



Workshop 101 - Git, Github & Version Control



What is Version Control

Version control is a system that records changes to files over time so that you can recall specific versions later.



Benefits of Version Control

- Tracking Changes
- Collaboration
- Backup & Recovery
- Branching and Merging



Now..... if we know version control...

What is Git?

- **Git is a tool for version control.**

<https://github.com/git/git>

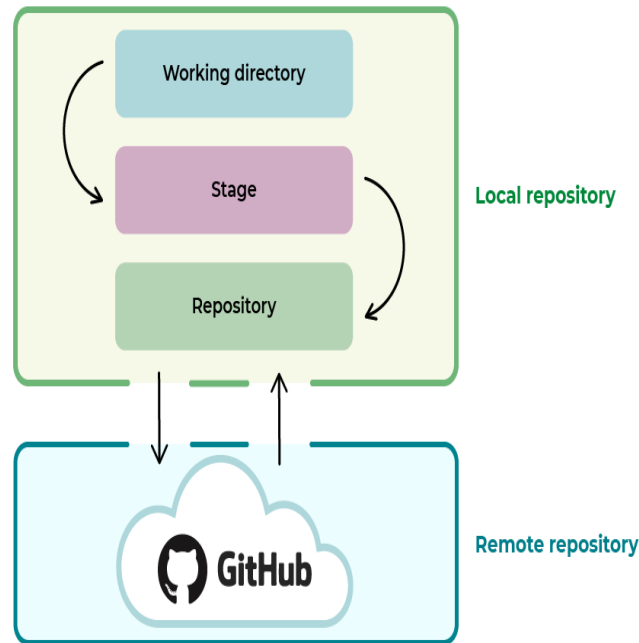


Git and Github

- GitHub is a web-based platform that uses Git for version control and allows developers to collaborate on projects.



AWS CodeCommit



Git flow

Modified/ working dictionary

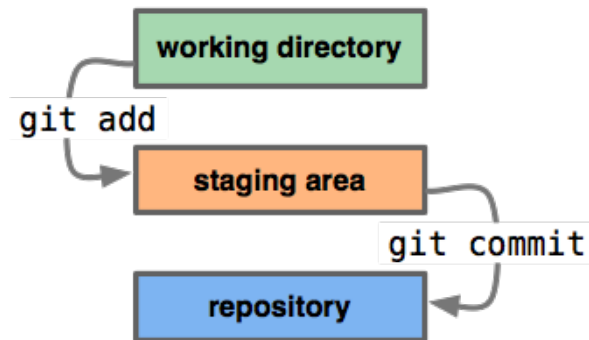
File has been changed. But has not been saved.

Staged

Modified file is ready to go for the next commit(save). File is on standby

Committed

File has been saved onto a database.



Key words to know

 **Repository-** Another word for database.

Branch- Separate/new versions of the main repository. Allows you to work on different parts of the project, without affecting the main repository/project. When work is done, can be merged onto main project.

Analogy- The project/main repository is the tree. The branches are still part of the tree.

Local Repositories- Databases hosted on a individual/personal computer

Remote Repositories- Databases hosted on a server that is accessible to everyone.(internet or some type of network)

Common Git Terms/Commands to Know



- **Git Init** - used to start a project


```
Akash Jha@LAPTOP-LJJ1U61G MINGW64 ~/Desktop/Git (master)
$ git init
Initialized empty Git repository in C:/Users/Akash Jha/Desktop/Git/.git/

Akash Jha@LAPTOP-LJJ1U61G MINGW64 ~/Desktop/Git (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Git Clone- Copy of a code source from a remote repository.(such as github)



```
# Syntax
$ git clone <repository>

# Example
$ git clone https://github.com/username/myproject.git
```




Git Add

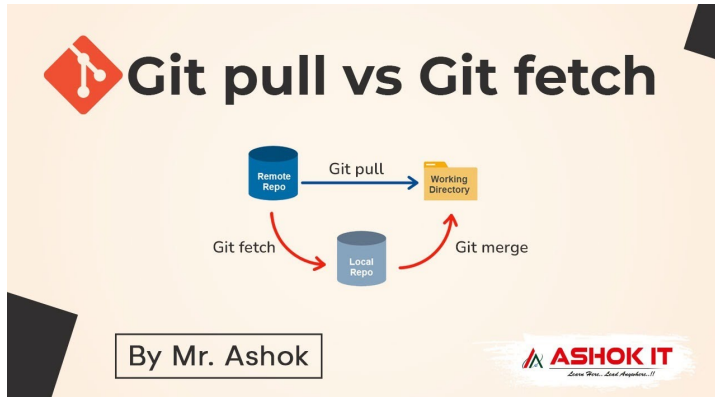
- Adds new or changed files with local repository
- These new files/file changes would basically be set in the stage area, which prepare said file to be used in the next commit.

Git Commit

- **One of the most important git commands.**
- Allows you to save changes. (acts as a checkpoint for a project/task)
- Also, when making a commit, you **MUST** write a message stating what you did.
- Also, commits list the times and date said change was made.

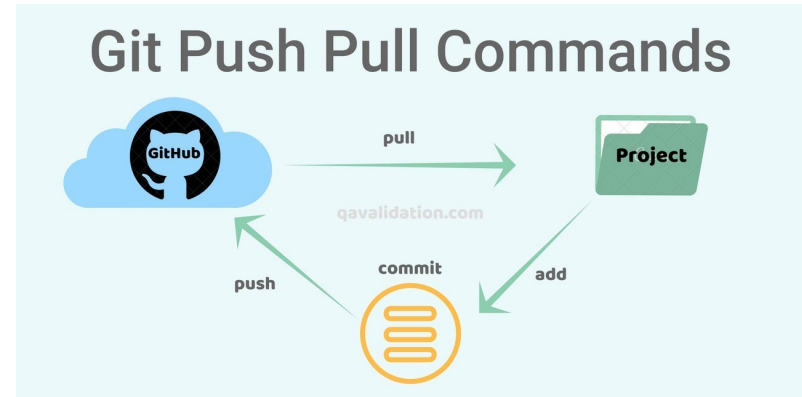
Git Pull

- Used to acquire/update information from the remote repository to the local repository.
- (acts as a updater)



Git Push

- Used to send modified information from local repository to remote repository
- (acts as an uploader)



Git Remote:

- Establishes a connection between the local repository to the remote repository server.

Git Status

- States all the necessary information about a current

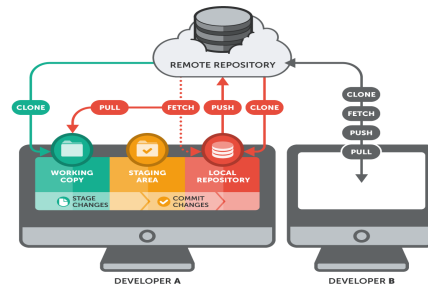
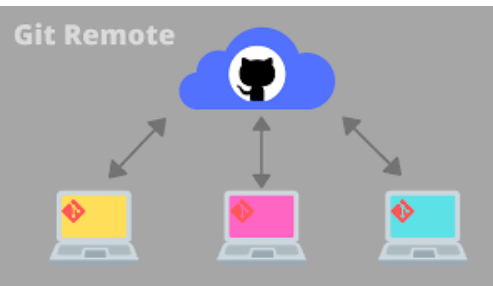
```
@Harish MINGW64 /e/ToolsQA/First Project (master)
$ touch ABC.txt

@Harish MINGW64 /e/ToolsQA/First Project (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        ABC.txt

nothing added to commit but untracked files present (use "git add" to track)

@Harish MINGW64 /e/ToolsQA/First Project (master)
$ |
```




Git Cheat-Sheet

Useful Commands


Basic Git Commands

- **Creating a Repository**
 - `git init`
 - `git clone <url>`
- **Staging and Committing**
 - `git add .`
 - `git commit -m "Initial commit"`
- **Pushing and Pulling**
 - `git push origin main`
 - `git pull`




GIT CHEAT SHEET

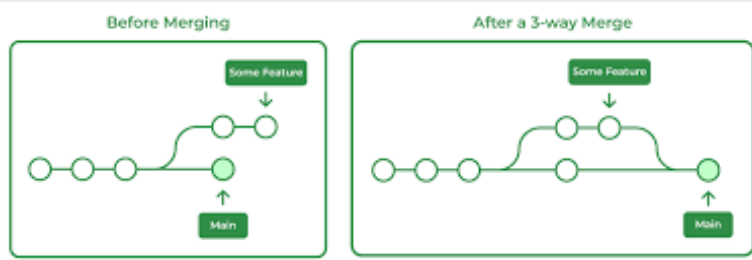
presented by Tower - the best Git client for Mac and Windows



CREATE <ul style="list-style-type: none">Clone an existing repository<ul style="list-style-type: none"><code>\$ git clone ssh://user@domain.com/repo.git</code>Create a new local repository<ul style="list-style-type: none"><code>\$ git init</code>	BRANCHES & TAGS <ul style="list-style-type: none">List all existing branches<ul style="list-style-type: none"><code>\$ git branch -av</code>Switch HEAD branch<ul style="list-style-type: none"><code>\$ git switch <branch></code>Create a new branch based on your current HEAD<ul style="list-style-type: none"><code>\$ git branch <new-branch></code>Create a new tracking branch based on a remote branch<ul style="list-style-type: none"><code>\$ git checkout --track <remote/branch></code>Delete a local branch<ul style="list-style-type: none"><code>\$ git branch -d <branch></code>Mark the current commit with a tag<ul style="list-style-type: none"><code>\$ git tag <tag-name></code>	MERGE & REBASE <ul style="list-style-type: none">Merge <branch> into your current HEAD<ul style="list-style-type: none"><code>\$ git merge <branch></code>Rebase your current HEAD onto <branch>. Don't rebase published commits!<ul style="list-style-type: none"><code>\$ git rebase <branch></code>Abort a rebase<ul style="list-style-type: none"><code>\$ git rebase --abort</code>Continue a rebase after resolving conflicts<ul style="list-style-type: none"><code>\$ git rebase --continue</code>Use your configured merge tool to solve conflicts<ul style="list-style-type: none"><code>\$ git mergetool</code>Use your editor to manually solve conflicts and (after resolving) mark file as resolved<ul style="list-style-type: none"><code>\$ git add <resolved-file></code><code>\$ git rm <resolved-file></code>
LOCAL CHANGES <ul style="list-style-type: none">Changed files in your working directory<ul style="list-style-type: none"><code>\$ git status</code>Changes to tracked files<ul style="list-style-type: none"><code>\$ git diff</code>Add all current changes to the next commit<ul style="list-style-type: none"><code>\$ git add .</code>Add some changes in <file> to the next commit<ul style="list-style-type: none"><code>\$ git add -p <file></code>Commit all local changes in tracked files<ul style="list-style-type: none"><code>\$ git commit -a</code>Commit previously staged changes<ul style="list-style-type: none"><code>\$ git commit</code>Change the last commit. Don't amend published commits!<ul style="list-style-type: none"><code>\$ git commit --amend</code>	UPDATE & PUBLISH <ul style="list-style-type: none">List all currently configured remotes<ul style="list-style-type: none"><code>\$ git remote -v</code>Show information about a remote<ul style="list-style-type: none"><code>\$ git remote show <remote></code>Add new remote repository, named <remote><ul style="list-style-type: none"><code>\$ git remote add <shortname> <url></code>Download all changes from <remote>, but don't integrate into HEAD<ul style="list-style-type: none"><code>\$ git fetch <remote></code>Download changes and directly merge/integrate into HEAD<ul style="list-style-type: none"><code>\$ git pull <remote> <branch></code>Publish local changes on a remote<ul style="list-style-type: none"><code>\$ git push <remote> <branch></code>Delete a branch on the remote<ul style="list-style-type: none"><code>\$ git push <remote> --delete <branch></code>Publish your tags<ul style="list-style-type: none"><code>\$ git push --tags</code>	UNDO <ul style="list-style-type: none">Discard all local changes in your working directory<ul style="list-style-type: none"><code>\$ git reset --hard HEAD</code>Discard local changes in a specific file<ul style="list-style-type: none"><code>\$ git checkout HEAD --file</code>Revert a commit (by producing a new commit with contrary changes)<ul style="list-style-type: none"><code>\$ git revert <commit></code>Reset your HEAD pointer to a previous commit...and discard all changes since then<ul style="list-style-type: none"><code>\$ git reset --hard <commit></code>...and preserve all changes as unstaged changes<ul style="list-style-type: none"><code>\$ git reset <commit></code>...and preserve uncommitted local changes<ul style="list-style-type: none"><code>\$ git reset --keep <commit></code>
COMMIT HISTORY <ul style="list-style-type: none">Show all commits, starting with newest<ul style="list-style-type: none"><code>\$ git log</code>Show changes over time for a specific file<ul style="list-style-type: none"><code>\$ git log -p <file></code>Who changed what and when in <file><ul style="list-style-type: none"><code>\$ git blame <file></code>		

30-day free trial available at www.git-tower.com

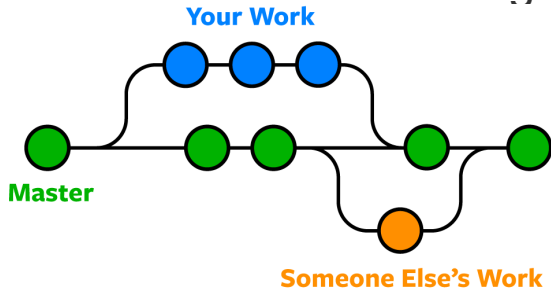




Git Merge:

Integrating a feature branch into a main/master branch

Analogy: merging onto another lane on a highway.



Git log:

Showing history of commits

Git Pull request:

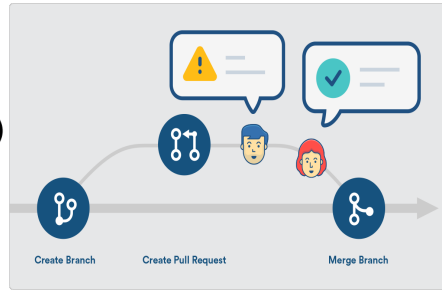
Used to make proposals to merge/add changes from feature branch to main branch.

Mainly used for code review

Examples:

-Feature Additions: New features is made. Pull request is created to merge feature onto main branch after team approval.

Bug fixes: Bug is founded, pull request is created for team to review said bug.





Merge Conflicts

What causes merge conflicts?

- **Merge conflicts occur in Git when multiple contributors make changes to the same part of a file**

How to Resolve Merge Conflicts?

- When a conflict occurs, Git stops the merge process and gives you the opportunity to resolve the conflict manually.



Steps: To resolve

1. **Identify the Conflict**
 - **"This branch has conflicts that must be resolved."**
1. Pushing Changes and Creating a Pull Request
 - **git fetch origin feature-branch**
 - **git checkout feature-branch**
1. **Merge the Target Branch into Your PR Branch**
2. **Stage the Resolved Files**
3. **Commit the Changes**
4. **Push the Changes**

Why is github important?



- Github allows for enhanced collaboration.
- Not only can you view your own work/projects on github, but you can also view the work/projects of other developers.
- Additionally, you can also contribute open source projects
 - Also applicable to team development
 - Used within public repositories.
- Within interviews, you can showcase your projects.

Two git terminal commands to know

Git -help

-List of all the git commands(use if you forget a certain command)

```
These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  restore    Restore working tree files
  rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
  bisect     Use binary search to find the commit that introduced a bug
  diff       Show changes between commits, commit and working tree, etc
  grep       Print lines matching a pattern
  log        Show commit logs
  show       Show various types of objects
  status     Show the working tree status

grow, mark and tweak your common history
  branch     List, create, or delete branches
  commit     Record changes to the repository
  merge      Join two or more development histories together
  rebase     Reapply commits on top of another base tip
  reset      Reset current HEAD to the specified state
  switch     Switch branches
  tag        Create, list, delete or verify a tag object signed with GPG
```

Git --version

To see if Git is already installed on your computer(more applicable to those who have macs).

Use this to check if git is installed on your device, after you download it.

```
PS C:\Users\phili> git --version
git version 2.41.0.windows.3
```

Installing Git on Windows OS,



For Windows OS Users:

Go to search- Type Command Prompt or Terminal

Once there, type the following command:

```
git --version
```

Go to this link:

<https://gitforwindows.org>

Follow the installers instructions

Once installation is complete, go back to command prompt/terminal and enter the command: `git --version` (verify if the installation actually occurred)



Signing up for github

<https://github.com>

Installing Git for MacOS: Homebrew:

Go to search- Type Terminal

Once there, type the following command:

`git --version`

Go to this link:

<https://brew.sh>

The process will take some time, Script explains what it will do. Than does it.

Than one homebrew is installed, you can type either one of the following:

`brew --version` or which brew

Once it is verified that it is installed, type the following command on terminal `bre`

~~When installing~~ Once installation is complete, go back to terminal and enter the command:
`git --version` (verify if the installation actually occurred)

Xcode

Open terminal

Enter in the following command:

```
xcode-select --install
```

Follow the prompts

To check if it is installed:

`xcode-select -p`

`git --version`



Mac OS

Linux Installation

Run with package management tool that comes with distribution

If you're on fedora: enter the following command:

```
sudo dnf install git-all
```

Ubuntu/Debian:

```
sudo apt install git-all
```



EXERCISES/MORE DEMONSTRATIONS!!!



Git Cheat-Sheet

Useful Commands

Creating Branches

Why use branches? Feature development, bug fixes.

- `git branch feature-branch`
- `git checkout -b feature-branch`

Merging Branches

- `git merge feature-branch`

Managing merge conflicts.



GIT CHEAT SHEET

presented by Tower - the best Git client for Mac and Windows



CREATE

Clone an existing repository
`$ git clone ssh://user@domain.com/repo.git`
Create a new local repository
`$ git init`

LOCAL CHANGES

Changed files in your working directory
`$ git status`
Changes to tracked files
`$ git diff`
Add all current changes to the next commit
`$ git add .`
Add some changes in <file> to the next commit
`$ git add -p <file>`
Commit all local changes in tracked files
`$ git commit -a`
Commit previously staged changes
`$ git commit`
Change the last commit
Don't amend published commits!
`$ git commit --amend`

COMMIT HISTORY

Show all commits, starting with newest
`$ git log`
Show changes over time for a specific file
`$ git log -p <file>`
Who changed what and when in <file>
`$ git blame <file>`

BRANCHES & TAGS

List all existing branches
`$ git branch -av`
Switch HEAD branch
`$ git switch <branch>`
Create a new branch based on your current HEAD
`$ git branch <new-branch>`
Create a new tracking branch based on a remote branch
`$ git checkout --track <remote/branch>`
Delete a local branch
`$ git branch -d <branch>`
Mark the current commit with a tag
`$ git tag <tag-name>`

UPDATE & PUBLISH

List all currently configured remotes
`$ git remote -v`
Show information about a remote
`$ git remote show <remote>`
Add new remote repository, named <remote>
`$ git remote add <shortname> <url>`
Download all changes from <remote>, but don't integrate into HEAD
`$ git fetch <remote>`
Download changes and directly merge/integrate into HEAD
`$ git pull <remote> <branch>`
Publish local changes on a remote
`$ git push <remote> <branch>`
Delete a branch on the remote
`$ git push <remote> --delete <branch>`
Publish your tags
`$ git push --tags`

MERGE & REBASE

Merge <branch> into your current HEAD
`$ git merge <branch>`
Rebase your current HEAD onto <branch>
Don't rebase published commits!
`$ git rebase <branch>`
Abort a rebase
`$ git rebase --abort`
Continue a rebase after resolving conflicts
`$ git rebase --continue`
Use your configured merge tool to solve conflicts
`$ git mergetool`
Use your editor to manually solve conflicts and (after resolving) mark file as resolved
`$ git add <resolved-file>`
`$ git rm <resolved-file>`

UNDO

Discard all local changes in your working directory
`$ git reset --hard HEAD`
Discard local changes in a specific file
`$ git checkout HEAD --file`
Revert a commit (by producing a new commit with contrary changes)
`$ git revert <commit>`
Reset your HEAD pointer to a previous commit...and discard all changes since then
`$ git reset --hard <commit>`
...and preserve all changes as unstaged changes
`$ git reset <commit>`
...and preserve uncommitted local changes
`$ git reset --keep <commit>`



CALCULATOR!!!!

Math template

add function(1st)
subtraction function(2nd)
multiplication function(3rd)
division function(4th)

actual calculator function

make it with a while/do while loop, until user says
exit^



Best Practices for Collaboration

Commit Messages

- Descriptive messages for commits. Example: “Add subtraction function.”

Pull Request Reviews:

- Remind participants to provide constructive feedback and to ask questions during the review process.

Branching Strategies:

- Discuss the use of feature branches for new features and hotfix branches for urgent fixes.

1. Setting Up the Repository

Create a New Repository:

- Create a new repository on GitHub (e.g., `simple-calculator`).

2. Clone the Repository

Clone the repository to their local machines and choose the directory where you want to create:

```
git clone https://github.com/username/simple-calculator.git  
cd simple-calculator
```

```
git config --global user.name "Your Name"  
git config --global user.email "you@example.co
```

3. Create a Branch for New Features

Each participant will create a feature branch for their changes:

```
git checkout -b feature/addition # Example
```

4. Implement the Feature

simple addition function in their own branch. For example, create a file called `calculator.py` with the following code:

```
def add(x, y):  
    return x + y
```

5. Commit Changes

participants to stage and commit their changes:

```
git add calculator.py or git add .  
git commit -m "Add addition function"
```

6. Push Changes to GitHub

Have participants push their feature branch to GitHub:

```
git push origin feature/addition
```

7. Create pull request

Create a pull request on GitHub:

- Go to the repository on GitHub.
- Click on "Compare & pull request."
- Add a title and description for the pull request, explaining the changes they made.

8. Review pull request

Review each other's pull requests:

- They can add comments, suggest changes, or approve the pull request.
- Discuss the importance of constructive feedback in code reviews.

8. Merge pull request

Once the pull request is approved, the owner can merge it:

- Click on "Merge pull request" on GitHub.
- Delete the branch after merging to keep the repository clean.

Merge Conflict

What is a Merge Conflict?

A **merge conflict** occurs in a version control system (like Git) when:

- Two or more branches contain changes to the **same part of a file**.
- Git is unable to **automatically resolve** which change should be kept.

Why Merge Conflicts Occur

Merge conflicts happen due to:

- **Simultaneous Edits:** Multiple contributors editing the same file at the same time.
- **Divergent Branches:** Significant differences between branches that need to be merged.
- **Complex Changes:** Changes that affect multiple related files can lead to conflicts.

Making an intentional conflicts

1. Setup the Initial Project

- Start with a simple file structure in your Git repository (e.g., `calculator.py`).

```
# calculator.py
```

```
def add(x, y):  
    return x + y
```

```
def subtract(x, y):  
    return x - y
```

Here's how you can demonstrate a merge conflict and resolve it using Visual Studio Code (VS Code). We'll set up a simple scenario involving two branches with conflicting changes, which participants can easily replicate in your workshop.

Example Scenario:

1. Setup the Initial Project

- Start with a simple file structure in your Git repository (e.g., `calculator.py`).

```
# calculator.py
```

```
def add(x, y):  
    return x + y
```

```
def subtract(x, y):  
    return x - y
```

2. Create Two Feature Branches

1. Create the first feature branch (e.g., `feature/branch1`):

```
git checkout -b feature/branch1
```


Example Scenario:

1. Setup the Initial Project

- Start with a simple file structure in your Git repository (e.g., `calculator.py`).

```
# calculator.py
def add(x, y):
    return x + y

def subtract(x, y):
    return x - y
```

2. Create Two Feature Branches

1. **Create the first feature branch** (e.g., `feature/branch1`):

```
git checkout -b feature/branch1
```

1. Modify the `subtract` function to:

```
def subtract(x, y):  
    return x - y + 1 # Intentional conflict
```

1. Commit the change (git add calculator.py or git add .)

```
git commit -m "Change subtract function to add 1"
```

3. Create the second feature branch (e.g., feature/branch2):

```
git checkout -b feature/branch2
```

- Modify the subtract function differently:

```
def subtract(x, y):  
    return x - y - 1 # Intentional conflict
```

- Commit the change: git add calculator.py

```
git commit -m "Change subtract function to subtract 1"
```

4. Merge the Branches and Create a Conflict

Switch back to the main branch:

```
git checkout main
```

Merge the first feature branch:

```
git merge feature/branch1
```

Merge the second feature branch:

```
git merge feature/branch2
```

- This will result in a merge conflict because both branches modified the subtract function.

5. View the Merge Conflict in VS Code

- Open calculator.py in VS Code. You will see something like this:

```
def subtract(x, y):
```

```
    <<<<<<< HEAD
```

```
        return x - y + 1 # From branch feature/branch1
```

```
    =====
```

```
        return x - y - 1 # From branch feature/branch2
```

```
    >>>>>>> feature/branch2
```

- The lines marked with <<<<<<< HEAD are from your current branch (main), while the lines after ===== are from the merging branch (feature/branch2).

5. Resolve the Merge Conflict

- Discuss how to resolve the conflict by choosing which line to keep, or by combining them. For example, you might want to keep both changes:

```
def subtract(x, y):  
    return x - y + 1 if x > y else x - y - 1 # Combining changes or modifying as needed
```

- After resolving the conflict, save the file.

6. Mark the Conflict as Resolved

- In the terminal, stage the resolved file:

```
git add calculator.py
```

- Complete the merge:

```
git commit -m "Resolve merge conflict between feature/branch1 and feature/branch2"
```

7. Push Changes to GitHub

- Finally, push the resolved changes to the main branch on GitHub:

```
git push origin main
```

Thank You!!