**52)Add Collaborators**

When we push code into the main branch it is also in production. So, if someone pushes a code with a bug onto the main branch the entire production will be in problem. So we use pull requests to merge code from feature branch to main branch. We need to add collaborators to review our code to be pushed into the main branch.

Settings

Manage Access

Invite a Collaborator

Add collaborators here

They can review the code and commit it to the main branch.

To make sure no one can commit the code to main branch

Settings

Branches

Branch Protection Rules

In branch name pattern add “main” then check “Require pull request reviews before merging” and “include administrators”

**53)Git Forking**

When you are having large open-source projects with lots of contributors.

They employ this forking strategy or workflow where there might be a handful of actual maintainers.

They cannot add thousands of people as direct contributors or collaborators.

This forking workflow enables anybody to try and contribute to the repository.

There is no need for permission. You can make your own copy. You try making changes and then you make a PR.

Anybody can make a pull request.

GitHub and other similar tools allow us to create personal copies of other people’s repositories.

We call those copies as a fork of the original.

When we fork a repo, we basically asking GitHub "Make me my own copy of this repo please"

Like pull requests, forking is not a git native feature.

The ability to fork is implemented by GitHub.

git clone URL

We clone the repository to our local repository, do some edits and try to push to that repository.

We get an error we can’t commit to that repository.

git push Won’t work

So, what we do is,

We go to the repository we want to fork and click fork on top right.

It’ll create a repository of that repository in our account.

We make changes and push it into our account

**54)Git Forking cont.**

We have learnt how to fork the others repository.

We have our own copy of the repo; we can do whatever we want!

We can clone my fork and make changes, add features, and break things without fear of disturbing the original repository

If we want to share my work, we can make a pull request from my fork to the original repo.

This means that a lot of people can fork and can work on the project without having permission to them.

Diagram

Description automatically generated

We fork it to our local repository, and work on the project but what if the organization repository made some changes in this time, the local repo won’t have these changes. So, we create an upstream to constantly update our local files with organization files.

The "Fork & Clone" workflow might seem complicated, but it's extremely common for good reasons!

It allows a project manager to accept contributions from developers all around the world without having to add them as actual owners of the main project repository or worry about giving them all permissions to push to the Repo.

We can do updates to the file and commit them to the local repo and raise a pull request for the changes to be added to the main repository.

The contributors can review these changes and if good they can add it to the main repository.

**55)Git Tagging, Lightweight Tags vs Annotated Tags**

Git Tags main idea is that we can tag a commit so we can label commits by creating a tag, a reference to a moment in time.

Tags are pointers that refer to points in Git history. We can mark a particular moment in time with a tag.

Tags are most often used to mark version releases in projects (v4.1.0, v4.1.1 etc).

Think of tags as branch references that do not change. Once a tag is created, it always refers to the same commit. (When we commit to a branch the branch head moves but tag stays at the same commit).

It's just a label for a commit.

There are two types of tags

1. Lightweight Tags

2. Annotated Tags

Lightweight tags are light weight. They are just a name/label that points to a particular commit.

Lightweight tag is much like a branch that doesn't change - It’s just a pointer to a commit

Annotated tags store extra meta data including the author's name and email, the date, and a tagging message (like a commit message).

Annotated tags, however, are stored as full objects in the git database. It's generally recommended that you create annotated tags, so you have full information.

**56)Semantic Versioning of Software**

Semantic versioning specs outlines a standardized versioning system for software releases.

It provides a consistent way for developers to give meaning to their software releases.

Version consists of three numbers separated by periods. (4.2.1)

4-Major

2-Minor

1-Patch

Typically, the initial release (first release) for any project will be 1.0.0.

**Patch Release (1.0.1)**

Patch releases normally do not contain new features or significant changes. They typically signify bug fixes and other changes that do not impact how the code is used.

**Minor Release (1.1.0)**

Minor releases signify that new features or functionality have been added, but the project is still backwards compatible. No breaking changes. The new functionality is optional and should not force users to rewrite their own code.

**Major Release (2.0-0)**

Major releases signify significant changes that is no longer backwards compatible. Features may be removed or changed substantially

**57)Git Tag Commands**

git tag

Git tag will print a list of au tags in the current repository.

We can search for tags that match a particular pattern by using git tag -l and then passing wildcard. For example, git tag -l "\*beta\*" will print a list of tags that include beta in their name.

git checkout <tag>

To view the state of a repo at a particular tag, we can use git checkout <tag>. This puts us in detached Head.

Let’s clone react repository in our local.

When we git tag It’ll print all tags in the repository.

When we git tag-l It’ll print all tags in the repository.

When we git tag-l beta It’ll print all tags that have beta as its name

When we git tag-l beta\* It’ll print all tags that have beta in its name followed by something

When we git tag-l \*beta\* It’ll print all tags that have beta in between its name

git checkout v17.0.0

Text

Description automatically generated

git diff v17.0.0 v17.0.1 Gives us all file changes between these two versions

**58)Create Lightweight Tag**

To create a tag, we do git tag v1.0.0

Now if we do git tag it’ll display the above tag as v1.0.0

A screenshot of a computer

Description automatically generated

Lightweight Tag has Commit, Author, Date, We can also see what edits we made to the file at the bottom of the pic above.

**59)Create Annotated Tag**

git tag -a v1.0.1 a means annotated

It’ll ask us to write message

Text

Description automatically generated

Annotated Tag has Tagger, Date, Comment, Commit, Author, Date, We can also see what edits we made to the file at the bottom of the pic above.

**60)Push Tags into GitHub Repo**

git remote add origin awebdev/git-tag.git adding origin

Text

Description automatically generated

We have 3 commits in the log

If we just push the commits into GitHub the commits go but tags do not go to the remote repository.

Graphical user interface, text

Description automatically generated with medium confidence

To do that we,(We have three tags)

We can push one tag at a time or all tags at once

**One Tag**

git push origin vl.0.0 This push one tag(v.1.0.0)

git push origin --tags This pushes all tags

Select draft a new release option in Github to name a tag,write its comment on GitHub.