Hands-On Lab (HOL): Understanding Git Reset, Restore, and Checkout

Author

Dr. Sandeep Kumar Sharma

Learning Objective

The objective of this Hands-On Lab is to help learners clearly understand **git reset**, **git restore**, and **git checkout** commands by mapping them to Git's internal architecture (Working Directory, Staging Area, and Repository). This lab removes confusion by explaining *what exactly changes*, *where it changes*, and *why the command behaves that way*.

Learning Outcome

After completing this lab, learners will be able to:

- Understand the difference between `git reset`, `git restore`, and `git checkout`
- Identify which Git area (WD, Index, HEAD) is affected by each command
- Safely undo changes without data loss
- Confidently answer interview questions related to undoing changes in Git
- Choose the correct command for real-world DevOps scenarios

Pre-requisite Concept (Must Recall)

Git has **three important areas**:

1. **Working Directory (WD)** – Where files are edited
2. **Staging Area (Index)** – Where changes are prepared
3. **Repository (HEAD)** – Last committed snapshot

```
Working Directory  <-->  Staging Area  <-->  Repository (HEAD)
```

Undoing changes means deciding **from which area** and **to which area** you want to move data back.

Initial Lab Assumption

- `login.php` file exists
- File is already committed once
- Repository is clean initially

Step 1: Modify the File (Working Directory)

```
vi login.php
```

- Add or modify some content
- Save and exit (`Esc :wq`)

State:

- Change exists only in **Working Directory**

Step 2: Check Status

```
git status
```

- File shows as **modified**
- Change is not staged

PART 1: git restore (Safe Undo Command)

Step 3: Discard Changes from Working Directory

```
git restore login.php
```

What happens internally:

- Working Directory is reset to match **Staging Area / HEAD**
- No commit history is affected

Simple Meaning:

> "Undo my local changes"

Step 4: Verify Status

```
git status
```

- Working tree is clean again

PART 2: git reset (Moving HEAD / Index)

Step 5: Modify and Stage the File Again

```
vi login.php
git add login.php
git status
```

- Change is now in **Staging Area**

Step 6: Unstage the File (Soft Reset)

```
git reset login.php
```

Internal Mapping:

- File moved from **Staging Area** → **Working Directory**
- Content is NOT lost

Simple Meaning:

> "I added by mistake, keep the code but unstage it"

Step 7: Verify Status

```
git status
```

- File appears as modified but not staged

Step 8: Hard Reset (⚠️Dangerous)

```
git reset --hard HEAD
```

Internal Mapping:

- WD + Index reset to match Repository
- All uncommitted changes are LOST

Trainer Warning:

> Never use `--hard` unless you are 100% sure

PART 3: git checkout (Older Method)

Step 9: Modify File Again

```
vi login.php
```

Step 10: Undo Changes Using Checkout

```
git checkout -- login.php
```

Internal Mapping:

- File restored from **Repository → Working Directory**

Note:

- `git checkout` is now discouraged for file restore
- Replaced by `git restore`

Quick Comparison Table

Command	Affects	Safe?	Use Case
`git restore`	WD	✅Yes	Undo local changes
`git reset`	Index / HEAD	⚠️Depends	Unstage or move commits

Command	Affects	Safe?	Use Case
`git reset --hard`	WD + Index	❌No	Discard everything
`git checkout -- file`	WD	⚠️Legacy	Older restore method

Common Interview Questions

Q1: Which command is safest to undo local changes?
➡️`git restore`

Q2: Which command moves HEAD?
➡️`git reset`

Q3: Difference between reset and restore?
➡️Reset changes history/index, restore changes files

Conclusion

Think of **reset** as *moving pointers*, **restore** as *fixing files*, and **checkout** as the *old all-in-one tool*. Once learners understand *which Git area is affected*, undoing changes becomes logical instead of scary.

✅**End of Hands-On Lab – Git Reset, Restore, and Checkout**