The Linux Kernel HOWTO

by Brian Ward, ward@blah.tu-graz.ac.at version 0.2, 3 Jan 1995

This is a detailed guide to kernel configuration, compilation, and upgrades.

Contents

1 Introduction

This is release 0.2 of the Kernel-HOWTO. Is this document for you? Well, if you've got any of the following symptoms:

- "Arg! This wizzo-46.5.6 package says it needs kernel 1.1.193 and I've still only got release 1.0!"
- 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 know how to compile and install a kernel, and people seem to know this. Therefore, they keep whining to you to help them install their kernels.

1.1 Assumptions

Some of the examples in this document assume that you have GNU tar, find, and xargs. These are standard with distributions. 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 doesn't change unless you repartition your disk, add a new one, reinstall your system, or something similar.

2 Some questions that usually come up before anything else

2.1 What does the kernel do, anyway?

In Unix, the kernel does a lot of things. First, it does all of that memory management for all of the running programs (processes), and makes sure that they all get a fair (or unfair, as the case may be) share of the processor's cycles. In addition, it provides a nice interface for the aforementioned programs to talk to your hardware.

Is that it? Well, it is a little more complicated, but it won't be discussed in detail here.

2.2 Why would I want to upgrade my kernel?

Newer kernels generally offer more the ability to talk to more kinds of weird hardware (ie, 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.

2.3 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`." "`make config`" will allow you to see all hardware supported by the standard kernel source code, but not everything that linux supports; many common device drivers (like PCMCIA drivers and some tape drivers) are loadable modules maintained and distributed seperately.

2.4 What version of gcc or libc do I need?

Linus recommends this in the README in the linux source. If you don't have at least that version, the new distribution of gcc should tell you if you need to upgrade your libc. Both of these aren't really scary operations, but do follow the instructions.

2.5 What's a loadable module?

These are parts of the kernel (device drivers, usually) which aren't compiled into the kernel. One compiles them separately, and can insert and remove them into the running kernel just about any time. This has become a preferred way of adding things onto the kernel, since it's obviously more modular. Many popular device drivers, such as the PCMCIA drivers and the QIC-80/40 tape driver, are loadable modules.

2.6 How much disk space do I need?

It depends on a few things. First off, the compressed linux source was 1.66 megs at version 1.1.45. You'll usually want to keep this around. Uncompressed, that was about 8 megs. But that's not the end – you'll need more to actually compile the thing. This depends on how much junk you've got configured into your kernel. For example, on my 386, I've networking, the 3Com 3C503 driver, and five filesystems configured, and it takes 16.2 megs. Adding the compressed linux source, you have about 18 megs for this particular configuration. On another system, without network device support (but still with networking support), and sound card support, it takes 12 megs. So, in general, if you've got a lot of hardware, make sure that you have a big enough hard disk in that mess..

2.7 How long does it take?

For most people, the answer is, "kind of long." The speed of your processor is and the amount of memory you have ultimately determines the time, but there's a small bit to do with the amount of stuff you've got configured into the kernel. On a 486DX4/100 notebook with 16 megs of RAM, on a kernel with five filesystems, networking support, and sound card drivers, it takes under 20 minutes. On a 386DX/40 (8MB RAM) with a similar configuration, it's about 1.5 hours. (meaningless aside: I know someone with a 386SX/16 w/4MB RAM who compiles his kernels overnight (it usually takes longer (yes, I have told him to get a real computer)))

3 How to actually configure the kernel

3.1 Getting the source

You can attain the source via ftp from ftp.funet.fi or mirror in `/pub/OS/Linux/PEOPLE/Linus`, a mirror of this site, or other sites. It is typically labelled `linux-x.x.x.tar.gz`, where `x.x.x` is the version number. Newer (better?) versions and the patches are typically in subdirectories like "`v1.1`" and "`v1.2`."

It is *strongly* suggested that you use a mirror ftp site instead of ftp.funet.fi! A short listing of mirrors and other sites follows.

```
(mirrors of ftp.funet.fi:)
Germany:        ftp.Germany.EU.net
UK:             doc.ic.ac.uk
Australia:      kirk.bu.oz.au
(sites that also carry the kernel files:)
USA:            tsx-11.mit.edu
USA:            sunsite.unc.edu
Germany:        ftp.dfv.rwth-aachen.de
```

If you don't have ftp access, there is a list of BBS systems carrying linux which is posted periodically to comp.os.linux.announce; try to get your hands on this.

3.2 Unpacking the source

Make sure you're "`root`," and cd to `/usr/src`. If you got your linux from anywhere reasonable, there will already be a directory there called "`linux`." That's the old linux source code. If you have the disk space and you want to play it safe, you'll want to preserve that directory. A good idea is to figure out what version you're running and rename the directory accordingly. The command "`uname -r`" will tell you the new version. Therefore, if "`uname -r`" said "`1.1.47`," you would rename "`linux`" to "`linux-1.1.47`." If you're the reckless kind of person, just wipe out the directory. In any case, just make sure there is no "`linux`" directory in `/usr/src` before unpacking the full source code.

In `/usr/src`, unpack the source with "`zcat linux.x.x.x.tar.gz | tar xvf -`" (You could also use "`tar zxvf linux.x.x.x.tar.gz`" for simplicity; if you've just got a `.tar` file (no `.gz` at the end), "`tar xvf linux.x.x.x.tar`" will work fine.). You'll see the contents of the source fly by. When finished, there will be a new "`linux`" directory. "`cd`" to linux and look over the `README` file. There will be a section labelled

"INSTALLING the kernel" or something like that. Carry out the instructions when appropriate – symlinks that should be in place, removal of stale .o files, etc.

3.3 Configuring the kernel

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

The command "`make config`" while in `/usr/src/linux` will start up a script that asks you all sorts of questions. It needs bash, so make sure bash is `/bin/bash`, `/bin/sh`, or `$BASH`.

You're ready to answer the questions, usually with "`y`" or "`n`." Some of the more obvious and non-critical options are not described.

3.3.1 Kernel math emulation

If you don't have a math coprocessor (ie, you've got a bare 386 or 486SX), you need to say "`y`" to this. If you do have a coprocessor and you still say "`y`," don't worry too much – the coprocessor will be used and the emulation ignored. The only consequence is that the kernel will be larger.

3.3.2 Normal harddisk support

You'll almost always need to say yes to this. This means that the kernel will support standard PC disks like IDE stuff, which most people have. This does not include SCSI drives.

3.3.3 Networking support

If you're on a network, say, if you are on the internet, or you're going to use SLIP, PPP, term, or something to dial up to places for internet access, say "`y`."

3.3.4 Limit memory to low 16MB

If you've got less than 16MB of RAM, you almost always want to say "`y`" to this; you save a little bit of RAM when the new kernel is in use. But, of course, if you've got more than 16, if you don't say "`n`," the new kernel will only use 16 of your 34 billion megs of RAM. There are supposedly bad 386 DMA controllers (or something hardware-related) out there which don't address anything above 16 megs correctly, either; you would want to say "`y`" in the (rare) case that you have one.

3.3.5 Use -m486 flag for 486-specific optimizations

This optimizes the kernel for the 486. The new kernel will be slightly bigger, but it will still work fine on a 386, if you're worried about that. It might be a "little" slower on the 386, but you won't notice it.

3.3.6 SCSI support

If you've got a SCSI device, say "`y`." You'll be prompted for further information, like if you want to support CD-ROM, disks, and what kind of SCSI adapter you've got. See the SCSI-HOWTO for more detail.

3.3.7 Network device support

If you've got an ethernet card, or you want to use SLIP, PPP, or a parallel port adapter, say "`y`." You'll be prompted for which kind of card you have, or which serial protocol to use.

3.3.8 Filesystems

You'll be prompted for support of a number of filesystems. They are:

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 optimal for floppy disks.

Extended fs - This was the first version of the extended filesystem, which isn't used much anymore. Chances are, you'll know it if you need it.

Second extended - This is widely used in new distributions. You probably have one of these.

xiafs filesystem - At one time, this was pretty common, but at the time of this writing, I didn't know anyone running it.

msdos - Well, you guessed it; if you want to use your MS-DOS hard disk partitions, or mount MS-DOS formatted floppy disks, say "`y`."

umsdos - This is a fairly slick filesystem which can make an MS-DOS filesystem have more features, like long filenames, etc. It's not really useful for people (like me) who don't run MS-DOS.

/proc - One of the most slick filesystems (idea shamelessly stolen from Bell Labs, I guess). It's not anything you partition disks with; but a filesystem interface to the kernel and processes. Many process-listers (like "`ps`") use it. If you've got it installed, try "`cat /proc/meminfo`" or "`cat /proc/devices`" sometime. Some shells, like rc, use `/proc/self/fd` (known as `/dev/fd` on other systems) for i/o. You should almost certainly say "`y`" to this; many important standard Linux tools depend on it!

NFS - If you're on a network and you want to share files, say "`y`."

ISO9660 - Found on most CD-ROMs.

OS/2 HPFS - At the time of this writing, a read-only fs for OS/2 HPFS.

System V and Coherent - for partitions of System V and Coherent systems.

But I don't know which filesystems I need! Ok, type "`mount`." It 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. On this example, my `/` and `/usr` partitions are the second extended type, I'm using `/proc`, and there's a floppy disk mounted using the msdos (bleah) filesystem.

You can try "`cat /proc/filesystems`" if you've got /proc. It will give you a list of what's in your current kernel.

3.3.9 Character devices

Here, you'll find the devices for your printer, busmouse, PS/2 mouse (most notebooks use builtin PS/2 mice), some tape drives, selection. Say "`y`" when appropriate.

Note: "Selection" allows one to run a program that lets you use the mouse outside of X-Windows to cut and paste between virtual consoles. It's fairly nice if you've got a serial mouse, because it interacts well with X Windows, but at the time of this writing, it didn't play nice with PS/2 type mice (it locks up the mouse port, making it impossible to run X Windows until you kill the selection process.).

3.3.10 Sound card

If you're interested in keeping the people next door awake, say "`y`," and later on, a config program will compile and ask you all about your sound board.

3.3.11 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.

3.4 Now what? (The Makefile)

You'll get a message telling 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 won't have to change it, but it doesn't hurt to look. You may also change the options in there with the "`rdev`" command once the new kernel is in place, too.

4 Compiling the kernel

4.1 Cleaning and depending

When the configure script ends, it will tell you to "`dep`" and "`clean`." So, do the "`make dep`." It doesn't take that long unless you've got a really slow computer. Making depend makes sure you've got all of the dependencies in place, like the include files and that kind of thing. When finished (you can also do it before the "`make dep`"), do a "`make clean`." This wipes out all of the object files and some other things that the kernel compiles. Don't forget this step.

4.2 Compile time

After depending and cleaning, you can now do "`make`" or "`make zdisk`." "`make`" will compile the kernel, and leave you with a file called "`zImage`" (among other things). That's the new kernel. "`make zdisk`" does the same thing, but sticks it on a floppy disk that you have hopefully put in drive A:. "`zdisk`" is fairly handy for testing new kernels; if it totally bombs out, just remove the floppy and boot with your old kernel. It's also a handy way to boot if you accidentally remove or kernel or something. You can also use it in installing new systems when you just dump the contents of one disk onto the other (all of this and more! NOW how much would you pay?).

All reasonably recent kernels are compressed, hence the "`z`" in front of the names. The kernel is compressed, and when run, automatically decompresses itself (a nice way to save disk space).

4.3 Other "make"ables

"`make mrproper`" will do a more extensive "`clean`"ing. It's sometimes necessary, so you may wish to do it at every patch. "`make zlilo`" 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 agrees.

4.4 Installing the kernel

After you've got 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. It's a fairly easy package to install which, however, can confuse people with the config file. If you don't have the latest version, look at the config file (either `/etc/lilo/config` for older versions or `/etc/lilo.conf` for new versions), and see what you've got. It will say something like this:

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

First off, that "`image = `" is set to what the installed kernel is. Most people seem to use `/vmlinuz`, but I use `/vmlinux`. "`label`" is used by lilo to tell which kernel or operating system you're booting, and "`root`" is the / of that particular OS. Make a backup copy of your old kernel (whatever it's called), and copy the zImage you made into place (like, you would say "`cp zImage /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 want to know more about LILO's configuration, or you don't have LILO but you want to, get the newest version from your favorite ftp site and follow the instructions.

To be able to boot one of your old kernels off the hard disk (another way to save yourself in case you floozie up the new kernel), copy the lines (including the line) "`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 name of the thing you saved your backup kernel to. Then, change the "`label = xXx`" to something like "`label = linux-backup`." Then, rerun lilo. You may have to put a line in the config file saying "`delay=x`," where x is an amount in tenths of a second. That's to make lilo wait before booting, so 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).

5 Patching the kernel

5.1 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 by applying the patch. You might want to make a backup of the source tree first ("`make clean`" and then "`cd /usr/src; tar cvf - linux | gzip -c > old-tree.tar.gz`" will make a compressed tar archive for you.).

So, continuing with the example in the above, let's suppose that you've got "`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 computer is kind of slow) telling you that it's trying to apply hunks, and if it succeeds or not. Usually, it goes by too fast for you to read, so you're not too sure whether it worked or not. So, to look for things that 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`" will do the trick.

If everything has gone right, do a "`make clean`," "`config`," and "`dep`" as described in sections 2 and 3.

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

5.2 If something goes wrong

The most frequent problem that arises used to be 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's left. 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 says it succeeded (no fails), then you're okay. The `config.in.rej` file will remain, but you can get rid of it.

If you've got further problems, you might have installed a patch out of order. If patch says "`previously applied patch detected: Assume -R?`," you are probably applying some patch that is below your current version number (This is not one of those recommended things to do.).

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

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

5.3 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 last cleaned out at 1.1.48 (I think). Removing the .orig files saved over a half a meg. "`find . -name '*.orig' -exec rm -f {} ';'` " will take care of it for you. Some versions of patch use a tilde instead of `.orig`.

There are other (better?) ways to get rid of the `.orig` files. Most use "`xargs`" instead of "`-exec`:" "`find . -name '*.orig' | xargs rm`" or "`find . -name '*.orig' -print0 | xargs --null rm --`" (the latter is more secure.).

5.4 Other patches

There's always 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, or fix the source or the patch accordingly. That's generally an ugly thing to do for novices, so if you're not into screwing around with the source, back out the nonstandard patches before applying Linus'. Then, you can see if the nonstandard patches still work. If they don't, you're either stuck with running at the old patchlevel, playing with the patch to get it to work, or waiting for someone to come out with a version of the nonstandard patch for your new patchlevel.

How common are the ones not in the standard distribution? You will probably hear of them. I use Bill Paul's noblink patch to make the cursor on my virtual consoles because I hate blinking cursors. As far as I know, this particular patch was last updated for kernel version 1.0, but I still used it at 1.1.51 – I have modified it quite a few times, because it often messes up Linus' patches to `drivers/char/console.c`.

6 Additional packages

Your Linux kernel has a lot of 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.

6.1 kbd

The Linux console has more features than you can shake a stick at. This includes the ability to switch fonts, remap your keyboard, switch videomodes (in newer kernels), etc. The kbd package has programs which allow the user to do all of this, plus a ton of fonts and keyboard maps for almost any keyboard.

6.2 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 seperately, as usual.

7 Some pitfalls

7.1 make clean

If your new kernel does really weird things (it's happened to me), chances are you forgot to make clean. Symptoms can be anything from your kernel outright crashing, to strange I/O problems, to crummy performance. Make sure you do a make dep, too.

7.2 Huge or slow kernels

If your kernel is sucking up a lot of memory, or is getting really big, or really takes forever to compile even when you've got your new 486DX6/440 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. If you've got less than 16 megs, make sure that you say yes to "`limit memory to low 16MB`;" it makes a lot of difference (especially on a 4MB system). The most obvious symptom of bloat is extreme swapping in and out of memory to disk. If your disk is making a lot of noise, 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 it from the amount of "total mem" in `/proc/meminfo` or the output of the command "`free`." You can also find out by doing a "`dmesg`" (or by looking at the kernel log file, wherever it is on your system). There will be a line that looks like this:

```
Memory:  15124k/16384k available (552k kernel code, 384k reserved, 324k data)
```

My 386 (which has slightly less junk configured) says this:

```
Memory:  7000k/8192k available (496k kernel code, 384k reserved, 312k data)
```

7.3 Kernel doesn't compile

If it doesn't compile, then a patch has probably failed, or you got corrupted sources from somewhere. Your version of gcc also might not be correct, or could be messed up. Make sure that the symlinks which Linus describes in the README are right. In general, if a kernel doesn't compile, something is seriously wrong.

7.4 New version of the kernel doesn't seem to boot

LILO either wasn't run, or isn't configured correctly. One thing that "got" me once was a problem in the config file. I had "`boot = /dev/hda1`" instead of "`boot = /dev/hda`" (This can be really annoying at first, but once you've got a working config file, you shouldn't need to change it.).

7.5 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 and prepare another bootable floppy (like "`make zdisk`" would do). You need to know where your root (`/`) partition is and what type it is (second extended, minix, etc). In the example below, you also need to know what partition your `/usr/src/linux` source tree is on, its type, and where it is normally mounted.

In the example, the `/` is `/dev/hda1`, and the partition which holds `/usr/src/linux` is `/dev/hda3`, normally mounted at `/usr`. They are both second extended filesystems. The working kernel image in `/usr/src/linux` is called `zImage`.

The idea is that if there was a working kernel image in `/usr/src/linux` called `zImage`, 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 or rescue disk, and then mount the partition that there was a working kernel image on:

```
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: `/mnt + /usr/src/linux - /usr = /mnt/src/linux`). Put a formatted disk in drive a: (make sure it's not your boot or root disk!), dump the image to the disk, and configure for your root partition:

```
cd /mnt/src/linux
dd if=zImage of=/dev/fd0
rdev /dev/fd0 /dev/hda1
```

`cd` to `/` and unmount the normal `/usr` partition:

```
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 very common alternative. If you happened to have a working kernel image in / (`/vmlinuz` for example), you can use that as well. 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 / partition), `/mnt/src/linux` to `/mnt`, and `if=zImage` to `if=vmlinuz`. The little note explaining how to derive `/mnt/src/linux` may be ignored.

7.6 It says "warning: bdflush not running"

This can be a pretty severe problem. Starting with a kernel release after 1.0 (I can't remember which, exactly, but it was 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), and compile it (You will probably want to run under the old kernel while compiling and installing). It will install itself as "`update`" and you should be fine with the new kernel after that.

7.7 It says weird things about obsolete routing requests

Get new versions of the route program and/or recompile the old stuff. `/usr/include/linux/route.h` (which is actually a file in `/usr/src/linux`) has changed.

8 Tips and tricks

8.1 Redirecting output of the make or patch commands

If you'd 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're running sh or bash, "`(command) 2>&1 | tee (output file)`" will place a copy of the make in the file "`(output file)`."

For csh or tcsh, the syntax "`(command) |& tee (output file)`" is for you.

For rc (Note: you probably aren't running rc) it's "`(command) >[2=1] | tee (output file)`."

9 Misc

9.1 Author

The original revision was -0.1 on 3 October 1994 by Brian Ward (ward@blah.tu-graz.ac.at). Please send me any comments, additions, corrections, or computers. Corrections are, in particular, the most important to me!

Even though I try to be attentive as possible with mail, please remember that I get a *lot* of mail per 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. I'd like to thank everyone who's given me feedback.

9.2 History and other forms of this document

I originally wrote this in ASCII, then I put some TeX around it, and then horrified the TeX so that I could convert the TeX into SGML with a perl script. Because of the original in TeX, the first version of this document that appeared on sunsite had two section numbers with each section (It wouldn't have looked that bad if they weren't different.). I also realize that it's kind of nonstandard to make revision numbers negative, but oh well. And if any of the above sounds kind of stupid, I don't blame you.

9.3 To do

The "Tricks and tips" 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.

9.4 Contributions

A small part of Linus' README (Re: 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@nataa.frmug.fr.net (Nat MAKAREVITCH)`: mrproper

The people who have sent me mail with questions and problems have been helpful, too!

9.5 Copyright notice and copying