Team collaboration

**Create a team**

To use GitHub with multiple developers on a project at a same time, there are two ways to set up GitHub for team collaboration which is Organizations and Collaborators.

Organizations is suitable for supervising several teams and every team is set with different level of permission and want to add different members to different repositories. To create Organizations for open source code repositories in Github, go to Settings > Organizations, set up the organization and invite members. We can set permission members in Settings > Member privileges. There are 4 different permission levels. None access that members will only be able to clone and pull public repositories. To give a member additional access, you’ll need to add them to teams or make them collaborators on individual repositories. Read access allows members to clone and pull all repositories. Write access allows members to clone, pull, and push all repositories. Admin access enable members to clone, pull, push, and add new collaborators to all repositories.

However, Collaborators is a better option for a repository owner to add one or few collaborators and assign Read + Write access on the particular repository to the collaborators. Collaborators give Read and Write access to the repository. To add collaborators, go to your repository Settings > Collaborators.

**To contribute**If collaborator wish to contribute to a repository, the collaborator can fork the repo and work on it independently. After the modification, the collaborator can send a pull request to the repository owner to merge his code changes. The team can make discussion of the pull request itself on the code quality, features or others. As a collaborator, to initiate a pull request, firstly, click the “Fork” button on the repository that we want to contribute to, a copy of the repository will be created in your own Github. Next, clone or download the repo in your local pc with either SSH URL or HTTPS URL. If you use SSH method, SSH keys need to be generated and used for authentication. After done cloning, you are suggested to create a new git branch for each new features because if the branch is further updated after some discussions, the pull request will also updated automatically. After making modifications or adding new features, add and commit the changes. Checkout to the git master branch again if you want to push the brunch to the remote repository. These are the command from clone to checkout:

|  |  |
| --- | --- |
|  | $ git clone [ssh-url] [directory]  $ cd [directory]  $ git checkout -b [branch-name]  $ git add .  $ git commit -m "information added in readme"  $ git checkout master |

We can first check the branch name as well as the remote repository aliases, and then push the changes using command:

$ git branch

$ git remote -v

$ git push origin [branch-name]

In your forked repository page, change the branch from master to the new branch that you just pushed, click the “Pull Request” button and submit your pull request with some comment. The repository owner will receive your pull request and is able to write comment on it, or if the owner wants, he or she can merge your pull request with the master branch. Every time when you make a push, in both the owner and collaborator repository page, the pull request will be updated automatically with different pull request code. Before an owner merge a pull request, the owner can run some test locally with command:

$ git checkout -b [new branch]

$ git pull [contributor’s repo branch]

$ git checkout master

$ git merge [new branch]

$ git push origin master

After the marge, pull request will be closed. In the case of the present of merge conflict, Github will not allow the merging before the conflict is removed.

**Bugs and Issues**Issues are helpful for collaborator to track the Bugs, Features, and To-do list. Bugs are things that are broken and need to be fix, features are new ideas to be implement and to-do list contain the items to be complete. There are some useful features in Issues including Labels, Milestones and Search. Each issues can be labeled with colored categories for filtering. Milestones are the dated categories of issues to identify the issues that need to be worked on for next release. The progress bar signifies the number of issues that are closed within the associated Milestone. The progress bar will be updated automatically after each issues that associated with the Milestone are closed. Besides that, a collaborator can be assign to be the person in-charge to fix an issue. Team member can add issues, discuss, and solve the issues together. Closed issues indicate all the issues in the past while open issues are the issues that waiting to be solved. We can Mention related issues by hashtag #[issues-number] in our message and that will create a hyperlink that linked to the hashtaged issue. Issues can be closed with the “close issue” button in the dialog box, or commit message with Fixes/Fixed or Close/Closes/Closed #[issues-number]. As an example: $ git commit -m "bug fixed. fixes #2"  
 $ git push origin master

**Analytics**

Github provide tools which is Github Graphs to provide an insight into the collaborators and their activities. There are 6 different graphs which is Contributors, Traffic, Commits, Code frequency, Punch card and Network. Contributors graph shows who were the contributors and the number of lines of code they add or delete. Commit graph shows the number of commits in each week. Code frequency shows the number of lines that were committed throughout the entire life cycle of the project. Punch card shows the number of commits in each hour of the days. Network graph is the most powerful graph tool as it shows every contributors’ commits across all forked repositories and also shows how the contributors related to one another.

**Code Review**

Github provide interface for team members to comments or raise questions on each other code line by line. We can review the inline comments by just toggle on-off the checkbox beside of commit.