# Git Rebase

## What is Rebase?

- **Rebase** moves your branch commits on top of another branch.
- Instead of merging (which creates a new merge commit), rebase **replays commits in a linear history**.
- Makes history cleaner (no unnecessary merge commits).

---

## Syntax

```
# Rebase current branch onto main
git checkout feature-branch
git rebase main
```

👉 This takes all commits from `feature-branch` and applies them **on top of `main`**.

---

## Example Flow

```
main:    A---B---C
feature:       D---E
```

After `git rebase main` on feature:

```
main:    A---B---C
feature:           D'---E'
```

✅ History is linear, as if you started working after commit `C`.

---

## Use Cases

- Keeping feature branches updated before merging.
- Avoiding “merge hell” in large projects.
- Maintaining **clean project history** in open source (many maintainers ask for rebased PRs).

⚠️ **Don’t rebase public branches** (others already pulled them). Safe to use on personal/local branches.

---

---

# 🔹 Git Squash (Merging Multiple Commits into One)

## What is Squashing?

- Combines multiple commits into a **single commit**.
- Used to clean up messy history (e.g., `fix typo`, `update readme`, `oops bugfix`).

---

## How to Squash

git rebase -i HEAD~3

(Interactive rebase of last 3 commits)

You’ll see something like:

pick abc123 First commit
pick def456 Second commit
pick ghi789 Third commit

- 

Change to:

pick abc123 First commit
squash def456 Second commit
squash ghi789 Third commit

- 
- Git will open editor to combine commit messages.
- Final result: **1 commit instead of 3**.

---

### Example

Before:

* ghi789 (HEAD -> feature) Added error handling
* def456 Fixed typo
* abc123 Initial API implementation

After squashing:

* xyz111 (HEAD -> feature) Initial API implementation + fixes + error handling

---

---

## 🔹 Git Merge vs Rebase vs Squash

| Command | What It Does | History |
|---|---|---|
| **Merge** | Combines two branches, creates a **merge commit** | Non-linear (keeps all commits as-is) |
| **Rebase** | Moves branch commits on top of another | Linear, but keeps all commits separate |
| **Squash** | Compresses multiple commits into one | Linear + cleaner history |

---

### DevOps Use Case

- **Merge**: When you want to preserve full history (audit, compliance).
- **Rebase**: When you want clean, linear commit history before merging a PR.

- **Squash**: When a feature branch has many small commits → squash into one meaningful commit before merging into `main`.

---

✅ Quick Summary Commands:

```
# Rebase feature branch on main
git checkout feature
git rebase main

# Interactive rebase (squash last 3 commits)
git rebase -i HEAD~3

# Merge feature into main (with merge commit)
git checkout main
git merge feature
```