GIT DAY-2

Thursday, 23 October 2025 6:41 PM

1️⃣ Interactive Rebase (git rebase -i)

Think of **commits as steps in a story**. Sometimes your story has unnecessary steps, repeated steps, or steps in the wrong order. git rebase -i lets you **tidy up your story** before sharing it.

How it works:

git checkout feature-branch
git rebase -i HEAD~3

- HEAD~3 = last 3 commits.
- A list of your last 3 commits will open in your editor.

What you can do in interactive mode:

- **pick**: Keep the commit as is.
- **squash (or s)**: Combine this commit with the one above.
- **Reorder commits**: Move lines up/down to change the order.
- **Edit commit messages**: Fix typos or make messages clearer.
- **Delete commits**: Remove unnecessary commits entirely.

Analogy:

- Imagine you wrote 3 lines in your diary: “Made login page”, “Fixed login bug”, “Updated README”. You realize the bug fix and login page can be combined into one clean entry. git rebase -i lets you merge and clean them up.

2️⃣ Cherry-picking (git cherry-pick)

Sometimes you **only want one specific change** from another branch, not everything. That’s what cherry-pick does.

git checkout target-branch
git cherry-pick <commit-hash>

- <commit-hash> is the unique ID of the commit you want.
- Git will copy that commit and apply it to your current branch.

Analogy:

- Think of a cherry tree. You don’t take all the fruit; you pick the cherry you want. Similarly, you pick only the commit you need.

3️⃣ Undoing Changes

a) git reset – Go back in time

- Moves your branch pointer backwards, basically **undoing commits locally**.

git reset --soft HEAD~1 # Undo last commit, keep changes staged
git reset --mixed HEAD~1 # Undo last commit, unstage changes
git reset --hard HEAD~1 # Undo last commit and discard changes completely

Warning: Be careful with --hard! You can lose work permanently.

Analogy:

- Imagine you wrote a paragraph on paper.
  - soft = erase the paragraph from your notebook but keep the text on a sticky note.
  - mixed = erase and throw the sticky note in front of you.
  - hard = erase and burn the sticky note—gone forever.

b) git revert – Safe undo for shared changes

- Creates a **new commit** that **reverses a previous commit**, without touching history.

git revert <commit-hash>

Analogy:

- You wrote a wrong sentence in a shared Google Doc. Instead of deleting it, you write a new line that says: “Undo: wrong sentence”. Everyone still sees the history.

- You wrote a wrong sentence in a shared Google Doc. Instead of deleting it, you write a new line that says: “Undo: wrong sentence”. Everyone still sees the history.

Key difference:

- reset = rewrites history (good for local work).
- revert = adds a new commit (safe for shared branches).

4️⃣ Tagging (git tag)

- Tags are **labels for important commits**, like milestones or releases.

```
git tag v1.0          # Simple tag
git tag -a v1.0 -m "Version 1.0"   # Annotated tag with message
git push origin --tags   # Push tags to remote
```

Analogy:

- Think of it like marking **chapters in a book**. You can quickly say, “This is version 1.0” without remembering the commit ID.

✅ Summary

Concept	Command Example	What it does	Analogy
Interactive Rebase	git rebase -i HEAD~3	Clean up last N commits	Fix your diary entries
Cherry-pick	git cherry-pick <commit-hash>	Apply a single commit from another branch	Pick a cherry from the tree
Undo Commit	git reset	Remove commits locally	Erase your paragraph (soft/mixed/hard)
Undo Safely	git revert	Reverse a commit safely	Add a “correction line” in shared doc
Tagging	git tag -a v1.0 -m "msg"	Mark important commits	Label a chapter in a book