



DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

Title: GitHub

INTEGRATED DESIGN PROJECT II
CSE 406



GREEN UNIVERSITY OF BANGLADESH

1 Objective(s)

- To learn about the version control system
- To differentiate between Git and GitHub
- To implement a version control system for the projects

2 Introduction

What is a version control system?

Version control systems (VCS) are software tools that help software teams manage changes to source code over time. Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members. Examples: Git, Mercurial, Bitbucket, etc.

What are Git and GitHub?

Git is an open-source distributed version control system. Developers can access Git via a command line (terminal) or a desktop app that has a GUI (graphical user interface), such as Sourcetree, GitKraken etc. Git runs locally and the files and history are stored on the local machine. By contrast, GitHub is a web-based hosting service for Git repositories. It makes Git more user-friendly and also provides a platform for developers to share code with others. In addition, GitHub makes it easy for others to contribute to projects. You do not need GitHub to use Git, but you cannot use GitHub without using git.

Benefits of the version control system.

- Tracking changes: provides the ability to track changes in files and code, keeping an extensive history of all activity on a collection of code (a repository).
- Collaboration: When a developer starts working on the project, they can pull (or download) the code locally to their computer. After finishing making changes, they push (or upload) the code back to the remote repository on a separate branch.
- Continuous Integration and continuous deployment, CI/CD: In the process of continuous integration and deployment, a variety of tasks can be automated using Git providers such as GitHub, GitLab, etc. They connect to your VCS provider, track changes, and perform actions based on the type of activity. Actions include but are not limited to running unit tests and automated integration tests, deploying the application, generating packages for external use (Nuget, npm, ...), etc.
- Management summary generation: Management can get a thorough picture of how the project is doing thanks to version control. They know who is responsible for the modifications, what they are intended to accomplish when they are completed, and how the changes will affect the document's long-term objective. It helps management spot persistent issues that particular team members could bring on.
- Useful for Software testers: A VCS is not only a useful tool for developers, but it also makes up for an integral part of a QA engineer's daily work. Tasks can range from having a dedicated repository for test automation code, approving merge requests of developers to managing the CI/CD pipelines, including the automated tests in the setup for CI/CD, setting up automatic deployment to test environments, etc.

GitHub Terminologies:

GitHub repository: A repository is the most basic element of GitHub and is usually used to organize a single project. They're easiest to imagine as a project's folder. A repository contains all of the project files (including documentation) and stores each file's revision history. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything a project needs. Often, repositories include a README file, a file with information about the project. README files are written in the plain text Markdown language. GitHub lets the users add a README file at the same time they create a new repository. Repositories can have multiple collaborators and can be either public or private.

Branch: Branching allows to have different versions of a repository at one time.

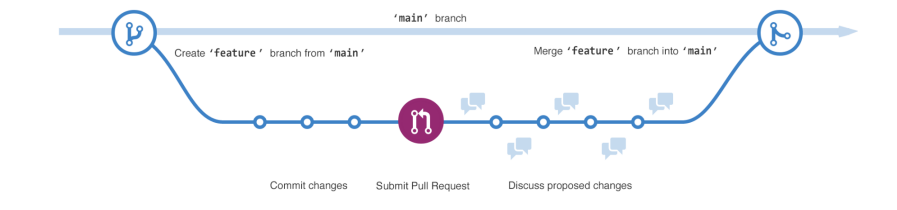
By default, a repository has one branch named main that is considered to be the definitive branch. We can create additional branches off of the main in that repository. We can use branches to have different versions of

a project at one time. This is helpful when we want to add new features to a project without changing the main source of code. The work done on different branches will not show up on the main branch until it is merged, which we will cover later in this guide.

While creating a branch off the main branch, we are 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 you were working on your branch, you could pull in those updates.

This diagram shows:

- The main branch
- A new branch called feature
- The journey that feature takes before it's merged into main



Commit:

Developers can make and save changes to the files in their repository. On GitHub, saved changes are called commits. When you make a commit to save your work, Git creates a unique ID (a.k.a. the "SHA" or "hash") that allows keeping a record of the specific changes committed along with who made them and when. Each commit has an associated commit message, which is a description explaining why a particular change was made. Commit messages to capture the history of the changes so that other contributors can understand what has been done and why.

Pull request:

Now that changes have been committed in a branch off of main, we can open a pull request. When we open a pull request, we're proposing the changes and requesting that someone review and pull in this contribution and merge them into their branch. Developers can even open pull requests in their own repository and merge them themselves. Pull requests show diffs, or differences, of the content from both branches. The changes, additions, and subtractions are shown in different colors.

Merge:

In this final step, the new branch is merged into the main branch. After merging the pull request, the changes on the new branch will be incorporated into main. Sometimes, a pull request may introduce changes to code that conflict with the existing code on main. If there are any conflicts, GitHub will alert about the conflicting code and prevent merging until the conflicts are resolved. You can make a commit that resolves the conflicts or use comments in the pull request to discuss the conflicts with team members.

3 Implementation

Here in this tutorial, you learn how to:

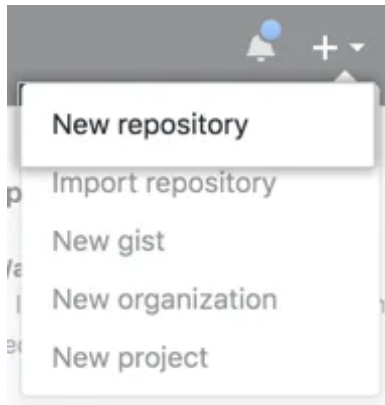
- Create an open source repository
- Start and manage a new branch
- Change a file and Commit changes
- Open and merge a pull request

First, Git must be installed on the computer.

Creating a repository:

Step 1: Create a GitHub account and sign in.

Step 2: In the upper-right corner of any page, use the drop-down menu, and select New repository.



Step 3: Type a short, memorable name for your repository. For example, "hello-world".

A screenshot of the 'Create a new repository' page on GitHub. The 'Repository name' field is highlighted with a box and contains the text 'hello-world' with a green checkmark to its right. The 'Owner' dropdown shows 'octocat'. Below the name field, there is a hint: 'Great repository names are short and memorable. Need inspiration? How about potential-eureka.' The 'Description (optional)' field is empty.

Step 4: Optionally, add a description of your repository. For example, "My first repository on GitHub."

A screenshot of the 'Create a new repository' page. The 'Repository name' field still contains 'hello-world'. The 'Description (optional)' field is now highlighted with a box and contains the text 'My first repository on GitHub'.

Step 5: Choose a repository visibility.

A screenshot showing the visibility selection options for the repository. The 'Description (optional)' field is at the top. Below it, a box highlights three options: 'Public' (selected with a blue radio button), 'Internal' (unselected), and 'Private' (unselected). Each option has a brief description of its permissions. At the bottom, there is a link that says 'Skip this step if you're importing an existing repository.'

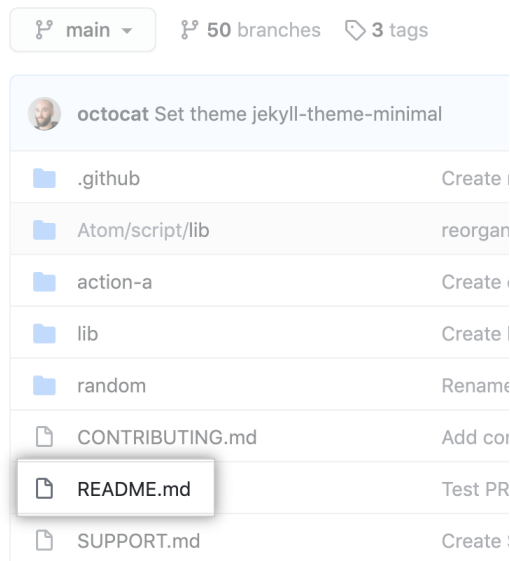
Step 6: Select Initialize this repository with a README.

Step 7: Click Create repository.

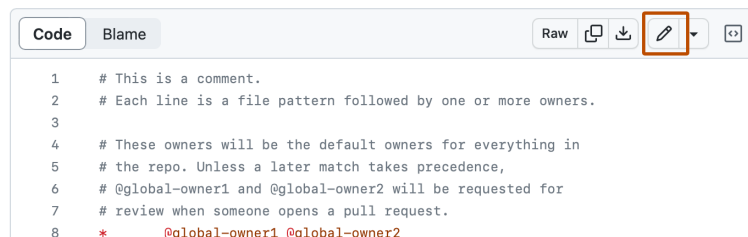
Committing changes: Changes in any file or code stored in the repository can be committed. When you created your new repository, you initialized it with a README file. README files are a great place to describe your project in more detail or add some documentation, such as how to install or use your project. The contents of your README file are automatically shown on the front page of your repository.

Let's commit a change to the README file.

Step 1: In your repository's list of files, click README.md.

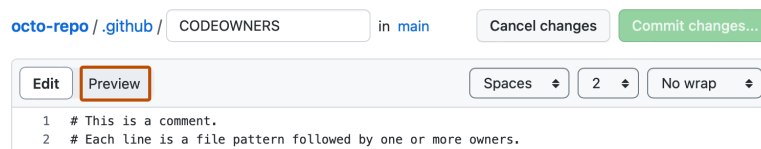


Step 2: In the upper right corner of the file view, click to open the file editor.

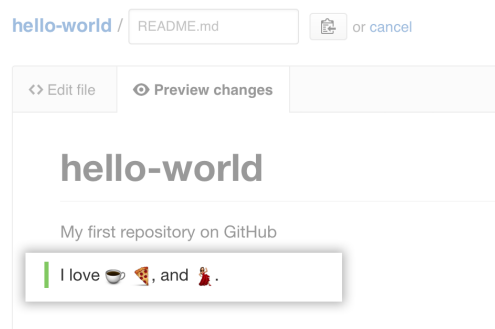


Step 3: Have some changes in the file. In the text box, type some information about yourself.

Step 4: Above the new content, click Preview.



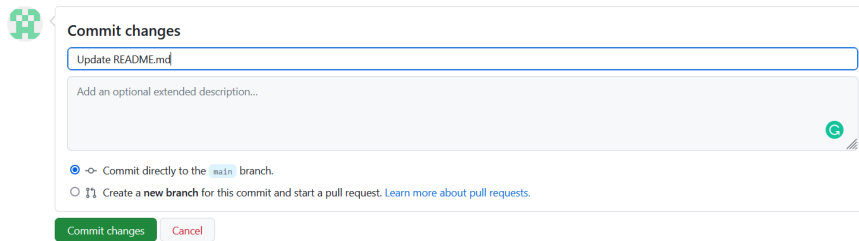
Step 5: Review the changes you made to the file. You will see the new content in green.



Step 6: Go to commit changes.

Step 7: In the "Commit message" field, type a short, meaningful commit message that describes the change

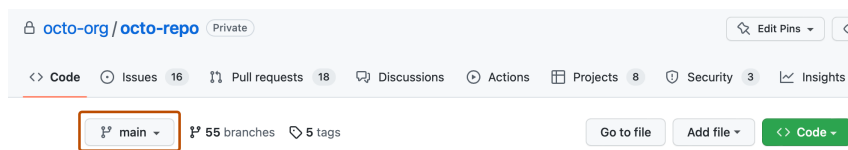
you made to the file. You can also add an optional extended description.

A screenshot of the GitHub 'Commit changes' dialog. It features a text input field containing 'Update README.md' and a larger text area below it with the placeholder 'Add an optional extended description...'. At the bottom, there are two radio button options: 'Commit directly to the main branch.' (which is selected) and 'Create a new branch for this commit and start a pull request. Learn more about pull requests.' Below these options are two buttons: 'Commit changes' in green and 'Cancel' in red.

Creating a branch:

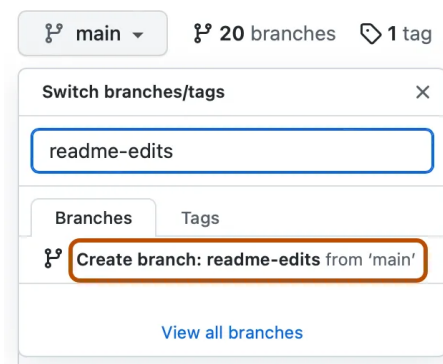
Step 1: Click the Code tab of your hello-world repository.

Step 2: Above the file list, click the dropdown menu that says main.



Step 3: Type a branch name, readme-edits, into the text box.

Step 4: Click Create branch: readme-edits from main.



Opening a pull request: When you created a new branch in the previous step, GitHub brought you to the code page for your new readme-edits branch, which is a copy of main. You can make and save changes to the files in your repository.

Now that you have changes in a branch off of main, you can open a pull request. As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.



Step 1: Click the Pull requests tab of your hello-world repository.

Step 2: Click New pull request

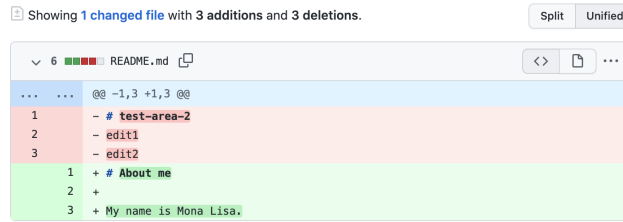
Step 3: In the Example Comparisons box, select the branch you made, readme-edits, to compare with main (the original).

Compare and review just about anything

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

Example comparisons	
 readme-edits	22 minutes ago
 main@{1day}...main	24 hours ago

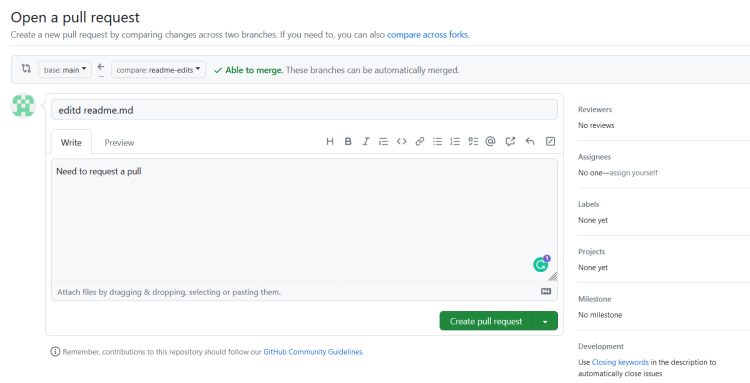
Step 4: Look over your changes in the diffs on the Compare page, make sure they're what you want to submit.



Step 5: Click Create pull request.

Step 6: Give your pull request a title and write a brief description of your changes.

Optionally, to the right of your title and description, click the next to Reviewers. Assignees, Labels, Projects, or Milestone to add any of these options to your pull request.



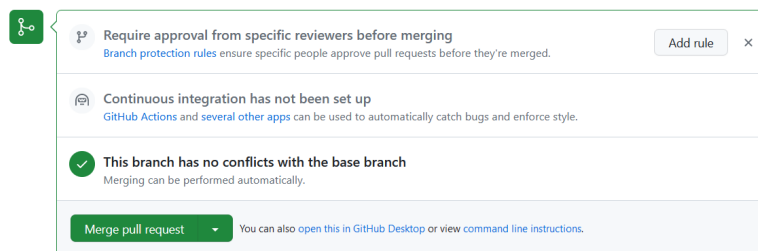
Step 7: Click Create pull request.

Merging your pull request:

In this final step, you will merge your readme-edits branch into the main branch. After you merge your pull request, the changes on your readme-edits branch will be incorporated into main.

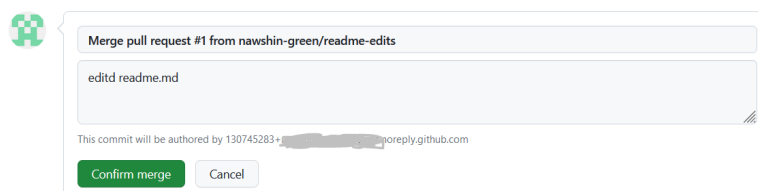
Sometimes, a pull request may introduce changes to code that conflict with the existing code on main. If there are any conflicts, GitHub will alert you about the conflicting code and prevent merging until the conflicts are resolved.

Step 1: At the bottom of the pull request, click Merge pull request to merge the changes into main.



Step 2: Click Confirm merge.

You will receive a message that the request was successfully merged and the request was closed.



Step 3: Click Delete branch. Now that your pull request is merged and your changes are on main, you can safely delete the readme-edits branch. If you want to make more changes to your project, you can always create a new branch and repeat this process.

Take a look at your GitHub profile and you'll see your work reflected on your contribution graph.

4 Discussion & Conclusion

After reading this tutorial on the version control system, GitHub, you would have understood how the VCS works. You will also be able to create a GitHub repository and integrate it into any project for better management and collaboration.

5 Lab Task (Please implement yourself and show the output to the instructor)

1. Create a GitHub repository for the given project.
2. Implement the branching, committing, pull request and merge options to track and manage the changes of the given project.

6 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.