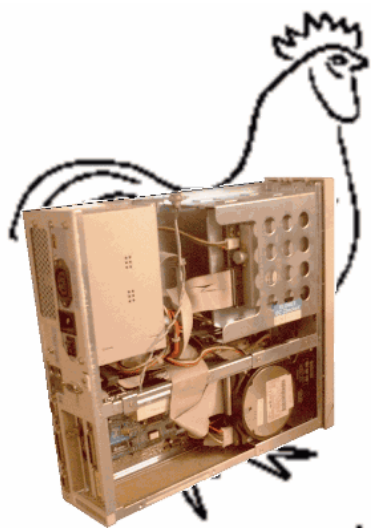
Salvage Server Project 'Junk Ideas' No.1:

Swappable Hard Drives for Backing-up



Produced by the Free Range Salvage Server Project, February 2003
email: fraw@gn.apc.org web: http://www.fraw.org.uk/ssp/

Old hard disks are easy to come by. In the mid-1990s people thought 300 or 600 megabyte hard disks provided massive storage capacity. Today they are relatively useless on most operating systems (Slackware Linux excepted). This sheet looks at putting all these old disks to a practical use as bulk storage devices. For older hard disk the storage capacity is relatively similar to CD-Rs, but perhaps more expensive if you buy the drives. But when using recent, large capacity, hard disks you can achieve storage capacities that are cheaper per gigabyte than DVD-R discs.

"*Hell is truth seen too late*" – John Locke

The concept...

One of the great ideas of information security is 'always back-up your work' (see quote from Locke above). But backing-up requires storage capacity. It's also a good idea to keep a separate back-up off-site so that if the place burns down you still don't lose the bulk of your data. But today, the quantities of data are so large that only voluminous or expensive options are viable to do this.

This 'Junk Idea' seeks to use up all those little hard disks that slowly accrete in computer shops and workshops. Or, if you want some serious storage capacity, you can use the same idea with the latest hard disk. What we do is create a swappable hard drive on a Linux system. This hard drive is treated as if it were a removable media, like a CD-ROM or floppy disk.

Potentially, this creates a huge and highly redundant storage capacity:

- We can back up data to old disks, just to put them to good use.
- We can back up to new, large hard disks so that even the biggest digital video project can be easily held off the system.
- We can, with large or small disks, hold a large quantity of data in a small space, making off-site backing-up far easier.

To make this work we need to install a hard disk 'rack' or 'caddy' in the system. This can be done retrospectively, or when you install the system. Then we modify the file system setting to make this drive user mountable. This allows us to mount/un-mount the drive, for backing-up, and then power down the system to swap it with another hard drive when required. Unfortunately, only the most expensive motherboards have 'hot swappable' hard drives, so we have to power down the system each time we want to change the hard disk. On the plus side, this solution is

Relative costs of back-up media

If we only consider the costs of the media (not the drive it sits in) then the relative costs of different backing-up options are:

Media	Cost, £	Size, MB	Cost/GB data
Floppy disk	£0.40	1.44	£284.00
CD-R	£0.50	600	£0.85
CD-RW	£2.00	600	£3.41
DVD-R	£4.00	3,750	£1.09
DVD-RW	£6.00	3,750	£1.64
Small scrap hard drive	£5.00	400	£12.80
Large scrap hard drive	£10.00	6,440	£1.59
New huge hard drive	£75.00	81,920	£0.94

*Note: Cost/GB = (1024/Size) * Cost*

not only read-write (whereas CD-R/DVD-R are write-once). The transfer rates are higher, and more reliable, than burning CDs.

Setting up

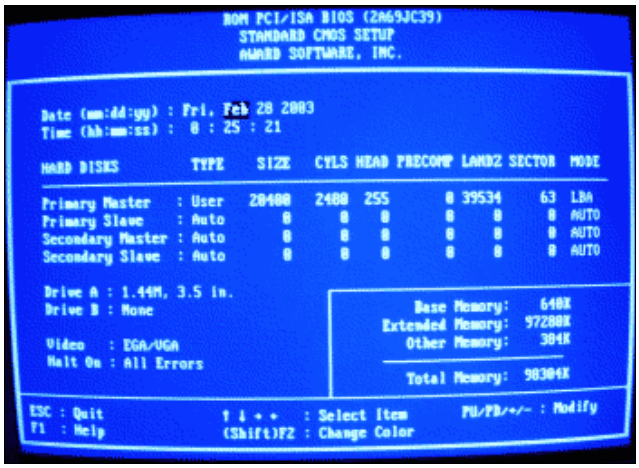
To begin we need to install a hard disk caddy in a 5.25" drive bay and some spare hard disks (top picture in the box to the right). The caddy will cost £6 to £12. But for new, high capacity drives you might want to get one for around £20 with a built in fan because the disks can get very hot. The only restriction on the hard disks is the size of disk your motherboard can handle (see next section).

For the system pictures, the hard disks are wired to the 'master' plug on the IDE drive cable. To ensure the drive function properly you must set the small jumper connector at the rear of the drive. You can then just fit the drive inside the caddy (second picture) not even a need for screws is the caddy has a good shell.

Then you stick the caddy into the rack (third picture). Most caddies have a catch, but some have small key locks. These can be useful to secure against the unauthorised removal of the drive whilst in use.

Now we can configure the Linux file system to accept this hard disk as a removable media. The option we're taking is that the drive will be unmounted at boot time, and mounted only when it is required to store data upon (in our view, this is safer as there's less risk of accidental erasure/corruption).

The first thing to do, before booting the system, is to modify the BIOS setting for the IDE drives. The removable drive on this system is the Secondary Master (see below). So that we can swap different sizes of hard drive without having to keep changing the BIOS settings, we configure BIOS to auto-detect the Type and Mode of the Secondary Master drive. This allows the system to boot normally with a very wide range of disks being inserted in the disk caddy.



Drive Caddy Installation

Changes to the `/etc/fstab` file

/etc/fstab line for the drive, before modification

```
LABEL=/rmhd /rmhd ext3 defaults 1 2
```

/etc/fstab line for the drive, after modification

```
/dev/hdc1 /storage auto noauto,user,sync 0 0
```

Next, either by installing a Linux system, or booting the system and letting it find the drive at the hardware check, we get the new drive to install on the Linux system. This will result in a new line being added to the `/etc/fstab` file which controls the Linux file system (see box above).

When the drive is installed we give it an arbitrary mount point, e.g. '`rmhd`'. The first thing we have to do is create a directory in the Linux file system to act as a target for the mount point of the removable drive. On our systems we put it in the root directory and called it storage. We also need to give it the appropriate permissions. We do all this as the root user using the commands:

```
mkdir /storage
chmod -R 777 /storage
```

We then edit the `/etc/fstab` file with a text editor. Find the line that refers to the removable hard drive and change it as shown in the box above. Note that these lines were taken from a Red Hat 7.3 fstab file – some of the others have subtly different lines, but all work the same way.

The modification make the 'hdc' drive – the secondary master drive – mount to the /storage directory. We let it mount as an 'auto' file system so that we can use any type of file system, Linux native or even Windows, in the removable drive. The rest of the line controls the way the drive is mounted.

If all's well we can mount the drive using the command –

```
mount /storage
```

...and un-mount it using the command –

```
umount /storage
```

On our system we actually configure the /storage drive to be exported as part of the Network File System (NFS). This allows any machine on the network to dump data into the drive once it's been mounted. The drive can be mounted from the system/server it is resident on, or people can rlogin or telnet to the server and mount/umount it remotely. This way the drive can be used for backing-up data from across the network.

Configuring the hard disks for use

To make like simple we configure each hard drive as a single primary partition. If you put a new hard drive in the caddy you'll need to check it's configuration before use, working as the root user, with `fdisk`. You can get more information on using `fdisk` from the `fdisk` man page.

First, you do is run `fdisk` with the command:

```
fdisk /dev/hdc
```

To display the partition information enter 'p'. You then delete any existing partitions using the 'd' command. Finally, you add a new primary partition (partition no.1) using the 'n' command. To conclude, save the changes with the 'w' command (which then quits the program).

We now have a hard drive containing one large partition, but with no file system. We add a file system, of whatever type we need, using the `mkfs` command (with `hdc` as an example):

- `mkfs.ext2 /dev/hdc1` – makes an ext2 filesystem (used on older Linux versions)
- `mkfs.ext3 /dev/hdc1` – makes an ext3 journalling filesystem (used on newer Linux versions)
- `mkreiserfs /dev/hdc1` – make a Reiser filesystem, used as the default on some distributions like SuSE Linux.

In our tests on a 266MHz Pentium II, an old 380MB disk formatted with ext2 in about 30 seconds. A 13GB hard disk formatted with ext3 in about 1 minute.

Once you've formatted the hard disk it's ready to mount and use.

Restrictions on disk size

Different types of motherboard present different restrictions of the maximum size of disk they can accommodate. You have to be aware of this when trying to plug disks into your system.

Many '386/'486-based systems have an absolute limit of 528MB. This is because of the limitations of the older IDE interface (before EIDE). The late '486s and early Pentiums have various limits – either, 528MB, 2.1GB or 4.2GB. You can do a 'workaround' by fiddling with the BIOS settings to extend the 4GB limit, but we don't really want to do that because we want the flexibility of just dropping in a disk without going through BIOS every time.

Later Pentium I boards raised the limit to 8.4GB – but there's no workaround to increase this. Pentium II boards have various limits. Some early ones are stuck at 8.4GB,

whilst later ones can work up to 30GB or more. However, now that more drives in the 10GB to 20GB range are becoming available as people upgrade their systems, you can still get quite a reasonable storage capacity with a Pentium II system. If you want a really huge storage capacity you're going to have to go for a Pentium III board. That'll let you use drives of 160GB to 200GB.

In any case, if you set up your system to accept a wide range of sizes as possible, you can use whatever disks are available as junk, or as shop clearance. That way you keep storage costs down.

And finally...

Some members of the Free Range Network, who are data junkies, have been using this method of backing-up since late 2000. Basically, there isn't time in the day to burn that many CDs! But as well as backing-up, you can use this system to extend the range of data available on your system. For example, you could hold a large library of data set on different hard disk, and plug the one you want in when you need it.

Note also that this method, if people can remote login to mount/un-mount the drive, and using NFS to provide network access, also provides a very low intensity method of backing-up compared to having to supervise/manage CD or tape backups.

We have come across one problem with this system. If moving drive between machines, and sometimes when you've swapped different drives on the same machine, the system sometimes has problem mounting the drive. It fails to recognise the file format, and although it mounts, what you get when doing a directory listing is a load of mush.

To solve this problem you have to define the file system type when you mount the drive. To do this you login as root and use the full version `mount` command, not the abbreviated version that refers to the `/etc/fstab` file for its parameters. For example, using an ext3 file system:

```
mount -t ext3 /dev/hdc1 /storage
```

Note that we give not only the file system type, but also the device ID (`/dev/hdc1`) and the mount point (`/storage`).

Apart from this minor glitch, after a year of usage we've found this method of backing up to be very useful. It's especially useful for doing off-site backups because you can use an old hard disk to dump data to, seal it up in a static/water proof bag, and store it in another location.