Linux Assignment 6: Comprehensive Lab Manual

A Complete Educational Guide to Inodes, Hard Links, and Soft Links

Table of Contents

Introduction & Core Concepts

What You'll Learn

- **Inodes**: The fundamental data structure that stores file information
- **Hard Links**: Multiple names for the same file data
- **Soft Links**: Shortcuts that point to other files
- **Link Counts**: How the file system tracks file references
- **File System Structure**: How Linux organizes and manages files

Key Concepts Explained

What is an Inode?

An **inode** (index node) is a data structure that stores information about a file or directory, **except for its name**. Think of it as the file's "ID card" in the file system.

Hard Links vs Soft Links

- **Hard Link**: Another name for the same file data (same inode)
- **Soft Link (Symbolic Link)**: A pointer/shortcut to another file (different inode)

Why This Matters

Understanding these concepts is crucial for:

- **System Administration**: Managing disk space and file organization
- **Backup Systems**: Understanding how files are actually stored
- **Troubleshooting**: Resolving file system issues
- **Security**: Understanding file access and permissions

Task 1: Understanding Inodes

Learning Objective

Master the concept of inodes and understand how the Linux file system organizes and tracks files.

Step 1.1: Setup Directory Structure

```bash
# Create the required directory structure
mkdir -p ~/Assignments/Assignment6

# Navigate to the Assignment6 directory
cd ~/Assignments/Assignment6
```

Step 1.2: Check Inode Usage

```bash
# Navigate to HOME directory first
cd ~

# Check inode usage on your file system
df -i
```

Understanding `df -i` Output:

- **Filesystem**: Storage device or partition
- **Inodes**: Total number of inodes available
- **IUsed**: Number of inodes currently in use
- **IFree**: Number of available inodes
- **IUse%**: Percentage of inodes used

Why This Matters: If you run out of inodes, you can't create new files even if you have disk space available!

📸 **Screenshot Required**: Show the `df -i` output

Step 1.3: Check Link Count for Assignment6 Directory

```bash
# Check link count for Assignment6 directory
ls -ld ~/Assignments/Assignment6
```

Understanding the Output:

```
drwxr-xr-x 2 user group 4096 date Assignment6
    ↑
Link count = 2
```

Answer: Assignment6 directory has a link count of **2**.

Explanation: Every directory starts with a link count of 2 because:

1. **One link** from its parent directory (Assignments)
2. **One link** from the "." (dot) entry inside the directory itself

Step 1.4: Create Files and Directories

```bash
# Navigate to Assignment6 directory
cd ~/Assignments/Assignment6

# Create files
touch f1 f2 f3 f4

# Create directories
mkdir dir1 dir2

# Verify creation
ls -l
```

Step 1.5: Check Link Count Changes

```bash
# Check Assignment6 link count again
ls -ld ~/Assignments/Assignment6

# Check link counts for new directories
ls -ld dir1 dir2
```

Answer for Assignment6: Link count changed from 2 to **4**.

Explanation:

- Original 2 links (from parent + ".")
- +1 link from dir1's ".." entry
- +1 link from dir2's ".." entry
- **Total**: 4 links

Answer for dir1 and dir2: Each has a link count of **2**.

Explanation: Each new directory starts with:

- 1 link from parent directory (Assignment6)
- 1 link from its own "." entry

Step 1.6: Find Inode Numbers

```bash
# Display inode numbers for all items
ls -li
```

Understanding `ls -li`:

- `-l`: Long format listing
- `-i`: Show inode numbers
- **First column**: Inode number (unique identifier)

📸 **Screenshot Required**: Show all inode numbers

Step 1.7: Analyze Inode Numbers

Answer: No files or directories should have the same inode number.

What same inode numbers mean: If two entries have the same inode, they are actually the **same file with different names** (hard links). Each file/directory gets a unique inode number.

Step 1.8: Information Stored in Inodes

Answer: Inodes store:

- **File size** and **type** (regular file, directory, etc.)
- **Permissions** (read, write, execute)
- **Owner** and **group** information
- **Timestamps** (creation, modification, access)
- **Link count** (number of hard links)
- **Data block locations** (where file content is stored)
- **File attributes** and **flags**

Answer: **No**, the filename is **NOT** stored in the inode. Filenames are stored in directory entries, which map names to inode numbers.

Step 1.9: Find Assignment6 Inode from Inside

```bash
# Make Assignment6 your working directory
cd ~/Assignments/Assignment6

# Find the inode number of Assignment6 directory
ls -ld ~/Assignments/Assignment6
```

📸 **Screenshot Required**: Record the inode number

Step 1.10: Verify "." Points to Same Inode

```bash
# From inside Assignment6, check the "." inode
ls -lid .

# Compare with Assignment6 inode from parent
ls -lid ~/Assignments/Assignment6
```

Answer: Both inode numbers should be **exactly the same**.

Explanation: The "." entry in any directory always points to the directory itself, so they must have the same inode number.

📸 **Screenshot Required**: Show both inode numbers are identical

---

Task 2: Hard Links vs Soft Links

Learning Objective

Understand the fundamental differences between hard and soft links through hands-on creation and analysis.

Understanding the Concepts

Hard Links

- **Share the same inode** as the original file
- **Same file data** - modifying one affects the other
- **Cannot cross file systems**
- **Cannot link to directories** (with standard commands)
- **File survives** even if original is deleted

Soft Links (Symbolic Links)

- **Have their own inode**
- **Point to another file's path**
- **Can cross file systems**
- **Can link to directories**
- **Become broken** if target is deleted

Step 2.1: Create Hard Link

```bash
# Navigate to Assignment6 directory
cd ~/Assignments/Assignment6

# Create hard link to f2
ln f2 HL_f2

# Verify creation
ls -li f2 HL_f2
```

Understanding `ln` command:

- `ln target linkname`: Creates hard link
- No special flags needed for hard links

📸 **Screenshot Required**: Show hard link creation command

Step 2.2: Create Soft Link to File

```bash
# Create soft link to f3
ln -s f3 SL_f3

# Verify creation
ls -li f3 SL_f3
```

Understanding `ln -s`:

- `-s`: Creates symbolic (soft) link
- Points to the target file's **path**

📸 **Screenshot Required**: Show soft link creation command

Step 2.3: Create Soft Link to Directory

```bash
# Create soft link to directory dir1
ln -s dir1 SL_dir1

# Verify creation
ls -li dir1 SL_dir1
```

📸 **Screenshot Required**: Show directory soft link creation

Step 2.4: Attempt Hard Link to Directory

```bash
# Try to create hard link to directory dir2
ln dir2 HL_dir2
```

Expected Result: This command will **fail** with an error.

Answer: **No**, it's not possible to create hard links to directories using the standard `ln` command.

Explanation:

- Hard links to directories could create **circular references**
- This could cause infinite loops in file system traversal
- Only the file system itself creates hard links to directories (like "." and "..")
- Some systems allow it with special privileges, but it's generally prohibited

Step 2.5: Analyze Inode Numbers

Fill in the table by running:

```bash
ls -li f2 f3 dir1 HL_f2 SL_f3 SL_dir1
```

Inode number	**name**
[Same as f2]	f2
[Unique]	f3
[Unique]	dir1
[Same as f2]	HL_f2
[Different]	SL_f3
[Different]	SL_dir1

Which inode numbers are the same? Answer: f2 and HL_f2 have the same inode number because HL_f2 is a hard link to f2.

Which inode numbers are different? Answer: All others have different inode numbers because:

- f3 is the original file
- dir1 is a directory (different from files)
- SL_f3 and SL_dir1 are soft links (have their own inodes)

Step 2.6: Find Files by Inode Number

```bash
# Find files with specific inode number (replace XXXX with f2's inode)
find ~/Assignments -inum XXXX
```

Command Template:

```bash
find [directory] -inum [inode_number]
```

Answer: This command searches for all files with the specified inode number.

Step 2.7: Test Soft Link Behavior

```bash
# Create files in dir1
cd dir1
touch xx yy
cd ..

# List contents of dir1
ls dir1

# List contents through soft link
ls SL_dir1
```

Answer: The results should be **exactly the same**.

Explanation: SL_dir1 is a soft link that points to dir1, so accessing it is like accessing dir1 directly. The soft link transparently redirects to the target directory.

Step 2.8: Create Multiple Hard Links

```bash
# Navigate to Assignments directory
cd ~/Assignments

# Create hard link to f1
ln Assignment6/f1 f11

# Navigate to HOME directory
cd ~

# Create another hard link to f11 (which is already a hard link to f1)
ln Assignments/f11 f111
```

Understanding the Chain:

- f1 (original file)
- f11 (hard link to f1)
- f111 (hard link to f11, which is the same as f1)

Answer: File f1 will have a link count of **3**.

Explanation:

- f1, f11, and f111 all point to the same inode
- Each hard link increases the link count
- All three names refer to the exact same file data

Step 2.9: Find All Hard Links

```bash
# Find all files with the same inode as f1 (replace XXXX with f1's inode)
find ~ -inum XXXX 2>/dev/null
```

Expected Output: Should show paths to f1, f11, and f111.

📸 **Screenshot Required**: Show command and output

Does output make sense?: Yes, because all three files share the same inode number - they are the same file with different names.

Step 2.10: Test Link Count After Deletion

```bash
# Delete the original f1 file
rm ~/Assignments/Assignment6/f1

# Check link count for f11
ls -l ~/Assignments/f11
```

Answer: The link count decreased by one to **2**.

Explanation:

- Deleting f1 removed one hard link
- The file data still exists because f11 and f111 still reference it
- **No disk space was freed** because the inode and data blocks are still in use

Did we free disk space?: **No**, the file data remains on disk because other hard links still exist.

Step 2.11: Find Remaining Hard Links

```bash
# Find remaining files with the same inode (use f1's recorded inode number)
find ~ -inum XXXX 2>/dev/null
```

📸 **Screenshot Required**: Show remaining hard links

Step 2.12: Delete All Remaining Hard Links

```bash
# Delete all remaining hard links found in previous step
find ~ -inum XXXX -delete 2>/dev/null

# Verify they're gone
find ~ -inum XXXX 2>/dev/null
```

📸 **Screenshot Required**: Show deletion command and verification

Important Note: Only when **all hard links** are deleted does the file system actually free the disk space occupied by the file data.

Task 3: Managing Links

Learning Objective

Learn proper techniques for managing and cleaning up hard and soft links.

Step 3.1: Delete Links

```bash
# Navigate to Assignment6 directory
cd ~/Assignments/Assignment6

# Delete soft link SL_f3
rm SL_f3

# Delete hard link HL_f2
rm HL_f2

# Verify deletion
ls -l
```

Understanding Link Deletion:

- `rm` command works the same for both hard and soft links
- Deleting a **soft link** only removes the link, target file remains
- Deleting a **hard link** decreases the target's link count

📸 **Screenshot Required**: Show successful deletion

Step 3.2: Find Files with Multiple Hard Links

```bash
# Find all files in HOME directory with more than one hard link
find ~ -type f -links +1 2>/dev/null
```

Understanding the Command:

- `find ~`: Search in HOME directory
- `-type f`: Only regular files (not directories)
- `-links +1`: Files with more than 1 hard link
- `2>/dev/null`: Suppress permission errors

What this finds: Files that have multiple names (hard links) pointing to the same data.

📸 **Screenshot Required**: Show command and output

Task 4: Advanced Link Scenarios

Learning Objective

Analyze complex linking scenarios to understand how link counts change with different operations.

Understanding the Scenarios

Before executing these commands, let's think through what happens step by step.

Scenario 1: Copy vs Link Operations

Commands:

```bash
touch sid ; ln sid bar
cp bar x ; ln x y ; ln bar z
```

Step-by-step Analysis:

1. `touch sid`: Creates file "sid" with link count = 1
2. `ln sid bar`: Creates hard link "bar" to "sid"
   - Both "sid" and "bar" now have link count = 2
3. `cp bar x`: **Copies** content to new file "x"
   - "x" is a **new file** with its own inode, link count = 1
   - "bar" and "sid" still have link count = 2
4. `ln x y`: Creates hard link "y" to "x"
   - "x" and "y" now have link count = 2
   - "bar" and "sid" still have link count = 2
5. `ln bar z`: Creates hard link "z" to "bar"
   - "bar", "sid", and "z" now have link count = 3

Answer: File "bar" has a link count of **3**.

Explanation: "bar" is hard-linked with "sid" and "z". The `cp` operation created a separate file "x", which doesn't affect "bar"'s link count.

Scenario 2: Move vs Link Operations

Commands:

```bash
touch sid ; ln sid bar
mv bar x ; ln x y ; ln y z
```

Step-by-step Analysis:

1. `touch sid`: Creates file "sid" with link count = 1
2. `ln sid bar`: Creates hard link "bar" to "sid"
   - Both "sid" and "bar" have link count = 2
3. `mv bar x`: **Renames** "bar" to "x"
   - This is just a rename operation, same inode
   - "sid" and "x" have link count = 2
4. `ln x y`: Creates hard link "y" to "x"
   - "sid", "x", and "y" have link count = 3
5. `ln y z`: Creates hard link "z" to "y"
   - "sid", "x", "y", and "z" have link count = 4

Answer: File "sid" has a link count of **4**.

Explanation: The `mv` operation just renamed "bar" to "x" but didn't create a new file. All subsequent hard links ("y" and "z") point to the same inode as the original "sid".

Key Differences Between Scenarios

Operation	Effect on Link Count	Creates New Inode?
`cp` (copy)	No change to original	Yes - new file
`mv` (move/rename)	No change	No - same file
`ln` (hard link)	Increases by 1	No - shared inode

Verification Commands

If you want to verify these scenarios:

```bash
# For Scenario 1
mkdir test1
cd test1
touch sid ; ln sid bar
cp bar x ; ln x y ; ln bar z
ls -li  # Check inode numbers and link counts
cd ..

# For Scenario 2
mkdir test2
cd test2
touch sid ; ln sid bar
mv bar x ; ln x y ; ln y z
ls -li  # Check inode numbers and link counts
cd ..
```

Summary & Key Takeaways

Fundamental Concepts Mastered

1. **Inodes**: The core data structure storing file metadata
2. **Hard Links**: Multiple names for the same file data
3. **Soft Links**: Pointers to other files or directories
4. **Link Counts**: How the file system tracks references
5. **File System Operations**: How copying, moving, and linking affect file structure

Critical Understanding Points

Hard Links

- **Same inode** as target file
- **Increase link count** of target
- **Cannot cross file systems**
- **File survives** deletion of other hard links
- **Cannot link to directories** (standard operation)

Soft Links

- **Own inode**, different from target
- **Can cross file systems**
- **Can link to directories**
- **Become broken** if target is deleted
- **Act as shortcuts** to other files

File Operations Impact

- **Copy (`cp`)**: Creates new file with new inode
- **Move (`mv`)**: Renames file, same inode
- **Hard Link (`ln`)**: Same inode, increases link count
- **Soft Link (`ln -s`)**: New inode, points to target

Real-World Applications

System Administration

- **Backup strategies**: Understanding when files are truly deleted
- **Disk space management**: Knowing when space is actually freed
- **File organization**: Using links to organize without duplicating data

Development & DevOps

- **Configuration management**: Using links to share config files
- **Application deployment**: Creating shortcuts to different versions
- **Log management**: Understanding when log files can be safely removed

Security & Forensics

- **File tracking**: Finding all references to sensitive data
- **Incident response**: Understanding file relationships
- **Data recovery**: Locating files through inode analysis

Commands Mastered

- **File System Analysis**: `df -i`, `ls -li`, `ls -ld`
- **Link Creation**: `ln`, `ln -s`
- **File Operations**: `touch`, `mkdir`, `cp`, `mv`, `rm`
- **Search Operations**: `find -inum`, `find -links`, `find -type`
- **Link Management**: Understanding when and how to use each type

Best Practices Learned

1. **Use hard links** for backup scenarios where you want file survival
2. **Use soft links** for shortcuts and cross-file system references
3. **Check link counts** before assuming file deletion frees space
4. **Understand inode implications** when managing disk space
5. **Be cautious with directory links** to avoid circular references

🎉 **Congratulations!** You've mastered fundamental Linux file system concepts that are essential for system administration, development, and advanced Linux usage. These concepts form the foundation for understanding how Linux manages files and storage at the deepest level.