sgi®

XFS® for Linux® Administration

007–4273–006

New Features in This Guide

This revision contains clarifications and corrections.

Record of Revision

Version	Description
001	January 2003 First printing, incorporating information for the SGI™ ProPack™ v 2.1 for Linux® release
002	May 2003 Incorporates information for the SGI ProPack v 2.2 for Linux release
003	January 2004 Incorporates information for the SGI ProPack v 2.4 for Linux release
004	January 2011 Incorporates information for the SGI InfiniteStorage Software Platform (ISSP) 2.3 release and XFS & XVM media kit 2.3 for Red Hat® Enterprise Linux® (RHEL) 6
005	April 2013 Incorporates information for the ISSP 3.0 release
006	October 2014 Incorporates information for the ISSP 3.3 release

Contents

About This Guide

This guide tells you how to plan, create, and maintain XFS® filesystems on a system running the Linux operating system.

Related Publications

For information about this release, see the following SGI InfiniteStorage Software Platform (ISSP) `README.txt` release note.

The following documents contain additional information:

- *DMF 6 Administrator Guide*
- *CXFS 7 Client-Only Guide for SGI InfiniteStorage*
- *XVM Volume Manager Administrator Guide*
- *Linux Configuration and Operations Guide*
- The user guide and quick start guide for your hardware
- *NIS Administrator's Guide*
- *Personal System Administration Guide*
- *Performance Co-Pilot for Linux User's and Administrator's Guide*
- *SGI L1 and L2 Controller Software User's Guide*

Obtaining Publications

You can obtain SGI documentation as follows:

- See the SGI Technical Publications Library at http://docs.sgi.com. Various formats are available. This library contains the most recent and most comprehensive set of online books, man pages, and other information.
- The `/docs` directory on the ISSP DVD or in the Supportfolio download directory contains information about the release, such as the following:

  - The ISSP release note: `/docs/README.txt`
  - Other release notes: `/docs/README_`*NAME*`.txt`
  - A complete list of the packages and their location on the media: `/docs/RPMS.txt`
  - The packages and their respective licenses: `/docs/PACKAGE_LICENSES.txt`
- The `/docs` directory on the SGI XFS & XVM media kit for RHEL CD or in the Supportfolio download directory contains information about the release, such as the following :
  - The XFS & XVM media kit release note: `/docs/xfs_xvm-`*VERSION*`-reademe.txt`
  - A complete list of the packages and their location on the media: `/docs/xfs_xvm-`*VERSION*`-rpms.txt`
  - The packages and their respective licenses: `/docs/PACKAGE_LICENSES.txt`
- The ISSP release notes and manuals are provided in the `noarch/sgi-isspdocs` RPM and will be installed on the system into the following location:

  `/usr/share/doc/packages/sgi-issp-`*VERSION*/*TITLE*

- You can view man pages by typing `man` *title* at a command line.

Note: The external websites referred to in this guide were correct at the time of publication, but are subject to change.

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.

variable	Italic typeface denotes variable entries and words or concepts being defined.
`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. (Output is shown in nonbold, fixed-space font.)
[ ]	Brackets enclose optional portions of a command or directive line.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this publication, contact SGI. Be sure to include the title and document number of the publication with your comments. (Online, the document number is located in the front matter of the publication. In printed publications, the document number is located at the bottom of each page.)

You can contact SGI in any of the following ways:

- Send e-mail to the following address:

  techpubs@sgi.com

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

- Send mail to the following address:

  SGI
  Technical Publications
  46600 Landing Parkway
  Fremont, CA 94538

SGI values your comments and will respond to them promptly.

Chapter 1

The XFS® Filesystem

The XFS® filesystem provides the following major features:

- Full 64-bit file capabilities (files larger than 2 GB)
- Rapid and reliable recovery after system crashes because of journaling technology
- Efficient support of large, sparse files (files with “holes”)
- Integrated, full-function volume manager support
- Extremely high I/O performance that scales well on multiprocessing systems
- User-specified filesystem block sizes ranging from 512 bytes up to a maximum of the filesystem page size

At least 64 MB of memory is recommended for systems with XFS filesystems.

The maximum size of an XFS filesystem is 2^{64} bytes. The maximum size of an XFS file is 2^{63}-1 bytes.

XFS uses database journaling technology to provide high reliability and rapid recovery. Recovery after a system crash is completed within a few seconds, without the use of a filesystem checker such as the `fsck` command. Recovery time is independent of filesystem size.

XFS is designed to be a very high performance filesystem. XFS as a filesystem is capable of delivering near-raw I/O performance. While traditional filesystems suffer from reduced performance as they grow in size, with XFS there is no performance penalty.

You can create filesystems with block sizes ranging from 512 bytes to a maximum of the filesystem page size. The filesystem page size is a kernel compile option and may be set to 4K on x86_64 systems or to 4K, 8K, or 16K on ia64 systems.

Filesystem extents, which provide for contiguous data within a file, are created automatically for normal files and may be configured at file creation time using the `fcntl()` system call. Extents are multiples of a filesystem block.

Inodes are created as needed by XFS filesystems. You can specify the size of inodes with the `-i size=` option to the filesystem creation command, `mkfs.xfs`. You can also specify the maximum percentage of the space in a filesystem that can be occupied by inodes with the `-i maxpct=` option of the `mkfs.xfs` command.

XFS implements fully journaled extended attributes. An *extended attribute* is a name/value pair associated with a file. Attributes can be attached to all types of inodes: regular files, directories, symbolic links, device nodes, and so forth. Attribute values can contain up to 64 KB of arbitrary binary data.

XFS implements two attribute namespaces:

- A user namespace available to all users, protected by the normal file permissions
- A system namespace, accessible only to privileged users

The system namespace can be used for protected filesystem metadata such as access control lists (ACLs) and hierarchical storage manager (HSM) file migration status. For more information see the, `attr`(1) man page.

To dump XFS filesystems, you must use the command `xfsdump`(8) (not the `dump` command). Restoring from these dumps is done using `xfsrestore`(8). For more information about the relationships between `xfsdump`, `xfsrestore` on XFS filesystems, see the man pages and Chapter 6, "Backup and Recovery Procedures".

Chapter 2

Planning an XFS Filesystem

This chapter discusses the following:

- "Choosing the Filesystem Block Size" on page 3
- "Choosing the Filesystem Directory Block Size" on page 4
- "Choosing the Log Type and Size" on page 4
- "Choosing Allocation Groups and Stripe Units" on page 6
- "Repartitioning the Disks" on page 7

Choosing the Filesystem Block Size

XFS lets you choose the logical block size for each filesystem by using the `-b size=` option of the `mkfs.xfs` command. (Physical disk blocks remain 512 bytes.)

For XFS filesystems on disk partitions and logical volumes and for the data subvolume of filesystems on logical volumes, the block size guidelines are as follows:

- The minimum block size is 512 bytes. Small block sizes increase allocation overhead which decreases filesystem performance. In general, the recommended block size for filesystems under 100 MB and for filesystems with many small files is 512 bytes. The filesystem block size must be a power of two.
- The default block size is 4096 bytes (4 KB). This is the recommended block size for filesystems over 100 MB.
- The maximum block size is the page size of the kernel, which is 4 KB on x86 systems (both 32-bit and 64-bit) and is configurable on ia64 systems. Because large block sizes can waste space, in general block sizes should not be larger than 4096 bytes (4 KB).

Block sizes are specified in bytes as follows:

- Decimal (default)
- Octal (prefixed by `0`)
- Hexadecimal (prefixed by `0x` or `0X`)

If the number has the suffix “K” it is multiplied by 1024.

Choosing the Filesystem Directory Block Size

To select a logical block size for the filesystem directory that is greater than the logical block size of the filesystem, use the `-n` option of the `mkfs.xfs` command. This lets you choose a filesystem block size to match the distribution of data file sizes without adversely affecting directory operation performance. Using this option could improve performance for a filesystem with many small files, such as a news or mail filesystem. In this case, the filesystem logical block size could be small (512 bytes, 1 KB, or 2 KB) and the logical block size for the filesystem directory could be large (4 KB or 8 KB); this can improve the performance of directory lookups because the tree storing the index information has larger blocks and less depth.

You should consider setting a logical block size for a filesystem directory that is greater than the logical block size for the filesystem if you are supporting an application that reads directories (with the `readdir`(3C) or `getdents`(2) system calls) many times in relation to how much it creates and removes files. Using a small filesystem block size saves on disk space and on I/O throughput for the small files.

The data needed to perform a `readdir` operation is segregated from the index information. Directory data blocks can be “read-ahead” in a `readdir`. Performing read-ahead improves the `readdir` performance dramatically. Because the data needed for a `readdir` operation and index information are separate in a directory block, the offset in a directory is limited to 32 bits.

Choosing the Log Type and Size

Each XFS filesystem has a log that contains filesystem journaling records. This log requires dedicated disk space. This disk space does not show up in listings from the `df` command, nor can you access it with a filename.

The location of the disk space depends on the type of log you choose:

Log Type	**Description**
External log	Log records that are maintained in a dedicated log device. To make the XFS filesystem on a logical volume with a log subvolume, use the `mkfs.xfs -l` option.

You should use an external log in the following circumstances:

- If you want the data and log records to be on different partitions
- If you want the data and the log subvolume of a logical volume to be on different partitions or to use different subvolume configurations
- If you want the log subvolume of a logical volume to be striped independently from the data subvolume

Internal log

Log records that are put into a dedicated portion of the disk partition (or data subvolume) that contains user files. This is used when an XFS filesystem is created on a disk partition or logical volume that does not have a log subvolume. This is the default.

The amount of disk space that should be allocated for the log is a function of how the filesystem is used. The amount of disk space required for log records is proportional to the transaction rate and the size of transactions on the filesystem, not the size of the filesystem. Larger block sizes result in larger transactions. Transactions from directory updates (for example, the `mkdir` and `rmdir` commands and the `create()` and `unlink()` system calls) cause more log data to be generated.

You can choose the amount of disk space to dedicate to the log (called the *log size*). The minimum log size for a filesystem is enforced by the size of the largest transaction, which depends on the filesystem and directory block sizes. The maximum log size is 64K blocks or 128 MB, whichever is smaller (this will depend on the block size).

For internal logs, the size of the log is specified with the `-l size=` option when you create the filesystem with the `mkfs.xfs` command. The default log size grows with the size of the filesystem up to the maximum log size, 128 MB, on a 1–TB filesystem. The log size is specified in bytes as described in "Choosing the Filesystem Block Size" on page 3, or as a multiple of the filesystem block size by using the suffix “`b`.”

For a filesystem that is contained in a striped logical volume, the default internal log size is rounded up to a multiple of the stripe unit size. In this case, the user-specified size value must be a multiple of the stripe unit size.

For external logs, the default size of the log is the same as the size of the log device. You can specify the size of the log with the `-l size= option` of the `mkfs.xfs` command, but any additional space in the log device cannot be used. You may find that you need to repartition a disk to create a properly sized log subvolume.

For filesystems with a very high transaction activity, a large log size is recommended. You should avoid making your log too large because a large log can increase filesystem mount time after a crash.

Choosing Allocation Groups and Stripe Units

The data section of an XFS filesystem is divided into allocation groups. You can select the number of allocation groups when you create an XFS filesystem or, alternatively, you can select the size of an allocation group. The larger the number of allocation groups, the more parallelism can be achieved when allocating blocks and inodes. You should avoid selecting a very large number of allocation groups or an allocation group size that will yield a very large number of allocation groups; a large number of allocation groups causes an unreasonable amount of CPU time to be used when the filesystem is close to full.

The minimum allocation group size is 16 MB; the maximum size is just under 4 GB.

The default number of allocation groups is 8, unless the filesystem is smaller than 128 MB or larger than 8 GB. When the filesystem is smaller than 128 MB, the default number of allocation groups is fewer than 8, since the minimum allocation group size is 16 MB. In this case, the data section, by default, will be divided into as many allocation groups as possible that are at least 16 MB. When the filesystem is larger than 8 GB, but smaller than 64 GB, the default number of allocation groups is greater than 8, with each allocation group approximately 1 GB in size. When the filesystem is larger than 64 GB, the default number of allocation groups is still greater than 8, but the allocation group size is 4 GB.

XFS lets you select the stripe unit for a RAID device or stripe volume. This ensures that data allocations, inode allocations, and the internal log will be aligned along stripe units when the end-of-file is extended and the file size is larger than 512 KB. You specify stripe units in 512-byte block units or in bytes. See the `mkfs.xfs`(1M) man page for information on specifying stripe units.

When you specify a stripe unit, you also specify a stripe width in 512-byte block units or in bytes. The stripe width must be a multiple of the stripe unit. The stripe width will be the preferred I/O size returned in the `stat()` system call. See the `mkfs.xfs`(8) man page for information on specifying stripe width.

When used in conjunction with the `-b` (block size) option of the `mkfs.xfs` command, you can use the `-d su=` and `-d sw=` options to specify the stripe unit and stripe width, respectively, in filesystem blocks.

For a RAID device, the default stripe unit is 0, indicating that the feature is disabled. You should configure the stripe unit and width sizes of RAID devices in order to avoid unexpected performance anomalies caused by the filesystem doing non-optimal I/O operations to the RAID unit. For example, if a block write is not aligned on a RAID stripe unit boundary and is not a full stripe unit, the RAID will be forced to do a read/modify/write cycle to write the data. This can have a significant performance impact. By setting the stripe unit size properly, XFS will avoid unaligned accesses.

For a striped volume, the stripe unit that was specified when the volume was created is provided by default.

Repartitioning the Disks

Many system administrators may find that they want or need to repartition disks when they switch to XFS filesystems and/or logical volumes. Some of the reasons to consider repartitioning are:

- Repartitioning can result in a larger pool of free space for all of the formerly separate filesystems
- If you plan to use logical volumes, you may want to put the XFS log into a small subvolume. This requires disk repartitioning to create a small partition for the log subvolume.
- If you plan to use logical volumes, you may want to repartition to create disk partitions of equal size that can be striped or plexed.

Chapter 3

Creating XFS Filesystems

This chapter discusses the following:

- "Making a Filesystem" on page 9
- "Growing a Filesystem" on page 14

Caution: When you create a filesystem, all files already on the disk partition or logical volume are destroyed.

Making a Filesystem

This section discusses the following:

- "Procedure to Make a Filesystem" on page 9
- "`mkfs.xfs` Using the Defaults" on page 12
- "`mkfs.xfs` Specifying Block and Log Size of Internal Log" on page 12
- "`mkfs.xfs` for a Logical Volume with a Log Subvolume" on page 13
- "`mkfs.xfs` for a Directory Block Size Larger than Filesystem Block Size" on page 13

Procedure to Make a Filesystem

Use the following procedure to make an XFS filesystem:

1. Review Chapter 2, "Planning an XFS Filesystem" to verify that you are ready to begin this procedure.
2. Identify the device name of the partition or logical volume where you plan to create the filesystem. This is the value of *partition* in the examples below. For simplicity, the examples in this chapter use an example partition name of `/dev/sdc1`. (For more information about partitioning, see the `parted`(8) man page.)

3. If the disk partition is already mounted, unmount it:

   ```
   # umount partition
   ```

   Caution: Any data that is on the disk partition is destroyed.

   For example:

   ```
   # umount /dev/sdc1
   ```

4. Use the `mkfs.xfs`(8) command to make the filesystem. See the following examples:

   - "`mkfs.xfs` Using the Defaults" on page 12
   - "`mkfs.xfs` Specifying Block and Log Size of Internal Log" on page 12
   - "`mkfs.xfs` for a Logical Volume with a Log Subvolume" on page 13
   - "`mkfs.xfs` for a Directory Block Size Larger than Filesystem Block Size" on page 13

5. Make a mount directory:

   ```
   # mkdir -p mountdir
   ```

   mountdir is the directory to be mounted. For example:

   ```
   # mkdir -p /mnt/scratch_space
   ```

6. Mount the filesystem on the mount directory:

   ```
   # mount partition mountdir
   ```

   For example:

   ```
   # mount /dev/sdc1 /mnt/scratch_space
   ```

7. To configure the system so that the new filesystem is automatically mounted when the system is booted, add the following line to the file `/etc/fstab`:

   ```
   partition mountdir xfs defaults 0 0
   ```

   For example:

   ```
   /dev/sdc1 /mnt/scratch_space xfs defaults 0 0
   ```

Note: Do not run `fsck` for XFS filesystems listed in `/etc/fstab` that use XVM devices (that is, you should set the `fsck` flag to `0`), because XVM devices are not always available. If an `fsck` is run on an XFS filesystem when XVM devices are not available, the system may suspend the system boot sequence and require input from the administrator. XVM includes a helper service that mounts all filesystems listed in `/etc/fstab` that use XVM devices at the time XVM is started during the boot sequence.

`mkfs.xfs` Using the Defaults

If you are making a filesystem on a disk partition or on a logical volume that does not have a log subvolume and want to use the default values for block size and log size, use the following command to create the new XFS filesystem:

```
# mkfs.xfs partition
```

The following example shows the command line to create an XFS filesystem using the defaults and system output:

```
# mkfs.xfs /dev/sdc1
meta-data=/dev/sdc1 isize=256    agcount=18, agsize=1048576 blks
data     =                       bsize=4096   blocks=17921788, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=0
naming   =version 2              bsize=4096
log      =internal log           bsize=4096   blocks=2187, version=1
         =                       sunit=0 blks
realtime =none                   extsz=65536  blocks=0, rtextents=0
```

`mkfs.xfs` Specifying Block and Log Size of Internal Log

If you are making a filesystem on a disk partition or on a logical volume that does not have a log subvolume and want to specify the block size and log size, use the following `mkfs.xfs` command to create the new XFS filesystem:

```
# mkfs.xfs -b size=blocksize -l size=logsize partition
```

blocksize is the filesystem block size (see "Choosing the Filesystem Block Size" on page 3), *logsize* is the size of the area dedicated to log records (see "Choosing the Log Type and Size" on page 4), and *partition* is the device name or logical volume. The default values are 4-KB blocks and a 1000-block log.

The following example shows the command line used to create an XFS filesystem and the system output. The filesystem has a 10–MB internal log and a block size of 1 KB and is on the partition `/dev/dsk/dks0d4s7`.

```
# mkfs.xfs -b size=1k -l size=10m /dev/sdc1
meta-data=/dev/sdc1 isize=256    agcount=18, agsize=4194304 blks
data     =                       bsize=1024   blocks=71687152, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=0
naming   =version 2              bsize=4096
log      =internal log           bsize=1024   blocks=10240, version=1
         =                       sunit=0 blks
realtime =none                   extsz=65536  blocks=0, rtextents=0
```

`mkfs.xfs` for a Logical Volume with a Log Subvolume

If you are making a filesystem on a logical volume that has a log subvolume (for an external log), use the following `mkfs.xfs` command to make the new XFS filesystem:

```
# mkfs.xfs -l logdev=device,size=blocksize partition
```

For example, to make a filesystem on partition `/dev/sdc1`, with an external log on the entire device `/dev/sdh`, whose size is 65536 filesystem blocks, enter the following:

```
# mkfs.xfs -l logdev=/dev/sdh,size=65536b /dev/sdc1
meta-data=/dev/sdc1              isize=256    agcount=4, agsize=76433916
blks
         =                       sectsz=512   attr=2
data     =                       bsize=4096   blocks=305735663,
imaxpct=5
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =/dev/sdh               bsize=4096   blocks=65536, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

`mkfs.xfs` for a Directory Block Size Larger than Filesystem Block Size

If you are making a filesystem with a directory block size that is larger than the filesystem block size, use the following `mkfs.xfs` command to create the new XFS filesystem:

```
# mkfs.xfs -b size=blocksize -n size=dirblocksize  partition
```

dirblocksize is the directory block size (see "Choosing the Filesystem Directory Block Size" on page 4).

For example:

```
# mkfs.xfs -b size=2k -n size=4k /dev/sdc1
meta-data=/dev/sdc1             isize=256    agcount=4,
agsize=152867832 blks
         =                      sectsz=512   attr=2
data     =                      bsize=2048   blocks=611471327,
imaxpct=5
         =                      sunit=0      swidth=0 blks
naming   =version 2             bsize=4096   ascii-ci=0
log      =internal log          bsize=2048   blocks=298569, version=2
         =                      sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                  extsz=4096   blocks=0, rtextents=0
```

Growing a Filesystem

To grow an existing XFS filesystem, increase the available disk space and use the `xfs_growfs`(8) command. The filesystem must be mounted to be grown. The existing contents of the filesystem are undisturbed, and the added space becomes available for additional file storage.

Growing an XFS filesystem is supported on XVM volumes. You must first grow the XVM volume before growing the XFS filesystem. For information on XVM volumes, see the *XVM Volume Manager Administrator's Guide.*

The following example grows a filesystem mounted at `/mnt`:

```
# xfs_growfs /mnt
meta-data=/mnt                   isize=256    agcount=30, agsize=262144 blks
data     =                       bsize=4096   blocks=7680000, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=0
naming   =version 2              bsize=4096
log      =internal               bsize=4096   blocks=1200 version=1
         =                       sunit=0 blks
realtime =none                   extsz=65536  blocks=0, rtextents=0
data blocks changed from 7680000 to 17921788
```

Filesystem Maintenance

The chapter discusses the following:

Filesystem Reorganization

Filesystems can become fragmented over time. When a filesystem is fragmented, blocks of free space are small and files have many extents. The `xfs_fsr` command reorganizes filesystems so that the layout of the extents is improved. This improves overall performance. See the `xfs_fsr`(8) man page for more information.

Filesystem Corruption

Most often, a filesystem is corrupted because the system experienced a panic. This can be caused by system software failure, hardware failure, or human error (for example, pulling the plug). Another possible source of filesystem corruption is overlapping partitions.

There is no foolproof way to predict hardware failure. The best way to avoid hardware failures is to conscientiously follow recommended diagnostic and maintenance procedures.

Human error is probably the greatest single cause of filesystem corruption. To avoid problems, follow these rules closely:

- Always shut down the system properly. Do not simply turn off power to the system. Use a standard system shutdown tool, such as the `shutdown`(8) command.
- Never remove a filesystem physically (never pull out a hard disk) without first turning off power.

- Never physically write-protect a mounted filesystem, unless it is mounted read-only.
- Do not mount filesystems on dual-hosted disks on two systems simultaneously.

The best way to ensure against data loss is to make regular, careful backups.

In some cases, XFS filesystem corruption, even on the root filesystem, can be repaired with the command `xfs_repair`. For more information about `xfs_repair`(8) see the man page and "Checking Filesystem Consistency" on page 16

Checking Filesystem Consistency

This section discusses the following:

- "Overview of the Commands to Check Filesystem Consistency" on page 16
- "`xfs_repair -n` Command Line" on page 17
- "`xfs_check` Command Line" on page 18

Overview of the Commands to Check Filesystem Consistency

You can use the following commands to check the consistency of a filesystem:

- `xfs_repair -n` *(no-modify mode)*

  The `xfs_repair -n` command is optimized to quickly and efficiently check an XFS filesystem. The `xfs_repair -n` command checks XFS filesystem consistency without making any attempt to repair problems, and it performs a more complete check than `xfs_check`. However, it performs only limited checking of extended attributes.

  Note: The `xfs_repair` command without the `-n` option makes modifications and should be used with caution; see "Repairing XFS Filesystem Problems" on page 18

- `xfs_check`

  The `xfs_check` command calls the checking routines of the general-purpose XFS filesystem debugger `xfs_db`, which requires more memory and time to check a filesystem than does `xfs_repair -n`. You can use `xfs_check` on filesystems with extended attributes. (For more information about extended attributes, see the `attr`(1) man page.)

The filesystem to be checked must have been unmounted cleanly using normal system administration procedures (the `umount` command or system shutdown), not as a result of a crash or system reset. If the filesystem has not been unmounted cleanly, mount it and unmount it cleanly before running `xfs_check` or `xfs_repair -n`.

Unlike `fsck`, `xfs_check` and `xfs_repair -n` are not invoked automatically on system startup. You should use these commands if you suspect a filesystem consistency problem.

`xfs_repair -n` Command Line

Caution: If you suspect problems with the root filesystem, you should use a boot disk or an alternate root to run `xfs_repair`.

The command line for `xfs_repair -n` is:

```
# xfs_repair -n device
```

device is the device file for a disk partition or logical volume that contains an XFS filesystem, such as `/dev/xscsi/pci02.02.0-1/target3/lun0/part1`

The following example shows output with no consistency problems found:

```
# xfs_repair -n /dev/xscsi/pci02.02.0-1/target3/lun0/part1
Phase 1 - find and verify superblock...
Phase 2 - using internal log
        - scan filesystem freespace and inode maps...
        - found root inode chunk
Phase 3 - for each AG...
        - scan (but don’t clear) agi unlinked lists...
        - process known inodes and perform inode discovery...
        - agno = 0
        - agno = 1
        ...
        - process newly discovered inodes...
Phase 4 - check for duplicate blocks...
        - setting up duplicate extent list...
        - check for inodes claiming duplicate blocks...
        - agno = 0
        - agno = 1
        ...
No modify flag set, skipping phase 5
Phase 6 - check inode connectivity...
        - traversing filesystem starting at / ...
        - traversal finished ...
        - traversing all unattached subtrees ...
        - traversals finished ...
        - moving disconnected inodes to lost+found ...
Phase 7 - verify link counts...
No modify flag set, skipping filesystem flush and exiting.
```

For information about potential errors, see "Common `xfs_repair` Error Messages" on page 20.

For more details, see the `xfs_repair`(8) man page.

`xfs_check` Command Line

The command line for `xfs_check` is:

```
# xfs_check device
```

device is the disk or volume device for the filesystem.

If no consistency problems were found, `xfs_check` returns without displaying any output, as shown in the following example:

```
# xfs_check /dev/xscsi/pci02.02.0-1/target3/lun0/part1
#
```

If a problem is reported, use `xfs_repair -n` to obtain more information. See "`xfs_repair -n` Command Line" on page 17.

For more information, see the `xfs_check`(8) man page.

Repairing XFS Filesystem Problems

The `xfs_repair` command without the `-n` option checks XFS filesystem consistency and sometimes repairs problems that are found. This section discusses the following:

- "Repairing Inconsistent Filesystems with `xfs_repair`" on page 19
- "Common `xfs_repair` Error Messages" on page 20
- "`xfs_repair` Error Messages When Files Are in `lost+found`" on page 21
- "What to Do If `xfs_repair` Cannot Repair a Filesystem" on page 23
- "Mounting a Filesystem Without Log Recovery" on page 24

Repairing Inconsistent Filesystems with `xfs_repair`

Caution: To avoid filesystem damage when using `xfs_repair` without the `-n` option, you must ensure that the storage hardware, including RAID and interconnect hardware, are not experiencing any problems.

If you suspect problems with the root filesystem, you should use a boot disk or an alternate boot disk to run `xfs_repair`.

The `xfs_repair` (without the `-n` option) checks XFS filesystem consistency and, if problems are detected, also corrects them if possible. The filesystem to be checked and repaired must have been unmounted cleanly using normal system administration procedures (the `umount` command or system shutdown), not as a result of a crash or system reset. If the filesystem has not been unmounted cleanly, mount it and unmount it cleanly before running `xfs_repair`.

The command line for `xfs_repair` when you want it to repair any inconsistencies it finds is:

```
# xfs_repair device
```

device is the disk or volume device for the filesystem. It must not be mounted.

The following example shows the output you see from running `xfs_repair` on a clean filesystem:

```
# xfs_repair /dev/xscsi/pci02.02.0-1/target3/lun0/part1
Phase 1 - find and verify superblock...
Phase 2 - using internal log
        - zero log...
        - scan filesystem freespace and inode maps...
        - found root inode chunk
```

```
Phase 3 - for each AG...
        - scan and clear agi unlinked lists...
        - process known inodes and perform inode discovery...
        - agno = 0
        - agno = 1
        ...
        - process newly discovered inodes...
Phase 4 - check for duplicate blocks...
        - setting up duplicate extent list...
        - clear lost+found (if it exists) ...
        - check for inodes claiming duplicate blocks...
        - agno = 0
        - agno = 1
        ...
Phase 5 - rebuild AG headers and trees...
        - reset superblock...
Phase 6 - check inode connectivity...
        - resetting contents of realtime bitmap and summary inodes
        - ensuring existence of lost+found directory
        - traversing filesystem starting at / ...
        - traversal finished ...
        - traversing all unattached subtrees ...
        - traversals finished ...
        - moving disconnected inodes to lost+found ...
Phase 7 - verify and correct link counts...
done
```

Common `xfs_repair` Error Messages

Some common error messages from `xfs_repair` and the repairs that it performs are the following:

`disconnected inode 242002, moving to lost+found`

> `xfs_repair` found an inode that is in use, but is not connected to the filesystem. The inode is moved to the filesystem's `lost+found` directory. Its name is its inode number (in this example, `242002`). If the disconnected inode is a directory, the directory's subtree is preserved—all of its child inodes are automatically moved with it, so the entire directory subtree moves to `lost+found`.

`imap claims in-use inode 2444941 is free, correcting imap`

The inode allocation map in the filesystem behaves as if inode `2444941`(in this example) is free, but the inode itself looks like it is still in use. `xfs_repair` corrects the inode map to say that the inode is in use.

`entry references free inode 2444940 in shortform directory 2444922 junking entry "fb" in directory inode 2444922`

A directory entry points to an inode (in this example, `2444940`) that `xfs_repair` has determined is actually free. `xfs_repair` junks the directory entry. The term *shortform* means a small directory. In larger directories, the entry deletion is usually a two-pass process. In this case, the second part of the message reads something like `marking bad entry`, `marking entry to be deleted`, or `will clear entry`.

`resetting inode 241996 nlinks from 5 to 3`

`xfs_repair` detected a mismatch between the number of directory entries pointing to the inode (in this example, `241996`) and the number of links recorded in the inode. It corrected the number (from `5` to `3` in this case).

`cleared inode 2444926`

There was something wrong with the inode that was not correctable, so `xfs_repair` turned it into a zero-length free inode. This usually happens because the inode claims blocks that are used by something else or the inode itself is badly corrupted. Typically, the `cleared inode` message is preceded by one or more messages indicating why the inode must be cleared.

`xfs_repair` Error Messages When Files Are in `lost+found`

If `xfs_repair` has put files and directories in a filesystem's `lost+found` directory and you do not remove them, the next time you run `xfs_repair` it temporarily disconnects the inodes for those files and directories. They are reconnected before `xfs_repair` terminates. As a result of the disconnected inodes in `lost+found`, you see output like this:

```
Phase 1 - find and verify superblock...
Phase 2 - zero log...
        - scan filesystem freespace and inode maps...
        - found root inode chunk
Phase 3 - for each AG...
        - scan and clear agi unlinked lists...
        - process known inodes and perform inode discovery...
        - agno = 0
        - agno = 1
        ...
        - process newly discovered inodes...
Phase 4 - check for duplicate blocks...
        - setting up duplicate extent list...
        - clear lost+found (if it exists) ...
        - clearing existing ‘‘lost+found’’ inode
        - deleting existing ‘‘lost+found’’ entry
        - check for inodes claiming duplicate blocks...
        - agno = 0
imap claims in-use inode 242000 is free, correcting imap
        - agno = 1
        - agno = 2
        ...
Phase 5 - rebuild AG headers and trees...
        - reset superblock counters...
Phase 6 - check inode connectivity...
        - ensuring existence of lost+found directory
        - traversing filesystem starting at / ...
        - traversal finished ...
        - traversing all unattached subtrees ...
        - traversals finished ...
        - moving disconnected inodes to lost+found ...
disconnected inode 242000, moving to lost+found
Phase 7 - verify and correct link counts...
done
```

In this example, inode `242000` was an inode that was moved to `lost+found` during a previous `xfs_repair` run. This run of `xfs_repair` found that the filesystem is consistent. If the `lost+found` directory had been empty, in phase 4 only the messages about clearing and deleting the `lost+found` directory would have appeared. The `imap claims` and `disconnected inode` messages appear (one pair of messages per inode) if there are inodes in the `lost+found` directory.

What to Do If `xfs_repair` Cannot Repair a Filesystem

If `xfs_repair` fails to repair the filesystem successfully, try giving the same `xfs_repair` command twice more; `xfs_repair` may be able to make more repairs on successive runs. If `xfs_repair` fails to fix the consistency problems in three tries, your next step depends upon where it failed:

- If `xfs_repair` failed in phase 1, you must restore lost files from backups.
- If `xfs_repair` failed in phase 2 or later, you may be able to restore files from the disk by backing up and restoring the files on the filesystem.

If `xfs_repair` failed in phase 2 or later, follow these steps:

1. Mount the filesystem read-only using `mount -r`.
2. Make a filesystem backup with `xfsdump`.
3. Use `mkfs.xfs` to a make new filesystem on the same disk partition or logical volume.
4. Restore the files from the backup with `xfsrestore`.

See Chapter 6, "Backup and Recovery Procedures" for information about `xfsdump` and `xfsrestore`.

Mounting a Filesystem Without Log Recovery

If a filesystem is damaged to the extent that you are unable to mount the filesystem successfully in the standard fashion, you may be able to recover some of its data by mounting the filesystem with the `-o norecover` option of the `mount` command. This option mounts the filesystem without running log recovery. You must mount the filesystem as read-only when you use this option.

Chapter 5

Disk Quotas

This chapter discusses the following:

- "Overview of Disk Quotas" on page 25
- "Enabling Quotas" on page 27
- "Setting Quota Limits" on page 29
- "Displaying Quota Information" on page 31
- "Administering Quotas" on page 31
- "Monitoring Disk Space Usage with Quota Accounting" on page 32
- "Checking Disk Space Usage" on page 33

For more information, see the `xfs_quota`(8) man pages.

Overview of Disk Quotas

If your system is constantly short of disk space and you cannot increase the amount of available space, you an use disk quotas to manage your existing space.

Disk quotas let you limit the amount of space a user can occupy and the number of files (inodes) each user can own. You can implement *hard* or *soft* limits; hard limits are enforced by the system, soft limits merely remind the user to trim disk usage. Disk usage limits are not enforced for `root`.

With soft limits, whenever a user logs in with a usage greater than the assigned soft limit, that user is warned (by the `login` command). When the user exceeds the soft limit, the timer is enabled. Any time the quota drops below the soft limits, the timer is disabled. If the timer is enabled longer than a time period set by the system administrator, the particular limit that has been exceeded is treated as if the hard limit has been reached, and no more disk space is allocated to the user. The only way to reset this condition is to reduce usage below the quota. Only `root` may set the time limits, and this is done on a per-filesystem basis.

You can impose limits on some users and not others, some filesystems and not others, and on total disk usage per user, or total number of files. There is no limit to the number of accounts and there is little performance penalty for large numbers of users.

You can also impose limits according to user ID, group ID, or project ID. You can associate a directory in the filesystem hierarchy with a project ID by including it in the `/etc/projects` file. (You can use `/etc/projid` to map each project name to its number.) With project quotas in effect, such a directory and all files and directories below it can be subjected to a quota, meaning that the aggregate resource used thereunder is limited. For more information, see the `xfs_quota`(8) man page.

Note: Group quotas and project quotas are mutually exclusive per filesystem because XFS records either the project ID or the group ID of a file in the same physical location; how the number is interpreted depends upon whether project or group quotas are in force.

Disk quotas can be used to do disk usage accounting. Disk usage accounting monitors disk usage, but does not enforce disk usage limits. See "Monitoring Disk Space Usage with Quota Accounting" on page 32 for more information.

You must first turn on disk quotas on a filesystem, then you can set quotas on that filesystem for individual users and for projects or groups.

For more details about disk quotas, see the `quotas`(4) man page.

Enabling Quotas

This section discusses the following:

- "Enabling Quotas for Users" on page 27
- "Enabling Quotas for Groups" on page 27
- "Enabling Quotas for Projects" on page 28

Enabling Quotas for Users

You can enable quotas for users in these ways:

- To turn on disk quotas automatically for users on a non-root filesystem, include the option `quota` in the `/etc/fstab` entry, for example:

  ```
  /dev/foo / xfs rw,quota 0 0
  ```

- To turn on disk quotas manually for users on a non-root filesystem, mount the filesystem with this command:

  ```
  # mount -o quota fsname rootdir
  ```

  fsname is the device name of the filesystem, *rootdir* is the directory where the filesystem is mounted.

- To turn on disk quotas for users on the root filesystem, you must pass the quota mount options into the kernel at boot time through the Linux `rootflags` boot option. The following example adds the `rootflags=quota` option to the append line in `elilo.conf`:

  ```
  append="root=/dev/xscsi/pci00.01.0-1/tsrget0/lun0/part3 rootflags=quota"
  ```

Enabling Quotas for Groups

You can enable quotas for groups in these ways:

- To turn on disk quotas automatically for groups on a non-root filesystem, include the option `gquota` in the `/etc/fstab` entry, for example:

  ```
  /dev/foo / xfs rw,gquota 0 0
  ```

- To turn on disk quotas manually for groups on a non-root filesystem, mount the filesystem with this command:

  ```
  # mount -o gquota fsname  rootdir
  ```

  fsname is the device name of the filesystem, *rootdir* is the directory where the filesystem is mounted.

- To turn on disk quotas for groups on the root filesystem, you must pass the quota mount options into the kernel at boot time through the Linux `rootflags` boot option. The following example adds the `rootflags=gquota` option to the append line in `elilo.conf`:

```
append="root=/dev/xscsi/pci00.01.0-1/tsrget0/lun0/part3 rootflags=gquota"
```

Enabling Quotas for Projects

Note: Group and project quotas are mutually exclusive per filesystem.

You can enable quotas for projects in these ways:

- To turn on disk quotas automatically for projects on a non-root filesystem, include the option `prjquota` in the `/etc/fstab` entry, for example:

  ```
  /dev/foo / xfs rw,prjquota 0 0
  ```

- To turn on disk quotas manually for projects on a non-root filesystem, mount the filesystem with this command:

  ```
  # mount -o prjquota fsname  rootdir
  ```

  fsname is the device name of the filesystem, *rootdir* is the directory where the filesystem is mounted.

- To turn on disk quotas for projects on the root filesystem, you must pass the quota mount options into the kernel at boot time through the Linux `rootflags` boot option. The following example adds the `rootflags=prjquota` option to the append line in `elilo.conf`:

```
append="root=/dev/xscsi/pci00.01.0-1/tsrget0/lun0/part3 rootflags=prjquota"
```

Setting Quota Limits

After enabling quotas, you can set limits for users, groups, or projects:

- "Setting Quota Limits for Users" on page 29
- "Setting Quota Limits for Groups" on page 29
- "Setting Quota Limits for Projects" on page 30

Note: Group and project quotas are mutually exclusive per filesystem.

Setting Quota Limits for Users

After completing "Enabling Quotas for Users" on page 27, do the following to specify quota limits for a user:

```
# xfs_quota -x -c 'limit -u bsoft=N bhard=N user' rootdir
```

where:

- *N* is a soft or hard limit for disk usage in blocks of the specified unit: `k` (kilobytes), `m` (megabytes), `g` (gigabytes), or `t` (terabytes)
- *user* is a user name or numeric user ID
- *rootdir* is the mount point of the XFS filesystem.

For example, to set limits for user `userA` on `/mnt/myxfs` using a soft limit of 5 Mbytes and a hard limit of 6 Mbytes:

```
# xfs_quota -x -c 'limit -u bsoft=5m bhard=6m userA' /mnt/myxfs
```

Setting Quota Limits for Groups

After completing "Enabling Quotas for Groups" on page 27, setting disk quota limits for groups is similar to setting limits for users (as described in "Setting Quota Limits for Users" on page 29), but uses the `-g` option and the group name or ID.

To specify quota limits for a group:

```
# xfs_quota -x -c 'limit -g bsoft=N bhard=N group' rootdir
```

where:

- *N* is a soft or hard limit for disk usage in blocks of the specified unit: k (kilobytes), `m` (megabytes), `g` (gigabytes), or `t` (terabytes)
- *group* is a group name or numeric group ID
- *rootdir* is the mount point of the XFS filesystem.

For example, to set limits for group `groupA` on `/mnt/myxfs` using a soft limit of 5 Mbytes and a hard limit of 6 Mbytes:

```
# xfs_quota -x -c 'limit -g bsoft=5m bhard=6m groupA' /mnt/myxfs
```

Setting Quota Limits for Projects

After completing "Enabling Quotas for Projects" on page 28, setting limits for projects is similar to setting limits for groups (as described in "Setting Quota Limits for Groups" on page 29), but uses the `-p` option and the project name or ID.

Note: Group and project quotas are mutually exclusive per filesystem.

To specify quota limits for a project:

```
# xfs_quota -x -c 'limit -p bsoft=N bhard=N project' rootdir
```

where:

- *N* is a soft or hard limit for disk usage in blocks of the specified unit: k (kilobytes), `m` (megabytes), `g` (gigabytes), or `t` (terabytes)
- *project* is a project name or numeric group ID
- *rootdir* is the mount point of the XFS filesystem.

For example, to set limits for project `projectA` on `/mnt/myxfs` using a soft limit of 5 Mbytes and a hard limit of 6 Mbytes:

```
# xfs_quota -x -c 'limit -p bsoft=5m bhard=6m projectA' /mnt/myxfs
```

For more information about projects, see the `xfs_quota`(8) man page.

Displaying Quota Information

Some commands that display information about disk quotas are as follows:

- To display a report that shows whether disk quotas are on or off for each filesystem:

  ```
  # xfs_quota -x -c state
  ```

- To see filesystem quota information for a specific filesystem:

  ```
  # xfs_quota -x -c report rootdir
  ```

  For example, to see quota information for the `/mnt/myxfs` filesystem:

  ```
  # xfs_quota -x -c report /mnt/myxfs
  ```

- To get information about group disk quotas for each filesystem:

  ```
  # xfs_quota -x -c 'report -g'
  ```

Administering Quotas

If the filesystem being dumped contains quotas, `xfsdump` will use `xfs_quota`(8) to store the quotas in the following files in the root of the filesystem to be dumped:

`xfsdump_quotas`	User quotas
`xfsdump_quotas_group`	Group quotas

These files will then be included in the dump. These files will appear only for those quotas that are enabled on the filesystem being dumped. Upon restoration, you can use `xfs_quota` to reactivate the quotas for the filesystem.

Note: The `xfsdump_quotas` file will probably require modification to change the filesystem or UIDs if the filesystem has been restored to a different partition or system.

To create quota reports, do the following:

- To create a file that lists the current quota limits of all the filesystems for users, enter this command as superuser:

  ```
  # xfs_quota -x -c 'report -f quotafile'
  ```

- To create a file that lists the current quota limits of all the filesystems for groups, enter this command as superuser:

  ```
  # xfs_quota -x -c 'report -g -f quotafile'
  ```

Monitoring Disk Space Usage with Quota Accounting

The disk quotas system can be used to monitor disk space usage without enforcing disk usage limits. Disk quota accounting can be enabled by user or by group.

Use the following commands to turn on disk usage accounting without enforcement, stop disk usage accounting, and report disk space usage:

- To turn on disk usage accounting automatically on a filesystem for user quotas, include the option `qnoenforce` in the `/etc/fstab` entry:

  ```
  /dev/foo / xfs rw,qnoenforce 0 0
  ```

- To turn on disk usage accounting automatically on a filesystem for group quotas, include the option `gqnoenforce` in the `/etc/fstab` entry:

  ```
  /dev/foo / xfs rw,
  gqnoenforce 0 0
  ```

- To turn on disk usage accounting manually for user quotas on a non-root filesystem, when mounting the filesystem:

  ```
  # mount -o qnoenforce fsname rootdir
  ```

  fsname is the device name of the filesystem, *rootdir* is the directory where the filesystem is mounted.

- To turn on disk usage accounting manually on a non-root filesystem for group quotas when mounting the filesystem:

  ```
  # mount -o gqnoenforce fsname rootdir
  ```

- To turn on disk usage accounting manually on the root filesystem for user quotas, execute the following commands. The `quotaon` command turns on disk accounting with enforcement, and the `quotaoff -o` command turns off the enforcement:

  ```
  # quotaon -v /
  # quotaoff -v -o enforce /
  # reboot
  ```

- To turn on disk usage accounting manually on the root filesystem (`/`) for group quotas:

  ```
  # quotaon -v -o gquota /
  # quotaoff -v -o gqenforce /
  # reboot
  ```

- To stop disk usage accounting on a filesystem for user quotas:

  ```
  # quotaoff fsname
  ```

- To stop disk usage accounting on a filesystem for group quotas:

  ```
  # quotaoff -o gquota fsname
  ```

- To get information about disk usage, use the commands described in "Checking Disk Space Usage" on page 33.

Checking Disk Space Usage

The `quota` command reports the amount of disk usage per user, per group, or per project on a filesystem, as well as additional information about the disk quotas. You must turn on quotas to use this feature, even if you are not going to enforce quota limits. For instructions on monitoring disk space usage without enforcing disk usage limits see "Monitoring Disk Space Usage with Quota Accounting" on page 32.

For information on the output of the quota command, see "Displaying Quota Information" on page 31.

Chapter 6

Backup and Recovery Procedures

This section discusses the following:

For more information, see the `xfsdump`(8) and `xfsrestore`(8) man pages.

Features of `xfsdump` and `xfsrestore`

The `xfsdump` and `xfsrestore` utilities fully support XFS filesystems. With `xfsdump` and `xfsrestore`, you can back up and restore data using local or remote drives. You can back up filesystems, directories, and individual files, and then restore them independently of how they were backed up. `xfsdump` also allows you to back up “live” (mounted, in-use) filesystems.

With `xfsdump` and `xfsrestore`, you can recover from intentional or accidental interruptions—this means you can interrupt a dump or restore at any time, and then resume it whenever desired. `xfsdump` and `xfsrestore` support incremental dumps, and multiple dumps can be placed on a single media object.

`xfsdump` and `xfsrestore` support the following:

- XFS features including 64-bit inode numbers, file lengths, and holes
- Multiple media types (disk and various kinds of tape)

- File types:

  Regular
  Directory
  Symbolic link
  Block and character special
  FIFO
  socket

`xfsdump` and `xfsrestore` retain hard links. `xfsdump` does not affect the state of the filesystem being dumped (for example, access times are retained). `xfsrestore` detects and bypasses media errors and recovers rapidly after encountering them. `xfsdump` does not cross mount points, local or remote.

`xfsdump` optionally prompts for additional media when the end of the current media is reached. Operator estimates of media capacity are not required and `xfsdump` also supports automated backups. `xfsdump` maintains an extensive online inventory of all dumps performed. Inventory contents can be viewed through various filters to quickly locate specific dump information. `xfsrestore` supports interactive operation, allowing selection of individual files or directories for recovery. It also permits selection from among backups performed at different times when multiple dumps are available. Dump contents may also be viewed noninteractively.

Note: If you are using disk quotas on XFS filesystems, see Chapter 5, "Disk Quotas".

Media Layout for `xfsdump`

The following section introduces some terminology and then describes the way `xfsdump` formats data on the storage media for use by `xfsrestore`.

While `xfsdump` and `xfsrestore` are often used with tape media, the utilities actually support multiple kinds of media, so in the following discussions, the term *media object* is used to refer to the media in a generic fashion. The term *dump* refers to the result of a single use of the `xfsdump` command to output data files to the selected media objects. An instance of the use of `xfsdump` is referred to as a **dump session**.

The dump session sends a single *dump stream* to the media objects. The dump stream may contain as little as a single file or as much as an entire filesystem. The dump stream is composed of *dump objects*, which are:

- One or more *data segments*

- An optional *dump inventory*
- A *stream terminator*

The data segment contains the actual data, the dump inventory contains a list of the dump objects in the dump, and the stream terminator marks the end of the dump stream. When a dump stream is composed of multiple dump objects, each object is contained in a *media file*. Some output devices, for example standard output, do not support the concept of media files—the dump stream is only the data.

Possible `xfsdump` Layouts

The simplest dump, for example the dump of a small amount of data to a single tape, produces a data segment and a stream terminator as the only dump objects. If the optional inventory object is added, you have a dump like that illustrated in Figure 6-1. (In the data layout diagrams in this section, the optional inventory object is always included.)

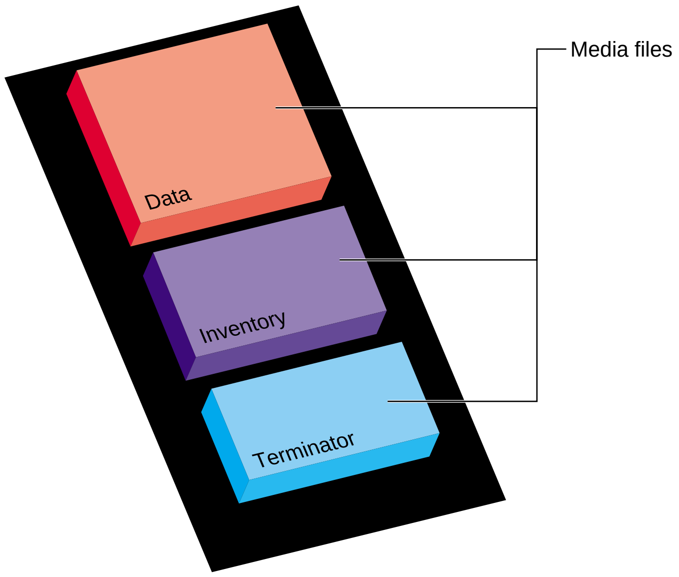

Figure 6-1 Single Dump on Single Media Object

You can also dump data streams that are larger than a single media object. The data stream can be broken between any two media files including data segment boundaries. (The inventory is never broken into segments.) In addition, if you specify multiple drives, the dump is automatically broken into multiple streams. The `xfsdump` utility prompts for a new media object when the end of the current media object is reached.

Figure 6-2 illustrates the data layout of a single dump session that requires two media objects on each of two devices.

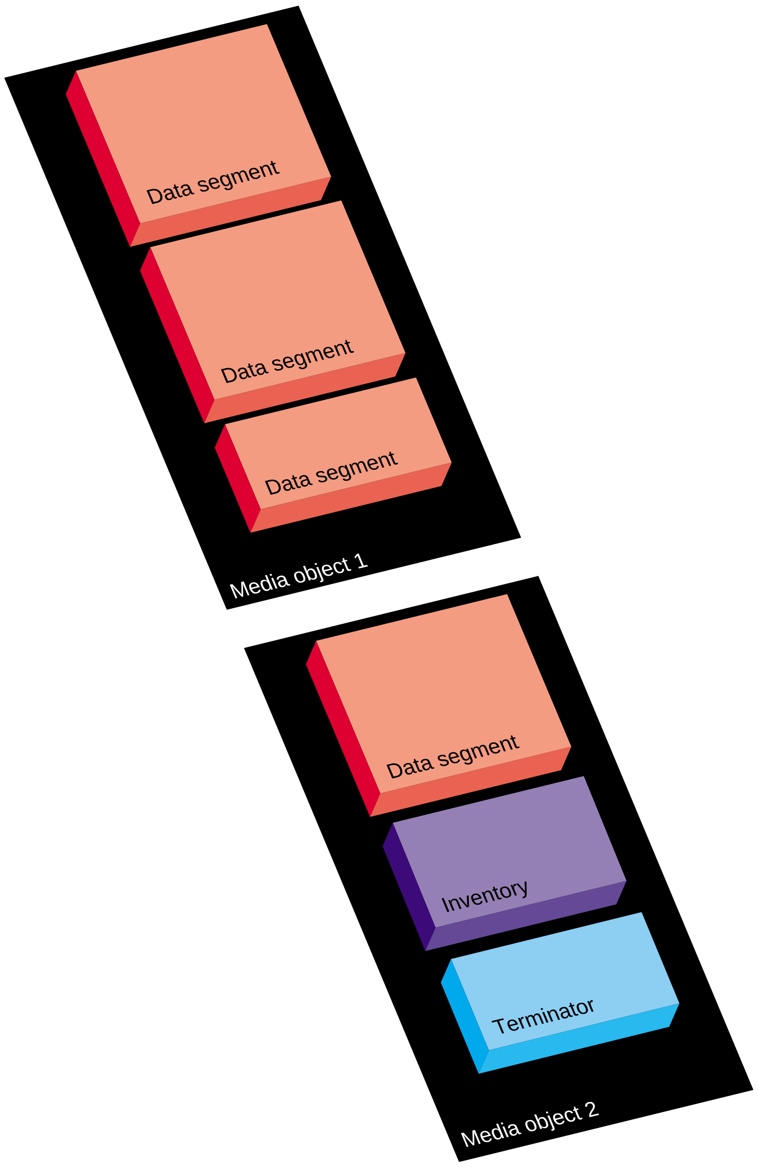

Figure 6-2 Single Dump on Multiple Media Objects

The `xfsdump` utility also accommodates multiple dumps on a single media object. When dumping to tape, for example, the tape is automatically advanced past the existing dump sessions and the existing stream terminator is erased. The new dump data is then written, followed by the new stream terminator. (For drives that do not permit termination to operate in this way, other means are used to achieve the same effective result.)

Figure 6-3 illustrates the layout of media files for two dumps on a single media object.

Figure 6-4 illustrates a case in which multiple dumps use multiple media objects. If media files already exist on the additional media objects, the `xfsdump` utility finds the existing stream terminator, erases it, and begins writing the new dump data stream.

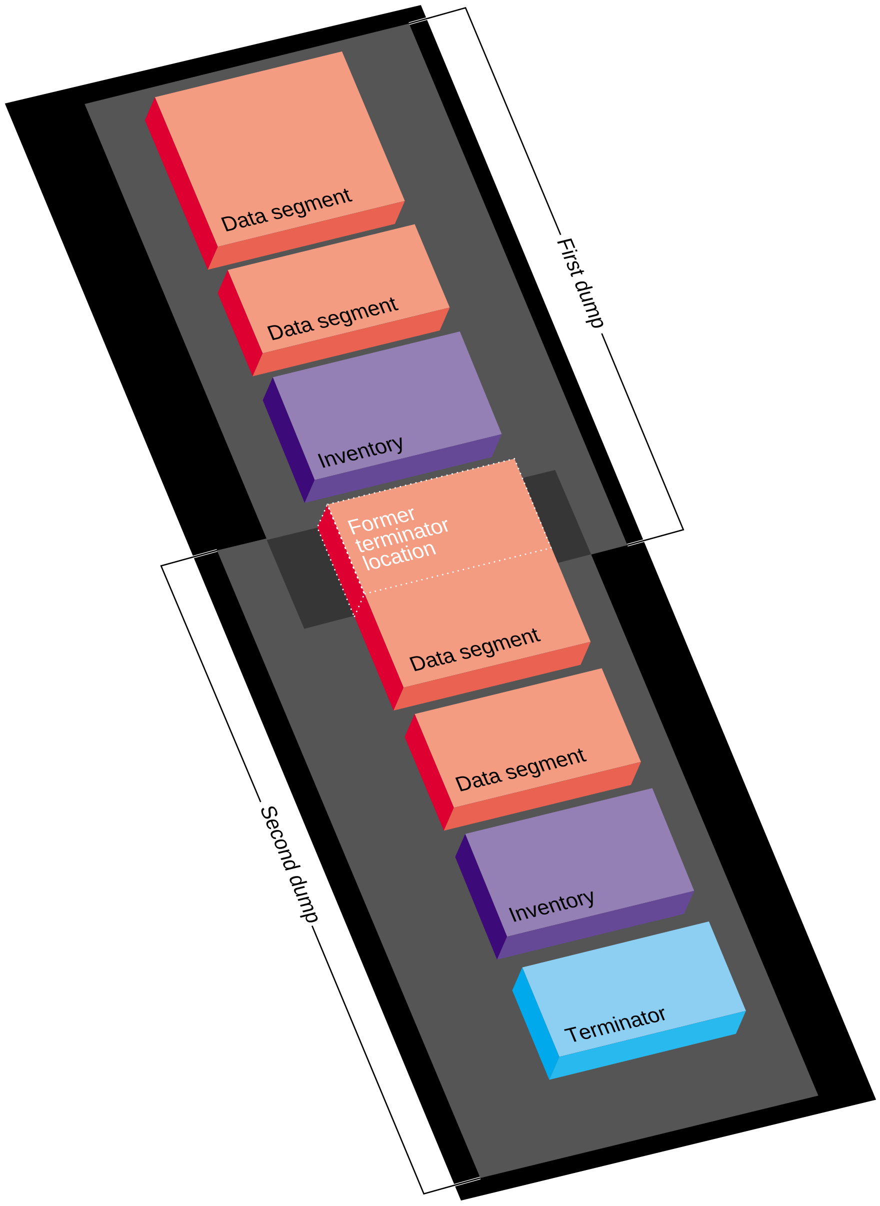

Figure 6-3 Multiple Dumps on Single Media Object

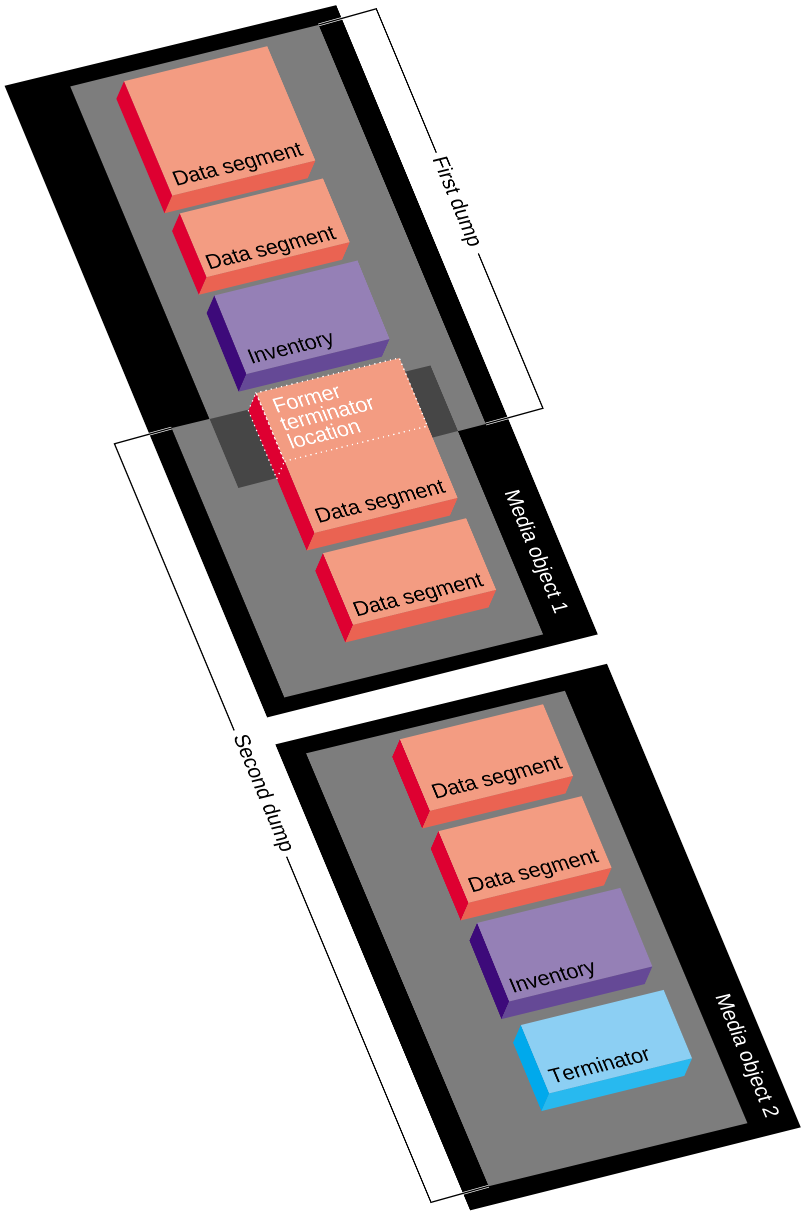

Figure 6-4 Multiple Dumps on Multiple Media Objects

Saving Data with xfsdump

This section discusses the following:

- "xfsdump Syntax" on page 43
- "Specifying Local Media" on page 44
- "Specifying a Remote Tape Drive" on page 45
- "Backing Up to a File" on page 47
- "Reusing Tapes" on page 47
- "Erasing Used Tapes" on page 48
- "About Incremental and Resumed Dumps" on page 48
- "Performing an Incremental Dump" on page 49
- "Performing a Resumed Dump" on page 50

xfsdump Syntax

You must be the superuser to use xfsdump. To display a summary of xfsdump syntax, use the -h option:

```
# xfsdump -h
xfsdump: version X.X
xfsdump: usage: xfsdump [ -b <blocksize> (with minimal rmt option) ]
                        [ -c <media change alert program> ]
                        [ -f <destination> ... ]
                        [ -h (help) ]
                        [ -l <level> ]
                        [ -m <force usage of minimal rmt> ]
                        [ -o <overwrite tape > ]
                        [ -p <seconds between progress reports> ]
                        [ -s <subtree> ... ]
                        [ -v <verbosity {silent, verbose, trace}> ]
                        [ -A (don’t dump extended file attributes) ]
                        [ -B <base dump session id> ]
                        [ -E (pre-erase media) ]
                        [ -F (don’t prompt) ]
                        [ -I (display dump inventory) ]
```

```
                                               [ -J (inhibit inventory update) ]
                                               [ -L <session label> ]
                                               [ -M <media label> ... ]
                                               [ -O <options file> ]
                                               [ -R (resume) ]
                                               [ -T (don’t timeout dialogs) ]
                                               [ -Y <I/O buffer ring length> ]
                                               [ - (stdout) ]
                                               [ <source (mntpnt|device)> ]
```

Specifying Local Media

You can use `xfsdump` to back up data to various media. For example, you can dump data to a tape or hard disk. The drive containing the media object may be connected to the local system or accessible over the network.

Following is an example of a level–0 dump to a local tape drive.

Note: The dump level does not need to be specified for a level–0 dump. For a discussion of dump levels, see "About Incremental and Resumed Dumps" on page 48.

```
# xfsdump -f /dev/tape -L testers_11_21_94 -M test_1 /disk2
xfsdump: version 2.0 - type ^C for status and control
xfsdump: level 0 dump of cumulus:/disk2
xfsdump: dump date: Wed Oct 25 16:19:13 1995
xfsdump: session id: d2a6123b-b21d-1001-8938-08006906dc5c
xfsdump: session label: ‘‘testers_11_21_94’’
xfsdump: ino map phase 1: skipping (no subtrees specified)
xfsdump: ino map phase 2: constructing initial dump list
xfsdump: ino map phase 3: skipping (no pruning necessary)
xfsdump: ino map phase 4: skipping (size estimated in phase 2)
xfsdump: ino map phase 5: skipping (only one dump stream)
xfsdump: ino map construction complete
xfsdump: preparing drive
xfsdump: creating dump session media file 0 (media 0, file 0)
xfsdump: dumping ino map
xfsdump: dumping directories
xfsdump: dumping non-directory files
xfsdump: ending media file
xfsdump: media file size 16777216 bytes
```

```
xfsdump: dumping session inventory
xfsdump: beginning inventory media file
xfsdump: media file 1 (media 0, file 1)
xfsdump: ending inventory media file
xfsdump: inventory media file size 4194304 bytes
xfsdump: writing stream terminator
xfsdump: beginning media stream terminator
xfsdump: media file 2 (media 0, file 2)
xfsdump: ending media stream terminator
xfsdump: media stream terminator size 2097152 bytes
xfsdump: I/O metrics: 3 by 2MB ring; 14/22 (64%) records streamed; 145889B/s
xfsdump: dump complete: 141 seconds elapsed
```

In this case, a session label (`-L` option) and a media label (`-M` option) are supplied, and the entire filesystem is dumped. Since no verbosity option is supplied, the default of `verbose` is used, resulting in the detailed screen output. The dump inventory is updated with the record of this backup because the `-J` option is not specified.

Following is an example of a backup of a subdirectory of a filesystem. In the following example, the verbosity is set to `silent`, and the dump inventory is not updated (`-J` option):

```
# xfsdump -f /dev/tape -v silent -J -s people/fred /usr
```

The subdirectory backed up (`/usr/people/fred`) was specified relative to the filesystem, so the specification did not include the name of the filesystem (in this case, `/usr`). Because `/usr` may be a very large filesystem and the `-v silent` option was used, this could take a long time during which there would be no screen output.

Specifying a Remote Tape Drive

To back up data to a remote tape drive, use the standard remote system syntax, specifying the system (by hostname if supported by a name server or IP address if not) followed by a colon (`:`), then the pathname of the special file.

Note: For remote backups, use the variable block size tape device if the device supports variable block size operation; otherwise, use the fixed block size device. For more information, see `intro`(7) .

The following example shows a subtree backup to a remote tape device:

```
# xfsdump -f magnolia:/dev/tape -L mag_10-95 -s engr /disk2
xfsdump: version 2.0 - type ^C for status and control
xfsdump: level 0 dump of cumulus:/disk2
xfsdump: dump date: Wed Oct 25 16:27:39 1995
xfsdump: session id: d2a6124b-b21d-1001-8938-08006906dc5c
xfsdump: session label: ‘‘mag_10-95’’
xfsdump: ino map phase 1: parsing subtree selections
xfsdump: ino map phase 2: constructing initial dump list
xfsdump: ino map phase 3: pruning unneeded subtrees
xfsdump: ino map phase 4: estimating dump size
xfsdump: ino map phase 5: skipping (only one dump stream)
xfsdump: ino map construction complete
xfsdump: preparing drive
xfsdump: positioned at media file 0: dump 0, stream 0
xfsdump: positioned at media file 1: dump 0, stream 0
xfsdump: positioned at media file 2: dump 0, stream 0
xfsdump: stream terminator found
xfsdump: creating dump session media file 0 (media 0, file 2)
xfsdump: dumping ino map
xfsdump: dumping directories
xfsdump: dumping non-directory files
xfsdump: ending media file
xfsdump: media file size 6291456 bytes
xfsdump: dumping session inventory
xfsdump: beginning inventory media file
xfsdump: media file 1 (media 0, file 3)
xfsdump: ending inventory media file
xfsdump: inventory media file size 4194304 bytes
xfsdump: writing stream terminator
xfsdump: beginning media stream terminator
xfsdump: media file 2 (media 0, file 4)
xfsdump: ending media stream terminator
xfsdump: media stream terminator size 2097152 bytes
xfsdump: I/O metrics: 3 by 2MB ring; 12/22 (55%) records streamed; 99864B/s
xfsdump: dump complete: 149 seconds elapsed
```

In this case, `/disk2/engr` is backed up to the variable block size tape device on the remote system `magnolia`. Existing dumps on the tape mounted on magnolia were skipped before recording the new data.

Note: The superuser account on the local system must be able to `rsh` to the remote system without a password. For more information, see `hosts.equiv`(4) .

Backing Up to a File

You can back up data to a file instead of a device. In the following example, a file (`Makefile`) and a directory (`Source`) are backed up to a dump file (`monday_backup`) in `/usr/tmp` on the local system:

```
# xfsdump -f /usr/tmp/monday_backup -v silent -J -s \
people/fred/Makefile -s people/fred/Source /usr
```

You may also dump to a file on a remote system, but the file must be in the remote system's `/dev` directory. For example, the following command backs up the `/usr/people/fred` subdirectory on the local system to the regular file `/dev/fred_mon_12-2` on the remote system `theduke`:

```
# xfsdump -f theduke:/dev/fred_mon_12-2 -s people/fred /usr
```

Alternatively, you could dump to any remote file if that file is on an NFS-mounted filesystem. In any case, permission settings on the remote system must allow you to write to the file.

For information on using the standard input and standard output capabilities of `xfsdump` and `xfsrestore` to pipe data between filesystems or across the network, see "Using `xfsdump` and `xfsrestore` to Copy Filesystems" on page 66.

Reusing Tapes

When you use a new tape as the media object of a dump session, `xfsdump` begins writing dump data at the beginning of the tape without prompting. If the tape already has dump data on it, `xfsdump` begins writing data after the last dump stream, again without prompting.

If, however, the tape contains data that is not from a dump session, `xfsdump` prompts you before continuing:

```
# xfsdump -f /dev/tape /test
xfsdump: version X.X - type ^C for status and control
xfsdump: dump date: Fri Dec 2 11:25:19 1994
xfsdump: level 0 dump
xfsdump: session id: d23cc072-b21d-1001-8f97-080069068eeb
xfsdump: preparing tape drive
xfsdump: this tape contains data that is not part of an XFS dump
xfsdump: do you want to overwrite this tape?
type y to overwrite, n to change tapes or abort (y/n):
```

You must answer `y` if you want to continue with the dump session, or `n` to quit. If you answer `y`, the dump session resumes and the tape is overwritten. If you do not respond to the prompt, the session eventually times out.

Note: This means that an automatic backup, for example one initiated by a `crontab` entry, will not succeed unless you specified the `-F` option with the `xfsdump` command, which forces it to overwrite the tape rather than prompt for approval.

Erasing Used Tapes

Erase preexisting data on tapes with the `mt erase` command. Make sure the tape is not write-protected. For example, to prepare a used tape in the local default tape drive, enter:

```
# mt -f /dev/tape erase
```

Caution: This erases all data on the tape, including any dump sessions

The tape can now be used by `xfsdump` without prompting for approval.

About Incremental and Resumed Dumps

Incremental dumps are a way of backing up less data at a time but still preserving current versions of all your backed-up files, directories, and so on. Incremental backups are organized numerically by levels from 0 through 9. A level-0 dump always backs up the complete filesystem. A dump level of any other number backs up all files that have changed since a dump with a lower dump level number.

For example, if you perform a level–2 backup on a filesystem one day and your next dump is a level–3 backup, only those files that have changed since the level–2 backup are dumped with the level–3 backup. In this case, the level–2 backup is called the *base dump* for the level–3 backup. The base dump is the most recent backup of that filesystem with a lower dump level number.

Resumed dumps work in much the same way. When a dump is resumed after it has been interrupted, the remaining files that had been scheduled to be backed up during the interrupted dump session are backed up, and any files that changed during the interruption are also backed up.

Note: You must restore an interrupted dump as if it is an incremental dump (see"Performing Cumulative Restores with `xfsrestore`" on page 60).

Performing an Incremental Dump

In the following example, a level-0 dump is the first backup written to a new tape:

```
# xfsdump -f /dev/tape -l 0 -M Jun_94 -L week_1 -v silent /usr
```

A week later, a level–1 dump of the filesystem is performed on the same tape:

```
# xfsdump -f /dev/tape -l 1 -L week_2 /usr
```

The tape is forwarded past the existing dump data and the new data from the level 1 dump is written after it. (Note that it is not necessary to specify the media label for each successive dump on a media object.)

A week later, a level 2 dump is taken and so on, for the four weeks of a month in this example, the fourth week being a level 3 dump (up to nine dump levels are supported):

```
# xfsdump -f /dev/tape -l 2 -L week_3 /usr
```

For information on the proper procedure for restoring incremental dumps, see"Performing Cumulative Restores with `xfsrestore`" on page 60.

Performing a Resumed Dump

You can interrupt a dump session and resume it later. To interrupt a dump session, type the interrupt character (typically **<CTRL-C>**). You receive a list of options that allow you to interrupt the session, change verbosity level, or resume the session.

In the following example, `xfsdump` is interrupted after dumping approximately 37% of a filesystem:

```
# xfsdump -f /dev/tape -M march95 -L week_1 -v silent /disk2
========================= status and control dialog ==========================
status at 16:49:16: 378/910 files dumped, 37.8% complete, 32 seconds elapsed
please select one of the following operations
1: interrupt this session
2: change verbosity
3: display metrics
4: other controls
5: continue (default) (timeout in 60 sec)
 -> 1
please confirm
1: interrupt this session
2: continue (default) (timeout in 60 sec)
 -> 1
interrupt request accepted
--------------------------------- end dialog ---------------------------------
xfsdump: initiating session interrupt
xfsdump: dump interrupted prior to ino 1053172 offset 0
```

You can later continue the dump by including the`-R` option and a different session label:

```
# xfsdump -f /dev/tape -R -L week_1.contd -v silent /disk2p
```

Any files that were not backed up before the interruption, and any file changes that were made during the interruption, are backed up after the dump is resumed.

Note: Use of the `-R` option requires that the dump was made with a dump inventory taken, that is, the `-J` option was not used with `xfsdump`.

Examining xfsdump Archives

This section describes how to use the xfsdump command to view an xfsdump inventory.

The xfsdump inventory is maintained in the directory /var/xfsdump created by xfsdump. You can view the dump inventory at any time with the xfsdump -I command. With no other arguments, xfsdump -I displays the entire dump inventory. (The xfsdump -I command does not require root privileges.)

The following output presents a section of a dump inventory:

```
# xfsdump -I | more
file system 0:
        fs id:          d23cb450-b21d-1001-8f97-080069068eeb
        session 0:
                mount point:    magnolia.abc.xyz.com:/test
                device:         magnolia.abc.xyz.com:/dev/rdsk/dks0d3s2
                time:           Mon Nov 28 11:44:04 1994
                session label:  ""
                session id:     d23cbf44-b21d-1001-8f97-080069068eeb
                level:          0
                resumed:        NO
                subtree:        NO
                streams:        1
                stream 0:
                        pathname:       /dev/tape
                        start:          ino 4121 offset 0
                        end:            ino 0 offset 0
                        interrupted:    YES
                        media files:    2
                        media file 0:
                                mfile index:    0
---more---
```

The dump inventory records are presented sequentially and are indented to illustrate the hierarchical order of the dump information.

You can view a subset of the dump inventory by specifying the level of depth (1, 2, or 3) that you want to view. For example, specifying depth=2 filters out a lot of the specific dump information, as you can see by comparing the previous output with the following:

```
# xfsdump -I depth=2
file system 0:
        fs id:          d23cb450-b21d-1001-8f97-080069068eeb
        session 0:
                mount point:    magnolia.abc.xyz.com:/test
                device:         magnolia.abc.xyz.com:/dev/rdsk/dks0d3s2
                time:           Mon Nov 28 11:44:04 1994
                session label:  ""
                session id:     d23cbf44-b21d-1001-8f97-080069068eeb
                level:          0
                resumed:        NO
                subtree:        NO
                streams:        1
        session 1:
                mount point:    magnolia.abc.xyz.com:/test
                device:         magnolia.abc.xyz.com:/dev/rdsk/dks0d3s2
               ...
```

You can also view a filesystem-specific inventory by specifying the filesystem mount point with the `mnt` option. The following output shows an example of a dump inventory display in which the `depth` is set to `1`, and only a single filesystem is displayed:

```
# xfsdump -I depth=1,mnt=magnolia.abc.xyz.com:/test
filesystem 0:
        fs id:          d23cb450-b21d-1001-8f97-080069068eeb
```

You can also look at a list of contents on the dump media itself by using the `-t` option with `xfsrestore`. See "Displaying the Contents of the Dump Media with `xfsrestore`" on page 55.

About `xfsrestore`

This section discusses the following:

- "`xfsrestore` Syntax" on page 53
- "Displaying the Contents of the Dump Media with `xfsrestore`" on page 55
- "Performing Simple Restores with `xfsrestore`" on page 56
- "Restoring Individual Files with `xfsrestore`" on page 58

- "Performing Network Restores with xfsrestore" on page 58
- "Performing Interactive Restores with xfsrestore" on page 59
- "Performing Cumulative Restores with xfsrestore" on page 60
- "Interrupting xfsrestore" on page 64
- "About the housekeeping and orphanage Directories" on page 65

For more information, see the xfsrestore(8) man page.

xfsrestore Syntax

You can use the xfsrestore command to view and extract data from the dump data created by xfsdump.

You can get a summary of xfsrestore syntax with the --h option:

```
# xfsrestore -h
xfsrestore: version X.X
xfsrestore: usage: xfsrestore [ -a <alt. workspace dir> ... ]
                     [ -e (don't overwrite existing files) ]
                     [ -f <source> ... ]
                     [ -h (help) ]
                     [ -i (interactive) ]
                     [ -n <file> (restore only if newer than) ]
                     [ -o (restore owner/group even if not root) ]
                     [ -p <seconds between progress reports> ]
                     [ -r (cumulative restore) ]
                     [ -s <subtree> ... ]
                     [ -t (contents only) ]
                     [ -v <verbosity {silent, verbose, trace}> ]
                     [ -A (don't restore extended file attributes) ]
                     [ -C (check tape record checksums) ]
                     [ -D (restore DMAPI event settings) ]
                     [ -E (don't overwrite if changed) ]
                     [ -F (don't prompt) ]
                     [ -I (display dump inventory) ]
                     [ -J (inhibit inventory update) ]
                     [ -L <session label> ]
                     [ -N (timestamp messages) ]
                     [ -O <options file> ]
                     [ -P (pin down I/O buffers) ]
                     [ -Q (force interrupted session completion) ]
                     [ -R (resume) ]
                     [ -S <session id> ]
                     [ -T (don’t timeout dialogs) ]
                     [ -U (unload media when change needed) ]
                     [ -V (show subsystem in messages) ]
                     [ -W (show verbosity in messages) ]
                     [ -X <excluded subtree> ... ]
                     [ -Y <I/O buffer ring length> ]
                     [ -Z (miniroot restrictions) ]
                     [ - (stdin) ]
                     [ <destination> ]
```

Use `xfsrestore` to restore data backed up with `xfsdump`. You can restore files, subdirectories, and filesystems regardless of the way they were backed up. For example, if you back up an entire filesystem in a single dump, you can select individual files and subdirectories from within that filesystem to restore.

You can use `xfsrestore` interactively or noninteractively. With interactive mode, you can peruse the filesystem or files backed up, selecting those you want to restore. In noninteractive operation, a single command line can restore selected files and subdirectories, or an entire filesystem. You can restore data to its original filesystem location or any other location in an XFS filesystem.

By using successive invocations of `xfsrestore`, you can restore incremental dumps on a base dump. This restores data in the same sequence it was dumped.

Displaying the Contents of the Dump Media with `xfsrestore`

To list the contents of the dump tape currently in the local tape drive, type:

```
# xfsrestore -f /dev/tape -t -v silent | more
xfsrestore: dump session found
xfsrestore: session label: "week_1"
xfsrestore: session id: d23cbcb4-b21d-1001-8f97-080069068eeb
xfsrestore: no media label
xfsrestore: media id: d23cbcb5-b21d-1001-8f97-080069068eeb
do you want to select this dump? (y/n): y
selected
one
A/five
people/fred/TOC
people/fred/ch3.doc
people/fred/ch3TOC.doc
people/fred/questions
A/four
people/fred/script_0
people/fred/script_1
people/fred/script_2
people/fred/script_3
people/fred/sub1/TOC
people/fred/sub1/ch3.doc
people/fred/sub1/ch3TOC.doc
people/fred/sub1/questions
people/fred/sub1/script_0
people/fred/sub1/script_1
people/fred/sub1/script_2
people/fred/sub1/script_3
people/fred/sub1/xdump1.doc
people/fred/sub1/xdump1.doc.backup
people/fred/sub1/xfsdump.doc
people/fred/sub1/xfsdump.doc.auto
people/fred/sub1/sub2/TOC
---more---
```

Performing Simple Restores with `xfsrestore`

A simple restore is a non-cumulative restore (for information on restoring incremental dumps, refer to "Performing Cumulative Restores with `xfsrestore`" on page 60). An example of a simple, noninteractive use of `xfsrestore` is:

```
# xfsrestore -f /dev/tape /disk2
xfsrestore: version 2.0 - type ^C for status and control
xfsrestore: searching media for dump
xfsrestore: preparing drive
xfsrestore: examining media file 0

 =========================== dump selection dialog ============================

the following dump has been found on drive 0

hostname: cumulus
mount point: /disk2
volume: /dev/rdsk/dks0d2s0
session time: Wed Oct 25 16:59:00 1995
level: 0
session label: ‘‘tape1’’
media label: ‘‘media1’’
file system id: d2a602fc-b21d-1001-8938-08006906dc5c
session id: d2a61284-b21d-1001-8938-08006906dc5c
media id: d2a61285-b21d-1001-8938-08006906dc5c

restore this dump?
1: skip
2: restore (default)
 -> 2
this dump selected for restoral

 --------------------------------- end dialog ---------------------------------

xfsrestore: using online session inventory
xfsrestore: searching media for directory dump
xfsrestore: reading directories
xfsrestore: directory post-processing
xfsrestore: restoring non-directory files
xfsrestore: I/O metrics: 3 by 2MB ring; 9/13 (69%) records streamed; 204600B/s
xfsrestore: restore complete: 104 seconds elapsed
```

In this case, xfsrestore went to the first dump on the tape and asked if this was the dump to restore. If you had entered 1 for “skip,” xfsrestore would have proceeded to the next dump on the tape (if there was one) and asked if this was the dump you wanted to restore.

You can request a specific dump if you used xfsdump with a session label. For example:

```
# xfsrestore -f /dev/tape -L Wed_11_23 /usr
xfsrestore: version X.X - type ^C for status and control
xfsrestore: preparing tape drive
xfsrestore: dump session found
xfsrestore: advancing tape to next media file
xfsrestore: dump session found
xfsrestore: restore of level 0 dump of magnolia.abc.xyz.com:/usr created Wed Nov 23 11:17:54 1994
xfsrestore: beginning media file
xfsrestore: reading ino map
xfsrestore: initializing the map tree
xfsrestore: reading the directory hierarchy
xfsrestore: restoring non-directory files
xfsrestore: ending media file
xfsrestore: restoring directory attributes
xfsrestore: restore complete: 200 seconds elapsed
```

In this way you recover a dump with a single command line and do not have to answer y or n to the prompts asking you if the dump session found is the correct one. To be even more exact, use the -S option and specify the unique session ID of the particular dump session:

```
# xfsrestore -f /dev/tape -S \ d23cbf47-b21d-1001-8f97-080069068eeb /usr2/tmp
xfsrestore: version X.X - type ^C for status and control
xfsrestore: preparing tape drive
xfsrestore: dump session found
xfsrestore: advancing tape to next media file
xfsrestore: advancing tape to next media file
xfsrestore: dump session found
xfsrestore: restore of level 0 dump of magnolia.abc.xyz.com:/test resumed Mon Nov 28 11:50:41 1994
xfsrestore: beginning media file
xfsrestore: media file 0 (media 0, file 2)
xfsrestore: reading ino map
xfsrestore: initializing the map tree
xfsrestore: reading the directory hierarchy
```

```
xfsrestore: restoring non-directory files
xfsrestore: ending media file
xfsrestore: restoring directory attributes
xfsrestore: restore complete: 229 seconds elapsed
```

You can find the session ID by viewing the dump inventory (see "Examining `xfsdump` Archives" on page 51). Session labels might be duplicated, but session IDs never are.

Restoring Individual Files with `xfsrestore`

On the `xfsrestore` command line, you can specify an individual file or subdirectory to restore. In this example, the file `people/fred/notes` is restored and placed in the `/usr/tmp` directory (that is, the file is restored in `/usr/tmp/people/fred/notes`):

```
# xfsrestore -f /dev/tape -L week_1 -s people/fred/notes /usr/tmp
```

You can also restore a file “in place” that is, restore it directly to where it came from in the original backup.

Note: However, if you do not use the `-e`, `-E`, or `-n` option, you will overwrite any existing files of the same name.

In the following example, the subdirectory `people/fred` is restored in the destination `/usr`, which overwrites any files and subdirectories in `/usr/people/fred` with the data on the dump tape:

```
# xfsrestore -f /dev/tape -L week_1 -s people/fred /usr
```

Performing Network Restores with `xfsrestore`

You can use standard network references to specify devices and files on the network. For example, to use the tape drive on a network host named `magnolia` as the source for a restore, you can use the following command:

```
# xfsrestore -f magnolia:/dev/tape -L 120694u2 /usr2
xfsrestore: version X.X - type ^C for status and control
xfsrestore: preparing tape drive
xfsrestore: dump session found
xfsrestore: advancing tape to next media file
xfsrestore: dump session found
```

```
xfsrestore: restore of level 0 dump of magnolia.abc.xyz.com:/usr2 created Tue Dec 6 10:55:17 1994
xfsrestore: beginning media file
xfsrestore: media file 0 (media 0, file 1)
xfsrestore: reading ino map
xfsrestore: initializing the map tree
xfsrestore: reading the directory hierarchy
xfsrestore: restoring non-directory files
xfsrestore: ending media file
xfsrestore: restoring directory attributes
xfsrestore: restore complete: 203 seconds elapsed
```

In this case, the dump data is extracted from the tape on `magnolia`, and the destination is the directory `/usr2` on the local system. For an example of using the standard input option of `xfsrestore`, see "Using `xfsdump` and `xfsrestore` to Copy Filesystems" on page 66.

Performing Interactive Restores with `xfsrestore`

Use the `-i` option of `xfsrestore` to perform interactive file restoration. With interactive restoration, you can use the commands `ls`, `pwd`, and `cd` to peruse the filesystem, and the `add` and `delete` commands to create a list of files and subdirectories you want to restore. Then you can enter the `extract` command to restore the files, or `quit` to exit the interactive restore session without restoring files. (The use of wildcards is not allowed with these commands.)

Note: Interactive restore is not allowed when the `xfsrestore` source is standard input (`stdin`).

The following screen output shows an example of a simple interactive restoration.

```
# xfsrestore -f /dev/tape -i -v silent .
xfsrestore: dump session found
xfsrestore: no session label
xfsrestore: session id:    d23cbeda-b21d-1001-8f97-080069068eeb
xfsrestore: no media label
xfsrestore: media id:      d23cbedb-b21d-1001-8f97-080069068eeb
do you want to select this dump? (y/n): y
selected

 --- interactive subtree selection dialog ---
```

```
the following commands are available:
        pwd
        ls [ { <name>, ".." } ]
        cd [ { <name>, ".." } ]
        add [ <name> ]
        delete [ <name> ]
        extract
        quit
        help
 -> ls
            4122 people/
            4130 two
            4126 A/
            4121 one
 -> add two
 -> cd people
 -> ls
            4124 fred/
 -> add fred
 -> ls
    *       4124 fred/
 -> extract

 ---------------- end dialog ----------------
```

In the interactive restore session above, the subdirectory `people/fred` and the file `two` were restored relative to the current working directory (“`.`”). An asterisk (`*`) in your `ls` output indicates your selections.

Performing Cumulative Restores with `xfsrestore`

Cumulative restores sequentially restore incremental dumps to re-create filesystems and are also used to restore interrupted dumps. To perform a cumulative restore of a filesystem, begin with the media object that contains the base-level dump and recover it first, then recover the incremental dump with the next higher dump level number, then the next, and so on. Use the `-r` option to inform `xfsrestore` that you are performing a cumulative recovery.

In the following example, the level–0 base dump and succeeding higher-level dumps are on `/dev/tape`. First the level-0 dump is restored, then each higher-level dump in succession:

```
# /usr/tmp/xfsrestore -f /dev/tape -r -v silent .

 =========================== dump selection dialog ============================

the following dump has been found on drive 0

hostname: cumulus
mount point: /disk2
volume: /dev/rdsk/dks0d2s0
session time: Wed Oct 25 14:37:47 1995
level: 0
session label: "week_1"
media label: "Jun_94"
file system id: d2a602fc-b21d-1001-8938-08006906dc5c
session id: d2a60b26-b21d-1001-8938-08006906dc5c
media id: d2a60b27-b21d-1001-8938-08006906dc5c

restore this dump?
1: skip
2: restore (default)
 -> Enter
this dump selected for restoral

 --------------------------------- end dialog ---------------------------------

#
```

Next, enter the same command again. The program goes to the next dump and again you select the default:

```
# xfsrestore -f /dev/tape -r -v silent .

 =========================== dump selection dialog ============================

the following dump has been found on drive 0

hostname: cumulus
mount point: /disk2
```

```
volume: /dev/rdsk/dks0d2s0
session time: Wed Oct 25 14:40:54 1995
level: 1
session label: "week_2"
media label: "Jun_94"
file system id: d2a602fc-b21d-1001-8938-08006906dc5c
session id: d2a60b2b-b21d-1001-8938-08006906dc5c
media id: d2a60b27-b21d-1001-8938-08006906dc5c

restore this dump?
1: skip
2: restore (default)
 -> Enter
this dump selected for restoral

 --------------------------------- end dialog ---------------------------------
#
```

You then repeat this process until you have recovered the entire sequence of incremental dumps. The full and latest copy of the filesystem will then have been restored. In this case, it is restored relative to "`.`", that is, in the directory you are in when the sequence of `xfsrestore` commands is issued.

Restore an interrupted dump just as if it were an incremental dump. Use the `-r` option to inform `xfsrestore` that you are performing an incremental restore, and answer `y` and `n` appropriately to select the proper "increments" to restore (see "Performing Cumulative Restores with `xfsrestore`" on page 60).

Note: If you try to restore an interrupted dump as if it were a non-interrupted, non-incremental dump, the portion of the dump that occurred before the interruption is restored, but not the remainder of the dump. You can determine if a dump is an interrupted dump by looking in the online inventory.

Following is an example of a dump inventory showing an interrupted dump session (the crucial fields are in bold type):

```
# xfsdump -I depth=3,mobjlabel=AugTape,mnt=indy4.xyz.com:/usr
file system 0:
        fs id:          d23cb450-b21d-1001-8f97-080069068eeb
        session 0:
                mount point:    indy4.xyz.com.com:/usr
                device:         indy4.xyz.com.com:/dev/rdsk/dks0d3s2
                time:           Tue Dec  6 15:01:26 1994
                session label:  "180894usr"
                session id:     d23cc0c3-b21d-1001-8f97-080069068eeb
                level:          0
                resumed:        NO
                subtree:        NO
                streams:        1
                stream 0:
                        pathname:       /dev/tape
                        start:          ino 4121 offset 0
                        end:            ino 0 offset 0
                        interrupted:    YES
                        media files:    2
        session 1:
                mount point:    indy4.xyz.com.com:/usr
                device:         indy4.xyz.com.com:/dev/rdsk/dks0d3s2
                time:           Tue Dec  6 15:48:37 1994
                session label:  "Resumed180894usr"
                session id:     d23cc0cc-b21d-1001-8f97-080069068eeb
                level:          0
                resumed:        YES
                subtree:        NO
                streams:        1
                stream 0:
                        pathname:       /dev/tape
                        start:          ino 4121 offset 0
                        end:            ino 0 offset 0
                        interrupted:    NO
                        media files:    2
...
```

From this it can be determined that session 0 was interrupted and then resumed and completed in session 1.

To restore the interrupted dump session in the example above, use the following sequence of commands:

```
# xfsrestore -f /dev/tape -r -L 180894usr .
# xfsrestore -f /dev/tape -r -L Resumed180894usr .
```

This restores the entire `/usr` backup relative to the current directory. (You should remove the `housekeeping` directory from the destination directory when you are finished.)

Interrupting `xfsrestore`

In a manner similar to `xfsdump` interruptions, you can interrupt an `xfsrestore` session. This allows you to interrupt a restore session and then resume it later. To interrupt a restore session, type the interrupt character (typically **<CTRL-C>**). You receive a list of options, which include interrupting the session or continuing.

```
# xfsrestore -f /dev/tape -v silent /disk2

 =========================== dump selection dialog ============================

the following dump has been found on drive 0

hostname: cumulus
mount point: /disk2
volume: /dev/rdsk/dks0d2s0
session time: Wed Oct 25 17:20:16 1995
level: 0
session label: "week1"
media label: "newtape"
file system id: d2a602fc-b21d-1001-8938-08006906dc5c
session id: d2a6129e-b21d-1001-8938-08006906dc5c
media id: d2a6129f-b21d-1001-8938-08006906dc5c

restore this dump?
1: skip
2: restore (default)
 -> 2
```

```
this dump selected for restoral

 --------------------------------- end dialog ---------------------------------


 ======================== status and control dialog ==========================

status at 17:23:52: 131/910 files restored, 14.4% complete, 42 seconds elapsed

please select one of the following operations
1: interrupt this session
2: change verbosity
3: display metrics
4: other controls
5: continue (default) (timeout in 60 sec)
 -> 1

please confirm
1: interrupt this session
2: continue (default) (timeout in 60 sec)
 -> 1
interrupt request accepted

 --------------------------------- end dialog ---------------------------------

xfsrestore: initiating session interrupt
```

Resume the `xfsrestore` session with the `--R` option:

```
# xfsrestore -f /dev/tape -R -v silent /disk2
```

Data recovery continues from the point of the interruption.

About the housekeeping and orphanage Directories

The `xfsrestore` utility can create two subdirectories in the destination called `housekeeping` and `orphanage`:

- `housekeeping` is a temporary directory used during cumulative recovery to pass information from one invocation of `xfsrestore` to the next. It must not be removed during the process of performing the cumulative recovery but should be removed after the cumulative recovery is completed.

- `orphanage` is created if a file or subdirectory is restored that is not referenced in the filesystem structure of the dump. For example, if you dump a very active filesystem, it is possible for new files to be in the non-directory portion of the dump, yet none of the directories dumped reference that file. A warning message is displayed, and the file is placed in the `orphanage` directory, named with its original inode number and generation count (for example, `123479.14`).

Using `xfsdump` and `xfsrestore` to Copy Filesystems

You can use `xfsdump` and `xfsrestore` to pipe data across filesystems or across the network with a single command line. By piping `xfsdump` standard output to `xfsrestore` standard input you create an exact copy of a filesystem.

For example, to make a copy of `/usr/people/fred` in the `/usr2` directory, enter:

```
# xfsdump -J -s people/fred - /usr | xfsrestore - /usr2
```

To copy `/usr/people/fred` to the network host `magnolia`'s `/usr/tmp` directory:

```
# xfsdump -J -s people/fred - /usr | rsh magnolia \
xfsrestore - /usr/tmp
```

This creates the directory `/usr/tmp/people/fred` on `magnolia`.

Note: The superuser account on the local system must be able to `rsh` to the remote system without a password. For more information, see `hosts.equiv`(4).

Chapter 7

Enhanced XFS Extensions

This chapter discusses the following enhanced XFS extensions:

- "`agskip` Mount Option for Allocation Group Specification" on page 67
- "`ibound` Mount Option for SSD Media" on page 67

`agskip` Mount Option for Allocation Group Specification

The `agskip` mount option specifies the allocation group (AG) for a new file, relative to the last previously created file (that is, it has the opposite effect of the `rotorstep` system tunable parameter. see "`rotorstep`" on page 83). Using `agskip=`*agskipvalue* causes each new file to be placed in the AG *initialAG+agskipvalue*, where *initialAG* is the allocation group used for the previously created new file. For example, `agskip=3` means each new file thereafter will be allocated three AGs away from the AG used for the most recently created file.

Use the following formula to determine an appropriate *agskipvalue*:

`(`*number_of AGs* `/` *number_of_concats*`) + 1 =` *agskipvalue*

For example, if you have six AGs and two concats, you would use a value of 4:

```
(6/2) + 1 = 4
```

Note: The `agskip` mount option disables the `rotorstep` system tunable parameter.

`ibound` Mount Option for SSD Media

This section discusses the following:

- "`ibound` Overview" on page 68
- "`ibound` and Tuning XFS" on page 68
- "`ibound` and Kernel Messages" on page 69
- "`ibound` Examples" on page 70

ibound Overview

Different types of media are appropriate for different uses:

- Solid-state drive (SSD) media is appropriate for small latency-sensitive operations
- Rotating hard-disk drive (HDD) media is appropriate for larger bandwidth- and capacity-intensive operations

The `ibound` mount option specifies where the filesystem places the inodes, which lets you use SSD media for a filesystem's inodes and HDD media for the file data. In this case, you should create an XVM volume that concatenates a slice of SSD media with HDD media and then use the `ibound` mount option to restrict filesystem inode allocation to the fast SSD media at the beginning of the XVM volume.

If the `ibound` address falls within the middle of an XFS filesystem allocation group (AG), that AG is not used for inode allocation. Only the lower-numbered AGs are used for inode allocation. Should the `ibound` address fall within the first AG (AG 0), the value of `ibound` is changed to include the entire first AG.

Note: The `ibound` mount option implies `inode32` behavior and is therefore incompatible with the `inode64` mount option. Behavior of the `inode32` mount option is not affected.

To maximize performance of the filesystem with an SSD drive, you should use an external log. You can use a partition of the SSD media or a separate HDD media.

ibound and Tuning XFS

The space for user data is rotored among the AGs. This behavior may result in decreased I/O efficiency, especially for streaming workloads that create sequential files.

When configuring an XFS filesystem with the `ibound` mount option, you should take special care to use the appropriate disk media and `mkfs.xfs`(8) settings. Doing so will ensure that `xfsrestore`(8) and other file-creation workloads perform as expected.

When using the `ibound` or `inode32` mount options, you can also adjust the allocation behavior for newly created files by using the `rotorstep` system tunable kernel parameter (see "`rotorstep`" on page 83). Setting the value to `255` may provide more favorable behavior by allocating space for user data in 255 new files in an AG before moving to the next AG for allocating space for a new file in the next AG.

Do the following:

1. Configure the XVM volume so that the inode area described by `ibound` is on very fast disk that is at the beginning of the volume. SSD disk is ideal.
2. Use an external XFS log on very fast disk. SSD disk is ideal.
3. Set the `rotorstep` system tunable parameter to 255. See:
   - "Permanently Changing a Parameter" on page 78
   - "Temporarily Changing a Parameter" on page 78
4. Set the allocation group (AG) size using the `mkfs.xfs agsize` option to so that it is 1/8 the size of the SSD (meaning that 8 AGs can span the SSD).

   Note: The `ibound` mount option is not compatible with the `inode64` mount option. If you specify both options, the `mount`(8) command will ignore the first option specified.

   The configuration rules for filesystems using `ibound` may result in an XFS filesystem with thousands of AGs. With this many AGs, XFS will consume more CPU resources searching for free space in a nearly full filesystem. For best performance, ensure that the filesystem is less than ~90% full as reported by the `df`(1) command.

`ibound` and Kernel Messages

When the `ibound` mount option is used, the XFS kernel module will log an `INFO` log message, indicating the maximum possible inode identification number (which is different from the count of inodes). For example:

```
XFS: filesystem filesystem_name maximum new inode number is new_inode_number
```

Note: The maximum inode identification number may be used by SGI Support during troubleshooting to verify that inodes are in the correct area of the filesystem.

The new maximum inode identification number is not reflected in the `xvm show` output and is not the same as the value you specify for the `ibound` mount option.

If the `ibound` value you specify is smaller than the size of the first AG, the XFS kernel module will log a `WARN` message, indicating that it has changed the value to something appropriate. For example:

`XFS: filesystem` *filesystem_name* `ibound is too small, using` *corrected_inode_number*

`ibound` Examples

This section discusses the following:

- "Maximizing SSD Storage of Inodes for an SSD/HDD Filesystem" on page 70
- "Example Using a Value for `ibound` that is Too Small" on page 74
- "Determining the SSD Size Required for a Given Number of Inodes" on page 74

Maximizing SSD Storage of Inodes for an SSD/HDD Filesystem

This example describes how to create a filesystem using both SSD and HDD so that the SSD is used for storing as many inodes as possible. The volume is constructed so that the first 8 allocation groups (AGs) and external log are placed on the SSD. The external log is the maximum size of 1 GiB. The remainder of the volume is a two-disk stripe.

1. Partition disks `sdc`, `sdb`, and `sdd` using a GPT label and primary partition that starts at MB 34:

```
cxfsxe4:~ # parted /dev/sdc
GNU Parted 2.3
Using /dev/sdc
Welcome to GNU Parted! Type ’help’ to view a list of commands.
(parted) mklabel gpt
Warning: The existing disk label on /dev/sdc will be destroyed and all data on
this disk will be lost. Do you want to continue?
Yes/No? yes
(parted) unit s
(parted) mkpart primary xfs 34 -34
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? ignore
(parted) quit
Information: You may need to update /etc/fstab.
```

```
cxfsxe4:~ # parted /dev/sdb
GNU Parted 2.3
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel gpt
Warning: The existing disk label on /dev/sdb will be destroyed and all data on
this disk will be lost. Do you want to continue?
Yes/No? yes
(parted) unit s
(parted) mkpart primary xfs 34 -34
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? ignore
(parted) quit
Information: You may need to update /etc/fstab.

cxfsxe4:~ # parted /dev/sdd
GNU Parted 2.3
Using /dev/sdd
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel gpt
Warning: The existing disk label on /dev/sdd will be destroyed and all data on
this disk will be lost. Do you want to continue?
Yes/No? yes
(parted) unit s
(parted) mkpart primary xfs 34 -34
Warning: The resulting partition is not properly aligned for best performance.
Ignore/Cancel? ignore
(parted) quit
Information: You may need to update /etc/fstab.
```

2. Use the `xvm` command to show the unlabeled devices:

```
cxfsxe4:~ # xvm
xvm:local> show unlabeled
unlabeled/dev/pm/ATA-HDT722525DLA380---VDS41LT8CAA9RH          * *
unlabeled/dev/pm/ATA-INTEL_SSDSA2M080---CVPO006500CD080BGN          * *
unlabeled/dev/pm/ATA-ST3500631NS---9QG4F29A          * *
unlabeled/dev/pm/ATA-ST3500841AS---3PM1TSNN          * *
```

3. Assign the disks to XVM by using the `xvm label` command:

```
xvm:local> label -name ssd0 unlabeled/dev/pm/ATA-INTEL_SSDSA2M080---CVPO006500CD080BGN
ssd0
xvm:local> label -name disk0 unlabeled/dev/pm/ATA-ST3500631NS---9QG4F29A
disk0
xvm:local> label -name disk1 unlabeled/dev/pm/ATA-ST3500841AS---3PM1TSNN
disk1
```

4. Construct a volume named `hybridvol` with a `data` subvolume that is a concatenation of SSD and HDD media and an external `log` subvolume:

```
xvm:local> slice -length 262144 phys/ssd0
</dev/lxvm/ssd0s0>  slice/ssd0s0
xvm:local> slice -start 262144 phys/ssd0
</dev/lxvm/ssd0s1>  slice/ssd0s1
xvm:local> slice -all phys/disk0
</dev/lxvm/disk0s0>  slice/disk0s0
xvm:local> slice -all phys/disk1
</dev/lxvm/disk1s0>  slice/disk1s0
xvm:local> subvolume -volname ssdlog -type log slice/ssd0s0
</dev/lxvm/ssdlog,log>  subvol/ssdlog/log
xvm:local> stripe -volname diskstripe -vename diskstripe slice/disk0s0 slice/disk1s0
</dev/lxvm/diskstripe>  stripe/diskstripe
xvm:local> concat -volname hybridvol -vename hybridconcat slice/ssd0s1 stripe/diskstripe
</dev/lxvm/hybridvol>  concat/hybridconcat
xvm:local> attach subvol/ssdlog/log vol/hybridvol
vol/hybridvol
xvm:local> delete -all vol/ssdlog
xvm:local> delete -all vol/diskstripe
xvm:local> show -top vol/hybridvol
vol/hybridvol                    0 online,accessible
    subvol/hybridvol/data    2109536416 online,accessible
        concat/hybridconcat      2109536416 online,accessible
            slice/ssd0s1              156022944 online,accessible
            stripe/diskstripe        1953513472 online,accessible
                slice/disk0s0             976756768 online,accessible
                slice/disk0s0             976756768 online,accessible
                slice/disk1s0             976756768 online,accessible
    subvol/hybridvol/log         262144 online,accessible
        slice/ssd0s0
```

```
xvm:local> quit
```

The above shows that the SSD slice is 156022944 sectors.

For more information about the `xvm` commands shown, see *XVM Volume Manager Administrator Guide* or the `xvm`(8) man page.

5. Determine the appropriate AG sector size for the SSD slice to be supplied to the `mkfs.xfs`(8) command, which must be a multiple of 8 (because there are eight sectors in a filesystem block):

   a. Divide the size of `slice/ssd0s1` (which is `156022944`) by the number of allocation groups (`8` in this case) and truncate the result to an integer value (resulting in `19502868`).

   b. Divide the result of step 5a by `8` (eight sectors per block) and truncate the result to an integer (resulting in `2437858`).

   c. Multiply the result of step 5b by `8`, resulting in `19502864`. This is the `agsize` value to be used in step 6 and step 7.

6. Make the filesystem, specifying the largest disk address (sector) allowed to be used for storing an inode (`19502864` in this case, as determined in step 5c) for the `agsize` value:

```
cxfsxe4:~ # mkfs.xfs -f -d agsize=19502864s -l logdev=/dev/lxvm/hybridvol_log -l size=128m /dev/lxvm/hybridvol
warning: unable to probe device topology for device /dev/lxvm/hybridvol
meta-data=/dev/lxvm/hybridvol    isize=256    agcount=109, agsize=2437858 blks
         =                       sectsz=512   attr=2, projid32bit=0
data     =                       bsize=4096   blocks=263692052, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =/dev/lxvm/hybridvol_log bsize=4096   blocks=32768, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

7. Mount the filesystem, supplying the size of `slice/ssd0s1` (which is `156022944`) for the `ibound` mount option:

```
cxfsxe4:~ # mount -o ibound=156022944,logdev=/dev/lxvm/hybridvol_log /dev/lxvm/hybridvol /mnt
```

8. Display the kernel messages to determine the new maximum inode identification number (which is not a value that you manually specify and is filesystem-dependent):

```
cxfsxe4:~ # dmesg | tail -3
XFS (xvm-46): XFS: filesystem xvm-46 maximum new inode number is 508767775
XFS (xvm-46): Mounting Filesystem
XFS (xvm-46): Ending clean mount
```

Example Using a Value for `ibound` that is Too Small

If you use a value for `ibound` that is smaller than the size of the first AG, the filesystem will determine an appropriate value to use instead. To illustrate this, carrying on from the example in "Maximizing SSD Storage of Inodes for an SSD/HDD Filesystem":

1. Unmount the filesystem:

```
cxfsxe4:~ # umount /mnt
```

2. Mount the filesystem with an `ibound` value that is obviously too small, such as `1`:

```
cxfsxe4:~ # mount -o ibound=1,logdev=/dev/lxvm/hybridvol_log /dev/lxvm/hybridvol /mnt
```

3. Display the kernel messages, which show that the improper value specified in the previous step is overridden:

```
 cxfsxe4:~ # dmesg | tail -4
XFS (xvm-46): XFS: filesystem xvm-46 ibound is too small, using 19502856
XFS (xvm-46): XFS: filesystem xvm-46 maximum new inode number is 39005727
XFS (xvm-46): Mounting Filesystem
XFS (xvm-46): Ending clean mount
```

Determining the SSD Size Required for a Given Number of Inodes

To determine the required SSD size, use the following guidelines: multiply the number of inodes by the inode size and add some overhead space for other metadata, such as file names and extended attributes:

(*number_of_inodes* X *inode_size*) + *overhead* = *SSD_size*

Note: The size of other metadata is highly variable and depends heavily upon how the filesystem is used. If there are large extended attributes in the filesystem or if there are long filenames, more overhead space will be required.

The default inode size is 256 bytes.

Appendix A

XFS System-Tunable Kernel Parameters

This appendix discusses the following:

- "Overview of the XFS System-Tunable Kernel Parameters" on page 77
- "Parameter Types" on page 79

Overview of the XFS System-Tunable Kernel Parameters

This section discusses the following:

- "Using Appropriate Parameter Settings" on page 77
- "Time Unit of Measure" on page 77
- "Prefix" on page 78
- "Permanently Changing a Parameter" on page 78
- "Temporarily Changing a Parameter" on page 78
- "Querying a Current Parameter Setting" on page 79

Using Appropriate Parameter Settings

All XFS parameters are dynamic. Before changing any parameter, you should understand the ramifications of doing so on your system. You should change debugging parameters only at the recommendation of SGI Support.

The values of these parameters vary in different releases of the product. When upgrading the product, consult SGI Support to determine whether any changes made to the parameters in this chapter should be carried forward. Setting these parameters incorrectly may render the system unstable or otherwise unusable.

Time Unit of Measure

Times are measured in centisecs (100ths of a second).

Prefix

Each of the parameters uses a prefix of `fs.xfs`. For example, the full name of the `stats_clear` parameter is `fs.xfs.stats_clear`.

Permanently Changing a Parameter

To ensure that a parameter is set upon reboot, modify or create a line like the following in the `/etc/modprobe.d/sgi-xfs.conf` file or equivalent:

```
install xfs /sbin/modprobe --ignore-install xfs; echo value > /proc/sys/fs/xfs/systune
```

where:

- *value* is the value you want to set
- *systune* is the parameter name

For example, to permanently set the `rotorstep` parameter to `255`:

```
install xfs /sbin/modprobe --ignore-install xfs; echo 255 >/proc/sys/fs/xfs/rotorstep
```

The change will take effect upon reboot.

Note: This is the recommended method to permanently set an XFS system tunable parameter. Setting the parameter in the `/etc/sysctl.conf` file is not recommended because the file may be parsed at boot time before the `xfs` module is loaded.

Temporarily Changing a Parameter

For a temporary change to a dynamic parameter, use the Linux `sysctl`(8) command as follows:

```
# sysctl -w "fs.xfs.systune=value"
```

where:

- *systune* is the parameter name
- *value* is the value you want to set for the parameter

Note: Do not use spaces around the = character.

For example, to temporarily set the `rotorstep` parameter (which has the `fs.xfs` prefix) to `255`, enter the following:

```
# sysctl fs.xfs.rotorstep=255
fs.xfs.rotorstep = 255
```

Querying a Current Parameter Setting

To query the current setting of a parameter, use the Linux `sysctl`(8) command:

```
# sysctl fs.xfs.systune
```

where:

- *systune* is the parameter name

For example, to query the current setting of the `rotorstep` parameter (which has the `fs.xfs` prefix):

```
# sysctl fs.xfs.rotorstep
rotorstep = 255
```

Parameter Types

This section discusses the following:

- "Parameters to Set at Initial Configuration" on page 79
- "Mount-Time Parameter for Initial Configuration" on page 82
- "Parameters for Special-Case Performance Tuning" on page 82
- "Mount-Time Parameter for Special-Case Performance Tuning" on page 84
- "Debugging Parameters Restricted to SGI Support" on page 85

Parameters to Set at Initial Configuration

Most of the dynamic parameters in this section affect the behavior of all mounted XFS filesystems, therefore you should set them at initial configuration and you should change them only to follow the desired site policy:

inherit_noatim

Specifies whether the noatim flag set by the xfs_io(8) chattr command will be inherited by files in a given directory.

Range of values:

- 0 prevents inheritance
- 1 causes the noatim

  flag to be inherited

inherit_nodfrg

Specifies whether the nodfrg flag set by the xfs_io(8) chattr command will be inherited by files in a given directory.

Range of values:

- 0 prevents inheritance
- 1 causes the nodfrg flag to be inherited

inherit_nodump

Specifies whether the nodump flag set by the xfs_io(8) chattr command will be inherited by files in a given directory.

Range of values:

- `0` prevents inheritance
- `1` causes the `nodump` flag to be inherited

`inherit_nosym`

Specifies whether the `nosymlinks` flag set by the `xfs_io`(8) `chattr` command will be inherited by files in a given directory.

Range of values:

- `0` prevents inheritance
- `1` causes the `nosymlinks` flag to be inherited

`inherit_sync`

Specifies whether the `sync` flag set by the `xfs_io`(8) `chattr` command will be inherited by files in a given directory.

Range of values:

- `0` prevents inheritance
- `1` causes the `sync` flag to be inherited

`sgid_inherit`

Controls the action taken for a file created in a *set group ID* (SGID) directory if the group ID of the new file does not match the effective group ID or one of the supplementary group IDs of the parent directory.

Range of values:

- `0` does not clear the `S_ISGID` bit
- `1` clears the `S_ISGID` bit

`stats_clear`

Specifies whether or not the accumulated XFS statistics in the `/proc/fs/xfs/stat` file are cleared

Range of values:

- `0` does not change the file
- `1` resets the statistics in the file to `0`

`symlink_mode`

Controls the permissions set for symbolic links.

Range of values:

- `0` creates symbolic links with mode `0777` (default)
- `1` specifies that the `umask` value is used

Mount-Time Parameter for Initial Configuration

The parameter in this section affects the behavior of all mounted XFS filesystems, therefore you should it at initial configuration and you should change it only to follow the desired site policy. Changes to this parameter takes effect at mount time.

`probe_dmapi`

Determines whether or not XFS attempts to load the `xfs_dmapi` module and enable the `dmi`/`dmapi`/`xdsm` mount option when mounting a filesystem.

Range of values:

- `0` does not load the module or enable the mount option
- `1` loads the module and enable the mount option

Parameters for Special-Case Performance Tuning

The default values for the following dynamic parameters are optimal for most workload, and you should take extra caution when changing them for performance tuning:

- "`probe_limit`" on page 83
- "`rotorstep`" on page 83

- "`syncd_timer`" on page 84
- "`xfs_buf_age`" on page 84
- "`xfs_buf_timer`" on page 84

`probe_limit`

Note: The `probe_limit` parameter is part of enhanced XFS.

Specifies the maximum number of pages that XFS will cluster together when probing, in order to optimize the conversion of delayed allocation or unwritten extents into real extents.

Range of values:

- Default: `4096` (0x1000)
- Minimum: `0`
- Maximum: `2097151` (0x1fffff)

`rotorstep`

In `inode32` allocation mode, determines how many files the allocator attempts to allocate before moving to the next allocation group. The intent is to control the rate at which the allocator moves between allocation groups when allocating extents for new files.

Range of values:

- Default: `1`
- Minimum: `1`
- Maximum: `255`

See also:

- "`agskip` Mount Option for Allocation Group Specification" on page 67
- "`ibound` Mount Option for SSD Media" on page 67

`syncd_timer`

Specifies the interval (in centiseconds) at which the `xfssyncd` thread flushes metadata such as log activity out to disk does some processing on unlinked inodes.

Range of values:

- Default: `3000`
- Minimum: `100`
- Maximum: `720000`

`xfs_buf_age`

Specifies the age (in centiseconds) at which `xfsbufd` flushes dirty metadata buffers to disk.

Range of values:

- Default: `1500`
- Minimum: `100`
- Maximum: `720000`

`xfs_buf_timer`

Specifies the interval (in centiseconds) at which `xfsbufd` scans the dirty metadata buffers list.

Range of values:

- Default: `100`
- Minimum: `50`
- Maximum: `3000`

Mount-Time Parameter for Special-Case Performance Tuning

The default value for the following parameter is optimal for most workload, and you should take extra caution when changing it for performance tuning. Changes to this parameter takes effect at mount time.

`fstrm_timer`

Specifies the filestream timer, which is the required time interval (in centiseconds) between file creates in a directory to maintain a stream of files.

Range of values:

- Default: `3000`
- Minimum: `1`
- Maximum: `360000`

Debugging Parameters Restricted to SGI Support

Caution: Do not change these parameters unless instructed to do so by SGI Support.

This section discusses debugging parameters that should be changed at the recommendation of SGI Support:

- "`error_level`" on page 85
- "`panic_mask`" on page 86

`error_level`

Specifies the reporting volume when internal errors occur, such as the number of detailed messages and backtraces for filesystem shutdowns. XFS macros use the following threshold values:

`XFS_ERRLEVEL_OFF` is `0`
`XFS_ERRLEVEL_LOW` is `3`
`XFS_ERRLEVEL_HIGH` is `5`

Range of values:

- Default: `3`
- Minimum: `0` (turns off error reporting)
- Maximum: `11`

`panic_mask`

Specifies a bitmask that causes certain error conditions to call `BUG()`. The value is the `AND` value of the following tags representing errors that should cause panics:

XFS_NO_PTAG	0
XFS_PTAG_IFLUSH	0x00000001
XFS_PTAG_LOGRES	0x00000002
XFS_PTAG_AILDELETE	0x00000004
XFS_PTAG_ERROR_REPORT	0x00000008
XFS_PTAG_SHUTDOWN_CORRUPT	0x00000010
XFS_PTAG_SHUTDOWN_IOERROR	0x00000020
XFS_PTAG_SHUTDOWN_LOGERROR	0x00000040

Range of values:

- Default: `0`
- Minimum: `0`
- Maximum: `255`

Index