The Linux Kernel HOWTO

Table of Contents

Table of Contents

The Linux Kernel HOWTO

Brian Ward, `bri@cs.uchicago.edu`

v2.3, 23 Feb 2001

This is a detailed guide to kernel configuration, compilation, upgrades, and troubleshooting for ix86−based systems.

---

1. Introduction

2. Quick Steps − Kernel Compile

3. Important questions and their answers

4. How to actually configure the kernel

5. Compiling the kernel

12. Other relevant HOWTOs that might be useful

13. Misc

14. Other Formats of this Document

---

1. Introduction

Should you read this document? Well, see if you've got any of the following symptoms:

- ``Arg! This wizzo−46.5.6 package says it needs kernel release 2.8.193 and I still only have release 1.0.9!''
- There's a device driver in one of the newer kernels that you just gotta have
- You really have no idea at all how to compile a kernel
- ``Is this stuff in the README *really* the whole story?''
- You came, you tried, it didn't work
- You need something to give to people who insist on asking you to install their kernels for them

1.1 Read this first! (I mean it)

Some of the examples in this document assume that you have GNU `tar`, `find`, and `xargs`. These are quite standard; this should not cause problems. It is also assumed that you know your system's filesystem structure; if you don't, it is critical that you keep a written copy of the `mount` command's output during normal system operation (or a listing of `/etc/fstab`, if you can read it). This information is important, and does not change unless you repartition your disk, add a new one, reinstall your system, or something similar.

The latest ``production'' kernel version at the time of this writing was 2.2.9, meaning that the references and examples correspond to that release. Even though I try to make this document as version−independent as possible, the kernel is constantly under development, so if you get a newer release, it will inevitably have some differences. Again, this should not cause major problems, but it may create some confusion.

There are two versions of the linux kernel source, ``production'' and ``development.'' Production releases are the even−minor−numbered releases; 1.2.x was production, 2.0.x is production, as well as 2.2.x. These kernels are considered to be the most stable, bug−free versions available at the time of release. The development kernels (2.1.x, 2.3.x, etc) are meant as testing kernels, for people willing to test out new and possibly very buggy kernels. You have been warned.

1.2 A word on style

`Text that looks like this` is either something that appears on your screen, a filename, or something that can be directly typed in, such as a command, or options to a command (if you're looking at a plain−text file, it doesn't look any different). Commands and other input are frequently quoted (with ` '), which causes the following classic punctuation problem: if such an item appears at the end of a sentence in quotes, people often type a `.' along with the command, because the American quoting style says to put the period inside of the quotation marks. Even though common sense (and unfortunately, this assumes that the one with the ``common sense'' is used to the so−called American style of quotation) should tell one to strip off the punctuation first, many people simply do not remember, so I will place it outside the quotation marks in such cases. In other words, when indicating that you should type ``make config'' I would write `make config', not `make config.'

---

2. Quick Steps − Kernel Compile

This section is written by Al Dev

The latest version of this section is at http://www.aldev.8m.com and click on "Quick Steps to recompile linux kernel". Mirror sites are at − http://aldev.webjump.com, angelfire, geocities, virtualave, bizland, theglobe, spree, infoseek, bcity, 50megs, NBCi, Terrashare, Fortunecity, Freewebsites, Tripod. These sites have **lots of linux goodies** and tips.

A copy of the above web−site is reproduced here −

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. **Note:** Below 'bash#' denotes the bash prompt, you should type the commands that appear after the 'bash#' prompt. Below are commands tested on Redhat Linux, but it should work for other distributions with very minor changes.

1. 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−sources*.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 http://rpmfind.net/linux/RPM/mandrake/7.1/Mandrake/RPMS/bin86−0.4−12mdk.i586.html , http://rpmfind.net/linux/RPM/kondara/jirai/i586/bin86−0.4−8k.i586.html).

2. Start X−windows with 'startx'.

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

   The **"make xconfig"** brings up a user friendly GUI interface! **DO NOT** use 'make config' which is a command−line option (use this only if you **CANNOT** bring up X−window). You load the configuration file from */usr/src/linux/arch/i386/config.in*

3. 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 these man pages

```
bash# man lsmod
bash# man insmod
bash# man rmmod
bash# man depmod
```

4. Save and Exit "make xconfig". And now, do −

```
bash# make dep
bash# make clean
```

5. Read the following file (to gain some knowledge about kernel building...) −

```
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.
```

6. Now, give the make command −

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

7. 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. And then read the manual page on lilo (see also http://www.linuxdoc.org/HOWTO/LILO−crash−rescue−HOWTO.html) −

```
bash# cp /usr/src/linux/arch/i386/boot/bzImage     /boot/bzImage.myker
bash# man lilo
bash# man lilo.conf
And edit /etc/lilo.conf file and put these lines −
        image=/boot/bzImage.myker
        label=myker
        root=/dev/hda1
        read−only
You can check device name for 'root=' with the command −
        bash# df   /boot
```

8. Now give

```
bash# lilo
```

```
bash# lilo −q
```

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

9. 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*
10. Loadable Modules: Check for insmod command which is extensively used for loading the modules.

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

    This step 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)
    ```
11. This step is required **ONLY if you had downloaded a new version** of linux kernel source. Loadable module are located in /lib/modules.

    ```
    bash# cd /usr/src/linux
    bash# make modules
    bash# make install_modules
    ```
12. 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
```

2.1 Troublshoot Common Mistakes

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
```

3. Important questions and their answers

3.1 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.

3.2 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.

3.3 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.

3.4 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.

3.5 What's a loadable module?

These 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.

3.6 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.

3.7 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.

---

4. How to actually configure the kernel

4.1 Getting the source

You can obtain the source via anonymous ftp from `ftp.kernel.org` in `/pub/linux/kernel/vx.y`, where `x.y` is the version (eg 2.2), and as mentioned before, the ones that end with an odd number are development releases and may be unstable. It is typically labelled `linux–x.y.z.tar.gz`, where `x.y.z` is the version number. The sites also typically carry ones with a suffix of `.bz2`, which have been compressed with bzip2 (these files will be smaller and take less time to transfer).

It's best to use `ftp.xx.kernel.org` where `xx` is your country code; examples being `ftp.at.kernel.org` for Austria, and `ftp.us.kernel.org` for the United States.

4.2 Unpacking the source

Log in as or `su` to \``root`', and `cd` to `/usr/src`. If you installed kernel source when you first installed linux (as most do), there will already be a directory called \``linux`' there, which contains the entire old source tree. If you have the disk space and you want to play it safe, preserve that directory. A good idea is to figure out what version your system runs now and rename the directory accordingly. The command \``uname -r`' prints the current kernel version. Therefore, if \``uname -r`' said \``1.0.9`', you would rename (with \``mv`') \``linux`' to \``linux-1.0.9`'. If you feel mildly reckless, just wipe out the entire directory. In any case, make certain there is no \``linux`' directory in `/usr/src` before unpacking the full source code.

Now, in `/usr/src`, unpack the source with \``tar zxpvf linux-x.y.z.tar.gz`' (if you've just got a `.tar` file with no `.gz` at the end, \``tar xpvf linux-x.y.z.tar`' works.). The contents of the source will fly by. When finished, there will be a new \``linux`' directory in `/usr/src`. `cd` to `linux` and look over the `README` file. There will be a section with the label \``INSTALLING the kernel`'. Carry out the instructions when appropriate –– symbolic links that should be in place, removal of stale `.o` files, etc.

If you have a `.bz2` file and the bzip2 program (read about it at `http://www.muraroa.demon.co.uk/`), do this:

```
bz2cat linux-x.y.z.tar.bz2 | tar xvf -
```

4.3 Configuring the kernel

Note: Some of this is reiteration/clarification of a similar section in Linus' `README` file.

The command \``make config`' while in `/usr/src/linux` starts a configure script which asks you many questions. It requires bash, so verify that bash is `/bin/bash`, `/bin/sh`, or `$BASH`.

However, there are some much more pleasant alternatives to \``make config`' and you may very well find them easier and more comfortable to use. \``make menuconfig`' is probably the most widely–used. Whatever you choose, it's best to get familiar with the interface because you may find yourself back at it sooner than you think. For those ``running X,'' you can try \``make xconfig`' if you have Tk installed (\`click–o–rama' – Nat). \``make menuconfig`' is for those who have (n)curses and would prefer a text–based menu. These interfaces have a rather clear advantage: If you goof up and make a wrong choice during configuration, it is simple to go back and fix it.

The configuration options will appear in hierarchies with \``make menuconfig`' and \``make xconfig`'.

You are ready to answer the questions, usually with \``y`' (yes) or \``n`' (no). Device drivers typically have an \``m`' option. This means ``module,'' meaning that the system will compile it, but not directly into the kernel, but as a loadable module. A more comical way to describe it is as ``maybe.'' Some of the more obvious and non–critical options are not described here; see the section ``Other configuration options'' for short descriptions of a few others. With \``make menuconfig`', the space bar toggles the selection.

In 2.0.x and later, there is a \`?' option, which provides a brief description of the configuration parameter. That information is likely to be the most up–to–date. Here are a listing of some of the important features, which hierarchy they are in, and brief description.

Kernel math emulation (Processor type and features)

If you don't have a math coprocessor (you have a bare 386 or 486SX), you must say `y' to this. If you do have a coprocessor and you still say `y', don't worry too much −− the coprocessor is still used and the emulation ignored. For any halfway modern machine, the answer will be no, but don't worry if you say yes accidentally; if not needed, it is not used.

Enhanced (MFM/RLL) disk and IDE disk/cdrom support (Block Devices)

You probably need to support this; it means that the kernel will support standard PC hard disks, which most people have. This driver does not include SCSI drives; they come later in the configuration.

You will then be asked about the ``old disk−only'' and ``new IDE'' drivers. You want to choose one of them; the main difference is that the old driver only supports two disks on a single interface, and the new one supports a secondary interface and IDE/ATAPI cdrom drives. The new driver is 4k larger than the old one and is also supposedly ``improved,'' meaning that aside from containing a different number of bugs, it might improve your disk performance, especially if you have newer (EIDE−type) hardware.

Networking support (General Setup)

In principle, you would only say `y' if your machine is on a network such as the internet, or you want to use SLIP, PPP, term, etc to dial up for internet access. However, as many packages (such as the X window system) require networking support even if your machine does not live on a real network, you should say `y'. Later on, you will be asked if you want to support TCP/IP networking; again, say `y' here if you are not absolutely sure.

System V IPC (General Setup)

One of the best definitions of IPC (Interprocess Communication) is in the Perl book's glossary. Not surprisingly, some Perl programmers employ it to let processes talk to each other, as well as many other packages (DOOM, most notably), so it is not a good idea to say n unless you know exactly what you are doing.

Processor family (Processor type and features)

(in older kernels: Use −m486 flag for 486−specific optimizations)

Traditionally, this compiled in certain optimizations for a particular processor; the kernels ran fine on other chips, but the kernel was perhaps a bit larger. In newer kernels, however, this is no longer true, so you should enter the processor for which you are compiling the kernel. A ``386'' kernel will work on all machines.

SCSI support

If you have SCSI devices, say `y'. You will be prompted for further information, such as support for CD−ROM, disks, and what kind of SCSI adapter you have. See the SCSI−HOWTO for greater detail.

Network device support

If you have a network card, or you would like to use SLIP, PPP, or a parallel port adapter for connecting to the Internet, say `y'. The config script will prompt for which kind of card you have, and which protocol to use.

Filesystems

The configure script then asks if you wish to support the following filesystems:

Standard (minix) − Newer distributions don't create minix filesystems, and many people don't use it, but it may still be a good idea to configure this one. Some ``rescue disk'' programs use it, and still more floppies may have a minix filesystem, since the minix filesystem is less painful to use on a floppy.

Second extended − This is the standard Linux filesystem. You almost definitely have one of these, and need to say `y'.

msdos − If you want to use your MS−DOS hard disk partitions, or mount MS−DOS formatted floppy disks, say `y'.

There are various other foreign operating system filesystem types available.

/proc − (idea from Bell Labs, I guess). One doesn't make a proc filesystem on a disk; this is a filesystem interface to the kernel and processes. Many process listers (such as `ps') use it. Try `cat /proc/meminfo' or `cat /proc/devices' sometime. Some shells (rc, in particular) use /proc/self/fd (known as /dev/fd on other systems) for I/O. You should almost certainly say `y' to this; many important linux tools depend on it.

NFS − If your machine lives on a network and you want to use filesystems which reside on other systems with NFS, say `y'.

ISO9660 − Found on most CD−ROMs. If you have a CD−ROM drive and you wish to use it under Linux, say `y'.

But I don't know which filesystems I need!

Ok, type `mount'. The output will look something like this:

```
blah# mount
/dev/hda1 on / type ext2 (defaults)
/dev/hda3 on /usr type ext2 (defaults)
none on /proc type proc (defaults)
/dev/fd0 on /mnt type msdos (defaults)
```

Look at each line; the word next to `type' is the filesystem type. In this example, my / and /usr filesystems are second extended, I'm using /proc, and there's a floppy disk mounted using the msdos (bleah) filesystem.

You can try `cat /proc/filesystems' if you have /proc currently enabled; it will list your current kernel's filesystems.

The configuration of rarely−used, non−critical filesystems can cause kernel bloat; see the section on modules for a way to avoid this and the ``Pitfalls'' section on why a bloated kernel is undesirable.

Character devices

Here, you enable the drivers for your printer (parallel printer, that is), busmouse, PS/2 mouse (many notebooks use the PS/2 mouse protocol for their built−in trackballs), some tape drives, and other such ``character'' devices. Say `y' when appropriate.

Note: gpm is a program which allows the use of the mouse outside of the X window system for cut and paste between virtual consoles. It's fairly nice if you have a serial mouse, because it coexists well with X, but you need to do special tricks for others.

Sound

If you feel a great desire to hear biff bark, say `y', and you can tell the configuration program all about your sound board. (A note on sound card configuration: when it asks you if you want to install the full version of the driver, you can say `n' and save some kernel memory by picking only the features which you deem necessary.)

If you are serious about sound card support, have a look at both the free drivers at http://www.linux.org.uk/OSS/ and the commercial Open Sound System at http://www.opensound.com/.

Other configuration options

Not all of the configuration options are listed here because they change too often or fairly self−evident (for instance, 3Com 3C509 support to compile the device drive for this particular ethernet card). There exists a fairly comprehensive list of all the options (plus a way to place them into the Configure script) in an effort started and maintained by Axel Boldt (boldt@math.ucsb.edu) and it's the online help. It's also available as one big file at the Documentation/Configure.help in your Linux kernel source tree as of version 2.0.

Kernel hacking

>From Linus' README:

the ``kernel hacking'' configuration details usually result in a bigger or slower kernel (or both), and can even make the kernel less stable by configuring some routines to actively try to break bad code to find kernel problems (kmalloc()). Thus you should probably answer `n' to the questions for a ``production'' kernel.

4.4 Now what? (The Makefile)

After you finish configuration, a message tells you that your kernel has been configured, and to ``check the top−level Makefile for additional configuration,'' etc.

So, look at the Makefile. You probably will not need to change it, but it never hurts to look. You can also change its options with the `rdev' command once the new kernel is in place. If you're feel lost when you look at the file, then don't worry about it.

5. Compiling the kernel

5.1 Cleaning and depending

When the configure script ends, it also tells you to `make dep' and (possibly) `clean'. So, do the `make dep'. This insures that all of the dependencies, such the include files, are in place. It does not take long, unless your computer is fairly slow to begin with. For older versions of the kernel, when finished, you should do a `make clean'. This removes all of the object files and some other things that an old version leaves behind. In any case, *do not* forget this step before attempting to recompile a kernel.

5.2 Compile time

After depending and cleaning, you may now `make bzImage' or `make bzdisk' (this is the part that takes a long time.). `make bzImage' will compile the kernel, and leave a file in arch/i386/boot called `bzImage' (among other things). This is the new compressed kernel. `make bzdisk' does the same thing, but also places the new bzImage on a floppy disk which you hopefully put in drive ``A:''. `bzdisk' is fairly handy for testing new kernels; if it bombs (or just doesn't work right), just remove the floppy and boot with your old kernel. It can also be a handy way to boot if you accidentally remove your kernel (or something equally as dreadful). You can also use it to install new systems when you just dump the contents of one disk onto the other (``all this and more! NOW how much would you pay?'').

All even halfway reasonably recent kernels are compressed, hence the `bz' in front of the names. A compressed kernel automatically decompresses itself when executed.

In older kernels, you don't have the option to build a bzImage; it was simply a zImage. That option is at the moment still available, however, given the code size of newer kernels, it is now more or less mandatory to build a bzImage because the older methods can't handle a kernel that's just too large.

5.3 Other ``make''ables

`make mrproper' will do a more extensive `clean'ing. It is sometimes necessary; you may wish to do it at every patch. `make mrproper' will also delete your configuration file, so you might want to make a backup of it (.config) if you see it as valuable.

`make oldconfig' will attempt to configure the kernel from an old configuration file; it will run through the `make config' process for you. If you haven't ever compiled a kernel before or don't have an old config file, then you probably shouldn't do this, as you will most likely want to change the default configuration.

See the section on modules for a description of `make modules'.

5.4 Installing the kernel

After you have a new kernel that seems to work the way you want it to, it's time to install it. Most people use LILO (Linux Loader) for this. `make bzlilo' will install the kernel, run LILO on it, and get you all ready to boot, BUT ONLY if lilo is configured in the following way on your system: kernel is /vmlinuz, lilo is in

`/sbin`, and your lilo config (`/etc/lilo.conf`) agrees with this.

Otherwise, you need to use LILO directly. It's a fairly easy package to install and work with, but it has a tendency to confuse people with the configuration file. Look at the config file (either `/etc/lilo/config` for older versions or `/etc/lilo.conf` for new versions), and see what the current setup is. The config file looks like this:

```
image = /vmlinuz
    label = Linux
    root = /dev/hda1
    ...
```

The `` `image =' `` is set to the currently installed kernel. Most people use `/vmlinuz`. `` `label' `` is used by lilo to determine which kernel or operating system to boot, and `` `root' `` is the `/` of that particular operating system. Make a backup copy of your old kernel and copy the `bzImage` which you just made into place (you would say `` `cp bzImage /vmlinuz' `` if you use `` `/vmlinuz' ``). Then, rerun lilo −− on newer systems, you can just run `` `lilo' ``, but on older stuff, you might have to do an `/etc/lilo/install` or even an `/etc/lilo/lilo −C /etc/lilo/config`.

If you would like to know more about LILO's configuration, or you don't have LILO, get the newest version from your favorite ftp site and follow the instructions.

To boot one of your old kernels off the hard disk (another way to save yourself in case you screw up the new kernel), copy the lines below (and including) `` `image = xxx' `` in the LILO config file to the bottom of the file, and change the `` `image = xxx' `` to `` `image = yyy' ``, where `` `yyy' `` is the full pathname of the file you saved your backup kernel to. Then, change the `` `label = zzz' `` to `` `label = linux−backup' `` and rerun `lilo`. You may need to put a line in the config file saying `` `delay=x' ``, where x is an amount in tenths of a second, which tells LILO to wait that much time before booting, so that you can interrupt it (with the shift key, for example), and type in the label of the backup boot image (in case unpleasant things happen).

---

6. Patching the kernel

6.1 Applying a patch

Incremental upgrades of the kernel are distributed as patches. For example, if you have version 1.1.45, 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 | patch −p0' `` (or `` `patch −p0 < 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

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 # 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. Additional packages

Your linux kernel has many features which are not explained in the kernel source itself; these features are typically utilized through external packages. Some of the most common are listed here.

7.1 kbd

The linux console probably has more features than it deserves. Among these are the ability to switch fonts, remap your keyboard, switch video modes (in newer kernels), etc. The kbd package has programs which allow the user to do all of this, plus many fonts and keyboard maps for almost any keyboard, and is available from the same sites that carry the kernel source.

7.2 util−linux

Rik Faith (`faith@cs.unc.edu`) put together a large collection of linux utilities which are, by odd coincidence, called util−linux. These are now maintained by Andries Brouwer (`util-linux@math.uio.no`). Available via anonymous ftp from sunsite.unc.edu in `/pub/Linux/system/misc`, it contains programs such as `setterm`, `rdev`, and `ctrlaltdel`, which are relevant to the kernel. As Rik says, *do not install without thinking;* you do not need to install everything in the package, and it could very well cause serious problems if you do.

7.3 hdparm

As with many packages, this was once a kernel patch and support programs. The patches made it into the official kernel, and the programs to optimize and play with your hard disk are distributed separately.

7.4 gpm

gpm stands for general purpose mouse. This program allows you to cut and paste text between virtual consoles and do other things with a large variety of mouse types.

---

8. Some pitfalls

8.1 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.

8.2 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 lots 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`' ``.

8.3 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.

8.4 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/`.

8.5 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.).

8.6 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.

8.7 It says `warning: bdflush not running'

This can be a severe problem. Starting with a kernel release after 1.0 (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.

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

Strangely enough, lots 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 linux in that versions 1.2.x don't support a third IDE interface (there is support in starting somewhere in the 1.3.x series but that's development, remember, and it doesn't auto−probe). 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.

8.9 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.

8.10 Firewalling not working in 1.2.0

Upgrade to at least version 1.2.1.

8.11 ``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.

8.12 Problems with console terminal after upgrade to 1.3.x

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

8.13 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.

8.14 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
```

9. Note for upgrade to version 2.0.x, 2.2.x

Kernel versions 2.0.x and 2.2.x introduced quite a bit of changes for kernel installation. The file `Documentation/Changes` in the 2.0.x source tree contains information that you should know when upgrading to either of these versions. You will most likely need to upgrade several key packages, such as gcc, libc, and SysVInit, and perhaps alter some system files, so expect this. Don't panic, though.

---

10. 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.

10.1 Installing the module utilities

The module utilities are available from wherever you got your kernel source as `modutils-x.y.z.tar.gz`; choose the highest patchlevel `x.y.z` that is equal to or below that of your current kernel. Unpack it with `` `tar zxvf modutils-x.y.z.tar.gz' ``, `cd` to the directory it creates (`modutils-x.y.z`), look over the `README`, and carry out its installation instructions (which is usually something simple, such as `make install`). You should now have the programs `insmod`, `rmmod`, `ksyms`, `lsmod`, `genksyms`, `modprobe`, and `depmod` in `/sbin`. If you wish, test out the utilities with the ``hw'' example driver in `insmod`; look over the `INSTALL` file in that subdirectory for details.

`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.

10.2 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.

11. Tips and tricks

11.1 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)
```

11.2 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.

11.3 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.emlist.com` in `pub/kchanges` or through the URL

```
http://www.crynwr.com/kchanges
```

12. Other relevant HOWTOs that might be useful

- Sound−HOWTO: sound cards and utilities
- SCSI−HOWTO: all about SCSI controllers and devices
- 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..
- 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

13. Misc

13.1 Author

The author and maintainer of the Linux Kernel−HOWTO is Brian Ward (`bri@cs.uchicago.edu`). Please send me any comments, additions, corrections (Corrections are, in particular, the most important to me.).

You can take a look at my `home page' at one of these URLs:

```
http://www.math.psu.edu/bri/
http://blah.math.tu-graz.ac.at/~bri/
```

Even though I try to be attentive as possible with mail, please remember that I get a *lot* of it every day, so it may take a little time to get back to you. Especially when emailing me with a question, please try extra hard to be clear and detailed in your message. If you're writing about non−working hardware (or something like that), I need to know what your hardware configureation is. If you report an error, don't just say ``I tried this but it gave an error;'' I need to know what the error was. I would also like to know what versions of the kernel, gcc, and libc you're using. If you just tell me you're using this−or−that distribution, it won't tell me much at all. I don't care if you ask simple questions; remember, if you don't ask, you may never get an answer! I'd like to thank everyone who has given me feedback.

If your question does not relate to the kernel, or is in some language that I don't understand, I may not answer.

If you mailed me and did not get an answer within a resonable amount of time (three weeks or more), then chances are that I accidentally deleted your message or something (sorry). Please try again.

I get a lot of mail about thing which are actually hardware problems or issues. That's OK, but please try to keep in mind that I'm not familiar with all of the hardware in the world. I use AMD processors, Adaptec and Sybios SCSI controllers, and IBM SCSI disks.

Version −0.1 was written on October 3, 1994. This document is available in SGML, PostScript, TeX, roff, and plain−text formats.

13.2 To do

The ``Tips and tricks'' section is a little small. I hope to expand on it with suggestions from others.

So is ``Additional packages.''

More debugging/crash recovery info needed.

13.3 Contributions

A small part of Linus' README (kernel hacking options) is inclusive. (Thanks, Linus!)

`uc@brian.lunetix.de` (Ulrich Callmeier): patch −s and xargs.

`quinlan@yggdrasil.com` (Daniel Quinlan): corrections and additions in many sections.

`nat@nat@nataa.fr.eu.org` (Nat Makarevitch): mrproper, tar −p, many other things

`boldt@math.ucsb.edu` (Axel Boldt): collected descriptions of kernel configuration options on the net; then provided me with the list

`lembark@wrkhors.psyber.com` (Steve Lembark): multiple boot suggestion

`kbriggs@earwax.pd.uwa.edu.au` (Keith Briggs): some corrections and suggestions

`rmcguire@freenet.columbus.oh.us` (Ryan McGuire): makeables additions

`dumas@excalibur.ibp.fr` (Eric Dumas): French translation

`simazaki@ab11.yamanashi.ac.jp` (Yasutada Shimazaki): Japanese translation

`jjamor@lml.ls.fi.upm.es` (Juan Jose Amor Iglesias): Spanish translation

`mva@sbbs.se` (Martin Wahlen): Swedish translation

`jzp1218@stud.u-szeged.hu` (Zoltan Vamosi): Hungarian translation

`bart@mat.uni.torun.pl` (Bartosz Maruszewski): Polish translation

`donahue@tiber.nist.gov` (Michael J Donahue): typos, winner of the ``sliced bread competition'' `rms@gnu.ai.mit.edu` (Richard Stallman): ``free'' documentation concept/distribution notice

`dak@Pool.Informatik.RWTH−Aachen.DE` (David Kastrup): NFS thing

`esr@snark.thyrsus.com` (Eric Raymond): various tidbits

The people who have sent me mail with questions and problems have also been quite helpful.

13.4 Copyright notice, License, and all that stuff

Recommendations: Commercial redistribution is allowed and encouraged; however, it is strongly recommended that the redistributor contact the author before the redistribution, in the interest of keeping things up−to−date (you could send me a copy of the thing you're making while you're at it). Translators are also advised to contact the author before translating. The printed version looks nicer. Recycle.

---

14. Other Formats of this Document

This section is written by Al Dev (at site http://www.aldev.8m.com mirrors at http://aldev.webjump.com, angelfire, geocities, virtualave, bizland, theglobe, spree, infoseek, bcity, 50megs)

This document is published in 12 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 and SGML.

- You can get this HOWTO document as a single file tar ball in HTML, DVI, Postscript or SGML formats from − ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/other−formats/ and http://www.linuxdoc.org/docs.html#howto
- Plain text format is in: ftp://sunsite.unc.edu/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
- Translations to other languages like French, German, Spanish, Chinese, Japanese are in ftp://sunsite.unc.edu/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 Kernel−HOWTO.sgml (to generate html file)
- sgml2rtf Kernel−HOWTO.sgml (to generate RTF file)
- sgml2latex Kernel−HOWTO.sgml (to generate latex file)

LaTeX documents may be converted into PDF files simply by producing a Postscript output using **sgml2latex** (and dvips) and running the output through the Acrobat **distill** (http://www.adobe.com) command as follows:

```
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.

This howto document is located at −

- http://sunsite.unc.edu/LDP/HOWTO/Kernel−HOWTO.html

Also you can find this document at the following mirrors sites −

- http://www.caldera.com/LDP/HOWTO/Kernel−HOWTO.html
- http://www.WGS.com/LDP/HOWTO/Kernel−HOWTO.html
- http://www.cc.gatech.edu/linux/LDP/HOWTO/Kernel−HOWTO.html
- http://www.redhat.com/linux−info/ldp/HOWTO/Kernel−HOWTO.html
- Other mirror sites near you (network−address−wise) can be found at http://sunsite.unc.edu/LDP/hmirrors.html select a site and go to directory /LDP/HOWTO/Kernel−HOWTO.html

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 | Applications | Publishing | 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 | Applications | 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.
