**Git Branching, Merging, and Pull Requests**

**Main Branch:** The default branch is called the main branch, which has some commitments that we’ve made.

**Feature Branch:** But we can create new branches. Each individual branch DO NOT know what the other commits are in other branches, or the main branch. Each branch only keeps track of their own commitments.

Chart

Description automatically generated

**Hot Fix Branch:** Say we are working on the main branch and we see that there’s a bug that we want to fix. We don’t want to affect the main program, so we can create a new branch just to fix the code, and then merge it back.

**1. Git Branching in Codes:**

- List branches: The star \* shows which branch we are currently on

**git branch**

Press q to return from the list

- Create a new branch

**git checkout -b <feature-branch-name>**

- Switch between branches

**git checkout <feature-branch-name>**

We can then work on the new branch. Follow with add and commit (will only save on the branch)

**2. Git Merging:**

First, before merging the code, we can double check and see the code that you’re merging in. To do this:

**git diff <feature-branch-name>** *(what changes have been made. Press q to go back)*

*We can also use* ***git diff*** *when we want to know which changes have been made*

This will show the changes that the main branch does not have.

We can then continue to merge:

**git merge <feature-branch-name>**

**3. Pull request on GitHub:**

1. Switch to feature branch
2. push to GitHub: no upstream branch yet, so use

**git push origin <feature-branch-name-on-GitHub>**

1. Create a pull request (on GitHub)
2. After all comments are resolved, and the branch is merged, the changes are only made on GitHub. Therefore, we’ll have to pull the main branch again down to VS Code if we want to see the change reflected on local computer.

**git pull origin main**

1. Now that we’re done with the feature branch, we need to go ahead and delete the branch.

**git branch -d <feature-branch-name-to-delete>**

**4. Merge Conflict:**

To merge, we use:

**git merge main**

However, this might show the merge conflicts. The automatic merge will be failed. To fix merge conflict, we can easily fix directly on VS Code (can also do on GitHub or Git)

On VS Code:

A picture containing background pattern

Description automatically generated

The green lines show the changes from the current branch. The blue lines show the changes from the other branches. We can then manually handle the conflict. \

1. **Git Diff**

**git diff**

By default, this command shows the **uncommitted changes** to your repository

- To show git diff between 2 commits:

**git diff <commit1> <commit2>**

Each commit has an ID, which can be seen by pressing

**git log**

- git diff between 2 branches

**git diff <branch1>..<branch2>**

This will run git diff between **2 latest commits** of the branches. If we want to instead run git diff on specific files:

**$ git diff <branch1>..<branch2> -- <file>**

Diagram

Description automatically generated*NOTE:* Using 3 dots instead of 2 will compare HEAD of right branch with their common ancestor

1. **Viewing Old Commits**

- To view a file as it was committed at a particular commit, use **show**

**git show d6bac6:./path/to/file.c**

To see the difference between your working directory and the files in a certain commit, use diff

**git diff d6bac6**