer machines, the compilation takes dramatically less time than older ones; an AMD K6−2/300 with a fast disk can do a 2.2.x kernel in about four minutes. As for old Pentiums, 486s, and 386s, if you plan to compile one, be prepared to wait, possibly hours, days..

If this troubles you, and you happen to have a faster machine around to compile on, you can build on the fast machines (assuming you give it the right parameters, that your ulilities are up−to−date, and so on), and then transfer the kernel image to the slower machine.

---

6. Patching the kernel

6.1. Applying a patch

Incremental upgrades of the kernel are distributed as patches. For example, if you have Linux v1.1.45, and you notice that there's a `patch46.gz' out there for it, it means you can upgrade to version 1.1.46 through application of the patch. You might want to make a backup of the source tree first (`make clean' and then `cd /usr/src; tar zcvf old−tree.tar.gz linux' will make a compressed tar archive for you.).

So, continuing with the example above, let's suppose that you have `patch46.gz' in `/usr/src. cd` to `/usr/src` and do a `zcat patch46.gz [verbar] patch −p0'(or `patch −p0 [lt] patch46' if the patch isn't compressed). You'll see things whizz by (or flutter by, if your system is that slow) telling you that it is trying to apply hunks, and whether it succeeds or not. Usually, this action goes by too quickly for you to read, and you're not too sure whether it worked or not, so you might want to use the `−s` flag to `patch`, which tells `patch` to only report error messages (you don't get as much of the ``hey, my computer is actually doing something for a change!" feeling, but you may prefer this..). To look for parts which might not have gone smoothly, cd to `/usr/src/linux` and look for files with a `.rej` extension. Some versions of `patch` (older versions which may have been compiled with on an inferior filesystem) leave the rejects with a `[num ]` extension. You can use `find' to look for you; find . −name '*.rej' −print prints all files who live in the current directory or any subdirectories with a `.rej` extension to the standard output.

If everything went right, do a `make clean', `config', and `dep' as described in sections 3 and 4.

There are quite a few options to the `patch` command. As mentioned above, `patch −s` will suppress all messages except the errors. If you keep your kernel source in some other place than `/usr/src/linux`, `patch −p1` (in that directory) will patch things cleanly. Other `patch` options are well−documented in the manual page.

6.2. If something goes wrong

(Note: this section refers mostly to quite old kernels)

The most frequent problem that used to arise was when a patch modified a file called `config.in' and it didn't look quite right, because you changed the options to suit your machine. This has been taken care of, but one still might encounter it with an older release. To fix it, look at the `config.in.rej` file, and see what remains of the original patch. The changes will typically be marked with `+' and `−' at the beginning of the line. Look at the lines surrounding it, and remember if they were set to `y' or `n'. Now, edit `config.in`, and change `y' to `n' and `n' to `y' when appropriate. Do a patch −p0 < config.in.rej and if it reports that it succeeded (no fails), then you can continue on with a configuration and compilation. The `config.in.rej` file will remain, but you can get delete it.

If you encounter further problems, you might have installed a patch out of order. If patch says `previously applied patch detected: Assume −R?', you are probably trying to apply a patch which is below your current version number; if you answer `y', it will attempt to degrade your source, and will most likely fail; thus, you will need to get a whole new source tree (which might not have been such a bad idea in the first place).

To back out (unapply) a patch, use ` `patch -R' on the original patch.

The best thing to do when patches really turn out wrong is to start over again with a clean, out-of-the-box source tree (for example, from one of the `linux-x.y.z.tar.gz` files), and start again.

6.3. Getting rid of the .orig files

After just a few patches, the `.orig` files will start to pile up. For example, one 1.1.51 tree I had was once last cleaned out at 1.1.48. Removing the .orig files saved over a half a meg. find . -name '*.orig' -exec rm -f {} ';' will take care of it for you. Versions of `patch` which use `[num ]` for rejects use a tilde instead of `.orig`.

There are better ways to get rid of the `.orig` files, which depend on GNU `xargs`: find . -name '*.orig' | xargs rm or the ``quite secure but a little more verbose'' method: find . -name '*.orig' -print0 | xargs --null rm --

6.4. Other patches

There are other patches (I'll call them ``nonstandard'') than the ones Linus distributes. If you apply these, Linus' patches may not work correctly and you'll have to either back them out, fix the source or the patch, install a new source tree, or a combination of the above. This can become very frustrating, so if you do not want to modify the source (with the possibility of a very bad outcome), back out the nonstandard patches before applying Linus', or just install a new tree. Then, you can see if the nonstandard patches still work. If they don't, you are either stuck with an old kernel, playing with the patch or source to get it to work, or waiting (possibly begging) for a new version of the patch to come out.

How common are the patches not in the standard distribution? You will probably hear of them. I used to use the noblink patch for my virtual consoles because I hate blinking cursors (This patch is (or at least was) frequently updated for new kernel releases.). With most newer device drivers being developed as loadable modules, though, the frequecy of ``nonstandard'' patches is decreasing significantly.

7. Tips and tricks

7.1. Redirecting output of the make or patch commands

If you would like logs of what those ` make ' or ` patch ' commands did, you can redirect output to a file. First, find out what shell you're running: ` grep root /etc/passwd ' and look for something like ` /bin/csh '.

If you use sh or bash, (command) 2>&1 | tee (output file) will place a copy of (command) 's output in the file ` (output file)'.

For csh or tcsh, use (command) |& tee (output file)

For rc (Note: you probably do not use rc) it's (command) >[2=1] | tee (output file)

7.2. Conditional kernel install

Other than using floppy disks, there are several methods of testing out a new kernel without touching the old one. Unlike many other Unix flavors, LILO has the ability to boot a kernel from anywhere on the disk (if you have a large (500 MB or above) disk, please read over the LILO documentation on how this may cause problems). So, if you add something similar to image = /usr/src/linux/arch/i386/boot/bzImage label = new_kernel to the end of your LILO configuration file, you can choose to run a newly compiled kernel without touching your old /vmlinuz (after running lilo , of course). The easiest way to tell LILO to boot a new kernel is to press the shift key at bootup time (when it says LILO on the screen, and nothing else), which gives you a prompt. At this point, you can enter ` new_kernel ' to boot the new kernel.

If you wish to keep several different kernel source trees on your system at the same time (this can take up a *lot* of disk space; be careful), the most common way is to name them /usr/src/linux-x.y.z , where x.y.z is the kernel version. You can then ``select'' a source tree with a symbolic link; for example, ` ln -sf linux-1.2.2 /usr/src/linux ' would make the 1.2.2 tree current. Before creating a symbolic link like this, make certain that the last argument to ln is not a real directory (old symbolic links are fine); the result will not be what you expect.

7.3. Kernel updates

Russell Nelson (nelson@crynwr.com) summarizes the changes in new kernel releases. These are short, and you might like to look at them before an upgrade. They are available with anonymous ftp from "ftp://ftp.emlist.com" in pub/kchanges or through the URL "http://www.crynwr.com/kchanges"

---

8. Mount RPMs With FTPFS

By this time, your kernel is compiled and running ok. You will have the need to access countless number of RPMs which you may need to install in near future. One way is to physically mount the Linux CDROMS, but there are more than 3 Linux cdroms and it is cumbersome to remove and change the Linux cdroms. Hence, here comes the FTPFS.

FTP File System is a Linux kernel module, enhancing the VFS with FTP volume mounting capabilities. That is, you can "mount" FTP shared directories in your very personal file system and take advantage of local files ops. It is at "http://lufs.sourceforge.net/lufs" and at "http://ftpfs.sourceforge.net" .

8.1. Using the ftpfs

Download the ftpfs and install it on your system. The ftpfs is installed as a module in /lib/modules/2.4.18−19.8.0/kernel/fs/ftpfs/ftpfs.o. And also the command ftpmount is in /usr/bin/ftpmount. And you can do the following:

Login as root (su − root) and run this script:

```
#!/bin/sh −x
# Use this script to mount ftp redhat cdroms rpms directory disk1,2,3
# Built rpm by name ftpfs.
# http://lufs.sourceforge.net/main/projects.html
# ftpmount −−help
# Try this: ftpmount [user[:pass]@]host_name[:port][/root_dir] mount_point [−o]
# [−uid=id] [gid=id] [fmask=mask] [dmask=mask]
#ftpmount anonymous:pass@ftp.kernel.org /mnt/ftpfs
#mkdir −p /mnt/ftpfs /mnt/ftpfs/updates /mnt/ftpfs/rpms /mnt/ftpfs/contrib
# Redhat ftp mirror sites − http://www.redhat.com/download/mirror.html
FTPSITE="csociety−ftp.ecn.purdue.edu"
USER="anonymous:pass"
ftpmount $USER@$FTPSITE/pub/redhat/redhat  /mnt/ftpfs/site
ftpmount $USER@$FTPSITE/pub/redhat/redhat/linux/updates/8.0/en/os  /mnt/ftpfs/updates
ftpmount $USER@$FTPSITE/pub/redhat/redhat/linux/8.0/en/os/i386/RedHat /mnt/ftpfs/rpms
ftpmount $USER@$FTPSITE/pub/redhat−contrib /mnt/ftpfs/contrib
```

8.2. The ftpfs Commands

Before you even start thinking about mounting FTP volumes, make sure you have a decent bandwidth or it's gonna suck.

8.2.1. The autofs way − A must try!

If you were wise enough to install the autofs/automount bridge (check out the installation notes) there is a cool way to use ftpfs: just try to access any file/dir on the desired server under /mnt/ftpfs.

```
cd /mnt/ftpfs/[user:pass@]ftp_server[:port]
```

Something like cd /mnt/ftpfs/ftp.kernel.org. And guess what? You're there!

Normally you will only use this for anonymous ftp since you don't want your user/pass info to show up in the /mnt/ftpfs/ tree.

8.2.2. The ftpmount way

ftpmount [lsqb]user[lsqb]:password]@]hostname[lsqb]:port][lsqb]/root_dir] mount_point [lsqb]−own] [lsqb]−uid=id] [lsqb]−gid=id] [lsqb]−fmask=mask] [lsqb]−dmask=mask] [lsqb]−active]

```
The parameters: [defaults]
    * user: The user to be used for logging on the FTP server. [anonymous]
    * password: The password for that user. [user@ftpfs.sourceforge.net]
    * hostname: The FTP server.
    * port: The port the server is listening on. [21]
    * root_dir: The directory on the FTP server you want to be mounted. This should be sp
    * mount_point: The local directory you want to mount the FTP server onto.
    * own: Flag to force ownership on all remote files. Useful for FTP servers that list
    * uid: The local user ID you want to be the owner of the mounted tree.
    * gid: The local group ID you want to own the mounted tree.
    * fmask: The numeric mode to be ORed on all mounted files.
    * dmask: The numeric mode to be ORed on all mounted dirs.
    * active: Flag to enable active mode for FTP transfers. Useful if you're behind some
```

Eg: ftpmount mali@ftp.linuxnet.wox.org /mnt/ftpfs −uid=500 −gid=500 −dmask=555

It is generally a good idea not to provide your password as a parameter, since ftpmount will ask for it.

8.2.3. The mount way

If for some reason you choose not to use ftpmount (you probably installed the kernel patch and are too lazy to install ftpmount too), here's the way to use good−ol mount:

mount −n −t ftpfs none mount_point −o ip=server_ip [lsqb],user=user_name] [lsqb],pass=password] [lsqb],port=server_port] [lsqb],root= root_dir] [lsqb],own] [lsqb],uid=id] [lsqb],gid=id] [lsqb],fmode=mask] [lsqb],dmode=mask] [lsqb],active]

Please note that you have to provide the server's IP and that the only way to enter a password is in clear. For example, while testing, I used the following command:

mount −n −t ftpfs none /mnt/ftpfs −o ip=127.0.0.1,user=mali,pass=my_pass

8.2.4. Some notes

To unmount the volume, you go like

```
umount mount_point
```

The own option (−o for ftpmount) forces ownership by the mounting user on all files. This is useful for accommodating servers with strange user/permissions management (SERVU & stuff).

A few words of wisdom:

- Use −n mount option! I bet you don't want your user/password information listed in mtab.
- Don't push it! (pushing it = a dozen processes reading on the mount point)
- It works best for one process! While concurrent access (under normal circumstances) shouldn't cause any problem, the output is optimized for one process reading (the TCP connection is kept alive). So, if you're gonna watch a movie, you don't want other processes to access the mount point and kill the throughoutput (trust me!).
- The address in IP format sucks! − Go get ftpmount.

---

9. Linux Kernel Textbooks and Documents

Check the following books on "The Linux Kernel" at

- Kernel book "http://kernelbook.sourceforge.net" and at "http://sourceforge.net/projects/kernelbook"
- Linux Kernel books like 'The Linux Kernel Module Programming Guide', 'Linux Kernel 2.4 Internals', 'The Linux System Administrators Guide', 'The Linux Network Administrator's Guide' and others at "http://www.tldp.org/guides.html"
- FreeTech books "http://www.tcfb.com/freetechbooks/booklinuxdev.html"
- Rusty's "http://www.netfilter.org/unreliable−guides"
- Linux Kernel links "http://www.topology.org/soft/lkernel.html"
- Linux Kernel Internals "http://www.moses.uklinux.net/patches/lki.html"
- Books links "http://linux−mm.org/kernel−links.shtml"

Refer also to other relevant HOWTOs at:

- Bootdisk−HOWTO
- Sound−HOWTO : sound cards and utilities
- SCSI−HOWTO: all about SCSI controllers and devices. And see also SCSI−2.4−HOWTO
- NET−2−HOWTO: networking
- PPP−HOWTO: PPP networking in particular
- PCMCIA−HOWTO: about the drivers for your notebook
- ELF−HOWTO: ELF: what it is, converting.. Mirror sites at ELF−HOWTO−mirror . See also GCC−HOWTO
- Hardware−HOWTO: overview of supported hardware
- Module mini−HOWTO: more on kernel modules
- Kerneld mini−HOWTO: about kerneld
- BogoMips mini−HOWTO: in case you were wondering

---

10. Kernel Files Information

This section gives a "very brief" and "introduction" to some of the Linux Kernel System. If you have time you can give one reading.

Caution: You should be extra careful about these Kernel Files and you must not edit or touch or move/delete/rename them.

10.1. vmlinuz and vmlinux

The vmlinuz is the Linux kernel executable. This is located at /boot/vmlinuz. This can be a soft link to something like /boot/vmlinuz−2.4.18−19.8.0

The vmlinux is the uncompressed built kernel, vmlinuz is the compressed one, that has been made bootable. (Note both names vmlinux and vmlinuz look same except for last letter z). Generally, you don't need to worry about vmlinux, it is just an intermediate step.

The kernel usually makes a bzImage, and stores it in arch/i386/boot, and it is up to the user to copy it to /boot and configure GRUB or LILO.

10.2. Bootloader Files

The .b files are "bootloader" files. they are part of the dance required to get a kernel into memory to begin with. You should NOT touch them.

```
ls −l /boot/*.b
−rw−r−−r−−    1 root     root         5824 Sep  5  2002 /boot/boot.b
−rw−r−−r−−    1 root     root          612 Sep  5  2002 /boot/chain.b
−rw−r−−r−−    1 root     root          640 Sep  5  2002 /boot/os2_d.b
```

10.3. Message File

The 'message' file contains the message your bootloader will display, prompting you to choose an OS. So DO NOT touch it.

```
ls −l /boot/message*
−rw−r−−r−−    1 root     root        23108 Sep  6  2002 /boot/message
−rw−r−−r−−    1 root     root        21282 Sep  6  2002 /boot/message.ja
```

10.4. initrd.img

See the Appendix A at Section 13 .

10.5. bzImage

The bzImage is the compressed kernel image created with command 'make bzImage' during kernel compile.

10.6. module−info

This file 'module−info' is created by anaconda/utils/modlist (specific to Redhat Linux Anaconda installer). Other Linux distributions may be having equivalent command. Refer to your Linux distributor's manual pages.

See this script and search for "module−info" updmodules .

Below is a cut from this script:

```
#!/bin/bash
# updmodules.sh
        MODLIST=$PWD/../anaconda/utils/modlist
-- snip cut
blah blah blah
-- snip cut
    # create the module-info file
    $MODLIST --modinfo-file $MODINFO --ignore-missing --modinfo \
                $(ls *.o | sed 's/\.o$//') > ../modinfo
```

The program anaconda/utils/modlist is located in anaconda−runtime*.rpm on the Redhat CDROM

```
cd /mnt/cdrom/RedHat/RPMS
rpm -i anaconda-8.0-4.i386.rpm
rpm -i anaconda-runtime-8.0-4.i386.rpm
ls -l /usr/lib/anaconda-runtime/modlist
```

Get the source code for anaconda/utils/modlist.c from anaconda*.src.rpm at "http://www.rpmfind.net/linux/rpm2html/search.php?query=anaconda" Read online at modlist.c . .

The file 'module−info' is generated during the compile. It is an information file that is at least used during filing proper kernel OOPS reports. It is a list of the module entry points. It may also be used by depmod in building the tables that are used by insmod and its kith and kin. This includes dependancy information for other modules needed to be loaded before any other given module, etc. "Don't remove it."

Some points about module−info:

- Is provided by the kernel rpms (built by anaconda−runtime*.rpm)
- Is a link to module−info−[lcub]kernel−version[rcub]
- Contains information about all available modules (at least those included in the default kernel config.)
- Important to anaconda − in anaconda/utils/modlist command.
- Might be used by kudzu to determine default parameters for modules when it creates entries in /etc/modules.conf. If you move module−info out of the way, shut down, install a new network card, and re−boot then kudzu would complain loudly. Look at the kudzu source code.

10.7. config

Everytime you compile and install the kernel image in /boot, you should also copy the corresponding config file to /boot area, for documentation and future reference. Do NOT touch or edit these files!!

```
ls -l /boot/config-*
-rw-r--r--    1 root     root        42111 Sep  4  2002 /boot/config-2.4.18-14
-rw-r--r--    1 root     root        42328 Jan 26 01:29 /boot/config-2.4.18-19.8.0
-rw-r--r--    1 root     root        51426 Jan 25 22:21 /boot/config-2.4.18-19.8.0BOOT
-rw-r--r--    1 root     root        52328 Jan 28 03:22 /boot/config-2.4.18-19.8.0-26mar2
```

10.8. grub

If you are using GRUB, then there will be 'grub' directory.

```
ls /boot/grub
device.map     ffs_stage1_5  menu.lst        reiserfs_stage1_5  stage2
e2fs_stage1_5  grub.conf     minix_stage1_5  splash.xpm.gz      vstafs_stage1_5
fat_stage1_5   jfs_stage1_5  stage1          xfs_stage1_5
```

See also Section 15 file.

10.9. System.map

System.map is a "phone directory" list of function in a particular build of a kernel. It is typically a symlink to the System.map of the currently running kernel. If you use the wrong (or no) System.map, debugging crashes is harder, but has no other effects. Without System.map, you may face minor annoyance messages.

Do NOT touch the System.map files.

```
ls -ld /boot/System.map*
lrwxrwxrwx    1 root     root           30 Jan 26 19:26 /boot/System.map -> System.map-2.
-rw-r--r--    1 root     root       501166 Sep  4  2002 /boot/System.map-2.4.18-14
-rw-r--r--    1 root     root       510786 Jan 26 01:29 /boot/System.map-2.4.18-19.8.0
-rw-r--r--    1 root     root       331213 Jan 25 22:21 /boot/System.map-2.4.18-19.8.0BOO
-rw-r--r--    1 root     root       503246 Jan 26 19:26 /boot/System.map-2.4.18-19.8.0cus
```

How The Kernel Symbol Table Is Created ? System.map is produced by 'nm vmlinux' and irrelevant or uninteresting symbols are grepped out, When you compile the kernel, this file 'System.map' is created at /usr/src/linux/System.map. Something like below:

```
nm /boot/vmlinux-2.4.18-19.8.0 > System.map
# Below is the line from /usr/src/linux/Makefile
nm vmlinux | grep -v '\(compiled\)\|\(\.o$$\)\|\( [aUw] \)\|\(\.\.ng$$\)\|\(LASH[RL]DI\)'
cp /usr/src/linux/System.map /boot/System.map-2.4.18-14   # For v2.4.18
```

From "http://www.dirac.org/linux/systemmap.html"

10.9.1. System.map

There seems to be a dearth of information about the System.map file. It's really nothing mysterious, and in the scheme of things, it's really not that important. But a lack of documentation makes it shady. It's like an earlobe; we all have one, but nobody really knows why. This is a little web page I cooked up that explains the why.

Note, I'm not out to be 100[percnt] correct. For instance, it's possible for a system to not have /proc filesystem support, but most systems do. I'm going to assume you "go with the flow" and have a fairly typical system.

Some of the stuff on oopses comes from Alessandro Rubini's "Linux Device Drivers" which is where I learned most of what I know about kernel programming.

10.9.2. What Are Symbols?

In the context of programming, a symbol is the building block of a program: it is a variable name or a function name. It should be of no surprise that the kernel has symbols, just like the programs you write. The difference is, of course, that the kernel is a very complicated piece of coding and has many, many global symbols.

10.9.3. What Is The Kernel Symbol Table?

The kernel doesn't use symbol names. It's much happier knowing a variable or function name by the variable or function's address. Rather than using size_t BytesRead, the kernel prefers to refer to this variable as (for example) c0343f20.

Humans, on the other hand, do not appreciate names like c0343f20. We prefer to use something like size_t BytesRead. Normally, this doesn't present much of a problem. The kernel is mainly written in C, so the compiler/linker allows us to use symbol names when we code and allows the kernel to use addresses when it runs. Everyone is happy.

There are situations, however, where we need to know the address of a symbol (or the symbol for an address). This is done by a symbol table, and is very similar to how gdb can give you the function name from a address (or an address from a function name). A symbol table is a listing of all symbols along with their address. Here is an example of a symbol table:

```
c03441a0 B dmi_broken
c03441a4 B is_sony_vaio_laptop
c03441c0 b dmi_ident
c0344200 b pci_bios_present
c0344204 b pirq_table
c0344208 b pirq_router
c034420c b pirq_router_dev
c0344220 b ascii_buffer
c0344224 b ascii_buf_bytes
```

You can see that the variable named dmi_broken is at the kernel address c03441a0.

10.9.4. What Is The System.map File?

There are 2 files that are used as a symbol table:

1. /proc/ksyms
2. System.map

There. You now know what the System.map file is.

Every time you compile a new kernel, the addresses of various symbol names are bound to change.

/proc/ksyms is a "proc file" and is created on the fly when a kernel boots up. Actually, it's not really a file; it's simply a representation of kernel data which is given the illusion of being a disk file. If you don't believe me, try finding the filesize of /proc/ksyms. Therefore, it will always be correct for the kernel that is currently running..

However, System.map is an actual file on your filesystem. When you compile a new kernel, your old System.map has wrong symbol information. A new System.map is generated with each kernel compile and you need to replace the old copy with your new copy.

10.9.5. What Is An Oops?

What is the most common bug in your homebrewed programs? The segfault. Good ol' signal 11.

What is the most common bug in the Linux kernel? The segfault. Except here, the notion of a segfault is much more complicated and can be, as you can imagine, much more serious. When the kernel dereferences an invalid pointer, it's not called a segfault −− it's called an "oops". An oops indicates a kernel bug and should always be reported and fixed.

Note that an oops is not the same thing as a segfault. Your program cannot recover from a segfault. The kernel doesn't necessarily have to be in an unstable state when an oops occurs. The Linux kernel is very robust; the oops may just kill the current process and leave the rest of the kernel in a good, solid state.

An oops is not a kernel panic. In a panic, the kernel cannot continue; the system grinds to a halt and must be restarted. An oops may cause a panic if a vital part of the system is destroyed. An oops in a device driver, for example, will almost never cause a panic.

When an oops occurs, the system will print out information that is relevent to debugging the problem, like the contents of all the CPU registers, and the location of page descriptor tables. In particular, the contents of the EIP (instruction pointer) is printed. Like this:

```
EIP: 0010:[<00000000>]
Call Trace: [<c010b860>]
```

10.9.6. What Does An Oops Have To Do With System.map?

You can agree that the information given in EIP and Call Trace is not very informative. But more importantly, it's really not informative to a kernel developer either. Since a symbol doesn't have a fixed address, c010b860 can point anywhere.

To help us use this cryptic oops output, Linux uses a daemon called klogd, the kernel logging daemon. klogd intercepts kernel oopses and logs them with syslogd, changing some of the useless information like c010b860 with information that humans can use. In other words, klogd is a kernel message logger which can perform name−address resolution. Once klogd tranforms the kernel message, it uses whatever logger is in place to log system wide messages, usually syslogd.

To perform name−address resolution, klogd uses System.map. Now you know what an oops has to do with System.map.

Fine print: There are actually two types of address resolution are performed by klogd.

- Static translation, which uses the System.map file.
- Dynamic translation which is used with loadable modules, doesn't use

System.map and is therefore not relevant to this discussion, but I'll describe it briefly anyhow.

Klogd Dynamic Translation

Suppose you load a kernel module which generates an oops. An oops message is generated, and klogd intercepts it. It is found that the oops occured at d00cf810. Since this address belongs to a dynamically loaded module, it has no entry in the System.map file. klogd will search for it, find nothing, and conclude that a loadable module must have generated the oops. klogd then queries the kernel for symbols that were exported by loadable modules. Even if the module author didn't export his symbols, at the very least, klogd will know what module generated the oops, which is better than knowing nothing about the oops at all.

There's other software that uses System.map, and I'll get into that shortly.

10.9.7. Where Should System.map Be Located?

System.map should be located wherever the software that uses it looks for it. That being said, let me talk about where klogd looks for it. Upon bootup, if klogd isn't given the location of System.map as an argument, it will look for System.map in 3 places, in the following order:

1. /boot/System.map
2. /System.map
3. /usr/src/linux/System.map

System.map also has versioning information, and klogd intelligently searches for the correct map file. For instance, suppose you're running kernel 2.4.18 and the associated map file is /boot/System.map. You now compile a new kernel 2.5.1 in the tree /usr/src/linux. During the compiling process, the file /usr/src/linux/System.map is created. When you boot your new kernel, klogd will first look at /boot/System.map, determine it's not the correct map file for the booting kernel, then look at /usr/src/linux/System.map, determine that it is the correct map file for the booting kernel and start reading the symbols.

A few nota bene's:

- Somewhere during the 2.5.x series, the Linux kernel started to untar into linux−version, rather than just linux (show of hands −− how many people have been waiting for this to happen?). I don't know if klogd has been modified to search in /usr/src/linux−version/System.map yet. TODO: Look at the

klogd srouce. If someone beats me to it, please email me and let me know if klogd has been modified to look in the new directory name for the linux source code.

- The man page doesn't tell the whole the story. Look at this:

```
# strace -f /sbin/klogd | grep 'System.map'
31208 open("/boot/System.map-2.4.18", O_RDONLY|O_LARGEFILE) = 2
```

Apparently, not only does klogd look for the correct version of the map in the 3 klogd search directories, but klogd also knows to look for the name "System.map" followed by "-kernelversion", like System.map-2.4.18. This is undocumented feature of klogd.

A few drivers will need System.map to resolve symbols (since they're linked against the kernel headers instead of, say, glibc). They will not work correctly without the System.map created for the particular kernel you're currently running. This is NOT the same thing as a module not loading because of a kernel version mismatch. That has to do with the kernel version, not the kernel symbol table which changes between kernels of the same version!

10.9.8. What else uses the System.map

Don't think that System.map is only useful for kernel oopses. Although the kernel itself doesn't really use System.map, other programs such as klogd, lsof,

```
satan# strace lsof 2>&1 1> /dev/null | grep System
readlink("/proc/22711/fd/4", "/boot/System.map-2.4.18", 4095) = 23
```

and ps :

```
satan# strace ps 2>&1 1> /dev/null | grep System
open("/boot/System.map-2.4.18", O_RDONLY|O_NONBLOCK|O_NOCTTY) = 6
```

and many other pieces of software like dosemu require a correct System.map.

10.9.9. What Happens If I Don't Have A Healthy System.map?

Suppose you have multiple kernels on the same machine. You need a separate System.map files for each kernel! If boot a kernel that doesn't have a System.map file, you'll periodically see a message like: System.map does not match actual kernel Not a fatal error, but can be annoying to see everytime you do a ps ax. Some software, like dosemu, may not work correctly (although I don't know of anything off the top of my head). Lastly, your klogd or ksymoops output will not be reliable in case of a kernel oops.

10.9.10. How Do I Remedy The Above Situation?

The solution is to keep all your System.map files in /boot and rename them with the kernel version. Suppose you have multiple kernels like:

- /boot/vmlinuz-2.2.14
- /boot/vmlinuz-2.2.13

Then just rename your map files according to the kernel version and put them in /boot, like:

```
/boot/System.map−2.2.14
/boot/System.map−2.2.13
```

Now what if you have two copies of the same kernel? Like:

- /boot/vmlinuz−2.2.14
- /boot/vmlinuz−2.2.14.nosound

The best answer would be if all software looked for the following files:

```
/boot/System.map−2.2.14
/boot/System.map−2.2.14.nosound
```

You can also use symlinks:

```
System.map−2.2.14
System.map−2.2.14.sound
ln −s System.map−2.2.14.sound System.map     # Here System.map −> System.map−2.2.14.so
```

11. Advanced Topics − Linux Boot Process

This section may not be interesting for 'average Joe home PC user' but will be more directed towards someone with computer science background.

The chain of events at boot are: CPU−> VGA−> Power−On−Self−Test−> SCSI−> Boot Manager−> Lilo boot loader−> kernel−> init−> bash. The firmware and software programs output various messages as the computer and Linux come to life.

A guided tour of a Linux Boot process:

1. The Motherboard BIOS Triggers the Video Display Card BIOS Initialization
2. Motherboard BIOS Initializes Itself
3. SCSI Controller BIOS Initializes
4. Hardware Summary: The motherboard BIOS then displays the following summary of its hardware inventory. And runs its Virus checking code that looks for changed boot sectors.
5. BootManager Menu : The Master Boot Record (MBR) on the first hard disk is read, by DOS tradition, into address 0x00007c00, and the processor starts executing instructions there. This MBR boot code loads the first sector of code on the active DOS partition.
6. Lilo is started: If the Linux selection is chosen and if Linux has been installed with Lilo, Lilo is loaded into address 0x00007c00. Lilo prints LILO with its progress revealed by individually printing the letters. The first "L" is printed after Lilo moves itself to a better location at 0x0009A000. The "I" is printed just before it starts its secondary boot loader code. Lilo's secondary boot loader prints the next "L", loads descriptors pointing to parts of the kernel, and then prints the final "O". The descriptors are placed at 0x0009d200. The boot message and a prompt line, if specified, are printed. The pressing "Tab" at the prompt, allows the user to specify a system and to provide command−line specifications to the Linux Kernel, its drivers, and the "init" program. Also, environment variables may be defined at this point.

```
The following line is from /boot/message:
>
>
>
 Press  to list available boot image labels.
The following line is the prompt from /sbin/lilo:
boot:
Note: If Lilo is not used, then the boot code built into the head
      of the Linux kernel, linux/arch/i386/boot/bootsect.S
          prints "Loading" and continues.
Lilo displays the following as it loads the kernel code. It gets the
text "Linux−2.2.12" from the "label=..." specification in lilo.conf.
Loading linux−2.2.12.........
```

7. The kernel code in /linux/arch/i386/boot/setup.S arranges the transition from the processor running in real mode (DOS mode) to protected mode (full 32−bit mode). Blocks of code named Trampoline.S and Trampoline32.S help with the transition. Small kernel images (zImage) are decompressed and loaded at 0x00010000. Large kernel images (bzImage) are loaded instead at 0x00100000. This code sets up the registers, decompresses the compressed kernel (which has linux/arch/i386/head.S at its start), printing the following 2 lines from linux/arch/i386/boot/compressed/misc.c Uncompressing Linux... Ok. Booting the kernel. The i386−specific setup.S code has now completed its job and it jumps to 0x00010000 (or 0x00100000) to start the generic Linux kernel code.

    - Processor, Console, and Memory Initialization : This runs linux/arch/i386/head.S which in

turn jumps to start_kernel(void) in linux/init/main.c where the interrupts are redefined. linux/kernel/module.c then loads the drivers for the console and pci bus. From this point on the kernel messages are also saved in memory and available using /bin/dmesg. They are then usually transferred to /var/log/message for a permanent record.

- PCI Bus Initialization : mpci_init() in linux/init/main.c causes the following lines from linux/arch/i386/kernel/bios32.c to be printed:
- Network Initialization: socket_init() in linux/init/main.c causes the following network initializations:

```
linux/net/socket.c prints:
Linux NET4.0 for Linux 2.2
Based upon Swansea University Computer Society NET3.039
linux/net/unix/af_unix.c prints:
NET4: Unix domain sockets 1.0 for Linux NET4.0.
linux/net/ipv4/af_inet.c prints:
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
linux/net/ipv4/ip_gre.c prints:
GRE over IPv4 tunneling driver
linux/net/core/dev.c prints:
early initialization of device gre0 is deferred
linux/net/core/rtnetlink.c prints:
Initializing RT netlink socket
```

- The Kernel Idle Thread (Process 0) is Started : At this point a kernel thread is started running init() which is one of the routines defined in linux/init/main.c. This init() must not be confused with the program /sbin/init that will be run after the Linux kernel is up and running. mkswapd_setup() in linux/init/main.c causes the following line from linux/mm/vmscan.c to be printed: Starting kswapd v 1.5
- Device Driver Initialization : The kernel routine linux/arch/i386/kernel/setup.c then initializes devices and file systems (built into the kernel??). It produces the following lines and then forks to run /sbin/init:
    - ◊ Generic Parallel Port Initialization : The parallel port initialization routine linux/drivers/misc/parport_pc.c prints the following:
    - ◊ Character Device Initializations : The following 3 lines are from linux/drivers/char/serial.c:
    - ◊ Block Device Initializations : linux/drivers/block/rd.c prints: RAM disk driver initialized: 16 RAM disks of 8192K size linux/drivers/block/loop.c prints: loop: registered device at major 7 linux/drivers/block/floppy.c prints: Floppy drive(s): fd0 is 1.44M, fd1 is 1.44M FDC 0 is a post−1991 82077
    - ◊ SCSI Bus Initialization: The following lines are from aic7xxx.c, scsi.c, sg.c, sd.c or sr.c in the subdirectory linux/drivers/scsi:
- Initialization of Kernel Support for Point−to−Point Protocol : The following initialization is done by linux/drivers/net/ppp.c.
- Examination of Fixed Disk Arrangement : The following lines are from linux/drivers/block/genhd.c:

8. Init Program (Process 1) Startup : The program /sbin/init is started by the "idle" process (Process 0) code in linux/init/main.c and becomes process 1. /sbin/init then completes the initialization by running scripts and forking additional processes as specified in /etc/inittab. It starts by printing: INIT: version 2.76 booting and reads /etc/inittab.
9. The Bash Shell is Started : The bash shell, /bin/bash is then started up. Bash initialization begins by executing script in /etc/profile which set the system−wide environment variables:

11.1. References for Boot Process

Refer to following resources :

- The Linux Boot Process
- Bootdisks and Boot Process
- Linux Boot Process − by San Gabreil LUG
- Boot Process (Netmag)
- Boot Process (LUG Victoria)

---

12. Other Formats of this Document

This section is written by Al Dev (at site "http://www.milkywaygalaxy.freeservers.com" mirrors at angelfire , geocities , virtualave , Fortunecity , Freewebsites , Tripod , 101xs , 50megs)

This document is published in 14 different formats namely − DVI, Postscript, Latex, Adobe Acrobat PDF, LyX, GNU−info, HTML, RTF(Rich Text Format), Plain−text, Unix man pages, single HTML file, SGML (Linuxdoc format), SGML (Docbook format), MS WinHelp format.

This howto document is located at −

- "http://www.linuxdoc.org" and click on HOWTOs and search for howto document name using CTRL+f or ALT+f within the web−browser.

You can also find this document at the following mirrors sites −

- "http://www.caldera.com/LDP/HOWTO"
- "http://www.linux.ucla.edu/LDP"
- "http://www.cc.gatech.edu/linux/LDP"
- "http://www.redhat.com/mirrors/LDP"
- Other mirror sites near you (network−address−wise) can be found at "http://www.linuxdoc.org/mirrors.html" select a site and go to directory /LDP/HOWTO/xxxxx−HOWTO.html

- You can get this HOWTO document as a single file tar ball in HTML, DVI, Postscript or SGML formats from − "ftp://www.linuxdoc.org/pub/Linux/docs/HOWTO/other−formats/" and "http://www.linuxdoc.org/docs.html#howto"
- Plain text format is in: "ftp://www.linuxdoc.org/pub/Linux/docs/HOWTO" and "http://www.linuxdoc.org/docs.html#howto"
- Single HTML file format is in: "http://www.linuxdoc.org/docs.html#howto" Single HTML file can be created with command (see man sgml2html) − sgml2html −split 0 xxxxhowto.sgml
- Translations to other languages like French, German, Spanish, Chinese, Japanese are in "ftp://www.linuxdoc.org/pub/Linux/docs/HOWTO" and "http://www.linuxdoc.org/docs.html#howto" Any help from you to translate to other languages is welcome.

The document is written using a tool called "SGML−Tools" which can be got from − "http://www.sgmltools.org" Compiling the source you will get the following commands like

- sgml2html xxxxhowto.sgml (to generate html file)
- sgml2html −split 0 xxxxhowto.sgml (to generate a single page html file)
- sgml2rtf xxxxhowto.sgml (to generate RTF file)
- sgml2latex xxxxhowto.sgml (to generate latex file)

12.1. Acrobat PDF format

PDF file can be generated from postscript file using either acrobat *distill* or *Ghostscript* . And postscript file is generated from DVI which in turn is generated from LaTex file. You can download distill software from "http://www.adobe.com" . Given below is a sample session:

```
bash$ man sgml2latex
bash$ sgml2latex filename.sgml
bash$ man dvips
bash$ dvips −o filename.ps filename.dvi
bash$ distill filename.ps
bash$ man ghostscript
bash$ man ps2pdf
bash$ ps2pdf input.ps output.pdf
bash$ acroread output.pdf &
```

Or you can use Ghostscript command *ps2pdf* . ps2pdf is a work−alike for nearly all the functionality of Adobe's Acrobat Distiller product: it converts PostScript files to Portable Document Format (PDF) files. *ps2pdf* is implemented as a very small command script (batch file) that invokes Ghostscript, selecting a special "output device" called *pdfwrite* . In order to use ps2pdf, the pdfwrite device must be included in the makefile when Ghostscript was compiled; see the documentation on building Ghostscript for details.

12.2. Convert Linuxdoc to Docbook format

This document is written in linuxdoc SGML format. The Docbook SGML format supercedes the linuxdoc format and has lot more features than linuxdoc. The linuxdoc is very simple and is easy to use. To convert linuxdoc SGML file to Docbook SGML use the program *ld2db.sh* and some perl scripts. The ld2db output is not 100[percnt] clean and you need to use the *clean_ld2db.pl* perl script. You may need to manually correct few lines in the document.

- Download ld2db program from "http://www.dcs.gla.ac.uk/~rrt/docbook.html" or from Milkyway Galaxy site
- Download the cleanup_ld2db.pl perl script from from Milkyway Galaxy site

The ld2db.sh is not 100[percnt] clean, you will get lot of errors when you run

```
bash$ ld2db.sh file−linuxdoc.sgml db.sgml
bash$ cleanup.pl db.sgml > db_clean.sgml
bash$ gvim db_clean.sgml
bash$ docbook2html db.sgml
```

And you may have to manually edit some of the minor errors after running the perl script. For e.g. you may need to put closing tag < /Para> for each < Listitem>

12.3. Convert to MS WinHelp format

You can convert the SGML howto document to Microsoft Windows Help file, first convert the sgml to html using:

```
bash$ sgml2html xxxxhowto.sgml     (to generate html file)
bash$ sgml2html −split 0   xxxxhowto.sgml (to generate a single page html file)
```

Then use the tool HtmlToHlp . You can also use sgml2rtf and then use the RTF files for generating winhelp files.

12.4. Reading various formats

In order to view the document in dvi format, use the xdvi program. The xdvi program is located in tetex−xdvi*.rpm package in Redhat Linux which can be located through ControlPanel [verbar] Applications [verbar] Publishing [verbar] TeX menu buttons. To read dvi document give the command −

```
xdvi −geometry 80x90 howto.dvi man xdvi
```

And resize the window with mouse. To navigate use Arrow keys, Page Up, Page Down keys, also you can use 'f', 'd', 'u', 'c', 'l', 'r', 'p', 'n' letter keys to move up, down, center, next page, previous page etc. To turn off expert menu press 'x'.

You can read postscript file using the program 'gv' (ghostview) or 'ghostscript'. The ghostscript program is in ghostscript*.rpm package and gv program is in gv*.rpm package in Redhat Linux which can be located through ControlPanel [verbar] Applications [verbar] Graphics menu buttons. The gv program is much more user friendly than ghostscript. Also ghostscript and gv are available on other platforms like OS/2, Windows 95 and NT, you view this document even on those platforms.

- Get ghostscript for Windows 95, OS/2, and for all OSes from "http://www.cs.wisc.edu/~ghost"

To read postscript document give the command −

```
gv howto.ps ghostscript howto.ps
```

You can read HTML format document using Netscape Navigator, Microsoft Internet explorer, Redhat Baron Web browser or any of the 10 other web browsers.

You can read the latex, LyX output using LyX a X−Windows front end to latex.

13. Appendix A − Creating initrd.img file

The *initrd* is the "initial ramdisk". It is enough files stored in a ramdisk to store needed drivers . You need the drivers so that the kernel can mount / and kick off init.

You can avoid this file 'initrd.img' and eliminate the need of 'initrd.img', if you build your scsi drivers right into the kernel, instead of into modules. (Many persons recommend this).

13.1. Using mkinitrd

The mkinitrd utility creates an initrd image in a single command. This is command is peculiar to RedHat. There may be equivalent command of mkinitrd in other distributions of Linux. This is very convenient utility.

You can read the mkinitrd man page.

```
/sbin/mkinitrd --help   # Or simply type 'mkinitrd --help'
usage: mkinitrd [--version] [-v] [-f] [--preload <module>]
       [--omit-scsi-modules] [--omit-raid-modules] [--omit-lvm-modules]
       [--with=<module>] [--image-version] [--fstab=<fstab>] [--nocompress]
       [--builtin=<module>] [--nopivot] <initrd-image> <kernel-version>
       (example: mkinitrd /boot/initrd-2.2.5-15.img 2.2.5-15)
# Read the online manual page with .....
man mkinitrd
su - root
# The command below creates the initrd image file
mkinitrd  ./initrd-2.4.18-19.8.0custom.img   2.4.18-19.8.0custom
ls -l initrd-2.4.18-19.8.0custom.img
-rw-r--r--    1 root     root       127314 Mar 19 21:54 initrd-2.4.18-19.8.0custom.img
cp  ./initrd-2.4.18-19.8.0custom.img   /boot
```

See the following sections for the manual method of creating an initrd image.

13.2. Kernel Docs

To create /boot/initrd.img see the documentation at /usr/src/linux/Documentation/initrd.txt and see also Loopback−Root−mini−HOWTO .

13.3. Linuxman Book

A cut from "http://www.linuxman.com.cy/rute/node1.html" chapter 31.7.

SCSI Installation Complications and initrd

Some of the following descriptions may be difficult to understand without knowledge of kernel modules explained in Chapter 42. You may want to come back to it later.

Consider a system with zero IDE disks and one SCSI disk containing a LINUX installation. There are BIOS interrupts to read the SCSI disk, just as there were for the IDE, so LILO can happily access a kernel image somewhere inside the SCSI partition. However, the kernel is going to be lost without a kernel module [lsqb

]See Chapter 42. The kernel doesn't support every possible kind of hardware out there all by itself. It is actually divided into a main part (the kernel image discussed in this chapter) and hundreds of modules (loadable parts that reside in /lib/modules/) that support the many type of SCSI, network, sound etc., peripheral devices.] that understands the particular SCSI driver. So although the kernel can load and execute, it won't be able to mount its root file system without loading a SCSI module first. But the module itself resides in the root file system in /lib/modules/. This is a tricky situation to solve and is done in one of two ways: either (a) using a kernel with preenabled SCSI support or (b) using what is known as an initrd preliminary root file system image.

The first method is what I recommend. It's a straightforward (though time−consuming) procedure to create a kernel with SCSI support for your SCSI card built−in (and not in a separate module). Built−in SCSI and network drivers will also autodetect cards most of the time, allowing immediate access to the device−−they will work without being given any options [lsqb]Discussed in Chapter 42.] and, most importantly, without your having to read up on how to configure them. This setup is known as compiled−in support for a hardware driver (as opposed to module support for the driver). The resulting kernel image will be larger by an amount equal to the size of module. Chapter 42 discusses such kernel compiles.

The second method is faster but trickier. LINUX supports what is known as an initrd image (initial rAM disk image). This is a small, +1.5 megabyte file system that is loaded by LILO and mounted by the kernel instead of the real file system. The kernel mounts this file system as a RAM disk, executes the file /linuxrc, and then only mounts the real file system.

31.6 Creating an initrd Image

Start by creating a small file system. Make a directory [nbsp]/initrd and copy the following files into it.

```
drwxr−xr−x    7 root     root         1024 Sep 14 20:12 initrd/
drwxr−xr−x    2 root     root         1024 Sep 14 20:12 initrd/bin/
−rwxr−xr−x    1 root     root       436328 Sep 14 20:12 initrd/bin/insmod
−rwxr−xr−x    1 root     root       424680 Sep 14 20:12 initrd/bin/sash
drwxr−xr−x    2 root     root         1024 Sep 14 20:12 initrd/dev/
crw−r−−r−−    1 root     root       5,   1 Sep 14 20:12 initrd/dev/console
crw−r−−r−−    1 root     root       1,   3 Sep 14 20:12 initrd/dev/null
brw−r−−r−−    1 root     root       1,   1 Sep 14 20:12 initrd/dev/ram
crw−r−−r−−    1 root     root       4,   0 Sep 14 20:12 initrd/dev/systty
crw−r−−r−−    1 root     root       4,   1 Sep 14 20:12 initrd/dev/tty1
crw−r−−r−−    1 root     root       4,   1 Sep 14 20:12 initrd/dev/tty2
crw−r−−r−−    1 root     root       4,   1 Sep 14 20:12 initrd/dev/tty3
crw−r−−r−−    1 root     root       4,   1 Sep 14 20:12 initrd/dev/tty4
drwxr−xr−x    2 root     root         1024 Sep 14 20:12 initrd/etc/
drwxr−xr−x    2 root     root         1024 Sep 14 20:12 initrd/lib/
−rwxr−xr−x    1 root     root           76 Sep 14 20:12 initrd/linuxrc
drwxr−xr−x    2 root     root         1024 Sep 14 20:12 initrd/loopfs/
```

On my system, the file initrd/bin/insmod is the statically linked [lsqb]meaning it does not require shared libraries.] version copied from /sbin/insmod.static−−a member of the modutils−2.3.13 package. initrd/bin/sash is a statically linked shell from the sash−3.4 package. You can recompile insmod from source if you don't have a statically linked version. Alternatively, copy the needed DLLs from /lib/ to initrd/lib/. (You can get the list of required DLLs by running ldd /sbin/insmod. Don't forget to also copy symlinks and run strip −s [lcub]lib[rcub] to reduce the size of the DLLs.)

Now copy into the initrd/lib/ directory the SCSI modules you require. For example, if we have an Adaptec AIC−7850 SCSI adapter, we would require the aic7xxx.o module from /lib/modules/[lcub]version[rcub

]/scsi/aic7xxx.o. Then, place it in the initrd/lib/ directory.

```
−rw−r−−r−−    1 root     root       129448 Sep 27  1999 initrd/lib/aic7xxx.o
```

The file initrd/linuxrc should contain a script to load all the modules needed for the kernel to access the SCSI partition. In this case, just the aic7xxx module [lsqb] insmod can take options such as the IRQ and IO−port for the device. See Chapter 42.]:

```
#!/bin/sash

aliasall

echo "Loading aic7xxx module"
insmod /lib/aic7xxx.o
```

Now double−check all your permissions and then chroot to the file system for testing.

```
chroot ~/initrd /bin/sash
/linuxrc
```

Now, create a file system image similar to that in Section 19.9:

```
dd if=/dev/zero of=~/file−inird count=2500 bs=1024
losetup /dev/loop0 ~/file−inird
mke2fs /dev/loop0
mkdir ~/mnt
mount /dev/loop0 ~/mnt
cp −a initrd/* ~/mnt/
umount ~/mnt
losetup −d /dev/loop0
```

Finally, gzip the file system to an appropriately named file:

```
gzip −c ~/file−inird > initrd−<kernel−version>
```

31.7 Modifying lilo.conf for initrd

Your lilo.conf file can be changed slightly to force use of an initrd file system. Simply add the initrd option. For example:

```
boot=/dev/sda
prompt
timeout = 50
compact
vga = extended
linear
image = /boot/vmlinuz−2.2.17
        initrd = /boot/initrd−2.2.17
        label = linux
        root = /dev/sda1
        read−only
```

Notice the use of the linear option. This is a BIOS trick that you can read about in lilo(5). It is often necessary but can make SCSI disks nonportable to different BIOSs (meaning that you will have to rerun lilo if you move the disk to a different computer).

---

14. Appendix B − Sample lilo.conf

See also Section 15 file.

Always give a date extension to the filename, because it tells you when you built the kernel, as shown below:

```
bash# man lilo
bash# man lilo.conf
And edit /etc/lilo.conf file and put these lines −
        image=/boot/bzImage.myker.26mar2001
        label=myker
        root=/dev/hda1
        read-only
You can check device name for 'root=' with the command −
        bash# df   /
Now give −
        bash# lilo
        bash# lilo -q
```

You must re−run lilo even if the entry 'myker' exists, everytime you create a new bzImage.

Given below is a sample /etc/lilo.conf file. You should follow the naming conventions like ker2217 (for kernel 2.2.17), ker2214 (for kernel 2.2.14). You can have many kernel images on the same /boot system. On my machine I have something like:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
default=firewall
image=/boot/vmlinuz-2.2.14-5.0
        label=ker2214
        read-only
        root=/dev/hda9
image=/boot/vmlinuz-2.2.17-14
        label=ker2217
        read-only
        root=/dev/hda9
#image=/usr/src/linux/arch/i386/boot/bzImage
#       label=myker
#       root=/dev/hda7
#       read-only
image=/boot/bzImage.myker.11feb2001
        label=myker11feb
        root=/dev/hda9
        read-only
image=/boot/bzImage.myker.01jan2001
        label=myker01jan
        root=/dev/hda9
        read-only
image=/boot/bzImage.myker-firewall.16mar2001
        label=firewall
        root=/dev/hda9
        read-only
```

15. Appendix C − GRUB Details And A Sample grub.conf

See

- "http://www.tldp.org/HOWTO/Linux+Win9x+Grub−HOWTO/intro.html"
- GNU GRUB "http://www.gnu.org/software/grub"
- Redhat Manual .
- Multiboot−with−GRUB minihowto
- Grub Manual

```
bash# man grub
bash# man grubby   # (command line tool for configuring grub, lilo, and elilo)
bash# man grub-install
```

Edit the file /etc/grub.conf to make entries for the new kernel. See the sample file below:

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You do not have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /, eg.
#          root (hd0,8)
#          kernel /boot/vmlinuz-version ro root=/dev/hda9
#          initrd /boot/initrd-version.img
#boot=/dev/hda
# By default boot the second entry
default=1
# Fallback to the first entry.
fallback 0
# Boot automatically after 2 minutes
timeout=120
splashimage=(hd0,8)/boot/grub/splash.xpm.gz
title Windows 2000
unhide (hd0,0)
hide (hd0,1)
hide (hd0,2)
rootnoverify (hd0,0)
chainloader +1
makeactive
title Red Hat Linux (2.4.18-19.8.0.19mar2003)
        root (hd0,8)
        kernel /boot/bzImage.2.4.18-19.8.0.19mar2003 ro root=LABEL=/ hdd=ide-scsi
        initrd /boot/initrd-2.4.18-19.8.0custom.img.19mar03
title Red Hat Linux (2.4.18-19.8.0custom)
        root (hd0,8)
        kernel /boot/vmlinuz-2.4.18-19.8.0custom ro root=LABEL=/ hdd=ide-scsi
        initrd /boot/initrd-2.4.18-19.8.0custom.img
title Red Hat Linux (2.4.18-14)
        root (hd0,8)
        kernel /boot/vmlinuz-2.4.18-14 ro root=LABEL=/ hdd=ide-scsi
        initrd /boot/initrd-2.4.18-14.img
title MyKernel.26jan03 (Red Hat Linux 2.4.18-14)
        root (hd0,8)
        kernel /boot/bzImage.myker.26jan03 ro root=LABEL=/ hdd=ide-scsi
        initrd /boot/initrd-2.4.18-19.8.0.img
title Windows 98
```

```
        hide (hd0,0)
        hide (hd0,1)
        unhide (hd0,2)
        rootnoverify (hd0,2)
        chainloader +1
        makeactive
        title DOS 6.22
        hide (hd0,0)
        unhide (hd0,1)
        hide (hd0,2)
        rootnoverify (hd0,1)
        chainloader +1
        makeactive
        title Partition 2 (floppy)
        hide (hd0,0)
        unhide (hd0,1)
        hide (hd0,2)
        chainloader (fd0)+1
        title Partition 3 (floppy)
        hide (hd0,0)
        hide (hd0,1)
        unhide (hd0,2)
        chainloader (fd0)+1
```

16. Appendix D − Post Kernel Building

After successfully building and booting the Linux kernel, you may be required to do these additional steps to make some of the devices to work with Linux. (The steps below were tested on Redhat Linux but should work with other distributions as well.)

Video card/Monitor configuration:

- Please see the video card manual which is usually shipped with the PC. You should look for a "Technical Specifications" page.
- Please see the monitor's manual and look for a "Technical Specifications" page.

If you are using latest version of Linux (2.4 or later) and inside KDE/GNOME desktop click on Start−>"System Settings"−>Display.

For older versions of Linux follow the steps below:

You can configure the Video card and monitor by using these commands:

```
bash$ su − root
bash# man Xconfigurator
bash# /usr/bin/X11/Xconfigurator −−help
bash# /usr/bin/X11/Xconfigurator
bash# /usr/bin/X11/Xconfigurator −−expert
See also:
bash# man xf86config
bash# /usr/bin/X11/xf86config
```

If your card is not detected automatically, then you can use the −−expert option and select the "Unlisted card". If your monitor is not listed then select the generic monitor type SVGA 1024x768.

Sound card configuration:

- Connect your external speakers to the sound card's audio port.
- Connect your CDROM audio wire to sound card's audio 4−pin socket. (Otherwise your cdrom drive will not play the music from your music cd)
- Refer to HOWTO docs on 'Sound' at "http://www.linuxdoc.org"

If you are using latest version of Linux (2.4 or later) and inside KDE/GNOME desktop click on Start−>"System Settings"−>Soundcard Detection.

For older versions of Linux follow the steps below:

```
bash$ su − root
bash# man sndconfig
bash# /usr/sbin/sndconfig
```

Then start X−window 'KDE desktop' with 'startx' command. Click on 'K Start−>ControlCenter−>SoundServer−>General−>Test Sound'. This should play the test sound. Then click on 'K Start−>MultiMedia−>SoundMixer−>SoundVolumeSlider' and adjust the sound volume.

Network card configuration: If you are using latest version of Linux (2.4 or later) and inside KDE/GNOME desktop click on Start−>"System Settings"−>Network.

For older versions of Linux follow the steps below:

- Use /sbin/linuxconf
- Or use KDE control panel
- Refer to HOWTO docs on 'Networking' at "http://www.linuxdoc.org"

Configure Firewall and IP Masquerading : For Linux kernel version 2.4 and above, the firewall and IP Masquerading is implemented by NetFilter package. Hence in kernel config you should enable Netfilter and run the Firewall/IPMasq script. Download the scripts from Firewall−IPMasq scripts , main page of Netfilter is at "http://netfilter.samba.org" . Related materials at firewalling−matures and Netfilter−FAQ .

For kernel version below 2.4 you should install the firewall rpms from rpmfind.net or firewall.src.rpm .

Configuration of other devices: Refer to HOWTO docs relating to your devices at "http://www.linuxdoc.org"

17. Appendix E − Troubleshoot Common Mistakes

17.1. Compiles OK but does not boot

If the kernel compiles ok but booting never works and it always complains with a kernel panic about /sbin/modprobe.

Solution: You did not create initrd image file. See the Appendix A at Section 13 . Also, you must do 'make modules' and 'make modules_install' in addition to creating the initrd image file.

17.2. The System Hangs at LILO

Sympton: After you build the kernel and reboot, the system hangs just before LILO.

Reason: Probably you did not set the BIOS to pick up the proper Primary Master IDE and Secondary Slave IDE hard disk partition.

Solution: Power on the machine and press DEL key to do setup of the BIOS (Basic Input Output system). Select the IDE settings and set proper primary hard disk partition and slave drives. When the system boots it looks for the primary IDE hard disk and the Master Boot Record partition. It reads the MBR and starts loading the Linux Kernel from the hard disk partition.

17.3. No init found

The following mistake is commited very frequently by new users.

If your new kernel does not boot and you get −

```
Warning: unable to open an initial console
Kernel panic: no init found. Try passing init= option to kernel
```

The problem is that you *did not* set the "root=" parameter properly in the /etc/lilo.conf. In my case, I used root=/dev/hda1 which is having the root partition "/". You must properly point the root device in your lilo.conf, it can be like /dev/hdb2 or /dev/hda7.

The kernel looks for the init command which is located in /sbin/init. And /sbin directory lives on the root partition. For details see −

```
bash# man init
```

See the Section 15 file and see the Section 14 .

17.4. Lot of Compile Errors

The 'make', 'make bzImage', 'make modules' or 'make modules_install' gives compile problems. You should give 'make mrproper' before doing make.

```
bash# make mrproper
```

If this problem persists, then try menuconfig instead of xconfig. Sometimes GUI version xconfig causes some problems:

```
bash# export TERM=VT100
bash# make menuconfig
```

17.5. The 'depmod' gives "Unresolved symbol error messages"

When you run `depmod` it gives "Unresolved symbols". A sample error message is given here to demonstrate the case:

```
bash$ su − root
bash# man depmod
bash# depmod
depmod: *** Unresolved symbols in /lib/modules/version/kernel/drivers/md/linear.o
depmod: *** Unresolved symbols in /lib/modules/version/kernel/drivers/md/multipath.o
depmod: *** Unresolved symbols in /lib/modules/version/kernel/drivers/md/raid0.o
depmod: *** Unresolved symbols in /lib/modules/version/kernel/drivers/md/raid1.o
depmod: *** Unresolved symbols in /lib/modules/version/kernel/drivers/md/raid5.o
```

Reason: You did not make modules and install the modules after building the new kernel with `"make bzImage"` .

Solution: After you build the new kernel, you must do:

```
bash$ su − root
bash# cd /usr/src/linux
bash# make modules
bash# make modules_install
```

17.6. Kernel Does Not Load Module − "Unresolved symbols" Error Messages

When you boot kernel and system tries to load any modules and you get "Unresolved symbol : __some_function_name" then it means that you did not clean compile the modules and kernel. It is mandatory that you should do *make clean* and make the modules. Do this −

```
bash# cd /usr/src/linux
bash# make dep
```

```
bash# make clean
bash# make mrproper
bash# nohup make bzImage &
bash# tail −f nohup.out     (.... to monitor the progress)
bash# make modules
bash# make modules_install
```

17.7. Kernel fails to load a module

If the kernel fails to load a module (say loadable module for network card or other devices), then you may want to try to build the driver for device right into the kernel. Sometimes *loadable module will NOT work* and the driver needs to be built right inside the kernel. For example − some network cards do not support loadable module feature − you MUST build the driver of the network card right into linux kernel. Hence, in 'make xconfig' you MUST not select loadable module for this device.

17.8. Loadable modules

You can install default loadable modules with −

The step given below may not be required but is needed *ONLY FOR EMERGENCIES* where your /lib/modules files are damaged. If you already have the /lib/modules directory and in case you want replace them use the −−force to replace the package and select appropriate cpu architecture.

For new versions of linux redhat linux 6.0 and later, the kernel modules are included with kernel−2.2*.rpm. Install the loadable modules and the kernel with

```
        This will list the already installed package.
bash# rpm −qa | grep −i kernel

bash# rpm −U −−force  /mnt/cdrom/Redhat/RPMS/kernel−2.2.14−5.0.i686.rpm
(or)
bash# rpm −U −−force  /mnt/cdrom/Redhat/RPMS/kernel−2.2.14−5.0.i586.rpm
(or)
bash# rpm −U −−force  /mnt/cdrom/Redhat/RPMS/kernel−2.2.14−5.0.i386.rpm
```

This is only for old versions of redhat linux 5.2 and before. Boot new kernel and install the loadable modules from RedHat Linux "contrib" cdrom

```
bash# rpm −i /mnt/cdrom/contrib/kernel−modules*.rpm
....(For old linux systems which do not have insmod pre−installed)
```

17.9. See Docs

More problems. You can read the /usr/src/linux/README (at least once) and also /usr/src/linux/Documentation.

17.10. make clean

If your new kernel does really weird things after a routine kernel upgrade, chances are you forgot to `make clean` before compiling the new kernel. Symptoms can be anything from your system outright crashing, strange I/O problems, to crummy performance. Make sure you do a `make dep`, too.

17.11. Huge or slow kernels

If your kernel is sucking up a lot of memory, is too large, and/or just takes forever to compile even when you've got your new Quadbazillium−III/4400 working on it, you've probably got lot of unneeded stuff (device drivers, filesystems, etc) configured. If you don't use it, don't configure it, because it does take up memory. The most obvious symptom of kernel bloat is extreme swapping in and out of memory to disk; if your disk is making a lot of noise and it's not one of those old Fujitsu Eagles that sound like like a jet landing when turned off, look over your kernel configuration.

You can find out how much memory the kernel is using by taking the total amount of memory in your machine and subtracting from it the amount of ``total mem'' in `/proc/meminfo` or the output of the command `` `free` ''.

17.12. The parallel port doesn't work/my printer doesn't work

Configuration options for PCs are: First, under the category `General Setup', select `Parallel port support' and `PC−style hardware'. Then under `Character devices', select `Parallel printer support'.

Then there are the names. Linux 2.2 names the printer devices differently than previous releases. The upshot of this is that if you had an `lp1` under your old kernel, it's probably an `lp0` under your new one. Use `` `dmesg` `` or look through the logs in `/var/log` to find out.

17.13. Kernel doesn't compile

If it does not compile, then it is likely that a patch failed, or your source is somehow corrupt. Your version of gcc also might not be correct, or could also be corrupt (for example, the include files might be in error). Make sure that the symbolic links which Linus describes in the `README` are set up correctly. In general, if a standard kernel does not compile, something is seriously wrong with the system, and reinstallation of certain tools is probably necessary.

In some cases, gcc can crash due to hardware problems. The error message will be something like ``xxx exited with signal 15'' and it will generally look very mysterious. I probably would not mention this, except that it happened to me once − I had some bad cache memory, and the compiler would occasionally barf at random. Try reinstalling gcc first if you experience problems. You should only get suspicious if your kernel compiles fine with external cache turned off, a reduced amount of RAM, etc.

It tends to disturb people when it's suggested that their hardware has problems. Well, I'm not making this up. There is an FAQ for it −− it's at "http://www.bitwizard.nl/sig11" .

17.14. New version of the kernel doesn't seem to boot

You did not run LILO, or it is not configured correctly. One thing that ``got'' me once was a problem in the config file; it said ` boot = /dev/hda1 ' instead of ` boot = /dev/hda ' (This can be really annoying at first, but once you have a working config file, you shouldn't need to change it.).

17.15. You forgot to run LILO, or system doesn't boot at all

Ooops! The best thing you can do here is to boot off of a floppy disk or CDROM and prepare another bootable floppy (such as ` make zdisk ' would do). You need to know where your root (/) filesystem is and what type it is (e.g. second extended, minix). In the example below, you also need to know what filesystem your /usr/src/linux source tree is on, its type, and where it is normally mounted.

In the following example, / is /dev/hda1 , and the filesystem which holds /usr/src/linux is /dev/hda3 , normally mounted at /usr . Both are second extended filesystems. The working kernel image in /usr/src/linux/arch/i386/boot is called bzImage .

The idea is that if there is a functioning bzImage , it is possible to use that for the new floppy. Another alternative, which may or may not work better (it depends on the particular method in which you messed up your system) is discussed after the example.

First, boot from a boot/root disk combo or rescue disk, and mount the filesystem which contains the working kernel image:

mkdir /mnt mount −t ext2 /dev/hda3 /mnt

If mkdir tells you that the directory already exists, just ignore it. Now, cd to the place where the working kernel image was. Note that /mnt + /usr/src/linux/arch/i386/boot − /usr = /mnt/src/linux/arch/i386/boot Place a formatted disk in drive ``A:'' (not your boot or root disk!), dump the image to the disk, and configure it for your root filesystem:

cd /mnt/src/linux/arch/i386/boot dd if=bzImage of=/dev/fd0 rdev /dev/fd0 /dev/hda1

cd to / and unmount the normal /usr filesystem:

cd / umount /mnt

You should now be able to reboot your system as normal from this floppy. Don't forget to run lilo (or whatever it was that you did wrong) after the reboot!

As mentioned above, there is another common alternative. If you happened to have a working kernel image in / (/vmlinuz for example), you can use that for a boot disk. Supposing all of the above conditions, and that my kernel image is /vmlinuz , just make these alterations to the example above: change /dev/hda3 to /dev/hda1 (the / filesystem), /mnt/src/linux to /mnt , and if=bzImage to if=vmlinuz . The note explaining how to derive /mnt/src/linux may be ignored.

Using LILO with big drives (more than 1024 cylinders) can cause problems. See the LILO mini−HOWTO or documentation for help on that.

17.16. It says `warning: bdflush not running'

This can be a severe problem. Starting with a kernel release after Linux v1.0 (around 20 Apr 1994), a program called ` update ' which periodically flushes out the filesystem buffers, was upgraded/replaced. Get the sources to ` bdflush ' (you should find it where you got your kernel source), and install it (you probably want to run your system under the old kernel while doing this). It installs itself as ` update ' and after a reboot, the new kernel should no longer complain.

17.17. I can't get my IDE/ATAPI CD−ROM drive to work

Strangely enough, lot of people cannot get their ATAPI drives working, probably because there are a number of things that can go wrong.

If your CD−ROM drive is the only device on a particular IDE interface, it must be jumpered as ``master'' or ``single.'' Supposedly, this is the most common error.

Creative Labs (for one) has put IDE interfaces on their sound cards now. However, this leads to the interesting problem that while some people only have one interface to being with, many have two IDE interfaces built−in to their motherboards (at IRQ15, usually), so a common practice is to make the soundblaster interface a third IDE port (IRQ11, or so I'm told).

This causes problems with older Linux versions like 1.3 and below. in that versions Linux don't support a third IDE interface. To get around this, you have a few choices.

If you have a second IDE port already, chances are that you are not using it or it doesn't already have two devices on it. Take the ATAPI drive off the sound card and put it on the second interface. You can then disable the sound card's interface, which saves an IRQ anyway.

If you don't have a second interface, jumper the sound card's interface (not the sound card's sound part) as IRQ15, the second interface. It should work.

17.18. It says weird things about obsolete routing requests

Get new versions of the `route` program and any other programs which do route manipulation. `/usr/include/linux/route.h` (which is actually a file in `/usr/src/linux`) has changed.

17.19. ``Not a compressed kernel Image file''

Don't use the `vmlinux` file created in `/usr/src/linux` as your boot image; `[..]/arch/i386/boot/bzImage` is the right one.

17.20. Problems with console terminal after upgrade to Linux v1.3.x

Change the word `dumb` to `linux` in the console termcap entry in `/etc/termcap` . You may also have to make a terminfo entry.

17.21. Can't seem to compile things after kernel upgrade

The linux kernel source includes a number of include files (the things that end with `.h`) which are referenced by the standard ones in `/usr/include` . They are typically referenced like this (where `xyzzy.h` would be something in `/usr/include/linux`): #include <linux/xyzzy.h> Normally, there is a link called `linux` in `/usr/include` to the `include/linux` directory of your kernel source (`/usr/src/linux/include/linux` in the typical system). If this link is not there, or points to the wrong place, most things will not compile at all. If you decided that the kernel source was taking too much room on the disk and deleted it, this will obviously be a problem. Another way it might go wrong is with file permissions; if your `root` has a umask which doesn't allow other users to see its files by default, and you extracted the kernel source without the `p` (preserve filemodes) option, those users also won't be able to use the C compiler. Although you could use the `chmod` command to fix this, it is probably easier to re−extract the include files. You can do this the same way you did the whole source at the beginning, only with an additional argument:

blah# tar zxvpf linux.x.y.z.tar.gz linux/include Note: `` `make config` '' will recreate the `/usr/src/linux` link if it isn't there.

17.22. Increasing limits

The following few *example* commands may be helpful to those wondering how to increase certain soft limits imposed by the kernel: echo 4096 > /proc/sys/kernel/file−max echo 12288 > /proc/sys/kernel/inode−max echo 300 400 500 > /proc/sys/vm/freepages