Red Hat GFS 6.0

Administrator's Guide

Red Hat GFS 6.0: Administrator's Guide

Red Hat, Inc.

1801 Varsity Drive
Raleigh NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park NC 27709 USA

rh-gfsg(EN)-6.0-Print-RHI (2005-08-02T11:07-0400)

The GPG fingerprint of the security@redhat.com key is:
CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

Table of Contents

Introduction .... **i**
1. Audience i
2. Document Conventions i
3. More to Come iii
3.1. Send in Your Feedback iv
4. Activate Your Subscription iv
4.1. Provide a Red Hat Login iv
4.2. Provide Your Subscription Number v
4.3. Connect Your System v
5. Recommended References v

1. Red Hat GFS Overview **1**
1.1. New and Changed Features 1
1.2. Performance, Scalability, and Economy 2
1.2.1. Superior Performance and Scalability 2
1.2.2. Performance, Scalability, Moderate Price 3
1.2.3. Economy and Performance 4
1.3. GFS Functions 5
1.3.1. Cluster Volume Management 5
1.3.2. Lock Management 5
1.3.3. Cluster Management, Fencing, and Recovery 6
1.3.4. Cluster Configuration Management 6
1.4. GFS Software Subsystems 7
1.5. Before Configuring GFS 9

2. System Requirements **11**
2.1. Platform Requirements 11
2.2. TCP/IP Network 11
2.3. Fibre Channel Storage Network 11
2.4. Fibre Channel Storage Devices 12
2.5. Network Power Switches 12
2.6. Console Access 12
2.7. I/O Fencing 12

3. Installing GFS **13**
3.1. Prerequisite Tasks 13
3.1.1. Prerequisite Software 13
3.1.2. Specifying a Persistent Major Number 14
3.2. Installation Tasks 14
3.2.1. Installing GFS RPMs 14
3.2.2. Loading the GFS Kernel Modules 15

4. Initial Configuration **17**
4.1. Prerequisite Tasks 17
4.2. Initial Configuration Tasks 17
4.2.1. Setting Up Logical Devices 18
4.2.2. Setting Up and Starting the Cluster Configuration System 18
4.2.3. Starting Clustering and Locking Systems 19
4.2.4. Setting Up and Mounting File Systems 19

5. Using the Pool Volume Manager 21
5.1. Overview of GFS Pool Volume Manager 21
5.2. Synopsis of Pool Management Commands 21
5.2.1. pool_tool 22
5.2.2. pool_assemble 23
5.2.3. pool_info 23
5.2.4. pool_mp 24
5.3. Scanning Block Devices 25
5.3.1. Usage.. 25
5.3.2. Example 25
5.4. Creating a Configuration File for a New Volume 26
5.4.1. Examples.. 27
5.5. Creating a Pool Volume 27
5.5.1. Usage.. 28
5.5.2. Example 28
5.5.3. Comments 28
5.6. Activating/Deactivating a Pool Volume 28
5.6.1. Usage.. 29
5.6.2. Examples.. 29
5.6.3. Comments 29
5.7. Displaying Pool Configuration Information 30
5.7.1. Usage.. 30
5.7.2. Example 30
5.8. Growing a Pool Volume 30
5.8.1. Usage.. 30
5.8.2. Example procedure 31
5.9. Erasing a Pool Volume 31
5.9.1. Usage.. 32
5.9.2. Example 32
5.9.3. Comments 32
5.10. Renaming a Pool Volume.. 32
5.10.1. Usage.. 32
5.10.2. Example 33
5.11. Changing a Pool Volume Minor Number 33
5.11.1. Usage.. 33
5.11.2. Example 33
5.11.3. Comments 33
5.12. Displaying Pool Volume Information 34
5.12.1. Usage.. 34
5.12.2. Examples.. 34
5.13. Using Pool Volume Statistics 35
5.13.1. Usage.. 35
5.13.2. Examples.. 35
5.14. Adjusting Pool Volume Multipathing 35
5.14.1. Usage.. 36
5.14.2. Examples.. 36
6. Creating the Cluster Configuration System Files 37
6.1. Prerequisite Tasks 37
6.2. CCS File Creation Tasks 37
6.3. Dual Power and Multipath FC Fencing Considerations 38
6.4. GNBD Multipath Considerations for CCS Files 38
6.5. Creating the cluster.ccs File 39
6.6. Creating the fence.ccs File 41
6.7. Creating the nodes.ccs File 51

7. Using the Cluster Configuration System 75
7.1. Creating a CCS Archive 75
7.1.1. Usage 75
7.1.2. Example 76
7.1.3. Comments 76
7.2. Starting CCS in the Cluster 76
7.2.1. Usage 76
7.2.2. Example 77
7.2.3. Comments 77
7.3. Using Other CCS Administrative Options 77
7.3.1. Extracting Files from a CCS Archive 77
7.3.2. Listing Files in a CCS Archive 78
7.3.3. Comparing CCS Configuration Files to a CCS Archive 78
7.4. Changing CCS Configuration Files 78
7.4.1. Example Procedure 79
7.5. Alternative Methods to Using a CCA Device 79
7.5.1. CCA File and Server 79
7.5.2. Local CCA Files 82
7.6. Combining CCS Methods 82
8. Using Clustering and Locking Systems 85
8.1. Locking System Overview 85
8.2. LOCK_GULM 85
8.2.1. Selection of LOCK_GULM Servers 85
8.2.2. Number of LOCK_GULM Servers 86
8.2.3. Starting LOCK_GULM Servers 86
8.2.4. Fencing and LOCK_GULM 86
8.2.5. Shutting Down a LOCK_GULM Server 86
8.3. LOCK_NOLOCK 87
9. Managing GFS 89
9.1. Making a File System 89
9.1.1. Usage 89
9.1.2. Examples 90
9.1.3. Complete Options 90
9.2. Mounting a File System 91
9.2.1. Usage 92
9.2.2. Example 92
9.2.3. Complete Usage 92
9.3. Unmounting a File System 93
9.3.1. Usage 94
9.4. GFS Quota Management 94
9.4.1. Setting Quotas 94
9.4.2. Displaying Quota Limits and Usage 95
9.4.3. Synchronizing Quotas 96
9.4.4. Disabling/Enabling Quota Enforcement 97
9.4.5. Disabling/Enabling Quota Accounting 98
9.5. Growing a File System 99
9.5.1. Usage 99
9.5.2. Comments 100
9.5.3. Examples 100
9.5.4. Complete Usage 100
9.6. Adding Journals to a File System 101
9.6.1. Usage 101
9.6.2. Comments 101
9.6.3. Examples 101
9.6.4. Complete Usage 102

9.7. Direct I/O 103
9.7.1. `O_DIRECT` 103
9.7.2. GFS File Attribute 103
9.7.3. GFS Directory Attribute 104
9.8. Data Journaling 104
9.8.1. Usage 105
9.8.2. Examples 105
9.9. Configuring `atime` Updates 105
9.9.1. Mount with `noatime` 106
9.9.2. Tune GFS `atime` Quantum 106
9.10. Suspending Activity on a File System 107
9.10.1. Usage 107
9.10.2. Examples 108
9.11. Displaying Extended GFS Information and Statistics 108
9.11.1. Usage 108
9.11.2. Examples 109
9.12. Repairing a File System 109
9.12.1. Usage 109
9.12.2. Example 110
9.13. Context-Dependent Path Names 110
9.13.1. Usage 110
9.13.2. Example 111
9.14. Shutting Down a GFS Cluster 112
9.15. Starting a GFS Cluster 112
10. Using the Fencing System 115
10.1. How the Fencing System Works 115
10.2. Fencing Methods 115
10.2.1. APC MasterSwitch 116
10.2.2. WTI Network Power Switch 117
10.2.3. Brocade FC Switch 117
10.2.4. Vixel FC Switch 117
10.2.5. HP RILOE Card 118
10.2.6. GNBD 118
10.2.7. Manual 118
11. Using GNBD 121
11.1. GNBD Driver and Command Usage 121
11.1.1. Exporting a GNBD from a Server 121
11.1.2. Importing a GNBD on a Client 123
11.2. Considerations for Using GNBD Multipath 123
11.2.1. Linux Page Caching 123
11.2.2. Lock Server Startup 124
11.2.3. CCS File Location 124
11.2.4. Fencing GNBD Server Nodes 125
11.3. Running GFS on a GNBD Server Node 125
12. Using GFS `init.d` Scripts 127
12.1. GFS `init.d` Scripts Overview 127
12.2. GFS `init.d` Scripts Use 127
A. Using Red Hat GFS with Red Hat Cluster Suite 131
A.1. Terminology 131
A.2. Changes to Red Hat Cluster 132
A.3. Installation Scenarios 132
A.3.1. New Installations of Red Hat GFS and Red Hat Cluster Manager 132
A.3.2. Adding Red Hat GFS to an Existing Red Hat Cluster Manager Deployment 132
A.3.3. Upgrading Red Hat GFS 5.2.1 to Red Hat GFS 6.0 133

B. Upgrading GFS **135**
C. Basic GFS Examples **137**
C.1. LOCK_GULM, RLM Embedded 137
C.1.1. Key Characteristics 137
C.1.2. Kernel Modules Loaded 138
C.1.3. Setup Process 139
C.2. LOCK_GULM, RLM External 142
C.2.1. Key Characteristics 142
C.2.2. Kernel Modules Loaded 143
C.2.3. Setup Process 144
C.3. LOCK_GULM, SLM Embedded 148
C.3.1. Key Characteristics 148
C.3.2. Kernel Modules Loaded 150
C.3.3. Setup Process 150
C.4. LOCK_GULM, SLM External 153
C.4.1. Key Characteristics 154
C.4.2. Kernel Modules Loaded 155
C.4.3. Setup Process 155
C.5. LOCK_GULM, SLM External, and GNBD 159
C.5.1. Key Characteristics 159
C.5.2. Kernel Modules Loaded 161
C.5.3. Setup Process 161
C.6. LOCK_NOLOCK 166
C.6.1. Key Characteristics 166
C.6.2. Kernel Modules Loaded 167
C.6.3. Setup Process 167
Index **171**
Colophon **177**

Introduction

Welcome to the *Red Hat GFS Administrator's Guide*. This book provides information about installing, configuring, and maintaining GFS (Global File System). The document contains procedures for commonly performed tasks, reference information, and examples of complex operations and tested GFS configurations.

HTML and PDF versions of all the official Red Hat Enterprise Linux manuals and release notes are available online at http://www.redhat.com/docs/.

1. Audience

This book is intended primarily for Linux system administrators who are familiar with the following activities:

- Linux system administration procedures, including kernel configuration
- Installation and configuration of shared storage networks, such as Fibre Channel SANs

2. Document Conventions

When you read this manual, certain words are represented in different fonts, typefaces, sizes, and weights. This highlighting is systematic; different words are represented in the same style to indicate their inclusion in a specific category. The types of words that are represented this way include the following:

`command`

> Linux commands (and other operating system commands, when used) are represented this way. This style should indicate to you that you can type the word or phrase on the command line and press [Enter] to invoke a command. Sometimes a command contains words that would be displayed in a different style on their own (such as file names). In these cases, they are considered to be part of the command, so the entire phrase is displayed as a command. For example:
>
> Use the `cat testfile` command to view the contents of a file, named `testfile`, in the current working directory.

`file name`

> File names, directory names, paths, and RPM package names are represented this way. This style should indicate that a particular file or directory exists by that name on your system. Examples:
>
> The `.bashrc` file in your home directory contains bash shell definitions and aliases for your own use.
>
> The `/etc/fstab` file contains information about different system devices and file systems.
>
> Install the `webalizer` RPM if you want to use a Web server log file analysis program.

application

> This style indicates that the program is an end-user application (as opposed to system software). For example:
>
> Use **Mozilla** to browse the Web.

[key]

A key on the keyboard is shown in this style. For example:

To use [Tab] completion, type in a character and then press the [Tab] key. Your terminal displays the list of files in the directory that start with that letter.

[key]-[combination]

A combination of keystrokes is represented in this way. For example:

The [Ctrl]-[Alt]-[Backspace] key combination exits your graphical session and return you to the graphical login screen or the console.

text found on a GUI interface

A title, word, or phrase found on a GUI interface screen or window is shown in this style. Text shown in this style is being used to identify a particular GUI screen or an element on a GUI screen (such as text associated with a checkbox or field). Example:

Select the **Require Password** checkbox if you would like your screensaver to require a password before stopping.

top level of a menu on a GUI screen or window

A word in this style indicates that the word is the top level of a pulldown menu. If you click on the word on the GUI screen, the rest of the menu should appear. For example:

Under **File** on a GNOME terminal, the **New Tab** option allows you to open multiple shell prompts in the same window.

If you need to type in a sequence of commands from a GUI menu, they are shown like the following example:

Go to **Main Menu Button** (on the Panel) => **Programming** => **Emacs** to start the **Emacs** text editor.

button on a GUI screen or window

This style indicates that the text can be found on a clickable button on a GUI screen. For example:

Click on the **Back** button to return to the webpage you last viewed.

`computer output`

Text in this style indicates text displayed to a shell prompt such as error messages and responses to commands. For example:

The `ls` command displays the contents of a directory. For example:

```
Desktop                about.html        logs          paulwesterberg.png
Mail                   backupfiles       mail          reports
```

The output returned in response to the command (in this case, the contents of the directory) is shown in this style.

`prompt`

A prompt, which is a computer's way of signifying that it is ready for you to input something, is shown in this style. Examples:

`$`

`#`

`[stephen@maturin stephen]$`

`leopard login:`

`user input`

Text that the user has to type, either on the command line, or into a text box on a GUI screen, is displayed in this style. In the following example, **`text`** is displayed in this style:

To boot your system into the text based installation program, you must type in the **`text`** command at the `boot:` prompt.

replaceable

Text used for examples, which is meant to be replaced with data provided by the user, is displayed in this style. In the following example, *<version-number>* is displayed in this style:

The directory for the kernel source is /usr/src/*<version-number>*/, where *<version-number>* is the version of the kernel installed on this system.

Additionally, we use several different strategies to draw your attention to certain pieces of information. In order of how critical the information is to your system, these items are marked as a note, tip, important, caution, or warning. For example:

Note

Remember that Linux is case sensitive. In other words, a rose is not a ROSE is not a rOsE.

Tip

The directory `/usr/share/doc/` contains additional documentation for packages installed on your system.

Important

If you modify the DHCP configuration file, the changes do not take effect until you restart the DHCP daemon.

Caution

Do not perform routine tasks as root — use a regular user account unless you need to use the root account for system administration tasks.

Warning

Be careful to remove only the necessary Red Hat GFS partitions. Removing other partitions could result in data loss or a corrupted system environment.

3. More to Come

The *Red Hat GFS Administrator's Guide* is part of Red Hat's growing commitment to provide useful and timely support to Red Hat Enterprise Linux users.

3.1. Send in Your Feedback

If you spot a typo in the *Red Hat GFS Administrator's Guide*, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla (http://www.redhat.com/bugzilla) against the product `Red Hat Cluster Suite`, version `3`, component `rh-gfsg`.

Be sure to mention the manual's identifier:

```
rh-gfsg(EN)-6.0-Print-RHI (2005-08-02T11:07-0400)
```

If you mention this manual's identifier, we will know exactly which version of the guide you have.

If you have a suggestion for improving the documentation, try to be as specific as possible. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

4. Activate Your Subscription

Before you can access service and software maintenance information, and the support documentation included in your subscription, you must activate your subscription by registering with Red Hat. Registration includes these simple steps:

- Provide a Red Hat login
- Provide a subscription number
- Connect your system

The first time you boot your installation of Red Hat Enterprise Linux, you are prompted to register with Red Hat using the **Setup Agent**. If you follow the prompts during the **Setup Agent**, you can complete the registration steps and activate your subscription.

If you can not complete registration during the **Setup Agent** (which requires network access), you can alternatively complete the Red Hat registration process online at http://www.redhat.com/register/.

4.1. Provide a Red Hat Login

If you do not have an existing Red Hat login, you can create one when prompted during the **Setup Agent** or online at:

```
https://www.redhat.com/apps/activate/newlogin.html
```

A Red Hat login enables your access to:

- Software updates, errata and maintenance via Red Hat Network
- Red Hat technical support resources, documentation, and Knowledgebase

If you have forgotten your Red Hat login, you can search for your Red Hat login online at:

```
https://rhn.redhat.com/help/forgot_password.pxt
```

4.2. Provide Your Subscription Number

Your subscription number is located in the package that came with your order. If your package did not include a subscription number, your subscription was activated for you and you can skip this step.

You can provide your subscription number when prompted during the **Setup Agent** or by visiting http://www.redhat.com/register/.

4.3. Connect Your System

The Red Hat Network Registration Client helps you connect your system so that you can begin to get updates and perform systems management. There are three ways to connect:

1. During the **Setup Agent** — Check the **Send hardware information** and **Send system package list** options when prompted.
2. After the **Setup Agent** has been completed — From the **Main Menu**, go to **System Tools**, then select **Red Hat Network**.
3. After the **Setup Agent** has been completed — Enter the following command from the command line as the root user:
   - `/usr/bin/up2date --register`

5. Recommended References

For additional references about related topics, refer to the following table:

Topic	Reference	Comment
Shared Data Clustering and File Systems	*Shared Data Clusters* by Dilip M. Ranade. Wiley, 2002.	Provides detailed technical information on cluster file system and cluster volume-manager design.
Storage Area Networks (SANs)	*Designing Storage Area Networks: A Practical Reference for Implementing Fibre Channel and IP SANs, Second Edition* by Tom Clark. Addison-Wesley, 2003.	Provides a concise summary of Fibre Channel and IP SAN Technology.
	Building SANs with Brocade Fabric Switches by C. Beauchamp, J. Judd, and B. Keo. Syngress, 2001.	Best practices for building Fibre Channel SANs based on the Brocade family of switches, including core-edge topology for large SAN fabrics.
	Building Storage Networks, Second Edition by Marc Farley. Osborne/McGraw-Hill, 2001.	Provides a comprehensive overview reference on storage networking technologies.

Topic	Reference	Comment
Applications and High Availability	*Blueprints for High Availability: Designing Resilient Distributed Systems* by E. Marcus and H. Stern. Wiley, 2000.	Provides a summary of best practices in high availability.

Table 1. Recommended References Table

Chapter 1.

Red Hat GFS Overview

Red Hat GFS is a cluster file system that provides data sharing among Linux-based computers. GFS provides a single, consistent view of the file system name space across all nodes in a cluster. It allows applications to install and run without much knowledge of the underlying storage infrastructure. GFS is fully compliant with the IEEE POSIX interface, allowing applications to perform file operations as if they were running on a local file system. Also, GFS provides features that are typically required in enterprise environments, such as quotas, multiple journals, and multipath support.

GFS provides a versatile method of networking your storage according to the performance, scalability, and economic needs of your storage environment.

This chapter provides some very basic, abbreviated information as background to help you understand GFS. It contains the following sections:

- Section 1.1 *New and Changed Features*
- Section 1.2 *Performance, Scalability, and Economy*
- Section 1.3 *GFS Functions*
- Section 1.4 *GFS Software Subsystems*
- Section 1.5 *Before Configuring GFS*

1.1. New and Changed Features

This section lists new and changed features included with the initial release of Red Hat GFS 6.0 and Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5.

New and Changed Features with the Initial Release of Red Hat GFS 6.0

- File access control lists (ACLs) and extended file attributes in GFS file systems — This release adds the capability of setting and getting file ACLs and extended file attributes in a GFS file system. The Linux commands `setfacl` and `getfacl` set and get ACLs. The Linux commands `setfattr` and `getfattr` set and get file attributes. In addition, this release adds a GFS-specific `mount` command option, `-o acl`. The new option allows users to set ACLs. For more information about the `-o acl` option, refer to Section 9.2 *Mounting a File System*.
- Additional fencing agents — This release adds fencing agents for McData Fibre Channel (FC) switches, Egenera BladeFrame systems, and xCAT (Extreme Cluster Administration Toolkit) clusters.
- Initialization scripts — This release adds `init.d` scripts for the `pool`, `ccsd`, `lock_gulmd`, and `gfs` modules. For more information about the scripts, refer to Chapter 12 *Using GFS `init.d` Scripts*.
- Configurable node-failure detection parameters — This release adds optional parameters for setting heartbeat rate and allowed misses. Together, the parameters determine the time interval allowed without response from a node before the node is considered to have failed. For more information, refer to Section 6.5 *Creating the `cluster.ccs` File*.
- Removal of license mechanism — Previous GFS releases required a license file that defined the term of use and which GFS features were enabled. This release does not require a license file.

- Initial-configuration druid via Red Hat Cluster Suite — When GFS is installed with Red Hat Cluster Suite, a configuration druid is available with Cluster Suite for initial configuration of GFS. For more information about the druid, refer to the Cluster Suite documentation.

New and Changed Features with Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5

- Enhanced `gfs_fsck` performance and changes to the `gfs_fsck` command — The `gfs_fsck` function performs 10 times as fast as `gfs_fsck` in releases earlier than Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5. In addition, the enhanced `gfs_fsck` function includes changes to certain command options. For more information about changes to the command options, refer to Section 9.12 *Repairing a File System*.
- Optional `usedev` key available for use with the `nodes.ccs` file (`nodes.ccs:nodes`) — The value of the `usedev` key is a named device from the `ip_interfaces` section. If `usedev` is present, GULM uses the IP address from that device in the `ip_interfaces` section. Otherwise GULM uses the IP address from `libresolv`, as it does in releases earlier than Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5. For more information about the `usedev` key, refer to Section 6.7 *Creating the `nodes.ccs` File*

For information about using GFS with Red Hat Cluster Suite, refer to Appendix A *Using Red Hat GFS with Red Hat Cluster Suite*. For GFS upgrade instructions, refer to Appendix B *Upgrading GFS*.

1.2. Performance, Scalability, and Economy

You can deploy GFS in a variety of configurations to suit your needs for performance, scalability, and economy. For superior performance and scalability, you can deploy GFS in a cluster that is connected directly to a SAN. For more economical needs, you can deploy GFS in a cluster that is connected to a LAN with servers that use GNBD (Global Network Block Device). A GNBD provides block-level storage access over an Ethernet LAN. (For more information about GNBD, refer to Chapter 11 *Using GNBD*.)

The following sections provide examples of how GFS can be deployed to suit your needs for performance, scalability, and economy:

- Section 1.2.1 *Superior Performance and Scalability*
- Section 1.2.2 *Performance, Scalability, Moderate Price*
- Section 1.2.3 *Economy and Performance*

Note

The deployment examples in this chapter reflect basic configurations; your needs might require a combination of configurations shown in the examples.

1.2.1. Superior Performance and Scalability

You can obtain the highest shared-file performance when applications access storage directly. The GFS SAN configuration in Figure 1-1 provides superior file performance for shared files and file systems. Linux applications run directly on GFS clustered application nodes. Without file protocols or storage servers to slow data access, performance is similar to individual Linux servers with direct-connect storage; yet, each GFS application node has equal access to all data files. GFS supports over 300 GFS application nodes.

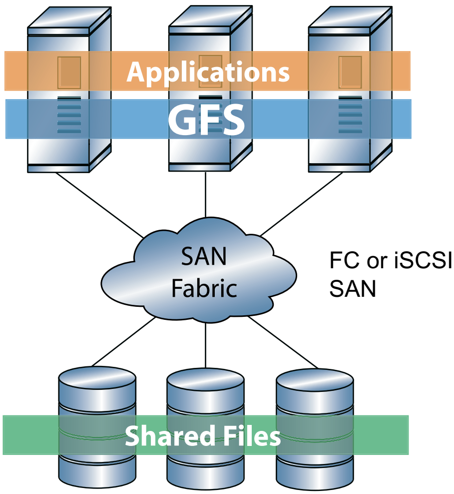

Figure 1-1. GFS with a SAN

1.2.2. Performance, Scalability, Moderate Price

Multiple Linux client applications on a LAN can share the same SAN-based data as shown in Figure 1-2. SAN block storage is presented to network clients as block storage devices by GNBD servers. From the perspective of a client application, storage is accessed as if it were directly attached to the server in which the application is running. Stored data is actually on the SAN. Storage devices and data can be equally shared by network client applications. File locking and sharing functions are handled by GFS for each network client.

Note

Clients implementing ext2 and ext3 file systems can be configured to access their own dedicated slice of SAN storage.

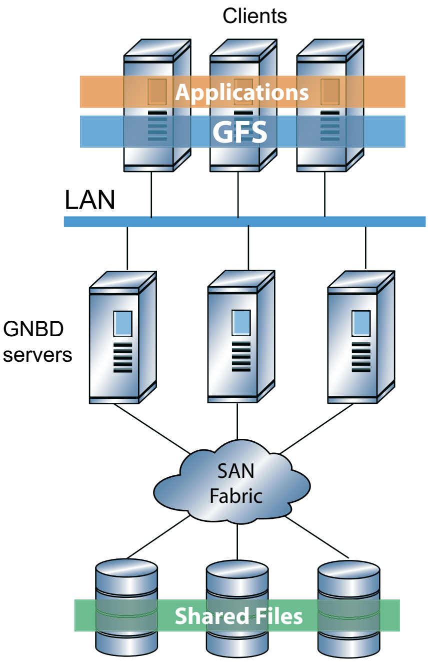

Figure 1-2. GFS and GNBD with a SAN

1.2.3. Economy and Performance

Figure 1-3 shows how Linux client applications can take advantage of an existing Ethernet topology to gain shared access to all block storage devices. Client data files and file systems can be shared with GFS on each client. Application failover can be fully automated with Red Hat Cluster Suite.

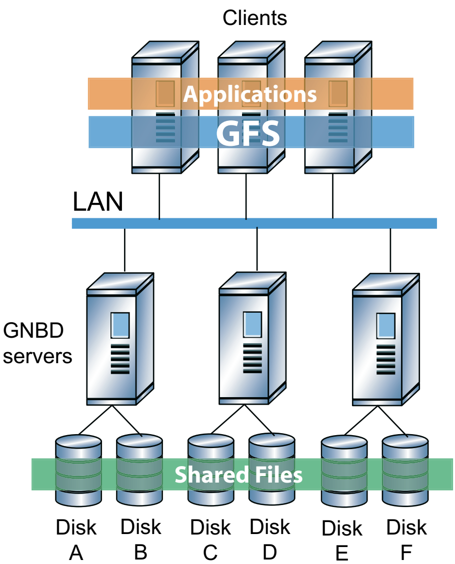

Figure 1-3. GFS and GNBD with Directly Connected Storage

1.3. GFS Functions

GFS is a native file system that interfaces directly with the VFS layer of the Linux kernel file-system interface. GFS is a cluster file system that employs distributed metadata and multiple journals for optimal operation in a cluster.

GFS provides the following main functions:

- Cluster volume management
- Lock management
- Cluster management, fencing, and recovery
- Cluster configuration management

1.3.1. Cluster Volume Management

Cluster volume management provides simplified management of volumes and the ability to dynamically extend file system capacity without interrupting file-system access. With cluster volume management, you can aggregate multiple physical volumes into a single, logical device across all nodes in a cluster.

Cluster volume management provides a logical view of the storage to GFS, which provides flexibility for the administrator in how the physical storage is managed. Also, cluster volume management provides increased availability because it allows increasing the storage capacity without shutting down the cluster. Refer to Chapter 5 *Using the Pool Volume Manager* for more information about cluster volume management.

1.3.2. Lock Management

A lock management mechanism is a key component of any cluster file system. The Red Hat GFS lock-management mechanism provides the following lock managers:

- *Single Lock Manager* (SLM) — A simple centralized lock manager that can be configured to run either on a file system node or on a separate dedicated lock manager node.
- *Redundant Lock Manager* (RLM) — A high-availability lock manager. It allows the configuration of a master and multiple hot-standby failover lock manager nodes. The failover nodes provide failover in case the master lock manager node fails.

The lock managers also provide cluster management functions that control node recovery. Refer to Chapter 8 *Using Clustering and Locking Systems* for a description of the GFS lock protocols.

1.3.3. Cluster Management, Fencing, and Recovery

Cluster management functions in GFS monitor node status through heartbeat signals to determine cluster membership. Also, cluster management keeps track of which nodes are using each GFS file system, and initiates and coordinates the recovery process when nodes fail. This process involves recovery coordination from the fencing system, the lock manager, and the file system. The cluster management functions are embedded in each of the lock management modules described earlier in Lock Management. Refer to Chapter 8 *Using Clustering and Locking Systems* for more information on cluster management.

Fencing is the ability to isolate or "fence off" a cluster node when that node loses its heartbeat notification with the rest of the cluster nodes. Fencing ensures that data integrity is maintained during the recovery of a failed cluster node. GFS supports a variety of automated fencing methods and one manual method. In addition, GFS provides the ability to configure each cluster node for cascaded fencing with the automated fencing methods. Refer to Chapter 10 *Using the Fencing System* for more information about the GFS fencing capability.

Warning

Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.

Recovery is the process of controlling reentry of a node into a cluster after the node has been fenced. Recovery ensures that storage data integrity is maintained in the cluster while the previously fenced node is reentering the cluster. As stated earlier, recovery involves coordination from fencing, lock management, and the file system.

1.3.4. Cluster Configuration Management

Cluster configuration management provides a centralized mechanism for the configuration and maintenance of configuration files throughout the cluster. It provides high-availability access to configuration-state information for all nodes in the cluster.

For information about cluster configuration management refer to Chapter 6 *Creating the Cluster Configuration System Files* and Chapter 7 *Using the Cluster Configuration System*.

1.4. GFS Software Subsystems

GFS consists of the following subsystems:

- Cluster Configuration System (CCS)
- Fence
- Pool
- LOCK_GULM
- LOCK_NOLOCK

Table 1-1 summarizes the GFS Software subsystems and their components.

Software Subsystem	Components	Description
Cluster Configuration System (CCS)	`ccs_tool`	Command used to create CCS archives.
	`ccs_read`	Diagnostic and testing command that is used to retrieve information from configuration files through `ccsd`.
	`ccsd`	CCS daemon that runs on all cluster nodes and provides configuration file data to cluster software.
	`ccs_servd`	CCS server daemon that distributes CCS data from a single server to ccsd daemons when a shared device is not used for storing CCS data.
Fence	`fence_node`	Command used by `lock_gulmd` when a fence operation is required. This command takes the name of a node and fences it based on the node's fencing configuration.
	`fence_apc`	Fence agent for APC power switch.
	`fence_wti`	Fence agent for WTI power switch.
	`fence_brocade`	Fence agent for Brocade Fibre Channel switch.
	`fence_mcdata`	Fence agent for McData Fibre Channel switch.
	`fence_vixel`	Fence agent for Vixel Fibre Channel switch.
	`fence_rib`	Fence agent for RIB card.
	`fence_gnbd`	Fence agent used with GNBD storage.
	`fence_egenera`	Fence agent used with Egenera BladeFrame system.
	`fence_xcat`	Fence agent used with xCAT-managed cluster.

Software Subsystem	Components	Description
	`fence_manual`	Fence agent for manual interaction. *WARNING:* Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.
	`fence_ack_manual`	User interface for `fence_manual` agent.
Pool	`pool.o`	Kernel module implementing the pool block-device driver.
	`pool_assemble`	Command that activates and deactivates pool volumes.
	`pool_tool`	Command that configures pool volumes from individual storage devices.
	`pool_info`	Command that reports information about system pools.
	`pool_grow`	Command that expands a pool volume.
	`pool_mp`	Command that manages pool multipathing.
LOCK_GULM	`lock_gulm.o`	Kernel module that is installed on GFS nodes using the LOCK_GULM lock module.
	`lock_gulmd`	Server/daemon that runs on each node and communicates with all nodes in GFS cluster.
	`gulm_tool`	Command that configures and debugs the `lock_gulmd` server.
LOCK_NOLOCK	`lock_nolock.o`	Kernel module installed on a node using GFS as a local file system.
GFS	`gfs.o`	Kernel module that implements the GFS file system and is loaded on GFS cluster nodes.
	`lock_harness.o`	Kernel module that implements the GFS lock harness into which GFS lock modules can plug.
	`gfs_mkfs`	Command that creates a GFS file system on a storage device.
	`gfs_tool`	Command that configures or tunes a GFS file system. This command can also gather a variety of information about the file system.
	`gfs_quota`	Command that manages quotas on a mounted GFS file system.
	`gfs_grow`	Command that grows a mounted GFS file system.
	`gfs_jadd`	Command that adds journals to a mounted GFS file system.

Software Subsystem	Components	Description
	`gfs_fsck`	Command that repairs an unmounted GFS file system.
GNBD	`gnbd.o`	Kernel module that implements the GNBD device driver on clients.
	`gnbd_serv.o`	Kernel module that implements the GNBD server. It allows a node to export local storage over the network.
	`gnbd_export`	Command to create, export and manage GNBDs on a GNBD server.
	`gnbd_import`	Command to import and manage GNBDs on a GNBD client.
Upgrade	`gfs_conf`	Command that retrieves from a cidev configuration information from earlier versions of GFS.

Table 1-1. GFS Software Subsystem Components

1.5. Before Configuring GFS

Before you install and configure GFS, note the following key characteristics of your GFS configuration:

Cluster name

Determine a cluster name for your GFS cluster. The cluster name is required in the form of a parameter variable, *ClusterName*, later in this book.The cluster name can be 1 to 16 characters long. For example, this book uses a cluster name *alpha* in some example configuration procedures.

Number of file systems

Determine how many GFS file systems to create initially. (More file systems can be added later.)

File system name

Determine a unique name for each file system. Each file system name is required in the form of a parameter variable, *FSName*, later in this book. For example, this book uses file system names `gfs1` and `gfs2` in some example configuration procedures.

Number of nodes

Determine how many nodes will mount the file systems. Note the hostname and IP address of each node for use in configuration files later.

LOCK_GULM servers

Determine the number of LOCK_GULM servers. Multiple LOCK_GULM servers (available with RLM) provide redundancy. RLM requires a minimum of three nodes, but no more than five nodes. Information about LOCK_GULM servers is required for a CCS (Cluster Configuration System) file, `cluster.ccs`. Refer to Section 6.5 *Creating the `cluster.ccs` File* for information about the `cluster.ccs` file.

GNBD server nodes

> If you are using GNBD, determine how many GNBD server nodes are needed. Note the hostname and IP address of each GNBD server node for use in configuration files later.

Fencing method

> Determine the fencing method for each GFS node. If you are using GNBD multipath, determine the fencing method for each GNBD server node (node that exports GNBDs to GFS nodes). Information about fencing methods is required later in this book for the CCS files, `fence.ccs` and `nodes.ccs`. (Refer to Section 6.6 *Creating the `fence.ccs` File* and Section 6.7 *Creating the `nodes.ccs` File* for more information.) To help determine the type of fencing methods available with GFS, refer to Chapter 10 *Using the Fencing System*. When using RLM, you must use a fencing method that shuts down and reboots the node being fenced.

Storage devices and partitions

> Determine the storage devices and partitions to be used for creating pool volumes in the file systems. Make sure to account for space on one or more partitions for storing cluster configuration information as follows: 2 KB per GFS node or 2 MB total, whichever is larger.

Chapter 2.

System Requirements

This chapter describes the system requirements for Red Hat GFS 6.0 and consists of the following sections:

- Section 2.1 *Platform Requirements*
- Section 2.2 *TCP/IP Network*
- Section 2.3 *Fibre Channel Storage Network*
- Section 2.4 *Fibre Channel Storage Devices*
- Section 2.5 *Network Power Switches*
- Section 2.6 *Console Access*
- Section 2.7 *I/O Fencing*

2.1. Platform Requirements

Table 2-1 shows the platform requirements for GFS.

Operating System	Hardware Architecture	RAM
Red Hat Enterprise Linux AS, ES, or WS, Version 3 Update 2 or later	ia64, x86-64, x86 SMP supported	256 MB, minimum

Table 2-1. Platform Requirements

2.2. TCP/IP Network

All GFS nodes must be connected to a TCP/IP network. Network communication is critical to the operation of the GFS cluster, specifically to the clustering and locking subsystems. For optimal performance and security, it is recommended that a private, dedicated, switched network be used for GFS. GFS subsystems do not use dual-network interfaces for failover purposes.

2.3. Fibre Channel Storage Network

Table 2-2 shows requirements for GFS nodes that are to be connected to a Fibre Channel SAN.

Requirement	Description
HBA (Host Bus Adapter)	One HBA minimum per GFS node
Connection method	Fibre Channel switch *Note:* If an FC switch is used for I/O fencing nodes, you may want to consider using Brocade, McData, or Vixel FC switches, for which GFS fencing agents exist. *Note:* When a small number of nodes is used, it may be possible to connect the nodes directly to ports on the storage device. *Note:* FC drivers may not work reliably with FC hubs.

Table 2-2. Fibre Channel Network Requirements

2.4. Fibre Channel Storage Devices

Table 2-3 shows requirements for Fibre Channel devices that are to be connected to a GFS cluster.

Requirement	Description
Device Type	FC RAID array or JBOD *Note:* Make sure that the devices can operate reliably when heavily accessed simultaneously from multiple initiators. *Note:* Make sure that your GFS configuration does not exceed the number of nodes an array or JBOD supports.
Size	2 TB maximum per GFS file system. Linux 2.4 kernels do not support devices larger than 2 TB; that limits the size of any GFS file system to 2 TB.

Table 2-3. Fibre Channel Storage Device Requirements

2.5. Network Power Switches

GFS provides fencing agents for APC and WTI network power switches.

2.6. Console Access

Make sure that you have console access to each GFS node. Console access to each node ensures that you can monitor the cluster and troubleshoot kernel problems.

2.7. I/O Fencing

You need to configure each node in your GFS cluster for at least one form of I/O fencing. For more information about fencing options, refer to Section 10.2 *Fencing Methods*.

Chapter 3.

Installing GFS

This chapter describes how to install GFS and includes the following sections:

- Section 3.1 *Prerequisite Tasks*
- Section 3.2 *Installation Tasks*
  - Section 3.2.1 *Installing GFS RPMs*
  - Section 3.2.2 *Loading the GFS Kernel Modules*

Note

For information about installing and using GFS with Red Hat Cluster Suite, refer to Appendix A *Using Red Hat GFS with Red Hat Cluster Suite*.

3.1. Prerequisite Tasks

Before installing GFS software, make sure that you have noted the key characteristics of your GFS configuration (refer to Section 1.5 *Before Configuring GFS*) and have completed the following tasks:

- Installed prerequisite software
- Specified a persistent major number (optional)

3.1.1. Prerequisite Software

Make sure that you have installed the following software:

- `perl-Net-Telnet` module
- Clock synchronization software
- `Stunnel` utility (optional)

3.1.1.1. `perl-Net-Telnet` Module

The `perl-Net-Telnet` module is used by several fencing agents and should be installed on all GFS nodes. The `perl-Net-Telnet` module should be installed before installing GFS; otherwise, GFS will not install.

You can install the `perl-Net-Telnet` module from the Red Hat GFS ISO.

3.1.1.2. Clock Synchronization Software

Make sure that each GFS node is running clock synchronization software. The system clocks in GFS nodes need to be within a few minutes of each other to prevent unnecessary inode time-stamp updates. If the node clocks are not synchronized, the inode time stamps will be updated unnecessarily, severely impacting cluster performance. Refer to Section 9.9 *Configuring `atime` Updates* for additional information.

Note

> One example of time synchronization software is the Network Time Protocol (NTP) software. You can find more information about NTP at http://www.ntp.org.

3.1.1.3. `Stunnel` Utility

The `Stunnel` utility needs to be installed only on nodes that use the HP RILOE PCI card for I/O fencing. (For more information about fencing with the HP RILOE card, refer to HP RILOE Card on page 147.) Verify that the utility is installed on each of those nodes by looking for `/usr/sbin/stunnel`. The `Stunnel` utility is available via `up2date`.

3.1.2. Specifying a Persistent Major Number

The major number is set dynamically when the `pool.o` module is loaded (either interactively or through an `init.d` script). In earlier releases of GFS, the major number was set to *121* each time `pool.o` was loaded. If you want to specify a persistent major number rather than a dynamic major number, edit the `/etc/modules.conf` file to include the following line, where *`MajorNumber`* is the major number that you want to use:

```
options pool pool_major=MajorNumber
```

For example, to specify a major number of *121*, edit `/etc/modules.conf` to include the following line:

```
options pool pool_major=121
```

You need to edit the `/etc/modules.conf` on each node running `pool`.

3.2. Installation Tasks

To install GFS, perform the following steps:

1. Install GFS RPMs.
2. Load the GFS kernel modules.

3.2.1. Installing GFS RPMs

Installing GFS RPMs consists of acquiring and installing two GFS RPMs: the GFS tools RPM (for example, `GFS-6.0.2.20-1.i686.rpm`) and the GFS kernel-modules RPM (for example, `GFS-modules-smp-6.0.2.20-1.i686.rpm`).

Note

You must install the GFS tools RPM before installing the GFS kernel-modules RPM.

To install GFS RPMs, follow these steps:

1. Acquire the GFS RPMs according to the kernels in the GFS nodes. Copy or download the RPMs to each GFS node.

   Note

   Make sure that you acquire the appropriate GFS kernel-modules RPM for each kernel type. For example, the following GFS kernel-modules RPM is for an SMP or hugemem kernel:

   `GFS-modules-smp-6.0.2.20-1.i686.rpm`

   The GFS tools RPM is common to all kernels.

2. At each node, install the GFS tools RPM using the `rpm -Uvh` command. For example:
   ```
   # rpm -Uvh GFS-6.0.2.20-1.i686.rpm
   ```
3. At each node, install the GFS kernel-modules RPM using the `rpm -Uvh` command. For example:
   ```
   # rpm -Uvh GFS-modules-smp-6.0.2.20-1.i686.rpm
   ```
4. At each node, issue the `rpm -qa` command to check the GFS version as follows:
   ```
   # rpm -qa | grep GFS
   ```
   This step verifies that the GFS software has been installed; it lists the GFS software installed in the previous step.

3.2.2. Loading the GFS Kernel Modules

Once the GFS RPMs have been installed on the GFS nodes, the following GFS kernel modules need to be loaded into the running kernel before GFS can be set up and used:

- `pool.o`
- `lock_harness.o`
- `lock_gulm.o`
- `gfs.o`

Note

The GFS kernel modules must be loaded into a GFS node each time the node is started. It is recommended that you use the `init.d` scripts included with GFS to automate loading the GFS kernel modules. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS `init.d` Scripts*.

Note

The procedures in this section are for a GFS configuration that uses LOCK_GULM. If you are using LOCK_NOLOCK, refer to Appendix C *Basic GFS Examples* for information about which GFS kernel modules you should load.

To load the GFS kernel modules, follow these steps:

1. Run `depmod -a`. For example:
   `# depmod -a`

   Note

   Run this only once after RPMs are installed.

2. Load `pool.o` and dependent files as follows:
   `# modprobe pool`

   Note

   To specify a persistent major number, edit `/etc/modules.conf` before loading `pool.o`. Refer to Section 3.1.2 *Specifying a Persistent Major Number*

3. Load `lock_gulm.o` and dependent files as follows:
   `# modprobe lock_gulm`

4. Load `gfs.o` and dependent files as follows:
   `# modprobe gfs`

Chapter 4.

Initial Configuration

This chapter describes procedures for initial configuration of GFS and contains the following sections:

- Section 4.1 *Prerequisite Tasks*
- Section 4.2 *Initial Configuration Tasks*
  - Section 4.2.1 *Setting Up Logical Devices*
  - Section 4.2.2 *Setting Up and Starting the Cluster Configuration System*
  - Section 4.2.3 *Starting Clustering and Locking Systems*
  - Section 4.2.4 *Setting Up and Mounting File Systems*

Note

If you are using GFS with Red Hat Cluster, you can configure GFS with **GFS Druid**. For information about configuring and using GFS with Red Hat Cluster Suite, refer to Appendix A *Using Red Hat GFS with Red Hat Cluster Suite*.

4.1. Prerequisite Tasks

Before setting up the GFS software, make sure that you have noted the key characteristics of your GFS configuration (refer to Section 1.5 *Before Configuring GFS*) and have loaded the GFS modules into each GFS node (refer to Chapter 3 *Installing GFS*). In addition, if you are using GNBD multipath, make sure that you understand GNBD multipath considerations (refer to Section 6.4 *GNBD Multipath Considerations for CCS Files* and Section 11.2 *Considerations for Using GNBD Multipath*).

4.2. Initial Configuration Tasks

Initial configuration consists of the following tasks:

1. Setting up logical devices (pools).
2. Setting up and starting the Cluster Configuration System (CCS).
3. Starting clustering and locking systems.
4. Setting up and mounting file systems.

Note

GFS kernel modules must be loaded prior to performing initial configuration tasks. Refer to Section 3.2.2 *Loading the GFS Kernel Modules*.

Note

For examples of GFS configurations, refer to Appendix C *Basic GFS Examples*.

The following sections describe the initial configuration tasks.

4.2.1. Setting Up Logical Devices

To set up logical devices (pools) follow these steps:

1. Create file system pools.
   a. Create pool configuration files. Refer to Section 5.4 *Creating a Configuration File for a New Volume*.
   b. Create a pool for each file system. Refer to Section 5.5 *Creating a Pool Volume*.

      Command usage:

      ```
      pool_tool -c ConfigFile
      ```

2. Create a CCS pool.
   a. Create pool configuration file. Refer to Section 5.4 *Creating a Configuration File for a New Volume*.
   b. Create a pool to be the Cluster Configuration Archive (CCA) device. Refer to Section 5.5 *Creating a Pool Volume*.

      Command usage:

      ```
      pool_tool -c ConfigFile
      ```

3. At each node, activate pools. Refer to Section 5.6 *Activating/Deactivating a Pool Volume*.

   Command usage:

   ```
   pool_assemble
   ```

Note

You can use GFS `init.d` scripts included with GFS to automate activating and deactivating pools. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS init.d Scripts*.

4.2.2. Setting Up and Starting the Cluster Configuration System

To set up and start the Cluster Configuration System, follow these steps:

1. Create CCS configuration files and place them into a temporary directory. Refer to Chapter 6 *Creating the Cluster Configuration System Files*.
2. Create a CCS archive on the CCA device. (The CCA device is the pool created in Step 2 of Section 4.2.1 *Setting Up Logical Devices*.) Put the CCS files (created in Step 1) into the CCS archive. Refer to Section 7.1 *Creating a CCS Archive*.

   Command usage:

   `ccs_tool create` *`Directory CCADevice`*.
3. At each node, start the CCS daemon, specifying the CCA device at the command line. Refer to Section 7.2 *Starting CCS in the Cluster*.

   Command usage:

   `ccsd -d` *`CCADevice`*

Note

You can use GFS `init.d` scripts included with GFS to automate starting and stopping the Cluster Configuration System. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS `init.d` Scripts*.

4.2.3. Starting Clustering and Locking Systems

To start clustering and locking systems, start `lock_gulmd` at each node. Refer to Section 8.2.3 *Starting LOCK_GULM Servers*.

Command usage:

`lock_gulmd`

Note

You can use GFS `init.d` scripts included with GFS to automate starting and stopping `lock_gulmd`. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS `init.d` Scripts*.

4.2.4. Setting Up and Mounting File Systems

To set up and mount file systems, follow these steps:

1. Create GFS file systems on pools created in Step 1 of Section 4.2.2 *Setting Up and Starting the Cluster Configuration System*. Choose a unique name for each file system. Refer to Section 9.1 *Making a File System*.

   Command usage:

```
gfs_mkfs -p lock_gulm -t ClusterName:FSName -j NumberJournals
BlockDevice
```

2. At each node, mount the GFS file systems. Refer to Section 9.2 *Mounting a File System*.

   Command usage:

   ```
   mount -t gfs BlockDevice MountPoint
   ```

   ```
   mount -t gfs -o acl BlockDevice MountPoint
   ```

   The `-o acl` mount option allows manipulating file ACLs. If a file system is mounted without the `-o acl` mount option, users are allowed to view ACLs (with `getfacl`), but are not allowed to set them (with `setfacl`).

Note

You can use GFS `init.d` scripts included with GFS to automate mounting and unmounting GFS file systems. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS `init.d` Scripts*.

Chapter 5.

Using the Pool Volume Manager

This chapter describes the GFS volume manager — named *Pool* — and its commands. The chapter consists of the following sections:

- Section 5.1 *Overview of GFS Pool Volume Manager*
- Section 5.2 *Synopsis of Pool Management Commands*
- Section 5.4 *Creating a Configuration File for a New Volume*
- Section 5.3 *Scanning Block Devices*
- Section 5.5 *Creating a Pool Volume*
- Section 5.6 *Activating/Deactivating a Pool Volume*
- Section 5.7 *Displaying Pool Configuration Information*
- Section 5.8 *Growing a Pool Volume*
- Section 5.9 *Erasing a Pool Volume*
- Section 5.10 *Renaming a Pool Volume*
- Section 5.11 *Changing a Pool Volume Minor Number*
- Section 5.12 *Displaying Pool Volume Information*
- Section 5.13 *Using Pool Volume Statistics*
- Section 5.14 *Adjusting Pool Volume Multipathing*

5.1. Overview of GFS Pool Volume Manager

Pool is a GFS software subsystem that presents physical storage devices (such as disks or RAID arrays) as logical volumes to GFS cluster nodes. Pool can aggregate storage devices either by concatenating the underlying storage or by striping the storage using RAID 0. Pool is a cluster-wide volume manager, presenting logical volumes to each GFS node as if the storage were attached directly to each node. Because Pool is a cluster-wide volume manager, changes made to a volume by one GFS node are visible to all other GFS nodes in a cluster.

Pool is a dynamically loadable kernel module, `pool.o`. When `pool.o` is loaded, it gets registered as a Linux kernel block-device driver. Before pool devices can be used, this driver module must be loaded into the kernel. (Once the driver module is loaded, the `pool_assemble` command can be run to activate pools.)

Pool includes a set of user commands that can be executed to configure and manage specific pool devices. Those commands are summarized in the next section.

More advanced, special-purpose features of the Pool volume manager are described later in this chapter.

5.2. Synopsis of Pool Management Commands

Four commands are available to manage pools:

- `pool_tool`
- `pool_assemble`
- `pool_info`
- `pool_mp`

The following sections briefly describe the commands and provide references to other sections in this chapter, where more detailed information about the commands and their use is described.

5.2.1. `pool_tool`

The `pool_tool` command provides a variety of functions for manipulating and controlling pools (refer to Table 5-1 and Table 5-2). Some `pool_tool` functions — such as creating a pool and growing a pool — require that a file be specified on the command line to provide inputs for the command. Other `pool_tool` functions — such as erasing a pool, renaming a pool, changing a minor number, and printing a pool configuration — act on existing pools and require one or more pool names to be specified on the command line.

Flag	Function	Section/Page Reference
`-c`	Create a pool	Section 5.5 *Creating a Pool Volume*
`-e`	Erase a pool	Section 5.9 *Erasing a Pool Volume*
`-s`	Scan devices	Section 5.3 *Scanning Block Devices*
`-g`	Grow a pool	Section 5.8 *Growing a Pool Volume* *Note:* The `pool_tool -g` command supersedes the `pool_grow` command as of GFS 5.2. Although the `pool_grow` command is still available, it is not supported in GFS 5.2 and later.
`-r`	Rename a pool	Section 5.10 *Renaming a Pool Volume* *Note:* In releases before GFS 5.2, the `-r` flag had a different usage.
`-p`	Print a pool configuration	Section 5.7 *Displaying Pool Configuration Information*
`-m`	Change a pool minor number	Section 5.11 *Changing a Pool Volume Minor Number*

Table 5-1. `pool_tool` Command Functions

Flag	Option
`-D`	Enable debugging output.
`-h`	Help. Show usage information.
`-O`	Override prompts.

Flag	Option
`-q`	Be quiet. Do not display output from the command.
`-V`	Display command version information, then exit.
`-v`	Verbose operation.

Table 5-2. `pool_tool` Command Options

5.2.2. `pool_assemble`

The `pool_assemble` command activates and deactivates pools on a system (refer to Table 5-3 and Table 5-4). One or more pool names can be specified on the command line, indicating the pools to be activated or deactivated. If no pools are specified on the command line, all pools will be acted upon.

Flag	Function	Section/Page Reference
`-a`	Activate pool(s)	Section 5.6 *Activating/Deactivating a Pool Volume*
`-r`	Deactivate pool(s)	Section 5.6 *Activating/Deactivating a Pool Volume*

Table 5-3. `pool_assemble` Command Functions

Flag	Option
`-D`	Enable debugging output.
`-h`	Help. Show usage information.
`-q`	Be quiet. Do not display output from the command.
`-V`	Display command version information, then exit.
`-v`	Verbose operation.

Table 5-4. `pool_assemble` Command Options

5.2.3. `pool_info`

The `pool_info` command scans disks directly and displays information about activated pools in a system; that is, pools that have been assembled with the `pool_assemble` command (refer to Table 5-5 and Table 5-6). Information about pools that are present but not assembled is excluded from the information displayed with a `pool_info` command. One or more pool names can be specified on the command line to select information about specific pools. If no pool name is specified, information on all pools is returned.

Flag	Function	Section/Page Reference
`-c`	Clear statistics	Section 5.13 *Using Pool Volume Statistics*

Flag	Function	Section/Page Reference
-i	Display information	Section 5.12 *Displaying Pool Volume Information*
-s	Display statistics	Section 5.13 *Using Pool Volume Statistics*
-p	Display an active configuration	Section 5.7 *Displaying Pool Configuration Information*

Table 5-5. `pool_info` Command Functions

Flag	Option
-D	Enable debugging output.
-H	Show capacity in human readable form.
-h	Help. Show usage information.
-q	Be quiet. Do not display output from the command.
-t	Set time interval for continual statistics updates.
-V	Display command version information, then exit.
-v	Verbose operation.

Table 5-6. `pool_info` Command Options

5.2.4. `pool_mp`

The `pool_mp` command is for managing multipathing on running pools (refer to Table 5-7 and Table 5-8). One or more pool names can be specified on the command line to adjust multipathing on specific pools. If no pools are specified on the command line, all pools will be acted upon.

Flag	Function	Section/Page Reference
-m	Tune multipathing	Section 5.14 *Adjusting Pool Volume Multipathing*
-r	Restore failed paths	Section 5.14 *Adjusting Pool Volume Multipathing*

Table 5-7. `pool_mp` Command Functions

Flag	Option
-D	Enable debugging output.
-h	Help. Show usage information.
-q	Be quiet. Do not display output from the command.
-V	Display command version information, then exit.

Flag	Option
`-v`	Verbose operation.

Table 5-8. `pool_mp` Command Options

5.3. Scanning Block Devices

Scanning block devices provides information about the availability and characteristics of the devices. That information is important for creating a pool configuration file. You can scan block devices by issuing the `pool_tool` command with the `-s` option. Issuing the `pool_tool` command with the `-s` option scans all visible block devices and reports whether they have an Ext2 or Ext3 file system, LVM version 1 labels, a partition table, a pool label, or an unknown label on them.

Note

The `pool_tool -s` command does not detect ondisk labels other than those mentioned in the preceding paragraph.

5.3.1. Usage

```
pool_tool -s
```

5.3.2. Example

In this example, the response to the command displays information about one GFS file system, other file systems that have no labels, and a local file system.

```
# pool_tool -s
  Device                                            Pool Label
  ======                                            ==========
  /dev/pool/stripe-128K                   <- GFS filesystem ->
  /dev/sda                          <- partition information ->
  /dev/sda1                                        stripe-128K
  /dev/sda2                                        stripe-128K
  /dev/sda3                                        stripe-128K
  /dev/sda4                         <- partition information ->
  /dev/sda5                                     <- unknown ->
  /dev/sda6                                     <- unknown ->
  /dev/sda7                                     <- unknown ->
  /dev/sda8                                     <- unknown ->
  /dev/sdb                          <- partition information ->
  /dev/sdb1                                     <- unknown ->
  /dev/sdb2                                     <- unknown ->
  /dev/sdb3                                     <- unknown ->
      .                                              .
      .                                              .
      .                                              .
  /dev/sdd4                         <- partition information ->
  /dev/sdd5                                     <- unknown ->
  /dev/sdd6                                     <- unknown ->
  /dev/sdd7                                     <- unknown ->
```

```
/dev/sdd8                                    <- unknown ->
/dev/hda                         <- partition information ->
/dev/hda1                            <- EXT2/3 filesystem ->
/dev/hda2                                  <- swap device ->
/dev/hda3                            <- EXT2/3 filesystem ->
```

5.4. Creating a Configuration File for a New Volume

A pool configuration file is used as input to the `pool_tool` command when creating or growing a pool volume. The configuration file defines the name and layout of a single pool volume. Refer to Figure 5-1 for the pool configuration file format. Refer to Table 5-9 for descriptions of the configuration file keywords and variables.

An arbitrary name can be used for a pool configuration file; however, for consistency, it is recommended that you name the file using the pool name followed by the `.cfg` extension (*poolname*`.cfg`). For example, the pool configuration file for a pool named *pool0* would be defined in configuration file `pool0.cfg`.

Before creating a configuration file, you can check to see what devices are available by using the `pool_tool` command with the `-s` option.

```
poolname name
minor number
subpools number
subpool id stripe devices [type]
pooldevice subpool id device
```

Figure 5-1. File Structure: Pool Configuration File

File Line and Keyword	Variable	Description
poolname	*name*	The name of the pool device that appears in the `/dev/pool/` directory.
minor	*number*	Assigns a device minor number (0 to 64) to a pool. If number is specified as 0 (or if the minor line is omitted), the minor number of the pool is assigned dynamically. The default value is 0.
subpools	*number*	Represents the total number of subpools in the pool. The *number* value should be set to a value of 1 unless special data or journal subpools are used.

File Line and Keyword	Variable	Description
subpool	*id stripe devices* [*type*]	The details of each subpool: *id* is the subpool identifier. Number the subpools in order beginning with 0. *stripe* is the stripe size in sectors (512 bytes per sector) for each device. A value of 0 specifies no striping (concatenation). *devices* specifies the number of devices in the subpool. *type* is optional and specifies the label type to attach to the subpool. Values of `gfs_data` or `gfs_journal` are acceptable. The default value is `gfs_data`.
pooldevice	*subpool id device*	Adds a storage device to a subpool: *subpool* specifies the subpool identifier to which the device is to be added. *id* is the device identifier. Number the devices in order beginning with 0. *device* specifies the device node to be used (for example, `/dev/sda1`).

Table 5-9. Pool Configuration File Keyword and Variable Descriptions

5.4.1. Examples

This example creates a 4-disk pool named pool0 that has a stripe size of 64K and an assigned minor number of 1:

```
poolname pool0
minor 1 subpools 1
subpool 0 128 4 gfs_data
pooldevice 0 0 /dev/sdb1
pooldevice 0 1 /dev/sdc1
pooldevice 0 2 /dev/sdd1
pooldevice 0 3 /dev/sde1
```

This example creates a 4-disk pool named pool1 that has a dynamic minor number composed of a striped subpool and a concatenated subpool:

```
poolname pool1
minor 0
subpools 2
# striped subpool
subpool 0 128 2 gfs_data
# concatenated subpool
subpool 1 0 2 gfs_data
pooldevice 0 0 /dev/sdb1
pooldevice 0 1 /dev/sdc1

pooldevice 1 0 /dev/sdd1
pooldevice 1 1 /dev/sde1
```

5.5. Creating a Pool Volume

Once a configuration file is created or edited (refer to Section 5.4 *Creating a Configuration File for a New Volume*), a pool volume can be created using the `pool_tool` command. Because the `pool_tool` command writes labels to the beginning of the devices or partitions, the new pool volume's devices or partitions must be accessible to the system. To create a pool, run the `pool_tool` command once from a single node.

Note

The pool can be activated on any node by running the `pool_assemble` command. Refer to Section 5.6 *Activating/Deactivating a Pool Volume*.

5.5.1. Usage

`pool_tool -c` *`ConfigFile`*

`ConfigFile`

Specifies the file that defines the pool.

5.5.2. Example

In this example, the `pool0.cfg` file describes the new pool, *`pool0`*, created by the command.

`pool_tool -c pool0.cfg`

5.5.3. Comments

Multiple pools can be created with one `pool_tool` command by listing multiple pool configuration files on the command line.

If no flag is specified in the `pool_tool` command, the function defaults to creating a pool (`-c`), with the configuration file specified after the command.

5.6. Activating/Deactivating a Pool Volume

The `pool_assemble` command activates or deactivates pools on a node.

Note

The `pool_assemble` command must be run on every node that accesses shared pools; also, it must be run each time a node reboots. You can use the `pool init.d` script included with GFS to automatically run the `pool_assemble` command each time a node reboots. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS `init.d` Scripts*.

The `pool_assemble` command only activates pools from devices visible to the node (those listed in `/proc/partitions/`). Disk labels created by the `pool_tool` command are scanned to determine which pools exist and, as a result, should be activated. The `pool_assemble` command also creates device nodes in the `/dev/pool/` directory for devices it has activated.

5.6.1. Usage

Activating a Pool Volume

```
pool_assemble -a [PoolName]
```

PoolName

Specifies the pool to activate. More than one name can be listed. If no pool names are specified, all pools visible to the system are activated.

Deactivating a Pool Volume

```
pool_assemble -r [PoolName]
```

PoolName

Specifies the pool to deactivate. More than one name can be listed. If no pool names are specified, all pools visible to the system are deactivated.

5.6.2. Examples

This example activates all pools on a node:

`pool_assemble -a`

This example deactivates all pools on a node:

`pool_assemble -r`

This example activates `pool0` on a node:

`pool_assemble -a pool0`

This example deactivates `pool0` on a node:

`pool_assemble -r pool0`

5.6.3. Comments

The `pool_assemble` command must be run on every GFS node.

The `pool_assemble` command should be put into the node's system-startup scripts so that pools are activated each time the node boots.

Note

You can use GFS `init.d` scripts included with GFS to automate activating and deactivating pools. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS `init.d` Scripts*.

5.7. Displaying Pool Configuration Information

Using the `pool_tool` command with the `-p` (print) option displays pool configuration information in configuration file format. The pool information is displayed in the format equivalent to the configuration file that was used to create the pool. The disk labels that were written when the pool was created are read to recreate the configuration file.

5.7.1. Usage

```
pool_tool -p [PoolName]
```

PoolName

Specifies the pool name(s) for which to display information. If no pool names are specified, all active pools are displayed.

5.7.2. Example

In this example, the `pool_tool -p` command displays the configuration for `pool0`:

```
# pool_tool -p pool0
poolname pool0
#minor <dynamically assigned>
subpools 1
subpool 0 0 1 gfs_data
pooldevice 0 0 /dev/sda1
```

5.8. Growing a Pool Volume

An existing pool can be expanded while it is activated or deactivated. You can grow a pool by creating a new pool configuration file (based on an existing pool configuration file), then adding one or more subpools containing the new devices to be added to the volume.

Refer to Section 5.7 *Displaying Pool Configuration Information* for information on creating a configuration file for an existing pool volume.

5.8.1. Usage

```
pool_tool -g [ConfigFile]
```

ConfigFile

Specifies the file describing the extended pool.

Note

The pool_tool -g command supersedes the pool_grow command as of GFS 5.2. Although the pool_grow command is still available, it is not supported in GFS 5.2 and later.

5.8.2. Example procedure

The following example procedure expands a pool volume.

1. Create a new configuration file from configuration information for the pool volume that you want to expand (in this example, *pool0*):

```
# pool_tool -p pool0 > pool0-new.cfg
# cat pool0-new.cfg
poolname pool0
subpools 1
subpool 0 128 4 gfs_data
pooldevice 0 0 /dev/sdb1
pooldevice 0 1 /dev/sdc1
pooldevice 0 2 /dev/sdd1
pooldevice 0 3 /dev/sde1
```

2. Edit the new file, pool0-new.cfg, by adding one or more subpools that contain the devices or partitions, as indicated in this example:

```
poolname pool0
subpools 2                    <--- Change
subpool 0 128 4 gfs_data
subpool 1 0 1 gfs_data        <--- Add
pooldevice 0 0 /dev/sdb1
pooldevice 0 1 /dev/sdc1
pooldevice 0 2 /dev/sdd1
pooldevice 0 3 /dev/sde1
pooldevice 1 0 /dev/sdf1      <--- Add
```

3. After saving the file, verify that the file has been changed:

```
# cat pool0-new.cfg
poolname pool0
subpools 2                    <--- Changed
subpool 0 128 4 gfs_data
subpool 1 0 1 gfs_data        <--- Added
pooldevice 0 0 /dev/sdb1
pooldevice 0 1 /dev/sdc1
pooldevice 0 2 /dev/sdd1
pooldevice 0 3 /dev/sde1
pooldevice 1 0 /dev/sdf1      <--- Added
```

4. Run the pool_tool command with the grow (-g) option specifying the configuration file:

```
pool_tool -g pool0-new.cfg
```

5.9. Erasing a Pool Volume

A deactivated pool can be erased by using the `-e` option of the `pool_tool` command. Using `pool_tool -e` erases the disk labels written when the pool was created.

5.9.1. Usage

```
pool_tool -e [PoolName]
```

PoolName

Specifies the pool to erase. If no pool names are specified, all pools are erased.

5.9.2. Example

This example erases all disk labels for *pool0*:

`pool_tool -e pool0`

5.9.3. Comments

The `-O` (override) flag bypasses the confirmation step.

5.10. Renaming a Pool Volume

The `pool_tool` command can be used to change the name of a pool.

5.10.1. Usage

```
pool_tool -r NewPoolName CurrentPoolName
```

NewPoolName

Specifies the new name of the pool.

CurrentPoolName

Specifies the pool name to be changed.

Note

In releases before GFS 5.2, the `-r` flag had a different usage.

Note

You must deactivate a pool before renaming it. You can deactivate a pool with the `pool_assemble -r` *`PoolName`* command or by using the `pool init.d` script. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS `init.d` Scripts*.

5.10.2. Example

This example changes the name for pool *`mypool`* to *`pool0`*:

```
pool_tool -r pool0 mypool
```

5.11. Changing a Pool Volume Minor Number

The `pool_tool` command can be used to change the minor number of a pool.

5.11.1. Usage

```
pool_tool -m Number PoolName
```

`Number`

Specifies the new minor number to be used.

`PoolName`

Specifies the name of the pool to be changed. The minor number must have a value between 0 and 64. Specifying a minor number of 0 dynamically selects an actual minor number between 65 and 127 at activation time.

Note

You must deactivate a pool before changing its pool volume minor number. You can deactivate a pool with the `pool_assemble -r` *`PoolName`* command or by using the `pool init.d` script. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS `init.d` Scripts*.

5.11.2. Example

This example changes the minor number for *`pool0`* to 6.

```
pool_tool -m 6 pool0
```

5.11.3. Comments

Before changing a pool volume minor number, deactivate the pool. For this command to take effect throughout the cluster, you must reload the pools on each node in the cluster by issuing a `pool_assemble -r` *PoolName* command followed by a `pool_assemble -a` *PoolName* command.

Note

You can use GFS `init.d` scripts included with GFS to automate activating and deactivating pools. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS init.d Scripts*.

5.12. Displaying Pool Volume Information

The `pool_info` command can be used to display information about pools.

Using the `pool_info` command with the `-i` option displays the following basic information about the named pool(s): the pool name, the minor number, the device node alias, the capacity, whether or not the pool is being used, and the multipathing type.

Using the `pool_info` command with the `-v` (verbose) option displays complete information about the named pools, which adds subpool and device details to the output display.

5.12.1. Usage

Basic Display

```
pool_info -i [PoolName]
```

PoolName

Specifies the pool name(s) for which to display information. If no pool names are specified, all active pools are displayed.

Complete Display

```
pool_info -v [PoolName]
```

PoolName

Specifies the pool name(s) for which to display information. If no pool names are specified, all active pools are displayed.

5.12.2. Examples

This example displays basic information about all activated pools:

```
pool_info -i
```

This example displays complete information about all activated pools:

pool_info -v

This example displays complete information about *pool0*:

pool_info -v pool0

5.13. Using Pool Volume Statistics

The `pool_info` command can be used to display pool read-write information and to clear statistics from pools.

Using the `pool_info` command with the `-s` option displays the number of reads and writes for the named pool(s) since the last time the pool was activated or statistics were cleared.

Using the `pool_info` command with the `-c` option clears statistics from the named pools.

5.13.1. Usage

Display the Number of Reads and Writes

```
pool_info -s [PoolName]
```

PoolName

Specifies the pool name for which to display information. If no pool names are specified, all active pools are displayed.

Clear Statistics

```
pool_info -c [PoolName]
```

PoolName

Specifies the pool name(s) from which statistics are cleared. If no pool names are specified, statistics are cleared from all active pools.

5.13.2. Examples

This example displays statistics for all activated pools:

pool_info -s

This example displays statistics for *pool0*:

pool_info -s pool0

This example clears statistics for *pool0*:

pool_info -c pool0

5.14. Adjusting Pool Volume Multipathing

The `pool_mp` command adjusts multipathing for running pools. Using the `pool_mp` command with the `-m` option, you can change the type of multipathing. Using the `pool_mp` command with the `-r` option, you can reintegrate failed paths.

5.14.1. Usage

Change the Type of Multipathing

```
pool_mp -m {none | failover | n} [PoolName]
```

{none | failover | *n*}

Specifies the type of multipathing to be used: either **none**, **failover**, or the number of kilobytes, *n*, used as a round-robin stripe value.

PoolName

Specifies the pool on which to adjust multipathing. If no pool names are specified, this action is attempted on all active pools.

Reintegrate Failed Paths

```
pool_mp -r [PoolName]
```

PoolName

Specifies the pool on which to attempt restoration of any failed paths. If no pool names are specified, this action is attempted on all active pools.

5.14.2. Examples

This example adjusts the multipathing for all pools to none.

```
pool_mp -m none
```

This example adjusts the multipathing for *pool0* to failover.

```
pool_mp -m failover pool0
```

This example adjusts the multipathing for *pool0* to round-robin with a stripe size of 512 KB.

```
pool_mp -m 512 pool0
```

This example restores failed paths for all active pools.

```
pool_mp -r
```

Chapter 6.

Creating the Cluster Configuration System Files

The GFS Cluster Configuration System (CCS) requires the following files:

- `cluster.ccs` — The cluster file contains the name of the cluster and the names of the nodes where LOCK_GULM servers are run.
- `fence.ccs` — The fence file describes each device used for fencing.
- `nodes.ccs` — The nodes file contains an entry for each GFS node and LOCK_GULM server node. This file specifies the IP address and fencing parameters of each node.

This chapter describes how to create the CCS files and contains the following sections:

- Section 6.1 *Prerequisite Tasks*
- Section 6.2 *CCS File Creation Tasks*
- Section 6.3 *Dual Power and Multipath FC Fencing Considerations*
- Section 6.4 *GNBD Multipath Considerations for CCS Files*
- Section 6.5 *Creating the `cluster.ccs` File*
- Section 6.6 *Creating the `fence.ccs` File*
- Section 6.7 *Creating the `nodes.ccs` File*

Note

If you are using GFS with Red Hat Cluster, you can create CCS files with **GFS Druid**. For information about configuring and using GFS with Red Hat Cluster Suite, refer to Appendix A *Using Red Hat GFS with Red Hat Cluster Suite*.

6.1. Prerequisite Tasks

Before creating the CCS files, make sure that you perform the following prerequisite tasks:

- Choose a cluster name (user variable, *`ClusterName`*).
- Create a temporary directory (for example, `/root/alpha/`) in which to place the new CCS files that are created. Note the temporary directory path; it is used later as a *`Directory`* parameter when the CCS files are written to a CCA (Cluster Configuration Archive) device. For more information, refer to Section 7.1 *Creating a CCS Archive*.
- Identify each node that runs the LOCK_GULM server daemons. Those nodes must have entries in the `nodes.ccs` file. Refer to Section 8.2 *LOCK_GULM*.
- Determine if any GFS node has dual power supplies or multiple paths to FC storage.
- Determine if you are using GNBD multipath.
- Determine the type of fencing for each node.

For more information about prerequisite tasks, refer to Section 1.5 *Before Configuring GFS*

6.2. CCS File Creation Tasks

To create the CCS files perform the following steps:

1. Create the `cluster.ccs` file.
2. Create the `fence.ccs` file.
3. Create the `nodes.ccs` file.

Note

The contents of CCS files are case sensitive.

6.3. Dual Power and Multipath FC Fencing Considerations

To ensure that fencing completely removes a node that has dual power supplies or multiple paths to FC storage, both power supplies and all paths to FC storage for that node must be fenced.

To fence dual power supplies and multiple paths to FC storage, you need to consider the following actions when creating the `fence.ccs` and `nodes.ccs` files:

- `fence.ccs` — For each power supply and each path to FC storage define a fencing device (`fence.ccs:fence_devices/`*`DeviceName`*).
- `nodes.ccs` — For each node with dual power supplies, include in the fencing method section (`nodes.ccs:nodes/`*`NodeName`*`/fence/`*`Methodname`*) a fencing device for each power supply . For each node having multiple paths to FC storage, include in the fencing method section a fencing device for each path to FC storage.

GFS supports dual power-supply fencing with the APC MasterSwitch only; it supports multipath FC fencing with Brocade and Vixel switches. For more information about creating the `fence.ccs` and `nodes.ccs` files, refer to Section 6.6 *Creating the `fence.ccs` File* and Section 6.7 *Creating the `nodes.ccs` File*. For more information about fencing, refer to Chapter 10 *Using the Fencing System*.

6.4. GNBD Multipath Considerations for CCS Files

GNBD multipath allows you to configure multiple GNBD server nodes (nodes that export GNBDs to GFS nodes) with redundant paths between the GNBD server nodes and storage devices. The GNBD server nodes, in turn, present multiple storage paths to GFS nodes (GNBD clients) via redundant GNBDs. With GNBD multipath, if a GNBD server node becomes unavailable, another GNBD server node can provide GFS nodes with access to storage devices.

Make sure to take the following actions when setting up CCS files for GNBD multipath:

- Configure a fencing method that physically removes each GNBD server node from the network.

Warning

Do *not* specify the GNBD fencing agent (`fence_gnbd`) as a fencing device for the GNBD server nodes.

- If you specify `fence_gnbd` as a fence device for a GFS node using GNBD multipath, the `fence.ccs` file must include an *`option = multipath`* parameter (in `fence.ccs:fence_devices/`*`DeviceName`*).

Note

If the GFS node is using another fencing device, the *`option = multipath`* parameter is not needed.

For more information about setting up CCS files for GNBD multipath, refer to Section 6.6 *Creating the `fence.ccs` File* and Section 6.7 *Creating the `nodes.ccs` File*. For more information and additional considerations about using GNBD multipath, refer to Chapter 11 *Using GNBD*. For more information about fencing, refer to Chapter 10 *Using the Fencing System*.

6.5. Creating the `cluster.ccs` File

Creating the `cluster.ccs` file consists of specifying the following parameters:

- Cluster name
- Each node that runs LOCK_GULM server
- Optional parameters

Note

Because of quorum requirements, the number of lock servers allowed in a GFS cluster can be 1, 3, 4, or 5. Any other number of lock servers — that is, 0, 2, or more than 5 — is not supported.

Note

Two optional `cluster.ccs` parameters, `heartbeat_rate` and `allowed_misses`, are included in this procedure for configuring node failure detection. For a description of other optional parameters, refer to the `lock_gulmd(5)` man page.

To create the `cluster.ccs` file, follow these steps:

1. Create a new file named `cluster.ccs` using the file structure shown in Figure 6-1. Refer to Table 6-1 for syntax description.
2. Specify *`ClusterName`* (for example, *`alpha`*). Refer to Example 6-1.
3. Specify each node (*`NodeName`*) that runs LOCK_GULM server (for example, *`n01`*, *`n02`*, and *`n03`*). Refer to Example 6-1.

4. (Optional) For the heartbeat rate (`heartbeat_rate =`), specify *`Seconds`*. Refer to Example 6-1.

   The *`Seconds`* parameter in combination with the `allowed_misses` *`Number`* parameter specify the amount of time for node failure detection as follows:

   `Seconds` x (*`Number`*+1) = Time (in seconds)

5. (Optional) For the allowed consecutively missed heartbeats (`allowed_misses =`), specify *`Number`*. Refer to Example 6-1.

6. Save the `cluster.ccs` file.

```
cluster {
   name = "ClusterName"
   lock_gulm {
       servers = ["NodeName",..., "NodeName"]
              heartbeat_rate = Seconds  <-- Optional
              allowed_misses = Number   <-- Optional
          }
}
```

Figure 6-1. File Structure: `cluster.ccs`

Parameter	Description
`ClusterName`	The name of the cluster, from 1 to 16 characters long.
`NodeName`	The name of each node that runs the LOCK_GULM server. Each node name must appear under `nodes.ccs:nodes`.
`Seconds` (Optional)	For the `heartbeat_rate =` parameter, the rate, in seconds, that a master node *checks* for heartbeats from other nodes. The default value of *`Seconds`* is 15. To ensure that nodes respond within the *`Seconds`* value, the interval for heartbeats *sent* by all nodes is automatically set to two-thirds of the *`Seconds`* parameter value. The *`Seconds`* parameter in combination with the *`Number`* parameter specify the amount of time for node failure detection as follows: *`Seconds`* x (*`Number`*+1) = Time (in seconds). To specify *`Seconds`* as a sub-second value, use floating point notation; however, refer to the following caution for sub-second values and other values less than the default value. *Caution:* If you *must* adjust *`Seconds`* to a different value than the default value, make sure that you understand in detail the characteristics of your cluster hardware and software. Smaller *`Seconds`* values can cause false node expirations under heavy network loads.
`Number` (Optional)	For `allowed_misses`, how many consecutive heartbeats can be missed before a node is marked as expired. The default value of *`Number`* is 2. The *`Seconds`* parameter in combination with the *`Number`* parameter specify the amount of time for node failure detection as follows: *`Seconds`* x (*`Number`*+1) = Time (in seconds).

Table 6-1. File Syntax Description: Variables for `cluster.ccs`

```
cluster {
          name = "alpha"
          lock_gulm {
              servers = ["n01", "n02", "n03"]
              heartbeat_rate = 20
              allowed_misses = 3
          }
}
```

Example 6-1. `cluster.ccs`

6.6. Creating the `fence.ccs` File

You can configure each node in a GFS cluster for a variety of fencing devices. To configure fencing for a node, you need to perform the following tasks:

- Create the `fence.ccs` file — Define the fencing devices available in the cluster (described in this section).
- Create the `nodes.ccs` file — Define which fencing method (or methods) each node should use (refer to Section 6.7 *Creating the `nodes.ccs` File*).

Creating the `fence.ccs` file consists of defining each fencing device you are going to use. You can define the following types of fencing devices in the `fence.ccs` file:

- APC MasterSwitch
- WTI NPS (Network Power Switch)
- Brocade FC (Fibre Channel) switch
- McData FC switch
- Vixel FC switch
- GNBD
- HP RILOE card
- xCAT
- Egenera BladeFrame system
- Manual

Warning

Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.

The `fence.ccs` file is used in conjunction with the `nodes.ccs` file to configure fencing in a cluster. The `nodes.ccs` file specifies fencing devices that are defined in the `fence.ccs` file. The `fence.ccs` file can define any combination of fencing devices.

If a node has dual power supplies, you must define a fencing device for each power supply. Similarly, if a node has multiple paths to FC storage, you must define a fencing device for each path to FC storage.

For more information about fencing, refer to Chapter 10 *Using the Fencing System*.

To create the `fence.ccs` file, follow these steps:

1. Create a file named `fence.ccs`. Use a file format according to the fencing method as follows. Refer to Table 6-2 for syntax description.
   - APC MasterSwitch — Refer to Figure 6-2.
   - WTI NPS (Network Power Switch) — Refer to Figure 6-3.
   - Brocade FC (Fibre Channel) switch — Refer to Figure 6-4.
   - McData FC (Fibre Channel) switch — Refer to Figure 6-5.
   - Vixel FC switch — Refer to Figure 6-6.
   - GNBD — For GNBD *without* GNBD multipath, refer to Figure 6-7. For GNBD *with* GNBD multipath, refer to Figure 6-8.
   - HP RILOE card — Refer to Figure 6-9.
   - xCAT — Refer to Figure 6-10.
   - Egenera BladeFrame system — Refer to Figure 6-11.
   - Manual — Refer to Figure 6-12.

   Warning

   Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.

2. Type parameters in the file according to the fencing device (or devices) needed:
   a. For each APC MasterSwitch fencing device, specify the following parameters: *`DeviceName`*, the fencing agent (`agent =`) as `fence_apc`, *`IPAddress`*, *`LoginName`*, and *`LoginPassword`*. Refer to Example 6-2 for a `fence.ccs` file that specifies an APC MasterSwitch fencing device.
   b. For each WTI NPS fencing device, specify the following parameters: *`DeviceName`*, the fencing agent (`agent =`) as `fence_wti`, *`IPAddress`*, and *`LoginPassword`*. Refer to Example 6-3 for a `fence.ccs` file that specifies a WTI NPS fencing device.
   c. For each Brocade FC-switch fencing device, specify the following parameters: *`DeviceName`*, the fencing agent (`agent =`) as `fence_brocade`, *`IPAddress`*, *`LoginName`*, and *`LoginPassword`*. Refer to Example 6-4 for a `fence.ccs` file that specifies a Brocade FC-switch fencing device.
   d. For each McData FC-switch fencing device, specify the following parameters: *`DeviceName`*, the fencing agent (`agent =`) as `fence_mcdata`, *`IPAddress`*, *`LoginName`*, and *`LoginPassword`*. Refer to Example 6-5 for a `fence.ccs` file that specifies a McData FC-switch fencing device.
   e. For each Vixel FC-switch fencing device, specify the following parameters: *`DeviceName`*, the fencing agent (`agent =`) as `fence_vixel`, *`IPAddress`*, and *`LoginPassword`*. Refer to Example 6-6 for a `fence.ccs` file that specifies a Vixel FC-switch fencing device.
   f. For each GNBD fencing device, specify the following parameters: *`DeviceName`*, the fencing agent (`agent =`) as `fence_gnbd`, and *`ServerName`*.

      For GNBD multipath, include an `option = "multipath"` line after the *`ServerName`* line. In addition, for GNBD multipath, you can add two optional lines: `retrys = "`*`Number`*`"` and `wait_time = "`*`Seconds`*`"`.

Note

Do *not* use `fence_gnbd` to fence GNBD server nodes.

For descriptions of those parameters refer to Table 6-2. Refer to Example 6-7 for a `fence.ccs` file that specifies a GNBD fencing device for a configuration that does not employ GNBD multipath. Refer to Example 6-8 for a `fence.ccs` file that specifies a GNBD fencing device for a configuration that *does* employ GNBD multipath.

g. For each HP-RILOE-card fencing device, specify the following parameters: *`DeviceName`*, the fencing agent (`agent =`) as `fence_rib`, *`HostName`*, *`LoginName`*, and *`LoginPassword`*. Refer to Example 6-9 for a `fence.ccs` file that specifies an HP-RILOE-card fencing device.

h. For each xCAT fencing device, specify the following parameters: *`DeviceName`*, the fencing agent (`agent =`) as `fence_xcat`, and *`RpowerBinaryPath`*. Refer to Example 6-10 for a `fence.ccs` file that specifies an xCAT fencing device.

i. For each Egenera BladeFrame fencing device, specify the following parameters: *`DeviceName`*, the fencing agent (`agent =`) as `fence_egenera`, and *`CserverName`*. Refer to Example 6-11 for a `fence.ccs` file that specifies an Egenera BladeFrame fencing device.

j. For each manual fencing device, specify *`DeviceName`* and the fencing agent (`agent =`) as `fence_manual`. Refer to Example 6-12 for a `fence.ccs` file that specifies a manual fencing device.

Warning

Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.

3. Save the file.

```
fence_devices{
   DeviceName {
      agent = "fence_apc"
      ipaddr = "IPAddress"
      login = "LoginName"
      passwd = "LoginPassword"
   }
   DeviceName {
   .
   .
   .
   }
}
```

Figure 6-2. File Structure: `fence_devices`, `fence_apc`

```
fence_devices{
   DeviceName {
      agent = "fence_wti"
      ipaddr = " IPAddress"
      passwd = " LoginPassword"
   }
   DeviceName {
   .
   .
   .
   }
}
```

Figure 6-3. File Structure: `fence_devices`, `fence_wti`

```
fence_devices{
   DeviceName {
      agent = "fence_brocade"
      ipaddr = "IPAddress"
      login = "LoginName"
      passwd = "LoginPassword"
   }
   DeviceName {
   .
   .
   .
   }
}
```

Figure 6-4. File Structure: `fence_devices`, `fence_brocade`

```
fence_devices{
   DeviceName {
      agent = "fence_mcdata"
      ipaddr = "IPAddress"
      login = "LoginName"
      passwd = "LoginPassword"
   }
   DeviceName {
   .
   .
   .
   }
}
```

Figure 6-5. File Structure: `fence_devices`, `fence_mcdata`

```
fence_devices{
   DeviceName {
      agent = "fence_vixel"
      ipaddr = "IPAddress"
      passwd = "LoginPassword"
   }
   DeviceName {
   .
   .
   .
   }
}
```

Figure 6-6. File Structure: `fence_devices`, `fence_vixel`

```
fence_devices{
   DeviceName {
      agent = "fence_gnbd"
      server = "ServerName"
      .
      .
      .
      server = "ServerName"
   }
   DeviceName {
   .
   .
   .
   }
}
```

Figure 6-7. File Structure: `fence_devices`, `fence_gnbd` without GNBD Multipath

```
fence_devices{
   DeviceName {
      agent = "fence_gnbd"
      server = "ServerName"
      .
      .
      .
      server = "ServerName"
      option = "multipath"
      retrys = "Number"
      wait_time = "Seconds"
   }
   DeviceName {
   .
   .
   .
   }
}
```

Figure 6-8. File Structure: `fence_devices`, `fence_gnbd` with GNBD Multipath

```
fence_devices{
   DeviceName {
      agent = "fence_rib"
      hostname = "HostName"
      login = "LoginName"
      passwd = "LoginPassword"
   }
   DeviceName {
   .
   .
   .
   }
}
```

Figure 6-9. File Structure: `fence_devices`, `fence_rib`

```
fence_devices{
   DeviceName {
      agent = "fence_xcat"
      rpower = "RpowerBinaryPath"
   }
   DeviceName {
   .
   .
   .
   }
}
```

Figure 6-10. File Structure: `fence_devices`, `fence_xcat`

```
fence_devices{
   DeviceName {
      agent = "fence_egenera"
      cserver = "CserverName"
   }
   DeviceName {
   .
   .
   .
   }
}
```

Figure 6-11. File Structure: `fence_devices`, `fence_egenera`

```
fence_devices{
   DeviceName {
      agent = "fence_manual"
   }
   DeviceName {
   .
   .
   .
   }
}
```

Figure 6-12. File Structure: `fence_devices`, `fence_manual`

Warning

Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.

Parameter	Description
CserverName	For Egenera BladeFrame fencing device: The name of an Egenera control blade, the Egenera control blade with which the fence agent communicates via ssh.
DeviceName	The name of a fencing device. The *DeviceName* parameter specifies the name of a fencing device and makes that fencing device available for use by the fence section of a `nodes.ccs` file (`nodes.ccs:`*NodeName*`/fence/`*MethodName*). The fence section of a `nodes.ccs` file also contains *DeviceName* parameters each mapping to a *DeviceName* in the `fence.ccs` file.
HostName	The host name of a RILOE card on the network to which stunnel connections can be made.
IPAddress	For fencing with power and Fibre Channel switches: The IP address of a switch to which Telnet connections can be established.
LoginName	The login name for a power switch, an FC switch, or a RILOE card.
LoginPassword	The password for logging in to a power switch, an FC switch, or a RILOE card.
multipath	Selects GNBD multipath style fencing. *CAUTION:* When multipath style fencing is used, if the `gnbd_servd` process of a GNBD server node cannot be contacted, it is fenced as well, using its specified fencing method. That means that when a GNBD client (GFS node) is fenced, any node listed as its GNBD server that does not have the `gnbd_serv` module loaded (which starts `gnbd_servd`) is also fenced.
RpowerBinaryPath	For xCAT fencing device, the path to the `rpower` binary.
Number	The number of times to retry connecting to the GNBD server after a failed attempt, before the server is fenced. The parameter entry is for the `retrys =` entry and is only valid when used with multipath style fencing. (Refer to the multipath entry in this table.) The default value of *Number* is 3.
Seconds	The length of time, in seconds, to wait between retries. This parameter entry is for the `wait_time =` entry and is only valid when used with multipath style fencing. (Refer to the multipath entry in this table.) The default value of *Seconds* is 2.
ServerName	The host name of a GNBD server. Each GNBD server is represented with a "`server =`" line in the `fence.ccs` file. For example, if you have three GNBD servers, then the `fence.ccs` file needs three "`server =`" lines one for each GNBD server.

Table 6-2. File Syntax Description: Variables for `fence.ccs`

```
fence_devices {
        apc1 {
              agent = "fence_apc"
              ipaddr = "10.0.3.3"
              login = "apc"
              passwd = "apc"
        }
        apc2 {
              agent = "fence_apc"
              ipaddr = "10.0.3.4"
              login = "apc"
              passwd = "apc"
        }
}
```

Example 6-2. APC MasterSwitch Fencing Devices Named `apc1` and `apc2`

```
fence_devices {
        wti1 {
              agent = "fence_wti"
              ipaddr = "10.0.3.3"
              passwd = "password"
        }
        wti2 {
              agent = "fence_wti"
              ipaddr = "10.0.3.4"
              passwd = "password"
        }
}
```

Example 6-3. WTI NPS Fencing Devices Named `wti1` and `wti2`

```
fence_devices {
        silkworm1 {
              agent = "fence_brocade"
              ipaddr = "10.0.3.3"
              login = "admin"
              passwd = "password"
        }
        silkworm2 {
              agent = "fence_brocade"
              ipaddr = "10.0.3.4"
              login = "admin"
              passwd = "password"
        }
}
```

Example 6-4. Brocade FC-Switch Fencing Devices Named `silkworm1` and `silkworm2`

```
fence_devices {
         mdfc1 {
               agent = "fence_mcdata"
               ipaddr = "10.0.3.3"
               login = "admin"
               passwd = "password"
         }
         mdfc2 {
               agent = "fence_mcdata"
               ipaddr = "10.0.3.4"
               login = "admin"
               passwd = "password"
         }
}
```

Example 6-5. McData FC-Switch Fencing Devices Named `mdfc1` and `mdfc2`

```
fence_devices {
         vixel1 {
               agent = "fence_vixel"
               ipaddr = "10.0.3.3"
               passwd = "password"
         }
         vixel2 {
               agent = "fence_vixel"
               ipaddr = "10.0.3.4"
               passwd = "password"
         }
}
```

Example 6-6. Vixel FC-Switch Fencing Device Named `vixel1` and `vixel2`

```
fence_devices {
         gnbd {
               agent = "fence_gnbd"
               server = "nodea"
               server = "nodeb"
          }
}
```

This example shows a fencing device named `gnbd` with two servers: `nodea` and `nodeb`.

Example 6-7. GNBD Fencing Device Named `gnbd`, without GNBD Multipath

```
fence_devices {
         gnbdmp {
               agent = "fence_gnbd"
               server = "nodea"
               server = "nodeb"
               option = "multipath"  <-- Additional entry
               retrys = "5"          <-- Number of retries set to 5
               wait_time = "3"       <-- Wait time between retries set to 3
          }
}
```

This example shows a fencing device named `gnbdmp` with two servers: `nodea` and `nodeb`. Because GNBD Multipath is employed, an additional configuration entry under gnbdmp is needed: `option = "multipath"`. Also, for GNBD multipath, the example sets the number of retries to 5 with `retrys = 5`, and sets the wait time between retries to 3 with `wait_time = 3`.

Example 6-8. GNBD Fencing Device Named `gnbdmp`, with GNBD Multipath

```
fence_devices {
         riloe1 {
               agent = "fence_rib"
               ipaddr = "10.0.4.1"
               login = "admin"
               passwd = "password"
         }
         riloe2 {
               agent = "fence_rib"
               ipaddr = "10.0.4.2"
               login = "admin"
               passwd = "password"
         }
}
```

In this example, two RILOE fencing devices are defined for two nodes.

Example 6-9. Two HP-RILOE-Card Fencing Device Named `riloe1` and `riloe2`

```
fence_devices {
         xcat {
               agent = "fence_xcat"
               rpower = "/opt/xcat/bin/rpower"
         }
}
```

Example 6-10. xCAT Fencing Device Named `xcat`

```
fence_devices {
         egenera {
               agent = "fence_egenera"
               cserver = "c-blade1"
         }
}
```

Example 6-11. Egenera BladeFrame Fencing Devices Named and `xcat2`

```
fence_devices {
         admin {
               agent = "fence_manual"
         }
}
```

Example 6-12. Manual Fencing Device Named `admin`

Warning

Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.

6.7. Creating the `nodes.ccs` File

The `nodes.ccs` file specifies the nodes that run in a GFS cluster and their fencing methods. The nodes specified include those that run GFS and those that run LOCK_GULM servers. The `nodes.ccs` file is used in conjunction with the `fence.ccs` file to configure fencing in a cluster; the `nodes.ccs` file specifies fencing devices that are defined in the `fence.ccs` file.

Creating the `nodes.ccs` file consists of specifying the identity and fencing method (or methods) of each node in a GFS cluster. Specifying the identity consists of assigning a name and an IP address to the node. Specifying a fencing method consists of assigning a name to the fencing method and specifying its fencing-device parameters; that is, specifying how a node is fenced.

The way in which a fencing method is specified depends on if a node has either dual power supplies or multiple paths to FC storage. If a node has dual power supplies, then the fencing method for the node must specify at least two fencing devices — one fencing device for each power supply. Similarly, if a node has multiple paths to FC storage, then the fencing method for the node must specify one fencing device for each path to FC storage. For example, if a node has two paths to FC storage, the fencing method should specify two fencing devices — one for each path to FC storage. If a node has *neither* dual power supplies nor multiple paths to FC storage, then the fencing method for the node should specify *only one* fencing device.

You can configure a node with one fencing method or multiple fencing methods. When you configure a node for one fencing method, that is the only fencing method available for fencing that node. When you configure a node for multiple fencing methods, the fencing methods are *cascaded* from one fencing method to another according to the order of the fencing methods specified in the `nodes.ccs` file. If a node fails, it is fenced using the first fencing method specified in the `nodes.ccs` file for that node. If the first fencing method is not successful, the next fencing method specified for that node is used. If none of the fencing methods is successful, then fencing starts again with the first fencing method specified, and continues looping through the fencing methods in the order specified in `nodes.ccs` until the node has been fenced.

Refer to Chapter 10 *Using the Fencing System* for basic fencing details, descriptions of how fencing is used, and descriptions of available fencing methods.

To create the `nodes.ccs` file, follow these steps:

1. Create a file named `nodes.ccs`.

   a. If you are configuring a node for one fencing method (not cascaded), specify only one fencing method per node in the `nodes.ccs` file. Use a file format according to the fencing method as follows. Refer to Table 6-3 for syntax description.

   - APC MasterSwitch — For a node with a single power supply, refer to Figure 6-13. For a node with dual power supplies, refer to Figure 6-14.
   - WTI NPS — Refer to Figure 6-15.
   - Brocade, McData, or Vixel FC switch — Refer to Figure 6-16.
   - GNBD — Refer to Figure 6-17.
   - HP RILOE — Refer to Figure 6-18.
   - xCAT — Refer to Figure 6-19.
   - Egenera BladeFrame — Refer to Figure 6-20.
   - Manual — Refer to Figure 6-21.

   Warning

   Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.

   b. If you are configuring a node for cascaded fencing, use the file format in Figure 6-22. Refer to Table 6-3 for syntax description.

   Note

   Figure 6-22 does not show device-specific parameters for fencing methods. To determine device-specific parameters, refer to the appropriate figures listed in Step 1, part a.

2. For each node, specify *`NodeName`*, *`IFName`*, and the IP address of the node name, *`IPAddress`*.

   If your cluster is running Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5 and later, you can use the optional `usedev` parameter to explicitly specify an IP address rather than relying on an IP address from `libresolv`. For more information about the optional `usedev` parameter, refer to the file format in Figure 6-23 and the example in Example 6-26. Refer to Table 6-3 for syntax description of the `usedev` parameter.

   Note

   Figure 6-23 and Example 6-26 do not show device-specific parameters for fencing methods. To determine device-specific parameters, refer to the appropriate figures listed in Step 1, part a.

Note

Make sure that you specify *`Nodename`* as the Linux hostname and that the primary IP address of the node is associated with the hostname. Specifying *`NodeName`* other than the Linux hostname (for example the interface name) can cause unpredictable results — especially if the node is connected to multiple networks. To determine the hostname of a node, use the `uname -n` command on the node. To verify the IP address associated with the hostname, issue a `ping` command to the hostname.

3. For each node, specify the fencing parameters according to the fencing method you are using, as follows:

   a. If using APC MasterSwitch fencing, specify *`MethodName`*, *`DeviceName`*, *`PortNumber`*, and *`SwitchNumber`*. If you are configuring for dual power supplies, specify the following parameters for the second fencing device: *`DeviceName`*, *`PortNumber`*, and *`SwitchNumber`*. Refer to Example 6-13 for a `nodes.ccs` file that specifies APC MasterSwitch fencing for a single power supply. Refer to Example 6-14 for a `nodes.ccs` file that specifies APC MasterSwitch fencing for dual power supplies.

   b. If using WTI NPS fencing, specify *`MethodName`*, *`DeviceName`*, and *`PortNumber`*. Refer to Example 6-15 for a `nodes.ccs` file that specifies WTI NPS fencing.

   c. If using Brocade, McData, or Vixel FC-switch fencing, specify *`MethodName`*, *`DeviceName`*, and *`PortNumber`*. If you are configuring for multiple paths to FC storage, specify the following parameters for each additional fencing device required: *`DeviceName`* and *`PortNumber`*. Refer to Example 6-16 for a `nodes.ccs` file that specifies Brocade FC-switch fencing. Refer to Example 6-17 for a `nodes.ccs` file that specifies McData FC-switch fencing. Refer to Example 6-18 for a `nodes.ccs` file that specifies Vixel FC-switch fencing.

   d. If using GNBD fencing, specify *`MethodName`*, *`DeviceName`*, and *`IPAddress`*. Refer to Example 6-19 for a `nodes.ccs` file that specifies GNBD fencing.

   e. If using HP RILOE fencing, specify *`MethodName`*, *`DeviceName`*, and *`PortNumber`*. Refer to Example 6-20 for a `nodes.ccs` file that specifies HP RILOE fencing.

   f. If using xCAT fencing, specify *`MethodName`*, *`DeviceName`*, and *`NodelistName`*. Refer to Example 6-21 for a `nodes.ccs` file that specifies xCAT fencing.

   g. If using Egenera BladeFrame fencing, specify *`MethodName`*, *`DeviceName`*, *`LPANName`*, and *`PserverName`*. Refer to Example 6-22 for a `nodes.ccs` file that specifies Egenera BladeFrame fencing.

   h. If using manual fencing, specify *`MethodName`*, *`DeviceName`*, and *`IPAddress`*. Refer to Example 6-23 for a `nodes.ccs` file that specifies manual fencing.

      Warning

      Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.

   i. If using cascaded fencing, specify parameters according to the type of fencing methods and in the order that the fencing methods are to cascade. Refer to Example 6-24 for a `nodes.ccs` file that specifies cascaded fencing.

   j. If using GNBD multipath, fence the GNBD server nodes using any of the fencing methods stated in previous steps in the procedure *except* for GNBD fencing (Step 3, part d). Refer to Example 6-25 for a `nodes.ccs` file that specifies fencing for a GNBD server node.

4. Save the `nodes.ccs` file.

```
nodes {
    NodeName {
    :
    }
    NodeName {
        ip_interfaces {
            IFNAME = " IPAddress "
        }
        fence {
            MethodName {
                DeviceName {
                    port = PortNumber
                    switch = SwitchNumber
                }
            }
        }
    }
    NodeName {
    :
    }
}
```

File format for node identification (same format for all nodes)

File format for APC MasterSwitch fencing method, for node with single power supply only

Figure 6-13. File Format: `nodes.ccs`, APC Single Fencing Method, Single Power Supply

```
nodes {
   NodeName {
   :
   }
   NodeName {                                              File format for node
      ip_interfaces {                                      identification (same
         IFNAME = "IPAddress"                              format for all nodes)
      }
      fence {
         MethodName {
            DeviceName {          Fencing device           File format for APC
               port = PortNumber  for pwr supply 1         MasterSwitch
               switch = SwitchNumber                       fencing method, for
               option = "off"     Power down               node with dual
            }                     pwr supply 1             power supplies
            DeviceName {          Fencing device
               port = PortNumber  for pwr supply 2         Fencing a node with
               switch = SwitchNumber                       dual power supplies
               option = "off"     Power down               requires that both
            }                     pwr supply 2             power supplies be
            DeviceName {          Fencing device           powered off before
               port = PortNumber  for pwr supply 1         rebooting the node.
               switch = SwitchNumber                       To accomplish that,
               option = "on"      Power up                 the nodes.ccs file
            }                     pwr supply 1             requires the use of
            DeviceName {          Fencing device           the option =
               port = PortNumber  for pwr supply 2         parameter: first, set
               switch = SwitchNumber                       to "off" for for the
               option = "on"      Power up                 fencing device of
            }                     pwr supply 2             each power supply,
         }                                                 then set to "on" for
      }                                                    the fencing device
   }                                                       of each power
   NodeName {                                              supply.
   :
   }
}
```

Figure 6-14. File Format: `nodes.ccs`, APC Single Fencing Method, Dual Power Supply

```
nodes {
    NodeName {
    :
    }
    NodeName {
        ip_interfaces {
            IFNAME = " IPAddress "
        }
        fence {
            MethodName {
                DeviceName {
                    port = PortNumber
                }
            }
        }
    }
    NodeName {
    :
    }
}
```

File format for node identification (same format for all nodes)

File format for WTI NPS fencing method

Figure 6-15. File Format: `nodes.ccs`, WTI NPS, Single Fencing Method

```
nodes {
    NodeName {
    :
    }
    NodeName {
        ip_interfaces {
            IFNAME = " IPAddress "
        }
        fence {
            MethodName {
                DeviceName {
                    port = PortNumber
                }
                DeviceName {
                    port = PortNumber
                }
            }
        }
    }
    NodeName {
    :
    }
}
```

File format for node identification (same format for all nodes)

File format for Brocade, McData, or Vixel FC-Switch fencing method

Additional fencing device: one required for each additional FC path

Figure 6-16. File Format: `nodes.ccs`, Brocade, McData, or Vixel FC Switch, Single Fencing Method

```
nodes {
   NodeName {
   .
   .
   .
   }
   NodeName {
      ip_interfaces {
         IFNAME = " IPAddress "
      }
      fence {
         MethodName {
            DeviceName {
               ipaddr = " IPAddress "
            }
         }
      }
   }
   NodeName {
   .
   .
   .
   }
}
```

File format for node identification (same format for all nodes)

File format for GNBD fencing method

Figure 6-17. File Format: `nodes.ccs`, GNBD, Single Fencing Method

```
nodes {
   NodeName {
   :
   }
   NodeName {
       ip_interfaces {
           IFNAME = " IPAddress "
       }
       fence {
           MethodName {
               DeviceName {
                   localport = PortNumber
               }
           }
       }
   }
   NodeName {
   :
   }
}
```

File format for node identification (same format for all nodes)

File format for HP RILOE fencing method

Figure 6-18. File Format: `nodes.ccs`, HP RILOE, Single Fencing Method

```
nodes {
    NodeName    {
    :
    }
    NodeName    {
        ip_interfaces    {
            IFNAME    = " IPAddress    "
        }
        fence {
            MethodName    {
                DeviceName    {
                    nodename =    "NodelistName"
                }
            }
        }
    }
    NodeName    {
    :
    }
}
```

File format for node identification (same format for all nodes)

File format for xCAT fencing method

Figure 6-19. File Format: `nodes.ccs`, xCAT Fencing Method

```
nodes {
    NodeName    {
    :
    }
    NodeName    {
        ip_interfaces    {
            IFNAME    = " IPAddress   "
        }
        fence {
            MethodName    {
                DeviceName    {
                    lpan =    "LPANName"
                    pserver =    "PserverName"
                }
            }
        }
    }
    NodeName    {
    :
    }
}
```

File format for node identification (same format for all nodes)

File format for Egenera BladeFrame fencing method

Figure 6-20. File Format: `nodes.ccs`, Egenera BladeFrame Fencing Method

```
nodes {
    NodeName {
    :
    }
    NodeName {
        ip_interfaces {
            IFNAME = " IPAddress "
        }
        fence {
            MethodName {
                DeviceName {
                    ipaddr = " IPAddress "
                }
            }
        }
    }
    NodeName {
    :
    }
}
```

File format for node identification (same format for all nodes)

File format for manual fencing method

Figure 6-21. File Format: `nodes.ccs`, Manual Fencing Method

Warning

Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.

```
nodes {
   NodeName {
   .
   .
   .
   }
   NodeName {
       ip_interfaces {
           IFName = "IPAddress"
       }
       fence {
           MethodName {
               DeviceName {
                   #Device-specific parameter(s)
               }
           }
           MethodName {
               DeviceName {
                   #Device-specific parameter(s)
               }
           }
           MethodName {
               DeviceName {
                   #Device-specific parameter(s)
               }
           }
       }
   }
   NodeName {
   .
   .
   .
   }
}
```

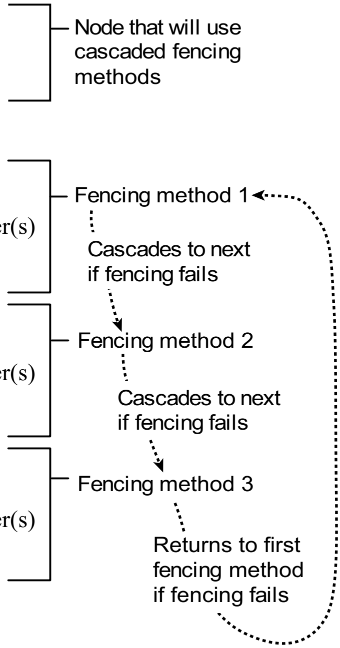

Figure 6-22. File Format: `nodes.ccs`, Cascaded Fencing

```
nodes {
  NodeName {
     ip_interfaces {
         IFNAME="IPAddress"  <-- Must be an IP address; not a name
     }
     usedev = "NamedDevice"  <-- Optional parameter usedev
     fence {
     .
     .
     .
     }
  }
  NodeName {
  .
  .
  .
  }
}
```

Figure 6-23. File Structure: Optional `usedev` Parameter

Parameter	Description
DeviceName	The name of a fencing device to use with a node. Use a valid fencing device name specified by a *DeviceName* parameter in the `fence.ccs` file (`fence.ccs:fence_devices/`*DeviceName*).
IFName	The interface name of the IP address specified. For example: *eth0*
IPAddress	For the `ip_interfaces` section: The IP address of the node on the interface specified. GULM uses this parameter only if the optional `usedev` parameter is specified in the `nodes.ccs` file. The `usedev` parameter is available only with Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5 and later. For the `fence` section: If GNBD fencing — The IP address of this node, the node to be fenced. If manual fencing — IP address of this node, the node that needs to be reset or disconnected from storage. *WARNING:* Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.
LPANName	For Egenera BladeFrame fencing: This is the name of the Logical Processing Area Network (LPAN), of which the node (an Egenera pServer) to be fenced is a member.
LoginPassword	This is the password of the node to be fenced.
MethodName	A name describing the fencing method performed by the listed devices. For example, a *MethodName* of `power` could be used to describe a fencing method using an APC MasterSwitch. Or, a *MethodName* of `Cascade1` could be used to describe a cascaded fencing method.

Parameter	Description
NamedDevice	Used with `usedev`. *NamedDevice* indicates that the IP address is specified by the optional parameter `usedev`, and *not* by the IP address pulled from `libresolv`. The `usedev` and *NamedDevice* parameters are available with Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5 or later.
NodelistName	For xCAT: The node name of the node to be fenced, as defined in the `nodelist.tab` file.
NodeName	The Linux hostname of the node. **Note**: Make sure that you use the Linux hostname and that the primary IP address of the node is associated with the hostname. Specifying a *NodeName* other than the Linux hostname (for example the interface name) can cause unpredictable results — especially if the node is connected to multiple networks. To determine the hostname of a node, you can use the `uname -n` command at the node. To verify the IP address associated with the hostname, you can issue a `ping` command to the hostname.
PortNumber	For power and FC switches: The port number on the switch to which this node is connected. For HP RILOE: This is an optional value that defines a local port to be used. The default value is 8888.
PserverName	For Egenera BladeFrame fencing: This is the name of an Egenera pServer, the node to be fenced.
SwitchNumber	For use with APC MasterSwitch: When chaining more than one switch, this parameter specifies the switch number of the port. This entry is not required when only one switch is present. (The default value is 1 if not specified.)
usedev	This is an optional parameter available with Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5 or later. If `usedev` is present, GULM uses the IP address from that device in the `ip_interfaces` section. Otherwise GULM uses the IP address from `libresolv` (as it does in releases earlier than Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5).
UserId	The user ID of the node to be fenced.

Table 6-3. File Syntax Description: Variables for `nodes.ccs`

```
nodes {
   n01 {
      ip_interfaces {
         hsi0 = "10.0.0.1"
      }
      fence {
         power {
             apc1 {
                port = 6
                switch = 2
             }
          }
       }
  }
  n02 {
  .
  .
  .
  }
}
```

Example 6-13. Node Defined for APC Fencing, Single Power Supply

```
nodes {
   n01 {
      ip_interfaces {
        hsi0 = "10.0.0.1"
      }
      fence {
         power {
            apc1 { <----------- Fencing device for power supply 1
               port = 6
               switch = 1
               option = "off" <-- Power down power supply 1
            }
            apc2 { <----------- Fencing device for power supply 2
               port = 7
               switch = 2
               option = "off" <-- Power down power supply 2
            }
            apc1 { <----------- Fencing device for power supply 1
               port = 6
               switch = 1
               option = "on" <--- Power up power supply 1
            }
            apc2 { <----------- Fencing device for power supply 2
               port = 7
               switch = 2
               option = "on" <--- Power up power supply 2
            }
         }
      }
   }
   n02 {
   .
   .
   .
   }
}
```

Example 6-14. Node Defined for APC Fencing, Dual Power Supplies

```
nodes {
   n01 {
      ip_interfaces {
         hsi0 = "10.0.0.1"
      }
      fence {
         power {
            wti1 {
               port = 1
            }
         }
      }
   }
   n02 {
   .
   .
   .
   }
}
```

Example 6-15. Node Defined for WTI NPS Fencing

```
nodes {
   n01 {
      ip_interfaces {
         hsi0 = "10.0.0.1"
      }
      fence {
         san {
            silkworm1 {
               port = 3
            }
            silkworm2 { <--- Additional fencing device, for additional
                port = 4     path to FC storage
            }
         }
      }
   }
   n02 {
   .
   .
   .
   }
}
```

Example 6-16. Node Defined for Brocade FC-Switch Fencing

```
nodes {
   n01 {
      ip_interfaces {
         hsi0 = "10.0.0.1"
      }
      fence {
         san {
            mdfc1 {
               port = 3
            }
            mdfc2 { <--- Additional fencing device, for additional
                port = 4     path to FC storage
            }
         }
      }
   }
   n02 {
   .
   .
   .
   }
}
```

Example 6-17. Node Defined for McData FC-Switch Fencing

```
nodes {
   n01 {
      ip_interfaces {
         hsi0 = "10.0.0.1"
      }
      fence {
         san {
            vixel1 {
               port = 3
            }
            vixel2 { <---- Additional fencing device, for additional
               port = 4    path to FC storage
            }
         }
      }
   }
   n02 {
   .
   .
   .
   }
}
```

Example 6-18. Node Defined for Vixel FC-Switch Fencing

```
nodes {
   n01 {
      ip_interfaces {
         hsi0 = "10.0.0.1"
      }
      fence {
         server {
            gnbd {
               ipaddr = "10.0.1.1"
            }
         }
      }
   }
   n02 {
   .
   .
   .
   }
}
```

Example 6-19. Node Defined for GNBD Fencing

```
nodes {
   n01 {
      ip_interfaces {
         hsi0 = "10.0.0.1"
      }
      fence {
         riloe {
            riloe1 {
               localport = 2345
            }
         }
      }
   }
   n02 {
   .
   .
   .
   }
}
```

Example 6-20. Node Defined for HP RILOE Fencing

```
nodes {
   n01 {
      ip_interfaces {
         hsi0 = "10.0.0.1"
      }
      fence {
         blade-center {
                 xcat {
                    nodename = "blade-01"
            }
         }
      }
   }
   n02 {
      ip_interfaces {
         hsi0 = "10.0.0.2"
      }
      fence {
         blade-center {
                 xcat {
                    nodename = "blade-02"
            }
         }
      }
   }
   n03 {
   .
   .
   .
   }
}
```

Example 6-21. Nodes Defined for xCAT Fencing

```
nodes {
   n01 {
      ip_interfaces {
         hsi0 = "10.0.0.1"
      }
      fence {
         blade-center {
                egenera {
                   lpan = "opsgroup"
                   pserver = "ops-1
            }
         }
      }
   }
   n02 {
      ip_interfaces {
         hsi0 = "10.0.0.2"
      }
      fence {
         blade-center {
                egenera {
                   lpan = "opsgroup"
                   pserver = "ops-2
            }
         }
      }
   }
   n03 {
   .
   .
   .
   }
}
```

Example 6-22. Nodes Defined for Egenera BladeFrame Fencing

```
nodes {
   n01 {
      ip_interfaces {
         hsi0 = "10.0.0.1"
      }
      fence {
         human {
            admin {
               ipaddr = "10.0.0.1"
            }
         }
      }
   }
   n02 {
   .
   .
   .
   }
}
```

Example 6-23. Nodes Defined for Manual Fencing

Warning

Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.

```
nodes {
   n01 {
      ip_interfaces {
         eth0 = "10.0.1.21"
      }
      fence {
         san {            <-- Fencing with Brocade FC switch
            brocade1 {
               port = 1
            }
         }
         power {          <-- Fencing with APC MasterSwitch
            apc {
               port = 1
               switch = 1
            }
         }
      }
   }
   n02 {
   .
   .
   .
   }
}
```

This example shows a node that can be fenced using a Brocade FC switch or an APC MasterSwitch. If the node must be fenced, the fencing system first attempts to disable the node's FC port. If that operation fails, the fencing system attempts to reboot the node using the power switch.

Example 6-24. Nodes Defined for Cascaded Fencing

```
nodes {
   n01 {
      ip_interfaces {
         hsi0 = "10.0.0.1"
      }
      fence {
         power { <------------- APC MasterSwitch fencing device
            apc1 {
               port = 6
               switch = 2
            }
      }
   }
   n02 {
   .
   .
   .
   }
}
```

Example 6-25. GNBD Server Node Defined for APC Fencing, Single Power Supply

```
nodes {
   n01 {
      ip_interfaces {
         wizzy = "10.0.0.1"  <-- Must be an IP address; not a name
      }
      usedev = "wizzy"  <-- Optional parameter usedev set to "wizzy"
      fence {
      .
      .
      .
      }
   }
   n02 {
   .
   .
   .
   }
}
```

Example 6-26. Optional `usedev` Parameter

Chapter 7.

Using the Cluster Configuration System

This chapter describes how to use the *cluster configuration system* (CCS) and consists of the following sections:

- Section 7.1 *Creating a CCS Archive*
- Section 7.2 *Starting CCS in the Cluster*
- Section 7.3 *Using Other CCS Administrative Options*
- Section 7.4 *Changing CCS Configuration Files*
- Section 7.5 *Alternative Methods to Using a CCA Device*
- Section 7.6 *Combining CCS Methods*

Note

If you are using GFS with Red Hat Cluster, you can create a CCS archive with **GFS Druid**. For information about configuring and using GFS with Red Hat Cluster Suite, refer to Appendix A *Using Red Hat GFS with Red Hat Cluster Suite*.

Note

If your GFS cluster is configured for GNBD multipath, there are some considerations you must take into account for the location of CCS files. Refer to Section 11.2 *Considerations for Using GNBD Multipath*.

7.1. Creating a CCS Archive

A *CCS archive* is a collection of CCS configuration files that can be accessed by the cluster. The `ccs_tool` command is used to create a CCS archive from a directory containing `.ccs` configuration files. This command writes the archive to a shared pool called the *CCA device*.

A small pool volume may be used as the CCA device. You can determine the size of the CCA device pool volume as follows: 2 KB per GFS node or 2 MB total, whichever is larger. (Refer to Section 5.5 *Creating a Pool Volume* and Section 5.6 *Activating/Deactivating a Pool Volume* for details on creating and activating a pool volume for the CCA device.)

7.1.1. Usage

```
ccs_tool create Directory CCADevice
```

`Directory`

The relative path to the directory containing the CCS files for the cluster.

CCADevice

Specifies the name of the CCA device.

7.1.2. Example

In this example, the name of the cluster is `alpha`, and the name of the pool is `/dev/pool/alpha_cca`. The CCS configuration files in directory `/root/alpha/` are used to create a CCS archive on the CCA device `/dev/pool/alpha_cca`.

```
ccs_tool create /root/alpha/ /dev/pool/alpha_cca
```

7.1.3. Comments

- The `-O` (override) option can be specified after the command name (`ccs_tool -O create`) to forcibly write over the current CCA device contents without a prompt.

  Warning

  Make sure that you specify the right device if you use the override option. Otherwise, data may be lost.

- Depending on the size of the device, it may take several seconds to create a CCA device for the first time due to initialization of the device.
- The `ccs_tool` command uses the Linux raw-device interface to update and read a CCA device directly, bypassing operating system caches. Caching effects could otherwise create inconsistent views of the CCA device between cluster nodes.

7.2. Starting CCS in the Cluster

Once a CCS archive has been created on a CCA device (refer to Section 7.1 *Creating a CCS Archive* for details, if necessary), the CCS daemon (`ccsd`) should be started on all cluster nodes. All cluster nodes must be able to see the CCA device before the daemon is started.

The CCS daemon provides an interface to configuration data that is independent of the specific location where the data is stored.

7.2.1. Usage

`ccsd -d` *`CCADevice`*

CCADevice

Specifies the name of the CCA device.

Note

You can use the `ccsd init.d` script included with GFS to automate starting and stopping `ccsd`. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS init.d Scripts*.

7.2.2. Example

In this example, the CCS daemon is started on a cluster node. This command should be run on all cluster nodes:

`ccsd -d /dev/pool/alpha_cca`

7.2.3. Comments

The CCS daemon (`ccsd`) uses the Linux raw-device interface to update and read a CCA device directly, bypassing operating system caches. Caching effects could otherwise create inconsistent views of the CCA device between cluster nodes.

7.3. Using Other CCS Administrative Options

The following sections detail other administrative functions that can be performed by the `ccs_tool` command.

7.3.1. Extracting Files from a CCS Archive

When extracting original CCS configuration files from a CCS archive, the `ccs_tool extract` command creates a new directory specified on the command line and recreates the CCS files in the directory. The CCS archive remains unaffected by this command.

7.3.1.1. Usage

```
ccs_tool extract CCADevice Directory
```

CCADevice

Specifies the name of the CCA device.

Directory

The relative path to the directory containing the CCS files for the cluster.

7.3.1.2. Example

In this example, the CCS files contained on the CCA device, `/dev/pool/alpha_cca`, are extracted and recreated in directory `/tmp/alpha-bak/`.

`ccs_tool extract /dev/pool/alpha_cca /tmp/alpha-bak/`

7.3.2. Listing Files in a CCS Archive

The CCS configuration files contained within a CCS archive can be listed by using the `ccs_tool` list command.

7.3.2.1. Usage

```
ccs_tool list CCADevice
```

CCADevice

Specifies the name of the CCA device.

7.3.2.2. Example

This example causes the CCS files contained on the CCA device, `/dev/pool/alpha_cca`, to be listed.

```
ccs_tool list /dev/pool/alpha_cca
```

7.3.3. Comparing CCS Configuration Files to a CCS Archive

The `ccs_tool diff` command can be used to compare a directory of CCS configuration files with the configuration files in a CCS archive. The output from the `ccs_tool diff` command is displayed for each corresponding file in the specified directory and the CCS archive.

7.3.3.1. Usage

```
ccs_tool diff CCADevice [Directory]
```

CCADevice

Specifies the name of the CCA device.

Directory

The relative path to the directory containing the CCS files for the cluster.

7.3.3.2. Example

In this example, the CCS configuration files in directory `/root/alpha/` are compared with the configuration files in CCA device `/dev/pool/alpha_cca`.

```
ccs_tool diff /dev/pool/alpha_cca /root/alpha/
```

7.4. Changing CCS Configuration Files

Based on the LOCK_GULM locking protocol, the following list defines what can or cannot be changed in a CCS archive while a cluster is running. There are no restrictions to making changes to configuration files when the cluster is offline.

- New nodes *can* be defined in the `nodes.ccs` file.
- Unused node definitions *can* be removed from the `nodes.ccs` file.
- New fencing devices *can* be defined in the `fence.ccs` file.
- The locking servers array (`servers =`) in `cluster.ccs:cluster/lock_gulm` *cannot* be changed.
- The fencing parameters for an existing node definition in `nodes.ccs` *can* be changed.
- The IP address of an existing node definition in the `nodes.ccs` file *can* only be changed *if* the node does not have any GFS file systems mounted and is not running a LOCK_GULM server.

7.4.1. Example Procedure

This example procedure shows how to change configuration files in a CCS archive.

1. Extract configuration files from the CCA device into temporary directory `/root/alpha-new/`.
   `ccs_tool extract /dev/pool/alpha_cca /root/alpha-new/`
2. Make changes to the configuration files in `/root/alpha-new/`.
3. Create a new CCS archive on the CCA device by using the `-O` (override) flag to forcibly overwrite the existing CCS archive.
   `ccs_tool -O create /root/alpha-new/ /dev/pool/alpha_cca`

7.5. Alternative Methods to Using a CCA Device

If it is not possible to reserve shared storage for use as a CCA device, you can use two alternative methods:

- Section 7.5.1 *CCA File and Server*
- Section 7.5.2 *Local CCA Files*

Neither of these methods requires shared storage to store CCS data.

7.5.1. CCA File and Server

The first alternative to a CCA device is to use a single network server to serve CCS configuration files to all nodes in the cluster. If used, this CCS server is a single point of failure in a cluster. If a single (non-redundant) LOCK_GULM server daemon is being used, it would be reasonable to run a CCS server on the same node as the LOCK_GULM server. The CCS server does not have failover capabilities.

The CCS server is called `ccs_servd`, it can be run on any node in or out of the cluster. When CCS daemons (`ccsd`) are started on cluster nodes, the IP address of the node running `ccs_servd` is specified instead of the name of the CCA device. The name of the cluster is also passed to `ccsd`.

The CCS server does not read CCS files directly; rather, it reads a CCA file that is a local file containing a CCS archive.

Steps related to CCS in the setup procedure must be modified to use a CCS server in place of a CCA device.

> **Note**
>
> `ccs_servd` provides information to any computer that can connect to it. Therefore, `ccs_servd` should not be used at sites where untrusted nodes can contact the CCS server.

7.5.1.1. Creating a CCA File

Like a CCA device, a CCA file is created by the `ccs_tool` command from a directory of CCS configuration files. Instead of specifying a CCA device as the last parameter when creating an archive, a local file name is specified. The `ccs_tool` command creates the named file, which is the CCA file. That file should be named *`ClusterName`* with a `.cca` extension. (*`ClusterName`* is the user-supplied variable that specifies the name of the cluster.) The CCA file must be located on the node that runs `ccs_servd`.

7.5.1.1.1. Usage

```
ccs_tool create Directory CCAFile
```

`Directory`

The relative path to the directory containing the CCS files for the cluster.

`CCAFile`

Specifies the CCA file to create.

7.5.1.1.2. Example

In this example, the name of the cluster is *alpha* and the name of the CCA file is `alpha.cca`. The CCA file is saved in the `/etc/sistina/ccs-build/` directory, which is the default location where `ccs_servd` looks for CCA files.

```
ccs_tool create /root/alpha/ /etc/sistina/ccs-build/alpha.cca
```

7.5.1.2. Starting the CCS Server

There are two parts to starting CCS in the cluster when using a CCS server. The first is starting `ccs_servd` and the second is starting `ccsd` on all the cluster nodes. When starting `ccs_servd`, no command line options are required unless the CCA file is saved in a location other than `/etc/sistina/ccs-build/`.

7.5.1.2.1. Usage

```
ccs_servd
```

or

```
ccs_servd -p Path
```

Path

Specifies an alternative location of CCA files.

7.5.1.2.2. Examples

This example shows starting the CCS server normally; that is, using the default location for CCA files.

ccs_servd

This example shows starting the CCS server using a user-defined location for CCA files. In this case, CCA files are saved in `/root/cca/`.

ccs_servd -p /root/cca/

7.5.1.3. Starting the CCS Daemon

When using a CCS server, `ccsd` must connect to it over the network, and requires two parameters on the `ccsd` command line: the IP address (and optional port number) of the node running `ccs_servd`, and the name of the cluster.

7.5.1.3.1. Usage

```
ccsd -s IPAddress[:PortNumber] -c ClusterName
```

IPAddress

The IP address of the node running the CCS server.

:PortNumber

(Optional) The non-default port number. A colon and port number can optionally follow the IPAddress to specify a non-default port number: *IPAddress:PortNumber*.

ClusterName

Specifies the name of the cluster. The CCS server uses this to pick the correct CCA file that is named for the cluster.

7.5.1.3.2. Example

This example starts `ccsd` on a node for cluster *alpha* when using a CCS server with the IP address shown.

ccsd -s 10.0.5.1 -c alpha

7.5.2. Local CCA Files

An alternative to both a CCA device and a CCS server is to replicate CCA files on all cluster nodes.

Note

Care must be taken to keep all the copies identical.

A CCA file is created using the same steps as for a CCS server. The CCA file is manually copied to all cluster nodes.

7.5.2.1. Starting the CCS Daemon

When the CCS daemon is started on each node, it must be given the location of the local copy of the CCA file.

7.5.2.2. Usage

```
ccsd -f File
```

File

Specifies the local copy of the CCA file.

7.5.2.3. Example

This example starts `ccsd` on a node using a local copy of a CCA file.

```
ccsd -f /etc/sistina/ccs-build/alpha.cca
```

7.6. Combining CCS Methods

The shared block-device methodology described at the beginning of this chapter is the recommended method for storing CCS data (Section 7.1 *Creating a CCS Archive* and Section 7.2 *Starting CCS in the Cluster*). The advantages are that there is no server point-of-failure and that updates to the CCS archive happen atomically.

However, not every cluster has every node connected to the shared storage. For example, a cluster may be built with external lock servers that do not have access to the shared storage. In that case, the client/server methodology (Section 7.5.1 *CCA File and Server*) could be employed, but that approach introduces a server point-of-failure. Also, local file archives could be used on each node (Section 7.5.2 *Local CCA Files*), but that approach makes updating the CCS archives difficult.

The best approach for storing CCS data may be a combination of the shared-device method and the local-files method. For example, the cluster nodes attached to shared storage could use the shared-device method, and the other nodes in the cluster could use the local-files approach. Combining those two methods eliminates the possible point-of-failure and reduces the effort required to update a CCS archive.

Note

When you update a CCS archive, update the shared-device archive first, then update the local archives. *Be sure to keep the archives synchronized*.

Chapter 8.

Using Clustering and Locking Systems

This chapter describes how to use the clustering and locking systems available with GFS, and consists of the following sections:

- Section 8.1 *Locking System Overview*
- Section 8.2 *LOCK_GULM*
- Section 8.3 *LOCK_NOLOCK*

8.1. Locking System Overview

The Red Hat GFS interchangeable locking/clustering mechanism is made possible by the `lock_harness.o` kernel module. The GFS kernel module `gfs.o` connects to one end of the harness, and lock modules connect to the other end. When a GFS file system is created, the lock protocol (or lock module) that it uses is specified. The kernel module for the specified lock protocol must be loaded subsequently to mount the file system. The following lock protocols are available with GFS:

- LOCK_GULM — Implements both RLM and SLM and is the recommended choice
- LOCK_NOLOCK — Provides no locking and allows GFS to be used as a local file system

8.2. LOCK_GULM

RLM and SLM are both implemented by the LOCK_GULM system.

LOCK_GULM is based on a central server daemon that manages lock and cluster state for all GFS/LOCK_GULM file systems in the cluster. In the case of RLM, multiple servers can be run redundantly on multiple nodes. If the master server fails, another "hot-standby" server takes over.

The LOCK_GULM server daemon is called `lock_gulmd`. The kernel module for GFS nodes using LOCK_GULM is called `lock_gulm.o`. The lock protocol (LockProto) as specified when creating a GFS/LOCK_GULM file system is called `lock_gulm` (lower case, with no `.o` extension).

Note

You can use the `lock_gulmd init.d` script included with GFS to automate starting and stopping `lock_gulmd`. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS `init.d` Scripts*.

8.2.1. Selection of LOCK_GULM Servers

The nodes selected to run the `lock_gulmd` server are specified in the `cluster.ccs` configuration file (`cluster.ccs:cluster/lock_gulm/servers`). Refer to Section 6.5 *Creating the `cluster.ccs` File*.

For optimal performance, lock_gulmd servers should be run on dedicated nodes; however, they can also be run on nodes using GFS. All nodes, including those only running lock_gulmd, must be listed in the nodes.ccs configuration file (nodes.ccs:nodes).

8.2.2. Number of LOCK_GULM Servers

You can use just one lock_gulmd server; however, if it fails, the entire cluster that depends on it must be reset. For that reason, you can run multiple instances of the lock_gulmd server daemon on multiple nodes for redundancy. The redundant servers allow the cluster to continue running if the master lock_gulmd server fails.

Over half of the lock_gulmd servers on the nodes listed in the cluster.ccs file (cluster.ccs:cluster/lock_gulm/servers) must be operating to process locking requests from GFS nodes. That quorum requirement is necessary to prevent split groups of servers from forming independent clusters — which would lead to file system corruption.

For example, if there are three lock_gulmd servers listed in the cluster.ccs configuration file, two of those three lock_gulmd servers (a quorum) must be running for the cluster to operate.

A lock_gulmd server can rejoin existing servers if it fails and is restarted.

When running redundant lock_gulmd servers, the minimum number of nodes required is three; the maximum number of nodes is five.

8.2.3. Starting LOCK_GULM Servers

If no lock_gulmd servers are running in the cluster, take caution before restarting them — you must verify that no GFS nodes are hung from a previous instance of the cluster. If there are hung GFS nodes, reset them before starting lock_gulmd servers. Resetting the hung GFS nodes before starting lock_gulmd servers prevents file system corruption. Also, be sure that all nodes running lock_gulmd can communicate over the network; that is, there is no network partition.

The lock_gulmd server is started with no command line options.

8.2.4. Fencing and LOCK_GULM

Cluster state is managed in the lock_gulmd server. When GFS nodes or server nodes fail, the lock_gulmd server initiates a fence operation for each failed node and waits for the fence to complete before proceeding with recovery.

The master lock_gulmd server fences failed nodes by calling the fence_node command with the name of the failed node. That command looks up fencing configuration in CCS to carry out the fence operation.

When using RLM, you need to use a fencing method that shuts down and reboots a node. With RLM you *cannot* use any method that does not reboot the node.

8.2.5. Shutting Down a LOCK_GULM Server

Before shutting down a node running a LOCK_GULM server, lock_gulmd should be terminated using the gulm_tool command. If lock_gulmd is not properly stopped, the LOCK_GULM server may be fenced by the remaining LOCK_GULM servers.

Caution

> Shutting down one of multiple redundant LOCK_GULM servers may result in suspension of cluster operation if the remaining number of servers is half or less of the total number of servers listed in the `cluster.ccs file` (`cluster.ccs:lock_gulm/servers`).

8.2.5.1. Usage

`gulm_tool shutdown` *`IPAddress`*

`IPAddress`

Specifies the IP address or hostname of the node running the instance of `lock_gulmd` to be terminated.

8.3. LOCK_NOLOCK

The LOCK_NOLOCK system allows GFS to be used as a local file system on a single node.

The kernel module for a GFS/LOCK_NOLOCK node is `lock_nolock.o`. The lock protocol as specified when creating a GFS/LOCK_NOLOCK file system is called `lock_nolock` (lower case, with no `.o` extension).

Caution

> *Do not* allow multiple nodes to mount the same file system while LOCK_NOLOCK is used. Doing so causes one or more nodes to panic their kernels, and may cause file system corruption.

Chapter 9.

Managing GFS

This chapter describes the tasks and commands for managing GFS and consists of the following sections:

- Section 9.1 *Making a File System*
- Section 9.2 *Mounting a File System*
- Section 9.3 *Unmounting a File System*
- Section 9.4 *GFS Quota Management*
- Section 9.5 *Growing a File System*
- Section 9.6 *Adding Journals to a File System*
- Section 9.7 *Direct I/O*
- Section 9.8 *Data Journaling*
- Section 9.9 *Configuring `atime` Updates*
- Section 9.10 *Suspending Activity on a File System*
- Section 9.11 *Displaying Extended GFS Information and Statistics*
- Section 9.12 *Repairing a File System*
- Section 9.13 *Context-Dependent Path Names*
- Section 9.14 *Shutting Down a GFS Cluster*
- Section 9.15 *Starting a GFS Cluster*

9.1. Making a File System

Making a GFS file system is one of the final tasks in the process of configuring and setting up a GFS cluster. (Refer to Chapter 4 *Initial Configuration* for more information.) Once a cluster is set up and running, additional GFS file systems can be made and mounted without additional cluster-configuration steps.

A file system is created on a block device, which is usually an activated Pool volume. (Refer to Chapter 5 *Using the Pool Volume Manager* for further details.) The following information is required to run the `gfs_mkfs` command:

- Lock protocol/module name (for example, `lock_gulm`)
- Cluster name (from `cluster.ccs`)
- Number of nodes that may be mounting the file system

9.1.1. Usage

```
gfs_mkfs -p LockProtoName -t LockTableName -j Number BlockDevice
```

Warning

Make sure that you are very familiar with using the *LockProtoName* and *LockTableName* parameters. Improper use of the *LockProtoName* and *LockTableName* parameters may cause file system or lock space corruption.

LockProtoName

Specifies the name of the locking protocol (typically `lock_gulm`) to use.

LockTableName

This parameter has two parts separated by a colon (no spaces) as follows: *ClusterName:FSName*

- *ClusterName*, the cluster name, is set in the `cluster.ccs` file (`cluster.ccs:cluster/name`).
- *FSName*, the file system name, can be 1 to 16 characters long, and the name must be unique among all file systems in the cluster.

Number

Specifies the number of journals to be created by the `gfs_mkfs` command. One journal is required for each node that mounts the file system. (More journals can be specified to allow for easier, future expansion.)

BlockDevice

Usually specifies a pool volume, but any block device can be specified.

9.1.2. Examples

In this example, `lock_gulm` is the locking protocol that the file system uses. The cluster name is `alpha`, and the file system name is `gfs1`. The file system contains eight journals and is created on the `pool0` block device.

`gfs_mkfs -p lock_gulm -t alpha:gfs1 -j 8 /dev/pool/pool0`

In this example, a second `lock_gulm` file system is made, which can be used in cluster `alpha`. The file system name is `gfs2`. The file system contains eight journals and is created on the `pool1` block device.

`gfs_mkfs -p lock_gulm -t alpha:gfs2 -j 8 /dev/pool/pool1`

9.1.3. Complete Options

Table 9-1 describes the `gfs_mkfs` command options (flags and parameters).

Flag	Parameter	Description
`-b`	*BlockSize*	Sets the file system block size to *BlockSize*. Default block size is 4096 bytes.
`-D`		Enables debugging output.
`-h`		Help. Displays available options, then exits.
`-J`	*MegaBytes*	Specifies the size of the journal in megabytes. Default journal size is 128 megabytes. The minimum size is 32 megabytes.
`-j`	*Number*	Specifies the number of journals to be created by the `gfs_mkfs` command. One journal is required for each node that mounts the file system. **Note:** More journals than are needed can be specified at creation time to allow for future expansion.
`-P`		Tells the `gfs_mkfs` command that the underlying device is a pool. The `gfs_mkfs` command then asks the pool about its layout. The `-p` flag overrides the `-j` and `-J` flags.
`-p`	*LockProtoName*	Specifies the name of the locking protocol to use. Recognized cluster-locking protocols include: LOCK_GULM — The standard GFS locking module. LOCK_NOLOCK — May be used when GFS is acting as a local file system (one node only).
`-O`		Prevents the `gfs_mkfs` command from asking for confirmation before writing the file system.
`-q`		Quiet. Do not display anything.
`-r`	*MegaBytes*	Specifies the size of the resource groups in megabytes. Default resource group size is 256 megabytes.
`-s`	*Blocks*	Specifies the journal-segment size in file system blocks.
`-t`	*LockTableName*	This parameter has two parts separated by a colon (no spaces) as follows: *ClusterName:FSName*. *ClusterName* is set in the `cluster.ccs` file (`cluster.ccs:cluster/name`). *FSName*, the file system name, can be 1 to 16 characters in length, and the name must be unique among all file systems in the cluster.
`-V`		Displays command version information, then exits.

Table 9-1. Command Options: `gfs_mkfs`

9.2. Mounting a File System

Before you can mount a GFS file system, the file system must exist (refer to Section 9.1 *Making a File System*), the pool volume where the file system exists must be activated, and the supporting clustering and locking systems must be started (refer to

Chapter 4 *Initial Configuration*). After those requirements have been met, you can mount the GFS file system as you would any Linux file system.

To manipulate file ACLs, you must mount the file system with the `-o acl` mount option. If a file system is mounted without the `-o acl` mount option, users are allowed to view ACLs (with `getfacl`), but are not allowed to set them (with `setfacl`).

9.2.1. Usage

Mounting Without ACL Manipulation

```
mount -t gfs BlockDevice MountPoint
```

Mounting With ACL Manipulation

```
mount -t gfs -o acl BlockDevice MountPoint
```

`-o acl`

GFS-specific option to allow manipulating file ACLs.

`BlockDevice`

Specifies the block device where the GFS file system resides.

`MountPoint`

Specifies the directory where the GFS file system should be mounted.

9.2.2. Example

In this example, the GFS file system on the `pool0` block device is mounted on the `/gfs1/` directory.

```
mount -t gfs /dev/pool/pool0 /gfs1
```

9.2.3. Complete Usage

```
mount -t gfs BlockDevice MountPoint -o option
```

The `-o` *`option`* consists of GFS-specific options (refer to Table 9-2) or acceptable standard Linux `mount -o` options, or a combination of both. Multiple *`option`* parameters are separated by a comma and no spaces.

Note

The `mount` command is a Linux system command. In addition to using GFS-specific options described in this section, you can use other, standard, `mount` command options (for example, `-r`). For information about other Linux `mount` command options, see the Linux `mount` man page.

Table 9-2 describes the available GFS-specific options that can be passed to GFS at mount time.

Option	Description
`-o acl`	Allows manipulating file ACLs. If a file system is mounted without the `-o acl` mount option, users are allowed to view ACLs (with `getfacl`), but are not allowed to set them (with `setfacl`).
`hostdata=`*`nodename`*	LOCK_GULM file systems use this information to set the local node name, overriding the usual selection of node name from `uname -n`.
`lockproto=`*`LockModuleName`*	Allows the user to specify which locking protocol to use with the file system. If *`LockModuleName`* is not specified, the locking protocol name is read from the file system superblock.
`locktable=`*`LockTableName`*	Allows the user to specify which locking table to use with the file system.
`upgrade`	Upgrade the on-disk format of the file system so that it can be used by newer versions of GFS.
`ignore_local_fs` **Caution:** This option should *not* be used when GFS file systems are shared.	Forces GFS to treat the file system as a multihost file system. By default, using LOCK_NOLOCK automatically turns on the `localcaching` and `localflocks` flags.
`localcaching` **Caution:** This option should *not* be used when GFS file systems are shared.	Tells GFS that it is running as a local file system. GFS can then turn on selected optimization capabilities that are not available when running in cluster mode. The `localcaching` flag is automatically turned on by LOCK_NOLOCK.
`localflocks` **Caution:** This option should not be used when GFS file systems are shared.	Tells GFS to let the VFS (virtual file system) layer do all flock and fcntl. The `localflocks` flag is automatically turned on by LOCK_NOLOCK.

Table 9-2. GFS-Specific Mount Options

9.3. Unmounting a File System

The GFS file system can be unmounted the same way as any Linux file system.

Note

The `umount` command is a Linux system command. Information about this command can be found in the Linux `umount` command man pages.

9.3.1. Usage

```
umount MountPoint
```

MountPoint

Specifies the directory where the GFS file system should be mounted.

9.4. GFS Quota Management

File system quotas are used to limit the amount of file-system space a user or group can use. A user or group does not have a quota limit until one is set. GFS keeps track of the space used by each user and group even when there are no limits in place. GFS updates quota information in a transactional way so system crashes do not require quota usages to be reconstructed.

To prevent a performance slowdown, a GFS node synchronizes updates to the quota file only periodically. The "fuzzy" quota accounting can allow users or groups to slightly exceed the set limit. To minimize this, GFS dynamically reduces the synchronization period as a "hard" quota limit is approached.

GFS uses its `gfs_quota` command to manage quotas. Other Linux quota facilities cannot be used with GFS.

9.4.1. Setting Quotas

Two quota settings are available for each user ID (UID) or group ID (GID): a *hard limit* and a *warn limit*.

A hard limit is the amount space that can be used. The file system will not let the user or group use more than that amount of disk space. A hard limit value of *zero* means that no limit is enforced.

A warn limit is usually a value less than the hard limit. The file system will notify the user or group when the warn limit is reached to warn them of the amount of space they are using. A warn limit value of *zero* means that no limit is enforced.

Limits are set using the `gfs_quota` command. The command only needs to be run on a single node where GFS is mounted.

9.4.1.1. Usage

Setting Quotas, Hard Limit

```
gfs_quota limit -u User -l Size -f MountPoint
```

```
gfs_quota limit -g Group -l Size -f MountPoint
```

Setting Quotas, Warn Limit

```
gfs_quota warn -u User -l Size -f MountPoint
```

```
gfs_quota warn -g Group -l Size -f MountPoint
```

User

A user ID to limit or warn. It can be either a user name from the password file or the UID number.

Group

A group ID to limit or warn. It can be either a group name from the group file or the GID number.

Size

Specifies the new value to limit or warn. By default, the value is in units of megabytes. The additional -k, -s and -b flags change the units to kilobytes, sectors, and file-system blocks, respectively.

MountPoint

Specifies the GFS file system to which the actions apply.

9.4.1.2. Examples

This example sets the hard limit for user *Bert* to 1024 megabytes (1 gigabyte) on file system /gfs.

```
gfs_quota limit -u Bert -l 1024 -f /gfs
```

This example sets the warn limit for group ID 21 to 50 kilobytes on file system /gfs.

```
gfs_quota warn -g 21 -l 50 -k -f /gfs
```

9.4.2. Displaying Quota Limits and Usage

Quota limits and current usage can be displayed for a specific user or group using the gfs_quota get command. The entire contents of the quota file can also be displayed using the gfs_quota list command, in which case all IDs with a non-zero hard limit, warn limit, or value are listed.

9.4.2.1. Usage

Displaying Quota Limits for a User

```
gfs_quota get -u User -f MountPoint
```

Displaying Quota Limits for a Group

```
gfs_quota get -g Group -f MountPoint
```

Displaying Entire Quota File

```
gfs_quota list -f MountPoint
```

User

A user ID to display information about a specific user. It can be either a user name from the password file or the UID number.

Group

A group ID to display information about a specific group. It can be either a group name from the group file or the GID number.

MountPoint

Specifies the GFS file system to which the actions apply.

9.4.2.2. Command Output

GFS quota information from the `gfs_quota` command is displayed as follows:

```
user User: limit:LimitSize warn:WarnSize value:Value

group Group: limit:LimitSize warn:WarnSize value:Value
```

The *LimitSize*, *WarnSize*, and *Value* numbers (values) are in units of megabytes by default. Adding the `-k`, `-s`, or `-b` flags to the command line change the units to kilobytes, sectors, or file system blocks, respectively.

User

A user name or ID to which the data is associated.

Group

A group name or ID to which the data is associated.

LimitSize

The hard limit set for the user or group. This value is zero if no limit has been set.

Value

The actual amount of disk space used by the user or group.

9.4.2.3. Comments

When displaying quota information, the `gfs_quota` command does not resolve UIDs and GIDs into names if the `-n` option is added to the command line.

Space allocated to GFS's hidden files can be left out of displayed values for the root UID and GID by adding the `-d` option to the command line. This is useful when trying to match the numbers from `gfs_quota` with the results of a `du` command.

9.4.2.4. Examples

This example displays quota information for all users and groups that have a limit set or are using any disk space on file system `/gfs`.

```
gfs_quota list -f /gfs
```

This example displays quota information in sectors for group `users` on file system `/gfs`.

```
gfs_quota get -g users -f /gfs -s
```

9.4.3. Synchronizing Quotas

GFS stores all quota information in its own internal file on disk. A GFS node does not update this quota file for every file-system write; rather, it updates the quota file once every 60 seconds. This is necessary to avoid contention among nodes writing to the quota file, which would cause a slowdown in performance.

As a user or group approaches their quota limit, GFS dynamically reduces the time between its quota-file updates to prevent the limit from being exceeded. The normal time period between quota synchronizations is a tunable parameter, `quota_quantum`, and can be changed using the `gfs_tool` command. By default, the time period is 60 seconds. Also, the `quota_quantum` parameter must be set on each node and each time the file system is mounted. (Changes to the `quota_quantum` parameter are not persistent across unmounts.)

You can use the `gfs_quota sync` command to synchronize the quota information from a node to the on-disk quota file between the automatic updates performed by GFS.

9.4.3.1. Usage

Synchronizing Quota Information

```
gfs_quota sync -f MountPoint
```

MountPoint

Specifies the GFS file system to which the actions apply.

Tuning the Time Between Synchronizations

```
gfs_tool settune MountPoint quota_quantum Seconds
```

MountPoint

Specifies the GFS file system to which the actions apply.

Seconds

Specifies the new time period between regular quota-file synchronizations by GFS. Smaller values may increase contention and slow down performance.

9.4.3.2. Examples

This example synchronizes the quota information from the node it is run on to file system `/gfs`.

```
gfs_quota sync -f /gfs
```

This example changes the default time period between regular quota-file updates to one hour (3600 seconds) for file system `/gfs` on a single node.

```
gfs_tool settune /gfs quota_quantum 3600
```

9.4.4. Disabling/Enabling Quota Enforcement

Enforcement of quotas can be disabled for a file system without clearing the limits set for all users and groups. Enforcement can also be enabled. Disabling and enabling of quota enforcement is done by changing a tunable parameter, `quota_enforce`, with the `gfs_tool` command. The `quota_enforce` parameter must be disabled or enabled on each node where quota enforcement should be disabled/enabled. Each time the file system is mounted, enforcement is enabled by default. (Disabling is not persistent across unmounts.)

9.4.4.1. Usage

```
gfs_tool settune MountPoint quota_enforce {0|1}
```

`MountPoint`

Specifies the GFS file system to which the actions apply.

`quota_enforce {0|1}`

0 = disabled

1 = enabled

9.4.4.2. Comments

A value of 0 disables enforcement. Enforcement can be enabled by running the command with a value of 1 (instead of 0) as the final command line parameter. Even when GFS is not enforcing quotas, it still keeps track of the file system usage for all users and groups so that quota-usage information does not require rebuilding after re-enabling quotas.

9.4.4.3. Examples

This example *disables* quota enforcement on file system `/gfs`.

```
gfs_tool settune /gfs quota_enforce 0
```

This example *enables* quota enforcement on file system `/gfs`.

```
gfs_tool settune /gfs quota_enforce 1
```

9.4.5. Disabling/Enabling Quota Accounting

By default, GFS keeps track of disk usage for every user and group even when no quota limits have been set. This accounting incurs some overhead that is unnecessary if quotas are not used. This quota accounting can be completely disabled by setting the `quota_account` tunable parameter to 0. This must be done on each node and after each mount. (The 0 setting is not persistent across unmounts.) Quota accounting can be enabled by setting the `quota_account` tunable parameter to 1.

9.4.5.1. Usage

```
gfs_tool settune MountPoint quota_account {0|1}
```

MountPoint

Specifies the GFS file system to which the actions apply.

quota_account {0|1}

0 = disabled

1 = enabled

9.4.5.2. Comments

To enable quota accounting on a file system, the quota_account parameter must be set back to 1. Afterward, the GFS quota file must be initialized to account for all current disk usage for users and groups on the file system. The quota file is initialized by running: gfs_quota init -f *MountPoint*.

Note

Initializing the quota file requires scanning the entire file system and may take a long time.

9.4.5.3. Examples

This example *disables* quota accounting on file system /gfs on a single node.

```
gfs_tool settune /gfs quota_account 0
```

This example enables quota accounting on file system /gfs on a single node and initializes the quota file.

```
gfs_tool settune /gfs quota_account 1

gfs_quota init -f /gfs
```

9.5. Growing a File System

The gfs_grow command is used to expand a GFS file system after the device where the file system resides has been expanded. Running a gfs_grow command on an existing GFS file system fills all spare space between the current end of the file system and the end of the device with a newly initialized GFS file system extension. When the fill operation is completed, the resource index for the file system is updated. All nodes in the cluster can then use the extra storage space that has been added.

The gfs_grow command can only be run on a mounted file system, but only needs to be run on one node in the cluster. All the other nodes sense that the expansion has occurred and automatically start using the new space.

To verify that the changes were successful, you can use the gfs_grow command with the -T (test) and -v (verbose) flags. Running the command with those flags displays the current state of the mounted GFS file system.

9.5.1. Usage

```
gfs_grow MountPoint
```

MountPoint

Specifies the GFS file system to which the actions apply.

9.5.2. Comments

Before running the `gfs_grow` command:

- Back up important data on the file system.
- Display the pool volume that is used by the file system to be expanded by running a `gfs_tool df` *`MountPoint`* command.
- Expand the underlying pool volume with a `pool_tool -g command`. Refer to Section 5.8 *Growing a Pool Volume* for additional information.

After running the `gfs_grow` command, run a `df` command to check that the new space is now available in the file system.

9.5.3. Examples

In this example, the file system on the `/gfs1/` directory is expanded.

```
gfs_grow /gfs1
```

In this example, the state of the mounted file system is checked.

```
gfs_grow -Tv /gfs1
```

9.5.4. Complete Usage

```
gfs_grow [Options] {MountPoint | Device} [MountPoint | Device]
```

MountPoint

Specifies the directory where the GFS file system is mounted.

Device

Specifies the device node of the file system.

Table 9-3 describes the GFS-specific options that can be used while expanding a GFS file system.

Option	Description
-h	Help. Display a short usage message, then exist.
-q	Quiet. Turn down the verbosity level.
-T	Test. Do all calculations, but do not write any data to the disk and do not expand the file system.
-V	Display command version information, then exit.
-v	Turn up the verbosity of messages.

Table 9-3. GFS-specific Options Available While Expanding A File System

9.6. Adding Journals to a File System

The `gfs_jadd` command is used to add journals to a GFS file system after the device where the file system resides has been expanded. Running a `gfs_jadd` command on a GFS file system uses space between the current end of the file system and the end of the device where the file system resides. When the fill operation is completed, the journal index is updated.

The `gfs_jadd` command can only be run on a mounted file system, but it only needs to be run on one node in the cluster. All the other nodes sense that the expansion has occurred.

To verify that the changes were successful, you can use the `gfs_jadd` command with the `-T` (test) and `-v` (verbose) flags. Running the command with those flags displays the current state of the mounted GFS file system.

9.6.1. Usage

```
gfs_jadd -j Number MountPoint
```

`Number`

Specifies the number of new journals to be added.

`MountPoint`

Specifies the directory where the GFS file system is mounted.

9.6.2. Comments

Before running the `gfs_jadd` command:

- Back up important data on the file system.
- Run a `gfs_tool df` *`MountPoint`* command to display the pool volume used by the file system where journals will be added.
- Expand the underlying pool volume with a `pool_tool -g` command. Refer to Section 5.8 *Growing a Pool Volume* for additional information.

After running the `gfs_jadd` command, run a `gfs_jadd` command with the `-T` and `-v` flags enabled to check that the new journals have been added to the file system.

9.6.3. Examples

In this example, one journal is added to the file system on the `/gfs1/` directory.

`gfs_jadd -j1 /gfs1`

In this example, two journals are added to the file system on the `/gfs1/` directory.

`gfs_jadd -j2 /gfs1`

In this example, the current state of the file system on the `/gfs1/` directory can be checked for the new journals.

`gfs_jadd -Tv /gfs1`

9.6.4. Complete Usage

`gfs_jadd [`*`Options`*`] {`*`MountPoint`*` | `*`Device`*`} [`*`MountPoint`*` | `*`Device`*`]`

`MountPoint`

Specifies the directory where the GFS file system is mounted.

`Device`

Specifies the device node of the file system.

Table 9-4 describes the GFS-specific options that can be used when adding journals to a GFS file system.

Flag	Parameter	Description
`-h`		Help. Displays short usage message, then exits.
`-J`	*`MegaBytes`*	Specifies the size of the new journals in MBytes. Default journal size is 128 MBytes. The minimum size is 32 MBytes. To add journals of different sizes to the file system, the `gfs_jadd` command must be run for each size journal. The size specified is rounded down so that it is a multiple of the journal-segment size that was specified when the file system was created.
`-j`	*`Number`*	Specifies the number of new journals to be added by the `gfs_jadd` command. The default value is 1.
`-T`		Test. Do all calculations, but do not write any data to the disk and do not add journals to the file system. Enabling this flag helps discover what the `gfs_jadd` command would have done if it were run without this flag. Using the `-v` flag with the `-T` flag turns up the verbosity level to display more information.
`-q`		Quiet. Turns down the verbosity level.
`-V`		Display command version information, then exit.
`-v`		Turn up the verbosity of messages.

Table 9-4. GFS-specific Options Available When Adding Journals

9.7. Direct I/O

Direct I/O is a feature of the file system whereby file reads and writes go directly from the applications to the storage device, bypassing the operating system read and write caches. Direct I/O is used by only a few applications that manage their own caches, such as databases.

Direct I/O is invoked by an application opening a file with the `O_DIRECT` flag. Alternatively, GFS can attach a direct I/O attribute to a file, in which case direct I/O is used regardless of how the file is opened.

When a file is opened with `O_DIRECT`, or when a GFS direct I/O attribute is attached to a file, all I/O operations must be done in block-size multiples of 512 bytes. The memory being read from or written to must also be 512-byte aligned.

One of the following methods can be used to enable direct I/O on a file:

- `O_DIRECT`
- GFS file attribute
- GFS directory attribute

9.7.1. `O_DIRECT`

If an application uses the `O_DIRECT` flag on an `open()` system call, direct I/O is used for the opened file.

To cause the `O_DIRECT` flag to be defined with recent glibc libraries, define `_GNU_SOURCE` at the beginning of a source file before any includes, or define it on the **cc** line when compiling.

Note

Linux kernels from some distributions do not support use of the `O_DIRECT` flag.

9.7.2. GFS File Attribute

The `gfs_tool` command can be used to assign a direct I/O attribute flag, `directio`, to a regular GFS file. The `directio` flag can also be cleared.

9.7.2.1. Usage

Set Direct I/O Attribute Flag

```
gfs_tool setflag directio File
```

Clear Direct I/O Attribute Flag

```
gfs_tool clearflag directio File
```

`File`

Specifies the file where the `directio` flag is assigned.

9.7.2.2. Example

In this example, the command sets the `directio` flag on the file named `datafile` in directory `/gfs1/`.

```
gfs_tool setflag directio /gfs1/datafile
```

9.7.3. GFS Directory Attribute

The `gfs_tool` command can be used to assign a direct I/O attribute flag, `inherit_directio`, to a GFS directory. Enabling the `inherit_directio` flag on a directory causes all newly created regular files in that directory to automatically inherit the `directio` flag. Also, the `inherit_directio` flag is inherited by any new subdirectories created in the directory. The `inherit_directio` flag can also be cleared.

9.7.3.1. Usage

Setting the `inherit_directio` flag

```
gfs_tool setflag inherit_directio Directory
```

Setting the `inherit_directio` flag

```
gfs_tool clearflag inherit_directio Directory
```

`Directory`

Specifies the directory where the `inherit_directio` flag is set.

9.7.3.2. Example

In this example, the command sets the `inherit_directio` flag on the directory named `/gfs1/data/`.

```
gfs_tool setflag inherit_directio /gfs1/data/
```

9.8. Data Journaling

Ordinarily, GFS writes only metadata to its journal. File contents are subsequently written to disk by the kernel's periodic sync that flushes file system buffers. An `fsync()` call on a file causes the file's data to be written to disk immediately. The call returns when the disk reports that all data is safely written.

Data journaling can result in a reduced `fsync()` time, especially for small files, because the file data is written to the journal in addition to the metadata. An `fsync()` returns as soon as the data is written to the journal, which can be substantially faster than the time it takes to write the file data to the main file system.

Applications that rely on `fsync()` to sync file data may see improved performance by using data journaling. Data journaling can be enabled automatically for any GFS files created in a flagged directory (and all its subdirectories). Existing files with zero length can also have data journaling turned on or off.

Using the `gfs_tool` command, data journaling is enabled on a directory (and all its subdirectories) or on a zero-length file by setting the `inherit_jdata` or `jdata` attribute flags to the directory or file, respectively. The directory and file attribute flags can also be cleared.

9.8.1. Usage

Setting and Clearing the `inherit_jdata` Flag

```
gfs_tool setflag inherit_jdata Directory
gfs_tool clearflag inherit_jdata Directory
```

Setting and Clearing the `jdata` Flag

```
gfs_tool setflag jdata File
gfs_tool clearflag jdata File
```

Directory

Specifies the directory where the flag is set or cleared.

File

Specifies the zero-length file where the flag is set or cleared.

9.8.2. Examples

This example shows setting the `inherit_jdata` flag on a directory. All files created in the directory or any of its subdirectories will have the `jdata` flag assigned automatically. Any data written to the files will be journaled.

```
gfs_tool setflag inherit_jdata /gfs1/data/
```

This example shows setting the `jdata` flag on a file. The file must be zero size. Any data written to the file will be journaled.

```
gfs_tool setflag jdata /gfs1/datafile
```

9.9. Configuring `atime` Updates

Each file inode and directory inode has three time stamps associated with it:

- `ctime` — The last time the inode status was changed
- `mtime` — The last time the file (or directory) data was modified
- `atime` — The last time the file (or directory) data was accessed

Note

For more information about `ctime`, `mtime`, and `atime` updates, refer to the `stat(2)` man page.

If `atime` updates are enabled as they are by default on GFS and other Linux file systems then every time a file is read, its inode needs to be updated.

Because few applications use the information provided by `atime`, those updates can require a significant amount of unnecessary write traffic and file-locking traffic. That traffic can degrade performance; therefore, it may be preferable to turn off `atime` updates.

Two methods of reducing the effects of `atime` updating are available:

- Mount with `noatime`
- Tune GFS `atime` quantum

9.9.1. Mount with `noatime`

A standard Linux mount option, `noatime`, may be specified when the file system is mounted, which disables `atime` updates on that file system.

9.9.1.1. Usage

```
mount -t gfs BlockDevice MountPoint -o noatime
```

`BlockDevice`

Specifies the block device where the GFS file system resides.

`MountPoint`

Specifies the directory where the GFS file system should be mounted.

9.9.1.2. Example

In this example, the GFS file system resides on the `pool0` block device and is mounted on directory `/gfs1/` with atime updates turned off.

```
mount -t gfs /dev/pool/pool0 /gfs1 -o noatime
```

9.9.2. Tune GFS `atime` Quantum

When `atime` updates are enabled, GFS (by default) only updates them once an hour. The time quantum is a tunable parameter that can be adjusted using the `gfs_tool` command.

Each node in a GFS cluster updates the access time based on the difference between its system time and the time recorded in the inode. It is required that system clocks of all nodes in a GFS cluster be in sync. If a node's system time is out of sync by a significant fraction of the tunable parameter, `atime_quantum`, then `atime` updates are written more frequently. Increasing the frequency of `atime` updates may cause performance degradation in clusters with heavy work loads.

By using the `gettune` action flag of the `gfs_tool` command, all current tunable parameters including `atime_quantum` (default is 3600 seconds) are displayed.

The `gfs_tool settune` command is used to change the `atime_quantum` parameter value. It must be set on each node and each time the file system is mounted. (The setting is not persistent across unmounts.)

9.9.2.1. Usage

Displaying Tunable Parameters

```
gfs_tool gettune MountPoint
```

MountPoint

Specifies the directory where the GFS file system is mounted.

Changing the `atime_quantum` Parameter Value

```
gfs_tool settune MountPoint atime_quantum Seconds
```

MountPoint

Specifies the directory where the GFS file system is mounted.

Seconds

Specifies the update period in seconds.

9.9.2.2. Examples

In this example, all GFS tunable parameters for the file system on the mount point `/gfs1` are displayed.

```
gfs_tool gettune /gfs1
```

In this example, the `atime` update period is set to once a day (86,400 seconds) for the GFS file system on mount point `/gfs1`.

```
gfs_tool settune /gfs1 atime_quantum 86400
```

9.10. Suspending Activity on a File System

All write activity to a file system can be suspended for a time by using the `gfs_tool` command's `freeze` action. The `unfreeze` action returns the file system to its ordinary state. That feature allows hardware-based device snapshots to be used to capture the file system in a consistent state.

9.10.1. Usage

Freeze Activity

```
gfs_tool freeze MountPoint
```

Unfreeze Activity

```
gfs_tool unfreeze MountPoint
```

MountPoint

Specifies the file system to freeze or unfreeze.

9.10.2. Examples

This example freezes file system `/gfs`.

```
gfs_tool freeze /gfs
```

This example unfreezes file system `/gfs`.

```
gfs_tool unfreeze /gfs
```

9.11. Displaying Extended GFS Information and Statistics

A variety of details can be gathered about GFS using the `gfs_tool` command. Typical usage of the `gfs_tool` command is described here.

9.11.1. Usage

Displaying Statistics

```
gfs_tool counters MountPoint
```

The `counters` action flag displays statistics about a file system. If `-c` is used, the `gfs_tool` command continues to run, displaying statistics once per second.

Displaying Space Usage

```
gfs_tool df MountPoint
```

The `df` action flag displays a space-usage summary of a given file system. The information is more detailed than a standard `df`.

Displaying Extended Status

```
gfs_tool stat File
```

The `stat` action flag displays extended status information about a file.

MountPoint

Specifies the file system to which the action applies.

File

Specifies the file from which to get information.

The gfs_tool command provides additional action flags (options) not listed in this section. For more information about other gfs_tool action flags, refer to the gfs_tool man page.

9.11.2. Examples

This example reports extended file system usage about file system /gfs.

```
gfs_tool df /gfs
```

This example reports extended file status about file /gfs/datafile.

```
gfs_tool stat /gfs/datafile
```

9.12. Repairing a File System

When nodes fail with the file system mounted, file system journaling allows fast recovery. However, if a storage device loses power or is physically disconnected, file system corruption may occur. (Journaling cannot be used to recover from storage subsystem failures.) When that type of corruption occurs, the GFS file system can be recovered by using the gfs_fsck command.

The gfs_fsck command must only be run on a file system that is unmounted from all nodes.

Note

On nodes running Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5 and later, the gfs_fsck command has changed from previous releases of Red Hat GFS in the following ways:

- You can no longer set the interactive mode with [Ctrl]-[C]. Pressing [Ctrl]-[C] now cancels the gfs_fsck command. Do *not* press [Ctr]-[C] unless you want to cancel the command.
- You can increase the level of verbosity by using the -v flag. Adding a second -v flag increases the level again.
- You can decrease the level of verbosity by using the -q flag. Adding a second -q flag decreases the level again.
- The -n option opens a file system as read-only and answers **no** to any queries automatically. The option provides a way of trying the command to reveal errors without actually allowing the gfs_fsck command to take effect.

Refer to the gfs_fsck man page, gfs_fsck(8), for additional information about other command options.

9.12.1. Usage

gfs_fsck -y *BlockDevice*

`-y`

The `-y` flag causes all questions to be answered with `yes`. With the `-y` specified, the `gfs_fsck` does not prompt you for an answer before making changes.

`BlockDevice`

Specifies the block device where the GFS file system resides.

9.12.2. Example

In this example, the GFS file system residing on block device `/dev/pool/pool0` is repaired. All queries to repair are automatically answered with `yes`.

```
gfs_fsck -y /dev/pool/pool0
```

9.13. Context-Dependent Path Names

Context-Dependent Path Names (CDPNs) allow symbolic links to be created that point to variable destination files or directories. The variables are resolved to real files or directories each time an application follows the link. The resolved value of the link depends on the node or user following the link.

CDPN variables can be used in any path name, not just with symbolic links. However, the CDPN variable name cannot be combined with other characters to form an actual directory or file name. The CDPN variable must be used alone as one segment of a complete path.

9.13.1. Usage

For a Normal Symbolic Link

```
ln -s Target LinkName
```

`Target`

Specifies an existing file or directory on a file system.

`LinkName`

Specifies a name to represent the real file or directory on the other end of the link.

For a Variable Symbolic Link

```
ln -s Variable LinkName
```

`Variable`

Specifies a special reserved name from a list of values (refer to Table 9-5) to represent one of multiple existing files or directories. This string is not the name of an actual file or directory itself. (The real files or directories must be created in a separate step using names that correlate with the type of variable used.)

LinkName

Specifies a name that will be seen and used by applications and will be followed to get to one of the multiple real files or directories. When *LinkName* is followed, the destination depends on the type of variable and the node or user doing the following.

Variable	Description
`@hostname`	This variable resolves to a real file or directory named with the hostname string produced by the following command entry: `echo ‘uname -n‘`
`@mach`	This variable resolves to a real file or directory name with the machine-type string produced by the following command entry: `echo ‘uname -m‘`
`@os`	This variable resolves to a real file or directory named with the operating-system name string produced by the following command entry: `echo ‘uname -s‘`
`@sys`	This variable resolves to a real file or directory named with the combined machine type and OS release strings produced by the following command entry: `echo ‘uname -m‘_‘uname -s‘`
`@uid`	This variable resolves to a real file or directory named with the user ID string produced by the following command entry: `echo ‘id -u‘`
`@gid`	This variable resolves to a real file or directory named with the group ID string produced by the following command entry: `echo ‘id -g‘`

Table 9-5. CDPN *`Variable`* Values

9.13.2. Example

In this example, there are three nodes with hostnames `n01`, `n02` and `n03`. Applications on each node uses directory `/gfs/log/`, but the administrator wants these directories to be separate for each node. To do this, no actual log directory is created; instead, a `@hostname` CDPN link is created with the name `log`. Individual directories `/gfs/n01/`, `/gfs/n02/`, and `/gfs/n03/` are created that will be the actual directories used when each node references `/gfs/log/`.

```
n01# cd /gfs
n01# mkdir n01 n02 n03
n01# ln -s @hostname log

n01# ls -l /gfs
lrwxrwxrwx 1 root root 9 Apr 25 14:04 log -> @hostname/
drwxr-xr-x 2 root root 3864 Apr 25 14:05 n01/
drwxr-xr-x 2 root root 3864 Apr 25 14:06 n02/
drwxr-xr-x 2 root root 3864 Apr 25 14:06 n03/
```

```
n01# touch /gfs/log/fileA
n02# touch /gfs/log/fileB
n03# touch /gfs/log/fileC

n01# ls /gfs/log/
fileA
n02# ls /gfs/log/
fileB
n03# ls /gfs/log/
fileC
```

9.14. Shutting Down a GFS Cluster

To cleanly shut down a GFS cluster, perform the following steps:

1. Unmount all GFS file systems on all nodes. Refer to Section 9.3 *Unmounting a File System* for more information.
2. Shut down all LOCK_GULM servers. Refer to Section 8.2.5 *Shutting Down a LOCK_GULM Server* for more information.
3. Kill the CCS daemon on all nodes.
4. Deactivate all pools on all nodes. Refer to Section 5.6 *Activating/Deactivating a Pool Volume* for more information.

Note

You can use GFS `init.d` scripts included with GFS to shut down nodes in a GFS cluster. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS `init.d` Scripts*.

9.15. Starting a GFS Cluster

When starting a GFS cluster, perform the following steps.

Note

You can use GFS `init.d` scripts included with GFS to start nodes in a GFS cluster. For more information about GFS `init.d` scripts, refer to Chapter 12 *Using GFS `init.d` Scripts*.

Note

The GFS kernel modules must be loaded prior to performing these steps. Refer to Section 3.2.2 *Loading the GFS Kernel Modules* for more information.

1. At each node, activate pools. Refer to Section 5.6 *Activating/Deactivating a Pool Volume* for more information.

   Command usage: `pool_assemble`

2. At each node, start the CCS daemon, specifying the CCA device on at the command line. Refer to Section 7.2 *Starting CCS in the Cluster* for more information.

   Command usage: `ccsd -d` *`CCADevice`*

3. Start the LOCK_GULM servers. At each lock server node, start `lock_gulmd`. Refer to Section 8.2.3 *Starting LOCK_GULM Servers* for more information.

   Command usage: `lock_gulmd`

4. At each node, mount the GFS file systems. Refer to Section 9.2 *Mounting a File System* for more information.

   Command usage: `mount -t gfs` *`BlockDevice MountPoint`*

Chapter 10.

Using the Fencing System

Fencing (or *I/O fencing*) is the mechanism that disables an errant GFS node's access to a file system, preventing the node from causing data corruption. This chapter explains the necessity of fencing, summarizes how the fencing system works, and describes each form of fencing that can be used in a GFS cluster. The chapter consists of the following sections:

- Section 10.1 *How the Fencing System Works*
- Section 10.2 *Fencing Methods*

10.1. How the Fencing System Works

Fencing consists of two main steps:

- Removal — Cutting an errant node off from contact with the storage
- Recovery — Returning the node safely back into the cluster.

A cluster manager monitors the heartbeat between GFS nodes to determine which nodes are running properly and which nodes are errant in a GFS cluster. (A cluster manager is part of the LOCK_GULM server). If a node fails, the cluster manager fences the node, then communicates to the lock manager and GFS to perform recovery of the failed node.

If a node falls out of contact (losing heartbeat) with the rest of the cluster, the locks it holds and the corresponding parts of the file system are unavailable to the rest of the nodes in the cluster. Eventually, that condition may bring the entire cluster to a halt as other nodes require access to those parts of the file system.

If a node fails, it cannot be permitted to rejoin the cluster while claiming the locks it held when the node failed. Otherwise, that node could write to a file system where another node — that legitimately has been issued locks to write to the file system — is writing, therefore corrupting the data. Fencing prevents a failed node from rejoining a cluster with invalid locks by disabling the path between the node and the file system storage.

When the cluster manager fences a node, it directs the fencing system to fence the node by name. The fencing system must read from CCS the appropriate method of fencing the node. Refer to Chapter 7 *Using the Cluster Configuration System* for details on how to specify each fencing method in the CCS configuration files.

Each device or method that can be used to fence nodes is listed in `fence.ccs` under `fence_devices`. Each device specification includes the name of a fencing agent. The fencing agent is a command that interacts with a specific type of device to disable a specific node. In order to use a device for fencing, an associated fence agent must exist.

10.2. Fencing Methods

Table 10-1 lists the fencing methods and associated fencing agents that you can use with GFS.

Fending Method	Fencing Agent
APC Network Power Switch	`fence_apc`
WTI Network Power Switch	`fence_wti`
Brocade FC Switch	`fence_brocade`
McData FC Switch	`fence_mcdata`
Vixel FC Switch	`fence_vixel`
HP RILOE	`fence_rib`
GNBD	`fence_gnbd`
xCAT	`fence_xcat`
Manual	`fence_manual`

Table 10-1. Fencing Methods and Agents

Warning

Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.

When a GFS cluster is operating, the fencing system executes those fencing agents. Specifically, when using LOCK_GULM, the cluster manager is the master `lock_gulmd` daemon. The daemon uses the `fence_node` command to dispatch a fencing agent.

Note

Contact an authorized Red Hat support representative if there is a device you wish to use for fencing that is not described in the following sections.

The following sections describe the fencing methods available with GFS.

10.2.1. APC MasterSwitch

APC MasterSwitch power switches are used to power cycle nodes that need to be fenced. The fencing agent, `fence_apc`, logs into the device and reboots the specific port for the failed node. The `fence_apc` fencing agent supports nodes with dual power supplies plugged into an APC MasterSwitch. Support for nodes with dual power supplies allows powering down both power supplies in a node, thereby allowing fencing of nodes with dual power supplies. Refer to Section 6.6 *Creating the `fence.ccs` File* and Section 6.7 *Creating the `nodes.ccs` File* for information on how to configure with this type of fencing.

Note

Lengthy Telnet connections to the APC should be avoided during the cluster operation. A fencing operation trying to use the APC will be blocked until it can log in.

10.2.2. WTI Network Power Switch

WTI network power switches (NPSs) are used to power cycle nodes that need to be fenced. The fencing agent, `fence_wti`, logs into the device and reboots the specific port for the offline node. The `fence_wti` fencing agent does not support nodes with dual power supplies plugged into a WTI NPS. Refer to Section 6.6 *Creating the `fence.ccs` File* and Section 6.7 *Creating the `nodes.ccs` File* for information on how to configure with this type of fencing.

Note

Lengthy Telnet connections to the WTI NPS should be avoided during the cluster operation. A fencing operation trying to use the WTI NPS will be blocked until it can log in.

10.2.3. Brocade FC Switch

A node connected to a Brocade FC (Fibre Channel) switch can be fenced by disabling the switch port that the node is connected to. The fencing agent, `fence_brocade`, logs into the switch and disables the specific port associated with the node.

Nodes with multiple FC paths can have each path disabled. Refer to Section 6.6 *Creating the `fence.ccs` File* and Section 6.7 *Creating the `nodes.ccs` File* for information on how to configure with this type of fencing.

Note

Lengthy Telnet connections to the switch should be avoided during the cluster operation. A fencing operation trying to use the switch will be blocked until it can log in.

10.2.4. Vixel FC Switch

A node connected to a Vixel FC (Fibre Channel) switch can be fenced by disabling the switch port that the node is connected to. The fencing agent, `fence_vixel`, logs into the switch and disables the specific port associated with the node. Nodes with multiple FC paths can have each path disabled. Refer to Section 6.6 *Creating the `fence.ccs` File* and Section 6.7 *Creating the `nodes.ccs` File* for information on how to configure with this type of fencing.

Note

Lengthy Telnet connections to the switch should be avoided during the cluster operation. A fencing operation trying to use the switch will be blocked until it can log in.

Caution

Red Hat GFS does not support the following Vixel firmware: Vixel 7xxx series firmware versions 4.0 or later, Vixel 9xxx series firmware versions 6.0 or later.

10.2.5. HP RILOE Card

A GFS node that has an HP RILOE (Remote Insight Lights-Out Edition) card can be fenced with the `fence_rib` fencing agent. Refer to Section 6.6 *Creating the `fence.ccs` File* and Section 6.7 *Creating the `nodes.ccs` File* for information on how to configure with this type of fencing.

Note

The `fence_rib` fencing agent requires the **Stunnel** software be installed on the system running the fencing agent. **Stunnel** is required to connect to the HP RILOE card.

10.2.6. GNBD

Nodes that only use GFS with storage devices via GNBD (Global Network Block Device) servers can use the `fence_gnbd` agent. The agent requires no special hardware. The `fence_gnbd` fencing agent instructs all GNBD servers to disallow all I/O from a fenced node. When a fenced node is reset and re-imports the GNBD devices, the GNBD servers again allow the node access to the devices. Refer to Section 6.6 *Creating the `fence.ccs` File* and Section 6.7 *Creating the `nodes.ccs` File* for information on how to configure with this type of fencing.

Note

You *must not* specify the GNBD fencing agent (`fence_gnbd`) as a fencing device for GNBD server nodes (nodes that export GNBDs to GFS nodes).

10.2.7. Manual

In the absence of fencing hardware, a manual fencing method can be used for testing or evaluation purposes.

Warning

Manual fencing should not be used in a production environment. Manual fencing depends on human intervention whenever a node needs recovery. Cluster operation is halted during the intervention.

The manual fencing agent, `fence_manual`, writes a message into the system log of the node on which the fencing agent is running. The message indicates the cluster node that requires fencing.

Upon seeing this message (by monitoring `/var/log/messages` or equivalent), an administrator must manually reset the node specified in the message. After the node is reset, the administrator must run the command `fence_ack_manual` to indicate to the system that the failed node has been reset. Recovery of the reset node will then proceed. Refer to Section 6.6 *Creating the `fence.ccs` File* and Section 6.7 *Creating the `nodes.ccs` File* for information on how to configure with this type of fencing.

10.2.7.1. Usage

```
fence_ack_manual -s IPAddress
```

IPAddress

The IP address of the node that was manually reset.

Chapter 11.

Using GNBD

GNBD (Global Network Block Device) provides block-level storage access over an Ethernet LAN. GNBD components run as a client in a GFS node and as a server in a GNBD server node. A GNBD server node exports block-level storage from its local storage (either directly attached storage or SAN storage) to a GFS node.

This chapter describes how to use GNBD with Red Hat GFS and consists of the following sections:

- Section 11.1 *GNBD Driver and Command Usage*
- Section 11.2 *Considerations for Using GNBD Multipath*
- Section 11.3 *Running GFS on a GNBD Server Node*

11.1. GNBD Driver and Command Usage

The Global Network Block Device (GNBD) driver allows a node to export its local storage as a GNBD over a network so that other nodes on the network can share the storage. Client nodes importing the GNBD use it like any other block device. Importing a GNBD on multiple clients forms a shared storage configuration through which GFS can be used.

The GNBD driver is implemented through the following client and server kernel modules.

- `gnbd.o` — Implements the GNBD device driver on GNBD clients (nodes using GNBD devices).
- `gnbd_serv.o` — Implements the GNBD server. It allows a node to export local storage over the network.

Two user commands are available to configure GNBD:

- `gnbd_export` (for servers) — User program for creating, exporting, and managing GNBDs on a GNBD server.
- `gnbd_import` (for clients) — User program for importing and managing GNBDs on a GNBD client.

11.1.1. Exporting a GNBD from a Server

The `gnbd_serv.o` kernel module must be loaded on a node before it can export storage as a GNBD. Once local storage has been identified to be exported, the `gnbd_export` command is used to export it.

Caution

When configured for GNBD multipath, a GNBD server (a server that is exporting a GNBD) ignores Linux page caching. Caching is ignored to ensure data integrity when using GNBD multipath. (By default, the `gnbd_export` command exports with caching turned off.)

Note

A server should not import the GNBDs to use them as a client would. If a server exports the devices uncached, they may also be used by `ccsd` and `gfs`.

11.1.1.1. Usage

`gnbd_export -d` *`pathname`* `-e` *`gnbdname`* `[-c]]`

`pathname`

Specifies a storage device to export.

`gnbdname`

Specifies an arbitrary name selected for the GNBD. It is used as the device name on GNBD clients. This name must be unique among all GNBDs exported in the network.

`-o`

Export the device as read-only.

Note

If a GNBD server that is exporting CCS files is also exporting GNBDs in multipath mode, it must export the CCS files as read-only. Under those circumstances, a GNBD client cannot use `ccs_tool` to update its copy of the CCS files. Instead, the CCS files must be updated on a node where the CCS files are stored locally or on FC-attached storage.

`-c`

Enable caching. Reads from the exported GNBD and takes advantage of the Linux page cache.

By default, the `gnbd_export` command does *not* enable caching.

Caution

For GNBD multipath, you must not specify the `-c` option. All GNBDs that are part of the pool *must run* with caching *disabled*. Pool, the GFS volume manager, does not check for caching being disabled; therefore, data corruption will occur if the GNBD devices are run with caching enabled.

Note

If you have been using GFS 5.2 or earlier and do *not* want to change your GNBD setup you *should* specify the `-c` option. Before GFS Release 5.2.1, Linux caching was enabled by default for `gnbd_export`. If the `-c` option is *not* specified, GNBD runs with a noticeable performance decrease. Also, if the `-c` option is *not* specified, the exported GNBD runs in timeout mode, using the default timeout value (the `-t` option). For more information about the `gnbd_export` command and its options, see the `gnbd_export` man page.

11.1.1.2. Examples

This example is for a GNBD server configured with GNBD multipath. It exports device /dev/sdc2 as GNBD gamma. Cache is disabled by default.

gnbd_export -d /dev/sdc2 -e gamma

This example is for a GNBD server *not* configured with GNBD multipath. It exports device /dev/sdb2 as GNBD delta with cache enabled.

gnbd_export -d /dev/sdb1 -e delta -c

11.1.2. Importing a GNBD on a Client

The gnbd.o kernel module must be loaded on a node before it can import GNBDs. When GNBDs are imported, device nodes are created for them in /dev/gnbd/ with the name assigned when they were exported.

11.1.2.1. Usage

gnbd_import -i *Server*

Server

Specifies a GNBD server by hostname or IP address from which to import GNBDs. All GNBDs exported from the server are imported on the client running this command.

11.1.2.2. Example

This example imports all GNBDs from the server named nodeA.

gnbd_import -i nodeA

11.2. Considerations for Using GNBD Multipath

GNBD multipath allows you to configure multiple GNBD server nodes (nodes that export GNBDs to GFS nodes) with redundant paths between the GNBD server nodes and storage devices. The GNBD server nodes, in turn, present multiple storage paths to GFS nodes via redundant GNBDs. With GNBD multipath, if a GNBD server node becomes unavailable, another GNBD server node can provide GFS nodes with access to storage devices.

If you are using GNBD multipath, you need to take the following into consideration:

- Linux page caching
- Lock server startup
- CCS file location
- Fencing GNBD server nodes

11.2.1. Linux Page Caching

For GNBD multipath, do *not* specify Linux page caching (the `-c` option of the `gnbd_export` command). All GNBDs that are part of the pool must run with caching *disabled*. Data corruption occurs if the GNBDs are run with caching enabled. Refer to Section 11.1.1 *Exporting a GNBD from a Server* for more information about using the `gnbd_export` command for GNBD multipath.

11.2.2. Lock Server Startup

Lock servers can reside on the following types of nodes: dedicated lock server nodes, GFS nodes, or GNBD server nodes. In any case, a lock server must be running before the GNBD servers can be started.

11.2.3. CCS File Location

In a GFS cluster configured for GNBD multipath, the location of CCS files for each node depends on how a node is deployed. If a node is deployed as a dedicated GFS node, its CCS files can reside on a GNBD, local storage, or FC-attached storage (if available). If a node is deployed as a dedicated GNBD server, its CCS files must reside on local storage or FC-attached storage. If a node is deployed as a dedicated lock server, its CCS files must reside on local storage or FC-attached storage. Because lock servers need to start before GNBD servers can start, a lock server *cannot* access CCS files through a GNBD. If a lock server is running on a GFS node, the CCS files for that node must be located on local storage or FC-attached storage.

If a GNBD server that is exporting CCS files is also exporting GNBDs in multipath mode, it must export the CCS files as read-only. (Refer to Section 11.1.1 *Exporting a GNBD from a Server* for more information about exporting a GNBD as read-only.) Under those circumstances, a GNBD client cannot use `ccs_tool` to update its copy of the CCS files. Instead, the CCS files must be updated on a node where the CCS files are stored locally or on FC-attached storage.

Note

If FC-attached storage can be shared among nodes, the CCS files can be stored on that shared storage.

Note

A node with CCS files stored on local storage or FC-attached storage can serve the CCS files to other nodes in a GFS cluster via `ccs_servd`. However, doing so would introduce a single point of failure. For information about `ccs_servd`, refer to Section 7.5.1 *CCA File and Server*.

Table 11-1 summarizes where CCS files can be located according to node deployment. For information about using CCS, refer to Chapter 7 *Using the Cluster Configuration System*.

Node Deployment	CCS File Location
GFS dedicated	GNBD, local, or FC-attached storage
GFS with lock server	Local or FC-attached storage only
GNBD server dedicated	Local or FC-attached storage only
GNBD server with lock server	Local or FC-attached storage only
Lock server dedicated	Local or FC-attached storage only

Table 11-1. CCS File Location for GNBD Multipath Cluster

Before a GNBD client node can activate (using the `pool_assemble` command) a GNBD-multipath pool, it must activate the GNBD-exported CCS pool and start `ccsd` and `lock_gulmd`. The following example shows activating an GNBD-exported CCS pool labeled as `CCS`:

```
# pool_assemble CCS
```

11.2.4. Fencing GNBD Server Nodes

GNBD server nodes must be fenced using a fencing method that physically removes the nodes from the network. To physically remove a GNBD server node, you can use any of the following fencing devices: APC MasterSwitch (`fence_apc` fence agent), WTI NPS (`fence_wti` fence agent), Brocade FC switch (`fence_brocade` fence agent), McData FC switch (`fence_mcdata` fence agent), Vixel FC switch (`fence_vixel` fence agent), HP RILOE (`fence_rib` fence agent), or xCAT (`fence_xcat` fence agent). You *cannot* use the GNBD fencing device (`fence_gnbd` fence agent) to fence a GNBD server node. For information about configuring fencing for GNBD server nodes, refer to Chapter 6 *Creating the Cluster Configuration System Files*.

11.3. Running GFS on a GNBD Server Node

You can run GFS on a GNBD server node, with some restrictions. In addition, running GFS on a GNBD server node reduces performance. The following restrictions apply when running GFS on a GNBD server node.

Important

When running GFS on a GNBD server node you *must* follow the restrictions listed; otherwise, the GNBD server node will fail.

Note

You may need to increase the timeout period on the exported GNBDs to accommodate reduced performance. The need to increase the timeout period depends on the quality of the hardware.

1. A GNBD server node must have local access to all storage devices needed to mount a GFS file system. The GNBD server node must not import (`gnbd_import` command) other GNBD devices to run the file system.
2. The GNBD server must export all the GNBDs in uncached mode, and it must export the raw devices, not pool devices.
3. GFS must be run on top of a pool device, not raw devices.

Chapter 12.

Using GFS `init.d` Scripts

This chapter describes GFS `init.d` scripts and consists of the following sections:

- Section 12.1 *GFS `init.d` Scripts Overview*
- Section 12.2 *GFS `init.d` Scripts Use*

12.1. GFS `init.d` Scripts Overview

The GFS `init.d` scripts start GFS services during node startup and stop GFS services during node shutdown. Also, the scripts provide functions for querying the status of GFS services (for example, if a service is running or stopped).

The GFS `init.d` scripts are stored in the directory `/etc/init.d` and accept one of the following parameters: `start`, `stop` or `status`. For example, to start the `gfs.o` module, call the `gfs init.d` script as follows:

```
# /etc/init.d/gfs start
```

As with other `init.d` scripts, wrappers are available for using the scripts. For example you can use `service` or `serviceconf`.

GFS provides the following init.d scripts that are installed automatically when GFS is installed:

- `pool`
- `ccsd`
- `lock_gulmd`
- `gfs`

The scripts automatically start and stop GFS modules during startup and shutdown of a node. When GFS is installed, the scripts are stored in the `/etc/init.d` directory. In addition, installation automatically names and places the scripts into directories `rc0.d` through `rc6.d` so that the GFS modules will be started and stopped in the correct order.

If you use the scripts manually to start and shut down GFS modules, you must run the scripts in a certain order. For startup, follow this sequence: `pool`, `ccsd`, `lock_gulmd`, and `gfs`. For shutdown, follow this sequence: `gfs`, `lock_gulmd`, `ccsd`, and `pool`.

The following example shows running the GFS `init.d` scripts to start up GFS:

```
# service pool start
# service ccsd start
# service lock_gulmd start
# service gfs start
```

The following example shows running the GFS `init.d` scripts to shut down GFS:

```
# service gfs stop
# service lock_gulmd stop
# service ccsd stop
# service pool stop
```

12.2. GFS init.d Scripts Use

The following example procedure demonstrates using the GFS `init.d` scripts to start GFS:

1. Install GFS on each node.
2. Load the `pool` module:

   Note

   If you need to specify a persistent major number, edit `/etc/modules.conf` before loading `pool.o`. Refer to Section 3.1.2 *Specifying a Persistent Major Number*

   `# `**`modprobe pool`**

   or

   `# `**`service pool start`**
3. Create pool labels.
4. Write the labels to disk.
5. Load the pools. You can use the `init.d` script to reload or rediscover pool labels as follows:
   `# `**`service pool restart`**

   You have the option of specifying in a configuration file `/etc/sysconfig/gfs` the pools that you want assembled. If no pools are specified, then the `pool` script scans all the devices and loads any pool that it finds.

   To specify the pools on which to operate, the variable *`POOLS`* must be set in `/etc/sysconfig/gfs`. You can define multiple pools by separating the pool names with a space, as shown in the following example:
   `POOLS="trin.cca trin1.gfs"`
6. Create the CCS archive.
7. Write the CCS archive to disk or to a file.
8. Modify `/etc/sysconfig/gfs` to specify the location of the CCS archive by defining the variable *`CCS_ARCHIVE`* in `/etc/sysconfig/gfs`. For example:
   `CCS_ARCHIVE="/dev/pool/trin.cca"`
9. Start `ccsd` as follows:
   `# `**`service ccsd start`**

   If *`CCS_ARCHIVE`* is not defined in `/etc/sysconfig/gfs`, `pool_tool` is used to scan for assembled pools that have CCS archives. If a single archive is found, then that device is automatically used.
10. Start `lock_gulmd` as follows:
    `# `**`service lock_gulmd start`**

    No additional configuration is required. `ccsd` needs to be running.
11. Create GFS file systems using the `gfs_mkfs` command.

12. Modify `/etc/fstab` to include GFS file systems. For example, here is part of an `/ect/fstab` file that includes the GFS file system `trin1.gfs`:

    ```
    /dev/pool/trin1.gfs /gfs gfs defaults 0 0
    ```

    If you do not want a GFS file system to automatically mount on startup, add `noauto` to the options in the `/ect/fstab` file as follows:

    ```
    /dev/pool/trin1.gfs /gfs gfs noauto,defaults 0 0
                                 \______________/
                                        |
                                  noauto option
    ```

13. Start `gfs` as follows:

    ```
    # service gfs start
    ```

Appendix A.
Using Red Hat GFS with Red Hat Cluster Suite

This appendix provides information about considerations to take when running Red Hat GFS 6.0 with Red Hat Cluster Suite and consists of the following sections:

- Section A.1 *Terminology*
- Section A.2 *Changes to Red Hat Cluster*
- Section A.3 *Installation Scenarios*

A.1. Terminology

You may have encountered new terms associated with Red Hat Cluster Suite. The following list provides a brief description of terms used with Red Hat GFS and Red Hat Cluster Suite:

GFS Setup Druid

This application is a Red Hat Cluster GUI for initial configuration of Red Hat GFS. The GUI is launched separately from the Red Hat Cluster GUI, the **Cluster Configuration Tool**. The **GFS Setup Druid** uses `/etc/cluster.xml` as input. If `/etc/cluster.xml` does not exist, the **GFS Setup Druid** displays a message and exits.

Note

You must run the **Cluster Configuration Tool** before running the **GFS Setup Druid**; the **Cluster Configuration Tool** creates `/etc/cluster.xml`.

To run the **GFS Setup Druid**, enter the following at the command line:

```
# redhat-config-gfscluster
```

gulm-bridge

This is a fence method available for Red Hat Cluster nodes, *if* and *only if* the Red Hat GFS RPM is installed on the node that the **Cluster Configuration Tool** runs on. The gulm-bridge fence method has been added to Red Hat Cluster Suite specifically for the Red Hat Enterprise Linux 4 Update 3 release. Using this fence method on a Red Hat Cluster Manager member prevents it from being fenced twice.

Red Hat Cluster

Red Hat Cluster Manager is part of the Red Hat Cluster Suite. It provides cluster administration functionality for Red Hat Enterprise Linux 4. Red Hat Cluster Manager contains two major components:

- **Red Hat Cluster Manager** — The underlying software (non-GUI) that performs Red Hat Cluster administrations services.
- **Cluster Configuration Tool** — This component is the graphical user interface (GUI) for Red Hat Cluster Manager. The GUI provides a configuration interface and a status monitor for members and services in a Red Hat Cluster Manager system. The **Cluster Configuration Tool** accepts configuration data from a user and writes it to the `/etc/cluster.xml` file. The **Red Hat Cluster Manager** reads the configuration data from the `/etc/cluster.xml` file.

Also, the **Cluster Configuration Tool** wraps several command line calls into the **Red Hat Cluster Manager**, such as starting and stopping services.

A.2. Changes to Red Hat Cluster

The following changes to Red Hat Cluster enable running it with Red Hat GFS in RHEL-U3:

- The **Cluster Configuration Tool** has been changed. After entering members in the configuration section of the application, if a member is highlighted, and you click **Add Child**, a dialog box is displayed, offering fence method options. You can select a fence method by clicking a radio button next to the fence method in the dialog box. Earlier Red Hat Cluster releases provided only two fence method options (under **Power Controller Type**): **Serial** and **Network**. For Red Hat Enterprise Linux 4 Update 3, if Red Hat GFS is installed on the node, then a third fence-method option, **GULM-STONITH** (the gulm-bridge fence method), is available.
- The **Red Hat Cluster Manager** now provides support for **GULM-STONITH**, the gulm-bridge fence method.
- A druid application, the **GFS Setup Druid**, provides for configuring an initial instance of Red Hat GFS by writing the three Red Hat GFS configuration files: `cluster.ccs`, `nodes.ccs`, and `fence.ccs`. The **GFS Setup Druid** requires an `/etc/cluster.xml` file when started.

A.3. Installation Scenarios

When running Red Hat GFS with Red Hat Cluster Manager, you must take into account certain considerations, according to the following circumstances:

- New installations of Red Hat GFS and Red Hat Cluster Manager
- Adding Red Hat GFS to an existing Red Hat Cluster Manager deployment
- Upgrading Red Hat GFS 5.2.1 to Red Hat GFS 6.0

A.3.1. New Installations of Red Hat GFS and Red Hat Cluster Manager

When installing Red Hat GFS and Red Hat Cluster Manager for the first time into a cluster, install and configure Red Hat Cluster Suite before installing and configuring Red Hat GFS. With the **Cluster Configuration Tool**, you can configure up to 16 nodes — the maximum number of nodes allowed in Red Hat Cluster Manager system.

You can add services and failover domains (and other functions) after initially configuring Red Hat GFS with the **GFS Setup Druid**.

Note

The only configuration items in Red Hat Cluster that Red Hat GFS or the **GFS Setup Druid** depend on are setting up Red Hat Cluster Manager members and specifying fence devices.

A.3.2. Adding Red Hat GFS to an Existing Red Hat Cluster Manager Deployment

Adding Red Hat GFS to an existing Red Hat Cluster Manager deployment requires running the Red Hat GFS druid application, **GFS Setup Druid** (also known as **redhat-config-gfscluster**). As with the scenario in Section A.3.1 *New Installations of Red Hat GFS and Red Hat Cluster Manager*, while Red Hat GFS is scalable up to 300 nodes, a Red Hat Cluster Manager limits the total number of nodes in a cluster to 16. Therefore, in this scenario, Red Hat GFS scalability is limited. If the 16-node limit is too small for your deployment, you may want to consider using multiple Red Hat Cluster Manager clusters.

A.3.3. Upgrading Red Hat GFS 5.2.1 to Red Hat GFS 6.0

To upgrade Red Hat GFS 5.2.1 to Red Hat GFS 6.0, follow the procedures in Appendix B *Upgrading GFS*. Running an upgraded version of Red Hat GFS (Red Hat GFS 6.0) with Red Hat Cluster Manager, requires the following actions:

1. Install and configure Red Hat Cluster Manager.
2. Install and configure Red Hat GFS, configuring the Red Hat GFS CCS files according to the procedures in Chapter 6 *Creating the Cluster Configuration System Files*. It is recommended that you edit the CCS files manually rather than by using the Red Hat Cluster Manager **GFS Setup Druid**.

Note

For assistance with installing Red Hat Cluster Manager and performing the **Red Hat GFS** upgrade in this scenario, consult with Red Hat Support.

Appendix B.

Upgrading GFS

This appendix contains instructions for upgrading GFS 5.2.1 to GFS 6.0 software.

Note

> If you are using GFS with Red Hat Cluster, the order in which you upgrade GFS compared to other Red Hat Cluster installation and configuration tasks may vary. For information about installing and using GFS with Red Hat Cluster Suite, refer to Appendix A *Using Red Hat GFS with Red Hat Cluster Suite*.

To upgrade the software follow these steps:

1. Halt the cluster nodes and the lock servers. The remaining steps require that the GFS cluster be stopped (all GFS nodes shut down). Stopping the GFS cluster consists of the following actions:
   a. Unmount GFS file systems from all nodes.
   b. Stop lock servers.
   c. Stop `ccsd` on all nodes.
   d. Deactivate pools.
   e. Unload kernel modules.

2. Install new software. This step consists of the following actions:

   Reference: Chapter 3 *Installing GFS*

   a. Install (or verify that) the Red Hat Enterprise Linux 3 Update 2 kernel (the stock "2.4.21-15.EL" kernel) is installed.
   b. Install `perl-Net-Telnet` RPM.
   c. Install GFS 6.0 RPMs.

3. Load new kernel modules on GFS nodes.

   Reference: Chapter 3 *Installing GFS*

   Example:
   ```
   insmod pool
   insmod lock_harness
   insmod lock_gulm
   insmod gfs
   ```

4. (Optional) Modify CCS files.

   With the cluster being shut down, if you need to make changes to the Cluster Configuration System (CCS) files, you have the option of doing that now. In addition, you can remove the `licens.ccs` file from the CCA — GFS 6.0 requires no `license.ccs` file.

Note

Although GFS 6.0 requires no `license.ccs` file, you can safely leave the license file in the CCA.

Tip

You can use the `ccs_tool extract` command to extract the Cluster Configuration System (CCS) files for modification.

5. (Optional) Activate pools on all nodes.

   Command usage: `pool_assemble -a`

   Reference: Section 5.6 *Activating/Deactivating a Pool Volume*

   Example:
   `pool_assemble -a`

6. (Optional) Create CCS archive on CCA device. The CCS archive is created from the directory of new CCS files as described in Step 5.

   Command usage: `ccs_tool create` *`Directory Device`*

   Reference: Section 7.1 *Creating a CCS Archive*

   Example:
   `ccs_tool create /root/alpha/ /dev/pool/alpha_cca`

7. Start `ccsd` on all nodes.

   This includes all GFS nodes and all nodes that will run the LOCK_GULM server.

   Command usage: `ccsd -d` *`Device`*

   Reference: Section 7.2 *Starting CCS in the Cluster*

   Example:
   `ccsd -d /dev/pool/alpha_cca`

8. Start LOCK_GULM server.

   Start `lock_gulmd` on all nodes.

   Command usage: `lock_gulmd`

   Reference: Section 8.2.3 *Starting LOCK_GULM Servers*

   Example:
   `lock_gulmd`

9. Mount GFS file systems on all GFS nodes.

   Command usage: `mount -t gfs` *`BlockDevice MountPoint`*

   Reference: Section 9.2 *Mounting a File System*

   Example:
   `mount -t gfs /dev/pool/pool0 /gfs`

Appendix C.

Basic GFS Examples

This appendix contains examples of setting up and using GFS in the following basic scenarios:

- Section C.1 *LOCK_GULM, RLM Embedded*
- Section C.2 *LOCK_GULM, RLM External*
- Section C.3 *LOCK_GULM, SLM Embedded*
- Section C.4 *LOCK_GULM, SLM External*
- Section C.5 *LOCK_GULM, SLM External, and GNBD*
- Section C.6 *LOCK_NOLOCK*

The examples follow the process structure for procedures and associated tasks defined in Chapter 4 *Initial Configuration*.

C.1. LOCK_GULM, RLM Embedded

This example sets up a cluster with three nodes and two GFS file systems. It requires three nodes for the GFS cluster. All nodes in the cluster mount the GFS file system and run the LOCK_GULM servers.

This section provides the following information about the example:

- Section C.1.1 *Key Characteristics*
- Section C.1.2 *Kernel Modules Loaded*
- Section C.1.3 *Setup Process*

C.1.1. Key Characteristics

This example configuration has the following key characteristics:

- Fencing device — An APC MasterSwitch (single-switch configuration). Refer to Table C-1 for switch information.
- Number of GFS nodes — 3. Refer to Table C-2 for node information.
- Number of lock server nodes — 3. The lock servers are run on the GFS nodes (embedded). Refer to Table C-2 for node information.
- Locking protocol — LOCK_GULM. The LOCK_GULM server is run on every node that mounts GFS.
- Number of shared storage devices — 2. Refer to Table C-3 for storage device information.
- Number of file systems — 2.
- File system names — `gfs01` and `gfs02`.
- File system mounting — Each GFS node mounts the two file systems.
- Cluster name — `alpha`.

Host Name	IP Address	Login Name	Password
apc	10.0.1.10	apc	apc

Table C-1. APC MasterSwitch Information

Host Name	IP Address	APC Port Number
n01	10.0.1.1	1
n02	10.0.1.2	2
n03	10.0.1.3	3

Table C-2. GFS and Lock Server Node Information

Major	Minor	#Blocks	Name
8	16	8388608	sda
8	17	8001	sda1
8	18	8377897	sda2
8	32	8388608	sdb
8	33	8388608	sdb1

Table C-3. Storage Device Information

Notes

For shared storage devices to be visible to the nodes, it may be necessary to load an appropriate device driver. If the shared storage devices are not visible on each node, confirm that the device driver is loaded and that it loaded without errors.

The small partition (`/dev/sda1`) is used to store the cluster configuration information. The two remaining partitions (/`dev/sda2`, `sdb1`) are used for the GFS file systems.

You can display the storage device information at each node in your GFS cluster by running the following command: `cat /proc/partitions`. Depending on the hardware configuration of the GFS nodes, the names of the devices may be different on each node. If the output of the `cat /proc/partitions` command shows only entire disk devices (for example, `/dev/sda` instead of `/dev/sda1`), then the storage devices have not been partitioned. To partition a device, use the `fdisk` command.

C.1.2. Kernel Modules Loaded

Each node must have the following kernel modules loaded:

- `gfs.o`
- `lock_harness.o`
- `lock_gulm.o`
- `pool.o`

C.1.3. Setup Process

The setup process for this example consists of the following steps:

1. Create pool configurations for the two file systems.

   Create pool configuration files for each file system's pool: `pool_gfs01` for the first file system, and `pool_gfs02` for the second file system. The two files should look like the following:

   ```
   poolname pool_gfs01
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sda2
   ```

   ```
   poolname pool_gfs02
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sdb1
   ```

2. Create a pool configuration for the CCS data.

   Create a pool configuration file for the pool that will be used for CCS data. The pool does not need to be very large. The name of the pool will be `alpha_cca`. (The name of the cluster, `alpha`, followed by `_cca`). The file should look like the following:

   ```
   poolname alpha_cca
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sda1
   ```

3. Use the `pool_tool` command to create all the pools as follows:

   ```
   n01# pool_tool -c pool_gfs01.cf pool_gfs02.cf alpha_cca.cf
   Pool label written successfully from pool_gfs01.cf
   Pool label written successfully from pool_gfs02.cf
   Pool label written successfully from alpha_cca.cf
   ```

4. Activate the pools on all nodes.

   Note

   This step must be performed every time a node is rebooted. If it is not, the pool devices will not be accessible.

   Activate the pools using the `pool_assemble -a` command for each node as follows:

   ```
   n01# pool_assemble -a  <-- Activate pools
   alpha_cca assembled
   pool_gfs01 assembled
   pool_gfs02 assembled

   n02# pool_assemble -a  <-- Activate pools
   alpha_cca assembled
   pool_gfs01 assembled
   pool_gfs02 assembled

   n03# pool_assemble -a  <-- Activate pools
   alpha_cca assembled
   pool_gfs01 assembled
   pool_gfs02 assembled
   ```

5. Create CCS files.

   a. Create a directory called `/root/alpha` on node `n01` as follows:

      ```
      n01# mkdir /root/alpha
      n01# cd /root/alpha
      ```

b. Create the `cluster.ccs` file. This file contains the name of the cluster and the name of the nodes where the LOCK_GULM server is run. The file should look like the following:

```
cluster {
   name = "alpha"
   lock_gulm {
      servers = ["n01", "n02", "n03"]
   }
}
```

c. Create the `nodes.ccs` file. This file contains the name of each node, its IP address, and node-specific I/O fencing parameters. The file should look like the following:

```
nodes {
   n01 {
      ip_interfaces {
         eth0 = "10.0.1.1"
      }
      fence {
         power {
            apc {
            port = 1
            }
         }
      }
   }
   n02 {
      ip_interfaces {
         eth0 = "10.0.1.2"
      }
      fence {
         power {
            apc {
            port = 2
            }
         }
      }
   }
   n03 {
      ip_interfaces {
         eth0 = "10.0.1.3"
      }
      fence {
         power {
            apc {
            port = 3
            }
         }
      }
   }
}
```

Note

If your cluster is running Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5 and later, you can use the optional `usedev` parameter to explicitly specify an IP address rather than relying on an IP address from `libresolv`. For more information about the optional `usedev` parameter, refer to the file format in Figure 6-23 and the example in Example 6-26. Refer to Table 6-3 for syntax description of the `usedev` parameter.

d. Create the `fence.ccs` file. This file contains information required for the fencing method(s) used by the GFS cluster. The file should look like the following:

```
fence_devices {
   apc {
      agent = "fence_apc"
      ipaddr = "10.0.1.10"
      login = "apc"
      passwd = "apc"
   }
}
```

6. Create the CCS Archive on the CCA Device.

   > **Note**
   >
   > This step only needs to be done once and from a single node. It should *not* be performed every time the cluster is restarted.

   Use the `ccs_tool` command to create the archive from the CCS configuration files:

   ```
   n01# ccs_tool create /root/alpha /dev/pool/alpha_cca
   Initializing device for first time use... done.
   ```

7. Start the CCS daemon (`ccsd`) on all the nodes.

   > **Note**
   >
   > This step must be performed each time the cluster is rebooted.

   The CCA device must be specified when starting `ccsd`.

   ```
   n01# ccsd -d /dev/pool/alpha_cca

   n02# ccsd -d /dev/pool/alpha_cca

   n03# ccsd -d /dev/pool/alpha_cca
   ```

8. At each node, start the LOCK_GULM server:

   ```
   n01# lock_gulmd

   n02# lock_gulmd

   n03# lock_gulmd
   ```

9. Create the GFS file systems.

   Create the first file system on `pool_gfs01` and the second on `pool_gfs02`. The names of the two file systems are `gfs01` and `gfs02`, respectively, as shown in the example:

   ```
   n01# gfs_mkfs -p lock_gulm -t alpha:gfs01 -j 3 /dev/pool/pool_gfs01
   Device: /dev/pool/pool_gfs01
   Blocksize: 4096
   Filesystem Size:1963216
   Journals: 3
   Resource Groups:30
   Locking Protocol:lock_gulm
   Lock Table: alpha:gfs01

   Syncing...
   All Done

   n01# gfs_mkfs -p lock_gulm -t alpha:gfs02 -j 3 /dev/pool/pool_gfs02
   Device: /dev/pool/pool_gfs02
   ```

```
Blocksize: 4096
Filesystem Size:1963416
Journals: 3
Resource Groups:30
Locking Protocol:lock_gulm
Lock Table: alpha:gfs02

Syncing...
All Done
```

10. Mount the GFS file systems on all the nodes.

    Mount points `/gfs01` and `/gfs02` are used on each node:

```
n01# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n01# mount -t gfs /dev/pool/pool_gfs02 /gfs02

n02# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n02# mount -t gfs /dev/pool/pool_gfs02 /gfs02

n03# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n03# mount -t gfs /dev/pool/pool_gfs02 /gfs02
```

C.2. LOCK_GULM, RLM External

This example sets up a GFS cluster with three nodes, two GFS file systems, and three lock server nodes that are external to the GFS nodes. The lock server nodes are dedicated to running LOCK_GULM only.

This section provides the following information about the example:

- Section C.2.1 *Key Characteristics*
- Section C.2.2 *Kernel Modules Loaded*
- Section C.2.3 *Setup Process*

C.2.1. Key Characteristics

This example configuration has the following key characteristics:

- Fencing device — An APC MasterSwitch (single-switch configuration). Refer to Table C-4 for switch information.
- Number of GFS nodes — 3. Refer to Table C-5 for node information.
- Number of lock server nodes — 3. The lock server nodes are dedicated to running as lock server nodes only, and are external to the GFS nodes. Refer to Table C-6 for node information.
- Locking protocol — LOCK_GULM. The LOCK_GULM server is run on each lock server node.
- Number of shared storage devices — 2. Refer to Table C-7 for storage device information.
- Number of file systems — 2.
- File system names — `gfs01` and `gfs02`.
- File system mounting — Each GFS node mounts the two file systems.
- Cluster name — `alpha`.

Host Name	IP Address	Login Name	Password
apc	10.0.1.10	apc	apc

Table C-4. APC MasterSwitch Information

Host Name	IP Address	APC Port Number
n01	10.0.1.1	1
n02	10.0.1.2	2
n03	10.0.1.3	3

Table C-5. GFS Node Information

Host Name	IP Address	APC Port Number
lck01	10.0.1.4	4
lck02	10.0.1.5	5
lck03	10.0.1.6	6

Table C-6. Lock Server Node Information

Major	Minor	#Blocks	Name
8	16	8388608	sda
8	17	8001	sda1
8	18	8377897	sda2
8	32	8388608	sdb
8	33	8388608	sdb1

Table C-7. Storage Device Information

Notes

For shared storage devices to be visible to the nodes, it may be necessary to load an appropriate device driver. If the shared storage devices are not visible on each node, confirm that the device driver is loaded and that it loaded without errors.

The small partition (`/dev/sda1`) is used to store the cluster configuration information. The two remaining partitions (`/dev/sda2`, `sdb1`) are used for the GFS file systems.

You can display the storage device information at each node in your GFS cluster by running the following command: `cat /proc/partitions`. Depending on the hardware configuration of the GFS nodes, the names of the devices may be different on each node. If the output of the `cat /proc/partitions` command shows only entire disk devices (for example, `/dev/sda` instead of `/dev/sda1`), then the storage devices have not been partitioned. To partition a device, use the `fdisk` command.

C.2.2. Kernel Modules Loaded

Each node must have the following kernel modules loaded:

- `gfs.o`
- `lock_harness.o`
- `lock_gulm.o`
- `pool.o`

C.2.3. Setup Process

The setup process for this example consists of the following steps:

1. Create pool configurations for the two file systems.

   Create pool configuration files for each file system's pool: `pool_gfs01` for the first file system, and `pool_gfs02` for the second file system. The two files should look like the following:
   ```
   poolname pool_gfs01
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sda2
   ```
   ```
   poolname pool_gfs02
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sdb1
   ```
2. Create a pool configuration for the CCS data.

   Create a pool configuration file for the pool that will be used for CCS data. The pool does not need to be very large. The name of the pool will be `alpha_cca`. (The name of the cluster, `alpha`, followed by `_cca`). The file should look like the following:
   ```
   poolname alpha_cca
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sda1
   ```
3. Use the `pool_tool` command to create all the pools as follows:
   ```
   n01# pool_tool -c pool_gfs01.cf pool_gfs02.cf alpha_cca.cf
   Pool label written successfully from pool_gfs01.cf
   Pool label written successfully from pool_gfs02.cf
   Pool label written successfully from alpha_cca.cf
   ```
4. Activate the pools on all nodes.

   > **Note**
   >
   > This step must be performed every time a node is rebooted. If it is not, the pool devices will not be accessible.

   Activate the pools using the `pool_assemble -a` command for each node as follows:
   ```
   n01# pool_assemble -a  <-- Activate pools
   alpha_cca assembled
   pool_gfs01 assembled
   pool_gfs02 assembled

   n02# pool_assemble -a  <-- Activate pools
   ```

```
alpha_cca assembled
pool_gfs01 assembled
pool_gfs02 assembled

n03# pool_assemble -a  <-- Activate pools
alpha_cca assembled
pool_gfs01 assembled
pool_gfs02 assembled

lck01# pool_assemble -a  <-- Activate pools
alpha_cca assembled
pool_gfs01 assembled
pool_gfs02 assembled

lck02# pool_assemble -a  <-- Activate pools
alpha_cca assembled
pool_gfs01 assembled
pool_gfs02 assembled

lck03# pool_assemble -a  <-- Activate pools
alpha_cca assembled
pool_gfs01 assembled
pool_gfs02 assembled
```

5. Create CCS files.

   a. Create a directory called `/root/alpha` on node `n01` as follows:

   ```
   n01# mkdir /root/alpha
   n01# cd /root/alpha
   ```

   b. Create the `cluster.ccs` file. This file contains the name of the cluster and the name of the nodes where the LOCK_GULM server is run. The file should look like the following:

   ```
   cluster {
      name = "alpha"
      lock_gulm {
         servers = ["lck01", "lck02", "lck03"]
      }
   }
   ```

   c. Create the `nodes.ccs` file. This file contains the name of each node, its IP address, and node-specific I/O fencing parameters. The file should look like the following:

   ```
   nodes {
      n01 {
         ip_interfaces {
            eth0 = "10.0.1.1"
         }
         fence {
            power {
               apc {
               port = 1
               }
            }
         }
      }
      n02 {
         ip_interfaces {
            eth0 = "10.0.1.2"
         }
         fence {
            power {
               apc {
               port = 2
               }
   ```

```
         }
      }
   }
   n03 {
      ip_interfaces {
         eth0 = "10.0.1.3"
      }
      fence {
         power {
            apc {
            port = 3
            }
         }
      }
   }

   lck01 {
      ip_interfaces {
         eth0 = "10.0.1.4"
      }
      fence {
         power {
            apc {
            port = 4
            }
         }
      }
   }
   lck02 {
      ip_interfaces {
         eth0 = "10.0.1.5"
      }
      fence {
         power {
            apc {
            port = 5
            }
         }
      }
   }
   lck03 {
      ip_interfaces {
         eth0 = "10.0.1.6"
      }
      fence {
         power {
            apc {
            port = 6
            }
         }
      }
   }
}
```

Note

If your cluster is running Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5 and later, you can use the optional `usedev` parameter to explicitly specify an IP address rather than relying on an IP address from `libresolv`. For more information about the optional `usedev` parameter, refer to the file format in Figure 6-23 and the example in Example 6-26. Refer to Table 6-3 for syntax description of the `usedev` parameter.

d. Create the `fence.ccs` file. This file contains information required for the fencing method(s) used by the GFS cluster. The file should look like the following:

```
fence_devices {
   apc {
      agent = "fence_apc"
      ipaddr = "10.0.1.10"
      login = "apc"
      passwd = "apc"
   }
}
```

6. Create the CCS Archive on the CCA Device.

Note

This step only needs to be done once and from a single node. It should *not* be performed every time the cluster is restarted.

Use the `ccs_tool` command to create the archive from the CCS configuration files:

```
n01# ccs_tool create /root/alpha /dev/pool/alpha_cca
Initializing device for first time use... done.
```

7. Start the CCS daemon (ccsd) on all the nodes.

Note

This step must be performed each time the cluster is rebooted.

The CCA device must be specified when starting ccsd.

```
n01# ccsd -d /dev/pool/alpha_cca

n02# ccsd -d /dev/pool/alpha_cca

n03# ccsd -d /dev/pool/alpha_cca

lck01# ccsd -d /dev/pool/alpha_cca

lck02# ccsd -d /dev/pool/alpha_cca

lck03# ccsd -d /dev/pool/alpha_cca
```

8. At each node, start the LOCK_GULM server. For example:

```
n01# lock_gulmd

lck01# lock_gulmd
```

9. Create the GFS file systems.

   Create the first file system on `pool_gfs01` and the second on `pool_gfs02`. The names of the two file systems are `gfs01` and `gfs02` , respectively, as shown in the example:

```
n01# gfs_mkfs -p lock_gulm -t alpha:gfs01 -j 3 /dev/pool/pool_gfs01
Device: /dev/pool/pool_gfs01
Blocksize: 4096
Filesystem Size:1963216
Journals: 3
Resource Groups:30
Locking Protocol:lock_gulm
Lock Table: alpha:gfs01

Syncing...
All Done

n01# gfs_mkfs -p lock_gulm -t alpha:gfs02 -j 3 /dev/pool/pool_gfs02
Device: /dev/pool/pool_gfs02
Blocksize: 4096
Filesystem Size:1963416
Journals: 3
Resource Groups:30
Locking Protocol:lock_gulm
Lock Table: alpha:gfs02

Syncing...
All Done
```

10. Mount the GFS file systems on all the nodes.

    Mount points `/gfs01` and `/gfs02` are used on each node:

```
n01# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n01# mount -t gfs /dev/pool/pool_gfs02 /gfs02

n02# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n02# mount -t gfs /dev/pool/pool_gfs02 /gfs02

n03# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n03# mount -t gfs /dev/pool/pool_gfs02 /gfs02
```

C.3. LOCK_GULM, SLM Embedded

This example sets up a cluster with three nodes and two GFS file systems. It requires three nodes for the GFS cluster. One of the nodes in the cluster runs the LOCK_GULM server in addition to mounting the GFS file system.

This section provides the following information about the example:

- Section C.3.1 *Key Characteristics*
- Section C.3.2 *Kernel Modules Loaded*
- Section C.3.3 *Setup Process*

C.3.1. Key Characteristics

This example configuration has the following key characteristics:

- Fencing device — An APC MasterSwitch (single-switch configuration). Refer to Table C-8 for switch information.
- Number of GFS nodes — 3. Refer to Table C-9 for node information.
- Number of lock server nodes — 1. The lock server is run on one of the GFS nodes (embedded). Refer to Table C-9 for node information.
- Locking protocol — LOCK_GULM. The LOCK_GULM server is run on the node that is designated as a lock server node.
- Number of shared storage devices — 2. Refer to Table C-10 for storage device information.
- Number of file systems — 2.
- File system names — `gfs01` and `gfs02`.
- File system mounting — Each GFS node mounts the two file systems.
- Cluster name — `alpha`.

Host Name	IP Address	Login Name	Password
apc	10.0.1.10	apc	apc

Table C-8. APC MasterSwitch Information

Host Name	IP Address	APC Port Number
n01	10.0.1.1	1
n02	10.0.1.2	2
n03	10.0.1.3	3

Table C-9. GFS and Lock Server Node Information

Major	Minor	#Blocks	Name
8	16	8388608	sda
8	17	8001	sda1
8	18	8377897	sda2
8	32	8388608	sdb
8	33	8388608	sdb1

Table C-10. Storage Device Information

Notes

For shared storage devices to be visible to the nodes, it may be necessary to load an appropriate device driver. If the shared storage devices are not visible on each node, confirm that the device driver is loaded and that it loaded without errors.

The small partition (/dev/sda1) is used to store the cluster configuration information. The two remaining partitions (/dev/sda2, sdb1) are used for the GFS file systems.

You can display the storage device information at each node in your GFS cluster by running the following command: `cat /proc/partitions`. Depending on the hardware configuration of the GFS nodes, the names of the devices may be different on each node. If the output of the `cat /proc/partitions` command shows only entire disk devices (for example, /dev/sda instead of /dev/sda1), then the storage devices have not been partitioned. To partition a device, use the `fdisk` command.

C.3.2. Kernel Modules Loaded

Each node must have the following kernel modules loaded:

- gfs.o
- lock_harness.o
- lock_gulm.o
- pool.o

C.3.3. Setup Process

The setup process for this example consists of the following steps:

1. Create pool configurations for the two file systems.

   Create pool configuration files for each file system's pool: pool_gfs01 for the first file system, and pool_gfs02 for the second file system. The two files should look like the following:
   ```
   poolname pool_gfs01
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sda2
   ```
   ```
   poolname pool_gfs02
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sdb1
   ```
2. Create a pool configuration for the CCS data.

   Create a pool configuration file for the pool that will be used for CCS data. The pool does not need to be very large. The name of the pool will be alpha_cca. (The name of the cluster, alpha, followed by _cca). The file should look like the following:
   ```
   poolname alpha_cca
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sda1
   ```
3. Use the pool_tool command to create all the pools as follows:
   ```
   n01# pool_tool -c pool_gfs01.cf pool_gfs02.cf alpha_cca.cf
   Pool label written successfully from pool_gfs01.cf
   Pool label written successfully from pool_gfs02.cf
   Pool label written successfully from alpha_cca.cf
   ```
4. Activate the pools on all nodes.

Note

This step must be performed every time a node is rebooted. If it is not, the pool devices will not be accessible.

Activate the pools using the `pool_assemble -a` command for each node as follows:

```
n01# pool_assemble -a  <-- Activate pools
alpha_cca assembled
pool_gfs01 assembled
pool_gfs02 assembled

n02# pool_assemble -a  <-- Activate pools
alpha_cca assembled
pool_gfs01 assembled
pool_gfs02 assembled

n03# pool_assemble -a  <-- Activate pools
alpha_cca assembled
pool_gfs01 assembled
pool_gfs02 assembled
```

5. Create CCS files.

   a. Create a directory called `/root/alpha` on node `n01` as follows:

   ```
   n01# mkdir /root/alpha
   n01# cd /root/alpha
   ```

   b. Create the `cluster.ccs` file. This file contains the name of the cluster and the name of the nodes where the LOCK_GULM server is run. The file should look like the following:

   ```
   cluster {
      name = "alpha"
      lock_gulm {
         servers = ["n01"]
      }
   }
   ```

   c. Create the `nodes.ccs` file. This file contains the name of each node, its IP address, and node-specific I/O fencing parameters. The file should look like the following:

   ```
   nodes {
      n01 {
         ip_interfaces {
            eth0 = "10.0.1.1"
         }
         fence {
            power {
               apc {
               port = 1
               }
            }
         }
      }
      n02 {
         ip_interfaces {
            eth0 = "10.0.1.2"
         }
         fence {
            power {
               apc {
               port = 2
               }
            }
   ```

```
            }
        }
        n03 {
            ip_interfaces {
                eth0 = "10.0.1.3"
            }
            fence {
                power {
                    apc {
                    port = 3
                    }
                }
            }
        }
    }
```

Note

If your cluster is running Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5 and later, you can use the optional `usedev` parameter to explicitly specify an IP address rather than relying on an IP address from `libresolv`. For more information about the optional `usedev` parameter, refer to the file format in Figure 6-23 and the example in Example 6-26. Refer to Table 6-3 for syntax description of the `usedev` parameter.

d. Create the `fence.ccs` file. This file contains information required for the fencing method(s) used by the GFS cluster. The file should look like the following:

```
fence_devices {
    apc {
        agent = "fence_apc"
        ipaddr = "10.0.1.10"
        login = "apc"
        passwd = "apc"
    }
}
```

6. Create the CCS Archive on the CCA Device.

Note

This step only needs to be done once and from a single node. It should *not* be performed every time the cluster is restarted.

Use the `ccs_tool` command to create the archive from the CCS configuration files:

```
n01# ccs_tool create /root/alpha /dev/pool/alpha_cca
Initializing device for first time use... done.
```

7. Start the CCS daemon (`ccsd`) on all the nodes.

note

This step must be performed each time the cluster is rebooted.

The CCA device must be specified when starting `ccsd`.

```
n01# ccsd -d /dev/pool/alpha_cca

n02# ccsd -d /dev/pool/alpha_cca

n03# ccsd -d /dev/pool/alpha_cca
```

8. Start the LOCK_GULM server on each node. For example:

```
n01# lock_gulmd
```

9. Create the GFS file systems.

   Create the first file system on `pool_gfs01` and the second on `pool_gfs02`. The names of the two file systems are `gfs01` and `gfs02`, respectively, as shown in the example:

```
n01# gfs_mkfs -p lock_gulm -t alpha:gfs01 -j 3 /dev/pool/pool_gfs01
Device: /dev/pool/pool_gfs01
Blocksize: 4096
Filesystem Size:1963216
Journals: 3
Resource Groups:30
Locking Protocol:lock_gulm
Lock Table: alpha:gfs01

Syncing...
All Done

n01# gfs_mkfs -p lock_gulm -t alpha:gfs02 -j 3 /dev/pool/pool_gfs02
Device: /dev/pool/pool_gfs02
Blocksize: 4096
Filesystem Size:1963416
Journals: 3
Resource Groups:30
Locking Protocol:lock_gulm
Lock Table: alpha:gfs02

Syncing...
All Done
```

10. Mount the GFS file systems on all the nodes.

   Mount points `/gfs01` and `/gfs02` are used on each node:

```
n01# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n01# mount -t gfs /dev/pool/pool_gfs02 /gfs02

n02# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n02# mount -t gfs /dev/pool/pool_gfs02 /gfs02

n03# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n03# mount -t gfs /dev/pool/pool_gfs02 /gfs02
```

C.4. LOCK_GULM, SLM External

This example sets up a cluster with three nodes and two GFS file systems. It requires three nodes for the GFS cluster and an additional (external) node to run the LOCK_GULM server.

This section provides the following information about the example:

- Section C.4.1 *Key Characteristics*
- Section C.4.2 *Kernel Modules Loaded*
- Section C.4.3 *Setup Process*

C.4.1. Key Characteristics

This example configuration has the following key characteristics:

- Fencing device — An APC MasterSwitch (single-switch configuration). Refer to Table C-11 for switch information.
- Number of GFS nodes — 3. Refer to Table C-12 for node information.
- Number of lock server nodes — 1. The lock server is run on one of the GFS nodes (embedded). Refer to Table C-13 for node information.
- Locking protocol — LOCK_GULM. The LOCK_GULM server is run on the node that is designated as a lock server node.
- Number of shared storage devices — 2. Refer to Table C-14 for storage device information.
- Number of file systems — 2.
- File system names — `gfs01` and `gfs02`.
- File system mounting — Each GFS node mounts the two file systems.
- Cluster name — `alpha`.

Host Name	IP Address	Login Name	Password
apc	10.0.1.10	apc	apc

Table C-11. APC MasterSwitch Information

Host Name	IP Address	APC Port Number
n01	10.0.1.1	1
n02	10.0.1.2	2
n03	10.0.1.3	3

Table C-12. GFS Node Information

Host Name	IP Address	APC Port Number
lcksrv	10.0.1.4	4

Table C-13. Lock Server Node Information

Major	Minor	#Blocks	Name
8	16	8388608	sda
8	17	8001	sda1
8	18	8377897	sda2
8	32	8388608	sdb
8	33	8388608	sdb1

Table C-14. Storage Device Information

Notes

For shared storage devices to be visible to the nodes, it may be necessary to load an appropriate device driver. If the shared storage devices are not visible on each node, confirm that the device driver is loaded and that it loaded without errors.

The small partition (`/dev/sda1`) is used to store the cluster configuration information. The two remaining partitions (/`dev/sda2`, `sdb1`) are used for the GFS file systems.

You can display the storage device information at each node in your GFS cluster by running the following command: `cat /proc/partitions`. Depending on the hardware configuration of the GFS nodes, the names of the devices may be different on each node. If the output of the `cat /proc/partitions` command shows only entire disk devices (for example, `/dev/sda` instead of `/dev/sda1`), then the storage devices have not been partitioned. To partition a device, use the `fdisk` command.

C.4.2. Kernel Modules Loaded

Each node must have the following kernel modules loaded:

- `gfs.o`
- `lock_harness.o`
- `lock_gulm.o`
- `pool.o`

C.4.3. Setup Process

The setup process for this example consists of the following steps:

1. Create pool configurations for the two file systems.

   Create pool configuration files for each file system's pool: `pool_gfs01` for the first file system, and `pool_gfs02` for the second file system. The two files should look like the following:

   ```
   poolname pool_gfs01
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sda2

   poolname pool_gfs02
   subpools 1
   ```

```
subpool 0 0 1
pooldevice 0 0 /dev/sdb1
```

2. Create a pool configuration for the CCS data.

   Create a pool configuration file for the pool that will be used for CCS data. The pool does not need to be very large. The name of the pool will be `alpha_cca`. (The name of the cluster, `alpha`, followed by `_cca`). The file should look like the following:

   ```
   poolname alpha_cca
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sda1
   ```

3. Use the `pool_tool` command to create all the pools as follows:

   ```
   n01# pool_tool -c pool_gfs01.cf pool_gfs02.cf alpha_cca.cf
   Pool label written successfully from pool_gfs01.cf
   Pool label written successfully from pool_gfs02.cf
   Pool label written successfully from alpha_cca.cf
   ```

4. Activate the pools on all nodes.

   Note

   This step must be performed every time a node is rebooted. If it is not, the pool devices will not be accessible.

   Activate the pools using the `pool_assemble -a` command for each node as follows:

   ```
   n01# pool_assemble -a  <-- Activate pools
   alpha_cca assembled
   pool_gfs01 assembled
   pool_gfs02 assembled

   n02# pool_assemble -a  <-- Activate pools
   alpha_cca assembled
   pool_gfs01 assembled
   pool_gfs02 assembled

   n03# pool_assemble -a  <-- Activate pools
   alpha_cca assembled
   pool_gfs01 assembled
   pool_gfs02 assembled

   lcksrv# pool_assemble -a  <-- Activate pools
   alpha_cca assembled
   pool_gfs01 assembled
   pool_gfs02 assembled
   ```

5. Create CCS files.

   a. Create a directory called `/root/alpha` on node `n01` as follows:

      ```
      n01# mkdir /root/alpha
      n01# cd /root/alpha
      ```

   b. Create the `cluster.ccs` file. This file contains the name of the cluster and the name of the nodes where the LOCK_GULM server is run. The file should look like the following:

      ```
      cluster {
         name = "alpha"
         lock_gulm {
            servers = ["lcksrv"]
         }
      }
      ```

c. Create the `nodes.ccs` file. This file contains the name of each node, its IP address, and node-specific I/O fencing parameters. The file should look like the following:

```
nodes {
   n01 {
      ip_interfaces {
         eth0 = "10.0.1.1"
      }
      fence {
         power {
            apc {
            port = 1
            }
         }
      }
   }
   n02 {
      ip_interfaces {
         eth0 = "10.0.1.2"
      }
      fence {
         power {
            apc {
            port = 2
            }
         }
      }
   }
   n03 {
      ip_interfaces {
         eth0 = "10.0.1.3"
      }
      fence {
         power {
            apc {
            port = 3
            }
         }
      }
   }
   lcksrv {
      ip_interfaces {
         eth0 = "10.0.1.4"
      }
      fence {
         power {
            apc {
            port = 4
            }
         }
      }
   }
}
```

Note

If your cluster is running Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5 and later, you can use the optional `usedev` parameter to explicitly specify an IP address rather than relying on an IP address from `libresolv`. For more information about the optional `usedev` parameter, refer to the file format in Figure 6-23 and the example in Example 6-26. Refer to Table 6-3 for syntax description of the `usedev` parameter.

d. Create the `fence.ccs` file. This file contains information required for the fencing method(s) used by the GFS cluster. The file should look like the following:

```
fence_devices {
   apc {
      agent = "fence_apc"
      ipaddr = "10.0.1.10"
      login = "apc"
      passwd = "apc"
   }
}
```

6. Create the CCS Archive on the CCA Device.

Note

This step only needs to be done once and from a single node. It should *not* be performed every time the cluster is restarted.

Use the `ccs_tool` command to create the archive from the CCS configuration files:

```
n01# ccs_tool create /root/alpha /dev/pool/alpha_cca
Initializing device for first time use... done.
```

7. Start the CCS daemon (`ccsd`) on all the nodes.

Note

This step must be performed each time the cluster is rebooted.

The CCA device must be specified when starting `ccsd`.

```
n01# ccsd -d /dev/pool/alpha_cca

n02# ccsd -d /dev/pool/alpha_cca

n03# ccsd -d /dev/pool/alpha_cca

lcksrv# ccsd -d /dev/pool/alpha_cca
```

8. At each node, start the LOCK_GULM server. For example:

```
n01# lock_gulmd

lcksrv# lock_gulmd
```

9. Create the GFS file systems.

Create the first file system on `pool_gfs01` and the second on `pool_gfs02`. The names of the two file systems are `gfs01` and `gfs02`, respectively, as shown in the example:

```
n01# gfs_mkfs -p lock_gulm -t alpha:gfs01 -j 3 /dev/pool/pool_gfs01
Device: /dev/pool/pool_gfs01
Blocksize: 4096
Filesystem Size:1963216
Journals: 3
Resource Groups:30
Locking Protocol:lock_gulm
Lock Table: alpha:gfs01

Syncing...
All Done

n01# gfs_mkfs -p lock_gulm -t alpha:gfs02 -j 3 /dev/pool/pool_gfs02
Device: /dev/pool/pool_gfs02
Blocksize: 4096
Filesystem Size:1963416
Journals: 3
Resource Groups:30
Locking Protocol:lock_gulm
Lock Table: alpha:gfs02

Syncing...
All Done
```

10. Mount the GFS file systems on all the nodes.

    Mount points `/gfs01` and `/gfs02` are used on each node:

```
n01# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n01# mount -t gfs /dev/pool/pool_gfs02 /gfs02

n02# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n02# mount -t gfs /dev/pool/pool_gfs02 /gfs02

n03# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n03# mount -t gfs /dev/pool/pool_gfs02 /gfs02
```

C.5. LOCK_GULM, SLM External, and GNBD

This example configures a cluster with three GFS nodes and two GFS file systems. It will require three nodes for the GFS cluster, one node to run a LOCK_GULM server, and another node for a GNBD server. (A total of five nodes are required in this example.)

This section provides the following information about the example:

- Section C.5.1 *Key Characteristics*
- Section C.5.2 *Kernel Modules Loaded*
- Section C.5.3 *Setup Process*

C.5.1. Key Characteristics

This example configuration has the following key characteristics:

- Fencing device — An APC MasterSwitch (single-switch configuration). Refer to Table C-15 for switch information.
- Number of GFS nodes — 3. Refer to Table C-16 for node information.

- Number of lock server nodes — 1. The lock server is run on one of the GFS nodes (embedded). Refer to Table C-17 for node information.
- Number of GNBD server nodes — 1. Refer to Table C-18 for node information.
- Locking protocol — LOCK_GULM. The LOCK_GULM server is run on a node (the lock server node) that is not mounting GFS but is dedicated as a LOCK_GULM server.
- Number of shared storage devices — 2. GNBD will be used as the transport layer for the storage devices. Refer to Table C-19 for storage device information.
- Number of file systems — 2.
- File system names — `gfs01` and `gfs02`.
- File system mounting — Each GFS node mounts the two file systems.
- Cluster name — `alpha`.

Host Name	IP Address	Login Name	Password
apc	10.0.1.10	apc	apc

Table C-15. APC MasterSwitch Information

Host Name	IP Address	APC Port Number
n01	10.0.1.1	1
n02	10.0.1.2	2
n03	10.0.1.3	3

Table C-16. GFS Node Information

Host Name	IP Address	APC Port Number
lcksrv	10.0.1.4	4

Table C-17. Lock Server Node Information

Host Name	IP Address	APC Port Number
gnbdsrv	10.0.1.5	5

Table C-18. GNBD Server Node Information

Major	Minor	#Blocks	Name
8	16	8388608	sda
8	17	8001	sda1
8	18	8377897	sda2
8	32	8388608	sdb
8	33	8388608	sdb1

Table C-19. Storage Device Information

Notes

The storage must only be visible on the GNBD server node. The GNBD server node will ensure that the storage is visible to the GFS cluster nodes via the GNBD protocol.

For shared storage devices to be visible to the nodes, it may be necessary to load an appropriate device driver. If the shared storage devices are not visible on each node, confirm that the device driver is loaded and that it loaded without errors.

The small partition (`/dev/sda1`) is used to store the cluster configuration information. The two remaining partitions (/`dev/sda2`, `sdb1`) are used for the GFS file systems.

You can display the storage device information at each node in your GFS cluster by running the following command: `cat /proc/partitions`. Depending on the hardware configuration of the GFS nodes, the names of the devices may be different on each node. If the output of the `cat /proc/partitions` command shows only entire disk devices (for example, `/dev/sda` instead of `/dev/sda1`), then the storage devices have not been partitioned. To partition a device, use the `fdisk` command.

C.5.2. Kernel Modules Loaded

Each node must have the following kernel modules loaded:

- `gfs.o`
- `gnbd.o`
- `lock_harness.o`
- `lock_gulm.o`
- `pool.o`

C.5.3. Setup Process

The setup process for this example consists of the following steps:

1. Create and export GNBD devices.

   Create and export a GNBD device for the storage on the GNBD server (`gnbdsrv`) to be used for the GFS file systems and CCA device. In the following example, `gfs01` is the GNBD device used for the pool of the first GFS file system, `gfs02` is the device used for the pool of the second GFS file system, and `cca` is the device used for the CCA device.

```
gnbdsrv# gnbd_export -e cca -d /dev/sda1 -c
gnbdsrv# gnbd_export -e gfs01 -d /dev/sda2 -c
gnbdsrv# gnbd_export -e gfs02 -d /dev/sdb1 -c
```

Caution

The GNBD server should not attempt to use the cached devices it exports — either directly or by importing them. Doing so can cause cache coherency problems.

2. Import GNBD devices on all GFS nodes and the lock server node.

   Use `gnbd_import` to import the GNBD devices from the GNBD server (`gnbdsrv`):

```
n01# gnbd_import -i gnbdsrv
n02# gnbd_import -i gnbdsrv
n03# gnbd_import -i gnbdsrv
lcksrv# gnbd_import -i gnbdsrv
```

3. Create pool configurations for the two file systems.

   Create pool configuration files for each file system's pool: `pool_gfs01` for the first file system, and `pool_gfs02` for the second file system. The two files should look like the following:

```
poolname pool_gfs01
subpools 1
subpool 0 0 1
pooldevice 0 0 /dev/gnbd/gfs01
```

```
poolname pool_gfs02
subpools 1
subpool 0 0 1
pooldevice 0 0 /dev/gnbd/gfs02
```

4. Create a pool configuration for the CCS data.

   Create a pool configuration file for the pool that will be used for CCS data. The pool does not need to be very large. The name of the pool will be `alpha_cca`. (The name of the cluster, `alpha`, followed by `_cca`). The file should look like the following:

```
poolname alpha_cca
subpools 1
subpool 0 0 1
pooldevice 0 0 /dev/gnbd/cca
```

5. Create the pools using the `pool_tool` command.

Note

This operation must take place on a GNBD client node.

   Use the `pool_tool` command to create all the pools as follows:

```
n01# pool_tool -c pool_gfs01.cf pool_gfs02.cf alpha_cca.cf
Pool label written successfully from pool_gfs01.cf
Pool label written successfully from pool_gfs02.cf
Pool label written successfully from alpha_cca.cf
```

6. Activate the pools on all nodes.

Note

This step must be performed every time a node is rebooted. If it is not, the pool devices will not be accessible.

Activate the pools using the `pool_assemble -a` command for each node as follows:

```
n01# pool_assemble -a  <-- Activate pools
alpha_cca assembled
pool_gfs01 assembled
pool_gfs02 assembled

n02# pool_assemble -a  <-- Activate pools
alpha_cca assembled
pool_gfs01 assembled
pool_gfs02 assembled

n03# pool_assemble -a  <-- Activate pools
alpha_cca assembled
pool_gfs01 assembled
pool_gfs02 assembled

lcksrv# pool_assemble -a  <-- Activate pools
alpha_cca assembled
pool_gfs01 assembled
pool_gfs02 assembled
```

7. Create CCS files.

   a. Create a directory called `/root/alpha` on node `n01` as follows:

   ```
   n01# mkdir /root/alpha
   n01# cd /root/alpha
   ```

   b. Create the `cluster.ccs` file. This file contains the name of the cluster and the name of the nodes where the LOCK_GULM server is run. The file should look like the following:

   ```
   cluster {
      name = "alpha"
      lock_gulm {
         servers = ["lcksrv"]
      }
   }
   ```

   c. Create the `nodes.ccs` file. This file contains the name of each node, its IP address, and node-specific I/O fencing parameters. The file should look like the following:

   ```
   nodes {
      n01 {
         ip_interfaces {
            eth0 = "10.0.1.1"
         }
         fence {
            power {
               apc {
               port = 1
               }
            }
         }
      }
      n02 {
         ip_interfaces {
            eth0 = "10.0.1.2"
         }
         fence {
   ```

```
            power {
               apc {
               port = 2
               }
            }
         }
      }
      n03 {
         ip_interfaces {
            eth0 = "10.0.1.3"
         }
         fence {
            power {
               apc {
               port = 3
               }
            }
         }
      }
      lcksrv {
         ip_interfaces {
            eth0 = "10.0.1.4"
         }
         fence {
            power {
               apc {
               port = 4
               }
            }
         }
      }
      gnbdsrv {
         ip_interfaces {
            eth0 = "10.0.1.5"
         }
         fence {
            power {
               apc {
               port = 5
               }
            }
         }
      }
   }
```

Note

If your cluster is running Red Hat GFS 6.0 for Red Hat Enterprise Linux 3 Update 5 and later, you can use the optional `usedev` parameter to explicitly specify an IP address rather than relying on an IP address from `libresolv`. For more information about the optional `usedev` parameter, refer to the file format in Figure 6-23 and the example in Example 6-26. Refer to Table 6-3 for syntax description of the `usedev` parameter.

d. Create the `fence.ccs` file. This file contains information required for the fencing method(s) used by the GFS cluster. The file should look like the following:

```
fence_devices {
   apc {
      agent = "fence_apc"
      ipaddr = "10.0.1.10"
      login = "apc"
      passwd = "apc"
   }
}
```

8. Create the CCS Archive on the CCA Device.

   Note

   This step only needs to be done once and from a single node. It should *not* be performed every time the cluster is restarted.

   Use the `ccs_tool` command to create the archive from the CCS configuration files:

   ```
   n01# ccs_tool create /root/alpha /dev/pool/alpha_cca
   Initializing device for first time use... done.
   ```

9. Start the CCS daemon (`ccsd`) on all the nodes.

   Note

   This step must be performed each time the cluster is rebooted.

   The CCA device must be specified when starting `ccsd`.

   ```
   n01# ccsd -d /dev/pool/alpha_cca

   n02# ccsd -d /dev/pool/alpha_cca

   n03# ccsd -d /dev/pool/alpha_cca

   lcksrv# ccsd -d /dev/pool/alpha_cca
   ```

10. At each node, start the LOCK_GULM server. For example:

    ```
    n01# lock_gulmd

    lcksrv# lock_gulmd
    ```

11. Create the GFS file systems.

    Create the first file system on `pool_gfs01` and the second on `pool_gfs02`. The names of the two file systems are `gfs01` and `gfs02`, respectively, as shown in the example:

    ```
    n01# gfs_mkfs -p lock_gulm -t alpha:gfs01 -j 3 /dev/pool/pool_gfs01
    Device: /dev/pool/pool_gfs01
    Blocksize: 4096
    Filesystem Size:1963216
    Journals: 3
    Resource Groups:30
    Locking Protocol:lock_gulm
    Lock Table: alpha:gfs01

    Syncing...
    All Done

    n01# gfs_mkfs -p lock_gulm -t alpha:gfs02 -j 3 /dev/pool/pool_gfs02
    Device: /dev/pool/pool_gfs02
    ```

```
Blocksize: 4096
Filesystem Size:1963416
Journals: 3
Resource Groups:30
Locking Protocol:lock_gulm
Lock Table: alpha:gfs02

Syncing...
All Done
```

12. Mount the GFS file systems on all the nodes.

    Mount points `/gfs01` and `/gfs02` are used on each node:

```
n01# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n01# mount -t gfs /dev/pool/pool_gfs02 /gfs02

n02# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n02# mount -t gfs /dev/pool/pool_gfs02 /gfs02

n03# mount -t gfs /dev/pool/pool_gfs01 /gfs01
n03# mount -t gfs /dev/pool/pool_gfs02 /gfs02
```

C.6. LOCK_NOLOCK

This example sets up a single node mounting two GFS file systems. Only a single node is required because the file system will not be mounted in cluster mode.

This section provides the following information about the example:

- Section C.6.1 *Key Characteristics*
- Section C.6.2 *Kernel Modules Loaded*
- Section C.6.3 *Setup Process*

C.6.1. Key Characteristics

This example configuration has the following key characteristics:

- Number of GFS nodes — 1. Refer to Table C-20 for node information.
- Locking protocol — LOCK_NOLOCK.
- Number of shared storage devices — 1. One direct-attached storage device is used. Refer to Table C-21for storage device information.
- Number of file systems — 2.
- File system names — `gfs01` and `gfs02`.
- File system mounting — The GFS node mounts the two file systems.

Host Name	IP Address
n01	10.0.1.1

Table C-20. GFS Node Information

Major	Minor	#Blocks	Name
8	16	8388608	sda
8	17	8001	sda1
8	32	8388608	sdb
8	33	8388608	sdb1

Table C-21. Storage Device Information

Notes

For storage to be visible to the node, it may be necessary to load an appropriate device driver. If the storage is not visible on the node, confirm that the device driver is loaded and that it loaded without errors.

The two partitions (`/dev/sda1`, `sdb1`) are used for the GFS file systems.

You can display the storage device information at each node in your GFS cluster by running the following command: `cat /proc/partitions`. Depending on the hardware configuration of the GFS nodes, the names of the devices may be different on each node. If the output of the `cat /proc/partitions` command shows only entire disk devices (for example, `/dev/sda` instead of `/dev/sda1`), then the storage devices have not been partitioned. To partition a device, use the `fdisk` command.

C.6.2. Kernel Modules Loaded

Each node must have the following kernel modules loaded:

- `gfs.o`
- `lock_harness.o`
- `lock_nolock.o`
- `pool.o`

C.6.3. Setup Process

The setup process for this example consists of the following steps:

1. Create pool configurations for the two file systems.

   Create pool configuration files for each file system's pool: `pool_gfs01` for the first file system, and `pool_gfs02` for the second file system. The two files should look like the following:
   ```
   poolname pool_gfs01
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sda1
   ```
   ```
   poolname pool_gfs02
   subpools 1
   subpool 0 0 1
   pooldevice 0 0 /dev/sdb1
   ```

2. Use the `pool_tool` command to create all the pools as follows:

```
n01# pool_tool -c pool_gfs01.cf pool_gfs02.cf
Pool label written successfully from pool_gfs01.cf
Pool label written successfully from pool_gfs02.cf
```

3. Activate the pools.

   Note

   This step must be performed every time a node is rebooted. If it is not, the pool devices will not be accessible.

   Activate the pools using the `pool_assemble -a` command as follows:

   ```
   n01# pool_assemble -a
   pool_gfs01 assembled
   pool_gfs02 assembled
   ```

4. Create the CCS Archive.

   a. Create a directory called `/root/alpha` on node `n01` as follows:

      ```
      n01# mkdir /root/alpha
      n01# cd /root/alpha
      ```

   b. Create the CCS Archive on the CCA Device.

      Note

      This step only needs to be done once. It should *not* be performed every time the cluster is restarted.

      Use the `ccs_tool` command to create the archive from the CCS configuration files:

      ```
      n01# ccs_tool create /root/alpha /root/alpha_cca
      Initializing device for first time use... done.
      ```

5. Start the CCS daemon (`ccsd`).

   Note

   This step must be performed each time the node is rebooted.

   The CCA device must be specified when starting `ccsd`.

   ```
   n01# ccsd -d /dev/pool/alpha_cca
   ```

6. Create the GFS file systems.

   Create the first file system on `pool_gfs01` and the second on `pool_gfs02`. The names of the two file systems are `gfs01` and `gfs02`, respectively, as shown in the example:

   ```
   n01# gfs_mkfs -p lock_gulm -t alpha:gfs01 -j 1 /dev/pool/pool_gfs01
   Device: /dev/pool/pool_gfs01
   Blocksize: 4096
   Filesystem Size:1963216
   Journals: 1
   Resource Groups:30
   Locking Protocol:lock_nolock
   Lock Table:

   Syncing...
   ```

```
All Done

n01# gfs_mkfs -p lock_gulm -t alpha:gfs02 -j 1 /dev/pool/pool_gfs02
Device: /dev/pool/pool_gfs02
Blocksize: 4096
Filesystem Size:1963416
Journals: 1
Resource Groups:30
Locking Protocol:lock_nolock
Lock Table:

Syncing...
All Done
```

7. Mount the GFS file systems on the nodes.

   Mount points `/gfs01` and `/gfs02` are used on the node:

   ```
   n01# mount -t gfs /dev/pool/pool_gfs01 /gfs01

   n01# mount -t gfs /dev/pool/pool_gfs02 /gfs02
   ```

Index

A

activating your subscription, iv
adding journals to a file system, 101
administrative options, 77
  comparing CCS configuration files to a CCS archive, 78
  extracting files from a CCS archive, 77
  listing files in a CCS archive, 78
APC MasterSwitch information (examples) table, 137, 142, 149, 154, 160
atime, configuring updates, 105
  mounting with noatime, 106
  tuning atime quantum, 106
audience, i

B

block devices
  checking before creating pool configuration file, 25
  scanning for, 25

C

CCA
  CCA file and server
    alternative methods to using a CCA device, 79
    creating a CCA file, 80
    starting the CCS daemon, 81
    starting the CCS server, 80
  local CCA files
    alternative methods to using a CCA device, 82
CCS archive
  creating, 75
CCS file location for GNBD multipath cluster table, 125
CDPN variable values table, 111
cluster configuration management, 6
cluster configuration system (CCS)
  administrative options, using, 77
    comparing CCS configuration files to a CCS archive, 78
    extracting files from a CCS archive, 77
    listing files in a CCS archive, 78
  alternative methods to using a CCA device, 79
    CCA file and server, 79
    local CCA files, 82
  changing CCS configuration files, 79
  combining CCS methods, 82
  creating a CCS archive, 75
  starting CCS in the cluster, 76
  using, 75
  using clustering and locking systems, 85
    fencing and LOCK_GULM, 86
    locking system overview, 85
    LOCK_GULM, 85
    LOCK_NOLOCK, 87
    number of LOCK_GULM servers, 86
    selection of LOCK_GULM servers, 85
    shutting down a LOCK_GULM server, 86
    starting LOCK_GULM servers, 86
Cluster Configuration System Files
  (see system files)
Cluster Configuration System, setting up and starting
  configuration, initial, 19
cluster management, fencing, recovery, 6
cluster volume management, 5
cluster.ccs, 39
cluster.ccs variables table, 40
clustering, starting
  configuration, initial, 19
configuration file
  creating for a new volume, 26
    examples (config file), 27
configuration, before, 9
configuration, initial, 17
  initial tasks, 17
    Cluster Configuration System, setting up and starting, 19
    file systems, setting up and mounting, 19
    logical devices, setting up, 18
    starting clustering and locking systems, 19
  prerequisite tasks, 17
console access
  system requirements, 12
conventions
  document, i

D

data journaling, 104
direct I/O, 103
  directory attribute, 104
  file attribute, 103
  O_DIRECT, 103
displaying extended GFS information and statistics, 108
dual power
  multipath FC fencing, 38

E

examples
basic GFS examples, 137
LOCK_GULM, RLM embedded, 137
key characteristics, 137
setup process, 139
LOCK_GULM, RLM external, 142
key characteristics, 142
setup process, 144
LOCK_GULM, SLM embedded, 148
key characteristics, 149, 154
setup process, 150, 155
LOCK_GULM, SLM embedded, and GNBD
key characteristics, 159
LOCK_GULM, SLM external, 154
LOCK_GULM, SLM external, and GNBD, 159
setup process, 161
LOCK_NOLOCK, 166
key characteristics, 166
setup process, 167

F

features, new and changed, 1
feedback, iv
fence.ccs, 41
fence.css variables table, 47
fencing, 115
fencing methods, 115
APC MasterSwitch, 116
Brocade FC switch, 117
GNBD, 118
HP RILOE card, 118
manual, 118
Vixel FC switch, 117
WTI network power switch, 117
how the fencing system works, 115
fencing methods and agents table, 116
fibre channel network requirements table, 12
fibre channel storage device requirements table, 12
fibre channel storage devices
system requirements, 12
fibre channel storage network
system requirements, 11
file system
adding journals, 101
atime, configuring updates, 105
mounting with noatime, 106
tuning atime quantum, 106
context-dependent path names (CDPNs), 110
data journaling, 104
direct I/O, 103
directory attribute, 104
file attribute, 103
O_DIRECT, 103
growing, 99
making, 89
mounting, 91
quota management, 94
disabling/enabling quota accounting, 98
disabling/enabling quota enforcement, 98
displaying quota limits, 95
setting quotas, 94
synchronizing quotas, 97
repairing, 109
suspending activity, 107
unmounting, 93
file systems, setting up and mounting
configuration, initial, 19

G

GFS
atime, configuring updates, 105
mounting with noatime, 106
tuning atime quantum, 106
direct I/O, 103
directory attribute, 104
file attribute, 103
O_DIRECT, 103
displaying extended information and statistics, 108
managing, 89
quota management, 94
disabling/enabling quota accounting, 98
disabling/enabling quota enforcement, 98
displaying quota limits, 95
setting quotas, 94
synchronizing quotas, 97
shutting down a cluster, 112
starting a cluster, 112
upgrading
(see upgrading GFS)
GFS and lock server node information (examples) table, 138, 149
GFS functions, 5
cluster configuration management, 6
cluster management, fencing, recovery, 6
cluster volume management, 5
lock management, 6
GFS kernel modules, loading
installation tasks, 15
GFS node information (examples) table, 143, 154, 160, 166
GFS RPM installation
installation tasks, 15
GFS software subsystem components table, 7
GFS software subsystems, 7
GFS-specific options for adding journals table, 102
GFS-specific options for expanding file systems table, 101

gfs_mkfs command options table, 90
GNBD
driver and command usage, 121
exporting from a server, 121
importing on a client, 123
using, 121
using GFS on a GNBD server node, 125
using GNBD multipath, 123
CCS file location, 124
fencing GNBD server nodes, 125
Linux page caching, 124
lock server startup, 124
GNBD multipath, 38
GNBD server node information (examples) table, 160
growing a file system, 99

I

I/O fencing
(see fencing)
system requirements, 12
init.d
usage, 128
using GFS init.d scripts, 127
init.d scripts
using, 127
initial tasks
Cluster Configuration System, setting up and starting, 19
configuration, initial, 17
file systems, setting up and mounting, 19
logical devices, setting up, 18
starting clustering and locking systems, 19
installation tasks, 14
GFS kernel modules, loading, 15
GFS RPM installation, 15
installing system software, 13
installation tasks, 14
GFS RPM installation, 15
loading GFS kernel modules, 15
prerequisite tasks, 13
clock synchronization software, 14
perl-Net-Telnet module, 13
persistent major number utility, 14
stunnel utility, 14
introduction, i
audience, i
references, v

L

lock management, 6
lock server node information (examples) table, 143, 154, 160
locking system
LOCK_GULM, 85
fencing, 86
LOCK_GULM severs
number of, 86
selection of, 85
shutting down, 86
starting, 86
LOCK_NOLOCK, 87
overview, 85
using, 85
locking systems
configuration, initial, 19
logical devices, setting up
configuration, initial, 18

M

making a file system, 89
managing GFS, 89
mount table, 93
mounting a file system, 91

N

network power switches
system requirements, 12
nodes.ccs, 51
nodes.css variables table, 64

O

overview, 1
configuration, before, 9
economy, 2
features, new and changed, 1
GFS functions, 5
cluster configuration management, 6
cluster management, fencing, recovery, 6
cluster volume management, 5
lock management, 6
GFS software subsystems, 7
performance, 2
scalability, 2

P

path names, context-dependent (CDPNs), 110
platform
  system requirements, 11
platform requirements table, 11
pool configuration
  displaying information, 30
pool configuration file keyword and variable descriptions table, 26
pool management
  commands, 22
    pool_assemble, 23
    pool_info, 23
    pool_mp, 24
    pool_tool, 22
pool volume, 21
  activating, 28
  adjusting multipathing, 36
  creating, 28
  deactivating, 28
  displaying information, 34
  erasing, 32
  examples (activate/deactivate), 29
  examples (displaying configuration file information), 30
  growing, 30
  minor number, changing, 33
  overview, 21
  renaming, 32
  statistics, 35
Pool Volume Manager
  configuration file
    key words and variable descriptions, 26
  using, 21
pool_assemble command functions table, 23
pool_assemble command options table, 23
pool_info command functions table, 23
pool_info command options table, 24
pool_mp command functions table, 24
pool_mp command options table, 24
pool_tool command functions table, 22
pool_tool command options table, 22
preface
  (see introduction)
prerequisite tasks
  configuration, initial, 17
  installing system software, 13
    clock synchronization software, 14
    perl-Net-Telnet module, 13
    persistent major number, 14
    stunnel utility, 14

Q

quota management, 94
  disabling/enabling quota accounting, 98
  disabling/enabling quota enforcement, 98
  displaying quota limits, 95
  setting quotas, 94
  synchronizing quotas, 97

R

recommended references table, v
Red Hat GFS with Red Hat Cluster Suite
  changes to Red Hat Cluster Suite, 132
  installation scenarios, 132
  terminology, 131
  using, 131
references, recommended, v
registering your subscription, iv
repairing a file system, 109

S

shutting down a GFS cluster, 112
software, installing, 13
starting a GFS cluster, 112
storage device information (examples) table, 138, 143, 149, 155, 161, 167
subscription registration, iv
suspending activity on a file system, 107
system files
  CCS file creation tasks, 38
  cluster.ccs, 39
  creating, 37
  dual power, 38
  fence.ccs, 41
  GNBD multipath, 38
  multipath FC fencing, 38
  nodes.ccs, 51
  prerequisite tasks, 37
system requirements, 11
  console access, 12
  fibre channel storage devices, 12
  fibre channel storage network, 11
  I/O fencing, 12
  network power switches, 12
  platform, 11
  TCP/IP network, 11

T

tables
APC MasterSwitch information (examples), 137, 142, 149, 154, 160
CCS file location for GNBD multipath cluster, 125
CDPN variable values, 111
cluster.ccs variables, 40
fence.css variables, 47
fencing methods and agents, 116
fibre channel network requirements, 12
fibre channel storage device requirements, 12
GFS and lock server node information (examples), 138, 149
GFS node information (examples), 143, 154, 160, 166
GFS software subsystem components, 7
GFS-specific options for adding journals, 102
GFS-specific options for expanding file systems, 101
gfs_mkfs command options, 90
GNBD server node information (examples), 160
lock server node information (examples), 143, 154, 160
mount options, 93
nodes.css variables, 64
platform requirements, 11
pool configuration file keyword and variable descriptions, 26
pool_assemble command functions, 23
pool_assemble command options, 23
pool_info command functions, 23
pool_info command options, 24
pool_mp command functions, 24
pool_mp command options, 24
pool_tool command functions, 22
pool_tool command options, 22
recommended references, v
storage device information (examples), 138, 143, 149, 155, 161, 167
TCP/IP network
system requirements, 11

U

unmounting a file system, 93
upgrading GFS, 135
upgrade procedure, 135
using Red Hat GFS with Red Hat Cluster Suite, 131

V

volume, new
checking for block devices before creating pool configuration file, 25
creating a configuration file, 26

Colophon

The manuals are written in DocBook SGML v4.1 format. The HTML and PDF formats are produced using custom DSSSL stylesheets and custom jade wrapper scripts. The DocBook SGML files are written in **Emacs** with the help of PSGML mode.

Garrett LeSage created the admonition graphics (note, tip, important, caution, and warning). They may be freely redistributed with the Red Hat documentation.

The Red Hat Product Documentation Team consists of the following people:

Sandra A. Moore — Primary Writer/Maintainer of the *Red Hat Enterprise Linux Installation Guide for x86, Itanium™, AMD64, and Intel® Extended Memory 64 Technology (Intel® EM64T)*; Primary Writer/Maintainer of the *Red Hat Enterprise Linux Installation Guide for the IBM® POWER Architecture*; Primary Writer/Maintainer of the *Red Hat Enterprise Linux Installation Guide for the IBM® S/390® and IBM® eServer™ zSeries® Architectures*

John Ha — Primary Writer/Maintainer of the *Red Hat Cluster Suite Configuring and Managing a Cluster*; Co-writer/Co-maintainer of the *Red Hat Enterprise Linux Security Guide*; Maintainer of custom DocBook stylesheets and scripts

Edward C. Bailey — Primary Writer/Maintainer of the *Red Hat Enterprise Linux Introduction to System Administration*; Primary Writer/Maintainer of the *Release Notes*; Contributing Writer to the *Red Hat Enterprise Linux Installation Guide for x86, Itanium™, AMD64, and Intel® Extended Memory 64 Technology (Intel® EM64T)*

Karsten Wade — Primary Writer/Maintainer of the *Red Hat SELinux Application Development Guide*; Primary Writer/Maintainer of the *Red Hat SELinux Policy Writing Guide*

Andrius Benokraitis — Primary Writer/Maintainer of the *Red Hat Enterprise Linux Reference Guide*; Co-writer/Co-maintainer of the *Red Hat Enterprise Linux Security Guide*; Contributing Writer to the *Red Hat Enterprise Linux System Administration Guide*

Paul Kennedy — Primary Writer/Maintainer of the *Red Hat GFS Administrator's Guide*; Contributing Writer to the *Red Hat Cluster Suite Configuring and Managing a Cluster*

Mark Johnson — Primary Writer/Maintainer of the *Red Hat Enterprise Linux Desktop Configuration and Administration Guide*

Melissa Goldin — Primary Writer/Maintainer of the *Red Hat Enterprise Linux Step By Step Guide*

The Red Hat Localization Team consists of the following people:

Amanpreet Singh Alam — Punjabi translations

Jean-Paul Aubry — French translations

David Barzilay — Brazilian Portuguese translations

Runa Bhattacharjee — Bengali translations

Chester Cheng — Traditional Chinese translations

Verena Fuehrer — German translations

Kiyoto Hashida — Japanese translations

N. Jayaradha — Tamil translations

Michelle Jiyeen Kim — Korean translations

Yelitza Louze — Spanish translations

Noriko Mizumoto — Japanese translations

Ankitkumar Rameshchandra Patel — Gujarati translations

Rajesh Ranjan — Hindi translations

Nadine Richter — German translations
Audrey Simons — French translations
Francesco Valente — Italian translations
Sarah Wang — Simplified Chinese translations
Ben Hung-Pin Wu — Traditional Chinese translations