ASK KLAUS!

Klaus Knopper is the creator of Knoppix and co-founder of the LinuxTag expo. He currently works as a teacher, programmer, and consultant. If you have a configuration problem, or if you just want to learn more about how Linux works, send your questions to:

klaus@linux-magazine.com

Adding RAID

How can I mount a RAID 1 drive via hotplug with mdadm on Debian and a 2.6.16 kernel?

That is to say: I want to add a second hard drive with the system already installed and with the configuration that follows:

```
dev/hda1=boot; ➥
dev/hda2=swap; ➥
dev/hda3=/;
dev/hda5=usr; ➥
dev/hda6=/var; ➥
dev/hda8 =home.
```

I cannot partition because the system already exists and it has to use mdadm [1] for managing the software RAID array.

For a full discussion of this topic, see the HOWTO on mdadm-Raid with Debian [2]. But first, let's try a few steps. For your setup, I assume that the first hard disk is */dev/hda* (master drive on first controller) and the second one is */dev/hdb* (slave disk on first controller). In theory, you get a better throughput when configuring the second disk as master on the second controller slot, but usually there is already a CD-ROM, which would slow down data transfer when you are using it on the same cable with a hard disk. Let's stay with master and slave disk on the first controller: *hda* for the original drive, and *hdb* for the mirror-disk-to-be. The jumpers on the new disk should be set to "slave disk" accordingly, and attach it to the same cable as the first disk.

Step 0: Make a backup of all relevant data, which is everything you won't be able to reinstall from a Debian installation disk (i.e., your own work and settings: */home* and at least */etc*).

For RAID autodetection, I recommend running a kernel with RAID compiled-in or an initial ramdisk with RAID modules. Alternatively, you can use a small boot system on a non-RAID partition that contains the necessary modules.

(In other words, your root partition including */lib/modules* would be on a plain partition with no RAID.) We'll try the full RAID setup first, where all partitions (except swap) are mirrored on *hdb*.

To make the array autodetectable, change the partition type of all Linux partitions (except swap) to type *fd*, which is *Linux raid autodetect*. I'm using *fdisk* for this, but *cfdisk* should work as well. Because you aren't changing the partition locations and sizes, this will have no bad effects, and the system will still boot normally. The command:

```
fdisk -l /dev/hda
```

should now report something like the output in Listing 1 for your existing partition table (*hda1 = boot; hda2 = swap; hda3 = /; hda5 = usr; hda6 = /var; hda8 = home*). Listing 1 omits the partition beginnings and endings, since these settings depend on your configuration.

You don't need to reformat these partitions and keep the previously installed file systems. Now, you copy the partition table (not the data) from *hda* to *hdb*:

```
sfdisk -d /dev/hda | ➥
sfdisk /dev/hdb
```

The procedure of adding the spare disk to form a RAID 1 array gets a bit odd. Instead of "synchronizing" each partition from *hda* to *hdb*, you create filesystems on a "degraded" RAID 1 array with the first disk (*hda*) missing; software RAID needs to store some meta information outside of the filesystem data structure, and if the filesystem occupies all of the RAID partitions, this metadata could get

overwritten or the filesystem could get damaged. The recommended procedure is to create a RAID 1 structure on *hdb* containing the data from *hda*, boot from this incomplete RAID array, and add *hda* as a "spare disk."

If the kernel fails to boot from a RAID 1 array, you still have the old configuration on *hda* until the synchronization.

Let's create the array:

```
mdadm --create /dev/md0 -l1 ➥
-n2 /dev/hdb1 missing
mdadm --create /dev/md1 -l1 ➥
-n2 /dev/hdb3 missing
mdadm --create /dev/md2 -l1 ➥
-n2 /dev/hdb5 missing
mdadm --create /dev/md3 -l1 ➥
-n2 /dev/hdb6 missing
mdadm --create /dev/md4 -l1 ➥
-n2 /dev/hdb8 missing
```

Because you did not say what is on *hda7*, I skipped it. Add *hda7* if necessary. Format all RAID partitions with your preferred filesystem:

```
for i in `seq 0 4`; do
 mkreiserfs /dev/md$i
done
```

This will only write onto the *hdb* partitions because your RAID does not know anything about *hda* yet. Next, mount and copy from *hda* to the array. I'm showing an example for the *hda1* (boot) partition; copy the other mounted partitions accordingly. The *-x* option of *rsync* makes sure that only files on the mounted file system are copied, with no mounted subdirectories:

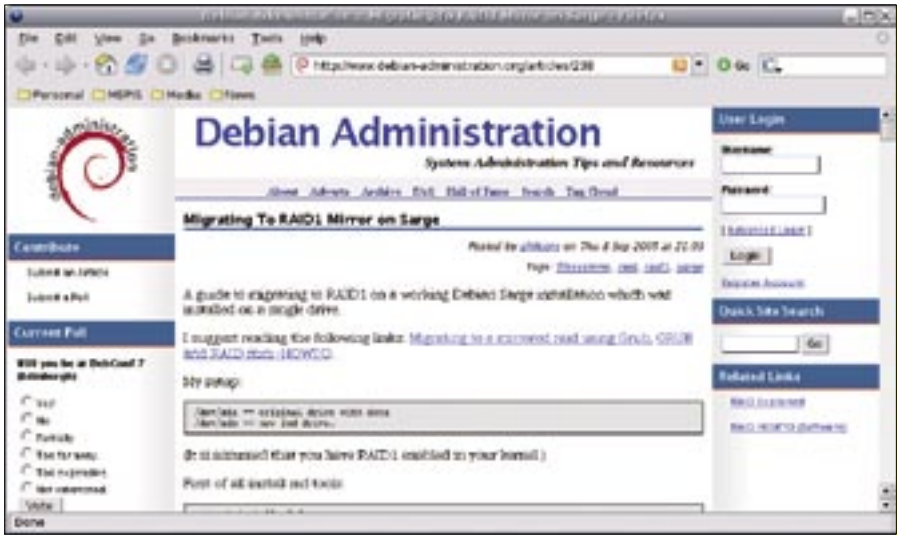

Figure 1: debian-administrator.org is an excellent source for information on Linux RAID and other configuration topics.

```
mkdir -p /mnt
mount /dev/md0 /mnt
rsync -Hax /boot/ /mnt
umount /mnt
```

If you don't have *rsync*, a *cp -dpRx* will do the same.

After you copy all partitions to a RAID device this way, you need to modify */etc/fstab* on the RAID. The RAID root partition in your example is */dev/md1*, so mount this under */mnt* and change the device files for */boot*, */*, */home*, */usr* and */var* to the corresponding */dev/md** partitions. To make use of the duplicate swap partition, use *mkswap /dev/hdb2* and add this partition as additional swap space as well.

/mnt/etc/fstab should now contain lines similar to the example in Listing 2, and you can *umount /mnt/* again. To boot from the RAID, you need to tell the bootloader the new root partition:

```
mount /dev/md0 /mnt
```

Edit */mnt/boot/grub/menu.lst*, keep a backup of the old entries, and add new entries for RAID 1, which look like:

```
title       Linux (new)
root        (hd0,0)
kernel      /vmlinuz-2.6.21 ➥
root=/dev/md1 md=1,➥
/dev/hda3,/dev/hdb3 ro
boot

title    Linux (RAID Recovery)
root      (hd1,0)
kernel    /vmlinuz-2.6.21 ➥
root=/dev/md1 md=1,➥
/dev/hdb3 ro
```

Listing 1: Partitions

```
01 Disk /dev/hda: xxx heads, 63
   sectors, xxxx cylinders
02 Units = cylinders of xxxxx *
   512 bytes
03
04    Device Boot    Start
   End    Blocks   Id  System
05 /dev/hda1    *     ....
   ....  .........  fd  Linux
   raid autodetect
06 /dev/hda2          ....
   ....  .........  82  Linux
   swap
07 /dev/hda3          ....
   ....  .........  fd  Linux
   raid autodetect
08 /dev/hda4          ....
   ....  .........   5  Extended
09 /dev/hda5          ....
   ....  .........  fd  Linux
   raid autodetect
10 /dev/hda6          ....
   ....  .........  fd  Linux
   raid autodetect
11 /dev/hda7          ....
   ....  .........  83  Linux
   raid autodetect
12 /dev/hda8          ....
   ....  .........  fd  Linux
   raid autodetect
```

Listing 2: Sample fstab

```
01 /dev/md0   /boot  reiserfs
   defaults  1  2
02 /dev/md1   /      reiserfs
   defaults  1  1
03 /dev/md2   /usr   reiserfs
   defaults  1  2
04 /dev/md3   /var   reiserfs
   defaults  1  2
05 /dev/md4   /home  reiserfs
   defaults  1  2
06 /dev/hda2  none   swap
   defaults  0  0
07 /dev/hdb2  none   swap
   defaults  0  0
```

```
boot
```

Remember that */dev/md1* is your new root partition. Change */vmlinuz-2.6.21* to match your kernel on the */boot* partition, and if your system uses an *initrd*, add

```
initrd      /initrd.img-2.6.21
```

right after the *kernel* line, as in the existing entries. For this setting to be used immediately when booting from the first disk, enter:

```
cp /mnt/boot/grub/menu.lst ➥
/boot/grub/menu.lst
```

and you can *umount /mnt* again.

Now you should be ready to reboot. Usually, you should not have to reinstall GRUB on *hda*, but if you need to, enter the following line:

```
grub-install /dev/hda
```

After reboot, choosing the GRUB *Linux (RAID recovery)* entry is the safe option. If this does not work, watch the boot (or panic) messages carefully for anything that could have gone wrong. In case of failure, you may just boot your old GRUB entry because you have not changed anything (except for the GRUB config) on the *hda* disk.

If the system boots up in degraded RAID mode, then a *df* should show all partitions that were formerly */dev/hda** as */dev/md**.

If everything is working nicely, you can start the RAID synchronization of *hda* from the currently running *hdb*:

```
mdadm --add /dev/md0 /dev/hda1
mdadm --add /dev/md1 /dev/hda3
mdadm --add /dev/md2 /dev/hda5
mdadm --add /dev/md3 /dev/hda6
mdadm --add /dev/md4 /dev/hda8
```

All data on the partitions of *hda* will now get overwritten. In the shell, you can monitor the progress with:

```
watch cat /proc/mdstat
```

Let this run until synchronization is complete.

Dual-boot Question

I like to try out different Linux distros, and I often dual-boot two Linux systems. Sometimes the last or second distro installed will not show the other system in the bootloader. How do I get the bootloader that shows the first operating system?

It's not a matter of which bootloader to use; you can even use the Windows bootloader to boot several Linux installations and edit *boot.ini* accordingly. However, the bootloader will have to know which operating systems are installed where, so no matter which bootloader you use (LILO, GRUB, or any other), you will have to edit the configuration file in order to give a hint on which systems to show in the boot menu. Assuming your bootloader installed on the master boot record is GRUB, and its configuration file is located in */boot/grub/menu.lst*, Listing 3 shows examples for adding several operating systems to the boot menu.

Listing 3: Adding entries to the GRUB boot menu.

```
01 # /boot/grub/menu.lst for GRUB
   MBR
02 # By default boot the first
   menu entry.
03 default 0
04
05 # Allow 30 seconds before
   booting the default.
06 timeout 30
07
08 # Debian GNU/Linux on
   Partition 4
09 title Debian Etch
10 root (hd0,3)
11 kernel /boot/vmlinuz-2.6.20.1
   root=/dev/hda4
12
13 # Fedora on Partition 3
14 title Fedora
15 root (hd0,2)
16 kernel /boot/kernel-2.6.18.1
   root=/dev/hda3
17 initrd /boot/initrd-2.6.18.1
18
19 # Partition 2 is Linux Swap,
   ignore.
20
21 # Windows on Partition 1
22 title Windows
23 rootnoverify (hd0,0)
24 chainloader +1
```

UMASK

I am pretty new to Linux and found *umask*. When I set *umask 0007* and use *chmod ug+s* on the desired path, it should be possible to log out and log in again. The *umask* should survive the disconnect, but it doesn't. I am using Ubuntu 6.06 and 6.1, Centos 4.3, and openSUSE 10.2, and all have the same problem. I would like a general solution in that I might end up using NFS, ftp, or something else.

The shell internal command *umask*, simplified, sets a mask of permission bits to be taken away from the full set of permissions of created files from within the shell. In other words, *umask 022* removes write permission from newly created files, whereas *chmod 644* changes permissions to be read-write for the owner and read-only for others. *umask* applies only to the current shell and its children, and must be set on each login. It's possible to tell your system to set this option automatically through *.profile* (for login shells), *.bashrc* (for all shells), or */etc/profile* or */etc/bash.bashrc* (globally for all users). Of course, *umask* also works at the beginning of a shell script.

The *umask* permissions of the client process apply when accessing remote files over NFS. For Samba, the relevant *smb.conf* parameters would be *create mask* for files and *directory mask* for directories created on the server side. For ftp, the *umask* configuration depends on the server you choose. ■

INFO

[1] mdadm: *http://neil.brown.name/blog/mdadm*

[2] mdadm with Debian: *http://www.debian-administration.org/articles/238*

Send your Linux questions to klaus@linux-magazine.com.