**SCRIPT MODULE 6**

**Objective:**

This module teaches how to fix common mistakes in Git using reset, restore, revert, and reflog. These commands help you recover from errors without fear.

**🔹 Topic 1: Unstaging Files (git reset HEAD)**

**➤ Step-by-Step:**

1. Create and modify a file:

bash

echo "line one" > test.txt

git add test.txt

1. Now you decide to unstage the file:

bash

git reset HEAD test.txt

1. Run:

bash

git status

“You’ve staged a file using git add, but then realize you’re not ready to commit. git reset HEAD <file> will *unstage* it but keep your edits.”

**🔹 Topic 2: Revert a Commit (git revert)**

**➤ Step-by-Step:**

1. Commit a change:

bash

echo "1st line" >> test1.txt

git add test1.txt

git commit -m "Added 1st line"

echo "2nd line" >> test1.txt

git add test1.txt

git commit -m "Added 2nd line"

1. Revert that commit:

bash

git revert HEAD

“git revert creates a *new commit* that undoes the effect of a previous commit. It’s a safe way to undo history because it doesn’t change commit history—ideal for public repos.”

**🔹 Topic 3: Restore File to Last Commit (git restore)**

**➤ Step-by-Step:**

1. Modify the file again:

bash

echo "temporary text" >> test.txt

1. To discard the uncommitted changes:

bash

git restore test.txt

“git restore is a powerful command to discard changes in your working directory. If you made a mess and want to go back to the last committed version, use this.”

**🔹 Topic 4: git reset (Soft, Mixed, Hard)**

Let’s explore how to reset your commits using 3 different levels.

**➤ Step-by-Step:**

**Create 3 Commits:**

bash

echo "one" > file.txt

git add . && git commit -m "Commit 1"

echo "two" >> file.txt

git add . && git commit -m "Commit 2"

echo "three" >> file.txt

git add . && git commit -m "Commit 3"

**✅ git reset --soft HEAD~1**

bash

git reset --soft HEAD~1

✔️ **Effect:** Removes last commit, but keeps changes *staged*.

“Soft reset keeps your work staged and ready for a new commit. Great if you just want to rewrite the message or squash commits.”

**✅ git reset --mixed HEAD~1**

bash

git reset --mixed HEAD~1

✔️ **Effect:** Unstages the changes but keeps your edits.

“Mixed reset unstages your files — like git reset HEAD — but for older commits.”

**✅ git reset --hard HEAD~1**

bash

git reset --hard HEAD~1

✔️ **Effect:** Removes commit **AND** discards changes from working directory.

“Hard reset is dangerous! It completely erases history and changes. Use only when you're absolutely sure.”

**🔹 Topic 5: Reflog — Emergency Recovery Tool**

**➤ Step-by-Step:**

1. Check Git history:

bash

git reflog

1. You’ll see something like:

perl

abc1234 HEAD@{0}: reset: moving to HEAD~1

bcd2345 HEAD@{1}: commit: Added feature

1. Restore to an earlier point:

bash

git reset --hard abc1234

“Git keeps a hidden log of all your moves. reflog is your time machine — even after a hard reset, you can recover old commits using this.”

**Summary Table**

| **Command** | **Purpose** | **Keeps Changes?** |
| --- | --- | --- |
| git reset HEAD file | Unstage file | ✅ Yes |
| git revert | Undo commit (safe) | ✅ Yes |
| git restore file | Discard file changes | ❌ No |
| git reset --soft | Undo commit, keep staged | ✅ Yes |
| git reset --mixed | Undo commit, keep unstaged | ✅ Yes |
| git reset --hard | Undo commit + delete changes | ❌ No |
| git reflog | View all HEAD changes | ✅ Yes |

**Real-Life Analogy for Undo Actions**

| **Git Command** | **Analogy** |
| --- | --- |
| git revert | Using **Ctrl+Z** to undo but saving the history |
| git reset --hard | Throwing away your paper and burning it |
| git restore | Reverting the Word file back to saved version |
| reflog | Checking your browser history to revisit a closed tab |