



Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'xfs_db.8' command

\$ man xfs_db.8

xfs_db(8) System Manager's Manual xfs_db(8)

NAME

xfs_db - debug an XFS filesystem

SYNOPSIS

xfs_db [-c cmd] ... [-i|r|x|F] [-f] [-l logdev] [-p progname]

device

xfs_db -V

DESCRIPTION

xfs_db is used to examine an XFS filesystem. Under rare circumstances it can also be used to modify an XFS filesystem, but that task is normally left to xfs_repair(8) or to scripts such as xfs_admin(8) that run xfs_db.

OPTIONS

- c cmd xfs_db commands may be run interactively (the default) or as arguments on the command line. Multiple -c arguments may be given. The commands are run in the sequence given, then the program exits.
- f Specifies that the filesystem image to be processed is stored in a regular file at device (see the mkfs.xfs(8) -d file option). This might happen if an image copy of a filesystem has been made into an ordinary file with xfs_copy(8).
- F Specifies that we want to continue even if the superblock magic is not correct. For use in xfs_metadump.

-i Allows execution on a mounted filesystem, provided it is mounted read-only. Useful for shell scripts which must only operate on filesystems in a guaranteed consistent state (either unmounted or mounted read-only). These semantics are slightly different to that of the -r option.

-l logdev

Specifies the device where the filesystems external log resides.

Only for those filesystems which use an external log. See the

mkfs.xfs(8) -l option, and refer to xfs(5) for a detailed de?

scription of the XFS log.

-p progname

Set the program name to progname for prompts and some error mes?

sages, the default value is xfs_db.

-r Open device or filename read-only. This option is required if the filesystem is mounted. It is only necessary to omit this flag if a command that changes data (write, blocktrash, crc) is to be used.

-x Specifies expert mode. This enables the (write, blocktrash, crc invalidate/revalidate) commands.

-V Prints the version number and exits.

CONCEPTS

xfs_db commands can be broken up into two classes. Most commands are for the navigation and display of data structures in the filesystem.

Other commands are for scanning the filesystem in some way.

Commands which are used to navigate the filesystem structure take argu?

ments which reflect the names of filesystem structure fields. There

can be multiple field names separated by dots when the underlying

structures are nested, as in C. The field names can be indexed (as an

array index) if the underlying field is an array. The array indices

can be specified as a range, two numbers separated by a dash.

xfs_db maintains a current address in the filesystem. The granularity

of the address is a filesystem structure. This can be a filesystem

block, an inode or quota (smaller than a filesystem block), or a direc?

tory block (could be larger than a filesystem block). There are a variety of commands to set the current address. Associated with the current address is the current data type, which is the structural type of this data. Commands which follow the structure of the filesystem always set the type as well as the address. Commands which examine pieces of an individual file (inode) need the current inode to be set, this is done with the inode command.

The current address/type information is actually maintained in a stack that can be explicitly manipulated with the push, pop, and stack commands. This allows for easy examination of a nested filesystem structure. Also, the last several locations visited are stored in a ring buffer which can be manipulated with the forward, back, and ring commands.

XFS filesystems are divided into a small number of allocation groups. xfs_db maintains a notion of the current allocation group which is manipulated by some commands. The initial allocation group is 0.

COMMANDS

Many commands have extensive online help. Use the help command for more details on any command.

a See the addr command.

ablock filoff

Set current address to the offset filoff (a filesystem block number) in the attribute area of the current inode.

addr [field-expression]

Set current address to the value of the field-expression. This is used to "follow" a reference in one structure to the object being referred to. If no argument is given, the current address is printed.

agf [agno]

Set current address to the AGF block for allocation group agno. If no argument is given, use the current allocation group.

agfl [agno]

Set current address to the AGFL block for allocation group agno.

If no argument is given, use the current allocation group.

agi [agno]

Set current address to the AGI block for allocation group agno.

If no argument is given, use the current allocation group.

agresv [agno]

Displays the length, free block count, per-AG reservation size, and per-AG reservation usage for a given AG. If no argument is given, display information for all AGs.

attr_remove [-r|-u|-s] [-n] name

Remove the specified extended attribute from the current file.

-r Sets the attribute in the root namespace. Only one namespace option can be specified.

-u Sets the attribute in the user namespace. Only one namespace option can be specified.

-s Sets the attribute in the secure namespace. Only one namespace option can be specified.

-n Do not enable 'noattr2' mode on V4 filesystems.

attr_set [-r|-u|-s] [-n] [-R|-C] [-v namelen] name

Sets an extended attribute on the current file with the given name.

-r Sets the attribute in the root namespace. Only one namespace option can be specified.

-u Sets the attribute in the user namespace. Only one namespace option can be specified.

-s Sets the attribute in the secure namespace. Only one namespace option can be specified.

-n Do not enable 'noattr2' mode on V4 filesystems.

-R Replace the attribute. The command will fail if the attribute does not already exist.

-C Create the attribute. The command will fail if the attribute already exists.

-v Set the attribute value to a string of this length containing the letter 'v'.

b See the back command.

back Move to the previous location in the position ring.

blockfree

Free block usage information collected by the last execution of the blockget command. This must be done before another blockget command can be given, presumably with different arguments than the previous one.

blockget [-npvs] [-b bno] ... [-i ino] ...

Get block usage and check filesystem consistency. The information is saved for use by a subsequent blockuse, ncheck, or blocktrash command.

-b is used to specify filesystem block numbers about which verbose information should be printed.

-i is used to specify inode numbers about which verbose information should be printed.

-n is used to save pathnames for inodes visited, this is used to support the xfs_ncheck(8) command. It also means that pathnames will be printed for inodes that have problems. This option uses a lot of memory so is not enabled by default.

-p causes error messages to be prefixed with the filesystem name being processed. This is useful if several copies of xfs_db are run in parallel.

-s restricts output to severe errors only. This is useful if the output is too long otherwise.

-v enables verbose output. Messages will be printed for every block and inode processed.

blocktrash [-z] [-o offset] [-n count] [-x min] [-y max] [-s seed] [-0|1|2|3] [-t type] ...

Trash randomly selected filesystem metadata blocks. Trashing occurs to randomly selected bits in the chosen blocks. This command is available only in debugging versions of xfs_db. It is useful for testing xfs_repair(8).

-0 | -1 | -2 | -3

These are used to set the operating mode for blocktrash.

Only one can be used: -0 changed bits are cleared; -1 changed bits are set; -2 changed bits are inverted; -3 changed bits are randomized.

-n supplies the count of block-trashings to perform (default 1).

-o supplies the bit offset at which to start trashing the block. If the value is preceded by a '+', the trashing will start at a randomly chosen offset that is larger than the value supplied. The default is to randomly choose an offset anywhere in the block.

-s supplies a seed to the random processing.

-t gives a type of blocks to be selected for trashing. Multiple -t options may be given. If no -t options are given then all metadata types can be trashed.

-x sets the minimum size of bit range to be trashed. The default value is 1.

-y sets the maximum size of bit range to be trashed. The default value is 1024.

-z trashes the block at the top of the stack. It is not necessary to run blockget if this option is supplied.

blockuse [-n] [-c count]

Print usage for current filesystem block(s). For each block, the type and (if any) inode are printed.

-c specifies a count of blocks to process. The default value is 1 (the current block only).

-n specifies that file names should be printed. The prior blockget command must have also specified the -n option.

bmap [-a] [-d] [block [len]]

Show the block map for the current inode. The map display can be restricted to an area of the file with the block and len arguments. If block is given and len is omitted then 1 is assumed

for len.

The `-a` and `-d` options are used to select the attribute or data area of the inode, if neither option is given then both areas are shown.

`bt dump [-a] [-i]`

If the cursor points to a btree node, dump the btree from that block downward. If instead the cursor points to an inode, dump the data fork block mapping btree if there is one. If the cursor points to a directory or extended attribute btree node, dump that. By default, only records stored in the btree are dumped.

`-a` If the cursor points at an inode, dump the extended attribute block mapping btree, if present.

`-i` Dump all keys and pointers in intermediate btree nodes, and all records in leaf btree nodes.

`btheight [-b blksz] [-n recs] [-w max|-w min] btree types...`

For a given number of btree records and a btree type, report the number of records and blocks for each level of the btree, and the total number of blocks. The btree type must be given after the options.

A raw btree geometry can be provided in the format "record_bytes:key_bytes:ptr_bytes:header_type", where header_type is one of "short", "long", "shortcrc", or "longcrc".

The supported btree types are: `bnobt`, `cntbt`, `inobt`, `finobt`, `bmapbt`, `refcountbt`, and `rmapbt`.

Options are as follows:

`-b` is used to override the btree block size. The default is the filesystem block size.

`-n` is used to specify the number of records to store. This argument is required.

`-w max`

shows only the best case scenario, which is when the btree blocks are maximally loaded.

`-w min`

shows only the worst case scenario, which is when the
btree blocks are half full.

check See the blockget command.

convert type number [type number] ... type

Convert from one address form to another. The known types, with
alternate names, are:

agblock or agbno (filesystem block within an allocation
group)

agino or aginode (inode number within an allocation group)

agnumber or agno (allocation group number)

bboff or daddroff (byte offset in a daddr)

blkoff or fsboff or agboff (byte offset in a agblock or fs?
block)

byte or fsbyte (byte address in filesystem)

daddr or bb (disk address, 512-byte blocks)

fsblock or fsb or fsbno (filesystem block, see the fsblock
command)

ino or inode (inode number)

inoidx or offset (index of inode in filesystem block)

inooff or inodeoff (byte offset in inode)

Only conversions that "make sense" are allowed. The compound
form (with more than three arguments) is useful for conversions
such as convert agno ag agbno agb fsblock.

crc [-i|-r|-v]

Invalidates, revalidates, or validates the CRC (checksum) field
of the current structure, if it has one. This command is avail-
able only on CRC-enabled filesystems. With no argument, valida-
tion is performed. Each command will display the resulting CRC
value and state.

-i Invalidate the structure's CRC value (incrementing it by
one), and write it to disk.

-r Recalculate the current structure's correct CRC value,
and write it to disk.

-v Validate and display the current value and state of the structure's CRC.

daddr [d]

Set current address to the daddr (512 byte block) given by d.

If no value for d is given, the current address is printed, expressed as a daddr. The type is set to data (uninterpreted).

dblock filoff

Set current address to the offset filoff (a filesystem block number) in the data area of the current inode.

debug [flagbits]

Set debug option bits. These are used for debugging xfs_db. If no value is given for flagbits, print the current debug option bits. These are for the use of the implementor.

dquot [-g|-p|-u] id

Set current address to a group, project or user quota block for the given ID. Defaults to user quota.

echo [arg] ...

Echo the arguments to the output.

f See the forward command.

forward

Move forward to the next entry in the position ring.

frag [-adflqRrv]

Get file fragmentation data. This prints information about fragmentation of file data in the filesystem (as opposed to fragmentation of freespace, for which see the freesp command). Every file in the filesystem is examined to see how far from ideal its extent mappings are. A summary is printed giving the totals.

-v sets verbosity, every inode has information printed for it. The remaining options select which inodes and extents are examined. If no options are given then all are assumed set, otherwise just those given are enabled.

-a enables processing of attribute data.

-d enables processing of directory data.

- f enables processing of regular file data.
- l enables processing of symbolic link data.
- q enables processing of quota file data.
- R enables processing of realtime control file data.
- r enables processing of realtime file data.

freesp [-bcds] [-A alignment] [-a ag] ... [-e i] [-h h1] ... [-m m]

Summarize free space for the filesystem. The free blocks are examined and totalled, and displayed in the form of a histogram, with a count of extents in each range of free extent sizes.

- A reports only free extents with starting blocks aligned to alignment blocks.
- a adds ag to the list of allocation groups to be processed. If no -a options are given then all allocation groups are processed.
- b specifies that the histogram buckets are binary-sized, with the starting sizes being the powers of 2.
- c specifies that freesp will search the by-size (cnt) space Btree instead of the default by-block (bno) space Btree.
- d specifies that every free extent will be displayed.
- e specifies that the histogram buckets are equal-sized, with the size specified as i.
- h specifies a starting block number for a histogram bucket as h1. Multiple -h's are given to specify the complete set of buckets.
- m specifies that the histogram starting block numbers are powers of m. This is the general case of -b.
- s specifies that a final summary of total free extents, free blocks, and the average free extent size is printed.

fsb See the fsblock command.

fsblock [fsb]

Set current address to the fsblock value given by fsb. If no value for fsb is given the current address is printed, expressed as an fsb. The type is set to data (uninterpreted). XFS

filesystem block numbers are computed $((agno \ll agshift) | ag?$
block) where $agshift$ depends on the size of an allocation group.
Use the `convert` command to convert to and from this form. Block
numbers given for file blocks (for instance from the `bmap` com?
mand) are in this form.

`fsmap [start] [end]`

Prints the mapping of disk blocks used by an XFS filesystem.
The `map` lists each extent used by files, allocation group meta?
data, journaling logs, and static filesystem metadata, as well
as any regions that are unused. All blocks, offsets, and
lengths are specified in units of 512-byte blocks, no matter
what the filesystem's block size is. The optional start and end
arguments can be used to constrain the output to a particular
range of disk blocks.

`fuzz [-c] [-d] field action`

Write garbage into a specific structure field on disk. Expert
mode must be enabled to use this command. The operation happens
immediately; there is no buffering.

The `fuzz` command can take the following actions against a field:

`zeroes`

Clears all bits in the field.

`ones`

Sets all bits in the field.

`firstbit`

Flips the first bit in the field. For a scalar value,
this is the highest bit.

`middlebit`

Flips the middle bit in the field.

`lastbit`

Flips the last bit in the field. For a scalar value,
this is the lowest bit.

`add` Adds a small value to a scalar field.

`sub` Subtracts a small value from a scalar field.

random

Randomizes the contents of the field.

The following switches affect the write behavior:

-c Skip write verifiers and CRC recalculation; allows in?
valid data to be written to disk.

-d Skip write verifiers but perform CRC recalculation; al?
lows invalid data to be written to disk to test detection
of invalid data.

hash string

Prints the hash value of string using the hash function of the
XFS directory and attribute implementation.

help [command]

Print help for one or all commands.

info Displays selected geometry information about the filesystem.

The output will have the same format that mkfs.xfs(8) prints
when creating a filesystem or xfs_info(8) prints when querying a
filesystem.

inode [inode#]

Set the current inode number. If no inode# is given, print the
current inode number.

label [label]

Set the filesystem label. The filesystem label can be used by
mount(8) instead of using a device special file. The maximum
length of an XFS label is 12 characters - use of a longer label
will result in truncation and a warning will be issued. If no
label is given, the current filesystem label is printed.

log [stop | start filename]

Start logging output to filename, stop logging, or print the
current logging status.

logformat [-c cycle] [-s sunit]

Reformats the log to the specified log cycle and log stripe
unit. This has the effect of clearing the log destructively.

If the log cycle is not specified, the log is reformatted to the

current cycle. If the log stripe unit is not specified, the stripe unit from the filesystem superblock is used.

logres Print transaction reservation size information for each transac?

tion type. This makes it easier to find discrepancies in the reservation calculations between xfsprogs and the kernel, which will help when diagnosing minimum log size calculation errors.

ls [-i] [paths]...

List the contents of a directory. If a path resolves to a directory, the directory will be listed. If no paths are supplied and the IO cursor points at a directory inode, the contents of that directory will be listed.

The output format is: directory cookie, inode number, file type, hash, name length, name.

-i Resolve each of the given paths to an inode number and print that number. If no paths are given and the IO cursor points to an inode, print the inode number.

metadump [-egow] filename

Dumps metadata to a file. See xfs_metadump(8) for more information.

ncheck [-s] [-i ino] ...

Print name-inode pairs. A blockget -n command must be run first to gather the information.

-i specifies an inode number to be printed. If no -i options are given then all inodes are printed.

-s specifies that only setuid and setgid files are printed.

p See the print command.

path dir_path

Walk the directory tree to an inode using the supplied path. Absolute and relative paths are supported.

pop Pop location from the stack.

print [field-expression] ...

Print field values. If no argument is given, print all fields in the current structure.

push [command]

Push location to the stack. If command is supplied, set the current location to the results of command after pushing the old location.

q See the quit command.

quit Exit xfs_db.

ring [index]

Show position ring (if no index argument is given), or move to a specific entry in the position ring given by index.

sb [agno]

Set current address to SB header in allocation group agno. If no agno is given, use the current allocation group number.

source source-file

Process commands from source-file. source commands can be nested.

stack View the location stack.

type [type]

Set the current data type to type. If no argument is given, show the current data type. The possible data types are: agf, agfl, agi, attr, bmapbta, bmapbtd, bnobt, cntbt, data, dir, dir2, dqblk, inobt, inode, log, refcntbt, rmapbt, rtbitmap, rtsummary, sb, symlink and text. See the TYPES section below for more information on these data types.

timelimit [OPTIONS]

Print the minimum and maximum supported values for inode time stamps, quota expiration timers, and quota grace periods supported by this filesystem. Options include:

--bigtime

Print the time limits of an XFS filesystem with the bigtime feature enabled.

--classic

Print the time limits of a classic XFS filesystem.

--compact

Print all limits as raw values on a single line.

--pretty

Print the timestamps in the current locale's date and time format instead of raw seconds since the Unix epoch.

uuid [uuid | generate | rewrite | restore]

Set the filesystem universally unique identifier (UUID). The filesystem UUID can be used by mount(8) instead of using a device special file. The uuid can be set directly to the desired UUID, or it can be automatically generated using the generate option. These options will both write the UUID into every copy of the superblock in the filesystem. On a CRC-enabled filesystem, this will set an incompatible superblock flag, and the filesystem will not be mountable with older kernels. This can be reverted with the restore option, which will copy the original UUID back into place and clear the incompatible flag as needed. rewrite copies the current UUID from the primary superblock to all secondary copies of the superblock. If no argument is given, the current filesystem UUID is printed.

version [feature | versionnum features2]

Enable selected features for a filesystem (certain features can be enabled on an unmounted filesystem, after mkfs.xfs(8) has created the filesystem). Support for unwritten extents can be enabled using the extflg option. Support for version 2 log format can be enabled using the log2 option. Support for extended attributes can be enabled using the attr1 or attr2 option. Once enabled, extended attributes cannot be disabled, but the user may toggle between attr1 and attr2 at will (older kernels may not support the newer version).

If no argument is given, the current version and feature bits are printed. With one argument, this command will write the updated version number into every copy of the superblock in the filesystem. If two arguments are given, they will be used as numeric values for the versionnum and features2 bits respectively.

tively, and their string equivalent reported (but no modifications are made).

write [-c|-d] [field value] ...

Write a value to disk. Specific fields can be set in structures (struct mode), or a block can be set to data values (data mode), or a block can be set to string values (string mode, for symlink blocks). The operation happens immediately: there is no buffering.

Struct mode is in effect when the current type is structural, i.e. not data. For struct mode, the syntax is "write field value".

Data mode is in effect when the current type is data. In this case the contents of the block can be shifted or rotated left or right, or filled with a sequence, a constant value, or a random value. In this mode write with no arguments gives more information on the allowed commands.

- c Skip write verifiers and CRC recalculation; allows invalid data to be written to disk.
- d Skip write verifiers but perform CRC recalculation. This allows invalid data to be written to disk to test detection of invalid data. (This is not possible for some types.)

TYPES

This section gives the fields in each structure type and their meanings. Note that some types of block cover multiple actual structures, for instance directory blocks.

agf The AGF block is the header for block allocation information; it is in the second 512-byte block of each allocation group.

The following fields are defined:

- magicnum** AGF block magic number, 0x58414746 ('XAGF').
- versionnum** version number, currently 1.
- seqno** sequence number starting from 0.
- length** size in filesystem blocks of the allocation

group. All allocation groups except the last one of the filesystem have the superblock's agblocks value here.

bnoroot block number of the root of the Btree holding free space information sorted by block number.

cntroot block number of the root of the Btree holding free space information sorted by block count.

bnolevel number of levels in the by-block-number Btree.

cntlevel number of levels in the by-block-count Btree.

ffirst index into the AGFL block of the first active entry.

fllast index into the AGFL block of the last active entry.

flcount count of active entries in the AGFL block.

freeblks count of blocks represented in the freespace Btrees.

longest longest free space represented in the freespace Btrees.

btreeblks number of blocks held in the AGF Btrees.

agfl The AGFL block contains block numbers for use of the block allocator; it is in the fourth 512-byte block of each allocation group. Each entry in the active list is a block number within the allocation group that can be used for any purpose if space runs low. The AGF block fields **ffirst**, **fllast**, and **flcount** designate which entries are currently active. Entry space is allocated in a circular manner within the AGFL block. Fields defined:

bno array of all block numbers. Even those which are not active are printed.

agi The AGI block is the header for inode allocation information; it is in the third 512-byte block of each allocation group.

Fields defined:

magicnum AGI block magic number, 0x58414749 ('XAGI').

versionnum version number, currently 1.

seqno sequence number starting from 0.

length size in filesystem blocks of the allocation group.

count count of inodes allocated.

root block number of the root of the Btree holding inode allocation information.

level number of levels in the inode allocation Btree.

freecount count of allocated inodes that are not in use.

newino last inode number allocated.

dirino unused.

unlinked an array of inode numbers within the allocation group. The entries in the AGI block are the heads of lists which run through the inode next_unlinked field. These inodes are to be unlinked the next time the filesystem is mounted.

attr An attribute fork is organized as a Btree with the actual data embedded in the leaf blocks. The root of the Btree is found in block 0 of the fork. The index (sort order) of the Btree is the hash value of the attribute name. All the blocks contain a blkinfo structure at the beginning, see type dir for a description. Nonleaf blocks are identical in format to those for version 1 and version 2 directories, see type dir for a description. Leaf blocks can refer to "local" or "remote" attribute values. Local values are stored directly in the leaf block. Leaf blocks contain the following fields:

hdr header containing a blkinfo structure info (magic number 0xfbee), a count of active en?

tries, usedbytes total bytes of names and values, the firstused byte in the name area, holes set if the block needs compaction, and array freemap as for dir leaf blocks.

entries array of structures containing a hashval, nameidx (index into the block of the name), and flags incomplete, root, and local.

nvlist array of structures describing the attribute names and values. Fields always present: val? uelen (length of value in bytes), namelen, and name. Fields present for local values: value (value string). Fields present for remote values: valueblk (fork block number of containing the value).

Remote values are stored in an independent block in the attribute fork. Prior to v5, value blocks had no structure, but in v5 they acquired a header structure with the following fields:

magic attr3 remote block magic number, 0x5841524d ('XARM').

offset Byte offset of this data block within the overall attribute value.

bytes Number of bytes stored in this block.

crc Checksum of the attribute block contents.

uuid Filesystem UUID.

owner Inode that owns this attribute value.

bno Block offset of this block within the inode's attribute fork.

lsn Log serial number of the last time this block was logged.

data The attribute value data.

bmapbt Files with many extents in their data or attribute fork will have the extents described by the contents of a Btree for

that fork, instead of being stored directly in the inode.

Each bmap Btree starts with a root block contained within the inode. The other levels of the Btree are stored in filesystem blocks. The blocks are linked to sibling left and right blocks at each level, as well as by pointers from parent to child blocks. Each block contains the following fields:

magic bmap Btree block magic number, 0x424d4150
('BMAP').

level level of this block above the leaf level.

numrecs number of records or keys in the block.

leftsib left (logically lower) sibling block, 0 if
 none.

rightsib right (logically higher) sibling block, 0 if
 none.

recs [leaf blocks only] array of extent records.
Each record contains startoff, startblock,
blockcount, and extentflag (1 if the extent
is unwritten).

keys [non-leaf blocks only] array of key records.
These are the first key value of each block
in the level below this one. Each record con-
tains startoff.

ptrs [non-leaf blocks only] array of child block
pointers. Each pointer is a filesystem block
number to the next level in the Btree.

bnobt There is one set of filesystem blocks forming the by-block-
number allocation Btree for each allocation group. The root
block of this Btree is designated by the bnoroot field in the
corresponding AGF block. The blocks are linked to sibling
left and right blocks at each level, as well as by pointers
from parent to child blocks. Each block has the following
fields:

magic BNOBT block magic number, 0x41425442

('ABTB').

level level number of this block, 0 is a leaf.

numrecs number of data entries in the block.

leftsib left (logically lower) sibling block, 0 if none.

rightsib right (logically higher) sibling block, 0 if none.

recs [leaf blocks only] array of freespace records. Each record contains startblock and blockcount.

keys [non-leaf blocks only] array of key records. These are the first value of each block in the level below this one. Each record contains startblock and blockcount.

ptrs [non-leaf blocks only] array of child block pointers. Each pointer is a block number within the allocation group to the next level in the Btree.

cntbt There is one set of filesystem blocks forming the by-block-count allocation Btree for each allocation group. The root block of this Btree is designated by the cntroot field in the corresponding AGF block. The blocks are linked to sibling left and right blocks at each level, as well as by pointers from parent to child blocks. Each block has the following fields:

magic CNTBT block magic number, 0x41425443 ('ABTC').

level level number of this block, 0 is a leaf.

numrecs number of data entries in the block.

leftsib left (logically lower) sibling block, 0 if none.

rightsib right (logically higher) sibling block, 0 if none.

recs [leaf blocks only] array of freespace records. Each record contains startblock and blockcount.

keys [non-leaf blocks only] array of key records. These are the first value of each block in the level below this one. Each record contains blockcount and startblock.

ptrs [non-leaf blocks only] array of child block pointers. Each pointer is a block number within the allocation group to the next level in the Btree.

data User file blocks, and other blocks whose type is unknown, have this type for display purposes in xfs_db. The block data is displayed in hexadecimal format.

dir A version 1 directory is organized as a Btree with the directory data embedded in the leaf blocks. The root of the Btree is found in block 0 of the file. The index (sort order) of the Btree is the hash value of the entry name. All the blocks contain a blkinfo structure at the beginning with the following fields:

forw next sibling block.

back previous sibling block.

magic magic number for this block type.

The non-leaf (node) blocks have the following fields:

hdr header containing a blkinfo structure info (magic number 0xfebe), the count of active entries, and the level of this block above the leaves.

btree array of entries containing hashval and before fields. The before value is a block number within the directory file to the child block, the hashval is the last hash value in that block.

The leaf blocks have the following fields:

hdr header containing a blkinfo structure info (magic number 0xfeeb), the count of active entries, namebytes (total name string bytes), holes flag (block needs compaction), and freemap (array of base, size entries for free regions).

entries array of structures containing hashval, nameidx (byte index into the block of the name string), and namelen.

namelist array of structures containing inumber and name.

dir2 A version 2 directory has four kinds of blocks. Data blocks start at offset 0 in the file. There are two kinds of data blocks: single-block directories have the leaf information embedded at the end of the block, data blocks in multi-block directories do not. Node and leaf blocks start at offset 32GiB (with either a single leaf block or the root node block). Freespace blocks start at offset 64GiB. The node and leaf blocks form a Btree, with references to the data in the data blocks. The freespace blocks form an index of longest free spaces within the data blocks.

A single-block directory block contains the following fields:

bhdr header containing magic number 0x58443242 ('XD2B') and an array bestfree of the longest 3 free spaces in the block (offset, length).

bu array of union structures. Each element is either an entry or a freespace. For entries, there are the following fields: inumber, namelen, name, and tag. For freespace, there are the following fields: freetag (0xffff), length, and tag. The tag value is the byte offset in the block of the start of the entry

it is contained in.

bleaf array of leaf entries containing hashval and address. The address is a 64-bit word offset into the file.

btail tail structure containing the total count of leaf entries and stale count of unused leaf entries.

A data block contains the following fields:

dhdr header containing magic number 0x58443244 ('XD2D') and an array **bestfree** of the longest 3 free spaces in the block (offset, length).

du array of union structures as for **bu**.

Leaf blocks have two possible forms. If the Btree consists of a single leaf then the freespace information is in the leaf block, otherwise it is in separate blocks and the root of the Btree is a node block. A leaf block contains the following fields:

lhdr header containing a **blkinfo** structure info (magic number 0xd2f1 for the single leaf case, 0xd2ff for the true Btree case), the total count of leaf entries, and stale count of unused leaf entries.

lents leaf entries, as for **bleaf**.

lbests [single leaf only] array of values which represent the longest freespace in each data block in the directory.

ltail [single leaf only] tail structure containing **bestcount** count of **lbests**.

A node block is identical to that for types **attr** and **dir**.

A freespace block contains the following fields:

fhdr header containing magic number 0x58443246 ('XD2F'), **firstdb** first data block number covered by this freespace block, **nvalid** num?

ber of valid entries, and nused number of en?

tries representing real data blocks.

fbests array of values as for lbests.

dqblk The quota information is stored in files referred to by the superblock uquotino and pquotino fields. Each filesystem block in a quota file contains a constant number of quota en? tries. The quota entry size is currently 136 bytes, so with a 4KiB filesystem block size there are 30 quota entries per block. The dqquot command is used to locate these entries in the filesystem. The file entries are indexed by the user or project identifier to determine the block and offset. Each quota entry has the following fields:

magic magic number, 0x4451 ('DQ').

version version number, currently 1.

flags flags, values include 0x01 for user quota,
0x02 for project quota.

id user or project identifier.

blk_hardlimit absolute limit on blocks in use.

blk_softlimit preferred limit on blocks in use.

ino_hardlimit absolute limit on inodes in use.

ino_softlimit preferred limit on inodes in use.

bcount blocks actually in use.

icount inodes actually in use.

itimer time when service will be refused if soft
limit is violated for inodes.

btimer time when service will be refused if soft
limit is violated for blocks.

iwarns number of warnings issued about inode
limit violations.

bwarns number of warnings issued about block
limit violations.

rtb_hardlimit absolute limit on realtime blocks in use.

rtb_softlimit preferred limit on realtime blocks in use.

rtbcount realtime blocks actually in use.
rtbtimer time when service will be refused if soft
 limit is violated for realtime blocks.
rtbwarns number of warnings issued about realtime
 block limit violations.

inobt There is one set of filesystem blocks forming the inode allo-
 cation Btree for each allocation group. The root block of
 this Btree is designated by the root field in the correspond-
 ing AGI block. The blocks are linked to sibling left and
 right blocks at each level, as well as by pointers from par-
 ent to child blocks. Each block has the following fields:

magic INOBT block magic number, 0x49414254
 ('IABT').

level level number of this block, 0 is a leaf.

numrecs number of data entries in the block.

leftsib left (logically lower) sibling block, 0 if
 none.

rightsib right (logically higher) sibling block, 0 if
 none.

recs [leaf blocks only] array of inode records.
 Each record contains startino allocation-
 group relative inode number, freecount count
 of free inodes in this chunk, and free bit-
 map, LSB corresponds to inode 0.

keys [non-leaf blocks only] array of key records.
 These are the first value of each block in
 the level below this one. Each record con-
 tains startino.

ptrs [non-leaf blocks only] array of child block
 pointers. Each pointer is a block number
 within the allocation group to the next level
 in the Btree.

inode Inodes are allocated in "chunks" of 64 inodes each. Usually a

chunk is multiple filesystem blocks, although there are cases with large filesystem blocks where a chunk is less than one block. The inode Btree (see inobt above) refers to the inode numbers per allocation group. The inode numbers directly reflect the location of the inode block on disk. Use the `inode` command to point `xfs_db` to a specific inode. Each inode contains four regions: `core`, `next_unlinked`, `u`, and `a`. `core` contains the fixed information. `next_unlinked` is separated from the `core` due to journaling considerations, see type `agi` field `unlinked`. `u` is a union structure that is different in size and format depending on the type and representation of the file data ("data fork"). `a` is an optional union structure to describe attribute data, that is different in size, format, and location depending on the presence and representation of attribute data, and the size of the `u` data ("attribute fork"). `xfs_db` automatically selects the proper union members based on information in the inode.

The following are fields in the inode core:

- `magic` inode magic number, 0x494e ('IN').
- `mode` mode and type of file, as described in `chmod(2)`, `mknod(2)`, and `stat(2)`.
- `version` inode version, 1 or 2.
- `format` format of `u` union data (0: `xfs_dev_t`, 1: local file - in-inode directory or symlink, 2: extent list, 3: Btree root, 4: unique id [unused]).
- `nlinkv1` number of links to the file in a version 1 inode.
- `nlinkv2` number of links to the file in a version 2 inode.
- `projid_lo` owner's project id (low word; version 2 inode only). `projid_hi` owner's project id (high word; version 2 inode only).

uid owner's user id.

gid owner's group id.

atime time last accessed (seconds and nanoseconds).

mtime time last modified.

ctime time created or inode last modified.

size number of bytes in the file.

nblocks total number of blocks in the file including indirect and attribute.

extsize basic/minimum extent size for the file.

nextents number of extents in the data fork.

naextents number of extents in the attribute fork.

forkoff attribute fork offset in the inode, in 64-bit words from the start of u.

aformat format of a data (1: local attribute data, 2: extent list, 3: Btree root).

dmevmask DMAPI event mask.

dmstate DMAPI state information.

newrtbm file is the realtime bitmap and is "new" for? mat.

prealloc file has preallocated data space after EOF.

realtime file data is in the realtime subvolume.

gen inode generation number.

The following fields are in the u data fork union:

bmbt bmap Btree root. This looks like a bmapbtd block with redundant information removed.

bmx array of extent descriptors.

dev dev_t for the block or character device.

sfdir shortform (in-inode) version 1 directory. This consists of a hdr containing the parent inode number and a count of active entries in the directory, followed by an array list of hdr.count entries. Each such entry contains inumber, namelen, and name string.

`sdir2` shortform (in-inode) version 2 directory.

This consists of a `hdr` containing a count of active entries in the directory, an `i8count` of entries with inumbers that don't fit in a 32-bit value, and the `parent inode` number, followed by an array list of `hdr.count` entries. Each such entry contains `namelen`, a saved offset used when the directory is converted to a larger form, a `name string`, and the `inumber`.

`symlink` symbolic link string value.

The following fields are in the `a` attribute fork union if it exists:

`bmbt` `bmap Btree` root, as above.

`bmx` array of extent descriptors.

`sfattr` shortform (in-inode) attribute values. This consists of a `hdr` containing a `totsize` (total size in bytes) and a count of active entries, followed by an array list of `hdr.count` entries. Each such entry contains `namelen`, `valuelen`, `root flag`, `name`, and `value`.

`log` Log blocks contain the journal entries for XFS. It's not useful to examine these with `xfs_db`, use `xfs_logprint(8)` instead.

`refcntbt` There is one set of filesystem blocks forming the reference count Btree for each allocation group. The root block of this Btree is designated by the `refcntroot` field in the corresponding AGF block. The blocks are linked to sibling left and right blocks at each level, as well as by pointers from parent to child blocks. Each block has the following fields:

`magic` REFC block magic number, 0x52334643 ('R3FC').

`level` level number of this block, 0 is a leaf.

`numrecs` number of data entries in the block.

leftsib left (logically lower) sibling block, 0 if none.

rightsib right (logically higher) sibling block, 0 if none.

recs [leaf blocks only] array of reference count records. Each record contains startblock, blockcount, and refcount.

keys [non-leaf blocks only] array of key records. These are the first value of each block in the level below this one. Each record contains startblock.

ptrs [non-leaf blocks only] array of child block pointers. Each pointer is a block number within the allocation group to the next level in the Btree.

rmapbt There is one set of filesystem blocks forming the reverse mapping Btree for each allocation group. The root block of this Btree is designated by the rmaproot field in the corresponding AGF block. The blocks are linked to sibling left and right blocks at each level, as well as by pointers from parent to child blocks. Each block has the following fields:

magic RMAP block magic number, 0x524d4233 ('RMB3').

level level number of this block, 0 is a leaf.

numrecs number of data entries in the block.

leftsib left (logically lower) sibling block, 0 if none.

rightsib right (logically higher) sibling block, 0 if none.

recs [leaf blocks only] array of reference count records. Each record contains startblock, blockcount, owner, offset, attr_fork, bmbt_block, and unwritten.

keys [non-leaf blocks only] array of double-key

records. The first ("low") key contains the first value of each block in the level below this one. The second ("high") key contains the largest key that can be used to identify any record in the subtree. Each record contains startblock, owner, offset, attr_fork, and bmbt_block.

ptrs [non-leaf blocks only] array of child block pointers. Each pointer is a block number within the allocation group to the next level in the Btree.

rtbitmap If the filesystem has a realtime subvolume, then the rbmino field in the superblock refers to a file that contains the realtime bitmap. Each bit in the bitmap file controls the allocation of a single realtime extent (set == free). The bitmap is processed in 32-bit words, the LSB of a word is used for the first extent controlled by that bitmap word. The atime field of the realtime bitmap inode contains a counter that is used to control where the next new realtime file will start.

rtsummary If the filesystem has a realtime subvolume, then the rsumino field in the superblock refers to a file that contains the realtime summary data. The summary file contains a two-dimensional array of 16-bit values. Each value counts the number of free extent runs (consecutive free realtime extents) of a given range of sizes that starts in a given bitmap block. The size ranges are binary buckets (low size in the bucket is a power of 2). There are as many size ranges as are necessary given the size of the realtime subvolume. The first dimension is the size range, the second dimension is the starting bitmap block number (adjacent entries are for the same size, adjacent bitmap blocks).

sb There is one sb (superblock) structure per allocation group.

It is the first disk block in the allocation group. Only the first one (block 0 of the filesystem) is actually used; the other blocks are redundant information for `xfstool` to use if the first superblock is damaged. Fields defined:

`magicnum` superblock magic number, 0x58465342 ('XFSB').

`blocksize` filesystem block size in bytes.

`dblocks` number of filesystem blocks present in the data subvolume.

`rblocks` number of filesystem blocks present in the realtime subvolume.

`rextents` number of realtime extents that `rblocks` contain.

`uuid` unique identifier of the filesystem.

`logstart` starting filesystem block number of the log (journal). If this value is 0 the log is "external".

`rootino` root inode number.

`rbmino` realtime bitmap inode number.

`rsumino` realtime summary data inode number.

`rextsize` realtime extent size in filesystem blocks.

`agblocks` size of an allocation group in filesystem blocks.

`agcount` number of allocation groups.

`rbmblocks` number of realtime bitmap blocks.

`logblocks` number of log blocks (filesystem blocks).

`versionnum` filesystem version information. This value is currently 1, 2, 3, or 4 in the low 4 bits.

If the low bits are 4 then the other bits have additional meanings. 1 is the original value. 2 means that attributes were used. 3 means that version 2 inodes (large link counts) were used. 4 is the bitmask version of the version number. In this case, the

other bits are used as flags (0x0010: at? tributes were used, 0x0020: version 2 inodes were used, 0x0040: quotas were used, 0x0080: inode cluster alignment is in force, 0x0100: data stripe alignment is in force, 0x0200: the shared_vn field is used, 0x1000: unwrit? ten extent tracking is on, 0x2000: version 2 directories are in use).

sectsize sector size in bytes, currently always 512.

This is the size of the superblock and the other header blocks.

inodesize inode size in bytes.

inopblock number of inodes per filesystem block.

fname obsolete, filesystem name.

fpack obsolete, filesystem pack name.

blocklog log2 of blocksize.

sectlog log2 of sectsize.

inodelog log2 of inodesize.

inopblog log2 of inopblock.

agblklog log2 of agblocks (rounded up).

rextslog log2 of rextents.

inprogress mkfs.xfs(8) or xfs_copy(8) aborted before completing this filesystem.

imax_pct maximum percentage of filesystem space used for inode blocks.

icount number of allocated inodes.

ifree number of allocated inodes that are not in use.

fdblocks number of free data blocks.

frextents number of free realtime extents.

uquotino user quota inode number.

pquotino project quota inode number; this is currently unused.

qflags quota status flags (0x01: user quota account?
ing is on, 0x02: user quota limits are en?
forced, 0x04: quotacheck has been run on user
quotas, 0x08: project quota accounting is on,
0x10: project quota limits are enforced,
0x20: quotacheck has been run on project quo?
tas).

flags random flags. 0x01: only read-only mounts are
allowed.

shared_vn shared version number (shared readonly
filesystems).

inoalignmt inode chunk alignment in filesystem blocks.

unit stripe or RAID unit.

width stripe or RAID width.

dirblklog log2 of directory block size (filesystem
blocks).

symlink Symbolic link blocks are used only when the symbolic link
value does not fit inside the inode. The block content is
just the string value. Bytes past the logical end of the
symbolic link value have arbitrary values.

text User file blocks, and other blocks whose type is unknown,
have this type for display purposes in xfs_db. The block
data is displayed in two columns: Hexadecimal format and
printable ASCII chars.

DIAGNOSTICS

Many messages can come from the check (blockget) command. If the
filesystem is completely corrupt, a core dump might be produced instead
of the message

device is not a valid filesystem

If the filesystem is very large (has many files) then check might run
out of memory. In this case the message

out of memory

is printed.

The following is a description of the most likely problems and the associated messages. Most of the diagnostics produced are only meaningful with an understanding of the structure of the filesystem.

agf_freeblks n, counted m in ag a

The freeblocks count in the allocation group header for allocation group a doesn't match the number of blocks counted free.

agf_longest n, counted m in ag a

The longest free extent in the allocation group header for allocation group a doesn't match the longest free extent found in the allocation group.

agi_count n, counted m in ag a

The allocated inode count in the allocation group header for allocation group a doesn't match the number of inodes counted in the allocation group.

agi_freecount n, counted m in ag a

The free inode count in the allocation group header for allocation group a doesn't match the number of inodes counted free in the allocation group.

block a/b expected inum 0 got i

The block number is specified as a pair (allocation group number, block in the allocation group). The block is used multiple times (shared), between multiple inodes. This message usually follows a message of the next type.

block a/b expected type unknown got y

The block is used multiple times (shared).

block a/b type unknown not expected

SEE ALSO

mkfs.xfs(8), xfs_admin(8), xfs_copy(8), xfs_logprint(8), xfs_metadump(8), xfs_ncheck(8), xfs_repair(8), mount(8), chmod(2), mknod(2), stat(2), xfs(5).

xfs_db(8)