Red Hat Enterprise Linux 4

Global File System

Red Hat Global File System

Red Hat Enterprise Linux 4 Global File System
Red Hat Global File System
Edition 1.1

1801 Varsity Drive
Raleigh, NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701

This book provides information about installing, configuring, and maintaining Red Hat GFS (Red Hat Global File System) for Red Hat Enterprise Linux 4.

Introduction **v**
1. Audience v
2. Related Documentation v
3. Document Conventions vi
3.1. Typographic Conventions vi
3.2. Pull-quote Conventions vii
3.3. Notes and Warnings viii
4. Feedback viii
5. Recommended References ix

1. GFS Overview **1**
1.1. Performance, Scalability, and Economy 1
1.1.1. Superior Performance and Scalability 1
1.1.2. Performance, Scalability, Moderate Price 2
1.1.3. Economy and Performance 3
1.2. GFS Functions 3
1.3. GFS Software Subsystems 4
1.4. Before Setting Up GFS 5

2. System Requirements **7**
2.1. Platform Requirements 7
2.2. Red Hat Cluster Suite 7
2.3. Fencing 7
2.4. Fibre Channel Storage Network 7
2.5. Fibre Channel Storage Devices 8
2.6. Network Power Switches 8
2.7. Console Access 8
2.8. Installing GFS 8

3. Getting Started **11**
3.1. Prerequisite Tasks 11
3.2. Initial Setup Tasks 11

4. Managing GFS **13**
4.1. Making a File System 13
4.2. Mounting a File System 15
4.3. Unmounting a File System 17
4.4. GFS Quota Management 18
4.4.1. Setting Quotas 18
4.4.2. Displaying Quota Limits and Usage 19
4.4.3. Synchronizing Quotas 20
4.4.4. Disabling/Enabling Quota Enforcement 21
4.4.5. Disabling/Enabling Quota Accounting 22
4.5. Growing a File System 23
4.6. Adding Journals to a File System 25
4.7. Direct I/O 26
4.7.1. **O_DIRECT** 27
4.7.2. GFS File Attribute 27
4.7.3. GFS Directory Attribute 27
4.8. Data Journaling 28
4.9. Configuring **atime** Updates 29
4.9.1. Mount with **noatime** 29
4.9.2. Tune GFS **atime** Quantum 30
4.10. Suspending Activity on a File System 31
4.11. Displaying Extended GFS Information and Statistics 31
4.12. Repairing a File System 32

4.13. Context-Dependent Path Names 33
A. Upgrading GFS **37**
B. Revision History **41**
Index **43**

Introduction

Welcome to the *Global File System Configuration and Administration* document. This book provides information about installing, configuring, and maintaining Red Hat GFS (Red Hat Global File System). Red Hat GFS depends on the cluster infrastructure of Red Hat Cluster Suite. For information about Red Hat Cluster Suite refer to *Red Hat Cluster Suite Overview* and *Configuring and Managing a Red Hat Cluster*.

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. Related Documentation

For more information about using Red Hat Enterprise Linux, refer to the following resources:

- *Red Hat Enterprise Linux Installation Guide* — Provides information regarding installation of Red Hat Enterprise Linux.
- *Red Hat Enterprise Linux Introduction to System Administration* — Provides introductory information for new Red Hat Enterprise Linux system administrators.
- *Red Hat Enterprise Linux System Administration Guide* — Provides more detailed information about configuring Red Hat Enterprise Linux to suit your particular needs as a user.
- *Red Hat Enterprise Linux Reference Guide* — Provides detailed information suited for more experienced users to reference when needed, as opposed to step-by-step instructions.
- *Red Hat Enterprise Linux Security Guide* — Details the planning and the tools involved in creating a secured computing environment for the data center, workplace, and home.

For more information about Red Hat Cluster Suite for Red Hat Enterprise Linux, refer to the following resources:

- *Red Hat Cluster Suite Overview* — Provides a high level overview of the Red Hat Cluster Suite.
- *Configuring and Managing a Red Hat Cluster* — Provides information about installing, configuring and managing Red Hat Cluster components.
- *LVM Administrator's Guide: Configuration and Administration* — Provides a description of the Logical Volume Manager (LVM), including information on running LVM in a clustered environment.
- *Using GNBD with Global File System* — Provides an overview on using Global Network Block Device (GNBD) with Red Hat GFS.
- *Using Device-Mapper Multipath* — Provides information about using the Device-Mapper Multipath feature of Red Hat Enterprise Linux.

- *Linux Virtual Server Administration* — Provides information on configuring high-performance systems and services with the Linux Virtual Server (LVS).
- *Red Hat Cluster Suite Release Notes* — Provides information about the current release of Red Hat Cluster Suite.

Red Hat Cluster Suite documentation and other Red Hat documents are available in HTML and PDF versions online at the following location:

http://www.redhat.com/docs

3. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

3.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F2`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F1`** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

[1] https://fedorahosted.org/liberation-fonts/

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **ssh *username*@*domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.
>
> The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.
>
> To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

3.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
```

```
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

3.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

4. Feedback

If you spot a typo, 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://bugzilla.redhat.com/bugzilla/*) against the component **rh-cs**.

Be sure to mention the manual's identifier:

```
```

```
rh-gfsg(EN)-4.8 (2010-03-17T16:33)
```

By mentioning this manual's identifier, we 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.

5. Recommended References

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

Table 1. Recommended References 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.
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.

GFS Overview

Red Hat GFS is a cluster file system that is available with Red Hat Cluster Suite. Red Hat GFS nodes are configured and managed with Red Hat Cluster Suite configuration and management tools. Red Hat GFS provides data sharing among GFS nodes in a Red Hat cluster. GFS provides a single, consistent view of the file-system name space across the GFS nodes in a Red Hat cluster. GFS 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, “Performance, Scalability, and Economy”*
- *Section 1.2, “GFS Functions”*
- *Section 1.3, “GFS Software Subsystems”*
- *Section 1.4, “Before Setting Up GFS”*

1.1. 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).

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

- *Section 1.1.1, “Superior Performance and Scalability”*
- *Section 1.1.2, “Performance, Scalability, Moderate Price”*
- *Section 1.1.3, “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.

1.1.1. 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, “GFS with a SAN”* provides superior file performance for shared files and file systems. Linux applications run directly on GFS nodes. Without file protocols or storage servers to slow data access, performance is similar to individual Linux servers with directly connected storage; yet, each GFS application node has equal access to all data files. GFS supports up to 16 GFS nodes.

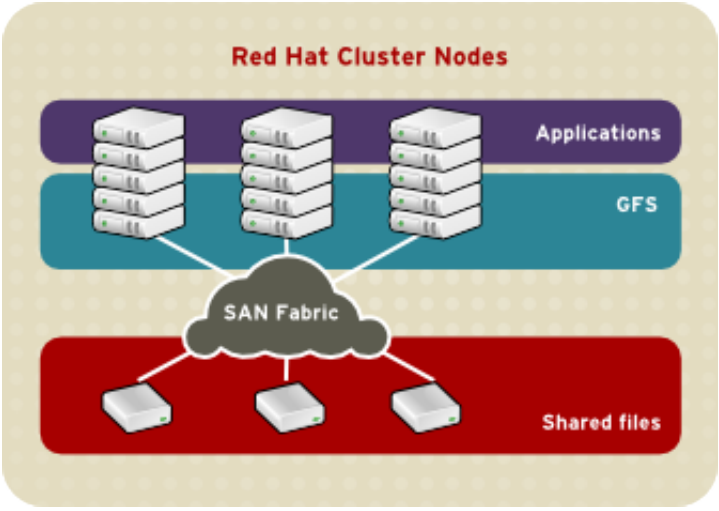

Figure 1.1. GFS with a SAN

1.1.2. Performance, Scalability, Moderate Price

Multiple Linux client applications on a LAN can share the same SAN-based data as shown in *Figure 1.2, "GFS and GNBD with a SAN"*. 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.

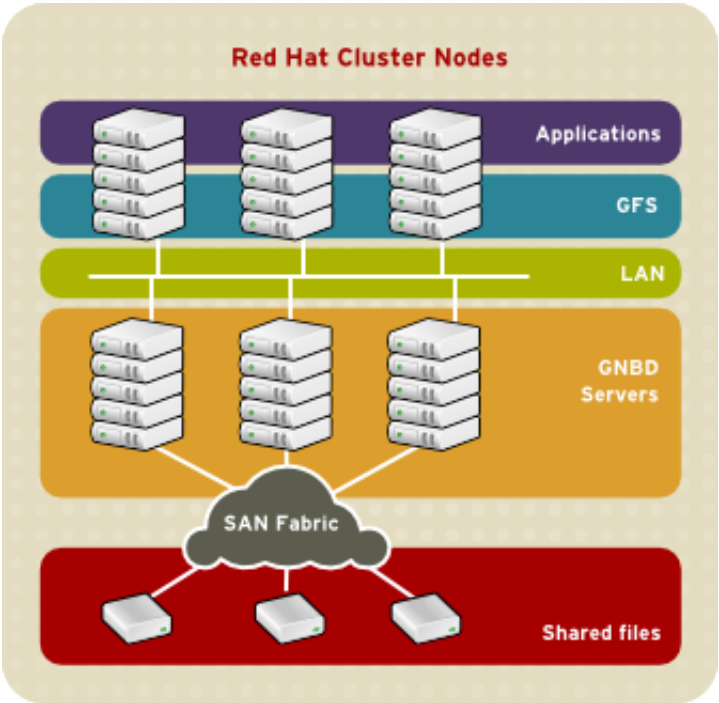

Figure 1.2. GFS and GNBD with a SAN

1.1.3. Economy and Performance

Figure 1.3, "GFS and GNBD with Directly Connected Storage" 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.

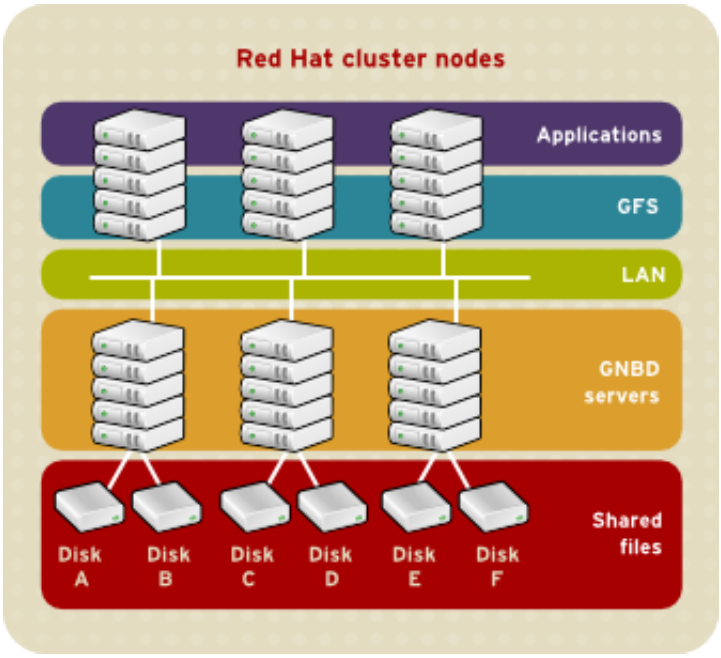

Figure 1.3. GFS and GNBD with Directly Connected Storage

1.2. 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. Cluster management of GFS nodes is managed through Red Hat Cluster Suite. Volume management is managed through CLVM (Cluster Logical Volume Manager). For information about Red Hat Cluster Suite refer to *Configuring and Managing a Red Hat Cluster*. For information about using CLVM, refer to *LVM Administrator's Guide*.

Note

CLVM is a cluster-wide implementation of LVM, enabled by the CLVM daemon, **`clvmd`** running in a Red Hat Cluster Suite cluster. The daemon makes it possible to use LVM2 to manage logical volumes across a cluster, allowing all nodes in the cluster to share the logical volumes.

GFS provides the following main functions:

- Making a File System
- Mounting a File System
- Unmounting a File System
- GFS Quota Management
- Growing a File System

- Adding Journals to a File System
- Direct I/O
- Data Journaling
- Configuring **atime** Updates
- Suspending Activity on a File System
- Displaying Extended GFS Information and Statistics
- Repairing a File System
- Context-Dependent Path Names (CDPN)

1.3. GFS Software Subsystems

Table 1.1, "GFS Software Subsystem Components" summarizes the GFS Software subsystems and their components.

Table 1.1. GFS Software Subsystem Components

Software Subsystem	Components	Description
GFS	**gfs.ko**	Kernel module that implements the GFS file system and is loaded on GFS cluster nodes.
	gfs_fsck	Command that repairs an unmounted 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.
	gfs_mkfs	Command that creates a GFS file system on a storage device.
	gfs_quota	Command that manages quotas on a mounted GFS file system.
	gfs_tool	Command that configures or tunes a GFS file system. This command can also gather a variety of information about the file system.
	lock_harness.ko	Implements a pluggable lock module interface for GFS that allows for a variety of locking mechanisms to be used (for example, the DLM lock module, **lock_dlm.ko**).
	lock_dlm.ko	A lock module that implements DLM locking for GFS. It plugs into the lock harness, **lock_harness.ko** and communicates with the DLM lock manager in Red Hat Cluster Suite.

Software Subsystem	Components	Description
	lock_gulm.ko	A lock module that implements GULM locking for GFS. It plugs into the lock harness, **lock_harness.ko** and communicates with the GULM lock manager in Red Hat Cluster Suite.
	lock_nolock.ko	A lock module for use when GFS is used as a local file system only. It plugs into the lock harness, **lock_harness.ko** and provides local locking.

1.4. Before Setting Up GFS

Before you install and set up GFS, note the following key characteristics of your GFS file systems:

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. For example, this book uses file-system names **gfs1** and **gfs2** in some example procedures.

Journals
: Determine the number of journals for your GFS file systems. One journal is required for each node that mounts a GFS file system. Make sure to account for additional journals needed for future expansion.

GFS nodes
: Determine which nodes in the Red Hat Cluster Suite will mount the GFS file systems.

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 setting up GNBD clients later. For information on using GNBD with GFS, see the *Using GNBD with Global File System* document.

Storage devices and partitions
: Determine the storage devices and partitions to be used for creating logical volumes (via CLVM) in the file systems.

System Requirements

This chapter describes the system requirements for Red Hat GFS with Red Hat Enterprise Linux 5 and consists of the following sections:

- *Section 2.1, "Platform Requirements"*
- *Section 2.2, "Red Hat Cluster Suite"*
- *Section 2.3, "Fencing"*
- *Section 2.4, "Fibre Channel Storage Network"*
- *Section 2.5, "Fibre Channel Storage Devices"*
- *Section 2.6, "Network Power Switches"*
- *Section 2.7, "Console Access"*

2.1. Platform Requirements

Table 2.1, "Platform Requirements" shows the platform requirements for GFS.

Table 2.1. Platform Requirements

Operating System	Hardware Architecture	RAM
Red Hat Enterprise Linux AS, ES, or WS, Version 4 or later	ia64, x86-64, x86 SMP supported	256 MB, minimum

2.2. Red Hat Cluster Suite

Red Hat GFS runs with Red Hat Cluster Suite 4.0 or later. The Red Hat Cluster Suite software must be installed on the cluster nodes before you can install and run Red Hat GFS.

Note

Red Hat Cluster Suite 4.0 and later provides the infrastructure for application failover in the cluster and network communication among GFS nodes (and other Red Hat Cluster Suite nodes).

2.3. Fencing

You must configure each GFS node in your Red Hat cluster for at least one form of fencing. Fencing is configured and managed in Red Hat Cluster Suite. For more information about fencing options, refer to *Configuring and Managing a Red Hat Cluster*.

2.4. Fibre Channel Storage Network

Table 2.2, "Fibre Channel Network Requirements" shows requirements for GFS nodes that are to be connected to a Fibre Channel SAN.

Table 2.2. Fibre Channel Network Requirements

Requirement	Description
HBA (Host Bus Adapter)	One HBA minimum per GFS node

Requirement	Description
Connection method	Fibre Channel switch *Note:* If an FC switch is used for fencing, you may want to consider using Brocade, McData, or Vixel FC switches, for which Red Hat Cluster Suite fencing agents exist. Refer to *Configuring and Managing a Red Hat Cluster* for more information about supported fencing agents. *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.

2.5. Fibre Channel Storage Devices

Table 2.3, "Fibre Channel Storage Device Requirements" shows requirements for Fibre Channel devices that are to be connected to a GFS cluster.

Table 2.3. Fibre Channel Storage Device Requirements

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	GFS is based on a 64-bit architecture, which can theoretically accommodate an 8 EB file system. However, the current supported maximum size of a GFS file system is 25 TB. If your system requires GFS file systems larger than 25 TB, contact your Red Hat service representative. When determining the size of your file system, you should consider your recovery needs. Running the **`fsck`** command on the file system can take a long time and consume a large amount of memory. Additionally, in the event of a disk or disk-subsytem failure, recovery time is limited by the speed of your backup media.

2.6. Network Power Switches

You can fence GFS nodes with power switches and fencing agents available with Red Hat Cluster Suite. For more information about fencing with network power switches, refer to *Configuring and Managing a Red Hat Cluster*.

2.7. Console Access

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

2.8. Installing GFS

Installing GFS consists of installing Red Hat GFS RPMs on nodes in a Red Hat cluster. Before installing the RPMs, make sure of the following:

- The cluster nodes meet the system requirements described in this chapter.
- You have noted the key characteristics of your GFS configuration (refer to *Section 1.4, “Before Setting Up GFS”*).
- The correct Red Hat Cluster Suite software is installed in the cluster.

For information on installing RPMS for Red Hat Cluster Suite and Red Hat GFS, see *Configuring and Managing a Red Hat Cluster*. If you have already installed the appropriate Red Hat Cluster Suite RPMs, follow the procedures that pertain to installing the Red Hat GFS RPMs.

Getting Started

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

- *Section 3.1, “Prerequisite Tasks”*
- *Section 3.2, “Initial Setup Tasks”*

3.1. Prerequisite Tasks

Before setting up Red Hat GFS, make sure that you have noted the key characteristics of the GFS nodes (refer to *Section 1.4, “Before Setting Up GFS”*) and have loaded the GFS modules into each GFS node.Also, make sure that the clocks on the GFS nodes are synchronized. It is recommended that you use the Network Time Protocol (NTP) software provided with your Red Hat Enterprise Linux distribution. In addition, if you are using GNBD multipath, make sure that you understand GNBD multipath considerations. For information on GNBD multipath, see the document *Using GNBD with Global Filesystem*.

Note

The system clocks in GFS nodes must be within a few minutes of each other to prevent unnecessary inode time-stamp updating. Unnecessary inode time-stamp updating severely impacts cluster performance.

3.2. Initial Setup Tasks

Initial GFS setup consists of the following tasks:

1. Setting up logical volumes.
2. Making a GFS files system.
3. Mounting file systems.

Follow these steps to set up GFS initially.

1. Using CLVM (Cluster Logical Volume Manager), create a logical volume for each Red Hat GFS file system.

   

   Note

   You can use **`init.d`** scripts included with Red Hat Cluster Suite to automate activating and deactivating logical volumes. For more information about **`init.d`** scripts, refer to *Configuring and Managing a Red Hat Cluster*.

2. Create GFS file systems on logical volumes created in Step 1. Choose a unique name for each file system. For more information about creating a GFS file system, refer to *Section 4.1, “Making a File System”*.

   Command usage:

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

3. At each node, mount the GFS file systems. For more information about mounting a GFS file system, refer to *Section 4.2, "Mounting a File System"*.

   Command usage:

   mount -t gfs *BlockDevice MountPoint*

   mount -t gfs -o acl *BlockDevice MountPoint*

   The **-o aclmount** 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 **init.d** scripts included with Red Hat Cluster Suite to automate mounting and unmounting GFS file systems. For more information about **init.d** scripts, refer to *Configuring and Managing a Red Hat Cluster*.

Managing GFS

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

- *Section 4.1, “Making a File System”*
- *Section 4.2, “Mounting a File System”*
- *Section 4.3, “Unmounting a File System”*
- *Section 4.4, “GFS Quota Management”*
- *Section 4.5, “Growing a File System”*
- *Section 4.6, “Adding Journals to a File System”*
- *Section 4.7, “Direct I/O”*
- *Section 4.8, “Data Journaling”*
- *Section 4.9, “Configuring **atime** Updates”*
- *Section 4.10, “Suspending Activity on a File System”*
- *Section 4.11, “Displaying Extended GFS Information and Statistics”*
- *Section 4.12, “Repairing a File System”*
- *Section 4.13, “Context-Dependent Path Names”*

4.1. Making a File System

Once a cluster is set up and running, you can create a GFS file system with the **gfs_mkfs** command. A file system is created on an activated CLVM volume.The following information is required to run the **gfs_mkfs** command:

- Lock protocol/module name (for example, **lock_dlm**)
- Cluster name
- Number of journals (one journal required for each node that may be mounting the file system)

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 (for example, **lock_dlm**) to use.

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

- *ClusterName*, the name of the Red Hat cluster for which the GFS file system is being created.
- *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 than are needed can be specified at creation time to allow for future expansion.)

BlockDevice
Specifies a volume.

Examples

In this example, **lock_dlm** 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 **/dev/vg01/lvol0**.

```
gfs_mkfs -p lock_dlm -t alpha:gfs1 -j 8 /dev/vg01/lvol0
```

In this example, a second **lock_dlm** 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 **/dev/vg01/lvol1**.

```
gfs_mkfs -p lock_dlm -t alpha:gfs2 -j 8 /dev/vg01/lvol1
```

Complete Options

*Table 4.1, "Command Options: **gfs_mkfs**"* describes the **gfs_mkfs** command options (flags and parameters).

Table 4.1. Command Options: **gfs_mkfs**

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.
-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.

Flag	Parameter	Description
		Note: More journals than are needed can be specified at creation time to allow for future expansion.
-p	*LockProtoName*	Specifies the name of the locking protocol to use. Recognized cluster-locking protocols include: **lock_dlm** — The standard locking module. **lock_gulm** — The locking module compatible with earlier versions of GFS. **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 the name of the Red Hat cluster for which the GFS file system is being created. The cluster name is set in the **/etc/cluster/cluster.conf** file via the **Cluster Configuration Tool** and displayed at the **Cluster Status Tool** in the Red Hat Cluster Suite cluster management GUI. *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.

4.2. Mounting a File System

Before you can mount a GFS file system, the file system must exist (refer to *Section 4.1, "Making a File System"*), the volume where the file system exists must be activated, and the supporting clustering and locking systems must be started (refer to *Chapter 3, Getting Started* and *Configuring and Managing a Red Hat Cluster*. 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**).

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.

Example

In this example, the GFS file system on **/dev/vg01/lvol0** is mounted on the **/gfs1** directory.

```
mount -t gfs /dev/vg01/lvol0 /gfs1
```

Complete Usage

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

The **-o option** argument consists of GFS-specific options (refer to *Table 4.2, "GFS-Specific Mount Options"*) 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 4.2, "GFS-Specific Mount Options" describes the available GFS-specific **-o option** values that can be passed to GFS at mount time.

Table 4.2. GFS-Specific Mount Options

Option	Description
acl	Allows manipulating file ACLs. If a file system is mounted without the **acl** mount option, users are allowed to view ACLs (with **getfacl**), but are not allowed to set them (with **setfacl**).
hostdata=HostIDInfo	This field provides host (the computer on which the file system is being mounted) identity information to the lock module. The format and behavior of **HostIDInfo** depends on the lock module used. For **lock_gulm**, it overrides the **uname -n** network node name used as

Option	Description
	the default value by **lock_gulm**. This field is ignored by the **lock_dlm** and **lock_nolock** lock modules.
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**.
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.
oopses_ok	This option allows a GFS node to *not* panic when an oops occurs. (By default, a GFS node panics when an oops occurs, causing the file system used by that node to stall for other GFS nodes.) A GFS node *not* panicking when an oops occurs minimizes the failure on other GFS nodes using the file system that the failed node is using. There may be circumstances where you do not want to use this option — for example, when you need more detailed troubleshooting information. Use this option with care. Note: This option is turned on automatically if **lock_nolock** locking is specified; however, you can override it by using the **ignore_local_fs** option.
upgrade	Upgrade the on-disk format of the file system so that it can be used by newer versions of GFS.

4.3. Unmounting a File System

The GFS file system can be unmounted the same way as any Linux file system — by using the **umount** command.

Note

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

Usage

```
umount MountPoint
```

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

4.4. 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.

4.4.1. 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 of 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.

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.

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
```

4.4.2. 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.

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.

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.

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.

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
```

4.4.3. 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.

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.

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
```

4.4.4. 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.)

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

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.

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
```

4.4.5. Disabling/Enabling Quota Accounting

By default, quota accounting is enabled; therefore, GFS keeps track of disk usage for every user and group even when no quota limits have been set. Quota accounting incurs unnecessary overhead if quotas are not used. You can disable quota accounting completely 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.

Usage

```
fs_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

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.

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
```

4.5. 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 must be run on a mounted file system, but only needs to be run on one node in a 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, 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.

Usage

```
gfs_grow MountPoint
```

MountPoint
Specifies the GFS file system to which the actions apply.

Comments

Before running the **gfs_grow** command:

- Back up important data on the file system.
- Display the volume that is used by the file system to be expanded by running a **gfs_tool df *MountPoint*** command.
- Expand the underlying cluster volume with LVM. For information on administering LVM volumes, see the *LVM Administrator's Guide*

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

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
```

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 4.3, "GFS-specific Options Available While Expanding A File System" describes the GFS-specific options that can be used while expanding a GFS file system.

Table 4.3. GFS-specific Options Available While Expanding A File System

Option	Description
-h	Help. Displays a short usage message.
-q	Quiet. Turns 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	Displays command version information.

Option	Description
-v	Turns up the verbosity of messages.

4.6. 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 must be run on 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, 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.

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.

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 volume used by the file system where journals will be added.
- Expand the underlying cluster volume with LVM. For information on administering LVM volumes, see the *LVM Administrator's Guide*

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.

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 is checked for the new journals.

```
gfs_jadd -Tv /gfs1
```

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 4.4, "GFS-specific Options Available When Adding Journals" describes the GFS-specific options that can be used when adding journals to a GFS file system.

Table 4.4. GFS-specific Options Available When Adding Journals

Flag	Parameter	Description
-h		Help. Displays short usage message.
-J	*MegaBytes*	Specifies the size of the new journals in megabytes. Default journal size is 128 megabytes. The minimum size is 32 megabytes. 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		Displays command version information.
-v		Turns up the verbosity of messages.

4.7. 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 only by applications (such as databases) that manage their own caches.

An application invokes direct I/O by 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

4.7.1. 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.

4.7.2. GFS File Attribute

The **gfs_tool** command can be used to assign (set) a direct I/O attribute flag, **directio**, to a GFS file. The **directio** flag can also be cleared.

Usage

Setting the directio Flag

```
gfs_tool setflag directio File
```

Clearing the directio Flag

```
gfs_tool clearflag directio File
```

File
Specifies the file where the **directio** flag is assigned.

Example

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

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

4.7.3. GFS Directory Attribute

The **gfs_tool** command can be used to assign (set) 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.

Usage

Setting the inherit_directio flag

```
gfs_tool setflag inherit_directio Directory
```

Clearing the inherit_directio flag

```
gfs_tool clearflag inherit_directio Directory
```

Directory
Specifies the directory where the **inherit_directio** flag is set.

Example

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

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

4.8. 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.

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.

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
```

4.9. 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

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

4.9.1. Mount with `noatime`

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

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.

Example

In this example, the GFS file system resides on the **/dev/vg01/lvol0** and is mounted on directory **/gfs1** with atime updates turned off.

```
mount -t gfs /dev/vg01/lvol0 /gfs1 -o noatime
```

4.9.2. 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 GFS node 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 GFS nodes in a cluster be synchronized. If a node's system time is out of synchronization 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** 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.)

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.

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
```

4.10. Suspending Activity on a File System

You can suspend write activity to a file system by using the **gfs_tool freeze** command. Suspending write activity allows hardware-based device snapshots to be used to capture the file system in a consistent state. The **gfs_tool unfreeze** command ends the suspension.

Usage

Start Suspension

```
gfs_tool freeze MountPoint
```

End Suspension

```
gfs_tool unfreeze MountPoint
```

MountPoint
Specifies the file system.

Examples

This example suspends writes to file system **/gfs**.

```
gfs_tool freeze /gfs
```

This example ends suspension of writes to file system **/gfs**.

```
gfs_tool unfreeze /gfs
```

4.11. Displaying Extended GFS Information and Statistics

You can use the **gfs_tool** command to gather a variety of details about GFS. This section describes typical use of the **gfs_tool** command for displaying statistics, space usage, and extended status.

Usage

Displaying Statistics

```
gfs_tool counters MountPoint
```

The **counters** 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** 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** 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** flags, refer to the **gfs_tool** man page.

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
```

4.12. 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, you can recover the GFS file system 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

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 **Ctrl**+**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.

Usage

```
gfs_fsck -y BlockDevice
```

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

BlockDevice
Specifies the block device where the GFS file system resides.

Example

In this example, the GFS file system residing on block device **/dev/vg01/lvol0** is repaired. All queries to repair are automatically answered with **yes**.

```
gfs_fsck -y /dev/vg01/lvol0
```

4.13. 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.

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 4.5, "CDPN Variable Values"*) 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.

Table 4.5. CDPN *Variable* Values

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

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, an **@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
```

Appendix A. Upgrading GFS

To upgrade a node to Red Hat GFS 6.1 from earlier versions of Red Hat GFS, you must convert the GFS cluster configuration archive (CCA) to a Red Hat Cluster Suite cluster configuration system (CCS) configuration file (**/etc/cluster/cluster.conf**) and convert GFS **pool** volumes to LVM2 volumes.

This appendix contains instructions for upgrading from GFS 6.0 (or GFS 5.2.1) to Red Hat GFS 6.1, using GULM as the lock manager.

Note

You must retain GULM lock management for the upgrade to Red Hat GFS 6.1; that is, you cannot change from GULM lock management to DLM lock management during the upgrade to Red Hat GFS 6.1. However, after the upgrade to GFS 6.1, you can change lock managers.

The following procedure demonstrates upgrading to Red Hat GFS 6.1 from a GFS 6.0 (or GFS 5.2.1) configuration with an example **pool** configuration for a pool volume named **argus**.

```
poolname argus
subpools 1
subpool 0 512 1 gfs_data
pooldevice 0 0 /dev/sda1
```

1. Halt the GFS nodes and the lock server nodes as follows:
   a. Unmount GFS file systems from all nodes.
   b. Stop the lock servers; at each lock server node, stop the lock server as follows:

      ```
      # service lock_gulmd stop
      ```

   c. Stop **ccsd** at all nodes; at each node, stop **ccsd** as follows:

      ```
      # service ccsd stop
      ```

   d. Deactivate pools; at each node, deactivate GFS **pool** volumes as follows:

      ```
      # service pool stop
      ```

   e. Uninstall Red Hat GFS RPMs.
2. Install new software:
   a. Install Red Hat Enterprise Linux version 4 software (or verify that it is installed).
   b. Install Red Hat Cluster Suite and Red Hat GFS RPMs.
3. At *all* GFS 6.1 nodes, create a cluster configuration file directory (**/etc/cluster**) and upgrade the CCA (in this example, located in **/dev/pool/cca**) to the new Red Hat Cluster Suite CCS

configuration file format by running the **ccs_tool upgrade** command as shown in the following example:

```
# mkdir /etc/cluster
# ccs_tool upgrade /dev/pool/cca > /etc/cluster/cluster.conf
```

4. At *all* GFS 6.1 nodes, start **ccsd**, run the **lock_gulmd -c** command, and start **clvmd** as shown in the following example:

```
# ccsd
# lock_gulmd -c
Warning! You didn't specify a cluster name before --use_ccs
Letting ccsd choose which cluster we belong to.
# clvmd
```

Note

Ignore the warning message following the **lock_gulmd -c** command. Because the cluster name is already included in the converted configuration file, there is no need to specify a cluster name when issuing the **lock_gulmd -c** command.

5. At *all* GFS 6.1 nodes, run **vgscan** as shown in the following example:

```
# vgscan
  Reading all physical volumes.  This may take a while...
  Found volume group "argus" using metadata type pool
```

6. At *one* GFS 6.1 node, convert the **pool** volume to an LVM2 volume by running the **vgconvert** command as shown in the following example:

```
# vgconvert -M2 argus
  Volume group argus successfully converted
```

7. At *all* GFS 6.1 nodes, run **vgchange -ay** as shown in the following example:

```
# vgchange -ay
  1 logical volume(s) in volume group "argus" now active
```

8. At the first node to mount a GFS file system, run the **mount** command with the **upgrade** option as shown in the following example:

```
# mount -t gfs -o upgrade /dev/pool/argus /mnt/gfs1
```

Note

This step only needs to be done once — on the first mount of the GFS file system.

Note

If static minor numbers were used on **pool** volumes and the GFS 6.1 nodes are using LVM2 for other purposes (root file system) there may be problems activating the **pool** volumes under GFS 6.1. That is because of static minor conflicts. Refer to the following Bugzilla report for more information:

https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=146035

Appendix B. Revision History

Revision 1.1 **Wed Mar 17 2010** **Paul Kennedy** *pkennedy@redhat.com*

Resolves #570798

Clarifies the number of nodes supported.

Revision 1.0 **Wed Apr 01 2009**

Index

A

adding journals to a file system, 25
atime, configuring updates, 29
  mounting with noatime, 29
  tuning atime quantum, 30
audience, v

C

CDPN variable values table, 34
configuration, before, 5
configuration, initial, 11
  prerequisite tasks, 11
console access
  system requirements, 8

D

data journaling, 28
direct I/O, 26
  directory attribute, 27
  file attribute, 27
  O_DIRECT, 27
displaying extended GFS information and statistics, 31

F

feedback, viii, viii
fencing
  system requirements, 7
fibre channel network requirements table, 7
fibre channel storage device requirements table, 8
fibre channel storage devices
  system requirements, 8
fibre channel storage network
  system requirements, 7
file system
  adding journals, 25
  atime, configuring updates, 29
    mounting with noatime, 29
    tuning atime quantum, 30
  context-dependent path names (CDPNs), 33
  data journaling, 28
  direct I/O, 26
    directory attribute, 27
    file attribute, 27
    O_DIRECT, 27
  growing, 23
  making, 13
  mounting, 15
  quota management, 18
    disabling/enabling quota accounting, 22
    disabling/enabling quota enforcement, 21
    displaying quota limits, 19
    setting quotas, 18
    synchronizing quotas, 20
  repairing, 32
  suspending activity, 31
  unmounting, 17

G

GFS
  atime, configuring updates, 29
    mounting with noatime, 29
    tuning atime quantum, 30
  direct I/O, 26
    directory attribute, 27
    file attribute, 27
    O_DIRECT, 27
  displaying extended information and statistics, 31
  managing, 13
  quota management, 18
    disabling/enabling quota accounting, 22
    disabling/enabling quota enforcement, 21
    displaying quota limits, 19
    setting quotas, 18
    synchronizing quotas, 20
GFS functions, 3
GFS software subsystem components table, 4
GFS software subsystems, 4
GFS-specific options for adding journals table, 26
GFS-specific options for expanding file systems table, 24
gfs_mkfs command options table, 14
growing a file system, 23

I

initial tasks
  setup, initial, 11
introduction, v
  audience, v
  references, ix

M

making a file system, 13
managing GFS, 13
mount table, 16
mounting a file system, 15

N

network power switches
  system requirements, 8

O

overview, 1
  configuration, before, 5
  economy, 1
  GFS functions, 3
  GFS software subsystems, 4
  performance, 1
  scalability, 1

P

path names, context-dependent (CDPNs), 33
platform
  system requirements, 7
platform requirements table, 7
preface (see introduction)
prerequisite tasks
  configuration, initial, 11

Q

quota management, 18
  disabling/enabling quota accounting, 22
  disabling/enabling quota enforcement, 21
  displaying quota limits, 19
  setting quotas, 18
  synchronizing quotas, 20

R

recommended references table, ix
Red Hat Cluster Suite
  system requirements, 7
references, recommended, ix
repairing a file system, 32

S

setup, initial
  initial tasks, 11
suspending activity on a file system, 31
system requirements, 7
  console access, 8
  fencing, 7
  fibre channel storage devices, 8
  fibre channel storage network, 7
  network power switches, 8
  platform, 7
  Red Hat Cluster Suite, 7

T

tables
  CDPN variable values, 34
  fibre channel network requirements, 7
  fibre channel storage device requirements, 8
  GFS software subsystem components, 4
  GFS-specific options for adding journals, 26
  GFS-specific options for expanding file systems, 24
  gfs_mkfs command options, 14
  mount options, 16
  platform requirements, 7
  recommended references, ix

U

unmounting a file system, 17