Red Hat Enterprise Linux 6

Global File System 2

Red Hat Global File System 2

Red Hat Enterprise Linux 6 Global File System 2
Red Hat Global File System 2
Edition 7

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 configuring and maintaining Red Hat GFS2 (Red Hat Global File System 2) for Red Hat Enterprise Linux 6.

Introduction

This book provides information about configuring and maintaining Red Hat GFS2 (Red Hat Global File System 2), which is included in the Resilient Storage Add-On.

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:

- *Installation Guide* — Documents relevant information regarding the installation of Red Hat Enterprise Linux 6.
- *Deployment Guide* — Documents relevant information regarding the deployment, configuration and administration of Red Hat Enterprise Linux 6.
- *Storage Administration Guide* — Provides instructions on how to effectively manage storage devices and file systems on Red Hat Enterprise Linux 6.

For more information about the High-Availabilty Add-On and the Resilient Storage Add-On for Red Hat Enterprise Linux 6, refer to the following resources:

- *Red Hat Cluster Suite Overview* — Provides a high level overview of the High Availability Add-On, Resilient Storage Add-On, and the Load Balancer Add-On.
- *Configuring and Managing a Red Hat Cluster* — Provides information about installing, configuring and managing the High Availability Add-On.
- *Logical Volume Manager Administration* — Provides a description of the Logical Volume Manager (LVM), including information on running LVM in a clustered environment.
- *DM 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 Red Hat Load Balancer Add-On (Formerly known as Linux Virtual Server [LVS]),
- *Release Notes* — Provides information about the current release of Red Hat products.

High Availability Add-On documentation and other Red Hat documents are available in HTML, PDF, and RPM versions on the Red Hat Enterprise Linux Documentation CD and online at *http://www.redhat.com/docs/*.

3. We Need Feedback!

If you find a typographical error in this manual, 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/*

against the product **Red Hat Enterprise Linux 6** and the component **doc-Global_File_System_2**. When submitting a bug report, be sure to mention the manual's identifier: **rh-gfs2(EN)-6 (2010-10-14T15:15)**.

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

4. 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.

4.1. 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.

Proportional Bold

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

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.

4.2. 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"));
   }
}
```

4.3. 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.

GFS2 Overview

The Red Hat GFS2 file system is included in the Resilient Storage Add-On. It is a native file system that interfaces directly with the Linux kernel file system interface (VFS layer). When implemented as a cluster file system, GFS2 employs distributed metadata and multiple journals. Red Hat supports the use of GFS2 file systems only as implemented in the High Availability Add-On.

Note

Although a GFS2 file system can be implemented in a standalone system or as part of a cluster configuration, for the Red Hat Enterprise Linux 6 release Red Hat does not support the use of GFS2 as a single-node file system. Red Hat does support a number of high-performance single node file systems which are optimized for single node and thus have generally lower overhead than a cluster file system. Red Hat recommends using these file systems in preference to GFS2 in cases where only a single node needs to mount the file system.

Red Hat will continue to support single-node GFS2 file systems for mounting snapshots of cluster file systems (for example, for backup purposes).

Note

Red Hat does not support using GFS2 for cluster file system deployments greater than 16 nodes.

GFS2 is based on a 64-bit architecture, which can theoretically accommodate an 8 EB file system. However, the current supported maximum size of a GFS2 file system is 25 TB. If your system requires GFS2 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.gfs2** command on a very large file system can take a long time and consume a large amount of memory. Additionally, in the event of a disk or disk-subsystem failure, recovery time is limited by the speed of your backup media. For information on the amount of memory the **fsck.gfs2** command requires, see *Section 3.11, "Repairing a File System"*.

When configured in a cluster, Red Hat GFS2 nodes can be configured and managed with High Availability Add-On configuration and management tools. Red Hat GFS2 then provides data sharing among GFS2 nodes in a cluster, with a single, consistent view of the file system name space across the GFS2 nodes. This allows processes on different nodes to share GFS2 files in the same way that processes on the same node can share files on a local file system, with no discernible difference. For information about the High Availability Add-On refer to *Configuring and Managing a Red Hat Cluster*.

While a GFS2 file system may be used outside of LVM, Red Hat supports only GFS2 file systems that are created on a CLVM logical volume. CLVM is included in the Resilient Storage Add-On. It is a cluster-wide implementation of LVM, enabled by the CLVM daemon **clvmd**, which manages LVM logical volumes in a 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. For information on the LVM volume manager, see *Logical Volume Manager Administration*

The **gfs2.ko** kernel module implements the GFS2 file system and is loaded on GFS2 cluster nodes.

Note

When you configure a GFS2 file system as a cluster file system, you must ensure that all nodes in the cluster have access to the shared storage. Asymmetric cluster configurations in which some nodes have access to the shared storage and others do not are not supported. This does not require that all nodes actually mount the GFS2 file system itself.

This chapter provides some basic, abbreviated information as background to help you understand GFS2. It contains the following sections:

- *Section 1.1, "New and Changed Features"*
- *Section 1.2, "Before Setting Up GFS2"*
- *Section 1.3, "Differences between GFS and GFS2"*
- *Section 1.4, "GFS2 Node Locking"*

1.1. New and Changed Features

This section lists new and changed features of the GFS2 file system and the GFS2 documentation that are included with the initial release of Red Hat Enterprise Linux 6.

- For the Red Hat Enterprise Linux 6 release, Red Hat does not support the use of GFS2 as a single-node file system.
- For the Red Hat Enterprise Linux 6 release, the **gfs2_convert** command to upgrade from a GFS to a GFS2 file system has been enhanced. For information on this command, see *Appendix A, Converting a File System from GFS to GFS2*.
- The Red Hat Enterprise Linux 6 release supports the **discard**, **nodiscard**, **barrier**, **nobarrier**, **quota_quantum**, **statfs_quantum**, and **statfs_percent** mount options. For information about mounting a GFS2 file system, see *Section 3.2, "Mounting a File System"*.
- The Red Hat Enterprise Linux 6 version of this document contains a new section, *Section 1.4, "GFS2 Node Locking"*. This section describes some of the internals of GFS2 file systems.

1.2. Before Setting Up GFS2

Before you install and set up GFS2, note the following key characteristics of your GFS2 file systems:

GFS2 nodes
: Determine which nodes in the cluster will mount the GFS2 file systems.

Number of file systems
: Determine how many GFS2 file systems to create initially. (More file systems can be added later.)

File system name
: Determine a unique name for each file system. The name must be unique for all **lock_dlm** file systems over the cluster. Each file system name is required in the form of a parameter variable. For example, this book uses file system names **mydata1** and **mydata2** in some example procedures.

Journals
: Determine the number of journals for your GFS2 file systems. One journal is required for each node that mounts a GFS2 file system. GFS2 allows you to add journals dynamically at a later point as additional servers mount a file system. For information on adding journals to a GFS2 file system, see *Section 3.7, "Adding Journals to a File System"*.

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

Note

You may see performance problems with GFS2 when many create and delete operations are issued from more than one node in the same directory at the same time. If this causes performance problems in your system, you should localize file creation and deletions by a node to directories specific to that node as much as possible.

1.3. Differences between GFS and GFS2

This section lists the improvements and changes that GFS2 offers over GFS.

Migrating from GFS to GFS2 requires that you convert your GFS file systems to GFS2 with the **gfs2_convert** utility. For information on the **gfs2_convert** utility, see *Appendix A, Converting a File System from GFS to GFS2*.

1.3.1. GFS2 Command Names

In general, the functionality of GFS2 is identical to GFS. The names of the file system commands, however, specify GFS2 instead of GFS. *Table 1.1, "GFS and GFS2 Commands"* shows the equivalent GFS and GFS2 commands.

Table 1.1. GFS and GFS2 Commands

GFS Command	GFS2 Command	Description
mount	**mount**	Mount a file system. The system can determine whether the file system is a GFS or GFS2 file system type. For information on the GFS2 mount options see the gfs2_mount(8) man page.
umount	**umount**	Unmount a file system.
fsck **gfs_fsck**	**fsck** **fsck.gfs2**	Check and repair an unmounted file system.
gfs_grow	**gfs2_grow**	Grow a mounted file system.
gfs_jadd	**gfs2_jadd**	Add a journal to a mounted file system
gfs_mkfs **mkfs -t gfs**	**mkfs.gfs2** **mkfs -t gfs2**	Create a file system on a storage device.
gfs_quota	**gfs2_quota**	Manage quotas on a mounted file system.
gfs_tool	**gfs2_tool**	Configure, tune, or gather information about a file system.
gfs_edit	**gfs2_edit**	Display, print, or edit file system internal structures. The **gfs2_edit** command can be used for GFS file systems as well as GFS2 file system.

GFS Command	GFS2 Command	Description
gfs_tool setflag jdata/ inherit_jdata	**chattr +j** (preferred)	Enable journaling on a file or directory.
setfacl/ getfacl	**setfacl/ getfacl**	Set or get file access control list for a file or directory.
setfattr/ getfattr	**setfattr/ getfattr**	Set or get the extended attributes of a file.

For a full listing of the supported options for the GFS2 file system commands, see the man pages for those commands.

1.3.2. Additional Differences Between GFS and GFS2

This section summarizes the additional differences in GFS and GFS2 administration that are not described in *Section 1.3.1, "GFS2 Command Names"*.

Context-Dependent Path Names

GFS2 file systems do not provide support for context-dependent path names, which allow you to create symbolic links that point to variable destination files or directories. For this functionality in GFS2, you can use the **bind** option of the **mount** command. For information on bind mounts and context-dependent pathnames in GFS2, see *Section 3.12, "Bind Mounts and Context-Dependent Path Names"*.

gfs2.ko Module

The kernel module that implements the GFS file system is **gfs.ko**. The kernel module that implements the GFS2 file system is **gfs2.ko**.

Enabling Quota Enforcement in GFS2

In GFS2 file systems, quota enforcement is disabled by default and must be explicitly enabled. To enable and disable quotas for GFS2 file systems, you use the **quota=on|off|account** option for the **mount** command. For information on enabling and disabling quota enforcement, see *Section 3.5.4, "Enabling/Disabling Quota Enforcement"*.

Data Journaling

GFS2 file systems support the use of the **chattr** command to set and clear the **j** flag on a file or directory. Setting the **+j** flag on a file enables data journaling on that file. Setting the **+j** flag on a directory means "inherit jdata", which indicates that all files and directories subsequently created in that directory are journaled. Using the **chattr** command is the preferred way to enable and disable data journaling on a file.

Adding Journals Dynamically

In GFS file systems, journals are embedded metadata that exists outside of the file system, making it necessary to extend the size of the logical volume that contains the file system before adding journals. In GFS2 file systems, journals are plain (though hidden) files. This means that for GFS2 file systems, journals can be dynamically added as additional servers mount a file system, as long as

space remains on the file system for the additional journals. For information on adding journals to a GFS2 file system, see *Section 3.7, "Adding Journals to a File System"*.

atime_quantum parameter removed

The GFS2 file system does not support the **atime_quantum** tunable parameter, which can be used by the GFS file system to specify how often **atime** updates occur. In its place GFS2 supports the **relatime** and **noatime** mount options. The **relatime** mount option is recommended to achieve similar behavior to setting the **atime_quantum** parameter in GFS.

The data= option of the mount command

When mounting GFS2 file systems, you can specify the **data=ordered** or **data=writeback** option of the **mount**. When **data=ordered** is set, the user data modified by a transaction is flushed to the disk before the transaction is committed to disk. This should prevent the user from seeing uninitialized blocks in a file after a crash. When **data=writeback** is set, the user data is written to the disk at any time after it is dirtied. This does not provide the same consistency guarantee as **ordered** mode, but it should be slightly faster for some workloads. The default is **ordered** mode.

The gfs2_tool command

The **gfs2_tool** command supports a different set of options for GFS2 than the **gfs_tool** command supports for GFS:

- The **gfs2_tool** command supports a **journals** parameter that prints out information about the currently configured journals, including how many journals the file system contains.
- The **gfs2_tool** command does not support the **counters** flag, which the **gfs_tool** command uses to display GFS statistics.
- The **gfs2_tool** command does not support the **inherit_jdata** flag. To flag a directory as "inherit jdata", you can set the **jdata** flag on the directory or you can use the **chattr** command to set the **+j** flag on the directory. Using the **chattr** command is the preferred way to enable and disable data journaling on a file.

The gfs2_edit command

The **gfs2_edit** command supports a different set of options for GFS2 than the **gfs_edit** command supports for GFS. For information on the specific options each version of the command supports, see the **gfs2_edit** and **gfs_edit** man pages.

1.3.3. GFS2 Performance Improvements

There are many features of GFS2 file systems that do not result in a difference in the user interface from GFS file systems but which improve file system performance.

A GFS2 file system provides improved file system performance in the following ways:

- Better performance for heavy usage in a single directory.
- Faster synchronous I/O operations
- Faster cached reads (no locking overhead)
- Faster direct I/O with preallocated files (provided I/O size is reasonably large, such as 4M blocks)
- Faster I/O operations in general

- Execution of the **df** command is much faster, because of faster **statfs** calls.
- The **atime** mode has been improved to reduce the number of write I/O operations generated by **atime** when compared with GFS.

GFS2 file systems provide broader and more mainstream support in the following ways.

- GFS2 is part of the upstream kernel (integrated into 2.6.19).
- GFS2 supports the following features:
  - SELinux extended attributes.
  - the **lsattr**() and **chattr**() attribute settings via standard **ioctl**() calls.
  - nanosecond timestamps

A GFS2 file system provides the following improvements to the internal efficiency of the file system.

- GFS2 uses less kernel memory
- GFS2 requires no metadata generation numbers.

  Allocating GFS2 metadata does not require reads. Copies of metadata blocks in multiple journals are managed by revoking blocks from the journal before lock release.
- GFS2 includes a much simpler log manager that knows nothing about unlinked inodes or quota changes.
- The **gfs2_grow** and **gfs2_jadd** commands use locking to prevent multiple instances running at the same time.
- The ACL code has been simplified for calls like **creat**() and **mkdir**().
- Unlinked inodes, quota changes, and **statfs** changes are recovered without remounting the journal.

1.4. GFS2 Node Locking

In order to get the best performance from a GFS2 file system, it is very important to understand some of the basic theory of its operation. A single node file system is implemented alongside a cache, the purpose of which is to eliminate latency of disk accesses when using frequently requested data. In Linux the page cache (and historically the buffer cache) provide this caching function.

With GFS2, each node has its own page cache which may contain some portion of the on-disk data. GFS2 uses a locking mechanism called *glocks* (pronounced gee-locks) to maintain the integrity of the cache between nodes. The glock subsystem provides a cache management function which is implemented using the *distributed lock manager* (DLM) as the underlying communication layer.

The glocks provide protection for the cache on a per-inode basis, so there is one lock per inode which is used for controlling the caching layer. If that glock is granted in shared mode (DLM lock mode: PR) then the data under that glock may be cached upon one or more nodes at the same time, so that all the nodes may have local access to the data.

If the glock is granted in exclusive mode (DLM lock mode: EX) then only a single node may cache the data under that glock. This mode is used by all operations which modify the data (such as the **write** system call).

If another node requests a glock which cannot be granted immediately, then the DLM sends a message to the node or nodes which currently hold the glocks blocking the new request to ask them to drop their locks. Dropping glocks can be (by the standards of most file system operations) a long process. Dropping a shared glock requires only that the cache be invalidated, which is relatively quick and proportional to the amount of cached data.

Dropping an exclusive glock requires a log flush, and writing back any changed data to disk, followed by the invalidation as per the shared glock.

The different between a single node file system and GFS2 then, is that a single node file system has a single cache and GFS2 has a separate cache on each node. In both cases, latency to access to cached data is of a similar order of magnitude, but the latency to access uncached data is much greater in GFS2 if another node has previously cached that same data.

Note

Due to the way in which GFS2's caching is implemented the best performance is obtained when either of the following takes place:

- An inode is used in a read only fashion across all nodes
- An inode is written or modified from a single node only.

Note that inserting and removing entries from a directory during file creation and deletion counts as writing to the directory inode.

It is possible to break this rule provided that it is broken relatively infrequently. Ignoring this rule too often will result in a severe performance penalty.

If you **mmap**() a file on GFS2 with a read/write mapping, but only read from it, this only counts as a read. On GFS though, it counts as a write, so GFS2 is much more scalable with **mmap**() I/O.

If you do not set the **noatime** **mount** parameter, then reads will also result in writes to update the file timestamps. We recommend that all GFS2 users should mount with **noatime** unless they have a specific requirement for **atime**.

1.4.1. Performance Tuning With GFS2

It is usually possible to alter the way in which a troublesome application stores its data in order to gain a considerable performance advantage.

A typical example of a troublesome application is an email server. These are often laid out with a spool directory containing files for each user (**mbox**), or with a directory for each user containing a file for each message (**maildir**). When requests arrive over IMAP, the ideal arrangement is to give each user an affinity to a particular node. That way their requests to view and delete email messages will tend to be served from the cache on that one node. Obviously if that node fails, then the session can be restarted on a different node.

When mail arrives via SMTP, then again the individual nodes can be set up so as to pass a certain user's mail to a particular node by default. If the default node is not up, then the message can be saved directly into the user's mail spool by the receiving node. Again this design is intended to keep particular sets of files cached on just one node in the normal case, but to allow direct access in the case of node failure.

This setup allows the best use of GFS2's page cache and also makes failures transparent to the application, whether **imap** or **smtp**.

Backup is often another tricky area. Again, if it is possible it is greatly preferable to back up the working set of each node directly from the node which is caching that particular set of inodes. If you have a backup script which runs at a regular point in time, and that seems to coincide with a spike in the response time of an application running on GFS2, then there is a good chance that the cluster may not be making the most efficient use of the page cache.

Obviously, if you are in the (enviable) position of being able to stop the application in order to perform a backup, then this won't be a problem. On the other hand, if a backup is run from just one node, then after it has completed a large portion of the file system will be cached on that node, with a performance penalty for subsequent accesses from other nodes. This can be mitigated to a certain extent by dropping the VFS page cache on the backup node after the backup has completed with following command:

```
echo -n 3 >/proc/sys/vm/drop_caches
```

However this is not as good a solution as taking care to ensure the working set on each node is either shared, mostly read only across the cluster, or accessed largely from a single node.

1.4.2. Troubleshooting GFS2 Performance with the GFS2 Lock Dump

If your cluster performance is suffering because of inefficient use of GFS2 caching, you may see large and increasing I/O wait times. You can make use of GFS2's lock dump information to determine the cause of the problem.

The GFS2 lock dump information can be gathered from the **debugfs** file which can be found at the following path name, assuming that **debugfs** is mounted on **/sys/kernel/debug/**:

```
/sys/kernel/debug/gfs2/fsname/glocks
```

The content of the file is a series of lines. Each line starting with G: represents one glock, and the following lines, indented by a single space, represent an item of information relating to the glock immediately before them in the file.

The best way to use the **debugfs** file is to use the **cat** command to take a copy of the complete content of the file (it might take a long time if you have a large amount of RAM and a lot of cached inodes) while the application is experiencing problems, and then looking through the resulting data at a later date.

Tip

It can be useful to make two copies of the **debugfs** file, one a few seconds or even a minute or two after the other. By comparing the holder information in the two traces relating to the same glock number, you can tell whether the workload is making progress (that is, it is just slow) or whether it has become stuck (which is always a bug and should be reported to Red Hat support immediately).

Lines in the **debugfs** file starting with H: (holders) represent lock requests either granted or waiting to be granted. The flags field on the holders line f: shows which: The 'W' flag refers to a waiting request, the 'H' flag refers to a granted request. The glocks which have large numbers of waiting requests are likely to be those which are experiencing particular contention.

Table 1.2, "Glock flags" shows the meanings of the different glock flags and *Table 1.3, "Glock holder flags"* shows the meanings of the different glock holder flags.

Table 1.2. Glock flags

Flag	Name	Meaning
d	Pending demote	A deferred (remote) demote request
D	Demote	A demote request (local or remote)
f	Log flush	The log needs to be committed before releasing this glock
F	Frozen	Replies from remote nodes ignored - recovery is in progress.
i	Invalidate in progress	In the process of invalidating pages under this glock
I	Initial	Set when DLM lock is associated with this glock
l	Locked	The glock is in the process of changing state
p	Demote in progress	The glock is in the process of responding to a demote request
r	Reply pending	Reply received from remote node is awaiting processing
y	Dirty	Data needs flushing to disk before releasing this glock

Table 1.3. Glock holder flags

Flag	Name	Meaning
a	Async	Do not wait for glock result (will poll for result later)
A	Any	Any compatible lock mode is acceptable
c	No cache	When unlocked, demote DLM lock immediately
e	No expire	Ignore subsequent lock cancel requests
E	exact	Must have exact lock mode
F	First	Set when holder is the first to be granted for this lock
H	Holder	Indicates that requested lock is granted
p	Priority	Enqueue holder at the head of the queue
t	Try	A "try" lock
T	Try 1CB	A "try" lock that sends a callback
W	Wait	Set while waiting for request to complete

Having identified a glock which is causing a problem, the next step is to find out which inode it relates to. The glock number (n: on the G: line) indicates this. It is of the form *`type`*/*`number`* and if *`type`* is 2, then the glock is an inode glock and the *`number`* is an inode number. To track down the inode, you can then run **`find -inum number`** where *`number`* is the inode number converted from the hex format in the glocks file into decimal.

Warning

If you run the **`find`** on a file system when it is experiencing lock contention, you are likely to make the problem worse. It is a good idea to stop the application before running the **`find`** when you are looking for contended inodes.

Table 1.4, "Glock types" shows the meanings of the different glock types.

Table 1.4. Glock types

Type number	Lock type	Use
1	Trans	Transaction lock
2	Inode	Inode metadata and data
3	Rgrp	Resource group metadata
4	Meta	The superblock
5	Iopen	Inode last closer detection
6	Flock	**`flock`**(2) syscall
8	Quota	Quota operations
9	Journal	Journal mutex

If the glock that was identified was of a different type, then it is most likely to be of type 3: (resource group). If you see significant numbers of processes waiting for other types of glock under normal loads, then please report this to Red Hat support.

If you do see a number of waiting requests queued on a resource group lock there may be a number of reason for this. One is that there are a large number of nodes compared to the number of resource groups in the file system. Another is that the file system may be very nearly full (requiring, on average, longer searches for free blocks). The situation in both cases can be improved by adding more storage and using the **`gfs2_grow`** command to expand the file system.

Getting Started

This chapter describes procedures for initial setup of GFS2 and contains the following sections:

- *Section 2.1, “Prerequisite Tasks”*
- *Section 2.2, “Initial Setup Tasks”*
- *Section 2.3, “Deploying a GFS2 Cluster”*

2.1. Prerequisite Tasks

Before setting up Red Hat GFS2, make sure that you have noted the key characteristics of the GFS2 nodes (refer to *Section 1.2, “Before Setting Up GFS2”*). Also, make sure that the clocks on the GFS2 nodes are synchronized. It is recommended that you use the Network Time Protocol (NTP) software provided with your Red Hat Enterprise Linux distribution.

Note

The system clocks in GFS2 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.

2.2. Initial Setup Tasks

Initial GFS2 setup consists of the following tasks:

1. Setting up logical volumes.
2. Making a GFS2 files system.
3. Mounting file systems.

Follow these steps to set up GFS2 initially.

1. Using LVM, create a logical volume for each Red Hat GFS2 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 GFS2 file systems on logical volumes created in Step 1. Choose a unique name for each file system. For more information about creating a GFS2 file system, refer to *Section 3.1, “Making a File System”*.

   You can use either of the following formats to create a clustered GFS2 file system:

   ```
   mkfs.gfs2 -p lock_dlm -t ClusterName:FSName -j NumberJournals BlockDevice
   ```

```
mkfs -t gfs2 -p lock_dlm -t LockTableName -j NumberJournals BlockDevice
```

For more information on creating a GFS2 file system, see *Section 3.1, "Making a File System"*.

3. At each node, mount the GFS2 file systems. For more information about mounting a GFS2 file system, see *Section 3.2, "Mounting a File System"*.

   Command usage:

   mount *BlockDevice MountPoint*

   mount -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 **init.d** scripts included with Red Hat Cluster Suite to automate mounting and unmounting GFS2 file systems. For more information about **init.d** scripts, refer to *Configuring and Managing a Red Hat Cluster*.

2.3. Deploying a GFS2 Cluster

Deploying a cluster filesystem is not a "drop in" replacement for a single node deployment. We recommend that you allow a period of around 8-12 weeks of testing on new installations in order to test the system and ensure that it is working at the required performance level. During this period any performance or functional issues can be worked out and any queries should be directed to the Red Hat support team. We also recommend that customers considering deploying clusters have their configurations reviewed by Red Hat support before deployment to avoid any possible support issues later on.

Managing GFS2

This chapter describes the tasks and commands for managing GFS2 and consists of the following sections:

3.1. Making a File System

You create a GFS2 file system with the **mkfs.gfs2** command. You can also use the **mkfs** command with the **-t gfs2** option specified. A file system is created on an activated LVM volume. The following information is required to run the **mkfs.gfs2** command:

- Lock protocol/module name (the lock protocol for a cluster is **lock_dlm**)
- Cluster name (when running as part of a cluster configuration)
- Number of journals (one journal required for each node that may be mounting the file system)

When creating a GFS2 file system, you can use the **mkfs.gfs2** command directly, or you can use the **mkfs** command with the *-t* parameter specifying a filesystem of type **gfs2**, followed by the gfs2 file system options.

Note

Once you have created a GFS2 file system with the **mkfs.gfs2** command, you cannot decrease the size of the file system. You can, however, increase the size of an existing file system with the **gfs2_grow** command, as described in *Section 3.6, "Growing a File System"*.

Usage

When creating a clustered GFS2 filesystem, you can use either of the following formats:

```
mkfs.gfs2 -p LockProtoName -t LockTableName -j NumberJournals BlockDevice
```

```
mkfs -t gfs2 -p LockProtoName -t LockTableName -j NumberJournals BlockDevice
```

When creating a local GFS2 filesystem, you can use either of the following formats:

Note

For the Red Hat Enterprise Linux 6 release, Red Hat does not support the use of GFS2 as a single-node file system.

```
mkfs.gfs2 -p LockProtoName -j NumberJournals BlockDevice
```

```
mkfs -t gfs2 -p LockProtoName -j NumberJournals 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 to use. The lock protocol for a cluster is **lock_dlm**.

LockTableName
: This parameter is specified for GFS2 filesystem in a cluster configuration. It has two parts separated by a colon (no spaces) as follows: *ClusterName:FSName*

- *ClusterName*, the name of the cluster for which the GFS2 file system is being created.
- *FSName*, the file system name, can be 1 to 16 characters long. The name must be unique for all **lock_dlm** filesystems over the cluster, and for all filesystems (**lock_dlm** and **lock_nolock**) on each local node.

Number
: Specifies the number of journals to be created by the **mkfs.gfs2** command. One journal is required for each node that mounts the file system. For GFS2 file systems, more journals can be added later without growing the filesystem, as described in *Section 3.7, "Adding Journals to a File System"*.

BlockDevice
: Specifies a logical or physical volume.

Examples

In these example, **lock_dlm** is the locking protocol that the file system uses, since this is a clustered file system. The cluster name is **alpha**, and the file system name is **mydata1**. The file system contains eight journals and is created on **/dev/vg01/lvol0**.

```
mkfs.gfs2 -p lock_dlm -t alpha:mydata1 -j 8 /dev/vg01/lvol0
```

```
mkfs -t gfs2 -p lock_dlm -t alpha:mydata1 -j 8 /dev/vg01/lvol0
```

In these examples, a second **lock_dlm** file system is made, which can be used in cluster **alpha**. The file system name is **mydata2**. The file system contains eight journals and is created on **/dev/vg01/lvol1**.

```
mkfs.gfs2 -p lock_dlm -t alpha:mydata2 -j 8 /dev/vg01/lvol1
```

```
mkfs -t gfs2 -p lock_dlm -t alpha:mydata2 -j 8 /dev/vg01/lvol1
```

Complete Options

*Table 3.1, "Command Options: **mkfs.gfs2**"* describes the **mkfs.gfs2** command options (flags and parameters).

Table 3.1. Command Options: **mkfs.gfs2**

Flag	Parameter	Description
-c	*Megabytes*	Sets the initial size of each journal's quota change file to *Megabytes*.
-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 8 megabytes. Larger journals improve performance, although they use more memory than smaller journals.
-j	*Number*	Specifies the number of journals to be created by the **mkfs.gfs2** command. One journal is required for each node that mounts the file system. If this option is not specified, one journal will be created. For GFS2 file systems, you can add additional journals at a later time without growing the file system.
-O		Prevents the **mkfs.gfs2** command from asking for confirmation before writing the file system.
-p	*LockProtoName*	Specifies the name of the locking protocol to use. Recognized locking protocols include: **lock_dlm** — The standard locking module, required for a clustered file system. **lock_nolock** — Used when GFS2 is acting as a local file system (one node only).

Flag	Parameter	Description
-q		Quiet. Do not display anything.
-r	*MegaBytes*	Specifies the size of the resource groups in megabytes. The minimum resource group size is 32 MB. The maximum resource group size is 2048 MB. A large resource group size may increase performance on very large file systems. If this is not specified, mkfs.gfs2 chooses the resource group size based on the size of the file system: average size file systems will have 256 MB resource groups, and bigger file systems will have bigger RGs for better performance.
-t	*LockTableName*	A unique identifier that specifies the lock table field when you use the **lock_dlm** protocol; the **lock_nolock** protocol does not use this parameter. This parameter has two parts separated by a colon (no spaces) as follows: *ClusterName:FSName*. *ClusterName* is the name of the cluster for which the GFS2 file system is being created; only members of this cluster are permitted to use this file system. 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.
-u	*MegaBytes*	Specifies the initial size of each journal's unlinked tag file.
-V		Displays command version information.

3.2. Mounting a File System

Before you can mount a GFS2 file system, the file system must exist (refer to *Section 3.1, "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 *Configuring and Managing a Red Hat Cluster*). After those requirements have been met, you can mount the GFS2 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 BlockDevice MountPoint
```

Mounting With ACL Manipulation

```
```

```
mount -o acl BlockDevice MountPoint
```

-o acl
GFS2-specific option to allow manipulating file ACLs.

BlockDevice
Specifies the block device where the GFS2 file system resides.

MountPoint
Specifies the directory where the GFS2 file system should be mounted.

Example

In this example, the GFS2 file system on **/dev/vg01/lvol0** is mounted on the **/mygfs2** directory.

```
mount /dev/vg01/lvol0 /mygfs2
```

Complete Usage

```
mount BlockDevice MountPoint -o option
```

The **-o option** argument consists of GFS2-specific options (refer to *Table 3.2, "GFS2-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 GFS2-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 3.2, "GFS2-Specific Mount Options" describes the available GFS2-specific **-o option** values that can be passed to GFS2 at mount time.

Note

This table includes descriptions of options that are used with local file systems only. Note, however, that for the Red Hat Enterprise Linux 6 release, Red Hat does not support the use of GFS2 as a single-node file system. Red Hat will continue to support single-node GFS2 file systems for mounting snapshots of cluster file systems (for example, for backup purposes).

Table 3.2. GFS2-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**).

Option	Description
data=[ordered|writeback]	When **data=ordered** is set, the user data modified by a transaction is flushed to the disk before the transaction is committed to disk. This should prevent the user from seeing uninitialized blocks in a file after a crash. When **data=writeback** mode is set, the user data is written to the disk at any time after it is dirtied; this does not provide the same consistency guarantee as **ordered** mode, but it should be slightly faster for some workloads. The default value is **ordered** mode.
ignore_local_fs **Caution:** This option should *not* be used when GFS2 file systems are shared.	Forces GFS2 to treat the file system as a multihost file system. By default, using **lock_nolock** automatically turns on the **localflocks** flag.
localflocks **Caution:** This option should not be used when GFS2 file systems are shared.	Tells GFS2 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.
quota=[off/account/on]	Turns quotas on or off for a file system. Setting the quotas to be in the **account** state causes the per UID/GID usage statistics to be correctly maintained by the file system; limit and warn values are ignored. The default value is **off**.
errors=panic|withdraw	When **errors=panic** is specified, file system errors will cause a kernel panic. The default behavior, which is the same as specifying **errors=withdraw**, is for the system to withdraw from the file system and make it inaccessible until the next reboot; in some cases the system may remain running. For information on the GFS2 withdraw function, see *Section 3.14, "The GFS2 Withdraw Function"*.
discard/nodiscard	Causes GFS2 to generate "discard" I/O requests for blocks that have been freed. These can be used by suitable hardware to implement thin provisioning and similar schemes.
barrier/nobarrier	Causes GFS2 to send I/O barriers when flushing the journal. The default value is **on**. This option is automatically turned **off** if the underlying device does not support I/O barriers. Use of I/O barriers with GFS2 is highly recommended at all times unless the block device is designed so that it cannot lose its write cache content (for example, if it is on a UPS or it does not have a write cache).
quota_quantum=*secs*	Sets the number of seconds for which a change in the quota information may sit on one node before being written to the quota file. This is the preferred way to

Option	Description
	set this parameter. The value is an integer number of seconds greater than zero. The default is 60 seconds. Shorter settings result in faster updates of the lazy quota information and less likelihood of someone exceeding their quota. Longer settings make filesystem operations involving quotas faster and more efficient.
statfs_quantum=*secs*	Setting **statfs_quantum** to 0 is the preferred way to set the slow version of **statfs**. The default value is 30 secs which sets the maximum time period before **statfs** changes will be synced to the master **statfs** file. This can be adjusted to allow for faster, less accurate **statfs** values or slower more accurate values. When this option is set to 0, **statfs** will always report the true values.
statfs_percent=*value*	Provides a bound on the maximum percentage change in the **statfs** information on a local basis before it is synced back to the master **statfs** file, even if the time period has not expired. If the setting of **statfs_quantum** is 0, then this setting is ignored.

3.3. Unmounting a File System

The GFS2 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 GFS2 file system is currently mounted.

3.4. Special Considerations when Mounting GFS2 File Systems

GFS2 file systems that have been mounted manually rather than automatically through an entry in the **fstab** file will not be known to the system when file systems are unmounted at system shutdown. As a result, the GFS2 script will not unmount the GFS2 file system. After the GFS2 shutdown script is run, the standard shutdown process kills off all remaining user processes, including the cluster infrastructure, and tries to unmount the filesystem. This unmount will fail without the cluster infrastructure and the system will hang.

To prevent the system from hanging when the GFS2 file systems are unmounted, you should do one of the following:

- Always use an entry in the **fstab** file to mount the GFS2 file system.
- If a GFS2 file system has been mounted manually with the **mount** command, be sure to unmount the file system manually with the **umount** command before rebooting or shutting down the system.

If your file system hangs while it is being unmounted during system shutdown under these circumstances, perform a hardware reboot. It is unlikely that any data will be lost since the file system is synced earlier in the shutdown process.

3.5. GFS2 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. GFS2 keeps track of the space used by each user and group even when there are no limits in place. GFS2 updates quota information in a transactional way so system crashes do not require quota usages to be reconstructed.

To prevent a performance slowdown, a GFS2 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, GFS2 dynamically reduces the synchronization period as a "hard" quota limit is approached.

GFS2 uses its **gfs2_quota** command to manage quotas. Other Linux quota facilities cannot be used with GFS2.

3.5.1. 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 **gfs2_quota** command. The command only needs to be run on a single node where GFS2 is mounted.

By default, quota enforcement is not set on GFS2 file systems. To enable quota accounting, use the **quota=** of the **mount** command when mounting the GFS2 file system, as described in *Section 3.5.4, "Enabling/Disabling Quota Enforcement"*.

Usage

Setting Quotas, Hard Limit

```
gfs2_quota limit -u User -l Size -f MountPoint
```

```
gfs2_quota limit -g Group -l Size -f MountPoint
```

Setting Quotas, Warn Limit

```
gfs2_quota warn -u User -l Size -f MountPoint
```

```
gfs2_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 GFS2 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 **/mygfs2**.

```
gfs2_quota limit -u Bert -l 1024 -f /mygfs2
```

This example sets the warn limit for group ID 21 to 50 kilobytes on file system **/mygfs2**.

```
gfs2_quota warn -g 21 -l 50 -k -f /mygfs2
```

3.5.2. Displaying Quota Limits and Usage

Quota limits and current usage can be displayed for a specific user or group using the **gfs2_quota get** command. The entire contents of the quota file can also be displayed using the **gfs2_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

```
gfs2_quota get -u User -f MountPoint
```

Displaying Quota Limits for a Group

```
gfs2_quota get -g Group -f MountPoint
```

Displaying Entire Quota File

```
```

```
gfs2_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 GFS2 file system to which the actions apply.

Command Output

GFS2 quota information from the **gfs2_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 **gfs2_quota** command does not resolve UIDs and GIDs into names if the **-n** option is added to the command line.

Space allocated to GFS2'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 **gfs2_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 **/mygfs2**.

```
gfs2_quota list -f /mygfs2
```

This example displays quota information in sectors for group **users** on file system **/mygfs2**.

```
gfs2_quota get -g users -f /mygfs2 -s
```

3.5.3. Synchronizing Quotas

GFS2 stores all quota information in its own internal file on disk. A GFS2 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, GFS2 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 **gfs2_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 **gfs2_quota sync** command to synchronize the quota information from a node to the on-disk quota file between the automatic updates performed by GFS2.

Usage

Synchronizing Quota Information

```
gfs2_quota sync -f MountPoint
```

MountPoint
Specifies the GFS2 file system to which the actions apply.

Tuning the Time Between Synchronizations

```
gfs2_tool settune MountPoint quota_quantum Seconds
```

MountPoint
Specifies the GFS2 file system to which the actions apply.

Seconds
Specifies the new time period between regular quota-file synchronizations by GFS2. 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 **/mygfs2**.

```
gfs2_quota sync -f /mygfs2
```

This example changes the default time period between regular quota-file updates to one hour (3600 seconds) for file system **/mygfs2** on a single node.

```
gfs2_tool settune /mygfs2 quota_quantum 3600
```

3.5.4. Enabling/Disabling Quota Enforcement

In GFS2 file systems, quota enforcement is disabled by default. To enable quota enforcement for a file system, mount the file system with the **quota=on** option specified.

Usage

```
mount -o quota=on BlockDevice MountPoint
```

To mount a file system with quota enforcement disabled, mount the file system with the **quota=off** option specified. This is the default setting.

```
mount -o quota=off BlockDevice MountPoint
```

-o quota={on|off}
Specifies that quota enforcement is enabled or disabled when the file system is mounted.

BlockDevice
Specifies the block device where the GFS2 file system resides.

MountPoint
Specifies the directory where the GFS2 file system should be mounted.

Examples

In this example, the GFS2 file system on **/dev/vg01/lvol0** is mounted on the **/mygfs2** directory with quota enforcement enabled.

```
mount -o quota=on /dev/vg01/lvol0 /mygfs2
```

3.5.5. Enabling Quota Accounting

It is possible to keep track of disk usage and maintain quota accounting for every user and group without enforcing the limit and warn values. To do this, mount the file system with the **quota=account** option specified.

Usage

```
mount -o quota=account BlockDevice MountPoint
```

-o quota=account
Specifies that user and group usage statistics are maintained by the file system, even though the quota limits are not enforced.

BlockDevice
Specifies the block device where the GFS2 file system resides.

MountPoint
Specifies the directory where the GFS2 file system should be mounted.

Example

In this example, the GFS2 file system on **/dev/vg01/lvol0** is mounted on the **/mygfs2** directory with quota accounting enabled.

```
mount -o quota=account /dev/vg01/lvol0 /mygfs2
```

3.6. Growing a File System

The **gfs2_grow** command is used to expand a GFS2 file system after the device where the file system resides has been expanded. Running a **gfs2_grow** command on an existing GFS2 file system fills all spare space between the current end of the file system and the end of the device with a newly initialized GFS2 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 **gfs2_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.

Note

Once you have created a GFS2 file system with the **mkfs.gfs2** command, you cannot decrease the size of the file system.

Usage

```
gfs2_grow MountPoint
```

MountPoint
Specifies the GFS2 file system to which the actions apply.

Comments

Before running the **gfs2_grow** command:

- Back up important data on the file system.
- Determine the volume that is used by the file system to be expanded by running a **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 **gfs2_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 **/mygfs2fs** directory is expanded.

```
[root@dash-01 ~]# gfs2_grow /mygfs2fs
FS: Mount Point: /mygfs2fs
FS: Device:      /dev/mapper/gfs2testvg-gfs2testlv
FS: Size:        524288 (0x80000)
FS: RG size:     65533 (0xfffd)
DEV: Size:       655360 (0xa0000)
The file system grew by 512MB.
gfs2_grow complete.
```

Complete Usage

```
gfs2_grow [Options] {MountPoint | Device} [MountPoint | Device]
```

MountPoint
Specifies the directory where the GFS2 file system is mounted.

Device
Specifies the device node of the file system.

Table 3.3, "GFS2-specific Options Available While Expanding A File System" describes the GFS2-specific options that can be used while expanding a GFS2 file system.

Table 3.3. GFS2-specific Options Available While Expanding A File System

Option	Description
-h	Help. Displays a short usage message.
-q	Quiet. Turns down the verbosity level.
-r MegaBytes	Specifies the size of the new resource group. The default size is 256MB.
-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.

3.7. Adding Journals to a File System

The **gfs2_jadd** command is used to add journals to a GFS2 file system. You can add journals to a GFS2 file system dynamically at any point without expanding the underlying logical volume. The **gfs2_jadd** command must be run on a mounted file system, but it needs to be run on only one node in the cluster. All the other nodes sense that the expansion has occurred.

Note

If a GFS2 file system is full, the **gfs2_jadd** will fail, even if the logical volume containing the file system has been extended and is larger than the file system. This is because in a GFS2 file system, journals are plain files rather than embedded metadata, so simply extending the underlying logical volume will not provide space for the journals.

Before adding journals to a GFS file system, you can use the **journals** option of the **gfs2_tool** to find out how many journals the GFS2 file system currently contains. The following example displays the number and size of the journals in the file system mounted at **/mnt/gfs2**.

```
[root@roth-01 ../cluster/gfs2]# gfs2_tool journals /mnt/gfs2
journal2 - 128MB
journal1 - 128MB
journal0 - 128MB
3 journal(s) found.
```

Usage

```
gfs2_jadd -j Number MountPoint
```

Number
Specifies the number of new journals to be added.

MountPoint
Specifies the directory where the GFS2 file system is mounted.

Examples

In this example, one journal is added to the file system on the **/mygfs2** directory.

```
gfs2_jadd -j1 /mygfs2
```

In this example, two journals are added to the file system on the **/mygfs2** directory.

```
gfs2_jadd -j2 /mygfs2
```

Complete Usage

```
gfs2_jadd [Options] {MountPoint | Device} [MountPoint | Device]
```

MountPoint
Specifies the directory where the GFS2 file system is mounted.

Device
Specifies the device node of the file system.

Table 3.4, "GFS2-specific Options Available When Adding Journals" describes the GFS2-specific options that can be used when adding journals to a GFS2 file system.

Table 3.4. GFS2-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 **gfs2_jadd** command must be

Flag	Parameter	Description
		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 **gfs2_jadd** command. The default value is 1.
-q		Quiet. Turns down the verbosity level.
-V		Displays command version information.

3.8. Data Journaling

Ordinarily, GFS2 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 for very small files because the file data is written to the journal in addition to the metadata. This advantage rapidly reduces as the file size increases. Writing to medium and larger files will be much slower with data journaling turned on.

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 GFS2 files created in a flagged directory (and all its subdirectories). Existing files with zero length can also have data journaling turned on or off.

Enabling data journaling on a directory sets the directory to "inherit jdata", which indicates that all files and directories subsequently created in that directory are journaled. You can enable and disable data journaling on a file with the **chattr** command.

The following commands enable data journaling on the **/mnt/gfs2/gfs2_dir/newfile** file and then check whether the flag has been set properly.

```
[root@roth-01 ~]# chattr +j /mnt/gfs2/gfs2_dir/newfile
[root@roth-01 ~]# lsattr /mnt/gfs2/gfs2_dir
---------j--- /mnt/gfs2/gfs2_dir/newfile
```

The following commands disable data journaling on the **/mnt/gfs2/gfs2_dir/newfile** file and then check whether the flag has been set properly.

```
[root@roth-01 ~]# chattr -j /mnt/gfs2/gfs2_dir/newfile
[root@roth-01 ~]# lsattr /mnt/gfs2/gfs2_dir
------------- /mnt/gfs2/gfs2_dir/newfile
```

You can also use the **chattr** command to set the **j** flag on a directory. When you set this flag for a directory, all files and directories subsequently created in that directory are journaled. The following set of commands sets the **j** flag on the **gfs2_dir** directory, then checks whether the flag has been set properly. After this, the commands create a new file called **newfile** in the **/mnt/gfs2/gfs2_dir** directory and then check whether the **j** flag has been set for the file. Since the **j** flag is set for the directory, then **newfile** should also have journaling enabled.

```
[root@roth-01 ~]# chattr -j /mnt/gfs2/gfs2_dir
```

```
[root@roth-01 ~]# lsattr /mnt/gfs2
---------j--- /mnt/gfs2/gfs2_dir
[root@roth-01 ~]# touch /mnt/gfs2/gfs2_dir/newfile
[root@roth-01 ~]# lsattr /mnt/gfs2/gfs2_dir
---------j--- /mnt/gfs2/gfs2_dir/newfile
```

3.9. 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 GFS2 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 or reduce the frequency of **atime** updates.

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

- Mount with **relatime** (relative atime), which updates the **atime** if the previous **atime** update is older than the **mtime** or **ctime** update.
- Mount with **noatime**, which disables **atime** updates on that file system.

3.9.1. Mount with relatime

The **relatime** (relative atime) Linux mount option can be specified when the file system is mounted. This specifies that the **atime** is updated if the previous **atime** update is older than the **mtime** or **ctime** update.

Usage

```
mount  BlockDevice MountPoint -o relatime
```

BlockDevice
Specifies the block device where the GFS2 file system resides.

MountPoint
Specifies the directory where the GFS2 file system should be mounted.

Example

In this example, the GFS2 file system resides on the **/dev/vg01/lvol0** and is mounted on directory **/mygfs2**. The **atime** updates take place only if the previous **atime** update is older than the **mtime** or **ctime** update.

```
mount /dev/vg01/lvol0 /mygfs2 -o relatime
```

3.9.2. Mount with noatime

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

Usage

```
mount BlockDevice MountPoint -o noatime
```

BlockDevice
Specifies the block device where the GFS2 file system resides.

MountPoint
Specifies the directory where the GFS2 file system should be mounted.

Example

In this example, the GFS2 file system resides on the **/dev/vg01/lvol0** and is mounted on directory **/mygfs2** with **atime** updates turned off.

```
mount /dev/vg01/lvol0 /mygfs2 -o noatime
```

3.10. Suspending Activity on a File System

You can suspend write activity to a file system by using the **gfs2_tool freeze** command. Suspending write activity allows hardware-based device snapshots to be used to capture the file system in a consistent state. The **gfs2_tool unfreeze** command ends the suspension.

Usage

Start Suspension

```
gfs2_tool freeze MountPoint
```

End Suspension

```
gfs2_tool unfreeze MountPoint
```

MountPoint
Specifies the file system.

Examples

This example suspends writes to file system **/mygfs2**.

```
gfs2_tool freeze /mygfs2
```

This example ends suspension of writes to file system **/mygfs2**.

```
gfs2_tool unfreeze /mygfs2
```

3.11. 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 GFS2 file system by using the **fsck.gfs2** command.

Warning

The **fsck.gfs2** command must be run only on a file system that is unmounted from all nodes.

Note

If you have previous experience using the gfs_fsck command on GFS file systems, note that the **fsck.gfs2** command differs from some earlier releases of **gfs_fsck** in the in the following ways:

- Pressing **Ctrl**+**C** while running the **fsck.gfs2** interrupts processing and displays a prompt asking whether you would like to abort the command, skip the rest of the current pass, or continue processing.
- 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 **fsck.gfs2** command to take effect.

Refer to the **fsck.gfs2** man page for additional information about other command options.

Running the **fsck.gfs2** command requires system memory above and beyond the memory used for the operating system and kernel. Each block of memory in the GFS2 file system itself requires approximately five bits of additional memory, or 5/8 of a byte. So to estimate how many bytes of memory you will need to run the **fsck.gfs2** command on your file system, determine how many blocks the file system contains and multiply that number by 5/8.

For example, to determine approximately how much memory is required to run the **fsck.gfs2** command on a GFS2 file system that is 16TB with a block size of 4K, first determine how many blocks of memory the file system contains by dividing 16Tb by 4K:

```
17592186044416 / 4096 = 4294967296
```

Since this file system contains 4294967296 blocks, multiply that number by 5/8 to determine how many bytes of memory are required:

```
4294967296 * 5/8 = 2684354560
```

This file system requires approximately 2.6GB of free memory to run the **fsck.gfs2** command. Note that if the block size was 1K, running the **fsck.gfs2** command would require four times the memory, or approximately 11GB.

Usage

```
fsck.gfs2 -y BlockDevice
```

-y
The **-y** flag causes all questions to be answered with **yes**. With the **-y** flag specified, the **fsck.gfs2** command does not prompt you for an answer before making changes.

BlockDevice
Specifies the block device where the GFS2 file system resides.

Example

In this example, the GFS2 file system residing on block device **/dev/testvol/testlv** is repaired. All queries to repair are automatically answered with **yes**.

```
[root@dash-01 ~]# fsck.gfs2 -y /dev/testvg/testlv
Initializing fsck
Validating Resource Group index.
Level 1 RG check.
(level 1 passed)
Clearing journals (this may take a while)...
Journals cleared.
Starting pass1
Pass1 complete
Starting pass1b
Pass1b complete
Starting pass1c
Pass1c complete
Starting pass2
Pass2 complete
Starting pass3
Pass3 complete
Starting pass4
Pass4 complete
Starting pass5
Pass5 complete
Writing changes to disk
fsck.gfs2 complete
```

3.12. Bind Mounts and Context-Dependent Path Names

GFS2 file systems do not provide support for Context-Dependent Path Names (CDPNs), which allow you to create symbolic links that point to variable destination files or directories. For this functionality in GFS2, you can use the **bind** option of the **mount** command.

The **bind** option of the **mount** command allows you to remount part of a file hierarchy at a different location while it is still available at the original location. The format of this command is as follows.

```
mount --bind olddir newdir
```

After executing this command, the contents of the *olddir* directory are available at two locations: *olddir* and *newdir*. You can also use this option to make an individual file available at two locations.

For example, after executing the following commands the contents of **/root/tmp** will be identical to the contents of the previously mounted **/var/log** directory.

```
[root@menscryfa ~]# cd ~root
[root@menscryfa ~]# mkdir ./tmp
[root@menscryfa ~]# mount --bind /var/log /tmp
```

Alternately, you can use an entry in the **/etc/fstab** file to achieve the same results at mount time. The following **/etc/fstab** entry will result in the contents of **/root/tmp** being identical to the contents of the **/var/log** directory.

```
/var/log               /root/tmp              none    bind           0 0
```

After you have mounted the file system, you can use the **mount** command to see that the file system has been mounted, as in the following example.

```
[root@menscryfa ~]# mount | grep /tmp
/var/log on /root/tmp type none (rw,bind)
```

With a file system that supports Context-Dependent Path Names, you might have defined the **/bin** directory as a Context-Dependent Path Name that would resolve to one of the following paths, depending on the system architecture.

```
/usr/i386-bin
/usr/x86_64-bin
/usr/ppc64-bin
```

You can achieve this same functionality by creating an empty **/bin** directory. Then, using a script or an entry in the **/etc/fstab** file, you can mount each of the individual architecture directories onto the **/bin** directory with a **mount -bind** command. For example, you can use the following command as a line in a script.

```
mount --bind /usr/i386-bin /bin
```

Alternately, you can use the following entry in the **/etc/fstab** file.

```
/usr/1386-bin            /bin              none    bind           0 0
```

A bind mount can provide greater flexibility than a Context-Dependent Path Name, since you can use this feature to mount different directories according to any criteria you define (such as the value of **%fill** for the file system). Context-Dependent Path Names are more limited in what they can encompass. Note, however, that you will need to write your own script to mount according to a criteria such as the value of **%fill**.

Warning

When you mount a file system with the **bind** option and the original file system was mounted **rw**, the new file system will also be mounted **rw** even if you use the **ro** flag; the **ro** flag is silently ignored. In this case, the new file system might be marked as **ro** in the **/proc/mounts** directory, which may be misleading.

3.13. Bind Mounts and File System Mount Order

When you use the **bind** option of the **mount** command, you must be sure that the file systems are mounted in the correct order. In the following example, the **/var/log** directory must be mounted before executing the bind mount on the **/tmp** directory:

```
# mount --bind /var/log /tmp
```

The ordering of file system mounts is determined as follows:

- In general, file system mount order is determined by the order in which the file systems appear in the **fstab** file. The exceptions to this ordering are file systems mounted with the **_netdev** flag or filesystems that have their own **init** scripts.
- A file system with its own **init** script is mounted later in the initialization process, after the file systems in the **fstab** file.
- File systems mounted with the **_netdev** flag are mounted when the network has been enabled on the system.

If your configuration requires that you create a bind mount on which to mount a GFS2 file system, you can order your **fstab** file as follows:

1. Mount local filesystems that are required for the bind mount.
2. Bind mount the directory on which to mount the GFS2 file system.
3. Mount the GFS2 file system.

If your configuration requires that you bind mount a local directory or file system onto a GFS2 file system, listing the file systems in the correct order in the **fstab** file will not mount the file systems correctly since the GFS2 file system will not be mounted until the GFS2 **init** script is run. In this case, you should write an **init** script to execute the bind mount so that the bind mount will not take place until after the GFS2 file system is mounted.

The following script is an example of a custom **init** script. This script performs a bind mount of two directories onto two directories of a GFS2 filesystem. In this example, there is an existing GFS2 mount point at **/mnt/gfs2a**, which is mounted when the GFS2 **init** script runs, after cluster startup.

In this example script, the values of the **chkconfig** statement indicate the following:

- 345 indicates the run levels that the script will be started in
- 29 is the start priority, which in this case indicates that the script will run at startup time after the GFS2 **init** script, which has a start priority of 26
- 73 is the stop priority, which in this case indicates that the script will be stopped during shutdown before the GFS2 script, which has a stop priority of 74

The start and stop values indicate that you can manually perform the indicated action by executing a **service start** and a **service stop** command. For example, if the script is named **fredwilma**, then you can execute **service fredwilma start**.

This script should be put in the **/etc/init.d** directory with the same permissions as the other scripts in that directory. You can then execute a **chkconfig on** command to link the script to the indicated run levels. For example, if the script is named **fredwilma**, then you can execute **chkconfig fredwilma on**.

```
#!/bin/bash
#
# chkconfig: 345 29 73
# description: mount/unmount my custom bind mounts onto a gfs2 subdirectory
#
#
### BEGIN INIT INFO
# Provides:
### END INIT INFO

. /etc/init.d/functions
case "$1" in
  start)
 # In this example, fred and wilma want their home directories
 # bind-mounted over the gfs2 directory /mnt/gfs2a, which has
 # been mounted as /mnt/gfs2a
 mkdir -p /mnt/gfs2a/home/fred &> /dev/null
 mkdir -p /mnt/gfs2a/home/wilma &> /dev/null
 /bin/mount --bind /mnt/gfs2a/home/fred /home/fred
 /bin/mount --bind /mnt/gfs2a/home/wilma /home/wilma
        ;;

  stop)
 /bin/umount /mnt/gfs2a/home/fred
 /bin/umount /mnt/gfs2a/home/wilma
        ;;

  status)
        ;;

  restart)
        $0 stop
        $0 start
        ;;

  reload)
        $0 start
        ;;
  *)
        echo $"Usage: $0 {start|stop|restart|reload|status}"
        exit 1
esac

exit 0
```

3.14. The GFS2 Withdraw Function

The GF2S *withdraw* function is a data integrity feature of GFS2 file systems in a cluster. If the GFS2 kernel module detects an inconsistency in a GFS2 file system following an I/O operation, the file system becomes unavailable to the cluster. The I/O operation stops and the system waits for further I/O operations to error out, preventing further damage. When this occurs, you can stop any other

services or applications manually, after which you can reboot and remount the GFS2 file system to replay the journals. If the problem persists, you can unmount the file system from all nodes in the cluster and perform file system recovery with the **fsck.gfs2** command. The GFS withdraw function is less severe than a kernel panic, which would cause another node to fence the node.

If your system is configured with the **gfs2** startup script enabled and the GFS2 file system is included in the **/etc/fstab** file, the GFS2 file system will be remounted when you reboot. If the GFS2 file system withdrew because of perceived file system corruption, it is recommended that you run the **fsck.gfs2** command before remounting the file system. In this case, in order to prevent your file system from remounting at boot time, you can perform the following procedure:

1. Temporarily disable the startup script on the affected node with the following command:

```
# chkconfig gfs2 off
```

2. Reboot the affected node, starting the cluster software. The GFS2 file system will not be mounted.
3. Unmount the file system from every node in the cluster.
4. Run the **fsck.gfs2** on the file system from one node only to ensure there is no file system corruption.
5. Re-enable the startup script on the affected node by running the following command:

```
# chkconfig gfs2 on
```

6. Remount the GFS2 file system from all nodes in the cluster.

An example of an inconsistency that would yield a GFS2 withdraw is an incorrect block count. When the GFS kernel deletes a file from a file system, it systematically removes all the data and metadata blocks associated with that file. When it is done, it checks the block count. If the block count is not one (meaning all that is left is the disk inode itself), that indicates a file system inconsistency since the block count did not match the list of blocks found.

You can override the GFS2 withdraw function by mounting the file system with the **-o errors=panic** option specified. When this option is specified, any errors that would normally cause the system to withdraw cause the system to panic instead. This stops the node's cluster communications, which causes the node to be fenced.

Appendix A. Converting a File System from GFS to GFS2

Since the Red Hat Enterprise Linux 6 release does not support GFS file systems, you must upgrade any existing GFS file systems to GFS2 file systems with the **gfs2_convert** command. Note that you must perform this conversion procedure on a Red Hat Enterprise Linux 5 system before upgrading to Red Hat Enterprise Linux 6.

Warning

Before converting the GFS file system, you must back up the file system, since the conversion process is irreversible and any errors encountered during the conversion can result in the abrupt termination of the program and consequently an unusable file system.

Before converting the GFS file system, you must use the **gfs_fsck** command to check the file system and fix any errors.

If the conversion from GFS to GFS2 is interrupted by a power failure or any other issue, restart the conversion tool. Do not attempt to execute the **fsck.gfs2** command on the file system until the conversion is complete.

GFS2 file systems do not provide support for context-dependent path names (CDPNs), which allow you to create symbolic links that point to variable destination files or directories. The **gfs2_convert** command identifies CDPNs and replaces them with empty directories with the same name. To achieve the same functionality as CDPNs in GFS2 file systems, you can use the **bind** option of the **mount** command. For more information on bind mounts and context-dependent pathnames in GFS2, see *Section 3.12, "Bind Mounts and Context-Dependent Path Names"*.

When converting full or nearly full file systems, it is possible that there will not be enough space available to fit all the GFS2 file system data structures. In such cases, the size of all the journals is reduced uniformly such that everything fits in the available space.

Use the following procedure to convert a GFS file system to a GFS2 file system.

1. On a Red Hat Enterprise Linux system, make a backup of your existing GFS file system.
2. Unmount the GFS file system from all nodes in the cluster.
3. Execute the **gfs_fsck** command on the GFS file system to ensure there is no file system corruption.
4. Execute **gfs2_convert *gfsfilesystem***. The system will display warnings and confirmation questions before converting *gfsfilesystem* to GFS2.
5. Upgrade to Red Hat Enterprise Linux 6.

The following example converts a GFS file system on block device **/dev/shell_vg/500g** to a GFS2 file system.

```
[root@shell-01 ~]#  /root/cluster/gfs2/convert/gfs2_convert /dev/shell_vg/500g
gfs2_convert version 2 (built May 10 2010 10:05:40)
Copyright (C) Red Hat, Inc.  2004-2006  All rights reserved.
```

```
Examining file system..................
This program will convert a gfs1 filesystem to a gfs2 filesystem.
WARNING: This can't be undone.  It is strongly advised that you:

   1. Back up your entire filesystem first.
   2. Run gfs_fsck first to ensure filesystem integrity.
   3. Make sure the filesystem is NOT mounted from any node.
   4. Make sure you have the latest software versions.
Convert /dev/shell_vg/500g from GFS1 to GFS2? (y/n)y
Converting resource groups..................
Converting inodes.
24208 inodes from 1862 rgs converted.
Fixing file and directory information.
18 cdpn symlinks moved to empty directories.
Converting journals.
Converting journal space to rg space.
Writing journal #1...done.
Writing journal #2...done.
Writing journal #3...done.
Writing journal #4...done.
Building GFS2 file system structures.
Removing obsolete GFS1 file system structures.
Committing changes to disk.
/dev/shell_vg/500g: filesystem converted successfully to gfs2.
```

Appendix B. Revision History

Revision 6.0-1 Wed Nov 15 2010 **Steven Levine** *slevine@redhat.com*

Initial release for Red Hat Enterprise Linux 6

Index

Q

R

S

T

U

W