Open Source Programming

Digital Assignment – 1

Slot:-B2-TB2

Submitted to:-Mr. Jayakumar S

Name-Pranav Bansal

Reg. No. – 18BIT0244

Portfolio Link:-

https://PRANAVBANSAL2019.github.io/portfolio

Q. Write down the step by step process of GitHub working methodology and different ways to access GitHub.

Ans:- I will be Making a Project named “Project1” and discuss GitHub working methodology.

Step 1. Create a Repository

A repository is usually used to organize a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs. We recommend including a *README*, or a file with information about your project. GitHub makes it easy to add one at the same time you create your new repository. *It also offers other common options such as a license file.*

My Project1 repository can be a place where you store ideas, resources, or even share and discuss things with others.

To create a new repository

I Name repository Project1.

1. Write a short description.
2. Select Initialize this repository with a README.

Click Create repository**.**

Step 2. Create a Branch

Branching is the way to work on different versions of a repository at one time.

By default your repository has one branch named `main` which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to `main`.
When I create a branch off the `main` branch, I am making a copy, or snapshot, of `main` as it was at that point in time. If someone else made changes to the `main` branch while I am working on other branch, I could pull in those updates.

This diagram shows:

- The `main` branch
- A new branch called `feature` (because we're doing 'feature work' on this branch)
- The journey that `feature` takes before it's merged into `main`

To create a new branch

1. I Go to your new repository Project1.
2. I Click the drop down at the top of the file list that says branch: main.
3. I Type a branch name, `readme-edits`, into the new branch text box.
4. I Select the blue Create branch box or hit "Enter" on your keyboard.

Now I have two branches, `main` and `readme-edits`. They look exactly the same, but not for long! Next we'll add our changes to the new branch.

Step 3. Make and commit changes

I on the code view for your `readme-edits` branch, which is a copy of `main`. Let's make some edits.

On GitHub, saved changes are called *commits*. Each commit has an associated *commit message*, which is a description explaining why a particular change was made. Commit messages capture the history of your changes, so other contributors can understand what you've done and why.

Make and commit changes

Click the README.md file.

1. I Click the pencil icon in the upper right corner of the file view to edit.
2. In the editor, write a bit about yourself.
3. I Write a commit message that describes your changes.
4. I Click **Commit changes** button.

These changes will be made to just the README file on your `readme-edits` branch, so now this branch contains content that's different from `main`.

Step 4. Open a Pull Request

Pull Requests are the heart of collaboration on GitHub. When I open a *pull request*, I am proposing my changes and requesting that someone review and pull in my contribution and merge them into their branch. Pull requests show *diffs*, or differences, of the content from both branches. The changes, additions, and subtractions are shown in green and red.

As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.

I can even open pull requests in my own repository and merge them yourself. It's a great way to learn the GitHub flow before working on larger projects.

Open a Pull Request for changes to the README

When I am done with my message, I click Create pull request**!**

Step 5. Merge your Pull Request

In this final step, it's time to bring your changes together – merging your `readme-edits` branch into the `main` branch.

1. I Click the green **Merge pull request** button to merge the changes into `main`.
2. I Click **Confirm merge**.
3. I Go ahead and delete the branch, since its changes have been incorporated, with the **Delete branch** button in the purple box.

Q:-Write down the pros and cons of GitHub.

Ans:-

Pros:-

1. Version Control: GitHub, being built over Git, makes it fast and easy to develop projects in versions/branches and easily rollback to previous versions when necessary.
2. Pull Requests/Review: GitHub has a powerful UI for creating pull requests, with useful tools like inline commenting and more recently "suggested changes". Pull request history is always maintained and easy to search.

3. Collaboration/Auditing: It's easy for multiple team members to work on the same project and merge changes (often) seamlessly. All contributions are tracked so it's easy to identify contributors.
4. Industry Standard: GitHub is used by virtually all major open source projects so it's very easy to find and contribute to projects of interest if you're well versed with GitHub.
5. Reliability is solid and we never have to worry much about Github being available

Cons:-

1. Reviewing large pull requests can be tedious and it can be tough to identify recent changes (e.g. a one line change) in new files or files with lots of changes.
2. It should be a bit harder to push unresolved merge conflicts, we've had these slip through once in a while.
3. You have to be careful with merge operations; a bad merge can be painful to reverse.
4. Github's status as an industry leader means it's often targeted by sophisticated attackers with DDOS attacks, which has kicked it offline a handful of times in the past few years
5. Lacks first-party support for mobile (no app component)
6. Uncertainty in how Microsoft will manage the company post-deal-close

Q:-List down the features needs to be added in GitHub.

Ans:- Features Need to be added in GitHub:-

1. Better handling or notification of deleted forked repos. If you delete the repo, the pull request will show up as "unknown repository" which creates odd dead ends

2.Discovering new repos could be improved

3. When browsing history of a file, GitHub could make it easier to see the file after a particular commit instead of just being able to quickly view the commit. I'd like to be able to see the commit or the file itself in one click.

4. I would like to be able to view commits by user.

5. I would love to be able to traverse code on GitHub (go to definition, etc) - the good news here is that they are working on these features!

Q:-Compare the minimum of three version control applications.

Ans:- Here 3 version control systems discussed by me are Git (distributed), Mercurial (distributed), and Subversion (centralized).

SVN is different from Git and Mercurial, in that it is a single repository that all users have to pull and commit to.

Git and Mercurial have been very similar, but to tweak Mercurial, we can do it, because it's written in Python. Git is at least somewhat in C, which I'm not as familiar with.

Git and Mercurial have a distributed model. This means that there is a repository on every computer and there is usually an "Official" repository that people will choose to commit their changes to and pull from. Subversion, on the other hand, consists of a single central repository, and each working copy is a slave to that central server, pushing and pulling (committing and updating, in this case) changes from it and it alone.

Installing Git or Mercurial for a couple of people consists of getting SSH access to the same server and installing a couple of packages. Whereas for SVN, as far as I know you need to configure and run an actual server application under Apache, and then mess with an SSL cert and .htaccess, etc. to secure it.

Generally for personal projects, we go with Mercurial or Git. But someone is working with a large team, they probably go Subversion because you get centralized authentication and hosting. But for two people, they pick one of the distributed ones, because then you don't have to mess with centralized authentication and hosting.