A Project report

On

**“Space Up”**

With

**Source Code Management**

(CS181)

Submitted by

Team Member 1 Vanshika Gupta Roll No.2110991516

Team Member 2 Smridhi Agnihotri Roll No.2110991373

Team Member 3 Sneha Chawla Roll No.2110991376

Team Member 4 Aakanksha Roll No.2110991604



**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/Batch | **2nd/2021** |
| Vertical Name | **Alpha** | Group No | G24 |
| Course Coordinator | **Dr. Neeraj Singla** | | |
| Faculty Name | **Dr.\_\_\_\_\_\_\_Gagandeep kaur\_\_\_\_\_\_\_\_\_\_** | | |

Submission

Name:

Signature:

Date:

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Title** | **Page No.** |
| 1 | Version control with Git | 1-2 |
| 2 | Problem Statement | 3-6 |
| 3 | Objective | 7 |
| 4 | Resources Requirements – Frontend / Backend | 8-10 |
| 5 | Concepts and commands | 11-18 |
| 6 | Workflow and Discussion | 19-30 |
| 7 | Reference | 31 |

**1. Version control with Git**

A **Version Control System** (VCS) is an integrated fool-proof framework

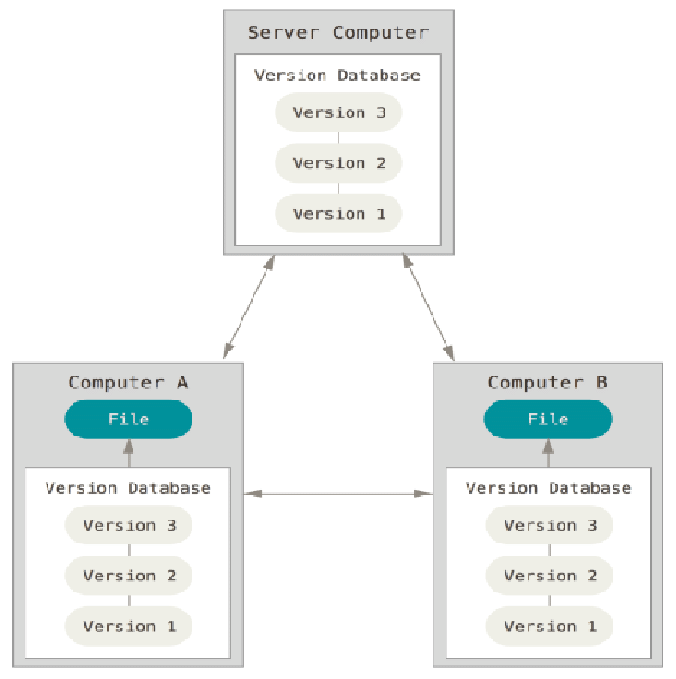
For

* Backup and Restore
* Short and long-term undo
* Tracking changes
* Synchronization
* Collaborating
* Sandboxing

... With minimal overhead.

To overcome problems related to centralization**, Distributed VCSs**

(DVCSs) were invented. Keeping a complete copy of database in every working directory.



Actually the most simple and most powerful implementation of any VCS.

**Git** is the most commonly used version control system. Git tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions should you ever need to. Git also makes collaboration easier, allowing changes by multiple people to all be merged into one source.

So regardless of whether you write code that only you will see, or work as part of a team, Git will be useful for you.



Git is software that runs locally. Your files and their history are stored on your computer. You can also use online hosts (such as [GitHub](https://github.com/) or [Bitbucket](https://bitbucket.org/)) to store a copy of the files and their revision history. Having a centrally located place where you can upload your changes and download changes from others, enable you to collaborate more easily with other developers. Git can automatically merge the changes, so two people can even work on different parts of the same file and later merge those changes without losing each other’s work!

**2. Problem Statement**

For almost all software projects, the source code is like the crown jewels - a precious asset whose value must be protected. For most software teams, the source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have collected and refined through careful effort.

Software developers working in teams are continually writing new source code and changing existing source code. The code for a project, app or software component is typically organized in a folder structure or “file tree”. One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.

It is obvious that while working in huge organizations like this, managing the source code efficiently would pose a huge problem.

How does one ensure that the source code is stored safely, that the code is easily accessible to all the developers working on it, that one can track all code modifications at hand?

Without a proper tool, sharing the source code and all tasks mentioned above would be very tedious and lacking efficiency.

Say you decided to share your code with your team through email- it would be disastrous, to say the least. Imprudently sending and receiving modified versions of the code would make it very hard to keep track of its different versions and would hinder the productivity of all the developers in the team.

**Hence, the problem is to manage the source code professionally, in the most efficient way possible while working in a team with multiple developers.**

# **How to add collaborators to a repository in GitHub?**

# Even if you have a public repository in GitHub, not everyone has the permission to push code into your repository. Other users have a read-only access and cannot modify the repository. In order to allow other individuals to make changes to your repository, you need to invite them to collaborate to the project.

# 

# **How to push to github?**

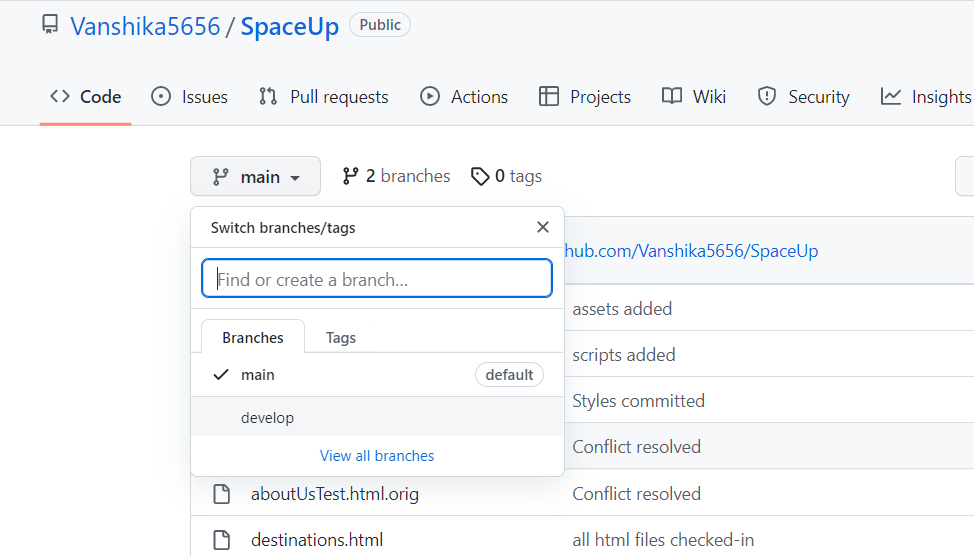
# After you make and commit changes locally, you can share them with the remote repository using git push. Pushing changes to the remote makes your commits accessible to others who you may be collaborating with. This will also update any open pull requests with the branch that you're working on.

# 

# **How to work with branches in git?**

# In Git, a branch is a new/separate version of the main repository.

# Let's say you have a large project, and you need to update the design on it, you need different branches to work in other than just the main branch.

****

# **How to merge and resolve conflicts?**

# The git merge command helps a contributor add to a project from a [branch](https://phoenixnap.com/kb/git-create-new-branch). The concept is one of the core ideas of collaborative programming, allowing multiple people to work on their part of the code without any conflicts.

# 

# **How to pull code from github?**

# The term pull is used to receive data from GitHub. It fetches and merges changes from the remote server to your working directory. The **git pull command** is used to pull a repository.

# 

**3. Objective**

Basically, using version control systems like github or bitbucket keeps track on changes made on a particular software and take a snapshot of every modification.

Let’s suppose if a team of developer add some new functionalities in an application and the updated version is not working properly so as version control system keeps track of our work so with the help of version control system we can omit the new changes and continue with the previous version.

Using version control systems, we aim to achieve the following:

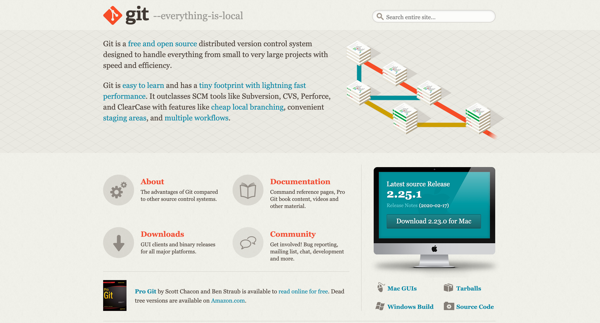
* Enhance the project development speed by providing efficient collaboration,
* Leverage the productivity, expedite product delivery, and skills of the employees through better communication and assistance,
* Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change,
* Employees or contributors of the project can contribute from anywhere irrespective of the different geographical locations through this VCS,
* For each different contributor to the project, a different working copy is maintained and not merged to the main file unless the working copy is validated. The most popular example is Git, Helix core, Microsoft TFS,
* Helps in recovery in case of any disaster or contingent situation,
* Informs us about Who, What, When, Why changes have been made.

**4. Resources requirement**

We had used resources for code management in this project as listed below:

1. **Gitbash**: Git Bash is a source control management system for Windows. It allows users to type Git commands that make source code management easier through versioning and commit history. Bash is a Linux-based command line (that has been ported over to Windows) while **Shell** is a native Windows command line.

*Git Bash is not a GUI software, it is a command-line prompt. You will only use Git Bash to write and run commands on the terminal.*



2. **GitHub**: GitHub hosts Git repositories and provides developers with tools to ship better code through command line features, issues (threaded discussions), pull requests, code review, or the use of a collection of free and for-purchase apps in the GitHub Marketplace.

Work is organized into repositories where developers can outline requirements or direction and set expectations for team members. Then, using the GitHub flow, developers simply create a branch to work on updates, commit changes to save them, open a pull request to propose and discuss changes, and merge pull requests once everyone is on the same page.

Also for frontend part we had used:

1. **HTML**: HTML is a *markup language* that defines the structure of your content. HTML consists of a series of [elements](https://developer.mozilla.org/en-US/docs/Glossary/Element), which you use to enclose, or wrap, different parts of the content to make it appear a certain way, or act a certain way. The enclosing [tags](https://developer.mozilla.org/en-US/docs/Glossary/Tag) can make a word or image hyperlink to somewhere else, can italicize words, and can make the font bigger or smaller, and so on.



2. **CSS**: Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

I will list down some of the key advantages of learning CSS:

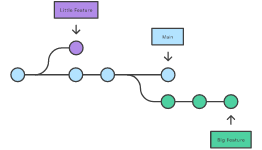
* Create Stunning Web site - Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs, variations in display for different devices and screen sizes as well as a variety of other effects.
* Control web - CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.

3. **JavaScrip**t:Javascript is a MUST for students and working professionals to become a great Software Engineer especially when they are working in Web Development Domain. I will list down some of the key advantages of learning JavaScript:

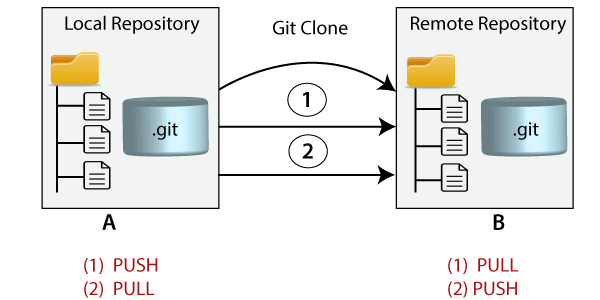
* Once you learnt Javascript, it helps you developing great front-end as well as back-end softwares using different Javascript based frameworks like jQuery, Node.JS etc.
* JavaScript is everywhere, it comes installed on every modern web browser and so to learn Javascript you really do not need any special environment setup.
* JavaScript helps you create really beautiful and crazy fast websites. You can develop your website with a console like look and feel and give your users the best Graphical User Experience.
* JavaScript usage has now extended to mobile app development, desktop app development, and game development.
* Due to high demand, there is tons of job growth and high pay for those who know JavaScript.

**5. CONCEPTS AND COMMANDS**

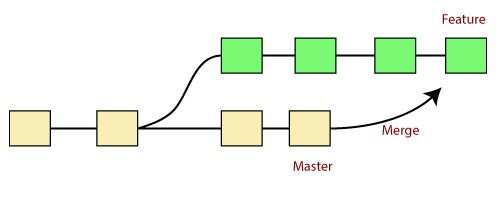
**Git branch:** In Git, branches are a part of your everyday development process. Git branches are effectively a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug—no matter how big or how small—you spawn a new branch to encapsulate your changes.



**Git clone:** Git clone is a Git command line utility which is used to target an existing repository and create a clone, or copy of the target repository. Git clone is primarily used to point to an existing repo and make a clone or copy of that repo at in a new directory, at another location. The original repository can be located on the local filesystem or on remote machine accessible supported protocols.



**Git Merge and Merge conflict:** In Git, the merging is a procedure to connect the forked history. It joins two or more development history together. The git merge command facilitates you to take the data created by git branch and integrate them into a single branch. Git merge will associate a series of commits into one unified history. Generally, git merge is used to combine two branches.



The git merge command is used to merge the branches. The syntax for the git merge command is as:

1. $ git merge <query>

It can be used in various context. Some are as follows:

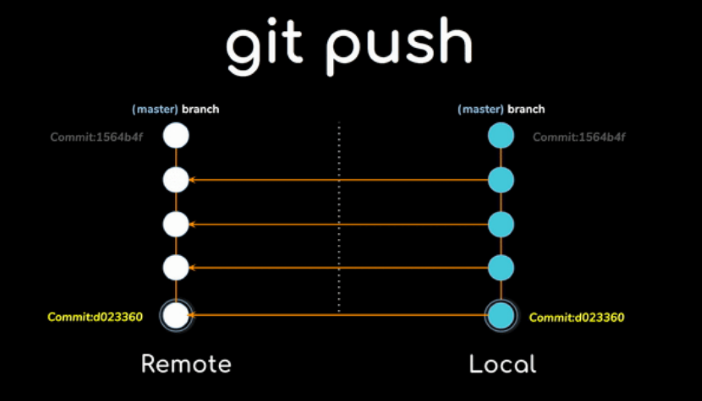
**Scenario1: To merge the specified commit to currently active branch:**

Use the below command to merge the specified commit to currently active branch.

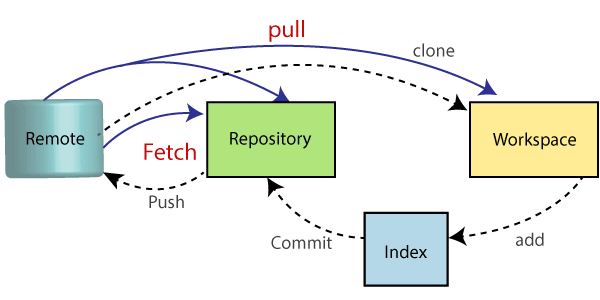
1. $ git merge <commit>

The above command will merge the specified commit to the currently active branch. You can also merge the specified commit to a specified branch by passing in the branch name in <commit>.

**Git push:** The git push command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repo. Git push is most commonly used to publish an upload local changes to a central repository. After a local repository has been modified a push is executed to share the modifications with remote team members.

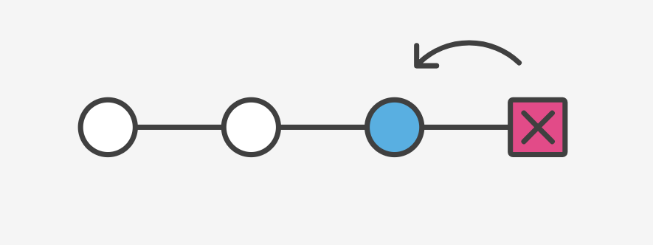


**Git Pull:** The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content. Merging remote upstream changes into your local repository is a common task in Git-based collaboration work flows. The git pull command first runs git fetch which downloads content from the specified remote repository. Then a git merge is executed to merge the remote content refs and heads into a new local merge commit.



**Git Reset:** The git reset command is a complex and versatile tool for undoing changes. It has three primary forms of invocation. These forms correspond to command line arguments --soft, --mixed, --hard. The three arguments each correspond to Git's three internal state management mechanisms, The Commit Tree (HEAD), The Staging Index, and The Working Directory. At a surface level, git reset is similar in behavior to git checkout. Where git checkout solely operates on the HEAD ref pointer, git reset will move the HEAD ref pointer and the current branch ref pointer.

**Git Revert:** The git revert command can be considered an 'undo' type command, however, it is not a traditional undo operation. Instead of removing the commit from the project history, it figures out how to invert the changes introduced by the commit and appends a new commit with the resulting inverse content. This prevents Git from losing history, which is important for the integrity of your revision history and for reliable collaboration.



**Git Checkout**: In Git terms, a "checkout" is the act of switching between different versions of a target entity. The git checkout command operates upon three distinct entities: files, commits, and branches. In addition to the definition of "checkout" the phrase "checking out" is commonly used to imply the act of executing the git checkout command. Checking out branches is similar to checking out old commits and files in that the working directory is updated to match the selected branch/revision; however, new changes are saved in the project history—that is, it’s not a read-only operation.

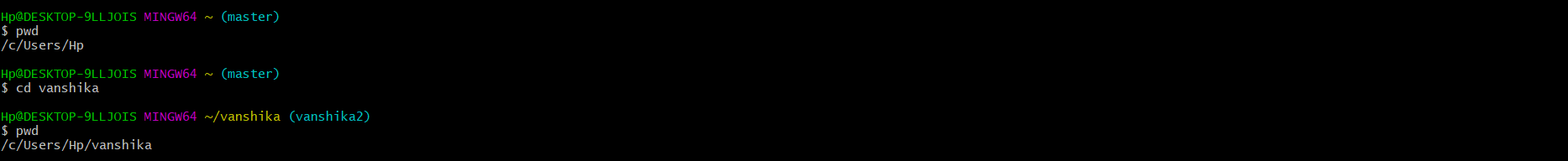
**cd**

The cd command stands for change directory. It takes the path of the directory where we want to move to as the argument.

In the case below, we move from the current working directory to /c/example/ using cd. To confirm that we have moved to the new directory, check the current working directory using the pwd command.

*Syntax:* cd <existing folder name>

**SNAPSHOT:**

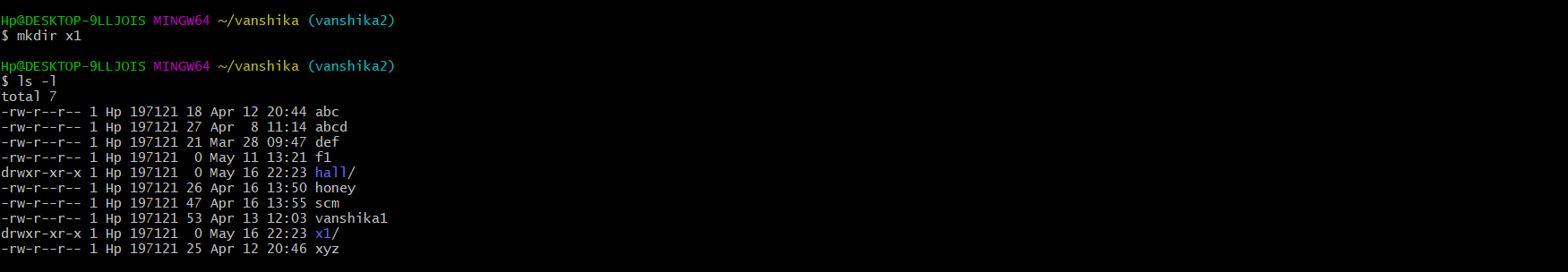


**mkdir**

The mkdir command stands for making a Directory. It helps to create a directory wherever we want to create it.

*Syntax:* mkdir <directoryName>

**SNAPSHOT:**



**Cat**

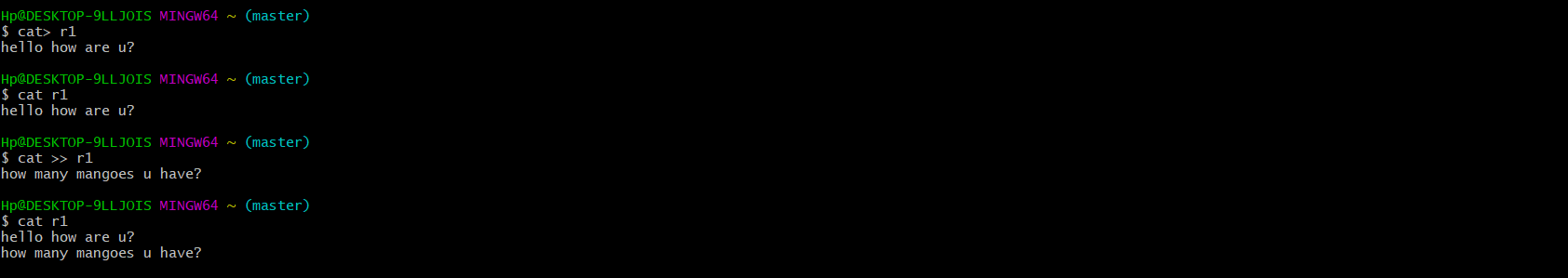
The cat (short for “concatenate “) command is one of the most frequently used commands in Git. cat command allows us to create single or multiple files, view content of a file, concatenate files and redirect output in terminal or files.

*Syntax:* cat > filename ->to create and write in a file

cat >> filename ->to append in a file

cat filename ->to read a file

**SNAPSHOT:**

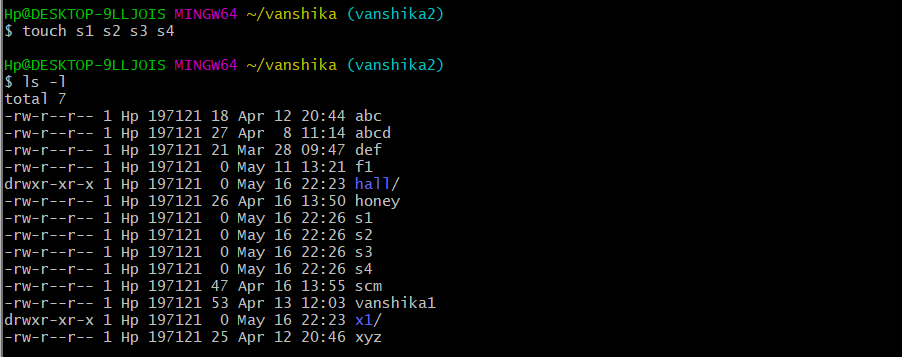


**Touch**

The touch command is the easiest way to create new, empty files. It is also used to change the timestamps (i.e., dates and times of the most recent access and modification) on existing files and directories. Touch’s

*Syntax:* touch [options] filename(s)

**SNAPSHOT:**

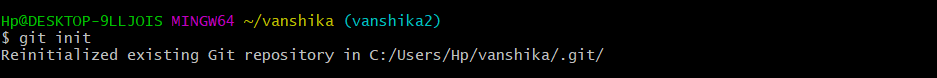


**git init**

The git init command creates a new Git repository. It can be used to convert an existing, unversioned project to a Git repository or initialize a new, empty repository. Most other Git commands are not available outside of an initialized repository, so this is usually the first command you'll run in a new project.

*Syntax:* git init

**SNAPSHOT:**

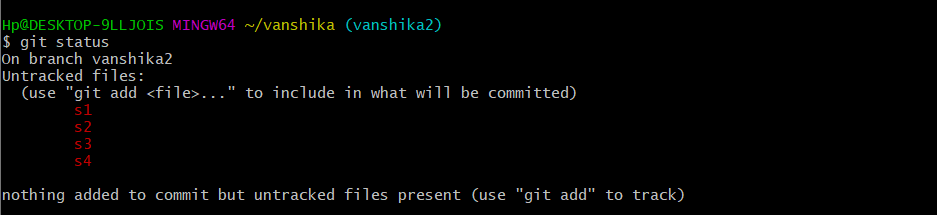


**git status**

The git status command is used to display the state of the repository and staging area. It allows us to see the tracked, untracked files and changes. This command will not show any commit records or information. Mostly, it is used to display the state between Git Add and Git commit command.

*Syntax:* git status

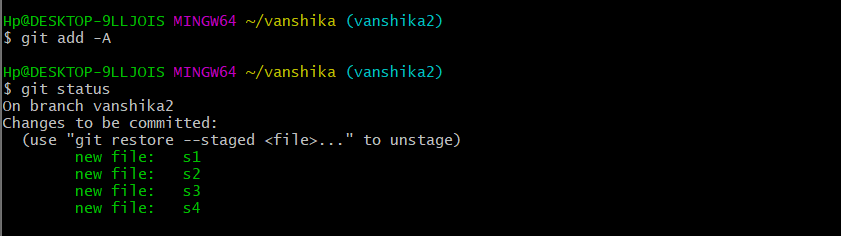
**SNAPSHOT:**



**git add**

The git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run git commit.

**SNAPSHOT:**

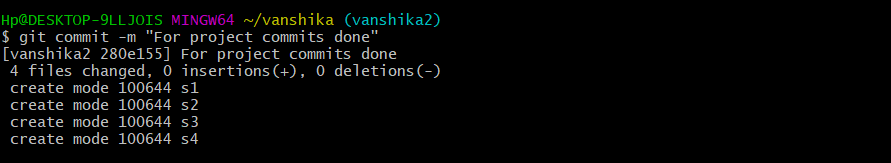


**git commit**

First, you need to stage the file with git add, then you can commit the staged snapshot. This command will add hello.py to the Git staging area. We can examine the result of this action by using the git status command. # with '#' will be ignored, and an empty message aborts the commit.

*Syntax***:** git commit -m "message to be printed"

**SNAPSHOT:**



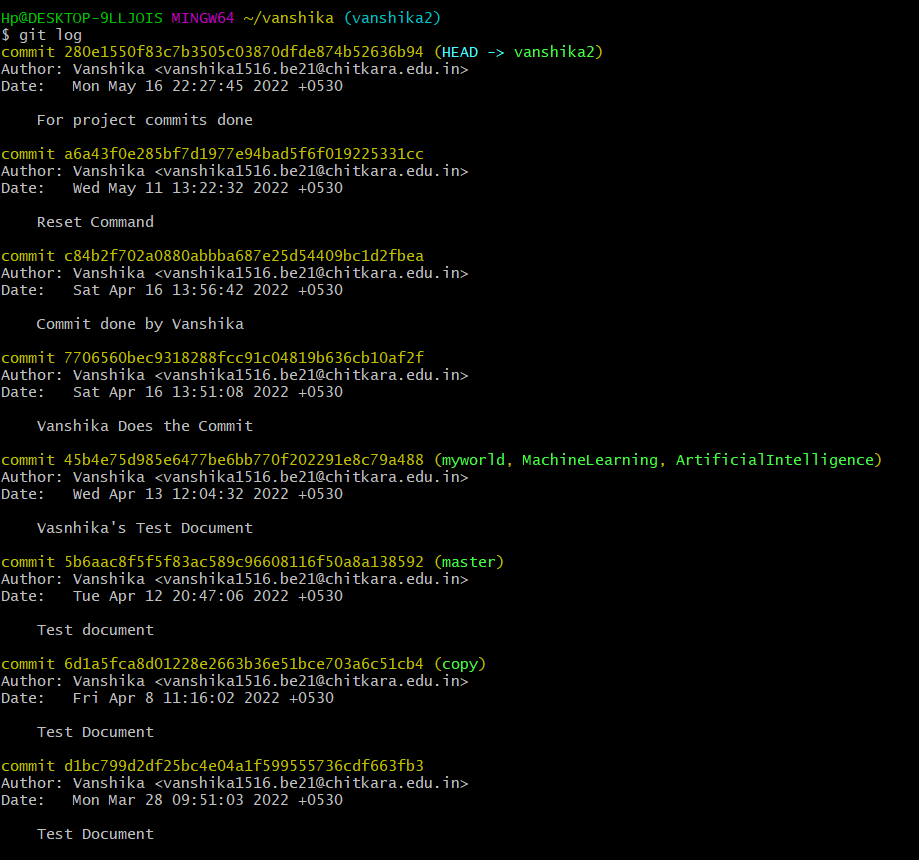
**git log**

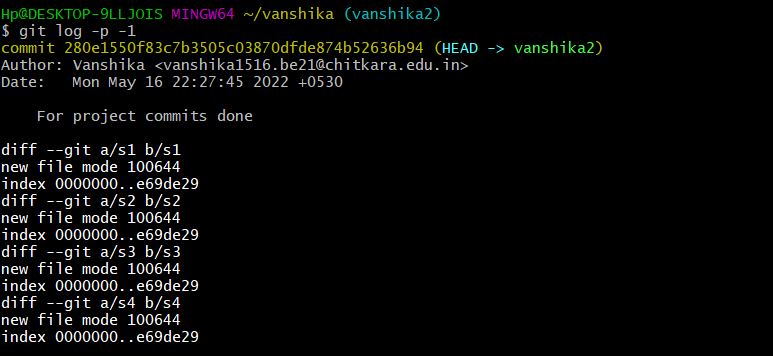
The git log command displays all of the commits in a repository's history. By default, the command displays each commit.

*Syntax:* git log

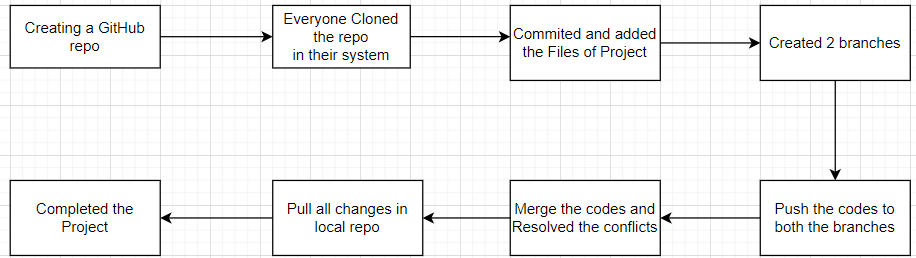
git log -p -3 ->to see the last 3 commits done

**SNAPSHOT:**





**6. Workflow and discussion**



Our workflow and discussion will cover following points:

* Initializing a new GitHub repo
* Cloning an existing GitHub repo
* Committing a modified version of a file to the repo- committing the entire project
* Created two branches
* Pushing the codes
* Merge and resolve conflict
* Pulling the changes so as to maintain a local copy of the entire project

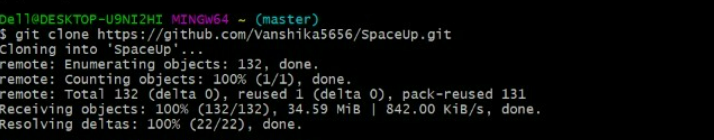
**Git repository:** A [Git repository](https://bitbucket.org/product/code-repository) is a virtual storage of your project. It allows you to save versions of your code, which you can access when needed.

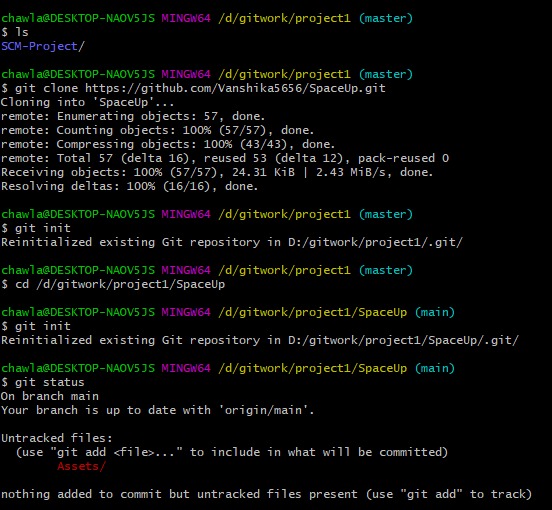
Initializing a new repository

* To create a new repo, we visited GitHub and then created a repository named SpaceUp

**Cloning an existing repository**: git clone

* If a project has already been set up in a central repository, the clone command is the most common way for users to obtain a local development clone. Like git init, cloning is generally a one-time operation. Once a developer has obtained a working copy, all [version control](https://bitbucket.org/product/version-control-software) operations are managed through their local repository.
* git clone <repo url>
* git clone is used to create a copy or clone of remote repositories. You pass git clone a repository URL. Git supports a few different network protocols and corresponding URL formats.





**Saving changes to the repository**: git add and git commit

Now that you have a repository cloned or initialized, you can commit file version changes to it. The following example assumes you have set up a project at SpaceUp. The steps being taken in this example are:

* Change directories to SpaceUp
* Create a new file SpaceUp
* git add -A to the repository staging area
* Create a new commit with a message describing what work was done in the commit

After executing this example, your repo will now have CommitTest.txt added to the history and will track future updates to the file.

This example introduced two additional git commands: add and commit. This was a very limited example, but both commands are covered more in depth on the [git add](https://www.atlassian.com/git/tutorials/saving-changes) and [git commit](https://www.atlassian.com/git/tutorials/saving-changes/git-commit) pages. Another common use case for git add is the --all option. Executing git add --all will take any changed and untracked files in the repo and add them to the repo and update the repo's working tree.

**Repo-to-repo collaboration**: git push

* We created two branches in it named Main and develop
* Every member pushed the work in the develop branch
* When the project got completed, we pushed the work to the main branch

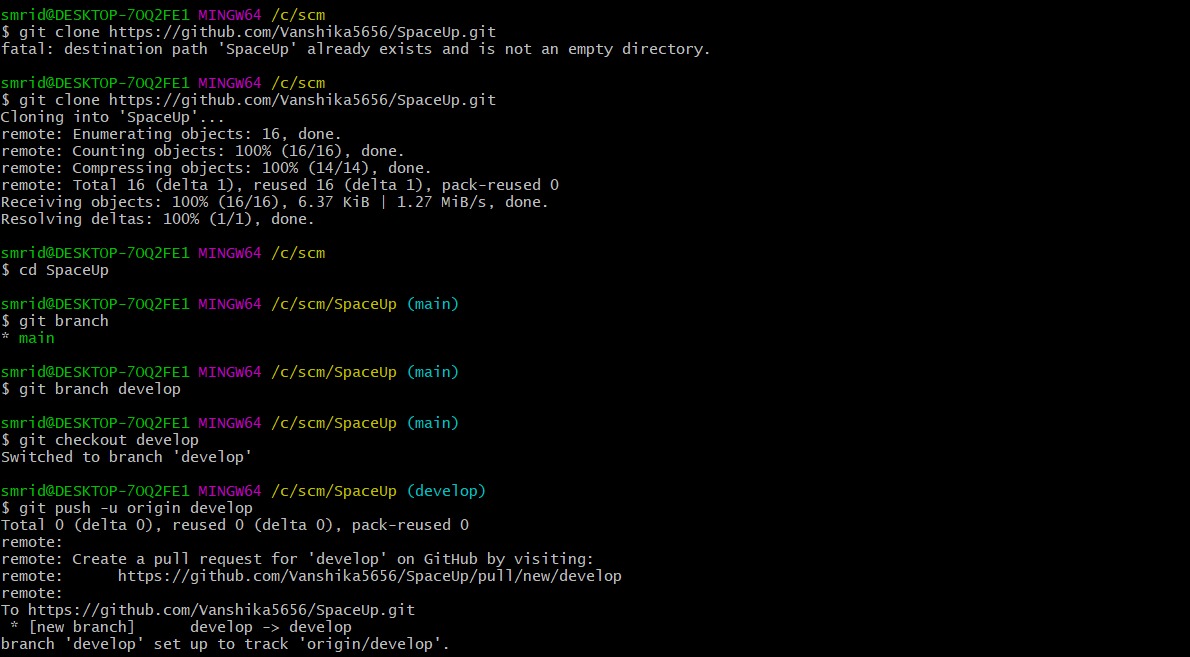
**Branches**: In Git, a branch is a new/separate version of the main repository.

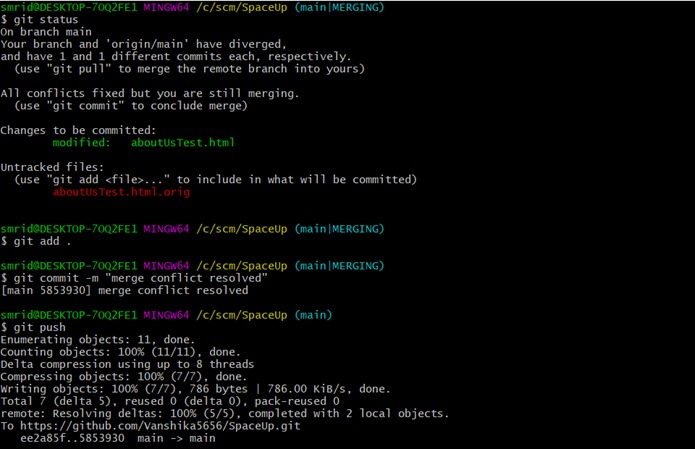
Branches allow you to work on different parts of a project without impacting the main branch.

When the work is complete, a branch can be merged with the main project.

The steps to take would be:

1. Fork a repository on GitHub
2. Clone it onto your computer
3. Make a branch and move to it: git checkout -b fixingBranch
4. Make changes to the files
5. Commit the changes to the history
6. Push the branch up to your forked version: git push origin fixingBranch
7. On GitHub, submit a Pull Request of your fixingBranch
8. Once the pull request is merged, [delete](https://github.com/blog/1377-create-and-delete-branches) the fixingBranch on your forked repo on GitHub and on your computer





**Git Push:** The git push command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repo.

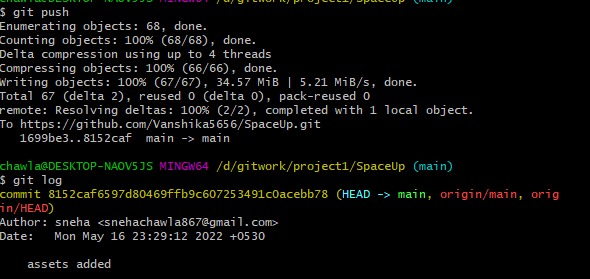
Using Command line to PUSH to GitHub:-

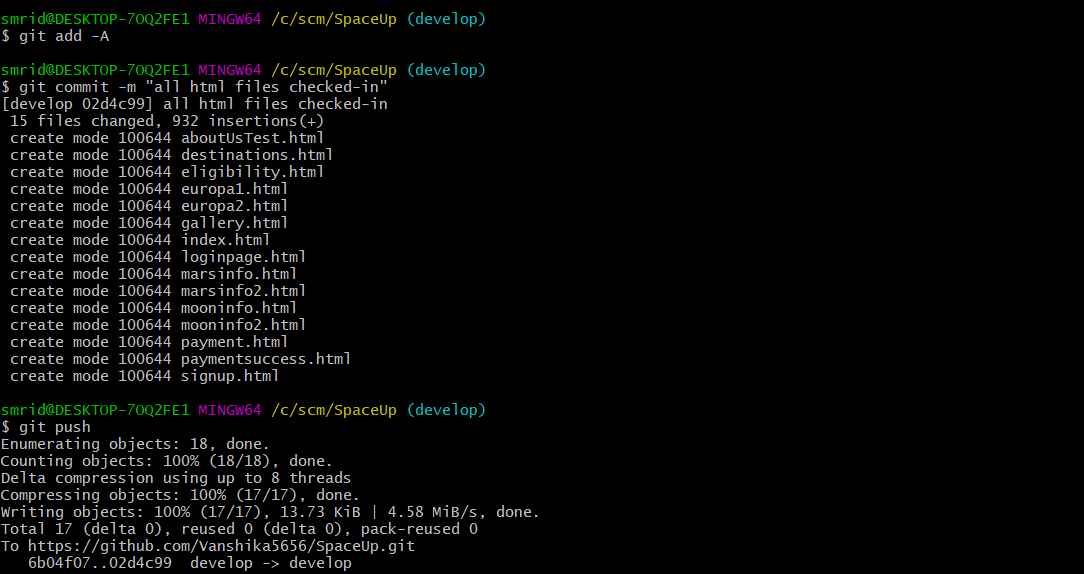
1. Creating a new repository.
2. Open your Git Bash.
3. Create your local project in your desktop directed towards a current working directory.
4. Initialize the git repository.
5. Add the file to the new local repository.
6. Commit the files staged in your local repository by writing a commit message.

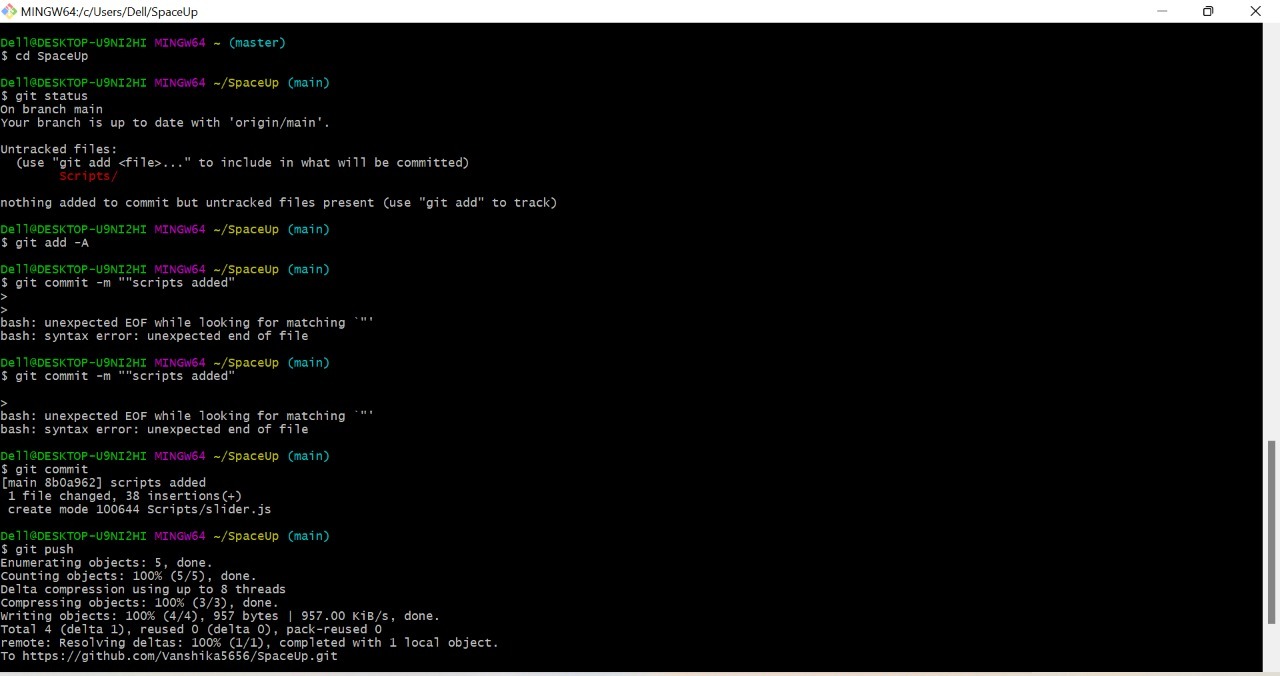
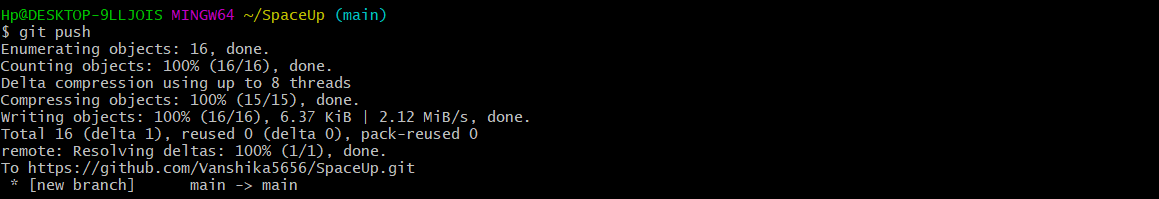
7. Copy your remote repository's URL from GitHub.

8. Add the URL copied, which is your remote repository to where your local content from your repository is pushed.

### 9. **Push the code in your local repository to GitHub**





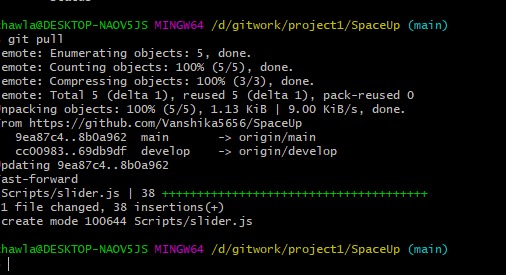


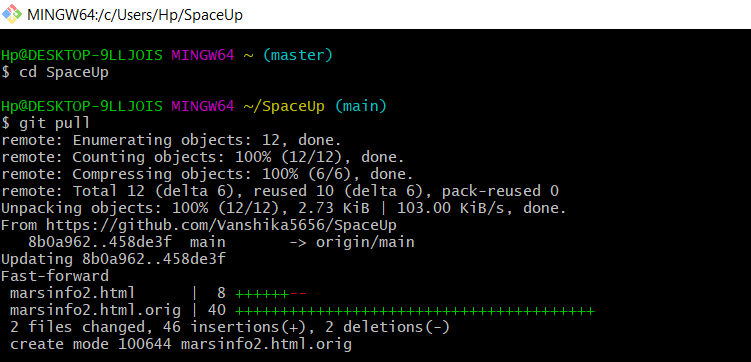
**Pulling the Changes:**

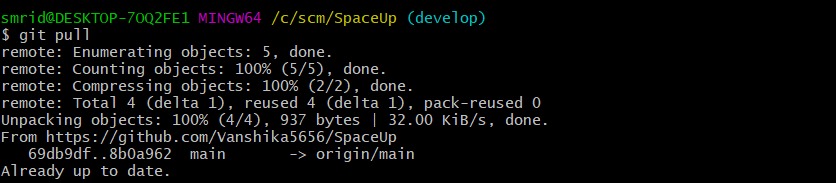
Git pull, in a nutshell, is a two-part process. First, your remote-tracking branch is synced with the “true” branch in the remote repository.

Then, your local branch is compared to the remote-tracking branch and receives the new commits so it can catch up to the current state of the remote branch.

1. First, run git status. Git will tell you the repository is clean, nothing to worry about.
2. Then run git fetch.
3. Next, run git status again. Git will say your branch is one commit behind.
4. Finally, run git pull to update your local branch.







**Merge and Resolve Conflict**

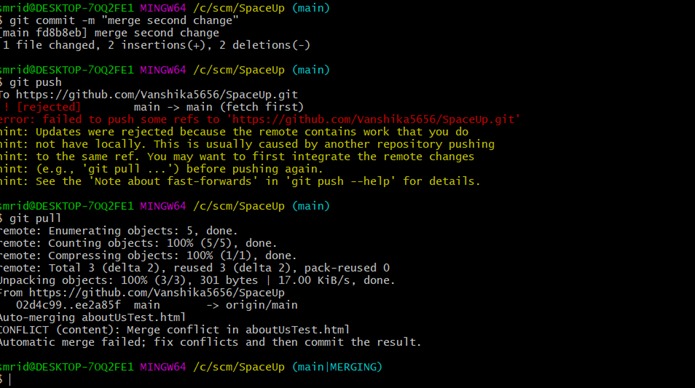
A merge conflict is an event that takes place when Git is unable to automatically resolve differences in code between two commits. Git can merge the changes automatically only if the commits are on different lines or branches.

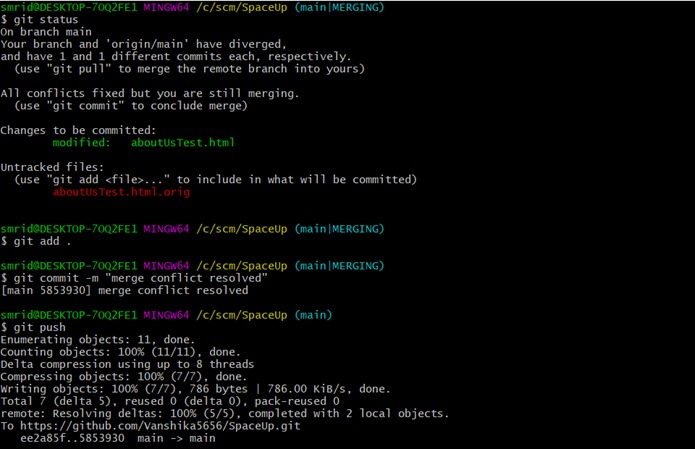
The following is an example of how a Git merge conflict works:

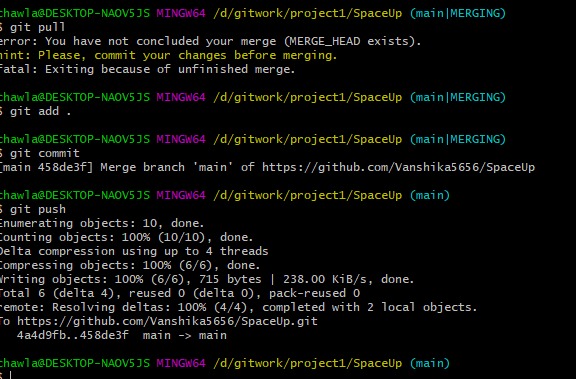
pull-push.

Let’s assume there are two developers: Developer A and Developer B. Both of them pull the same code file from the remote repository and try to make various amendments in that file. After making the changes, Developer A pushes the file back to the remote repository from his local repository. Now, when Developer B tries to push that file after making the changes from his end, he is unable to do so, as the file has already been changed in the remote repository.

To prevent such conflicts, developers work in separate isolated branches. The Git merge command combines separate branches and resolves any conflicting edits.



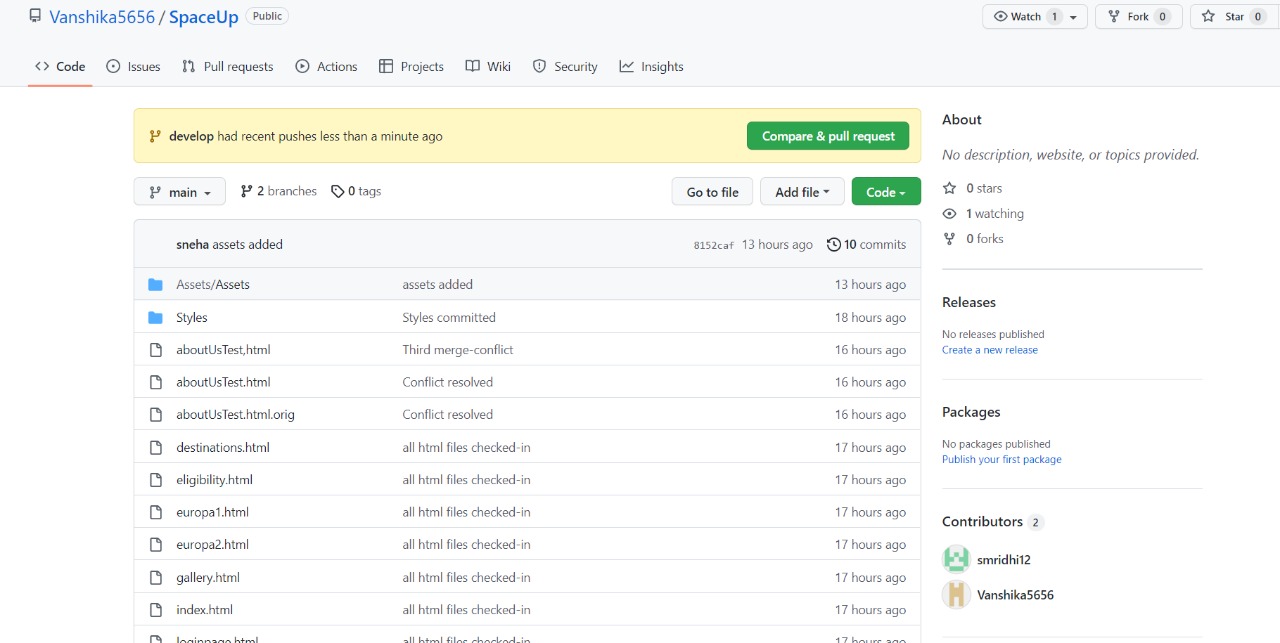


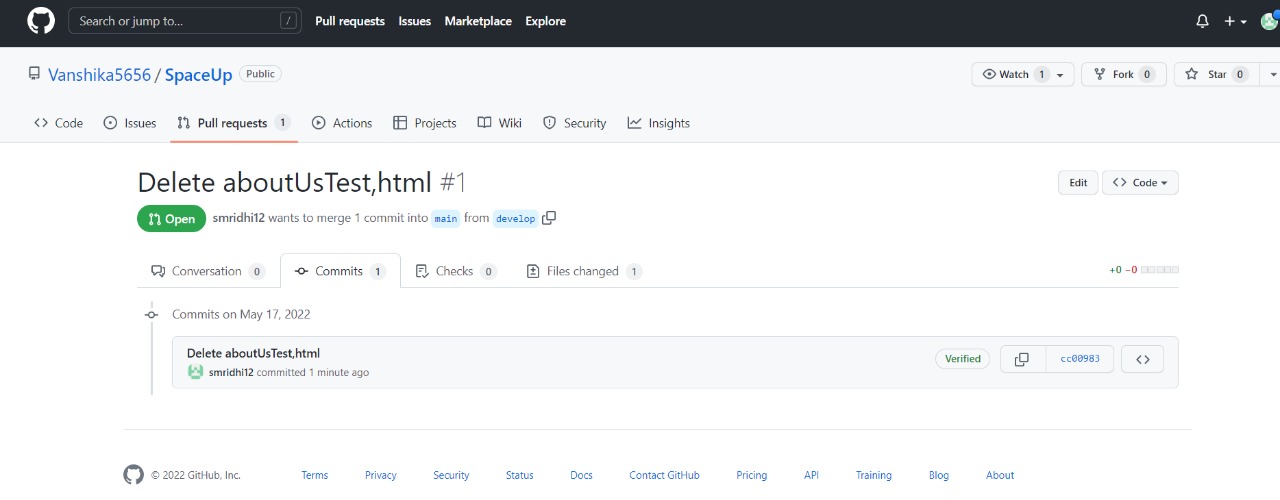


**Pull Request:** A pull request is an event in Git where a contributor asks a maintainer of a Git repository to review code they want to merge into a project.

Steps to be followed:-

1. Find your project on github.
2. Fork it.
3. Clone it to your local system.
4. Make a new branch.
5. Make your changes.
6. Push it back to your repo.
7. Click the Compare & pull request button.
8. Click Create pull request to open a new pull request.





**Network Graph**

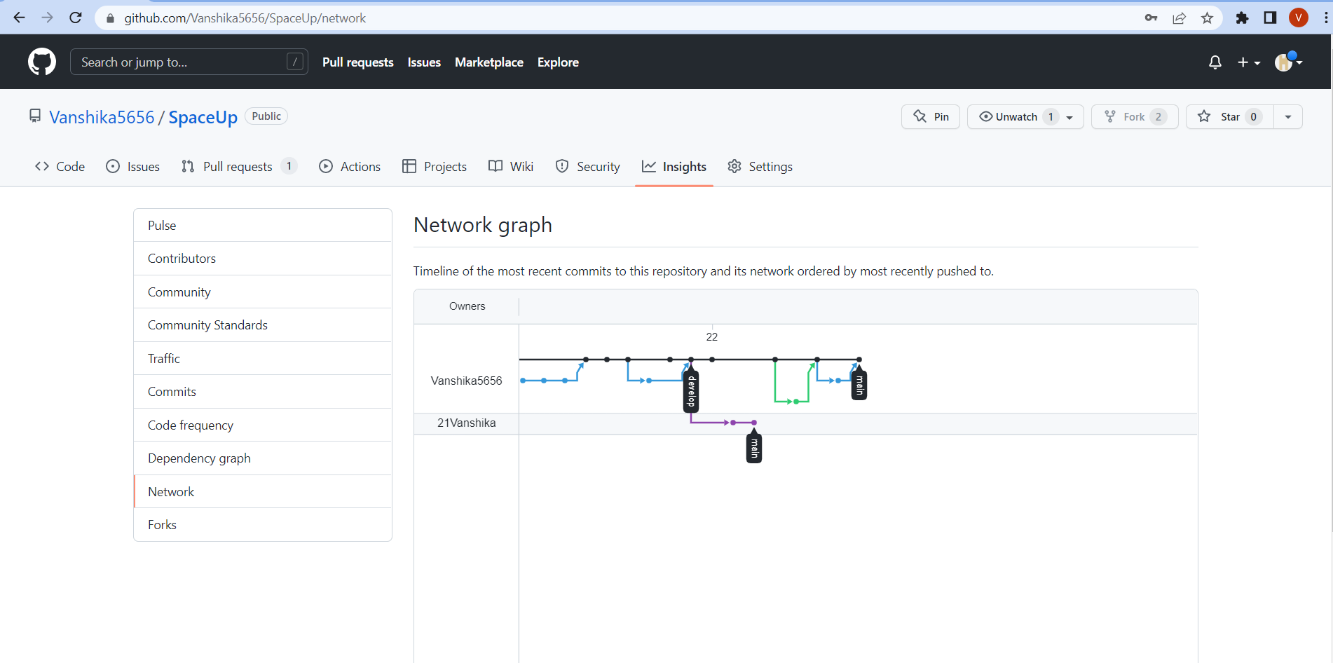
You can better understand the connections that exist between repositories by viewing a repository's network and forks and the projects that depend on the repository.

## *Viewing a repository's network*

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.

## Accessing the network graph

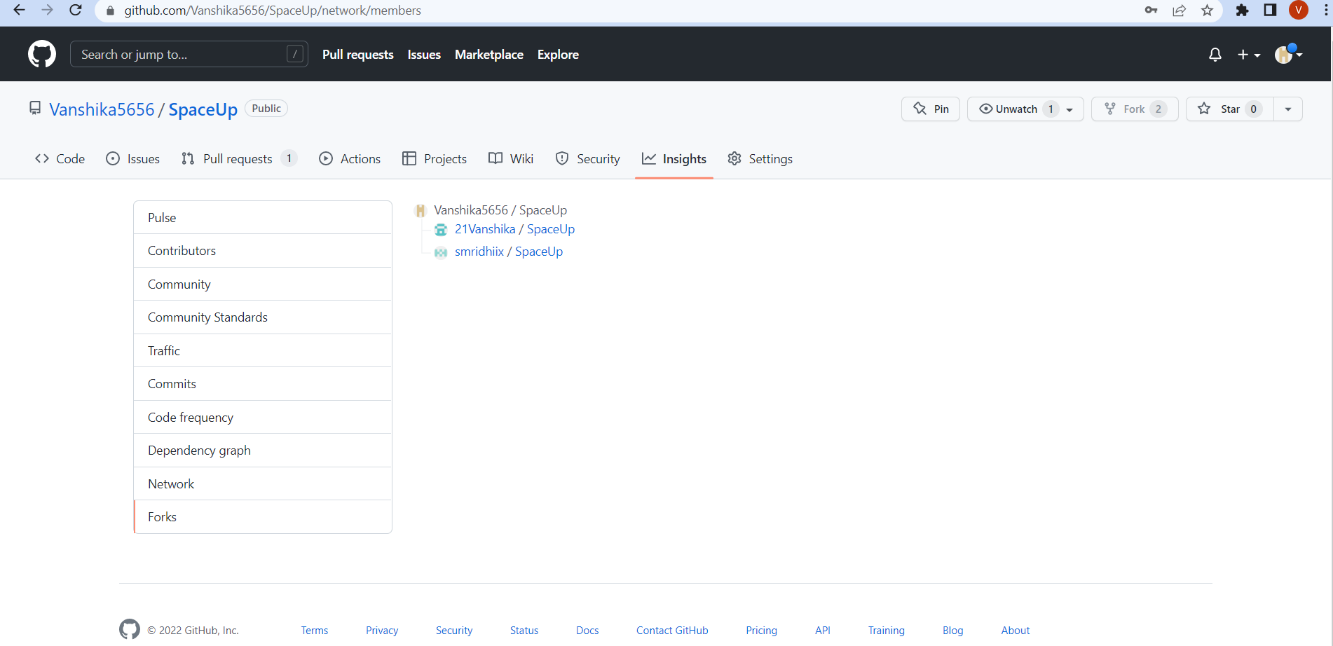
1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click  **Insights**.
3. In the left sidebar, click **Network**.



## Listing the forks of a repository

The Members graph displays all the forks of a repository.

Forks are listed alphabetically by the username of the person who forked the repository. You can click on the username to be redirected to the user's GitHub profile page or click on the fork name to be redirected to the specific fork of the repository.



**7. REFRENCES**

1. https://www.atlassian.com/git

2. <https://www.simplilearn.com/tutorials/git-tutorial>

3. <https://www.geeksforgeeks.org/>

4. <https://github.com/datacamp>

5. https://www.javatpoint.com/git-commands