The Linux Bootdisk HOWTO

Tom Fawcett and Graham Chapman v2.1, 18 August 1996

This document describes how to create Linux boot, boot/root and utility maintenance disks. These disks could be used as rescue disks or to test new kernels. Note: if you haven't read the Linux FAQ and related documents such as the Linux Installation HOWTO and the Linux Install Guide, then you should not be trying to build boot diskettes.

Contents

1 Introduction

1.1 Why Build Boot Disks?

Linux boot disks are useful in a number of situations, such as:

- Testing a new kernel.
- Recovering from disk or system failure. Such a failure could be anything from a lost boot sector to a disk head crash.

There are several ways of producing 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 various types of disk to operate, then build your own.

I originally chose the last option - learn how it works so that you can do it yourself. That way, if something breaks, you can work out what to do to fix it. Plus you learn a lot about how Linux works along the way.

Experienced Linux users may find little of use in this document. However users new to Linux system administration who wish to protect against root disk loss and other mishaps may find it useful.

A note on versions - this document has been updated to support the following packages and versions:

- Linux 2.0.6

- LILO 0.19

1.2 Feedback and Credits

We welcome any feedback, good or bad, on the content of this document. Please let us know if you find any errors or omissions. Send comments, corrections and questions to Tom Fawcett (`fawcett@nynexst.com`) or Graham Chapman (`grahamc@zeta.org.au`).

I thank the following people for correcting errors and providing useful suggestions for improvement:

```
Randolph Bentson
Grant R. Bowman
Scott Burkett
Cameron Davidson
Bruce Elliot
HARIGUCHI Youichi
Bjxrn-Helge Mevik
Dwight Spencer
Cameron Spitzer
Johannes Stille
```

1.3 Change History

v2.1, 18 August 1996. Changes in this version:

Summary: this was a major cleanup to reflect changes between kernel version 1.2 and 2.0. Specific changes are:

- Chg: replaced shell scripts and directory listings.
- Chg: removed most of the text of "oversize ramdisk" FAQ question.
- Fix: mkfs -i should have been mke2fs -i.
- Fix: missing parameter name in dd command to zero rootdisk device.
- Fix: remove accidental extra parameter from mke2fs command to create rootdisk filesystem.
- Chg: minor changes to reflect less reliance on the Bootkit utility.
- Chg: change section cross-references to refer to section title, not number (sometime I'll add hypertext links...)

- Add: use cpio as an alternate way of copying files.
- Add: tips for removing unnecessary device special files.
- Add: FAQ question - what to do if nothing happens at boot time.
- Add: various minor changes.

v2.0, 12 June 1996. Changes in this version:

- Add: additional author and maintainer, Tom Fawcett.
- Add: section 6.3, Ramdisk Usage.
- Add: section titled Advanced Bootdisk Creation, which describes how to take advantage of ramdisk changes in kernels 1.3.48+
- Chg: rewrite section on /lib directory.
- Chg: various minor tips on changed ramdisk usage.

Version history:

- v1.02, 25 June 1995 - minor changes.
- v1.01, 6 February 1995 - minor changes.
- v1.0, 2 January 1995 - first release in standard HOWTO layout.
- v0.10, 1 Novemer 1994 - original version, labelled "Draft".

2 Disks

2.1 Summary of Disk Types

I classify boot-related disks into 4 types. The discussion here and throughout this document uses the term "disk" to refer to diskettes unless otherwise specified. Most of the discussion could be equally well applied to hard disks.

A summary of disk types and uses is:

boot

A disk containing a kernel which can be booted. The disk can contain a filesystem and use a boot loader to boot, or it can simply contain the kernel only at the start of the disk. The disk can be used to boot the kernel using a root file system on another disk. This could be useful if you lost your boot loader due to, for example, an incorrect installation attempt.

root

A disk with a file system containing everything required to run a Linux system. It does not necessarily contain either a kernel or a boot loader.

This disk can be used to run the system independently of any other disks, once the kernel has been booted. A special kernel feature allows a separate root disk to be mounted after booting, with the root disk being automatically copied to a ramdisk.

You could use this type of disk to check another disk for corruption without mounting it, or to restore another disk after a disk failure or loss of files.

boot/root

A disk which is the same as a root disk, but contains a kernel and a boot loader. It can be used to boot from, and to run the system. 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 won't necessarily always be possbile to fit everything on a single diskette, even with compression.

utility

A disk which contains a file system, 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.

The term "utility" only really applies to diskettes, where you would use a utility disk to store additional recovery utility software.

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.

2.2 Boot

2.2.1 Overview

All PC systems start the boot process by executing code in ROM to load the sector from sector 0, cylinder 0 of the boot drive and try and execute it. 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.
- the start of an operating system kernel, such as Linux.

If a Linux kernel has been written to a diskette as a raw device, then the first sector will be the first sector of the Linux kernel itself, and this sector will continue the boot process by loading the rest of the kernel and running Linux. For a more detailed description of the boot sector contents, see the documentation in lilo-01.5 or higher.

An alternative method of storing a kernel on a boot disk is to create a filesystem, not as a root filesystem, but simply as a means of installing LILO and thus allowing boot-time command line options to be specified. For example, the same kernel could then be used to boot using a hard disk root filesystem, or a diskette root filesystem. This could be useful if you were trying to rebuild the hard disk filesystem, and wanted to repeatedly test results.

2.2.2 Setting Pointer to Root

The kernel must somehow obtain a pointer to the drive and partititon to be mounted as the root drive. This can be provided in several ways:

- By setting `ROOT_DEV = devicename` in the Linux kernel makefile and rebuilding the kernel (for advice on how to rebuild the kernel, read the Linux FAQ and look in `/usr/src/linux`). Comments in the Linux makefile describe the valid values for `devicename`.
- By running the rdev utility:

  ```
  rdev filename devicename
  ```

  This will set the root device of the kernel contained in `filename` to be `devicename`. For example:

  ```
  rdev zImage /dev/sda1
  ```

  This sets the root device in the kernel in zImage to the first partition on the first SCSI drive.

There are some alternative ways of issuing the rdev command. Try:

```
rdev -h
```

and it will display command usage.

There is usually no need to configure the root device for boot diskette use, because the kernel currently used to boot from probably already points to the root drive device. The need can arise, howoever, if you obtain a kernel from another machine, for example, from a distribution, or if you want to use the kernel to boot a root diskette. It is probably a good idea to check the current root drive setting, just in case it is wrong. To get rdev to check the current root device in a kernel file, enter the command:

```
rdev <filename>
```

It is possible to change the root device set in a kernel by means other than using rdev. For details, see the FAQ at the end of this document.

2.2.3 Copying Kernel to Boot Diskette

Once the kernel has been configured then it must be copied to the boot diskette.

The commands described below (and throughout the HOWTO) assume that the diskettes have been formatted. If not, then use fdformat to format the diskettes before continuing.

If the disk is not intended to contain a file system, then the kernel can be copied using the dd command, as follows:

```
dd if=infilename of=devicename

where   infilename is the name of the kernel
and     devicename is the diskette raw device,
        usually /dev/fd0
```

The cp command can also be used:

```
cp filename devicename
```

For example:

```
        dd if=zImage of=/dev/fd0
    or
        cp zImage /dev/fd0
```

The seek parameter to the dd command should NOT be used. The file must be copied to start at the boot sector (sector 0, cylinder 0), and omitting the seek parameter will do this.

The output device name to be used is usually `/dev/fd0` for the primary diskette drive (i.e. drive "A:" in DOS), and `/dev/fd1` for the secondary. These device names will cause the kernel to autodetect the attributes of the drives. Drive attributes can be specified to the kernel by using other device names: for example `/dev/fd0H1440` specifies a high density 1.44 Mb drive. It is rare to need to use these specific device names.

Where the kernel is to be copied to a boot disk containing a filesystem, then the disk is mounted at a suitable point in a currently-mounted filesystem, then the cp command is used. For example:

```
mount -t ext2 /dev/fd0 /mnt
cp zImage /mnt
umount /mnt
```

Note that for almost all operations in this HOWTO, the user should be operating as the superuser.

2.3 Root

2.3.1 Overview

A root disk contains a complete working Linux system, but without necessarily including a kernel. In other words, the disk may not be bootable, but once the kernel is running, the root disk contains 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:

- File system.
- Minimum set of directories - dev, proc, bin, etc, lib, usr, tmp.
- Basic set of utilities - bash (to run a shell), ls, cp etc.

- Minimum set of config files - rc, inittab, fstab 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/compression utilities including cpio, tar, gzip and ftape.

2.4 Boot/Root

This is essentially the same as the root disk, with the addition of a kernel and a boot loader such as LILO.

With this configuration, a kernel file is copied to the root file system, and LILO is then run to install a configuration which points to the kernel file on the target disk. At boot time, LILO will boot the kernel from the target disk.

Several files must be copied to the diskette for this method to work. Details of these files and the required LILO configuration, including a working sample, are given below in the section titled "LILO".

2.4.1 RAM Disks and Root Filesystems on Diskette

For a diskette root filesystem to be efficient, you need to be able to run it from a ramdisk, i.e. an emulated disk drive in main memory. This avoids having the system run at a snail's pace, which a diskette would impose. The Ftape HOWTO states that a ramdisk will be required when using Ftape because Ftape requires exclusive use of the diskette controller.

There is an added benefit from using a ramdisk - the Linux kernel includes an automatic ramdisk root feature, whereby it will, under certain circumstances, automatically copy the contents of a root diskette to a ramdisk, and then switch the root drive to be the ramdisk instead of the diskette. This has three major benefits:

- The system runs a lot faster.
- The diskette drive is freed up to allow other diskettes to be used on a single-diskette drive system.
- With compression, the ramdisk image on a disk can be substantially smaller than (eg, 40% the size of) the corresponding disk image. This means that a 1.44 meg floppy disk may hold a root containing roughly 3.6 meg.

For kernels 1.3.48+, the ramdisk code was substantially rewritten. You have some more options and the commands for using the ramdisk are somewhat different. The section titled "Advanced Bootdisk Creation" discusses how to take advantage of these.

You must configure your kernel to have ramdisk support, but the ramdisk is dynamically expandible so you need not specify the size. `rdev -r` is no longer used to specify the ramdisk size, but instead sets a ramdisk word in the kernel image. The section titled "Advanced Bootdisk Creation" discusses this in more detail.

If you have a kernel *before* 1.3.48, the following requirements apply. Note that this applies ONLY to kernels prior to 1.3.48.

- The file system on the diskette drive must be either a minix or an ext2 file system. The ext2 file system is generally the preferred file system to use. Note that if you have a Linux kernel earlier than 1.1.73, then you should see the comments in the section titled "File Systems" to see whether your kernel will support ext2. If your kernel is old then you may have to use minix. This will not cause any significant problems.
- A ramdisk must be configured into the kernel, and it must be at least as big as the diskette drive.

A ramdisk can be configured into the kernel in several ways:

- By uncommenting the RAMDISK macro in the Linux kernel makefile, so that it reads:

  ```
  RAMDISK = -DRAMDISK=1440
  ```

  to define a ramdisk of 1440 1K blocks, the size of a high-density diskette.

- By running the rdev utility, available on most Linux systems. This utility displays or sets values for several things in the kernel, including the desired size for a ramdisk. To configure a ramdisk of 1440 blocks into a kernel in a file named zImage, enter:

  ```
  rdev -r zImage 1440
  ```

  this might change in the future, of course. To see what your version of rdev does, enter the command:

  ```
  rdev -h
  ```

  and it should display its options.

- By using the boot loader package LILO to configure it into your kernel at boot time. This can be done using the LILO configuration parameter:

  ```
  ramdisk = 1440
  ```

  to request a ramdisk of 1440 1K blocks at boot time.

- By interrupting a LILO automatic boot and adding ramdisk=1440 to the command line. For example, such a command line might be:

  ```
  zImage ramdisk=1440
  ```

  See the section on LILO for more details.

- By editing the kernel file and altering the values near the start of the file which record the ramdisk size. This is definitely a last resort, but can be done. See the FAQ near the end of this document for more details.

The easiest of these methods is LILO configuration, because you need to set up a LILO configuration file anyway, so why not add the ramdisk size here?

LILO configuration is briefly described in a section titled "LILO" below, but it is advisable to obtain the latest stable version of LILO from your nearest Linux mirror site, and read the documentation that comes with it.

Ramdisks can be made larger than the size of a diskette, and made to contain a filesystem as large as the ramdisk. This can be useful to load all the software required for rescue work onto a single high-performance ramdisk. The method of doing this is described in the FAQ section under the question "How can I create an oversize ramdisk filesystem?"

2.5 Utility

Often one disk is not sufficient to hold all the software you need to be able to perform rescue functions of analysing, repairing and restoring corrupted disk drives. By the time you include tar, gzip e2fsck, fdisk, Ftape and so on, there is enough for a whole new diskette, maybe even more if you want lots of tools.

This means that a rescue set often requires a utility diskette, with a file system containing any extra files required. This file system can then be mounted at a convenient point, such as /usr, on the boot/root system.

Creating a file system is fairly easy, and is described in the section titled "File Systems".

3 Components

3.1 File Systems

The Linux kernel now supports two file system types for root disks to be automatically copied to ramdisk. These are minix and ext2, of which ext2 is the preferred file system. The ext2 support was added sometime between 1.1.17 and 1.1.57, I'm not sure exactly which. If you have a kernel within this range then edit /usr/src/linux/drivers/block/ramdisk.c and look for the word "ext2". If it is not found, then you will have to use a minix file system, and therefore the "mkfs" command to create it. If using ext2, then 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.

To create an ext2 file system on a diskette on my system, I issue the following command:

```
mke2fs -m 0 /dev/fd0
```

The mke2fs command will automatically detect the space available and configure itself accordingly. If desired, the diskette size in 1Kb blocks can be specified to speed up mke2fs operation. The -m 0 parameter prevents it from reserving space for root, and hence provides more usable space on the disk.

An easy way to test the result is to create a system using the above command or similar, and then attempt to mount the diskette. If it is an ext2 system, then the command:

```
mount -t ext2 /dev/fd0 /<mount point>
```

should work.

3.2 Kernel

3.2.1 Building a Custom 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. The kernel is one of the largest files in a minimum system, so if you want to build a boot/root diskette, then you will have to reduce the size of the kernel as much as possible. The kernel now supports changing the diskette after booting and before mounting root, so it is not necessary any more to squeeze the kernel into the same disk as everything else, therefore these comments apply only if you choose to build a boot/root diskette.

There are two ways of reducing kernel size:

- Building 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.
- Compressing it, using the standard compressed-kernel option included in the makefile:

  ```
  make zImage
  ```

  Refer to the documentation included with the kernel source for up-to-date information on building compressed kernels. Note that the kernel source is usually in /usr/src/linux.

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 and read your backups.

It is probably desirable to maintain a copy of the same version of backup utilities used to write the backups, so that you don't waste time trying to install versions that cannot read your backup tapes.

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. Note that if you have trouble building a kernel, then you should probably not attempt to build boot/root systems anyway.

3.3 Devices

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 the mkdir command 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 the -R option. [1] This will copy the directory without attempting to copy the contents of the files. Note that if you use lower caser, as in "-r", there will be a vast difference, because you will probably end up copying the entire contents of all of your hard disks - or at least as much of them as will fit on a diskette! Therefore, take care, and use the command:

```
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 the target file being copied) and that the original file attributes are preserved, thus preserving ownership information.

You can also use the -p option of cpio, because cpio will handle device special files correctly, and not try and copy the contents. For example:

```
cd /dev
find . -print | cpio -pmd /mnt/dev
```

will copy all device special files from /dev to /mnt/dev. In fact it will copy all files in the directory tree starting at /dev, and will create any required subdirectories in the target directory tree.

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.

Many distributions include a shell script called MAKEDEV in the /dev directory. This shell script could be used to create the devices, but it is probably easier to just copy your existing ones, especially for rescue disk purposes.

Whichever way the device directory is 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.

Note that an 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. Many devices are obviously unnecessary on specific systems. For example, if you do not have SCSI disks, then you can safely remove all the device files starting with "sd". Similarly, if you don't intend to use your serial port then all the device files starting with "cua" can go.

[1] Warning: The `cp` command supplied with the most recent version of fileutils (3.13) is reported not to respect the `-R` flag.

3.4 Directories

It might be possible to get away with just /dev, /proc and /etc to run a Linux system. I don't know - I've never tested it. However it will certainly be difficult, because without shared libraries all your executables would have to be statically linked. A reasonable minimum set of directories consists of the following:

/dev

Required to perform I/O with devices

/proc

Required by the ps command

/etc

System configuration files

/bin

Utility executables 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

Note that the directory tree presented here is for root diskette use only. Refer to the Linux File System Standard for much better information on how file systems should be structured in "standard" Linux systems.

Four of these directories can be created very easily:

- /dev is described above in the section titled DEVICES.
- /proc only needs to exist. Once the directory is created using mkdir, nothing more is required.
- Of the others, /mnt and /usr are included in this list only as mount points for use after the boot/root system is running. Hence again, these directories only need to be created.

The remaining 3 directories are described in the following sections.

3.4.1 /etc

This directory must contain a number of configuration files. On most systems, these can be divided into 3 groups:

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

Files which are not essential can 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, then 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:

- The ones I must configure for a boot/root system:

  ```
  rc.d/*  system startup and run level change scripts
  fstab   list of file systems to be mounted
  inittab parameters for the init process - the
          first process started at boot time.
  ```

- the ones I should tidy up for a boot/root system:

  ```
  passwd  list of logins
  shadow  contains passwords
  ```

  These should be pruned on secure systems to avoid copying user's passwords off the system, and so that when you boot from diskette, unwanted logins are rejected. [2]

- 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
  /etc/mount -av
  /bin/hostname boot_root
  ```

[2] Note that there is a reason *not* to prune passwd and shadow. Tar (and probably other archivers) stores user and group names with files. If you restore files to your hard disk from tape, the files will be restored with their original names. If these names do not exist in passwd/group when they are restored, the UID/GID will not be correct.

and I don't really need to run hostname - it just looks nicer if I do. Even mount is actually only needed to mount /proc to support the ps command - Linux will run without it, although rescue operations are rather limited without mount!

- fstab should contain:

```
/dev/ram      /        ext2   defaults
/dev/fd0      /        ext2   defaults
/proc         /proc    proc   defaults
```

  I don't think that the first entry is really needed, but I find that if I leave it out, mount won't mount /proc.

Inittab should be ok as is, unless you want to ensure that users on serial ports cannot login. To prevent this, comment out all the entries for /etc/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.

Inittab defines what the system will run or rerun in various states including startup, move to multi-user mode, powerfail, and others. A point to be careful of here is to carefully check that the commands entered in inittab refer to programs which are present and to the correct directory. If you place your command files on your rescue disk using the sample directory listing in this HOWTO as a guide, and then copy your inittab to your rescue disk without checking it, then the probability of failure will be quite high, because half of the inittab entries will refer to missing programs or to the wrong directory.

It is worth noting here as well that some programs cannot be moved from one directory to another or they will fail at runtime because they have hardcoded the name of another program which they attempt to run. 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 can't 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 ls listing in "Sample Boot/Root ls-lR Directory Listing" - this is what I have, so probably it will be sufficient for you if you copy only those files - but note that systems differ a great deal, so you cannot be sure that the same set of files on your system is equivalent to the files on mine. 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 absolute minimum is a single rc script, but it will probably be a lot simpler in practice to copy the inittab and /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.

3.4.2 /bin

Here is a convenient point to place the extra utilities you need to perform basic operations, utilities such as ls, mv, cat, dd etc.

See the section titled "Sample Boot/Root ls-lR Directory Listing" for the list of files that I place in my boot/root /bin directory. You may notice that it does not include any of the utilities required to restore from backup, such as cpio, tar, gzip etc. That is because I place these on a separate utility diskette, to save space on the boot/root diskette. Once I have booted my boot/root diskette, it then copies itself 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 the section titled "Adding Utility Diskettes".

3.4.3 /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.X
```

where X is the current version number. Check your /lib directory. Note that libc.so.4 may be a symlink to a libc library with version number in the filename. If you issue the command:

```
ls -l /lib
```

you will see something like:

```
libc.so.4 -> libc.so.4.5.21
```

In this case, the libc library you want is libc.so.4.5.21. This is an example only - the ELF libc library is currently libc.so.5.xxxx.

To find other libraries you should go through all the binaries you plan to include and check their dependencies. You can do this with ldd command. For example, on my system the command:

```
ldd /bin/mount
```

produces the output:

```
/bin/mount:
        libc.so.5 => /lib/libc.so.5.2.18
```

indicating that /bin/mount needs the library libc.so.5, which is a symbolic link to libc.so.5.2.18.

In /lib you must also include one or more loaders to load the libraries. The loader file is either ld.so (for a.out libraries) or ld-linux.so (for ELF libraries). If you're not sure which you need, run the "file" command on the library. For example, on my system:

```
file /lib/libc.so.5.2.18
```

tells me:

```
/lib/libc.so.5.2.18: ELF 32-bit LSB shared object ...
```

so it needs an ELF loader. If you have an a.out library you'll instead see something like:

```
/lib/libc.so.4.7.2: Linux/i386 demand-paged executable (QMAGIC) ...
```

Copy the specific loader(s) you need.

Libraries and loaders should be checked carefully against the included binaries. If the kernel cannot load a necessary library, the kernel will usually hang with no error message.

3.5 LILO

3.5.1 Overview

For the boot/root to be any use, it must be bootable. To achieve this, the easiest way is to install a boot loader, which is a piece of executable code stored at sector 0, cylinder 0 of the diskette. See the section above titled "BOOT DISKETTE" for an overview of the boot process.

LILO is a tried and trusted boot loader available from any Linux mirror site. It allows you to configure the boot loader, including:

- Which device is to be mounted as the root drive.
- Whether to use a ramdisk.

3.5.2 Sample LILO Configuration

This provides a very convenient place to specify to the kernel how it should boot. My root/boot LILO configuration file, used with LILO 0.15, is:

```
boot = /dev/fd0
install = ./mnt/boot.b
map = ./mnt/lilo.map
delay = 50
message = ./mnt/lilo.msg
timeout = 150
compact
image = ./mnt/zImage
        ramdisk = 1440
        root = /dev/fd0
```

Note that I have not tested this recently, because I no longer use LILO-based boot/root diskettes. There is no reason to suppose that it does not still work, but if you try it and it fails, you must read the LILO documentation to find out why.

Note also that boot/root systems no longer rely on LILO, because since 1.3.48, the kernel supports loading a compressed root filesystem from the same diskette as the kernel. See the section "Advanced Bootdisk Creation" for details.

If you have a kernel later than 1.3.48, the "ramdisk = 1440" line is unnecessary and should be removed.

Note that boot.b, lilo.msg and the kernel must first have been copied to the diskette using a command similar to:

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

If this is not done, then LILO will not run correctly at boot time if the hard disk is not available, and there is little point setting up a rescue disk which requires a hard disk in order to boot.

I run lilo using the command:

```
/sbin/lilo -C <configfile>
```

I run it from the directory containing the mnt directory where I have mounted the diskette. This means that I am telling LILO to install a boot loader on the boot device (/dev/fd0 in this case), to boot a kernel in the root directory of the diskette.

I have also specified that I want the root device to be the diskette, and I want a ramdisk created of 1440 1K blocks, the same size as the diskette. Since I have created an ext2 file system on the diskette, this completes all the conditions required for Linux to automatically switch the root device to the ramdisk, and copy the diskette contents there as well.

The ramdisk features of Linux are described further in the section above titled "RAM DISKS AND BOOT/ROOT SYSTEMS".

It is also worth considering using the "single" parameter to cause Linux to boot in single-user mode. This could be useful to prevent users logging in on serial ports.

I also use the "DELAY" "MESSAGE" and "TIMEOUT" statements so that when I boot the disk, LILO will give me the opportunity to enter command line options if I wish. I don't need them at present, but I never know when I might want to set a different root device or mount a filesystem read-only.

The message file I use contains the message:

```
Linux Boot/Root Diskette
========================

Enter a command line of the form:

      zImage [ command-line options]

If nothing is entered, linux will be loaded with
defaults after 15 seconds.
```

This is simply a reminder to myself what my choices are.

Readers are urged to read the LILO documentation carefully before atttempting to install anything. It is relatively easy to destroy partitions if you use the wrong "boot = " parameter. If you are inexperienced, do NOT run LILO until you are sure you understand it and you have triple-checked your parameters.

Note that you must re-run LILO every time you change the kernel, so that LILO can set up its map file to correctly describe the new kernel file. It is in fact possible to replace the kernel file with one which is almost identical without rerunning LILO, but it is far better not to gamble - if you change the kernel, re-run LILO.

3.5.3 Removing LILO

One other thing I might as well add here while I'm on the LILO topic: if you mess up lilo on a drive containing DOS, you can always replace the boot sector with the DOS boot loader by issuing the DOS command:

```
FDISK /MBR
```

where MBR stands for "Master Boot Record". Note that some purists disagree with this, and they may have grounds, but it works.

3.5.4 Useful LILO Options

LILO has several useful options which are worth keeping in mind when building boot disks:

- Command line options - you can enter command line options to set the root device, ramdisk size (for kernels less than 1.3.48), special device parameters, or other things. If you include the DELAY = nn statement in your LILO configuration file, then LILO will pause to allow you to select a kernel image to boot, and to enter, on the same line, any options. For example:

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

  will pass the aha152x parameters through to the aha152x scsi disk driver (provided that driver has been included when the kernel was built) and will ask for the root filesystem to be mounted read-only.

- Command line "lock" option - this option asks LILO to store the command line entered as the default command line to be used for all future boots. This is particularly useful where you have a device which cannot be autoselected. By using "lock" you can avoid having to type in the device parameter string every time you boot. For example:

  ```
  zImage aha152x=0x340,11,3,1 root=/dev/sda8 ro lock
  ```

- APPEND configuration statement - this allows device parameter strings to be stored in the configuration, as an alternative to using the "lock" command line option. Note that any keywords of the form word=value MUST be enclosed in quotes. For example:

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

- DELAY configuration statement - this pauses for DELAY tenths of seconds and allows the user to interrupt the automatic boot of the default command line, so that the user can enter an alternate command line.

4 Advanced Bootdisk Creation

4.1 Overview

Previous Sections of this document covered the basics of creating boot/root disks, and are applicable to nearly all kernels up to the present (2.0, the latest stable kernel).

Kernels 1.3.48+ involved a substantial rewrite of the ramdisk code, adding significant new capabilities. These kernels could automatically detect compressed filesystems, uncompress them and load them into the ramdisk on boot-up. Root filesystems could be placed on a second disk, and as of kernel 1.3.98 or so, ramdisks are dynamically expandable.

Altogether, these new capabilities mean that boot disks can contain substantially more than they used to. With compression, a 1722K disk may now hold up to about 3.5 megs of files. As anyone who has created bootdisks knows, much time is spent pruning down the file set and finding trimmed-down versions of files that will all fit in a small filesystem. With the new capabilities this is no longer such a concern.

Unfortunately, creating bootdisks to exploit these new features is slightly more difficult now. To build a compressed filesystem on a floppy, the filesystem has to be built on another device and then compressed and transferred to the floppy. This means a few more steps.

The basic strategy is to create a compressed root filesystem, copy the kernel to the floppy disk, tell the kernel where to find the root filesystem, then copy the compressed root filesystem to the floppy.

Here's a simple ASCII drawing of what the disk will look like:

```
|<--- zImage --->|<------ Compressed root filesystem -------->|
|________________|____________________________________________|
               Floppy disk space
```

Here are the steps to create the boot floppy:

4.2 Creating a root filesystem

The root filesystem is created pretty much the same way as outlined in Section 2.3 of this document. The primary difference is that you can no longer create a root filesystem directly on a floppy – you must create it on a separate device larger than the floppy area it will occupy.

4.2.1 Choosing a device

In order to build such a root filesystem, you need a spare device that is large enough. There are several choices:

- If you have an unused hard disk partition that is large enough (several megabytes), this is the easiest solution. Alternatively, if you have enough physical RAM you can simply turn off swapping and build the filesystem in your swap partition.

  However, most people don't have a spare partition and can't afford to turn swapping off, so...

- Use a loopback device. A loopback device allows a disk file on an existing filesystem to be treated as a device. In order to use loopback devices you need specially modified mount and unmount programs. You can find these at:

  ```
  ftp://ftp.win.tue.nl:/pub/linux/util/mount-2.5X.tar.gz
  ```

  where X is the latest modification letter.

  If you do not have loop devices (/dev/loop0, /dev/loop1, etc) on your system, you'll have to create them first. The command

  ```
  MAKEDEV loop
  ```

  will do this. [3]

  One you've installed these special mount/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.

- A final option is to use the ramdisk (DEVICE = /dev/ram or /dev/ramdisk). In this case, RAM is used to simulate a disk drive. The ramdisk must be large enough to hold a filesystem of the appropriate size. Check your Lilo configuration file (/etc/lilo.conf) for a line like:

  ```
  RAMDISK_SIZE = nnn
  ```

  which determines how much RAM will be allocated. The default is 4096K.

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

```
dd if=/dev/zero of=DEVICE bs=1k count=3000
```

This command zeroes out the device. This step is important because the filesystem on the device will be compressed later, so all unused portions should be filled with zeroes to achieve maximum compression.

Next, create the filesystem with:

```
mke2fs -m 0 DEVICE
```

(If you're using a loopback device, the disk file you're using should be supplied in place of this DEVICE. In this case, mke2fs will ask if you really want to do this; say yes.)

Then mount the device:

[3] Older MAKEDEV scripts may not know about loop devices. A MAKEDEV that can create loop devices is available in the directory `ftp://tsx-11.mit.edu/pub/linux/sources/sbin/`

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

Proceed as before, copying files into /mnt, as specified in Section 2.3.

4.2.2 Compressing the filesystem

After you're done copying files into the root filesystem, you need to copy it back out and compress it. First, umount it:

```
umount /mnt
```

(Technically you can copy the filesystem without unmounting it first, but that's somewhat dangerous, and bad practice.)

Next, copy data off the device to a disk file. Call the disk file `rootfs`:

```
dd if=DEVICE of=rootfs bs=1k
```

Then compress it. Use the '-9' option of gzip for maximal compression:

```
gzip -9 rootfs
```

This may take several minutes. When it finishes, you'll have a file rootfs.gz that is your compressed root filesystem.

If you're tight on disk space you can combine dd and gzip:

```
dd if=DEVICE bs=1k | gzip -9 > rootfs.gz
```

4.3 Calculating the space

At this point, check the space to make sure both the kernel and the root filesystem will fit on the floppy. An "ls -l" will show how many bytes each occupies; divide by 1024 to determine how many blocks each will need. For partial blocks, be sure to round up to the next block.

For example, if the kernel size is 453281 bytes, it will need

```
ceil(453281 / 1024) = 443
```

blocks, so it will occupy blocks 0-442 on the floppy disk. The compressed root filesystem will begin at block 443. Remember this block number for the commands to follow; call it ROOTBEGIN.

You must tell the kernel where on the floppy to find the root filesystem. Inside the kernel image is a ramdisk word that specifies where the root filesystem is to be found, along with other options. The word is defined in /usr/src/linux/arch/i386/kernel/setup.c and is interpreted as follows:

```
bits  0-10:     Offset to start of ramdisk, in 1024 byte blocks
                (This is ROOTBEGIN, calculated above)
bits 11-13:     unused
bit     14:     Flag indicating that ramdisk is to be loaded
bit     15:     Flag indicating to prompt for floppy
```

(If bit 15 is set, on boot-up you will be prompted to place a new floppy disk in the drive. This is necessary for a two-disk boot set, discussed below in the section "Making a two-disk set". For now, this will be zero.)

If the root filesystem is to begin at block 443, the ramdisk word is

```
   1BB (hex)       443 (decimal)     (bits 0-10)
+ 4000 (hex)       Ramdisk load flag (bit 14)
  ----------
= 41BB (hex)
=16827 (decimal)
```

This ramdisk word will be set in the kernel image using the "rdev -r" command in the next section.

4.4 Copying files to the floppy

At this point you're ready to create the boot floppy. First copy the kernel:

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

Next, tell the kernel to find its root filesystem on the floppy:

```
rdev /dev/fd0 /dev/fd0
```

Next, you have to set the ramdisk word in the kernel image now residing on the floppy. The ramdisk word is set using the "rdev -r" command. Using the figure calculated above in the section titled "Calculating the space":

```
rdev -r zImage 16827
```

Finally, place the root filesystem on the floppy after the kernel. The `dd` command has a `seek` option that allows you to specify how many blocks to skip:

```
dd if=rootfs.gz of=/dev/fd0 bs=1k seek=443
```

(The value 443 is ROOTBEGIN from the section "Calculating the space" above.)

Wait for the floppy drive to finish writing, and you're done.

4.5 Making a two-disk set

If you want more space, you can make a two-disk boot set. In this case, the first floppy disk will contain the kernel alone, and the second will contain the compressed root filesystem. With this configuration you can use a compressed filesystem of up to 1440K.

A two-disk set is created using a simple variation of the instructions above. First, you must set the ramdisk PROMPT flag to 1 to instruct the kernel to prompt and wait for the second floppy. The root filesystem will begin at byte 0 of the second floppy.

From the section "Calculating the space" above, the ramdisk PROMPT flag (bit 15) will be set to 1, and the ramdisk offset will be zero. In our example the new calculation would be:

```
  4000 (hex)      Ramdisk load flag (bit 14)
+ 8000 (hex)      Ramdisk prompt flag (bit 15)
------------
= C000 (hex)
=49152 (decimal)
```

which would be used in the 'rdev -r' calculation as before.

Follow the instructions of "Copying files to the floppy" above, but after issuing the 'rdev -r' command, put a new floppy in the drive and issue the command:

```
dd if=rootfs.gz of=/dev/fd0
```

The `seek` option is not needed since the root filesystem starts at block zero.

5 Samples

5.1 Disk Directory Listings

This lists the contents of directories from my root and utility diskettes. These lists are provided as an example only of the files included to create a working system. I have added some explanatory notes where it seemed useful.

5.1.1 Root Disk ls-lR Directory Listing

```
total 18
drwxr-xr-x   2 root     root         1024 Jul 29 21:16 bin/
drwxr-xr-x   2 root     root         9216 Jul 28 16:21 dev/
drwxr-xr-x   3 root     root         1024 Jul 29 20:25 etc/
drwxr-xr-x   2 root     root         1024 Jul 28 19:53 lib/
drwxr-xr-x   2 root     root         1024 Jul 24 22:47 mnt/
drwxr-xr-x   2 root     root         1024 Jul 24 22:47 proc/
drwxr-xr-x   2 root     root         1024 Jul 28 19:07 sbin/
drwxr-xr-x   2 root     root         1024 Jul 29 20:57 tmp/
```

```
drwxr-xr-x   4 root     root         1024 Jul 29 21:35 usr/
drwxr-xr-x   3 root     root         1024 Jul 28 19:52 var/

bin:
total 713
-rwxr-xr-x   1 root     bin          7737 Jul 24 22:16 cat*
-rwxr-xr-x   1 root     bin          9232 Jul 24 22:48 chmod*
-rwxr-xr-x   1 root     bin          8156 Jul 24 22:48 chown*
-rwxr-xr-x   1 root     bin         19652 Jul 24 22:48 cp*
-rwxr-xr-x   1 root     root         8313 Jul 29 21:16 cut*
-rwxr-xr-x   1 root     bin         12136 Jul 24 22:48 dd*
-rwxr-xr-x   1 root     bin          9308 Jul 24 22:48 df*
-rwxr-xr-x   1 root     root         9036 Jul 29 20:24 dircolors*
-rwxr-xr-x   1 root     bin          9064 Jul 24 22:48 du*
-rwxr-x---   1 root     bin         69252 Jul 24 22:51 e2fsck*
-rwxr-xr-x   1 root     bin          5361 Jul 24 22:48 echo*
-rwxr-xr-x   1 root     bin          5696 Jul 24 22:16 hostname*
-rwxr-xr-x   1 root     bin          6596 Jul 24 22:49 kill*
-rwxr-xr-x   1 root     bin         10644 Jul 24 22:17 ln*
-rwxr-xr-x   1 root     bin         13508 Jul 24 22:17 login*
-rwxr-xr-x   1 root     bin         26976 Jul 24 22:17 ls*
-rwxr-xr-x   1 root     bin          7416 Jul 24 22:49 mkdir*
-rwxr-x---   1 root     bin         34596 Jul 24 22:51 mke2fs*
-rwxr-xr-x   1 root     bin          6712 Jul 24 22:49 mknod*
-rwxr-xr-x   1 root     bin         20304 Jul 24 22:17 more*
-rwxr-xr-x   1 root     bin         24704 Jul 24 22:17 mount*
-rwxr-xr-x   1 root     bin         12464 Jul 24 22:17 mv*
-rwxr-xr-x   1 root     bin         20829 Jul 24 22:50 ps*
-rwxr-xr-x   1 root     bin          9424 Jul 24 22:50 rm*
-rwxr-xr-x   1 root     bin          4344 Jul 24 22:50 rmdir*
-rwxr-xr-x   1 root     root       299649 Jul 27 14:12 sh*
-rwxr-xr-x   1 root     bin          9853 Jul 24 22:17 su*
-rwxr-xr-x   1 root     bin           380 Jul 27 14:12 sync*
-rwxr-xr-x   1 root     bin         13620 Jul 24 22:17 umount*
-rwxr-xr-x   1 root     root         5013 Jul 29 20:03 uname*

dev:
total 0
lrwxrwxrwx   1 root     root           10 Jul 24 22:34 cdrom -> /dev/sbpcd
crw--w--w-   1 root     tty        4,   0 Jul 24 21:49 console
brw-rw----   1 root     floppy     2,   0 Apr 28  1995 fd0
lrwxrwxrwx   1 root     root            4 Jul 24 22:34 ftape -> rft0
crw-rw-rw-   1 root     sys       10,   2 Jul 18  1994 inportbm
crw-rw----   1 root     kmem       1,   2 Jul 28 16:21 kmem
crw-rw----   1 root     kmem       1,   1 Jul 18  1994 mem
lrwxrwxrwx   1 root     root            4 Jul 24 22:34 modem -> cua0
lrwxrwxrwx   1 root     root            4 Jul 24 22:34 mouse -> cua1
crw-rw-rw-   1 root     sys        1,   3 Jul 18  1994 null
```

```
brw-rw----   1 root     disk       1,   1 Jul 18  1994 ram
crw-rw----   1 root     disk      27,   0 Jul 18  1994 rft0
brw-rw----   1 root     disk      25,   0 Jul 19  1994 sbpcd
***  I have only included devices for the SCSI partitions I use.
***  If you use IDE, then use /dev/hdxx instead.
brw-rw----   1 root     disk       8,   0 Apr 29  1995 sda
brw-rw----   1 root     disk       8,   6 Apr 29  1995 sda6
brw-rw----   1 root     disk       8,   7 Apr 29  1995 sda7
brw-rw----   1 root     disk       8,   8 Apr 29  1995 sda8
lrwxrwxrwx   1 root     root            7 Jul 28 12:56 systty -> console
***  this link from systty to console is required
crw-rw-rw-   1 root     tty        5,   0 Jul 18  1994 tty
crw--w--w-   1 root     tty        4,   0 Jul 18  1994 tty0
crw--w----   1 root     tty        4,   1 Jul 24 22:33 tty1
crw--w----   1 root     tty        4,   2 Jul 24 22:34 tty2
crw--w--w-   1 root     root       4,   3 Jul 24 21:49 tty3
crw--w--w-   1 root     root       4,   4 Jul 24 21:49 tty4
crw--w--w-   1 root     root       4,   5 Jul 24 21:49 tty5
crw--w--w-   1 root     root       4,   6 Jul 24 21:49 tty6
crw-rw-rw-   1 root     tty        4,   7 Jul 18  1994 tty7
crw-rw-rw-   1 root     tty        4,   8 Jul 18  1994 tty8
crw-rw-rw-   1 root     tty        4,   9 Jul 19  1994 tty9
crw-rw-rw-   1 root     sys        1,   5 Jul 18  1994 zero

etc:
total 20
-rw-r--r--   1 root     root         2167 Jul 29 20:25 DIR_COLORS
-rw-r--r--   1 root     root           20 Jul 28 12:37 HOSTNAME
-rw-r--r--   1 root     root          109 Jul 24 22:57 fstab
-rw-r--r--   1 root     root          271 Jul 24 22:21 group
-rw-r--r--   1 root     root         2353 Jul 24 22:27 inittab
-rw-r--r--   1 root     root            0 Jul 29 21:02 issue
-rw-r--r--   1 root     root         2881 Jul 28 19:38 ld.so.cache
***  Lots of things get upset at boot time if ld.so.cache is missing, but
***  make sure that ldconfig is included and run from rc.x to
***  update it.
-rw-r--r--   1 root     root           12 Jul 24 22:22 motd
-rw-r--r--   1 root     root          606 Jul 28 19:25 passwd
-rw-r--r--   1 root     root         1065 Jul 24 22:21 profile
drwxr-xr-x   2 root     root         1024 Jul 29 21:01 rc.d/
-rw-r--r--   1 root     root           18 Jul 24 22:21 shells
-rw-r--r--   1 root     root          774 Jul 28 13:43 termcap
-rw-r--r--   1 root     root          126 Jul 28 13:44 ttys
-rw-r--r--   1 root     root            0 Jul 24 22:47 utmp

etc/rc.d:
total 5
*** I didn't bother with shutdown scripts - everthing runs on a
```

```
*** ramdisk, so there's not much point shutting it down.
-rwxr-xr-x   1 root     root         1158 Jul 24 22:23 rc.K*
-rwxr-xr-x   1 root     root         1151 Jul 28 19:08 rc.M*
-rwxr-xr-x   1 root     root          507 Jul 29 20:25 rc.S*

lib:
total 588
*** I have an ELF system, so I include the ELF loader ld-linux.so. if
*** you are still on a.out, then you need ld.so. Use the file command to
*** see which libraries you should include.
lrwxrwxrwx   1 root     root           17 Jul 24 23:36 ld-linux.so.1 -> ld-linux.so.1.7.3*
-rwxr-xr-x   1 root     root        20722 Aug 15  1995 ld-linux.so.1.7.3*
lrwxrwxrwx   1 root     root           13 Jul 24 23:36 libc.so.5 -> libc.so.5.0.9*
-rwxr-xr-x   1 root     root       562683 May 19  1995 libc.so.5.0.9*
***  Must include libtermcap
lrwxrwxrwx   1 root     root           19 Jul 28 19:53 libtermcap.so.2 -> libtermcap.so.2.0.0*
-rwxr-xr-x   1 root     root        11360 May 19  1995 libtermcap.so.2.0.0*

mnt:
total 0

proc:
total 0

sbin:
total 191
***  I use Slackware, which uses agetty. Many systems use getty.
***  Check your /etc/inittab to see which it uses. Note that you
***  need (a)getty and login to be able to start doing much.
-rwxr-xr-x   1 root     bin         11309 Jul 24 22:54 agetty*
-rwxr-xr-x   1 root     bin          5204 Jul 24 22:19 halt*
***  Must have this to boot
-rwxr-xr-x   1 root     bin         20592 Jul 24 22:19 init*
-rwxr-xr-x   1 root     root        86020 Jul 28 19:07 ldconfig*
-rwxr-xr-x   1 root     bin          5329 Jul 27 14:10 mkswap*
-rwxr-xr-x   1 root     root         5204 Jul 24 22:20 reboot*
-rwxr-xr-x   1 root     bin         12340 Jul 24 22:20 shutdown*
-rwxr-xr-x   1 root     root         5029 Jul 24 22:20 swapoff*
-rwxr-xr-x   1 root     bin          5029 Jul 24 22:20 swapon*
-rwxr-xr-x   1 root     root        20592 Jul 27 18:18 telinit*
-rwxr-xr-x   1 root     root         7077 Jul 24 22:20 update*

tmp:
total 0

usr:
total 2
drwxr-xr-x   2 root     root         1024 Jul 29 21:00 adm/
```

```
drwxr-xr-x   2 root     root         1024 Jul 29 21:16 lib/

usr/adm:
total 0

usr/lib:
total 0

var:
total 1
***  Several things complained until I included this and
***  the /etc/rc.S code to initialise /var/run/utmp, but this
***  won't necessarily apply to your system.
drwxr-xr-x   2 root     root         1024 Jul 28 19:52 run/

var/run:
total 0
```

5.1.2 Utility Disk ls-lR Directory Listing

```
total 579
-rwxr-xr-x   1 root     root        42333 Jul 28 19:05 cpio*
-rwxr-xr-x   1 root     root       103560 Jul 29 21:31 elvis*
-rwxr-xr-x   1 root     root        56401 Jul 28 19:06 find*
-rw-r--r--   1 root     root       128254 Jul 28 19:03 ftape.o
-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*
```

5.2 Shell Scripts to Build Diskettes

These shell scripts are provided as examples only. I use them on my system to create rescue diskettes. You may find it convenient to use them, but if so, read the instructions carefully - for example, if you specify the wrong swap device, you will find your root filesystem has been throroughly and permanently erased.... so just be darn sure you have it correctly configured before you use it!

The upside of the scripts are that they provide a quick way to get a rescue set together, by doing the following:

- copy a kernel to a bootdisk, and use rdev to configure it, as explained above.

- adjust mkroot to your system and build a root disk. Use the directory listing above as a guide to what to include.
- use mkutil to throw your favourite utilities onto one or more utility disks.

There are two shell scripts:

- mkroot - builds a root or boot/root diskette.
- mkutil - builds a utility diskette.

Both are currently configured to run in the parent directory of boot_disk and util_disk, each of which contains everything to be copied to it's diskette. Note that these shell scripts will *NOT* automatically set up and copy all the files for you - you work out which files are needed, set up the directories and copy the files to those directories. The shell scripts are samples which will copy the contents of those directories. Note that they are primitive shell scripts and are not meant for the novice user.

The scripts both contain configuration variables at the start which allow them to be easily configured to run anywhere. First, set up the model directories and copy all the required files into them. To see what directories and files are needed, have a look at the sample directory listings in the previous sections.

Check the configuration variables in the shell scripts and change them as required before running the scripts.

5.2.1 mkroot - Make Root Diskette

```
# mkroot: make a root disk - creates a root diskette
#         by building a file system on it, then mounting it and
#         copying required files from a model.
#         Note: the model to copy from from must dirst be set up,
#         then change the configuration variables below to suit
#         your system.
#
# usage: mkroot [ -d swap | ram ]
#       where swap means use $SWAPDEV swap device
#       and ram means use $RAMDISKDEV ramdisk device

# Copyright (c) Graham Chapman 1996. All rights reserved.
# Permission is granted for this material to be freely
# used and distributed, provided the source is acknowledged.
# No warranty of any kind is provided. You use this material
# at your own risk.

# Configuration variables - set these to suit your system
#
####  set the device to use to build the root filesystem on.
####  ramdisk is safer - swap is ok only if you have plenty of
####  free memory. If linux can't swap then things get nasty.
USEDEVICE="ramdisk"             # set to either "ramdisk" or "swap"
```

```
RAMDISKDEV="/dev/ram"          # ramdisk device <==== CHANGE if using ramdisk
SWAPDEV="/dev/sda7"            # swap device    <==== CHANGE if using swap
FSBLOCKS=3072                  # desired filesystem size in blocks
#
####  set name or directory where you have set up your rootdisk
####  model
ROOTDISKDIR="./root_disk"      # name of root disk directory
MOUNTPOINT="/mnt"              # temporary mount point for diskette
DISKETTEDEV="/dev/fd0"         # device name of diskette drive
LOGFL="`pwd`/mkroot.log"       # log filename
TEMPROOTFS="/tmp/mkrootfs.gz"  # temp file for compressed filesystem
# End of Configuration variables

# Internal variables
ROOTDISKDEV=

case $USEDEVICE in
swap|ramdisk)   :;;
*)      echo "Invalid setting for USEDEVICE variable"
        exit;;
esac

clear
echo "    ***************** W A R N I N G ******************

Use this script with care. If you don't understand it, then
exit NOW!"

if [ "$USEDEVICE" = "swap" ]
then
        ROOTDISKDEV=$SWAPDEV
        echo -e "\nThis script will temporarily remove the swap file $SWAPDEV"
        echo "and use the space to build a compressed root filesystem from"
        echo "the files in the directory tree below $ROOTDISKDIR. To do this"
        echo "safely you must have 8Mb or more of memory, and you should"
        echo "switch to single user mode via 'init 1'."
        echo -e "\nIf you have used a ramdisk since the last reboot, then"
        echo "reboot NOW before using this script."
        echo -e "\nIf the script fails, you may not have a swap partition. Run 'free'"
        echo "and check the total size to see if it is correct. If the swap"
        echo "partition $SWAPDEV is missing, do the following:"
        echo "  umount $MOUNTPOINT"
        echo "  mkswap $SWAPDEV"
        echo "  swapon $SWAPDEV"
        echo "to restore the swap partition $SWAPDEV."
else
        ROOTDISKDEV=$RAMDISKDEV
        echo -e "\nThis script will use a ramdisk of $FSBLOCKS Kb. To do this safely"
```

```
        echo "you must have at least 8Mb of memory. If you have only 8Mb you should"
        echo "ensure nothing else is running on the machine."
        echo -e "\nWhen the script is complete, the ramdisk will still be present, so"
        echo "you should reboot to reclaim the memory allocated to the ramdisk."
fi

echo -e "
Do you want to continue (y/n)? \c"
read ans
if [ "$ans" != "Y" -a $ans != "y" ]
then
        echo "not confirmed - aborting"
        exit
fi

echo "Starting mkroot at `date`" > $LOGFL

if [ "$USEDEVICE" = "swap" ]
then
        echo "Unmounting swap device $SWAPDEV" | tee -a $LOGFL
        swapoff $SWAPDEV >> $LOGFL 2>&1
fi

echo "Zeroing device $ROOTDISKDEV" | tee -a $LOGFL
dd if=/dev/zero of=$ROOTDISKDEV bs=1024 count=$FSBLOCKS >> $LOGFL 2>&1
if [ $? -ne 0 ]
then
        echo "dd zeroing $ROOTDISKDEV failed" | tee -a $LOGFL
        exit 1
fi

echo "Creating filesystem on device $ROOTDISKDEV" | tee -a $LOGFL
mke2fs -m0 $ROOTDISKDEV $FSBLOCKS >> $LOGFL 2>&1

echo "Mounting $ROOTDISKDEV filesystem at $MOUNTPOINT" | tee -a $LOGFL
mount -t ext2 $ROOTDISKDEV $MOUNTPOINT >> $LOGFL 2>&1
if [ $? -ne 0 ]
then
        echo "mount failed"
        exit 1
fi

# copy the directories containing files
echo "Copying files from $ROOTDISKDIR to $MOUNTPOINT" | tee -a $LOGFL
currdir=`pwd`
cd $ROOTDISKDIR
find . -print | cpio -dpumv $MOUNTPOINT >> $LOGFL 2>&1
if [ $? -ne 0 ]
```

```
then
        echo "cpio step failed."
        cd $currdir
        exit 1
fi
cd $currdir

fssize=`du -sk $MOUNTPOINT|cut -d"      " -f1`
echo "Uncompressed root filesystem size is $fssize Kb" | tee -a $LOGFL
echo "Unmounting filesystem from $ROOTDISKDEV" | tee -a $LOGFL
umount $MOUNTPOINT >> $LOGFL 2>&1

echo "Compressing filesystem from $ROOTDISKDEV into $TEMPROOTFS
        This may take a few minutes..." | tee -a $LOGFL

#       We don't bother with gzip -9 here - takes more than twice as long
#       and saves less than 1% in space on my root disk...
dd if=$ROOTDISKDEV bs=1024 count=$FSBLOCKS 2>>$LOGFL | gzip -c > $TEMPROOTFS

fssize=`du -k $TEMPROOTFS|cut -d"       " -f1`
echo "Compressed root filesystem size is $fssize Kb" | tee -a $LOGFL

echo -e "Insert diskette in $DISKETTEDEV and press any key
        ***  Warning: data on diskette will be overwritten!\c"
read ans

echo "Copying compressed filesystem from $TEMPROOTFS to $DISKETTEDEV" | tee -a $LOGFL
dd if=$TEMPROOTFS of=$DISKETTEDEV >>$LOGFL 2>&1
if [ $? -ne 0 ]
then
        echo "copy step failed."
        exit 1
fi

if [ "$USEDEVICE" = "swap" ]
then
        echo "Reinitialising swap device $SWAPDEV" | tee -a $LOGFL
        mkswap $SWAPDEV >> $LOGFL 2>&1
        echo "Starting swapping to swap device $SWAPDEV" | tee -a $LOGFL
        swapon $SWAPDEV >> $LOGFL 2>&1
fi

echo "Deleting $TEMPROOTFS" | tee -a $LOGFL
rm $TEMPROOTFS

echo "mkroot completed at `date`" >> $LOGFL

echo "Root diskette creation complete - please read log file $LOGFL"
```

5.2.2 mkutil - Make Utility Diskette

```
# mkutil: make a utility diskette - creates a utility diskette
#         by building a file system on it, then mounting it and
#         copying required files from a model.
#         Note: the model to copy from from must first be set up,
#         then change the configuration variables below to suit
#         your system.

# Copyright (c) Graham Chapman 1996. All rights reserved.
# Permission is granted for this material to be freely
# used and distributed, provided the source is acknowledged.
# No warranty of any kind is provided. You use this material
# at your own risk.

# Configuration variables...
UTILDISKDIR=./util_disk        # name of directory containing model
MOUNTPOINT=/mnt                # temporary mount point for diskette
DISKETTEDEV=/dev/fd0           # device name of diskette drive

echo $0: create utility diskette
echo Warning: data on diskette will be overwritten!
echo Insert diskette in $DISKETTEDEV and and press any key...
read anything

mke2fs $DISKETTEDEV
if [ $? -ne 0 ]
then
        echo mke2fs failed
        exit
fi

# Any file system type would do here
mount -t ext2 $DISKETTEDEV $MOUNTPOINT
if [ $? -ne 0 ]
then
        echo mount failed
        exit
fi

# copy the directories containing files
cp -dpr $UTILDISKDIR/* $MOUNTPOINT

umount $MOUNTPOINT

echo Utility diskette complete
```

6 FAQ

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

This is the most common problem experienced by first-time root disk users, and it can be quite tricky finding the cause. The basic problem is that there is very little that is exactly the same in all Linux systems, but to get a system to boot to the stage where it will talk to you requires several components to be present and correctly configured.

The first thing to be understood is that creating a root disk is usually an iterative process. Requirements change as the kernel changes, and different distributions arrange things in different ways. The normal boot process is for the kernel to execute /sbin/init, which in turn reads /etc/inittab to find out what to execute next... but anything is possible, so to understand things you must RTFM, including:

- the man pages for init, inittab , agetty or getty, login, ldconfig.
- the kernel documentation in /usr/src/linux/Documentation.
- the /etc/rc.x scripts and /etc/inittab in your system.

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.

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 really is to read the man pages for init and inittab and work out exactly what your existing system is doing when it boots.

The recommended procedure for investigating the problem where the system will not talk to you is as follows:

- 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 /root_disk/bin instead of /bin on your root diskette.
- check /sbin/init is present, and /etc/inittab has a system initialisation entry. This should contain a command of the form /etc/rc.x, to execute one of the /etc/rc scripts. The specific script in the inittab must exist.
- check what the the /etc/rc.x script specified in inittab does exist. All commands that it executes must be present on the disk.
- check that there is a /lib/libc.so and /lib/libtermcap.so, with the same links as appear 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 on your root disk.
- check that you have included /dev/null, /dev/zero, /dev/mem, /dev/ram and /dev/kmem devices.
- check your kernel configuration - support for all resources required up to login point must be built in, not modules. Also, ramdisk support must be included.
- check that your kernel root device and ramdisk settings are correct. Refer to the Advanced Bootdisk Creation section for details.

6.2 Q. How can I make a boot disk with a XXX driver?

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 rdev command:

```
rdev zImage
```

Rdev will then display the current root device in the kernel. If this is not the same as the root device you want, then 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".

6.3 Q. How do I update my boot floppy with a new kernel?

Just copy the kernel to your boot diskette using the dd command for a boot diskette without a filesystem, or the cp command for a boot/root disk. Refer to the section in this HOWTO titled "Boot" for details on creating a boot disk. The description applies equally to updating a kernel on a boot disk.

6.4 Q. How do I remove LILO so that I can use DOS to boot again?

This is not really a Bootdisk topic, but it is asked so often, so: the answer is, use the DOS command:

```
FDISK /MBR
```

MBR stands for Master Boot Record, and it 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.

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.

6.5 Q. How can I boot if I've lost my kernel AND my boot disk?

If you don't have a boot disk standing by, then 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:

```
0x01F8  Low byte of RAMDISK size
0x01F9  High byte of RAMDISK size
0x01FC  Root minor device number - see below
0X01FD  Root major device number - see below
```

The ramdisk size is the number of blocks of ramdisk to create. If you want to boot from a root diskette then set this to decimal 1440, which is 0x05A0, thus set offset 0x01F8 to 0xA0 and offset 0x01F9 to 0x05. This will allocate enough space for a 1.4Mb diskette.

Note that the meaning of the ramdisk size word changed in kernel version 1.3.48. This meaning is described in the section titled "Advanced Bootdisk Creation".

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 several distributions, including the SLS and Slackware 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, then you must write the file to a physical disk starting at the beginning of the disk.

6.6 Q. How can I make extra copies of boot/root diskettes?

It is never desirable to have just one set of rescue disks - 2 or 3 should be kept in case one 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 the device name of the diskette
        drive
and     filename the name of the file where you
        want to copy to
```

For example, to copy from /dev/fd0 to a temporary file called /tmp/diskette.copy, I would enter the command:

```
dd if=/dev/fd0 of=/tmp/diskette.copy
```

Omitting the "count" parameter, as we have done here, means that the whole diskette of 2880 (for a high-density) blocks will be copied.

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, then you can copy diskettes using a command like:

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

6.7 Q. How can I boot without typing in "ahaxxxx=nn,nn,nn" every time?

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 the LILO "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 = "aha152x=0x340,11,3,1"
```

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, cd 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.

6.8 Q. How can I create an oversize ramdisk filesystem?

For kernels 1.3.48+, it is best to create a compressed filesystem as described in Advanced Bootdisk Creation. If your kernel is earlier than this, you can either upgrade, or refer to version 2.0 or below of this HOWTO.

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

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.

6.10 Q. My kernel has ramdisk support, but initializes ramdisks of 0K

Where this occurs, a kernel message similar to:

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

appears as the kernel is booting. The size should be either the default of 4096K, or the size specified in kernel parameters ramdisk_size or ramdisk. If the size is 0K, it 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 included in some older distributions, and was put there to override any previous kernel setting. Since 1.3.48 it is irrelevant, because the ramdisk_size kernel parameter now sets the maximum ramdisk size, not the size allocated at boot time. No ramdisk memory is allocated at boot time.

The solution is to remove the LILO ramdisk parameter.

Note that if you attempt to use a ramdisk which has been set to 0K, then behaviour can be unpredictable, and can result in kernel panics.

7 References

In this section, vvv is used in package names in place of the version, to avoid referring here to specific versions. When retrieving a package, always get the latest version unless you have good reasons for not doing so.

7.1 LILO - 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: `tsx-11.mit.edu: /pub/linux/packages/lilo/lilo.vvv.tar.gz` also on sunsite and mirror sites.

7.2 Linux FAQ and HOWTOs

These are available from many sources. Look at the usenet newsgroups `news.answers` and `comp.os.linux.announce`.

Ftp from: `sunsite.unc.edu:/pub/Linux/docs`

- FAQ is in `/pub/linux/docs/faqs/linux-faq`
- HOWTOs are in `/pub/Linux/docs/HOWTO`

For WWW, start at the Linux documentation home page:

```
http://sunsite.unc.edu/mdw/linux.html
```

If desperate, send mail to:

```
mail-server@rtfm.mit.edu
```

with the word "help" in the message, then follow the mailed instructions.

Note: if you haven't read the Linux FAQ and related documents such as the Linux Installation HOWTO and the Linux Install Guide, then you should not be trying to build boot diskettes.

7.3 Ramdisk Usage

An excellent description of the how the new 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, similar to the section titled "Advanced Bootdisk Creation" of this HOWTO.

7.4 Rescue Packages

7.4.1 Bootkit

Written by Scott Burkett. Bootkit provides a flexible menu-driven framework for managing rescue disk creation and contents. It uses the Dialog package to provide nice menus, and a straight-forward directory tree to contain definitions of rescue disk contents. The package includes samples of the main files needed. The package aims to provide only the framework; it is up to the user to work out what to put on the disks and set up the config files accordingly. For those users who don't mind doing this, it is a good choice.

Ftp from: `sunsite.unc.edu: /pub/Linux/system/Recovery/Bootkit-vvv.tar.gz`

7.4.2 CatRescue

Written by Oleg Kibirev. This package concentrates on saving space on the rescue diskettes by extensive use of compression, and by implementing executables as shells scripts. The doco includes some tips on what to do in various disaster situations.

Ftp from: `gd.cs.csufresno.edu/pub/sun4bin/src/CatRescue100.tgz`

7.4.3 Rescue Shell Scripts

Written by Thomas Heiling. This contains shell scripts to produce boot and boot/root diskettes. It has some dependencies on specific versions of other software such as LILO, and so might need some effort to convert to your system, but it might be useful as a starting point if you wanted more comprehensive shell scripts than are provided in this document.

Ftp from: `sunsite.unc.edu:/pub/Linux/system/Recovery/rescue.tgz`

7.4.4 SAR - Search and Rescue

Written by Karel Kubat. SAR produces a rescue diskette, using several techniques to minimize the space required on the diskette. The manual includes a description of the Linux boot/login process.

Ftp from: `ftp.icce.rug.nl:/pub/unix/SAR-vvv.tar.gz`

The manual is available via WWW from:

`http://www.icce.rug.nl/karel/programs/SAR.html`

7.4.5 YARD

Written by Tom Fawcett. Yard produces customized rescue diskettes using the compressed ramdisk option of more recent kernels (1.3.48+). Yard was designed to automate most of the instructions in Section 4, above. In addition, Yard checks your file selections (loaders and libraries, and /etc/fstab, rc, and passwd) to make sure you've included everything needed to make a bootable rescue disk. Yard needs Perl 5 and LILO, and works with kernel 1.3.48+.

Ftp from: `sunsite.unc.edu:/pub/Linux/system/Recovery/yard-1.n.tar.gz`

A companion set of customized rescue specifications is available for use with Yard.

Ftp from: `sunsite.unc.edu:/pub/Linux/system/Recovery/yard-prefabs-n.tar.gz`

7.5 Slackware Distribution

Apart from being one of the more popular Linux distributions around, it is also a good place to get a generic kernel. It is available from almost everywhere, so there is little point in putting addresses here.