VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Name: Aashish Bansal

Registration No.: 19BIT0346

Course Code: ITE1008

Course Name: Open Source Programming

Course Faculty: Prof. Jayakumar S.

Course Slot: B1 + TB1

Assignment No.: 1 [Theory]

My GitHub Profile: https://www.github.com/aashish22bansal

My Portfolio Website: https://aashish22bansal.github.io

My GitHub Link for Portfolio Website Repository: https://github.com/aashish22bansal/aashish22bansal.github.io

Assignment Link on Personal GitHub: https://github.com/aashish22bansal/VIT-Assignments/tree/master/ITE1008%20PE%20Open%20Source%20Programming/Digital%20Assignment%201

Assignment Link on Classroom GitHub: https://github.com/JayakumarClassroom/fs20-ite1008-da1-b1-aashish22bansal

Case Study on GitHub Version Control

Question 1

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

Answer

Working Methodology

GitHub is a web-based platform used for version control. Git simplifies the process of working with other people and makes it easy to collaborate on projects. Team mates can work on files and easily merge their changes in with the master branch of the project.

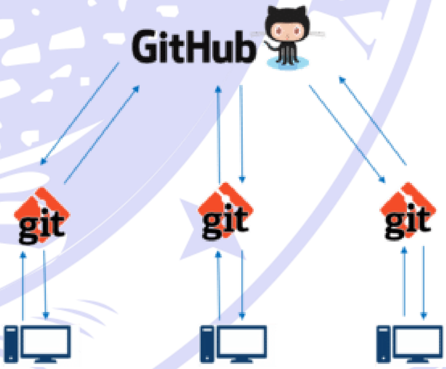
Step 1: What is GitHub?

GitHub is a file or code-sharing service to collaborate with different people. GitHub is a highly used software that is typically used for version control. It is helpful when more than just one person is working on a project. For example, a software developer team wants to build a website and everyone has to update their codes simultaneously while working on the project. In this case, GitHub helps them to build a centralized repository where everyone can upload, edit, and manage the code files.

Why is GitHub so popular?

GitHub has various advantages but many people often have a doubt as to why not use dropbox or any cloud-based system? Say more than two software developers are working on the same file and they want to update it simultaneously. Unfortunately, the person who save the file first will get precedence over the others. While in GitHub, this is not the case. GitHub document the changes and reflect them in an organized manner to avoid any chaos between any of the files uploaded. Therefore, using GitHub centralized repository, it avoids all the confusion and working on the same code becomes very easy.

If you look at the image on the right, GitHub is a central repository and Git is a tool which allows you to create a local repository. Now people usually get confused between git and GitHub but it's actually very different. Git is a version control tool that will allow you to perform all kinds of operations to fetch data from the central server or push data to it whereas GitHub is a core hosting platform for version control collaboration. GitHub is a company that allows you to host a central repository in a remote server.



Now, ways in which GitHub makes git simple:

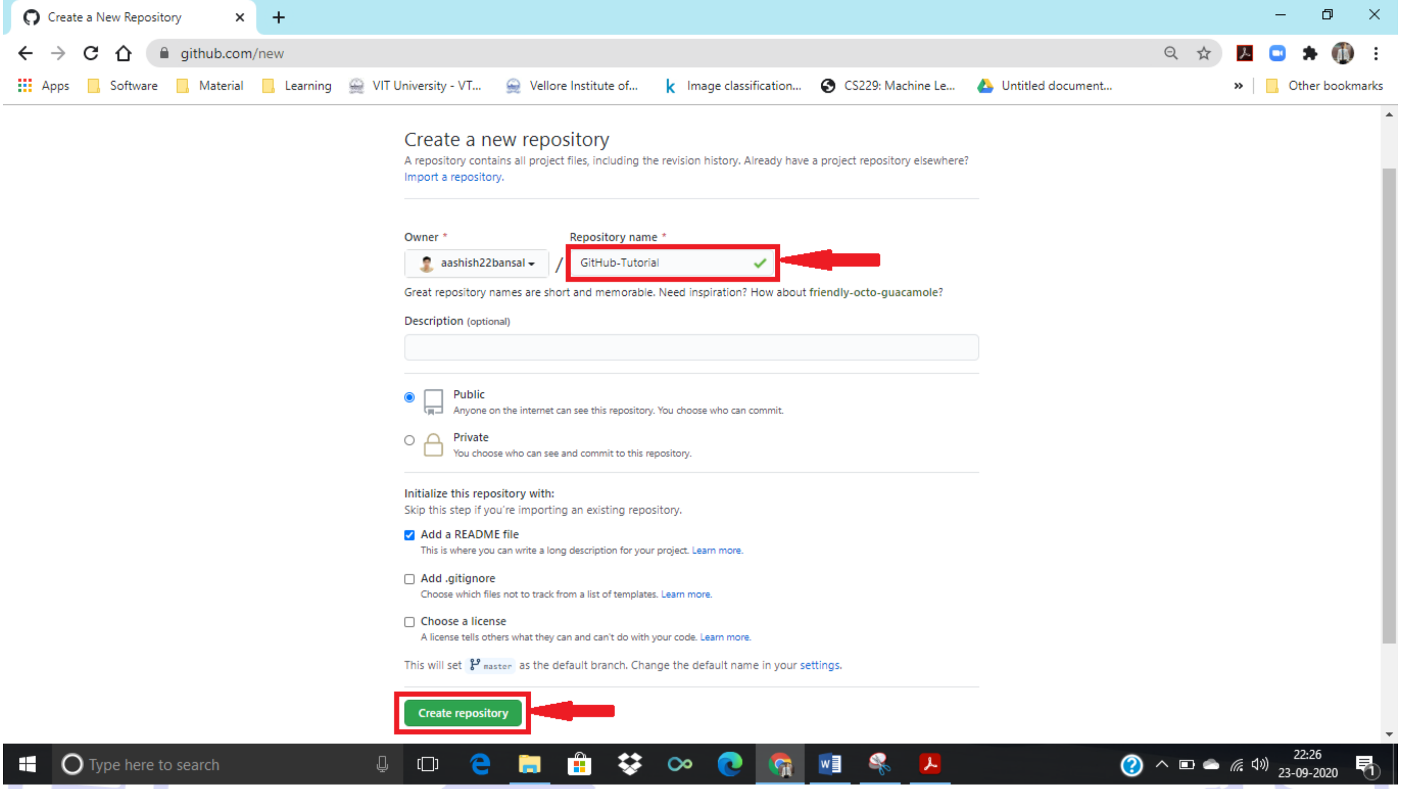
- GitHub provides you a beautiful visual interface which helps you to track or manage your version-controlled projects locally.
- Once you register on GitHub, you can connect with social network and build a strong profile.

Step 2: How to create a GitHub Repository?

A repository is a storage space where your project lives. It can be local to a folder on your computer, or it can be a storage space on GitHub or another online host. You can keep code files, text files, images or any kind of a file in a repository. You need a GitHub repository when you have done some changes and are ready to be uploaded. This GitHub repository acts as your remote repository. So, let me make your task easy, just follow these simple steps to create a GitHub repository:

- Go to the link: https://github.com/ . Fill the sign-up form and click on "Sign up for GitHub".
- Click on 'New' in the 'Repositories' Section.

- Enter any repository name and click on “Create Repository”. You can also give a description to your repository (optional).



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * aashish22bansal / Repository name * GitHub-Tutorial ✓

Great repository names are short and memorable. Need inspiration? How about friendly-octo-guacamole?

Description (optional)

● Public
Anyone on the internet can see this repository. You choose who can commit.

○ Private
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☑ Add a README file
This is where you can write a long description for your project. Learn more.

□ Add .gitignore
Choose which files not to track from a list of templates. Learn more.

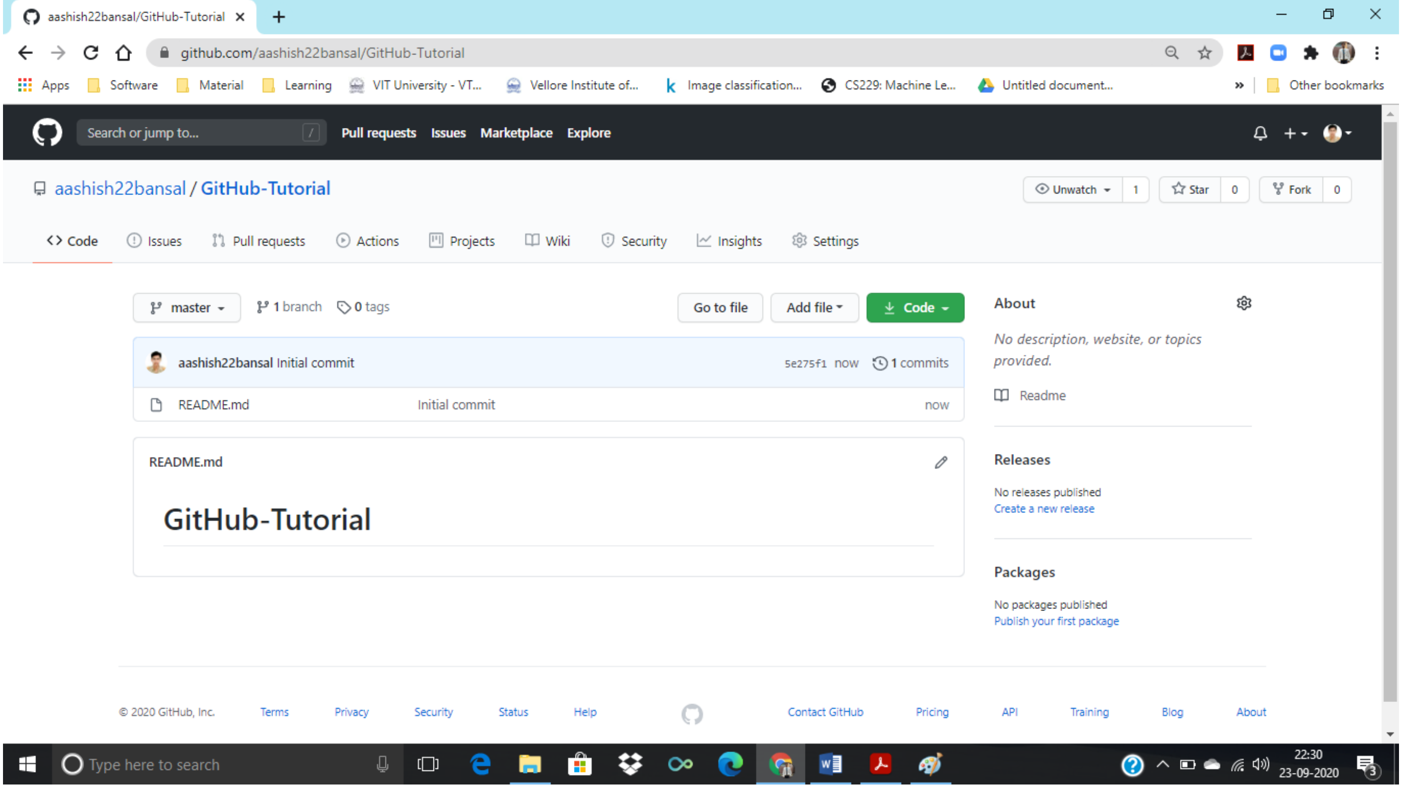
□ Choose a license
A license tells others what they can and can't do with your code. Learn more.

This will set master as the default branch. Change the default name in your settings.

Create repository

Now, if you noticed by default a GitHub repository is public which means that anyone can view the contents of this repository whereas in a private repository, you can choose who can view the content. Also, private repository is a paid version. Also, if you refer the above screenshot, initialize the repository with a README file. This file contains the description of the file and once you check this box, this will be the first file inside your repository.

Your repository is successfully created! It will look like the below screenshot:



aashish22bansal / GitHub-Tutorial

aashish22bansal Initial commit 5e275f1 now 1 commits

README.md Initial commit now

README.md

GitHub-Tutorial

About

No description, website, or topics provided.

Readme

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

So now a central repository has been successfully created! Once this is done, you are ready to commit, pull, push and perform all the other operations.

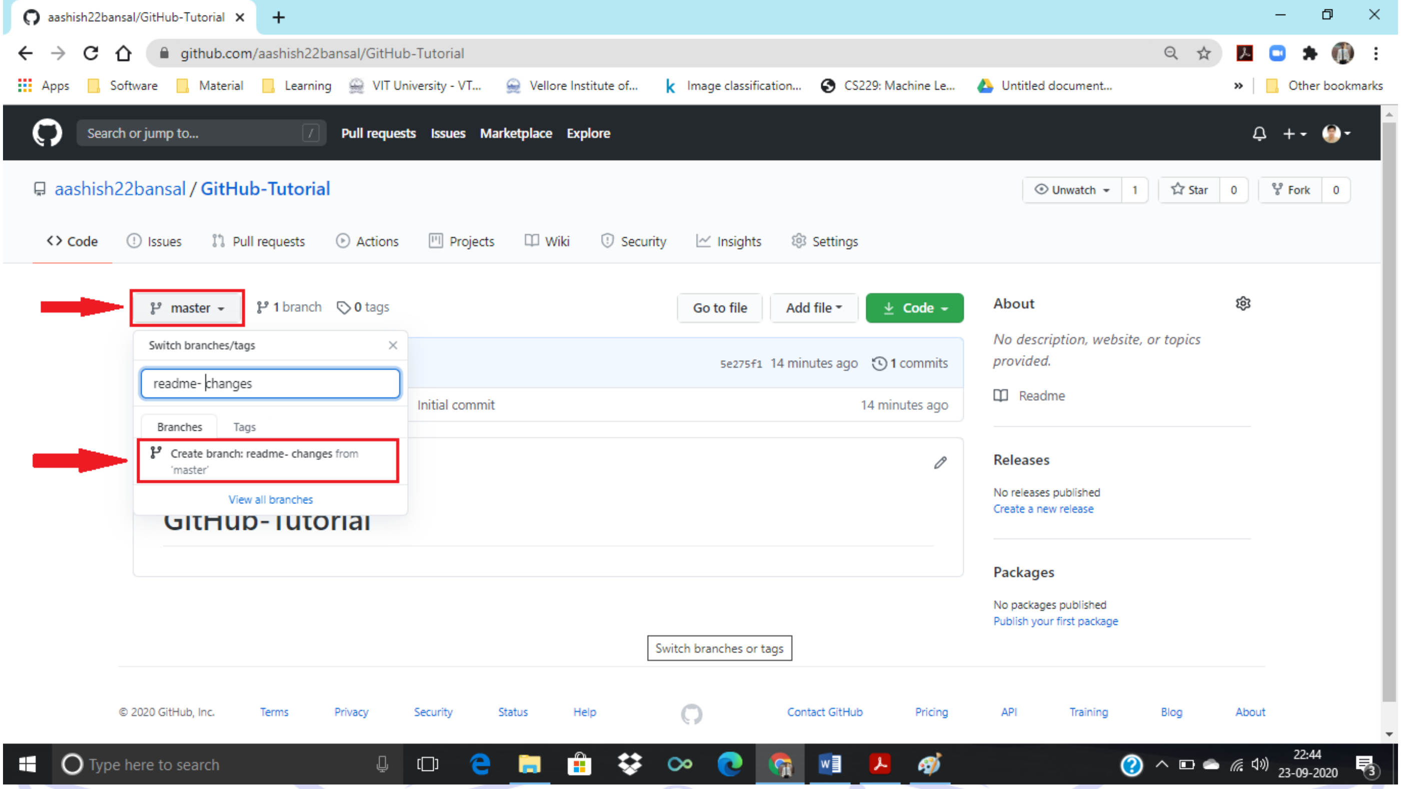
Step 3: Create Branches and Perform Operations

Branching: Branches help you to work on different versions of a repository at one time. Let’s say you want to add a new feature (which is in the development phase), and you are afraid at the same time whether to make changes to your main project or not. This is where git branching comes to rescue. Branches allow you to move back and forth between the different states/versions of a project. In the above scenario, you can create a new branch and test the new feature without affecting the main branch. Once you are done with it, you can merge the changes from new branch to the main branch. Here the main branch is the master branch, which is there in your repository by default.

There is a master branch which has a new branch for testing. Under this branch, two set of changes are done and once it completed, it is merged back to the master branch. This is how branching works.

To create a branch in GitHub, follow the below steps:

- Click on the dropdown “Branch: master”
- As soon as you click on the branch, you can find an existing branch or you can create a new one. In my case, I am creating a new branch with a name “readme- changes”. Refer to the below screenshot for better understanding.



Once you have created a new branch, you have two branches in your repository now i.e. read-me (master branch) and ‘readme- changes’. The new branch is just the copy of master branch. So, let’s perform some changes in our new branch and make it look different from the master branch.

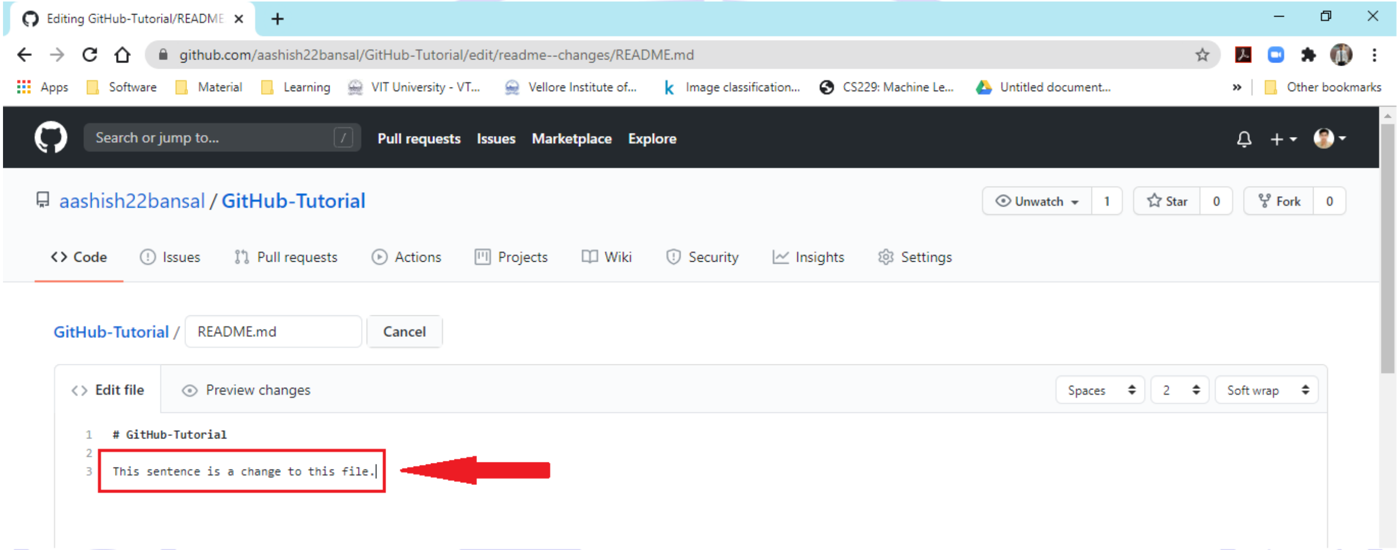
How to use GitHub: Operations

Commit Command:

This operation helps you to save the changes in your file. When you commit a file, you should always provide the message, just to keep in the mind the changes done by you. Though this message is not compulsory but it is always recommended so that it can differentiate the various versions or commits you have done so far to your repository. These commit messages maintain the history of changes which in turn help other contributors to understand the file better. Now let’s make our first commit, follow the below steps:

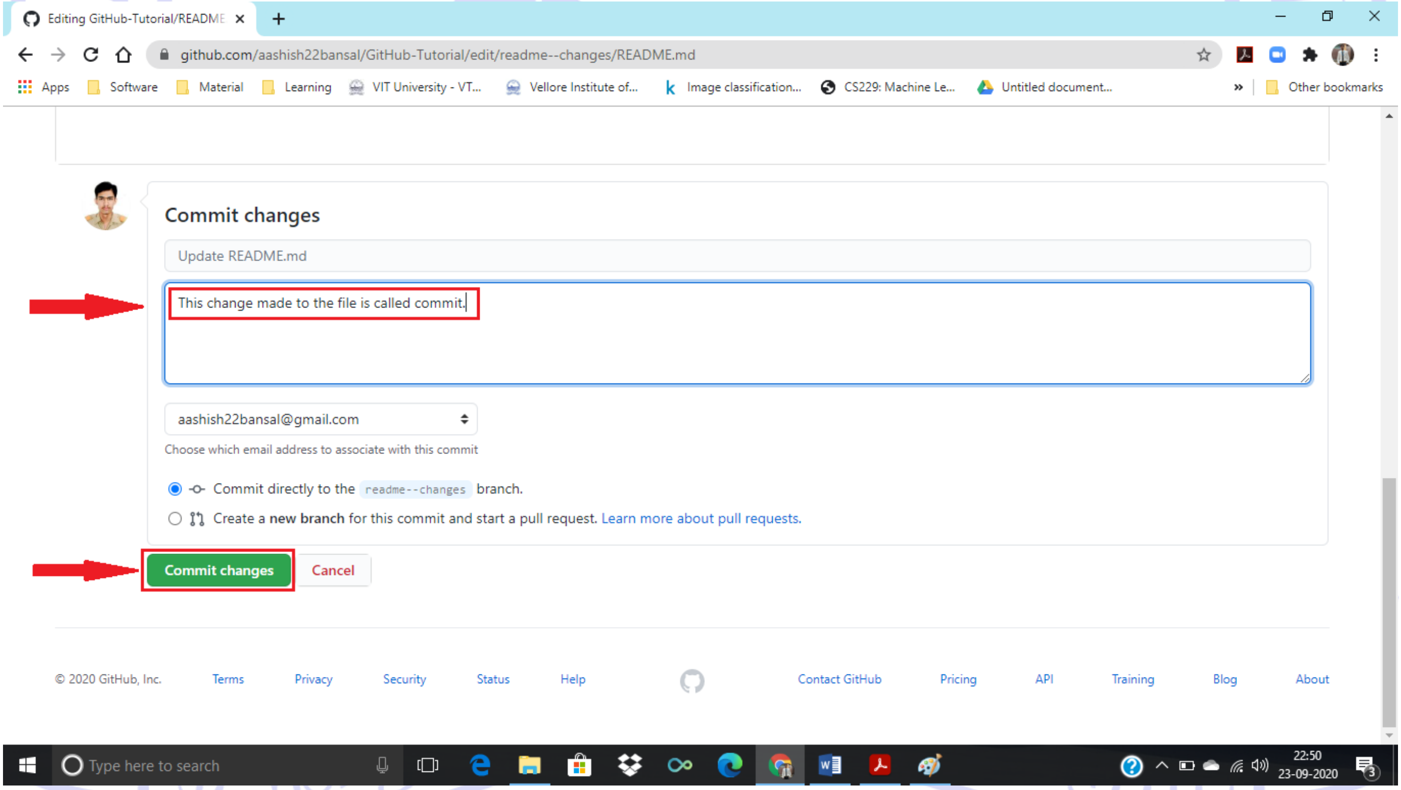
- Click on “readme- changes” file which we have just created.
- Click on the “edit” or a pencil icon in the rightmost corner of the file.
- Once you click on that, an editor will open where you can type in the changes or anything.
- Write a commit message which identifies your changes.
- Click commit changes in the end.

Refer to the below screenshot for better understanding:



```
# GitHub-Tutorial

This sentence is a change to this file.
```



Commit changes

Update README.md

This change made to the file is called commit.

aashish22bansal@gmail.com

Choose which email address to associate with this commit

● Commit directly to the `readme--changes` branch.

○ Create a **new branch** for this commit and start a pull request. Learn more about pull requests.

Commit changes Cancel

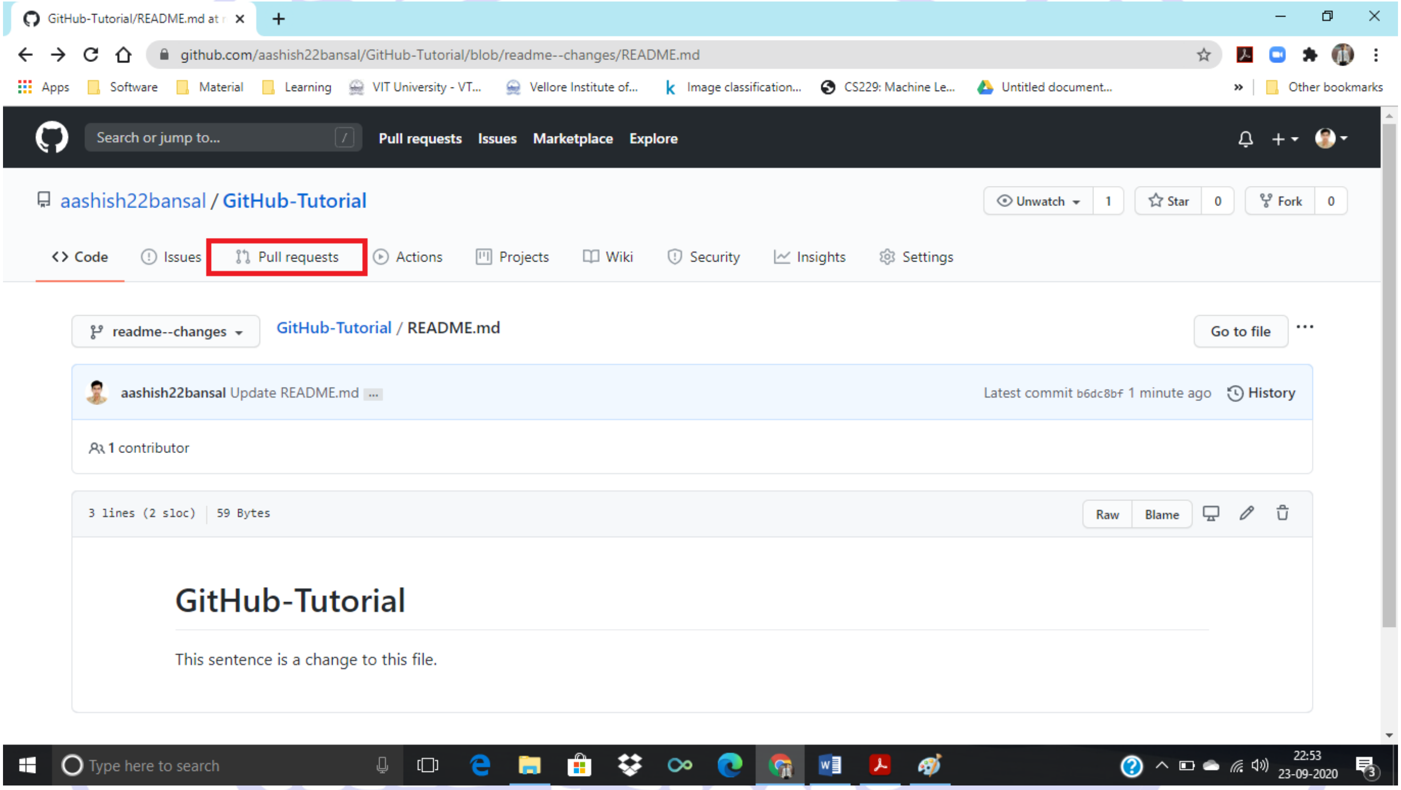
Next

We have successfully made our first commit. Now this “readme- changes” file is different from the master branch. Next, let us see how can we open a pull request.

Pull Command

Pull command is the most important command in GitHub. It tells the changes done in the file and request other contributors to view it as well as merge it with the master branch. Once the commit is done, anyone can pull the file and can start a discussion over it. Once it’s all done, you can merge the file. Pull command compares the changes which are done in the file and if there are any conflicts, you can manually resolve it. Now let us see different steps involved to pull request in GitHub.

- Click the ‘Pull requests’ tab.
- Click ‘New pull request’.
- Once you click on pull request, select the branch and click ‘readme- changes’ file to view changes between the two files present in our repository.
- Click “Create pull request”.
- Enter any title, description to your changes and click on “Create pull request”. Refer to the below screenshots.



aashish22bansal Update README.md

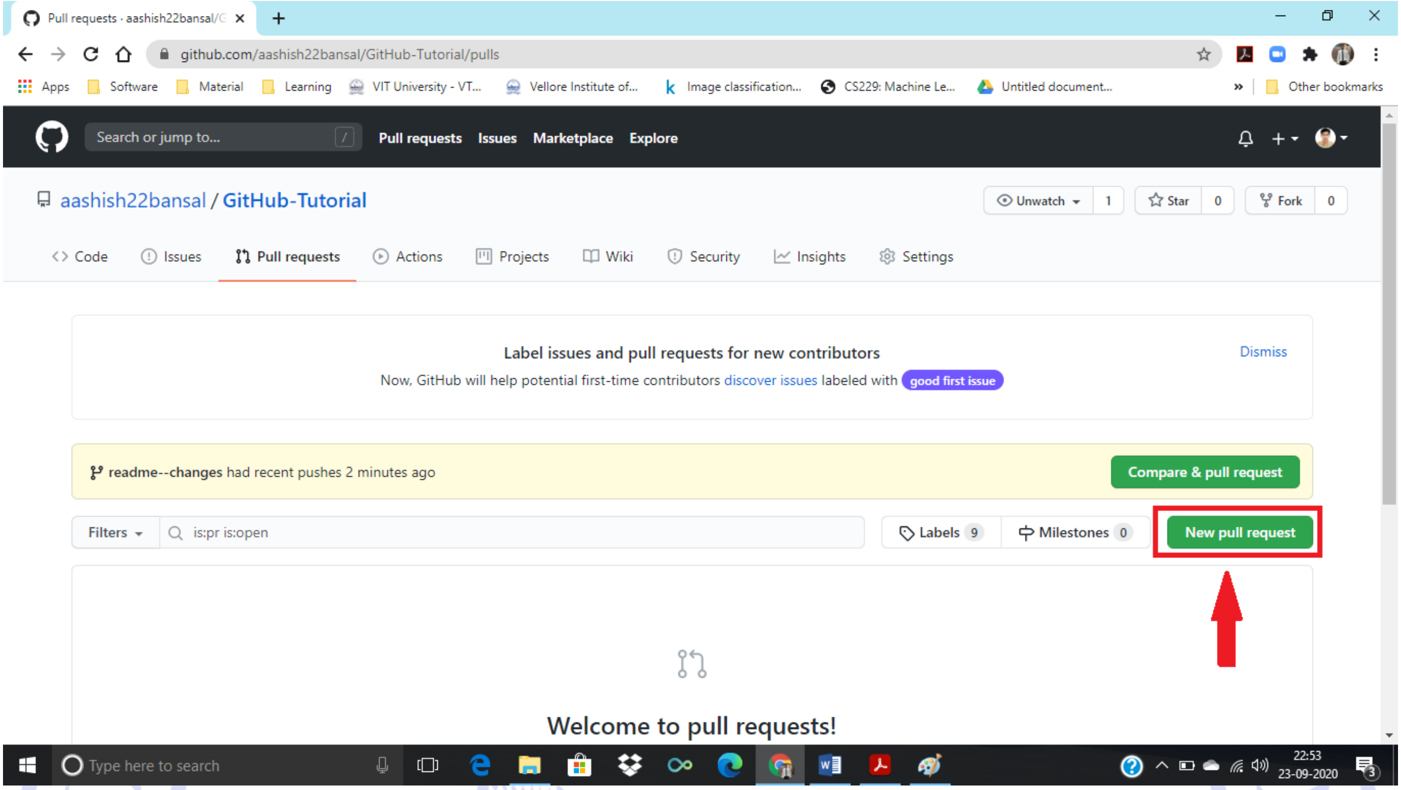
Latest commit b6dc8bf 1 minute ago

1 contributor

3 lines (2 sloc) | 59 Bytes

GitHub-Tutorial

This sentence is a change to this file.

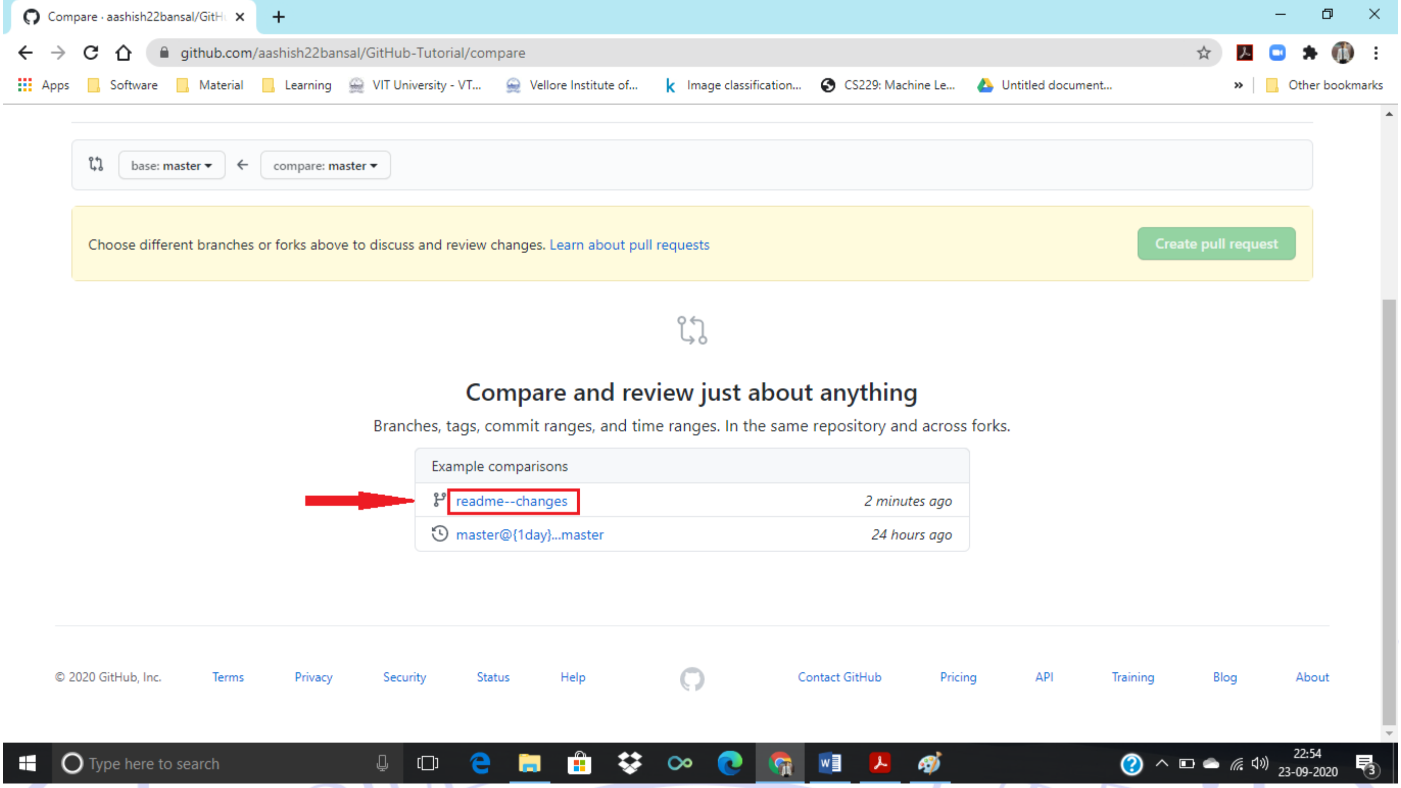


Label issues and pull requests for new contributors

Now, GitHub will help potential first-time contributors discover issues labeled with good first issue

readme--changes had recent pushes 2 minutes ago

Welcome to pull requests!

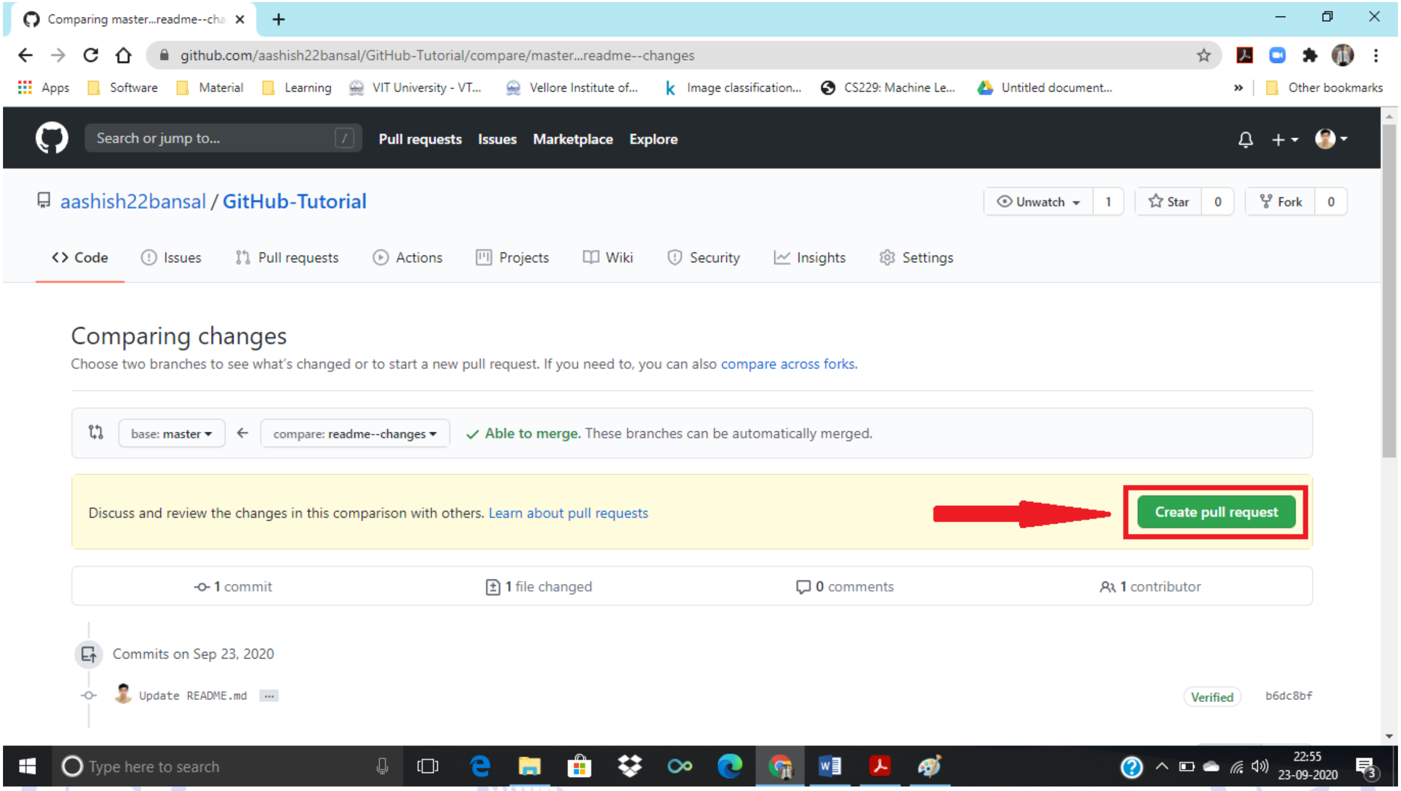


Choose different branches or forks above to discuss and review changes. Learn about pull requests

Compare and review just about anything

Branches, tags, commit ranges, and time ranges. In the same repository and across forks.

Example comparisons	
readme--changes	2 minutes ago
master@{1day}...master	24 hours ago



Comparing changes

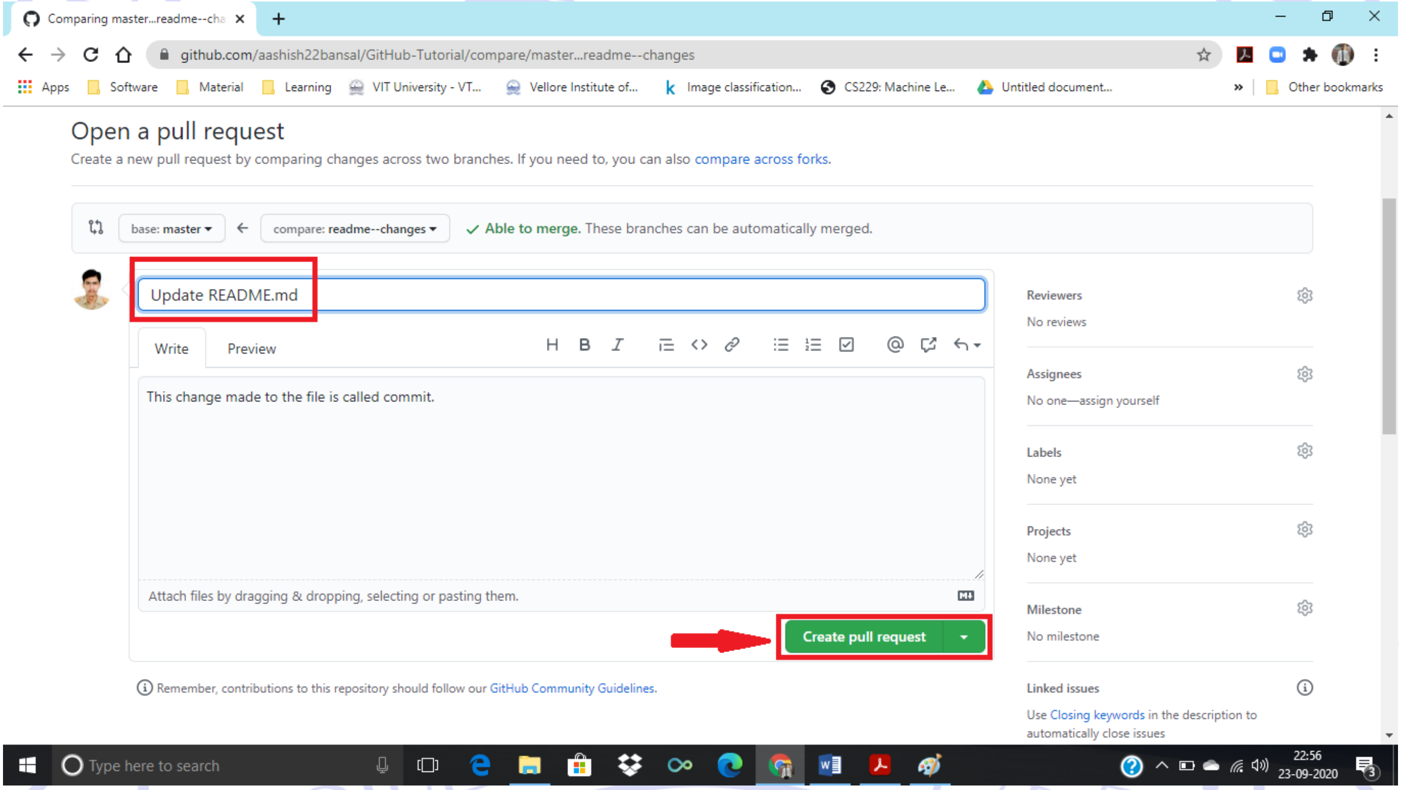
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base: master ← compare: readme--changes ✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others. Learn about pull requests

Commits on Sep 23, 2020

Update README.md



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base: master ← compare: readme--changes ✓ Able to merge. These branches can be automatically merged.

Update README.md

This change made to the file is called commit.

Attach files by dragging & dropping, selecting or pasting them.

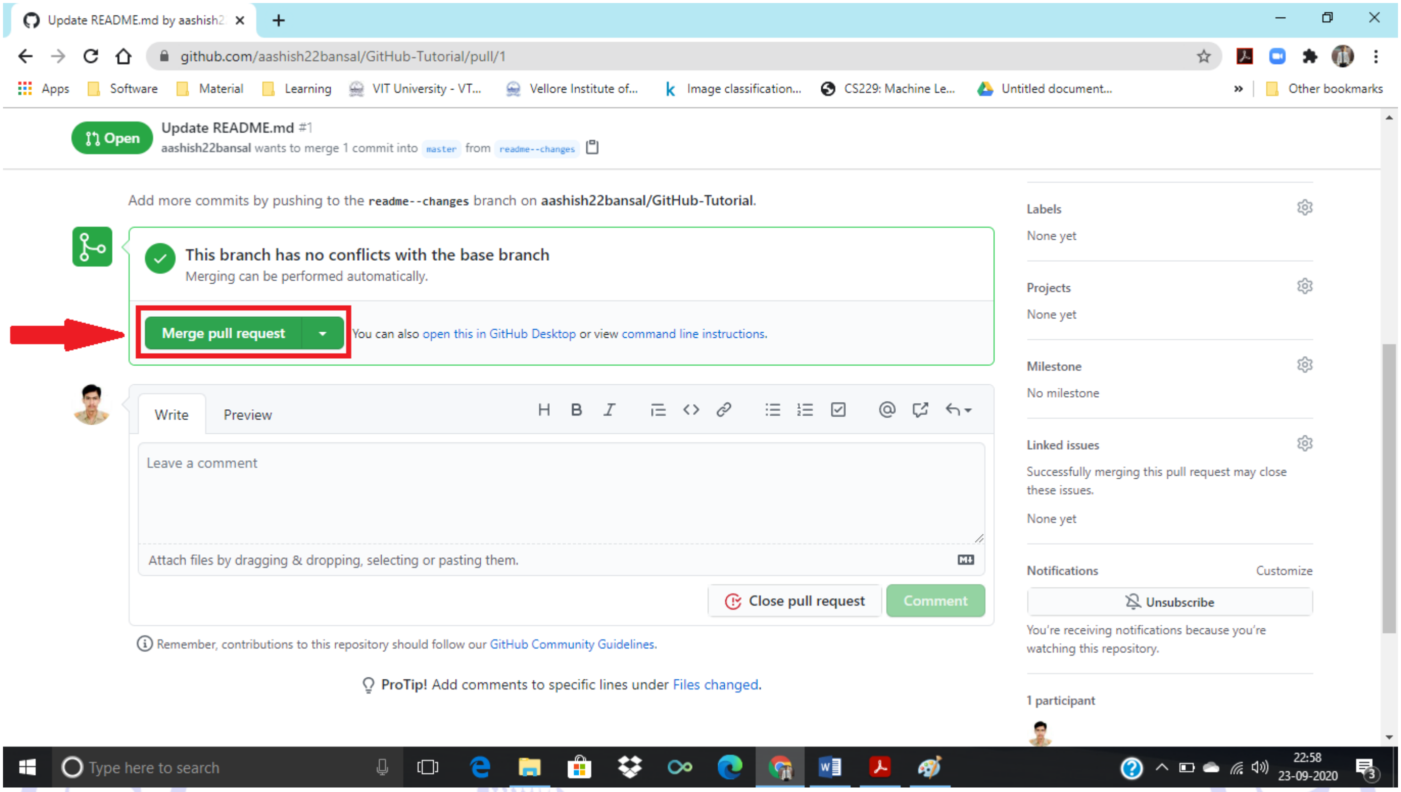
Remember, contributions to this repository should follow our GitHub Community Guidelines.

Use Closing keywords in the description to automatically close issues

Merge Command

Here comes the last command which merge the changes into the main master branch. We saw the changes in pink and green colour, now let’s merge the “readme- changes” file with the master branch/ read-me. Go through the below steps to merge pull request.

- Click on “Merge pull request” to merge the changes into master branch.
- Click “Confirm merge”.
- You can delete the branch once all the changes have been incorporated and if there are no conflicts. Refer to the below screenshots.



Update README.md #1

aashish22bansal wants to merge 1 commit into master from readme--changes

Add more commits by pushing to the readme--changes branch on aashish22bansal/GitHub-Tutorial.

This branch has no conflicts with the base branch

Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Remember, contributions to this repository should follow our GitHub Community Guidelines.

ProTip! Add comments to specific lines under Files changed.

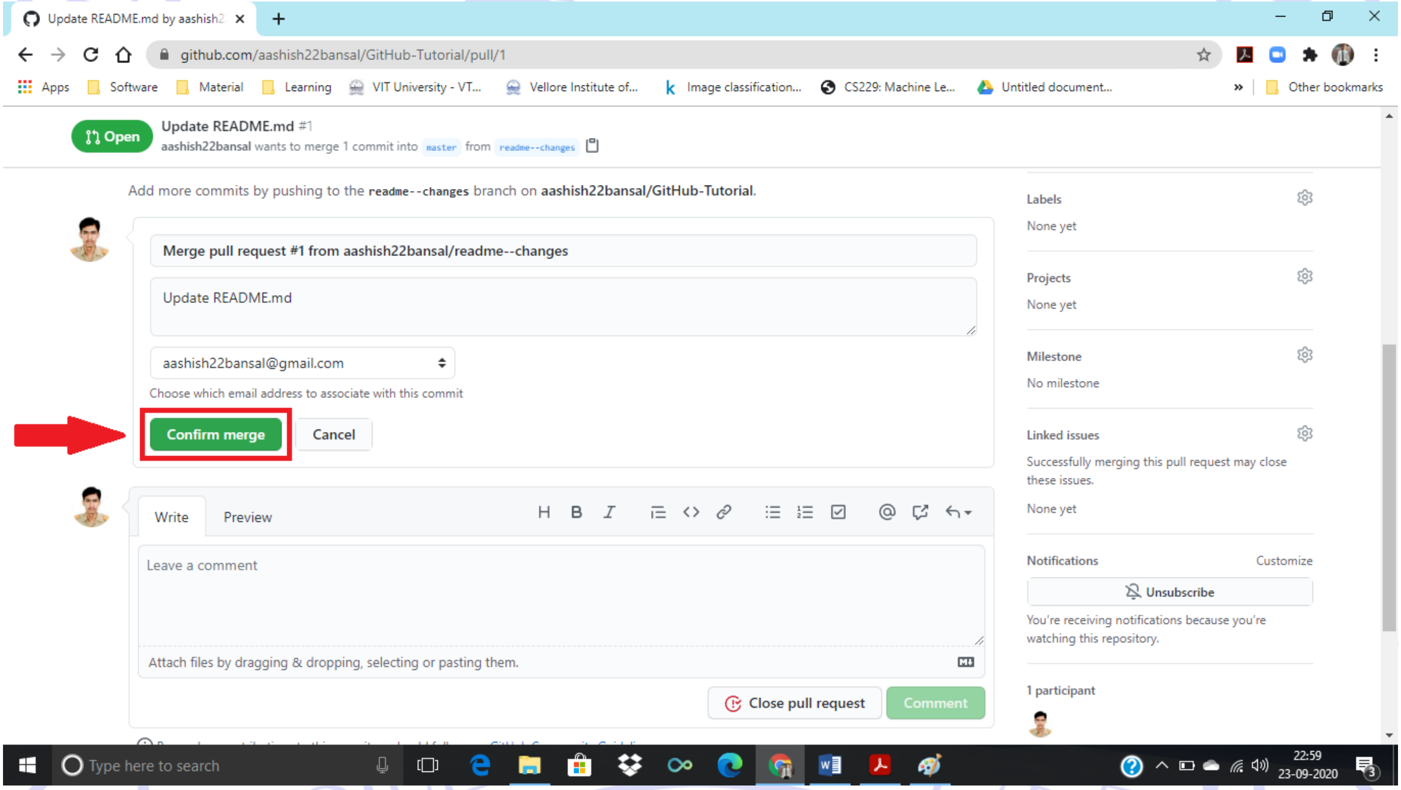
Labels: None yet

Projects: None yet

Milestone: No milestone

Linked issues: Successfully merging this pull request may close these issues. None yet

You're receiving notifications because you're watching this repository.



Update README.md #1

aashish22bansal wants to merge 1 commit into master from readme--changes

Add more commits by pushing to the readme--changes branch on aashish22bansal/GitHub-Tutorial.

Merge pull request #1 from aashish22bansal/readme--changes

Update README.md

aashish22bansal@gmail.com

Choose which email address to associate with this commit

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

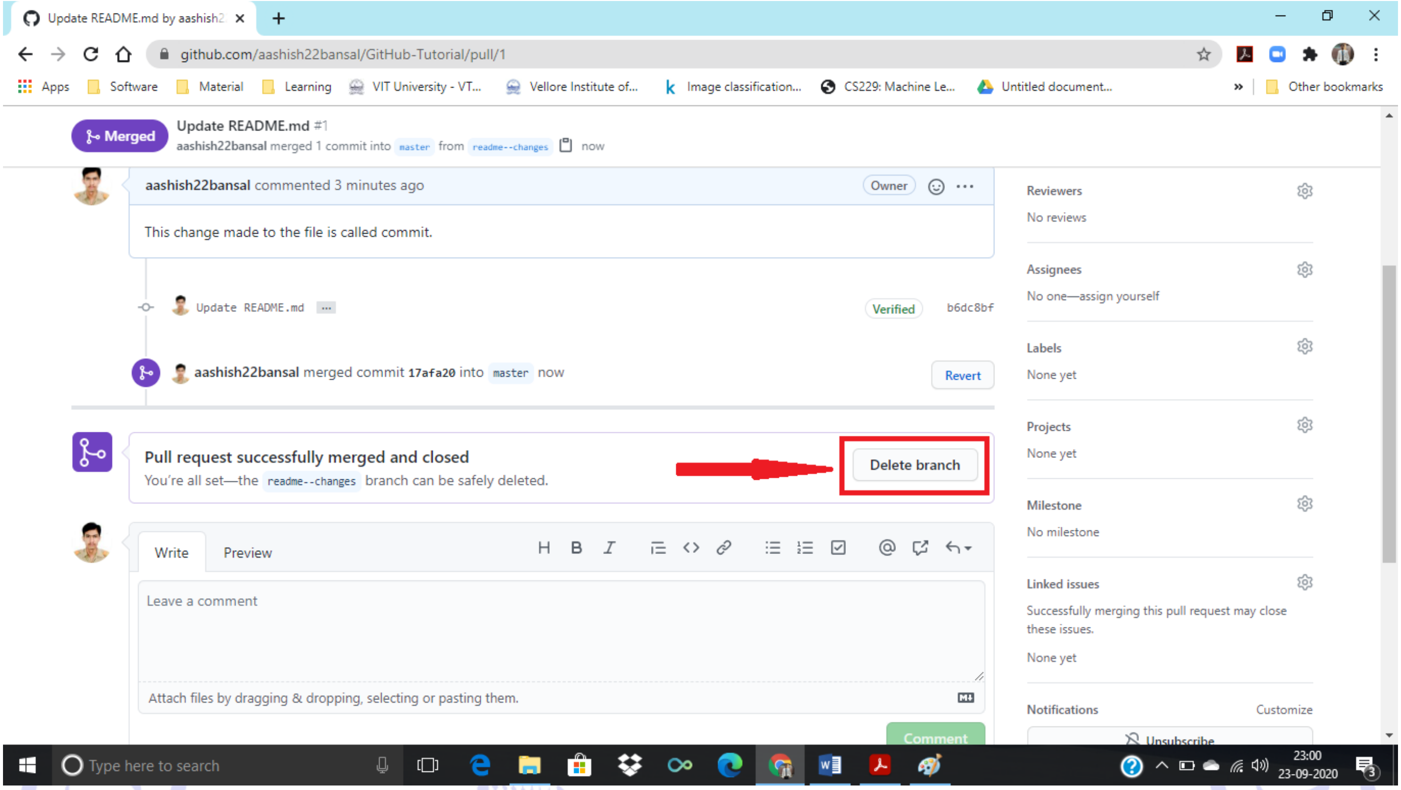
Labels: None yet

Projects: None yet

Milestone: No milestone

Linked issues: Successfully merging this pull request may close these issues. None yet

You're receiving notifications because you're watching this repository.



Update README.md by aashish2

github.com/aashish22bansal/GitHub-Tutorial/pull/1

Merged

Update README.md #1

aashish22bansal merged 1 commit into `master` from `readme--changes` now

aashish22bansal commented 3 minutes ago

This change made to the file is called commit.

Update README.md ... Verified b6dc8bf

aashish22bansal merged commit `17afa20` into `master` now

Pull request successfully merged and closed

You're all set—the `readme--changes` branch can be safely deleted.

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Reviewers: No reviews

Assignees: No one—assign yourself

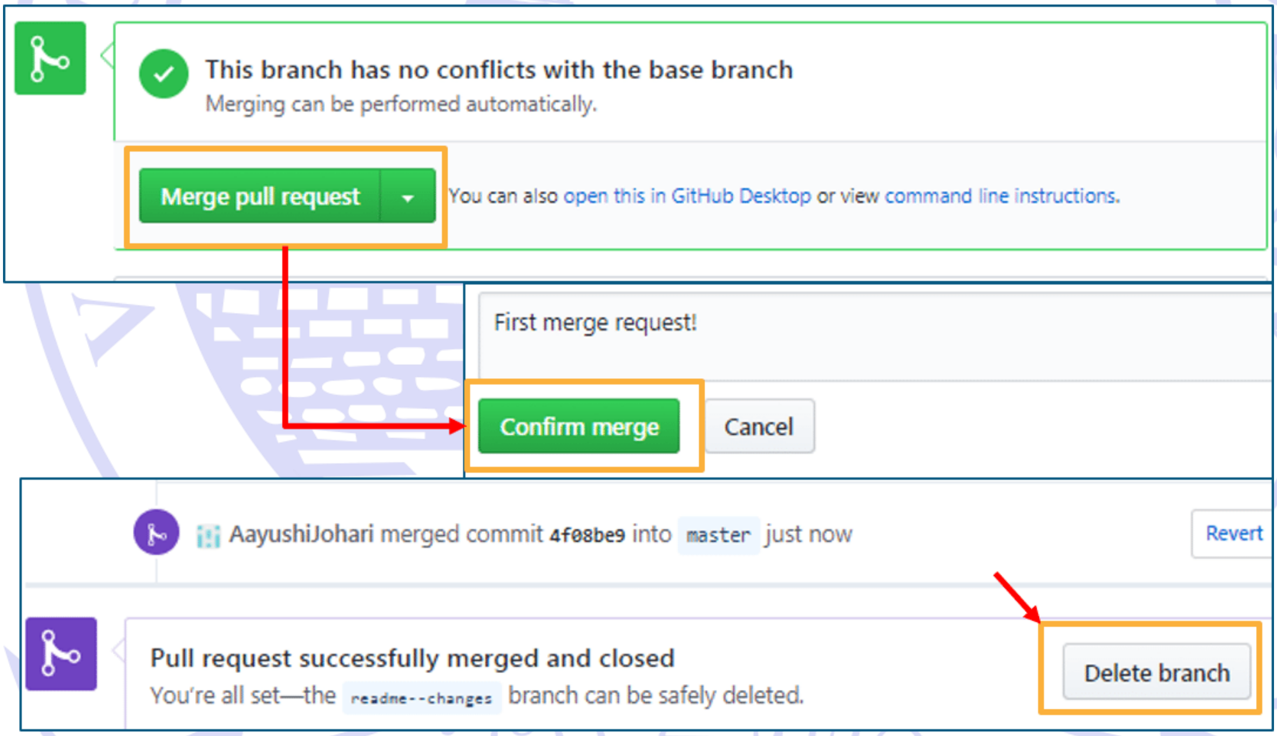
Labels: None yet

Projects: None yet

Milestone: No milestone

Linked issues: Successfully merging this pull request may close these issues. None yet

Notifications



This branch has no conflicts with the base branch

Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

First merge request!

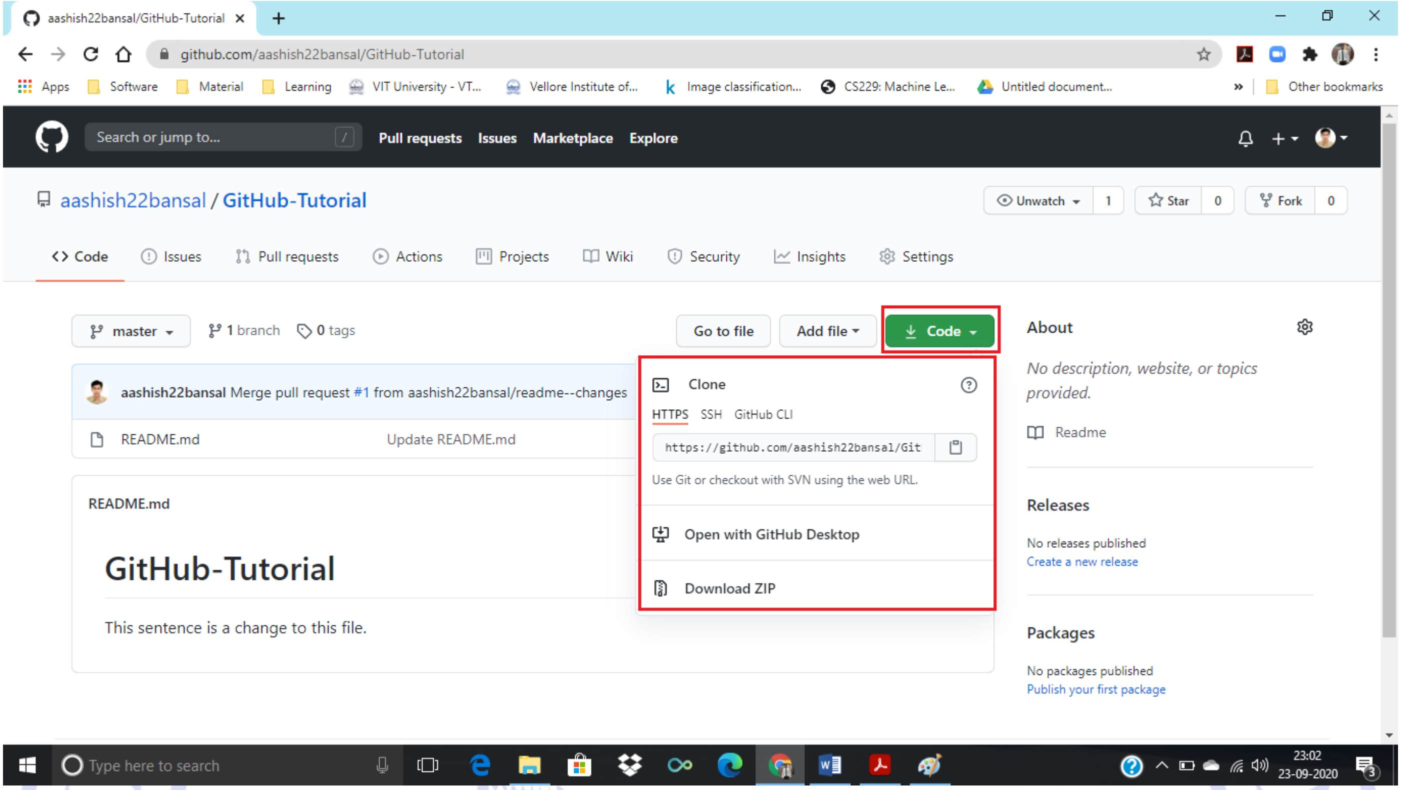
AayushiJohari merged commit `4f08be9` into `master` just now

Pull request successfully merged and closed

You're all set—the `readme--changes` branch can be safely deleted.

Step 4: Cloning and Forking GitHub Repository

Cloning: Suppose you want to use some code which is present in a public repository, you can directly copy the contents by cloning or downloading.



aashish22bansal Merge pull request #1 from aashish22bansal/readme--changes

README.md Update README.md

https://github.com/aashish22bansal/Git

Use Git or checkout with SVN using the web URL.

No description, website, or topics provided.

No releases published

No packages published

GitHub-Tutorial

This sentence is a change to this file.

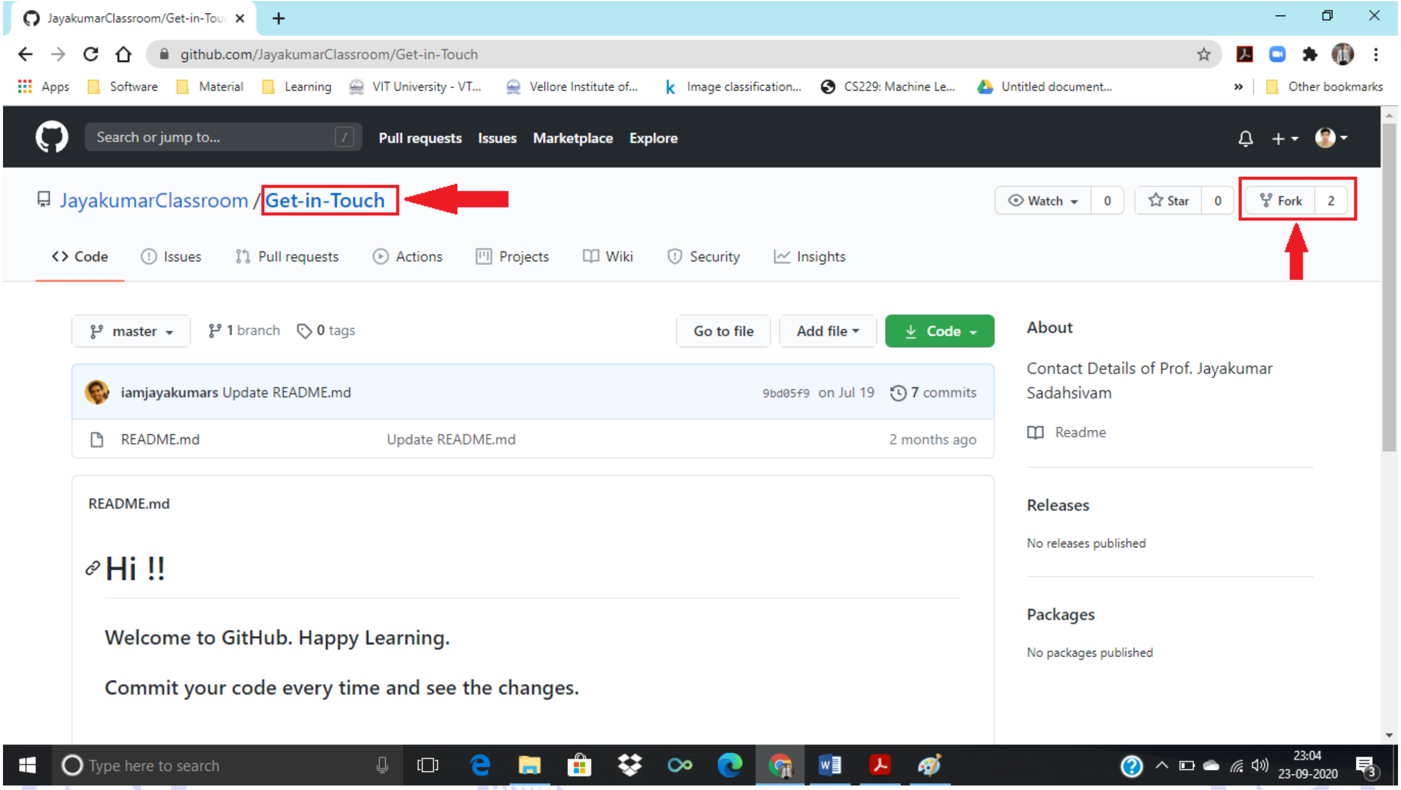
Forking: Suppose, you need some code which is present in a public repository, under your repository and GitHub account. For this, we need to fork a repository.

Before we get started with forking, there are some important points which you should always keep in mind.

- Changes done to the original repository will be reflected back to the forked repository.
- If you make a change in forked repository, it will not be reflected to the original repository until and unless you have made a pull request.

Now let’s see how can you want to fork a repository. For that, follow the below steps:

- Click “fork”. Note that this “Get-in-Touch” repository is already forked 2 times and it is under “google” account.

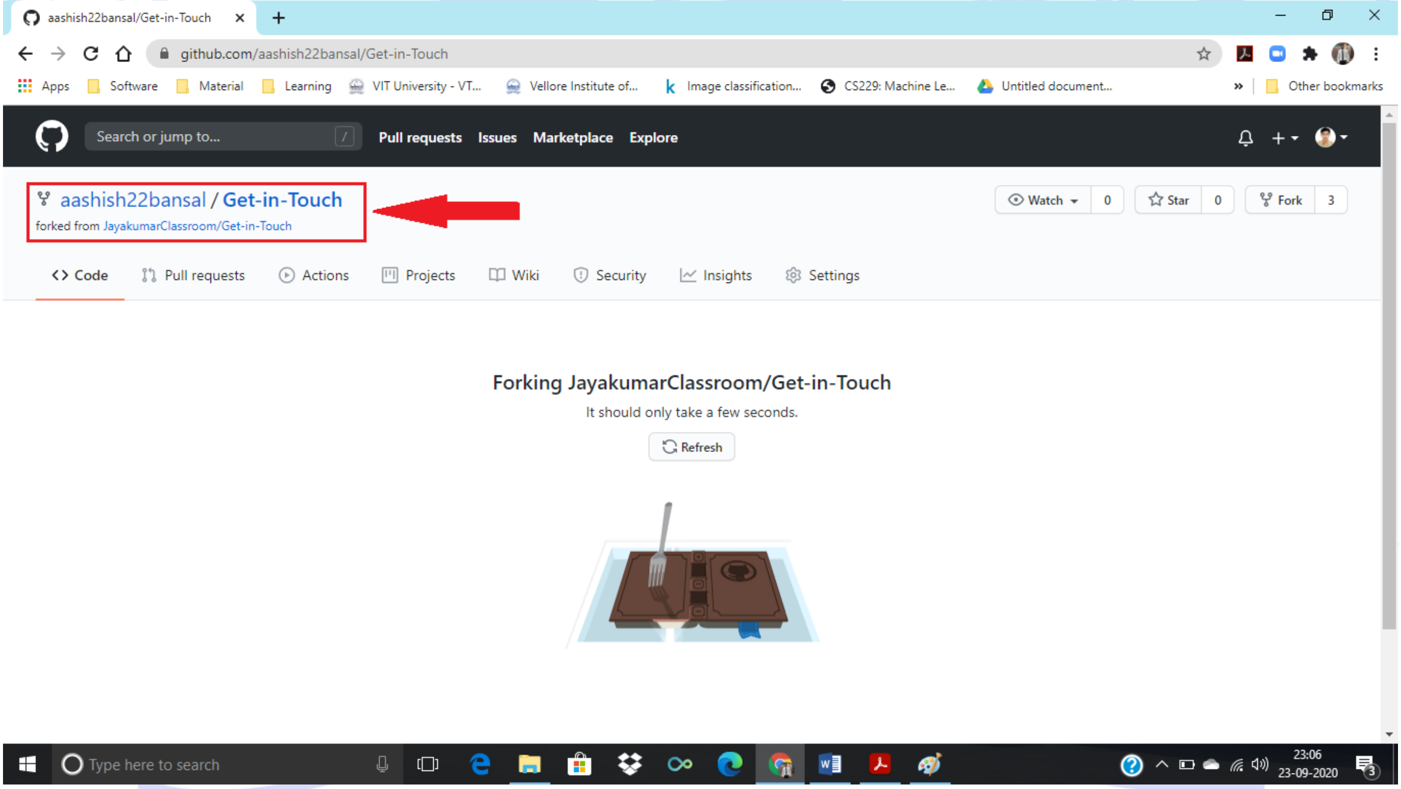


Hi !!

Welcome to GitHub. Happy Learning.

Commit your code every time and see the changes.

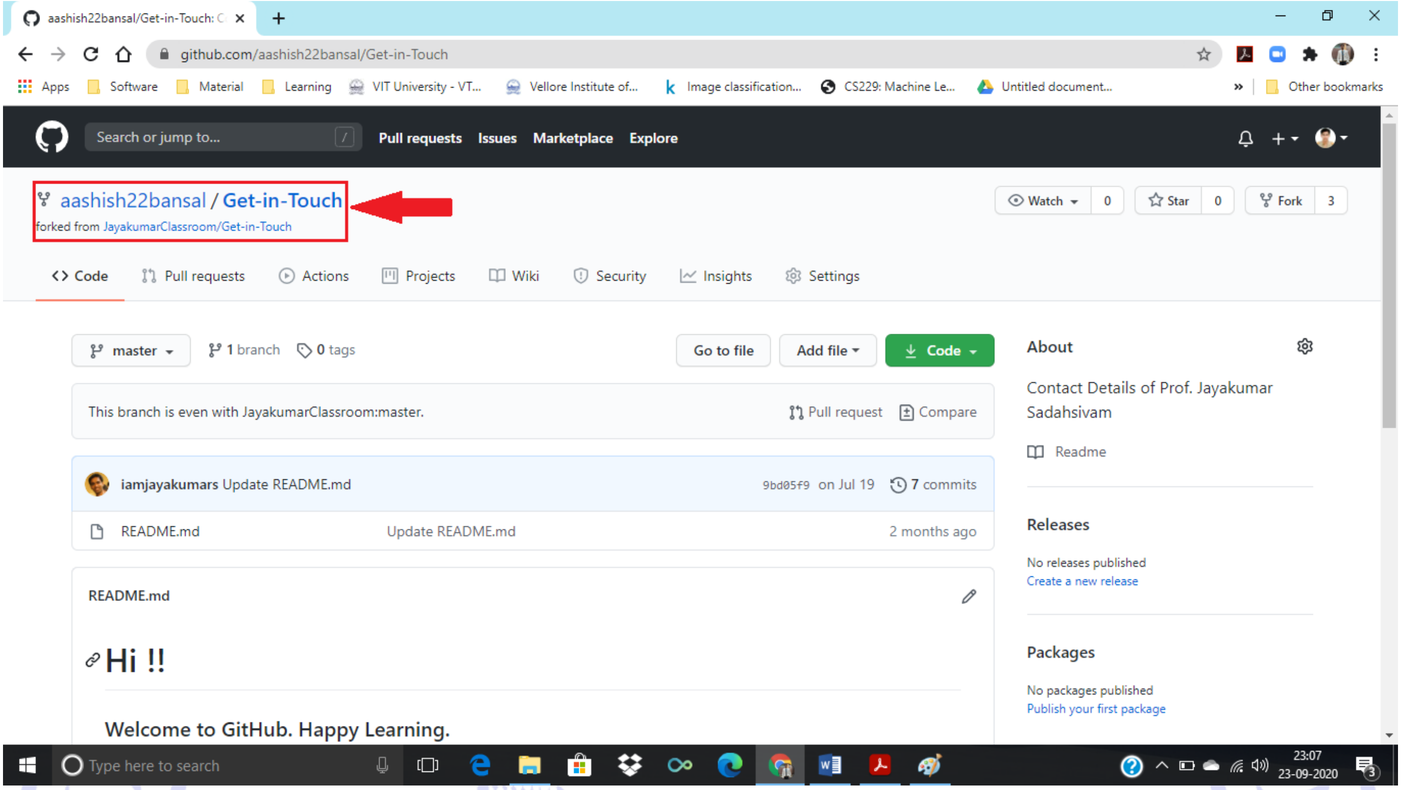
As soon as you click on “Fork”, it will take some time to fork the repository. Once done you will notice that the repository name is under your account.



github.com/aashish22bansal/Get-in-Touch
aashish22bansal / Get-in-Touch
forked from JayakumarClassroom/Get-in-Touch
Watch 0
Star 0
Fork 3
Code
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Refresh

Forking JayakumarClassroom/Get-in-Touch

It should only take a few seconds.



aashish22bansal / Get-in-Touch
forked from JayakumarClassroom/Get-in-Touch

This branch is even with JayakumarClassroom:master.

iamjayakumars Update README.md 9bd05f9 on Jul 19 7 commits

README.md Update README.md 2 months ago

README.md

Hi !!

Welcome to GitHub. Happy Learning.

About

Contact Details of Prof. Jayakumar Sadahsivam

Readme

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Different Ways to Access GitHub

While you can grant read/write access to collaborators on a personal repository, members of an organization can have more granular access permissions for the organization's repositories.

Personal user accounts

A repository owned by a user account has two permission levels: the *repository owner* and *collaborators*.

Organization accounts

Organization members can have *owner*, *billing manager*, or *member* roles. Owners have complete administrative access to your organization, while billing managers can manage billing settings. Member is the default role for everyone else. You can manage access permissions for multiple members at a time with teams

Enterprise accounts

Enterprise owners have ultimate power over the enterprise account and can take every action in the enterprise account. *Billing managers* can manage your enterprise account's billing settings. Members and outside collaborators of organizations owned by your enterprise account are automatically members of the enterprise account, although they have no access to the enterprise account itself or its settings. Enterprise accounts are available with GitHub Enterprise Cloud and GitHub Enterprise Server.

Question 2

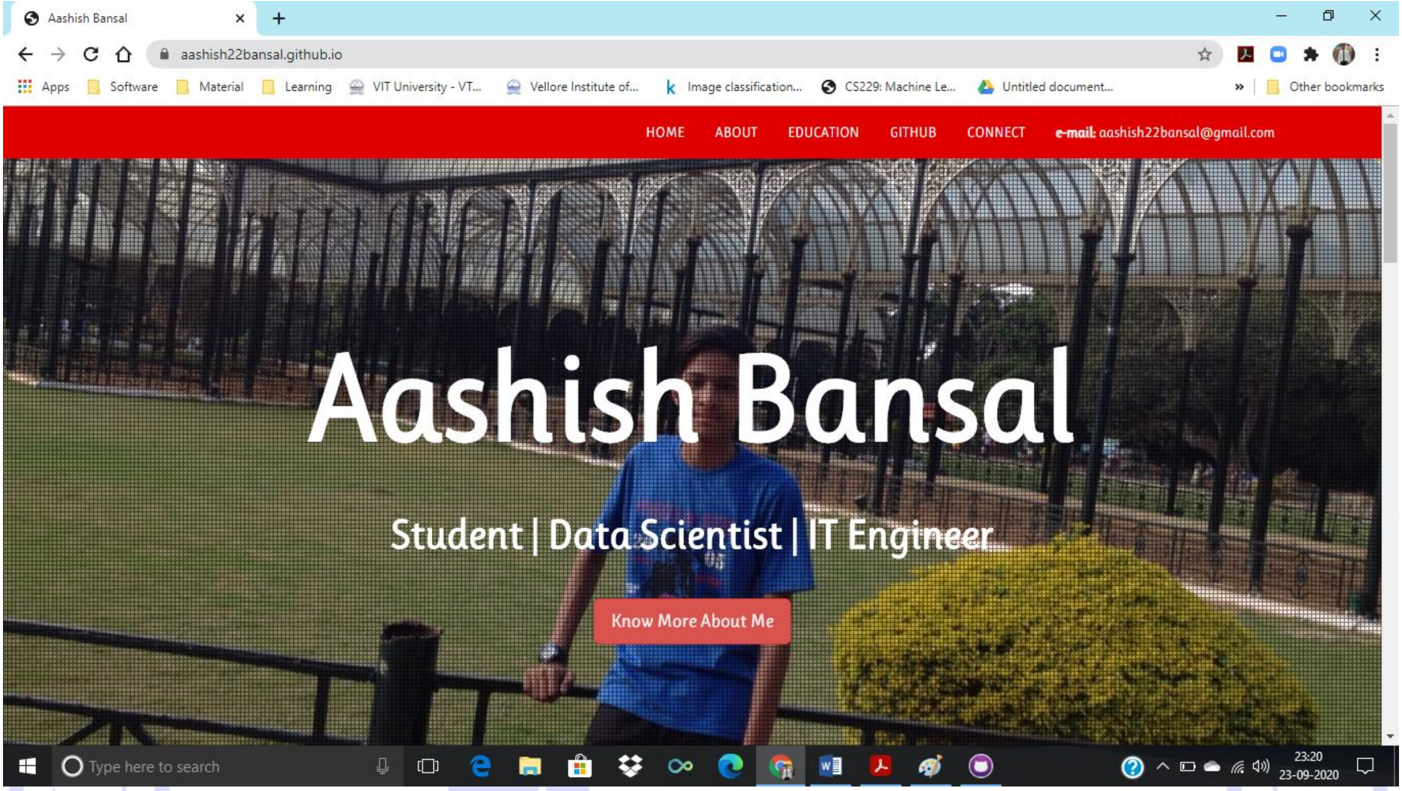
Host your Personal Portfolio in GitHub and provide the screenshot of the project and version history.

Answer

Website: https://aashish22bansal.github.io

Project Portfolio Website

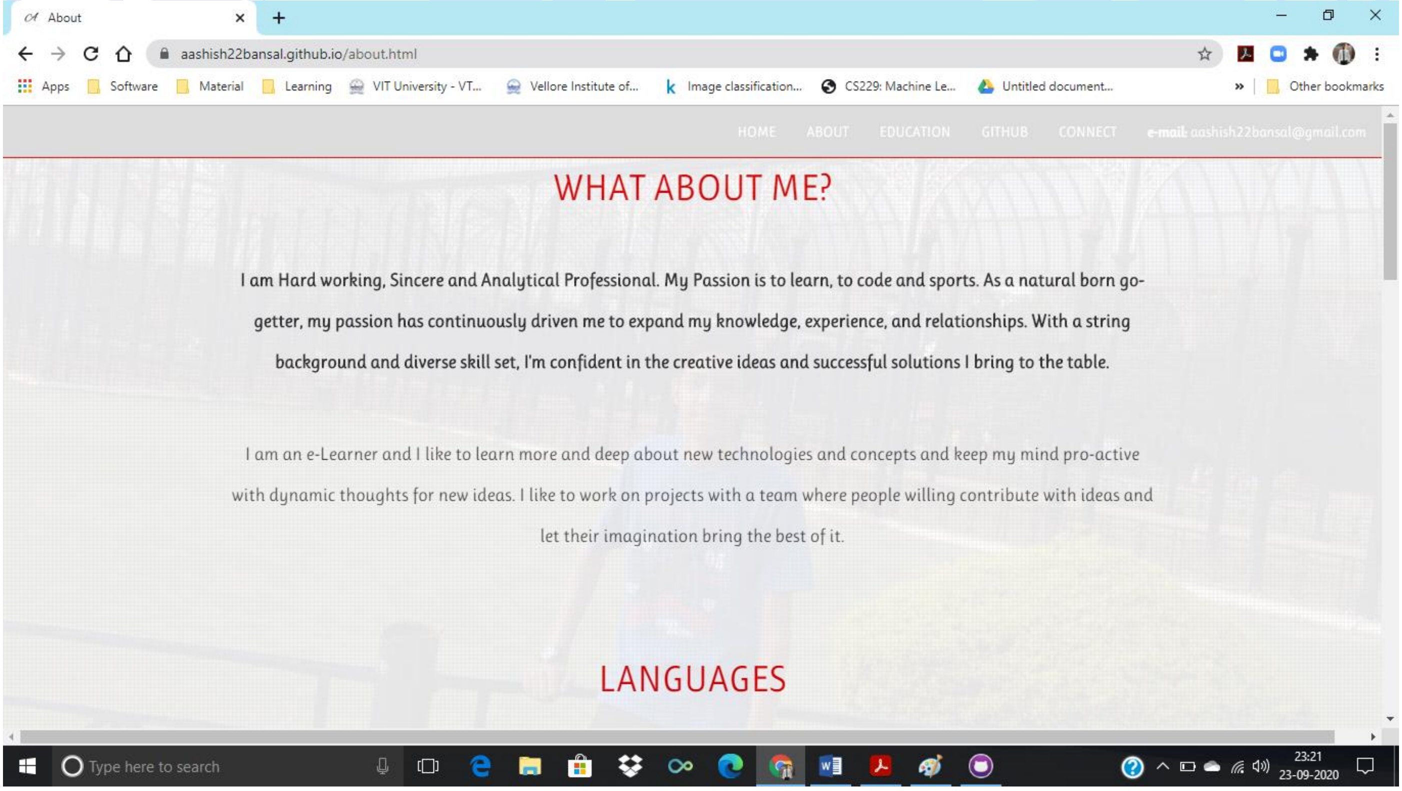
Home Page



Aashish Bansal

Student | Data Scientist | IT Engineer

About Page

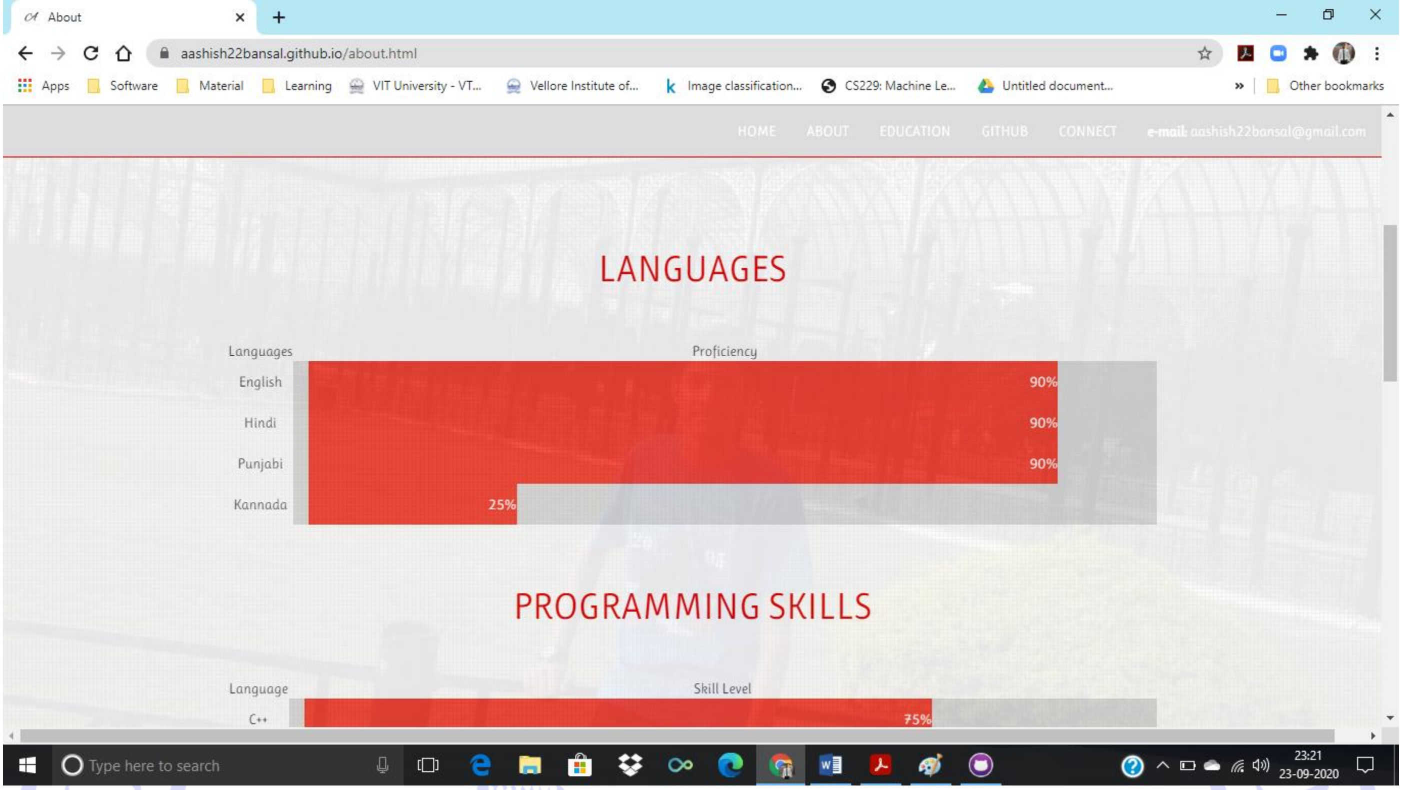


WHAT ABOUT ME?

I am Hard working, Sincere and Analytical Professional. My Passion is to learn, to code and sports. As a natural born go-getter, my passion has continuously driven me to expand my knowledge, experience, and relationships. With a string background and diverse skill set, I'm confident in the creative ideas and successful solutions I bring to the table.

I am an e-Learner and I like to learn more and deep about new technologies and concepts and keep my mind pro-active with dynamic thoughts for new ideas. I like to work on projects with a team where people willing contribute with ideas and let their imagination bring the best of it.

LANGUAGES

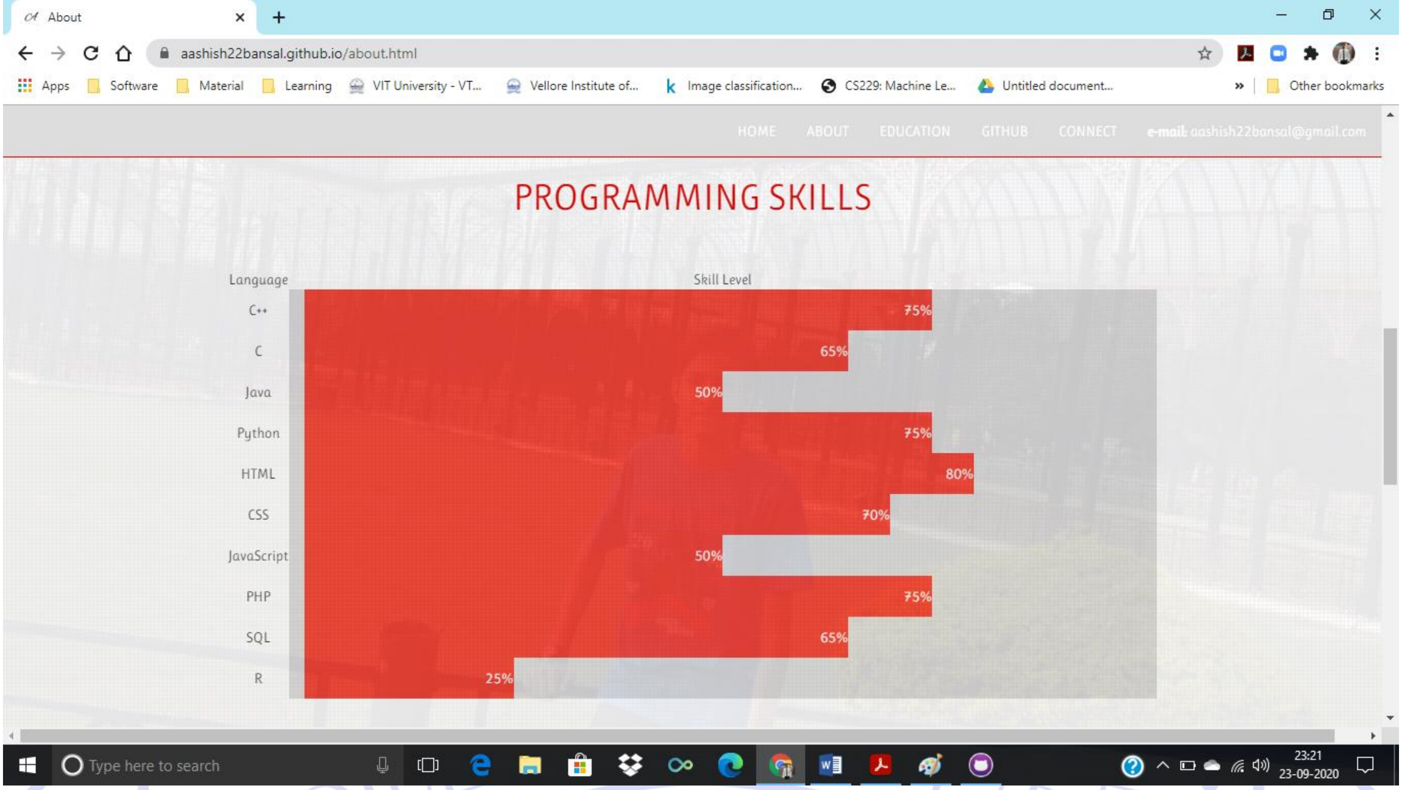


LANGUAGES

Languages	Proficiency
English	90%
Hindi	90%
Punjabi	90%
Kannada	25%

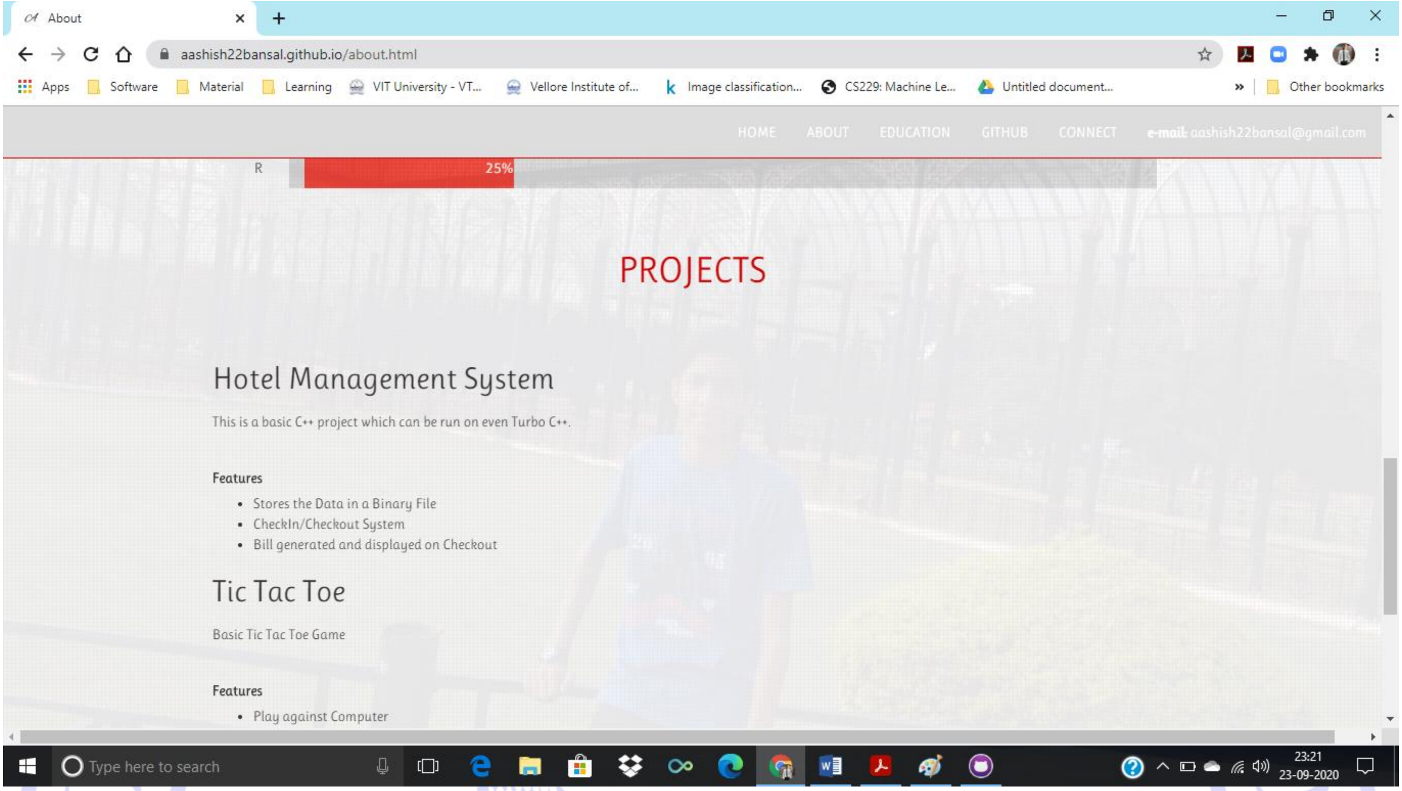
PROGRAMMING SKILLS

Language	Skill Level
C++	75%



PROGRAMMING SKILLS

Language	Skill Level
C++	75%
C	65%
Java	50%
Python	75%
HTML	80%
CSS	70%
JavaScript	50%
PHP	75%
SQL	65%
R	25%



PROJECTS

Hotel Management System

This is a basic C++ project which can be run on even Turbo C++.

Features

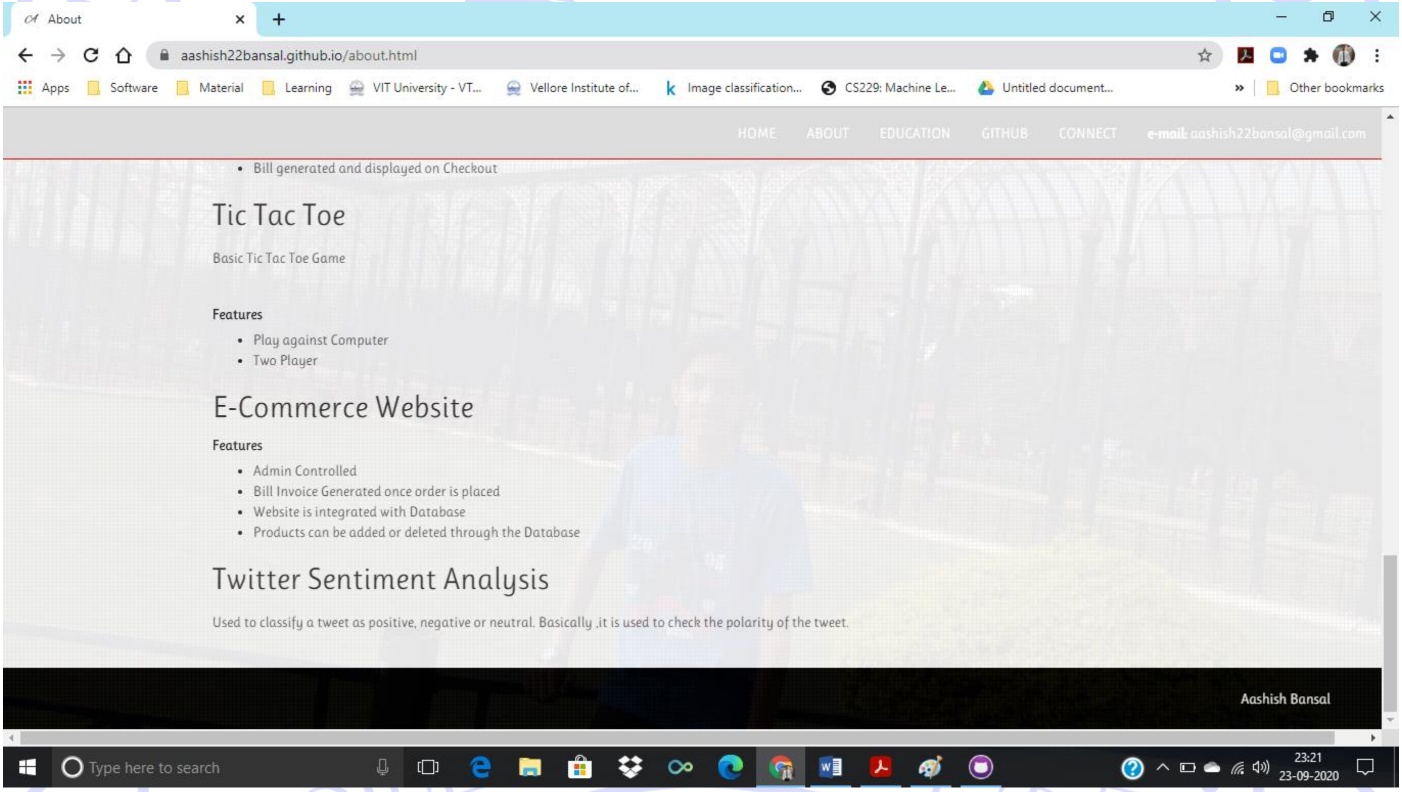
- Stores the Data in a Binary File
- CheckIn/Checkout System
- Bill generated and displayed on Checkout

Tic Tac Toe

Basic Tic Tac Toe Game

Features

- Play against Computer



- Bill generated and displayed on Checkout

Tic Tac Toe

Basic Tic Tac Toe Game

Features

- Play against Computer
- Two Player

E-Commerce Website

Features

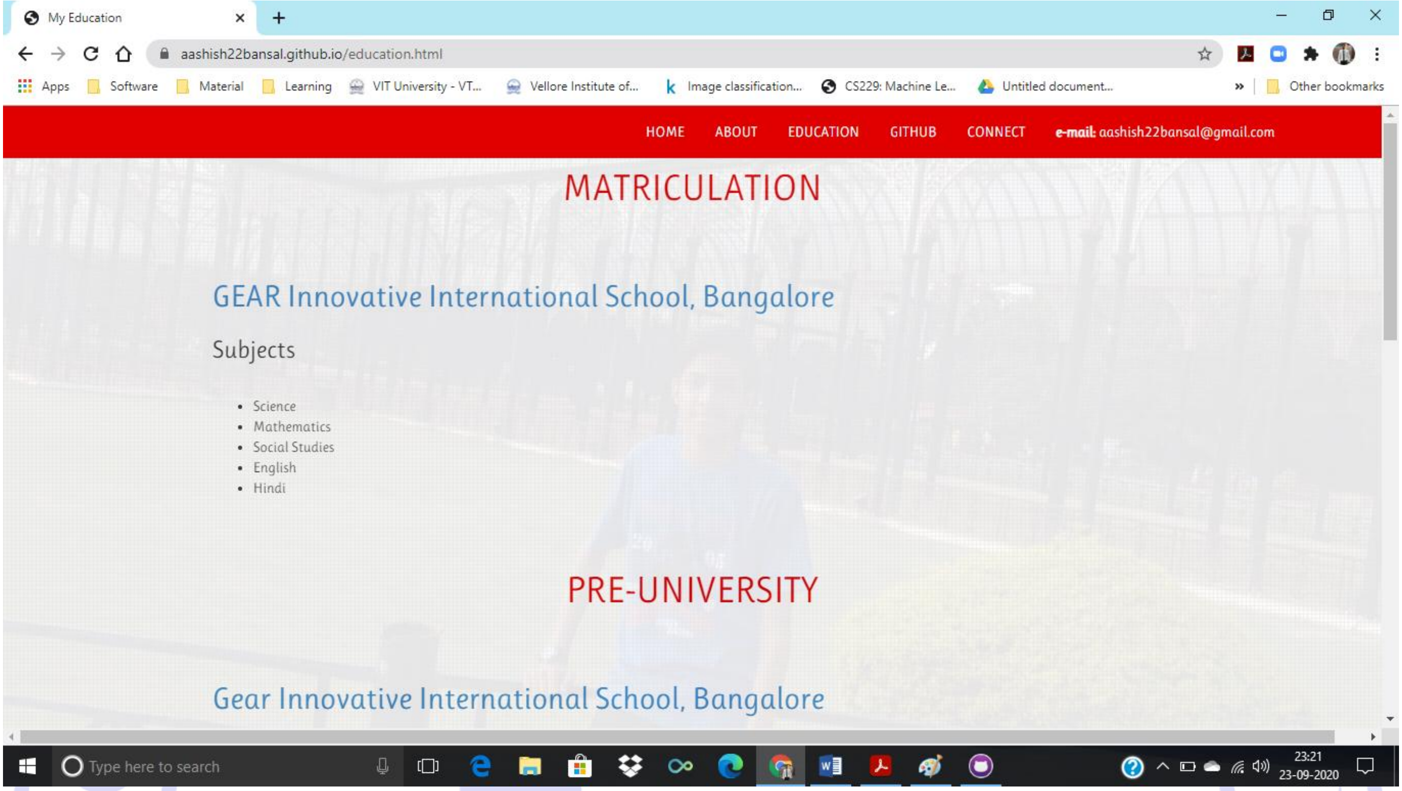
- Admin Controlled
- Bill Invoice Generated once order is placed
- Website is integrated with Database
- Products can be added or deleted through the Database

Twitter Sentiment Analysis

Used to classify a tweet as positive, negative or neutral. Basically ,it is used to check the polarity of the tweet.

Aashish Bansal

Education



MATRICULATION

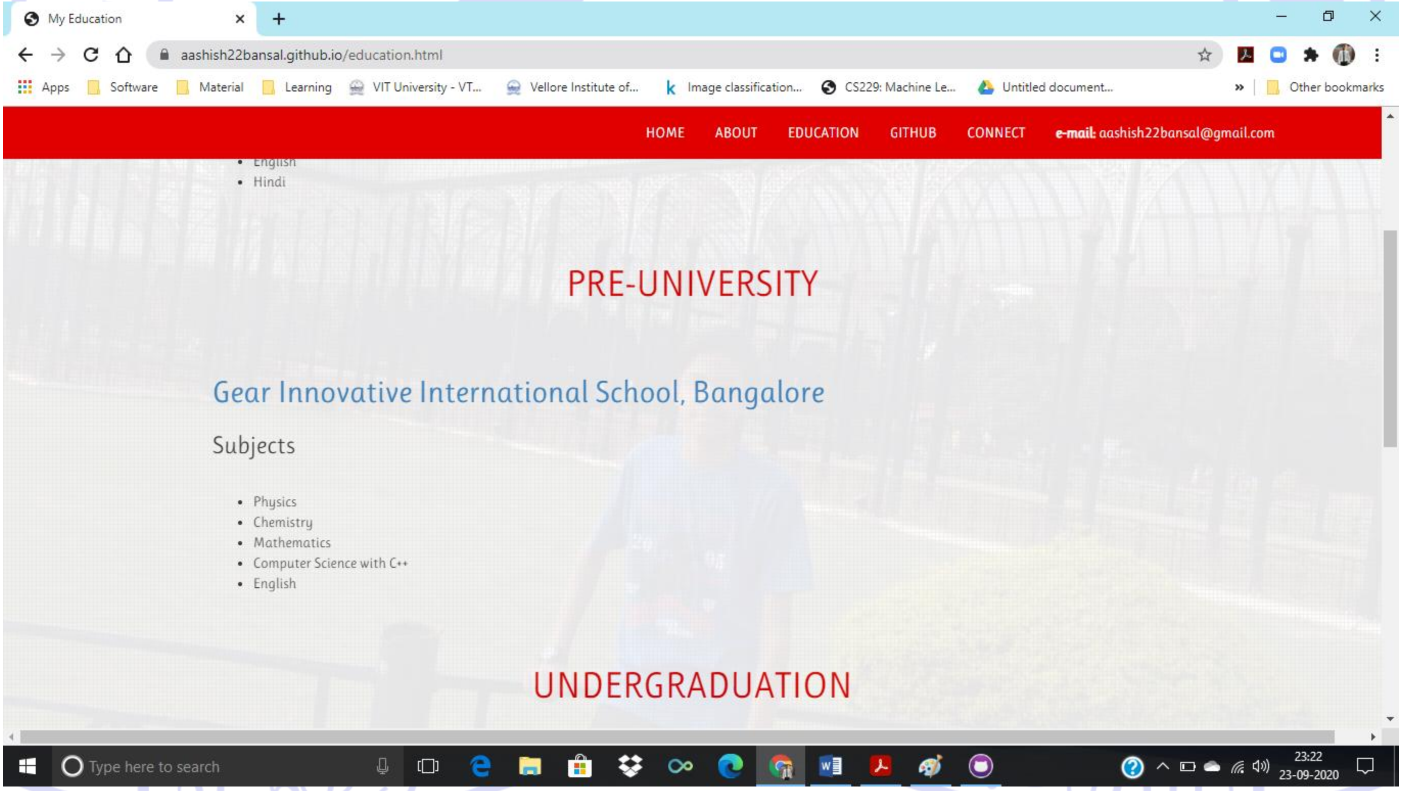
GEAR Innovative International School, Bangalore

Subjects

- Science
- Mathematics
- Social Studies
- English
- Hindi

PRE-UNIVERSITY

Gear Innovative International School, Bangalore



- English
- Hindi

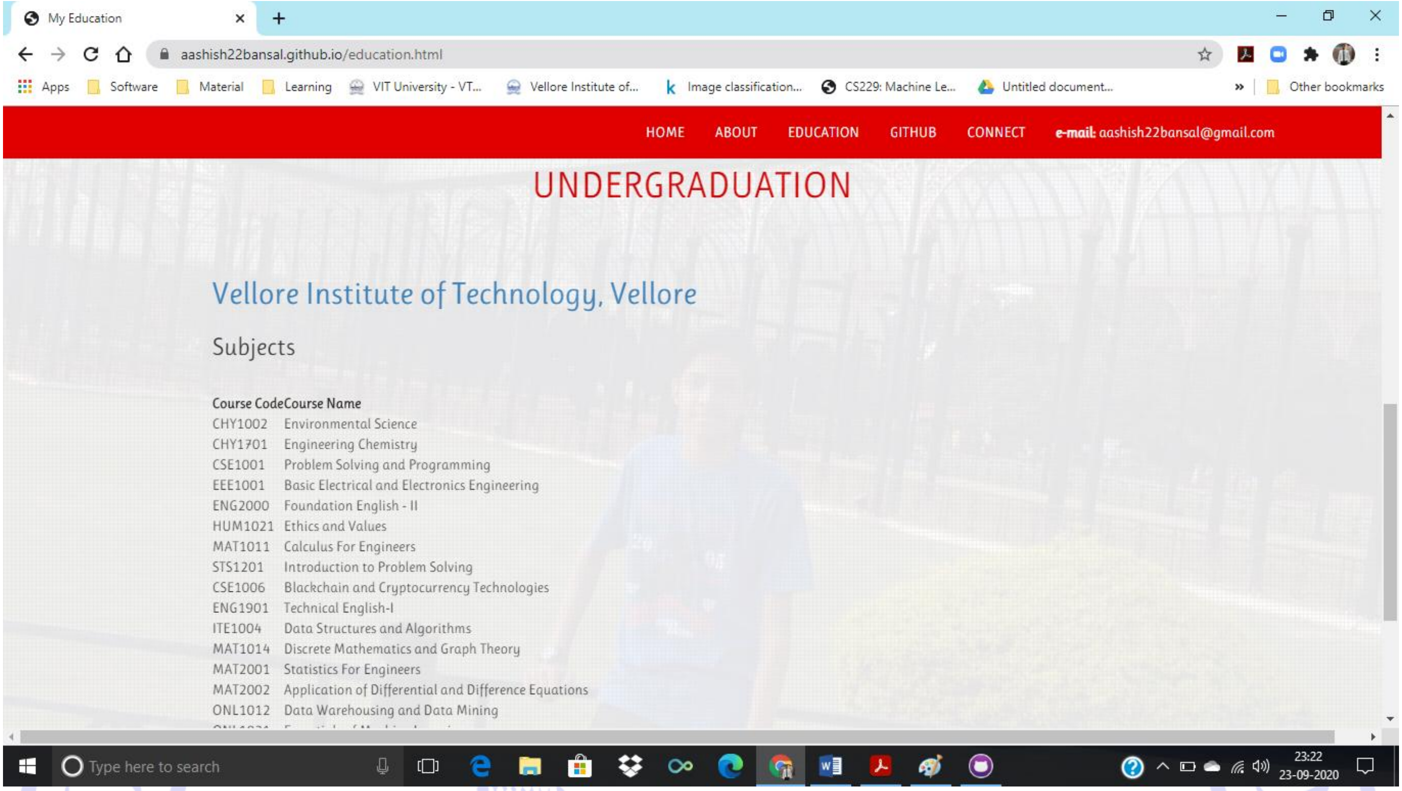
PRE-UNIVERSITY

Gear Innovative International School, Bangalore

Subjects

- Physics
- Chemistry
- Mathematics
- Computer Science with C++
- English

UNDERGRADUATION



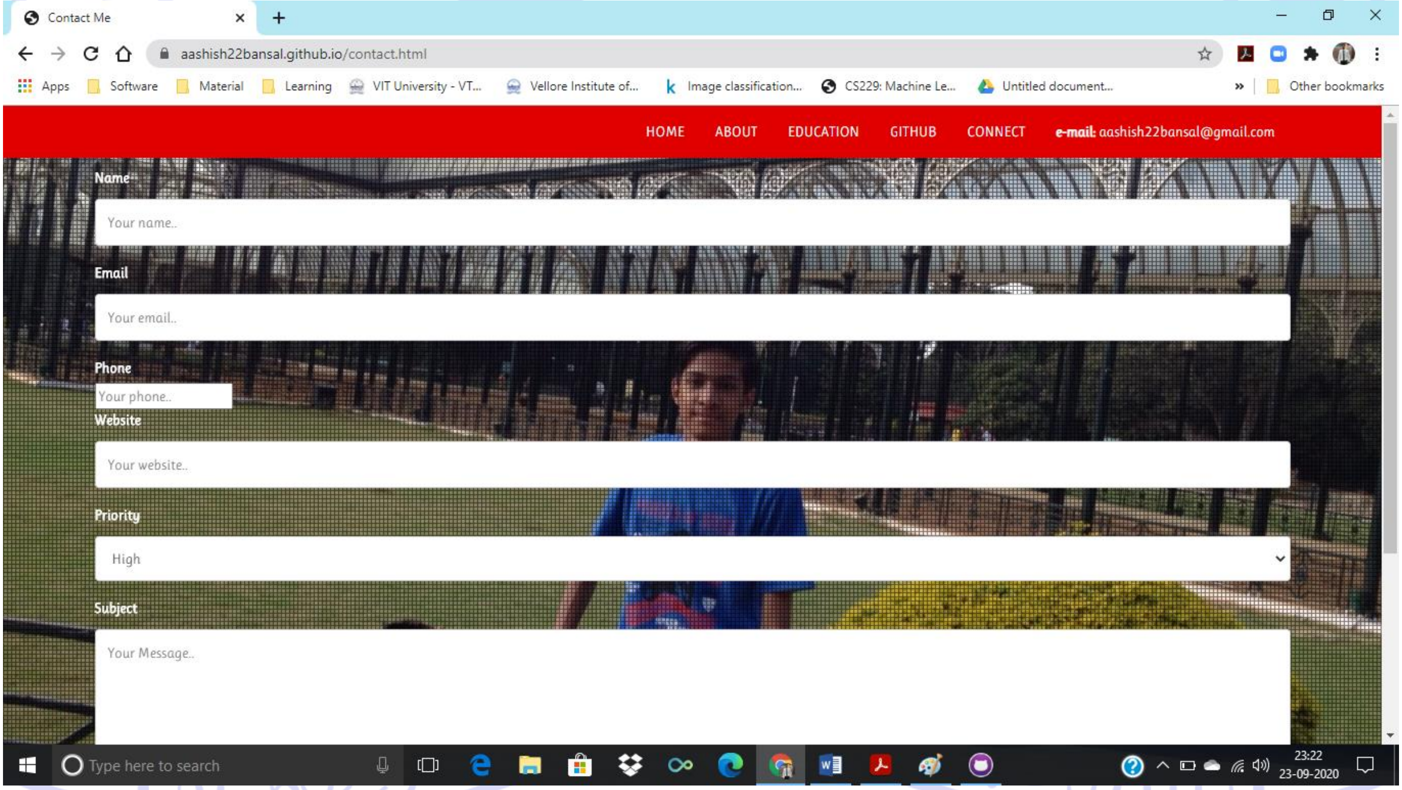
UNDERGRADUATION

Vellore Institute of Technology, Vellore

Subjects

Course Code	Course Name
CHY1002	Environmental Science
CHY1701	Engineering Chemistry
CSE1001	Problem Solving and Programming
EEE1001	Basic Electrical and Electronics Engineering
ENG2000	Foundation English - II
HUM1021	Ethics and Values
MAT1011	Calculus For Engineers
STS1201	Introduction to Problem Solving
CSE1006	Blackchain and Cryptocurrency Technologies
ENG1901	Technical English-I
ITE1004	Data Structures and Algorithms
MAT1014	Discrete Mathematics and Graph Theory
MAT2001	Statistics For Engineers
MAT2002	Application of Differential and Difference Equations
ONL1012	Data Warehousing and Data Mining

Connect



Name

Your name..

Email

Your email..

Phone

Your phone..

Website

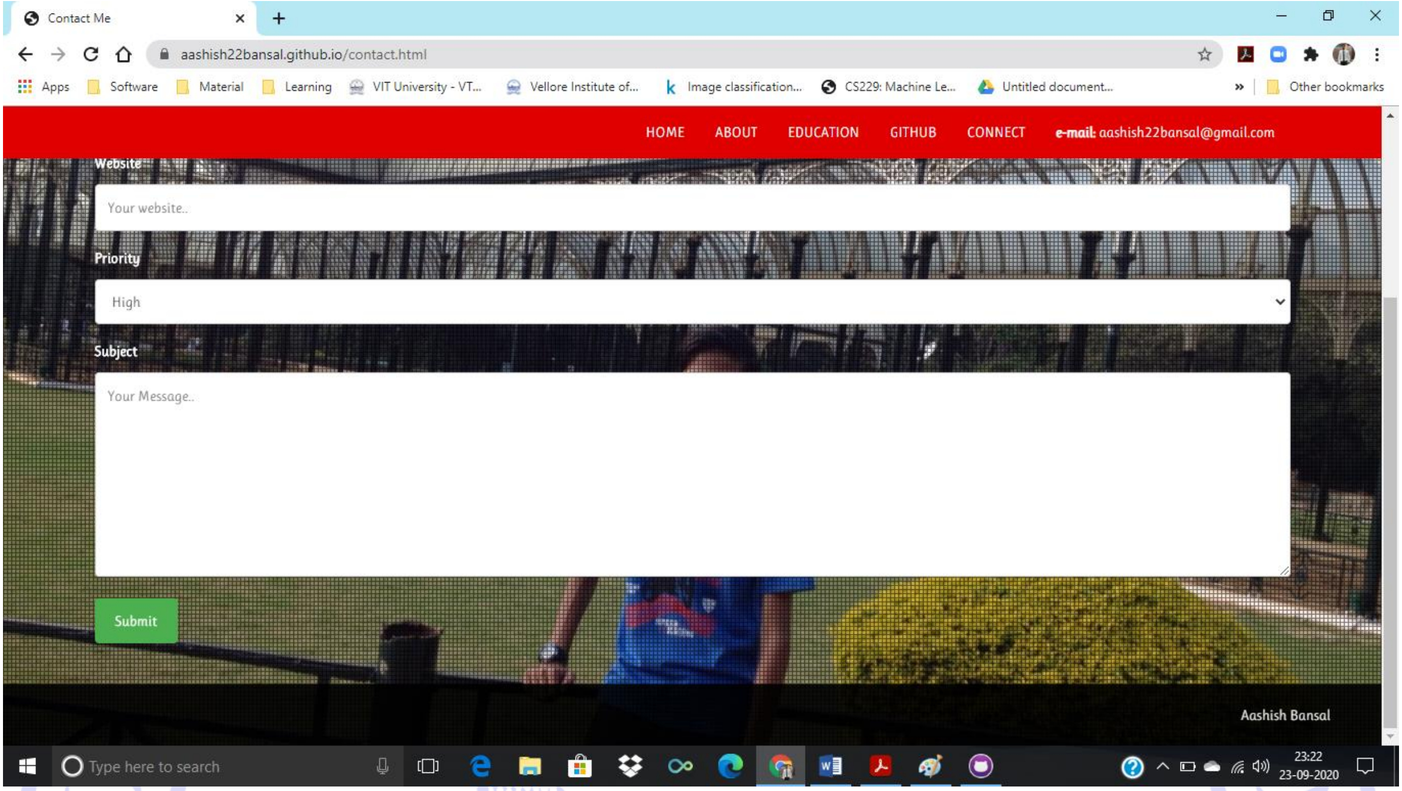
Your website..

Priority

High

Subject

Your Message..



Contact Me
aashish22bansal.github.io/contact.html
Apps Software Material Learning VIT University - VT... Vellore Institute of... Image classification... CS229: Machine Le... Untitled document... Other bookmarks
HOME ABOUT EDUCATION GITHUB CONNECT e-mail: aashish22bansal@gmail.com

Website

Your website..

Priority

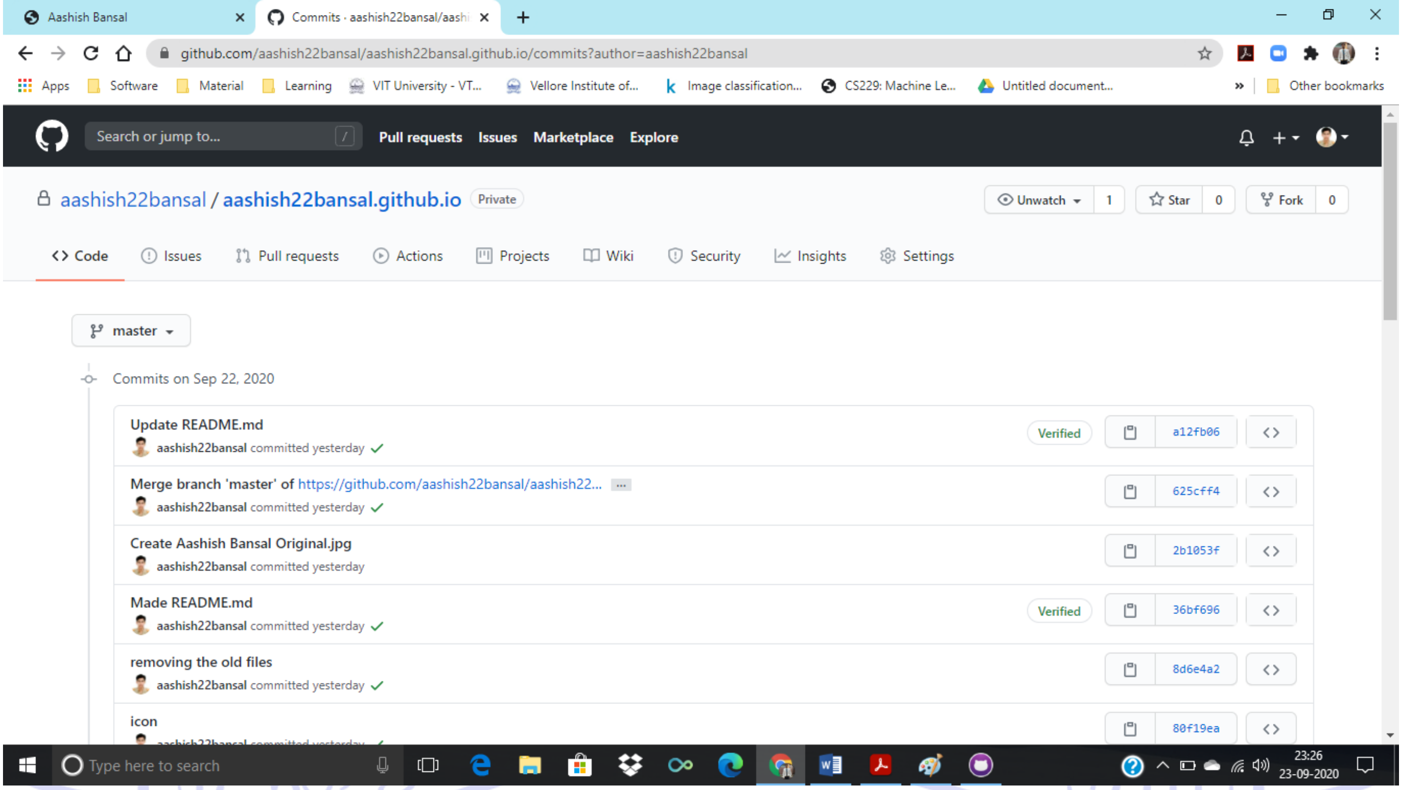
High

Subject

Your Message..

Submit
Aashish Bansal
Type here to search
23:22
23-09-2020

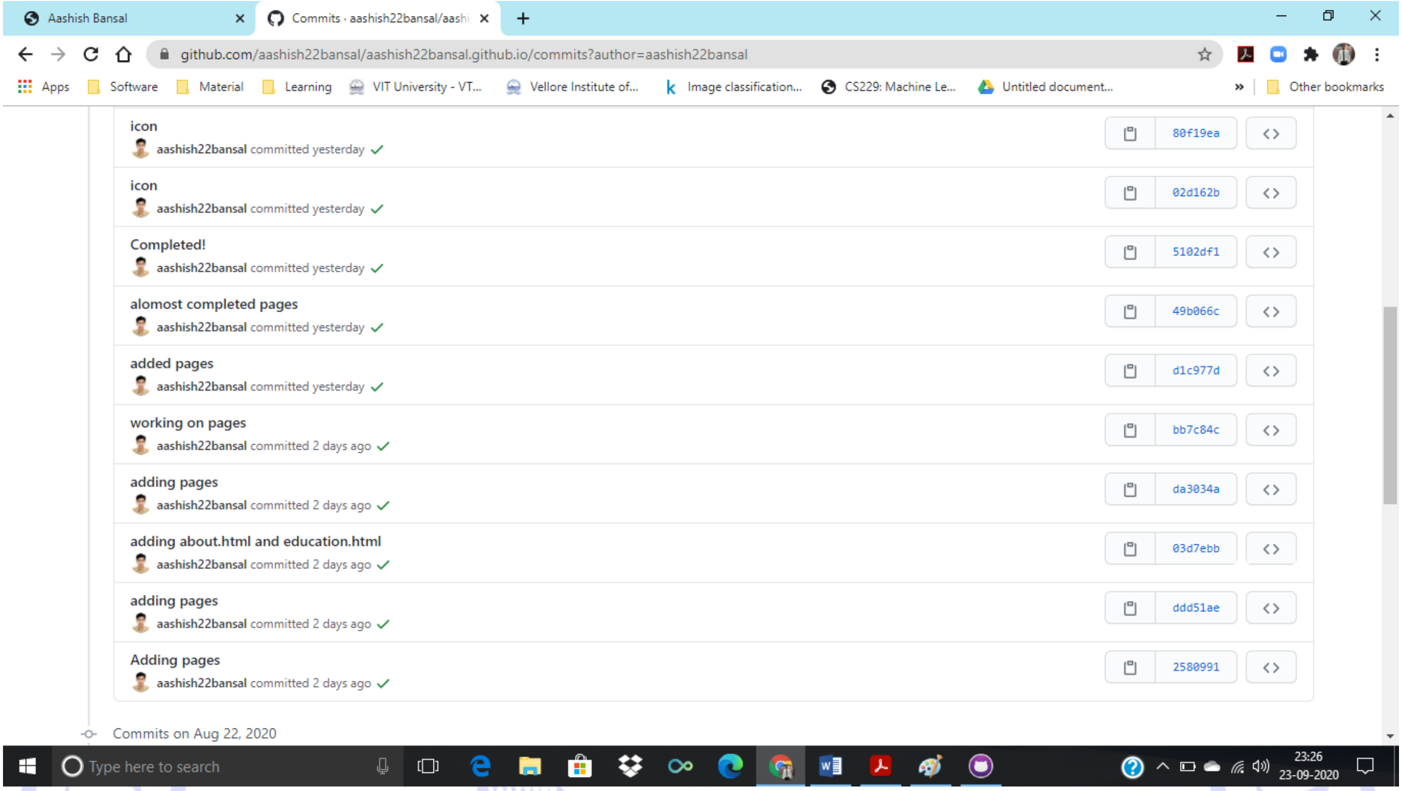
Version History



Aashish Bansal
Commits · aashish22bansal/aash
github.com/aashish22bansal/aashish22bansal.github.io/commits?author=aashish22bansal
Apps Software Material Learning VIT University - VT... Vellore Institute of... Image classification... CS229: Machine Le... Untitled document... Other bookmarks
Search or jump to...
Pull requests Issues Marketplace Explore
aashish22bansal / aashish22bansal.github.io Private
Unwatch 1 Star 0 Fork 0
Code Issues Pull requests Actions Projects Wiki Security Insights Settings
master

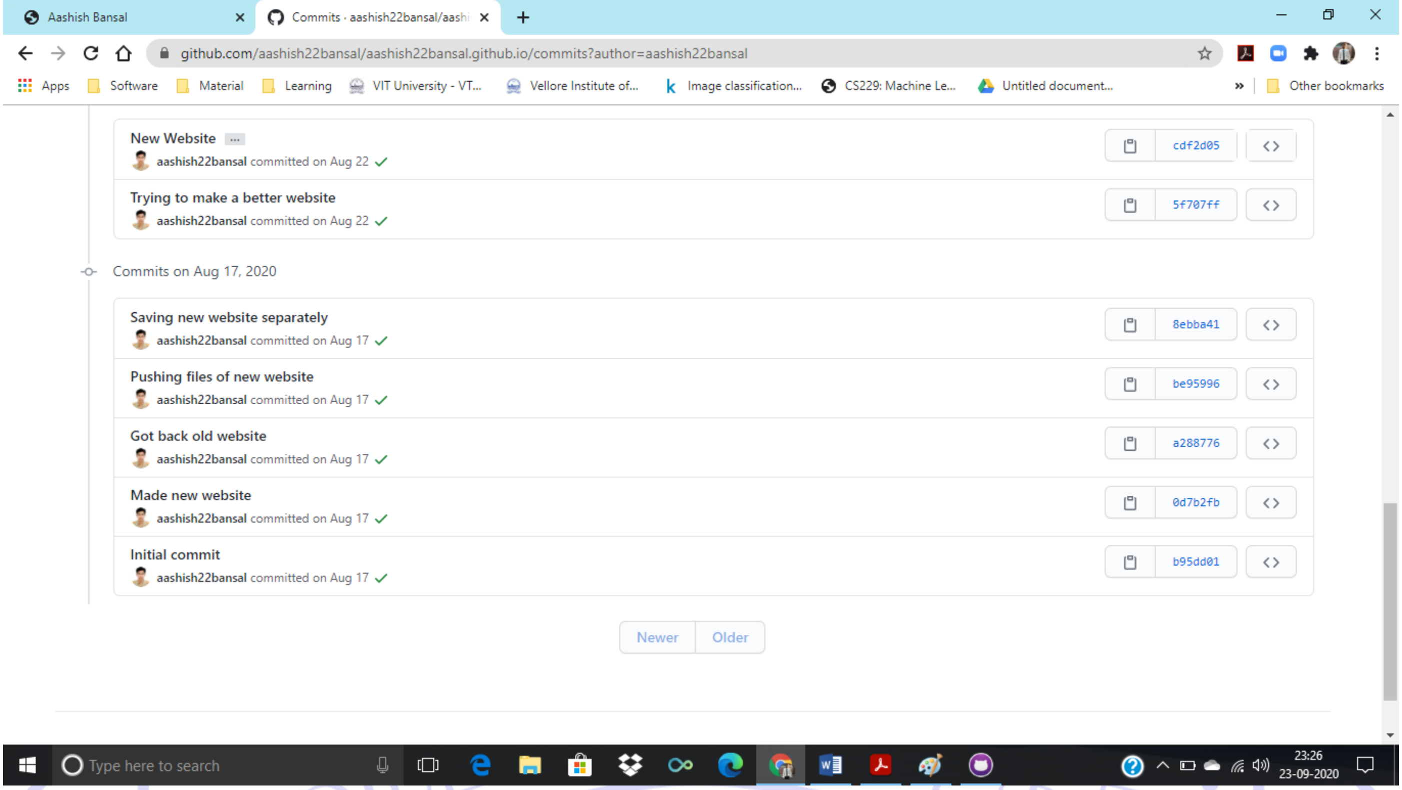
Commits on Sep 22, 2020

Update README.md aashish22bansal committed yesterday ✓	Verified	a12fb06
Merge branch 'master' of https://github.com/aashish22bansal/aashish22... … aashish22bansal committed yesterday ✓		625cff4
Create Aashish Bansal Original.jpg aashish22bansal committed yesterday		2b1053f
Made README.md aashish22bansal committed yesterday ✓	Verified	36bf696
removing the old files aashish22bansal committed yesterday ✓		8d6e4a2
icon aashish22bansal committed yesterday ✓		80f19ea



Commit	Author	Hash
icon	aashish22bansal committed yesterday ✓	80f19ea
icon	aashish22bansal committed yesterday ✓	02d162b
Completed!	aashish22bansal committed yesterday ✓	5102df1
alomost completed pages	aashish22bansal committed yesterday ✓	49b066c
added pages	aashish22bansal committed yesterday ✓	d1c977d
working on pages	aashish22bansal committed 2 days ago ✓	bb7c84c
adding pages	aashish22bansal committed 2 days ago ✓	da3034a
adding about.html and education.html	aashish22bansal committed 2 days ago ✓	03d7ebb
adding pages	aashish22bansal committed 2 days ago ✓	ddd51ae
Adding pages	aashish22bansal committed 2 days ago ✓	2580991

Commits on Aug 22, 2020



Commit	Author	Hash
New Website …	aashish22bansal committed on Aug 22 ✓	cdf2d05
Trying to make a better website	aashish22bansal committed on Aug 22 ✓	5f707ff

Commits on Aug 17, 2020

Commit	Author	Hash
Saving new website separately	aashish22bansal committed on Aug 17 ✓	8ebba41
Pushing files of new website	aashish22bansal committed on Aug 17 ✓	be95996
Got back old website	aashish22bansal committed on Aug 17 ✓	a288776
Made new website	aashish22bansal committed on Aug 17 ✓	0d7b2fb
Initial commit	aashish22bansal committed on Aug 17 ✓	b95dd01

Question 3

Write down the pros and cons of GitHub.

Answer

#	Pros	Cons
1	Different offering for continous providers	Lack of Command line configuration options

2	brings social aspect of programming into future	Reviewing large pull requests can be tedious and it can be tough to identify recent changes in new files or files with a lot of changes.
3	It makes it fast and easy to develop projects in versions/branches and easily to previous versions when necessary	It is hard to push unresolved merge conflicts
4	GitHub has a powerful UI for creating pull requests, with useful tools like inline commenting and suggested changes	we have to be careful with merge operations because a bad reverse will be painful to reverse
5	Pull History is always maintained and easy to search	when browsing the 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
6	It is easy for multiple team members to work on the same project and merge changes. All contributors are tracked so it is easy to identify contributors.	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
7	GitHub makes it easy to find what code has been changed and when	There's so much you can do with GitHub that it's fairly common for a user to possibly only use a small fraction of what GitHub can do. Improving GitHub’s discovery features would help surface some of the non-essential features that are quite useful.
8	It is easy to integrate with other tools	Lacks first-party support for mobile

Question 4

List down the features needs to be added in GitHub.

Answer

Some of the features which need to be added to GitHub are:

- Ability to move an issue from one project to another
- A way to distinguish the most active fork in a project
- Something to preview HTML based gists
- Ability to star issues
- A way to prioritize issues and sort by priority
- Prevent co-editing an issue comment
- Prevent editing another author comment/issue
- A Google group like or forum about a particular project

Question 5

Compare the minimum of three version control applications.

Answer

Concurrent Versions System (CVS)

CVS has been around since the 80s, and has been very popular with both commercial and open source developers.

It is released under the GNU license, and uses a system to let users “check out” the code they are going to work on and “check in” their changes.

Originally, CVS handled conflicts between two programmers by only allowing for the latest version of the code to be worked on and updated. As such, it was a first come, first serve system where the user must publish changes quickly to ensure that other users haven’t beat them to the punch.

Now, CVS can handle branching projects so the developed software can diverge into different products with unique features and will be reconciled at a later time.

The CVS server runs on Unix-like systems with client software that runs on multiple operating systems. It is considered the most mature version control system because it has been developed for such a long time and does not receive many requests for new features at this time.

A fork project of CVS, CVSNT was created to run CVS on Windows servers, and it is currently being actively developed to increase functionality.

Pros:

- Has been in use for many years and is considered mature technology

Cons:

- Moving or renaming files does not include a version update
- Security risks from symbolic links to files
- No atomic operation support, leading to source corruption
- Branch operations are expensive as it is not designed for long-term branching

Comparison with Git

The main difference is that (as it was already said in other responses) CVS is (old) centralized version control system, while Git is distributed.

But even if you use version control for single developer, on single machine (single account), there are a few differences between Git and CVS:

- **Setting up repository**. Git stores repository in .git directory in top directory of your project; CVS require setting up CVSROOT, a central place for storing version control info for different projects (modules). The consequence of that design for user is that importing existing sources into version control is as simple as "git init && git add . && git commit" in Git, while it is more complicated in CVS.
- **Atomic operations**. Because CVS at beginning was a set of scripts around per-file RCS version control system, commits (and other operations) are not atomic in CVS; if an operation on the repository is interrupted in the middle, the repository can be left in an inconsistent state. In Git all operations are atomic: either they succeed as whole, or they fail without any changes.
- **Changesets**. Changes in CVS are per file, while changes (commits) in Git they always refer to the whole project. This is very important *paradigm shift*. One of consequences of this is that it is very easy in Git to revert (create a change that undoes) or undo *whole* change; other consequence is that in CVS is easy to do partial checkouts, while it is currently next to impossible in Git. The fact that changes are per-file, grouped together led to invention of GNU Changelog format for commit messages in CVS; Git users use (and some Git tools expect) different convention, with single line describing (summarizing) change, followed by empty line, followed by more detailed description of changes.

If you are collaboration with at least one other developer, you would find also the following differences between Git and CVS:

- **Commit before merge** Git uses *commit-before-merge* rather than, like CVS, *merge-before-commit* (or *update-then-commit*). If while you were editing files, preparing for creating new commit (new revision) somebody other created new commit on the same branch and it is now in repository, CVS forces you to first update your working directory and resolve conflicts before allowing you to commit. This is not the case with Git. You first commit, saving your state in version control, then you merge other developer changes. You can also ask the other developer to do the merge and resolve conflicts.

- If you prefer to have linear history and avoid merges, you can always use *commit-merge-recommit* workflow via "git rebase" (and "git pull --rebase"), which is similar to CVS in that you replay your changes on top of updated state. But you always commit first.
- **No need for central repository** With Git there is no need to have single central place where you commit your changes. Each developer can have its own repository (or better repositories: private one in which he/she does development, and public bare one where she/he publishes that part which is ready), and they can pull/fetch from each other repositories, in symmetric fashion. On the other hand, it is common for larger project to have *socially* defined/nominated central repository from which everyone pull from (get changes from).

Apache Subversion (SVN)

SVN was created as an alternative to CVS that would fix some bugs in the CVS system while maintaining high compatibility with it.

Like CVS, SVN is free and open source with the difference of being distributed under the Apache license as opposed to GNU.

To prevent corruption in the database from being corrupted, SVN employs a concept called atomic operations. Either all of the changes made to the source are applied or none are applied, meaning that no partial changes will break the original source.

Many developers have switched to SVN as it is a newer technology that takes the best features of CVS and improves upon them.

While CVS's branch operations are expensive and do not really lend themselves to long-term forks in the project, SVN is designed to allow for it, lending itself better to large, forked projects with many directions.

Criticism of SVN includes slower comparative speed and the lack of distributed revision control. Distributed revision control uses a peer-to-peer model rather than using a centralized server to store code updates. While a peer-to-peer model would work better for world-wide, open source projects, it may not be ideal in other situations. The downside to a dedicated server approach is that when the server is down, no clients are able to access the code.

Pros:

- Newer system based on CVS
- Includes atomic operations
- Cheaper branch operations
- Wide variety of plug-ins for IDEs
- Does not use peer-to-peer model

Cons:

- Still contains bugs relating to renaming files and directories
- Insufficient repository management commands
- Slower comparative speed

Comparison with Git

While Git and SVN are both enterprise version control systems (VCS) that help with workflow and project management in coding, they do have their differences. The difference between Git and SVN version control systems is that Git is a distributed version control system, whereas SVN is a centralized version control system. Git uses multiple repositories including a centralized repository and server, as well as some local repositories. SVN does not have a centralized repository or server.

SVN allows you to check out sub-trees (or branches) only whereas Git requires you to check out the entire repository as a unit. This is because there is a .svn in each one of your folders while git only has one .git at the top level parent directory.

Git and SVN are each viable workflow and version control systems, but for different reasons. Git may have more difficulty compressing and storing binary files, while SVN doesn't as much. That said, many claim Git is better than SVN because it works well even for developers who aren't always connected to the master repository, as it is available offline. Branching and merging support are also thought to be superior with Git. When it comes to disk space storage, it's pretty close to equal between both SVN and Git repositories. Git is also a bit newer than SVN.

Mercurial

Mercurial began close to the same time as Git and is also a distributed revision control tool.

It was originally made to compete with Git for Linux kernel development, and as Git was selected, Mercurial has seen less success in that area. However, that is not to say that it is not used as many major developments use it, including OpenOffice.org.

It's different from other revision control systems in that Mercurial is primarily implemented in Python as opposed to C, but there are some instances where C is used.

Due to its distributed nature and its creation in Python, the Python language developers are considering a switch to Mercurial as it would allow non-core developers to have easier access to creating new trees and reverting changes.

Users have noted that Mercurial shares some features with SVN as well as being a distributed system, and because of the similarities, the learning curve for those already familiar with SVN will be less steep. The documentation for Mercurial also is more complete and will facilitate learning the differences faster.

Some of the major drawbacks to Mercurial include that it doesn't allow for two parents to be merged and unlike Git, it uses an extension system rather than being scriptable. That may be ideal for some programmers, but many find the power of Git to be a feature they don't want to trade off.

Pros:

- Easier to learn than Git
- Better documentation
- Distributed model

Cons:

- No merging of two parents
- Extension-based rather than script ability
- Less out of the box power

Comparison with Git

Both Mercurial and Git are DVCS. This allows developers to bring a repo full of code down to their workstations, perform their work items, and then put it back into a central server. Although this means you can work offline, that's less of a concern than it used to be. (Who isn't always online?)

Once each developer's work is completed, it's merged with everyone else's work in the "dev codeline." It gets tested, and then, at some point (hopefully sooner rather than later), it goes into production.

At the end of the day, Git and Mercurial do the same thing — they help you manage the version history of your source code. Yet there are big differences between them.

Here's a breakdown of all of the key differences between Git vs. Mercurial.

1. Usability: Is Git or Mercurial Easier to Use?

When comparing Git vs. Mercurial, one of the biggest differences is the level of expertise needed to use each system.

Git Is More Complex, With More Commands

Git is more complex, and it requires your team to know it inside and out before using it safely and effectively. With Git, one sloppy developer can cause major damage for the whole team. Its documentation is also harder to understand.

In general, if your team is less experienced, or if you have non-technical people on your team, Mercurial might be the better solution.

If you opt for Git, consider choosing an intuitive GUI to accompany it. Using Git on the command-line can cause nightmares, especially in the beginning. The commands in Git are long and come with multiple options, which increases the difficulty.

Mercurial Is Simpler

Mercurial's syntax is simpler, and the documentation is easier to understand. Furthermore, it works the way a tool should — you don't think about it while using it. Conversely, with Git, you might end up spending time figuring out finicky behavior and pouring over forums for help.

Once you get over the learning curve, Git offers teams more flexibility.

2. Security: Is Git or Mercurial More Secure?

It seems strange, but you could argue that Git and Mercurial are each more secure than the other, and you wouldn't be contradicting yourself. Which is to say, neither of them offers the security most teams need.

Git Is Better for Experienced Users

Security in Git vs. Mercurial depends on your level of technical expertise. Mercurial may be safer for less experienced users, but Git can offer ways to enhance safety (once you know what you are doing).

Neither VCS offers the security that prevents people from accidentally or intentionally compromising code. In short, neither offers the robust security that most teams need.

Mercurial Is Safer for Less Experienced Users

By default, Mercurial doesn't allow you to change history. However, Git allows all involved developers to change the version history. Obviously, this can have disastrous consequences. With basic Mercurial, you can only change your last commit with "hg commit – amend".

Git also stores every change made for 30 days in reflog. For example, you can modify the commit message of the latest commit, and revert it for 30 days. However, the changes can only be made locally, as these changes are not pushed to the remote repository by default. After 30 days, the changes are collected, meaning you can no longer revert.

3. Branching: Is Mercurial vs. Git Better?

Branching means working with files — source code — that you want to modify. Branches allow you to work on different versions of your code at the same time. Then developers can merge the changes, without (hopefully) breaking the code base.

Git's Branching Model Is More Effective

One of the main reasons developers swear by Git is its effective branching model. In Git, branches are only references to a certain commit. This makes them lightweight yet powerful.

Git allows you to create, delete, and change a branch anytime, without affecting the commits. If you need to test a new feature or find a bug — make a branch, do the changes, and then delete the branch.

Git supports the idea of a staging area, which is also known as the index file.

Staging is the practice of *adding* files for your next commit. It allows you to choose which files to commit next. This is useful when you don't want to commit all changed files together.

Mercurial’s Branching Model Can Cause Confusion

Branching in Mercurial doesn't share the same meaning. This can make it more cumbersome. In Mercurial, branches refer to a linear line of consecutive changesets. Changesets (csets) refer to a complete set of changes made to a file in a repository.

Mercurial embeds the branches in the commits, where they are stored forever. This means that branches cannot be removed because that would alter the history. However, you can refer to certain commits with bookmarks, and use them in a similar manner to Git’s branches.

Lastly, Mercurial requires you take added care not to push code to the wrong branch, especially if you haven’t clearly named them. You can easily avoid this in Git.

In Mercurial, there is no index or staging area before the commit. Changes are committed as they are in the working directory. This might be an adequate and simple solution for many people. But if you want the option to choose which parts of the working directory you commit, you can use a Mercurial extension, such as DirState or Mercurial Queues.