DevOps Shack Git Assignment | Task:9

Task 9: Reset vs Revert – Undoing Changes

9.1 Introduction to Undoing Changes in Git

In any **corporate development workflow**, mistakes happen:

- Commits pushed to the wrong branch.
- Buggy code merged into production.
- Half-baked changes that shouldn't be there.

Git provides **powerful tools** for **undoing changes** safely:

1. **git reset**:
    - Moves the **branch pointer backward** (modifies history).
    - Optionally modifies the **working directory** and **staging area**.
2. **git revert**:
    - Creates a **new commit** that **undoes the changes** introduced by an earlier commit.
    - **Preserves history**—important for **shared/public branches**.

Analogy:

- **Reset**: Like **rewinding time**, erasing events.
- **Revert**: Like **correcting a mistake in your diary**, leaving a record.

9.2 Why Understanding Reset vs Revert is Crucial in Corporate Workflows

- In a **team environment**, history integrity matters.
- Developers must know **when to rewrite history** (reset) and **when to preserve it** (revert).
- Using the **wrong undo method** can:
    - Cause **confusion**.
    - **Break shared repositories**.
    - Lead to **lost work**.

Undo Method	**Purpose**	**Rewrites History?**	**Where to Use**
git reset	Move branch pointer backward, optionally undo changes	Yes	**Local branches only**
git revert	Add a commit that undoes previous commit's changes	No	**Shared/public branches**

9.3 Deep Dive: How git reset Works Internally

git reset moves the HEAD and branch pointer.

Three Modes of Reset:

Mode	**Affects Commit History**	**Affects Staging Area (Index)**	**Affects Working Directory**
Soft	Yes	No	No
Mixed	Yes	Yes	No
Hard	Yes	Yes	Yes

Visualizing Reset Modes:

Example Setup:

A---B---C---D (HEAD)

You want to **reset to B**.

Soft Reset (--soft):

- Moves **HEAD** and **branch pointer** back to **B**.
- Leaves **staging area** and **working directory unchanged**.

```
A---B (HEAD)
     \
      C---D (staged/working)
```

- **Staged and working directory** still contain **C** and **D's changes**.

Mixed Reset (--mixed):

- Moves **HEAD** and **branch pointer** back to **B**.
- **Unstages changes** in **C** and **D**.
- Keeps **working directory unchanged**.

```
A---B (HEAD)
     \
      C---D (working directory)
```

- Changes from **C** and **D** are now in **unstaged** state.

Hard Reset (--hard):

- Moves **HEAD** and **branch pointer** back to **B**.
- **Wipes staging area** and **working directory**.

A---B (HEAD)

- **C** and **D's changes** are **gone**.

Warning:

- **Hard reset** is **destructive**.
- Use cautiously, especially if **uncommitted changes exist**.

9.4 Deep Dive: How git revert Works Internally

git revert:

- Creates a **new commit** that **reverses the changes** of a specific commit.

Visualizing Revert:

Example Setup:

A---B---C---D (HEAD)

You **revert commit C**:

A---B---C---D---E (HEAD)

- **E** contains **inverse changes** of **C**.

- **History is preserved**:
    - Everyone can **trace what happened**.

Note:

- **Revert** works best for:
    - **Shared/public branches**.
    - **Critical rollbacks** (production fixes).

9.5 Step-by-Step Implementation: git reset

Scenario Setup:

- You accidentally **committed a temporary debug statement**.

Commits:

a1b2c3d Add main feature

d4e5f6g Debug statement

Step 1: View Commit History

```
git log --oneline
```

Step 2: Soft Reset (Preserve Changes)

```
git reset --soft HEAD~1
```

- Moves **HEAD** back one commit (to **Add main feature**).
- Leaves **debug statement changes staged**.

Step 3: Mixed Reset (Unstage Changes)

```
git reset --mixed HEAD~1
```

- Moves **HEAD** back.
- **Unstages** the **debug statement** (but keeps it in working directory).

Step 4: Hard Reset (Delete Changes)

git reset --hard HEAD~1

- **Deletes** the **debug statement**.

9.6 Step-by-Step Implementation: git revert

Scenario Setup:

- Commit history:

a1b2c3d Add main feature

d4e5f6g Debug statement

Step 1: Revert the Debug Commit

git revert d4e5f6g

- Creates a **new commit**:

a1b2c3d Add main feature

d4e5f6g Debug statement

e7f8g9h Revert "Debug statement"

- History **preserved**.

9.7 Visualizations of Reset vs Revert

Reset (soft/mixed/hard):

A---B---C (HEAD)

- After **reset to B**:

A---B (HEAD)

- **Commits removed**.

Revert:

A---B---C (HEAD)

- After **revert C**:

A---B---C---D (HEAD)

- **D** reverses **C**, but **history remains intact**.

9.8 Best Practices for Reset and Revert

Action	**When to Use**
git reset --soft	Keep changes staged, undo last commit.
git reset --mixed	Unstage changes but keep working directory.
git reset --hard	Erase all changes (working directory too).
git revert	Undo a commit **without rewriting history**.

Golden Rules:

- **Never reset shared/public branches**.
- Use **revert for public branches**.
- **Reset only for local work**.

9.9 Common Mistakes & Pitfalls

Mistake	**How to Avoid**
Hard resetting **shared branches**	Use **revert** instead.
Losing uncommitted work with hard reset	Always **stash or commit** before reset.
Reverting a **merge commit** incorrectly	Use git revert -m 1 <merge-commit>.

9.10 Reset and Revert in CI/CD Workflows

1. **Revert for production rollbacks**:
   - Revert buggy commits without breaking pipelines.
2. **Reset in local development**:

- Clean up **before pushing to remote**.