

Software RAID installation on Linux.

Under Red Hat 7.2 and later, there is an excellent implementation of software RAID, using the "raidtools" utilities.

It is well supported under Disk Druid, and the only problem with that is that Disk Druid makes some assumptions about your knowledge and how to implement these tools.

Even in the case of a hardware RAID controller, such as a Promise, Highpoint or 3Ware, we have found that better performance can be had using raidtools than by using the hardware RAID utilities in the device firmware.

For an example of this, there is Appendix C at the end of this document.

This is a performance shootout between a 3Ware 3w-7850 running 8 Western Digital Caviar 100 UDMA100 disks, using the hardware RAID manager in the 3Ware BIOS, versus raidtools using the same card as a simple IDE 8 port interface..

The benchmark was run on bonnie++ 1.02a

I am quite certain that you will find the results quite interesting, perhaps even astonishing.

Section 1: Red Hat 7.2 usage as quoted from Red Hat Installation documents.
<http://www.redhat.com/docs/manuals/linux/RHL-7.2-Manual/custom-guide/raid-intro.html>

Section 2: Manual Configuration - as quoted from "Software-RAID-HOWTO"
<http://www.linux.com/howto/Software-RAID-HOWTO.html>

Appendix A: EXAMPLE: Software RAID /etc/raidtab file

Appendix B: EXAMPLE: /etc/fstab file:

Appendix C: Shootout at the RAID Corral

Appendix D: 3Ware Red Hat 7.2 Linux installation notes

Section 1: Red Hat 7.2 usage as quoted from Red Hat Installation documents

First off, to read the official Red Hat docs on this, please refer to:

<http://www.redhat.com/docs/manuals/linux/RHL-7.2-Manual/custom-guide/raid-intro.html>

Bear in mind, that if you need to use extended and logical partition types, you will need to do this with `fdisk` first, however you can exit `fdisk` after doing what is needed there, and resume with Disk Druid.

Disk Druid does a nice job of helping you set up the RAID device, partition types, and so on, and creates the needed files for you.

Chapter 4. Redundant Array of Independent Disks (RAID)

What is RAID?

The basic idea behind RAID is to combine multiple small, inexpensive disk drives into an array to accomplish performance or redundancy goals not attainable with one large and expensive drive.

This array of drives will appear to the computer as a single logical storage unit or drive.

RAID is a method in which information is spread across several disks, using techniques such as:

Disk striping (RAID Level 0)

Disk mirroring (RAID level 1)

Disk striping with parity (RAID Levels 4 and 5) to achieve redundancy, lower latency and/or increase bandwidth for reading or writing to disks.

It also will maximize the ability to recover from hard disk crashes.

The underlying concept of RAID is that data may be distributed across each drive in the array in a consistent manner. To do this, the data must first be broken into consistently-sized "chunks" (often 32K or 64K in size, although different sizes can be used).

Each chunk is then written to a hard drive in RAID according to the RAID level used.

When the data is to be read, the process is reversed, giving the illusion that multiple drives are actually one large drive.

Who Should Use RAID?

Anyone who needs to keep large quantities of data on would benefit by using RAID technology.

The primary reasons to use RAID include:

Enhanced speed

Increased storage capacity using a single virtual disk

Lessening the impact of a disk failure

Hardware RAID versus Software RAID

There are two possible RAID approaches: Hardware RAID and Software RAID.

Hardware RAID

The hardware-based system manages the RAID subsystem independently from the host and presents to the host only a single disk per RAID array.

An example of a Hardware RAID device would be one that connects to a SCSI controller and presents the RAID arrays as a single SCSI drive. An external RAID system moves all RAID handling "intelligence" into a controller located in the external disk subsystem.

The whole subsystem is connected to the host via a normal SCSI controller and appears to the host as a single disk. These devices are often called "host independent"

RAID controllers also come in the form of cards that act like a SCSI controller to the operating system but handle all of the actual drive communications themselves. In these cases, you plug the drives into the RAID controller just like you would a SCSI controller, but then you add them to the RAID controller's configuration, and the operating system never knows the difference.

Software RAID

Software RAID implements the various RAID levels in the kernel disk (block device) code. It offers the cheapest possible solution, as expensive disk controller cards or hot-swap chassis are not required. Software RAID also works with cheaper IDE disks as well as SCSI disks. With today's fast CPUs, Software RAID performance can excel against Hardware RAID.

The MD driver in the Linux kernel is an example of a RAID solution that is completely hardware independent. The performance of a software-based array is dependent on the server CPU performance and load.

For information on configuring Software RAID in the Red Hat Linux installation program, refer to Chapter 5.

For those interested in learning more about what Software RAID has to offer here is a brief list of the most important features:

- Threaded rebuild process
- Fully kernel-based configuration
- Portability of arrays between Linux machines without reconstruction
- Backgrounded array reconstruction using idle system resources
- Hot-swappable drive support
- Automatic CPU detection to take advantage of certain CPU optimizations

Note:

A hot-swap chassis allows you to remove a hard drive without having to power-down your system.

For proper hotswap with IDE devices it is best to have disk trays that allow the device to be power off before removal or re-insertion. For SCSI devices it is best to use SCA (80 pin connector) drives, as they fully support hot-swap.

RAID Levels and Linear Support

RAID supports various configurations, including levels 0, 1, 4, 5, and linear.

These RAID types are defined as follows:

Level 0 RAID level 0, often called "striping," is a performance-oriented striped data mapping technique.

This means the data being written to the array is broken down into strips and written across the member disks of the array, allowing high I/O performance at low inherent cost but provides no redundancy.

The storage capacity of a level 0 array is equal to the total capacity of the member disks in a Hardware RAID or the total capacity of member partitions in a Software RAID.

Level 1 RAID level 1, or "mirroring," has been used longer than any other form of RAID.

Level 1 provides redundancy by writing identical data to each member disk of the array, leaving a "mirrored" copy on each disk.

Mirroring remains popular due to its simplicity and high level of data availability.

Level 1 operates with two or more disks that may use parallel access for high data-transfer rates when reading but more commonly operate independently to provide high I/O transaction rates.

Level 1 provides very good data reliability and improves performance for read-intensive applications but at a relatively high cost.

The storage capacity of the level 1 array is equal to the capacity of one of the mirrored hard disks in a Hardware RAID or one of the mirrored partitions in a Software RAID.

Level 4 Level 4 uses parity concentrated on a single disk drive to protect data.

It is better suited to transaction I/O rather than large file transfers.

Because the dedicated parity disk represents an inherent bottleneck, level 4 is seldom used without accompanying technologies such as write-back caching.

Although RAID level 4 is an option in some RAID partitioning schemes, it is not an option allowed in Red Hat Linux RAID installations.

The storage capacity of Hardware RAID level 4 is equal to the capacity of member disks, minus the capacity of one member disk.

The storage capacity of Software RAID level 4 is equal to the capacity of the member partitions, minus the size of one of the partitions if they are of equal size.

Level 5 This is the most common type of RAID.

By distributing parity across some or all of an array's member disk drives, RAID level 5 eliminates the write bottleneck inherent in level 4.

The only performance bottleneck is the parity calculation process.

With modern CPUs and Software RAID, that usually isn't a very big problem.

As with level 4, the result is asymmetrical performance, with reads substantially outperforming writes.

Level 5 is often used with write-back caching to reduce the asymmetry.

The storage capacity of Hardware RAID level 5 is equal to the capacity of member disks, minus the capacity of one member disk.

The storage capacity of Software RAID level 5 is equal to the capacity of the member partitions, minus the size of one of the partitions if they are of equal size.

Linear RAID Linear RAID is a simple grouping of drives to create a larger virtual drive.

In linear RAID, the chunks are allocated sequentially from one member drive, going to the next drive only when the first is completely filled.

This grouping provides no performance benefit, as I/O operations will not be split between member drives.

Linear RAID also offers no redundancy and, in fact, decreases reliability.

If any one member drive fails, the entire array cannot be used.

The capacity is the total of all member disks.

Notes

1) RAID level 1 comes at a high cost because you write the same information to all of the disks in the array, which wastes drive space.

For example, if you have RAID level 1 set up so that your root (/) partition exists on two 40G drives, you have 80G total but are only able to access 40G of that 80G.

The other 40G acts like a mirror of the first 40G.

2) Parity information is calculated based on the contents of the rest of the member disks in the array. This information can then be used to reconstruct data when one disk in the array fails.

The reconstructed data can then be used to satisfy I/O requests to the failed disk before it is replaced and to repopulate the failed disk after it has been replaced.

3) RAID level 4 takes up the same amount of space as RAID level 5, but level 5 has more advantages than level 4. For this reason, level 4 is not supported.

4) There are also the so-called RAID 0+1 and RAID10 variants.

These are methods of striping sets of RAID1 into mirror sets, or mirroring RAID0 sets.

In either case they involve the use of 4 or more drives, and, as you will see in appendix C, this does not make a lot of sense from either a performance or price point of view.

5) Do not bother creating swap space on RAID partitions, if possible. If you want to improve swap performance by using multiple swap partitions, mkswap and swapon already make provision for this internally. See "man swap" for more info on this feature.

Software RAID Configuration

<http://www.redhat.com/docs/manuals/linux/RHL-7.2-Manual/custom-guide/software-raid.html>

Software RAID can be configured during the graphical installation of Red Hat Linux or during `kickstart` installation. You can use `fdisk` or Disk Druid to create your RAID configuration, but these instructions will focus mainly on using Disk Druid to complete this task.

Before you can create a RAID device, you must first create RAID partitions, using the following step-by-step instructions.

Tip:

If you are using `fdisk` to create a RAID partition, remember that instead of creating a partition as type 83, which is Linux native, you must create the partition as type `fd` (Linux RAID). It is also advisable to make these partitions bootable with the `fdisk "a"` command. Also, for best performance, partitions within a given RAID array should span identical cylinders on drives.

Create a partition.

In Disk Druid, choose New to create a new partition.

Choose software RAID from the Filesystem Type pull-down menu.

You will not be able to enter a mount point.

You will be able to do that once you have created your RAID device).

For Allowable Drives, select the drive on which RAID will be created.

If you have multiple drives, all drives will be selected here and you must deselect those drives which will not have the RAID array on them.

Enter the size that you want the partition to be.

Select "Fill to maximum allowable size" if you want the partition to grow to fill all available space on the hard disk. If you make more than one partition growable, the partitions will share the available free space on the disk.

Select Force to be a primary partition if you want the partition to be a primary partition.

Select "Check for bad blocks?" if you want the installation program to check for bad blocks on the hard drive before formatting it.

Continue these steps to create as many partitions as needed for your RAID setup.

Notice that all the partitions do not have to be RAID partitions.

For example, you can configure only the `/home` partition as a software RAID device.

Once you have all of your partitions created as software RAID partitions, select the Make RAID button on the Disk Druid main partitioning screen.

A screen will appear, where you can make a RAID device.

Making a RAID Device

First, enter a mount point.

Next, choose the partition type for the partition.

Choose your RAID type. You can choose from RAID 0, RAID 1, and RAID 5

Please Note If you are making a RAID partition of /boot you must choose RAID level 1 and it must use one of the first two drives (IDE first, SCSI second).

If you are not creating a RAID partition of /boot, and you are making a RAID partition of /, it must be RAID level 1 and it must use one of the first two drives (IDE first, SCSI second).

Select which partitions will go into this RAID array and then click Ok.

A spare partition can be specified for RAID 1 and RAID 5.

If a software RAID partition fails, the spare will automatically be used as a replacement.

For each spare you want to specify, you must create an additional software RAID partition (in addition to the partitions for the RAID device).

In the previous step, select the partitions for the RAID device and the partition(s) for the spare(s).

Select the number of spares.

Select whether you want the partition formatted.

The RAID device will appear in the Drive Summary list.

At this point, you can continue with your installation process.

END OF RED HAT DOCUMENTATION SECTION

Section 2: RAID Manual Configuration

- as quoted from "Software-RAID-HOWTO"

<http://www.linux.com/howto/Software-RAID-HOWTO.html>

Firstly, it is important to know that the software RAID tools under Linux have undergone 2 complete revision versions. Here we are discussing the second one.

To quote:

"This HOWTO describes how to use Software RAID under Linux.

It addresses a specific version of the Software RAID layer, namely the 0.90 RAID layer made by IngoMolnar and others. This is the RAID layer that will be standard in Linux-2.4, and it is the version that is also used by Linux-2.2 kernels shipped from some vendors.

The 0.90 RAID support is available as patches to Linux-2.0 and Linux-2.2, and is by many considered far more stable than the older RAID support already in those kernels."

4.1 General setup

This is what you need for any of the RAID levels:

A kernel. Preferably a stable 2.2.X kernel, or the latest 2.4.X.

The RAID patches. There usually is a patch available for the recent kernels.

(If you found a 2.4 kernel, the patches are already in and you can forget about them)

The RAID tools

Patience, Pizza, and your favorite caffeinated beverage.

All this software can be found at <ftp://ftp.fi.kernel.org/pub/linux> The RAID tools and patches are in the daemons/raid/alpha subdirectory.

The kernels are found in the kernel subdirectory.

Patch the kernel, configure it to include RAID support for the level you want to use.

Compile it and install it. Then unpack, configure, compile and install the RAID tools.

Ok, so far so good.

If you reboot now, you should have a file called `/proc/mdstat`.

Remember it, that file is your friend!

See what it contains, by doing a `cat /proc/mdstat`.

It should tell you that you have the right RAID personality

(eg. RAID mode) registered, and that no RAID devices are currently active.

Create the partitions you want to include in your RAID set.

Now, let's go mode-specific.

4.2 Linear mode

Ok, so you have two or more partitions which are not necessarily the same size (but of course can be), which you want to append to each other.

Set up the `/etc/raidtab` file to describe your setup.

I set up a `raidtab` for two disks in linear mode, and the file looked like this:

```
raiddev /dev/md0
raid-level          linear
nr-raid-disks      2
chunk-size         32
persistent-superblock 1
device             /dev/sdb6
raid-disk          0
device             /dev/sdc5
raid-disk          1
```

Spare-disks are not supported here.

If a disk dies, the array dies with it.
There's no information to put on a spare disk.

You're probably wondering why we specify a `chunk-size` here when linear mode just appends the disks into one large array with no parallelism.

Well, you're completely right, it's odd.
Just put in some chunk size and don't worry about this any more.

Ok, let's create the array.

Run the command: `"mkraid /dev/md0"`
This will initialize your array, write the persistent superblocks, and start the array.

Have a look in `/proc/mdstat`.

You should see that the array is running.

Now, you can create a filesystem, just like you would on any other device, mount it, include it in `yourfstab` and so on.

4.3 RAID-0

You have two or more devices, of approximately the same size, and you want to combine their storage capacity and also combine their performance by accessing them in parallel.

Set up the `/etc/raidtab` file to describe your configuration.

An example `raidtab` looks like:

```
raiddev /dev/md0
raid-level          0
nr-raid-disks      2
persistent-superblock 1
chunk-size         4
device              /dev/sdb6
raid-disk          0
device              /dev/sdc5
raid-disk          1
```

As in Linear mode, spare disks are not supported here either.

RAID-0 has no redundancy, so when a disk dies, the array goes with it.

Again, you just run `"mkraid /dev/md0"` to initialize the array.

This should initialize the superblocks and start the raid device.

Have a look in `/proc/mdstat` to see what's going on.

You should see that your device is now running.

`/dev/md0` is now ready to be formatted, mounted, used and abused.

4.4 RAID-1

You have two devices of approximately same size, and you want the two to be mirrors of each other.

Eventually you have more devices, which you want to keep as stand-by spare disks, that will automatically become a part of the mirror if one of the active devices break.

Set up the `/etc/raidtab` file like this:

```
raiddev /dev/md0
raid-level          1
nr-raid-disks      2
nr-spare-disks     0
chunk-size         4
persistent-superblock 1
device              /dev/sdb6
raid-disk           0
device              /dev/sdc5
raid-disk           1
```

If you have spare disks, you can add them to the end of the device specification like:

```
device              /dev/sdd5
spare-disk          0
```

Remember to set the `nr-spare-disks` entry correspondingly.

Ok, now we're all set to start initializing the RAID.

The mirror must be constructed, eg. the contents (however unimportant now, since the device is still not formatted) of the two devices must be synchronized.

Issue the `"mkraid /dev/md0"` command to begin the mirror initialization.

Check out the `/proc/mdstat` file.

It should tell you that the `/dev/md0` device has been started, that the mirror is being reconstructed, and an ETA of the completion of the reconstruction.

Reconstruction is done using idle I/O bandwidth.

So, your system should still be fairly responsive, although your disk LEDs should be glowing nicely.

The reconstruction process is transparent, so you can actually use the device even though the mirror is currently under reconstruction.

Try formatting the device, while the reconstruction is running. It will work.

Also you can mount it and use it while reconstruction is running.

Of Course, if the wrong disk breaks while the reconstruction is running you're out of luck.

4.5 RAID-4

The setup below is my best guess, not something I have actually had up running.

In my opinion, there is little to recommend RAID4, when we can use RAID5 instead.

RAID4 puts ALL of the parity data onto one disk, so that disk is a performance bottleneck, and if it fails we lose all redundancy.

However, if you have three or more devices of roughly the same size, one device is significantly faster than the other devices, and you want to combine them all into one larger device, still maintaining some redundancy information, perhaps RAID4 is what you want..

If so read on..

Set up the /etc/raidtab file like this:

```
raiddev /dev/md0
raid-level          4
nr-raid-disks      4
nr-spare-disks     0
persistent-superblock 1
chunk-size         32
device             /dev/sdb1
raid-disk          0
device             /dev/sdc1
raid-disk          1
device             /dev/sdd1
raid-disk          2
device             /dev/sde1
raid-disk          3
```

If we had any spare disks, they would be inserted in a similar way, following the raid-disk specifications;

```
device             /dev/sdf1
spare-disk         0
```

as usual.

Your array can be initialized with the "mkraid /dev/md0" command as usual.

You should see the section on special options for mke2fs before formatting the device.

4.6 RAID-5

You have three or more devices of roughly the same size you want to combine them into a larger device, but still to maintain a degree of redundancy for data safety.

Eventually you have a number of devices to use as spare disks, that will not take part in the array before another device fails.

If you use N devices where the smallest has size S, the size of the entire array will be $(N-1)*S$. This "missing" space (the "-1") is used for parity (redundancy) information.

Thus, if any disk fails, all data stay intact. But if two disks fail, all data is lost.

Set up the `/etc/raidtab` file like this:

```
raiddev /dev/md0
    raid-level          5
    nr-raid-disks      7
    nr-spare-disks     0
    persistent-superblock 1
    parity-algorithm    left-symmetric
    chunk-size         32
    device              /dev/sda3
    raid-disk           0
    device              /dev/sdb1
    raid-disk           1
    device              /dev/sdc1
    raid-disk           2
    device              /dev/sdd1
    raid-disk           3
    device              /dev/sde1
    raid-disk           4
    device              /dev/sdf1
    raid-disk           5
    device              /dev/sdg1
    raid-disk           6
```

If we had any spare disks, they would be inserted in a similar way, following the raid-disk specifications

```
device          /dev/sdh1
spare-disk      0
```

And so on..

A chunk size of 32 KB is a good default for many general purpose filesystems of this size.

The array on which the above `raidtab` is used, is a 7 times 6 GB = 36 GB (remember the $(n-1)*s = (7-1)*6 = 36$) device.

It holds an ext2 filesystem with a 4 KB block size.

You could go higher with both array chunk-size and filesystem block-size if your filesystem is either much larger, or just holds very large files.

Ok, enough talking. You set up the `raidtab`, so let's see if it works.

Run the `"mkraid /dev/md0"` command, and see what happens. Hopefully your disks start working like mad, as they begin the reconstruction of your array.

Have a look in `/proc/mdstat` to see what's going on.

If the device was successfully created, the reconstruction process has now begun.

Your array is not consistent until this reconstruction phase has completed.

However, the array is fully functional (except for the handling of device failures of course), and you can format it and use it even while it is reconstructing.

See the section on special options for `mke2fs` before formatting the array.

Ok, now when you have your RAID device running, you can always stop it or re-start it using the `"raidstop /dev/md0"` or `"raidstart /dev/md0"` commands.

Instead of putting these into `init`-files and rebooting a zillion times to make that work, read on, and get autodetection running.

4.7 The Persistent Superblock

Back in "The Good Old Days" (TM), the `raidtools` would read your `/etc/raidtab` file, and then initialize the array.

However, this would require that the filesystem on which `/etc/raidtab` resided was mounted.

This is unfortunate if you want to boot on a RAID. Also, the old approach led to complications when mounting filesystems on RAID devices.

They could not be put in the `/etc/fstab` file as usual, but would have to be mounted from the `init`-scripts. The persistent superblocks solve these problems.

When an array is initialized with the `persistent-superblock` option in the `/etc/raidtab` file, a special superblock is written in the beginning of all disks participating in the array.

This allows the kernel to read the configuration of RAID devices directly from the disks involved, instead of reading from some configuration file that may not be available at all times.

You should, however, still maintain a consistent `/etc/raidtab` file, since you may need this file for later reconstruction of the array.

The persistent superblock is mandatory if you want auto-detection of your RAID devices upon system boot.

This is described in the Autodetection section.

4.8 Chunk Sizes

The chunk-size deserves an explanation.

You can never write completely parallel to a set of disks. If you had two disks and wanted to write a byte, you would have to write four bits on each disk, actually, every second bit would go to disk 0 and the others to disk 1.

Hardware just doesn't support that.

Instead, we choose some chunk-size, which we define as the smallest "atomic" mass of data that can be written to the devices.

A write of 16 KB with a chunk size of 4 KB, will cause the first and the third 4 KB chunks to be written to the first disk, and the second and fourth chunks to be written to the second disk, in the RAID-0 case with two disks.

Thus, for large writes, you may see lower overhead by having fairly large chunks, whereas arrays that are primarily holding small files may benefit more from a smaller chunk size.

Chunk sizes must be specified for all RAID levels, including linear mode.

However, the chunk-size does not make any difference for linear mode.

For optimal performance, you should experiment with the value, as well as with the block-size of the filesystem you put on the array.

The argument to the chunk-size option in `/etc/raidtab` specifies the chunk-size in kilobytes. So "4" means "4 KB".

RAID-0

Data is written "almost" in parallel to the disks in the array.

Actually, chunk-size bytes are written to each disk, serially.

If you specify a 4 KB chunk size, and write 16 KB to an array of three disks, the RAID system will write 4 KB to disks 0, 1 and 2, in parallel, then the remaining 4 KB to disk 0.

A 32 KB chunk-size is a reasonable starting point for most arrays.

But the optimal value depends very much on the number of drives involved, the content of the file system you put on it, and many other factors.

Experiment with it, to get the best performance.

RAID-1

For writes, the chunk-size doesn't affect the array, since all data must be written to all disks no matter what.

For reads however, the chunk-size specifies how much data to read serially from the participating disks.

Since all active disks in the array contain the same information, reads can be done in a parallel RAID-0 like manner.

RAID-4

When a write is done on a RAID-4 array, the parity information must be updated on the parity disk as well.

The chunk-size is the size of the parity blocks.

If one byte is written to a RAID-4 array, then chunk-size bytes will be read from the N-1 disks, the parity information will be calculated, and chunk-size bytes written to the parity disk.

The chunk-size affects read performance in the same way as in RAID-0 since reads from RAID-4 are done in the same way.

RAID-5

On RAID-5 the chunk-size has exactly the same meaning as in RAID-4.

A reasonable chunk-size for RAID-5 is 128 KB, but as always, you may want to experiment with this.

Also see the section on special options for `mke2fs`. This affects RAID-5 performance significantly

4.9 Options for mke2fs

There is a special option available when formatting RAID-4 or -5 devices with mke2fs.

The `-R stride=nn` option will allow mke2fs to better place different ext2 specific data-structures in an intelligent way on the RAID device.

If the chunk-size is 32 KB, it means, that 32 KB of consecutive data will reside on one disk.

If we want to build an ext2 filesystem with 4 KB block-size, we realize that there will be eight filesystem blocks in one array chunk.

We can pass this information on the mke2fs utility, when creating the filesystem:

```
"mke2fs -b 4096 -R stride=8 /dev/md0 RAID-{4,5}"
```

For ext2fs with journaling (ext3) we simply add the "-j" option. This is STRONGLY recommended!! Like this:

```
"mke2fs -b 4096 -j -R stride=32 /dev/md0 RAID-5"
```

Performance is severely influenced by this option.

I am unsure how the stride option will affect other RAID levels. If anyone has information on this, please send it in my direction.

The ext2fs blocksize severely influences the performance of the filesystem.

You should always use 4KB block size on any filesystem larger than a few hundred megabytes, unless you store a very large number of very small files on it.

4.10 Autodetection

Autodetection allows the RAID devices to be automatically recognized by the kernel at boot-time, right after the ordinary partition detection is done.

This requires several things:

- 1) You need autodetection support in the kernel.
Check this!
- 2) You must have created the RAID devices using persistent superblock
- 3) The partition-types of the devices used in the RAID must be set to 0xFD (use fdisk and set the type to `fd`)

NOTE: Be sure that your RAID is NOT RUNNING before changing the partition types. Use `raidstop /dev/md0` to stop the device.

If you set up 1, 2 and 3 from above, autodetection should be set up. Try rebooting.

When the system comes up, cat'ing `/proc/mdstat` should tell you that your RAID is running.

During boot, you could see messages similar to these:

```
Oct 22 00:51:59 malthe kernel: SCSI device sdg: hdwr sector= 512
bytes. Sectors= 12657717 [6180 MB] [6.2 GB]
Oct 22 00:51:59 malthe kernel: Partition check:
Oct 22 00:51:59 malthe kernel: sda: sda1 sda2 sda3 sda4
Oct 22 00:51:59 malthe kernel: sdb: sdb1 sdb2
Oct 22 00:51:59 malthe kernel: sdc: sdc1 sdc2
Oct 22 00:51:59 malthe kernel: sdd: sdd1 sdd2
Oct 22 00:51:59 malthe kernel: sde: sde1 sde2
Oct 22 00:51:59 malthe kernel: sdf: sdf1 sdf2
Oct 22 00:51:59 malthe kernel: sdg: sdg1 sdg2
Oct 22 00:51:59 malthe kernel: autodetecting RAID arrays
Oct 22 00:51:59 malthe kernel: (read) sdb1's sb offset: 6199872
Oct 22 00:51:59 malthe kernel: bind<sdb1,1>
Oct 22 00:51:59 malthe kernel: (read) sdc1's sb offset: 6199872
Oct 22 00:51:59 malthe kernel: bind<sdcl,2>
Oct 22 00:51:59 malthe kernel: (read) sdd1's sb offset: 6199872
Oct 22 00:51:59 malthe kernel: bind<sdd1,3>
Oct 22 00:51:59 malthe kernel: (read) sde1's sb offset: 6199872
Oct 22 00:51:59 malthe kernel: bind<sde1,4>
Oct 22 00:51:59 malthe kernel: (read) sdf1's sb offset: 6205376
Oct 22 00:51:59 malthe kernel: bind<sdf1,5>
Oct 22 00:51:59 malthe kernel: (read) sdg1's sb offset: 6205376
Oct 22 00:51:59 malthe kernel: bind<sdg1,6>
Oct 22 00:51:59 malthe kernel: autorunning md0
Oct 22 00:51:59 malthe kernel: running: <sdg1><sdf1><sde1><sdd1><sdcl><sdb1>
Oct 22 00:51:59 malthe kernel: now!
Oct 22 00:51:59 malthe kernel: md: md0: raid array is not clean --
starting background reconstruction
```

This is the output from the autodetection of a RAID-5 array that was not cleanly shut down (e.g. the machine crashed).

Reconstruction is automatically initiated.

Mounting this device is perfectly safe, since reconstruction is transparent and all data are consistent (it's only the parity information that is inconsistent - but that isn't needed until a device fails).

Autostarted devices are also automatically stopped at shutdown.

Don't worry about init scripts. Just use the /dev/md devices as any other /dev/sd or /dev/hd devices. Yes, it really is that easy.

You may want to look in your init-scripts for any raidstart/raidstop commands.

These are often found in the standard RedHat init scripts. They are used for old-style RAID, and has no use in new-style RAID with autodetection.

Just remove the lines, and everything will be just fine.

4.11 Booting on RAID

There are several ways to set up a system that mounts its root filesystem on a RAID device. At the moment, only the graphical install of Red Hat Linux 6.1 and later allow direct installation to a RAID device. So most likely you're in for a little tweaking if you want this, but it is indeed possible.

The latest official lilo distribution (Version 21) doesn't handle RAID devices and thus the kernel cannot be loaded at boot-time from a RAID device.

If you use this version, your /boot filesystem will have to reside on a non-RAID device.

A way to ensure that your system boots no matter what is, to create similar /boot partitions on all drives in your RAID, that way the BIOS can always load data from.
eg. the first drive available.

This requires that you do not boot with a failed disk in your system.
With redhat 6.1 a patch to lilo 21 has become available that can handle /boot on RAID-1.

Note that it doesn't work for any other level, RAID-1 (mirroring) is the only supported RAID level.

This patch (lilo.raid1) can be found in:

dist/redhat-6.1/SRPMS/SRPMS/lilo-0.21-10.src.rpm

on any redhat mirror.

The patched version of LILO will accept `boot=/dev/md0` in `lilo.conf` and will make each disk in the mirror bootable.

Another way of ensuring that your system can always boot is, to create a boot floppy when all the setup is done. If the disk on which the /boot filesystem resides dies, you can always boot from the floppy.

4.12 Root filesystem on RAID

In order to have a system booting on RAID, the rootfilesystem (/) must be mounted on a RAID device.

Two methods for achieving this is supplied below.

Because none of the current distributions (that I know of at least) support installing on a RAID device, the methods assume that you install on a normal partition, and then - when the installation is complete - move the contents of your non-RAID rootfilesystem onto a new RAID device.

Method 1

This method assumes you have a spare disk you can install the system on, which is not part of the RAID you will be configuring.

First, install a normal system on your extra disk.

Get the kernel you plan on running, get the raid-patches and the tools, and make your system boot with this new RAID-aware kernel. Make sure that RAID-support is in the kernel, and is not loaded as modules.

Ok, now you should configure and create the RAID you plan to use for the rootfilesystem. This is standard procedure, as described elsewhere in this document.

Just to make sure everything's fine, try rebooting the system to see if the new RAID comes up on boot. It should.

Put a filesystem on the new array (using mke2fs), and mount it under /mnt/newroot

Now, copy the contents of your current root-filesystem (the spare disk) to the new root-filesystem (the array).

There are lots of ways to do this, one of them is:

```
"cd / find . -xdev | cpio -pm /mnt/newroot"
```

You should modify the /mnt/newroot/etc/fstab file to use the correct device (the /dev/md? root device) for the root filesystem.

Now, unmount the current /boot filesystem, and mount the boot device on /mnt/newroot/boot instead. This is required for LILO to run successfully in the next step.

Update /mnt/newroot/etc/lilo.conf to point to the right devices.

The boot device must still be a regular disk (non-RAID device), but the root device should point to your new RAID.

When done, run:

```
"lilo -r /mnt/newroot"
```

This LILO run should complete with no errors.

Reboot the system, and watch everything come up as expected :)

If you're doing this with IDE disks, be sure to tell your BIOS that all disks are "auto-detect" types, so that the BIOS will allow your machine to boot even when a disk is missing.

Method 2

This method requires that you use `araidtools/patch` that includes the `failed-disk` directive.

This will be the `tools/patch` for all kernels from 2.2.10 and later.

You can only use this method on RAID levels 1 and above.

The idea is to install a system on a disk which is purposely marked as failed in the RAID, then copy the system to the RAID which will be running in degraded mode, and finally making the RAID use the no-longer needed "install-disk", zapping the old installation but making the RAID run in non-degraded mode.

First, install a normal system on one disk (that will later become part of your RAID).

It is important that this disk (or partition) is not the smallest one.

If it is, it will not be possible to add it to the RAID later on!

Then, get the kernel, the patches, the tools etc. etc. You know the drill.

Make your system boot with a new kernel that has the RAID support you need, compiled into the kernel.

Now, set up the RAID with your current root-device as the `failed-disk` in the `theraidtab` file.

Don't put the `failed-disk` as the first disk in the `theraidtab`, that will give you problems with starting the RAID. Create the RAID, and put a filesystem on it.

Try rebooting and see if the RAID comes up as it should.

Copy the system files, and reconfigure the system to use the RAID as root-device, as described in the previous section.

When your system successfully boots from the RAID, you can modify the `theraidtab` file to include the previously failed-disk as a normal raid-disk.

Now, "raidhotadd" the disk to your RAID.

You should now have a system that can boot from a non-degraded RAID.

4.13 Making the system boot on RAID

For the kernel to be able to mount the rootfilesystem, all support for the device on which the rootfilesystem resides, must be present in the kernel.

Therefore, in order to mount the rootfilesystem on a RAID device, the kernel must have RAID support.

The normal way of ensuring that the kernel can see the RAID device is to simply compile a kernel with all necessary RAID support compiled in.

Make sure that you compile the RAID support into the kernel, and not asloadable modules.

The kernel cannot load a module (from the rootfilesystem) before the root filesystem is mounted.

However, since RedHat-6.0 ships with a kernel that has new-style RAID support as modules, I here describe how one can use the standard RedHat-6.0 kernel and still have the system boot on RAID.

Booting with RAID as module

You will have to instruct LILO to use a RAM-disk in order to achieve this.

Use the mkinitrd command to create a ramdisk containing all kernelmodules needed to mount the root partition.

This can be done as:

```
"mkinitrd --with=<module> <ramdisk name> <kernel>"
```

For example:

```
"mkinitrd --with=raid5 raid-ramdisk 2.2.5-22"
```

This will ensure that the specified RAID module is present at boot-time, for the kernel to use when mounting the root device.

4.14 Pitfalls

Never NEVER ever! re-partition disks that are part of a running RAID.

If you must alter the partition table on a disk which is a part of a RAID, stop the array first, then repartition.

It is easy to put too many disks on a bus.

A normal Fast-Wide SCSI bus can sustain 10 MB/s which is less than many disks can do alone today.

Putting six such disks on the bus will NOT give you the expected performance boost.

More SCSI controllers will only give you extra performance, if the SCSI busses are nearly maxed out by the disks on them.

You will not see a performance improvement from using two 2940s with two old SCSI disks, instead of just running the two disks on one controller.

If you forget the persistent-superblock option, your array may not start up willingly after it has been stopped.

Just re-create the array with the option set correctly in the raidtab.

If a RAID-5 fails to reconstruct after a disk was removed and re-inserted, this may be because of the ordering of the devices in the raidtab.

Try moving the first "device ..." and "raid-disk ..." pair to the bottom of the array description in the raidtab file.

Most of the "error reports" we see on linux-kernel, are from people who somehow failed to use the right RAID-patch with the right version of the raidtools.

Make sure that if you're running 0.90 RAID, you're using the raidtools for it!

5. Testing

If you plan to use RAID to get fault-tolerance, you may also want to test your setup, to see if it really works.

Now, how does one simulate a disk failure ?

The short story is, that you can't, except perhaps by putting a fire axethru the drive you want to ``simulate" the fault on. You can never know what will happen if a drive dies.

It may electrically take the bus it's attached to with it, rendering all drives on that bus inaccessible. I've never heard of that happening though.

The drive may also just report a read/write fault to the SCSI/IDE layer which in turn makes the RAID layer handle this situation gracefully.

This is fortunately the way things often go.

5.1 Simulating a drive failure

If you want to simulate a drive failure, then plug out the drive.

You should do this with the power off.

If you are interested in testing whether your data can survive with a disk less than the usual number, there is no point in being a "hot-plug cowboy" here. Take the system down, unplug the disk, and boot it up again.

Look in the syslog, and look at /proc/mdstat to see how the RAID is doing.

Did it work ?

Remember, that you must be running RAID-{1,4,5} for your array to be able to survive a disk failure.

Linear- or RAID-0 WILL fail completely when a device is missing!

When you've re-connected the disk again (with the power off, of course remember), you can add the ``new" device to the RAID again, with the "raidhotadd" command.

5.2 Simulating data corruption

RAID (be it hardware- or software-), assumes that if a write to a disk doesn't return an error, then the write was successful.

Therefore, if your disk corrupts data without returning an error, your data will become corrupted. This is of course very unlikely to happen, but it is possible, and it would result in a corrupt filesystem.

RAID cannot and is not supposed to guard against data corruption on the media.

Therefore, it doesn't make any sense either, to purposely corrupt data (using dd for example) on a disk to see how the RAID system will handle that.

It is most likely (unless you corrupt the RAID superblock) that the RAID layer will never find out about the corruption, but your filesystem on the RAID device will be corrupted.

This is the way things are supposed to work.

RAID is not a guarantee for data integrity, it just allows you to keep your data if a disk dies (that is, with RAID levels above or equal one, of course).

6. Reconstruction

If you've read the rest of this HOWTO, you should already have a pretty good idea about what reconstruction of a degraded RAID involves.

I'll summarize:

Power down the system
Replace the failed disk
Power up the system once again.
Use `raidhotadd /dev/mdX /dev/sdX` to re-insert the disk in the array
Have coffee while you watch the automatic reconstruction running

And that's it.

Well, it usually is, unless you're unlucky and your RAID has been rendered unusable because more disks than the ones covered by redundancy failed.

This can actually happen if a number of disks reside on the samebus, and one disk takes the bus with it as it crashes. The other disks, however fine, will be unreachable to the RAID layer, because the bus is down, and they will be marked as faulty.

On a RAID-5 where you can spare one disk, losing two or more disks can be fatal.

The following section is the explanation that MartinBene gave to me, and describes a possible recovery from the scary scenario outlined above.

It involves using the failed-disk directive in your `/etc/raidtab`, so this will only work on kernels 2.2.10 and later.

6.1 Recovery from a multiple disk failure

The scenario is (pick one):

- *A controller dies and takes two disks offline at the same time
- *All disks on one SCSI bus can no longer be reached if a disk dies
- *A cable comes loose...

In short: quite often you get a temporary failure of several disks at once afterwards the RAID superblocks are out of sync and you can no longer `init` your RAID array.

One thing left: rewrite the RAID superblocks by `"mkraid --force"`

To get this to work, you'll need to have an up-to-date `/etc/raidtab` -

IF it doesn't EXACTLY match the devices and ordering of the original disks this will NOT work!

Look at the `sylog` produced by trying to start the array, you'll see the event count for each superblock; usually it's best to leave out the disk with the lowest event count, i.e the oldest one.

If you `mkraid` without "failed-disk", the recovery thread will kick in immediately and start rebuilding the parity blocks - not necessarily what you want at that moment.

With "failed-disk" you can specify exactly which disks you want to be active and perhaps try different combinations for best results.

BTW, only mount the filesystem as read-only while trying this out...

This has been successfully used by at least two guys I've been in contact with.

7. Credits

The following people contributed to the creation of this documentation:

Ingo Molnar
Jim Warren
Louis Mandelstam
Allan Noah
Yasunori Taniike
Martin Bene
Bennett Todd
Maurice Hilarius
The Linux-RAID mailing list people
The ones I forgot, sorry :)

Please submit corrections, suggestions etc. to the author. It's the only way this HOWTO can improve.

Appendix A:

EXAMPLE: Software RAID /etc/raidtab example:

An 8 disk RAID on an 8 port 3Ware controller, which is mounted on thefstab example as "/data".

A 2 disk RAID on an the motherboard IDE channels using 2 IDE disks, with all partitions as RAID1 mirrors:

These are mounted on the fstab example as:

```
/
/boot
/tmp
/var
/home
/usr
/usr/local

raiddev          /dev/md7
raid-level       5
nr-raid-disks   8
chunk-size      64k
persistent-superblock 1
nr-spare-disks  0
  device        /dev/sda1
  raid-disk     0
  device        /dev/sdb1
  raid-disk     1
  device        /dev/sdc1
  raid-disk     2
  device        /dev/sdd1
  raid-disk     3
  device        /dev/sde1
  raid-disk     4
  device        /dev/sdf1
  raid-disk     5
  device        /dev/sdg1
  raid-disk     6
  device        /dev/sdh1
  raid-disk     7

raiddev          /dev/md3
raid-level       1
nr-raid-disks   2
chunk-size      64k
persistent-superblock 1
nr-spare-disks  0
  device        /dev/hda9
  raid-disk     0
  device        /dev/hdb5
  raid-disk     1
```

continued..

```
raiddev          /dev/md5
raid-level       1
nr-raid-disks   2
chunk-size      64k
persistent-superblock 1
nr-spare-disks  0
  device        /dev/hda1
  raid-disk     0
  device        /dev/hda2
  raid-disk     1
```

```
raiddev          /dev/md6
raid-level       1
nr-raid-disks   2
chunk-size      64k
persistent-superblock 1
nr-spare-disks  0
  device        /dev/hda10
  raid-disk     0
  device        /dev/hdb8
  raid-disk     1
```

```
raiddev          /dev/md4
raid-level       1
nr-raid-disks   2
chunk-size      64k
persistent-superblock 1
nr-spare-disks  0
  device        /dev/hda8
  raid-disk     0
  device        /dev/hdb6
  raid-disk     1
```

```
raiddev          /dev/md0
raid-level       1
nr-raid-disks   2
chunk-size      64k
persistent-superblock 1
nr-spare-disks  0
  device        /dev/hda3
  raid-disk     0
  device        /dev/hdb1
  raid-disk     1
```

```
raiddev          /dev/md1
raid-level       1
nr-raid-disks   2
chunk-size      64k
persistent-superblock 1
nr-spare-disks  0
  device        /dev/hda5
  raid-disk     0
  device        /dev/hdb2
  raid-disk     1
```

continued..

```

raiddev          /dev/md2
raid-level       1
nr-raid-disks   2
chunk-size      64k
persistent-superblock 1
nr-spare-disks  0
  device        /dev/hda7
  raid-disk     0
  device        /dev/hdb7
  raid-disk     1

```

Appendix B:

EXAMPLE: /etc/fstab file:

```

LABEL=/          /          ext3    defaults        1 1
LABEL=/boot     /boot      ext3    defaults        1 2
none            /dev/pts   devpts  gid=5,mode=620 0 0
LABEL=/home     /home      ext3    defaults        1 2
none            /proc      proc    defaults        0 0
none            /dev/shm   tmpfs   defaults        0 0
LABEL=/tmp      /tmp       ext3    defaults        1 2
LABEL=/usr      /usr       ext3    defaults        1 2
LABEL=/usr/local /usr/local ext3    defaults        1 2
LABEL=/var      /var       ext3    defaults        1 2
/dev/md7        /data      ext3    defaults        1 2
/dev/hda6       swap       swap    defaults        0 0
/dev/hdb3       swap       swap    defaults        0 0
/dev/cdrom      /mnt/cdrom iso9660 noauto,owner,kudzu,ro 0 0
/dev/fd0        /mnt/floppy auto     noauto,owner,kudzu 0 0

```

Appendix C: Shootout at the RAID Corral

Testbed - Hard Data MP+ server / array

Motherboard: Tyan Tiger S2460MP

Processors: Dual AthlonMP 1800+ (1.53GHz Palomino)

RAM: 2 x 1024MB Kentron PC2100, ECC, Registered DDR

Video: Matrox G450-32MB

Ethernet: 3COM 3C905C

Disk controller: 3Ware 7850 (8 channel IDE RAID controller)

Hard Disks:

2 x Maxtor 740L (UDMA133, 20GB, 7200rpm) for HDA and HDB on motherboard IDE interface

Filesystems all configured as RAID1, ext3

8 x Western Digita Caviar 100 UDMA100, 7200rpm on 3Ware interface

Configuration is as per the above example for fstab and raidtab

Filesystem as a single RAID5, no spares, ext3. Mounted on /data

In the first example we installed the hard disks using the 3Ware BIOS interface to build a single RAID5 set, with no hot spares. Capacity of roughly 640GB

In the second example we installed the hard disks using raidtools, and the disks used the 3Ware card as a hard disk interface only.

Again, we built a single RAID5 set, with no hot spares. Capacity of roughly 640GB

Bonnie++ 1.02a

Command line:

```
./bonnie++ -d /data -s 4096 -r 2048M -u root > bonoutX
```

Command line explanation:

-d /data is the directory the RAID5 set is mounted to

-s 4096 is telling bonnie++ to use a block size of 4096MB. This is double the installed system RAM, and will prevent caching artifacts and false performance due to OS or disk caching.

-r 2048M is telling bonnie++ how much RAM is in the system.

-u root is telling bonnie that we are running as root

bonoutX is our output file.

Appendix C – Benchmark results

Results for the first run, using the 3Ware "native" RAID5:

```
Version 1.02a      -----Sequential Output----- --Sequential Input- --Random-
                  -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Machine           Size K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP /sec %CP
dynamic-181.hard 4G  5182  72 14542  25 13625  18 18667  94 67454  33 384.2  1
                  -----Sequential Create----- -----Random Create-----
                  -Create-- --Read--- -Delete-- -Create-- --Read--- -Delete--
files            /sec %CP /sec %CP /sec %CP /sec %CP /sec %CP /sec %CP
                  16  1296  97 +++++ + 16413  93  1203  99 +++++ + 4791  97
```

Results for the second run, using the raidtools /dev/md7 software RAID5:

```
Version 1.02a      -----Sequential Output----- --Sequential Input- --Random-
                  -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Machine           Size K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP /sec %CP
dynamic-181.hard 4G 17132  99 78071  76 44695  61 19854  99 127400  89 346.1  4
                  -----Sequential Create----- -----Random Create-----
                  -Create-- --Read--- -Delete-- -Create-- --Read--- -Delete--
files            /sec %CP /sec %CP /sec %CP /sec %CP /sec %CP /sec %CP
                  16  1334  96 +++++ + 16413  93  1383  99 +++++ + 2874  99
```

Write performance:

On sequential character the software RAID was 230% faster (5182 versus 17,132)

On sequential block the software RAID was 436% faster (14542 versus 78071)

On sequential rewrites the software RAID was 228% faster (13625 versus 44695)

Read performance:

On sequential character the software RAID was 6.3% faster (18667 versus 19854)

On sequential block the software RAID was 88.8% faster (67454 versus 127400)

I/O performance on seek commands was slightly faster with the hardware RAID, at 384 per sec, versus 346 for the software RAID.

On the file creation and deletion tests the results were fairly close.

The software RAID was a small amount ahead on most tests, while the hardware RAID was much faster at random file deletion.

CPU use was generally higher using the software RAID.

This CPU usage pretty well mirrors the actual data throughput in each case.

Appendix D: 3Ware Red Hat 7.2 Linux installation notes

3Ware driver installation on Red Hat Linux 7.2

First the good news:

Although 3Ware claims that their driver software is unsupported in Red Hat Linux 7.2, the driver is actually in the current distribution.

If you do not plan to boot from drives attached to the 3Ware card, you should simply do a normal Red Hat 7.2 installation, then install the updates, and you will have a 3Ware driver that is newer than the one on the 3Ware installation disks and what is on their website!

Now the bad news:

If you are planning to boot from a device attached to a 3Ware card, you need to do some steps to help the Red Hat Anaconda installer properly deal with this installation.

Firstly, there is a patch for Anaconda that is needed.

See:

<http://www.redhat.com/support/errata/RHBA-2001-131.html>

You will need to download the file:

<ftp://updates.redhat.com/7.2/en/os/images/i386/update-disk-20011009.img>

Download the update disk image, then create an update diskette. This can be done by using the same procedure used to create a boot diskette.

The choices are:

From Linux:

```
dd if= update-disk-20011009.img of=/dev/fd0 bs=1440k
```

From DOS:

```
C:\>
```

```
C:\> cd \dosutils
```

```
D:\dosutils> rawrite
```

```
Enter disk image source file name: updat~1.img
```

```
Enter target diskette drive: a:
```

```
Please insert a formatted diskette into drive A: and
```

```
press --ENTER-- : [Enter]
```

Note that rawrite is provided on the Red Hat distribution CROM set.

When booting into the anaconda installation program, type "linux updates" at the boot prompt, followed by any other installation options which are required (such as "expert" or "text" or "ks").

The installation process will prompt you to insert the updates disk when it is required, and the install will then proceed normally.

3Ware 3DM installation on Linux.

3Ware includes a script to install their web browser based 3dm utilities on Linux. If your installation is not a completely original and stock SuSE or Red Hat 7.1 or earlier version, it will fail, and not complete.

The main reason is that they obviously never actually tested this script on much of anything

Here is the whole "fix":

```
--- install.3dm.orig      Wed Aug  1 08:00:00 2001
+++ install.3dm         Fri Nov 23 11:47:17 2001
@@ -390,7 +390,7 @@
  /bin/echo "Red Hat 6.2, 7.0 and 7.1"
  /bin/echo "SuSE 6.4, 7.0 and 7.1"
  /bin/echo ""
- /bin/echo "*** Note: 3ware has tested and supports the operating systems listed above."
+ /bin/echo "*** Note: 3ware has tested and supports the operating systems listed above."
  /bin/echo "You will need to configure the following manually on other systems."
  /bin/echo ""
  /bin/echo "You will need to create kill script links to /etc/rc.d/init.d/3dmd"
```

In other words - one quotation mark is missing. After that it runs and installs everything but skips the last step. Once you have run the script, you will have a configuration file on your system, which has these settings inside it:

```
EMAIL No
SERVER dynamic-151.harddata.net
SENDER root
RCPT 3ware_admin
AUDIO No
CALL3WARE No
PORT 1080
FW 25
HELP /usr/local/doc/3dm
SNMP No
PASSWORD No
KEY OmwmsK8IKk2
```

Note that the location for documentation is up to your local admin to choose, with the default '/usr/local/doc' perhaps not the best choice in all instances. /usr/local/share/doc/3dm may be preferable.

Here is what to do:

1. Copy two files '3dm-lnx.tgz' and 'install.3dm' to some convenient location, such as `tmp`

Unpack the tgz file

2. Edit this 'install.3dm' and add the missing "" as per the patch above

3. Run `./install.3dm -i`

4. Disregard what is printed about "supported distributions".

5. Type:

```
"install -o root -g root -m 755 3w-xxxx.rc.redhat /etc/rc.d/init.d/3w-xxxx"
```

```
"chkconfig --add 3w-xxxx"
```

6. If you want to start the service without waiting for a reboot type:

```
"service 3w-xxxx start"
```

7. Also, you may check validity with:

```
"chkconfig --list 3w-xxxx"
```

You should see something like this:

```
3w-xxxx      0:off  1:off  2:on   3:on   4:on   5:on   6:off
```

Written by:

Jakob Østergaard (jakob@ostenfeld.dk)
v. 0.90.7 19th of January 2000

Updated and amended by:

Maurice Hilarius (maurice@harddata.com)

Copyright 2002, Hard Data Ltd.