Introduction to Git and GitHub

Points to be covered today:-

- Git log,tagging and comparing commits
- Parallel Development in a repository(branching,merging and rebasing)

Git Log

The Git Log tool allows you to view information about previous commits that have occurred in a project.

The n most recent commits can be given by

git log --n

Filter Commits by Author name/commiter name-

Git log --author <name>

Filter commits by date

Git log --before <date>

Git log --after <date>

View Summary of Changes for Each Commit

To view a summary of the changes made in each commit (# of lines added, removed, etc), use the –stat option:

$ git log --stat

Any other

git help log

Git tagging

Tags are ref's that point to specific points in Git history.

Creating a tag

git tag <tagname>

Git supports two different types of tags, annotated and lightweight tags.
The previous example created a lightweight tag.

Lightweight tags and Annotated tags differ in the amount of accompanying meta data they store.

Annotated Tags and Lightweight Tags

Annotated tags are stored as full objects in the Git database.

They store extra meta data such as: the tagger name, email, and date. Similar to commits and commit messages Annotated tags have a tagging message.

To list stored tags in a repo execute the following:

git tag

This will output a list of tags

Sharing and Pushing Tags to Remote

Sharing tags is similar to pushing branches. By default, git push will not push tags. Tags have to be explicitly passed to git push.

Checking Out Tags

You can view the state of a repo at a tag by using the git checkout command.

git checkout v1.4

Deleting Tags

Deleting tags is a straightforward operation. Passing the -d option and a tag identifier to git tag will delete the identified tag.

```
$ git tag

$ git tag -d v1

$ git tag
```

v3

Comparing Commits

Comparing changes with git diff

Diff is a function that takes two input data sets and outputs the changes between them. git diff is a multi-use Git command that when executed runs a diff function on Git data sources.

Comparison input

diff --git a/diff_test.txt b/diff_test.txt

This line displays the input sources of the diff. We can see that a/diff_test.txt and b/diff_test.txt have been passed to the diff.

Highlighting changes

1. git diff --color-words

git diff also has a special mode for highlighting changes with much better granularity:

--color-words.

Parallel Development in a Repository

Branching allows us to work in multiple parallel workspaces.

Let's create two branches. Create one with the name "issue2" and another with the name"issue3", then switch over to "issue2".

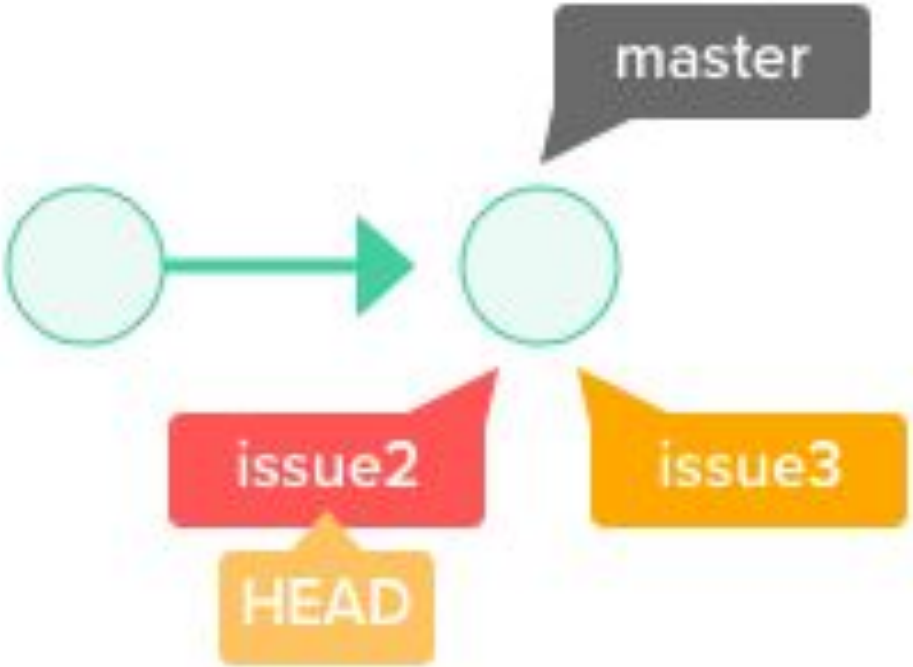
$ **git branch issue2**

$ **git branch issue3**

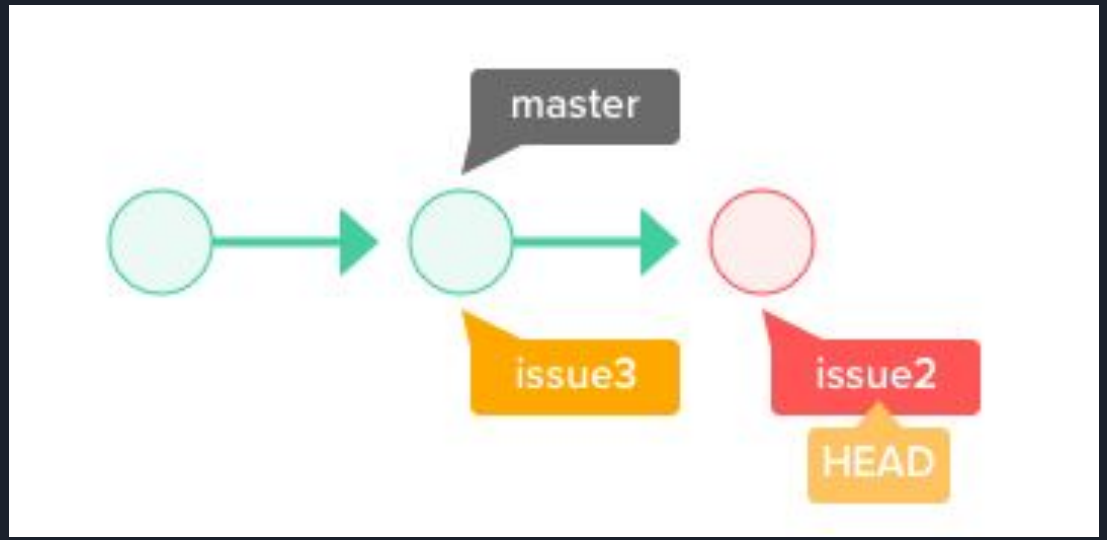
$ **git checkout issue2**

Switched to branch 'issue2'

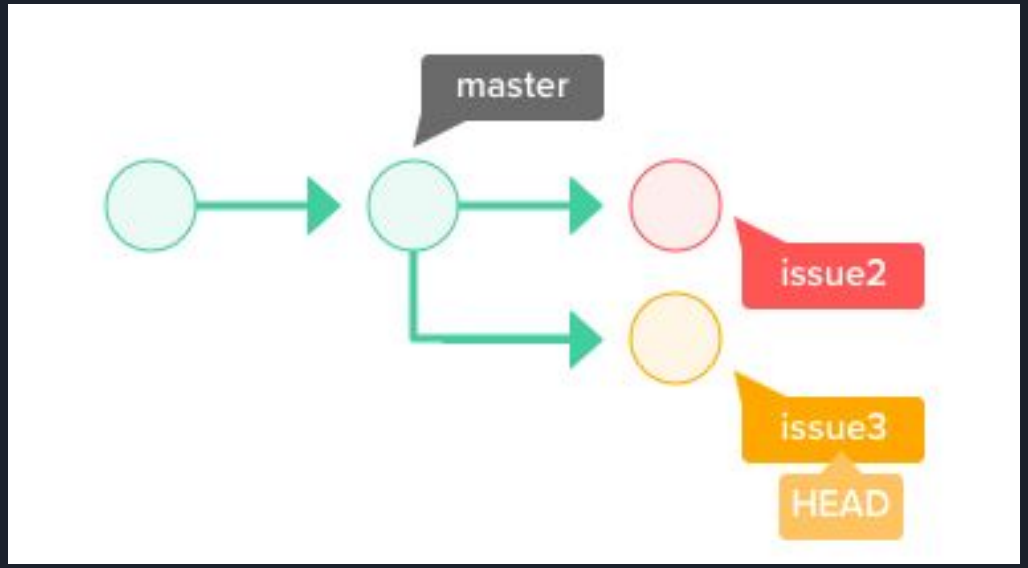
$ **git branch**


master
issue2
issue3
HEAD

- git add myfile.txt
- git commit -m "append description of the commit command"

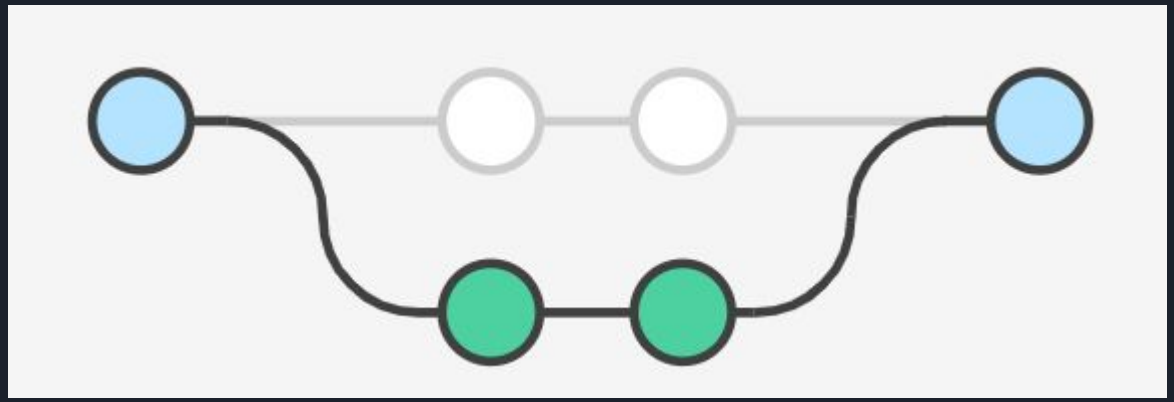


We have now added two different line of texts to two different branches in parallel.



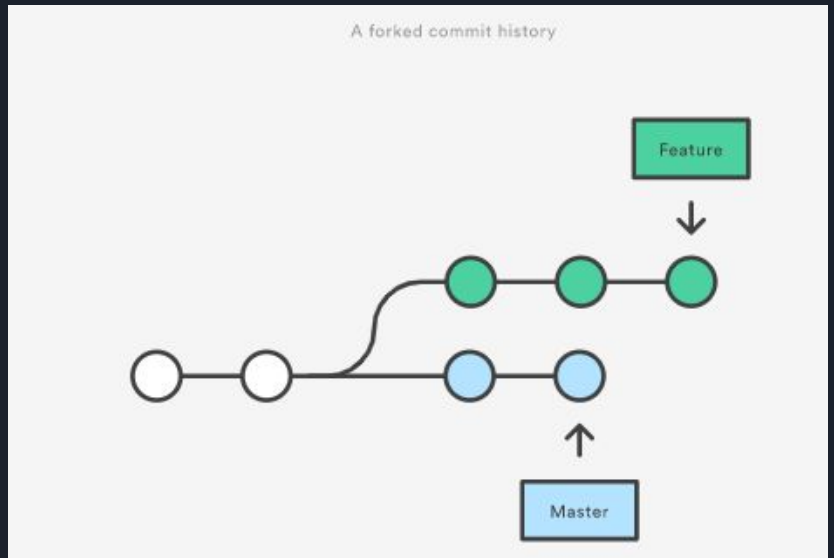
Branching

As we covered in the last slides



Merging and Rebasing

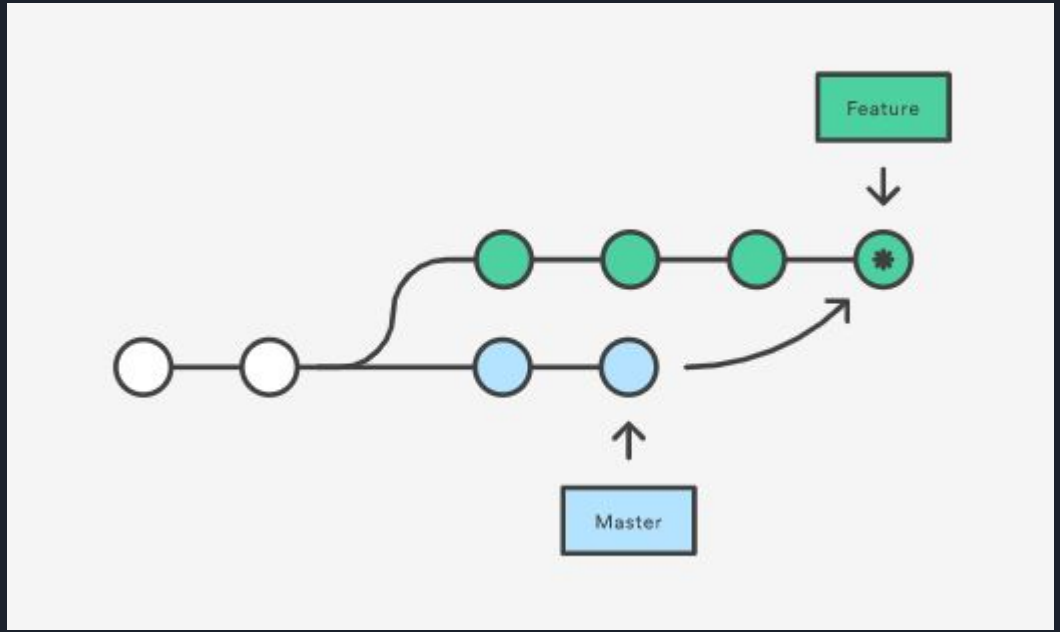
The first thing to understand about git rebase is that it solves the same problem as git merge. Both of these commands are designed to integrate changes from one branch into another branch—they just do it in very different ways.



The Merge Option

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

git merge feature master

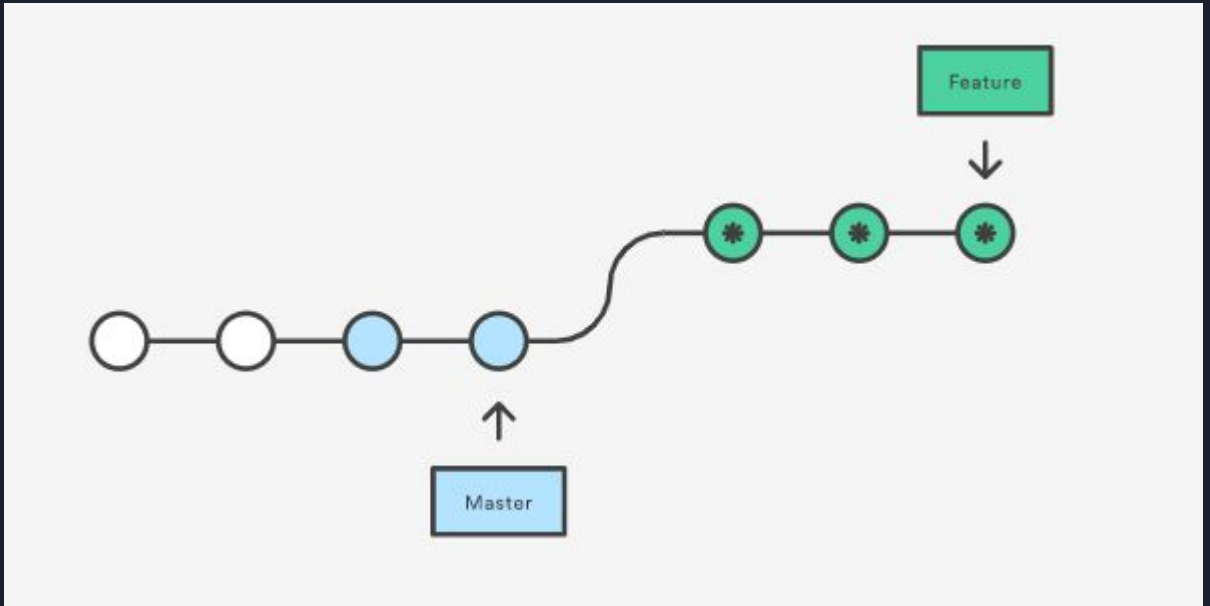


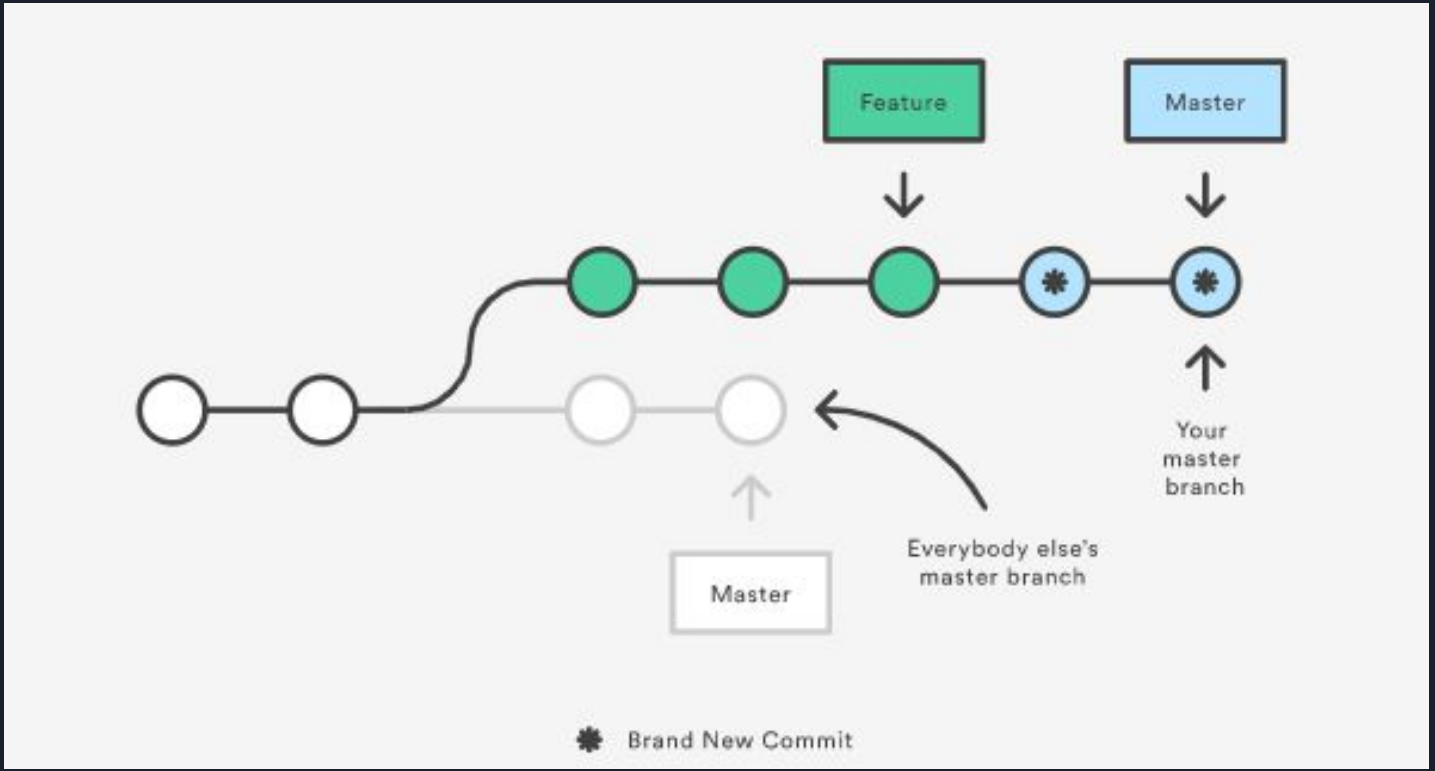
The Rebase Option

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

git checkout feature

git rebase master




Feature
Master
Your
master
branch
Everybody else's
master branch
Master
Brand New Commit

Force Pushing

If you try to push the rebased master branch back to a remote repository, Git will prevent you from doing so because it conflicts with the remote master branch. But, you can force the push to go through by passing the --force flag, like so:

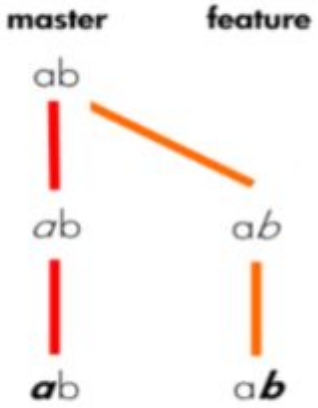
```
git push --force
```

Let Us Summarise

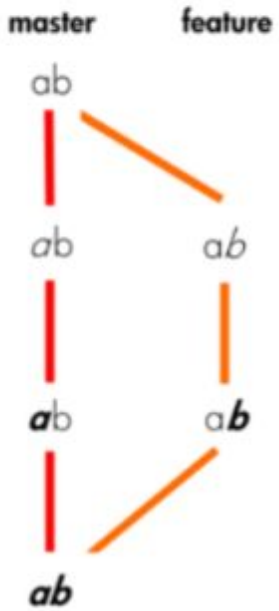
And that’s all you really need to know to start rebasing your branches. If you would prefer a clean, linear history free of unnecessary merge commits, you should reach for git rebase instead of git merge when integrating changes from another branch.

On the other hand, if you want to preserve the complete history of your project and avoid the risk of re-writing public commits, you can stick with git merge. Either option is perfectly valid, but at least now you have the option of leveraging the benefits of git rebase.

Commits



Merge



Rebase