Lab Exercise 2- Working with Git Reset

Lab Exercise: Git Reset

This lab exercise will guide you through the usage of the git reset command in various scenarios. The git reset command is used to undo changes in the Git history, working directory, or staging area. There are three main modes: **soft**, **mixed**, and **hard**.

Objective

- Learn how to use git reset to modify the commit history, unstage files, or discard changes.
- Understand the differences between --soft, --mixed, and --hard reset modes.

Prerequisites

1. Install Git on your system.
2. Set up a Git repository:

```
git init git-reset-lab

cd git-reset-lab
```

```
PS E:\> git init git-reset-lab
Initialized empty Git repository in E:/git-reset-lab/.git/
PS E:\> cd git-reset-lab
```

Steps

1. Set Up the Repository

1. Create and commit an initial file:

```
echo "Line 1" > file.txt

git add file.txt

git commit -m "Initial commit: Add Line 1"
```

```
PS E:\git-reset-lab> echo "Line 1" > file.txt
PS E:\git-reset-lab> git add file.txt
PS E:\git-reset-lab> git commit -m "Initial commit: Add Line 1"
[master (root-commit) 926b48c] Initial commit: Add Line 1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file.txt
```

2. Add a second change:

```
echo "Line 2" >> file.txt

git commit -am "Add Line 2"
```

```
PS E:\git-reset-lab> echo "Line 2" >> file.txt
PS E:\git-reset-lab> git commit -am "Add Line 2"
[master 49317d3] Add Line 2
 1 file changed, 0 insertions(+), 0 deletions(-)
```

3. Add a third change:

```
echo "Line 3" >> file.txt

git commit -am "Add Line 3"
```

```
PS E:\git-reset-lab> echo "Line 3" >> file.txt
PS E:\git-reset-lab> git commit -am "Add Line 3"
[master df69478] Add Line 3
 1 file changed, 0 insertions(+), 0 deletions(-)
```

4. Check the commit history:

```
git log –oneline
```

```
PS E:\git-reset-lab> git log --oneline
df69478 (HEAD -> master) Add Line 3
49317d3 Add Line 2
926b48c Initial commit: Add Line 1
```

---

2. Use git reset --soft

This mode moves the HEAD pointer to an earlier commit but keeps the changes in the staging area.

1. Reset to the second commit:

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

```
PS E:\git-reset-lab> git reset --soft HEAD~1
```

2. Check the commit history:

git log –oneline

```
PS E:\git-reset-lab> git log --oneline
49317d3 (HEAD -> master) Add Line 2
926b48c Initial commit: Add Line 1
```

3. Verify the staged changes:

git status

```
PS E:\git-reset-lab> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file.txt
```

4. If needed, re-commit the changes:

git commit -m "Recommit Line 3"

```
PS E:\git-reset-lab> git commit -m "Recommit Line 3"
[master da59d0c] Recommit Line 3
 1 file changed, 0 insertions(+), 0 deletions(-)
```

---

3. Use git reset --mixed

This mode moves the HEAD pointer and unstages the changes but keeps them in the working directory.

1. Reset to the first commit:

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

```
PS E:\git-reset-lab> git reset --mixed HEAD~1
Unstaged changes after reset:
M       file.txt
```

2. Check the commit history:

```
git log –oneline
```

```
PS E:\git-reset-lab> git log --oneline
49317d3 (HEAD -> master) Add Line 2
926b48c Initial commit: Add Line 1
```

3. Verify the changes in the working directory:

```
git status
```

```
PS E:\git-reset-lab> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

4. If needed, stage and re-commit:

```
git add file.txt
```

git commit -m "Recommit Line 2 and Line 3"

```
PS E:\git-reset-lab> git add file.txt
PS E:\git-reset-lab> git commit -m "Recommit Line 2 and Line 3"
[master 6f74d13] Recommit Line 2 and Line 3
 1 file changed, 0 insertions(+), 0 deletions(-)
```

---

4. Use git reset --hard

This mode moves the HEAD pointer and discards all changes in the staging area and working directory.

1. Reset to the initial commit:

git reset --hard HEAD~1

```
PS E:\git-reset-lab> git reset --hard HEAD~1
HEAD is now at 49317d3 Add Line 2
```

2. Check the commit history:

git log –oneline

```
PS E:\git-reset-lab> git log --oneline
49317d3 (HEAD -> master) Add Line 2
926b48c Initial commit: Add Line 1
PS E:\git-reset-lab> cat file.txt
Line 1
Line 2
```

3. Verify the working directory:

cat file.txt

```
PS E:\git-reset-lab> cat file.txt
Line 1
Line 2
```

---

5. Use git reset with a Commit Hash

1. Add some changes for demonstration:

```
echo "Line 3" >> file.txt

git commit -am "Add Line 3"
```

```
PS E:\git-reset-lab> echo "Line 3" >> file.txt
PS E:\git-reset-lab> git commit -am "Add Line 3"
[master cda2694] Add Line 3
 1 file changed, 0 insertions(+), 0 deletions(-)
```

2. Get the commit hash for the initial commit:

```
git log –oneline
```

```
PS E:\git-reset-lab> git log --oneline
cda2694 (HEAD -> master) Add Line 3
49317d3 Add Line 2
926b48c Initial commit: Add Line 1
```

3. Reset to the initial commit using the hash:

```
git reset --hard <commit-hash>
```

```
PS E:\git-reset-lab> git reset --hard 49317d3
HEAD is now at 49317d3 Add Line 2
```

4. Verify the working directory and commit history:

```
git log --oneline

cat file.txt
```

```
PS E:\git-reset-lab> git log --oneline
49317d3 (HEAD -> master) Add Line 2
926b48c Initial commit: Add Line 1
```

Summary of Commands

Mode	Effect	Command Example
--soft	Moves HEAD, keeps changes staged.	git reset --soft HEAD~1
--mixed	Moves HEAD, unstages changes, keeps them in working dir.	git reset --mixed HEAD~1
--hard	Moves HEAD, discards all changes in staging and working dir.	git reset --hard HEAD~1