



Whamcloud



Lustre Trash Can Undelete (TCU)

Yingjin Qian, Oleg Drokin, Andreas Dilger
September 2025



Background & Objectives (Why TCU?)

► The Problem:

- Accidental file deletion by users or applications leads to permanent data loss and service interruption:
- "rm -rf" command with wrong filenames/wildcards ("fat finger mistakes")
- Script errors
- Malicious deletion

► The Solution:

- Provide a "second chance" for deleted files
- Store deleting files temporarily by moving them to a hidden holding area before permanently deleting them
- Delayed purging of deleted files by policy or explicit commands
- Allow marking files/directories to avoid trash (e.g. temporary files)

► Critical User Benefit:

- Provide a way for regular users to restore or retrieve deleted files immediately
- Drastically lower the number of data recovery tickets to administrators
- Files can be recovered from trash even if created+deleted after last backup
- **Improved Data Safety:** Provide a configurable grace period against human/tool error

Trash Can/Undelete Core Features Overview

- ▶ **“Delete” effectively becomes “Rename” on MDS**
 - Upon last unlink files are not immediately purged but renamed to hidden `.lustre/trash/MDTxxxx` directory
 - Applies to all forms of file **deletion** operations, not just “rm” or “`unlink()`” on updated clients
- ▶ **File Restoration (Undelete)**
 - Users can restore files from trash to their original location or specified path
- ▶ **Permanent Deletion / Empty Trash**
 - Users/admins can permanently delete files from Trash Can to free the used space
 - Support emptying all deleted files at once
- ▶ **Trash Can Space Usage**
 - Deleted file quota space usage is hidden from users by default
- ▶ **Retention Policy**
 - Files are kept in Trash for a configurable "Grace Period" before cleanup
- ▶ Admin can enable/disable Trash Can feature on whole FS, users can disable on specific directory/files

Virtual “.Trash” Directory & User Access

- ▶ **Concept:** A virtual “.Trash” directory is available in every filesystem directory
 - Allow users to easily browse deleted files/directories under the current subdirectory in the Trash Can
 - Access deleted files/dirs for recovery via standard POSIX API
 - Deleted files can be accessed in **read-only mode** to avoid (ab)use by users/applications
- ▶ **User Experience:** Easy access to deleted files via normal access methods
 - “`ls -la $dir/.Trash`” lists files deleted and moving into Trash Can for the directory \$dir
 - If no files are deleted from a directory, its “.Trash” directory does not exist (ENOENT)
 - Standard POSIX APIs (`readdir()`, `stat()`) work as normal on .Trash and deleted files/subdirs therein
 - Can use any command-line tools (`ls`, `rm`, `rmdir`, etc.), or file browsers if mounted on workstation
- ▶ **Implementation:** Essentially a shortcut to the stub directory with parent directory’s *pFID* name “.lustre/trash/MDTxxxx/*NODEMAP/UID/pFID*” (*NODEMAP/* and *UID/* optionally configured)
- ▶ **Special handling for striped and remote directories**
 - Access is unified through the virtual “.Trash” directory, transparent to the user

File organization under TCU

- ▶ Usually configured via “mdd.*.trash_can_type”
- ▶ “plain”: all stub dir *pFID* are directly created under “.lustre/trash/MDTxxxx”
- ▶ “uid”: Per-User Trash Can (default)
 - Per-User “.lustre/trash/MDTxxxx/UID”, owned by UID, mode 0700
 - Avoid world readable access to deleted files on Trash Can
 - De-conflict files/directories of the same name created by users
 - Avoid exposing files to other users that may be private
 - Allow tracking space usage more clearly of each UID
 - A user’s data can be found and purged more quickly if they are exceeding their quota
- ▶ NODEMAP: Per-Nodemap (optionally + per-user) Trash Can
 - NODEMAP TCU type can only (and always) be set from clients that are part of a nodemap
 - Cannot configure via “mdd.*.trash_can_type”
 - Divide into two categories according to the “mdd.*.trash_can_type” setting
 - “plain”: lustre/trash/MDTxxxx/NODEMAP
 - “uid”: lustre/trash/MDTxxxx/NODEMAP/UID

The Deletion Process: Step-by-Step

- ▶ On `unlink()/rmdir()` or `unlink` via `rename()` during the last unlink
 1. Trash Can directory path depends if unlink sent from client in a nodemap or not
 - a) Nodemap: create local `".lustre/trash/MDTxxxx/NODEMAP"` according to nodemap name
 - b) UID: create `".lustre/trash/MDTxxxx/NODEMAP/UID"` (if needed) and `mdd.*.trash_can_type=uid`
 2. Stub directory Setup: Create the `pFID` stub directory if not exist:
 - a) Plain TCU type: Under `".lustre/trash/MDTxxxx/NODEMAP/pFID"`
 - b) UID TCU type: Under `".lustre/trash/MDTxxxx/NODEMAP/UID/pFID"`
 3. Move/Migrate the victim file or directory object into the `pFID` stub directory
 4. Change the original UID/GID/PROJID of file with Trash Can UID/GID/PROJID and update the quota accounting
 - a) `mdd.*.trash_can_uid`
 - b) `mdd.*.trash_can_gid`
 - c) `mdd.*.trash_can_projid`
 5. Save the original UID/GID/PROJID and timestamp to `"trusted.unrm"` XATTR which can use for `unrm/restore`
 6. Set `LUSTRE_UNRM_FL=FS_UNRM_FL` flag on the file moving into Trash Can

Quota & Space Accounting ([LU-19143](#))



- ▶ **Quota Transfer:** Upon deletion, file's original UID/GID/PROJID changed to dedicated Trash Can IDs
 - **User View:** their quota usage decreases and reflects only “in use” files
 - **Trash Can View:** quota usage is tracked under `trash_can_uid/trash_can_gid/trash_can_projid`
 - Default value for `mdd.*.trash_can_[uid|gid|projid]=-2` means “deleted files change to these IDs”
 - Optionally `mdd.*.trash_can_[uid|gid|projid]=-1` (original) means “keep original file IDs in Trash”
 - Respective UID, GID, and/or PROJID on the deleted file object is unchanged when it is deleted
 - Quota continues to be tracked against original IDs for deleted files
 - In this case **it is up to user to manage their own Trash Can usage** to reduce quota usage
 - To handle case where users abuse Trash Can to store files in rotation to avoid quota limits
- ▶ `getattr()` on deleted files has original ACL, and UID/GID/PROJID from “`trusted.unrm`” XATTR
 - Allow file access/cleanup by original user, prevents access by other users
- ▶ **Undelete/restore files** in Trash Can
 - Restore the original UID/GID/PROJID saved in “`trusted.unrm`” XATTR during rename
 - Quota accounting is updated atomically by the backend OSD when IDs are changed
 - Delete “`trusted.unrm`” XATTR from restored file

Quota & Space Accounting (cont')

- ▶ `df/statfs()` reporting
 - "df" shows free inodes/space as if there were no files in the Trash Can
 - The space/inodes used by Trash Can are added to the free space via `trash_can_projid` quota
 - "`lfs df [--notrash]`" also adjusts free inodes/space for space used by Trash Can
 - "`lfs df --trash`" shows the actual free inodes/space without adjustment for Trash Can usage
- ▶ Nodemap Interaction
 - Allow configuring UID/GID/PROJID to which files in the Trash Can are assigned
 - Can set per-nodemap `trash_can_uid`, `trash_can_gid` and `trash_can_projid` parameters
 - These IDs are within the ID offset range of the nodemap
 - Allow nodemap project quota group to account for **all** space used by the nodemap
 - This isolates Trash Can usage for each nodemap from the regular UID/GID/PROJID of the nodemap users
 - A nodemap mounts with Lustre fileset, with `llite.*.statfs_project=1` parameter set (default)
 - PROJID is set on the fileset with `PROJINHERIT` flag
 - "`df SUBDIR`" will use total = PROJID quota limits, and subtract PROJID quota usage from used space/inodes counters

User Commands & Usage

► Configuration (Administrator):

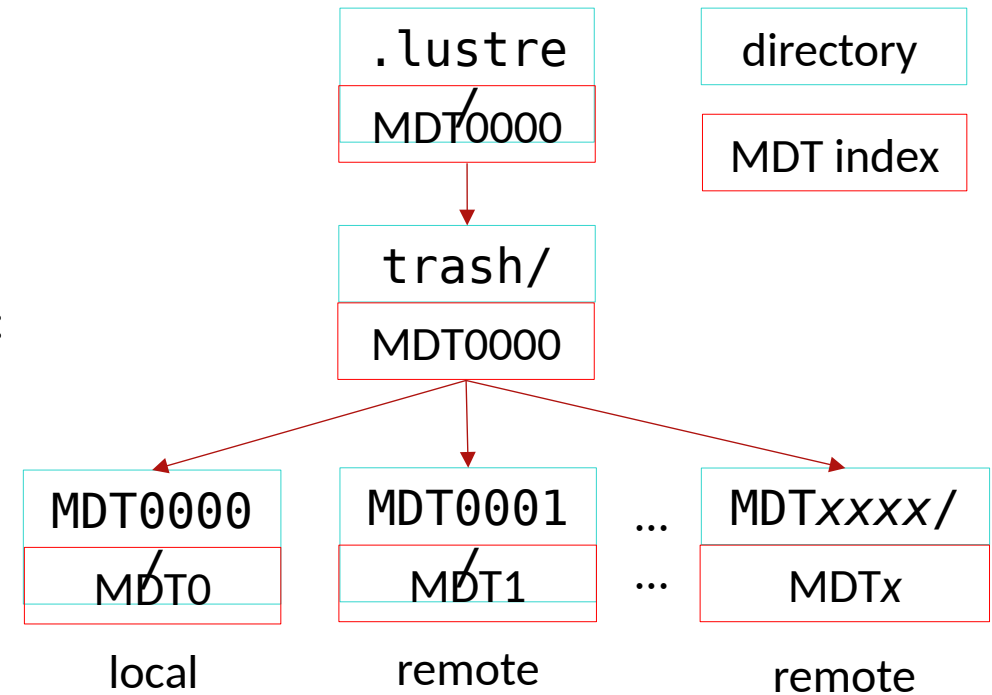
- `lctl set_param mdd.*.trash_can_enable={0,1}`
- `lctl set_param mdd.*.trash_can_{uid|gid|projid}=ID`

► User Commands:

- **List files/directories deleted from current *DIR*:** `ls DIR/.Trash`
- **Display state of Trash Can:** `lfs trash state TARGETDIR`
- **Restore *FILENAME* to parent *DIR*:** `lfs trash unrm
DIR/.Trash/FILENAME`
- **Partially restore file/dir to different parent:** `lfs trash mv
DIR/.Trash/SRCDIR TGTDIR`
- **Permanently delete files in/under *DIR* from Trash:** `lfs trash clean [--
recursive] DIR`
 or directly via POSIX API: `rm [-rf]
DIR/.Trash/FILENAME`
- **Empty Trash for a specific user:** `lfs trash clean --user USER
MOUNTPOINT`

Make trash/ directory visible in client namespace

- ▶ Deleted files moved to “.lustre/trash/MDTxxxx”
 - Local directory on each MDT where the files are located
 - Avoids cross-MDT rename overhead/locking
- ▶ Configure trash directories all MDTs after MDS stack setup:
 - Trash directories for MDTs are created (if not already present)
 - Done after MDS stack setup and MDT has finished recovery
 - Visible in ROOT directory by path “.lustre/trash/MDTxxxx”
 - Can be accessed via normal POSIX I/O interface from clients
 - “.lustre/trash/MDT0000” is a local directory for MDT0
 - “.lustre/trash/MDTxxxx” for other MDTs are remote subdirs



Moving a regular file into trash

- ▶ “Last unlink” for file moving into trash will lookup (or create) a directory named with its parent’s FID
- ▶ This is the **stub dir “pFID”** in the MDT Trash Can directory where the file inode is located
- ▶ The regular file is moved into this pFID subdirectory on Trash Can



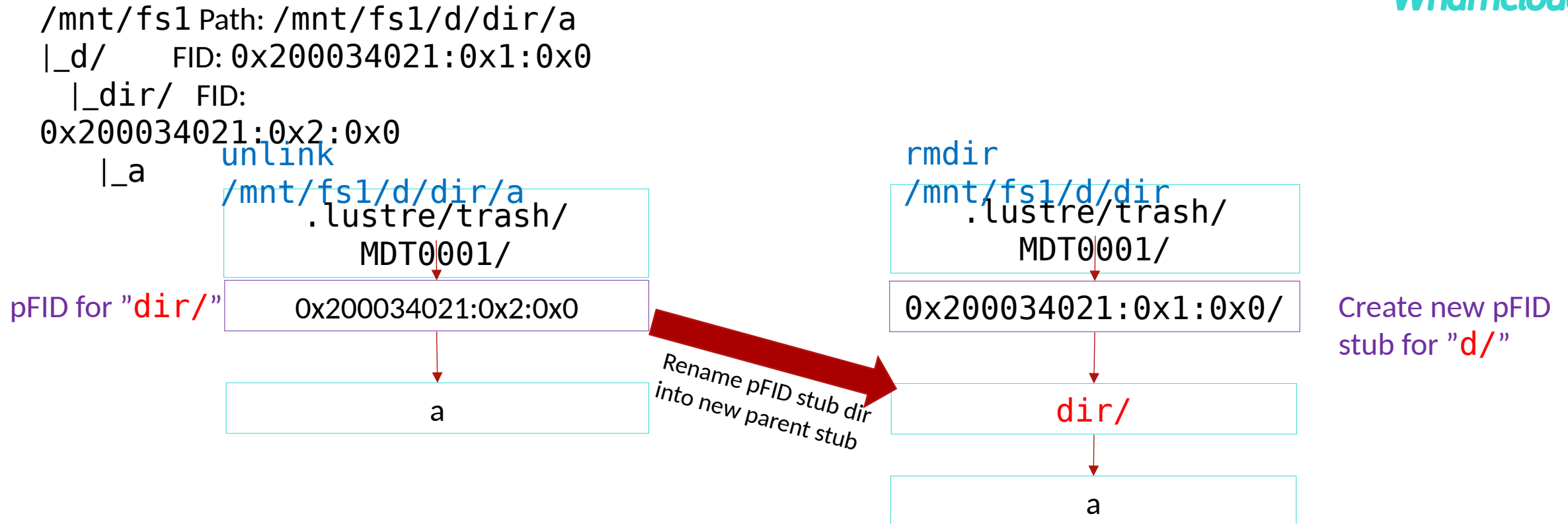
Access trash from Lustre mountpoint on a client:

```
# ls -R /mnt/fs1/.lustre/trash/MDT0001
.lustre/trash/MDT0001/0x200034021:0x1:0x0
.lustre/trash/MDT0001/0x200034021:0x1:0x0/a
```

Access via “.Trash” virtual directory on a client:

```
# ls -R /mnt/fs1/d1/dir/.Trash
/mnt/fs1/d1/dir/.Trash/a
```

Migrate a "undeleted" directory into Trash Can



- ▶ If deleting parent directory was previously empty, the operation is same as deleting a regular file
- ▶ When directory with pFID-named stub in trash is deleted, rename pFID stub to its original name
- ▶ Move into new pFID stub directory for *its* parent in trash
 - Replicate its xattrs like crypt, selinux, etc. to preserve access permissions, fscrypt state, user xattrs, etc.
 - Finally destroy the original directory object that is now empty

Advanced Topics & Special Cases

- ▶ LUSTRE_NOTRASH_FL/FS_NODUMP_FL Attribute ([LU-18958](#))
 - Files or directories can be marked with the NOTRASH flag: `chattr +d FILE`; `chattr -d FILE`
 - Objects with this flag bypass Trash Can completely, destroyed immediately upon the last unlink
 - Ideal for directories with temporary files, or sensitive files that should not be recovered
- ▶ Repeated deletion of file/dir with same name and directory ([LU-19239](#))
 - Configurable whether only oldest or newest file is saved, or unique name is created
 - Can append a timestamp to the filename in Trash to help identify file version
e.g. `file.2025-04-03-00:11:24`, `file.2025-04-03-01:32:17`
 - Configurable upper limit on versions to prevent trash from being overwhelmed by short-lived files (TODO)
- ▶ Deleting symbolic links (symlink) ([LU-19303](#))
 - Treated as regular files. Link target (path) is preserved
 - Deletion and restoration process is same as regular files
- ▶ Deletion via `rename ( )` operation ([LU-19258](#))
 - Running `"mv FILE.tmp FILE"` will delete `FILE` into the Trash Can

Further Development

- ▶ File overwrite via `open(O_TRUNC) / truncate(0)` ([LU-19259](#))
 - Allow moving data deleted via truncation into Trash Can
 - Create a new temporary file on MDT with new OST objects in the Trash Can
 - Copy attributes/xattrs from current file to temporary file
 - Use existing **layout swap** functionality to swap objects and avoid data copy
 - New data written transparently to new OST objects on old MDT inode
- ▶ Cleanup files from Trash Can for nearly full FS
 - Need to automatically clean up Trash Can when filesystem nearly full or files are too old
 - Scan MDT or OST devices for files in Trash Can via policy engine
 - Check TRASH and N0TRASH attributes on files (both MDT and OST)
 - Check the original UID/GID/PROJID and deletion timestamp stored in “trusted.unrm” XATTR
 - Choose candidate files and/or directories to be purged from Trash Can to free up space
 - Potentially via MDS kernel thread for “basic/emergency” policy (e.g. delete oldest pFID directory tree)
 - Userspace policy engine for more sophisticated policy (age/space by user/project, filename, etc.)

Summary

- ▶ Lustre Trash Can Undelete (TCU) enhances data protection and operational efficiency for Lustre
- ▶ TCU Core Value:
 - **Data Protection:** Significantly reduces risk of data loss from accidental or malicious deletion
 - **User Experience:** Provides a familiar, convenient and user-friendly "undelete" operation
 - **Operational Efficiency:** Reduces administrative interaction and overhead for data recovery
- ▶ Limitations:
 - **Hardlink** deletion, truncate, overwrite does not trigger Trash Can functionality
 - May have negative impact on bulk unlink performance
- ▶ Current Status:
 - Feature under active development ([LU-18456](#))
 - HLD document:



Whamcloud

DDN[®]
STORAGE