Git Merge Strategies

# Merge Commit (default strategy)

Creates a new commit that combines the changes from the two branches being merged.

Preserves the history of both branches and results in a non-linear history.

Used by running `git merge branchname` without specifying any additional options.

# Fast-Forward Merge

Moves the pointer forward if the branch has not diverged, making the history linear and clean.

Does not create a merge commit.

Explicitly request this by using `git merge --ff branchname`. If fast-forward is not possible, Git defaults to a merge commit.

# Squash Merge

Combines all changes from a feature branch into a single commit which is then merged.

Results in a cleaner, linear history by condensing all commits into one.

Often used in pull request workflows. Use `git merge --squash branchname`.

# Rebase

Not a merge strategy, but an alternative that rewrites the commit history by placing commits on top of another branch.

Changes the base of your branch to be the tip of another branch, transferring your changes on top.

Execute with `git rebase branchname`. Leads to a cleaner, more linear history.

# No-Fast-Forward Merge

Forces Git to create a new merge commit even if the merge could be performed with a fast-forward.

Useful for preserving the context of an isolated feature branch.

Use by running `git merge --no-ff branchname`.