knóldus

REBASE

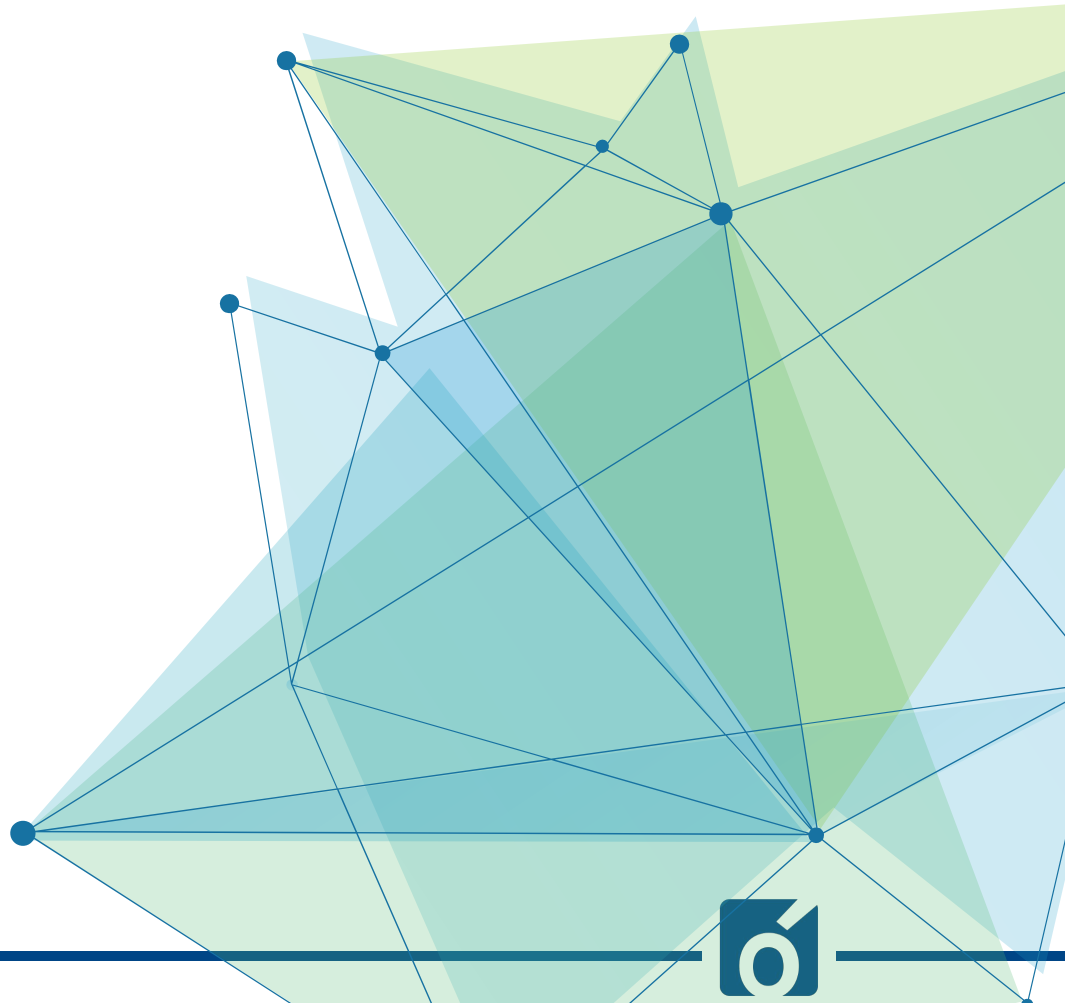
Presented By : Group 9

Group 9 Members

1. Gaurav Kumar
2. Himanshu Gupta
3. Lucky Kumar
4. Nitesh Kumar
5. Shivam Saxena

Agenda

- Introduction
- Git Rebase Command
- Some Basic Commands
- Closing



Introduction

- Rebase is an action in git that allow you to rewrite commit from one branch to another branch.
- Rebasing is most useful and easily visualized in the context of features branching workflow
- Two method of Rebase are:

1

Regular Rebase

With a regular rebase you can update your feature branch with the default branch

2

Interactive Rebase

It is simply used to modify commit. Such as edit or delete commit

How To Rebase

When you made some commits on a feature branch (test branch) and some in the master branch. You can rebase any of these branches. Use the git log command to track the changes (commit history). Checkout to the desired branch you want to rebase. Now perform the rebase command as follows:

Syntax:

```
$git rebase <branch name>
```

Regular Rebase

With a regular rebase you can update your feature branch with the default branch (or any other branch). This is an important step for Git-based development strategies. You can ensure that the changes you're adding to the codebase do not break any existing changes added to the target branch after you created your feature branch.

Regular Rebase Example

To update your branch my-feature-branch with your default branch (here, using main):

1. Fetch the latest changes from main:

   git fetch origin main

2. Checkout your feature branch:

   git checkout my-feature-branch

3. Rebase it against main:

   git rebase origin/main

4. Force-push to your branch.

Interactive Rebase

- It is a tool that provide more manual control of your history revision process.
- When the Rebase Command accepts an -i argument then it will be in interactive mode
- Interactive Rebase can be used to modify commits .
- For example:
  - Amend a commit message
  - Squash
  - Edit or Delete Commits

Basic Commands

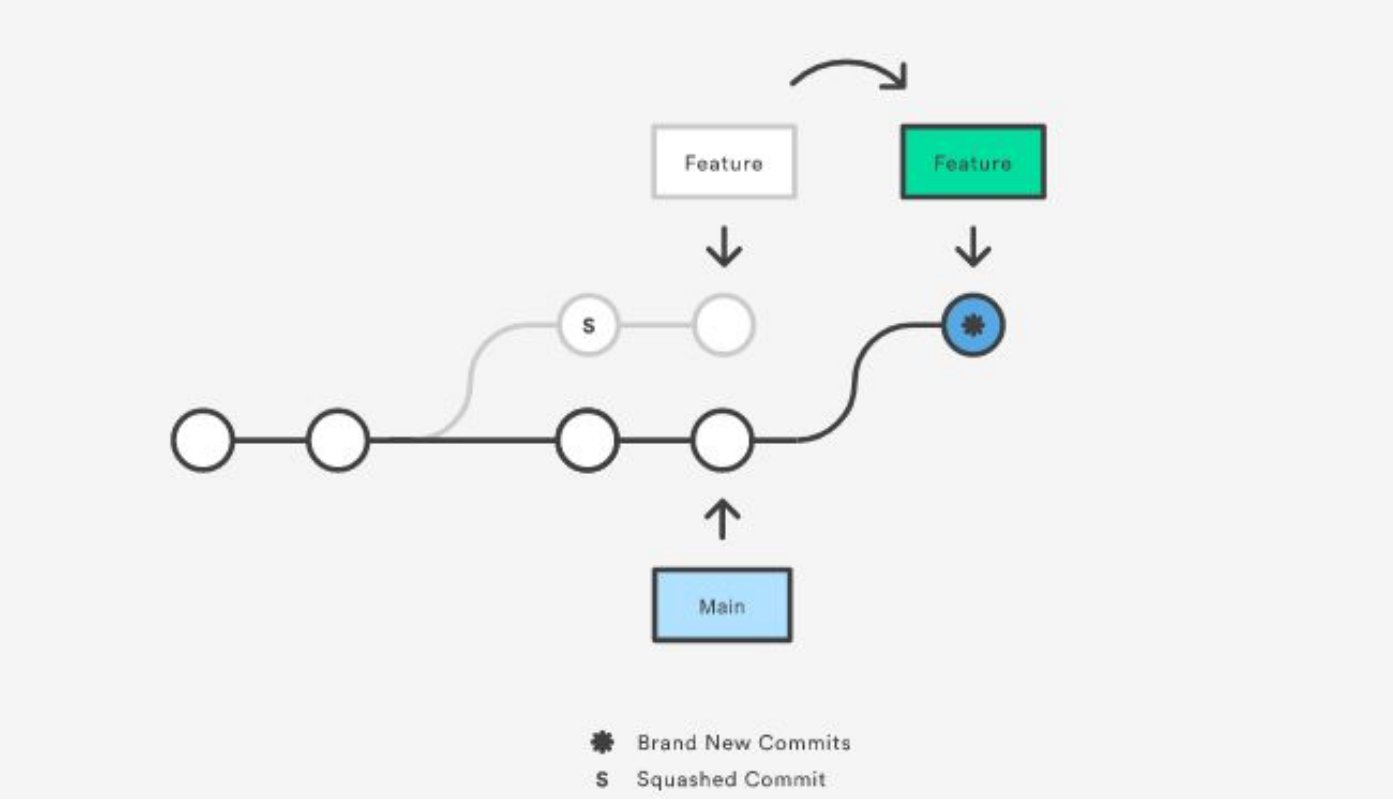
- **git commit - -amend**

This command is convenient way to modify the most recent commit.

- **Squash commit for clean history**

The s “Squash” allow you to specify which commit you want to merge into previous commit

Squash Commit For Clean History



Force-Push

When you perform more complex operations, for example, squash commits, reset or rebase your branch, you must force an update to the remote branch. These operations imply rewriting the commit history of the branch. To force an update, pass the flag `--force` or `-f` to the `push` command. For example:

```
git push --force origin my-feature-branch
```

Forcing an update is **not** recommended when you're working on shared branches.

How a "Force Push" Works ?

As described above, Git will normally only allow you to push your changes if you have previously updated your local branch with the latest commits from its remote counterpart. Only when you are up-to-date will you be able to push your own new commits to the remote.

Git prevents you from overwriting the central repository history by refusing push requests when they result in a non-fast-forward merge. So, if the remote history has diverged from your history, you need to pull the remote branch and merge it into your local one, then try pushing again.

Safety Rules

Since it is so easy to destroy or at least impede your colleagues' work, here are a few "safety rules" around git push --force:

Use --force-with-lease instead of --force.

The push command has another option called --force-with-lease. This helps to make sure that you are at least not overwriting work from others: it will present an error message and refuse to push if the remote was modified since you last fetched.

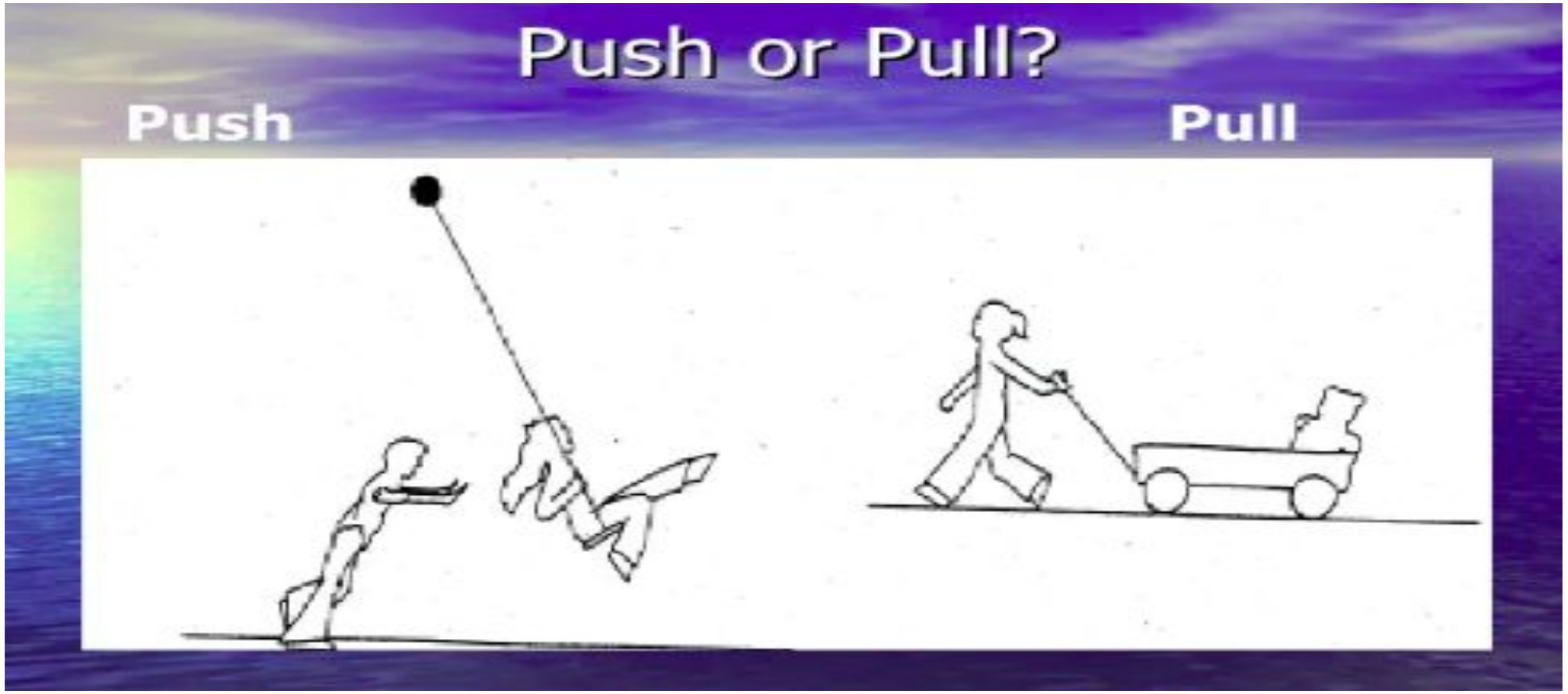
Don't use it on shared history

Whenever you have pushed commits to a remote branch that is shared with your team, you should try NOT to use force push. If, on the other hand, you were working on a feature branch that only you yourself are using, then of course feel free to step on the gas and use the --force option.

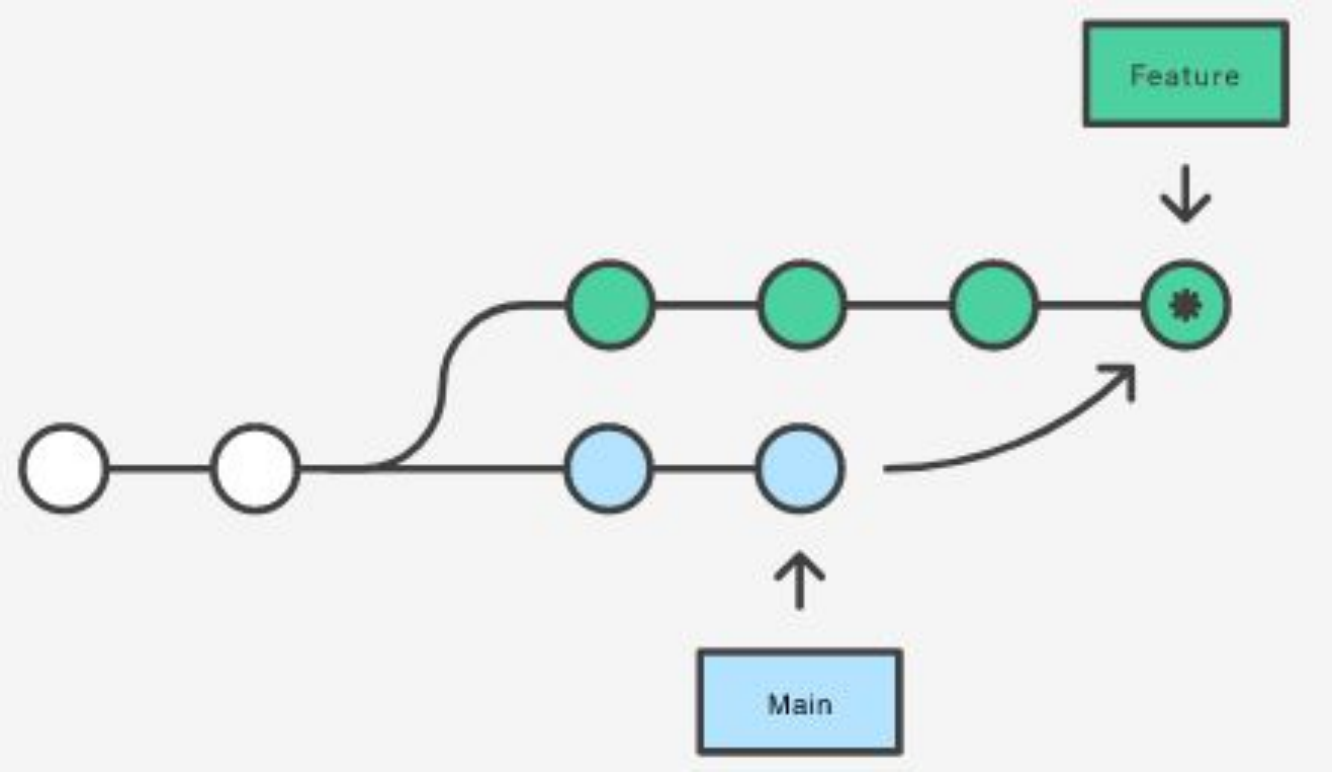
Consider using git revert instead

The basic need for a tool to correct a mistake that you've already pushed, of course, remains. However, consider using a tool that does NOT rewrite commit history, like git revert for example. This provides a much less obtrusive way to undo a mistake.

Understand Difference In Pull or Push by this Diagram

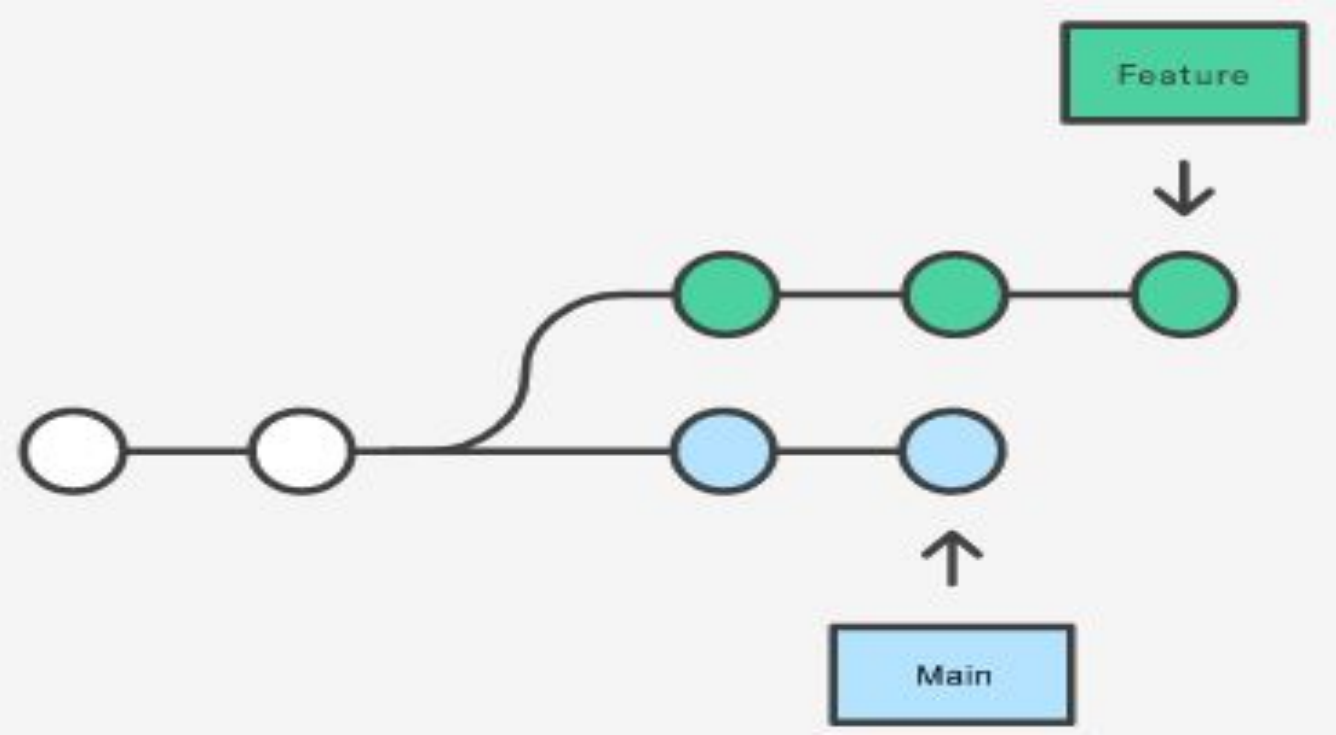


Git Merge vs Git Rebase



What they do?

The first thing to understand about git rebase is that it solves the same problem as git merge. Both of these commands are designed to integrate changes from one branch into another branch—they just do it in very different ways.

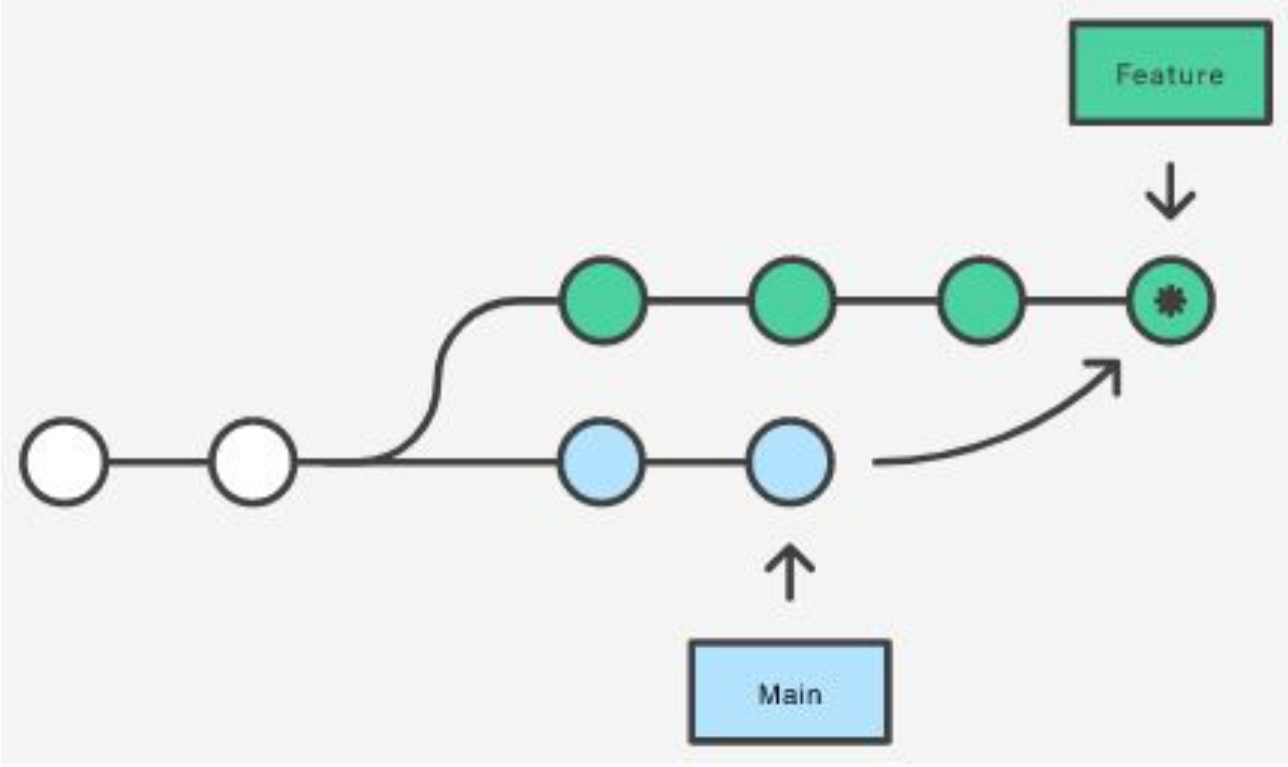


Easiest option? Merge!

The easiest option is to merge the main branch into the feature branch using something like the following command:

```
git merge feature main
```

This creates a new “merge commit” in the feature branch that ties together the histories of both branches, giving you a branch structure that looks like this:

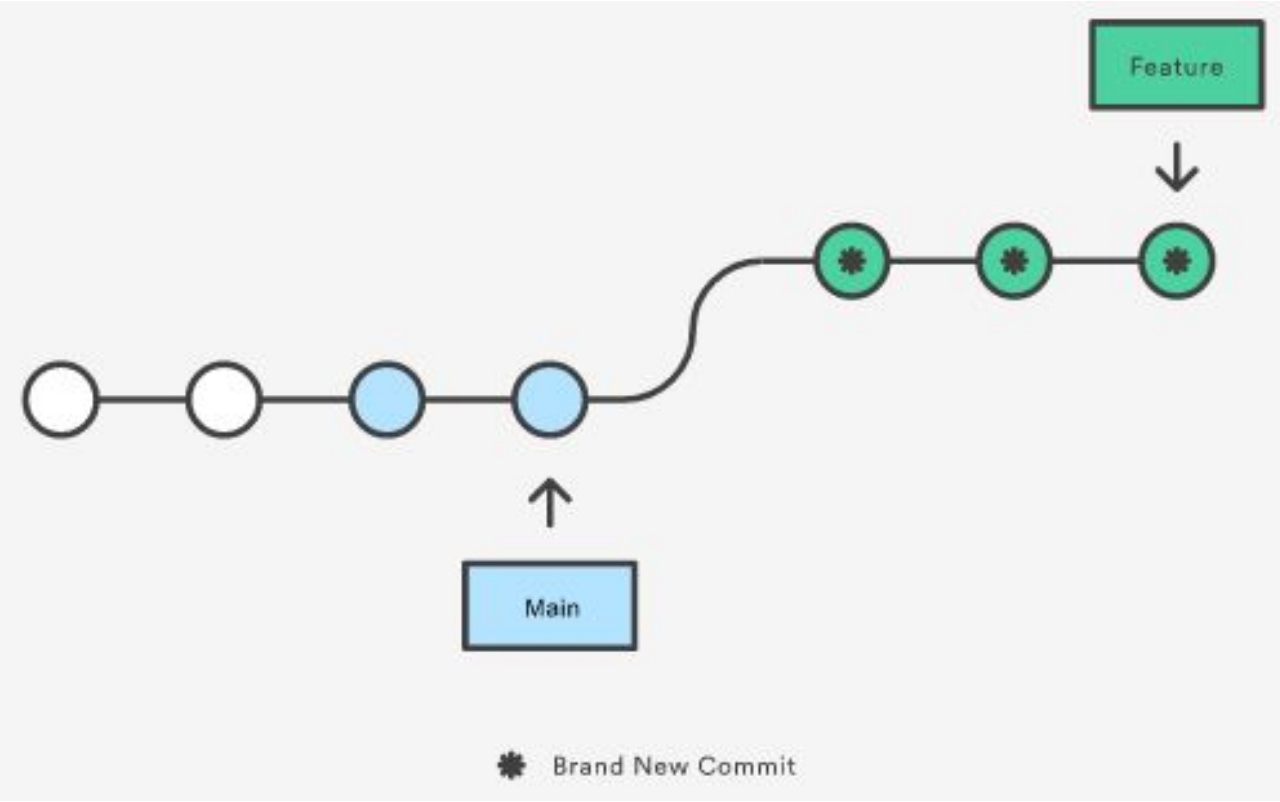


Why Rebase!

As an alternative to merging, you can rebase the feature branch onto main branch using the following commands:

```
git checkout feature
git rebase main
```

The major benefit of rebasing is that you get a much cleaner project history. First, it eliminates the unnecessary merge commits required by git merge. Second, as you can see in the diagram, rebasing also results in a linear project history, you can follow the tip of feature to the beginning of the project without any forks.



Thank You!