Antares−RAID−sparcLinux−HOWTO

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Antares−RAID−sparcLinux−HOWTO

Thom Coates (tdc3@psu.edu), Carl Munio, Jim Ludemann

v0.1, 28 April 2000

This document describes how to install, configure, and maintain a hardware RAID built around the 5070 SBUS host based RAID controller by Antares Microsystems. Other topics of discussion include RAID levels, the 5070 controller GUI, and 5070 command line. A complete command reference for the 5070's K9 kernel and Bourne−like shell is included.

---

14. Advanced Topics: SCSI Monitor Daemon (SMON)

15. Further Reading

---

1. Preamble

---

2. Acknowledgements and Thanks

- Carl and Jim at Antares for the hardware, man pages, and other support/contributions they provided during the writing of this document.
- Penn State University − Hershey Medical Center, Department of Radiology, Section of Clinical Image Management (My home away from my home away from home).
- The software−raid−HOWTO Copyright 1997 by Linas Vepstas under the GNU public license agreement. The software−raid−HOWTO is Available from : http://www.linuxdoc.org

---

3. New Versions

- The most recent version of this document can be found at my homepage: http://www.xray.hmc.psu.edu/~tcoates/
- Other versions may be found in different formats at the LDP homepage: http://www.linuxdoc.org and mirror sites.

---

4. Introduction

The Antares 5070 is a high performance, versatile, yet relatively inexpensive host based RAID controller. Its embedded operating system (K9 kernel) is modelled on the Plan 9 operating system whose design is discussed in several papers from AT&T (see the "Further Reading" section). K9 is a kernel targeted at embedded controllers of small to medium complexity (e.g. ISDN−ethernet bridges, RAID controllers, etc). It supports multiple lightweight processes (i.e. without memory management) on a single CPU with a non−pre−emptive scheduler. Device driver architecture is based on Plan 9 (and Unix SVR4) streams. Concurrency control mechanisms include semaphores and signals.

The 5070 has three single ended ultra 1 SCSI channels and two onboard serial interfaces one of which provides command line access via a connected serial terminal or modem. The other is used to upgrade the firmware. The command line is robust, implementing many of the essential Unix commands (e.g. dd, ls, cat, etc.) and a scaled down Bourne shell for scripting. The Unix command set is augmented with RAID specific configuration commands and scripts. In addition to the command line interface an ASCII text based GUI is provided to permit easy configuration of level 0, 1, 3, 4, and 5 RAIDs.

4.1 5070 Main Features

- RAID levels 0, 1, 3, 4, and 5 are supported.
- Text based GUI for easy configuration for all supported RAID levels.
- A Multidisk RAID volume appears as an individual SCSI drive to the operating system and can be managed with the standard utilities (fdisk, mkfs, fsck,etc.). RAID Volumes may be assigned to different SCSI IDs or the same SCSI IDs but different LUNs.
- No special RAID drivers required for the host operating system.
- Multiple RAID volumes of different levels can be mixed among the drives forming the physical plant. For example in a hypothetical drive plant consisting of 9 drives:
    - 2 drives form a level 3 RAID assigned to SCSI ID 5, LUN 0
    - 2 drives form a level 0 RAID assigned to SCSI ID 5, LUN 1
    - 5 drives form a level 5 RAID assigned to SCSI ID 6, LUN 0
- Three single ended SCSI channels which can accommodate 6 drives each (18 drives total).
- Two serial interfaces. The first permits configuration/control/monitoring of the RAID from a local serial terminal. The second serial port is used to upload new programming into the 5070 (using PPP and TFTP).
- Robust Unix−like command line and NVRAM based file system.
- Configurable ASCII SCSI communication channel for passing commands to the 5070's command line interpreter. Allows programming running on host OS to directly configure/control/monitor all parameters of the 5070.

---

5. Background

Much of the information/knowledge pertaining to RAID levels in this section is adapted from the software−raid−HOWTO by Linas Vepstas . See the acknowledgements section for the URL where the full document may be obtained.

RAID is an acronym for "Redundant Array of Inexpensive Disks" and is used to create large, reliable disk storage systems out of individual hard disk drives. There are two basic ways of implementing a RAID, software or hardware. The main advantage of a software RAID is low cost. However, since the OS of the host system must manage the RAID directly there is a substantial penalty in performance. Furthermore if the RAID is also the boot device, a drive failure could prove disastrous since the operating system and utility software needed to perform the recovery is located on the RAID. The primary advantages of hardware RAID is performance and improved reliability. Since all RAID operations are handled by a dedicated CPU on the controller, the host system's CPU is never bothered with RAID related tasks. In fact the host OS is completely oblivious to the fact that its SCSI drives are really virtual RAID drives. When a drive fails on the 5070 it can be replaced on−the−fly with a drive from the spares pool and its data reconstructed without the host's OS ever knowing anything has happened.

5.1 Raid Levels

The different RAID levels have different performance, redundancy, storage capacity, reliability and cost characteristics. Most, but not all levels of RAID offer redundancy against drive failure. There are many different levels of RAID which have been defined by various vendors and researchers. The following describes the first 7 RAID levels in the context of the Antares 5070 hardware RAID implementation.

5.2 RAID Linear

RAID−linear is a simple concatenation of drives to create a larger virtual drive. It is handy if you have a number small drives, and wish to create a single, large drive. This concatenation offers no redundancy, and in fact decreases the overall reliability: if any one drive fails, the combined drive will fail.

SUMMARY

- Enables construction of a large virtual drive from a number of smaller drives
- No protection, less reliable than a single drive
- RAID 0 is a better choice due to better I/O performance

5.3 Level 1

Also referred to as "mirroring". Two (or more) drives, all of the same size, each store an exact copy of all data, disk−block by disk−block. Mirroring gives strong protection against drive failure: if one drive fails, there is another with the an exact copy of the same data. Mirroring can also help improve performance in I/O−laden systems, as read requests can be divided up between several drives. Unfortunately, mirroring is also one of the least efficient in terms of storage: two mirrored drives can store no more data than a single drive.

SUMMARY

- Good read/write performance
- Inefficient use of storage space (half the total space available for data)
- RAID 6 may be a better choice due to better I/O performance.

5.4 Striping

Striping is the underlying concept behind all of the other RAID levels. A stripe is a contiguous sequence of disk blocks. A stripe may be as short as a single disk block, or may consist of thousands. The RAID drivers split up their component drives into stripes; the different RAID levels differ in how they organize the stripes, and what data they put in them. The interplay between the size of the stripes, the typical size of files in the file system, and their location on the drive is what determines the overall performance of the RAID subsystem.

5.5 Level 0

Similar to RAID−linear, except that the component drives are divided into stripes and then interleaved. Like RAID−linear, the result is a single larger virtual drive. Also like RAID−linear, it offers no redundancy, and therefore decreases overall reliability: a single drive failure will knock out the whole thing. However, the 5070 hardware RAID 0 is the fastest of any of the schemes listed here.

SUMMARY:

- Use RAID 0 to combine smaller drives into one large virtual drive.
- Best Read/Write performance of all the schemes listed here.
- No protection from drive failure.
- ADVICE: Buy very reliable hard disk drives if you plan to use this scheme.

5.6 Level 2 and 3

RAID−2 is seldom used anymore, and to some degree has been made obsolete by modern hard disk technology. RAID−2 is similar to RAID−4, but stores ECC information instead of parity. Since all modern disk drives incorporate ECC under the covers, this offers little additional protection. RAID−2 can offer greater data consistency if power is lost during a write; however, battery backup and a clean shutdown can offer the same benefits. RAID−3 is similar to RAID−4, except that it uses the smallest possible stripe size.

SUMMARY

- RAID 2 is largely obsolete
- Use RAID 3 to combine separate drives together into one large virtual drive.
- Protection against single drive failure,
- Good read/write performance.

5.7 Level 4

RAID−4 interleaves stripes like RAID−0, but it requires an additional drive to store parity information. The parity is used to offer redundancy: if any one of the drives fail, the data on the remaining drives can be used to reconstruct the data that was on the failed drive. Given N data disks, and one parity disk, the parity stripe is computed by taking one stripe from each of the data disks, and XOR'ing them together. Thus, the storage capacity of a an (N+1)−disk RAID−4 array is N, which is a lot better than mirroring (N+1) drives, and is almost as good as a RAID−0 setup for large N. Note that for N=1, where there is one data disk, and one parity disk, RAID−4 is a lot like mirroring, in that each of the two disks is a copy of each other. However, RAID−4 does NOT offer the read−performance of mirroring, and offers considerably degraded write performance. In brief, this is because updating the parity requires a read of the old parity, before the new parity can be calculated and written out. In an environment with lots of writes, the parity disk can become a bottleneck, as each write must access the parity disk.

SUMMARY

- Similar to RAID 0
- Protection against single drive failure.
- Poorer I/O performance than RAID 3
- Less of the combined storage space is available for data [than RAID 3] since an additional drive is needed for parity information.

5.8 Level 5

RAID−5 avoids the write−bottleneck of RAID−4 by alternately storing the parity stripe on each of the drives. However, write performance is still not as good as for mirroring, as the parity stripe must still be read and XOR'ed before it is written. Read performance is also not as good as it is for mirroring, as, after all, there is only one copy of the data, not two or more. RAID−5's principle advantage over mirroring is that it offers redundancy and protection against single−drive failure, while offering far more storage capacity when used with three or more drives.

SUMMARY

- Use RAID 5 if you need to make the best use of your available storage space while gaining protection against single drive failure.
- Slower I/O performance than RAID 3

---

6. Installation

NOTE: The installation procedure given here for the SBUS controller is similar to that found in the manual. It has been modified so minor variations in the SPARCLinux installation may be included.

6.1 SBUS Controller Compatibility

The 5070 / Linux 2.2 combination was tested on SPARCstation (5, 10, & 20), Ultra 1, and Ultra 2 Creator. The 5070 was also tested on Linux with Symmetrical Multiprocessing (SMP) support on a dual processor Ultra 2 creator 3D with no problems. Other 5070 / Linux / hardware combinations may work as well.

6.2 Hardware Installation Procedure

If your system is already up and running, you must halt the operating system.

GNOME:

1. From the login screen right click the "Options" button.
2. On the popup menu select System −> Halt.
3. Click "Yes" when the verification box appears

KDE:

1. From the login screen right click shutdown.
2. On the popup menu select shutdown by right clicking its radio button.
3. Click OK

XDM:

1. login as root
2. Left click on the desktop to bring up the pop−up menu
3. select "New Shell"
4. When the shell opens type "halt" at the prompt and press return

Console Login (systems without X windows):

1. Login as root
2. Type "halt"

All Systems:

Wait for the message "power down" or "system halted" before proceeding. Turn off your SPARCstation system (Note: Your system may have turned itself off following the power down directive), its video monitor, external disk expansion boxes, and any other peripherals connected to the system. Be sure to check that the green power LED on the front of the system enclosure is not lit and that the fans inside the system are not running. Do not disconnect the system power cord.

SPARCstation 4, 5, 10, 20 & UltraSPARC Systems:

1. Remove the top cover on the CPU enclosure. On a SPARCstation 10, this is done by loosening the captive screw at the top right corner of the back of the CPU enclosure, then tilting the top of the enclosure forward while using a Phillips screwdriver to press the plastic tab on the top left corner.
2. Decide which SBUS slot you will use. Any slot will do. Remove the filler panel for that slot by removing the two screws and rectangular washers that hold it in.
3. Remove the SBUS retainer (commonly called the handle) by pressing outward on one leg of the retainer while pulling it out of the hole in the printed circuit board.
4. Insert the board into the SBUS slot you have chosen. To insert the board, first engage the top of the 5070 RAIDium backpanel into the backpanel of the CPU enclosure, then rotate the board into a level position and mate the SBUS connectors. Make sure that the SBUS connectors are completely engaged.
5. Snap the nylon board retainers inside the SPARCstation over the 5070 RAIDium board to secure it inside the system.
6. Secure the 5070 RAIDium SBUS backpanel to the system by replacing the rectangular washers and screws that held the original filler panel in place.
7. Replace the top cover by first mating the plastic hooks on the front of the cover to the chassis, then rotating the cover down over the unit until the plastic tab in back snaps into place. Tighten the captive screw on the upper right corner.

Ultra Enterprise Servers, SPARCserver 1000 & 2000 Systems, SPARCserver 6XO MP Series:

1. Remove the two Allen screws that secure the CPU board to the card cage. These are located at each end of the CPU board backpanel.
2. Remove the CPU board from the enclosure and place it on a static−free surface.
3. Decide which SBUS slot you will use. Any slot will do. Remove the filler panel for that slot by removing the two screws and rectangular washers that hold it in. Save these screws and washers.
4. Remove the SBUS retainer (commonly called the handle) by pressing outward on one leg of the retainer while pulling it out of the hole in the printed circuit board.
5. Insert the board into the SBUS slot you have chosen. To insert the board, first engage the top of the 5070 RAIDium backpanel into the backpanel of the CPU enclosure, then rotate the board into a level position and mate the SBUS connectors. Make sure that the SBUS connectors are completely engaged.
6. Secure the 5070 RAIDium board to the CPU board with the nylon screws and standoffs provided on the CPU board. The standoffs may have to be moved so that they match the holes used by the SBUS retainer, as the standoffs are used in different holes for an MBus module. Replace the screws and rectangular washers that originally held the filler panel in place, securing the 5070 RAIDium SBus backpanel to the system enclosure.
7. Re−insert the CPU board into the CPU enclosure and re−install the Allen−head retaining screws that secure the CPU board.

All Systems:

1. Mate the external cable adapter box to the 5070 RAIDium and gently tighten the two screws that extend through the cable adapter box.
2. Connect the three cables from your SCSI devices to the three 68−pin SCSI−3 connectors on the Antares 5070 RAIDium. The three SCSI cables must always be reconnected in the same order after a RAID set has been established, so you should clearly mark the cables and disk enclosures for future disassembly and reassembly.
3. Configure the attached SCSI devices to use SCSI target IDs other than 7, as that is taken by the 5070 RAIDium itself. Configuring the target number is done differently on various devices. Consult the manufacturer's installation instructions to determine the method appropriate for your device.
4. As you are likely to be installing multiple SCSI devices, make sure that all SCSI buses are properly terminated. This means a terminator is installed only at each end of each SCSI bus daisy chain.

Verifying the Hardware Installation:

These steps are optional but recommended. First, power−on your system and interrupt the booting process by pressing the "Stop" and "a" keys (or the "break" key if you are on a serial terminal) simultaneously as soon as the Solaris release number is shown on the screen. This will force the system to run the Forth Monitor in the system EPROM, which will display the "ok" prompt. This gives you access to many useful low−level commands, including:

```
ok show-devs

. . .

/iommu@f,e0000000/sbus@f,e000100SUNW, isp@1,8800000
```

```
. . .
```

The first line in the response shown above means that the 5070 RAIDium host adapter has been properly recognized. If you don't see a line like this, you may have a hardware problem.

Next, to see a listing of all the SCSI devices in your system, you can use the probe−scsi−all command, but first you must prepare your system as follows:

```
ok setenv auto-boot? False

ok reset

ok probe-scsi-all
```

This will tell you the type, target number, and logical unit number of every SCSI device recognized in your system. The 5070 RAIDium board will report itself attached to an ISP controller at target 0 with two Logical Unit Numbers (LUNs): 0 for the virtual hard disk drive, and 7 for the connection to the Graphical User Interface (GUI). Note: the GUI communication channel on LUN 7 is currently unused under Linux. See the discussion under "SCSI Monitor Daemon (SMON)" in the "Advanced Topics" section for more information.

REQUIRED: Perform a reconfiguration boot of the operating system:

```
ok boot -r
```

If no image appears on your screen within a minute, you most likely have a hardware installation problem. In this case, go back and check each step of the installation procedure. This completes the hardware installation procedure.

6.3 Serial Terminal

If you have a serial terminal at your disposal (e.g. DEC−VT420) it may be connected to the controller's serial port using a 9 pin DIN male to DB25 male serial cable. Otherwise you will need to supplement the above cable with a null modem adapter to connect the RAID controller's serial port to the serial port on either the host computer or a PC. The terminal emulators I have successfully used include Minicom (on Linux), Kermit (on Caldera's Dr. DOS), and Hyperterminal (on a windows CE palmtop), however, any decent terminal emulation software should work. The basic settings are 9600 baud , no parity, 8 data bits, and 1 stop bit.

6.4 Hard Drive Plant

Choosing the brand and capacity of the drives that will form the hard drive physical plant is up to you. I do have some recommendations:

- Remember, you generally get what you pay for. I strongly recommend paying the extra money for better (i.e. more reliable) hardware especially if you are setting up a RAID for a mission critical project. For example, consider purchasing drive cabinets with redundant hot−swappable power supplies, etc.
- You will also want a UPS for your host system and drive cabinets. Remember, RAID levels 3 and 5 protect you from data loss due to drive failure NOT power failure.
- The drive cabinet you select should have hot swappable drive bays, these cost more but are definitely worth it when you need to add/change drives.
- Make sure the cabinet(s) have adequate cooling when fully loaded with drives.

- Keep your SCSI cables (internal and external) as short as possible
- Mark the drives/cabinet(s) in such a way that you will be able to reconnect them to the controller in their original configuration. Once the RAID is configured you cannot re−organize you drives without re−configuring the RAID (and subsequently erasing the data stored on it).
- Keep in mind that although it is physically possible to connect/configure up to 6 drives per channel, performance will sharply decrease for RAIDs with more than three drives per channel. This is due to the 25 MHz bandwidth limitation of the SBUS. Therefore, if read/write performance is an issue go with a small number of large drives. If you need a really large RAID (~ 1 terabyte) then you will have no other choice but to load the channels to capacity and pay the performance penalty. NOTE: if you are serving files over a 10/100 Base T network you may not notice the performance decrease since the network is usually the bottleneck not the SBUS.

---

8. Linux Configuration

These instructions cover setting up the virtual RAID drives on RedHat Linux 6.1. Setting it up under other Linux distributions should not be a problem. The same general instructions apply.

If you are new to Linux you may want to consider installing Linux from scratch since the RedHat installer will do most of the configuration work for you. If so skip to section titled "New Linux Installation." Otherwise go to the "Existing Linux Installation" section (next).

8.1 Existing Linux Installation

Follow these instructions if you already have Redhat Linux installed on your system and you do not want to re−install. If you are installing the RAID as part of a new RedHat Linux installation (or are re−installing) skip to the "New Linux Installation" section.

QLogic SCSI Driver

The driver can either be loaded as a module or compiled into your kernel. If you want to boot from the RAID then you may want to use a kernel with compiled in QLogic support (see the kernel−HOWTO available from http://www.linuxdoc.org. To use the modular driver become the superuser and add the following lines to /etc/conf.modules:

```
alias qlogicpti /lib/modules/preferred/scsi/qlogicpti
```

Change the above path to where ever your SCSI modules live. Then add the following line to you /etc/fstab (with the appropriate changes for device and mount point, see the fstab man page if you are unsure)

```
/dev/sdc1 /home ext2 defaults 1 2
```

Or, if you prefer to use a SYSV initialization script, create a file called "raid" in the /etc/rc.d/init.d directory with the following contents (NOTE: while there are a few good reasons to start the RAID using a script, one of the aforementioned methods would be preferable):

```
#!/bin/bash

case "$1" in
```

```
       start)

       echo "Loading raid module"

/sbin/modprobe qlogicpti

echo

echo "Checking and Mounting raid volumes..."

mount −t ext2 −o check /dev/sdc1 /home

touch /var/lock/subsys/raid

;;

stop)

       echo "Unmounting raid volumes"

umount /home

echo "Removing raid module(s)"

/sbin/rmmod qlogicpti

rm −f /var/lock/subsys/raid

echo

;;

restart)

       $0 stop

$0 start

;;

*)

       echo "Usage: raid {start|stop|restart}"

exit 1

esac

exit 0
```

You will need to edit this example and substitute your device name(s) in place of /dev/sdc1 and mount point(s) in place of /home. The next step is to make the script executable by root by doing:

```
chmod 0700 /etc/rc.d/init.d/raid
```

Now use your run level editor of choice (tksysv, ksysv, etc.) to add the script to the appropriate run level.

Device mappings

Linux uses dynamic device mappings you can determine if the drives were found by typing:

```
more /proc/scsi/scsi
```

one or more of the entries should look something like this:

```
Host: scsi1 Channel: 00 Id: 00 Lun: 00

Vendor: ANTARES Model: CX106 Rev: 0109

Type: Direct−Access ANSI SCSI revision: 02
```

There may also be one which looks like this:

```
Host: scsi1 Channel: 00 Id: 00 Lun: 07

Vendor: ANTARES Model: CX106−SMON Rev: 0109

Type: Direct−Access ANSI SCSI revision: 02
```

This is the SCSI monitor communications channel which is currently un−used under Linux (see SMON in the advanced topics section below).

To locate the drives (following reboot) type:

```
dmesg | more
```

Locate the section of the boot messages pertaining to you SCSI devices. You should see something like this:

```
qpti0: IRQ 53 SCSI ID 7 (Firmware v1.31.32)(Firmware 1.25 96/10/15)

[Ultra Wide, using single ended interface]

QPTI: Total of 1 PTI Qlogic/ISP hosts found, 1 actually in use.

scsi1 : PTI Qlogic,ISP SBUS SCSI irq 53 regs at fd018000 PROM node ffd746e0
```

Which indicates that the SCSI controller was properly recognized, Below this look for the disk section:

```
Vendor ANTARES Model: CX106 Rev: 0109

Type: Direct−Access ANSI SCSI revision: 02

Detected scsi disk sdc at scsi1, channel 0, id 0, lun 0

SCSI device sdc: hdwr sector= 512 bytes. Sectors= 20971200 [10239
 MB] [10.2 GB]
```

Note the line that reads "Detected scsi disk sdc ..." this tells you that this virtual disk has been mapped to device /dev/sdc. Following partitioning the first partition will be /dev/sdc1, the second will be /dev/sdc2, etc. There should be one of the above disk sections for each virtual disk that was detected. There may also be an entry like the following:

```
Vendor ANTARES Model: CX106-SMON Rev: 0109

Type: Direct-Access ANSI SCSI revision: 02

Detected scsi disk sdd at scsi1, channel 0, id 0, lun 7

SCSI device sdd: hdwr sector= 512 bytes. Sectors= 20971200 [128 MB]
 [128.2 MB]
```

BEWARE: this is not a drive DO NOT try to fdisk, mkfs, or mount it!! Doing so WILL hang your system.

Partitioning

A virtual drive appears to the host operating system as a large but otherwise ordinary SCSI drive. Partitioning is performed using fdisk or your favorite utility. You will have to give the virtual drive a disk label when fdisk is started. Using the choice "Custom with autoprobed defaults" seems to work well. See the man page for the given utility for details.

Installing a filesystem

Installing a filesystem is no different from any other SCSI drive:

```
mkfs -t <filesystem_type> /dev/<device>
```

for example:

```
mkfs -t ext2 /dev/sdc1
```

Mounting

If QLogic SCSI support is compiled into you kernel OR you are loading the "qlogicpti" module at boot from /etc/conf.modules then add the following line(s) to the /etc/fstab:

```
/dev/<device> <mount point> ext2 defaults 1 1
```

If you are using a SystemV initialization script to load/unload the module you must mount/unmount the drives there as well. See the example script above.

8.2 New Linux Installation

This is the easiest way to install the RAID since the RedHat installer program will do most of the work for you.

1. Configure the host port, RAID sets, and spares as outlined in "Onboard Configuration." Your computer must be on to perform this step since the 5070 is powered from the SBUS. It does not matter if the computer has an operating system installed at this point all we need is power to the controller card.
2. Begin the RedHat SparcLinux installation
3. The installation program will auto detect the 5070 controller and load the Qlogic driver

4. Your virtual RAID drives will appear as ordinary SCSI hard drives to be partitioned and formatted during the installation. NOTE: When using the graphical partitioning utility during the RedHat installation DO NOT designate any partition on the virtual drives as type RAID since they are already hardware managed virtual RAID drives. The RAID selection on the partitioning utilities screen is for setting up a software RAID. IMPORTANT NOTE: you may see a small SCSI drive (usually ~128 MB) on the list of available drives. DO NOT select this drive for use. It is the SMON communication channel NOT a drive. If setup tries to use it it will hang the installer.
5. Thats it, the installation program takes care of everything else !!

---

9. Maintenance

9.1 Activating a spare

When running a RAID 3 or 5 (if you configured one or more drives to be spares) the 5070 will detect when a drive goes offline and automatically select a spare from the spares pool to replace it. The data will be rebuilt on−the−fly. The RAID will continue operating normally during the re−construction process (i.e. it can be read from and written to just is if nothing has happened). When a backend fails you will see messages similar to the following displayed on the 5070 console:

```
930 secs: Redo:1:1 Retry:1 (DIO_cim_homes_D1.1.0_q1) CDB=28(Read_10)Re-/Selection
 Time-out @682400+16

932 secs: Redo:1:1 Retry:2 (DIO_cim_homes_D1.1.0_q1) CDB=28(Read_10)Re-/Selection
 Time-out @682400+16

933 secs: Redo:1:1 Retry:3 (DIO_cim_homes_D1.1.0_q1) CDB=28(Read_10)Re-/Selection
 Time-out @682400+16

934 secs: CIO_cim_homes_q3 R5_W(3412000, 16): Pre-Read drive 4 (D1.1.0)
 fails with result "Re-/Selection Time-out"

934 secs: CIO_cim_homes_q2 R5: Drained alternate jobs for drive 4 (D1.1.0)

934 secs: CIO_cim_homes_q2 R5: Drained alternate jobs for drive 4 (D1.1.0)
 RPT 1/0

934 secs: CIO_cim_homes_q2 R5_W(524288, 16): Initial Pre-Read drive 4 (D1.1.0)
 fails with result "Re-/Selection Time-out"

935 secs: Redo:1:0 Retry:1 (DIO_cim_homes_D1.0.0_q1) CDB=28(Read_10)SCSI
 Bus ~Reset detected @210544+16

936 secs: Failed:1:1 Retry:0 (rconf) CDB=2A(Write_10)Re-/Selection Time-out
 @4194866+128
```

Then you will see the spare being pulled from the spares pool, spun up, tested, engaged, and the data reconstructed.

```
937 secs: autorepair pid=1149 /raid/cim_homes: Spinning up spare device

938 secs: autorepair pid=1149 /raid/cim_homes: Testing spare device/dev/hd/1.5.0/data
```

```
939 secs: autorepair pid=1149 /raid/cim_homes: engaging hot spare ...

939 secs: autorepair pid=1149 /raid/cim_homes: reconstructing drive 4 ...

939 secs: 1054

939 secs: Rebuild on /raid/cim_homes/repair: Max buffer 2800 in 7491 reads,
 priority 6 sleep 500
```

The rebuild script will printout its progress every 10% of the job completed

```
939 secs: Rebuild on /raid/cim_homes/repair @ 0/7491

1920 secs: Rebuild on /raid/cim_homes/repair @ 1498/7491

2414 secs: Rebuild on /raid/cim_homes/repair @ 2247/7491

2906 secs: Rebuild on /raid/cim_homes/repair @ 2996/7491
```

9.2 Re−integrating a repaired drive into the RAID (levels 3 and 5)

After you have replaced the bad drive you must re−integrate it into the RAID set using the following procedure.

1. Start the text GUI
2. Look the list of backends for the RAID set(s).
3. Backends that have been marked faulty will have a (−) to the right of their ID (e.g. D1.1.0−).
4. If you set up spares the ID of the faulty backend will be followed by the ID of the spare that has replaced it (e.g. D1.1.0−D1.5.0) .
5. Write down the ID(s) of the faulty backend(s) (NOT the spares).
6. Press Q to exit agui
7. At the husky prompt type:
   ```
   replace <name> <backend>
   ```

Where <name> is whatever you named the raid set and <backend> is the ID of the backend that is being re−integrated into the RAID. If a spare was in use it will be automatically returned to the spares pool. Be patient, reconstruction can take a few minutes minutes to several hours depending on the RAID level and the size. Fortunately, you can use the RAID as you normally would during this process.

---

10. Troubleshooting / Error Messages

10.1 Out of band temperature detected...

- Probable Cause: The 5070 SBUS card is not adequately cooled.
- Solution: Try to improve cooling inside the case. Clean dust from the fans, re−organize the cards so the raid card is closest to the fan, etc. On some of the "pizza box" sun cases (e.g. SPARC 20) you may need to add supplementary cooling fans especially if you have it loaded with cards.

10.2 ... failed ... cannot have more than 1 faulty backend.

- Cause: More than one backend in the RAID 3/4/5 has failed (i.e. there is no longer sufficient redundancy to enable the lost data to be reconstructed).
- Solution: You're hosed ... Sorry. If you did not assign spares when you configured you RAID 3/4/5 now may be a good time to re−consider the wisdom of that decision. Hopefully you have been making regular backups. Since now you will have to replace the defective drives, re−configure the RAID, and restore the data from a secondary source.

10.3 When booting I see: ... Sun disklabel: bad magic 0000 ... unknown partition table.

- Suspected Cause: Incorrect settings in the disk label set by fdisk (or whatever partitioning utility you used). This message seems to happen when you choose one of the preset disk labels rather than "Custom with autoprobed defaults."
- Solution: Since this error does not seem to effect the operation of the drive you can choose to do nothing and be ok. If you want to correct it you can try re−labeling the disk or re−partitioning the disk and choose "Custom with autoprobed defaults." If you are installing RedHat Linux from scratch the installer will get all of this right for you.

---

11. Bugs

None yet! Please send bug reports to tdc3@psu.edu

---

12. Frequently Asked Questions

12.1 How do I reset/erase the onboard configuration?

At the husky prompt issue the following command:

```
rconf −init
```

This will delete all of the RAID configuration information but not the global variables and scsi monitors. the remove ALL configuration information type:

```
rconf −fullinit
```

Use these commands with caution!

12.2 How can I tell if a drive in my RAID has failed?

In the text GUI faulty backends appear with a (−) to the right of their ID. For example the list of backends:

```
D0.0.0,D1.0.0−,D2.0.0,D0.1.0,D1.1.0,D2.1.0
```

Indicates that backend (drive) D1.0.0 is either faulty or not present. If you assigned spares (RAID 3 or 5) then you should also see that one or more spares are in use. Both the main and the and the RaidSets screens will show information on faulty/not present drives in a RAID set.

---

13. Advanced Topics: 5070 Command Reference

In addition to the text based GUI the RAID configuration may also be manipulated from the husky prompt (the : raid; prompt) of the onboard controller. This section describes commands that a user can input interactively or via a script file to the K9 kernel. Since K9 is an ANSI C Application Programming Interface (API) a shell is needed to interpret user input and form output. Only one shell is currently available and it is called husky. The K9 kernel is modelled on the Plan 9 operating system whose design is discussed in several papers from AT&T (See the "Further Reading" section for more information). K9 is a kernel targeted at embedded controllers of small to medium complexity (e.g. ISDN−ethernet bridges, RAID controllers, etc). It supports multiple lightweight processes (i.e. without memory management) on a single CPU with a non−pre−emptive scheduler. Device driver architecture is based on Plan 9 (and Unix SVR4) STREAMS. Concurrency control mechanisms include semaphores and signals. The husky shell is modelled on a scaled down Unix Bourne shell.

Using the built−in commands the user can write new scripts thus extending the functionality of the 5070. The commands (adapted from the 5070 man pages) are extensive and are described below.

13.1 AUTOBOOT − script to automatically create all raid sets and scsi monitors

- SYNOPSIS: autoboot
- DESCRIPTION: autoboot is a husky script which is typically executed when a RaidRunner boots. The following steps are taken −
    1. Start all configured scsi monitor daemons (smon).
    2. Test to see if the total cache required by all the raid sets that are to boot is not more than 90% of available memory.
    3. Start all the scsi target daemons (stargd) and set each daemon's mode to "spinning−up" which enables it to respond to all non medium access commands from the host. This is done to allow hosts to gain knowledge about the RaidRunner's scsi targets as quickly as possible.
    4. Bind into the root (ram) filesystem all unused spare backend devices.
    5. Build all raid sets.
    6. If battery backed−up ram is present, check for any saved writes and restore them into the just built raid sets.
    7. Finally, set the state of all scsi target daemons to "spun−up" enabling hosts to fully access the raid set's behind them.

13.2 AUTOFAULT − script to automatically mark a backend faulty after a drive failure

- SYNOPSIS: autofault raidset
- DESCRIPTION: autofault is a husky script which is typically executed by a raid file system upon the failure of a backend of that raid set when that raid file system cannot use spare backends or has been configured not to use spare backends. After parsing it's arguments (command and environment)

autofault issues a rconf command to mark a given backend as faulty.

- OPTIONS:
    - raidset: The bind point of the raid set whose backend failed.
    - $DRIVE_NUMBER: The index of the backend that failed. The first backend in a raid set is 0. This option is passed as an environment variable.
    - $BLOCK_SIZE: The raid set's io block size in bytes. (Ignored). This option is passed as an environment variable.
    - $QUEUE_LENGTH: The raid set's queue length. (Ignored). This option is passed as an environment variable.
- SEE ALSO: rconf

13.3 AUTOREPAIR − script to automatically allocate a spare and reconstruct a raid set

- SYNOPSIS: autorepair raidset size
- DESCRIPTION: autorepair is a husky script which is typically executed by either a raid type 1, 3 or 5 file system upon the failure of a backend of that raid set. After parsing it's arguments (command and environment) autorepair gets a spare device from the RaidRunner's spares spool. It then engages it in write−only mode and reads the complete raid device which reconstructs the data on the spare. The read is from the raid file system repair entrypoint. Reading from this entrypoint causes a read of a block immediately followed by a write of that block. The read/write sequence is atomic (i.e is not interruptible). Once the reconstruction has completed, a check is made to ensure the spare did not fail during reconstruction and if not, the access mode of the spare device is set to the access mode of the raid set. The process that reads the repair entrypoint is rebuild.

This device reconstruction will take anywhere from 10 minutes to one and a half hours depending on both the size and speed of the backends and the amount of activity the host is generating.

During device reconstruction, pairs of numbers will be printed indicating each 10% of data reconstructed. The pairs of numbers are separated by a slash character, the first number being the number of blocks reconstructed so far and the second being the number number of blocks to be reconstructed. Further status about the rebuild can be gained from running rebuild.

When the spare is allocated both the number of spares currently used on the backend and the spare device name is printed. The number of spares on a backend is referred to the depth of spares on the backend. Thus prior to re−engaging the spare after a reconstruction a check can be made to see if the depth is the same. If it is not, then the spare reconstruction failed and reconstruction using another spare is underway (or no spares are available), and hence we don't re−engage the drive.

- OPTIONS:
    - raidset: The bind point of the raid set whose backend failed.
    - size : The size of the raid set in 512 byte blocks.
    - $DRIVE_NUMBER: The index of the backend that failed. The first backend in a raid set is 0. This option is passed as an environment variable.
    - $BLOCK_SIZE: The raid set's io block size in bytes. This option is passed as an environment variable.
    - $QUEUE_LENGTH: The raid set's queue length. This option is passed as an environment variable.
- SEE ALSO: rconf, rebuild

13.4 BIND − combine elements of the namespace

- SYNOPSIS: bind [−k] new old
- DESCRIPTION: Bind replaces the existing old file (or directory) with the new file (or directory). If the"−k" switch is given then new must be a kernel recognized device (file system). Section 7k of the manual pages documents the devices (sometimes called file systems) that can be bound using the "−k" switch.

13.5 BUZZER − get the state or turn on or off the buzzer

- SYNOPSIS: buzzer or buzzer on|off|mute
- DESCRIPTION: Buzzer will either print the state of the buzzer, turn on or off the buzzer or mute it. If no arguments are given then the state of the buzzer is printed, that is on or off will be printed if the buzzer is currently on or off respectively. If the buzzer has been muted, then you will be informed of this. If the buzzer has not been used since the RaidRunner has booted then the special state, unused, is printed. If the argument on is given the buzzer is turned on, if off, the buzzer is turned off. If the argument mute is given then the muted state of the buzzer is changed.
- SEE ALSO: warble, sos

13.6 CACHE − display information about and delete cache ranges

- SYNOPSIS: cache [−D moniker] [−I moniker] [−F] [−g moniker first|last] lastoffset
- DESCRIPTION: cache will print (to standard output) information about the given cache range, delete a given cache range, flush the cache or return the last offset of all cache ranges.
- OPTIONS
    - −F: Flush all cache buffers to their backends (typically raid sets).
    - −D moniker: Delete the cache range with moniker (name) moniker.
    - −I moniker: Invalidate the cache for the given cache range (moniker). This is only useful for debugging or elaborate benchmarks.
    - g moniker first|last: Print either the first or last block number of a cache range with moniker (name) moniker.
    - lastoffset: Print the last offset of all cache ranges. The last offset is the last block number of all cache ranges.

13.7 CACHEDUMP − Dump the contents of the write cache to battery backed−up ram

- SYNOPSIS: cachedump
- DESCRIPTION: cachedump causes all unwritten data in the RaidRunner's cache to be written out to the battery backed−up ram. No data will be written to battery backed−up ram if there is currently valid data already stored there. This command is typically executed when there is something wrong with the data (or it's organization) in battery backed−up ram and you need to re−initialize it. cachedump will always return a NULL status.
- SEE ALSO: showbat, cacherestore

13.8 CACHERESTORE − Load the cache with data from battery backed−up ram

- SYNOPSIS: cacherestore
- DESCRIPTION: cacherestore will check the RaidRunner's battery backed−up ram for any data it has stored as a result of a power failure. It will copy any data directly into the cache. This command is typically executed automatically at boot time and prior to the RaidRunner making it's data available to a host. Having successfully copied any data from battery backed−up ram into the cache, it flushes the cache and then re−initializes battery backed−up ram to indicate it holds no data. cacherestore will return a NULL status on success or 1 if an error occurred during the loading (with a message written to standard error).
- SEE ALSO: showbat

13.9 CAT − concatenate files and print on the standard output

- SYNOPSIS: cat [file...]
- DESCRIPTION: cat writes the contents of each given file, or standard input if none are given or when a file named `−' is given, to standard output. If the nominated file is a directory then the filenames contained in that directory are sent to standard out (one per line). More information on a file (e.g. its size) can be obtained by using stat. The script file ls uses cat and stat to produce directory listings.
- SEE ALSO echo, ls, stat

13.10 CMP − compare the contents of 2 files

- SYNOPSIS: cmp [−b blockSize] [−c count] [−e] [−x] file1 file2
- DESCRIPTION: cmp compares the contents of the 2 named files. If file1 is "−" then standard input is used for that file. If the files are the same length and contain the same val- ues then nothing is written to standard output and the exit status NIL (i.e. true) is set. Where the 2 files dif- fer, the first bytes that differ and the position are out- put to standard out and the exit status is set to "differ" (i.e. false). The position is given by a block number (origin 0) followed by a byte offset within that block (origin 0). The optional "−b" switch allows the blockSize of each read operation to be set. The default blockSize is 512 (bytes). For big compares involving disks a relatively large blockSize may be useful (e.g. 64k). See suffix for allowable suffixes. The optional "−c" switch allows the count of blocks read to fixed. A value of 0 for count is interpreted as read to the end of file (EOF). To compare the first 64 Megabytes of 2 files the switches "−b 64k −c 1k" could be used. See suffix for allowable suffixes. The optional "−e" switch instructs ccmmpp to output to stan- dard out (usually overwriting the same line) the count of blocks compared, each time a multiple of 100 is reached. The final block count is also output. The optional "−x" switch instructs ccmmpp to continue after a comparison error (but not a file error) and keep a count of blocks in error. If any errors are detected only the last one will be output when the command exits. If the "−e" switch is also given then the current count of blocks in error is output to the right of the multiple of 100 blocks compared. This command is designed to compare very large files. Two buffers of blockSize are allocated dynamically so their size is bounded by the amount of memory (i.e. RAM in the target) available at the time of command execution. The count could be up to 2G. The number of bytes compared is the product of blockSize and count (i.e. big enough).
- SEE ALSO: suffix

13.11 CONS − console device for Husky

- SYNOPSIS: bind −k cons bind_point
- DESCRIPTION: cons allows an interpreter (e.g. Husky) to route console input and output to an appropriate device. That console input and output is available at bind_point in the K9 namespace. The special file cons should always be available.
- EXAMPLES: Husky does the following in its initialisation:

```
bind -k cons /dev/cons
```

On a Unix system this is equivalent to:

```
bind -k unixfd /dev/cons
```

On a DOS system this is equivalent to:

```
bind -k doscon /dev/cons
```

On target hardware using a SCN2681 chip this is equivalent to:

```
bind -k scn2681 /dev/cons
```

- SEE ALSO: unixfd, doscon, scn2681

13.12 DD − copy a file (disk, etc)

- SYNOPSIS: dd [if=file] [of=file] [ibs=bytes] [obs=bytes] [bs=bytes] [skip=blocks] [seek=blocks] [count=blocks] [flags=verbose]
- DESCRIPTION: dd copies a file (from the standard input to the standard output, by default) with a user−selectable blocksize.
- OPTIONS
    - if=file Read from file instead of the standard input.
    - of=file, Write to file instead of the standard output.
    - ibs=bytes, Read given number of bytes at a time.
    - obs=bytes, Write given number of bytes at a time.
    - bs=bytes, Read and write given number of bytes at a time. Override ibs and obs.
    - skip=blocks, Skip ibs−sized blocks at start of input.
    - seek=blocks, By−pass obs−sized blocks at start of output.
    - count=blocks, Copy only ibs−sized input blocks.
    - flags=verbose, Print (to standard output) the number of blocks copied every ten percent of the copy. The output is of the form X/T where X is the number of blocks copied so far and T is the total number of blocks to copy. This option can only be used if both the count= and of= options are also given.

  The decimal numbers given to "ibs", "obs", "bs", "skip", "seek" and "count" must not be negative. These numbers can optionally have a suffix (see suffix). dd outputs to standard out in all cases. A successful copy of 8 (full) blocks would cause the following output:

```
8+0 records in

8+0 records out
```

The number after the "+" is the number of fractional blocks (i.e. blocks that are less than the block size) involved. This number will usually be zero (and is otherwise when physical media with alignment requirements is involved).

A write failure outputting the last block on the previous example would cause the following output:

```
Write failed

8+0 records in

7+0 records out
```

- SEE ALSO: suffix

13.13 DEVSCMP − Compare a file's size against a given value

- SYNOPSIS: devscmp filename size
- DESCRIPTION: devscmp will find the size of the given file and compare it's size in 512−byte blocks to the given size (to be in 512−byte blocks). If the size of the file is less than the given value, then −1 is printed, if equal to then 0 is printed, and if the size of the given file is greater than the given size then 1 is printed. This routine is used in internal scripts to ensure that backends of raid sets are of an appropriate size.

13.14 DFORMAT− Perform formatting functions on a backend disk drive

- SYNOPSIS
    - dformat −p c.s.l −R bnum
    - dformat −p c.s.l −pdA|−pdP|−pdG
    - dformat −p c.s.l −S [−v] [−B firstbn]
    - dformat −p c.s.l −F
    - dformat −p c.s.l −D file
- DESCRIPTION: In it's first form dformat will either reassign a block on a nominated disk drive. via the SCSI−2 REASSIGN BLOCKS command. The second form will allow you to print out the current manufacturers defect list (−pdP), the grown defect list (−pdG) or both defect lists (−pdA). Each printed list is sorted with one defect per line in Physical Sector Format − Cylinder Number, Head Number and Defect Sector Number. The third form causes the drive to be scanned in a destructive write/read/compare manner. If a read or write or data comparison error occurs then an attempt is made to identify the bad sector(s). Typically the drive is scanned from block 0 to the last block on the drive. You can optionally give an alternative starting block number. The fourth form causes a low level format on the specified device. The fifth option allows you to download a device's microcode into the device.
- OPTIONS:
    - −R bnum: Specify a logical block number to reassign to the drive's grown defect list.
    - −pdA: Print both the manufacturer's and grown defect list.
    - \ −pdP: Print the manufacturer's defect list.
    - −pdG: Print the grown defect list.
    - −S: Perform a destructive scan of the disk reporting I/O errors.
    - −B firstbn: Specify the first logical block number to start a scan from.
    - −v: Turn on verbose mode − which prints the current block number being scanned.

    - −F: Issue a low−level SCSI format command to the given device. This will take some time.
    - −D file: Download into the specified device, the given file. The download is effected by a single SCSI Write−Buffer command in save microcode mode. This allows users to update a device's microcode. Use this command carefully as you could destroy the device by loading an incorrect file.
    - −p c.s.l: Identify the disk device by specifying it's channel, SCSI ID (rank) and SCSI LUN provided in the format "c.s.l"
- SEE ALSO: Product manual for disk drives used in your RAID.

13.15 DIAGS − script to run a diagnostic on a given device

- SYNOPSIS: diags disk −C count −L length −M io−mode −T io−type −D device
- DESCRIPTION: diags is a husky script which is used to run the randio diagnostic on a given device. When randio is executed, it is executed in verbose mode.
- OPTIONS:
    - disk: This is the device type of diagnostic we are to run.
    - −C count: Specify the number of times to execute the diagnostic.
    - −L length: Specify the "length" of the diagnostic to execute. This can be either short, medium or long and specified with the letter's s, m or l respectively. In the case of a disk, a short test will the first 10% of the device, a medium the first 50% and long the whole (100%) of the disk.
    - −M io−mode: Specify a destructive (read−write) or non−destructive (read−only) test. Use either read−write or read−only.
    - −T io−type: Specify a type of io − either sequential or random.
    - −D device: Specify the device to test.
- SEE ALSO: randio, scsihdfs

13.16 DPART − edit a scsihd disk partition table

- SYNOPSIS:
    - dpart −a|d|l|m −D file [−N name] [−F firstblock] [−L lastblock]
    - dpart −a −D file −N name −F firstblock −L lastblock
    - dpart −d −D file −N name
    - dpart −l −D file
    - dpart −m −D file −N name −F firstblock −L lastblock
- DESCRIPTION: Each scsihd device (typically a SCSI disk drive) can be divided up into eight logical partitions. By default when a scsihd device is bound into the RaidRunner's file system, it has four partitions, the whole device (raw), typically named bindpoint/raw, the partition file (bindpoint/partition), the RaidRunner backup configuration file (bindpoint/rconfig), and the "data" portion of the disk (bind− point/data) which represents the whole device less the backup configuration area and partition file. For more information, see scsihdfs. If other partitions are added, then they will appear as bindpoint/partitionname. dpart allows you to edit or list the partition table on a scsihd device (typically a disk).
- OPTIONS:
    - −a: Add a partition. When adding a partition, you need to specify the partition name (−N) and the partition range from the first block (−F) to the last block (−L).
    - −d: Delete a named (−N) partition.
    - −l: List all partitions.
    - −m: Modify an existing partition. You will need to specify the partition name (−N) and BOTH it's first (−F) and last (−L) blocknumbers even if you are just modifying the last block

number.
    ♦ −D file: Specify the partition file to be edited. Typically, this is the bindpoint/partition file.
    ♦ −N name: Specify the partition name.
    ♦ −F firstblock: Specify the first block number of the partition.
    ♦ −L lastblock: Specify the last block number of the partition.
* SEE ALSO: scsihd

13.17 DUP − open file descriptor device

* SYNOPSIS: bind −k dup bind_point
* DESCRIPTION: The dup device makes a one level directory with an entry in that directory for every open file descriptor of the invoking K9 process. These directory "entries" are the numbers. Thus a typical process (script) binding a dup device would at least make these files in the namespace: "bind_point/0", "bind_point/1" and "bind_point/2". These would correspond to its open standard in, standard out and standard error file descriptors. A dup device allows other K9 processes to access the open file descriptors of the invoking process. To do this the other processes simply "open" the required dup device directory entry whose name (a number) corresponds to the required file descriptor.

13.18 ECHO − display a line of text

* SYNOPSIS: echo [string ...]
* DESCRIPTION: echo writes each given string to the standard output, with a space between them and a newline after the last one. Note that all the string arguments are written in a single write kernel call. The following backslash−escaped characters in the strings are converted as follows: \b backspace

\c suppress trailing newline

\f form feed

\n new line

\r carriage return

\t horizontal tab

\v vertical tab

\\ backslash

\nnn the character whose ASCII code is nnn (octal)

* SEE ALSO: cat

13.19 ENV− environment variables file system

* SYNOPSIS: bind −k env bind_point
* DESCRIPTION: env file system associates a one level directory with the bind_point in the K9 namespace. Each file name in that directory is the name of the environment variable while the

contents of the file is that variable's current value. Conceptually each process sees their own copy of the env file system. This copy is either empty or inherited from this process's parent at spawn time (depending on the flags to spawn).

13.20 ENVIRON − RaidRunner Global environment variables − names and effects

- DESCRIPTION: The RaidRunner uses GLOBAL environment variables to control the functionality of automatic actions. GLOBAL environment variables are saved in the Raid configuration area so they retain their values between reboots/power downs. Certain RaidRunner internal run−time variables can also be set as a GLOBAL environment variables. See the internals manual entry for details. The table below describes those GLOBAL environment variables that are used by the RaidRunner in it's normal operation.
    - RebuildPri This variable, if set, controls the priority used when drive reconstruction occurs via the rebuild program. If the variable is not set then the default rebuild priority would be used. The variable is to be a comma separated list of raid set names and their associated rebuild priorities and sleep periods (colon separated). The form is

      ```
      Rname_1:Pri_1:Sleep_1,Rname_2:Pri_2:Sleep_2,...,Rname_N:Pri_N:Sleep_N
      ```

      where Pri_1 is to be the priority the rebuild program runs with when run on raid set Rname_1, Sleep_1 is the period, in milliseconds, to sleep between each rebuild action on the raid set, Pri_2 is to be the priority for raid set Rname_2, and so forth. For example, if the value of RebuildPri is

      ```
      R:5:30000
      ```

  then if a rebuild occurs (via replace, repair or autorepair) on raid set R then the rebuild will run with priority 5 (via the −p rebuild option) and will sleep 30000 milliseconds (30 seconds) between each rebuild action (specified via the −S rebuild option). The priority given must be valid for the rebuild program.
- BackendRanks On certain RaidRunner's where multiple controllers may exist, you can restrict a controller's access to the backend ranks of devices available. For example, you may have 2 controllers and 4 ranks of backend devices. You can specify that the first controller can only access the first two ranks and the second controller, the second two ranks. This variable along with other associated commands allows you to set up this restriction. Additionally, you may only have a single controller RaidRunner which is in an enclosure with multiple ranks. By default the controller will attempt to probe for all devices on all ranks. If you have only populated the RaidRunner with say, half it's possible compliment of backend devices, then the RaidRunner will still probe for the other half. Setting this variable appropriately will prevent this un−needed (and on occasion time consuming) process. This variable takes the form

  ```
  controller_id:ranklist controller_id:ranklist ...
  ```

  where controller_id is the controller number (from 0 upwards) and ranklist is a comma list of backend ranks which the given controller will access. Note that the backend rank is the scsi−id of that rank. For example, on a 2 rank (rank 1 and 2 − i.e scsi id 1 for the first rank and scsi id 2 for the second), 1 controller This variable takes the form For example, on a 2 rank (rank 1 and 2 − i.e scsi id 1 for the first rank and scsi id 2 for the second), 1 controller RaidRunner where only the first rank has devices you could prevent the controller from attempting to access the (empty) second rank by setting BackendRanks to

```
0:1
```

Typically, you would not set this variable directly, but use supporting commands to set it. These commands are pranks and sranks. See these manual entries for details.

* RAIDn_reference_PBUFS Raid types 3, 4 and 5 all make use of memory for temporary parity buffers when they need to create parity data. This memory is in addition to that allocated to a raid set's cache. When a raid set is created, it will also create a default number of parity buffers (which are the same size is a raid set's iosize). Sometimes, if the iosize of the raid set is large there will not be enough memory to create this default number of parity buffers. To overcome this situation, you can set GLOBAL environment variables to over−ride the default number of parity buffers that all raid sets of a particular type or a specific raid set will use. You need to set these variables before you define the raid set via agui and if you delete them and not the raid set, then the effect raid sets may not boot and hence will not be accessible by a host. The variables are of the form RAIDn_reference_PBUFS where n is the raid type (3, 4 or 5), and reference is the raid set's name or the string 'Default' You use the reference of 'Default' to specify all raid sets of a particular type. For example, to over−ride the number of parity buffers for a raid 5 named

```
: raid ; setenv RAID5_FRED_PBUFS 64
```

To over−ride the number of parity buffers for ALL raid 3's (and set only 72 parity buffers) set

```
: raid ; setenv RAID3_Default_PBUFS 128
```

If you set a default for all raid sets of a particular type, but want ONE of them to be different then set up a variable for that particular raid set as it's value will over−ride the default. In the above example, where all Raid Type 3 will have 128 parity buffers, you could set the variable

```
: raid ; setenv RAID3_Dbase_PBUFS 56
```

which will allow the raid 3 raid set named 'Dbase' to have 56 parity buffers, but all other raid 3's defined on the RaidRunner will have 128.

* SEE ALSO: setenv, printenv, rconf, rebuild, internals

13.21 EXEC − cause arguments to be executed in place of this shell

* SYNOPSIS: exec [arg ...]
* DESCRIPTION: exec causes the command specified by the first arg to be executed in place of this shell without creating a new process. Subsequent args are passed to the command specified by the first arg as its arguments. Shell redirection may appear and, if no other arguments are given, causes the shell input/output to be modified.

13.22 EXIT − exit a K9 process

* SYNOPSIS: exit [string]
* DESCRIPTION: exit has an optional string argument. If the optional argument is given the current K9 process is terminated with the given string as its exit value. (If the string has embedded spaces then the whole string should be a quoted_string). If no argument is given then the shell gets the string associated with the environment variable "status" and returns that string as the exit value. If the environment variable "status" is not found then the "true" exit status (i.e. NIL) is returned.

- SEE ALSO: true, K9exit

13.23 EXPR − evaluation of numeric expressions

- SYNOPSIS: expr numeric_expr ...
- DESCRIPTION: expr evaluates each numeric_expr command line argument as a separate numeric expression. Thus a single expression cannot contain unescaped whitespaces or needs to be placed in a quoted string (i.e. between "{" and "}"). Arithmetic is performed on signed integers (currently numbers in the range from −2,147,483,648 to 2,147,483,647). Successful calculations cause no output (to either standard out/error or environment variables). So each useful numeric_expr needs to include an assignment (or op−assignment). Each numeric_expr argument supplied is evaluated in the order given (i.e. left to right) until they all evaluate successfully (returning a true status). If evaluating a numeric_expr fails (usually due to a syntax error) then the expr command fails with "error" as the exit status and the error message is written to the environment variable "error".
- OPERATORS: The precedence of each operator is shown following the description in square brackets. "0" is the highest precedence. Within a single precedence group evaluation is left−to−right except for assignment operators which are right−to−left. Parentheses have higher precedence than all operators and can be used to change the default precedence shown below. UNARY OPERATORS

+

Does nothing to expression/number to the right.

−

negates expression/number to the right.

!

logically negate expression/number to the right.

~

Bitwise negate expression/number to the right.

BINARY ARITHMETIC OPERATORS

*

Multiply enclosing expressions [2]

/

Integer division of enclosing expressions

%

Modulus of enclosing expressions.

+

Add enclosing expressions

−

Subtract enclosing expressions.

<<

Shift left expression _left_ by number in right expression. Equivalent to: left * (2 ** right)

>>

Shift left expression _right_ by number in right expression. Equivalent to: left / (2 ** right)

&

Bitwise AND of enclosing expressions

^

Bitwise exclusive OR of enclosing expressions. [8]

|

Bitwise OR of enclosing expressions. [9]

BINARY LOGICAL OPERATORS

These logical operators yield the number 1 for a true comparison and 0 for a false comparison. For logical ANDs and ORs their left and right expressions are assumed to be false if 0 otherwise true. Both logical ANDs and ORs evaluate both their left and right expressions in all case (cf. C's short−circuit action).

<=

true when left less than or equal to right. [5]

>=

true when left greater than or equal to right. [5]

<

true when left less than right. [5]

>

true when left greater than right. [5]

==

true when left equal to right. [6]

!=

true when left not equal to right. [6]

&&

logical AND of enclosing expressions [10]

||

logical OR of enclosing expressions [11]

ASSIGNMENT OPERATORS

In the following descriptions "n" is an environment variable while "r_exp" is an expression to the right. All assignment operators have the same precedence which is lower than all other operators. N.B. Multiple assignment operators group right−to−left (i.e. same as C language).

=

Assign right expression into environment variable on left.

*=

n *= r_exp is equivalent to: n = n * r_exp

/=

n /= r_exp is equivalent to: n = n / r_exp

%=

n %= r_exp is equivalent to: n = n % r_exp

+=

n += r_exp is equivalent to: n = n + r_exp

−=

n −= r_exp is equivalent to: n = n − r_exp

<<=

n <<= r_exp is equivalent to: n = n << r_exp

>>=

n >>= r_exp is equivalent to: n = n >> r_exp

&=

n &= r_exp is equivalent to: n = n & r_exp

|=

n |= r_exp is equivalent to: n = n | r_exp

* NUMBERS: All number are signed integers in the range stated in the description above. Numbers can be input in base 2 through to base 36. Base 10 is the default base. The default base can be overridden by:
    1. a leading "0" : implies octal or hexadecimal
    2. a number of the form _base_#_num_

  Numbers prefixed with "0" are interpreted as octal. Numbers prefixed with "0x" or "0X" are interpreted as hexadecimal. For numbers using the "#" notation the _base_ must be in the range 2 through to 36 inclusive. For bases greater then 10 the letters "a" through "z" are utilised for the extra "digits". Upper and lower case letters are acceptable. Any single digit that exceeds (or is equal to) the base is consider an error. Base 10 numbers only may have a suffix. See suffix for a list of valid suffixes. Also note that since expr uses signed integers then "1G" is the largest magnitude number that can be represented with the "Gigabyte" suffix (assuming 32 bit signed integers, −2G is invalid due to the order of evaluation).
    * VARIABLES: The only symbolic variables allowed are K9 environment variables. Regardless of whether they are being read or written they should never appear preceded by a "$". Environment variables that didn't previous exist that appear as left argument of an assignment are created. When a non−existent environment variable is read then it is interpreted as the value 0.
    * EXAMPLES: Some simple examples:
      ```
              expr {n = 1 + 2} # create n
      echo $n

      3

      expr {n*=2} # 3 * 2 result back into n

      echo $n

      6

      expr { k = n > 5 } # 6 > 5 is true so create k = 1

      echo $k

      1
      ```
* NOTE: expr is a Husky "built−in" command. See the "Note" section in "set" to see the implications.
* SEE ALSO: husky, set, suffix, test

13.24 FALSE − returns the K9 false status

* SYNOPSIS: false
* DESCRIPTION: false does nothing other than return a K9 false status. K9 processes return a pointer to a C string (null terminated array of characters) on termination. If that pointer is NULL then a true exit value is assumed while all other returned pointer values are interpreted as false (with the string being some explanation of what went wrong). This command returns a pointer to the string "false" as its return value.

* EXAMPLE: The following script fragment will print "got here" to standard out:

```
        if false then
echo impossible

else

echo got here

end
```

* SEE ALSO: true

13.25 FIFO − bi−directional fifo buffer of fixed size

* SYNOPSIS:
    * bind −k {fifo size} bind_point
    * cat bind_point
    * bind_point/data
    * bind_point/ctl
* DESCRIPTION: fifo file system associates a one level directory with the bind_point in the K9 namespace with a buffer size of size bytes. bind_point/data and bind_point/ctl are the data and control channels for the fifo. Data written to the bind_point/data file is available for reading from the same file in a first−in first−out basis. A write of x bytes to the bind_point/data file will either complete and and transfer all the data, or will transfer sufficient bytes until the fifo buffer is full then block until data is removed from the fifo buffer by reading. A read of x bytes from the bind_point/data file will transfer the lessor of the current amount of data in the fifo buffer or x bytes. A read from the bind_point/ctl will return the size of the fifo buffer and the current usage. The number of opens (# Opens) is the number of processes that currently have the bind_point/data file open.
* EXAMPLE

```
       > /buffer
bind -k {fifo 2048} /buffer

ls -l /buffer

/buffer:

/buffer/ctl                     fifo    2 0x00000001    1 0

/buffer/data                    fifo    2 0x00000002    1 0

cat /buffer/ctl

Max: 2048 Cur: 0, # Opens: 0

echo hello > /buffer/data

cat /buffer/ctl

Max: 2048 Cur: 6, # Opens: 0

dd if=/buffer/data bs=512 count=1
```

```
hello

0+1 records in

0+1 records out

cat /buffer/ctl

Max: 2048 Cur: 0, # Opens: 0
```

* SEE ALSO: pipe

13.26 GET − select one value from list

* SYNOPSIS: get number [value ...]
* DESCRIPTION: get uses the given number to select one value from the given list. Indexing is origin 0 (e.g. "get 0 aaa bb c" returns "aaa"). If the number is out of range for an index on the given list of values then nothing is returned.

13.27 GETIV − get the value an internal RaidRunner variable

* SYNOPSIS:
    * getiv
    * getiv name
* DESCRIPTION: getiv prints the current value of an internal RaidRunner variable or prints a list of all variables. When a variable name is given it's current value is printed. If no value is given the all available internal variables are listed.
* NOTES: As different models of RaidRunners have different internal variables see your RaidRunner's Hardware Reference manual for a list of variables together with the meaning of their values. These variables are run−time variables and hence revert to their default value whenever the RaidRunner is booted.
* SEE ALSO: setiv

13.28 HELP − print a list of commands and their synopses

* SYNOPSIS: help or ?
* DESCRIPTION: help or the question mark character − ?, will print a list of all commands available to the command interpreter. Along with each command, it's synopsis is printed.

13.29 HUSKY − shell for K9 kernel

* SYNOPSIS
    * husky [−c command] [file [arg ...]]
    * hs [−c command] [file [arg ...]]
* DESCRIPTION: husky and hs are synonyms. husky is a command language interpreter that executes commands read from the standard input or from a file. husky is a scaled down model of Unix's Bourne shell (sh). One major difference is that husky has no concept of current working directory. If the "−c" switch is present then the following command is interpreted by husky in a newly thrown shell nested in the current environment. This newly thrown shell exits back to the current environment when the command finishes. Otherwise if arguments are given the first one is assumed

to be a file containing husky commands. Again a new shell is thrown to execute these commands. husky script files can access their command line arguments and the 2nd and subsequent arguments to husky (if present) are passed to the file for that purpose. If no arguments are given to husky then commands are read from standard in (and the shell is considered interactive).

* RETURN STATUS: husky places the K9 return status of a process (NIL if ok, otherwise a string explaining the error) in the file "/env/status" An example:

```
dd if=/xx

dd: could not open /xx

cat /env/status

open failed

cat /env/status

# empty because previous "cat" worked
```

As the file "/env/status" is an environment variable the return status of a command is also available in the variable $status. The exit status of a pipeline is the exit status of the last command in the pipeline.

* SIGNALS If an interactive shell receives an interrupt signal (i.e. K9_SIGINT − usually a control−C on the console) then the shell exits. The "init" process will then start a new instance of the husky shell with all the previously running processes (with the exception of the just killed shell) still running. This allows the user to kill the process that caused the previous shell problems. Alternatively a process that is acci- dentally run in foreground is effectively put in the background by sending an interrupt signal to the shell. Note that this is quite different to Unix shells which would forward the signal onto the foreground process.
* QUOTES, ESCAPING, STRING CONCATENATION, ETC: A quoted_string (as defined in the grammar) commences with a "{" and finishes with the matching "}". The term "matching" implies that all embedded "{" must have a corresponding embedded "}" before the final "}" is said to match the original "{". A quoted_string can be spread across several lines. No command line substitution occurs within quoted_strings. The character for escaping the following character is "\". If a "{" needs to be interpreted literally then it can be represented by "\{". If a string containing spaces (whitespaces) needs to be interpreted as a single token then space (whitespace) can be escaped (i.e. "\ "). If a "\" itself needs to be interpreted literally then it can be represented by "\\". The string concatenation character is "^". This is useful when a token such as "/d4" needs to built up by a script when "/d" is fixed and the "4" is derived from some variable:

```
       set n 4
> /d^$n
```

This example would create the file "/d4".

The output of another husky command or script can be made available inline by starting the sequence with "`" and finishing it with a "'". For example:

```
echo {ps output follows:

} `ps'
```

This prints the string "ps output follows:" followed on the next line by the current output from the command "ps". That output from "ps" would have its embedded newlines replaced by whitespaces.

- COMMAND LINE FILE REDIRECTION:
    - Redirection should appear after a command and its arguments in a line to be interpreted by husky. A special case is a line that just contains "> filename" which creates the filename with zero length if it didn't previously exist or truncates to zero length if it did.
    - Redirection of standard in to come from a file uses the token "<" with the filename appearing to its right. The default source of standard in is the console.
    - Redirection of standard out to go to a file uses the token ">" with the filename appearing to its right. The default destination of standard out is the console.
    - Redirection of standard error to go to a file uses the token ">[2]" with the filename appearing to its right. The default destination of standard error is the console.
    - Redirection of writes from within a command which uses a known file descriptor number (say "n") to go to a file uses the token ">[n]" with the filename appearing to its right.
    - Redirection of read from within a command which uses a known file descriptor number (say "n") to come from a file uses the token "<[n]" with the filename appearing to its right.
    - Redirection of reads and writes from within a command which uses a known file descriptor number (say "n") to a file uses the token "<>[n]" with the filename appearing to its right. In order to redirect both standard out and standard error to the one file the form " > filename >[2=1]" can be used. This sequence first redirects standard out (i.e. file descriptor 1) to filename and then redirects what is written to file descriptor 2 (i.e. standard error) to file descriptor 1 which is now associated with filename.
- ENVIRONMENT VARIABLES: Each process can access the name it was invoked by via the variable: "arg0" . The command line arguments (excluding the invocation name) can be accessed as a list in the variable: "argv" . The number of elements in the list "argv" is place in "argc". The get command is useful for fetching individual arguments from this list. The pid of the current process can be fetched from the variable: "pid". When a script launches a new process in the background then the child's pid can be accessed from the variable "child". The variable "ContollerId" is set to the RaidRunner controller number husky is running on. Environment variables are a separate "space" for each process. Depending on the way a process was created, its initial set of environment variables may be copied from its parent process at the "spawn" point.
- SEE ALSO: intro

13.30 HWCONF − print various hardware configuration details

- SYNOPSIS: hwconf [−D] [−M] [−I] [−d [−n]] [−f] [−h] [−i −p c.s.l] [−m] [−p c.s.l] [−s] [−S] [−t] [−T] [−P] [−W]
- DESCRIPTION: hwconf prints details about the RaidRunner hardware and devices attached.
- OPTIONS:
    - ♦ −h: Print the number of controllers, host interfaces per controller, the number of disk channels per controller, number of ranks of disks and the details memory (in bytes) on each controller. Four memory figures are printed, the first is the total memory in the controller, next is the amount of memory at boot time, next is the amount currently available and lastly is the largest available contiguous area of memory. This is the default option.
    - ♦ −f: Print the number of fans in the RaidRunner and then the speed for each fan in the system. The speeds values are in revolutions per minute (rpms). The fans in the system are labeled in your hardware specification sheet for your RaidRunner. The first speed printed from this command corresponds to fan number 0 on your specification sheet, the second is for fan 1, and so forth.
    - ♦ −d: Print out information on all the disk drives on the RaidRunner. For each disk on the RaidRunner, print out − the device name, in the format c.s.l where c is the channel, s is the SCSI ID (or rank) and l is the SCSI LUN of the device, the manufacturer's name (vendor id),

the disk's model name (product id), the disk's version id, the disk serial number, the disk geometry − number of cylinders, heads and sectors, and the last block number on the disk and the block size in bytes. the disk revolution count per minute (rpm's), the number of notches/zones available on the drive (if any)

- −n: Print out the disk drive notch/zone tables if available. This is a sub−option to the −d option. Not all disks appear to correctly report the notch/zone partition tables. For each notch/zone,
- the following is printed: the zone number, the zone's starting cylinder, the zone's starting head, the zone's ending cylinder, the zone's ending head, the zone's starting logical block number, the zone's ending logical block number, the zone's number of sectors per track
- −D: Print out the device names for all disk drives on the system.
- −I: Initialize back−end NCR SCSI chips. This flag may be used in conjunction with any other option and will done first. It has an effect only the first call to hwconf that has not yet used a −d, −D or −I options, or on those chips that have not yet had a −p on the channel associated with that chip.
- −m: Print out major flash and battery backed−up ram addresses (in hex). Additionally print out the size of the RaidRunner configuration area. Eight (8) addresses are printed in order RaidRunner configuration area start and end addresses (FLASH RAM), RaidRunner Husky Scripts area start and end addresses (FLASH RAM), RaidRunner Binary Image area start and end addresses (FLASH RAM), RaidRunner Battery Backed−up area start and end addresses. And the size of the RaidRunner configuration area (in bytes) is then printed.
- −p c.s.l: Probe a single device specified by the given channel, SCSI ID (rank) and SCSI LUN provided in the format "c.s.l". The output of this command is the same as the "−d" option but just for the given device. If the device is not present then nothing will be output and the exit status of the command will be 1.
- −i −p c.s.l: Re−initialize the SCSI device driver specified by the given channel, SCSI ID (rank) and SCSI LUN provided in the format "c.s.l". Typically this command is used when, on a running RaidRunner, a new drive is plugged in, and it will be used prior to the RaidRunner's next reboot.
- −M: Set the boottime memory. This option is executed internally by the controller at boot time and has no function (or effect) executed at any other time.
- −s: Print the 12 character serial number of the RaidRunner.
- −S: Issue SCSI spin up commands to all backends as quickly as possible. This option is intended for use at power−on stage only.
- −t: Probe the temperature monitor returning the internal temperature of the RaidRunner in degrees Celsius.
- −T: Print the temperatures being recorded by the hardware monitoring daemon (hwmon).
- −P: For both AC and DC power supplies, print the number of each present and the state of each supply. The state will be printed as ok or flt depending on whether the PSU is working or faulty.
- −W: This option will wait until all possible backends have spun up. It is used in conjunction with

- NOTES : The order of printing the disk information is by SCSI ID (rank), by channel, by SCSI LUN.

13.31 HWMON − monitoring daemon for temperature, fans, PSUs.

- SYNOPSIS: hwmon [−t seconds] [−d]
- DESCRIPTION: hwmon is a hardware monitoring daemon. It periodically probes the status of certain elements of a RaidRunner and if an out−of−band occurrence happens, will cause the alarm to

sound or light up fault leds as well as saving a message in the system log. Depending on the model of RaidRunner, the elements monitored are temperature, fans and power supplies. When an out−of−band occurrence is found, hwmon will reduce the time between probes to 5 seconds. If a buzzer is the alarm device, then the buzzer will turn on for 5 seconds then off for 5 seconds and repeat this cycle until the buzzer is muted or the occurrence is corrected. If the RaidRunner model supports a buzzer muting switch, then the buzzer will be muted if the switch is pressed during a cycle change as per the previous paragraph. When hwmon recognizes the mute switch it will beep twice.

Certain out−of−band occurrences can be considered to be catastrophic, meaning if the occurrence remains uncorrected, the RaidRunner's hardware is likely to be damaged. Occurrences such as total fan failure and sustained high temperature along with total or partial fan failure are considered as catastrophic. hwmon has a means of automatically placing the RaidRunner into a "shutdown" or quiescent state where minimal power is consumed (and hence less heat is generated). This is done by the execution of the shutdown command after a period of time where catastrophic out−of−band occurrences are sustained. This process is enabled, via the AutoShutdownSecs internal variable. See the internals manual for use of this variable. hwmon can be prevented from starting at boot time by creating the global environment variable NoHwmon and setting any value to it. A warning message will be stored in the syslog.

- OPTIONS:
  - t seconds: Specify the number of seconds to wait between probes of the hardware elements. If this option is not specified, the default period is 300 seconds.
  - −d: Turn on debugging mode which can produce debugging output.
- SEE ALSO: hwconf, pstatus, syslogd, shutdown, internals

13.32 INTERNALS − Internal variables used by RaidRunner to change dynamics of running kernel

- DESCRIPTION: Certain run−time features of the RaidRunner can be manipulated by changing internal variables via the setiv command. The table below describes each changeable variable, it's effect, it's default value and range of values it can be set to. The variables below are run−time features of a RaidRunner and hence are always set to their default values when a RaidRunner boots. Certain variables can be stored as a global environment variable and will over−ride the defaults at boot time. If you create a global environment variable of that variable's name with an appropriate value, it's default value will be over−ridden the next time the RaidRunner is re−booted. Note, that the values of these variables ARE NOT CHECKED when set in the global environment variable tables and, if incorrectly set, will generate errors at boot until deleted or corrected. In the table below, any variable that can have a value stored as a global environment variable is marked with (GEnv)
- write_limit: This variable is the maximum number of 512−byte blocks the cache filesystem will buffer for writes. If this limit is reached all writes to the cache filesystem will be blocked until the cache filesystem has written out (to it's backend) enough blocks to reach a low water mark − write_low_tide. This variable cannot be changed if battery backed−up RAM is available as it is tied to the amount of battery backed−up RAM available. The value of this variable is calculated when the cache is initialized. It's value is dependant on whether battery backed−up RAM is installed in the RaidRunner. If installed, the number of blocks of data that can be saved into the battery backed−up RAM is calculated. If no battery backed−up RAM is present, it's value is set to 75% of the RaidRunner's memory (expressed in a count of 512 byte blocks) then adjusted to reflect the amount of cache requested by configured raid sets. When write_limit is changed then both write_high_tide and write_low_tide are automatically changed to there default values (a function of the value of write_limit).
- write_high_tide: This variable is a high water mark for the number of written−to 512−byte blocks in

the cache. When the number of data blocks exceeds this value, to avoid the cache filesystem from blocking it's front end, the cache flushing mechanism continually flushes the cache buffer until the amount of unwritten (to the backend) cache buffers is below the low water mark (write_low_tide). This value defaults to 75% of write_limit. This variable can have values ranging from write_limit down to write_low_tide. It is recommended that this variable not be changed.

- write_low_tide: This variable is a low water mark for when the cache flushing mechanism is continually flushing data to it's backend. Once the number of written−to cache blocks yet to be flushed equals or is less than this value, the sustained flushing is stopped. This value defaults to 25% of write_limit. This variable can have values ranging from write_high_tide−1 down to zero (0). It is recommended that this variable not be changed.
- cache_nflush: This variable is the number of cache buffers (not 512−byte data blocks) that the cache flushing mechanism will attempt to write out in one flush cycle. Adjusting this value may improve performance on writes depending of the size of the cache buffers and type of disk drives used in the raid set backends. The default value is 128. It's value can range from 2 to 128.
- cache_nread: This variable is the number of cache buffers (not 512−byte data blocks) that the cache reading mechanism will attempt to read out in one read cycle. Adjusting this value may improve performance on reads depending of the size of the cache buffers and type of disk drives used in the raid set backends. The default value is 128. It's value can range from 2 to 128.
- cache_wlimit: This variable is the number of cache buffers (not 512−byte data blocks) that the cache flushing mechanism will attempt coalesce into a single sequential write. It is different to cache_nflush in that cache_nflush is the total number of cache buffers that can be written in a single cache flush cycle and these buffers can be non sequential whereas cache_wlimit is a limit on the number of sequential cache buffer's that can be written with one write. Adjusting this value may improve performance on writes depending of the size of the cache buffers and type of disk drives used in the raid set backends. The default value is 128. It's value can range from 2 to 128.
- cache_fperiod (GEnv): By default, the cache flushes any data to be written every 1000 milliseconds (unless it's forced to by the fact that the cache is getting full and then it flushes the cache and resets the timer). You can vary this flushing period by setting this variable. Given you have a large number of sustained reads and minimal writes, then you may want to delay the writes out of cache to the backends as long as possible. Note, that by setting this to a high value, you run the risk of loosing what you have written. The default value is 1000 milliseconds (i.e 1 second). It's value can range from 500ms to 300000ms.
- scsi_write_thru (GEnv): By default all writes (from a host) are buffered in the RaidRunner's cache and are flushed to the backend disks periodically. When battery backed−up RAM is available then this results in the most efficient write throughput. If no battery backed−up RAM is available or you do not want to depend on writes being saved in battery backed−up RAM in event of a power failure you can force the RaidRunner to write data straight thru to the backends prior to returning an OK status to the host. This essentially provides a write−thru cache. The default value of this variable is 0 − write−thru mode is DISABLED. The values this variable can take are
    - 0 − DISABLE write−thru mode, or
    - 1 − ENABLE write−thru mode.
- scsi_write_fua (GEnv): This variable effects what is done when the FUA (Force Unit Access) bit is set on a SCSI WRITE−10 command. When this variable is enabled and a SCSI WRITE−10 command has the FUA bit set is processed then the data is written directly thru the cache to the backend disks. If the variable is disabled, then the setting of the FUA bit on SCSI WRITE−10 commands is ignored. The default value for this variable is disabled (0) if battery backed−up RAM is present, or enabled (1) if battery backed−up RAM is NOT present. The values this variable can take are
    - 0 − IGNORE FUA bit on SCSI WRITE−10 commands, or
    - 1 − ACT on FUA bit on SCSI WRITE−10 commands.
- scsi_ierror (GEnv): This variable controls what is done when the RaidRunner receives a Initiator

Detected Error message on a SCSI host channel. If set (1), cause an Check Condition, If NOT set (0), follow the SCSI−2 standard and re−transmit the Data In / Out phase. The default value is 0. The values this variable can take are

    - 0 − follow SCSI−2 standard
    - 1 − ignore the SCSI−2 standard and cause a Check Condition.
- scsi_sol_reboot (GEnv): Determines whether to auto−detect a Solaris reboot and the clear any wide mode negotiations. If set (1), detect a Solaris reboot and clear wide mode. If NOT set (0), follow the SCSI−2 standard and not clear wide mode. The default value is 0. The values this variable can take are
    - 0 − follow SCSI−2 standard
    - 1 − ignore the SCSI−2 standard and clear wide mode.
- scsi_hreset (GEnv): Determines whether to issue a SCSI bus reset on host ports after power−on. If set (1), then a SCSI bus reset is done on the host port when starting the first smon/stargd process on that port. If NOT set (0), nothing is done. The default value is 0. The values this variable can take are
    - 0 − don't issue SCSI bus resets on power−on.
    - 1 − issue SCSI bus resets on power−on when the first smon/stargd process is started.
- scsi_full_log (GEnv): Determines whether or not stargd reports, via syslog, a Reset Check condition on Read, Write, Test Unit Ready and Start Stop commands. This reset check condition is always set when a RaidRunner boots or the raid detects a scsi−bus reset. Note that this variable only suppresses the logging of this Check condition into syslog, it does not effect the response to the host of this and any Check condition. If set (1), then all stargd detected reset Check condition error messages are logged. If NOT set (0), these messages are suppressed The default value is 0. The values this variable can take are
    - 0 − suppress logging these messages
    - 1 − log all messages.
- scsi_ms_badpage (GEnv): Determines whether or not stargd reports, via syslog, that it has received a non−supported page number in a MODE SENSE or MODE SELECT command it receives from a host. Note that stargd will issue the appropriate Check condition to the host ("Invalid Field in CDB") irrespective of the value of this variable. If set (1), then all stargd detected non−supported page numbers in MODE SENSE and MODE SELECT commands will be logged. If NOT set (0), these messages are suppressed The default value is 0. The values this variable can take are
    - 0 − suppress logging these messages
    - 1 − log all messages.
- scsi_bechng (GEnv): Determines whether or not the raid reports backend device parameter change errors. In a multi controller environment, backends are probed and some of their parameters are changed by a booting controller. This will generate parameter change mode sense errors. If cleared (0), then all parameter change errors will NOT be logged. If set (1), these messages are logged like any other backend error. The default value is 0. The values this variable can take are
    - 0 − suppress logging these messages
    - 1 − log all messages.
- scsi_dnotch (GEnv): Some disk drives take an inordinate amount of time to perform mode select commands. One set of information a RaidRunner will obtain from a device backend are the disk notch pages (if present). As this is for information only, then to reduce the boot time of a RaidRunner you can request that disk notches are not obtained. If cleared (0), backend disk notch information is not probed for. If set (1), then backend disk notch information is probed for. The default value is 1. The values this variable can take are:
    - 0 − don't probe for notch pages
    - 1 − probe for notch pages
- scsi_rw_retries (GEnv): Specify the number of read or write retries to perform on a device backend before effecting an error on the given operation. Note that ALL retries are reported via syslog. The default value is 3. It's value can range from 1 to 9.

- scsi_errpage_r (GEnv): Specify the number of internal read retries that a disk backend is to perform before reporting an error (to the raid). Setting this variable causes the Read Retry Count field in the Read−Write Error Recovery mode sense page. A value of −1 will cause the drive's default to be used. The default value is −1. It's value can range from −1 (use disk's default) or from 0 to 255.
- scsi_errpage_w (GEnv): Specify the number of internal write retries that a disk backend is to perform before reporting an error (to the raid). Setting this variable causes the Write Retry Count field in the Read−Write Error Recovery mode sense page. A value of −1 will cause the drive's default to be used. The default value is −1. It's value can range from −1 (use disk's default) or from 0 to 255.
- BackFrank: Specify the SCSI−ID of the first rank of backend disks on a RaidRunner. This variable should never be changed and is for informative purposes only. The default value is dependant on the model of RaidRunner being run. The values this variable can take are
    - 0 − the first rank SCSI−ID will be 0
    - 1 − the first rank SCSI−ID will be 1
- raid_drainwait (GEnv): Specify the number of milliseconds a raidset is to delay, before draining all backend I/O's when a backend fails. Setting this variable to a lower value will speed up the commencement of any error recovery procedures that would be performed on a raid set when a backend fails. The default value is 500 milliseconds. It's value can range from 50 to 10000 milliseconds.
- EnclosureType: Specify the enclosure type a raid controller is running within. This variable should never be changed and is for informative purposes only. The default value is dependant on the model of RaidRunner being run. The values this variable can take are integers starting from 0.
- fmt_idisc_tmo (GEnv): Specify the SCSI command timeout (in milliseconds) when a SCSI FORMAT command is issued on a backend. Disk drives take different amounts of time to perform a SCSI FORMAT command and hence a timeout is required to be set when the command is issued. As certain drives may take longer to format than the default timeout you can change it. The default value is 720000 milliseconds. It's value can range from 200000 to 1440000 milliseconds.
- AutoShutdownSecs (GEnv): Specify the number of seconds the RaidRunner should monitor catastrophic hardware failures before deciding to automatically shutdown. A catastrophic failure is one which will cause damage to the RaidRunner's hardware if not fixed immediately. Failures like all fans failing would be considered catastrophic. A value of 0 seconds (the default) will disable this feature, that is, with the exception of logging the errors, no action will occur. See the shutdown and hwmon for further details. The default value is 0 seconds. It's value can range from 20 to 125 seconds.
- SEE ALSO: setiv, getiv, syslog, setenv, printenv, hwmon, shutdown

13.33 KILL − send a signal to the nominated process

- SYNOPSIS: kill [−sig_name] pid
- DESCRIPTION: kill sends a signal to the process nominated by pid. If the pid is a positive number then only the nominated process is signaled. If the pid is a negative number then the signal is sent to all processes in the same process group as the process with the id of −pid. The switch is optional and if not given a SIGTERM (software termination signal) is sent. If the sig_name switch is given then it should be one of the following (lower case) abbreviations. Only the first 3 letters need to be given for the signal name to be recognized. Following each abbreviation is a brief explanation and the signal number in brackets: null − unused signal [0]

hup − hangup [1]

int − interrupt (rubout) [2]

quit − quit (ASCII FS) [3]

kill − kill (cannot be caught or ignored) [4]

pipe − write on a pipe with no one to read it [5]

alrm − alarm clock [6]

term − software termination signal [7]

cld − child process has changed state [8]

nomem − could not obtain memory (from heap) [9]

You cannot kill processes whose process id is between 0 and 5 inclusive. These are considered sacrosanct − hyena, init and console reader/writers.

* SEE ALSO: K9kill

13.34 LED− turn on/off LED's on RaidRunner

* SYNOPSIS:
    * led
    * led led_id led_function
* DESCRIPTION: led uses the given led_id to identify the LED to manipulate based on the led_function. When no arguments are given, an internal LED register is printed along with the current function the onboard LEDS, led1 and led2 are tracing. If a undefined led_id is given, the led command silently does nothing and returns NULL. If an incorrect number of arguments or invalid led_function is given a usage message is printed. Depending on the RaidRunner model the led_id can be one of
    * led1 − LED1 on the RaidRunner controller itself
    * led2 − LED2 on the RaidRunner controller itself
    * Dc.s.l − Device on channel c, scsi id s, scsi lun l
    * status − the status LED on the RaidRunner
    * io − the io LED on the RaidRunner

  and led_function can be one of
* on − turn on the given LED
* off − turn off the given LED
* ok − set the given LED to the defined OK state
* faulty − set the given LED to the defined FAULTY state
* warning − set the given LED to the defined WARNING state
* rebuild − set the given LED to the defined REBUILD state
* tprocsw − set the given LED to trace kernel process switching
* tparity − set the given LED to trace I/O parity generation
* tdisconn − set the given LED to trace host interface disconnect activity
* pid − set the given LED to trace the process pid as it runs

Different models of RaidRunner have various differences in number of LED's and their functionality. Depending on the type of LED, the ok, faulty, warning and rebuild functions perform different functions. See your RaidRunner's Hardware Reference manual to see what LED's exist and what different functions do.

* NOTES: Tracing activities can only occur on the `onboard` leds (LED1, LED2).

* SEE ALSO: lflash

13.35 LFLASH− flash a led on RaidRunner

* SYNOPSIS: lflash led_id period
* DESCRIPTION: lflash uses the given led_id to identify the LED to flash every period seconds. If a undefined led_id is given, the led command silently does nothing and returns NULL. Depending on the RaidRunner model the led_id can be one of: led1 − LED1 on the RaidRunner controller itself

led2 − LED2 on the RaidRunner controller itself

Dc.s.l − Device on channel c, scsi id s, scsi lun l

status − the status LED on the RaidRunner

io − the io LED on the RaidRunner

* NOTE: The number of seconds must be greater than or equal to 2.
* SEE ALSO: led

13.36 LINE − copies one line of standard input to standard output

* SYNOPSIS: line
* DESCRIPTION: line accomplishes the one line copy by reading up to a newline character followed by a single K9write.
* SEE ALSO: K9read, K9write

13.37 LLENGTH − return the number of elements in the given list

* SYNOPSIS: llength list
* DESCRIPTION: llength returns the number of elements in a given list.
* EXAMPLES: Some simple examples:

```
        set list D1 D2 D3 D4 D5 # create the list
set len `llength $list' # get it's length

echo $len

5

set list {D1 D2 D3 D4 D5} {D6 D7}  # create the
 list

set len `llength $list' # get it's length

echo $len

2
```

```
set list {} # create an empty list

set len `llength $list' # get it's length

echo $len

0
```

13.38 LOG − like zero with additional logging of accesses

- SYNOPSIS: bind −k {log fd error_rate tag} bind_point
- DESCRIPTION: log is a special file that when written to is a infinite sink of data (i.e. anything can be written to it and it will be disposed of quickly). When log is read it is an infinite source of zeros (i.e. the byte value 0). The log file will appear in the K9 namespace at the bind_point. Additionally, ASCII log data is written to the file associated with file descriptor fd. error_rate should be a number between 0 and 100 and is the percentages of errors (randomly distributed) that will be reported (as an EIO error) to the caller. Each line written to fd will have tag appended to it. There is one line output to fd for each IO operation on the log special file. The first character output is "R" or "W" indicating a read or write. The second character is blank if no error was reported and "*" if one was reported. Next (after a white space) is a (64 bit integer) offset into the file of the start of the operation, followed by the size (in bytes) of that operation. The line finishes with the tag.
- EXAMPLE: Bind a log special file at "/dev/log" that writes log information to standard error. Each line written to standard error has the tag string "scsi" appended to it. Approximately 30% of reads and writes (i.e. randomly distributed) return an EIO error to the caller. This is done as follows:

```
      bind "log 2 30 scsi" /dev/log
dd if=/dev/zero of=/dev/log count=5 bs=512

W  0000000000 512        scsi

W  0000000200 512        scsi

W  0000000400 512        scsi

W* 0000000600 512        scsi

Write failed.

4+0 records in

3+0 records out
```

- SEE ALSO: zero

13.39 LRANGE − extract a range of elements from the given list

- SYNOPSIS: lrange first last list
- DESCRIPTION: lrange returns a list consisting of elements first through last of list. 0 refers to the first element in the list. If first is greateR THAN last then the list is extracted in reverse order.
- EXAMPLES: Some simple examples:

```
set list D1 D2 D3 D4 D5 # create the list
```

```
set subl `lrange 0 3 $list' # extract from indices 0 to 3

echo $subl

D1 D2 D3 D4

set subl `lrange 3 1 $list' # extract from indices 3 to 1

echo $subl

D4 D3 D2

set subl `lrange 4 4 $list' # extract from indices 0 to 3

echo $subl # equivalent to get 4 $list

D5

set subl `lrange 3 100 $list'

echo $subl

D4 D5
```

13.40 LS − list the files in a directory

- SYNOPSIS: ls [−l] [directory...]
- DESCRIPTION: ls lists the files in the given directory on standard out. If no directory is given then the root directory (i.e. "/") is listed. Each file name contained in a directory is put on a separate line. Each listing has a lead−in line stating which directory is being shown. If there is more than one directory then they are listed sequentially separated by a blank line. If the "−l" switch is given then every listed file has data such as its length and the file system it belongs to shown on the same line as its name. See the stat command for more information. ls is not an inbuilt command but a husky script which utilizes cat and stat. The script source can be found in the file "/bin/ps".
- SEE ALSO: cat, stat

13.41 LSEARCH − find the a pattern in a list

- SYNOPSIS: lsearch pattern list
- DESCRIPTION: lsearch returns the index of the first element in list that matches pattern or −1 if none. 0 refers to the first element in the list
- EXAMPLES: Some simple examples:

```
set list D1 D2 D3 D4 D5 # create the list

set idx `lsearch D4 $list' # get index of D4 in list

echo $idx

3

set idx `lsearch D1 $list' # get index of D1 in list

echo $idx
```

```
0

set idx `lsearch D8 $list' # get index of D8 in list

echo $idx # equivalent to get 4 $list

-1
```

13.42 LSUBSTR − replace a character in all elements of a list

- SYNOPSIS: lsubstr find_char replacement_char list
- DESCRIPTION: lsubstr returns a list replacing every find_ch character found in any element of the list with the replacement_char character. replacement_char can be NULL which effectively deletes all find_char characters in the list.
- EXAMPLES: Some simple examples:

```
set list D1 D2 D3 D4 D5 # create the list

set subl `lsubstr D x $list' # replace all D's with x's

echo $subl

x1 x2 x3 x4 x5

set subl `lsubstr D {} $list' # delete all D's

echo $subl

1 2 3 4 5

set list −L −16 # create a list with embedded braces

set subl `lsubstr {} $list' # delete all open braces

set subl `lsubstr {} $subl' # delete all close braces

echo $subl

−L 16
```

13.43 MEM − memory mapped file (system)

- SYNOPSIS: bind −k {mem first last [r]} bind_point
- DESCRIPTION: mem allows machine memory to be accessed as a single K9 file (rather than a file system). The host system's memory is used starting at the first memory location up to and including the last memory location. Both first and last need to be given in hexadecimal. If successful the mem file will appear in the K9 namespace at the bind_point. The stat command will show it as a normal file with the appropriate size (i.e. last − first + 1). If the optional "r" is given then only read−only access to the file is permitted. In a target environment mem can usefully associate battery backed−up RAM (or ROM) with the K9 namespace. In a Unix environment it is of limited use (see unixfd instead). In a DOS environment it may be useful to access memory directly (IO space) but for accessing the DOS console see doscon. When mem is associated with the partition of Flash RAM

that stores the husky scripts, which is stored compressed, reading from that page will automatically decompress and return the data as it is read. When mem is associated with the writable partitions of Flash RAM (configuration partition, husky script partition and main binary partition) a write to the start of any partition will erase that partition.

- SEE ALSO: ram
- BUGS: Only a single file rather than a file system can be bound.

13.44 MDEBUG − exercise and display statistics about memory allocation

- SYNOPSIS: mdebug [off|on|trace|p|m size|f ptr|c nel elsize|r ptr size]
- DESCRIPTION: mdebug can be used to directly allocate and free memory. mdebug will also print (to standard output) information about the current state of memory allocation. With out any given options a brief five line summary of memory usage is printed, e.g.

```
      : raid; mdebug
Mdebug is off

nreq-nfree=87096-82951=4145(13905745)

size=15956672/16150000

waste=1%/2%

list=4251/8396

: raid;
```

The first line indicates the debug mode, either off, on or trace. The second line indicates the number times a request for memory is made (to Mmalloc() or Mcalloc() and related functions) and the number of times the memory allocator is called to free memory (via Mfree()). The difference between these first two numbers is the total number of currently allocated blocks of memory, with the number between the '(' and ')' being the total memory requested. Note that the amount of memory actually assign may be more than requested.

The third line indicates the amount of memory being managed. The second number is the total memory man aged (i.e. left over after loading the statically allocated text, data and bss space). The first number is that left over after various memory allocation tables have been subtracted out from that afore mention number. The fourth line is the total amount of extra memory assigned to requests in excess of the actual requested memory as compared with the totals on line 3.

The fifth line relates to the list of currently allocated memory. The first number is the number of free entries left and the second is the maximum table size. Note that the number of currently allocated blocks (third number on line 2) when added to the first number on line 5 gives the second number on line 5.

- OPTIONS:
    - p: Prints the above mentioned five line summary and then the free list.
    - P: Prints all the above plus dumps the list of currently allocated memory.
    - PP: Prints all the above plus the free bitmap.

The above three options can generate copious output and require a detailed knowledge of the source to understand their meaning.

- off: Turns off memory allocation debugging. This is the default condition after booting.
- on: Turns on memory allocation assertion checking.
- trace: Turns on memory allocation assertion checking and traces every memory allocation / deallocation.
- m: Uses Mmalloc() to allocate a block of memory of size bytes.
- f: Uses Mfree() to de−allocate a block of memory addressed by ptr.
- c: Uses Mcalloc() to allocate a contiguous block of memory consisting of nel elements each of size bytes.
- r: Uses Mrealloc() to re−allocate a block of previously allocated memory, ptr, changing the allocated size to be size bytes.
- SEE ALSO: Unix man pages on malloc()

13.45 MKDIR − create directory (or directories)

- SYNOPSIS: mkdir [directory_name ...]
- DESCRIPTION: mkdir creates the given directory (or directories). If all the given directories can be created then NIL is returned as the status; otherwise the first directory that could not be created is returned (and this command will continue trying to create directories until the list is exhausted). A directory cannot be created with a file name that previously existed in the enclosing directory.

13.46 MKDISKFS − script to create a disk filesystem

- SYNOPSIS: mkdiskfs disk_directory_root disk_name
- DESCRIPTION: mkdiskfs is a husky script which is used to perform all the necessary commands to create a disk filesystem given the root of the disk file system and the name of the disk.
- OPTIONS :
    - disk_directory_root: Specify the directory root under which the disk filesystems are bound. This is typically /dev/hd.
    - disk_name: Specify the name of the disk in the format Dc.s.l where c is the channel, s is the scsi id (or rank) and l is the scsi lun of the disk.

  After parsing it's arguments mkdiskfs creates the disk filesystem's bind point and binds in the disk at that point. set.
- SEE ALSO: rconf, scsihdfs

13.47 MKHOSTFS − script to create a host port filesystem

- SYNOPSIS: mkhostfs controller_number host_port host_bus_directory
- DESCRIPTION: mkhostfs is a husky script which is used to perform all the necessary commands to create a host port filesystem on the given RaidRunner controller given the root of the host port file systems and the host port number.
- OPTIONS:
    - controller_number: Specify the controller on which the host port filesystem is to be created.
    - host_port: Specify the host port number to create the filesystem for.
    - host_bus_directory: Specify the directory root under which host filesystems are bound. This is typically /dev/hostbus. After parsing it's arguments mkhostfs finds out what SCSI ID the host port is to present (see hconf and then binds in the host filesystem. set.
- SEE ALSO: hconf, scsihpfs

13.48 MKRAID − script to create a raid given a line of output of rconf

- SYNOPSIS: mkraid `rconf −list RaidSetName'
- DESCRIPTION: mkraid is a husky script which is used to perform all the necessary commands to create and enable host access to a given Raid Set. The arguments to mkraid is a line of output from a rconf −list command. After parsing it's arguments mkraid checks to see if a reconstruction was being performed when the RaidRunner was last operating, and if so, notes this. It then creates the raid filesystem (see mkraidfs) and adds a cache frontend to the raid filesystem. It then creates the required host filesystems (see mkhsotfs) and finally, if a reconstruction had been taking place when the RaidRunner was last operating, it restarts a reconstruction.
- NOTE: This husky script DOES NOT enable target access (stargd) to the raid set it creates.
- SEE ALSO: rconf, mkraidfs, mkhostfs

13.49 MKRAIDFS − script to create a raid filesystem

- SYNOPSIS: mkraidfs −r raidtype −n raidname −b backends [−c chunk] [−i iomode] [−q qlen] [−v] [−C capacity] [−S]
- DESCRIPTION: mkraidfs is a husky script which is used to perform all the necessary commands to create a Raid filesystem.
- OPTIONS:
    - −r raidtype: Specify the raid type as raidtype for the raid set. Must be 0, 1, 3 or 5.
    - −n raidname: Specify the name of the raid set as raidname.
    - −b backends: Specify the comma separated list of the raid set's backends in the format used by rconf.
    - −c iosize: Optionally specify the IOSIZE (in bytes) of the raid set.
    - −i iomode: Optionally specify the raid set's iomode − read−write, read−only, write−only.
    - −q qlen: Optionally specify the raid set's queue length for each backend.
    - −v: Enable verbose mode which prints out the main actions (binding, engage commands) as they are performed.
    - −C capacity: Optionally specify the raid set's size in 512−byte blocks.
    - −S: Optionally specify that spares pool access is required should a backend fail.

  After parsing it's arguments mkraidfs creates the Raid Set's backend filesystems, typically, disks (see mkdisfs) taking care of failed backends. It then binds in the raid filesystem and engages the backends into the filesystem. If spares access is requested, it enables the autorepair feature of the raid set.
- SEE ALSO: rconf, mkraidfs, mkhostfs, mkdiskfs, raid[0135]fs

13.50 MKSMON − script to start the scsi monitor daemon smon

- SYNOPSIS: mksmon controllerno hostport scsi_lun protocol_list
- DESCRIPTION: mksmon is a husky script which is used to perform all the necessary commands to start the scsi monitor daemon smon given the controller number, hostport, scsi lun, and the block protocol list. Typically, mksmon, is run with it's arguments from the output of a mconf −list command.
- OPTIONS:
    - controllerno: Specify the controller on which the scsi monitor daemon is to be run.
    - hostport: Specify the host port through which the scsi monitor daemon communicates.

- scsi_lun: Specify the SCSI LUN the scsi monitor daemon is to respond to.
- protocol_list: Specify the comma separated block protocol list the scsi monitor daemon is to implement.

After parsing it's arguments mksmon checks to see if it's already running and issues a message if so and exits. Otherwise, it creates the host filesystem (mkhostfs), creates a memory file and set of fifo's for smon to use and finally starts smon set.

- SEE ALSO: smon, mconf, mkhostfs, fifofs

13.51 MKSTARGD − script to initialize a scsi target daemon for a given raid set

- SYNOPSIS: mkstargd `rconf −list raidname'
- DESCRIPTION: mkstargd is a husky script which is used to perform all the necessary commands to start and initialize the scsi target daemon stargd for a given raid set. Typically, mkstargd, is run with it's arguments from the output of a rconf −list command. After parsing it's arguments mkstargd checks to see if it's already running and issues a message if so and exits. Otherwise, it creates the host filesystem (mkhostfs), then starts the scsi target daemon (stargd) for the given raid set. stargd is started in a mode that responds to non−medium access SCSI commands and sets a state of "spinning up". Typically stargd's are started as soon as possible but do not allow medium access commands until the underlying raid sets have been created and then are flagged as "spun up" (via mstargd −o command) which allows medium access.
- SEE ALSO: stargd, rconf, mkhostfs

13.52 MSTARGD − monitor for stargd

- SYNOPSIS: mstargd [−a] [−d level] [−h] [−hr] [−hw] [−l] [−m] [−n] [−o offset] [−R] [−s] [−t] [−v] [−W] [−z] [−Z spinstate] [−U] [−irgap nblks] pid
- DESCRIPTION: mstargd modifies the state or prints out information about the stargd daemon with the given pid. If such a process exists and no optional switches are given then the current debug level of that stargd daemon is printed to standard out by this call. mstargd works by looking for file named "/mon/stargd/pid". If it is not found or the pid does not represent an existing process then mstargd exits with an appropriate error message. If it is found then it is assumed to contain a reference into the associated stargd process's state (and statistics) structure. The "reference" for target hardware such as the raid (without memory management) is a memory address. The "reference" for Unix machines could be a shared memory identifier or a memory address (depending on how K9 processes are mapped to Unix processes). mstargd performs its monitoring (or modifying) task then exits immediately. A sanity check is performed on the associated stargd process's state (and statistics) structure before it is used. Operations that take a little time (e.g. "−h", "−s" and "−t") take an internal copy of this state structure. mstargd is designed to have no detrimental effect on a running stargd daemon. Note that the 3 modifying operations (i.e. "−o", "−d" and "−z") have no adverse effect either.
- OPTIONS
    - −a: This option has the same effect as giving mstargd the options, "−h −l −m −s −t".
    - −d level: This option will change the debug level of the nominated stargd daemon to the given level. The debug levels are:
        - 0 = no debug messages
        - 1 = debug messages on all non−read/write commands
        - 2 = debug messages on all commands
    - −h: This option will output a histogram for reads and a separate one for writes. The histogram currently consists of a header line followed by multiple lines. Each line has the

number of blocks in its 1st column, the number of invocations in the 2nd column and the cumulative time in the SCSI data phase in the 3rd column. Only lines with non−zero invocations are output. Block number 257 (if present) will be the last line and records all commands requesting 257 or more blocks.

- −hr: This option only prints out the histogram for reads.
- −hw: This option only prints out the histogram for writes.
- −l: This option prints out the stargd read lookahead statistics.
- −m: This option prints out the moniker (i.e name) of the raid set indicated by the given pid.
- −n: This option toggles the collection of statistics by the indicated stargd process.
- −o offset: This option informs the daemon that reads and writes into it's store are to be offset by the given number of blocks. The default value for offset is 0. The given offset must be in the range 0 to (2**32 − 1). The typical block size is 512 bytes. To simplify writing out large numbers certain suffixes can be used, see suffix. Additionally, this option informs the daemon that it's store is now available for access by setting it's spin state to 1. See "−Z" option below.
- −R: This option sets the write−protect flag which will result in any write command issued to the stargd process (as indicated by the given pid) to return a check condition with the sense key set to "DATA PROTECT" and the additional sense key set to "WRITE PROTECT".
- −s: This option prints the current state of the SCSI command state machine.
- −t: This option prints a row each for read(6), read(10), write(6), write(10) and others. These 5 rows represent a categorization of all incoming SCSI commands. Each row contains the number of invocations and the number of errors detected. Errors are divided into 2 categories: type 1 for situations when a "Check Condition" status is returned, and type 2 when some other failed status is returned (e.g. "Command terminated" or "Reservation conflict").
- −v: This option prints out the histogram (−h, −hw, −hr) and SCSI command- summary (−t) in vector (or line) form.
- −U: This option clears any SCSI−2 reservations set on the scsi target specified by the given pid. WARNING this clearance "controller system wide".
- −W: This option clears the write−protect flag which enables write commands issued to the stargd process (as indicated by the given pid) to write data.
- −z: This option zeroes the internal tables used by the histograms.
- −irgap nblks: Specify the inter−read gap, nblks, (in blocks). When sequential reads arrive from a host there may be a small gap between successive reads. Normally the lookahead algorithm will ignore these gaps providing they are no larger than the average length of the group of sequential reads that have occurred. By specifying this value, you can increase this gap.
- −Z spinstate: This option will change the spin state of the nominated stargd daemon to the given spinstate. The spin states are (State, Description, ASC,ASCQ):
    - 0, LOGICAL UNIT IS IN PROCESS OF BECOMING READY, 0x04, 0x01
    - 1, Logical unit is ready − medium access commands allowed, −, −
    - 2, LOGICAL UNIT NOT READY, MANUAL INTERVENTION REQUIRED, 0x04, 0x03
    - 3, LOGICAL UNIT HAS NOT SELF−CONFIGURED YET, 0x3e, 0x00
    - 4, LOGICAL UNIT HAS FAILED SELF−CONFIGURATION, 0x4c, 0x00
    - stargd's spin state is used to describe the condition of the tar- get whilst is is NOT READY. When stargd is not ready to accept SCSI−2 medium access commands it returns a CHECK CONDITION status to all medium access commands, sets the mode sense key to NOT READY and the additional mode sense code and code qualifier to values depending in the spin state value. Typically, when a RaidRunner boots it requires time to create the configured Raid Sets, although it needs to start the

scsi target daemons (stargd) as soon as possible. It does start all the required stargds and set's their spin state to 0. Once the raid sets have been built and are linked to their stargds, the spin state is set to 1 meaning it will allow and process SCSI−2 medium access commands. (See the "−o" option above). The additional spin states of 2, 3 and 4 can be used by systems with intelligent SCSI drivers in high availability environments.

- SEE ALSO: stargd

13.53 NICE − Change the K9 run−queue priority of a K9 process

- SYNOPSIS: nice pid priority
- DESCRIPTION: nice will change the run−queue priority of the process given by pid to priority.
- OPTIONS:
  - pid: This is the process identifier of the K9 process whose run−queue priority is to change.
  - priority: This the the priority to set. Priorities range from 0 (lowest) to 9 (highest).
- SEE ALSO: K9setpriority

13.54 NULL− file to throw away output in

- SYNOPSIS: bind −k null bind_point
- DESCRIPTION: null is a special file that when written to is a infinite sink of data (i.e. anything can be written to it and it will be disposed of quickly). When null is read it is an infinite source of end−of−files. The null file will appear in the K9 names− pace at the bind_point.
- EXAMPLE Husky installs a null special file as follows:

```
        bind null /dev/null
```

- SEE ALSO zero, log

13.55 PARACC − display information about hardware parity accelerator

- SYNOPSIS: paracc
- DESCRIPTION: paracc will print (to standard output) information about the hardware parity accelerator, if installed. The main output line of interest to all except those debugging the RaidRunner is the first line which displays, in bytes, the size of the memory on the hardward parity accelerator. IE

```
        Parity Memory available : 1048576
PAccLock@0xBF368{own=-1,cnt=1,pvt=0x0,nwait=0,name="PAc"}

Request failures: 0, Max Usage: 2, Alloc: 1, Free: 31

have_paracc is 1.

Req Fails: 0 0 0 0 0 0 0 0 0 0
```

All other lines are only informative for debugging purposes. If there is no accelerator present, then the Parity Memory available will be 0.

13.56 PEDIT − Display/modify SCSI backend Mode Parameters Pages

* SYNOPSIS:
    * pedit page_code c.s.l
    * pedit page_code c.s.l byte_modifier_list
* DESCRIPTION: pedit will either report the SCSI pages of mode parameters for a given page − page_code on a given SCSI backend device c.s.l and or allow you to change the mode parameters. In it's first form, pedit, for a given page code, will print five lines. The first is a header for ease of reading, the second will be the DEFAULT mode parameters, the second will be CHANGEABLE bitmask values, the third will be the CURRENT mode parameters and the last will be the SAVED mode parameters. In it's second form, pedit, for a given page code and device, will print the page codes but also will apply the byte_modifier_list to either the CURRENT or SAVED mode parameters. The supported SCSI pages are 0x1 ERROR RECOVERY page

0x2 DISCONNECT page

0x3 FORMAT page

0x4 GEOMETRY page

0x8 CACHE page

0xc NOTCH page

0xa CONTROL page

* OPTIONS
    * page_code: Specify the SCSI Page Code in hex.
    * c.s.l: Identify the disk device to select by specifying it's channel, SCSI ID (rank) and SCSI LUN provided in the format "c.s.l"
    * byte_modifier_list: The byte_modifier_list is of the form

```
C|Sbyte_no:set_val: clr_val,byte_no:set_val:clr_val,...
```

  where the C or S prefix specifies whether you want to change the CURRENT or SAVED mode pages respectively. This prefix is followed by a comma separated list of byte modifiers in the form byte_no:set_val:clr_val where byte_no is the byte number to change, set_val is a mask of which bits within that byte to SET (to 1) and clr_val is a mask of which bits within that byte to CLEAR (to 0).
* SEE ALSO: The mode sense page references in the relevant product manual for the disks used in the RAID.

13.57 PIPE − two way interprocess communication

* SYNOPSIS:
    * bind −k pipe bind_point
    * cat bind_point
    * bind_point/data
    * bind_point/ctl
    * bind_point/data1
    * bind_point/ctl1
* DESCRIPTION: pipe file system associates a one level directory with the bind_point in the K9

namespace. This device allocates two streams which are joined at the device end. bind_point/data and bind_point/ctl are the data and control channels for one stream while bind_point/data1 and bind_point/ctl1 are the data and control channels for the other stream. Data written to one channel is available for reading at the other. Write boundaries are preserved: each read terminates when the read buffer is full or after reading the last byte of a write, whichever comes first.

13.58 PRANKS − print or set the accessible backend ranks for the current controller

- SYNOPSIS
    - pranks
    - pranks rank1 [rank2 ... rankn]
- DESCRIPTION: pranks, without any arguments will print which backend ranks of devices the controller, on which the command is executed, can access. If a rank is not accessible, then a "−1" is printed in place of the rank number (i.e scsi id of that rank). If you wish to set which rank id's a controller is allowed to access then execute this command with those rank id's as it's arguments. When you execute this command to set access to certain (or all) ranks, then the access restrictions (if any) are effective immediately. Additionally, the GLOBAL environment variable BackendRanks is either defined or modified and when you next boot the RaidRunner, the settings you have just created will be set again automatically.
- EXAMPLE: Assume you have a RaidRunner system with four ranks of backend devices (rank 1, 2, 3 and 4) but you want to restrict the controller's access to the first two as you don't have any devices installed in the third and forth ranks. You would execute pranks 1 2
- SEE ALSO: environ, sranks

13.59 PRINTENV − print one or all GLOBAL environment variables

- SYNOPSIS
    - printenv
    - printenv name [name ...]
- DESCRIPTION: printenv prints the value (or list of values) associated with the GLOBAL environment variable name. If multiple GLOBAL environment variables are given, each value is printed on a line of it's own. When a GLOBAL environment variable is a list of values, each value is printed on a line of it's own. When printenv is called with no arguments, all GLOBAL environment variables and their associated value(s) are printed, one per line. If a value is a list then each element of the list is separated with the vertical bar () character.
- NOTE: A GLOBAL environment variable is one which is stored in a non volatile area on the RaidRunner and hence is available between successive power cycles or reboots. These variables ARE NOT the same as husky environment variables. The non volatile area is co−located with the RaidRunner Configuration area. If the given variable name is not a GLOBAL environment variable nothing is printed and no error status is set.
- SEE ALSO: setenv, unsetenv, rconf
- PROC − the process file system (device)
- SYNOPSIS:
    - bind −k proc bind_point
    - cat bind_point
    - cat bind_point/0
    - cat bind_point/1

    - ...
    - cat bind_point/N
    - signal
    - sigpgrp
    - status
- DESCRIPTION: The proc device creates a two level directory below its bind_point. The first level entries are the numbers of the processes that are known currently to K9. The second level entries are the filenames: "signal", "sigpgrp" and "status". It is an error to read from the second level directory files "signal" and "sigpgrp". When "status" is read it returns an ASCII string containing the following fields:
    - process name
    - process identifier (i.e. its pid)
    - this process's parent's pid
    - pid of process group leader
    - process state
    - process priority
    - maximum stack utilization as % of available stack
    - milliseconds of CPU time registered
    - semaphore id (if any) this process is waiting on

  These fields have an appropriate number of spaces between them so they look "reasonable" when output by ps. The process states are listed below:
- I interrupted
- R currently running (or has yielded control)
- S stirred (signaled while waiting)
- W waiting on a semaphore
- Z terminated and parent not waiting

It is an error to write to the second level directory file "status". A signal number or signal name (see kill for a list of signal names − which cannot be abbreviated in this case) may be written to the file "signal". This action will send a signal to the associated process. If a signal number or name is written to the file "sigpgrp" then all the processes in the process group which this process belongs to will receive the signal.

- SEE ALSO: ps, kill

13.60 PS − report process status

- SYNOPSIS: ps
- DESCRIPTION: ps prints information about all running K9 processes on standard out. The information output includes the process name, its process identification number (PID), its parent's PID, process group, process state, the maximum percentage utilization of its stack and the milliseconds of CPU time its has used. The process states are listed below:
    - I interrupted
    - R currently running (or has yielded control)
    - S stirred (signaled while waiting)
    - W waiting on a semaphore
    - Z terminated and parent not waiting
- Currently ps is implemented as a K9 Husky script (rather than a built in command). The script source can be found in the file "/bin/ps". The script utilizes the file system proc.
- EXAMPLE:

```
: raid; ps
```

```
NAME____________________PID__PPID__PGRP_S_P_ST%_TIME(ms)__SEMAPHORE+name

hyena                    0     0     0 R 9  18 385930     deadbeef

init                     1     0     1 W 0   9 90         8009b1a8   pau

SCN2681_reader           4     1     4 W 0   0 0          800702a4   2rd

SCN2681_writer           5     1     5 W 0   0 0          8007029c   2wr

SCN2681_putter           6     1     6 W 0   0 0          800702ac   2tp

DIO_R_drive3_q0        391     1   391 W 0   4 40120       8021a828   Ard

DIO_R_drive0_q0        397     1   397 W 0   4 13420       8007ac64   Ard

DIO_R_drive1_q0        404     1   404 W 0   5 25570       8007b224   Ard

husky                  28      1     1 W 0  10 50         8013a138   pau

cache_flusher          424     1   424 W 0  23 17700       8030c2c4   Cfr

CIO_R_q0               426     1   426 W 0  96 2320        8030d6f4   Ard

CIO_R_q1               427   426   426 W 0  96 2420        8030d6f4   Ard

CIO_R_q2               428   426   426 W 0  96 2410        8030d6f4   Ard

CIO_R_q3               429   426   426 W 0  96 2430        8030d6f4   Ard

CIO_R_q4               430   426   426 W 0  96 2240        8030d6f4   Ard

CIO_R_q5               431   426   426 W 0  96 2130        80c37540   Ard

CIO_R_q6               432   426   426 W 0  96 2300        8030d6f4   Ard

CIO_R_q7               433   426   426 W 0  96 2180        8030d6f4   Ard

smon                    65     1     1 W 0   5 30          8008d5e4   Nsl

DIO_R_drive2_q0        326     1   326 W 0   5 27680       8007b7e4   Ard

/bin/ps                871    28     1 W 0   8 40          80cfd020   pau

stargd                 107     1     1 R 0  48 23990       8007a648   Nsl

starg_107_L_R          119   107   119 W 0   0 0           8018c608   pau
```

- The fields are: process name, process identifier (i.e. its pid), this process's parent's pid, pid of process group leader, process state, process priority, maximum stack utilization as % of available stack, milliseconds of CPU time registered, semaphore id (if any) this process is waiting on along with the internal name of the semaphore. If a process is waiting on a semaphore then the last number is the address of the number it is waiting on.
- SEE ALSO: proc

13.61 PSCSIRES − print SCSI−2 reservation table for all or specific monikers

- SYNOPSIS:
    - pscsires
    - pscsires moniker
- DESCRIPTION: pscsires looks up the Global SCSI−2 Reservation table for a given moniker (see smon or stargd) and prints is current SCSI−2 reservation state. When no moniker is given, all entries in the Global SCSI−2 Reservation table are printed. For each table entry to be displayed, the moniker and four integers are printed − the controller number, host port number, reserved scsi id and reservor scsi id. The combination of controller number, host port number, and reservor scsi id can uniquely identify which host system issued the SCSI−2 Reserve command. The reserved scsi id field is used when a Third−Party reservation has been requested by the host system identified by the other three integers. If all four integers are −1, then no scsi target daemon (smon or stargd) on any controller or host port has reserved the moniker.
- SEE ALSO: smon, stargd, mstargd, SCSI−2 Reserve and Release Command Documentation

13.62 PSTATUS − print the values of hardware status registers

- SYNOPSIS: pstatus
- DESCRIPTION: pstatus will print the names and values of various hardware status registers. Each value is printed on a lineof it's own. Typical hardware status registers are BCDSW_0, BCDSW_1. Values of BCD host port SCSI ID selector switches
    - FANS: Value of Fan status register
    - AC_PWR: Value of AC Power supply status
    - DC_PWR: Value of DC Power supply status
- NOTE: Not all RaidRunner models support the same status registers, so consult you RaidRunner model's hardware reference manual to see which are supported and what their values imply.

13.63 RAIDACTION− script to gather/reset stats or stop/start a raid set's stargd

- SYNOPSIS: raidaction raidname up|down|getstats|getastats|zerostats
- DESCRIPTION: raidaction is a husky script which is used to either start or stop a raid set's scsi target daemon(s) (stargd) or gather/reset statistics about the raid set.
- OPTIONS :
    - raidset: Specify the raid set to perform the action on.
    - up: Start all the scsi target daemons associated with this raid set.
    - down: Stop all the scsi target daemons associated with this raid set.
    - getstats: Print the current statistics stored for the given raid set. The first line of output is prefixed with the string "RAIDSET:" and is the output of the stats command with arguments −r raidname −g. The second line of output is prefixed with the string "CACHE:" and is the output of the stats command with arguments −c raidname −g. The next line(s) are prefixed with the string "STARG: c.h.l" and is the output of the mstargd command with arguments −d 0 −v −h −H stargd_pid which stargd_pid is the process id of the scsi target daemon on controller c, host port h with scsi lun l. A line of this type is printed for each scsi target daemon belonging to this raid set.

    - ♦ getastats: Print the current statistics stored for the given raid set then zero all the stored statistics. By zeroing the stored statistics one can, through repeated timed calls to this code, form an average based on the gathered statistics. The output is the same as for the getstats option.
    - ♦ zerostats: Zero all statistics stored for the given raid set.
  - SEE ALSO: stats, mstargd

13.64 RAID0 − raid 0 device

  - SYNOPSIS:

```
      bind −k {raid0 nbackends} bind_point
echo moniker name=raid_set_name > bind_point/ctl

echo engage drive=driveNum qlen=queueLen fd=aFdNum blksize=blockSize name=backendname

<>[aFdNum] backEnd > bind_point/ctl

echo disengage drive=driveNum > bind_point/ctl

echo access drive=driveNum read−write > bind_point/ctl

echo access drive=driveNum read−only > bind_point/ctl

echo access drive=driveNum write−only > bind_point/ctl

echo access drive=driveNum offline > bind_point/ctl

cat bind_point

ctl

data

repair

stats
```

- DESCRIPTION: raid0 implements a raid 0 device. It has 1 "frontend" (i.e. bind_point/data) and typically multiple "backends" (i.e. one defined by each "engage" message with a new drive number). To associate an internal name (or moniker) with the raid device, send the message "moniker name=internal_name" to the device's control file, bind_point/ctl. This implementation of raid 0 uses nbackends files in its backend. Read and write operations to the frontend (i.e. bind_point/data) must be in integral units of blockSize. Each write of blockSize bytes is written on 1 backend file. The backend "files" referred to here will typically be disks. The name argument allows associates the given backendname string with the appropriate backend. This string will be used in reporting errors on the running raid.

  The queueLen argument must be 1 or greater and sets the maximum number of requests that can be put in a queue associated with each backend file. A daemon is spawned for each backend file to service this queue called async_io. Each backend file first needs to be identified to the raid0 device via the "engage" string sent to bind_point/ctl. If required a file can have its association with this device terminated with a "disengage" string. Once a backend file is engaged its access level can be varied between "read−write", "read−only", "write−only" and "offline" as required. The default is "offline" so in most initialization situations an "access read−write" string needs to be sent to this device. When the file bind_point/ctl is read then a line is output for every engaged backend file indicating its access status (e.g. "drive 3: engaged, read−write"). Also backend files that have been disengaged and not "re−"engaged output a line (e.g. "drive 5: disengaged").

When the file bind_point/stats is read then a line is output which shows the cumulative number of reads and writes performed (including failures) for each backend of the raid device. The format of this line is D0 r0_cnt r0_fails w0_cnt w0_fails; D1 r1_cnt r1_fails w1_cnt w1_fails; ... which indicates that backend 0 (typically the drive0) has made r0_cnt reads, w0_cnt writes, r0_fails read failures and w0_fails write failures and that backend 1 (drive 1) has made r1_cnt reads, w1_cnt writes, r1_fails read failures and w1_fails write failures and so forth for each backend in the raid set.

If the string "zerostats" is written to the file bind_point/stats then all cumulative read and write counts for each backend of the raid set are zeroed.

- EXAMPLE:

```
        > /raid0
bind -k {raid0 6} /raid0

echo moniker name=R_0 > /raid0/ctl

echo engage drive=0 qlen=8 fd=7 blksize=8192 name=D0

<>[7] /d0/data > /raid0/ctl

echo access drive=0 read-write > /raid0/ctl

...

echo engage drive=5 qlen=8 fd=7 blksize=8192 name=D5

<>[7] /d5/data > /raid0/ctl

echo access drive=5 read-write > /raid0/ctl
```

This example creates the file "/raid0" as a bind point and then binds the raid0 device on it. The first echo command establishes the internal raid device name as R_0. The subsequent echo commands are shown in pairs for each backend file: one sending an "engage" string and the other sending an "access" string to the file "/raid0/ctl". Each "engage" string associates a backend file (via file descriptor 7) with a block size of 8192 bytes and a maximum queue length of 8. The following "access" string adjusts the access level of the backend file from "offline" (the default) to "read−write". This is a six disk raid set.

- NOTES: The size of the resultant raid set will be the size of the smallest backend multiplied by the number of data backends adjusted downwards to align to be a multiple of the raid set's blocksize (blockSize).
- SEE ALSO: raid1, raid3, raid4, raid5

13.65 RAID1 − raid 1 device

- SYNOPSIS:

```
        bind -k raid1 bind_point
echo moniker name=raid_set_name > bind_point/ctl

echo engage drive=driveNum qlen=queueLen fd=aFdNum blksize=blockSize name=backendname

<>[aFdNum] backEnd > bind_point/ctl
```

```
echo disengage drive=driveNum > bind_point/ctl

echo access drive=driveNum read-write > bind_point/ctl

echo access drive=driveNum read-only > bind_point/ctl

echo access drive=driveNum write-only > bind_point/ctl

echo access drive=driveNum offline > bind_point/ctl

cat bind_point

ctl

data

repair

stats
```

- DESCRIPTION: raid1 implements a raid 1 device. Raid 1 is also known as "mirroring". It has 1 "frontend" (i.e. bind_point/data) and 2 "backends" (i.e. one defined by each "engage" message with a new drive number). To associate an internal name (or moniker) with the raid device, send the message "moniker name=internal_name" to the device's control file, bind_point/ctl. Read and write operations to the frontend (i.e. bind_point/data) must be in integral units of blockSize. Each write of blockSize bytes is written on both backend files. A read of blockSize bytes needs only to read 1 backend file (unless there is a problem). The backend file chosen to do the read is the one calculated to have its heads closer to the required block. The backend "files" referred to here will typically be disks.

  The "logical" block size is currently 512 bytes and the given blockSize must be a power of 2 times 512 (i.e. 2**n * 512 bytes). If, for example, the blockSize was 8 Kb then a write of 8 Kb would cause both backend files to have that 8 Kb written to them. An 8 Kb read would cause the file calculated to have its "heads" closer to be read. If this file was marked "offline", "write−only" or reported an IO error then the other file would be read.

  The queueLen argument must be 1 or greater and sets the maximum number of requests that can be put in a queue associated with each backend file. A daemon is spawned for each backend file to service this queue called async_io. The name argument allows associates the given backendname string with the appropriate backend. This string will be used in reporting errors on the running raid.

  Each backend file first needs to be identified to the raid1 device via the "engage" string sent to bind_point/ctl. If required a file can have its association with this device terminated with a "disengage" string. Once a backend file is engaged its access level can be varied between "read−write", "read−only", "write−only" and "offline" as required. The default is "offline" so in most initialization situations an "access read−write" string needs to be sent to this device.

  When the file bind_point/ctl is read then a line is output for every engaged backend file indicating its access status (e.g. "drive 3: engaged, read−write"). Also backend files that have been disengaged and not "re−"engaged output a line (e.g. "drive 5: disengaged").

  When the file bind_point/stats is read then a line is output which shows the cumulative number of reads and writes performed (including failures) for each backend of the raid device. The format of this line is:

```
D0 r0_cnt r0_fails w0_cnt w0_fails; D1 r1_cnt r1_fails w1_cnt w1_fails;
```

which indicates that backend 0 (typically the drive0) has made r0_cnt reads, w0_cnt writes, r0_fails read failures and w0_fails write failures and that backend 1 (drive 1) has made r1_cnt reads, w1_cnt writes, r1_fails read failures and w1_fails write failures.

If the string "zerostats" is written to the file bind_point/stats then all cumulative read and write counts for each backend of the raid set are zeroed.

- EXAMPLE

```
      > /raid1
bind -k raid1 /raid1

echo moniker name=R_1 > /raid1/ctl

echo engage drive=0 qlen=8 fd=7 blksize=8192 name=D1

<>[7] /d0/data > /raid1/ctl

echo access drive=0 read-write > /raid1/ctl

echo engage drive=1 qlen=8 fd=7 blksize=8192 name=D5

<>[7] /d5/data > /raid1/ctl

echo access drive=1 read-write > /raid1/ctl
```

This example creates the file "/raid1" as a bind point and then binds the raid1 device on it. The first echo command

establishes the internal raid device name as R_1. The subsequent echo commands are shown in pairs for both backend files: one sending an "engage" string and the other sending an "access" string to the file "/raid1/ctl". Each "engage" string associates a backend file (via file descriptor 7) with a block size of 8192 bytes and a maximum queue length of 8. The following "access" string adjusts the access level of the backend file from "offline" (the default) to "read−write".

- NOTES: The size of the resultant raid set will be the size of the smallest backend multiplied by the number of data backends (here just 1 as we are a mirror) adjusted downwards to align to be a multiple of the raid set's blocksize (blockSize).
- SEE ALSO: raid0, raid3, raid4, raid5

13.66 RAID3 − raid 3 device

- SYNOPSIS

```
      bind -k {raid3 nbackends} bind_point
echo moniker name=raid_set_name > bind_point/ctl

echo engage drive=driveNum qlen=queueLen fd=aFdNum blksize=blockSize name=backendname

<>[aFdNum] backEnd > bind_point/ctl

echo disengage drive=driveNum > bind_point/ctl
```

```
echo access drive=driveNum read-write > bind_point/ctl

echo access drive=driveNum read-only > bind_point/ctl

echo access drive=driveNum write-only > bind_point/ctl

echo access drive=driveNum offline > bind_point/ctl

cat bind_point

ctl

data

repair

stats
```

* DESCRIPTION: raid3 implements a raid 3 device. It has 1 "frontend" (i.e. bind_point/data) and typically multiple "backends" (i.e. one defined by each "engage" message with a new drive number).

  To associate an internal name (or moniker) with the raid device, send the message "moniker name=internal_name" to the device's control file, bind_point/ctl.

  This implementation of raid 3 uses at least 3 files in its backend. Read and write operations to the frontend (i.e. bind_point/data) must be in integral units of blockSize. Each write of blockSize bytes is striped (i.e. divided evenly) across (nbackends − 1) files with the "parity" on the other file. Subsequent writes will NOT rotate the file being used to store parity. [This rotation is a slight extension of the original raid 3 definition.] The backend "files" referred to here will typically be disks.

  The "logical" block size is currently 512 bytes and the given blockSize must be an integral multiple of (nbackends − 1) * 512 If, for example, the blockSize was 8 Kb and there were 5 backends then a write of 8 Kb would cause 4 backend files to have 2 Kb written on them and the other backend file to have 2 Kb of parity written on it. An 8 Kb read would cause the 4 files known to hold the data (as distinct from the parity) to be read. If any one of these files was marked "offline", "write−only" or reported an IO error then the 5th file containing the parity would be read and the 8 Kb block reconstructed.

  The queueLen argument must be 1 or greater and sets the maximum number of requests that can be put in a queue associated with each backend file. A daemon is spawned for each backend file to service this queue called async_io.

  The name argument allows associates the given backendname string with the appropriate backend. This string will be used in reporting errors on the running raid.

  Each backend file first needs to be identified to the raid3 device via the "engage" string sent to bind_point/ctl. If required a file can have its association with this device terminated with a "disengage" string. Once a backend file is engaged its access level can be varied between "read−write", "read−only", "write−only" and "offline" as required. The default is "offline" so in most initialization situations an "access read−write" string needs to be sent to this device. When the file bind_point/ctl is read then a line is output for every engaged backend file indicating its access status (e.g. "drive 3: engaged, read−write"). Also backend files that have been disengaged and not "re−"engaged output a line (e.g. "drive 5: disengaged").

  When the file bind_point/stats is read then a line is output which shows the cumulative number of reads and writes performed (including failures) for each backend of the raid device. The format of this line is

```
D0 r0_cnt r0_fails w0_cnt w0_fails; D1 r1_cnt r1_fails w1_cnt w1_fails;
 ...
```

which indicates that backend 0 (typically the drive0) has made r0_cnt reads, w0_cnt writes, r0_fails read failures and w0_fails write failures and that backend 1 (drive 1) has made r1_cnt reads, w1_cnt writes, r1_fails read failures and w1_fails write failures and so forth for each backend in the raid set.

If the string "zerostats" is written to the file bind_point/stats then all cumulative read and write counts for each backend of the raid set are zeroed.

- EXAMPLE:

```
      > /raid3
bind -k {raid3 5} /raid3

echo moniker name=R_3 > /raid3/ctl

echo engage drive=0 qlen=8 fd=7 blksize=8192 name=D0

<>[7] /d0/data > /raid3/ctl

echo access drive=0 read-write > /raid3/ctl

...

echo engage drive=5 qlen=8 fd=7 blksize=8192 name=D5

<>[7] /d5/data > /raid3/ctl

echo access drive=5 read-write > /raid3/ctl
```

This example creates the file "/raid3" as a bind point and then binds the raid3 device on it. The first echo command establishes the internal raid device name as R_3. The subsequent echo commands are shown in pairs for each backend file: one sending an "engage" string and the other sending an "access" string to the file "/raid3/ctl". Each "engage" string associates a backend file (via file descriptor 7) with a block size of 8192 bytes and a maximum queue length of 8. The following "access" string adjusts the access level of the backend file from "offline" (the default) to "read−write". This is a 6 disk raid set.

- NOTES: The size of the resultant raid set will be the size of the smallest backend multiplied by the number of data backends adjusted downwards to align to be a multiple of the raid set's blocksize (blockSize).
- SEE ALSO: raid0, raid1, raid4, raid5

13.67 RAID4 − raid 4 device

- SYNOPSIS:

```
      bind -k {raid4 nbackends} bind_point
echo moniker name=raid_set_name > bind_point/ctl

echo engage drive=driveNum qlen=queueLen fd=aFdNum blksize=blockSize name=backendname

<>[aFdNum] backEnd > bind_point/ctl
```

```
echo disengage drive=driveNum > bind_point/ctl

echo access drive=driveNum read-write > bind_point/ctl

echo access drive=driveNum read-only > bind_point/ctl

echo access drive=driveNum write-only > bind_point/ctl

echo access drive=driveNum offline > bind_point/ctl

cat bind_point

ctl

data

repair

stats
```

* DESCRIPTION: raid4 implements a raid 4 device. It has 1 "frontend" (i.e. bind_point/data) and typically multiple "backends" (i.e. one defined by each "engage" message with a new drive number). To associate an internal name (or moniker) with the raid device, send the message "moniker name=internal_name" to the device's control file, bind_point/ctl. This implementation of raid 4 uses at least 3 files in its backend. Read and write operations to the frontend (i.e. bind_point/data) must be in integral units of blockSize. Each write of blockSize bytes is written on 1 backend file. Its neighboring (nbackends − 2) files need to be read at the same offset to calculate a new parity block which needs to be re− written. The nbackends blocks at the same offset on the nbackends backend files are called a slice. The parity block is, like in raid3, fixed as the last backend. A read of blockSize bytes needs only to read 1 backend file (unless there is a problem). The backend "files" referred to here will typically be disks.

  The "logical" block size is currently 512 bytes and the given blockSize must be an integral multiple of (nbackends − 1) * 512 If, for example, the blockSize was 8 Kb then a write of 8 Kb would cause 1 backend file to have that 8 Kb written to it with the other (nbackends − 2) non−parity files in that slice having 8 Kb read from them in order to generate a new 8 Kb parity block which is then written to the parity file in this slice. An 8 Kb read would cause the file known to hold the data (as distinct from the parity) to be read. If this file was marked "offline", "write−only" or reported an IO error then the other ((nbackends − 1) files in the slice (i.e. (nbackends − 2) data and 1 parity) would be read and the 8 Kb block reconstructed.

  The queueLen argument must be 1 or greater and sets the maximum number of requests that can be put in a queue associated with each backend file. A daemon is spawned for each backend file to service this queue called async_io.

  The name argument allows associates the given backendname string with the appropriate backend. This string will be used in reporting errors on the running raid.

  Each backend file first needs to be identified to the raid5 device via the "engage" string sent to bind_point/ctl. If required a file can have its association with this device terminated with a "disengage" string. Once a backend file is engaged its access level can be varied between "read−write", "read−only", "write−only" and "offline" as required. The default is "offline" so in most initialization situations an "access read−write" string needs to be sent to this device.

  When the file bind_point/ctl is read then a line is output for every engaged backend file indicating its access status (e.g. "drive 3: engaged, read−write"). Also backend files that have been disengaged and not "re−"engaged output a line (e.g. "drive 5: disengaged").

When the file bind_point/stats is read then a line is output which shows the cumulative number of reads and writes performed (including failures) for each backend of the raid device. The format of this line is

```
D0 r0_cnt r0_fails w0_cnt w0_fails; D1 r1_cnt r1_fails w1_cnt w1_fails;
 ...
```

which indicates that backend 0 (typically the drive0) has made r0_cnt reads, w0_cnt writes, r0_fails read failures and w0_fails write failures and that backend 1 (drive 1) has made r1_cnt reads, w1_cnt writes, r1_fails read failures and w1_fails write failures and so forth for each backend in the raid set.

If the string "zerostats" is written to the file bind_point/stats then all cumulative read and write counts for each backend of the raid set are zeroed.

- EXAMPLE

```
      > /raid4
bind -k {raid4 5} /raid4

echo moniker name=R_4 > /raid4/ctl

echo engage drive=0 qlen=8 fd=7 blksize=8192 name=D0

<>[7] /d0/data > /raid4/ctl

echo access drive=0 read-write > /raid4/ctl

...

echo engage drive=5 qlen=8 fd=7 blksize=8192 name=D5

<>[7] /d5/data > /raid4/ctl

echo access drive=5 read-write > /raid4/ctl
```

This example creates the file "/raid4" as a bind point and then binds the raid4 device on it. The first echo command establishes the internal raid device name as R_4. The subsequent echo commands are shown in pairs for each backend file: one sending an "engage" string and the other sending an "access" string to the file "/raid4/ctl". Each "engage" string associates a backend file (via file descriptor 7) with a block size of 8192 bytes and a maximum queue length of 8. The following "access" string adjusts the access level of the backend file from "offline" (the default) to "read−write". This is a six disk raid set.

- NOTES: The size of the resultant raid set will be the size of the smallest backend multiplied by the number of data backends adjusted downwards to align to be a multiple of the raid set's blocksize (blockSize).
- SEE ALSO: raid0, raid1, raid3, raid5

13.68 RAID5 − raid 5 device

- SYNOPSIS

```
      bind -k {raid5 nbackends} bind_point
echo moniker name=raid_set_name > bind_point/ctl
```

```
echo engage drive=driveNum qlen=queueLen fd=aFdNum blksize=blockSize name=backendname

<>[aFdNum] backEnd > bind_point/ctl

echo disengage drive=driveNum > bind_point/ctl

echo access drive=driveNum read−write > bind_point/ctl

echo access drive=driveNum read−only > bind_point/ctl

echo access drive=driveNum write−only > bind_point/ctl

echo access drive=driveNum offline > bind_point/ctl

cat bind_point

ctl

data

repair

stats
```

* DESCRIPTION: raid5 implements a raid 5 device. It has 1 "frontend" (i.e. bind_point/data) and typically multiple "backends" (i.e. one defined by each "engage" message with a new drive number).

  To associate an internal name (or moniker) with the raid device, send the message "moniker name=internal_name" to the device's control file, bind_point/ctl.

  This implementation of raid 5 uses at least 3 files in its backend. Read and write operations to the frontend (i.e. bind_point/data) must be in integral units of blockSize. Each write of blockSize bytes is written on 1 backend file. Its neighboring (nbackends − 2) files need to be read at the same offset to calculate a new parity block which needs to be re−written. The nbackends blocks at the same offset on the nbackends backend files are called a slice. The parity block is rotated from one slice to the next. A read of blockSize bytes needs only to read 1 backend file (unless there is a problem). The backend "files" referred to here will typically be disks.

  The "logical" block size is currently 512 bytes and the given blockSize must be an integral multiple of (nbackends − 1) * 512 If, for example, the blockSize was 8 Kb then a write of 8 Kb would cause 1 backend file to have that 8 Kb written to it with the other (nbackends − 2) non−parity files in that slice having 8 Kb read from them in order to generate a new 8 Kb parity block which is then written to the parity file in this slice. An 8 Kb read would cause the file known to hold the data (as distinct from the parity) to be read. If this file was marked "offline", "write−only" or reported an IO error then the other ((nbackends − 1) files in the slice (i.e. (nbackends − 2) data and 1 parity) would be read and the 8 Kb block reconstructed.

  The queueLen argument must be 1 or greater and sets the maximum number of requests that can be put in a queue associated with each backend file. A daemon is spawned for each backend file to service this queue called async_io.

  The name argument allows associates the given backendname string with the appropriate backend. This string will be used in reporting errors on the running raid.

  Each backend file first needs to be identified to the raid5 device via the "engage" string sent to bind_point/ctl.

If required a file can have its association with this device terminated with a "disengage" string. Once a backend file is engaged its access level can be varied between "read−write", "read− only", "write−only" and "offline" as required. The default is "offline" so in most initialization situations an "access read−write" string needs to be sent to this device.

When the file bind_point/ctl is read then a line is output for every engaged backend file indicating its access status (e.g. "drive 3: engaged, read−write"). Also backend files that have been disengaged and not "re−"engaged output a line (e.g. "drive 5: disengaged").

When the file bind_point/stats is read then a line is output which shows the cumulative number of reads and writes performed (including failures) for each backend of the raid device. The format of this line is

```
D0 r0_cnt r0_fails w0_cnt w0_fails; D1 r1_cnt r1_fails w1_cnt w1_fails;
 ...
```

which indicates that backend 0 (typically the drive0) has made r0_cnt reads, w0_cnt writes, r0_fails read failures and w0_fails write failures and that backend 1 (drive 1) has made r1_cnt reads, w1_cnt writes, r1_fails read failures and w1_fails write failures and so forth for each backend in the raid set.

If the string "zerostats" is written to the file bind_point/stats then all cumulative read and write counts for each backend of the raid set are zeroed.

- EXAMPLE

```
       > /raid5
bind -k {raid5 5} /raid5

echo moniker name=R_5 > /raid5/ctl

echo engage drive=0 qlen=8 fd=7 blksize=8192 name=D0

<>[7] /d0/data > /raid5/ctl

echo access drive=0 read-write > /raid5/ctl

...

echo engage drive=5 qlen=8 fd=7 blksize=8192 name=D1

<>[7] /d5/data > /raid5/ctl

echo access drive=5 read-write > /raid5/ctl
```

This example creates the file "/raid5" as a bind point and then binds the raid5 device on it. The first echo command establishes the internal raid device name as R_5. The subsequent echo commands are shown in pairs for each backend file: one sending an "engage" string and the other sending an "access" string to the file "/raid5/ctl". Each "engage" string associates a backend file (via file descriptor 7) with a block size of 8192 bytes and a maximum queue length of 8. The following "access" string adjusts the access level of the backend file from "offline" (the default) to "read−write". This is a six disk raid set.

- NOTES: The size of the resultant raid set will be the size of the smallest backend multiplied by the number of data backends adjusted downwards to align to be a multiple of the raid set's blocksize (blockSize).
- SEE ALSO: raid0, raid1, raid3, raid4

13.69 RAM − ram based file system

- SYNOPSIS: bind −k ram bind_point
- DESCRIPTION: The ram file system uses memory on the target system's heap to support an hierarchical file system. Typically ram is the root file system in a husky environment. Unlike a normal file system ram lacks persistence. Therefore (assuming the heap resides in _non_ battery backed up RAM) when the target system loses power, the contents of the ram file system are lost. This is similar to the way the "/tmp" file system works in Unix.
- SEE ALSO: mem

13.70 RANDIO − simulate random reads and writes

- SYNOPSIS: randio −d device −n nop [−b cnt] [−f fill_ch] [−i init_ch] [−o offset] [−s size] [−G grp_size] [−S seed] [−X xfer_size] [−M rdwr|rdonly] [−T seq|ran] [−vBDON]
- DESCRIPTION: randio defines a range on the device from the start of the device to it's end, or for a length of size IN 512−byte BLOCKS, if specified, in which to perform. nop random write then read operations. All reads and writes are in multiples of 512 byte blocks. First, every block in the given range is initialized with a specific pattern. Then for each operation, a random location in the range is chosen and a random number of blocks are written with a specific data pattern. Once nop write operations have occurred, nop reads are performed at the appropriate locations and lengths to verify the previously written data. Lastly, all data on the device is verified. That is, any unwritten block should have only the initialization pattern and any block that was written to should have the appropriate data pattern. When a write of a given length occurs, each 512 byte block within the write has a header containing the operation number, the start of the write location, the length of write and the data fill pattern. The rest of each 512 byte block is initialized with a fill pattern. The output of this command provides the I/O transfer rates in Megabytes (MB) and million bytes (Mb) for the three phases of I/O, that is sequential- writes, random writes then reads, and finally sequential reads. The size and offset values may have a suffix.
- OPTIONS:
    - −b cnt: Restrict all read and write operations to be multiples of cnt x 512 blocks. The default is 1.
    - −f fill_ch: The data fill pattern used in writes should be fill_ch. fill_ch must be specified as a hex number. The default is 0x7F.
    - −i init_ch: The fill pattern used when initialising the device should be init_ch. fill_ch must be specified as a hex number. The default is 0x00.
    - −o offset: All reads and writes are to be performed offset 512−byte BLOCKS into the device. All reported values will be relative to this offset. The default is 0 i.e the start of the device.
    - −s size: The size, in 512−byte BLOCKS, of the range upon which we perform the io.
    - −v: Run in verbose mode printing out initialization information, and the final verification of the whole specified range on the device. If a second −v is given then details of individual read and write operations are printed.
    - −B: Align all random write locations to be a multiple of cnt x 512. Where cnt is specified by the −b option. This is useful when writing to raw raid devices which require both a fixed write size multiple and an aligned "start of write" address.
    - −G grp_siz: Normally, randio performs nop writes then nop reads. The −G option can change this to perform grp_siz writes then grp_siz reads looping until nop operations have occurred. The default "group size" if nop operations. If grp_siz is negative, then a random group size is chosen.
    - −M rdwr|rdonly: By default, a destructive read−write test is performed (−M rdwr). To run randio in a non−destructive way, specify this option with the sub−option of rdonly. This will

result in randio only performing reads.
  - −N: By default, data read is always compared against what was written. By specifying this option, the comparison will not be made. This option is usefull when you are only concerned with performing random reads and writes and not if the data corrupts.
  - −O: By default, the random write locations and sizes are chosen so that they overlap. By specifying this option, overlapped writes can NOT occur. By careful when using this option if your device is small.
  - −S seed: Specify a seed for the random number generators.
  - −T ran|seq: By default, randio performs a sequence of sequential writes, random writes then reads and finally sequential reads. To have only the sequential activity performed, use the −T seq option. The default is −T ran.
  - −X xfer_size: Restrict the maximum write size to be xfer_size 512−byte BLOCKS. The default is 256 blocks − 128K bytes.
- SEE ALSO: speedtst, suffix

13.71 RCONF, SPOOL, HCONF, MCONF, CORRUPT−CONFIG − raid configuration and spares management

- SYNOPSIS :
  - rconf −add −type raidtype −name raidname −size raidsize −iomode read−write|read−only|write−only −iosize raid− chunksize −hostif mflag controller host_port scsi_lun −backendsize backends_size −backends backend1,backend2,...,backendn [−boot auto|manual] [−cache raidcachesize] [−state active|inactive] [−usespares on|off] [−usezoneio on|off] [−qlen nqueues] [−stargs {xstargd_args}] [−hostif mflag controller host_port scsi_lun]
  - rconf −delete raidname
  - rconf −list [raidname] [−v]
  - rconf [−v]
  - rconf −mkfaulty −name raidname −drive drive_no
  - rconf −modify −name raidname [−iomode read−write|read−only|write−only] [−newname newraidname] [−hostif mflag controller host_port scsi_lun] [−boot auto|manual] [−cache raidcachesize] [−state active|inactive] [−usespares on|off] [−usezoneio on|off] [−qlen nqueues] [−stargs {xstargd_args}]
  - rconf −rebuild −name raidname −drive drive_no [−spare spare_backend]
  - rconf −repair −name raidname −drive drive_no −action start|finish [−depth value]
  - rconf −replace −name raidname −backend backendn −action start|finish [−depth value]
  - rconf −unrepair −name raidname −backend backendn −drive drive_no −depth value
  - rconf −init
  - rconf −fullinit
  - rconf −check
  - rconf −maxiochunk
  - rconf −validate
  - rconf −syslog
  - rconf −size
  - spool −add −name backend −backendsize backends_size −type hot|warm [−ctlr controller_number]
  - spool −delete −name backend
  - spool −list
  - spool

- hconf −delete controller hostport
- hconf −list [controller hostport]
- hconf
- hconf −modify controller hostport scsi_id
- mconf −add controller hostport scsi_lun [blkprotocol]
- mconf −delete controller hostport scsi_lun
- mconf −sml
- mconf −smldelete controller hostport
- mconf −list [controller]
- mconf [−C] [−c] corrupt−config

- DESCRIPTION: rconf, hconf, mconf and spool manipulate raid set definitions, host port connections and the spares pool on a RaidRunner. All information is stored the RaidRunner configuration area which is stored in the file /dev/flash/conf and duplicated on every disk (/dev/hd/.../rconfig) on the RaidRunner. The corrupt−config command will corrupt the configuration areas and immediately reboot the RaidRunner.
  - Raid Sets: A Raid Set has the following attributes
    - Type : the Raid Type − either 0 (stripped backends), 1 (mirrored backends), 3 (stripped backends with parity on one backend), 5 (stripped backends with parity spread across all backends).
    - Name : the Raid Set's unique name used to make identification and access easier. A Raid Set's name must start with an alphabetic character and then have alphanumeric elsewhere. The maximum length of the name is 32 characters.
    - IOmode: the mode of access to the Raid Set. Access modes can be either read−write, read−only or write−only.
    - Size: the size (in 512 byte blocks) of the raid set if it's to be less than the calculated size. Normally a raid set's size is a function of the size of the raid set's backends (i.e type 0 − sum of backends, 1 − size of one backend, 3/5 − sum of backends less one). The Size must be a multiple of of the IOsize.
    - IOsize: the size (in bytes) of io for read and write operations to the raid set. This is commonly called the chunksize.
    - Cachesize: the size (in bytes) of any cache that will front−end the raid set. The cache size must be a minimum of 256K bytes (quarter megabyte). The cache size must also be a multiple of the raid set's IOsize.
    - Host Interface(s): the interfaces to use for access by a host. A raid set can be either single−ported or multi−ported. That is to say, a raid set can present different scsi disks all directed at itself. Naturally, the host(s) that access these disks must co−operate to ensure data integrity. Each host interface is a quartet of − a master/slave access flag, the controller number, the host port on that controller and lastly the SCSI LUN that the raid set should propagate on the host port. The master/slave access flag is used at boot time to configure the Host Interface's SCSI target daemon into either a full access (and hence spun up) master SCSI target or a minimal access (and hence spun down) slave SCSI target. See details on the −Z option on the mstargd command. This master/slave concept can be used by co−opera− tive hosts to share access to a single raid set via multiple SCSI target daemons. A raid set can only be multi−ported within a controller.
    - Backends: the list of the raid set's backends. Backends can either be disks or other raid sets.
    - Backendsize: the size (in 512 byte blocks) of the raid set backends
    - Bootmode: the boot mode of the raid set. It can either be autoboot or manual. For the former, when the RaidRunner boots, the raid set is automatically made available on the host interface specified and in the later case, it must be manually made available.

    - ◊ State : this is the State of the raid set. The state of a raid set can either be active or inactive. If active, then the raid set's data is available for access. If inactive, then data on the raid set is not avail− able and hence the raid set is in a quiescent state.
    - ◊ ZoneioUsage: this is a flag to indicate to the cache filesystem that all I/O to this raid set be optimized for backend reads and writes based on the backends zone (notch) pages.
    - ◊ SparesUsage: this is a flag to indicate whether the raid set is to use spares. It's value can be either on or off.
    - ◊ QueueLen: this is the maximum number of requests than can be put in an io queue associated with each backend of the raid set. It's value must be 1 or greater.
  - ♦ Spares Pool: A spares pool can be established on a RaidRunner which provides a pool of disks for use by running raid sets which suffer a backend (disk) failure. Each disk in the spool may be either hot or warm. Hot disks are those that are immediately available i.e spun up. Warm disks are those that require spinning−up prior to use. As different raid sets may use different backend (disk) sizes, the spares pool must know the size of each disk in the pool. If the RaidRunner has more than one controller, then the usage of a spare can be restricted to a specific controller. Given a spare is available to a given controller, then the spare is allocated (rconf −repair command based on the following priority − hot spares of the same size on the same rank, hot spares of the same size on another rank, warm spares of the same size on the same rank, warm spares of the same size on another rank, hot spares of a larger size on the same rank, etc.
  - ♦ The rank is the SCSI ID of the device.
  - ♦ Host Port SCSI ID Assignment: On each controller in a RaidRunner there are a number of Host Ports through which data on the RaidRunner is accessed. As this host port is effectively a SCSI device then it has to have a SCSI ID assigned to it. There are two methods to assign a SCSI ID to a host port. The first uses a physical selector switch (usually providing selections from 0 to 16) and the second is to internally configure the host port's SCSI ID in the non−volatile raid configuration area − the hconf configuration command performs this assignment.
  - ♦ SCSI Monitor: On each controller in a RaidRunner one or more Monitors (smon) can be set up to run on a specified host port. This monitor allows a program on a host system to send RaidRunner commands which will be executed in a husky subshell as well as transfer of files to and from the RaidRunner. This monitor simulates a 32K "disk" on a given SCSI ID (as set up by the hconf command or selector switch) and SCSI LUN (which is set up using the mconf command). Execution of commands and file transfers are effected by reading from and writing to specific block locations on the "disk". The reads and writes are to follow a specific protocol based on the block locations and order of the reads and writes. See the smon for details of this protocol. As different systems use specific blocknumbers for their own internal use (eg block number 0 is typically used for disk labelling) writes to any block not used by the protocol are preserved for subsequent reads. Up to 16 blocks are preserved in the raid configuration area and are re−read by smon at boot time so data is preserved. If more than 16 blocks are written to, then this data is silently discarded. The block numbers used in the smon protocol can be changed to suit different systems which may use the default protocol block numbers. By default, the SCSI monitor will always start at boot time, even if the raid configuration area has just been initialized or is corrupt. RaidRunner model specific defaults for the host port, SCSI ID and SCSI LUN for the monitor are compiled into the RaidRunner binary. To prevent the scsi monitor from automatically executing at boot time the monitor will have to be specifically deleted from the controller using the mconf −delete option.
- Manual Backend Reconstruction: Raid types 1, 3 and 5 are resistant to a single backend device failure. That is, the raid set's data is still available even if one of it's backends fail. Should a

subsequent failure occur on a different backend then the raid set's data will be inaccessible. When a backend device does fail, the RaidRunner disengages that device from the raid set. To replace a failed backend of a type 1, 3 or 5 raid set, first physically correct the failure (eg replace and test the faulty disk), then engage the backend in write−only mode and read the entire raid set from the repair entry point. This will result in the reconstruction raid set's data and parity (if appropriate) onto the replaced backend. We engage the backend in write−only access mode to prevent the raid set from reading from this drive during the reconstruction process as data on this backend is incomplete. Once we have reconstructed the data on the replaced backend (i.e the read has completed), we then set the backend's access mode to be the access mode of the raid set and thus we now have a "complete" raid set. This reconstruction of data is passive, in that the raid set is still "running" during the reconstruction, although it will be running slowly.

- Automatic Backend Reconstruction: In the case of either a type 1, 3 or 5 raid set, a single backend device failure can result in the automatic replacement of the failed backend and subsequent reconstruction of the raid set. This is accomplished by specifying a husky script to execute when a backend failure occurs − see autorepair. When a backend failure occurs, the backend is disengaged from the raid set, and the husky script, if specified, is executed. The script typically allocates a disk from the spares pool, engages it to the raid set in write−only access mode, then reads the entire raid set (from the repair entry point) which reconstructs the raid set incorporating the new disk. Once complete, the disk's access mode is set to the access mode of the raid set.
- If, during the reconstruction phase (read of whole raid set) the newly engaged backend fails, the raid set's data is still available. All the RaidRunner will do, is disengage the backend and execute the script again. It will continue to do this until no more spares are available. Alternately, if another backend fails during the reconstruction, the raid set will fail.
- Configuration Areas: Typically the RaidRunner stores multiple copies of the raid configuration area. Copies are stored on all disks (in /dev/hd/c.s.l/rconfig) that can be written to and in a section of flash ram (/dev/flash/conf) on the RaidRunner controller board itself. This is done for redundancy. Whenever the raid configuration area is updated, copies are written to each disk (in chip, scsi id (rank), lun order) then lastly to flashram. When the raid configuration is read, it groups, all configuration sources with identical configurations then works out the most "correct" configuration based on the following rules −
    1. If all sources are unreadable or corrupt then we used the compiled default scsi host id's and scsi monitor configurations.
    2. If all sources are the same (i.e one group) then this is the normal configuration.
    3. If we have two groups of configurations, then the group with the highest number of identical areas is used. If both groups have the same number of identical areas then we pick the group with the highest revision. If both groups have the same number of areas and the same revision, the first group is chosen.
    4. If we have three groups we choose the group with the highest number of identical areas. If two groups (or all three) meet this criteria we use the defined master configuration area − FLASH RAM.
    5. If we have more than three groups, we use the master configuration area − FLASH RAM. The rconf −check option is used to re−read all configuration areas has per the above rules, and if there is differences (more than 1 group) it selects the most "correct" configuration and re−writes it out to all areas. When the RaidRunner initially boots it typically only has access to the flashram (as the disks have not been bound into the filesystem yet), so it uses the configuration stored in flashram to start (in a spun−down state) the scsi monitor(s) and scsi target daemons (stargd's). It then binds in all disks and executes a rconf −check command and if it had to re−write the configuration AND flashram was not in the "correct" group it may reboot the RaidRunner (as now we can assume than the flashram area is now correct).
- OPTIONS − all commands
    - −C: This option checks to see if any significant configuration change has been made since boot time

to any raid set, host port, scsi monitor or spares pool configuration. An appropriate message is printed and a return status ($status) of 1 is returned if a change has occurred, else 0 for no change. Any modification, addition, deletion is considered to be a significant change. The only NON−significant change is the changing of state of a raid set.
    - −c: This option checks to see if any significant configuration change has been made since boot time to the relevant configuration area depending on the command invoked − rconf − raid sets, hconf − host ports (if not using physical selectors), etc. The return status is as per the −C option.
- OPTIONS − mconf
    - Specifying no options is the same as specifying the −list option.
    - −delete: Delete the scsi monitor given controller number, hostport and SCSI LUN. This option takes three values, the controller number, host port and the SCSI LUN. The deletion of a monitor will not take effect until next boot of the RaidRunner.
    - −list: Print the controller/hostport assigned monitor SCSI LUN. If the optional controller number argument is given then print out the controller, hostport and assigned monitor SCSI LUNs for that controller. If a controller has no assigned monitor SCSI LUN, then a minus (−) character will be printed in place of the SCSI LUN. If an error occurs whilst opening or reading the RaidRunner configuration area, a RaidRunner model specific default with be listed.
    - −add: Set the monitor SCSI LUN on the given controller's host port. This option takes three mandatory values and one optional value. The first is the controller, the second the host port on that controller and the last is the SCSI LUN to assign the monitor. If the host port has not been assigned a SCSI ID, then the monitor will not be executed at boot. The optional value is a comma separated list of nine (9) numbers which change the default block protocol blocknumbers used by the SCSI monitor. See smon for details.
    - −sml: Print each scsi monitor's stored label information. The information is printed as follows − controller number, hostport number, number of stored blocks, followed by a colon, then the block addresses stored. Block number 0's at the end of this list mean that those available blocks have not been written to.
- OPTIONS − hconf
    - Specifying no options is the same as specifying the −list option.
    - −delete: If the controller does not have a physical SCSI ID selector switch, specifying this option will delete the controller/hostport assigned SCSI ID. This option takes two values, the controller number and host− port. By deleting an assigned scsi id, no other program (eg scsi monitor or raid set) can appear on this host port.
    - −list: Print the controller/hostport assigned SCSI ID. If the optional controller and hostport arguments are given then print out the controller, hostport and assigned scsi id for that pair. If no arguments are given, then for each host port on each controller in the RaidRunner, print out the controller number, host port and SCSI ID assigned to that host port. If the controller does not have host port SCSI ID selector switches, then if a host port has not had a SCSI ID assigned to it, a minus (−) character will be printed in place of the SCSI ID. For example (of a RaidRunner with two controllers each of which have two host ports)

```
        0 0 2
0 1 3

1 0 −

1 0 6
```

Which shows that on the first controller (0), the first host port (0) has been assigned SCSI ID 2, the second port (1) has been assigned SCSI ID 3. The second controller (1) has NO SCSI ID assigned to it's first host port, and SCSI ID 6 assigned to it's port. If an error occurs whilst opening or reading the RaidRunner configuration area, a RaidRunner model specific default with be listed.

- −modify: If the controller does not have a physical SCSI ID selector switch, specifying this option will set the SCSI ID on the given controller's host port. This option takes three values. The first is the controller, the second the host port on that controller and the last is the SCSI ID to assign. You cannot set a host port's SCSI ID if any raid set which uses that port is active.
- OPTIONS − rconf
    - Any size sub−options (i.e raidsize, cache, iosize, backendsize) can use the suffix values. Specifying no options is the same as specifying the −list option.
    - −add: Add a raid set on the RaidRunner. Sub−options are −
    - −type: Specify the raid type. This option takes values of either 0, 1, 3 or 5.
    - −name: Specify the raid name. A name must be alphanumeric, with a maximum of 32 characters and start with an alphabetic character.
    - −iomode: Specify the raid set's io mode. This option takes values of either RW, RD or WR for read−write, read−only or write−only respectively.
    - −size Specify the raid set's size. This is the size of the raid set in 512 byte blocks. It must be a multiple of the −iosize value. If it is not then it will be automatically adjusted to be so and a message (not error) will be printed. In the case of raid type 3 the size must be a multiple of the −iosize value multiplied by the number of data backends in the raid set.
    - −iosize: Specify the raid set's chunksize in bytes. All reads and writes on the raid set will be in multiples of this size. When cache is used, this is also the size of the cache buffers created (for Raid type 3 the cache buffer size is the number of data backends times this value). The RaidRunner establishes a maximum number of 512−byte blocks that can be written to at any instant − see the write_limit internal variable (internals). Accordingly iosize is limited to ensure that at least two of these buffers will fit into this "writable" portion of cache.
    - −hostif: Specify the raid set's host interface(s). The host interface is the device though which the raid set's data is externally accessed. A host interface is defined by the quartet − master/slave flag (mflag), controller, hostport and scsi lun. Where
        - mflag: is a target access flag − master(M): full access, slave(S): limited access
        - controller: is the controller on which the raid set is to run
        - hostport: is the host port a raid set's IO is to be directed
        - scsilun: is the scsi lun that the raid set propagates out the hostport

      The master/slave flag is either M for master, or S for slave mode. Multiple host interfaces are added by repeating this option.
- −backendsize: Specify the size of the raid set's backends. This is the size, in 512 byte blocks, of the raid set's backends. This value is used when searching for backends in the spares pool.
- −backends: Specify the raid set's backend devices in a comma separated list. A backend device of a raid set can be a disk or the frontend of another raid set. If it is a disk then it will have the format Dc.s.l where c is the channel, s is the scsi id (or rank) and l is the scsi lun of the disk. If it is a raid set, then it will have the format Rraidsetname where raidsetname is the name of the raid set.
- −boot: Specify the raid set's boot mode. Values can either be auto or manual for autobooting or manual booting raids. This option is optional and the default bootmode is auto.
- −cache: Specify the size, in bytes, of cache that should front−end the raid set. The default size is 0, which means that no cache will be used. The cache size must be a multiple of the raid set's iosize.
- −stargs: Specify additional arguments to the stargd process when it is created for the given raid set. The arguments are to be enclosed in braces − {}, IE {−L 16 −I 512}
- −state: Specify the initial state of the raid set. Values can either be active or inactive. This option is optional and default state is inactive.
- −usezoneio: Set the zone io usage flag to either on or off. If on, then the cache filesystem will fragment write and read requests to suit the zone (notch) partitions on the backend disks.
- −usespares: Set the spares usage flag to either on or off. If on, then a running raid set (1, 3 or 5) will, on an I/O error, attempt automatic spares allocation and reconstruction.
- −qlen: Specify the maximum number of io requests that can be queued to a backend. The minimum is 1 and

the default is 8.

- −delete: Delete a specified raid set from the RaidRunner. The raid set must be inactive.
- raidsetname: Specify the name of the raid set to delete.
- −list: Print raid set configuration. If a raidsetname name is not specified, all raid set configurations are printed, one per line in the form (if verbose mode is not set (−v) − name type flags size iosize cachesize iomode qlen nhostifs hostifs backendsize backends [optional backend status]. Where the flags is a comma separated list from "Active", "Inactive", "Used", "Unused", "Autoboot", "Manboot" and "Usespares". When a raid set is first flagged to be active, then the "Used" flag is per− manently set. This knowledge that a raid set has been used at some time is used by other programs and utilities on the RaidRunner. nhostifs is the number of host interfaces to present hostifs is a comma separated list of host interface quartets of master/slave flag, controller, host port and scsi lun (each separated by a period). For example M0.0.4,S0.1.0 means two host interfaces − both on controller 0, one on port 0 with a scsi LUN of 4 and the other on port 1 with a scsi LUN of 0. The first is to be a master target which provides full access by the host, and the second is to be a slave which will advise on any host access that the target is spun down. The backends are a comma separated list of backends. If a backend has failed then a "− the spare back−end will also be suffixed. If multiple spare backends have been used (i.e the spares failed) then each failed backend will be suffixed with a (−)

  ```
  D0.2.0−,D1s2l0,D2.2.0,D3.2.0,D4.2.0
  ```

  means that the backend D0.2.0 has failed and does not have a spare and the other backends are

  ```
  D1s2l0,D2.2.0,D3.2.0,D4.2.0.

  D0.2.0−D5.2.0,D1s2l0,D2.2.0,D3.2.0,D4.2.0
  ```

  means that the backend D0.2.0 has failed and the spare backend, D5.2.0, has replaced it and the other backends are D1s2l0,D2.2.0,D3.2.0,D4.2.0.

  ```
  D0.2.0−D5.2.0−,D1s2l0,D2.2.0,D3.2.0,D4.2.0
  ```

  means that the backend D0.2.0 has failed and the spare backend, D5.2.0, which replaced it has also failed and the other backends are D1.2.0,D2.2.0,D3.2.0,D4.2.0.

- raidsetname: Print the configuration for only the specified raid set.
- −modify: This command allows one to change the raid set's state, name, bootmode, iomode, host interface, spares usage, extra stargd args, cachesize or QueueLen. When specifying a state change, then no other modifications are allowed. To modify any of the preceding raid set attributes (except state), the raid set must be inactive. Sub−options are
- −name: The name of the raid set to modify
- −newname: The new name for the raid set. This new name must still be unique amongst the other defined raid sets.
- −boot: Change the raid set's boot mode. Values can either be auto or manual for autobooting or manual booting raids.
- −cache: Change the size, in bytes, of cache that should front−end the raid.
- −hostif: Change the raid set's host interface(s). The host interface is the device though which the raid set's data is externally accessed. A host interface is defined by the quartet − master/slave flag (mflag), controller, hostport and scsi lun. Where mflag is a target access flag − master(M): full access, slave(S): limited access controller is the controller on which the raid set is to run hostport is the host port a raid set's IO is to be directed scsilun is the scsi lun that the raid set propagates out the hostport The master/slave flag is either M for master, or S for slave mode. Multiple host interfaces are added by repeating this option. Note you must specify all host interfaces required in the one command. Thus to delete a host interface you modify without that host interface option.

* −iomode: Change the raid set's io mode. This option takes values of either RW, RD or WR for read−write, read−only or write−only respectively.
* −qlen: Change the maximum number of io requests that can be queued to a backend.
* −stargs: Specify additional arguments to the stargd process when it is created for the given raid set. The arguments are to be enclosed in braces − {}, IE {−L 16 −I 512}
* −state: Change state of the raid set. Values can either be active or inactive.
* −usespares: Set the spares usage flag to either on or off.
* −rebuild: The rebuild option is used when a stored raid set configuration has been corrupted and that raid set had faulty backends optionally supplemented with spares. Normally, if the stored raid set configuration was corrupted in some way and it did not have any faulty drives then one would just delete the raid set and then add it. If any of the backends were faulty then we need to store this information in the newly re−created raid set configuration. The rebuild option does this. If the faulty backend did not use any spares, then the −rebuild option will just mark that backend faulty. If it did use spares, then repeated invocations of this command (with the −spare sub−option) will add spares as appropriate, marking the backend faulty and any previously configured spares faulty also. The last invocation with the −spare sub−option will add the spare as in use but not faulty. If the last spare was also faulty, then a final invocation without the −spare sub−option will mark that last spare faulty.
* −name: The name of the raid set.
* −drive: Specify the backend to rebuild. The backend specification takes the form of the index of the backend in the list of backends. The first backend in a raid set has the index 0.
* −spar:e Optionally specify the spare device which is to be configured onto a backend. This device should already be in the spares pool and be unused. If no spares have been assigned to the backend, then the backend is marked faulty and the spare is added and marked as in use. If spares have been assigned to the backend, then the last spare is marked faulty and the additional spare is added and marked as in use.
* −mkfaulty: The mkfaulty option is used to mark a particular backend as faulty. When a backend of a raid set which does not use spares fails, we need to update the RaidRunner configuration of this fact. This command's sub−options are the same (and have the same effect) as the rebuild command's sub−options excluding the −spare sub−option.
* −repair: Reconfigure a raid set to replace a backend with a spare backend allocated from the spares pool. When the −action option is start then the depth of the new spare and the spare device name is printed. The depth is the number of spares allocated to that backend. When the first spare is allocated, then the depth will be 1. When the second is allocated then the depth is 2 and so on. The depth is used when re−engaging a spare which has just been reconstructed. If the depth is different after the reconstruction has occurred then we know that the spare has failed and another spare has been allocated so we wont attempt the re−engage.
* −name The name of the raid set.
* −drive Specify the backend to replace. The backend specification takes the form of the index of the backend in the list of backends. The first backend in a raid set has the index 0.
* −action: Specify the repair action. Values are either start or finished. Normally, when a spare is allocated, the next steps are to reconstruct the data on that spare then re−engage the spare. If a failure occurs on the spare during reconstruction, then the spare should not be re−engaged. Thus, a raid set should know that a reconstruction is in process. Once the reconstruction is complete then the reconstruction knowledge is cleared. The start value will print out the depth of the newly allocated spare and the spare's device name. If no spares are available an error will be printed and the depth of the last spare for the backend will be printed only. The finished action checks the current depth of spares against a given depth and clears the reconstruction flag if the depth is the same (that is, the spare reconstructed correctly).
* −depth: Specify the depth of the spare device when performing a finished action. A check is made to ensure the current depth of the backend is the same is the specified one. This essentially checks to see if another spare has been allocated on this backend during the reconstruction.
* −replace: This command is used when we have physically repaired (or replaced) the original failed backend and we wish to re−integrate it back into the raid set. This is done by deallocating the current running spare and reconstructing the raid on the replaced original backend. This command is similar to the −repair:

command except that the working spare is deallocated from the backend and returned to the spares pool PRIOR to the reconstruction. This way, if the newly replaced drive fails, we have a spare to reallocate.

- −name: The name of the raid set.
- −backend: The name of the backend device that is being re−integrated.
- −action: Specify the replacement action. Values are either start or finished. When the start action is specified, the last spare on the backend is returned to the spares pool if it is in a working state. Then the current depth of spares on the backend and the backend's index into the last of backends (the raid set's drive number) is printed out. If a failure occurs on the backend during reconstruction, then the backend should not be re−engaged. Thus, a raid set should know that a reconstruction is in process. Once the reconstruction is complete then the reconstruction knowledge is cleared. The finished action checks the current depth of spares against the given depth and clears the reconstruction flag if the depth is the same (that is, the spare reconstructed correctly) and then releases any spares back to the spares pool (all these spares will be faulty). If the depth is different then the reconstruction has failed and an error message is printed and the spares are left alone. −depth Specify the depth of the spare devices when per− forming a finished action. A check is made to ensure the current depth of spares of the backend is the same is the specified one. This essentially checks to see if a spare has been allocated on this backend during the reconstruction.
- −unrepair: Typically, when we repair a raid set using spares, we modify the raid set's configuration to indicate the faulty drive, the allocation of a spare and it's state of "under construction". We then perform the reconstruction, and on success we re−modify the configuration to clear the "under construction" state. IE a sequence of commands like −

```
        rconf −repair .. −action start ...
```

rebuild the raid set

```
rconf −repair .. −action finish ...
```

Now, if the spare we allocated is faulty or the reconstruction fails, we need to turn off the "under construction" state, deallocate the spare and re−mark the original failing drive as faulty (as well as the spare drive if it was faulty). The −unrepair option does this. The options are similar to the

- −repair: options with the addition of the name of the originally faulty backend.
- −name: The name of the raid set.
- −drive: Specify the backend to unrepair. The backend specification takes the form of the index of the backend in the list of backends. The first backend in a raid set has the index 0.
- −backend: The name of the backend device that is being un−repaired.
- −depth: Specify the depth of the spare device as allocated in the original rconf −repair −action start command. This essentially checks to see if another spare has been allocated on this backend during the failed reconstruction or spare testing.
- −init: Initialize the RaidRunner configuration area. With this option all raid set, host port, scsi monitor and spares spool configurations are initialized. The GLOBAL environment variables, and scsi monitor (smon) data blocks which are located in the Raid configuration area, will NOT be deleted. To enable at least one scsi monitor to come up, the scsi monitor lun and it's associated host port scsi id is initialized to a RaidRunner model specific default. Use this option with caution.
- −fullinit: Initialize the RaidRunner configuration area. This option performs the same function as the −init option and additionally clears all GLOBAL environment variables and the scsi monitor (smon) data blocks.
- −check: Cause the multiple configuration areas to be all re−read and, if different, re−written as per the raid configuration area multiple source rules. If more than one raid configuration area differs to the others AND the "correct" area does not include flashram, then rconf returns a non−zero exit status, if all areas are the same or all areas are unreadable (or corrupt) then rconf returns a zero exit status.
- −validate: Perform a consistency check on the current raid sets, spares etc and print out any inconsistencies. If any inconsistencies occur messages will be printed and the return status of rconf will be 1. If no

inconsistencies are present, then nothing will be printed and the return status will be NULL.

- −size: Print the number of bytes actually used in the raid configuration store.
- −syslog: Print any syslog entries stored in the configuration area. Notes that these entries will be deleted on an rconf −init or −fullinit.
- −maxiochunk: Search through all raid sets currently configured and print, in 512−byte blocks, the maximum IO Chunk size of all raid sets.
- −v: Set the verbose option. This option only effects the listing of Raid Sets. It prints additional detail when listing the Raid Sets.
- OPTIONS − spool

  Specifying no options is the same as specifying the −list option.

- −add: Add a disk to the spares pool of the RaidRunner. Sub−options are −
- −name the name of the disk. If the disk is already used by a raid set or is already in the spares pool, an error will be printed.
- −backendsize: the size (in 512 byte blocks) of the added disk. This value can be in the form specified by suffix.
- −type: the type of the spare. Spare devices can either be hot (spun up) or warm (spun down).
- −delete: delete the specified spare from the spares pool. A check is made to see that the spare is not in use.
- −list: Print out the current state of each spare in the pool. The format is

```
Backend BackendSize hot|warm controller_no|Any Used|Unused Faulty|Working
 comments
```

- Configuration Corruption: Under certain conditions, it is necessary to corrupt the RaidRunner configuration area. An example of this is a RaidRunner that has been configured in such a way that all memory is consumed and the RaidRunner will not respond to any commands either on the console or via the scsi monitor. Given there is not enough memory to delete raid sets or even execute the reboot process (or any other for that matter), then a command which corrupts all the RaidRunner configuration areas (and hence will prevent the automatic creation of raid sets at reboot − i.e the consumer of memory) and then immediately reboot's without using any additional memory will allow the user to reconfigure the RaidRunner when it has re−booted. The corrupt−config command performs this task.
    - RETURN VALUES: If an error occurs, the $status environment variable is set to 1, else null. When the configuration change options (−C or −c) are given, the return value ($status) will be 1 if a configuration has changed since boot else 0.
    - BUGS: Currently, if you duplicate sub−options within a command, the last occurrence will be the value set. For example if the command rconf −modify −name F −newname B −newname C −cache 1M −cache 2M is executed then the newname will be "C" and the cache size set to 2M (2 megabytes).
    - SEE ALSO: husky, autorepair, replace, suffix, setenv, unsetenv, printenv, smon

13.72 REBOOT − exit K9 on target hardware + return to monitor

- SYNOPSIS: reboot [−i] [−s] rboot [−i] [−s] rbootimed
- DESCRIPTION: reboot abruptly leaves the K9 kernel (on target hardware) and either re−enters the underlying hardware monitor or re−initializes the K9 kernel. Aside from flushing the write cache and initialising the battery backed−up ram no cleanup is performed on K9 processes (no signals, no nothing). If no options are given then the reboot will NOT do any memory tests prior re−initialising the K9 kernel. If the −s option is given then memory tests will be performed prior to re−initialising the K9 kernel. During these memory tests, a spinning − is displayed on the RaidRunner's console. This indicates the memory test is executing. If you press the space key during this test, then you will

drop into the underlying RaidRunner monitor. If the −i option is given then all I/O to the back−end drives will inhibited. That is, no spun−down drives will be started and no attempt to flush the cache is performed. rboot is an alias for reboot. rbootimed provides a special entry−point into the reboot process and is equivalent to calling reboot with the −i option. Unlike reboot, which is executed as a sub−process, rbootimed is interpreted via husky and immediately commences the reboot process and hence does not use any extra memory. This is useful when the RaidRunner has run out of memory and you need to reboot the RaidRunner into single user state. On simulation platforms such as Unix this command is not implemented (but typically the Unix "interrupt" signal has the similar effect of killing the Unix process containing the K9 simulation).

13.73 REBUILD − raid set reconstruction utility

- SYNOPSIS:
    - rebuild −d file −s size −b bufsize −D dno −R rname −n nbends −r rtype [−p pri] [−v] [−S sleep_period]
    - rebuild −d file −l [−v]
    - rebuild −L [−v]
- DESCRIPTION: To reconstruct data onto a newly incorporated drive in a raid set, the raid set's repair entry point is to be completely read. Additionally the read must be in multiples of the raid set's bufsize and the last read must align to the end of the raid set (i.e cannot attempt to read past the end of the raid set). The rebuild utility performs this reconstruction. When rebuild starts, it allocates an internal buffer whose size is maximized to the available memory and which is a multiple of the given buffer size. The raid set's repair entry point is read via these large buffers and the last read is guaranteed to align to the end of the raid set. As rebuild reads from the raid set's repair entry point, a check is made to see if the backend that is being rebuilt is still engaged to the raid set, and if the backend has failed (disengaged from raid set) rebuild prints a message and returns an exit status of 2. If the raid set fails during reconstruction (a second backend fails) then rebuild prints a message and returns an exit status of 1. Rebuilds, by default, consume large amounts of RaidRunner resources which would result in a large reduction in RaidRunner I/O throughput (to the host). A priority can be given to the rebuild program to reduce this demand on resources. This priority ranges between 0 (use minimal resources) and 9 (use maximum resources available). Naturally by specifying a low priority, the time to complete a rebuild will increase. The current state of any completed or "in−progress" rebuild is recorded for informative uses. Up to 10 of these records are kept at any one time. If a new rebuild is requested and we have already used all 10 records, we find the first "inactive" (or completed) record and re−use that one. If all 10 records are currently marked "active" (which means that we currently have 10 rebuilds currently in progress), then no record of the new rebuild will be kept. These records are initialized at boot time.
- OPTIONS
    - −d file: Specify the file to read. This is typically the repair entry point of the raid set.
    - −s size: Specify the raid set's size, in 512−byte blocks.
    - −b bufsize: Specify the raid set's IO chunksize − bufsize, in 512−byte blocks. In the case of a raid type 3, this should be the number of data drives times the size of the IO chunksize.
    - −n nbends: Specify the number of backends in the raid set. This is needed along with the bufsize option to ensure that the raid set's repair device is read in multiples of the raid set's stripe size.
    - −D dno: Specify the backend index (within the raid set) that is being rebuilt.
    - −R rname: Specify the name of the raid set that is being rebuilt. Specify the raid type of the raid set being rebuilt. Must be 1, 3 or 5.
    - −p pri: Specify a priority at which the rebuild is to be run at. The priority, pri, must be a number between 0 (lowest) and 9 (highest). When rebuild is executed by automatic or

manual scripts an environment vari− able, RebuildPri, is checked to see what priority a raid set should be re−built. If this variable is not set, then the default priority is 5. See environ for details.

- −S sleep_period: Specify a number of milliseconds to sleep between each reconstruct access of the raidset. The period, sleep_period, must be a number between 0 (no sleep) and 60000 (60 seconds). When rebuild is executed by automatic or manual scripts an environment variable, RebuildPri, is checked to see what sleep period a raid set should use. If this variable is not set, then the default period for the given priority is used. See environ for details.
- −v: Set verbose mode. When rebuilding, if verbose mode is set, the current number of reads and total expected number of reads is printed for every 10 percent of the reconstruction completed. When printing the state a specific or all rebuilds, verbose mode will produce a long listing of the rebuild status.
- −l: Print the current state of the specified rebuilding (−d) file.
    - By default the following colon separated fields are printed
    - the rebuild file (entry point to raid set repair device)
    - the size of the file (raid set) in 512 byte blocks
    - the IO chunksize of the file (raid set) in 512 byte blocks
    - the size of the buffer (in 512 byte blocks) being used for reading
    - the total number of reads needed to read the whole file
    - the current number of reads performed so far,
    - the priority of the rebuild
    - the number of milliseconds between each access to sleep
    - the state of the rebuild − either "active" or "complete" the error message that this rebuild failed on − set to "none" if the rebuild successfully completed.

  When verbose mode (−v) is set, the above is printed with appropriate labels.
- −L: Print the rebuild status of ALL rebuilds in progress or completed (in the same form as for the −l option).

- PRIORITY: The speed at which a rebuild occurs and impact it has on the system is determined by the priority and optional sleep period between accesses. When rebuild allocates the buffer to use it will limit this buffer's size to the maximum available contiguous memory segment or 2 Megabytes which ever is the lessor. The memory will then be scaled by pri + 1 / 10. That is, if you specify a priority of 0 then the buffer will be one tenth of the maximum buffer siz, if you specify a priority of 4 then it will be half. This resultant buffer size is then trimmed to ensure it is a multiple of a raid set's stripe size. Between each read/write access of the raid set's repair device, rebuild may sleep for a given period of milliseconds to reduce the overhead of the rebuild process. This sleep period may be given on the command line. If, not then the priority value is used to work out the period. The table below specifies what period is associated with which priority.
    - 0: 2000 milliseconds (2.00 seconds)
    - 1: 1750 milliseconds (1.75 seconds)
    - 2: 1500 milliseconds (1.50 seconds)
    - 3: 1250 milliseconds (1.25 seconds)
    - 4: 1000 milliseconds (1.00 seconds)
    - 5: 750 milliseconds (0.75 seconds)
    - 6: 500 milliseconds (0.50 seconds)
    - 7: 250 milliseconds (0.25 seconds)
    - 8: 100 milliseconds (0.10 seconds)
    - 9: 0 milliseconds (no sleep)
- EXIT STATUS

  The following exit values are returned:

- 0: Successful completion.

- 1: The raid set failed reconstruction. This means that the raid set has failed.
- 2: The raid set's backend that is being rebuilt has failed. This means that the raid set is still valid.
  - SEE ALSO: raid1, raid3, raid5, repair, replace, environ

13.74 REPAIR − script to allocate a spare to a raid set's failed backend

- SYNOPSIS: repair raidsetname backend
- DESCRIPTION: repair is a husky script which is used to allocate a spare drive to a failed backend of a raid set. This is typically done when a raid set has a failed backend and no spares were available at the time of failure and now you want to allocate the spare (as opposed to fixing the failed backend). After parsing it's arguments repair get's a spare backend (of appropriate type) from the spares pool. The spare drive is then engaged in write−only access mode in the raid set and a reconstruct of the raid occurs (read of the whole raid set). This read is from the raid file system repair entrypoint. Reading from this entrypoint causes a read of a block immediately followed by a write of that block. The read/write sequence is atomic (i.e is not interruptible). On successful completion of the reconstruction, the spare is then engaged in the correct iomode for the raid set. The process that reads the repair entrypoint is rebuild. This device reconstruction will take anywhere from 10 minutes to one and a half hours depending on both the size and speed of the backends and the amount of activity the host is generating. During device reconstruction, pairs of numbers will be printed indicating each 10% of data reconstructed. The pairs of numbers are separated by a slash character, the first number being the number of blocks reconstructed so far and the second being the number number of blocks to be reconstructed. Further status about the rebuild can be gained from running rebuild. Checks are made to ensure that the raid set is running, the spare works, and that there is no failure of the spare during reconstruction.
- OPTIONS
  - raidsetname: The name of the raid set.
  - backend: The name of the failed backend which is being replaced by a spare.
- SEE ALSO: rconf, rebuild

13.75 REPLACE − script to restore a backend in a raid set

- SYNOPSIS: replace raidsetname backend
- DESCRIPTION: replace is a husky script which is used to reconfigure back in a physically repaired backend of a type 3 or 5 raid set. After parsing it's arguments replace does a quick read/write test of the specified backend to ensure it's working. If a working spare is currently running in place of the backend, it disengages it and returns it back to the spares pool. The backend drive is then engaged in write−only access mode in the raid set and a reconstruct of the raid occurs (read of the whole raid set). This read is from the raid file system repair entrypoint. Reading from this entrypoint causes a read of a block immediately followed by a write of that block. The read/write sequence is atomic (i.e is not interruptible). On successful completion of the reconstruction, the backend is then engaged in the correct iomode for the raid set and any other faulty spare backends that were associated with that backend are returned to the spares pool. The process that reads the repair entrypoint is rebuild. This device reconstruction will take anywhere from 10 minutes to one and a half hours depending on both the size and speed of the backends and the amount of activity the host is generating. During device reconstruction, pairs of numbers will be printed indicating each 10% of data reconstructed. The pairs of numbers are separated by a slash character, the first number being the number of blocks reconstructed so far and the second being the number number of blocks to be reconstructed. Further status about the rebuild can be gained from running rebuild. Checks are made to ensure that the raid

set is running, the backend works, and that there is no failure of the backend during reconstruction.

* OPTIONS
    * raidsetname: The name of the raid set.
    * backend: The name of the backend which is being restored.
* SEE ALSO: rconf, rebuild

13.76 RM − remove the file (or files)

* SYNOPSIS: rm [−f] [file ...]
* DESCRIPTION: rm remove the named file (or files). [A directory is considered to be a file.] If all the given files can be removed then NIL (i.e. true) is returned as the status; otherwise the first file name that could not be removed is returned (and this command will continue trying to remove files until the list is exhausted). If the −f option is given, then files that could not be removed are ignored and NIL (i.e. true) is returned as the status.
* BUGS: The use of rm on non−empty directories orphans those child files.

13.77 RMON − Power−On Diagnostics and Bootstrap

* SYNOPSIS

```
      v1.0
DRAM: 010 Mb

Batt: B0000008-B007FFFC

PWROK 1:BAD 2:OK

0:720seE 1:720seE 2:720seE 3:720seE 4:720seE 5:720seE 6:770B 7:770B

S/N: ABC-12345 B

-
```

* DESCRIPTION: rmon will normally perform a power−on diagnostics and then bootstrap the main raid code. By typing a space before the count−down has finished rmon will abort the power−on diagnostics and prompt for a command to be entered from the console serial port.

  The first line printed by rmon is it's version number. The second line is the size (in hexa−decimal) of DRAM. The next line indicates the type and usage of Battery−Backup SRAM found. This can be a variety of values and if there are two hexa−decimal values printed, this is then the range of inclusive addresses upon which the power−on diagnostics will test Battery Backed−up SRAM. If there is only one value this indicates the size of Battery Backed−up SRAM, but no power−on diagnostics will performed upon it. All possible values are :−

```
Batt: 00000000          No Battery Backed-up SRAM present.



Batt: B0000000-B007FFFC 512Kb SRAM, B0000004==0 (No data).

Batt: B0000008-B007FFFC 512Kb SRAM, B0000004==1 (No data).

Batt: B00XXXXX-B007FFFC 512Kb SRAM, B0000004==2 (Saved data,
```

```
                        B0000000==B00XXXXX, where B00XXXXX is
                        start of unused data).
Batt: B0080000          512Kb SRAM present, but the contents of
                        SRAM is not one of the above, or could
                        be B0000004==2 and B0000000==B0080000.
                        No pon testing done on it.

Batt: B0000000-B03FFFFC 4Mb SRAM, B0000004==0 (No data).
Batt: B0000008-B03FFFFC 4Mb SRAM, B0000004==1 (No data).
Batt: B0XXXXXX-B03FFFFC 4Mb SRAM, B0000004==2 (Saved data,
                        B0000000==B0XXXXXX, where B0XXXXXX is
                        start of unused data).
Batt: B0400000          4Mb SRAM present, but the contents of
                        SRAM is not one of the above, or could
                        be B0000004==2 and B0000000==B0400000.
                        No pon testing done on it.
```

Note the case where location B0000004 has a value of 2 (Saved data present), but where B0000000 (the start of unused data) points to a word address just past the last byte of Battery Backed−up SRAM. This latter case will cause a single value (say B0080000, for 512Kb Battery Backed−up SRAM) to be printed. This case does not indicate that any contents of Battery Backed−up SRAM is deemed incorrect, but that Battery Backed−up SRAM is full of data and that no power−on diagnostics will be performed upon Battery Backed−up SRAM.

The DRAM memory sizing algorithm will cater for between 8Mb and 256Mb of DRAM, and there need be only 30 bits out of 32 bits words in each DRAM set that need work properly, for the sizing to function properly. The DRAM consists of two banks of memory, A and B, each of two slots. When using a bank of memory both slots of that bank must be occupied with a Simm of the same type and size. If only one bank is to be used, then it must be Bank A. The type of Simm memory may be single or double sided. Placing two double sided Simms (which, as mentioned above must be both the same size) into Bank A is equivalent to populating all four slots with single sided Simms of a size that is exactly one half of the double sided Simms size. The Bank B in such a case cannot contain any Simms. Each bank can potentially be using single sided Simms of different sizes, but the first bank must have the larger sized Simms.

```
DRAM Size DRAM last address
Hex Dec   Cached    Uncached
008  8    7FFFFF    A07FFFFF
```

```
010  16   FFFFFF    A0FFFFFF

020  32   1FFFFFF   A1FFFFFF

040  64   3FFFFFF   A3FFFFFF

080  128  7FFFFFF   A7FFFFFF

100  256  FFFFFFF   AFFFFFFF
```

On the fourth line is the power supply status. This will be one of two variations. The first for single power supply systems, the power supply summary status is printed at PWROK (for good) or PFAIL if this is not good. The second is for multi−power supply systems, the summary status will be followed by the individual status of each power supply. A PFAIL will be printed if any of the power supplies are faulty. At this point on non−pseudo−static DRAM systems the monitor will hang if the summary status is PFAIL. A pseudo−static DRAM system will never print this line or any of the preceding lines (i.e. version line, DRAM size or Battery Backed−up SRAM size) and always immediately hang.

On the fifth line is summary of the SCSI chips used in the system. Each item in this report consists of the index No. for that chip (from 0 to 7), followed by a ':' then a string of text indicating the model. This model consisting of it's chip type and revision level. A summary of the known part No.s vs the model is :−

```
                         Bits 7 to 4 of

 Model    Part No.    MACNTL CTEST3

  720B   609-0391071   0000   0001

  720C   609-0391324   0000   0010

  720E   609-0391955   0000   0100

720seE   609-0391949   0001   0100

  770A   609-0392179   0010   0000

  770B   609-0392393   0010   0001
```

There maybe a sixth indicating the serial number of the unit. If the last 12 bytes of the FlashRam boot partition is left in an unblown state (all ones), then this line will not be appear. Otherwise these last 12 bytes will be printed. These 12 bytes may include trailing blanks. The four bytes of FlashRam preceding the serial no. contain 26 bits representing the revision level of the board. If any of these bits are cleared and there is a serial no. present then the revision level from A to Z will be printed.

After printing the above configuration information, the DRAM and Battery Backed−up SRAM (if any found), will have a Knaizuk−Hartmann memory test performed upon them. This memory test does a quick non−exhaustive check for stuck−at faults in both the address lines as well as the data locations. This test is disabled on the following conditions :−

- If the upper 17 bits of first location, contain the hex bit pattern 0x3C1A0000 and there are no parity errors on thence reading all of the rest of DRAM. Such a condition will result in the code starting at the first location to subsequently executed.
- If no DRAM is found. The monitor will not autoboot the main raid code, but print a monitor prompt and wait

for a monitor command (see below).

By typing a space, any memory tests that may have been started will be aborted. The monitor will print a prompt and wait for a monitor command (see below).

All other conditions will result in the main raid code to be read from FlashRam and started. Whilst the memory test is in progress a sequence of −, /, | and characters are printed on the console giving the appearance of a rotating bar. On−board LED 1 will be flashed every 10ms and LED 2 will change state between each phase of the test. As noted above these tests can be abort by typing a space on the console. When rmon is in command mode a variety of commands may be issued. These mostly relate to various diagnostics that can be performed, or to how flash RAM may be upgraded with new firmware.

- GENERAL COMMANDS: All values and addresses used in the monitor are to be in hexadecimal. Commands must start at the beginning of a line and are case sensitive. Commands must be separated by semi−colons ';' or a newline. If a command syntax error occurs, then that command's usage line will be printed. If a command (e.g. command_name) is not recognized, the message unknown command: command_name will be printed. Note that any command that attempts to modify regions of memory that are occupied by the monitor will cause unpredictable results. Currently the monitor uses locations from CPU addresses 801E0000 to 801FFFFF inclusive.
- ?: This command, will print out a summary of the more generally useful commands available in the monitor.
- { some comment }: will ignore all text from the leading open−brace to the first closing − brace. This allows you to comment your monitor scripts (set of monitor commands) prior to downloading. Please NOTE that comments cannot be nested, that is there can only be one opening−brace and closing−brace on a line.
- * count commands ...: will repeat all commands up to the newline count times.
- (commands): will execute the sequence of commands from the opening parentheses to the closing parentheses (or new− line) as though they were on a line by themselves. For example to nest repeat (*) commands one could enter the line

```
* 20 (* 8 pb 5F 0); (* 10 pb 6F 1)
```

which will put the byte 0 into memory location 5F eight times, then put the byte 1 into memory location 6F ten times, and repeat both these put byte commands 20 times. Thus the location 5F will have the byte value 0 written into it a total of 160 times and the location 6F will have the byte value 1 writ− ten into it a total of 200 times.

- b: The command will boot the main raid code.
- ba: The command will auto−boot the main raid code, as if from power up.
- g <address>: Go to (jump to) a code address and start execution at that address.
- l: Download a program or data over the serial port. The data is expected in Motorola Packed S format. If an error occurs during download then the message Unrecoverable error in S−format loader.
- lg: Download a program over the serial port and commence execution of the program. The program data is expected in Motorola Packed S format. After the data or program has been downloaded successfully, then commence execution at the load address (specified within the downloaded file). If an error occurs during download then the message "Unrecoverable error in S−format loader". will be printed and the program will not be executed.
    - MEMORY COMMANDS: The following commands are used to display, change or test memory. Most of these commands have three forms depending on the size of memory access. These forms are − a byte form (8−bit), a short word form (16−bit) and long word form (32−bit). When using the short or long word form, any addresses used must be short word (16−bit) or long word (32−bit) aligned. If an unaligned address is given, the error message

```
cmd: bad address alignment
```

is printed. To differentiate between the three forms of memory access, the suffixes b, w and l are used for byte, short word and long word respectively.

* db|dw|dl start_address [end_address]: These commands display the contents of the specified memory location or memory range. Memory can be displayed in byte, short word or long word sizes using the db, dw and dl commands respectively. If no ending address is given, then only the contents of the first address is displayed. Each line of output from these commands have the format: address: contents contents .... contents. Where up to 16 bytes are printed on each line. For example

```
{rmon} db 60 7F

00000060: A0 9F 9E 9D 9C 9B 9A 99 98 97 96 95 94 93 92 91

00000070: 90 8F 8E 8D 8C 8B 8A 89 88 87 86 85 84 83 82 81

{rmon} dw 60 7F

00000060: FFA0 FF9E FF9C FF9A FF98 FF96 FF94 FF92

00000070: FF90 FF8E FF8C FF8A FF88 FF86 FF84 FF82

{rmon} dl 60 7F

00000060: FFFFFFA0 FFFFFF9C FFFFFF98 FFFFFF94

00000070: FFFFFF90 FFFFFF8C FFFFFF88 FFFFFF84
```

* pb|pw|pl address value ....: These commands put the given value(s) into the specified memory location. Memory can be addressed in byte, short word or long word sizes using the pb, pw and pl commands respectively. If more than one value is given, then the first value is put into the specified memory location, the next value is put into the next aligned memory location, and so forth. For example, the commands

```
       {rmon} pb 60 1 2
{rmon} pw 70 04F2 002F

{rmon} pl 80 8000FF04 8000FF0F 80FFFFFF
```

  will put the value 01 into location 00000060 and 02 into the location 00000061, put the value 04F2 into location 00000070 and 002F into location 00000072, and put the value 8000FF04 into location 00000080, 8000FF0F into location 00000084 and 80FFFFFF into location 00000088 respectively

* fb|fw|fl start_address end_address value: These commands fill the memory given by the start and end address with the given value. Memory can be addressed in byte, short word or long word sizes using the fb, fw and fl commands respectively. The start and ending addresses must be aligned to the given memory sizes. For example, the commands

```
       {rmon} fb 60 70 8F
{rmon} fw 60 70 FF8F

{rmon} fl 80 90 00FFFF11
```

  will put the value 8F into all bytes (inclusive) between locations 00000060 and 00000070, put the value FF8F into all short words (inclusive) between locations 00000060 00000070, and put the value 00FFFF11 into all long words (inclusive) between locations 00000080 and 00000090.

* sb|sw|sl start_address end_address value: These commands search the memory given by the start and end addresses for the first occurrence of the given value and will print out the address where the value was found and the value. If the given value is not found, then nothing will be printed. Memory can be addressed in byte, short word or long word sizes using the sb, sw and sl commands respectively.
* cb|cw|cl start_address end_address value: These commands compare successive memory locations given by the start and end addresses with the given value and will print out the address and it's contents of the end_address, if all the memory was the same, or the address of the first location where the value was different to the given value along with the different value. Memory can be addressed in byte, short word or long word sizes using the cb, cw and cl commands respectively.
* tb|tw|tl start_address end_address seed: These commands generate pseudo−random values and store these values in successive memory locations given by the start and end addresses. When all the memory has been filled, the memory is read and it's contents is compared with what was written. Memory can be addressed in byte, short word or long word sizes using the tb, tw and tl commands respectively. The given seed will initialize the pseudo−random number generator. If successive executions of this command use the same seed, then the random values will always be the same. When the random values are being written, a dot ('.') will be printed for each location. When the read is being performed, a dot ('.') will be printed if the data equals what was written, else an 'X' will be printed. For example

```
        {rmon} tw 60 70 7B
Writing:

Checking:

{rmon}
```

* m start_address end_address mask: This command will run low level memory tests on the given memory range. The start and end addresses must be long word aligned. If the test passes the message Passed. will be printed, and if it fails the message Failed. will be printed. The given mask will determine what type of tests are run. The table below shows what each mask will do. The tests essentially write to memory and then read from memory comparing what was read with what was written. The first twelve tests below first write to all the memory specified and then reads all the memory specified, the next twelve tests do an immediate read after the write. The masks can be or'd together to run multiple tests.

```
        0x0001         Fill each byte with 0x00
0x0002         Fill each byte with 0xFF

0x0004         Fill alternate bytes with 0x55 then 0xAA

0x0008         Fill each byte with the two's complement of it's address

0x0010         Fill each word with 0x0000

0x0020         Fill each word with 0xFFFF

0x0040         Fill alternate words with 0x5555 then 0xAAAA

0x0080         Fill each word with the two's complement of it's address

0x0100         Fill each long word with 0x00000000

0x0200         Fill each long word with 0xFFFFFFFF

0x0400         Fill alternate long words with 0x55555555 then 0xAAAAAAAA

0x0800         Fill each long word with the two's complement of it's address
```

```
0x0001000      Fill each byte with 0x00

0x0002000      Fill each byte with 0xFF

0x0004000      Fill alternate bytes with 0x55 then 0xAA

0x0008000      Fill each byte with the two's complement of it's address

0x0010000      Fill each word with 0x0000

0x0020000      Fill each word with 0xFFFF

0x0040000      Fill alternate words with 0x5555 then 0xAAAA

0x0080000      Fill each word with the two's complement of it's address

0x0100000      Fill each long word with 0x00000000

0x0200000      Fill each long word with 0xFFFFFFFF

0x0400000      Fill alternate long words with 0x55555555 then 0xAAAAAAAA

0x0800000      Fill each long word with the two's complement of it's address

0x1000000      Fill alternate long words with 0x00000000 then 0xFFFFFFFF
```

- r start_address end_address read_size:: This command will generate and print a 32−bit Cyclic−Redundancy−Check value for the block of memory extending from start_address to end_address using reads of the size specified by read_size which can either be 1, 2 or 4 for byte, word or long word reads respectively. The start and end addresses must be long word aligned.
- FLASH MEMORY COMMANDS: The Flash RAM used is AMD's Am29F040, a 4Mb (512k x 8bit) device that is comprised of 8 equi−sized sectors with each sector of 64Kb. The board may potentially have a second Flash RAM chip. The total storage, regardless of the number of Flash RAM chips is divided into 4 regions, the first 3 are each 1 (64kB) sector and contain the Bootstrap, Raid Configuration and Husky Scripts at the CPU addresses of BFC00000, BFC10000 and BFC20000 respectively. The Main region contains 5 64Kb sectors (starting at BFC30000) and optionally all 8 64Kb sectors of the second FLASHram (starting at BFC80000). If this second chip is present there is then a total of 13 64Kb sectors for the main raid code. The Manufacture ID and Device ID for the AMD Am29F040 is 01 and A4 respectively. Thus the text "Flash: 01A4" is printed on either of the E command or the W command accessing this device. Attempting to erase or write to the Boot sector will always cause an "Illegal Flash RAM address range".
    - E sector_address: An entire region can be bulk erased by the 'E' command by specifying an address in that region, the main region will erase all sectors starting with that sector containing the nominated address and erase to the end of that region, but not crossing a chip boundary. Thus the 'E BFC40000' command will erase the last 4 sectors of the first chip. The 'E BFC30000; E BFC8000' commands will erase the last 5 sectors of the first chip and then erase the entire second Flash RAM chip (if it is present). Attempting to erase a non−existent second chip cannot report an error, however a negative number is printed to indicate other errors, as described in the following table.

      ```
      Error                Fault Description

      −1                   Illegal Flash RAM address range.

      −2                   A sector within given range that is Write Protected.
      ```
- W sector_address source_address byte_count: This command will copy byte_count bytes of data from source_address to EEPROM locations starting at sector_address. The number of bytes copied or on error, a

negative number will be printed. The negative error numbers are described in the table below. Optionally a W command may appear without any arguments, in this case the sector_address will be that specified by a previous E command and the source_address and byte_count will be that given to and that printed by, respectively, a previous T command (see below). This command, unlike the E command, will modify the contents of the second Flash RAM chip, if it is implied by sector_address plus byte_count being equal or greater than the CPU address BFC80000. An attempt to write with a byte_count of 0 is not an error, and will cause only the Manufacturer and Device ID to be printed.

```
       Error                Fault Description
-1                    Illegal Flash RAM address range

-2                    A sector within the address range that is Write Protected

-3                    On a failed Byte Program command
```

- NETWORK COMMANDS: The second RS232 port on the RaidRunner can be used to download programs and data into memory from a host. This port can run the TCP/IP SLIP protocol and the Trivial File Transfer Protocol (TFTP) is available for file transfer. Three variable are available to effect download file transfers from a remote host. These are the remote host IP address, the local IP address (of the RaidRunner) and the name of the file on the remote host that is to be downloaded. These variables have default values of C02BC60E (192.43.198.14), C02BC6FD (192.43.198.253) and /usr/raid/lib/raid.bin where are the remote host IP address, local IP address and remote host's file to down− load respectively. These default values are dynamic in that, if you change them, they will revert back to their default values at power−on or whenever the boot monitor is started. The following commands manage these variables and file transfers.
    - A [remote_ip_address [local_ip_address]]: With no arguments, this command prints, in hexadecimal, the dynamic remote and local IP addresses used by TFTP and SLIP. To change the remote IP address, execute this command with only one argument − the IP address (in hexadecimal) of the remote host. To change the local IP address (of the RaidRunner), the execute this command with two arguments, the first being the IP address of the remote host and the second being the IP address of the RaidRunner. Remember both addresses must be in hexadecimal.
    - F [filename]: With no arguments, this command prints the filename of the file to download from the remote host. To change the filename, specify it as the argument to this command. Remember, the permissions on this file on the remote machine, must be set to allow remote TFTP access by the RaidRunner.
    - P: This command will ping (send an ICMP ECHO request packet and time the response) the remote internet host. If the remote host responds, the message

```
       Sending echo request...
Got response after 0x0000nn ms
```

will be printed. nn is the number of milliseconds it took for the remote host to respond. If the remote host does not respond within 10 seconds, the message

```
Sending echo request...

timeout after 10 seconds
```

will be printed.

- T start_address end_address: This command will transfer from the remote host the stored filename into a block of memory starting at
- start_address and limited to end_address. The number of bytes that will be transferred will be either the size of the remote file or the value computed by end_address − start_address + 1, whichever is the lessor. Whilst the data is being transferred, a dot ('.') will be printed for each packet of data transferred. At the end of a

successful transfer, the number of bytes transferred will be printed. If the transfer fails, an appropriate error message will be printed.

The following two examples show the transfer of the raid binary and the transfer of the RaidRunner's /bin directory from a remote host. The remote host's IP address is 192.43.198.101 (C02BC665), the RaidRunner's IP address is 192.43.198.200 (C02BC6C8) and the two files concerned will be /usr/raid/lib/raid.bin and /usr/raid/lib/raid.rc (raid binary and /bin directory respectively). The raid.bin file will be written into the bank of flash ram starting at address bfc10000 and the raid.rc file will be written into the two consecutive banks of flash ram at address bfc08000 and bfc0c000.

```
{rmon} {first transfer the raid binary − /usr/raid/lib/raid.bin}

{rmon} F /usr/raid/lib/raid.bin

{rmon} A C02BC665 C02BC6C8

{rmon} P

Sending echo request...

Got response after 0x000013 ms

{rmon} T 80300000 803fffff

{rmon} E bfc10000; {erase the Flash EEPROM address for raid.bin
 to be stored}

**

70000

{rmon} W bfc10000 80300000 5BDA0; {copy downloaded raid.bin
 into flash}

5BDA0

{rmon}

{rmon} {now transfer the raid /bin directory − /usr/raid/lib/raid.rc}

{rmon} F /usr/raid/lib/raid.rc

{rmon} A C02BC665 C02BC6C8

{rmon} P

Sending echo request...

Got response after 0x000013 ms

{rmon} T 80300000 803fffff

{rmon} E bfc08000; {erase the two consecutive blocks of
 flash ram where raid.rc}
```

```
4000

{rmon} E bfc0c000; {is to be stored}

4000

{rmon} W bfc08000 80300000 4000; {copy the first 0x4000
 bytes}

4000

{rmon} W bfc0c000 80304000 4000; {copy the next 0x4000 bytes}

4000

{rmon}
```

- Sno ABC−12345: Prints / sets the serial number. Without an argument the current serial number will be printed, if no serial number has been set then nothing is printed. If there any trailing characters after 'Sno' command, then this will indicate that these characters are to be set permanently as a serial number. A serial number may only be set once, there after any attempt to set another one will be ignored and the current serial number printed. If the serial number has to be re−set then the flashram needs to be reblown as if it were brand new. Note that when setting the the serial number for the first time, the exact format of the command is very important. There will be exactly one space after the the command 'Sno', after which the next 12 printable characters are taken as the serial number. If there are less than 12 characters (before the carriage return), then blank characters are padded on the right−hand side.

  It is suggested that a serial number be no greater than 10 characters and consist of upper case letters, decimal digits and only a limited number of special characters such as '−' or '.' characters. The main raid code will truncate the serial number to 10 characters if a revision level is present. The main raid code will, if the revision level is present, append the revision level immediately after the serial number and present the combined string as the serial number in an inquiry command.

- Rev A: Prints / sets the board revision level. Without an argument the current revision level will be printed as an upper case letter (from 'A' to 'Z'). If immediately following the 'Rev' command there is a space and a single upper case letter then this will indicate that this revision level is to set permanently. The revision level can be increased (e.g. from 'A' to 'B' to 'C' etc.), but can not ever be decreased.
- DIAGNOSTICS:

  x000 General form of a diagnostic command it that is it preceded by an x and then immediately followed by a hexi−decimal no. Most diagnostics will loop permanently requiring a power off/on cycle to stop the diagnostic.

- x000 chip: Will continually assert / de−assert most SCSI bus signals. These include DB0−15, I/O, REQ, C/D, SEL, MSG, RST, ACK, BSY, ATN. The DP0 and DP1 (i.e. parity bits) can not be assert / de−asserted in this fashion.
- x001 chip: Will continually assert / de−assert most SCSI bus signals in such a fashion as to indicate the number of the SCA connector that this pin would be connected to. This count is w.r.t. to the LED1 and LED2 when used to trigger a scope (on the falling edge). Each complete group of ten counts are coalesced to facilitate counting on a scope. The counts are:−

```
       7  − DB11       14 − SEL         21 − DB7        28 − DB0
8  − DB10       15 − MSG         22 − DB6        29 − DP1 (*2)
```

```
9  - DB9        16 - RST (*1)    23 - DB5        30 - DB15

10 - DB8        17 - ACK         24 - DB4        31 - DB14

11 - I/O        18 - BSY         25 - DB3        32 - DB13

12 - REQ        19 - ATN         26 - DB2        33 - DB12

13 - C/D        20 - DP0 (*2)    27 - DB1
```

Note that the RST line (marked *1 above) is pulsed only once after the last pin is pulsed (i.e after DB12) and that the parity lines (marked as *2 above) can not be pulsed.

- x002 dst_addr src_addr length: Calls the C library routine memcpy with three arguments of destination address, source address and length (all in hex).
- x003 200000 ffffff and x004 200000 ffffff: Two commands are used to exercise Battery Backed−up DRAM. The x003 command will set the memory range as suggested above. The value of 200000 is one location past the last used flash monitor memory location, ffffff is the last memory location of 8Mb of DRAM (this latter figure can be varied according to available DRAM memory). The DRAM starting at 0 will be modified to contain :−

```
        00: 0x3C1A9FC0   [ lui k0,0x9fc0 ]
04: 0x3C1B3C1A   [ lui k1,0x3c1a ]

08: 0xAC1B0000   [ sw  k1,(zero) ]

0C: 0x3C01A000   [ lui at,0xa000 ]
```

Location 0 is the start of the restart pseudo interrupt handler. The x004 command is then used after restoring power to check for this bit pattern. It is important that the arguments given to x003 are exactly the same as those given to x004. Any words detected that are different are printed, if there are no differences then nothing (but the next prompt) is printed.

  - NOTE: As a side effect of the above executing the above restart pseudo interrupt handler on restoring power, location 0 will be patched with 0x3C1A0000 to prevent memory tests and autobooting the main raid code. This feature can be re−enabled by depositing 0 into location 0 (i.e `pl 0 0`).

13.78 RRSTRACE − disassemble scsihpmtr monitor data

- SYNOPSIS: rrstrace −f monitor_file [−b bcnt] [−v]
- DESCRIPTION: rrstrace will disassemble the monitoring data collected by scsihpmtr which has been transferred to a host via ssmon. rrstrace will additionally analyze all scsi reads and look for sequences of reads and report them. This repeating sequence is referred to as a 'natural read'. This data can be then used for a more optimal configuration of the raid. The natural reads are reported via a starting block size followed by the total length (in blocks) of the natural read, how many reads made up the natural read and finally how many times it occurred. At the end the number of natural reads found are printed along with how many were looked for and how many actually existed. By default up to 256 different natural reads can be scanned for, but if more are required then the −b bcnt option can be given to allocate a deeper search. Note that a natural read that occurs only once is recorded but not printed out.
- OPTIONS:
  - −f monitor_file: Specify the name of the file from which the data is to be read
  - −b bcnt: Specify a larger natural read analysis count

- −v: Turn on verbose mode and hence print out every scsi instruction. In verbose mode, each line of output from rrstrace is of the form time chipno lun ssid msg_id tag command, where:
  - time: is the time the command occurred
  - chipno: is the chip number (hostport) upon which the command came in
  - lun: is the target lun the command was directed to
  - ssid: is the scsi id of the initiator (i.e host)
  - msg_id: is the scsi message id of the command
  - tag : is the scsi tag number of the command or −1 if no tag is associated
  - command: is the scsi command itself in text or if unknown to rrstrace the 12 byte scsi command block will be printed. In the case of read and write commands, the block address and length is printed along with the block address of the next sequential read/write command if the next command were to be sequential (this address is parenthesized). If the command has the 'Force Unit Access' bit set, then FUA is appended to the line. The scsi Read_6, Write_6, Read_10 and Write_10 commands will be printed as R6, W6, R10 and W10 respectively.
- SEE ALSO: ssmon, scsihpmtr

13.79 RSIZE − estimate the memory usage for a given raid set

- SYNOPSIS: rsize type cachesize iosize nbackends
- DESCRIPTION: rsize prints an estimate of the amount of memory (in bytes) a running raid set will use, given the raid set type, the amount of cache allocated to it, the raid set iosize and the number of backends in the raid set.
- SEE ALSO: rconf

13.80 SCN2681 − access a scn2681 (serial IO device) as console

- SYNOPSIS: bind −k {scn2681 [value]} bind_point
- DESCRIPTION: scn2681 allows the target hardware using the SCN2681 serial IO chip to be accessed via the bind_point in the K9 namespace. If no optional argument is given channel A in the chip is used. If value is 0 then channel A (serial port) is used. If value is 1 then channel B (serial port) is used. If value is 2 then the input port (parallel) is used. If value is 3 then the output port (parallel) is used. The special file scn2681 is only available when the target is special hardware (containing a SCN2681).
- SEE ALSO: cons

13.81 SCSICHIPS − print various details about a controller's scsi chips

- SYNOPSIS: scsichips
- DESCRIPTION: scsichips prints details about the RaidRunner scsi chips.
- OPTIONS: Three sets of numbers are printed. The first two numbers are the number of hostports (chips) and the number of backend channels (chips). The rest of the numbers printed are the hostport then backend channel chip indexes. The chip addresses can be used in binding them to a entrypoint in the kernel.

* EXAMPLES: For a RaidRunner controller with one hostport and six backend channels scsichips will print
```
1 6 6 0 1 2 3 4 5
```

which says that there is 1 hostport chip, 6 backend channels, the address of the hostport chip is 6 and the addresses of the backend channel chips are 0, 1, 2, 3, 4 and 5.

For a RaidRunner controller with two hostports and six backend channels scsichips will print

```
2 6 6 7 0 1 2 3 4 5
```

which says that there are 2 hostport chips, 6 backend channels, the addresses of the hostport chips are 6 and 7 and the addresses of the backend channel chips are 0, 1, 2, 3, 4 and 5.

* SEE ALSO: hwconf, scsihpfs, scsihdfs

13.82 SCSIHD − SCSI hard disk device (a SCSI initiator)

* SYNOPSIS
```
      bind −k {scsihd chipNumber scsiTargetId
[scsiTargetLUN]} bind_point

cat bind_point

ctl

raw

partition

data

rconfig
```

* DESCRIPTION: scsihd is a SCSI hard disk device. In SCSI jargon it is called the "initiator" while the disk it is talking to is referred to as the "target". The disk is known in SCSI as a "direct access device". Currently this device adopts the SCSI id 7 and attempts to access a disk with a SCSI id of scsiTargetId. The "logical unit number" (LUN) within the scsiTargetId that is accessed is scsiTargetLUN (or LUN 0 if not given). Multiple SCSI buses are supported (with typically one device per bus) and are distinguished by the chipNumber.

The SCSI disk device is bound into the namespace at bind_point. A one level directory is made at the bind_point containing the files: "raw", "data", "partition", "rconfig" and "ctl". The "partition" file is a small portion at the end of the disk used for storing partition information (in plain ASCII). The "ctl" and "rconfig" files are for internal RaidRunner use. (The "rconfig" file will be a backup copy of the RaidRunner configuration area). The "data" file is the usable part (i.e. the vast majority) of the space available on the SCSI disk connected to the SCSI bus on which this device is an initiator. The file "raw" addresses the whole disk less the partition block.

The current raid hardware has six fast wide SCSI−2 buses numbered 0 to 5 and two fast wide buses num− bered 6 and 7. Usually the first six fast wide SCSI−2 buses are set up as initiators (i.e. bind scsihd) while the two fast wide SCSI buses are set up as targets (i.e. bind scsihp).

Current implementations make the device name hd synonymous with scsihd. Therefore:

```
bind −k {hd chipNumber scsiTargetId
```

[scsiTargetLUN]} bind_point has the same effect as the similar line in the synopsis.

- SEE ALSO: scsihp

13.83 SCSIHP − SCSI target device

- SYNOPSIS

```
bind −k {scsihp chipNumber scsiid} bind_point


cat bind_point

scsiid         # this device's scsi id

cat bind_point/scsiid

0

...      # this device's LUN within SCSI id scsiid

7

cat bind_point/scsiid/0

data      # scsi data channel for SCSI id scsiid LUN 0

cmnd      # scsi command channel for SCSI id scsiid LUN 0
```

- DESCRIPTION: scsihp is a SCSI target device. A "target" in SCSI jargon performs commands issued by an "initiator" which is usually a general purpose computer. Normally a SCSI target is an integral part of a "direct access device" (i.e. a disk) but in a raid, the controller needs to look like a target to external users. Multiple SCSI buses are supported (with typically one device per bus) and are distinguished by the chipNumber.

The SCSI target device is bound into the namespace at bind_point. A three level directory is made at the bind_point. The first level is the SCSI id (as specified by scsiid) for this device. The second level is the LUN within that scsi id. The third level contains the files: "data" and "cmnd" which are used for SCSI data phases and command/status phases respectively.

A scsi target command interpreter called stargd is designed to "mate" with this device.

The current raid hardware has six fast narrow SCSI buses numbered 0 to 5 and two fast wide buses num− bered 6 and 7. Usually the six fast narrow SCSI buses are set up as initiators (i.e. bind scsihd) while the two fast wide SCSI buses are set up as targets (i.e. bind scsihp).

- SEE ALSO: scsihd, stargd

13.84 SET − set (or clear) an environment variable

- SYNOPSIS
    - set name
    - set name value ...
- DESCRIPTION: set allows the value (or list of values) associated with an environment variable to be changed. When no value is given then the environment variable name has its value set to NIL. When one value is given then the environment variable name is set to that value. When multiple values are given then the environment variable name is set to that list of values.
- NOTE: set is a "built−in" in the Husky shell. This means a new shell is _not_ thrown to execute this command unless some other action forces it. Setting an environment variable in a sub−shell has no effect once the sub shell exits to its parent shell. This is why throwing a sub−shell to execute set would be pointless. Operations such as command substitution (i.e. `set ...') and command line file redirection force a command to be executed in a sub−shell and hence defeat the purpose of set. Hence the set command (or at least "built−in") should appear in a simple expression to avoid surprises.

13.85 SCSIHPMTR − turn on host port debugging

- SYNOPSIS:
    - scsihpmtr −c portnumber storage_size
    - scsihpmtr −b
    - scsihpmtr −d
    - scsihpmtr −e
    - scsihpmtr −p count
    - scsihpmtr −r
    - scsihpmtr −w
- DESCRIPTION: scsihpmtr, if enabled in the RaidRunner kernel, will create a volatile internal store and save details of all scsi commands which come in on a specified host port or set of host ports. This data can then be uploaded to the host via the scsi monitor (ssmon) and analyzed as appropriate.
- OPTIONS:
    - −c portnumber storage_size: Specify and allocate a volatile storage area of storage_size bytes and monitor and save all scsi commands that appear on the given portnumber. The portnumber is the chip number of the host port on the raid controller. If you specify a port number of 8, then all host ports will be monitored.
    - −b: Perform natural read analysis for use in possible reconfiguration of RaidRunner.
    - −d: Delete any volatile storage containing monitored data and hence stop monitoring.
    - −e: Temporarily disable the monitoring (and hence storage) of scsi commands. Monitoring must be disabled BEFORE uploading the monitored data to the host via ssmon.
    - −p count: Print out both details of what is being monitored and the first count stored scsi commands. The details printed are the total number of commands that can be recorded, the current number of recorded commands, the current monitoring state (0 disabled, 1 enabled) and the monitored host port number. Each line of scsi commands printed will time the command occurred a hex word containing lower 3 bits the chip number (hostport) the command was on next 3 bits the scsi id of the command initiator (i.e host) next byte is the scsi message id of the command next two bytes is the scsi tag number of the command a 12 byte scsi command header block
    - −r: Reset the monitoring of scsi commands. This will cause all previously monitored data to be lost.
    - −s: Re−enable the monitoring (and hence storage) of scsi commands.
    - −w: Enable wrap around mode. By default, scsihpmtr will stop recording scsi commands

once it has used all the storage area allocated. If wrap around mode is enable, then scsi commands will continue to be recorded, wrapping round the storage area.
- SEE ALSO: smon, ssmon, SCSI Standards for formats of scsi commands.

13.86 SETENV − set a GLOBAL environment variable

- SYNOPSIS: setenv name value ...
- DESCRIPTION: setenv allows the value (or list of values) associated with a GLOBAL environment variable to be changed (or created). When one value is given then the GLOBAL environment variable name is set to that value. When multiple values are given then the GLOBAL environment variable name is set to that list of values.
- NOTE: A GLOBAL environment variable is one which is stored in a non volatile area on the RaidRunner and hence is available between successive power cycles or reboots. These variables ARE NOT the same as husky environment variables. The non volatile area is co−located with the RaidRunner Configuration area.
- SEE ALSO: printenv, unsetenv, rconf

13.87 SDLIST − Set or display an internal list of attached disk drives

- SYNOPSIS
    - sdlist
    - sdlist −p
    - sdlist backend_triplett_list
- DESCRIPTION: sdlist maintains a list of disk backends, in rank, chip, scsi lun form for a RaidRunner. This list is used by various commands to save those commands from continuously probing for all possible backends. When run without arguments, sdlist zero's the internal list and probes for all disk backends rebuilding the list. Typically, sdlist will be run with a list provided to it by way of it's arguments. For example, on a RaidRunner with one rank (at scsi id 1) and six disks with all luns at zero the following would be run.

```
sdlist `hwconf −D'
```

which would equate to

```
sdlist 0.1.0 1.1.0 2.1.0 3.1.0 4.1.0 5.1.0
```

and those six tripletts would be stored in the list, with sdlist having already deleted any already stored list. To see what this internal list contains, the −p option can be given and the list will be printed. This command is typically executed during autoboot and would not be executed interactively unless the user is performing unusual backend manipulations and is debugging the process.

- SEE ALSO: rconf

13.88 SETIV − set an internal RaidRunner variable

- SYNOPSIS:
    - setiv
    - setiv name value
- DESCRIPTION: setiv allows the value of an internal RaidRunner variable to be changed or lists all

changeable internal variables (no argument). When a value is given then the internal RaidRunner variable name is set to that value. If no value is given the all settable internal variables are listed.

- NOTES: As different models of RaidRunners have different settable internal variables see your RaidRunner's Hardware Reference manual for a list of variables together with the effect of changing them. These variables are run−time variables and hence must be set each time the RaidRunner is booted.
- SEE ALSO: getiv

13.89 SHOWBAT − display information about battery backed−up ram

- SYNOPSIS: showbat [−a] [−n] [−s] [−t]
- DESCRIPTION: showbat will print (to standard output) information about the installed battery backed−up ram.
- OPTIONS :
    - −a: Print address information about battery backed−up ram. The addresses printed are the base address, start of tag address, start of reserved area address, start of cache data address and the first unused byte address.
    - This is the default if no options are given.
    - −n: Print the number of tags in battery backed−up ram.
    - −s: Print the status of the battery backed−up ram.
    - −t: For each tag in battery backed−up ram, print it's value and the location of it's data along with details of the cache range it belongs to. If the cache filesystem has not been created and the ranges added then this option will only print out the tag addresses and inform you that no ranges are available. If the data associated with the tag has partially written data in it, the tag will be flagged in the list.
- SEE ALSO: cachedump, cacherestore, cacheflush

13.90 SHUTDOWN − script to place the RaidRunner into a shutdown or quiescent state

- SYNOPSIS: shutdown
- DESCRIPTION: shutdown is a husky script which is used to place the RaidRunner into the "shutdown" or quiescent state. The "shutdown state" is one in which the raid sets configured cannot be accessed by a host and all devices (disks) attached to the RaidRunner are in a low power usage state (i.e spun down). shutdown first set's all stargd's to a spin state of 2 which means that all host scsi medium−access commands will result in a CHECK CONDITION status and set the mode sense key to NOT READY and the additional mode sense code and code qualifier to "LOGICAL UNIT NOT READY, MANUAL INTERVENTION REQUIRED". It will then flush the cache and finally spin down all backend devices attached. Note that the scsi monitors − smon, are not effected.
- SEE ALSO: stargd, mstargd, cache, spind

13.91 SLEEP − sleep for the given number of seconds

- SYNOPSIS: sleep seconds
- DESCRIPTION: sleep causes the current K9 process to "sleep" for the given number of seconds. The argument seconds may be any integer or a fixed point number (e.g. 1.5). The resolution of timing is one clock tick (which varies according to the target hardware). If the sleep is interrupted before it

completes then a "sleep interrupted" status message is returned otherwise NIL (i.e. true) is returned.
- SEE ALSO: K9sleep

13.92 SMON − RaidRunner SCSI monitor daemon

- SYNOPSIS: smon [−b blkprotocol] [−d debug_level] −m moniker −h hport [−n] [−r] [−1 stdout_fifo_file] [−2 stderr_fifo_file] [−3 status_file] −s capacity device id lun
- DESCRIPTION: smon is a daemon process that simulates a SCSI−2 disk with modified Read and Write SCSI commands that implement a simple protocol which provides file transfer to and from the RaidRunner and remote command execution on the RaidRunner. The protocol is based on the SCSI Read and SCSI Write starting block addresses. smon usually works closely with a special device (file system) that does the hardware interfacing and message interpretation for a SCSI−2 target (see scsihp). smon has 3 mandatory switches, 8 optional switches and 3 mandatory positional arguments. The mandatory switches are the −s switch which specifies the size of the SCSI−2 monitor store, the −m switch which specifies the moniker (name) of the target and the −h switch which specifies the controller host port the monitor will communicate over. The 3 mandatory positional arguments are device, id and lun. These arguments are concatenated together to make the filename of the low level SCSI target device driver. The second argument is the SCSI identifier number this target will respond to while the third argument is the logical unit number (LUN) within that SCSI identifier which the target represents. An error will occur during daemon initialization unless the following files (with appropriate characteristics) are found:
    - device/id/lun/cmnd
    - device/id/lun/data
- OPTIONS The optional switches may appear in any order but if present must be before the positional arguments.
- −b blkprotocol: Change the block numbers used in the protocol (see below). By default the block numbers used in the protocol are 7, 8, 9 10, 11, 12, 13, 14 and 15 for protocol elements BLK_STDOUT, BLK_STDERR, BLK_STATUS, BLK_ENDPROC, BLK_UPLOAD, BLK_STDIN, BLK_MONITOR BLK_RESET and BLK_RCONFIG respectively. To change this default list of block numbers, specify a comma separated list of nine(9) unique block numbers.
- −d debuglevel: This switch is a debug flag. When given the command line arguments and other information derived during initialization are echoed back to standard out and the debug level is set to 1. The debug levels are:
    - 0 no debug messages (default when "−d" not given)
    - 1 debug messages on all non−read/write commands
    - 2 debug messages on all commands
- Debug messages are typically a single line per command and are sent to standard out. Serious errors are reported to standard error irrespective of the current debug level. If the error is related to incoming data (from the SCSI initiator) then the daemon will continue. If the error cannot be recovered from, then the daemon terminates with an error message as its exit status. The debug level can be changed during the execution of a smon daemon by using the "−d" option on mstargd.
- −m moniker: This switch associates the moniker (i.e name) with the scsi monitor that smon is about to execute on. It provides a method of naming scsi monitors. The moniker can be a maximum of 32 characters. This switch is mandatory.
- −h hport: This switch advises smon which controller host port number it will be communicating over. This switch is mandatory.
- −n: This switch stops statistics being gathered. The default action is to collect statistics.
- −r: This switch is for restarting the daemon with as little disturbance to the initiator as possible. Usually the SCSI Unit attention condition is set on reset (and when this daemon commences). When this switch is given the daemon starts in the idle state. Use with care. The default action (i.e. when the

"−r" is not given) is to set a SCSI Unit attention condition so that the first SCSI command received will have its status returned as "Check condition" (as required by the SCSI−2 standard).

- −1 stdout_fifo_file: Specify the name of the fifo data file where the standard output of any executed commands is to be sent (and read from).
- −2 stderr_fifo_file: Specify the name of the fifo data file where the standard error output of any executed commands is to be sent (and read from).
- −3 status_file: Specify the name of the fifo data file where the exit status of any executed command is to be written to (and read from).
- −s capacity: This mandatory switch informs the daemon what the size in blocks this SCSI target will represent. The value for capacity effects the response this target makes to the SCSI Read Capacity and Mode Sense (page 4) commands. To simplify writing out large numbers certain suffixes can be used, see suffix.
- PROTOCOL: smon appears as a disk with the given capacity, scsi id and scsi lun to a host computer. By reading and writing to specific block locations on this disk, the host computer can send and have executed husky commands on the RaidRunner and read back the command's standard output, error and status. Additionally files can be both downloaded from the host onto the RaidRunner and uploaded from the RaidRunner onto the local host computer. All reads and writes by the host need to be in multiples of 512 byte blocks with null padding as appropriate. When WRITEing (from the host) to smon, the data, depending on the block number, contained in the write will be treated as follows −
- BLK_ENDPROC: Will kill any processes, started by smon, currently running on the RaidRunner. The contents of the write's data will be read but ignored.
- BLK_RESET: Will terminate any file transfers, kill any process(s) (started by smon) currently running on the RaidRunner, close any files opened by smon and reset smon to it's initial state. The contents of the write's data will be read but ignored.
- BLK_STDIN: If smon is in DOWNLOAD file transfer mode, the contents of the write's data will be written out to the download file on the RaidRunner. If not in DOWNLOAD file transfer mode, will asynchronously execute the contents of the write as a husky command. Before execution of the command, the command is checked to see if it is an internal file transfer smon command, and if so, will initialize for file transfer.
- Any other: Will perform the write to the internal store. Up to 16 blocks of data will be stored in the raid configuration area.
- When READing (by the host) from smon, the data, depending on the block number, contained in the read will be treated as follows −
- BLK_STDOUT: This read will return the currently available standard output from the previously executed command. If the number of bytes of stan- dard output is less than the size of the read's data the data will be padded with NULL's ('\0'). If the number of bytes of standard output is more than the size of the read, then subsequent reads from BLK_STDOUT will transfer the rest of the standard output. Normally one would loop reads from BLK_STDOUT until the last character in the read buffer is NULL. If the previous command was a write to start a download, then a read of one block will next be expected and will return either a block of NULL's or an error message indicating a problem with the initialization of the download.
- BLK_STDERR: This read will return the currently available standard error out- put from the previously executed command in the same manner as BLK_STDOUT.
- BLK_ENDPROC: If the previously executed command (write to block BLK_STDIN) has not completed this read will return with it's buffer starting with "Process not finished" and NULL padded. If the command has completed then it will return a NULL filled buffer.
- BLK_STATUS: A read from this block will return the command's husky exit status (the $status variable).
- BLK_UPLOAD: Will first return either the UPLOAD handshake command (response to a UPLOAD write command) which is of the form UPLOAD nbytes where nbytes is the number of bytes in the file that has just been requested to be uploaded from the RaidRunner or an error message, then all

reads from this block onwards will contain the next buffer from the uploaded file.

- BLK_MONITOR: Will first return either the MONITOR handshake command (response to a MONITOR write command) which is of the form MONITORSIZ nbytes where nbytes is the number of bytes of monitor data that will need to be transferred up to the host OR an error message if there is no monitoring data. All subsequent reads will upload the monitoring data.
- BLK_RCONFIG: Successively reading in data from this block will return successive data from the in−core raid configuration area. When a BLK_RESET is written, the internal smon index into the in−core raid configuration area is reset (set to 0). When a number of blocks from BLK_RCONFIG are this internal index is incremented by the amount of data read. If more data is read than is available in the in−core raid configuration area, then the index is reset to 0.
- Any other: Will perform the read from the internal store.
- Internal Store: Typically, a host computer stores labels, boot and disk partition information on all disks. On most host computers, they usually only store this information in a few blocks either at the start of the disk and/or near the end of the disk. Rather than smon reserving this space as a continuous piece of memory, most of which would not be used, an internal store of 16 512−byte blocks is created and maintains a mapping of host block addresses to the store. This store is copied to the raid configuration area noting the controller number and host port number smon is running on. If the store fills as a host computer writes many blocks to the smon disk, the additional data is discarded. If a host computer attempts to read blocks that have not been stored, then null fill data is returned.
- SIGNALS: Two K9 signals are interpreted specially by smon.
- If the signal K9_SIGINT is received then smon will check to see whether there is a command in progress and if so terminate it with a "Check condition" status and set the associated sense key to "Command aborted". smon will then return to its command (waiting) mode.
- If the signal K9_SIGTERM is received then smon will do the same as it does for K9_SIGINT except it will exit the process rather than going to command (waiting) mode.

The SCSI initiator should interpret a sense key of "Command aborted" as the target unilaterally aborting a command in progress. The SCSI−2 standard suggests the initiator should retry the aborted command.

- EXAMPLE

```
smon −s 2080 −h 1 −m SMON167 /dev/hostbus/1 6 7
```

This line will invoke this daemon and try and open the following files:

```
/dev/hostbus/1/6/7/cmnd
```

and

```
/dev/hostbus/1/6/7/data
```

The "−s 2080" switch instructs this daemon to tell SCSI initiators that it is a 1040K disk.

- SEE ALSO: scsihp, suffix, mstargd, stargd, mconf

13.93 SOS − pulse the buzzer to emit sos's

- SYNOPSIS: sos [count]
- DESCRIPTION: sos will sound the buzzer in a plaintif "SOS" in Morse. If a count is given, it will repeat count times. The default count is 3.
- SEE ALSO: buzzer, warble

13.94 SPEEDTST − Generate a set number of sequential writes then reads

- SYNOPSIS: speedtst −d device −n io_cnt −X bufsize [−o offset] [−STRW]
- DESCRIPTION: By default speedtst will perform io_cnt sequential writes of size bufsize bytes onto the device from the start of the device. It will then do io_cnt sequential reads of size bufsize bytes from the start of the device device. By default reads and writes are not checked for success. The output of this command provides the I/O transfer rates in Megabytes (MB) and million bytes (mb) for both the write and read sequences. The bufsize and offset values may have a suffix.
- OPTIONS:
    - −o offset: All reads and writes are to be performed offset bytes into the device. All reported values will be relative to this offset. The default is 0 i.e the start of the device.
    - −r drives: The bufsize (set by −X bufsize) is expected to be to iosize of a Raid 5 raid set where drives is the number of data and parity drive. A single bufsize operation is performed to a drive in each strip of the raidset in such a fashion that either the data drive and the parity drive (the only two drives written too in a write operation) will not be the same drives as would have been written too, if the previous operation on the previous stripe had been a write. The default value is 0, causing normal sequential operations. This option is used to simulate random operations on a raidset without the penalty of significant seek overheads.
    - −R: By default speedtst will perform a write test then a read test. Specifying this option, the write test will NOT be performed.
    - −S: Normally the device, device, is opened in a default manner which may allow the host operating system to provide buffers when writing data to it. This may result in incorrect I/O through put rates. By setting this option, the device is opened in such a way to ensure that writes do not use any buffering and the individual write system calls do not return until the data is on the device.
    - −T: Normally the read and write operations are NOT checked for success, that is the return values for the read and write system calls are not checked for errors. To ensure each read and write is checked for any system error, use this flag.
    - −W: By default speedtst will perform a write test then a read test. Specifying this option, the read test will NOT be performed.
- SEE ALSO: randio, suffix

13.95 SPIND − Spin up or down a disk device

- SYNOPSIS:
    - spind up c.s.l
    - spind down c.s.l
- DESCRIPTION: spind will either spin down or spin up a nominated disk drive via the SCSI−2 START command.
- OPTIONS:
    - up: Spin up the disk drive. If the drive is already spun up nothing will occur.
    - down: Spin down the disk drive. If the drive is already spun down nothing will occur.
    - c.s.l: Identify the disk device by specifying it's channel, SCSI ID (rank) and SCSI LUN provided in the format "c.s.l"
- SEE ALSO: Product manual for disk drives used in your RaidRunner.

13.96 SPINDLE − Modify Spindle Synchronization on a disk device

- SYNOPSIS
    - spindle −c −p c.s.l [−o rpl_offset]
    - spindle −m|M −p c.s.l [−o rpl_offset]
    - spindle −s −p c.s.l [−o rpl_offset]
    - spindle −g −p c.s.l
- DESCRIPTION: spindle will either modify or report on a disk device's spindle synchronization. This command will set or clear the Rotational Position Locking (RPL) bits in the disk device's GeometryParameter's mode sense page. If spindle correctly modifies the device's RPL bits, it will print out the resultant RPL bits and the Rotational Offset byte of the device's Geometry Parameter mode sense page.
- OPTIONS:
    - −p c.s.l: Identify the disk device to modify by specifying it's channel, SCSI ID (rank) and SCSI LUN provided in the format "c.s.l"
    - −c: Clear the RPL bits.
    - −s: Set the RPL bits to 01b (0x1) which typically sets the device to operate as a synchronized−spindle slave.
    - −m: Set the RPL bits to 10b (0x2) which typically sets the device to operate as a synchronized−spindle master.
    - −M: Set the RPL bits to 11b (0x3) which typically sets the device to operate as a synchronized−spindle master control.
    - −g: Get and print the current RPL bits and the Rotational Offset byte from the device's Geometry Parameters Page.
    - −o rpl_offset: When clearing (−c) or setting (−m, −M, −s) spindle synchronization additionally set the rotational off− set to the given value. Must be in range from 0 to 256. This value is the numerator of a fractional multiplier that has 256 as it's denominator (eg a value of 128 indicates a one−half revolution skew). A value of zero (0) indicates that rotational offset shall not be used.
- SEE ALSO: The mode sense geometry page references in the relevant product manual for the disks used in the RaidRunner.

13.97 SRANKS − set the accessible backend ranks for a controller

- SYNOPSIS: sranks controller_id:ranklist [controller_id:ranklist]
- DESCRIPTION: sranks allows you to set backend rank restrictions, as specified by the arguments, for controller on which the command is set. Each argument is of the form

```
controller_id  the id of the controller (0, 1, ...).
```

ranklist a comma separated list of rank id's (scsi id's) for which the given controller is to have access. sranks will check each argument looking for the controller id corresponding to the controller it's running on and set the backend rank access as per the ranklist. Typically, this command is executed with the output of the BackendRanks GLOBAL environment variable at boot time.j

- SEE ALSO: environ, pranks

13.98 STARGD − daemon for SCSI−2 target

- SYNOPSIS: stargd [−c] [−d] −m moniker −h hport −s capacity [−n] [−r] [−L cnt] [−P nprocs] [−R] [−S sectorsize] [−nr nrread:nrlen] [−C] [−stripesize stripesize] [−irgap nblks] [−stripe stripe_args] device id lun store [store2]
- DESCRIPTION: stargd is a daemon process that interprets SCSI−2 commands as a "target". [In simple SCSI configurations the host computer is a SCSI "initiator" while its disk is a SCSI "target".] stargd usually works closely with a special device (file system) that does the hardware interfacing and message interpretation for a SCSI−2 target (see scsihp). When stargd first starts, it's "spin state", is set to indicate that thestore file has yet to "spin−up" and hence is NOT READY. This means that until the spin state is changed to be marked as READY (via mstargd "−o" option) all supported SCSI−2 medium−access commands will result in a CHECK CONDITION with the sense key set to "NOT READY" and additional sense information set to "LOGICAL UNIT IN PROCESS OF BECOMMING READY" When the spin state is marked as READY all supported SCSI−2 medium−access commands are processed correctly. Additional NOT READY states can be set by mstargd's −Z option.
- OPTIONS: stargd has 3 mandatory switches, 8 optional switches and 4 mandatory positional arguments and an optional trailing positional argument. The mandatory switches are the −m switch which associates the moniker (i.e name) with the raid set that stargd is about to execute on (this provides a method of naming raid sets), the −c switch which specifies the capacity of the backend and the −h switch which advises stargd which controller host port it will be communicating over. The 4 mandatory positional arguments are device, id, lun and store. The first 3 are concatenated together to make the filename of the low level SCSI target device driver.
    - The second argument is the SCSI identifier number this target will respond to while the third argument is the logical unit number (LUN) within that SCSI identifier which the target represents. An error will occur during daemon initialization unless the following files (with appropriate characteristics) are found:

```
        device/id/lun/cmnd
device/id/lun/data
```

- The fourth positional argument is store. It will typically be a cache device although it could be a single disk or a raid level. The trailing optional positional argument is store2. If the filename store2 can be opened and the "−c" argument is _not_ given then writes to store are echoed to store2. The SCSI commands that cause writes at this level are Write_6 and Write_10. If store2 cannot be opened (for writing) or the "−c" argument is given then a warning message is output during stargd initialization and it continues as if store2 had not been given.
- All switches may appear in any order but if present must be before the positional arguments.
- The "−L cnt" switch indicates that stargd is to implement a lookahead process for SCSI−2 reads (Read_6 and Read_10) which pre−fetches data into the cache based on the last cnt SCSI−2 reads. The default is a readahead of 16 reads (i.e −L 16). The minimum readahead is 2 and the maximum is 63. You can specify readahead to be 0 which turns off all lookahead. This switch can only be used when cache (−c) is being used. The "−nr nread:nrlen" is used to specify 'natural read' information to stargd. Some applications may perform reads of a certain size which is greater than the largest read the host operating system will allow. In this case, the host operating system will break the application's read up into smaller reads. stargd can trigger on a certain discontinuous read (by specifying a size in blocks − nread) and will prefetch prefetch the rest of the application read (nrlen). For example, if an application performs a read of 304 blocks on a host operating system which has a maximum read size of 128 blocks, then you could set the −nr arguments nread:nrlen to 128:304 which will cause stargd to, when it sees a discontinuous read of size 128 blocks, service the read and also prefectch into cache, the next 304 − 128 = 176 blocks of data. See scsihpmtr for analysis of host operating system reads.
- The "−stripesize stripesize" switch informs stargd the stripe size of the raid set which stargd is fronting. If the additional −C switch toggles whether this value (stripesize) will be used to calculate a cylinder size (heads x

sectors per track) that ensures that a cylinder is a multiple of the given stripe size. If the stripe size is such that there is no multiple of heads and sectors that, when multiplied together, is not a multiple of the stripe size, then the default head and sectors per track sizes are used − currently 16 head, 128 sectors per track.

- The "−stripe stripe_args" switch informs stargd that a number of host ports will be concurrently accessing this raid set. This switch is only usefull when the host based disk access software can "stripe" io to N different devices and each access (read or write) is always less than a set stripesize.
- The stripe_args are of the form N:S:I=c.h.l,I=c.h.l, where: N is the number of stripes (or host ports) that will be concurrently accessing the raid set, S is the stripe size (in 512−byte blocks) that the host will perform i/o in, I is an index or "stripe number" specifier ranging from 0 to N. c.h.l is the controller, host port, lun triplet, associated with the given index. For example, if we have two host ports and the given stripe−size from the host is, say, 1368 blocks, we would then have the following raid: set additions (in agui form) Host Interfaces (2): M 0.0.0 M 0.1.0 Additional stargd args: −stripe 2:1368:0=0.0.0,1=0.1.0 NOTE: that it has to be guaranteed that the host stripe software will send block 0 (size 1368 or less) to controller, host port, lun triplet 0.0.0 and block 1 (size 1368 or less) controller, host port, lun triplet 0.1.0, block 2 (size 1368 or less) to controller, host port, lun triplet 0.0.0, and so forth.
- The "−C" switch is used to toggle whether stargd calculates a cylinder size such that it is a multiple of the given stripe size. The default is to calculate a cylinder size based on the given stripe size.
- The "−irgap nblks" switch, specifies the inter−read gap, nblks, (in blocks). When sequential reads arrive from a host there may be a small gap between successive reads. Normally the lookahead algorithm will ignore these gaps providing they are no larger than the average length of the group of sequential reads that have occurred. By specifying this value, you can increase this gap. This switch has no meaning if the lookahead feature is turned off (i.e specifying "−L 0").
- The "−R" switch indicates that store should have read−only access. If this switch is NOT present, then the store is assumed to have read−write access.
- The "−c" switch indicates that store is a cache (rather than a disk or something else remote). This knowledge decreases the number of internal copies this daemon needs to do so it is a performance enhancement. The default (i.e. when this switch is not present) is to do the extra copy which is the safe course if the exact identity of the store is uncertain.
- The "−d" switch is a debug flag. When given the command line arguments and other information derived during initialization are echoed back to standard out and the debug level is set to 1. The debug levels are:
    - 0 no debug messages (default when "−d" not given)
    - 1 debug messages on all non−read/write commands
    - 2 debug messages on all commands

  Debug messages are typically a single line per command and are sent to standard out. Serious errors are reported to standard error irrespective of the current debug level. If the error is related to incoming data (from the SCSI initiator) then the daemon will continue. If the error cannot be recovered from, then the daemon terminates with an error message as its exit status. The debug level can be changed during the execution of a stargd daemon by using the "−d" option on mstargd.

- The "−m" switch associates the moniker (i.e name) with the raid set that stargd is about to execute on. It provides a method of naming raid sets. The moniker can be a maximum of 32 characters. This switch is mandatory.

- The "−h" switch informs stargd which controller host port it will be communicating over. This switch is mandatory.
- The "−n" switch toggles statistics being gathered. The default action is to collect statistics.
- The "−r" switch is for restarting the daemon with as little disturbance to the initiator as possible. Usually the SCSI Unit attention condition is set on reset (and when this daemon commences). When this switch is given the daemon starts in the idle state. Use with care. The default action (i.e. when the "−r" is not given) is to set a SCSI Unit attention condition so that the first SCSI command received will have its status returned as "Check

condition" (as required by the SCSI−2 standard).

- The "−s capacity" switch informs the daemon what the size in blocks this SCSI target will represent. The value for capacity effects the response this target makes to the SCSI Read Capacity and Mode Sense (page 4) commands. This switch is mandatory. To simplify writing out large numbers certain suffixes can be used, see suffix.
- The "−P nprocs" switch causes the daemon to spawn off nprocs copies of it self to allow concurrent processing of SCSI commands. If not specified, the default number of stargd processes created is 4. This value can range from 1 to 8 inclusive. The additional processes will not be created until the first access from the host port. The "−S sectorsize switch informs the daemon what sector size, in bytes, this SCSI target will present. The value defaults to 512 − the typical disk block size. To simplify writing out large numbers certain suffixes can be used, see suffix.
- SIGNALS: Two K9 signals are interpreted specially by stargd.
    - If the signal K9_SIGINT is received then stargd will check to see whether there is a command in progress and if so terminate it with a "Check condition" status and set the associated sense key to "Command aborted". stargd will then return to its command (waiting) mode.
    - If the signal K9_SIGTERM is received then stargd will do the same as it does for K9_SIGINT except it will exit the process rather than going to command (waiting) mode.
    - The SCSI initiator should interpret a sense key of "Command aborted" as the target unilaterally aborting a command in progress. The SCSI−2 standard suggests the initiator should retry the aborted command.
- EXAMPLE:

```
stargd −c −s 2M −m RS −h 1 /dev/hostbus/1 6 0 /cache/data
```

  This line will invoke this daemon and try and open the following files:

```
/dev/hostbus/1/6/0/cmnd
```

  and

```
/dev/hostbus/1/6/0/data
```

  The file "/cache/data" will be used as store. The "−c" switch identifies this file id as a cache to stargd. The "−s 2M" switch instructs this daemon to tell SCSI initiators that it is a 1 GigaByte disk (i.e. 2 MegaBlocks).

- SUPPORTED SCSI−2 COMMANDS: The table below lists the supported SCSI−2 commands (Code and Name) and their action under stargd.

```
00: Test Unit Ready  If backend is ready returns GOOD Status, else sets
 Sense Key to Not Ready and returns CHECK CONDITION Status

01: Rezero Unit      Does nothing, returns GOOD Status

03: Request Sense    Sense data held on a per initiator basis (plus extra
 for bad LUN's)

04: Format Unit      Does nothing, returns GOOD Status

07: Reassign Blocks  Consumes data but does nothing, returns GOOD Status

08: Read_6           DPO, FUA and RelAdr not supported

0a: Write_6          DPO, FUA and RelAdr not supported

0b: Seek_6           Does nothing, returns GOOD Status
```

```
12: Inquiry          Only standard 36 byte data format supported (not vital
 product data pages)

15: Mode Select      Support pages 1, 2, 3, 4, 8 and 10 (but none writable)

16: Reserve          Doesn't support extents + 3rd parties

17: Release          Doesn't support extents + 3rd parties

1a: Mode Sense       Support pages 1, 2, 3, 4, 8 and 10.

1b  Start Stop       If Start is requested and the Immediate bit is 0 then
 waits for backend to become ready, else does nothing and returns GOOD Status.
 If backend does not become ready within 20 seconds set Sense Key to Not Ready
 and returns CHECK  CONDITION Status

1d  Send Diagnostics Returns GOOD Status when self test else complains
 (does nothing internally)

25  Read Capacity    RelAdr, PMI and logical address > 0 are not supported

28  Read_10          Same as Read_6

2a  Write_10         Same as Write_6

2b  Seek_10          Does nothing, returns GOOD Status

2f  Verify           Does nothing, returns GOOD Status

55  Mode Select_10   Same as Mode Select

5a  Mode Sense_10    Same as Mode Sense
```

- SEE ALSO: scsihp, suffix, mstargd

13.99 STAT − get status information on the named files (or stdin)

- SYNOPSIS: stat [−b] [file...]
- DESCRIPTION: stat gets status information on each given file, or the standard input when a file named `−' is given, and sends it to the standard output. For files that are found a line with 6 columns is output. The meaning of each column is summarized below:
    - 1st filename
    - 2nd type of file system containing this file
    - 3rd instance of the containing file system
    - 4th unique (internal) file identifier
    - 5th version number of this file
    - 6th length of this file (in bytes)
- If the −b option is given, then the length of each file (6th field) is printed in 512−byte blocks. Only whole blocks are reported, so files with lengths less than 512−byte blocks will report is having a block size of zero (0).
- EXAMPLE

```
: raid; stat /bin/ps
```

```
ps                              ram    0 0x00049049    2 144

: raid;
```

* SEE ALSO: K9getstat, intro

13.100 STATS − Print cumulative performance statistics on a Raid Set or Cache Range

* SYNOPSIS: stats [−c cache_moniker] [−r raid_set] [−g] [−z]
* DESCRIPTION: stats is a process that will print and or zero the cumulative performance statistics which Raid Set's and Cache Ranges maintain.
* OPTIONS:
    * −c cache_moniker: Zero (−z) and or print (−g) the cache statistics of the cache range specified by cache_moniker which was set in the add moniker=cache_moniker first= ... command (see cache). The statistics printed are the number of cache hits, cache misses, cache probes per hit and probes per miss. Output is in the form
      `moniker : hits+misses hits misses hit+miss_probes hit_probes miss_probes`
* r raidset_name: Zero (−z) and or print (−g) the raid set statistics of the raid set specified by raidset_name. When printing (−g) statistics, for each backend in the raid set, the cumulative number of reads, writes, raid failures and write failures to that backend are printed. Output is in the form
  ```
  D0 r0_cnt r0_fails w0_cnt w0_fails; D1 r1_cnt r1_fails w1_cnt w1_fails;
   D...;
  ```

* GENERAL: When gathering (or zeroing) a Cache Range's statistics, a special system call to the RaidRunners cache is made. When gathering (or zeroing) a Raid Set's statistics, a read or write (of the control string "zerostats" is made to the raid_bind_point/stats control file.
* SEE ALSO: cache, raid0, raid1, raid3, raid5

13.101 STRING − perform a string operation on a given value

* SYNOPSIS: string option value
* DESCRIPTION: string provides various string type operations on the given value which is treated as a string. Options are length, range and split.
* OPTIONS:
    * length string: The length option returns the number of characters in the given string.
    * range string first last: The range option returns the substring from the given string that lies between the indices given by first and last. An index of 0 refers to the first character in the string. If last is beyond the length of the string, then it becomes the index of the last character in the string. If first is less than last then the substring is extracted backwards.
    * split string split_ch: The split option will replace each occurrence of the character split_ch in the given string with a space.
* EXAMPLES: Some simple examples:
  ```
  set string ABCDEFGHIJ             # create the string

  set subs `string length $string'   # get it's length

  echo $subs
  ```

```
10

set subs `string range $string 2 2'     # extract character
 at index 3

echo $subs

C

set subs `string range $string 3 6'     # extract from indices
 3 to 6

echo $subs

DEFG

set subs `string range $string 6 3'     # backwards

echo $subs

GFED

set subs `string range $string 4 70'    # extract from index
 4 to 70 (or end)

echo $subs

EFGHIJ

set string D1,D2,D4,D8              # create the string

set subs `string split $string ,'  # split the string

echo $subs

D1 D2 D4 D8
```

13.102 SUFFIX − Suffixes permitted on some big decimal numbers

- DESCRIPTION: In some commands that can take big decimal numbers as arguments certain suffixes are allowed. The suffix is a single alphabetical character that must follow immediately after the decimal number it is qualifying. The alphabetical character may be upper or lower case. Negative numbers cannot have suffixes. The accepted suffixes are:
- w: multiply number by 2
- b: multiply number by 512
- k: multiply number by 1024
- m: multiply number by 1048576
- g: multiply number by 1073741824

The resulting number must fit in a 32 bit unsigned number which means "3G" can be represented (== 3,221,225,472) but "4G" cannot. "0G" is allowable and is interpreted as zero.

* COMMANDS USING SUFFIXES: dd, stargd

13.103 SYSLOG − device to send system messages for logging

* SYNOPSIS: bind −k syslog bind_point
* DESCRIPTION: The syslog file system provides user level entry of messages into the system logging facility (see syslogd). Writes to this device will be time−stamped and stored with a message class of "NOTICE" in the system log. Reads from this device will return EOF. By default the system logging device is bound at /dev/syslog.
* EXAMPLE:

```
       > /dev/syslog
bind -k syslog /dev/syslog

ls -l /dev/syslog

/dev/syslog                   syslog    0 0x00000000    0 0

echo {Some important message} > /dev/syslog

syslogd

1: INFO: RaidRunner Syslog Boot

10: NOTICE: Some important message
```

* SEE ALSO: syslogd

13.104 SYSLOGD − initialize or access messages in the system log area

* SYNOPSIS: syslogd [−I] [−p cnt]
* DESCRIPTION: syslogd either initialize the system message logging area or print stored messages from that area.
* OPTIONS:
* −I: Initialize the system message logging area. This command is usually executed at boot time and is not normally invoked by the user.
* −p cnt: Print the last cnt messages stored in the system message logging area. /nr This is the default if no options are given and cnt is set to 20. Messages are printed in the format −

```
timestamp: message class: message
```

where timestamp is the time the message was logged recorded as the number of seconds from the time the RaidRunner was booted, message class is the type of message logged indication the importance (or class) of the message. message is the message itself

* MESSAGE CLASS There are currently nine (9) message classes

1. EMERG: messages of an extremely serious nature from which the RaidRunner cannot recover
2. ALERT: messages of a serious nature from which the RaidRunner can only partially recover
3. CRIT: messages of a serious nature from which the RaidRunner can almost fully recover
4. ERR: messages indicating internal errors

5. WARNING: messages of a serious from which the RaidRunner can fully recover, for example automatic allocation of hot spare to Raid 1, 3 or 5 file system.
6. NOTICE :messages logged via writes to syslog device
7. INFO: informative messages
8. DEBUG: debugging messages options are given and cnt is set to 20.
9. REPEATS: Indicates that the previous message has been repeated N times every S seconds since it's initial entry.

* SYSLOG OUTPUT: When messages are logged, they are stored in the system message logging area. If the global environment variable, SYSLOG_CONF, is set to a value of 'C' or is not set at all then messages will be printed to the console as well. If the message has a suffix of RPT N/S then this message has been logged N times every S seconds since it's initial entry into the system message logging area.
* SEE ALSO: syslog, setenv

13.105 TEST − condition evaluation command

* SYNOPSIS: test expr
* DESCRIPTION: test evaluates the expr and if its value is true then it returns a K9 status of NIL which is equivalent to the true command. Alternatively if test evaluates the expr and if its value is false then it returns a K9 status of "false" which is equivalent to the false command. If no arguments are given then a K9 status of "false" is returned. If the expression doesn't obey the syntax given below then an error message is written to standard error and returned as the K9 status. Note that any non−NIL K9 status is interpreted as false by husky condition logic (e.g. "if"). The basic expressions use by test fall into 3 categories: file tests, string tests and numeric tests. These basic expressions can be modified or combined into larger expressions by a 4th category called operators.
* FILE TESTS: The following file tests are supported:
    * −d file: True if file exists and is a directory.
    * −f file: True if file exists.
    * −s file: True if file exists and is non−zero length.
* STRING TESTS: The command "test {}" is syntactically correct and has the value "false" because "{}" is an empty string. The command "test" by itself has the value "false". The commands "test hello" and "test −z" both have the value "true". The following string tests are supported:
    * −n str: True if the length of str is non−zero.
    * −z str: True if the length of str is zero.
    * str1 = str2: True if str1 and str2 are identical. Spaces surrounding "=" are required.
    * str1 != str2: True if str1 and str2 are not identical. Spaces surrounding "!=" are required.
    * str: True if str is not a null (empty) string
* NUMERIC TESTS: These numeric arguments are evaluated as signed integers. Currently an internal 32 bit integer representation is used limiting the range of valid integers from −2,147,483,648 to 2,147,483,647 (inclusive). Thus fixed point and floating point numbers cannot be compared. The spaces surrounding the numeric comparison tokens (e.g. "−eq") are required. Instead of a number an expression of the form "−l str" can be given. This evaluates to the number of characters in the string str. The following numeric tests are supported:
    * n1 −eq n2: True if n1 and n2 are numerically equal
    * n1 −ge n2: True if n1 is greater than or equal to n2
    * n1 −gt n2: True if n1 is greater than n2
    * n1 −le n2: True if n1 is less than or equal to n2
    * n1 −lt n2: True if n1 is less than n2
    * n1 −ne n2: True if n1 is not equal to n2
* OPERATORS: The following operators are supported:

- ♦ !: Unary negation operator. Appears to the left of the expression it is negating.
- ♦ −a: Binary AND operator. Appears between the 2 expressions it is logically combining.
- ♦ −o: Binary OR operator. Appears between the 2 expressions it is logically combining.
- ♦ (expr): Parentheses are used to group expressions. Since parentheses are husky special characters they need to be quoted or escaped.
- ♦ The relative precedence of these operators from high to low is: () ! −a −o. Thus the expression:

```
test ! -f /bin/ls -a -d /env
```

is the same as:

```
test {(} ! -f /bin/ls {)} -a -d /env
```

The "!" operator should appear to the left of other unary operators. Basic binary operators have higher precedence than "−a" and "−o".

- SEE ALSO: husky

13.106 TIME − Print the number of seconds since boot (or reset of clock)

- SYNOPSIS: time
- DESCRIPTION: time will print the time in seconds since the RaidRunner was booted or since the clock was last reset.

13.107 TRAP − intercept a signal and perform some action

- SYNOPSIS:
  - ♦ trap
  - ♦ trap n ...
  - ♦ trap {} n ...
  - ♦ trap arg n ...
- DESCRIPTION: trap allows signals directed at this process to intercept signals and potentially take some special action. When trap is used with no arguments then all current (non− default) traps for this process are printed. When the first argument is a number (optionally followed by other numbers) then this number is interpreted as a signal number whose action is to be set to the default for that signal (extra numbers are treated in a similar fashion). When the first argument is an empty string (i.e. {}) then the signals nominated by the following numbers are ignored. When the first argument is a non−empty string then if the signals nominated by the following numbers occur, then "arg" will be executed. N.B. Signal number 4 (kill) can be neither caught nor ignored. The following table maps signal numbers to an explanation:
- 0 unused signal
- 1 hangup
- 2 interrupt (rubout)
- 3 quit (ASCII FS)
- 4 kill (cannot be caught or ignored)
- 5 write on a pipe with no one to read it
- 6 alarm clock
- 7 software termination signal
- 8 child process has changed state
- 9 process could not obtain memory (from heap)

* SEE ALSO: kill

13.108 TRUE − returns the K9 true status

* SYNOPSIS: true
* DESCRIPTION: true does nothing other than return the K9 true status. K9 processes return a pointer to a C string (null terminated array of characters) on termination. If that pointer is NULL then a true exit value is assumed while all other returned pointer values are interpreted as false (with the string being some explanation of what went wrong). This command returns a NULL pointer value as its return value. Returning a NULL pointer value is sometimes referred to as returning NIL. The husky shell interprets the token ":" to have the same meaning as true.
* SEE ALSO: false

13.109 STTY or TTY − print the user's terminal mount point or terminfo status

* SYNOPSIS:
    * tty
    * stty [ttyname]
* DESCRIPTION: tty examines the device on which the it is running, and if the device is on a DUART, then the mount point of the device is printed. If the device is not a DUART, then an appropriate message is printed. stty prints the terminfo structure associated with either the terminal device that stty is being executed from or the given terminal device − ttyname. If an invalid device name is given or the device stty is being executed from is not a DUART, an appropriate message is printed. This command is mainly useful for debugging the RaidRunner kernel.
* RETURN STATUS: On success a return code of 0 is returned, else 1 is returned.

13.110 UNSET − delete one or more environment variables

* SYNOPSIS
    * unset name
    * unset name name ...
* DESCRIPTION: unset removes the given environment variable(s). This is done by removing the given variable(s) from the /env file system.
* SEE ALSO: set, env

13.111 UNSETENV − unset (delete) a GLOBAL environment variable

* SYNOPSIS: unsetenv name [name ...]
* DESCRIPTION: unsetenv deletes the GLOBAL environment variable name along with it's contents. If multiple names are given, then each GLOBAL environment variable is deleted. If the given name is not GLOBAL environment variable then nothing is done and no error status is set.
* NOTE: A GLOBAL environment variable is one which is stored in a non volatile area on the RaidRunner and hence is available between successive power cycles or reboots. These variables ARE NOT the same as husky environment variables. The non volatile area is co−located with the RaidRunner Configuration area.

- SEE ALSO: printenv, setenv, rconf

13.112 VERSION − print out the version of the RaidRunner kernel

- SYNOPSIS: version
- DESCRIPTION: version prints out the version of the RaidRunner code which comprises the version number, date of creation and creator.

13.113 WAIT − wait for a process (or my children) to terminate

- SYNOPSIS: wait [pid]
- DESCRIPTION: wait either waits for the given pid (process identifier) to terminate or, if no argument is given, waits for all the children of this process to terminate. When a pid is given then the exit status of that process is returned by this command. When no pid is given then this command returns a NIL status (when all children have terminated).
- SEE ALSO: K9wait, K9kill

13.114 WARBLE − periodically pulse the buzzer

- SYNOPSIS: warble period count
- DESCRIPTION: warble will save the current state of the buzzer (on or off), then turn the buzzer on for period milliseconds, then off for period/2 and repeat this for count times.
- SEE ALSO: buzzer, sos

13.115 XD− dump given file(s) in hexa−decimal to standard out

- SYNOPSIS: xd [file ...]
- DESCRIPTION: xd dumps each given file in hexadecimal format to standard out. If no file is given (or it is "−") then standard in is used.

13.116 ZAP − write zeros to a file

SYNOPSIS: zap [−b blockSize] [−f byteVal] count offset <>[3] store

DESCRIPTION: zap writes count * 8192 bytes of zeros at byte position offset * 8192 into file store (which is opened and associated with file descriptor 3). Both count and offset may have a suffix. The optional "−b" switch allows the block size to be set to blockSize bytes. The default block size is 8192 bytes. The optional "−f" switch allows the fill character to be set to byteVal which should be a number in the range 0 to 255 (inclusive). The default fill character is 0 (i.e. zero). Every 100 write operations the current count is output (usually overwriting the previous count output). Errors on the write operations are ignored.

SEE ALSO: suffix

13.117 ZCACHE − Manipulate the zone optimization IO table of a Raid Set's cache

- SYNOPSIS: zcache −c cache_moniker [−E extents] [−P portion] [−W 0|1] [−p]
- DESCRIPTION: zcache is a utility that will print and or modify the zone optimized IO table of a given cache range.
- OPTIONS:
    - −c cache_moniker: Specify the cache range, cache_moniker, to print the zone table or modify it's contents. cache_moniker, is the name which was set in the add moniker=cache_moniker first= ... command (see cache). If no other options are given the zone table is printed. When the zone table is printed it is in the form of
      ```
      n state z1_lo..z1_hi@z1_blks z2_lo..z2_hi@z2_blks ... zn_lo..zn_hi@zn_blks
      ```
- where n is the number of IO optimized zones, state is either "on" for zone optimized IO, "off" for no zone optimized IO or "none" if no zones are available. z1_lo..z1_hi@z1_blks ... zn_lo..zn_hi@zn_blks lists the starting and ending blocks and the optimized block count for each zone.
- −E extents: Specify the number of data drives in the Raid Set for which the cache range exists.
- −P portion: Specify the percentage, limited to be between 50 and 300, that the optimized block count for each zone is to be adjusted. That is, if you need to reduce the optimized block count for each zone by say 10% you would set portion to 90.
- −W 0|1: Turn on (1) or off (0) zone IO optimizations for the given cache range.

- SEE ALSO: cache

13.118 ZERO − file when read yields zeros continuously

- SYNOPSIS: bind −k {zero [#]} bind_point
- DESCRIPTION: zero is a special file that when written to is an infinite sink of data (i.e. anything can be written to it and it will be disposed of quickly). When zero is read it is an infinite source of zeros (i.e. the byte value 0). The zero file will appear in the K9 namespace at the bind_point. If the optional "#" is given then the zero file still is an infinite sink of data. However, when read, each 512 bytes is viewed as a block (starting at block 0 at the beginning of the file). Each block contains 128 4 byte integers ("unsigned long" in C) each of which contain the current block number. This option is meant as a debugging aid.
- EXAMPLE: Husky installs a zero special file as follows:
  ```
  bind -k zero /dev/zero
  ```

Example of use to make 32 Kilobyte file (called "/fill") full of zeros.

```
dd if=/dev/zero of=/fill bs=8k count=4
```

- SEE ALSO: null, log

13.119 ZLABELS − Write zeros to the front and end of Raid Sets

- SYNOPSIS
    - zlabels −a
    - zlabels raidset [raidset ....]
- DESCRIPTION: zlabels is a husky script which writes zeros to both the front and end of either all

configured raid sets or to given raid sets. Typically a host operating system will write label(s) onto a disk. Nearly all operating systems write a few blocks at the start of a disk and some write copies at the end of the disk as well. As this label contains formatting and modified disk geometry information a change to a raid set that effects the geometry of it's offered disk drive will conflict with any labels written before. zlabels writes zero's at the start of a raid set and also at the end. The number of blocks zeroed is dependant on the raid set type and io chunksize. Typically 50 io chunk size blocks are written at the start and 49 at the end. In the case of a raid type 3, the number of data drives times 50 (and 49) are written.

- OPTIONS :
    - −a: All configured raid sets will be zeroed.
    - raidset: The named raid set, or raid sets, are zeroed.
- SEE ALSO: dd

---

14. Advanced Topics: SCSI Monitor Daemon (SMON)

Another way of communicating with the onboard controller from the host operating system is using the SCSI Monitor (SMON) facility. SMON provides an ASCII communication channel on an assigned SCSI ID and LUN. The commands discussed in section 7 may also be issued over this channel to manipulate the RAID configuration and operation. This mechanism is utilized under Solaris to provide a communication channel between an X Based GUI and the RAID controller. It is currently un−utilized under Linux. See the description of the smon daemon in the 5070 command reference above.

---

15. Further Reading

- The Linux software−RAID−HOWTO by Linas Vepstas
- The Plan9 pages at AT&T Bell Labs: http://plan9.bell−labs.com/plan9/index.htm

---