The Linux Bootdisk HOWTO

Tom Fawcett

Bootdisk−HOWTO@linuxdoc.org

v4.2, November 2000

This document describes how to design and build boot/root diskettes for Linux. These disks can be used as rescue disks or to test new system components. You should be reasonably familiar with system administration tasks before attempting to build a bootdisk. If you just want a rescue disk to have for emergencies, see Appendix A.1.

Table of Contents

Table of Contents

1. Preface

> **Important:** This document may be outdated. If the date on the title page is more than six months ago, please check the Bootdisk−HOWTO homepage to see if a more recent version exists.

Although this document should be legible in its text form, it looks much better in Postscript, PDF or HTML forms because of the typographical conventions used.

1.1. Version notes

Graham Chapman wrote the original Bootdisk−HOWTO and supported it through version 3.1. Tom Fawcett started as co−author around the time kernel v2 was introduced, and he is the document's current maintainer. Chapman has disappeared from the Linux community and his whereabouts are currently unknown.

This information is intended for Linux on the *Intel* platform. Much of this information may be applicable to Linux on other processors, but I have no first−hand experience or information about this. If you have experience with bootdisks on other platforms, please contact me.

1.2. Yet to do

1. Describe how to create bootable LS110 disks.
2. Describe how to deal with the huge `libc.so` shared libraries. The options are basically to get older, smaller libraries or to cut down existing libraries.
3. Re−analyze distribution bootdisks and update the "How the Pros do it" section.
4. Delete section that describes how to upgrade existing distribution bootdisks. This is usually more trouble than it's worth.

1.3. Feedback and credits

I welcome any feedback, good or bad, on the content of this document. I have done my best to ensure that the instructions and information herein are accurate and reliable. Please let me know if you find errors or omissions. When writing, please indicate the version number of the document you're referencing.

We thank the many people who assisted with corrections and suggestions. Their contributions have made it far better than we could ever have done alone.

Send comments and corrections to the author at the email address above. *Please read Section 7 before asking me questions*. Do *not* email me disk images.

1.4. Distribution policy

---

2. Introduction

Linux boot disks are useful in a number of situations, such as testing a new kernel, recovering from a disk failure (anything from a lost boot sector to a disk head crash), fixing a disabled system, or upgrading critical system files safely (such as `libc.so`).

There are several ways of obtaining boot disks:

- Use one from a distribution such as Slackware. This will at least allow you to boot.
- Use a rescue package to set up disks designed to be used as rescue disks.
- Learn what is required for each of the types of disk to operate, then build your own.

Some people choose the last option so they can do it themselves. That way, if something breaks, they can work out what to do to fix it. Plus it's a great way to learn about how a Linux system works.

This document assumes some basic familiarity with Linux system administration concepts. For example, you should know about directories, filesystems and floppy diskettes. You should know how to use **mount** and **df**. You should know what `/etc/passwd` and `fstab` files are for and what they look like. You should know that most of the commands in this HOWTO should be run as root.

Constructing a bootdisk from scratch can be complicated. If you haven't read the Linux FAQ and related documents, such as the Linux Installation HOWTO and the Linux Installation Guide, you should not be trying to build boot diskettes. If you just need a working bootdisk for emergencies, it is *much* easier to download a prefabricated one. See Appendix A.1, below, for where to find these.

3. Bootdisks and the boot process

A bootdisk is basically a miniature, self−contained Linux system on a diskette. It must perform many of the same functions that a complete full−size Linux system performs. Before trying to build one you should understand the basic Linux boot process. Here we present the basics, which are sufficient for understanding the rest of this document. Many details and alternative options have been omitted.

3.1. The boot process

All PC systems start the boot process by executing code in ROM (specifically, the *BIOS*) to load the sector from sector 0, cylinder 0 of the boot drive. The boot drive is usually the first floppy drive (designated `A:` in DOS and `/dev/fd0` in Linux). The BIOS then tries to execute this sector. On most bootable disks, sector 0, cylinder 0 contains either:

- code from a boot loader such as LILO, which locates the kernel, loads it and executes it to start the boot proper; or
- the start of an operating system kernel, such as Linux.

If a Linux kernel has been raw−copied to a diskette, the first sector of the disk will be the first sector of the Linux kernel itself. This first sector will continue the boot process by loading the rest of the kernel from the boot device.

When the kernel is completely loaded, it initializes device drivers and its internal data structures. Once it is completely initialized, it consults a special location in its image called the *ramdisk word*. This word tells it how and where to find its *root filesystem*. A root filesystem is simply a filesystem that will be mounted as ``/''. The kernel has to be told where to look for the root filesystem; if it cannot find a loadable image there, it halts.

In some boot situations   often when booting from a diskette   the root filesystem is loaded into a *ramdisk*, which is RAM accessed by the system as if it were a disk. RAM is several orders of magnitude faster than a floppy disk, so system operation is fast from a ramdisk. Also, the kernel can load a *compressed filesystem* from the floppy and uncompress it onto the ramdisk, allowing many more files to be squeezed onto the diskette.

Once the root filesystem is loaded and mounted, you see a message like:

```
        VFS: Mounted root (ext2 filesystem) readonly.
```

Once the system has loaded a root filesystem successfully, it tries to execute the `init` program (in `/bin` or `/sbin`). `init` reads its configuration file `/etc/inittab`, looks for a line designated `sysinit`, and executes the named script. The `sysinit` script is usually something like `/etc/rc` or `/etc/init.d/boot`. This script is a set of shell commands that set up basic system services, such as running **fsck** on hard disks, loading necessary kernel modules, initializing swapping, initializing the network, and mounting disks mentioned in `/etc/fstab`.

This script often invokes various other scripts to do modular initialization. For example, in the common SysVinit structure, the directory `/etc/rc.d/` contains a complex structure of subdirectories whose files specify how to enable and shut down most system services. However, on a bootdisk the sysinit script is often very simple.

When the sysinit script finishes control returns to **init**, which then enters the *default runlevel*, specified in `inittab` with the `initdefault` keyword. The runlevel line usually specifies a program like **getty**, which is responsible for handling commununications through the console and ttys. It is the **getty** program which prints the familiar ``login:'' prompt. The **getty** program in turn invokes the **login** program to handle login validation and to set up user sessions.

3.2. Disk types

Having reviewed the basic boot process, we can now define various kinds of disks involved. We classify disks into four types. The discussion here and throughout this document uses the term ``disk'' to refer to floppy diskettes unless otherwise specified, though most of the discussion could apply equally well to hard disks.

boot

> A disk containing a kernel which can be booted. The disk can be used to boot the kernel, which then may load a root file system on another disk. The kernel on a bootdisk usually must be told where to find its root filesystem.
>
> Often a bootdisk loads a root filesystem from another diskette, but it is possible for a bootdisk to be set up to load a hard disk's root filesystem instead. This is commonly done when testing a new kernel (in fact, ``**make zdisk**'' will create such a bootdisk automatically from the kernel source code).

root

> A disk with a filesystem containing files required to run a Linux system. Such a disk does not necessarily contain either a kernel or a boot loader.
>
> A root disk can be used to run the system independently of any other disks, once the kernel has been booted. Usually the root disk is automatically copied to a ramdisk. This makes root disk accesses much faster, and frees up the disk drive for a utility disk.

boot/root

> A disk which contains both the kernel and a root filesystem. In other words, it contains everything necessary to boot and run a Linux system without a hard disk. The advantage of this type of disk is that is it compact everything required is on a single disk. However, the gradually increasing size of everything means that it is increasingly difficult to fit everything on a single diskette, even with compression.

utility

> A disk which contains a filesystem, but is not intended to be mounted as a root file system. It is an additional data disk. You would use this type of disk to carry additional utilities where you have too much to fit on your root disk.

In general, when we talk about ``building a bootdisk'' we mean creating both the boot (kernel) and root (files) portions. They may be either together (a single boot/root disk) or separate (boot + root disks). The most flexible approach for rescue diskettes is probably to use separate boot and root diskettes, and one or more utility diskettes to handle the overflow.

4. Building a root filesystem

Creating the root filesystem involves selecting files necessary for the system to run. In this section we describe how to build a *compressed root filesystem*. A less common option is to build an uncompressed filesystem on a diskette that is directly mounted as root; this alternative is described in Section 8.2.

4.1. Overview

A root filesystem must contain everything needed to support a full Linux system. To be able to do this, the disk must include the minimum requirements for a Linux system:

- The basic file system structure,
- Minimum set of directories: `/dev`, `/proc`, `/bin`, `/etc`, `/lib`, `/usr`, `/tmp`,
- Basic set of utilities: `sh`, `ls`, `cp`, `mv`, etc.,
- Minimum set of config files: `rc, inittab, fstab`, etc.,
- Devices: `/dev/hd*, /dev/tty*, /dev/fd0`, etc.,
- Runtime library to provide basic functions used by utilities.

Of course, any system only becomes useful when you can run something on it, and a root diskette usually only becomes useful when you can do something like:

- Check a file system on another drive, for example to check your root file system on your hard drive, you need to be able to boot Linux from another drive, as you can with a root diskette system. Then you can run **fsck** on your original root drive while it is not mounted.
- Restore all or part of your original root drive from backup using archive and compression utilities such as `cpio`, `tar`, `gzip` and `ftape`.

We describe how to build a *compressed* filesystem, so called because it is compressed on disk and, when booted, is uncompressed onto a ramdisk. With a compressed filesystem you can fit many files (approximately six megabytes) onto a standard 1440K diskette. Because the filesystem is much larger than a diskette, it cannot be built on the diskette. We have to build it elsewhere, compress it, then copy it to the diskette.

4.2. Creating the filesystem

In order to build such a root filesystem, you need a spare device that is large enough to hold all the files before compression. You will need a device capable of holding about four megabytes. There are several choices:

- Use a *ramdisk* (DEVICE = `/dev/ram0`). In this case, memory is used to simulate a disk drive. The ramdisk must be large enough to hold a filesystem of the appropriate size. If you use LILO, check your configuration file (`/etc/lilo.conf`) for a line like `RAMDISK = nnn` which determines the maximum RAM that can be allocated to a ramdisk. The default is 4096K, which should be sufficient. You should probably not try to use such a ramdisk on a machine with less than 8MB of RAM. Check to make sure you have a device like `/dev/ram0, /dev/ram` or `/dev/ramdisk`. If not, create `/dev/ram0` with **mknod** (major number 1, minor 0).
- If you have an unused hard disk partition that is large enough (several megabytes), this is acceptable.
- Use a *loopback device*, which allows a disk file to be treated as a device. Using a loopback device you can create a three megabyte file on your hard disk and build the filesystem on it.

  Type **man losetup** for instructions on using loopback devices. If you don't have **losetup**, you can get it along with compatible versions of **mount** and **unmount** from the `util-linux` package in the directory `ftp://ftp.win.tue.nl/pub/linux/utils/util-linux/`.

  If you do not have a loop device (`/dev/loop0`, `/dev/loop1`, etc.) on your system, you will have to create one with ``**mknod /dev/loop0 b 7 0**''. Once you've installed these special **mount** and **umount** binaries, create a temporary file on a hard disk with enough capacity (eg, `/tmp/fsfile`). You can use a command like:

  ```
          dd if=/dev/zero of=/tmp/fsfile bs=1k count=nnn
  ```

  to create an `nnn`−block file.

  Use the file name in place of DEVICE below. When you issue a mount command you must include the option `-o loop` to tell mount to use a loopback device. For example:

  ```
          mount -o loop -t ext2 /tmp/fsfile /mnt
  ```

will mount `/tmp/fsfile` via a loopback device at the mount point `/mnt`. A **df** will confirm this.

After you've chosen one of these options, prepare the DEVICE with:

```
        dd if=/dev/zero of=DEVICE bs=1k count=4096
```

This command zeroes out the device.

> **Important:** Zeroing the device is critical because the filesystem will be compressed later, so all unused portions should be filled with zeroes to achieve maximum compression. Keep this in mind whenever you move or delete files on the filesystem. The filesystem will correctly de−allocate the blocks, *but it will not zero them out again*. If you do a lot of deletions and copying, your compressed filesystem may end up much larger than necessary.

Next, create the filesystem. The Linux kernel recognizes two file system types for root disks to be automatically copied to ramdisk. These are minix and ext2, of which ext2 is preferred. If using ext2, you may find it useful to use the `-i` option to specify more inodes than the default; `-i 2000` is suggested so that you don't run out of inodes. Alternatively, you can save on inodes by removing lots of unnecessary `/dev` files. **mke2fs** will by default create 360 inodes on a 1.44Mb diskette. I find that 120 inodes is ample on my current rescue root diskette, but if you include all the devices in the `/dev` directory then you will easily exceed 360. Using a compressed root filesystem allows a larger filesystem, and hence more inodes by default, but you may still need to either reduce the number of files or increase the number of inodes.

So the command you use will look like:

```
        mke2fs -m 0 -i 2000 DEVICE
```

(If you're using a loopback device, the disk file you're using should be supplied in place of this DEVICE.)

The **mke2fs** command will automatically detect the space available and configure itself accordingly. The ``-m 0'' parameter prevents it from reserving space for root, and hence provides more usable space on the disk.

Next, mount the device:

```
        mount -t ext2 DEVICE /mnt
```

(You must create a mount point `/mnt` if it does not already exist.) In the remaining sections, all destination directory names are assumed to be relative to `/mnt`.

4.3. Populating the filesystem

Here is a reasonable minimum set of directories for your root filesystem [1]:

- `/dev` −− Device files, required to perform I/O
- `/proc` −− Directory stub required by the proc filesystem
- `/etc` −− System configuration files
- `/sbin` −− Critical system binaries
- `/bin` −− Essential binaries considered part of the system
- `/lib` −− Shared libraries to provide run−time support
- `/mnt` −− A mount point for maintenance on other disks
- `/usr` −− Additional utilities and applications

Three of these directories will be empty on the root filesystem, so they only need to be created with **mkdir**. The `/proc` directory is basically a stub under which the proc filesystem is placed. The directories `/mnt` and `/usr` are only mount points for use after the boot/root system is running. Hence again, these directories only need to be created.

The remaining four directories are described in the following sections.

4.3.1. /dev

A `/dev` directory containing a special file for all devices to be used by the system is mandatory for any Linux system. The directory itself is a normal directory, and can be created with `mkdir` in the normal way. The device special files, however, must be created in a special way, using the **mknod** command.

There is a shortcut, though   copy your existing `/dev` directory contents, and delete the ones you don't want. The only requirement is that you copy the device special files using `−R` option. This will copy the directory without attempting to copy the contents of the files. Be sure to use an *upper case `R`*. The command is:

```
        cp -dpR /dev /mnt
```

assuming that the diskette is mounted at `/mnt`. The `dp` switches ensure that symbolic links are copied as links, rather than using the target file, and that the original file attributes are preserved, thus preserving ownership information.

If you want to do it the hard way, use `ls -l` to display the major and minor device numbers for the devices you want, and create them on the diskette using `mknod`.

However the devices are copied, it is worth checking that any special devices you need have been placed on the rescue diskette. For example, `ftape` uses tape devices, so you will need to copy all of these if you intend to access your floppy tape drive from the bootdisk.

Note that one inode is required for each device special file, and inodes can at times be a scarce resource, especially on diskette filesystems. It therefore makes sense to remove any device special files that you don't need from the diskette `/dev` directory. For example, if you do not have SCSI disks you can safely remove all the device files starting with `sd`. Similarly, if you don't intend to use your serial port then all device files starting with `cua` can go.

> **Important:** Be sure to include the following files from this directory: `console`, `kmem`, `mem`, `null`, `ram0` and `tty1`.

4.3.2. /etc

This directory contains important configuration files. On most systems, these can be divided into three groups:

1. Required at all times, *e.g.* `rc`, `fstab`, `passwd`.
2. May be required, but no one is too sure.
3. Junk that crept in.

Files which are not essential can usually be identified with the command:

```
        ls -ltru
```

This lists files in reverse order of date last accessed, so if any files are not being accessed, they can be omitted from a root diskette.

On my root diskettes, I have the number of config files down to 15. This reduces my work to dealing with three sets of files:

1. The ones I must configure for a boot/root system:
    a. `rc.d/*` –– system startup and run level change scripts
    b. `fstab` –– list of file systems to be mounted
    c. `inittab` –– parameters for the **init** process, the first process started at boot time.
2. The ones I should tidy up for a boot/root system:
    a. `passwd` –– Critical list of users, home directories, etc.
    b. `group` –– user groups.
    c. `shadow` –– passwords of users. You may not have this.
    d. `termcap` –– the terminal capability database.

If security is important, passwd and shadow should be pruned to avoid copying user passwords off the system, and so that unwanted logins are rejected when you boot from diskette.

Be sure that passwd contains at least root. If you intend other users to login, be sure their home directories and shells exist.

termcap, the terminal database, is typically several hundred kilobytes. The version on your boot/root diskette should be pruned down to contain only the terminal(s) you use, which is usually just the linux or linux-console entry.

3. The rest. They work at the moment, so I leave them alone.

Out of this, I only really have to configure two files, and what they should contain is surprisingly small.

- rc should contain:

```
        #!/bin/sh
        /bin/mount -av
        /bin/hostname Kangaroo
```

  Be sure it is executable, and be sure any absolute filenames are correct. You don't really need to run **hostname** it just looks nicer if you do.
- fstab should contain at least:

```
        /dev/ram0       /               ext2    defaults
        /dev/fd0        /               ext2    defaults
        /proc           /proc           proc    defaults
```

You can copy entries from your existing fstab, but you should not automatically mount any of your hard disk partitions; use the noauto keyword with them. Your hard disk may be damaged or dead when the bootdisk is used.

Your inittab should be changed so that its sysinit line runs rc or whatever basic boot script will be used. Also, if you want to ensure that users on serial ports cannot login, comment out all the entries for getty which include a ttys or ttyS device at the end of the line. Leave in the tty ports so that you can login at the console.

A minimal inittab file looks like this:

```
        id:2:initdefault:
        si::sysinit:/etc/rc
        1:2345:respawn:/sbin/getty 9600 tty1
        2:23:respawn:/sbin/getty 9600 tty2
```

The inittab file defines what the system will run in various states including startup, move to multi−user mode, etc. Check carefully the filenames mentioned in inittab; if init cannot find the program mentioned the bootdisk will hang, and you may not even get an error message.

Note that some programs cannot be moved elsewhere because other programs have hardcoded their locations. For example, on my system, /etc/shutdown has hardcoded in it /etc/reboot. If I move reboot to /bin/reboot, and then issue a shutdown command, it will fail because it cannot find the reboot file.

For the rest, just copy all the text files in your /etc directory, plus all the executables in your /etc directory that you cannot be sure you do not need. As a guide, consult the sample listing in Appendix C. Probably it will suffice to copy only those files, but systems differ a great deal, so you cannot be sure that the same set of files on your system is equivalent to the files in the list. The only sure method is to start with inittab and work out what is required.

Most systems now use an /etc/rc.d/ directory containing shell scripts for different run levels. The minimum is a single rc script, but it may be simpler just to copy inittab and the /etc/rc.d directory from your existing system, and prune the shell scripts in the rc.d directory to remove processing not relevent to a diskette system environment.

4.3.3. /bin and /sbin

The /bin directory is a convenient place for extra utilities you need to perform basic operations, utilities such as **ls**, **mv**, **cat** and **dd**. See Appendix C for an example list of files that go in a **/bin** and /sbin directories. It does not include any of the utilities required to restore from backup, such as **cpio**, **tar** and **gzip**. That is because I place these on a separate utility diskette, to save space on the boot/root diskette. Once the boot/root diskette is booted, it is copied to the ramdisk leaving the diskette drive free to mount another diskette, the utility diskette. I usually mount this as /usr.

Creation of a utility diskette is described below in Section 8.3. It is probably desirable to maintain a copy of the same version of backup utilities used to write the backups so you don't waste time trying to install versions that cannot read your backup tapes.

> **Important:** Be sure to include the following programs: init, getty or equivalent, login, mount, some shell capable of running your rc scripts, a link from sh to the shell.

4.3.4. /lib

In /lib you place necessary shared libraries and loaders. If the necessary libraries are not found in your /lib directory then the system will be unable to boot. If you're lucky you may see an error message telling you why.

Nearly every program requires at least the libc library, libc.so.*N*, where *N* is the current version number. Check your /lib directory. The file libc.so.N is usually a symlink to a filename with a complete version number:

```
% ls -l /lib/libc*
-rwxr-xr-x   1 root     root      4016683 Apr 16 18:48 libc-2.1.1.so*
lrwxrwxrwx   1 root     root           13 Apr 10 12:25 libc.so.6 -> libc-2.1.1.so*
```

In this case, you want libc-2.1.1.so. To find other libraries you should go through all the binaries you plan to include and check their dependencies with **ldd**. For example:

```
        % ldd /sbin/mke2fs
        libext2fs.so.2 => /lib/libext2fs.so.2 (0x40014000)
        libcom_err.so.2 => /lib/libcom_err.so.2 (0x40026000)
        libuuid.so.1 => /lib/libuuid.so.1 (0x40028000)
        libc.so.6 => /lib/libc.so.6 (0x4002c000)
```

```
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Each file on the right–hand side is required. The file may be a symbolic link.

Note that some libraries are *quite large* and will not fit easily on your root filesystem. For example, the `libc.so` listed above is about 4 meg. You will probably need to strip libraries when copying them to your root filesystem. See Section 8.1 for instructions.

In `/lib` you must also include a loader for the libraries. The loader will be either `ld.so` (for A.OUT libraries, which are no longer common) or `ld-linux.so` (for ELF libraries). Newer versions of **ldd** tell you exactly which loader is needed, as in the example above, but older versions may not. If you're unsure which you need, run the **file** command on the library. For example:

```
% file /lib/libc.so.4.7.2 /lib/libc.so.5.4.33 /lib/libc-2.1.1.so
/lib/libc.so.4.7.2: Linux/i386 demand-paged executable (QMAGIC), stripped
/lib/libc.so.5.4.33: ELF 32-bit LSB shared object, Intel 80386, version 1, stripped
/lib/libc-2.1.1.so: ELF 32-bit LSB shared object, Intel 80386, version 1, not stripped
```

The `QMAGIC` indicates that `4.7.2` is for A.OUT libraries, and `ELF` indicates that `5.4.33` and `2.1.1` are for ELF.

Copy the specific loader(s) you need to the root filesystem you're building. Libraries and loaders should be checked *carefully* against the included binaries. If the kernel cannot load a necessary library, the kernel may hang with no error message.

4.4. Providing for PAM and NSS

Your system may require dynamically loaded libraries that are not visible to `ldd`. If you don't provide for these, you may have trouble logging in or using your bootdisk.

4.4.1. PAM (Pluggable Authentication Modules)

If your system uses PAM (Pluggable Authentication Modules), you must make some provision for it on your bootdisk. Briefly, PAM is a sophisticated modular method for authenticating users and controlling their access to services. An easy way to determine if your system uses PAM is run `ldd` on your `login` executable; if the output includes `libpam.so`, you need PAM.

Fortunately, security is usually of no concern with bootdisks since anyone who has physical access to a machine can usually do anything they want anyway. Therefore, you can effectively disable PAM by creating a simple `/etc/pam.conf` file in your root filesystem that looks like this:

```
OTHER   auth       optional     /lib/security/pam_permit.so
OTHER   account    optional     /lib/security/pam_permit.so
OTHER   password   optional     /lib/security/pam_permit.so
OTHER   session    optional     /lib/security/pam_permit.so
```

Also copy the file `/lib/security/pam_permit.so` to your root filesystem. This library is only about 8K so it imposes minimal overhead.

This configuration allows anyone complete access to the files and services on your machine. If you care about security on your bootdisk for some reason, you'll have to copy some or all of your hard disk's PAM setup to your root filesystem. Be sure to read the PAM documentation carefully, and copy any libraries needed in `/lib/security` onto your root filesystem.

You must also include `/lib/libpam.so` on your bootdisk. But you already know this since you ran ldd on `/bin/login`, which showed this dependency.

4.4.2. NSS (Name Service Switch)

If you are using glibc (aka libc6), you will have to make provisions for name services or you will not be able to login. The file `/etc/nsswitch.conf` controls database lookups for various servies. If you don't plan to access services from the network (eg, DNS or NIS lookups), you need only prepare a simple `nsswitch.conf` file that looks like this:

```
passwd:     files
shadow:     files
group:      files
hosts:      files
services:   files
networks:   files
protocols:  files
rpc:        files
ethers:     files
netmasks:   files
bootparams: files
automount:  files
aliases:    files
netgroup:   files
publickey:  files
```

This specifies that every service be provided only by local files. You will also need to include `/lib/libnss_files.so.`*X*, where *X* is 1 for glibc 2.0 and 2 for glibc 2.1. This library will be loaded dynamically to handle the file lookups.

If you plan to access the network from your bootdisk, you may want to create a more elaborate `nsswitch.conf` file. See the `nsswitch` man page for details. You must include a file `/lib/libnss_`*service*`.so.1` for each *service* you specify.

4.5. Modules

If you have a modular kernel, you must consider which modules you may want to load from your bootdisk after booting. You might want to include **ftape** and **zftape** modules if your backup tapes are on floppy tape, modules for SCSI devices if you have them, and possibly modules for PPP or SLIP support if you want to access the net in an emergency.

These modules may be placed in `/lib/modules`. You should also include **insmod**, **rmmod** and **lsmod**. Depending on whether you want to load modules automatically, you might also include **modprobe**, **depmod** and **swapout**. If you use **kerneld**, include it along with `/etc/conf.modules`.

However, the main advantage to using modules is that you can move non−critical modules to a utility disk and load them when needed, thus using less space on your root disk. If you may have to deal with many different devices, this approach is preferable to building one huge kernel with many drivers built in.

> **Important:** In order to boot a compressed ext2 filesystem, you must have ramdisk and ext2 support built−in. *They cannot be supplied as modules.*

4.6. Some final details

Some system programs, such as **login**, complain if the file `/var/run/utmp` and the directory `/var/log` do not exist. So:

```
mkdir -p /mnt/var/{log,run{
touch /mnt/var/run/utmp
```

Finally, after you have set up all the libraries you need, run **ldconfig** to remake `/etc/ld.so.cache` on the root filesystem. The cache tells the loader where to find the libraries. You can do this with:

```
ldconfig −r /mnt
```

4.7. Wrapping it up

When you have finished constructing the root filesystem, unmount it, copy it to a file and compress it:

```
umount /mnt
dd if=DEVICE bs=1k | gzip −v9 > rootfs.gz
```

When this finishes you will have a file `rootfs.gz`. This is your compressed root filesystem. You should check its size to make sure it will fit on a diskette; if it doesn't you'll have to go back and remove some files. Some suggestions for reducing root filesystem size appear in Section 8.1.

5. Choosing a kernel

At this point you have a complete compressed root filesystem. The next step is to build or select a kernel. In most cases it would be possible to copy your current kernel and boot the diskette from that. However, there may be cases where you wish to build a separate one.

One reason is size. If you are building a single boot/root diskette, the kernel will be one of the largest files on the diskette so you will have to reduce the size of the kernel as much as possible. To reduce kernel size, build it with the minumum set of facilities necessary to support the desired system. This means leaving out everything you don't need. Networking is a good thing to leave out, as well as support for any disk drives and other devices which you don't need when running your boot/root system. As stated before, your kernel *must* have ramdisk and ext2 support built into it.

Having worked out a minimum set of facilities to include in a kernel, you then need to work out what to add back in. Probably the most common uses for a boot/root diskette system would be to examine and restore a corrupted root file system, and to do this you may need kernel support. For example, if your backups are all held on tape using Ftape to access your tape drive, then, if you lose your current root drive and drives containing Ftape, then you will not be able to restore from your backup tapes. You will have to reinstall Linux, download and reinstall **ftape**, and then try to read your backups.

The point here is that, whatever I/O support you have added to your kernel to support backups should also be added into your boot/root kernel.

The procedure for actually building the kernel is described in the documentation that comes with the kernel. It is quite easy to follow, so start by looking in `/usr/src/linux`. If you have trouble building a kernel, you should probably not attempt to build boot/root systems anyway. Remember to compress the kernel with ``**make zImage**''.

6. Putting them together: Making the diskette(s)

At this point you have a kernel and a compressed root filesystem. If you are making a boot/root disk, check their sizes to make sure they will both fit on one disk. If you are making a two disk boot+root set, check the root filesystem to make sure it will fit on a single diskette.

You should decide whether to use LILO to boot the bootdisk kernel. The alternative is to copy the kernel directly to the diskette and boot without LILO. The advantage of using LILO is that it enables you to supply some parameters to the kernel which may be necessary to initialize your hardware (Check the file `/etc/lilo.conf` on your system. If it exists and has a line like ``append=...'', you probably need this feature). The disadvantage of using LILO is that building the bootdisk is more complicated and takes slightly more space. You will have to set up a small separate filesystem, which we shall call the *kernel filesystem*, where you transfer the kernel and a few other files that LILO needs.

If you are going to use LILO, read on; if you are going to transfer the kernel directly, skip ahead to Section 6.2.

6.1. Transferring the kernel with LILO

The first thing you must do is create a small configuration file for LILO. It should look like this:

```
        boot      =/dev/fd0
        install   =/boot/boot.b
        map       =/boot/map
        read-write
        backup    =/dev/null
        compact
        image     = KERNEL
        label     = Bootdisk
        root      =/dev/fd0
```

For an explanation of these parameters, see LILO's user documentation. You will probably also want to add an `append=...` line to this file from your hard disk's `/etc/lilo.conf` file.

Save this file as `bdlilo.conf`.

You now have to create a small filesystem, which we shall call a *kernel filesystem*, to distinguish it from the root filesystem.

First, figure out how large the filesystem should be. Take the size of your kernel in blocks (the size shown by ``**ls −s KERNEL**'') and add 50. Fifty blocks is approximately the space needed for inodes plus other files. You can calculate this number exactly if you want to, or just use 50. If you're creating a two−disk set, you may as well overestimate the space since the first disk is only used for the kernel anyway. Call this number `KERNEL_BLOCKS`.

Put a floppy diskette in the drive (for simplicity we'll assume `/dev/fd0`) and create an ext2 kernel filesystem on it:

```
        mke2fs −i 8192 −m 0 /dev/fd0 KERNEL_BLOCKS
```

The ``-i 8192'' specifies one inode per 8192 bytes. Next, mount the filesystem, remove the lost+found directory, and create dev and boot directories for LILO:

```
        mount /dev/fd0 /mnt
        rm -rf /mnt/lost+found
        mkdir /mnt/{boot,dev}
```

Next, create devices /dev/null and /dev/fd0. Instead of looking up the device numbers, you can just copy them from your hard disk using -R:

```
        cp -R /dev/{null,fd0} /mnt/dev
```

LILO needs a copy of its boot loader, boot.b, which you can take from your hard disk. It is usually kept in the /boot directory.

```
        cp /boot/boot.b /mnt/boot
```

Finally, copy in the LILO configuration file you created in the last section, along with your kernel. Both can be put in the root directory:

```
        cp bdlilo.conf KERNEL /mnt
```

Everything LILO needs is now on the kernel filesystem, so you are ready to run it. LILO's -r flag is used for installing the boot loader on some other root:

```
        lilo -v -C bdlilo.conf -r /mnt
```

LILO should run without error, after which the kernel filesystem should look something like this:

```
total 361
  1 -rw-r--r--   1 root     root          176 Jan 10 07:22 bdlilo.conf
  1 drwxr-xr-x   2 root     root         1024 Jan 10 07:23 boot/
  1 drwxr-xr-x   2 root     root         1024 Jan 10 07:22 dev/
358 -rw-r--r--   1 root     root       362707 Jan 10 07:23 vmlinuz
boot:
total 8
  4 -rw-r--r--   1 root     root         3708 Jan 10 07:22 boot.b
  4 -rw-------   1 root     root         3584 Jan 10 07:23 map
dev:
total 0
  0 brw-r-----   1 root     root       2,   0 Jan 10 07:22 fd0
  0 crw-r--r--   1 root     root       1,   3 Jan 10 07:22 null
```

Do not worry if the file sizes are slightly different from yours.

Now leave the diskette in the drive and go to Section 6.3.

6.2. Transferring the kernel without LILO

If you are *not* using LILO, transfer the kernel to the bootdisk with **dd**:

```
        % dd if=KERNEL of=/dev/fd0 bs=1k
```

```
        353+1 records in
        353+1 records out
```

In this example, **dd** wrote 353 complete records + 1 partial record, so the kernel occupies the first 354 blocks of the diskette. Call this number `KERNEL_BLOCKS` and remember it for use in the next section.

Finally, set the root device to be the diskette itself, then set the root to be loaded read/write:

```
        rdev /dev/fd0 /dev/fd0
        rdev -R /dev/fd0 0
```

Be careful to use a capital `-R` in the second **rdev** command.

6.3. Setting the ramdisk word

Inside the kernel image is the *ramdisk word* that specifies where the root filesystem is to be found, along with other options. The word can be accessed and set via the **rdev** command, and its contents are interpreted as follows:

Bit field	Description
0−10	Offset to start of ramdisk, in 1024 byte blocks
11−13	unused
14	Flag indicating that ramdisk is to be loaded
15	Flag indicating to prompt before loading rootfs

If bit 15 is set, on boot−up you will be prompted to place a new floppy diskette in the drive. This is necessary for a two−disk boot set.

There are two cases, depending on whether you are building a single boot/root diskette or a double ``boot+root'' diskette set.

1. If you are building a single disk, the compressed root filesystem will be placed right after the kernel, so the offset will be the first free block (which should be the same as `KERNEL_BLOCKS`). Bit 14 will be set to 1, and bit 15 will be zero. For example, say you're building a single disk and the root filesystem will begin at block 253 (decimal). The ramdisk word value should be 253 (decimal) with bit 14 set to 1 and bit 15 set to 0. To calculate the value you can simply add the decimal values. 253 + (2^14) = 253 + 16384 = 16637. If you don't quite understand where this number comes from, plug it into a scientific calculator and convert it to binary,
2. If you are building a two−disk set, the root filesystem will begin at block zero of the second disk, so the offset will be zero. Bit 14 will be set to 1 and bit 15 will be 1. The decimal value will be 2^14 + 2^15 = 49152 in this case.

After carefully calculating the value for the ramdisk word, set it with **rdev −r**. Be sure to use the *decimal* value. If you used LILO, the argument to **rdev** here should be the *mounted kernel path*, e.g. `/mnt/vmlinuz`; if you copied the kernel with **dd**, instead use the floppy device name (*e.g.,* `/dev/fd0`).

```
        rdev -r KERNEL_OR_FLOPPY_DRIVE  VALUE
```

If you used LILO, unmount the diskette now.

6.4. Transferring the root filesystem

The last step is to transfer the root filesystem.

- If the root filesystem will be placed on the *same* disk as the kernel, transfer it using **dd** with the `seek` option, which specifies how many blocks to skip:
  ```
          dd if=rootfs.gz of=/dev/fd0 bs=1k seek=KERNEL_BLOCKS
  ```
- If the root filesystem will be placed on a *second* disk, remove the first diskette, put the second diskette in the drive, then transfer the root filesystem to it:
  ```
          dd if=rootfs.gz of=/dev/fd0 bs=1k
  ```

Congratulations, you are done!

> **Important:** You should always test a bootdisk before putting it aside for an emergency. If it fails to boot, read on.

7. Troubleshooting, or The Agony of Defeat

When building bootdisks, the first few tries often will not boot. The general approach to building a root disk is to assemble components from your existing system, and try and get the diskette−based system to the point where it displays messages on the console. Once it starts talking to you, the battle is half over because you can see what it is complaining about, and you can fix individual problems until the system works smoothly. If the system just hangs with no explanation, finding the cause can be difficult. The recommended procedure for investigating the problem where the system will not talk to you is as follows:

- You may see a message like this:
  ```
  Kernel panic: VFS: Unable to mount root fs on XX:YY
  ```
  This is a common problem and it has only a few causes. First, check the device *XX:YY* against the list of device codes in `/usr/src/linux/Documentation/devices.txt`. If it is incorrect, you probably didn't do an **rdev −R**, or you did it on the wrong image. If the device code is correct, then check carefully the device drivers compiled into your kernel. Make sure it has floppy disk, ramdisk and ext2 filesystem support built−in.
- If you see many errors like:
  ```
  end_request: I/O error, dev 01:00 (ramdisk), sector NNN
  ```
  This is an I/O error from the ramdisk driver, usually because the kernel is trying to write beyond the end of the device. The ramdisk is too small to hold the root filesystem. Check your bootdisk kernel's initialization messages for a line like:
  ```
          Ramdisk driver initialized : 16 ramdisks of 4096K size
  ```
  Check this size against the *uncompressed* size of the root filesystem. If the ramdisks aren't large enough, make them larger.
- Check that the root disk actually contains the directories you think it does. It is easy to copy at the wrong level so that you end up with something like `/rootdisk/bin` instead of `/bin` on your root diskette.
- Check that there is a `/lib/libc.so` with the same link that appears in your `/lib` directory on your hard disk.
- Check that any symbolic links in your `/dev` directory in your existing system also exist on your root diskette filesystem, where those links are to devices which you have included in your root diskette. In particular, `/dev/console` links are essential in many cases.
- Check that you have included `/dev/tty1, /dev/null, /dev/zero, /dev/mem, /dev/ram` and `/dev/kmem` files.
- Check your kernel configuration −− support for all resources required up to login point must be built in, not modules. So *ramdisk and ext2 support must be built−in*.
- Check that your kernel root device and ramdisk settings are correct.

Once these general aspects have been covered, here are some more specific files to check:

1. Make sure **init** is included as `/sbin/init` or `/bin/init`. Make sure it is executable.
2. Run **ldd init** to check init's libraries. Usually this is just `libc.so`, but check anyway. Make sure you included the necessary libraries and loaders.
3. Make sure you have the right loader for your libraries −− `ld.so` for a.out or `ld−linux.so` for ELF.
4. Check the `/etc/inittab` on your bootdisk filesystem for the calls to **getty** (or some **getty**−like program, such as **agetty**, **mgetty** or **getty_ps**). Double−check these against your hard disk `inittab`. Check the man pages of the program you use to make sure these make sense.

`inittab` is possibly the trickiest part because its syntax and content depend on the init program used and the nature of the system. The only way to tackle it is to read the man pages for **init** and `inittab` and work out exactly what your existing system is doing when it boots. Check to make sure `/etc/inittab` has a system initialisation entry. This should contain a command to execute the system initialization script, which must exist.
5. As with **init**, run **ldd** on your **getty** to see what it needs, and make sure the necessary library files and loaders were included in your root filesystem.
6. Be sure you have included a shell program (e.g., **bash** or **ash**) capable of running all of your rc scripts.
7. If you have a `/etc/ld.so.cache` file on your rescue disk, remake it.

If **init** starts, but you get a message like:

```
        Id xxx respawning too fast: disabled for 5 minutes
```

it is coming from **init**, usually indicating that **getty** or **login** is dying as soon as it starts up. Check the **getty** and **login** executables and the libraries they depend upon. Make sure the invocations in `/etc/inittab` are correct. If you get strange messages from **getty**, it may mean the calling form in `/etc/inittab` is wrong.

If you get a login prompt, and you enter a valid login name but the system prompts you for another login name immediately, the problem may be with PAM or NSS. See Section 4.4. The problem may also be that you use shadow passwords and didn't copy `/etc/shadow` to your bootdisk.

If you try to run some executable, such as **df**, which is on your rescue disk but you yields a message like: `df: not found`, check two things: (1) Make sure the directory containing the binary is in your PATH, and (2) make sure you have libraries (and loaders) the program needs.

8. Miscellaneous topics

8.1. Reducing root filesystem size

Sometimes a root filesystem is too large to fit on a diskette even after compression. Here are some ways to reduce the filesystem size:

1. *Increase the diskette density*. By default, floppy diskettes are formatted at 1440K, but higher density formats are available. `fdformat` will format disks for the following sizes: 1600, 1680, 1722, 1743, 1760, 1840, and 1920. Most 1440K drives will support 1722K, and this is what I always use for bootdisks. See the **fdformat** man page and `/usr/src/linux/Documentation/devices.txt`.
2. *Replace your shell*. Some of the popular shells for Linux, such as **bash** and **tcsh**, are large and require many libraries. Light−weight alternatives exist, such as **ash**, **lsh**, **kiss** and **smash**, which are much smaller and require few (or no) libraries. Most of these replacement shells are available from `http://metalab.unc.edu/pub/Linux/system/shells/`. Make sure any shell you use is capable of running commands in all the `rc` files you include on your bootdisk.
3. *Strip libraries and binaries*. Many libraries and binaries are distributed with debugging information. Running **file** on these files will tell you ``not stripped'' if so. When copying binaries to your root filesystem, it is good practice to use:

   ```
           objcopy --strip-all FROM TO
   ```

   Important: When copying libraries, be sure to use `strip−debug` instead of `strip−all`.

- If you deleted or moved many files when you were creating the root filesystem, create it again. See the NOTE ABOVE on the importance of not having dirty blocks in the filesystem.
- *Move non−critical files to a utility disk*. If some of your binaries are not needed immediately to boot or login, you can move them to a utility disk. See Section 8.3 for details. You may also consider moving modules to a utility disk as well.

8.2. Non−ramdisk root filesystems

Section 4 gave instructions for building a compressed root filesystem which is loaded to ramdisk when the system boots. This method has many advantages so it is commonly used. However, some systems with little memory cannot afford the RAM needed for this, and they must use root filesystems mounted directly from the diskette.

Such filesystems are actually easier to build than compressed root filesystems because they can be built on a diskette rather than on some other device, and they do not have to be compressed. We will outline the procedure as it differs from the instructions above. If you choose to do this, keep in mind that you will have *much less space* available.

1. Calculate how much space you will have available for root files. If you are building a single boot/root disk, you must fit all blocks for the kernel plus all blocks for the root filesystem on the one disk.
2. Using **mke2fs**, create a root filesystem on a diskette of the appropriate size.
3. Populate the filesystem as described above.

4. When done, unmount the filesystem and transfer it to a disk file but *do not compress it*.
5. Transfer the kernel to a floppy diskette, as described above. When calculating the ramdisk word, *set bit 14 to zero*, to indicate that the root filesystem is not to be loaded to ramdisk. Run the **rdev**'s as described.
6. Transfer the root filesystem as before.

There are several shortcuts you can take. If you are building a two−disk set, you can build the complete root filesystem directly on the second disk and you need not transfer it to a hard disk file and then back. Also, if you are building a single boot/root disk and using LILO, you can build a *single* filesystem on the entire disk, containing the kernel, LILO files and root files, and simply run LILO as the last step.

8.3. Building a utility disk

Building a utility disk is relatively easy −− simply create a filesystem on a formatted disk and copy files to it. To use it with a bootdisk, mount it manually after the system is booted.

In the instructions above, we mentioned that the utility disk could be mounted as `/usr`. In this case, binaries could be placed into a `/bin` directory on your utility disk, so that placing `/usr/bin` in your path will access them. Additional libraries needed by the binaries are placed in `/lib` on the utility disk.

There are several important points to keep in mind when designing a utility disk:

1. Do not place critical system binaries or libraries onto the utility disk, since it will not be mountable until after the system has booted.
2. You cannot access a floppy diskette and a floppy tape drive simultaneously. This means that if you have a floppy tape drive, you will not be able to access it while your utility disk is mounted.
3. Access to files on the utility disk will be slow.

Appendix D shows a sample of files on a utility disk. Here are some ideas for files you may find useful: programs for examining and manipulating disks (**format, fdisk**) and filesystems (**mke2fs, fsck, debugfs, isofs.o**), a lightweight text editor (**elvis, jove**), compression and archive utilities (**gzip, bzip, tar, cpio, afio**), tape utilities (**mt, ftmt, tob, taper**), communications utilities (**ppp.o, slip.o, minicom**) and utilities for devices (**setserial, mknod**).

9. How the pros do it

You may notice that the bootdisks used by major distributions such as Slackware, RedHat or Debian seem more sophisticated than what is described in this document. Professional distribution bootdisks are based on the same principles outlined here, but employ various tricks because their bootdisks have additional requirements. First, they must be able to work with a wide variety of hardware, so they must be able to interact with the user and load various device drivers. Second, they must be prepared to work with many different installation options, with varying degrees of automation. Finally, distribution bootdisks usually combine installation and rescue capabilities.

Some bootdisks use a feature called *initrd* (*initial ramdisk*). This feature was introduced around 2.0.x and allows a kernel to boot in two phases. When the kernel first boots, it loads an initial ramdisk image from the boot disk. This initial ramdisk is a root filesystem containing a program that runs before the real root fs is loaded. This program usually inspects the environment and/or asks the user to select various boot options, such as the device from which to load the real rootdisk. It typically loads additional modules not built in to the kernel. When this initial program exits, the kernel loads the real root image and booting continues normally. For further information on **initrd**, see your local file `/usr/src/linux/Documentation/initrd.txt` and `ftp://elserv.ffm.fgan.de/pub/linux/loadlin-1.6/initrd-example.tgz`

The following are summaries of how each distribution's installation disks seem to work, based on inspecting their filesystems and/or source code. We do not guarantee that this information is completely accurate, or that they have not changed since the versions noted.

Slackware (v.3.1) uses a straightforward LILO boot similar to what is described in Section 6.1. The Slackware bootdisk prints a bootup message (`Welcome to the Slackware Linux bootkernel disk!`) using LILO's `message` parameter. This instructs the user to enter a boot parameter line if necessary. After booting, a root filesystem is loaded from a second disk. The user invokes a **setup** script which starts the installation. Instead of using a modular kernel, Slackware provides many different kernels and depends upon the user to select the one matching his or her hardware requirements.

RedHat (v.4.0) also uses a LILO boot. It loads a compressed ramdisk on the first disk, which runs a custom **init** program. This program queries for drivers then loads additional files from a supplemental disk if necessary.

Debian (v.1.3) is probably the most sophisticated of the installation disk sets. It uses the SYSLINUX loader to arrange various load options, then uses an `initrd` image to guide the user through installation. It appears to use both a customized **init** and a customized shell.

10. Creating bootable CD−ROMs

> **Note:** This section was contributed by Rizwan Mohammed Darwe (`rizwan AT clovertechnologies dot com`)

This section assumes that you are familiar with the process and workings of writing CDs in linux. Consider this to be a quick reference to include the ability to boot the CD which you will burn. The CD−Writing−HOWTO should give you an in−depth reference.

10.1. What is El Torito?

For the x86 platform, many BIOS's have begun to support bootable CDs. The patches for mkisofs is based on the standard called "El Torito". Simply put, El Torito is a specification that says how a cdrom should be formatted such that you can directly boot from it.

The "El Torito" spec says that *any* cdrom drive should work (SCSI or EIDE) as long as the BIOS supports El Torito. So far this has only been tested with EIDE drives because none of the SCSI controllers that has been tested so far appears to support El Torito. The motherboard definately has to support El Torito. How do you if your motherboard supports "El Torito"? Well, the ones that support lets you choose booting from HD, Floppy, Network or CDROM.

10.2. How it Works

The El Torito standard works by making the CD drive appear, through BIOS calls, to be a normal floppy drive. This way you simply put any floppy size image (exactly 1440k for a 1.44 meg floppy) somewhere in the ISO filesystem. In the headers of the ISO fs you place a pointer to this image. The BIOS will then grab this image from the CD and for all purposes it acts as if it were booting from the floppy drive. This allows a working LILO boot disk, for example, to simply be used as is.

Roughly speaking, the first 1.44 (or 2.88 if supported) Mbytes of the CD−ROM contains a floppy−disk image supplied by you. This image is treated like a floppy by the BIOS and booted from. (As a consequence, while booting from this virtual floppy, your original drive A: (`/dev/fd0`) may not be accessible, but you can try with `/dev/fd1`).

10.3. How to make it work

First create a file, say "boot.img", which is an exact image of the bootable floppy−disk which you want to boot via the CD−ROM. This must be an 1.44 MB bootable floppy−disk. The command below will do this

```
        dd if=/dev/fd0 of=boot.img bs=10k count=144
```

assuming the floppy is in the A: drive.

Place this image somewhere in the hierarchy which will be the source for the iso9660 filesystem. It is a good idea to put all boot related files in their own directory ("boot/" under the root of the iso9660 fs, for example).

One caveat −− Your boot floppy *must* load any initial ramdisk via LILO, not the kernel ramdisk driver! This is because once the linux kernel starts up, the BIOS emulation of the CD as a floppy disk is circumvented and will fail. LILO will load the initial ramdisk using BIOS disk calls, so the emulation works as designed.

The El Torito specification requires a "boot catalog" to be created as well. This is a 2048 byte file which is of no interest except it is required. The patchwork done by the author of mkisofs will cause it to automatically create the boot catalog, but you must specify where the boot catalog will go in the iso9660 filesystem. Usually it is a good idea to put it in the same place as the boot image, and a name like `boot.catalog` seems appropriate.

So we have our boot image in the file `boot.img`, and we are going to put it in the directory `boot/` under the root of the iso9660 filesystem. We will have the boot catalog go in the same directory with the name `boot.catalog`. The command to create the iso9660 fs in the file `bootcd.iso` is then:

```
mkisofs −r −b boot/boot.img −c boot/boot.catalog −o bootcd.iso .
```

The `−b` option specifies the boot image to be used (note the path is relative to the root of the iso9660 disk), and the `−c` option is for the boot catalog file. The `−r` option will make approptiate file ownerships and modes (see the `mkisofs` manpage). The "." in the end says to take the source from the current directory.

Now burn the CD with the usual cdrecord command and it is ready to boot.

10.4. Create Win9x Bootable CD−Roms

The first step is to get hold of the bootable image used by the source CD. But you cannot simply mount the CD under linux and dd the first 1440k to a floppy disk or to a file like `boot.img`. Instead you simply boot with the source CD−ROM.

When you boot the Win98 CD you are dropped to A: prompt which is the actual ramdisk. And D: or Z: is where all the installables are residing. By using the diskcopy command of dos copy the A: image into the actual floppy drive which is now B: The command below will do this.

```
diskcopy A: B:
```

It works just like dd. You can try booting from this newly created disk to test if the booting process is similar to that of the source CD. Then the usual dd of this floppy to a file like boot.img and then rest is as usual.

11. Frequently Asked Question (FAQ) list

Q: I boot from my boot/root disks and nothing happens. What do I do?
Q: How does the Slackware/Debian/RedHat bootdisk work?
Q: How can I make a boot disk with a XYZ driver?
Q: How do I update my root diskette with new files?
Q: How do I remove LILO so that I can use DOS to boot again?
Q: How can I boot if I've lost my kernel and my boot disk?
Q: How can I make extra copies of boot/root diskettes?
Q: How can I boot without typing in ahaxxxx=nn,nn,nn every time?
Q: At boot time, I get error `A: cannot execute B`*. Why?*
Q: My kernel has ramdisk support, but initializes ramdisks of 0K. Why?

Q: *I boot from my boot/root disks and nothing happens. What do I do?*

A: See Section 7, above.

Q: *How does the Slackware/Debian/RedHat bootdisk work?*

A: See Section 9, above.

Q: *How can I make a boot disk with a XYZ driver?*

A: The easiest way is to obtain a Slackware kernel from your nearest Slackware mirror site. Slackware kernels are generic kernels which atttempt to include drivers for as many devices as possible, so if you have a SCSI or IDE controller, chances are that a driver for it is included in the Slackware kernel.

Go to the `a1` directory and select either IDE or SCSI kernel depending on the type of controller you have. Check the xxxxkern.cfg file for the selected kernel to see the drivers which have been included in that kernel. If the device you want is in that list, then the corresponding kernel should boot your computer. Download the xxxxkern.tgz file and copy it to your boot diskette as described above in the section on making boot disks.

You must then check the root device in the kernel, using the command **rdev zImage**. If this is not the same as the root device you want, use **rdev** to change it. For example, the kernel I tried was set to `/dev/sda2`, but my root SCSI partition is `/dev/sda8`. To use a root diskette, you would have to use the command **rdev zImage /dev/fd0**.

If you want to know how to set up a Slackware root disk as well, that's outside the scope of this HOWTO, so I suggest you check the Linux Install Guide or get the Slackware distribution. See the section in this HOWTO titled ``References''.

Q: *How do I update my root diskette with new files?*

A: The easiest way is to copy the filesystem from the rootdisk back to the DEVICE you used (from Section 4.2, above). Then mount the filesystem and make the changes. You have to remember where your root filesystem started and how many blocks it occupied:

```
dd if=/dev/fd0 bs=1k skip=ROOTBEGIN count=BLOCKS | gunzip > DEVICE
mount -t ext2 DEVICE /mnt
```

After making the changes, proceed as before (in Section 4.7) and transfer the root filesystem back to the disk. You should not have to re−transfer the kernel or re−compute the ramdisk word if you do not change the starting position of the new root filesystem.

Q: *How do I remove LILO so that I can use DOS to boot again?*

A: This is not really a Bootdisk topic, but it is asked often. Within Linux, you can run:

```
        /sbin/lilo −u
```

You can also use the **dd** command to copy the backup saved by LILO to the boot sector. Refer to the LILO documentation if you wish to do this.

Within DOS and Windows you can use the DOS command:

```
        FDISK /MBR
```

MBR stands for Master Boot Record. This command replaces the boot sector with a clean DOS one, without affecting the partition table. Some purists disagree with this, but even the author of LILO, Werner Almesberger, suggests it. It is easy, and it works.

Q: *How can I boot if I've lost my kernel and my boot disk?*

A: If you don't have a boot disk standing by, probably the easiest method is to obtain a Slackware kernel for your disk controller type (IDE or SCSI) as described above for ``How do I make a boot disk with a XXX driver?''. You can then boot your computer using this kernel, then repair whatever damage there is.

The kernel you get may not have the root device set to the disk type and partition you want. For example, Slackware's generic SCSI kernel has the root device set to `/dev/sda2`, whereas my root Linux partition happens to be `/dev/sda8`. In this case the root device in the kernel will have to be changed.

You can still change the root device and ramdisk settings in the kernel even if all you have is a kernel, and some other operating system, such as DOS.

rdev changes kernel settings by changing the values at fixed offsets in the kernel file, so you can do the same if you have a hex editor available on whatever systems you do still have running −− for example, Norton Utilities Disk Editor under DOS. You then need to check and if necessary change the values in the kernel at the following offsets:

```
HEX     DEC  DESCRIPTION
0x01F8  504  Low byte of RAMDISK word
0x01F9  505  High byte of RAMDISK word
0x01FC  508  Root minor device number - see below
0X01FD  509  Root major device number - see below
```

The interpretation of the ramdisk word was described in Section 6.3, above.

The major and minor device numbers must be set to the device you want to mount your root filesystem on. Some useful values to select from are:

```
DEVICE          MAJOR MINOR
/dev/fd0            2     0   1st floppy drive
```

```
/dev/hda1           3     1   partition 1 on 1st IDE drive
/dev/sda1           8     1   partition 1 on 1st SCSI drive
/dev/sda8           8     8   partition 8 on 1st SCSI drive
```

Once you have set these values then you can write the file to a diskette using either Norton Utilities Disk Editor, or a program called **rawrite.exe**. This program is included in all distributions. It is a DOS program which writes a file to the ``raw'' disk, starting at the boot sector, instead of writing it to the file system. If you use Norton Utilities you must write the file to a physical disk starting at the beginning of the disk.

Q: *How can I make extra copies of boot/root diskettes?*

A: Because magnetic media may deteriorate over time, you should keep several copies of your rescue disk, in case the original is unreadable.

The easiest way of making copies of any diskettes, including bootable and utility diskettes, is to use the **dd** command to copy the contents of the original diskette to a file on your hard drive, and then use the same command to copy the file back to a new diskette. Note that you do not need to, and should not, mount the diskettes, because **dd** uses the raw device interface.

To copy the original, enter the command:

```
        dd if=DEVICENAME of=FILENAME
```

where DEVICENAME is the device name of the diskette drive and FILENAME is the name of the (hard−disk) output file. Omitting the `count` parameter causes **dd** to copy the whole diskette (2880 blocks if high−density).

To copy the resulting file back to a new diskette, insert the new diskette and enter the reverse command:

```
        dd if=FILENAME of=DEVICENAME
```

Note that the above discussion assumes that you have only one diskette drive. If you have two of the same type, you can copy diskettes using a command like:

```
        dd if=/dev/fd0 of=/dev/fd1
```

Q: *How can I boot without typing in ahaxxxx=nn,nn,nn every time?*

A: Where a disk device cannot be autodetected it is necessary to supply the kernel with a command device parameter string, such as:

```
        aha152x=0x340,11,3,1
```

This parameter string can be supplied in several ways using LILO:

- By entering it on the command line every time the system is booted via LILO. This is boring, though.
- By using LILO's `lock` keyword to make it store the command line as the default command line, so that LILO will use the same options every time it boots.
- By using the `append=` statement in the LILO config file. Note that the parameter string must be enclosed in quotes.

For example, a sample command line using the above parameter string would be:

```
        zImage  aha152x=0x340,11,3,1 root=/dev/sda1 lock
```

This would pass the device parameter string through, and also ask the kernel to set the root device to `/dev/sda1` and save the whole command line and reuse it for all future boots.

A sample APPEND statement is:

```
        APPEND = 8220;aha152x=0x340,11,3,18221;
```

Note that the parameter string must *not* be enclosed in quotes on the command line, but it *must* be enclosed in quotes in the `APPEND` statement.

Note also that for the parameter string to be acted on, the kernel must contain the driver for that disk type. If it does not, then there is nothing listening for the parameter string, and you will have to rebuild the kernel to include the required driver. For details on rebuilding the kernel, go to `/usr/src/linux` and read the README, and read the Linux FAQ and Installation HOWTO. Alternatively you could obtain a generic kernel for the disk type and install that.

Readers are strongly urged to read the LILO documentation before experimenting with LILO installation. Incautious use of the `BOOT` statement can damage partitions.

Q: *At boot time, I get error* `A: cannot execute B`*. Why?*

A: There are several cases of program names being hardcoded in various utilities. These cases do not occur everywhere, but they may explain why an executable apparently cannot be found on your system even though you can see that it is there. You can find out if a given program has the name of another hardcoded by using the **strings** command and piping the output through **grep**.

Known examples of hardcoding are:

- **shutdown** in some versions has `/etc/reboot` hardcoded, so **reboot** must be placed in the `/etc` directory.
- **init** has caused problems for at least one person, with the kernel being unable to find **init**.

To fix these problems, either move the programs to the correct directory, or change configuration files (e.g. `inittab`) to point to the correct directory. If in doubt, put programs in the same directories as they are on your hard disk, and use the same `inittab` and `/etc/rc.d` files as they appear on your hard disk.

Q: *My kernel has ramdisk support, but initializes ramdisks of 0K. Why?*

A: Where this occurs, a kernel message like this will appear as the kernel is booting:

```
        Ramdisk driver initialized : 16 ramdisks of 0K size
```

This is probably because the size has been set to 0 by kernel parameters at boot time. This could possibly be because of an overlooked LILO configuration file parameter:

```
    ramdisk= 0
```

This was included in sample LILO configuration files in some older distributions, and was put there to override any previous kernel setting. If you have such a line, remove it.

Note that if you attempt to use a ramdisk of 0 size, the behaviour can be unpredictable, and can result in kernel panics.

A. Resources and pointers

When retrieving a package, always get the latest version unless you have good reasons for not doing so.

A.1. Pre−made Bootdisks

These are sources for distribution bootdisks. *Please use one of the mirror sites to reduce the load on these machines.*

- Slackware bootdisks, rootdisks and Slackware mirror sites
- RedHat bootdisks and Red Hat mirror sites
- Debian bootdisks and Debian mirror sites

In addition to the distribution bootdisks, the following rescue disk images are available. Unless otherwise specified, these are available in the directory `http://metalab.unc.edu/pub/Linux/system/recovery/!INDEX.html`

- `tomsrtbt`, by Tom Oehser, is a single−disk boot/root disk based on kernel 2.0, with a large set of features and support programs. It supports IDE, SCSI, tape, network adaptors, PCMCIA and more. About 100 utility programs and tools are included for fixing and restoring disks. The package also includes scripts for disassembling and reconstructing the images so that new material can be added if necessary.
- `rescue02`, by John Comyns, is a rescue disk based on kernel 1.3.84, with support for IDE and Adaptec 1542 and NCR53C7,8xx. It uses ELF binaries but it has enough commands so that it can be used on any system. There are modules that can be loaded after booting for all other SCSI cards. It probably won't work on systems with 4 mb of ram since it uses a 3 mb ram disk.
- `resque_disk-2.0.22`, by Sergei Viznyuk, is a full−featured boot/root disk based on kernel 2.0.22 with built−in support for IDE, many difference SCSI controllers, and ELF/AOUT. Also includes many modules and useful utilities for repairing and restoring a hard disk.
- `cramdisk` images, based on the 2.0.23 kernel, available for 4 meg and 8 meg machines. They include math emulation and networking (PPP and dialin script, NE2000, 3C509), or support for the parallel port ZIP drive. These diskette images will boot on a 386 with 4MB RAM. MSDOS support is included so you can download from the net to a DOS partition.

A.2. Rescue packages

Several packages for creating rescue disks are available on metalab.unc.edu. With these packages you specify a set of files for inclusion and the software automates (to varying degrees) the creation of a bootdisk. See `http://metalab.unc.edu/pub/Linux/system/recovery/!INDEX.html` for more information. *Check the file dates carefully*. Some of these packages have not been updated in several years and will not support the creation of a compressed root filesystem loaded into ramdisk. To the best of our knowledge, Yard is the only package that will.

A.3. LILO −− the Linux loader

Written by Werner Almesberger. Excellent boot loader, and the documentation includes information on the boot sector contents and the early stages of the boot process.

Ftp from ftp://tsx−11.mit.edu/pub/linux/packages/lilo/. It is also available on Metalab and mirrors.

A.4. Ramdisk usage

An excellent description of the how the ramdisk code works may be found with the documentation supplied with the Linux kernel. See `/usr/src/linux/Documentation/ramdisk.txt`. It is written by Paul Gortmaker and includes a section on creating a compressed ramdisk.

A.5. The Linux boot process

For more detail on the Linux boot process, here are some pointers:

- The *Linux System Administrators' Guide* has a section on booting.
- The LILO ``Technical overview'' has the definitive technical, low−level description of the boot process, up to where the kernel is started.
- The source code is the ultimate guide. Below are some kernel files related to the boot process. If you have the Linux kernel source code, you can find these under `/usr/src/linux` on your machine; alternatively, Shigio Yamaguchi (shigio at tamacom.com) has a very nice hypertext kernel browser for reading kernel source files. Here are some relevant files to look at:

`arch/i386/boot/bootsect.S` *and* *`setup.S`*

Contain assembly code for the bootsector itself.

`arch/i386/boot/compressed/misc.c`

Contains code for uncompressing the kernel.

`arch/i386/kernel/`

Directory containing kernel initialization code. `setup.c` defines the ramdisk word.

`drivers/block/rd.c`

Contains the ramdisk driver. The procedures **rd_load** and **rd_load_image** load blocks from a device into a ramdisk. The procedure **identify_ramdisk_image** determines what kind of filesystem is found and whether it is compressed.

B. LILO boot error codes

Questions about these codes are asked so often on Usenet that we include them here as a public service. This summary is excerpted from Werner Almsberger's LILO User Documentation.

When LILO loads itself, it displays the word `LILO`. Each letter is printed before or after performing some specific action. If LILO fails at some point, the letters printed so far can be used to identify the problem.

Output	**Problem**
(nothing)	No part of LILO has been loaded. LILO either isn't installed or the partition on which its boot sector is located isn't active.
L	The first stage boot loader has been loaded and started, but it can't load the second stage boot loader. The two−digit error codes indicate the type of problem. (See also section ``Disk error codes''.) This condition usually indicates a media failure or a geometry mismatch (e.g. bad disk parameters).
LI	The first stage boot loader was able to load the second stage boot loader, but has failed to execute it. This can either be caused by a geometry mismatch or by moving `/boot/boot.b` without running the map installer.
LIL	The second stage boot loader has been started, but it can't load the descriptor table from the map file. This is typically caused by a media failure or by a geometry mismatch.
LIL?	The second stage boot loader has been loaded at an incorrect address. This is typically caused by a subtle geometry mismatch or by moving `/boot/boot.b` without running the map installer.
LIL−	The descriptor table is corrupt. This can either be caused by a geometry mismatch or by moving /boot/map without running the map installer.
LILO	All parts of LILO have been successfully loaded.

If the BIOS signals an error when LILO is trying to load a boot image, the respective error code is displayed. These codes range from `0x00` through `0xbb`. See the LILO User Guide for an explanation of these.

C. Sample root filesystem listings

```
/:
drwx--x--x   2 root     root         1024 Nov  1 15:39 bin
drwx--x--x   2 root     root         4096 Nov  1 15:39 dev
drwx--x--x   3 root     root         1024 Nov  1 15:39 etc
drwx--x--x   4 root     root         1024 Nov  1 15:39 lib
drwx--x--x   5 root     root         1024 Nov  1 15:39 mnt
drwx--x--x   2 root     root         1024 Nov  1 15:39 proc
drwx--x--x   2 root     root         1024 Nov  1 15:39 root
drwx--x--x   2 root     root         1024 Nov  1 15:39 sbin
drwx--x--x   2 root     root         1024 Nov  1 15:39 tmp
drwx--x--x   7 root     root         1024 Nov  1 15:39 usr
drwx--x--x   5 root     root         1024 Nov  1 15:39 var

/bin:
-rwx--x--x   1 root     root        62660 Nov  1 15:39 ash
-rwx--x--x   1 root     root         9032 Nov  1 15:39 cat
-rwx--x--x   1 root     root        10276 Nov  1 15:39 chmod
-rwx--x--x   1 root     root         9592 Nov  1 15:39 chown
-rwx--x--x   1 root     root        23124 Nov  1 15:39 cp
-rwx--x--x   1 root     root        23028 Nov  1 15:39 date
-rwx--x--x   1 root     root        14052 Nov  1 15:39 dd
-rwx--x--x   1 root     root        14144 Nov  1 15:39 df
-rwx--x--x   1 root     root        69444 Nov  1 15:39 egrep
-rwx--x--x   1 root     root          395 Nov  1 15:39 false
-rwx--x--x   1 root     root        69444 Nov  1 15:39 fgrep
-rwx--x--x   1 root     root        69444 Nov  1 15:39 grep
-rwx--x--x   3 root     root        45436 Nov  1 15:39 gunzip
-rwx--x--x   3 root     root        45436 Nov  1 15:39 gzip
-rwx--x--x   1 root     root         8008 Nov  1 15:39 hostname
-rwx--x--x   1 root     root        12736 Nov  1 15:39 ln
-rws--x--x   1 root     root        15284 Nov  1 15:39 login
-rwx--x--x   1 root     root        29308 Nov  1 15:39 ls
-rwx--x--x   1 root     root         8268 Nov  1 15:39 mkdir
-rwx--x--x   1 root     root         8920 Nov  1 15:39 mknod
-rwx--x--x   1 root     root        24836 Nov  1 15:39 more
-rws--x--x   1 root     root        37640 Nov  1 15:39 mount
-rwx--x--x   1 root     root        12240 Nov  1 15:39 mt
-rwx--x--x   1 root     root        12932 Nov  1 15:39 mv
-r-x--x--x   1 root     root        12324 Nov  1 15:39 ps
-rwx--x--x   1 root     root         5388 Nov  1 15:39 pwd
-rwx--x--x   1 root     root        10092 Nov  1 15:39 rm
lrwxrwxrwx   1 root     root            3 Nov  1 15:39 sh -> ash
-rwx--x--x   1 root     root        25296 Nov  1 15:39 stty
-rws--x--x   1 root     root        12648 Nov  1 15:39 su
-rwx--x--x   1 root     root         4444 Nov  1 15:39 sync
-rwx--x--x   1 root     root       110668 Nov  1 15:39 tar
-rwx--x--x   1 root     root        19712 Nov  1 15:39 touch
-rwx--x--x   1 root     root          395 Nov  1 15:39 true
-rws--x--x   1 root     root        19084 Nov  1 15:39 umount
-rwx--x--x   1 root     root         5368 Nov  1 15:39 uname
-rwx--x--x   3 root     root        45436 Nov  1 15:39 zcat

/dev:
lrwxrwxrwx   1 root     root            6 Nov  1 15:39 cdrom -> cdu31a
brw-rw-r--   1 root     root      15,   0 May  5  1998 cdu31a
crw-------   1 root     root       4,   0 Nov  1 15:29 console
crw-rw-rw-   1 root     uucp       5,  64 Sep  9 19:46 cua0
crw-rw-rw-   1 root     uucp       5,  65 May  5  1998 cua1
```

```
crw-rw-rw-   1 root     uucp       5,  66 May  5  1998 cua2
crw-rw-rw-   1 root     uucp       5,  67 May  5  1998 cua3
brw-rw----   1 root     floppy     2,   0 Aug  8 13:54 fd0
brw-rw----   1 root     floppy     2,  36 Aug  8 13:54 fd0CompaQ
brw-rw----   1 root     floppy     2,  84 Aug  8 13:55 fd0D1040
brw-rw----   1 root     floppy     2,  88 Aug  8 13:55 fd0D1120
brw-rw----   1 root     floppy     2,  12 Aug  8 13:54 fd0D360
brw-rw----   1 root     floppy     2,  16 Aug  8 13:54 fd0D720
brw-rw----   1 root     floppy     2, 120 Aug  8 13:55 fd0D800
brw-rw----   1 root     floppy     2,  32 Aug  8 13:54 fd0E2880
brw-rw----   1 root     floppy     2, 104 Aug  8 13:55 fd0E3200
brw-rw----   1 root     floppy     2, 108 Aug  8 13:55 fd0E3520
brw-rw----   1 root     floppy     2, 112 Aug  8 13:55 fd0E3840
brw-rw----   1 root     floppy     2,  28 Aug  8 13:54 fd0H1440
brw-rw----   1 root     floppy     2, 124 Aug  8 13:55 fd0H1600
brw-rw----   1 root     floppy     2,  44 Aug  8 13:55 fd0H1680
brw-rw----   1 root     floppy     2,  60 Aug  8 13:55 fd0H1722
brw-rw----   1 root     floppy     2,  76 Aug  8 13:55 fd0H1743
brw-rw----   1 root     floppy     2,  96 Aug  8 13:55 fd0H1760
brw-rw----   1 root     floppy     2, 116 Aug  8 13:55 fd0H1840
brw-rw----   1 root     floppy     2, 100 Aug  8 13:55 fd0H1920
lrwxrwxrwx   1 root     root            7 Nov  1 15:39 fd0H360 -> fd0D360
lrwxrwxrwx   1 root     root            7 Nov  1 15:39 fd0H720 -> fd0D720
brw-rw----   1 root     floppy     2,  52 Aug  8 13:55 fd0H820
brw-rw----   1 root     floppy     2,  68 Aug  8 13:55 fd0H830
brw-rw----   1 root     floppy     2,   4 Aug  8 13:54 fd0d360
brw-rw----   1 root     floppy     2,   8 Aug  8 13:54 fd0h1200
brw-rw----   1 root     floppy     2,  40 Aug  8 13:54 fd0h1440
brw-rw----   1 root     floppy     2,  56 Aug  8 13:55 fd0h1476
brw-rw----   1 root     floppy     2,  72 Aug  8 13:55 fd0h1494
brw-rw----   1 root     floppy     2,  92 Aug  8 13:55 fd0h1600
brw-rw----   1 root     floppy     2,  20 Aug  8 13:54 fd0h360
brw-rw----   1 root     floppy     2,  48 Aug  8 13:55 fd0h410
brw-rw----   1 root     floppy     2,  64 Aug  8 13:55 fd0h420
brw-rw----   1 root     floppy     2,  24 Aug  8 13:54 fd0h720
brw-rw----   1 root     floppy     2,  80 Aug  8 13:55 fd0h880
brw-rw----   1 root     disk       3,   0 May  5  1998 hda
brw-rw----   1 root     disk       3,   1 May  5  1998 hda1
brw-rw----   1 root     disk       3,   2 May  5  1998 hda2
brw-rw----   1 root     disk       3,   3 May  5  1998 hda3
brw-rw----   1 root     disk       3,   4 May  5  1998 hda4
brw-rw----   1 root     disk       3,   5 May  5  1998 hda5
brw-rw----   1 root     disk       3,   6 May  5  1998 hda6
brw-rw----   1 root     disk       3,  64 May  5  1998 hdb
brw-rw----   1 root     disk       3,  65 May  5  1998 hdb1
brw-rw----   1 root     disk       3,  66 May  5  1998 hdb2
brw-rw----   1 root     disk       3,  67 May  5  1998 hdb3
brw-rw----   1 root     disk       3,  68 May  5  1998 hdb4
brw-rw----   1 root     disk       3,  69 May  5  1998 hdb5
brw-rw----   1 root     disk       3,  70 May  5  1998 hdb6
crw-r-----   1 root     kmem       1,   2 May  5  1998 kmem
crw-r-----   1 root     kmem       1,   1 May  5  1998 mem
lrwxrwxrwx   1 root     root           12 Nov  1 15:39 modem -> ttyS1
lrwxrwxrwx   1 root     root           12 Nov  1 15:39 mouse -> psaux
crw-rw-rw-   1 root     root       1,   3 May  5  1998 null
crwxrwxrwx   1 root     root      10,   1 Oct  5 20:22 psaux
brw-r-----   1 root     disk       1,   1 May  5  1998 ram
brw-rw----   1 root     disk       1,   0 May  5  1998 ram0
brw-rw----   1 root     disk       1,   1 May  5  1998 ram1
brw-rw----   1 root     disk       1,   2 May  5  1998 ram2
brw-rw----   1 root     disk       1,   3 May  5  1998 ram3
brw-rw----   1 root     disk       1,   4 May  5  1998 ram4
```

```
brw-rw----   1 root     disk       1,   5 May  5  1998 ram5
brw-rw----   1 root     disk       1,   6 May  5  1998 ram6
brw-rw----   1 root     disk       1,   7 May  5  1998 ram7
brw-rw----   1 root     disk       1,   8 May  5  1998 ram8
brw-rw----   1 root     disk       1,   9 May  5  1998 ram9
lrwxrwxrwx   1 root     root            4 Nov  1 15:39 ramdisk -> ram0
***  I have only included devices for the IDE partitions I use.
***  If you use SCSI, then use the /dev/sdXX devices instead.
crw-------   1 root     root       4,   0 May  5  1998 tty0
crw-w-----   1 root     tty        4,   1 Nov  1 15:39 tty1
crw-------   1 root     root       4,   2 Nov  1 15:29 tty2
crw-------   1 root     root       4,   3 Nov  1 15:29 tty3
crw-------   1 root     root       4,   4 Nov  1 15:29 tty4
crw-------   1 root     root       4,   5 Nov  1 15:29 tty5
crw-------   1 root     root       4,   6 Nov  1 15:29 tty6
crw-------   1 root     root       4,   7 May  5  1998 tty7
crw-------   1 root     tty        4,   8 May  5  1998 tty8
crw-------   1 root     tty        4,   9 May  8 12:57 tty9
crw-rw-rw-   1 root     root       4,  65 Nov  1 12:17 ttyS1
crw-rw-rw-   1 root     root       1,   5 May  5  1998 zero

/etc:
-rw-------   1 root     root          164 Nov  1 15:39 conf.modules
-rw-------   1 root     root          668 Nov  1 15:39 fstab
-rw-------   1 root     root           71 Nov  1 15:39 gettydefs
-rw-------   1 root     root          389 Nov  1 15:39 group
-rw-------   1 root     root          413 Nov  1 15:39 inittab
-rw-------   1 root     root           65 Nov  1 15:39 issue
-rw-r--r--   1 root     root          746 Nov  1 15:39 ld.so.cache
-rw-------   1 root     root           32 Nov  1 15:39 motd
-rw-------   1 root     root          949 Nov  1 15:39 nsswitch.conf
drwx--x--x   2 root     root         1024 Nov  1 15:39 pam.d
-rw-------   1 root     root          139 Nov  1 15:39 passwd
-rw-------   1 root     root          516 Nov  1 15:39 profile
-rwx--x--x   1 root     root          387 Nov  1 15:39 rc
-rw-------   1 root     root           55 Nov  1 15:39 shells
-rw-------   1 root     root          774 Nov  1 15:39 termcap
-rw-------   1 root     root           78 Nov  1 15:39 ttytype
lrwxrwxrwx   1 root     root           15 Nov  1 15:39 utmp -> ../var/run/utmp
lrwxrwxrwx   1 root     root           15 Nov  1 15:39 wtmp -> ../var/log/wtmp

/etc/pam.d:
-rw-------   1 root     root          356 Nov  1 15:39 other

/lib:
-rwxr-xr-x   1 root     root        45415 Nov  1 15:39 ld-2.0.7.so
lrwxrwxrwx   1 root     root           11 Nov  1 15:39 ld-linux.so.2 -> ld-2.0.7.so
-rwxr-xr-x   1 root     root       731548 Nov  1 15:39 libc-2.0.7.so
lrwxrwxrwx   1 root     root           13 Nov  1 15:39 libc.so.6 -> libc-2.0.7.so
lrwxrwxrwx   1 root     root           17 Nov  1 15:39 libcom_err.so.2 -> libcom_err.so.2.0
-rwxr-xr-x   1 root     root         6209 Nov  1 15:39 libcom_err.so.2.0
-rwxr-xr-x   1 root     root       153881 Nov  1 15:39 libcrypt-2.0.7.so
lrwxrwxrwx   1 root     root           17 Nov  1 15:39 libcrypt.so.1 -> libcrypt-2.0.7.so
-rwxr-xr-x   1 root     root        12962 Nov  1 15:39 libdl-2.0.7.so
lrwxrwxrwx   1 root     root           14 Nov  1 15:39 libdl.so.2 -> libdl-2.0.7.so
lrwxrwxrwx   1 root     root           16 Nov  1 15:39 libext2fs.so.2 -> libext2fs.so.2.4
-rwxr-xr-x   1 root     root        81382 Nov  1 15:39 libext2fs.so.2.4
-rwxr-xr-x   1 root     root        25222 Nov  1 15:39 libnsl-2.0.7.so
lrwxrwxrwx   1 root     root           15 Nov  1 15:39 libnsl.so.1 -> libnsl-2.0.7.so
-rwx--x--x   1 root     root       178336 Nov  1 15:39 libnss_files-2.0.7.so
lrwxrwxrwx   1 root     root           21 Nov  1 15:39 libnss_files.so.1 -> libnss_files-2.0.7.so
lrwxrwxrwx   1 root     root           14 Nov  1 15:39 libpam.so.0 -> libpam.so.0.64
```

```
-rwxr-xr-x   1 root     root        26906 Nov  1 15:39 libpam.so.0.64
lrwxrwxrwx   1 root     root           19 Nov  1 15:39 libpam_misc.so.0 -> libpam_misc.so.0.64
-rwxr-xr-x   1 root     root         7086 Nov  1 15:39 libpam_misc.so.0.64
-r-xr-xr-x   1 root     root        35615 Nov  1 15:39 libproc.so.1.2.6
lrwxrwxrwx   1 root     root           15 Nov  1 15:39 libpwdb.so.0 -> libpwdb.so.0.54
-rw-r-r---   1 root     root       121899 Nov  1 15:39 libpwdb.so.0.54
lrwxrwxrwx   1 root     root           19 Nov  1 15:39 libtermcap.so.2 -> libtermcap.so.2.0.8
-rwxr-xr-x   1 root     root        12041 Nov  1 15:39 libtermcap.so.2.0.8
-rwxr-xr-x   1 root     root        12874 Nov  1 15:39 libutil-2.0.7.so
lrwxrwxrwx   1 root     root           16 Nov  1 15:39 libutil.so.1 -> libutil-2.0.7.so
lrwxrwxrwx   1 root     root           14 Nov  1 15:39 libuuid.so.1 -> libuuid.so.1.1
-rwxr-xr-x   1 root     root         8039 Nov  1 15:39 libuuid.so.1.1
drwx--x--x   3 root     root         1024 Nov  1 15:39 modules
drwx--x--x   2 root     root         1024 Nov  1 15:39 security

/lib/modules:
drwx--x--x   4 root     root         1024 Nov  1 15:39 2.0.35

/lib/modules/2.0.35:
drwx--x--x   2 root     root         1024 Nov  1 15:39 block
drwx--x--x   2 root     root         1024 Nov  1 15:39 cdrom

/lib/modules/2.0.35/block:
drwx------   1 root     root         7156 Nov  1 15:39 loop.o

/lib/modules/2.0.35/cdrom:
drwx------   1 root     root        24108 Nov  1 15:39 cdu31a.o

/lib/security:
-rwx--x--x   1 root     root         8771 Nov  1 15:39 pam_permit.so

***  Directory stubs for mounting
/mnt:
drwx--x--x   2 root     root         1024 Nov  1 15:39 cdrom
drwx--x--x   2 root     root         1024 Nov  1 15:39 floppy

/proc:

/root:
-rw-------   1 root     root          176 Nov  1 15:39 .bashrc
-rw-------   1 root     root          182 Nov  1 15:39 .cshrc
-rwx--x--x   1 root     root          455 Nov  1 15:39 .profile
-rw-------   1 root     root         4014 Nov  1 15:39 .tcshrc

/sbin:
-rwx--x--x   1 root     root        23976 Nov  1 15:39 depmod
-rwx--x--x   2 root     root       274600 Nov  1 15:39 e2fsck
-rwx--x--x   1 root     root        41268 Nov  1 15:39 fdisk
-rwx--x--x   1 root     root         9396 Nov  1 15:39 fsck
-rwx--x--x   2 root     root       274600 Nov  1 15:39 fsck.ext2
-rwx--x--x   1 root     root        29556 Nov  1 15:39 getty
-rwx--x--x   1 root     root         6620 Nov  1 15:39 halt
-rwx--x--x   1 root     root        23116 Nov  1 15:39 init
-rwx--x--x   1 root     root        25612 Nov  1 15:39 insmod
-rwx--x--x   1 root     root        10368 Nov  1 15:39 kerneld
-rwx--x--x   1 root     root       110400 Nov  1 15:39 ldconfig
-rwx--x--x   1 root     root         6108 Nov  1 15:39 lsmod
-rwx--x--x   2 root     root        17400 Nov  1 15:39 mke2fs
-rwx--x--x   1 root     root         4072 Nov  1 15:39 mkfs
-rwx--x--x   2 root     root        17400 Nov  1 15:39 mkfs.ext2
-rwx--x--x   1 root     root         5664 Nov  1 15:39 mkswap
-rwx--x--x   1 root     root        22032 Nov  1 15:39 modprobe
```

```
lrwxrwxrwx   1 root     root            4 Nov  1 15:39 reboot -> halt
-rwx--x--x   1 root     root         7492 Nov  1 15:39 rmmod
-rwx--x--x   1 root     root        12932 Nov  1 15:39 shutdown
lrwxrwxrwx   1 root     root            6 Nov  1 15:39 swapoff -> swapon
-rwx--x--x   1 root     root         5124 Nov  1 15:39 swapon
lrwxrwxrwx   1 root     root            4 Nov  1 15:39 telinit -> init
-rwx--x--x   1 root     root         6944 Nov  1 15:39 update

/tmp:

/usr:
drwx--x--x   2 root     root         1024 Nov  1 15:39 bin
drwx--x--x   2 root     root         1024 Nov  1 15:39 lib
drwx--x--x   3 root     root         1024 Nov  1 15:39 man
drwx--x--x   2 root     root         1024 Nov  1 15:39 sbin
drwx--x--x   3 root     root         1024 Nov  1 15:39 share
lrwxrwxrwx   1 root     root           10 Nov  1 15:39 tmp -> ../var/tmp

/usr/bin:
-rwx--x--x   1 root     root        37164 Nov  1 15:39 afio
-rwx--x--x   1 root     root         5044 Nov  1 15:39 chroot
-rwx--x--x   1 root     root        10656 Nov  1 15:39 cut
-rwx--x--x   1 root     root        63652 Nov  1 15:39 diff
-rwx--x--x   1 root     root        12972 Nov  1 15:39 du
-rwx--x--x   1 root     root        56552 Nov  1 15:39 find
-r-x--x--x   1 root     root         6280 Nov  1 15:39 free
-rwx--x--x   1 root     root         7680 Nov  1 15:39 head
-rwx--x--x   1 root     root         8504 Nov  1 15:39 id
-r-sr-xr-x   1 root     bin          4200 Nov  1 15:39 passwd
-rwx--x--x   1 root     root        14856 Nov  1 15:39 tail
-rwx--x--x   1 root     root        19008 Nov  1 15:39 tr
-rwx--x--x   1 root     root         7160 Nov  1 15:39 wc
-rwx--x--x   1 root     root         4412 Nov  1 15:39 whoami

/usr/lib:
lrwxrwxrwx   1 root     root           17 Nov  1 15:39 libncurses.so.4 -> libncurses.so.4.2
-rw-r-r---   1 root     root       260474 Nov  1 15:39 libncurses.so.4.2

/usr/sbin:
-r-x--x--x   1 root     root        13684 Nov  1 15:39 fuser
-rwx--x--x   1 root     root         3876 Nov  1 15:39 mklost+found

/usr/share:
drwx--x--x   4 root     root         1024 Nov  1 15:39 terminfo

/usr/share/terminfo:
drwx--x--x   2 root     root         1024 Nov  1 15:39 l
drwx--x--x   2 root     root         1024 Nov  1 15:39 v

/usr/share/terminfo/l:
-rw-------   1 root     root         1552 Nov  1 15:39 linux
-rw-------   1 root     root         1516 Nov  1 15:39 linux-m
-rw-------   1 root     root         1583 Nov  1 15:39 linux-nic

/usr/share/terminfo/v:
-rw-------   2 root     root         1143 Nov  1 15:39 vt100
-rw-------   2 root     root         1143 Nov  1 15:39 vt100-am

/var:
drwx--x--x   2 root     root         1024 Nov  1 15:39 log
drwx--x--x   2 root     root         1024 Nov  1 15:39 run
drwx--x--x   2 root     root         1024 Nov  1 15:39 tmp
```

```
/var/log:
-rw-------   1 root     root            0 Nov  1 15:39 wtmp

/var/run:
-rw-------   1 root     root            0 Nov  1 15:39 utmp

/var/tmp:
```

D. Sample utility disk directory listing

```
total 579
-rwxr-xr-x   1 root     root        42333 Jul 28 19:05 cpio
-rwxr-xr-x   1 root     root        32844 Aug 28 19:50 debugfs
-rwxr-xr-x   1 root     root       103560 Jul 29 21:31 elvis
-rwxr-xr-x   1 root     root        29536 Jul 28 19:04 fdisk
-rw-r-r---   1 root     root       128254 Jul 28 19:03 ftape.o
-rwxr-xr-x   1 root     root        17564 Jul 25 03:21 ftmt
-rwxr-xr-x   1 root     root        64161 Jul 29 20:47 grep
-rwxr-xr-x   1 root     root        45309 Jul 29 20:48 gzip
-rwxr-xr-x   1 root     root        23560 Jul 28 19:04 insmod
-rwxr-xr-x   1 root     root          118 Jul 28 19:04 lsmod
lrwxrwxrwx   1 root     root            5 Jul 28 19:04 mt -> mt-st
-rwxr-xr-x   1 root     root         9573 Jul 28 19:03 mt-st
lrwxrwxrwx   1 root     root            6 Jul 28 19:05 rmmod -> insmod
-rwxr-xr-x   1 root     root       104085 Jul 28 19:05 tar
lrwxrwxrwx   1 root     root            5 Jul 29 21:35 vi -> elvis
```

Notes

[1]
: The directory structure presented here is for root diskette use only. Real Linux systems have a more complex and disciplined set of policies, called the Filesystem Hierarchy Standard, for determining where files should go.)