Git Under the Hood - Recovery Addendum

This extra section explains how to recover commits and branches after a destructive reset. This is one of the most useful Git recovery skills you can learn.

Recovering a branch after git reset --hard

Core idea: git reset --hard usually does not immediately delete commits. It moves the branch pointer backward. The commits often still exist and can be recovered using reflog.

Step 1: Create a branch with some work

```
git checkout -b recovery-demo

echo "important work" >> recover.txt
git add .
git commit -m "important work"

git log --oneline --graph
```

At this point the branch points to the latest commit containing your work.

Step 2: Simulate disaster

```
git reset --hard HEAD~1

git log --oneline --graph
```

The commit appears gone. This is the scary moment where many developers think the work has been destroyed.

Step 3: Use reflog to find the lost commit

```
git reflog
```

You should see something similar to:

```
abc1234 HEAD@{1}: commit: important work
```

Reflog remembers where HEAD and branch references used to point.

Step 4: Recover the lost commit

```
git checkout abc1234
```

You are now standing directly on the recovered commit in detached HEAD state.

Step 5: Restore the branch

```
git branch rescued-branch
git checkout rescued-branch
```

The branch is now fully restored and points to the recovered commit again.

Important mental model:

```
Before reset:
```

```
main -> A -> B -> C

After reset:

main -> A -> B

Commit C still exists.
Nothing points to it anymore.
```

Git usually loses references first, not data. That is why reflog is so powerful.

Important warning: Unreferenced commits can eventually be garbage collected. Recovery is usually possible for a while, but not forever.