Hands-On Lab (HOL): Git Internal Architecture Mapping – Working Directory, Staging Area, and Repository

Author

Dr. Sandeep Kumar Sharma

Learning Objective

The objective of this Hands-On Lab is to help learners deeply understand Git's internal architecture by mapping every Git command to its internal components such as the Working Directory, Staging Area (Index), and Local Repository. This lab focuses on *what happens internally* when Git commands are executed.

Learning Outcome

After completing this lab, learners will be able to:

- Clearly explain Git's internal architecture in a classroom or interview
- Map each Git command to the correct internal layer
- Understand how files move between Working Directory, Staging Area, and Repository
- Explain how Git stores data internally (objects, commits, HEAD)
- Debug common Git issues by understanding internal states

Git Internal Architecture – Conceptual Overview

Git internally works with **three major layers**:

1. **Working Directory (WD)**

2. Actual directory where files are created, edited, or deleted

3. Managed by the operating system

4. **Staging Area (Index)**

5. Intermediate area where changes are prepared before commit

6. Acts like a buffer between WD and Repository

7. **Local Repository (.git directory)**

8. Stores the complete history of the project

9. Contains commits, branches, HEAD pointer, and objects

```
Working Directory  --->  Staging Area  --->  Local Repository
       (edit)              (git add)           (git commit)
```

Step-by-Step Hands-On Lab with Architecture Mapping

Step 1: Create Project Directory (Working Directory Layer)

```
mkdir project
```

Internal Mapping:

- Directory created in the **Working Directory**
- Git is not involved yet

Step 2: Enter Project Directory

```
cd project
```

Internal Mapping:

- Still outside Git's control
- OS-level navigation only

Step 3: Check Git Status (Before Initialization)

```
git status
```

Internal Mapping:

- Git checks for `.git` directory

- Result: ❌Not a Git repository

Step 4: Initialize Git Repository

```
git init
```

Internal Mapping:

- `.git` directory created
- Git Repository layer initialized

Inside `.git`:

- `HEAD` → points to current branch
- `objects/` → stores blobs, trees, commits
- `refs/` → stores branch references

Step 5: Verify Hidden Git Files

```
ls -a
```

Internal Mapping:

- `.git` directory visible
- Git repository is now active

Step 6: Configure Git Identity (Metadata Layer)

```
git config --global user.name "sandeep"
git config --global user.email "sandeep@gmail.com"
```

Internal Mapping:

- Configuration stored in `~/.gitconfig`
- Used during commit object creation

Step 7: Create File in Working Directory

```
vi login.php
```

Internal Mapping:

- File exists only in **Working Directory**
- Git is aware but not tracking it yet

Step 8: Check Git Status (Untracked File)

```
git status
```

Internal Mapping:

- Git detects file in WD
- File state: **Untracked**

Step 9: Add File to Staging Area

```
git add login.php
```

Internal Mapping:

- Snapshot of file stored in **Staging Area (Index)**
- Git creates a *blob object* internally

```
WD  --->  INDEX
(login.php snapshot)
```

Step 10: Verify Staged State

```
git status
```

Internal Mapping:

- File marked as **staged**
- Ready for commit

Step 11: Commit File to Repository

```
git commit -m "this is my first commit" login.php
```

Internal Mapping:

- Commit object created
- Includes:
- Tree object
- Blob reference
- Parent commit (if any)
- Author & timestamp

```
INDEX  --->  LOCAL REPOSITORY
(commit object created)
```

Step 12: Check Repository State

```
git status
```

Internal Mapping:

- WD, Index, and Repository are in sync
- Clean working tree

Step 13: View Commit Internals

```
git show
```

Internal Mapping:

- Reads commit object from `.git/objects`
- Displays metadata and file diffs

Step 14: View Commit History

```
git log
```

Internal Mapping:

- Traverses commit graph using parent pointers
- Uses `HEAD` reference

Git Object Model (Conceptual)

Git internally stores everything as objects:

- **Blob** → File content
- **Tree** → Directory structure
- **Commit** → Snapshot + metadata
- **Tag** → Named reference (optional)

```
Commit
  └── Tree
        └── Blob (login.php)
```

Common Interview Mapping Question

Q: Where does `` store data?
➡️Staging Area (Index)

Q: Where does `` store data?
➡️Local Repository (.git/objects)

Q: What does `` compare?
➡️WD vs Index vs Repository

Conclusion

This lab explained Git not just as a command-line tool but as an **internally structured version control system**. Understanding this architecture allows developers and DevOps engineers to confidently use Git, troubleshoot issues, and explain Git behavior clearly in real-world projects and interviews.