Subject Name: **Source Code Management**

Subject Code: **CS121** Cluster: **Zeta** Department: **DCSE**



**Submitted By: Submitted To:**

Arsh Bedi Mr. Anuj Jain

2110992027

G5-A



**List of Tasks**

|  |  |  |
| --- | --- | --- |
| **S.**  **No** | **Task Title** | **Page No.** |
| 1 | Setting up of Git Client. |  |
| 2 | Setting up GitHub Account. |  |
| 3 | Generate logs. |  |
| 4 | Create and Visualize branches. |  |
| 5 | Git lifecycle description. |  |

Task 1

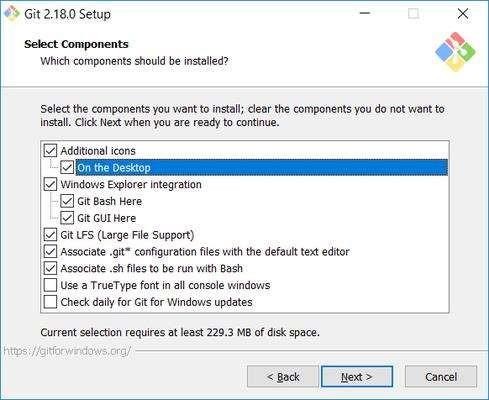
Installing and Configuring the Git client

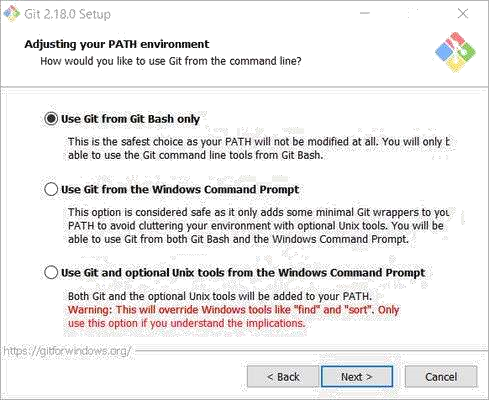
### Git installation

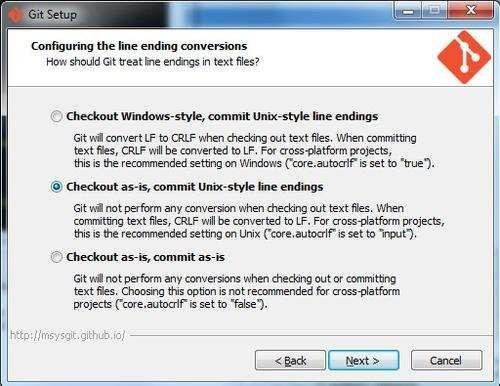
Download the Git installation program (Windows, Mac, or Linux) from [http://git-scm.com/downloads.](http://git-scm.com/downloads)

When running the installer, various screens appear (Windows screens shown). Generally, you can accept the default selections, ***except in the screens below where you do NOT want the default selections:***

In the **Select Components** screen, make sure **Windows Explorer Integration** is selected as shown:







Once Git is installed, there is some remaining custom configuration you must do. Follow the steps below:

* 1. From within File Explorer, right-click on any folder. A context menu appears containing the commands "**Git Bash here**" and "**Git GUI here**". These commands permit you to launch either Git client. For now, select **Git Bash here**.

# Task 2

This guide will walk you through setting up your GitHub account and getting started with GitHub's features for collaboration and community.

**Part 1: Configuring your GitHub account**

The first steps in starting with GitHub are to create an account, choose a product that fits your needs best, verify your email, set up two-factor authentication, and view your profile.

* 1. **Creating an account**

To sign up for an account on GitHub.com, navigate to <https://github.com/>and follow the prompts.

To keep your GitHub account secure you should use a strong and unique password. For more information, see "[Creating a strong password.](https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-strong-password)"

#### Choosing your GitHub product

You can choose GitHub Free or GitHub Pro to get access to different features for your personal account. You can upgrade at any time if you are unsure at first which product you want.

For more information on all of GitHub's plans, see "[GitHub's products.](https://docs.github.com/en/get-started/learning-about-github/githubs-products)"

#### Verifying your email address

To ensure you can use all the features in your GitHub plan, verify your email address after signing up for a new account. For more information, see "[Verifying your email](https://docs.github.com/en/github/getting-started-with-github/signing-up-for-github/verifying-your-email-address) [address.](https://docs.github.com/en/github/getting-started-with-github/signing-up-for-github/verifying-your-email-address)"

#### Configuring two-factor authentication

Two-factor authentication, or 2FA, is an extra layer of security used when logging into websites or apps. We strongly urge you to configure 2FA for the safety of your account. For more information, see "[About two-factor authentication.](https://docs.github.com/en/github/authenticating-to-github/securing-your-account-with-two-factor-authentication-2fa/about-two-factor-authentication)"

#### Viewing your GitHub profile and contribution graph

Your GitHub profile tells people the story of your work through the repositories and gists you've pinned, the organization memberships you've chosen to publicize, the contributions you've made, and the projects you've created. For more information, see ["About your profile"](https://docs.github.com/en/github/setting-up-and-managing-your-github-profile/customizing-your-profile/about-your-profile) and "[Viewing contributions on your profile.](https://docs.github.com/en/github/setting-up-and-managing-your-github-profile/managing-contribution-graphs-on-your-profile/viewing-contributions-on-your-profile)"

## Part 2: Using GitHub's tools and processes

To best use GitHub, you'll need to set up Git. Git is responsible for everything GitHubrelated that happens locally on your computer. To effectively collaborate on GitHub, you'll write in issues and pull requests using GitHub Flavored Markdown.

1. **Learning Git**

GitHub's collaborative approach to development depends on publishing commits from your local repository to GitHub for other people to view, fetch, and update using Git. For more information about Git, see the "[Git Handbook"](https://guides.github.com/introduction/git-handbook/) guide. For more information about how Git is used on GitHub, see "[GitHub flow.](https://docs.github.com/en/get-started/quickstart/github-flow)"

#### Setting up Git

If you plan to use Git locally on your computer, whether through the command line, an IDE or text editor, you will need to install and set up Git. For more information, see "[Set](https://docs.github.com/en/get-started/quickstart/set-up-git) [up Git.](https://docs.github.com/en/get-started/quickstart/set-up-git)"

If you prefer to use a visual interface, you can download and use GitHub Desktop. GitHub Desktop comes packaged with Git, so there is no need to install Git separately. For more information, see "[Getting started with GitHub Desktop.](https://docs.github.com/en/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop)"

Once you install Git, you can connect to GitHub repositories from your local computer, whether your own repository or another user's fork. When you connect to a repository on GitHub.com from Git, you'll need to authenticate with GitHub using either HTTPS or SSH. For more information, see "[About remote repositories.](https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories)"

#### Choosing how to interact with GitHub

Everyone has their own unique workflow for interacting with GitHub; the interfaces and methods you use depend on your preference and what works best for your needs.

For more information about how to authenticate to GitHub with each of these methods, see "[About authentication to GitHub.](https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/about-authentication-to-github)"

|  |  |
| --- | --- |
| **Method** | **Description Use cases** |
| Browse to GitHub.com | If you don't need to work with files locally, This method is useful if you GitHub lets you complete most Git-related want a visual interface and need actions directly in the browser, from to do quick, simple changes creating and forking repositories to editing that don't require working files and opening pull requests. locally. |

|  |  |
| --- | --- |
| GitHub Desktop | GitHub Desktop extends and simplifies This method is best if you need your GitHub.com workflow, using a visual or want to work with files interface instead of text commands on the locally, but prefer using a visual command line. For more information on interface to use Git and interact getting started with GitHub Desktop, see with GitHub. ["Getting](https://docs.github.com/en/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop) [started with GitHub Desktop.](https://docs.github.com/en/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop)" |
| IDE or text editor | You can set a default text editor, This is convenient if you are like [Atom](https://atom.io/) or [Visual Studio Code](https://code.visualstudio.com/) to open working with more complex and edit your files with Git, use extensions, files and projects and want and view the project structure. For more everything in one place, since information, see "[Associating text editors t](https://docs.github.com/en/github/using-git/associating-text-editors-with-git)ext editors or IDEs often allow [with Git.](https://docs.github.com/en/github/using-git/associating-text-editors-with-git)" you to directly access the  command line in the editor. |
| Command line, For the most granular control and This is most convenient if you with or customization of how you use Git and are already working from the without  interact with GitHub, you can use the command line, allowing you to  GitHub CLI command line. For more information on avoid switching context, or if using Git commands, see "[Git cheatsheet.](https://docs.github.com/en/github/getting-started-with-github/quickstart/git-cheatsheet)" you are more comfortable  using the command line.  GitHub CLI is a separate command-line tool you can install that brings pull requests, issues, GitHub Actions, and other GitHub features to your terminal, so you can do all  your work in one place. For | |

more information, see "[Getting started a](https://docs.github.com/en/github/extending-github/getting-started-with-the-api)utomate common tasks, back [with the API.](https://docs.github.com/en/github/extending-github/getting-started-with-the-api)" up your data, or create

|  |  |  |
| --- | --- | --- |
| **Method** | **Description**  more information, see "[GitHub CLI.](https://docs.github.com/en/github/getting-started-with-github/using-github/github-cli)" | **Use cases** |
| GitHub API | GitHub has a REST API and GraphQL API that you can use to interact with GitHub. | The GitHub API would be most helpful if you wanted to For |

integrations that extend GitHub.

#### Writing on GitHub

To make your communication clear and organized in issues and pull requests, you can use GitHub Flavored Markdown for formatting, which combines an easy-to-read, easyto- write syntax with some custom functionality. For more information, see "[About writing](https://docs.github.com/en/github/writing-on-github/about-writing-and-formatting-on-github) [and formatting on GitHub.](https://docs.github.com/en/github/writing-on-github/about-writing-and-formatting-on-github)"

You can learn GitHub Flavored Markdown with the "[Communicating using Markdown"](https://lab.github.com/githubtraining/communicating-using-markdown) course on GitHub Learning Lab.

#### Searching on GitHub

Our integrated search allows you to find what you are looking for among the many repositories, users and lines of code on GitHub. You can search globally across all of GitHub or limit your search to a particular repository or organization. For more information about the types of searches you can do on GitHub, see "[About searching on](https://docs.github.com/en/github/searching-for-information-on-github/getting-started-with-searching-on-github/about-searching-on-github) [GitHub.](https://docs.github.com/en/github/searching-for-information-on-github/getting-started-with-searching-on-github/about-searching-on-github)"

Our search syntax allows you to construct queries using qualifiers to specify what you want to search for. For more information on the search syntax to use in search, see ["Searching on GitHub.](https://docs.github.com/en/github/searching-for-information-on-github/searching-on-github)"

#### Managing files on GitHub

With GitHub, you can create, edit, move and delete files in your repository or any repository you have write access to. You can also track the history of changes in a file line by line. For more information, see "[Managing files on GitHub.](https://docs.github.com/en/github/managing-files-in-a-repository/managing-files-on-github)"

# Task 3

Package ‘logr’

Title Creates Log Files Version 1.2.9

Description Contains functions to help create log files. The

package aims to overcome the difficulty of the base R sink() command. The log\_print() function will print to both the console and the file log, without

interfering in other write operations.

License CC0 Encoding UTF-8

R topics documented:

logr . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . log\_close . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . log\_code . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. log\_open . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . log\_path . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

log\_print . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . log\_status . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Index

logr Creates log files

Description

The logr package contains functions to easily create log files.

Details

The logr package helps create log files for R scripts. The package provides easy logging, without the complexity of other logging systems. It is designed for analysts who simply want a written log of the their program execution. The package is designed as a wrapper to the base R sink() function.

How to use

There are only three logr functions:

* 1. log\_open
  2. log\_print
  3. log\_close

The log\_open() function initiates the log. The log\_print() function prints an object to the log. The log\_close() function closes the log. In normal situations, a user would place the call to log\_open at the top of the program, call log\_print() as needed in the program body, and call log\_close() once at the end of the program.

Logging may be controlled globally using the options "logr.on" and "logr.notes". Both options accept TRUE or FALSE values, and control log printing or log notes, respectively.

See function documentation for additional details.

# Task 4

How it works

A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process. You can think of them as a way to request a brand new working directory, staging area, and project history. New commits are recorded in the history for the current branch, which results in a fork in the history of the project.

# The git branch command lets you create, list, rename, and delete branches. It doesn’t let you switch between branches or put a forked history back together again. For this reason, git branch is tightly integrated with the [git checkout](https://www.atlassian.com/git/tutorials/using-branches/git-checkout) and [git merge](https://www.atlassian.com/git/tutorials/using-branches/git-merge) commands

How it works

A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process. You can think of them as a way to request a brand new working directory, staging area, and project history. New commits are recorded in the history for the current branch, which results in a fork in the history of the project.

The git branch command lets you create, list, rename, and delete branches. It doesn’t let you switch between branches or put a forked history back together again. For this reason, git branch is tightly integrated with the [git checkout](https://www.atlassian.com/git/tutorials/using-branches/git-checkout) and [git merge](https://www.atlassian.com/git/tutorials/using-branches/git-merge) commands.

Common Options

|  |
| --- |
| git branch |
| List all of the branches in your repository. This is synonymous with git branch --list. |
| git branch <branch> |
| Create a new branch called ＜branch＞. This does *not* check out the new branch. |
| git branch -d <branch> |
| Delete the specified branch. This is a “safe” operation in that Git  prevents you from deleting the branch if it has unmerged changes. |
| git branch -D <branch> |
| Force delete the specified branch, even if it has unmerged changes.  This is the command to use if you want to permanently throw away all of the commits associated with a particular line of development. |
| git branch -m <branch> |
| Rename the current branch to ＜branch＞. |
| git branch -a |

List all remote branches.

Creating Branches

It's important to understand that branches are just pointers to commits. When you create a branch, all Git needs to do is create a

new pointer, it doesn’t change the repository in any other way. If you

start with a repository that looks like this:

Then, you create a branch using the following command:

|  |
| --- |
| git branch crazy-experiment |
| The repository history remains unchanged. All you get is a new pointer to the current commit: |

Note that this only *creates* the new branch. To start adding commits to it, you need to select it with git checkout, and then use the standard git add and git commit commands.

Creating remote branches

So far these examples have all demonstrated local branch operations. The git branch command also works on remote branches. In order to operate on remote branches, a remote repo must first be configured and added to the local repo config.

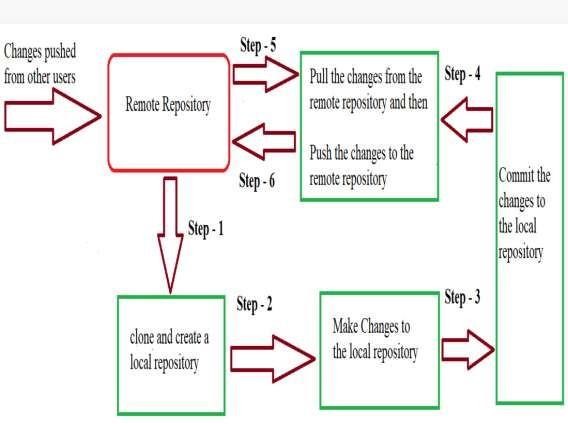
|  |
| --- |
| $ git remote add new-remote-  repo https://bitbucket.com/user/repo.git # Add remote repo to local repo config  $ git push <new-remote-repo> crazy-experiment~  # pushes the crazy-experiment branch to new-remote-repo |
| This command will push a copy of the local branch  crazyexperiment to the remote repo ＜remote＞. |

|  |  |
| --- | --- |
| Deleting Branches  Once you’ve finished working on a branch and have merged it into the main code base, you’re free to delete the branch without losing any history: | |
| git branch -d crazy-experiment |  |
| However, if the branch hasn’t been merged, the above command will  output an error message: |
| error: The branch 'crazy-experiment' is not fully merged. If you are sure you want to delete it, run 'git branch -D crazy-experiment'. |
| This protects you from losing access to that entire line of  development. If you really want to delete the branch (e.g., it’s a failed  experiment), you can use the capital -D flag: |
| git branch -D crazy-experiment |
| This deletes the branch regardless of its status and without warnings, so use it judiciously.  The previous commands will delete a local copy of a branch. The branch may still exist in remote repos. To delete a remote branch execute the following. |
| git push origin --delete crazy-experiment |
| Or |
| git push origin :crazy-experiment |

# Task 5

**Git – Life Cycle**

###### Git is used in our day-to-day work, we use git for keeping a track of our files, working in a collaboration with our team, to go back to our previous code versions if we face some error. Git helps us in many ways. Let us look at the Life Cycle that git has and understand more about its life cycle. Let us see some of the basic steps that we follow while working with Git –



###### When a directory is made a git repository, there are mainly 3 states which make the essence of Git Version Control System. The three states are –

* Working Directory

###### Staging Area

* Git Directory

Let us understand in detail about each state.

1. **Working Directory**

###### Whenever we want to initialize our local project directory to make it a git repository, we use the ***git init*** command. After this command, git becomes aware of the files in the project although it doesn’t track the files yet. The files are further tracked in the staging area.

##### 2.Staging Area

###### Now, to track the different versions of our files we use the command ***git add***. We can term a staging area as a place where different versions of our files are stored.

##### 3.Git Directory

###### Now since we have all the files that are to be tracked and are ready in the staging area, we are ready to commit our files using the ***git commit*** command. Commit helps us in keeping the track of the metadata of the files in our staging area. WeCommit also stores the name of the author who did the commit, files that are committed, and the date at which they are committed along with the commit message. *git commit .*

Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Zeta** Department: **DCSE**



**Submitted By: Submitted To:**

Arsh Bedi Mr. Anuj Jain

2110992027

G-27



**List of Tasks**

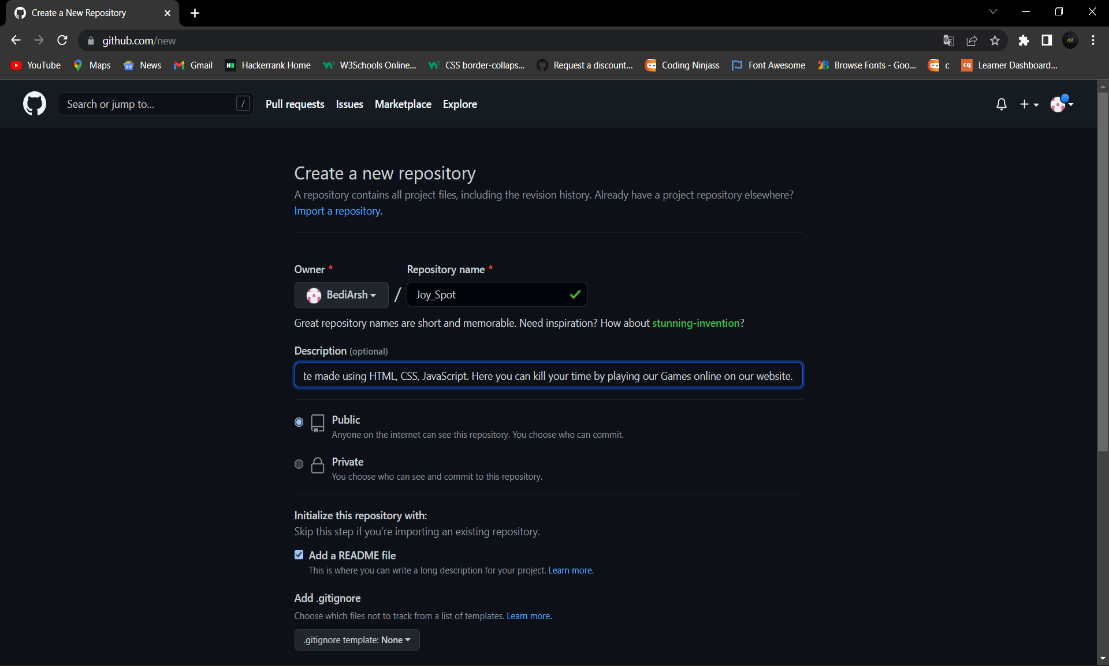
|  |  |  |
| --- | --- | --- |
| **S.**  **No** | **Task Title** | **Page No.** |
| 1 | Add Collaborators on GitHub Repo. | - |
| 2 | Fort And Commit. | - |
| 3 | Merge and resolve conflicts. | - |
| 4 | Reset and Revert. | - |
|  |  |  |

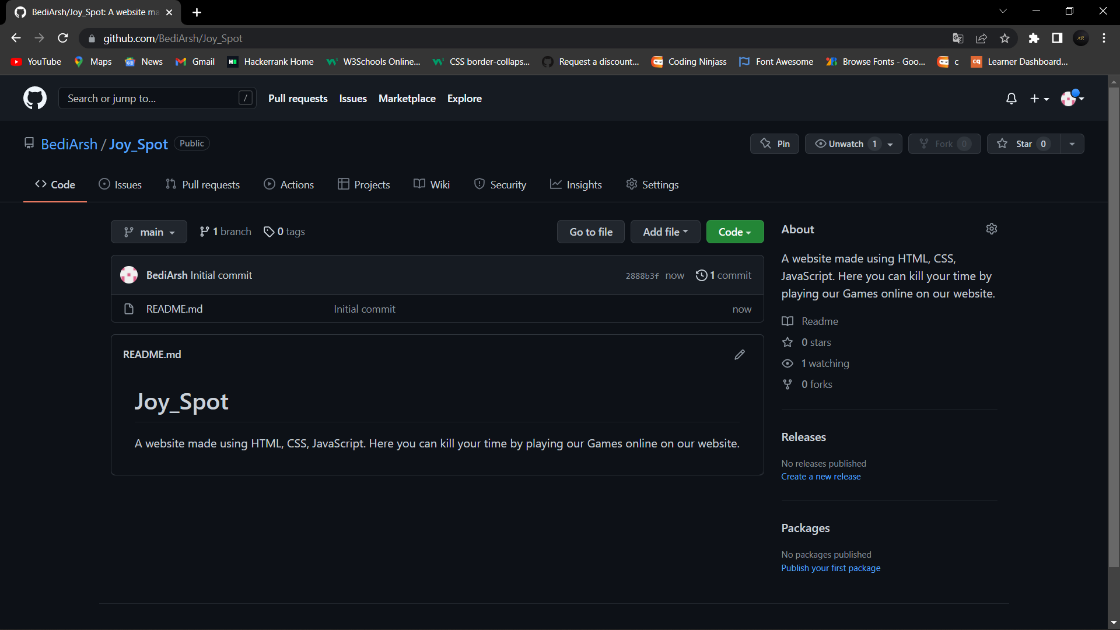
Task-1

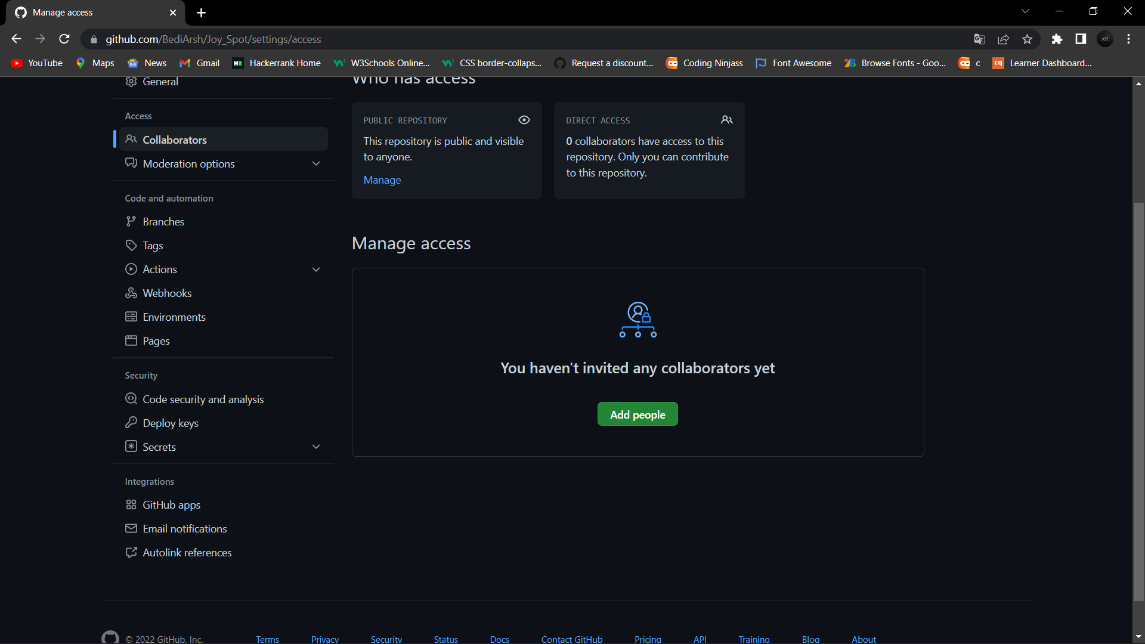
ADDING COLLABORATORS ON GITHUB REPO.

### CREATING A REPOSITORY

1. Click the new repository button in the top-right. You'll have an option there to initialize the repository with a README file, but I don't.
2. Click the “Create repository” button.







### ADDING A COLLABORATOR

### On GitHub, click the settings button on the right, select Manage access, click Invite a collaborator, and then enter your partner's username.

# 

# 

# 

# 

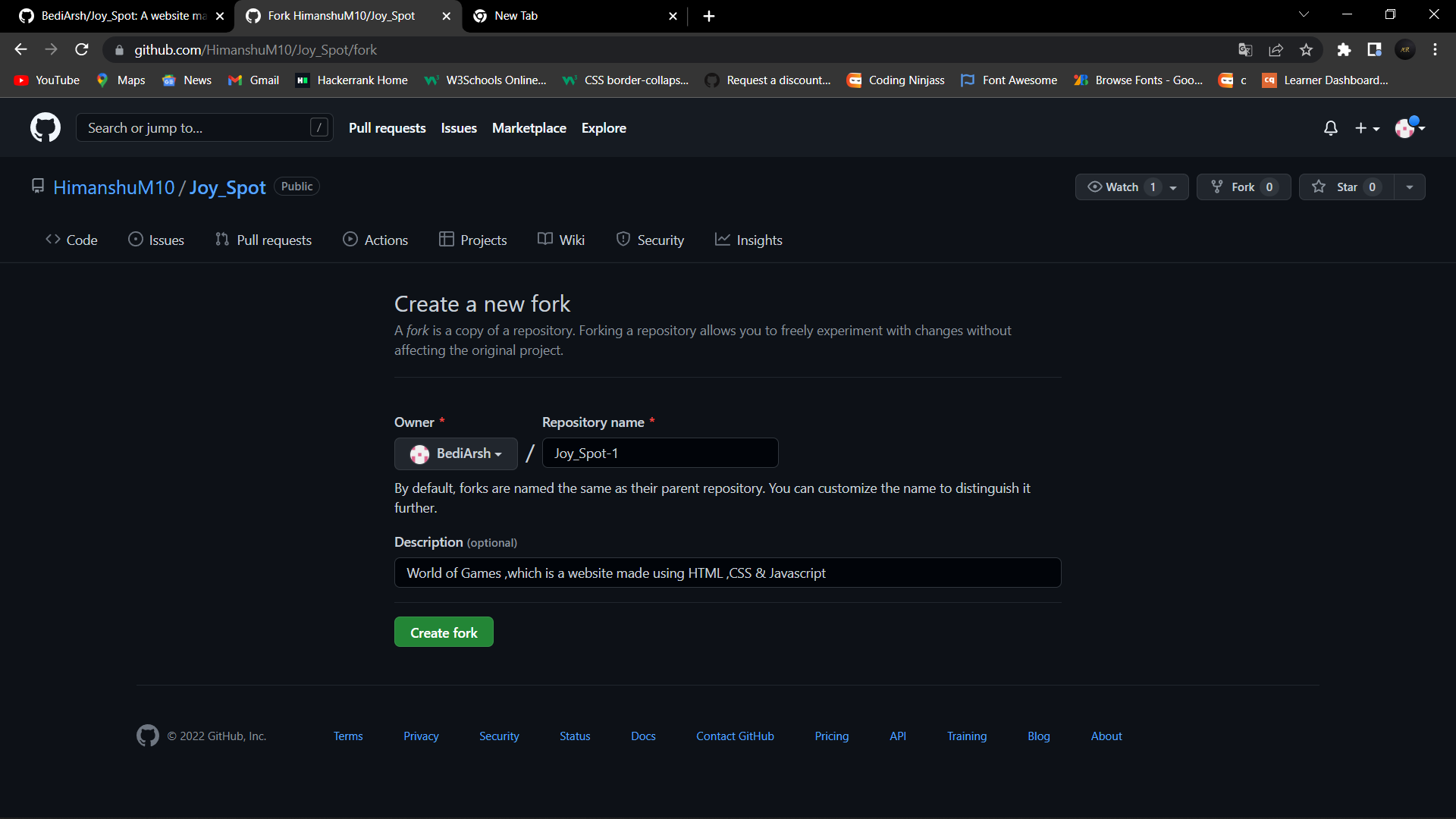
# Collaborator is added to my REPO.

# Task-2

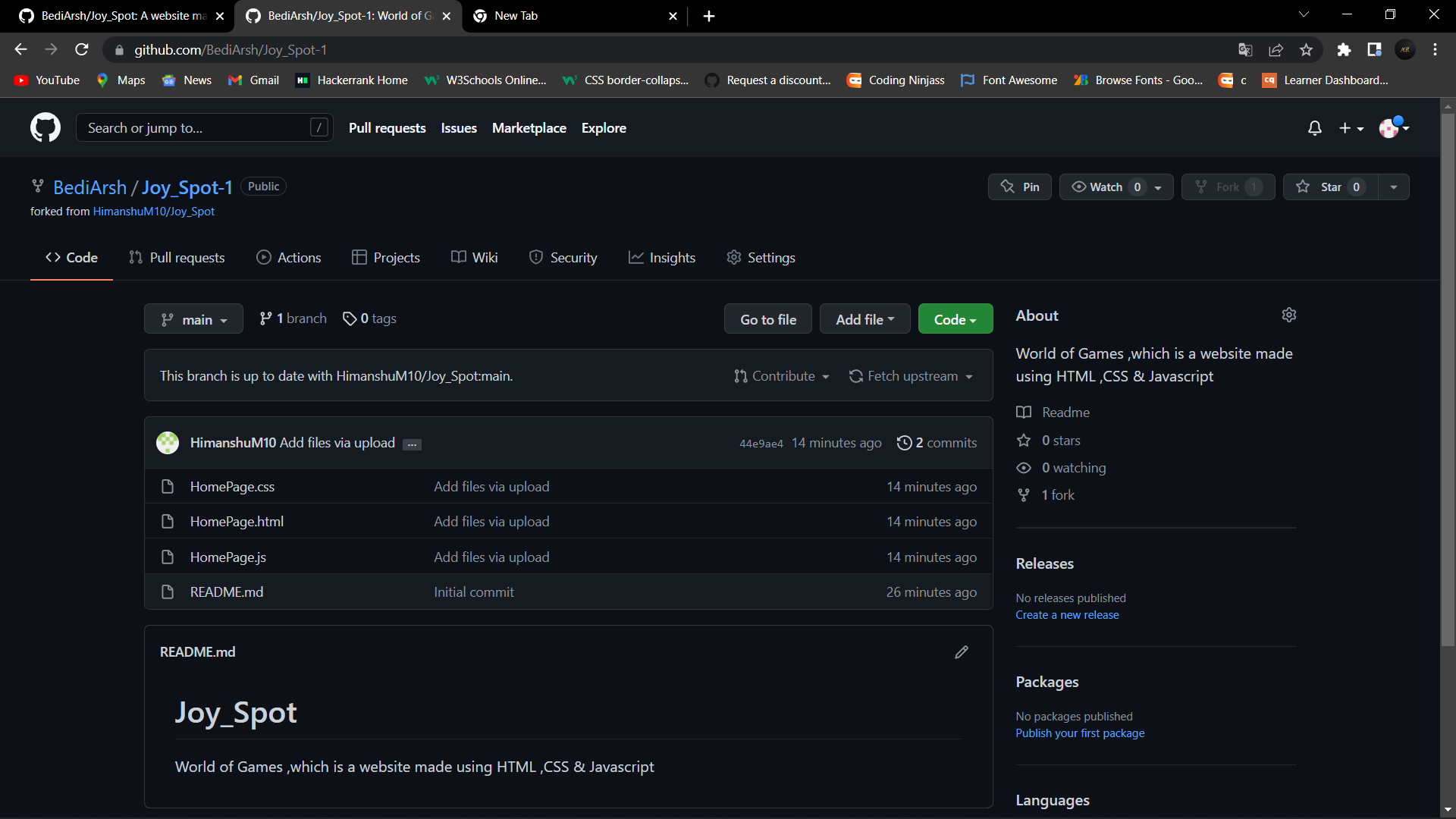
FORK AND COMMIT

* Fork a GitHub repository.
* Clone the forked repository to your local system.
* Add a Git remote for the original repository.
* Create a feature branch in which to place your changes.
* Make your changes to the new branch.
* Commit the changes to the branch.
* Push the branch to GitHub.

CREATING A FORK REPO.



COMMITING CHANGES IN FORK REPO.



# Task 3

Merging and Resolving conflicts.

## Merge conflicts

So things can go wrong, which usually starts with a **merge conflict**, due to both collaborators making incompatible changes to a file. While the error messages from merge conflicts can be daunting, getting things back to a normal state can be straightforward once you’ve got an idea where the problem lies.

A merge conflict occurs when both the owner and collaborator change the same lines in the same file without first pulling the changes that the other has made. This is most easily avoided by good communication about who is working on various sections of each file, and trying to avoid overlaps. But sometimes it happens, and git is there to warn you about potential problems. And git will not allow you to overwrite one person’s changes to a file with another’s changes to the same file if they were based on the same version.

## How to resolve a conflict

### Abort, abort, abort…

Sometimes you just made a mistake. When you get a merge conflict, the repository is

placed in a ‘Merging’ state until you resolve it. There’s a commandline command to abort

abort doing the merge altogether:

git merge –abort

**To see the conflict**, you can go back to the changes section. Here, you’ll see the files that contain conflicts. In this case, it is the README.md file.

### Resolving conflicts

The easiest way to resolve a conflict is to change the file on your computer. If you open README.md now, you’ll see lines that say this:

<<<<<< HEAD Hello world! This is my third Github repo!====== Hello world! This is my second Github repo! >>>&gt;>> snt2h1s3n4tnthd9au8d3324

The code between <<<;<<<; HEAD and ======= is the code in our local repository (our code).

The code between ====== and >&gt;>>>> is the code from the remote repository (their code).

These two lines of code are conflicting. We need to choose between the “second Github repo” or the “third Github repo”.

To fix the conflict, you choose the correct line of code. Then you delete everything else.

In this case, let’s say “third” is the correct version. What you’ll do is delete everything else that’s incorrect.

Hello world! This is my third Github repo!

Committing the merge

When you head back to Fork, you’ll see that the changes are updated. In our case, the change is the one on our local, so we don’t see any files that need to be staged.

If the change is different, you will need to stage the files.

After staging, you need to commit the merge.

If you look at the commit message area, you’ll see that Fork has filled in a commit message for you automatically. You can use the commit message directly.

Click on commit to commit the changes.

# Task 4

RESET AND REVERT

GIT RESET

The git reset command allows you to RESET your current head to a specified state. You can reset the state of specific files as well as an entire branch. This is useful if you haven't pushed your commit up to GitHub or another remote repository yet.

### Reset a file or set of files

The following command lets you selectively choose chunks of content and revert or unstage it.

git reset (--patch | -p) [tree-ish] [--] [paths]

### Unstage a file

If you moved a file into the staging area with git add, but no longer want it to be part of a commit, you can use git reset to unstage that file:

git reset HEAD FILE-TO-UNSTAGE

The changes you made will still be in the file, this command just removes that file from your staging area.

### Reset a branch to a prior commit

The following command resets your current branch's HEAD to the given COMMIT and updates the index. It basically rewinds the state of your branch, then all commits you make going forward write over anything that came after the reset point. If you omit the MODE, it defaults to --mixed:

git reset MODE COMMIT

The options for MODE are:

* --soft: does not reset the index file or working tree, but resets HEAD to commit. Changes all files to "Changes to be commited"
* --mixed: resets the index but not the working tree and reports what has not been updated
* --hard: resets the index and working tree. Any changes to tracked files in the working tree since commit are discarded
* --merge: resets the index and updates the files in the working tree that are different between commitand HEAD, but keeps those which are different between the index and working tree
* --keep: resets index entries and updates files in the working tree that are different between commitand HEAD. If a file that is different between commit and HEAD has local changes, the reset is aborted

GIT REVERT

## Git Revert

Both the git revert and git reset commands undo previous commits. But if you've already pushed your commit to a remote repository, it is recommended that you do not use git reset since it rewrites the history of commits. This can make working on a repository with other developers and maintaining a consistent history of commits very difficult.

Instead, it is better to use git revert, which undoes the changes made by a previous commit by creating an entirely new commit, all without altering the history of commits.

### Revert a commit or set of commits

The following command lets you revert changes from a previous commit or commits and create a new commit.

git revert [--[no-]edit] [-n] [-m parent-number] [-s] [-S[<keyid>]] <commit>…

git revert --continue

git revert --quit

git revert --abort

### Common options:

-e

--edit

* This is the default option and doesn't need to be explicitly set. It opens your system's default text editor and lets you edit the new commit message before commit the revert.
* This option does the opposite of -e, and git revert will not open the text editor.
* This option prevents git revert from undoing a previous commit and creating a new one. Rather than creating a new commit, -n will undo the changes from the previous commit and add them to the Staging Index and Working Directory.

--no-edit

-n

-no-commit

A Project report on

# “JOYSPOT”

with

# Source Code Management

(CS181)



Submitted by

|  |  |
| --- | --- |
| Himanshu | 2110992040 |
| Arsh Bedi | 2110992027 |
| Vansh Goel | 2110992047 |
| Bhaskar | 2110992056 |

# Department of Computer Science & Engineering

Chitkara University Institute of Engineering and Technology, Punjab

### Jan- June (2021-22)

|  |  |  |  |
| --- | --- | --- | --- |
| Institute/School Name | **Chitkara University Institute of Engineering and Technology** | | |
| Department Name | **Department of Computer Science & Engineering** | | |
| Programme Name | **Bachelor of Engineering (B.E.), Computer Science & Engineering** | | |
| Course Name | **Source Code Management** | Session | **2021-22** |
| Course Code | **CS181** | Semester/Batc h | **2nd/2021** |
| Vertical Name | **Zeta** | Group No | G27 |
| Course Coordinator | **Dr. Neeraj Singla** | | |
| Faculty Name | **Dr. Sachendra Singh Chauhan** | | |

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Title** | **Page No.** |
| 1 | Version control with Git | 1-4 |
| 2 | Problem Statement | 4-10 |
| 3 | Objective | 11 |
| 4 | Resources Requirements – Frontend | 12-13 |
| 5 | Concepts and commands | 14-26 |
| 6 | Workflow and Discussion | 27-30 |
| 7 | Reference | 31 |

# Version Control With Git

A **Version Control System (VCS)** is a tool that helps software developers keep track of how their software development projects desktop applications, websites, mobile apps, etc - change over time.

Each snapshot or state of the files and folders in a codebase at a given time can be called a "version." Version control systems were created to allow developers a convenient way to create, manage, and share those versions. It allows them to have *control* over managing the versions of their code as it evolves over time.

Version control systems also enable collaboration within a team of software developers, without losing or overwriting anyone's work. After a developer makes a set of code changes to one or more files, they tell the version control system to save a representation of those changes.

A version control system can also be referred to as a source control system, source code management, version control software, version control tools, or other combinations of these terms.

**Benefits of the Version Control System**

The Version Control System is very helpful and beneficial in software development; developing software without using version control is unsafe. It provides backups for uncertainty. Version control systems offer a speedy interface to developers. It also allows software teams to preserve efficiency and agility according to the team scales to include more developers.

Some key benefits of having a version control system are as follows.

* Complete change history of the file
* Simultaneously working
* Branching and merging

### History of VCS

There are 3 types of VCS:

1. **Local Version Control System:**

In a local version control system, files are simply copied into a separate directory locally. Versions of the same file are stored so as to allow the easy retrieval of any particular version at any point in time. This system is commonly used for small personal projects or files as it provides the facility of versioning your project in an easy manner locally.

###### Advantages:

* + Easy to set up
  + No internet needed
  + Cheap to run **Disadvantages:**
  + Error prone
  + Unsafe (stored locally)
  + Not suitable for team projects
  + As data is stored in local machine. If the local machine crashes, it would not be possible to retrieve the files, and all the information will be lost.

###### Centralized Version Control System:

In the Centralized Version Control Systems, there will be a single central server that contains all the files related to the project, and many collaborators checkout files from this single server (you will only have a working copy).

The problem with the Centralized Version Control Systems is if the central server crashes, almost everything related to the project will be lost.

###### Advantages:

* + Reasonably easy to set up
  + Various options (proprietary and open source)
  + Allows for file sharing amongst team members
  + Project is stored on a more reliable server (possibly cloud)
  + Admin can control the use and structure of the repository

###### DISADVANTAGES

* + Single point of failure (if server fails then changes will not be available)
  + File conflicts due to updates from different people

###### Distributed Version Control system:

In a distributed version control system, there will be one or more servers and many collaborators similar to the centralized system. But the difference is, not only do they check out the latest version, but each collaborator will have an exact copy of the main repository on their local machines.

Each user has their own repository and a working copy. This is very useful because even if the server crashes we would not lose everything as several copies are residing in several other computers.

###### ADVANTAGES

* + Reliable (everyone has a copy of all versions)
  + Allows for file share amongst team members
  + Various Options available **Disadvantages:**
  + More complex to use/set up
  + Heavy on Local Storage
* **What is Git?**

**Git** is an **open-source distributed version control system**. It is designed to handle minor to major projects with high speed and efficiency. It is developed to coordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.

Git was created by **Linus Torvalds** in **2005** to develop Linux Kernel. It is also used as an important distributed version-control tool for **the DevOps**.

#### Features of Git

Some remarkable features of Git are as follows:

* Open Source

Git is an **open-source tool**. It is released under the **GPL** (General Public License) license.

* Scalable

Git is **scalable**, which means when the number of users increases, the Git can easily handle such situations.

* Distributed

One of Git's great features is that it is **distributed**. Distributed means that instead of switching the project to another machine, we can create a "clone" of the entire repository. Also, instead of just having one central repository that you send changes to, every user has their own repository that contains the entire commit history of the project. We do not need to connect to the remote repository; the change is just stored on our local repository. If necessary, we can push these changes to a remote repository.

* Security

Git is secure. It uses the **SHA1 (Secure Hash Function)** to name and identify objects within its repository. Files and commits are checked and retrieved by its checksum at the time of checkout. It stores its history in such a way that the ID of particular commits depends upon the complete development history leading up to that commit. Once it is published, one cannot make changes to its old version.

* Speed

Git is very **fast**, so it can complete all the tasks in a while. Most of the git operations are done on the local repository, so it provides a **huge speed**. Also, a centralized version control system continually communicates with a server somewhere.

Performance tests conducted by Mozilla showed that it was **extremely fast compared to other VCSs**. Fetching version history from a locally

stored repository is much faster than fetching it from the remote server. The **core part of Git** is **written in C**, which **ignores** runtime overheads associated with other high-level languages.

Git was developed to work on the Linux kernel; therefore, it is

**capable** enough to **handle large repositories** effectively. From the beginning, **speed** and **performance** have been Git's primary goals.

* + Supports non-linear development

Git supports **seamless branching and merging**, which helps in visualizing and navigating a non-linear development. A branch in Git represents a single commit. We can construct the full branch structure with the help of its parental commit.

* + Branching and Merging

**Branching and merging** are the **great features** of Git, which makes it different from the other SCM tools. Git allows the **creation of multiple branches** without affecting each other. We can perform tasks like **creation, deletion,** and **merging** on branches, and these tasks take a few seconds only. Below are some features that can be achieved by branching:

* + We can **create a separate branch** for a new module of the project, commit and delete it whenever we want.
  + We can have a **production branch**, which always has what goes into production and can be merged for testing in the test branch.
  + We can create a **demo branch** for the experiment and check if it is working. We can also remove it if needed.
  + The core benefit of branching is if we want to push something to a remote repository, we do not have to push all of our branches. We can select a few of our branches, or all of them together.
  + Data Assurance

The Git data model ensures the **cryptographic integrity** of every unit of our project. It provides a **unique commit ID** to every commit through a **SHA algorithm**. We can **retrieve** and **update** the commit by commit ID. Most of the centralized version control systems do not provide such integrity by default.

* + Staging Area

The **Staging area** is also a **unique functionality** of Git. It can be

considered as a **preview of our next commit**, moreover, an **intermediate area** where commits can be formatted and reviewed before completion.

When you make a commit, Git takes changes that are in the staging area and make them as a new commit. We are allowed to add and remove changes from the staging area. The staging area can be considered as a place where Git stores the changes.

However, Git doesn't have a dedicated staging directory where it can store some objects representing file changes (blobs). Instead of this, it uses a file called index.

Another feature of Git that makes it apart from other SCM tools is that **it is possible to quickly stage some of our files and commit them without committing other modified files in our working directory.**

* Maintain the clean history

Git facilitates with Git Rebase; It is one of the most helpful features of Git. It fetches the latest commits from the master branch and puts our code on top of that. Thus, it maintains a clean history of the project.

# Problem Statement

Let’s take a scenario to know the importance of git and to know the real power and the importance of git in everyday life of a coder…

Just Imagine you are working on a website. You completed the first version of the project. You decided to modify it. You made a copy of your project folder and started making some changes without risking the integrity of your first version of the project. You made your second version of the project. Similarly, you made many versions of your project just to make it better and better.

Now the problem with this is or making duplicate folders of your project is not an optimized approach. We have to keep in mind the space and time complexities. But with git it becomes simpler to manage space and time as github is like a cloud where you will upload your data

If we are going to host our Education website Website over. So, we don’t need to make copies of the project and imagine we made a mistake and we want to go to the previous versions of the project. How will we roll back to a previous version?

# Objective

1. **To perform collaborations:** Git keeps track of changes to files and allows multiple users to coordinate updates to those files.
2. **To Track Histories:** Git is used to track changes in the source code.
3. **To enact distributed development:** Git enables the developers to manage the changes offline and allows you to branch and merge whenever required, giving them full control over the local code base.
4. **To perform branching:** Git allows you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository.
5. **To Speed up Tasks:** Git tries to minimize latency, which means good perceived performance. Which means it will be easier and faster to enact changes.

## Resources Requirements – Frontend

HTML stands for HyperText Markup Language. It is a relatively simple language that allows developers to create the basic structure of a website. Even the most complex websites have HTML at their core. It’s also the second-most-used programming language by developers, according to a recent Stack Overflow survey.

You may be asking yourself why HTML is called a “markup language.” The reason is that instead of using a programming language to perform the desired functions, HTML (like other markup languages) uses tags to annotate, or “mark up,” different types of content on a web page and identify the purposes they each serve to the page’s overall design. You likely see snippets of HTML more than you even realize. Have you ever noticed text at the bottom of a printed-out email that reads something like “ ”? That’s HTML. A markup language also helps web developers avoid formatting every instance of an item category separately (e.g., bolding the headlines on a website), which saves time and avoids errors.

HTML uses “elements,” or tags, to denote things like the beginning of a paragraph, the bolding of a font, or the addition of a photo caption. In this way, it controls how a webpage looks, how the text is separated and formatted, and what the user sees. For people who have never used programming languages before, HTML is an excellent place to start.

CSS

If HTML represents the building blocks of a website, CSS is a way to shape and enhance those blocks. CSS is a style sheet language used to specify the way different parts of a webpage appear to users. In other words, it’s a way to add some style and additional formatting to what you’ve already built with HTML.

For example, perhaps you’ve used HTML to add header text, and now you want that header to have a more pleasant font, a background color, or other formatting elements that make it more sleek, professional, and stylish. That’s where CSS comes in. CSS also helps websites adapt to different device types and screen sizes so that your pages render equally well onsmartphones, tablets, or desktop computers.

To understand the difference between HTML and CSS, it’s important to understand their histories. When HTML was invented in 1990, it was only designed to inform a document’s structural content (e.g., separating headlines from body text). However, when stylistic elements like fonts and colors were developed, HTML wasn’t able to adapt. To solve this issue, CSS was invented as a set of rules that can assign properties to HTML elements, building off of the existing markup language to create a more complex webpage.

JavaScript -

JavaScript is the most complex of the three front end languages discussed in this article, building on top of both HTML and CSS. If you’re trying to compare the languages, think of it like this: While HTML creates the basic structure for a website, CSS adds style to that structure, and JavaScript takes all of that work and makes it interactive and more functionally complex.

A classic example of how JavaScript works is the menu button that you’re used to seeing on the top corner of most websites. You know the one — the three stacked lines that show a list of website sections you can visit when clicked.

These buttons and their functionality are all present thanks to JavaScript. It can also help you develop keyboard shortcuts or change the color of a button when a cursor hovers over it.

JavaScript is crucial to all web development. It’s supported by all of the modern web browsers, and it is used on almost every site on the web. According to a recent Stack Overflow survey, JavaScript is the most commonly used programming language by developers around the world, with 67.7 percent of developers putting it to use in their work. So, if you’re interested in learning web development — whether professionally or even just as a hobby — you’d be smart to learn JavaScript.

**ExperimentNo.01**

**Aim: Create a distributed Repository and add members in project team**

**Distributed Repository:**

Now let’s discuss how to work with the distributed approach. While in centralized approach, everyone on team cloned the single central repository, here in distributed approach everyone clones *‘a copy of the source*

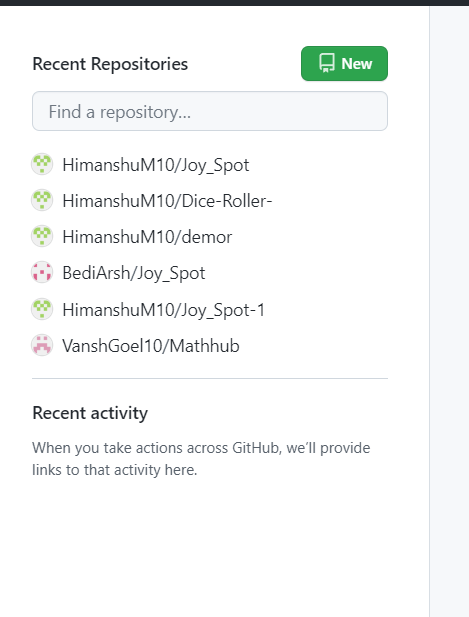
*repository’* also known as *‘forking’.*

In this approach, all the collaborators will ‘fork’ the source repo. Upon forking ever member gets their own copy of the source repository in their GitHub account. Collaborators can then go ahead and clone their ‘forked repository’ on their local machines.

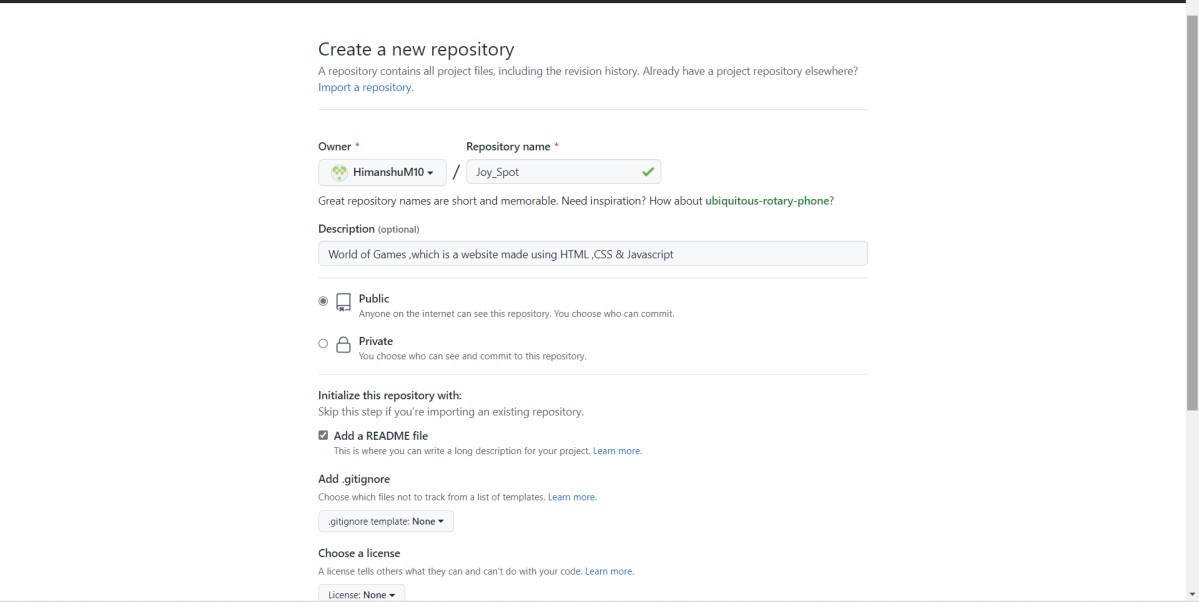
**Steps of creating Distributed Repository:**

we will create a new repository and invite a collaborator to our repository by sending an invitation. A detailed procedure on how to invite a collaborator to your [Github](https://recodehive.com/github-tutorials/) repository is mentioned below.

1. Went to Git hub and created new Repository, Click on new sign on top left side .

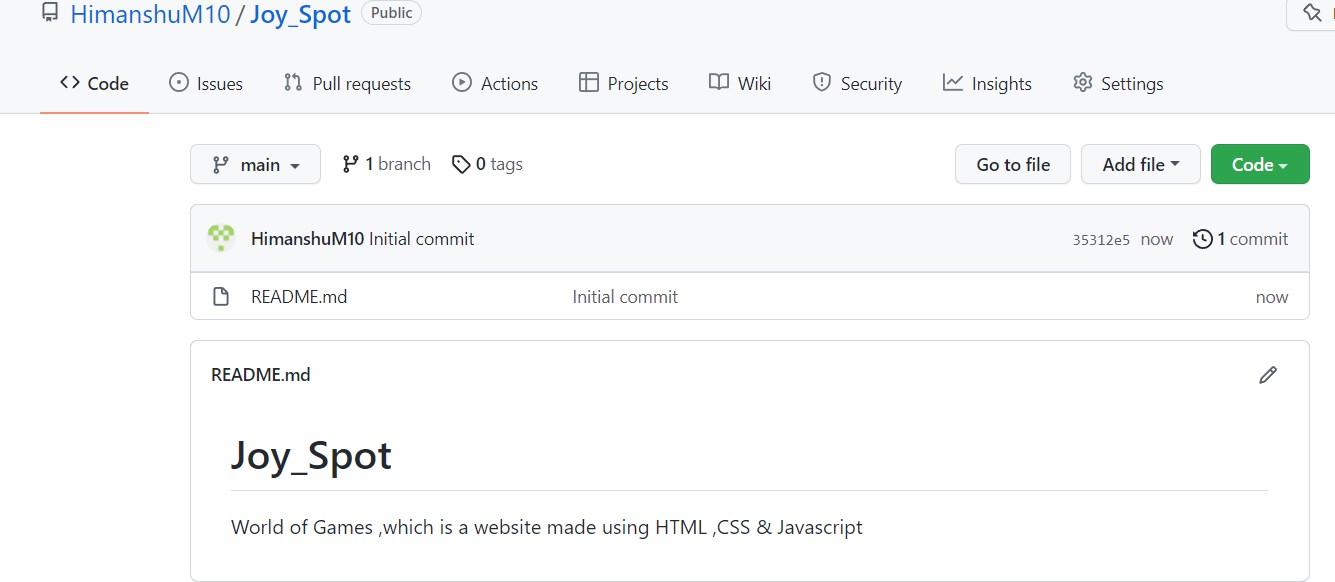


1. **Specify the Name of the Project,** make It public or primate, check on the readme file. Then click on Create repository. You can also see by default Brach name is main If you want you can change it. **(optional)**

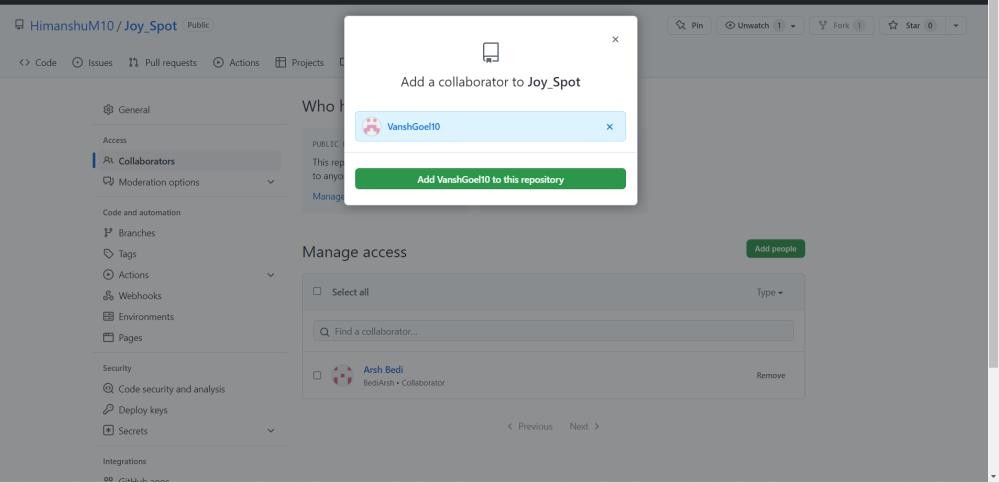


1. Now you can see the file created on that name with a default readme file. You can add content to the readme file. Now the next step is to invite the collaborators to the repository. currently, you have your repository ready to collobrate. Link to Git main Repository:

[HimanshuM10/Joy\_Spot: World of Games ,which is a website made using HTML ,CSS & Javascript](https://github.com/HimanshuM10/Joy_Spot) [(github.com)](https://github.com/HimanshuM10/Joy_Spot)

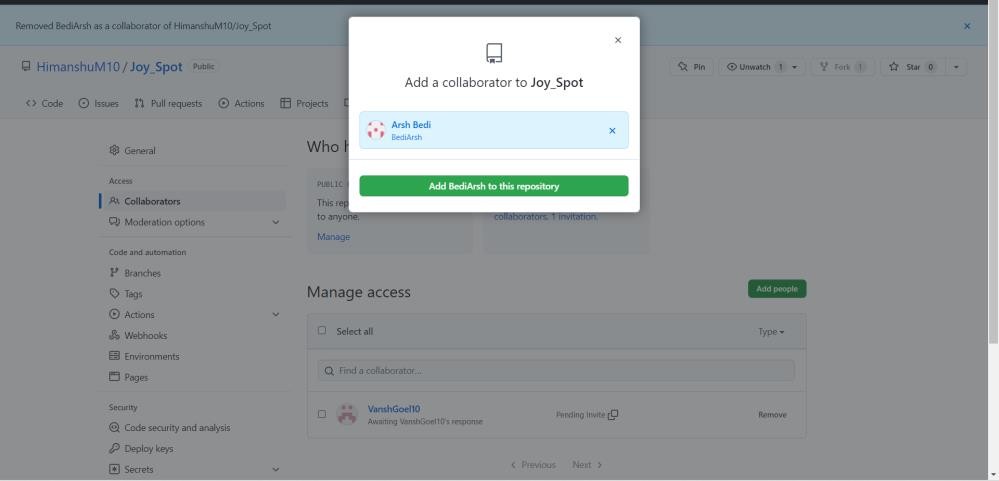


1. Now click on **settings,** you should be the one who created the repo to do this work, because only the author can sent the invite to others

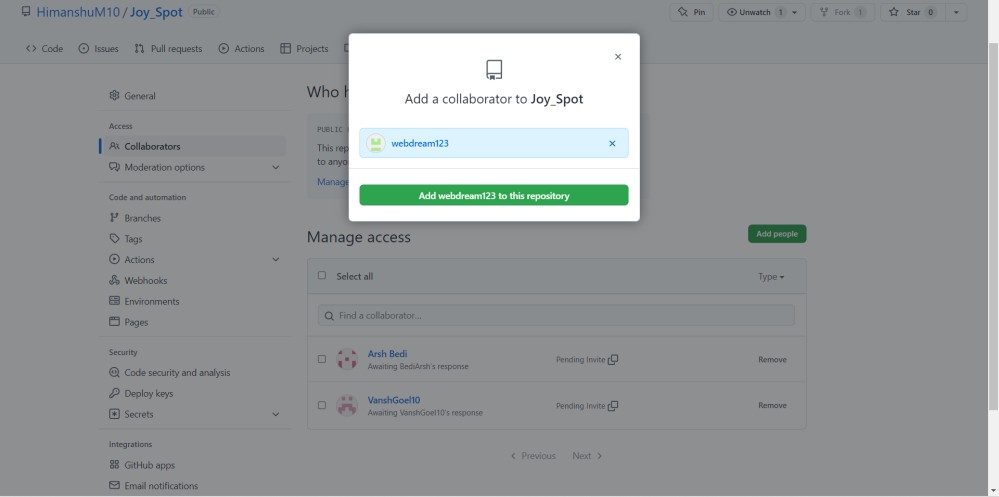


1. Click on the invited collaborator and enter the email ID of people you want to add to this repo. Email will go to them to accept the request.

Once it accepted they can push the changes to main branch.



1. The person will be getting an email invitation like this for Github collaboration, Click on the Email View and accept invitation.



1. You will be redirected to Github Windows there click on Accept Invitation.

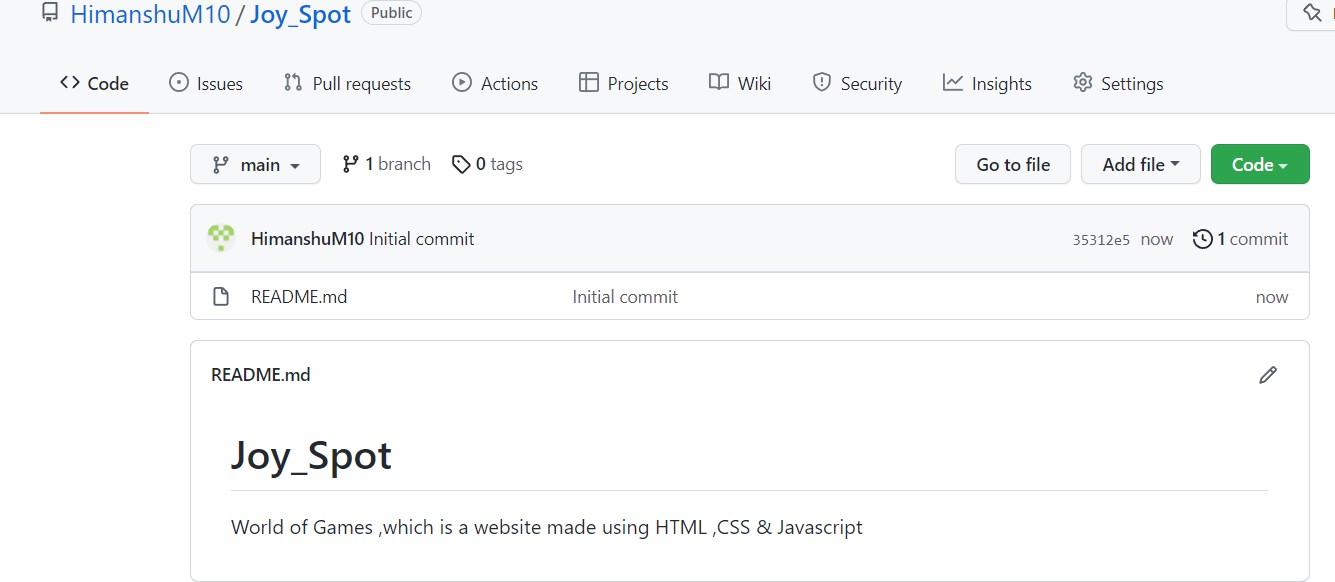
**Now all the collaborators will** [*‘***fork’**](https://help.github.com/en/enterprise/2.13/user/articles/fork-a-repo) **the source repo.**

Most commonly, forks are used to either propose changes to someone else's project to which you do not have write access, or to use someone else's project as a starting point for your own idea. You can fork a repository to create a

copy of the repository and make changes without affecting the upstream repository

1. On GitHub.com, navigate to the HimanshuM10/Joy\_Spot repository.
2. In the top-right corner of the page, click **Fork**.

We can make my changes here and then make a Pull Request to the maintainers of the project. Now it is in their



hand if they will accept or reject your changes to the main project.

After this, we will see how to work in the forked repositories which were forked by the collaborators. If the collaborator tries to change in his forked repository, it will not affect the main repository.

Upon forking ever member gets their own copy of the source repository in their GitHub account. Collaborators can then go ahead and clone their ‘forked repository’ on their local machines.

## Experiment No. 02

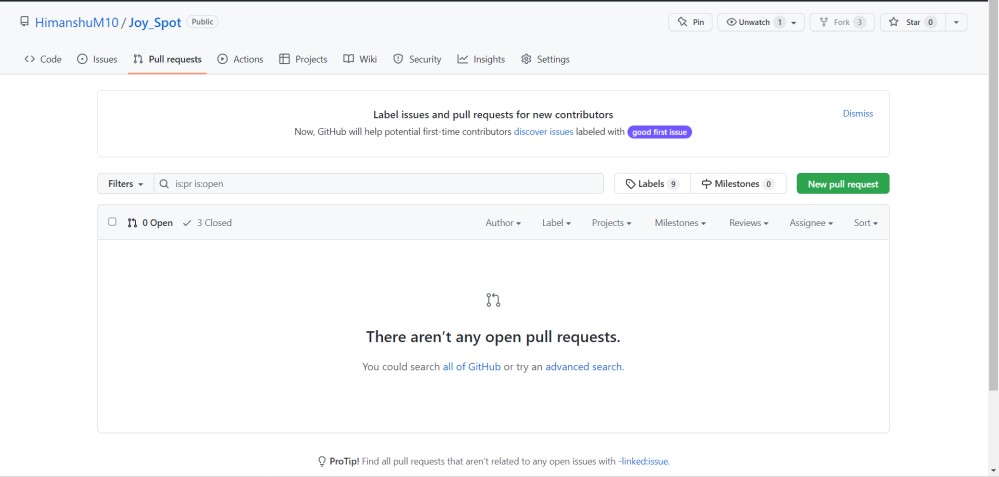
**Aim:** Open and close a pull request

Open a Pull Request:

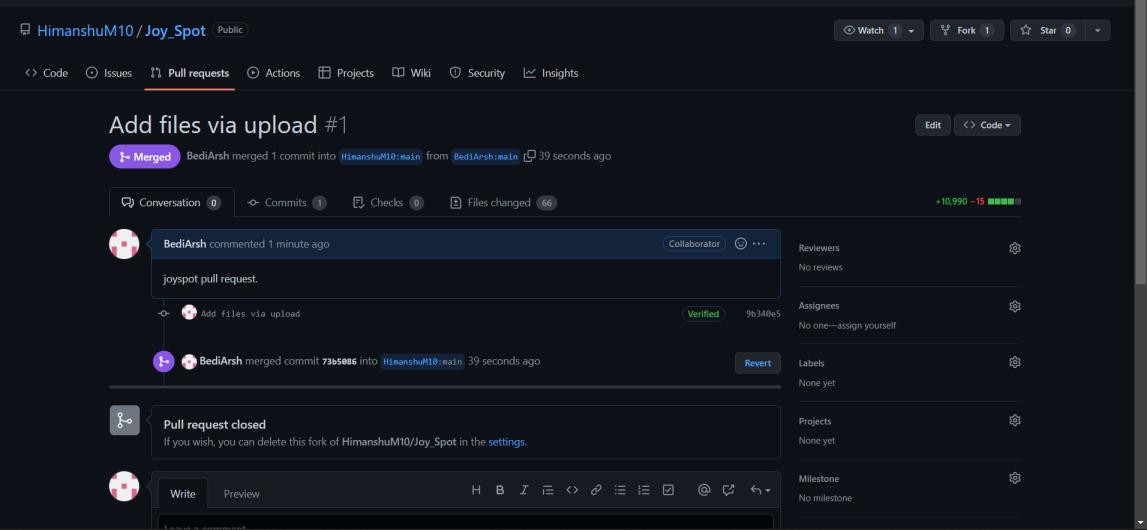
1. click on the **Fork** button in the top-right corner. This creates a new copy of my demo repo under your GitHub user account with a URL like: [https://github.com/<YourUserName>/demo](https://github.com/%3cYourUserName%3e/demo)

The copy includes all the code, branches, and commits from the original repo.Next,

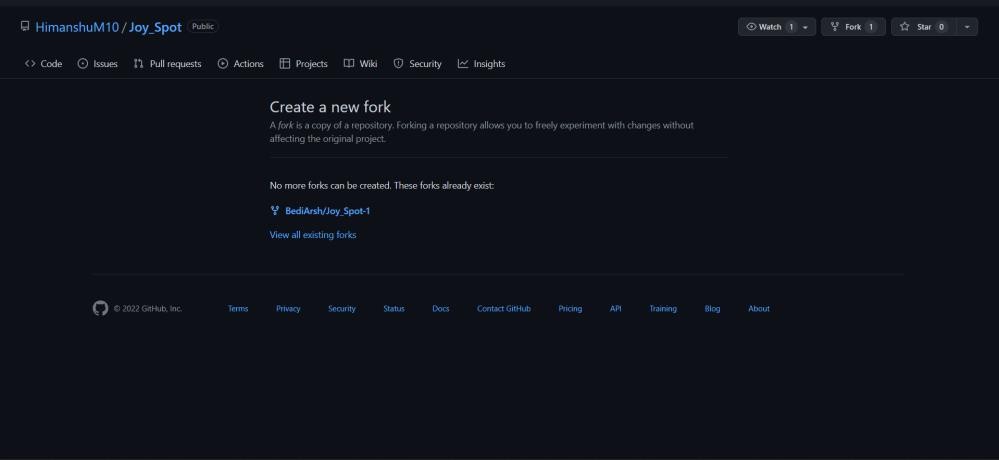
1. clone the repository and Create a new branch **.**
2. Now you can make changes to the code. The following code creates a new branch, makes an arbitrary change, and pushes it to **new\_branch**:
3. Once you push the changes to your repo, the **Compare & pull request** button will appear in GitHub.



1. Click it and you'll be taken to this screen:



3. Open a pull request by clicking the **Create pull request** button.

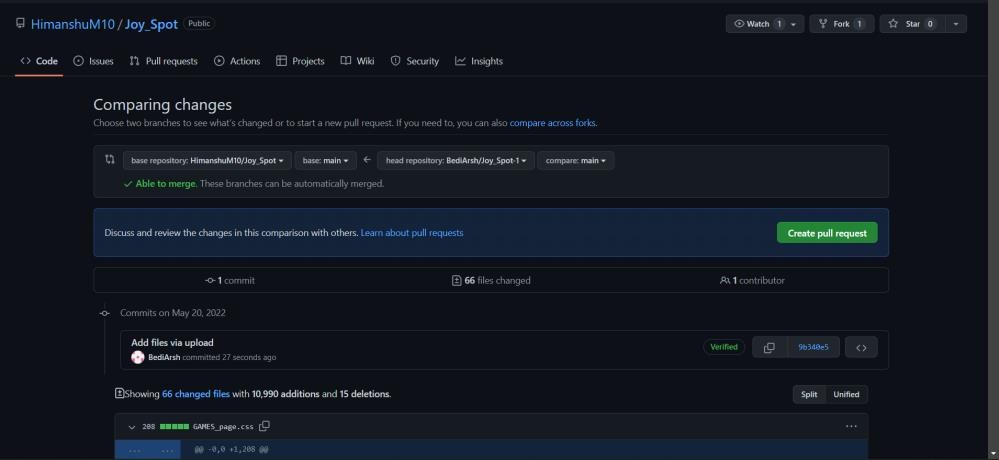


This allows the repo's maintainers to review your contribution. From here, they can merge it if it is good, or they may ask you to make some changes.

Closing a pull request

You may choose to *close* a pull request without [merging it into](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/incorporating-changes-from-a-pull-request/merging-a-pull-request) [the](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/incorporating-changes-from-a-pull-request/merging-a-pull-request) [upstream branch.](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/incorporating-changes-from-a-pull-request/merging-a-pull-request) This can be handy if the changes proposed in the branch are no longer needed, or if another solution has been proposed in another branch.

1. Under your repository name, click **Pull requests**



* 1. At the bottom of the pull request, below the comment box, click

**Close/Merge pull request**.

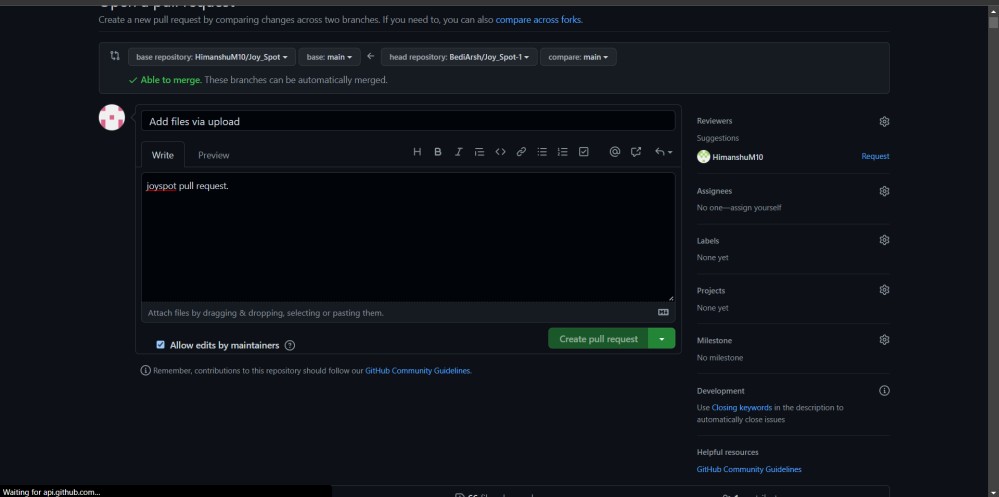
* 1. Optionally, [delete the branch](https://docs.github.com/en/articles/deleting-unused-branches). This keeps the list of branches in your repository tidy.

### Experiment No. 03

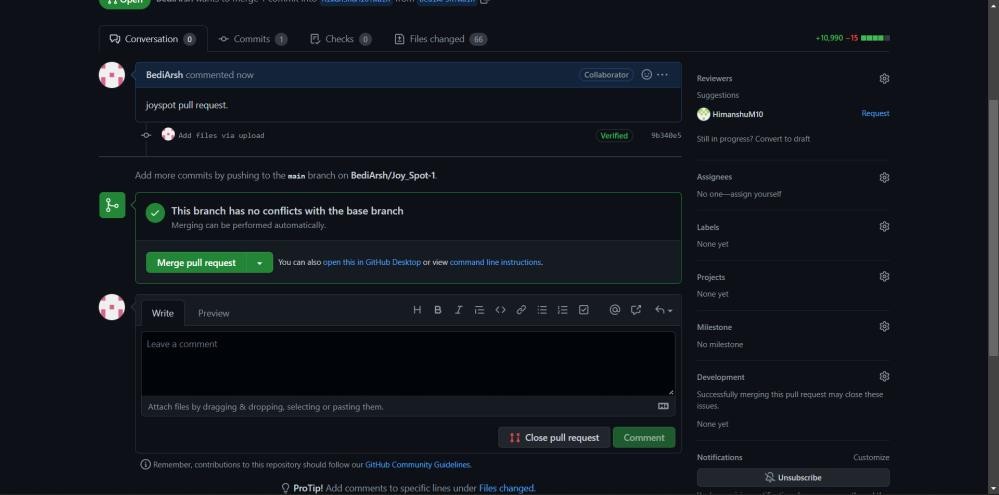
**Aim:** Each project member shall create a pull request on a team members repo and close pull requests generated by team members on own Repo as a maintainer.

Pull Request Generated by team members:

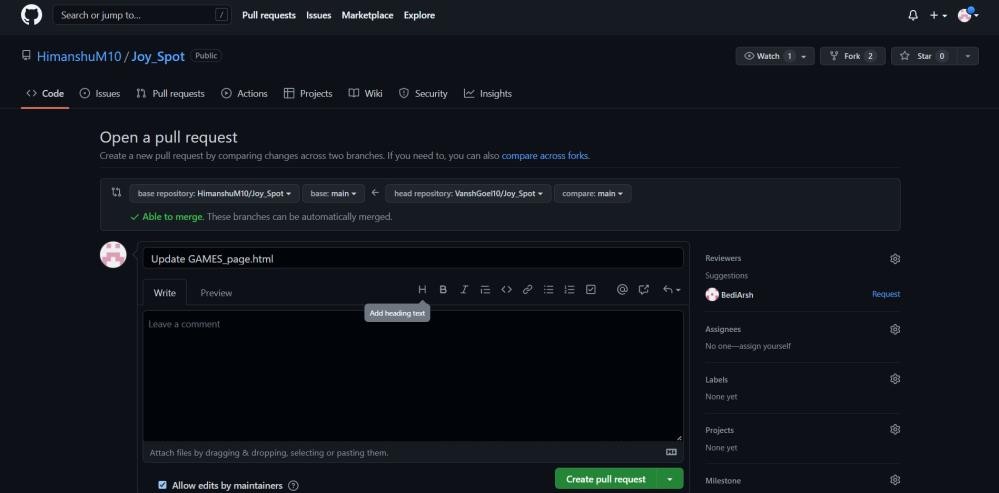
1. By Arsh Bedi



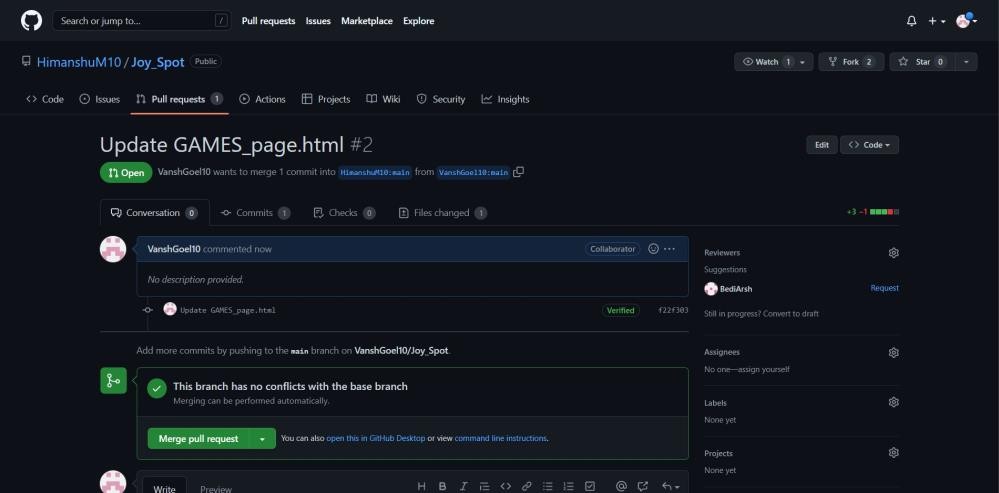
Accepted by Maintainer.



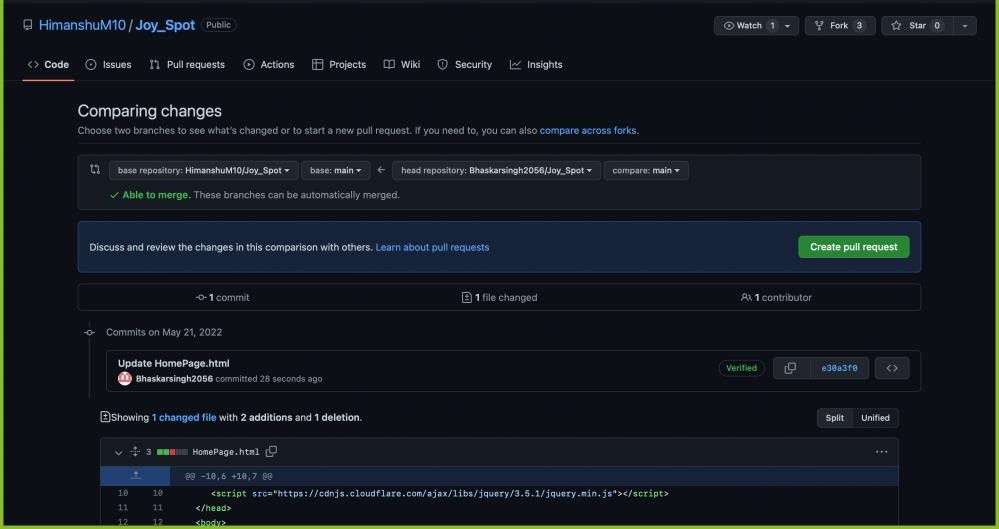
1. By Vansh Goel



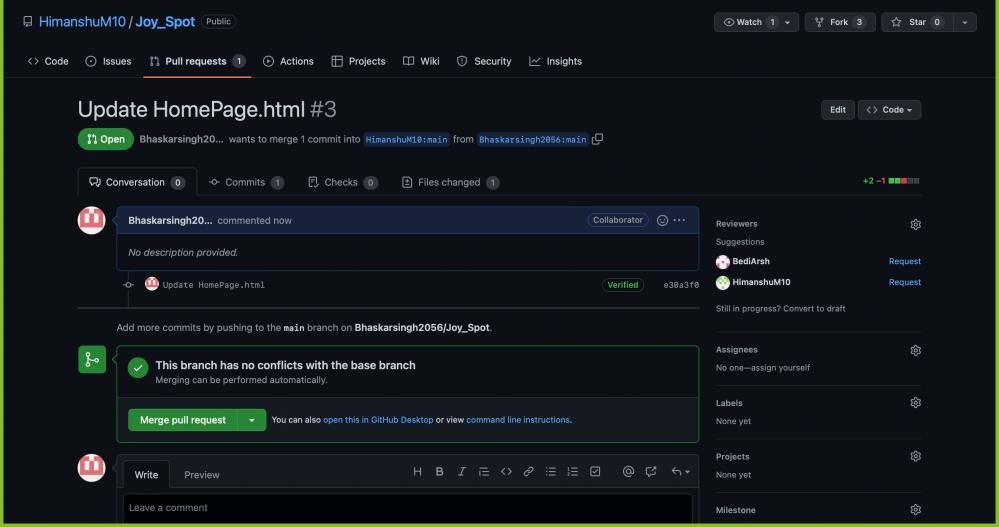
Accepted By Maintainer.



1. By Bhaskar



Accepted by Maintainer.

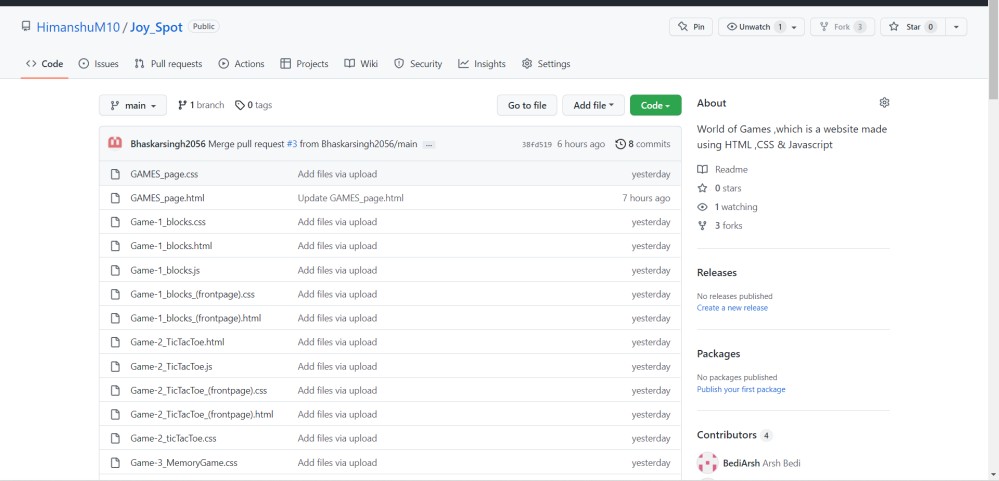


## Experiment No. 04

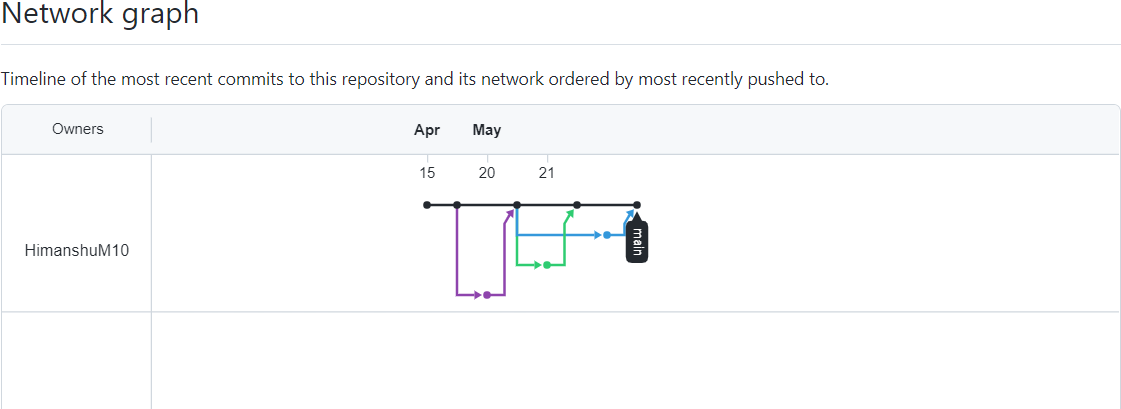
### Aim: Publish and print network graphs

Network Graph:

The network graph displays the branch history of the entire repository network, including branches of the root repository and branches of forks that contain commits unique to the network.

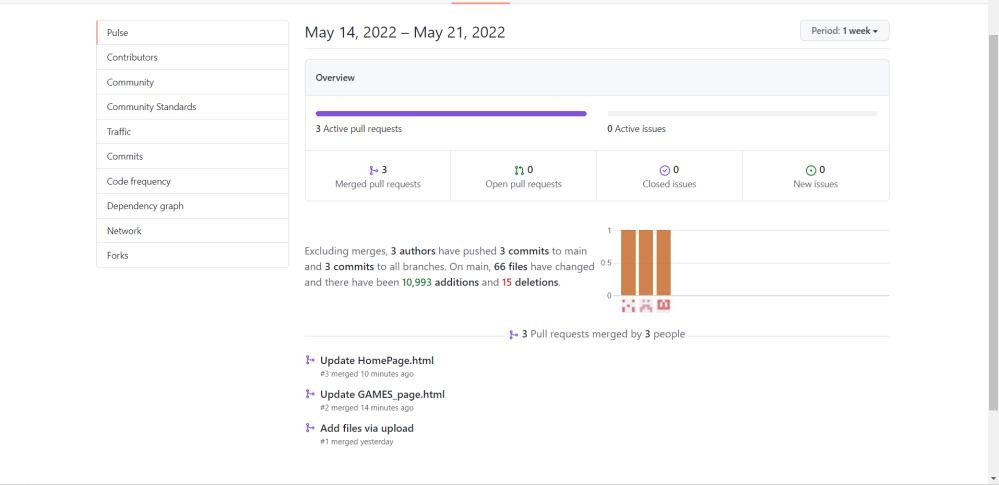


**Tip:** To see older branches, click and drag within the graph.

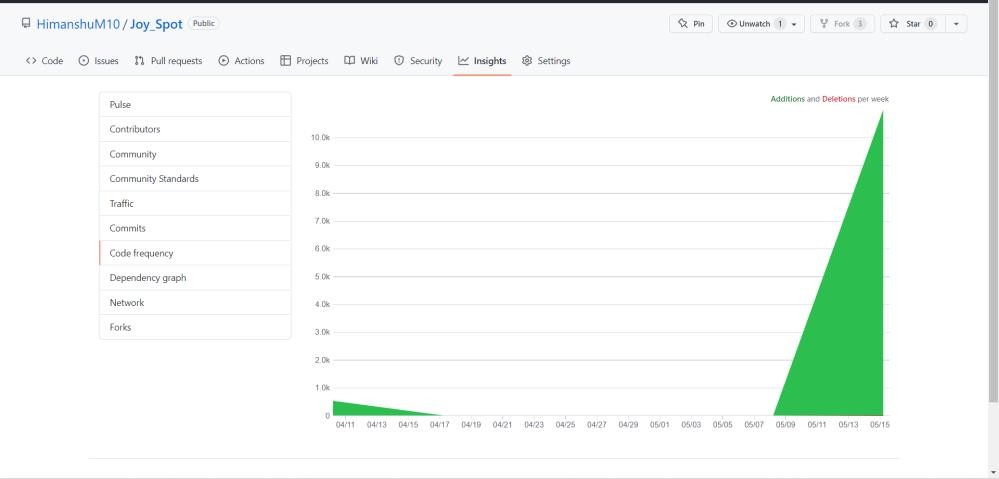


### Accessing the network graph

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Insights**.



Now we see,



1. In the left sidebar, click **Network**.