Lilo mini−HOWTO

Table of Contents

Lilo mini−HOWTO

Miroslav "Misko" Skoric, `m.skoric@eunet.yu`

v3.4, 15 November 2000

*LILO is the most used **Li**nux **Lo**ader for the x86 flavour of Linux; I'll call it Lilo rather than LILO here because I don't appreciate uppercase. This file describes some typical Lilo installations. It's intended as a supplement to the Lilo User's Guide. I think examples are informative even if your setup isn't much like mine. I hope this saves you trouble. Since Lilo's own documentation is very good, who's interested in the details is referred to /usr/doc/lilo* (once upon a time said gentlemen like Cameron Spitzer and Alessandro Rubini who have made early versions of this document)*

This version of Lilo mini−HOWTO is based on work of Cameron Spitzer (`cls@truffula.sj.ca.us`) and Alessandro Rubini (`rubini@linux.it`). There is also a contribution from Tony Harris (`tony@nmr.mgh.harvard.edu`).

---

1. Introduction

Although the documentation found in Lilo's sources (the one installed in `/usr/doc/lilo`−version) is very comprehensive, most Linux users experience some trouble in building their own `/etc/lilo.conf` file. This document is meant to support them by giving the minimal information and by showing five sample installations:

- The first example is the classical ``Linux and other'' installation.
- The next one shows how to install Lilo on a hard drive connected as `/dev/hdc` that will boot as `/dev/hda`. This is usually needed when you install a new Linux drive from your own running system. This also tells how to boot from SCSI disks when your BIOS is modern enough.
- The third example shows how to boot a Linux system whose root partition can't be accessed by the BIOS.
- The next sample file is used to access huge disks, that neither the BIOS nor DOS can access easily (this one is somehow outdated).
- The last example shows how to restore a damaged disk, if the damage resulted from installing another operating system).

The last three examples are by Cameron, `cls@truffula.sj.ca.us`, who wrote the original document. Alessandro `rubini@linux.it` doesn't run anything but Linux, so he can't check nor update them by himself. Needless to say, any feedback is welcome.

---

2. Background Information and Standard Installation

When Lilo boots the system, it uses BIOS calls to load the Linux kernel off the disk (IDE drive, floppy or whatever). Therefore, the kernel must live in some place that can be accessed by the bios.

At boot time, Lilo is not able to read filesystem data, and any pathname you put in `/etc/lilo.conf` is resolved at installation time (when you invoke **/sbin/lilo**). Installation time is when the program builds the tables that list which sectors are used by the files used to load the operating system. As a consequence, all of these files must live in a partition that can be accessed by the BIOS (the files are usually located in the `/boot` directory, this means that only the root partition of your Linux system needs to be accessed via the BIOS).

Another consequence of being BIOS−based is that you must reinstall the loader (i.e., you must reinvoke **/sbin/lilo**) any time you modify the Lilo setup. Whenever you recompile your kernel and overwrite your old image you must reinstall Lilo.

2.1 Where Should I Install Lilo?

The `boot=` directive in `/etc/lilo.conf` tells Lilo where it should place its primary boot loader. In general, you can either specify the master boot record (`/dev/hda`) or the root partition of your Linux installation (is usually is `/dev/hda1` or `/dev/hda2`).

If you have another operating system installed in your hard drive, you'd better install Lilo to the root partition instead of the MBR. In this case, you must mark the partition as ``bootable'' using the ``a'' command of **fdisk** or the ``b'' command of **cfdisk**. If you don't overwrite the master boot sector you'll find it easier to uninstall Linux and Lilo if needed.

Of course, you always have a way to avoid some "rules" like above. Well, you may install Lilo to the MBR even if you already have another operating system installed there. For example, if you installed Windows NT 4.0 as the first operating system on your machine, then NT's boot loader was placed into the MBR so you were able to boot NT without problems. After you installed Linux and chose to install Lilo to the MBR, Lilo rewrote NT's boot loader. Next time you boot your machine, you won't be able to boot NT. But, there is no problem. You should edit your `/etc/lilo.conf` and add a new entry for NT. Next time you re−boot your system, there will be the new added NT entry under Lilo menu.

2.2 How Should I Configure my IDE Hard Drives?

I personally don't use LBA or LARGE settings in the BIOS (but I only run Linux); they are horrible kludges forced on by design deficiencies in the PC world. This requires that the kernel lives in the first 1024 cylinders, but this is not a problem as long as you partition your hard drives and keep root small (as you should do anyways).

If your hard disk already carries another operating system, you won't be able to modify the BIOS settings, or the old system won't work any more. All recent Lilo distribution are able to deal with LBA and LARGE disk settings.

Note that the `"linear"` keyword in `/etc/lilo.conf` can help in dealing with geometry problems. The keyword instructs Lilo to use linear sector addresses instead of sector/head/cylinder tuples. Conversion to 3D addresses is delayed to run−time, therefore making the setup more immune to geometry problems.

If you have more than one hard disk and some of them are only used by Linux and are not involved in the boot process, you can tell your BIOS that they are not installed. Your system will boot more quickly and Linux will autodetect all the disks in no time. I often switch disks in my computers, but I never touch the BIOS configuration.

2.3 How Can I Interact at Boot Time?

When you see the Lilo prompt, you can hit the <Tab> key to show the list of possible choices. If Lilo is not configured to be interactive, press and hold the <Alt> or <Shift> key before the ``LILO'' message appears.

If you choose to boot a Linux kernel, you can add command−line arguments after the name of the system you choose. The kernel accepts many command−line arguments. All the arguments are listed in the ``BootPrompt−HOWTO'' by Paul Gortmaker, and I won't replicate it here. A few command line arguments, however, are particularly important and worth describing here:

- `root=`: you can tell the Linux kernel to mount as root a different partition than the one appearing in `/lilo.conf`. For example, my system has a tiny partition hosting a minimal Linux installation, and I've been able to boot the system after destroying my root partition by mistake.
- `init=`: version 1.3.43 and newer of the Linux kernel can execute another command instead of **/sbin/init**, as specified on the command line. If you experience bad problems during the boot process, you can access the bare system by specifying `init=/bin/sh` (when you are at the shell prompt you most likely will need to mount your disks: try ``**mount −w −n −o remount /; mount −a**'', and remember to ``**umount −a**'' before turning off the computer).
- A number: by specifying a number on the kernel command line, you instruct *init* to enter a specific run−level (the default is usually 3 or 2, according to the distribution you chose). Refer to the *init* documentation, to `/etc/inittab` and to `/etc/rc*.d` to probe further.

2.4 How Can I Uninstall Lilo?

When Lilo overwrites a boot sector, it saves a backup copy in `/boot/boot.`*xxyy*, where *xxyy* are the major and minor numbers of the device, in hex. You can see the major and minor numbers of your disk or partition by running ``**ls −l /dev/*/device***''. For example, the first sector of `/dev/hda` (major 3, minor 0) will be saved in `/boot/boot.0300`, installing Lilo on `/dev/fd0` creates `/boot/boot.0200` and installing on `/dev/sdb3` (major 8, minor 19) creates `/boot/boot.0813`. Note that Lilo won't create the file if there is already one so you don't need to care about the backup copy whenever you reinstall Lilo (for example, after recompiling your kernel). The backup copies found in `/boot/` are always the snapshot of the situation before installing any Lilo.

If you ever need to uninstall Lilo (for example, in the unfortunate case you need to uninstall Linux), you just need to restore the original boot sector. If Lilo is installed in `/dev/hda`, just do ``**dd if=/boot/boot.0300 of=/dev/hda bs=446 count=1**'' (I personally just do ``**cat /boot/boot.0300 > /dev/hda**'', but this is not safe, as this will restore the original partition table as well, which you might have modified in the meanwhile). This command is much easier to run than trying ``**fdisk /mbr**'' from a DOS shell: it allows you to cleanly remove Linux from a disk without ever booting anything but Linux. After removing Lilo remember to run Linux' **fdisk** to destroy any Linux partition (DOS' **fdisk** is unable to remove non−dos partitions).

If you installed Lilo on your root partition (e.g., `/dev/hda2`), nothing special needs to be done to uninstall Lilo. Just run Linux' **fdisk** to remove Linux partitions from the partition table. You must also mark the DOS partition as bootable.

2.5 How to make a ram disk?

Notice: If you find the next section hard to read, you may also look for the web page: `http://surfer.nmr.mgh.harvard.edu/partition/ramdisk.html` *where you would find the original of the contributed section.*

by Tony Harris

16 Oct 2000

ram disk eenie−weenie HOWTO

If your root file system is on a device for which your kernel has no compiled−in driver, you will need to use `lilo` to load that driver as a module very early in the boot cycle. There are only two easy steps:

- make a ram disk image with **/mkinitrd**
- modify `lilo.conf` to point to the image

First, I **cd** over to `/boot`:

```
System.map                  chain.b                 module−info−2.2.16−3ext3
System.map−2.2.16−3         initrd−2.2.16−3.img     vmlinux−2.2.16−3
System.map−2.2.16−3ext3     vmlinux−2.2.16−3ext3
vmlinuz                     kernel.h
boot.b                      map                     vmlinuz−2.2.16−3
bz.2.2.15.juke.Image        module−info             vmlinuz−2.2.16−3ext3
bzImage−2.2.14              module−info−2.2.16−3
```

Here you can see that I have a 2.2.16−3 kernel and I have added a second kernel with ext3 support (`vmlinuz−2.2.16−3ext3`). There is already a ram disk image for my first kernel (`initrd−2.2.16−3.img`)

To make a new image for the second kernel, I type the following (stuff I type is in bold):

boot# **mkinitrd initrd−2.2−16−3ext3.img 2.2.16−3ext3**

`mkinitrd` is a shellscript that looks at the modules needed by my kernel, then makes an ext2 filesystem containing them. If we look inside the image we see this is the case:

boot# **cat initrd−2.2.16−3ext3.img | gunzip** > **/tmp/myimage**

boot# **file /tmp/myimage**

/tmp/myimage: Linux/i386 ext2 filesystem/

You do not have to look inside your image. Only making the image and modifying lilo.conf are necessary steps. However, discussion of the ramdisk image is provided for pedagogic purposes.

In order to look inside, I need to mount the image as though it were a filesystem:

boot# **mount /tmp/myimage /mnt/tmp −t ext2 −o loop=/dev/loop3**

boot# **ls /mnt/tmp**

```
bin dev etc lib linuxrc
```

boot# **find /mnt/tmp**

mnt/tmp/

mnt/tmp/lib/

mnt/tmp/lib/aic7xxx.o/

mnt/tmp/bin/

mnt/tmp/bin/sh/

mnt/tmp/bin/insmod/

mnt/tmp/etc/

mnt/tmp/dev/

mnt/tmp/dev/console/

mnt/tmp/dev/null/

mnt/tmp/dev/ram/

mnt/tmp/dev/systty/

mnt/tmp/dev/tty1/

mnt/tmp/dev/tty2/

mnt/tmp/dev/tty3/

mnt/tmp/dev/tty4/

mnt/tmp/linuxrc/

The most important part of this ram disk image is `aic7xxx.o`, which is my scsi module.

Finally, I move on to the last step, modifying etc/lilo.conf/:

Here is my entry in `lilo.conf` that corresponds to the kernel and image I just created:

`image=`boot/vmlinuz−2.2.16−3ext3/

`label=linux.ext3`

`initrd=`boot/initrd−2.2.16−3ext3.img/

`read−only`

`root=`dev/hdb3/

That's it. Run **/lilo** as root and reboot.

If you have problems, check out the kernel `HOWTO`. There are a couple things you need to have covered: you need your kernel modules compiled and living in `/etc/conf.modules`.

3. The Simple Configuration

Most Lilo installations use a configuration file like the following one:

```
boot = /dev/hda    # or your root partition
delay = 10         # delay, in tenth of a second (so you can interact)
vga = 0            # optional. Use "vga=1" to get 80x50
#linear           # try "linear" in case of geometry problems.

image = /boot/vmlinux  # your zImage file
  root = /dev/hda1     # your root partition
  label = Linux        # or any fancy name
  read−only            # mount root read−only

other = /dev/hda4   # your dos partition, if any
  table = /dev/hda  # the current partition table
  label = dos       # or any non−fancy name
```

You can have multiple ``image'' and ``other'' sections if you want. It's not uncommon to have several kernel images configured in your *lilo.conf*, at least if you keep up to date with kernel development.

3.1 How to Deal with Big Kernels

If you compile a ``zImage'' kernel and it is too big to fit in half a megabyte (this is commong with new 2.1 kernels), you should build a ``big zImage'' instead: ``make bzImage''. To boot a big kernel image nothing special is needed, but you need version 18 or newer of Lilo. If your installation is older, you should upgrade your Lilo package.

3.2 How to Boot Windows NT from Lilo menu

Here I will give you an order of routines you have to do if you want to have both Linux and NT entries under Lilo menu:

- First of all, I would suggest you to install a fresh copy of Windows NT 4.0 on your hard disk. I suppose that you already made a backup of your important data, so the NT installation shouldn't be a problem. During the NT installation, setup is not going to ask you where to place NT's boot loader, so it would be placed into the MBR (Master Boot Record) of your hard disk. But, there is a possibility for a previous content of the MBR to remain within the MBR (especially any previous Lilo), so I would suggest you (before installation of NT) to boot the computer with a DOS floppy diskette having DOS version of FDISK. At the prompt a:\ just enter the command: fdisk /mbr and restart the computer again (without that floppy).

- After you have successfully installed your NT, you will see that it uses the whole hard disk or a specific partition of the hard disk (depending on what you decided during the setup process). So, it is advisible to 'shrink' the partition where NT resides in order to make some free space on the disk. Onto that free space you will install your Linux. After you have your NT configured and running, you have to boot your computer using a floppy diskette with Partition Magic utility by Power Quest. It is a graphical tool able to see all partitions on all hard disks you have. The best thing is that you can make some changes with your partitions but not to destroy your existing data. One of the available changes is to make your existing partition(s) smaller, so to get some free space on the disk(s) for other purposes. Although you are advised to make a backup before you make any changes to the partitions, I usually practise to 'shrink' NT's partition before I installed anything but NT itself (so, if needed, a repetitive re−installation wouldn't be a problem). Well, Partition Magic (or any other similar utility you are familiar with) will shrink your NT's partition (either NTFS or FAT) to a smaller measure and place it to either the beginning or to the end of the previous measure. It means that you may choose to have your 'shrinked' NT partition at the beginning or at the end of your disk (I usually choose NT to be at the beginning, so the ending part of the disk will become a 'free space'). After the 'shrinkin' is finished, you may re−boot your NT in order to check the new situation: you may use Windows Explorer or Disk Administrator for that.

- So far so good. Next step is to install your Linux. Case you are familiar with RedHat distribution (I hope with other distros is the same or similar), you start by putting your installation CD in the drive and re−boot the computer). Well, when you are about to choose what type of installation it will be (Gnome or KDE Workstation, Custom, etc.) you may choose whatever you planned before, but I would suggest to install a Workstation at first. This is good because Linux setup will find automatically the free space on the (first) hard disk, make all partitions needed for Linux, format them properly, make majority of option by default so you won't have much pain during the setup (later, if you want, you may either to add missing components or re−install Linux as Custom over the existing linux partitions). Lilo should go to the MBR.
- After it looks that Linux installation is finished, you are going to re−start the computer and there there you will only see Lilo with one Linux entry to boot (or maybe more than one Linux entry, in case your hardware is multi−processor one). But, don't panic! Your Windows NT is still there where you had installed it before Linux. You should become some familiar with Linux as soon as possible, in order to be able to find and edit your new /etc/lilo.conf file. When you open this file for the first time, you'll see that there is only one (or more) Linux entry. Well, you should know the exact position (read: a partition) where Windows NT has been installed, so you could add an appropriate entry into /etc/lilo.conf file. After you do that, restart Lilo and, after the next re−boot, you will have both 'linux' and 'nt' entries under Lilo menu.

4. Installing hdc to Boot as hda and Using bios=

Lilo allows to map the kernel image from one disk and instruct the BIOS to retrieve it from another disk. For example, it's common for me to install Linux on a disk I connect to `hdc` (master disk of secondary controller) and boot it as a standalong system on the primary IDE controller of another computer. I copied the installation floppy to a tiny partition, so I can run *chroot* in a virtual console to install `hdc` while I use the system to do something else.

The *lilo.conf* file I use to install Lilo looks like:

```
# This file must be used from a system running off /dev/hdc
boot = /dev/hdc   # overwrite MBR of hdc
disk = /dev/hdc   # tell how hdc will look like:
   bios = 0x80    #  the bios will see it as first drive
delay = 0
vga = 0

image = /boot/vmlinux  # this is on /dev/hdc1
  root = /dev/hda1     # but at boot it will be hda1
  label = Linux
  read−only
```

This configuration file must be read by a Lilo running **off /dev/hdc1**. The Lilo maps that get written the boot sector (`/dev/hdc`) must refer to the files in `/boot` (currently installed as hdc); such files will be accessed under hda when this disk will be booted as a standalone system.

I call this configuration file `/mnt/etc/lilo.conf.hdc` (`/mnt` is where hdc is mounted during the installation. I install Lilo by invoking ``` ``cd /mnt; chroot . sbin/lilo −C /etc/lilo.conf.hdc ```". Refer to the manual page for *chroot* if this looks magic.

The ``` ``bios= ```" directive in `lilo.conf` is used to tell Lilo what the BIOS thinks of your devices. BIOS calls identify floppy disks and hard drives with a number: 0x00 and 0x01 select the floppy drives, 0x80 and the following numbers select hard disks (old BIOSes can only access two disks). The meaning of ``` ``bios = 0x80 ``` in the previous sample file is therefore ``use 0x80 in your BIOS calls for `/dev/hdc`".

This Lilo directive can be handy in other situations, for example when your BIOS is able to boot from SCSI disks instead of IDE ones. When both IDE and SCSI devices are there, Lilo can't tell whether 0x80 will refer to one or the other because the user is able to choose it in the BIOS configuration menus, and the BIOS can't be accessed while Linux is running.

By default, Lilo assumes that IDE drives are mapped first by the BIOS, but this can be overridden by using instructions like these in `/etc/lilo.conf`:

```
disk = /dev/sda
  bios = 0x80
```

5. Using Lilo When the BIOS Can't See the Root Partition

I have two IDE drives, and a SCSI drive. The SCSI drive can't be seen from BIOS. The Linux Loader, Lilo, uses BIOS calls and can only see drives that BIOS can see. My stupid AMI BIOS will only boot from "A:" or "C:" My root file system is on a partition on the SCSI drive.

The solution consists in storing the kernel, map file, and chain loader in a Linux partition on the first IDE. Notice that it is not necessary to keep your kernel on your root partition.

The second partition on my first IDE (`/dev/hda2`, the Linux partition used to boot the system) is mounted on `/u2`. Here is the `/etc/lilo.conf` file I used.

```
#  Install Lilo on the Master Boot Record
#  on the first IDE.
#
boot = /dev/hda
#  /sbin/lilo (the installer) copies the Lilo boot record
#  from the following file to the MBR location.
install = /u2/etc/lilo/boot.b
#
#  I wrote a verbose boot menu.  Lilo finds it here.
message = /u2/etc/lilo/message
#  The installer will build the following file. It tells
#  the boot−loader where the blocks of the kernels are.
map = /u2/etc/lilo/map
compact
prompt
#  Wait 10 seconds, then boot the 1.2.1 kernel by default.
timeout = 100
#  The kernel is stored where BIOS can see it by doing this:
```

```
#      cp −p /usr/src/linux/arch/i386/boot/zImage /u2/z1.2.1
image = /u2/z1.2.1
        label = 1.2.1
#  Lilo tells the kernel to mount the first SCSI partition
#  as root.  BIOS does not have to be able to see it.
        root = /dev/sda1
#  This partition will be checked and remounted by /etc/rc.d/rc.S
        read−only
#  I kept an old Slackware kernel lying around in case I built a
#  kernel that doesn't work.  I actually needed this once.
image = /u2/z1.0.9
        label = 1.0.9
        root = /dev/sda1
        read−only
#  My DR−DOS 6 partition.
other = /dev/hda1
        loader=/u2/etc/lilo/chain.b
        label = dos
        alias = m
```

6. Accessing Huge Disks When the BIOS Can't

Notice: 1GB is "Huge"? Well, once upon a time...

The system in my office has a 1GB IDE drive. The BIOS can only see the first 504 MB of the IDE. (Where MB means 2**10 bytes, not 10**6 bytes.) So I have MS−DOS on a 350 MB partition `/dev/hda1` and my Linux root on a 120 MB partition `/dev/hda2`.

MS−DOS was unable to install itself correctly when the drive was fresh. Novell DOS 7 had the same problem. Luckily for me, "Options by IBM" forgot to put the "OnTrack" diskette in the box with the drive. The drive was supposed to come with a product called "OnTrack Disk Manager." If you only have MSDOS, I guess you have to use it.

So I made a partition table with Linux' fdisk. MSDOS−6.2 refused to install itself in `/dev/hda1`. It said something like ``this release of MS−DOS is for new installations. Your computer already has MS−DOS so you need to get an upgrade release from your dealer." Actually, the disk was brand new.

What a crock! So I ran Linux' fdisk again and deleted partition 1 from the table. This satisfied MS−DOS 6.2 which proceeded to create the exact same partition 1 I had just deleted and installed itself. MS−DOS 6.2 wrote its Master Boot Record on the drive, but it couldn't boot.

Luckily I had a Slackware kernel on floppy (made by the Slackware installation program "setup"), so I booted Linux and wrote Lilo over MS−DOS' broken MBR. This works. Here is the `/etc/lilo.conf` file I used:

```
boot = /dev/hda
map = /lilo−map
delay = 100
ramdisk = 0             # Turns off ramdisk in Slackware kernel
timeout = 100
prompt
disk = /dev/hda         # BIOS only sees first 500 MB.
   bios = 0x80          # specifies the first IDE.
   sectors = 63         # get the numbers from your drive's docs.
   heads = 16
   cylinders = 2100
image = /vmlinuz
  append = "hd=2100,16,63"
  root = /dev/hda2
  label = linux
  read−only
  vga = extended
other = /dev/hda1
  label = msdos
  table = /dev/hda
  loader = /boot/chain.b
```

After I installed these systems, I verified that the partition containing the zImage, boot.b, map, chain.b, and message files can use an msdos file system, as long as it is not "stackered" or "doublespaced." So I could have made the DOS partition on `/dev/hda1` 500 MB.

I have also learned that "OnTrack" would have written a partition table starting a few dozen bytes into the drive, instead of at the beginning, and it is possible to hack the Linux IDE driver to work around this problem. But installing would have been impossible with the precompiled Slackware kernel. Eventually, IBM sent me an "OnTrack" diskette. I called OnTrack's technical support. They told me Linux is broken because Linux doesn't use BIOS. I gave their diskette away.

7. Booting from a Rescue Floppy

Next, I installed Windows−95 on my office system. It blew away my nice Lilo MBR, but it left my Linux partitions alone. Kernels take a long time to load from floppy, so I made a floppy with a working Lilo setup on it, which could boot my kernel from the IDE.

I made the lilo floppy like so:

```
  fdformat /dev/fd0H1440      #  lay tracks on virgin diskette
  mkfs −t minix /dev/fd0 1440 #  make file system of type minix
  mount /dev/fd0 /mnt         #  mount in the standard tmp mount point
  cp −p /boot/chain.b /mnt    #  copy the chain loader over
  lilo −C /etc/lilo.flop      #  install Lilo and the map on the diskette.
  umount /mnt
```

Notice that the diskette **must be mounted when you run the installer** so that Lilo can write its map file properly.

This file is /etc/lilo.flop. It's almost the same as the last one:

```
#  Makes a floppy that can boot kernels from HD.
boot = /dev/fd0
map = /mnt/lilo−map
delay = 100
ramdisk = 0
timeout = 100
prompt
disk = /dev/hda     # 1 GB IDE, BIOS only sees first 500 MB.
   bios=0x80
   sectors = 63
   heads = 16
   cylinders = 2100
image = /vmlinuz
  append = "hd=2100,16,63"
  root = /dev/hda2
  label = linux
  read-only
  vga = extended
other = /dev/hda1
  label = msdos
  table = /dev/hda
  loader = /mnt/chain.b
```

Finally, I needed MS−DOS 6.2 on my office system, but I didn't want to touch the first drive. I added a SCSI controller and drive, made an msdos file system on it with Linux' mkdosfs, and Windows−95 sees it as "D:". But of course MSDOS will not boot off of D:. This is not a problem when you have Lilo. I added the following to the `lilo.conf` in Example 2.

```
other = /dev/sda1
  label = d6.2
  table = /dev/sda
  loader = /boot/any_d.b
```

With this modification MSDOS−6.2 runs, and it thinks it is on C: and Windows−95 is on D:.

8. Further Information

8.1 Copyright

8.2 Disclaimer

Use the information in this document at your own risk. I disavow any potential liability of this document. Use of the concepts, examples, and/or other content of this document is entirely at your own risk.

All copyrights are owned by their owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

Naming of particular products or brands should not be seen as endorsements.

You are strongly recommended to take a backup of your system before major installation and backups at regular intervals.

8.3 News

In addition to the Lilo docs, there are a number of mini−howto's that can be useful for your needs. All of them are called ``Linux+*foobarOS*", for some *foobarOS*, they deal with coexistence of Linux and other operationg system(s). For example, "NT OS Loader + Linux mini−HOWTO" by Bernd Reichert, describes how to add an entry for Linux under existing Windows NT Loader's menu. Next, you have "Linux+WindowsNT mini−HOWTO" by myself, covering how to add an entry for NT under existing Linux Lilo menu (more detailed than here). Also, "Multiboot−with−LILO" describes how the various Windows flavours can be made to coexist with Linux.

This mini−HOWTO would be improved from time to time. If you think that the HOWTO on your Linux installation CD is some out−of−date, you may check for newest release on the Internet. It could be found within the main Linux Documentation Project homepage.

8.4 Credits

This version of mini−HOWTO can thanks to:

```
        Cameron Spitzer (<tt/cls@truffula.sj.ca.us/)
        Alessandro Rubini (<tt/rubini@linux.it/)
        Tony Harris (<tt/tony@nmr.mgh.harvard.edu/)
```

Any comments or suggestions can be mailed to my email address: m.skoric@eunet.yu.

8.5 HOWTO

These are intended as the primary starting points to get the background information as well as show you how to solve a specific problem. Some relevant HOWTOs are `Bootdisk`, `Installation`, `SCSI` and `UMSDOS`. The main site for these is the LDP archive at Metalab (formerly known as Sunsite).

8.6 Mini−HOWTO

These are the smaller free text relatives to the HOWTOs. Some relevant mini−HOWTOs are `Backup−With−MSDOS`, `Diskless`, `LILO`, `Large Disk`, `Linux+DOS+Win95+OS2`, `Linux+OS2+DOS`, `Linux+Win95`, `Linux+WinNT`, `Linux+NT−Loader`, `NFS−Root`, `Win95+Win+Linux`, `ZIP Drive`, `FBB packet−radio BBS`. You can find these at the same place as the HOWTOs, usually in a sub directory called `mini`. Note that these are scheduled to be converted into SGML and become proper HOWTOs in the near future.

8.7 Local Resources

In most distributions of Linux there is a document directory installed, have a look in the /usr/doc directory. where most packages store their main documentation and README files etc. Also you will here find the HOWTO archive (/usr/doc/HOWTO) of ready formatted HOWTOs and also the mini−HOWTO archive (/usr/doc/HOWTO/mini) of plain text documents.

Many of the configuration files mentioned earlier can be found in the /etc directory. In particular you will want to work with the /etc/fstab file that sets up the mounting of partitions and possibly also /etc/mdtab file that is used for the `md` system to set up RAID.

The kernel source in /usr/src/linux is, of course, the ultimate documentation. In other words, *use the source, Luke*. It should also be pointed out that the kernel comes not only with source code which is even commented (well, partially at least) but also an informative documentation directory. If you are about to ask any questions about the kernel you should read this first, it will save you and many others a lot of time and possibly embarrassment.

Also have a look in your system log file (/var/log/messages) to see what is going on and in particular how the booting went if too much scrolled off your screen. Using `tail -f /var/log/messages` in a separate window or screen will give you a continuous update of what is going on in your system.

You can also take advantage of the /proc file system that is a window into the inner workings of your system. Use `cat` rather than `more` to view the files as they are reported as being zero length. Reports are that `less` works well here.

8.8 Web Pages

There is a huge number of informative web pages out there and by their very nature they change quickly so don't be too surprised if these links become quickly outdated.

A good starting point is of course the Linux Documentation Project home page, an information central for documentation, project pages and much, much more.

Please let me know if you have any other leads that can be of interest.

---

9. GETTING HELP

In the end you might find yourself unable to solve your problems and need help from someone else. The most efficient way is either to ask someone local or in your nearest Linux user group, search the web for the nearest one.

Another possibility is to ask on Usenet News in one of the many, many newsgroups available. The problem is that these have such a high volume and noise (called low signal−to−noise ratio) that your question can easily fall through unanswered.

No matter where you ask it is important to ask well or you will not be taken seriously. Saying just *my disk does not work* is not going to help you and instead the noise level is increased even further and if you are lucky someone will ask you to clarify.

Instead describe your problems in some detail that will enable people to help you. The problem could lie somewhere you did not expect. Therefore you are advised to list up the following information on your system:

Hardware

- Processor
- DMA
- IRQ
- Chip set (LX, BX etc)
- Bus (ISA, VESA, PCI etc)
- Expansion cards used (Disk controllers, video, IO etc)

Software

- BIOS (On motherboard and possibly SCSI host adapters)
- LILO, if used
- Linux kernel version as well as possible modifications and patches
- Kernel parameters, if any
- Software that shows the error (with version number or date)

Peripherals

- Type of disk drives with manufacturer name, version and type
- Other relevant peripherals connected to the same busses

Remember that booting text is logged to `/var/log/messages` which can answer most of the questions above. Obviously if the drives fail you might not be able to get the log saved to disk but you can at least scroll back up the screen using the `SHIFT` and `PAGE UP` keys. It may also be useful to include part of this in your request for help but do not go overboard, keep it *brief* as a complete log file dumped to Usenet News is more than a little annoying.