Advance Git & GitHub for DevOps Engineers.

Introduction to Git Branching

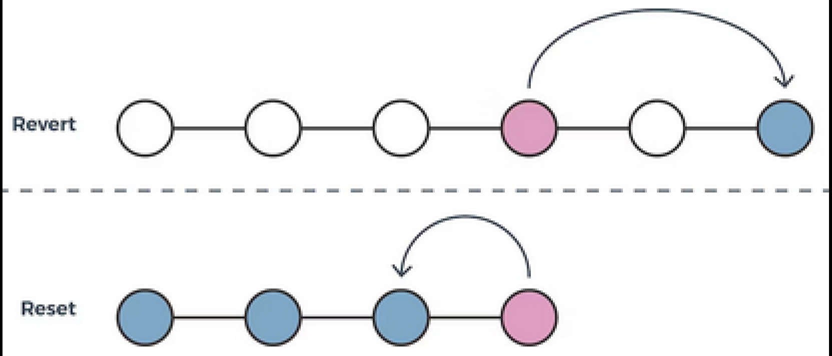
Git branching is a powerful feature that allows developers to work on different versions of the same codebase simultaneously. It enables multiple people to work on the same project without interfering with each other's work.

In Git, a branch is essentially a separate line of development. It allows developers to experiment with new features or make changes without affecting the main codebase. Branches can be merged back into the main codebase once they are complete.

Understanding Git Revert and Reset

Git revert and reset are two commands that allow developers to undo changes made to the codebase. Revert creates a new commit that undoes the changes made in a previous commit while keeping the original commit in the history. Reset, on the other hand, removes the commit and all subsequent commits, effectively erasing them from the history.

Revert is useful when you want to undo a specific change without affecting the rest of the codebase. Reset is more drastic and should be used carefully, as it permanently removes commits and changes.



Example

1. Create a new dir and add initialize git using git `init`, or clone your repo.
2. Create a new file `` `verison1.txt` `` with content “This is the first feature of our application" in the master branch
3. Switch to dev branch: git checkout -b dev
4. Make three commits like this:

- “This is the bug fix in the development branch” → Add some content to file
- `git add version01.txt` → Stage the changes
- `git commit -m "Added feature2 in development branch"` → Commit the changes

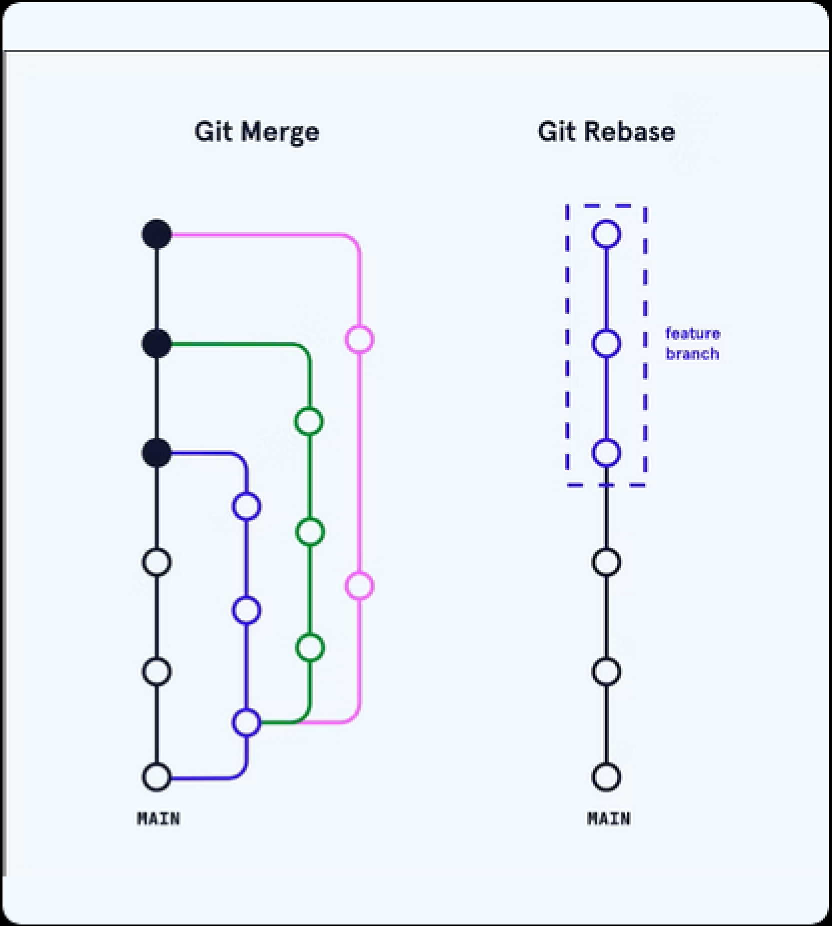
5. Restore the file to a previous version where the content should be of the first commit:

   `git revert HEAD~3..HEAD` #→ we have made 3 commits so, reverting the last 3 commits.

Git Rebase vs Merge

Git rebase and merge are two ways to combine changes from different branches. Merge combines the changes from two branches into a new commit, while rebase moves the entire branch onto another branch, essentially rewriting the commit history.

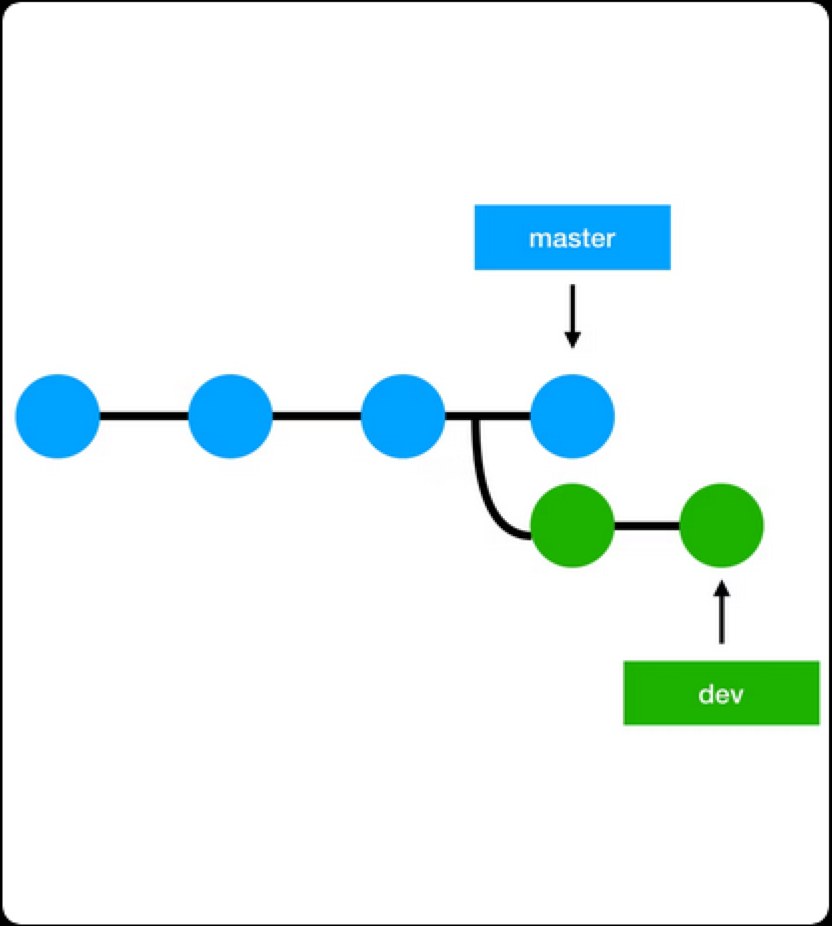
Rebase is useful when you want to keep a linear commit history and avoid merge conflicts. However, it can be risky if not done correctly, as it can cause confusion and make it difficult to track changes. Merge is safer and easier to understand, but can result in a more complex commit history.



Demonstrating Branching

To demonstrate the concept of branching, we will create two branches: `master` and `dev`. The master branch will contain the main codebase, while the dev branch will be used to experiment with new features.

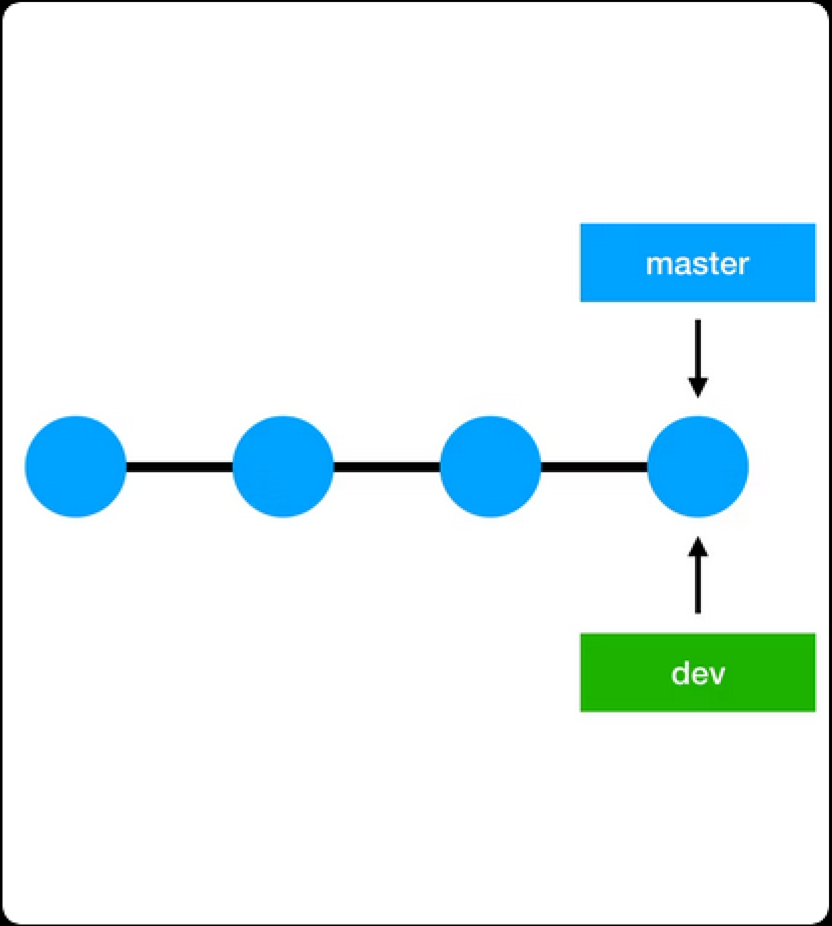
We will add some changes to the dev branch and then merge that branch into the master branch. We will also try using git rebase to see the difference it makes.



Example: git merge

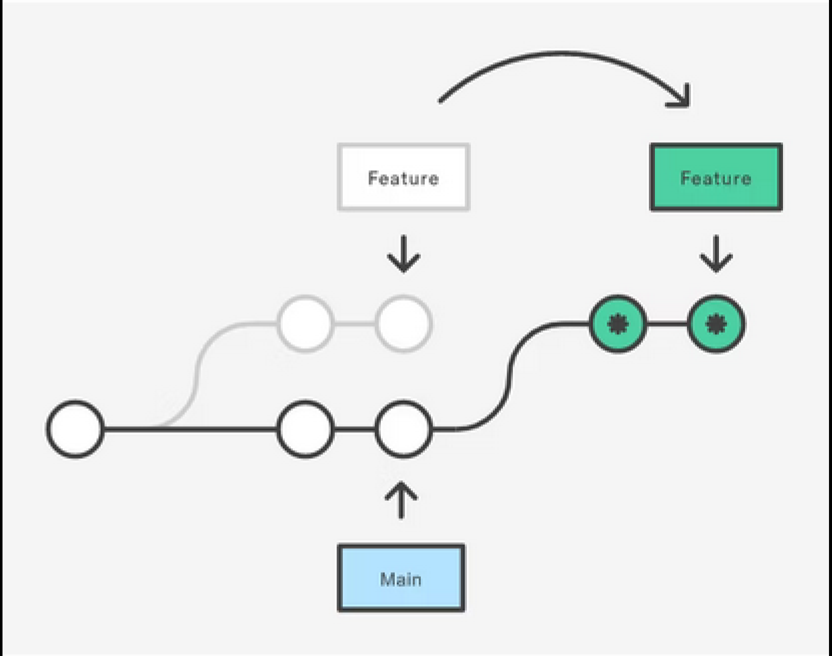
- Adding Changes to Dev Branch and Merging into Master:

1. Switch to the dev branch using the command '`git checkout dev'`.
2. We will then make some changes to the codebase and commit those changes using `git commit -m “commit message”`.
3. Switch back to the master branch using `git checkout master`
4. Merge the changes from the dev branch using `git merge dev`. This will create a new commit on the master branch that includes the changes from the dev branch.



Example: Git Rebase

1. First, switch back to the dev branch using `git checkout dev`.
2. Make some additional changes to the codebase and commit those changes using `git commit -m "commit message"`.
3. Switch back to the master branch using `git checkout master` .
4. Use the rebase command `git rebase dev`. This will move the entire master branch onto the dev branch, effectively rewriting the commit history.


Feature
Feature
Main