

XFS

Practical

Exercises

08 – Filesystem Repair

© Copyright 2006 Silicon Graphics Inc. All rights reserved.

Permission is granted to copy, distribute, and/or modify this document under the terms of the Creative Commons Attribution-Share Alike, Version 3.0 or any later version published by the Creative Commons Corp. A copy of the license is available at <http://creativecommons.org/licenses/by-sa/3.0/us/>.

Overview

Goals

This lab demonstrates several different corruption problems in SLES10 that `xfs_repair` fails to identify and/or fix, how to identify the problem and what information is needed in order to fix the problem.

Prerequisites

The lab assumes that you are already familiar with the XFS Filesystem Internals document and have tried many of the examples using `xfs_db` to examine on-disk structures.

Setup

All exercises assume the filesystem has a 4KB filesystem block directory block size. They assume the user is running SLES10 but also has available XFS command line tools 2.8.14 or later.

Exercises

Exercise 1 - v2 directory repair problems

This exercise demonstrates a valid filesystem state that `xfs_repair` incorrectly believes is wrong but `xfs_check` correctly believes is OK.

Setup

1. Create a directory with 508 files in it. Delete 1 file.

Exercise

2. Unmount the filesystem and run `xfs_repair` and observe the error that is generated. Rerun `xfs_repair` and note that the error persists – `xfs_repair` has not repaired the “problem”.
3. Run `xfs_check`. What is unexpected about the output from `xfs_check`?
4. Use `xfs_db` to investigate the directory inode and the single leaf block. You should observe there is no level field. Is this a problem?
5. Run the newer version of `xfs_repair` and note the incorrect error is no longer reported, `xfs_repair` has been fixed to understand this scenario.

Exercise 2 - v2 directory repair problems [16777216]

This exercise injects incorrect values into a directory inode and demonstrates how `xfs_repair` detected but did not rectify the problem.

Setup

1. Create a directory with 1000 entries
2. Note the directory inode number using

```
ls -ldi directory
```

3. Unmount the filesystem
4. Find the inode for the directory you created and decrement by 1 the `core.nextents` and `core.nblocks` fields from their current value.

Exercise

5. Run `xfs_check`
6. Run `xfs_repair` and observe
7. Rerun `xfs_repair` and note the problem is not fixed
8. Run `xfs_check` again - different problems, but still corrupted.
9. Run `xfs_db` to dump the inode and dblocks as using setup above:

```
> inode inum
> p
> dblock 0
> p
> dblock 1
```

```
> p
```

Repeat incrementing dblock until it fails

```
> dblock 8388608
```

```
> p
```

Increment dblock print until failure

```
> dblock 16777216
```

```
> p
```

10. Run new xfs_repair twice and then xfs_check. You should find the error is fixed but with some entries in lost+found.
11. Recreate initial conditions and only run new xfs_repair.

Exercise 3 - xfs_repair not correcting multiple entries with the same name

This exercise injects a duplicate name into a directory inode and demonstrates how xfs_repair failed to detect and rectify the problem.

Setup

1. Create a directory with 5 files in it, at least two with names the same length.
 2. Note the directory inode number using
- ```
ls -ldi directory
```
3. Unmount the filesystem
  4. Find the directory inode with xfs\_db and change one of the entries to have the same name as another entry. If the directory is in short form:

```
> write u.sfdir2.list[n].name "duplicate_name"
```

5. If the directory is in long form

```
> dblock 0
> p
> write du[n].name "duplicate_name"
```

### Exercise

6. Run the SLES10 version of xfs\_repair and observe the output
7. Use xfs\_db to observe the problem still exist in the directory inode
8. Run the new xfs\_repair to see this problem fixed

## Exercise 4 – xfs\_repair incorrectly flags extent btrees as corrupt

### Setup

1. Mount the filesystem
2. Create a directory and run

```
xfstests/src/makeextents -b 4096 -n 100000 <file>
```

3. Unmount the filesystem

## Exercise

4. Run the old `xfs_repair -n`
5. Run `xfs_check`
6. Run `xfs_db` to investigate inode which show that `u.bmbt.numrecs` is 2

```
> inode inum
> p
> fsb first_block_in_ptrs_field
> type bmapbtd
> p
> fsb second_block_in_ptrs_field
> type bmapbtd
> p
```

7. Look at the leaves, the beginning and end:

```
> fsb <keys[numrecs] of 1st block>
> type bmapbtd
> p
```

8. All the leaves point back and forth, `leftsib` and `rightsib` are non-null. The old `xfs_repair` is assuming that the first and last leaves under a node always point to null. This can be verified with:

```
> fsb keys[1]_of_2nd_block
> type bmapbtd
> p
```

9. `rightsib` of first should point to `fsb` of the 2nd command and the `leftsib` of the 2nd should point to the first. Based on XFS Filesystem Internals document, it's correct as the two point to each other (1st `rightsib` = 2nd, and 2nd `leftsib` = 1st, and 1st `leftsib` = 2nd `rightsib` = null).
10. Run new `xfs_repair -n` to see it's not an error anymore.

## Questions

---

1. In exercise 1 why are 508 directory entries important?
2. In exercise 2 how would you find the directory inode if the filesystem was not mounted?

## Answers

---

1. With 506 entries, the format is "leaf directory". 507 and 508 entries, the format is "node directory" but the hash values stay in a single block. 509 entries, and the hash values end up in a btree form. With the requirement to delete one entry, 507 will go to 506, back to "leaf" format. 509 and deleting one or two will not undo the btree structure for the hash values.
2. Find the root inode from the super block, and then search the directory entries

```
xfs_db -x device
> sb 0
> p rootino
> inode rootino
> p
```

If the inode is not in short form you may then need to:

```
> dblock 0
> p
> dblock 1
> p
> ...
```