**GIT/GITHUB**

Git is a powerful version control system that allows multiple developers to work on a project simultaneously, manage changes, and collaborate effectively. Below are the steps to set up and use Git for your project, allowing both client and server-side developers to add, commit, push, and pull changes, as well as work with branches.

**1. Initialize a Git Repository**

First, initialize a Git repository in the root of your project directory.

cd my-project

git init

**2. Create a Remote Repository**

Create a remote repository on a platform like GitHub, GitLab, or Bitbucket. Once created, add the remote repository URL to your local Git repository.

git remote add origin <REMOTE\_REPOSITORY\_URL> init

**3. Add and Commit Initial Files**

Add and commit your initial project files.

git add .

git commit -m "Initial commit"

**4. Push to the Remote Repository**

Push your initial commit to the remote repository.

git push -u origin master

**5. Set Up Branches**

It's good practice to use branches to manage different parts of the development process. For instance, you can have separate branches for client and server development.

**Create and Switch to a Branch**

For the server side:

git checkout -b server

For the client side:

git checkout -b client

**Push Branches to the Remote Repository**

Push the branches to the remote repository.

git push -u origin server

git push -u origin client

**6. Collaborating on Code**

Both developers can now clone the repository and start working on their respective branches.

**Cloning the Repository**

Each developer should clone the repository to their local machine.

git clone <REMOTE\_REPOSITORY\_URL>

cd my-project

**Switching Between Branches**

Each developer can switch to their respective branches.

git checkout server # For server-side developer

git checkout client # For client-side developer

**Pulling Updates**

Before starting work, each developer should pull the latest changes from the remote repository.

git pull origin server # For server-side developer

git pull origin client # For client-side developer

**Making Changes and Committing**

After making changes, each developer should add and commit their changes.

git add .

git commit -m "Description of the changes"

Copy code

**Pushing Changes**

Push the changes to the remote branch.

git push origin server # For server-side developer

git push origin client # For client-side developer

**7. Merging Branches**

When it's time to integrate changes, you can merge branches.

**Merge the Client Branch into the Master Branch**

First, switch to the master branch and pull the latest changes.

git checkout master

git pull origin master

Then, merge the client branch into the master branch.

git merge client

Push the changes to the remote repository.

git push origin master

**Merge the Server Branch into the Master Branch**

Repeat the same steps for the server branch.

git checkout master

git merge server

git push origin master

**8. Handling Conflicts**

If there are merge conflicts, Git will notify you. Resolve the conflicts manually in the affected files, then add and commit the resolved files.

git add <file\_with\_conflict>

git commit -m "Resolved merge conflict in <file>"

Finally, push the resolved changes.

git push origin master

**Working Together Example**

**User A Makes Changes and Pushes Them**

1. User A makes changes to the client branch.
2. User A commits the changes locally.

git add .

git commit -m "User A's changes"

1. User A pushes the changes to the remote repository.

git push origin client

**User B Pulls Changes Made by User A**

1. User B pulls the latest changes from the remote repository.

git pull origin client

1. If there are any conflicts, Git will notify User B. User B needs to resolve the conflicts, add the resolved files, and commit the changes.

# Resolve conflicts in the files

git add <resolved\_files>

git commit -m "Resolved merge conflicts"

1. Finally, User B can push the resolved changes back to the remote repository.

git push origin client

**Summary**

By following these steps, both client and server-side developers can efficiently collaborate on the same repository, push and pull each other's changes, and work on different branches to manage their work effectively. Regularly pulling changes before starting new work and committing and pushing frequently can help minimize conflicts and keep everyone in sync.