## NAME

xfs_repair − repair an XFS filesystem

## SYNOPSIS

**xfs_repair** [ **−dfLPv** ] [ **−n** | **-e** ] [ **−m** *maxmem* ] [ **−c** *subopt=value* ] [ **−o** *subopt[=value]* ] [ **−t** *interval* ] [ **−l** *logdev* ] [ **−r** *rtdev* ] *device*
**xfs_repair −V**

## DESCRIPTION

**xfs_repair** repairs corrupt or damaged XFS filesystems (see **xfs**(5)). The filesystem is specified using the *device* argument which should be the device name of the disk partition or volume containing the filesystem. If given the name of a block device, **xfs_repair** will attempt to find the raw device associated with the specified block device and will use the raw device instead.

Regardless, the filesystem to be repaired must be unmounted, otherwise, the resulting filesystem may be inconsistent or corrupt.

## OPTIONS

**−f** Specifies that the filesystem image to be processed is stored in a regular file at *device* (see the **mkfs.xfs −d** *file* option). This might happen if an image copy of a filesystem has been copied or written into an ordinary file. This option implies that any external log or realtime section is also in an ordinary file.

**−L** Force Log Zeroing. Forces **xfs_repair** to zero the log even if it is dirty (contains metadata changes). When using this option the filesystem will likely appear to be corrupt, and can cause the loss of user files and/or data. See the **DIRTY LOGS** section for more information.

**−l** *logdev*
Specifies the device special file where the filesystem's external log resides. Only for those filesystems which use an external log. See the **mkfs.xfs −l** option, and refer to **xfs**(5) for a detailed description of the XFS log.

**−r** *rtdev*
Specifies the device special file where the filesystem's realtime section resides. Only for those filesystems which use a realtime section. See the **mkfs.xfs −r** option, and refer to **xfs**(5) for a detailed description of the XFS realtime section.

**−n** No modify mode. Specifies that **xfs_repair** should not modify the filesystem but should only scan the filesystem and indicate what repairs would have been made. This option cannot be used together with **−e**.

**−P** Disable prefetching of inode and directory blocks. Use this option if you find **xfs_repair** gets stuck and stops proceeding. Interrupting a stuck **xfs_repair** is safe.

**−m** *maxmem*
Specifies the approximate maximum amount of memory, in megabytes, to use for **xfs_repair**. **xfs_repair** has its own internal block cache which will scale out up to the lesser of the process's virtual address limit or about 75% of the system's physical RAM. This option overrides these limits.

**NOTE:** These memory limits are only approximate and may use more than the specified limit.

**−c** *subopt=value*
Change filesystem parameters. Refer to **xfs_admin**(8) for information on changing filesystem parameters.

**−o** *subopt[=value]*
Override what the program might conclude about the filesystem if left to its own devices.

The *subopt*ions supported are:

**bhash**=*bhashsize*
overrides the default buffer cache hash size. The total number of buffer cache entries are limited to 8 times this amount. The default size is set to use up the remainder of 75% of the system's physical RAM size.

**ag_stride**=*ags_per_concat_unit*
This creates additional processing threads to parallel process AGs that span multiple concat units. This can significantly reduce repair times on concat based filesystems.

**force_geometry**
Check the filesystem even if geometry information could not be validated. Geometry information can not be validated if only a single allocation group exists and thus we do not have a backup superblock available, or if there are two allocation groups and the two superblocks do not agree on the filesystem geometry. Only use this option if you validated the geometry yourself and know what you are doing. If In doubt run in no modify mode first.

**noquota**
Don't validate quota counters at all. Quotacheck will be run during the next mount to recalculate all values.

**−t interval**
Modify reporting interval, specified in seconds. During long runs **xfs_repair** outputs its progress every 15 minutes. Reporting is only activated when ag_stride is enabled.

**−v** Verbose output. May be specified multiple times to increase verbosity.

**−d** Repair dangerously. Allow **xfs_repair** to repair an XFS filesystem mounted read only. This is typically done on a root filesystem from single user mode, immediately followed by a reboot.

**−e** If any metadata corruption was repaired, the status returned is 4 instead of the usual 0. This option cannot be used together with **−n**.

**−V** Prints the version number and exits.

**Checks Performed**

Inconsistencies corrected include the following:

1. Inode and inode blockmap (addressing) checks: bad magic number in inode, bad magic numbers in inode blockmap blocks, extents out of order, incorrect number of records in inode blockmap blocks, blocks claimed that are not in a legal data area of the filesystem, blocks that are claimed by more than one inode.
2. Inode allocation map checks: bad magic number in inode map blocks, inode state as indicated by map (free or in-use) inconsistent with state indicated by the inode, inodes referenced by the filesystem that do not appear in the inode allocation map, inode allocation map referencing blocks that do not appear to contain inodes.
3. Size checks: number of blocks claimed by inode inconsistent with inode size, directory size not block aligned, inode size not consistent with inode format.
4. Directory checks: bad magic numbers in directory blocks, incorrect number of entries in a directory block, bad freespace information in a directory leaf block, entry pointing to an unallocated (free) or out of range inode, overlapping entries, missing or incorrect dot and dotdot entries, entries out of hashvalue order, incorrect internal directory pointers, directory type not consistent with inode format and size.
5. Pathname checks: files or directories not referenced by a pathname starting from the filesystem root, illegal pathname components.
6. Link count checks: link counts that do not agree with the number of directory references to the inode.

7. Freemap checks: blocks claimed free by the freemap but also claimed by an inode, blocks unclaimed by any inode but not appearing in the freemap.

8. Super Block checks: total free block and/or free i-node count incorrect, filesystem geometry inconsistent, secondary and primary superblocks contradictory.

Orphaned files and directories (allocated, in-use but unreferenced) are reconnected by placing them in the *lost+found* directory. The name assigned is the inode number.

**Disk Errors**

**xfs_repair** aborts on most disk I/O errors. Therefore, if you are trying to repair a filesystem that was damaged due to a disk drive failure, steps should be taken to ensure that all blocks in the filesystem are readable and writable before attempting to use **xfs_repair** to repair the filesystem. A possible method is using **dd**(8) to copy the data onto a good disk.

**lost+found**

The directory *lost+found* does not have to already exist in the filesystem being repaired. If the directory does not exist, it is automatically created if required. If it already exists, it will be checked for consistency and if valid will be used for additional orphaned files. Invalid *lost+found* directories are removed and recreated. Existing files in a valid *lost+found* are not removed or renamed.

**Corrupted Superblocks**

XFS has both primary and secondary superblocks. **xfs_r epair** uses information in the primary superblock to automatically find and validate the primary superblock against the secondary superblocks before proceeding. Should the primary be too corrupted to be useful in locating the secondary superblocks, the program scans the filesystem until it finds and validates some secondary superblocks. At that point, it generates a primary superblock.

**Quotas**

If quotas are in use, it is possible that **xfs_repair** will clear some or all of the filesystem quota information. If so, the program issues a warning just before it terminates. If all quota information is lost, quotas are disabled and the program issues a warning to that effect.

Note that **xfs_repair** does not check the validity of quota limits. It is recommended that you check the quota limit information manually after **xfs_repair**. Also, space usage information is automatically regenerated the next time the filesystem is mounted with quotas turned on, so the next quota mount of the filesystem may take some time.

## DIAGNOSTICS

**xfs_repair** issues informative messages as it proceeds indicating what it has found that is abnormal or any corrective action that it has taken. Most of the messages are completely understandable only to those who are knowledgeable about the structure of the filesystem. Some of the more common messages are explained here. Note that the language of the messages is slightly different if **xfs_repair** is run in no-modify mode because the program is not changing anything on disk. No-modify mode indicates what it would do to repair the filesystem if run without the no-modify flag.

**disconnected inode** *ino***, moving to lost+found**

An inode numbered *ino* was not connected to the filesystem directory tree and was reconnected to the *lost+found* directory. The inode is assigned the name of its inode number (*ino*). If a *lost+found* directory does not exist, it is automatically created.

**disconnected dir inode** *ino***, moving to lost+found**

As above only the inode is a directory inode. If a directory inode is attached to *lost+found*, all of its children (if any) stay attached to the directory and therefore get automatically reconnected when the directory is reconnected.

**imap claims in-use inode** *ino* **is free, correcting imap**

The inode allocation map thinks that inode *ino* is free whereas examination of the inode indicates that the inode may be in use (although it may be disconnected). The program updates the inode allocation map.

**imap claims free inode** *ino* **is in use, correcting imap**

The inode allocation map thinks that inode *ino* is in use whereas examination of the inode indicates that the inode is not in use and therefore is free. The program updates the inode allocation map.

**resetting inode** *ino* **nlinks from** *x* **to** *y*

The program detected a mismatch between the number of valid directory entries referencing inode *ino* and the number of references recorded in the inode and corrected the the number in the inode.

*fork-type* **fork in ino** *ino* **claims used block** *bno*

Inode *ino* claims a block *bno* that is used (claimed) by either another inode or the filesystem itself for metadata storage. The *fork-type* is either **data** or **attr** indicating whether the problem lies in the portion of the inode that tracks regular data or the portion of the inode that stores XFS attributes. If the inode is a real-time (rt) inode, the message says so. Any inode that claims blocks used by the filesystem is deleted. If two or more inodes claim the same block, they are both deleted.

*fork-type* **fork in ino** *ino* **claims dup extent ...**

Inode *ino* claims a block in an extent known to be claimed more than once. The offset in the inode, start and length of the extent is given. The message is slightly different if the inode is a real-time (rt) inode and the extent is therefore a real-time (rt) extent.

**inode** *ino* − **bad extent ...**

An extent record in the blockmap of inode *ino* claims blocks that are out of the legal range of the filesystem. The message supplies the start, end, and file offset of the extent. The message is slightly different if the extent is a real-time (rt) extent.

**bad** *fork-type* **fork in inode** *ino*

There was something structurally wrong or inconsistent with the data structures that map offsets to filesystem blocks.

**cleared inode** *ino*

There was something wrong with the inode that was uncorrectable so the program freed the inode. This usually happens because the inode claims blocks that are used by something else or the inode itself is badly corrupted. Typically, this message is preceded by one or more messages indicating why the inode needed to be cleared.

**bad attribute fork in inode** *ino*, **clearing attr fork**

There was something wrong with the portion of the inode that stores XFS attributes (the attribute fork) so the program reset the attribute fork. As a result of this, all attributes on that inode are lost.

**correcting nextents for inode** *ino*, **was** *x* − **counted** *y*

The program found that the number of extents used to store the data in the inode is wrong and corrected the number. The message refers to nextents if the count is wrong on the number of extents used to store attribute information.

**entry** *name* **in dir** *dir_ino* **not consistent with .. value (***xxxx***) in dir ino** *ino***, junking entry** *name* **in directory inode** *dir_ino*

The entry *name* in directory inode *dir_ino* references a directory inode *ino*. However, the .. entry in directory *ino* does not point back to directory *dir_ino*, so the program deletes the entry *name* in directory inode *dir_ino*. If the directory inode*ino* winds up becoming a disconnected inode as a result of this, it is moved to *lost+found* later.

**entry** *name* **in dir** *dir_ino* **references already connected dir ino** *ino***, junking entry** *name* **in directory inode** *dir_ino*

The entry *name* in directory inode *dir_ino* points to a directory inode *ino* that is known to be a child of another directory. Therefore, the entry is invalid and is deleted. This message refers to an entry in a small directory. If this were a large directory, the last phrase would read "will clear entry".

**entry references free inode** *ino* **in directory** *dir_ino***, will clear entry**

An entry in directory inode *dir_ino* references an inode *ino* that is known to be free. The entry is therefore invalid and is deleted. This message refers to a large directory. If the directory were small, the message would read "junking entry ...".

## EXIT STATUS

**xfs_repair −n** (no modify mode) will return a status of 1 if filesystem corruption was detected and 0 if no filesystem corruption was detected. **xfs_r epair** run without the −n option will always return a status code of 0 if it completes without problems, unless the flag **-e** is used. If it is used, then status 4 is reported when any issue with the filesystem was found, but could be fixed. If a runtime error is encountered during operation, it will return a status of 1. In this case, **xfs_repair** should be restarted. If **xfs_repair is unable** to proceed due to a dirty log, it will return a status of 2. See below.

## DIRTY LOGS

Due to the design of the XFS log, a dirty log can only be replayed by the kernel, on a machine having the same CPU architecture as the machine which was writing to the log. **xfs_r epair** cannot replay a dirty log and will exit with a status code of 2 when it detects a dirty log.

In this situation, the log can be replayed by mounting and immediately unmounting the filesystem on the same class of machine that crashed. Please make sure that the machine's hardware is reliable before replaying to avoid compounding the problems.

If mounting fails, the log can be erased by running **xfs_repair** with the -L option. All metadata updates in progress at the time of the crash will be lost, which may cause significant filesystem damage. This should **only** be used as a last resort.

## BUGS

The filesystem to be checked and repaired must have been unmounted cleanly using normal system administration procedures (the **umount**(8) 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**.

**xfs_repair** does not do a thorough job on XFS extended attributes. The structure of the attribute fork will be consistent, but only the contents of attribute forks that will fit into an inode are checked. This limitation will be fixed in the future.

The no-modify mode (**−n** option) is not completely accurate. It does not catch inconsistencies in the freespace and inode maps, particularly lost blocks or subtly corrupted maps (trees).

The no-modify mode can generate repeated warnings about the same problems because it cannot fix the problems as they are encountered.

If a filesystem fails to be repaired, a metadump image can be generated with **xfs_metadump**(8) and be sent to an XFS maintainer to be analysed and **xfs_repair** fixed and/or improved.

## SEE ALSO

**dd**(1), **mkfs.xfs**(8), **umount**(8), **xfs_admin**(8), **xfs_metadump**(8), **xfs**(5).