Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **DCSE**

****

|  |  |  |
| --- | --- | --- |
| **Submitted By**  Deepender Singh  2110990418  G8-B |  | **Submitted To:**  Dr. Monit Kapoor |

## INDEX

## 

**TASK 1.1**

**TASK 1.2**

**TASK 2.0**

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Aim of Experiment | Page No. |  |
| 1 | Setting up of Git client | 5-6 |  |
| 2 | Setting up GitHub account | 7-10 |  |
| 3 | Program to generate log | 11-13 |  |
| 4 | Create and visualize branches | 14-17 |  |
| 5 | Git Lifecycle description | 18-21 |  |
| 6 | Add collaborators on Github repository | 22-25 |  |
| 7 | Fork and Commit | 26-31 |  |
| 8 | Merge and Resolve conflicts created due to own activity and collaborators activity | 32-41 |  |
| 9 | Reset and Revert | 42-49 |  |
| 10 | Project | 49-77 |  |

**INTRODUCTION**

**What is GIT and why is it used?**

Git is a source code management technology used by DevOps. Git is a piece of software that allows you to track changes in any group of files. It is a free and open-source version control system that may be used to efficiently manage small to big projects.

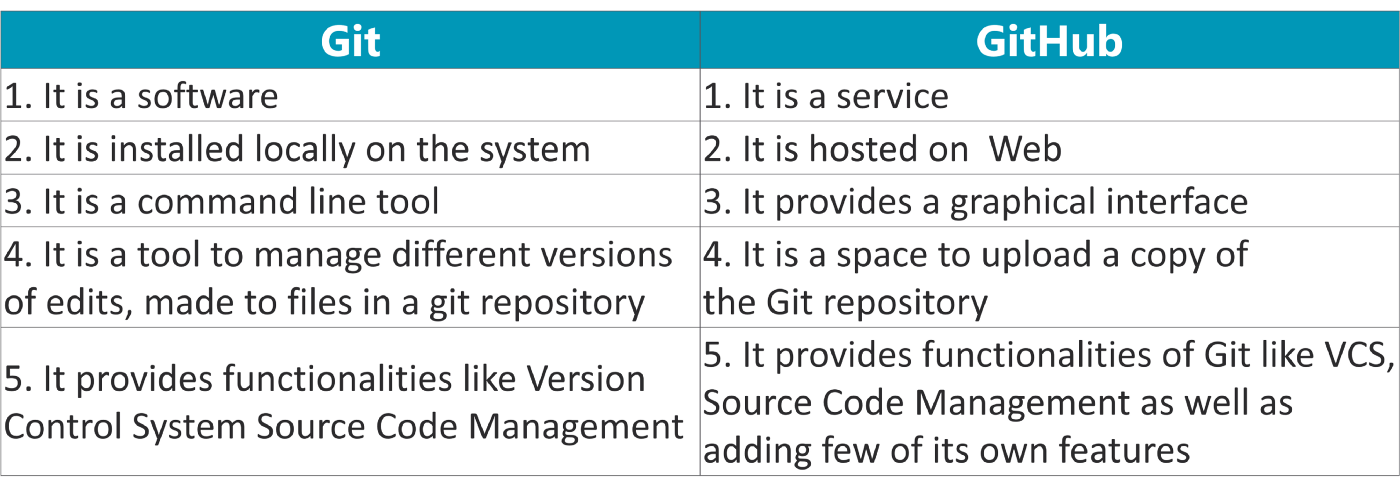
Git is a version control system that allows numerous developers to collaborate on non-linear development projects.

Git is an example of a distributed version control system (DVCS) (hence Distributed Version Control System).

**What is GITHUB?**

GitHub is a version management and collaboration tool for programming. It allows you and others to collaborate on projects from any location.

**What is the difference between GIT and GITHUB?**



**What is Repository?**

A repository stores all of your project's files, as well as the revision history for each one. Within the repository, you may discuss and monitor your project's progress. The.git/ subdirectory within a project is a Git repository. This repository keeps track of any changes made to files in your project over time, creating a history. That is, if you delete the.git/ subdirectory, you are also deleting the history of your project.

**What is Version Control System (VCS)?**

Version Control Systems are the software tools for tracking/managing all the changes made to the source code during the project development. It keeps a record of every single change made to the code. It also allows us to turn back to the previous version of the code if any mistake is made in the current version. Without a VCS in place, it would not be possible to monitor the development of the project.

## Types of VCS

* Local Version Control System
* Centralized Version Control System
* Distributed Version Control System

#### **Local Version Control System:** Local Version Control System is located in your local machine. If the local machine crashes, it would not be possible to retrieve the files, and all the information will be lost. If anything happens to a single version, all the versions made after that 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.

1. **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.

**Experiment No. 01**

**Aim:**Setting up of Git Client

**Theory:**

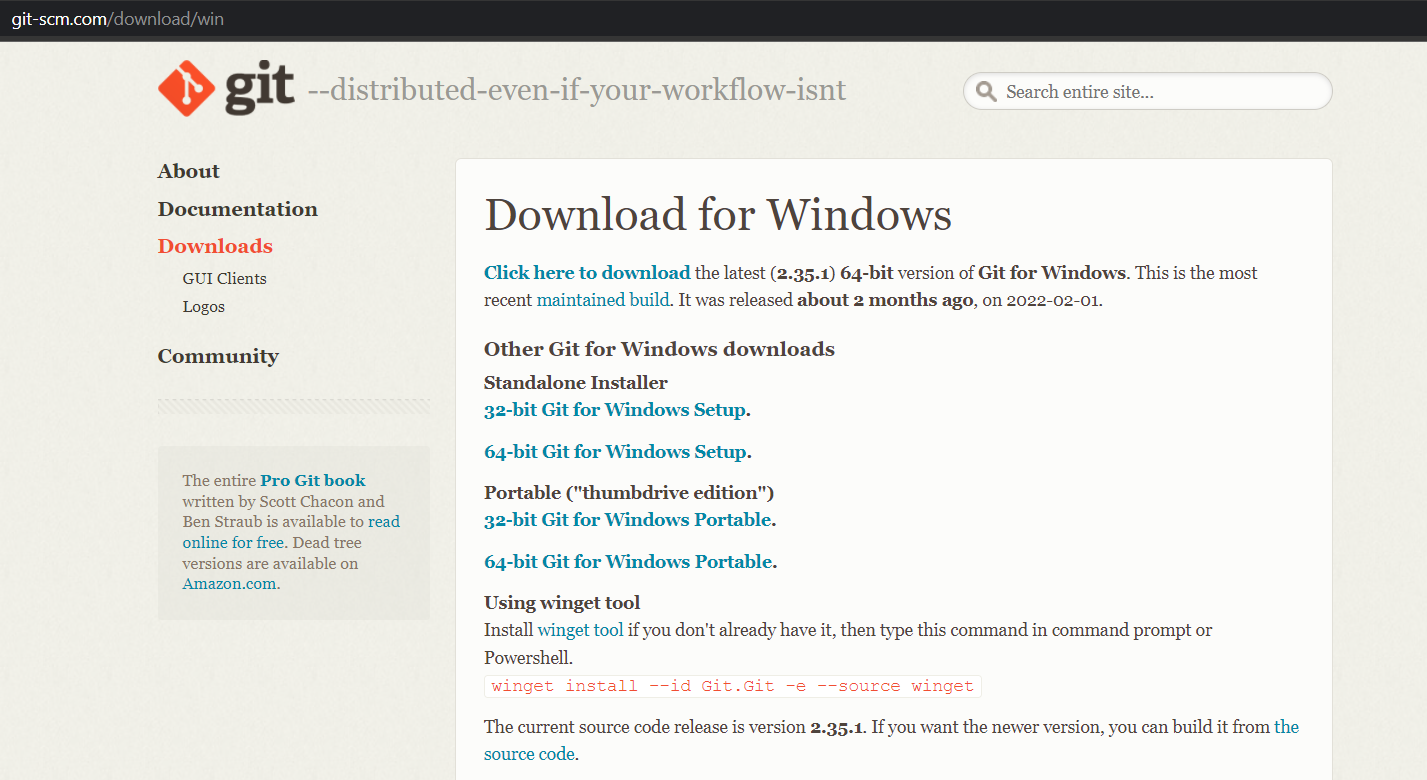
GIT –> It is basically used for pushing and pulling of code. We can use git and git-hub parallelly to work with multiple members or individually. We can make , edit , recreate ,copy or download any code on git hub using git.

What is GIT ? –> It’s a Version Control System(VCS) -> It is a software or we can say a server by which we are able to track all the previous changes in the code.

Advantages of GIT –>

**Procedure:**We can install Git on Windows, using the most official build which is available for download on the GIT’s official website or by just typing ( s c m git ) on any search engine . We can go on<https://git-scm.com/download/win>and can select the platform and bit-versionto download. And after clicking on your desired bit-version or ios it will start downloading automatically.

**Snapshots of download:**

****



**Experiment No. 02**

**Aim:**Setting up GitHub Account

**Theory:**

**What is GitHub ->**GitHub is a website and cloud-based service (client) that helpsan individual or a developers to store and manage their code. We can also trackas well as control changes to our or public code.

**Advantages of GitHub ->**GitHub’s has a user-friendly interface and is easy to use .We can connect the git-hub and git but using some commands shown below in figure 001. Without GitHub we cannot use Git because it generally requires a hostand if we are working for a project we need to share it will our team members, which can only be done by making a repository .Additionally, anyone can sign up and host a public code repository for free, which makes GitHub especially popular with open-source projects.

**Procedure:-**

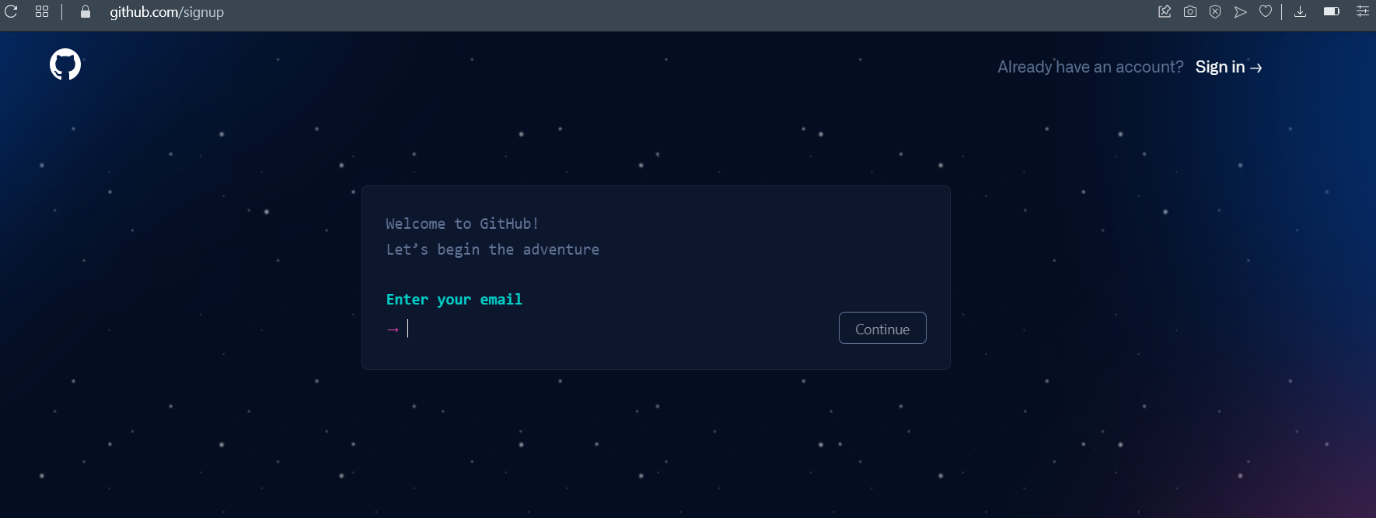
**Step1 :-**

Google (any search engine)

Search for git-hub or (<https://github.com/signup>).

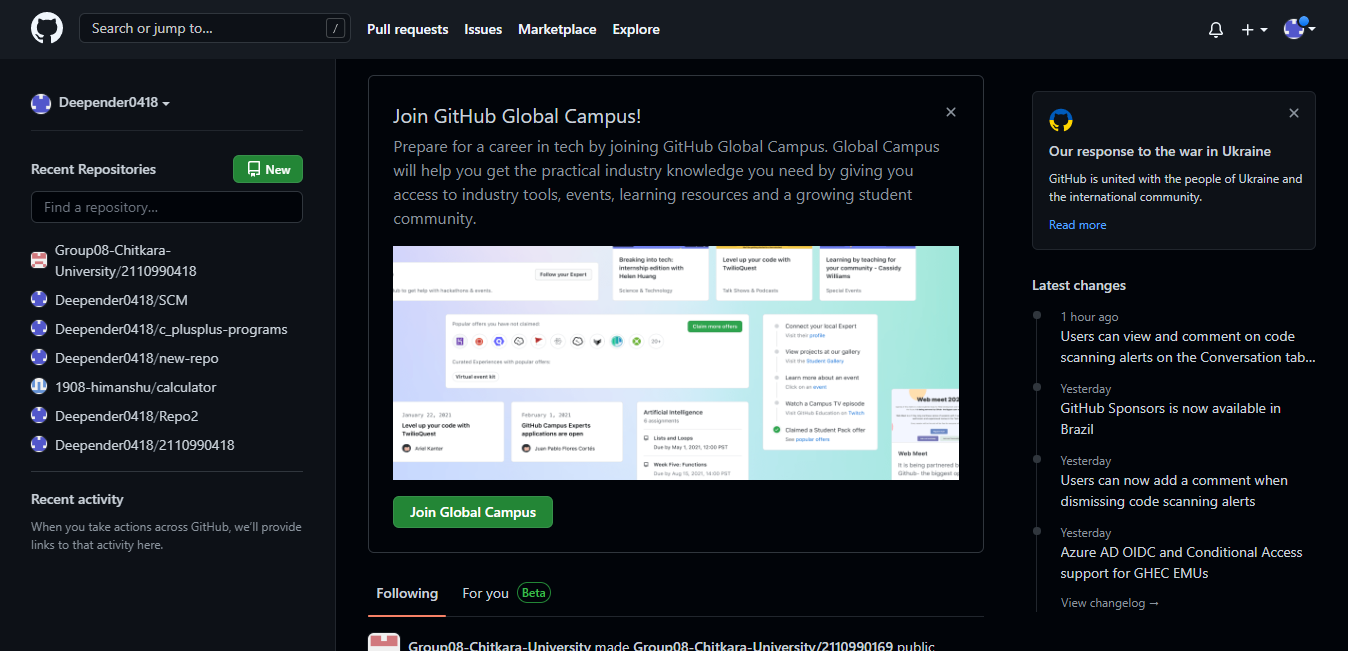
**Step2 :-**

**Snapshots** –



After visiting the link this type of interface will appear, if you already have account you can sign in and if not you can create.

**Interface of GitHub :-**



**To link GitHub account with Git bash –**

**For username:-**

git config --global user.name “username in git-hub”

**For user email:-**

git config --global user.email “your email in git-hub”

**To verify:-**

git config user.name

git config user.email

**Experiment No. 03**

**Aim:** Program to Generate log

**Theory:-**

**Logs ->** Logs are nothing but the history which we can see in git by using the code git log.

It contains all the past commits, insertions and deletions in it which we can see any time.

## Why logs -> Logs helps to check that what were the changes in the code or any other file and by whom. It also contains the number of insertions and deletions including at which time it was changed. First of all create a local repository using Git. For this, you have to make a folder in your device, right click and select “Git Bash Here”. This opens the Git terminal. To create a new local repository, use the command “gitinit” and it creates a folder.git

* When we use GIT for the first time, we have to give the user name and email so that if I am going to change in project, it will be visible to all.

For this, we use command 🡪

**“git config --global user.name *Name*”**

**“git config --global user.email*email*”**

For verifying the user’s name and email, we use 🡪

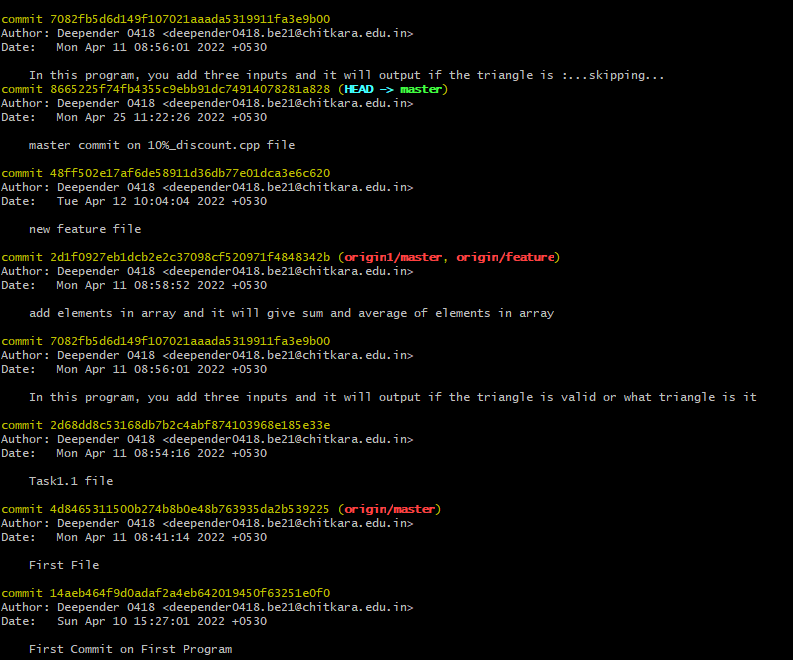
**“git config --global user.name”**

**“git config --global user.email”**

**Some Important Commands:**

* **ls🡪**It gives the file names in the folder.
* **ls -lart🡪**Gives the hidden files also.
* **git status🡪**Displays the state of the working directory and the staged snapshot.
* **touch filename 🡪**This command creates a new file in the repository.
* **Clear 🡪**It clears the terminal.
* **rm -rf .git🡪**It removes the repository.
* **git log 🡪**displays all of the commits in a repository's history
* **git diff 🡪**It compares my working tree to staging area.
* Now, we have to create some files in the repository. Suppose we created index.html Now type git status:  
    
  **git log:**The git log command displays a record of the commits in a Git repository. By default, the git log command displays a commit hash, the commit message, and other commit metadata.

**Snapshots –**

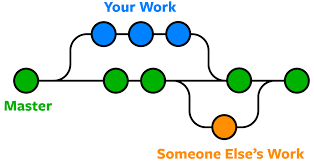
****

**Experiment No. 04**

**Aim:** Create and visualize branches

**Create branches :-**

* The main branch in git is called as master branch. But we can make branches out of this main master branch. All the files present in master can be shown in branch but the file which are created in branch are not shown in master branch. We can also merge both the parent (master) and child (other branches).  
    
  **Branching:** A branch in Git is an independent line of work(a pointer to a specific commit). It allows users to create a branch from the original code (master branch) and isolate their work. Branches allow you to work on different parts of a project without impacting the main branch.



Let us see the command of it:

Firstly, add a new branch, let us suppose the branch name is activity1.

For this use command 🡪

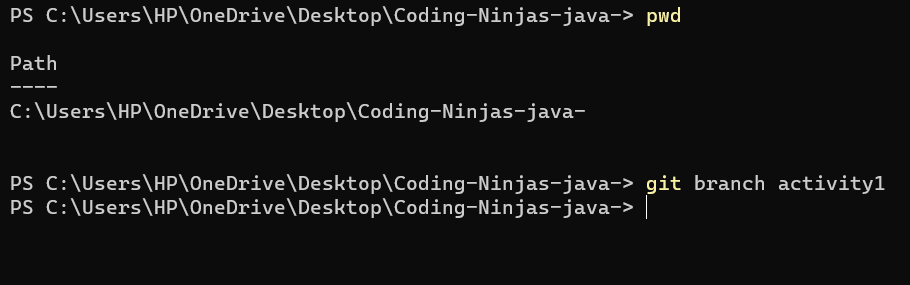
* **git branch name [**adding new branch**]**
* **git branch [**use to see the branch’s names**]**
* **git checkout *branch name* [**use to switch to the given branch**]**
* In this you can see that firstly ‘git branch’ shows only one branch in green colour but when we add a new branch using ‘git branch act1’, it shows 2 branches but the green colour and star is on master. So, we have to switch to act1 by using ‘git checkout act1’. If we use ‘git branch’, now you can see that the green colour and star is on act1. It means you are in activity1 branch and all the data of master branch is also on act1 branch. Use “ls” to see the files.
* Now add a new file in activity1 branch, do some changes in file and commit the file.

**Syntax:-**

1. For creating a new branch.

git branch name of branch

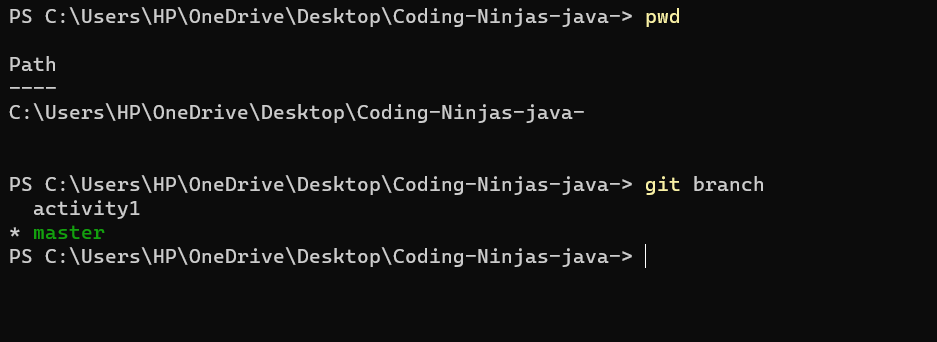
**Snapshots –**

****

1. We can also check how many branches we have.

git branch

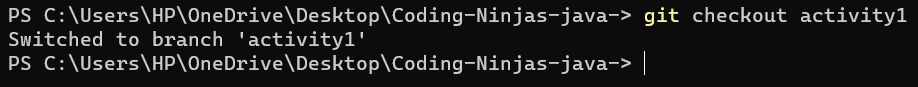
**Snapshots :-**



1. To change the present working branch.

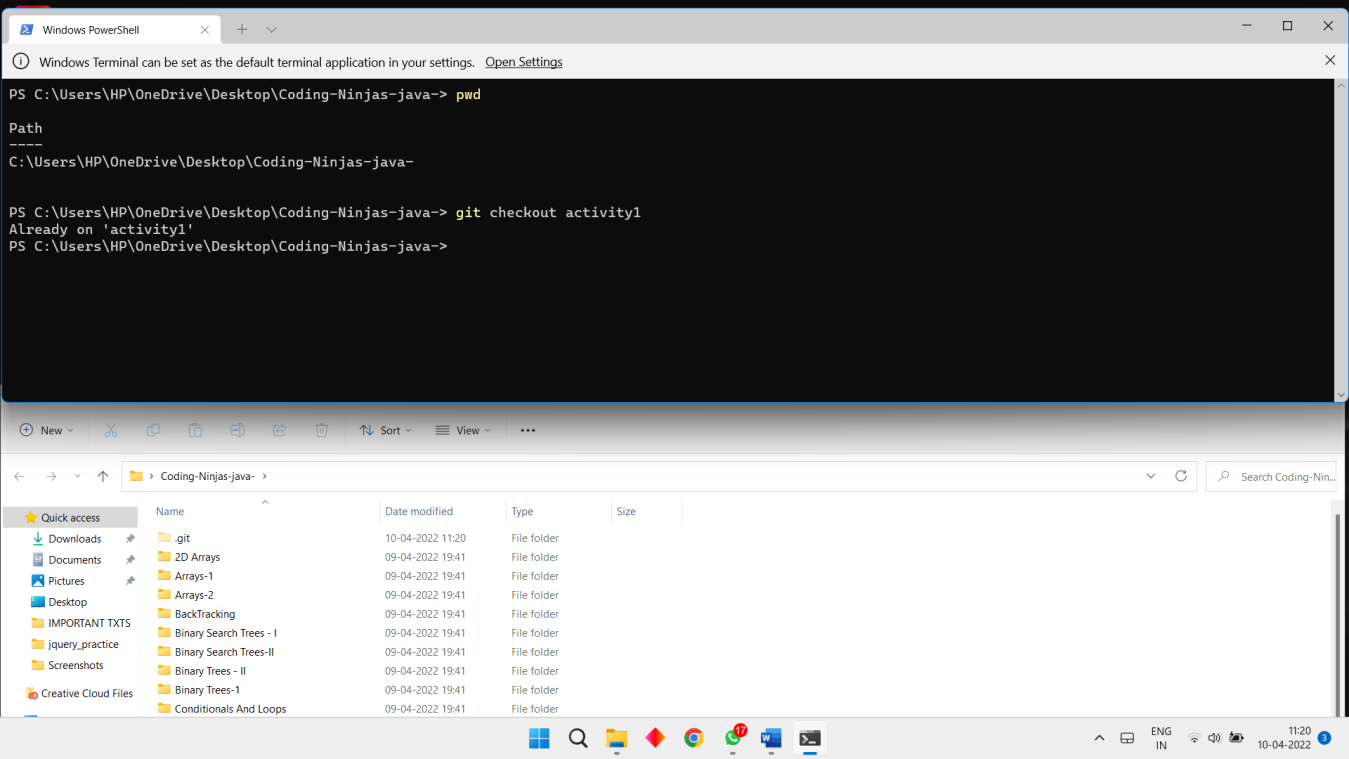
git checkout name of branch.

**Snapshots –**

****

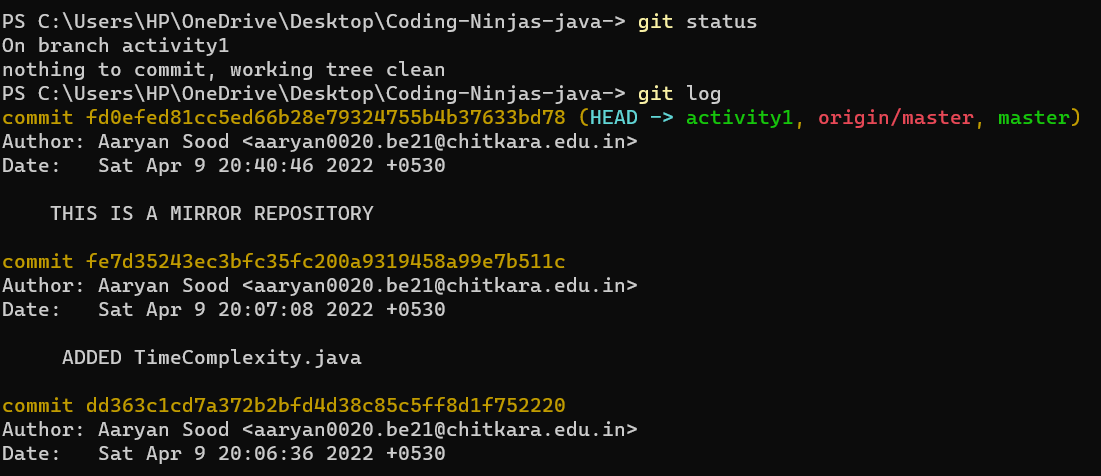
**Visualizing branches :-**

To visualize I have created a new file in a new branch activity 1 instead of master branch.



After this I have done the 3 step architecture which is tracking the file , send it to stagging

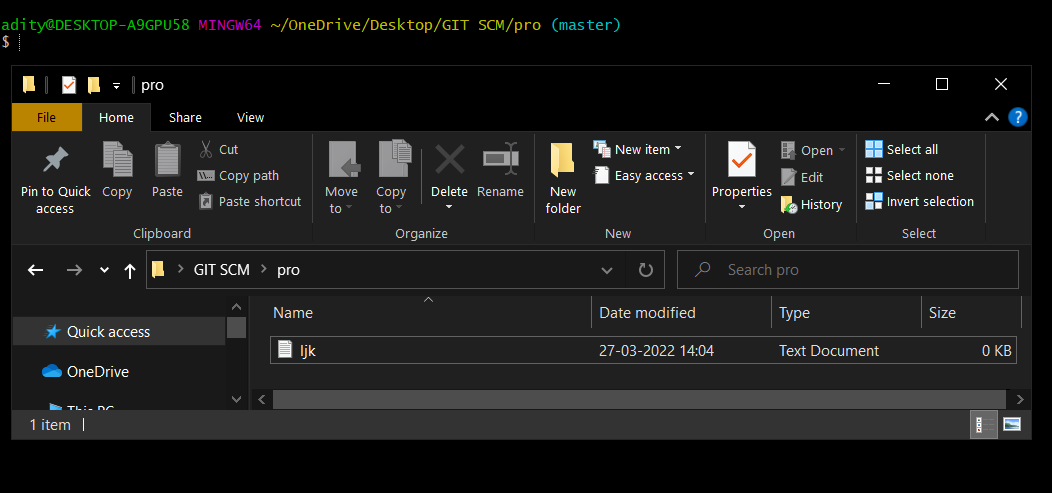
area and finally we can role back to any previously saved version of this file.



After this we will change the branch from activity1 to master, but when I will switch to the

master branch there will not be the same file in the master , it will not show the new file in

the master branch.



In this way we can create and change different branches . We can also merge the branches

by using git merge command.

**Experiment No. 05**

**Aim:** Git lifecycle description

**Theory:**

**Stages in GIT Life Cycle->**Files in a Git project have various stages like Creation, Modification, Refactoring, and Deletion and so on. Irrespective of whether this project is tracked by Git or not, these phases are still prevalent. However, when a project is under Git version control system, they are present in three major Git states in addition to these basic ones. Here are the three Git states:

* Working directory
* Staging area
* Git directory

**Working Directory ->**

Consider a project residing in your local system. This project may or may not be tracked by Git. In either case, this project directory is called your Working directory.

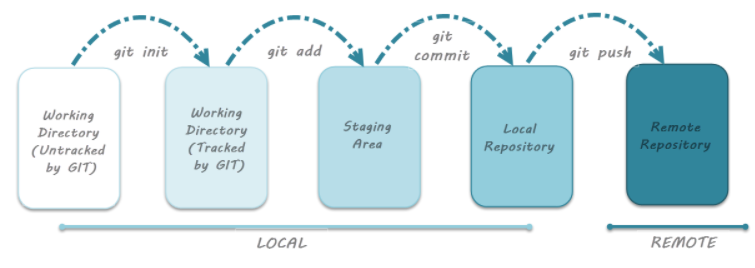
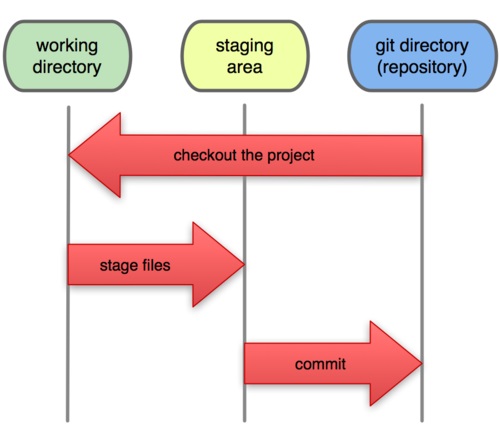
### **Staging Area ->**

**Staging area is the playground where you group, add and organize the files to be committed to Git for tracking their versions.**

**Git Directory ->**

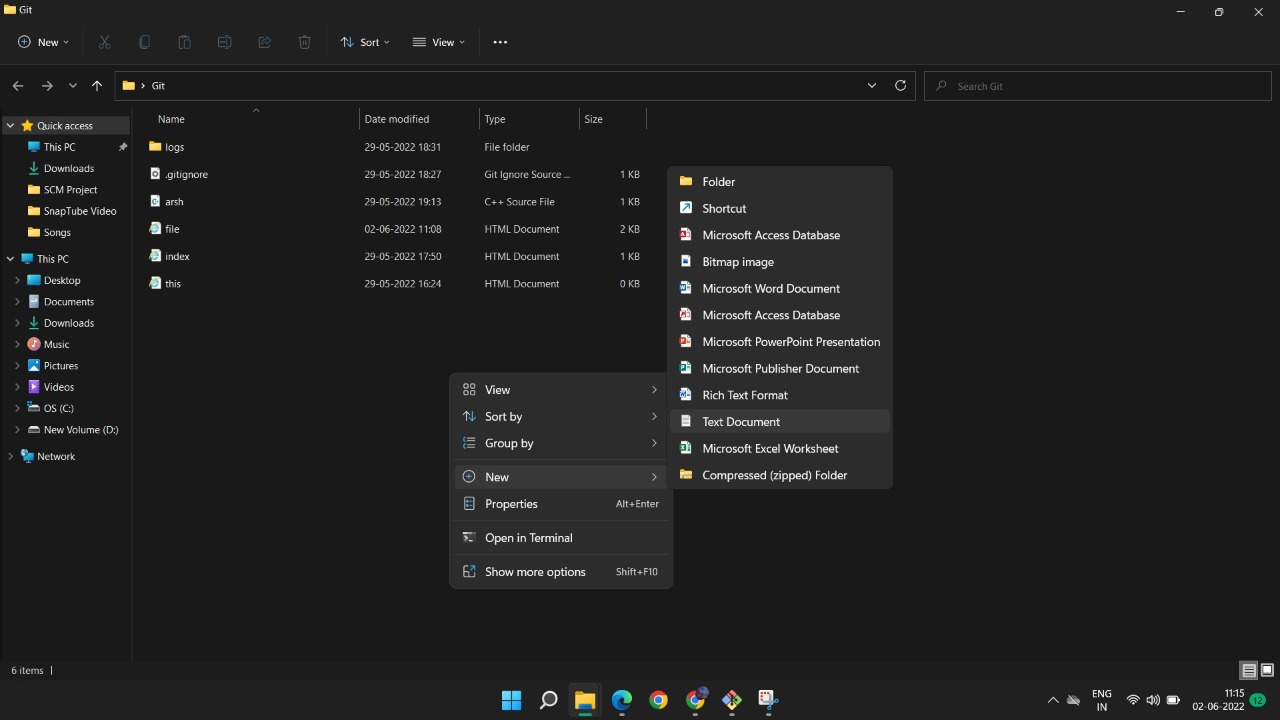
Now that the files to be committed are grouped and ready in the staging area, we can commit these files. So, we commit this group of files along with a commit message explaining what is the commit about. Apart from commit message, this step also records the author and time of the commit. Now, a snapshot of the files in the commit is recorded by Git. The information related to this commit is stored in the Git directory.

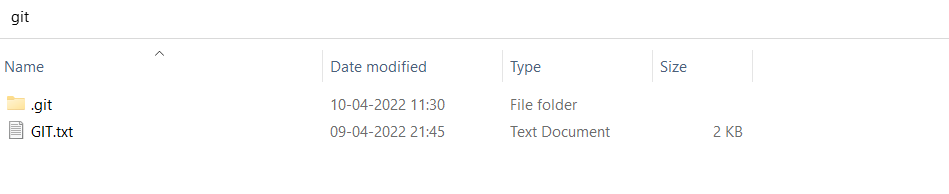
**Remote Repository-> means mirror or clone of the local Git repository in GitHub**. And **pushing means uploading the commits from local Git repository to remote repository hosted in GitHub.**

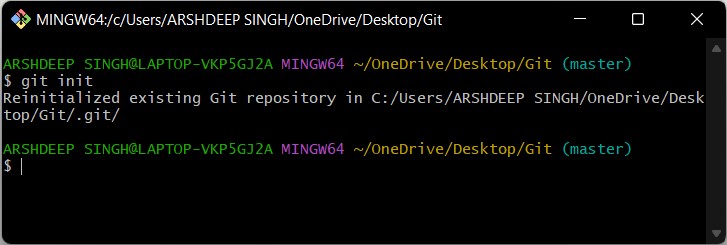
****

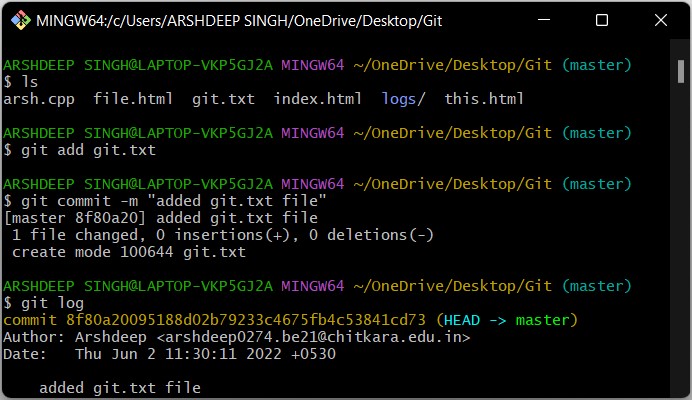
**Snapshots –**

****

****

****

****

****

****

**Experiment No. 06**

**Aim:** Add collaborators on GithubRepositoy

1. Ask for the username of the person you're inviting as a collaborator. If theydon'thaveausernameyet,theycansignupforGitHubFormoreinformation
2. OnGitHub.com,navigatetothemainpageoftherepository
3. Underyourrepositoryname,click**Settings**
4. Inthe"Access"sectionofthesidebar,click**Collaborators&teams**.
5. Click**Inviteacollaborator**..
6. In the search field, start typing the name of person you want to invite, thenclickaname inthelist of matches

**Experiment No. 07**

**Aim:** Fork and Commit

# **Aboutforks**

# Mostcommonly,forksareusedtoeitherproposechangestosomeoneelse'sprojectorto use someone else's project as a starting point for your own idea. You can fork arepository to create a copy of the repository and make changes without affecting theupstreamrepository.Formoreinformation,see"[Workingwithforks](https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/working-with-forks)."

## Proposechangestosomeoneelse's project

## For example, you can use forks to propose changes related to fixing a bug. Rather thanloggingan issueforabug you'vefound,youcan:

* Forktherepository.
* Makethefix.
* Submitapullrequesttotheprojectowner.

**Usesomeoneelse'sprojectasastartingpointforyourownidea**

Open source software is based on the idea that by sharing code, we can make better,more reliable software. For more information, see the "[About the Open Source Initiative](http://opensource.org/about)"ontheOpenSourceInitiative.Formoreinformationaboutapplyingopensourceprinciplestoyourorganization'sdevelopment work on GitHub.com, see GitHub's white paper "[An introduction toinner source](https://resources.github.com/whitepapers/introduction-to-innersource/)."

Whencreatingyourpublicrepositoryfromaforkofsomeone'sproject,makesuretoinclude a license file that determines how you want your project to be shared withothers. For more information, see "[Choose an open source license](https://choosealicense.com/)" atchoosealicense.com.

For more information on open source, specifically how to create and grow an opensource project, we've created [Open Source Guides](https://opensource.guide/) that will help you foster a healthyopensourcecommunitybyrecommendingbestpracticesforcreatingandmaintainingrepositories foryouropensourceproject.Youcanalsotakeafree[GitHubLearningLab](https://lab.github.com/)courseonmaintainingopensourcecommunities.

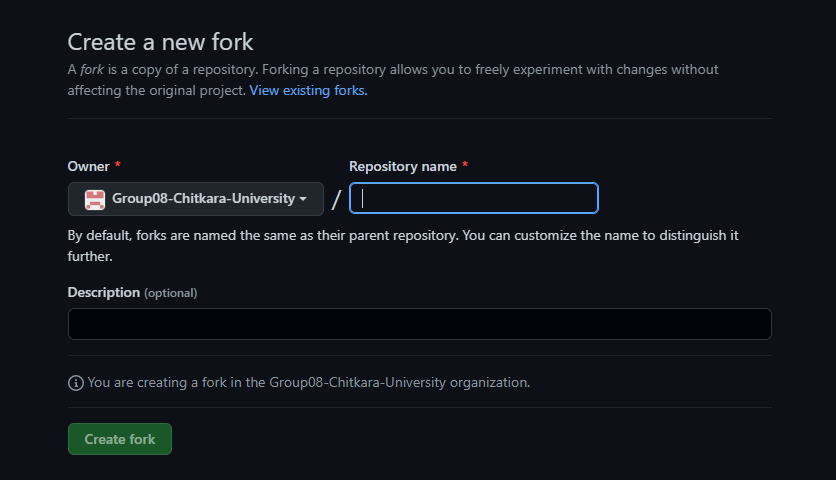
# **Prerequisites**

If you haven't yet, you should first [set up Git.](https://docs.github.com/en/articles/set-up-git) Don't forget to [set up authentication toGitHub.comfromGit](https://docs.github.com/en/articles/set-up-git#next-steps-authenticating-with-github-from-git) aswell.

# **Forkingarepository**

# Youmightforkaprojecttoproposechangestotheupstream,ororiginal,repository.Inthis case, it's good practice to regularly sync your fork with the upstream repository. Todo this, you'll need to use Git on the command line. You can practice setting theupstreamrepositoryusingthesame[octant/Spoon-Knife](https://github.com/octocat/Spoon-Knife)repositoryyoujustforked.

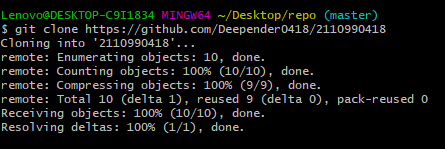
* OnGitHub.com,navigatetothe[octant/Spoon-Knife](https://github.com/octocat/Spoon-Knife)repository.
* Inthetop-rightcornerofthepage,click Fork.



# **Cloningyourforkedrepository**

Right now, you have a fork of the Spoon-Knife repository, but you don't have the files inthatrepositorylocally onyour computer.

* OnGitHub.com,navigateto yourforkoftheSpoon-Kniferepository.
* Above the list of files, click Code
* To clone the repository using HTTPS, under "Clone with HTTPS", click. To clonethe repository using an SSH key, including a certificate issued by yourorganization'sSSHcertificateauthority,click**UseSSH**,thenclick. Toclonea repositoryusingGitHubCLI,click**UseGitHubCLI**,thenclick.



* OpenGitBash.
* Changethecurrentworkingdirectorytothelocationwhereyouwantthecloneddirectory.
* Typegitclone,andthenpastetheURLyoucopiedearlier.Itwilllooklikethis,withyour GitHubusernameinstead of YOUR-USERNAME:
* $gitclonehttps://github.com/*YOUR-USERNAME*/Spoon-Knife
* PressEnter.Yourlocalclonewillbecreated.
* $gitclonehttps://github.com/*YOUR-USERNAME*/Spoon-Knife
* >Cloninginto`Spoon-Knife`...
* >remote:Countingobjects:10,done.
* >remote:Compressingobjects:100%(8/8),done.
* >remove:Total10(delta1),reused10(delta1)
* >Unpackingobjects:100%(10/10),done.

# Configuring Git to sync your fork with the originalrepository

# When you fork a project in order to propose changes to the original repository, you canconfigure Git to pull changes from the original, or upstream, repository into the localcloneofyour fork.

1. OnGitHub.com,navigatetothe[octocat/Spoon-Knife](https://github.com/octocat/Spoon-Knife)repository.
2. Abovethelistoffiles,click Code
3. To clone the repository using HTTPS, under "Clone with HTTPS", click . To clonethe repository using an SSH key, including a certificate issued by yourorganization'sSSHcertificateauthority,click**UseSSH**,thenclick.Toclonea repositoryusingGitHubCLI,click**UseGitHubCLI**,then click on
4. Open Git Bash.
5. Changedirectoriestothelocationoftheforkyoucloned.
   * To gotoyour homedirectory,typejustcdwithnoother text.
   * Tolistthefilesandfoldersinyourcurrentdirectory,type ls.
   * Togointooneofyourlisteddirectories,typecdyour\_listed\_directory.
   * Togo up onedirectory,typecd...
6. Typegitremote-vand press Enter. You'llseethecurrentconfiguredremoterepositoryforyour fork.
7. $gitremote-v
8. >originhttps://github.com/YOUR\_USERNAME/YOUR\_FORK.git(fetch)

>originhttps://github.com/YOUR\_USERNAME/YOUR\_FORK.git(push)

9. Typegitremoteaddupstream,andthenpastetheURLyoucopiedinStep2andpressEnter. It willlooklikethis:

$gitremoteaddupstreamhttps://github.com/octocat/Spoon-Knife.git

1. To verify the new upstream repository you've specified for your fork, type gitremote-vagain.You shouldsee theURLforyour fork asorigin, and theURLfortheoriginalrepositoryasupstream.
2. $gitremote-v
3. >origin https://github.com/YOUR\_USERNAME/YOUR\_FORK.git(fetch)
4. >origin https://github.com/YOUR\_USERNAME/YOUR\_FORK.git(push)
5. >upstreamhttps://github.com/ORIGINAL\_OWNER/ORIGINAL\_REPOSITORY.git(fetch)
6. >upstreamhttps://github.com/ORIGINAL\_OWNER/ORIGINAL\_REPOSITORY.git(push)

Now,youcankeepyourforksyncedwiththeupstreamrepositorywithafewGitcommands.For moreinformation, see"[Syncingafork](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/syncing-a-fork)."

## Nextsteps

## Youcanmakeanychangestoafork,including:

* Creatingbranches:[*Branches*](https://docs.github.com/en/articles/creating-and-deleting-branches-within-your-repository)allowyoutobuildnewfeaturesortestoutideaswithoutputting yourmainprojectat risk.
* Opening pull requests: If you are hoping to contribute back to the originalrepository,youcansendarequesttotheoriginalauthortopullyourforkintotheirrepositorybysubmitting a [pullrequest.](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests)

# **Find another repository to fork**

# Fork a repository to start contributing to a project. You can fork a repository to youruser account or any organization where you have repository creation permissions. Formoreinformation, see "[Rolesinanorganization](https://docs.github.com/en/organizations/managing-peoples-access-to-your-organization-with-roles/roles-in-an-organization)."Ifyouhaveaccesstoa privaterepositoryandtheownerpermitsforking,youcanforkthe repository to your user account or any organization on GitHub Team where youhave repository creation permissions. You cannot fork a private repository to anorganizationusingGitHubFree.Formoreinformation,see"[GitHub'sproducts](https://docs.github.com/en/articles/githubs-products)."

You can browse [Explore](https://github.com/explore)to find projects and start contributing to open sourcerepositories.Formoreinformation,see"[FindingwaystocontributetoopensourceonGitHub](https://docs.github.com/en/github/getting-started-with-github/finding-ways-to-contribute-to-open-source-on-github)."

# **Celebrate**

You have now forked a repository, practiced cloning your fork, and configured anupstreamrepository.Formoreinformationaboutcloningtheforkandsyncingthechangesinaforkedrepositoryfromyour computersee"[Set upGit](https://docs.github.com/en/articles/set-up-git)."Youcanalsocreateanewrepositorywhereyoucanput allyour projectsand sharethecodeonGitHub.For moreinformationsee, "[Createarepository](https://docs.github.com/en/articles/create-a-repo)."

Each repository in GitHub is owned by a person or an organization. You can interactwith the people, repositories, and organizations by connecting and following them onGitHub.For moreinformationsee"[Be social](https://docs.github.com/en/articles/be-social)."GitHub has a great support community where you can ask for help and talk to peoplefromaroundtheworld.Jointheconversationon[GithubSupportCommunity.](https://github.community/)

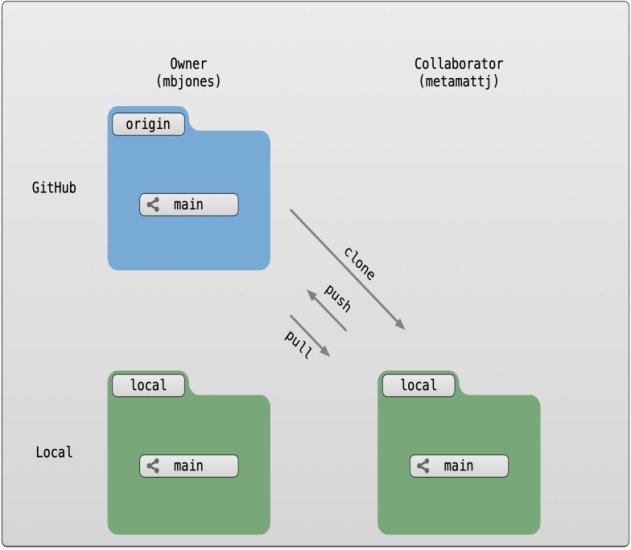
**Experiment No. 08**

**Aim:** Merge and Resolve conflicts created due to own activity and collaborators activity

Git is a great tool for working on your own, but even better for working withfriends and colleagues. Git allows you to work with confidence on your own local copy of files with the confidence that you will be able to successfully synchronize your changes with the changes made byothers.

The simplest way to collaborate with Git is to use a shared repository on a hosting service such as [GitHub,](https://github.com/) and use this shared repository as the mechanism to move changes from one collaborator to another. While there are other more advanced ways to sync git repositories, this “hub and spoke” model works really well due to itssimplicity.

Inthis model,thecollaboratorwillcloneacopyoftheowner’srepositoryfromGitHub, andthe ownerwillgrantthemcollaboratorstatus,enablingthecollaboratortodirectlypullandpushfrom the owner’s GitHubrepository.



#### **Collaborating with a trusted colleague without conflicts**

We start by enabling collaboration with a trusted colleague. We will designate the Owner as the person who owns the shared repository, and the Collaborator as the person that they wish to grant the ability to make changes to their reposity. We start by giving that person access to our GitHub repository.

We will start by having the collaborator make some changes and share those with the Owner without generating any conflicts, In an ideal world, this would be the normal workflow. Here are the typical steps.

##### **Step 1: Collaboratorclone**

##### To be able to contribute to a repository, the collaborator must clone the repository fromthe Owner’s github account.

##### To do this, the Collaborator should visit the github page for the Owner’s repository, and then copy the clone URL.

##### In R Studio, the Collaborator will create a new project from version control by pasting this clone URL into the

##### appropriate dialog (see the earlier chapter introducing GitHub).

##### **Step 2: CollaboratorEdits**

With a clone copied locally, the Collaborator can now make changes to theindex.Rmd file in the repository, adding a line or statement somewhere noticeable near the top. Save your changes.

##### **Step 3: Collaborator commit andpush**

To sync changes, the collaborator will need to add, commit, and push their changes to the Owner’s repository. But before doing so, its good practice to pull immediately before committing to ensure you have the most recent changes from the owner. So, in R Studio’s Git tab, first click the “Diff” button to open the git window, and then press the green “Pull” down arrow button. This will fetch any recent changes from the origin repository and merge them. Next, add the changed index.Rmd file to be committed by clicking the checkbox next to it, type in a commit message, and click ‘Commit.’ Once that finishes, then the collaborator can immediately click ‘Push’ to send the commits to the Owner’s GitHub repository.

##### **Step 4: Ownerpull**

Now, the owner can open their local working copy of the code in RStudio, and pull those changes down to their local copy.Congrats, the owner now has your changes!

##### **Step 5: Owner edits, commit, andpush**

Next, the owner should do the same. Make changes to a file in the repository, save it, pull to make sure no new changes have been made while editing, and then add, commit, and push the Owner changes to GitHub.

##### **Step 6: Collaboratorpull**

The collaborator can now pull down those owner changes, and all copies are once again fully synced. And you’re off to collaborating.

**Challenge**

Now that the instructors have demonstrated this conflict-free process, break into pairs and try the same with your partner. Start by designating one person as the Owner and one as the Collaborator, and then repeat the steps described above:

* Step 0: Setup permissions for yourcollaborator
* Step 1: Collaborator clones the Ownerrepository
* Step 2: Collaborator Edits the READMEfile
* Step 3: Collaborator commits and pushes the file toGitHub
* Step 4: Owner pulls the changes that the Collaboratormade
* Step 5: Owner edits, commits, and pushes some newchanges
* Step 6: Collaborator pulls the owners changes fromGitHub

#### **Mergeconflicts**

#### 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 gitwill 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 sameversion.

The main problem with merge conflicts is that, when the Owner and Collaborator both makechanges to the same line of a file, git doesn’t know whose changes take precedence. You have to tell git whose changes to use for that line.

#### **How to resolve aconflict**

##### 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 doing the merge altogether:

gitmerge--abort

Of course, after doing that you stull haven’t synced with your collaborator’s changes, so things are still unresolved. But at least your repository is now usable on your local machine.

##### Checkout

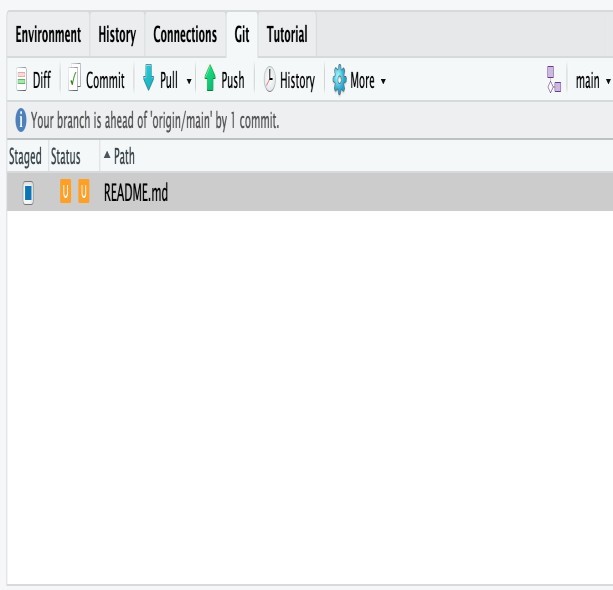
The simplest way to resolve a conflict, given that you know whose version of the file you want to keep, is to use the commandlinegit program to tell git to use either your changes (the person doing the merge), or their changes (the othercollaborator).

* keepyour collaboratorsfile:gitcheckout--theirsconflicted\_file.Rmd
* keepyour ownfile:gitcheckout--oursconflicted\_file.Rmd

Onceyouhaverunthat command,thenrunadd,commit, andpushthechangesasnormal.

##### **Pull and edit the file**

But that requires the commandline. If you want to resolve from RStudio, or if you want to pick and choose some of your changes and some of your collaborator’s, then instead you can manually edit and fix the file. When you pulled the file with a conflict, git notices that there is a conflict and modifies the file to show both your own changes and your collaborator’s changes in the file. It also shows the file in the Git tab with an orange U icon, which indicates that the file is Unmerged, and therefore awaiting you help to resolve the conflict. It delimits these blocks with a series of less than and greater than signs, so they are easy to find:



To resolve the conflicts, simply find all of these blocks, and edit them so that the file looks how you want (either pick your lines, your collaborators lines, some combination, or something altogether new), and save. Be sure you removed the delimiter lines that startedwith <<<<<<<, =======, and >>>>>>>.

Once you have made those changes, you simply add, commit, and push the files to resolve the conflict

##### **Producing and resolving mergeconflicts**

To illustrate this process, we’re going to carefully create a merge conflict step by step, show how to resolve it, and show how to see the results of the successful merge after it is complete. First,we will walk through the exercise to demonstrate theissues.

###### **Owner and collaborator ensure all changes areupdated**

First, start the exercise by ensuring that both the Owner and Collaborator have all of the changes synced to their local copies of the Owner’s repository in Studio. This includes doing a gitpull to ensure that you have all changes local, and make sure that the Git tab in RStudio doesn’t show any changes needing to be committed.

###### **Owner makes a change andcommits**

###### From that clean slate, the Owner first modifies and commits a small change inlcudingtheir name on a specific line of the README.md file (we will change line 4). Work to only change that one line, and add your username to the line in some form and commit the changes (but DO NOT push). We are now in the situation where the owner has unpushed changes that the collaborator can notyetsee.

###### **Collaborator makes a change and commits on the sameline**

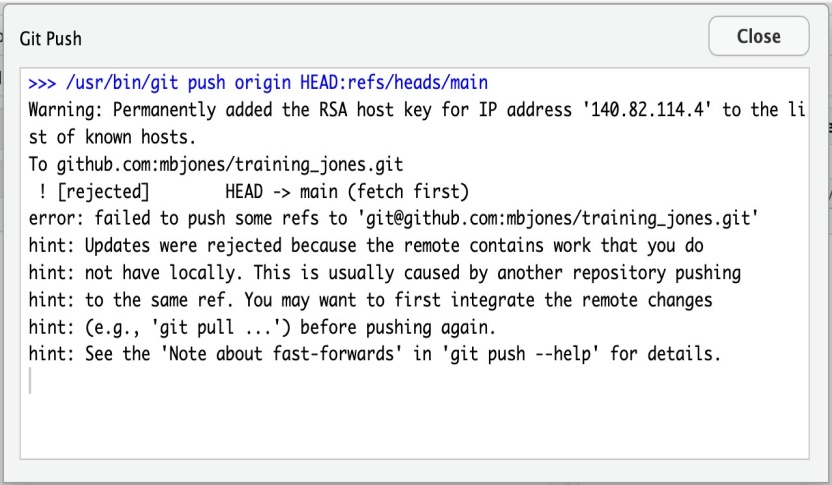
###### Now the collaborator also makes changes to the same (line 4) of the README.md file in their RStudio copy of the project, adding their name to the line. They then commit. At this point, both the owner and collaborator have committed changes based on their shared version of the README.md file, but neither has tried to share their changes via GitHub.

###### **Collaborator pushes the file toGitHub**

###### Sharing starts when the Collaborator pushes their changes to the GitHub repo, which updates GitHub to their version of the file. The owner is now one revision behind, but doesn’t yet know it.

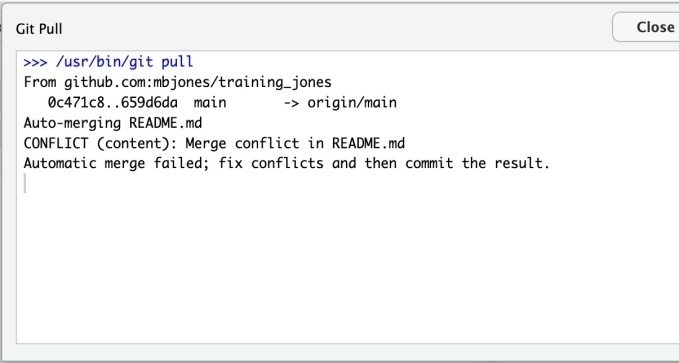
###### **Owner pushes their changes and gets anerror**

###### At this point, the owner tries to push their change to the repository, which triggers an error from GitHub. While the error message is long, it basically tells you everything needed ( that the owner’s repository doesn’t reflect the changes on GitHub, and that they need to pull before they can push).

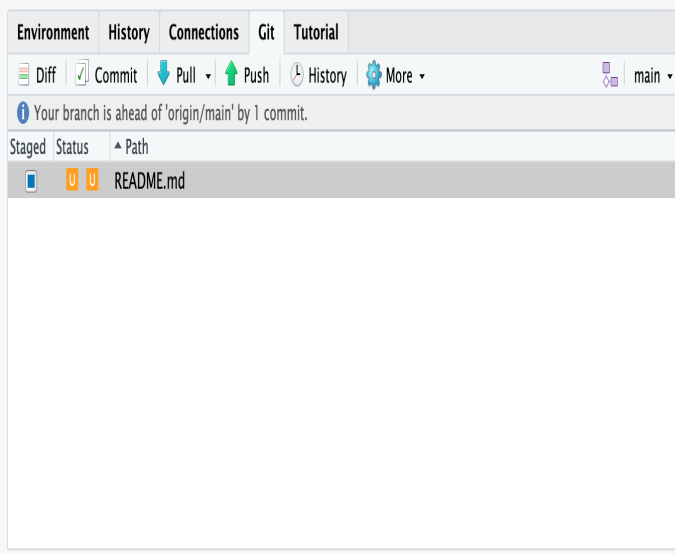


###### Owner pulls from GitHub to get Collaboratorchanges

Doing what the message says, the Owner pulls the changes from GitHub, and gets another, different error message. In this case, it indicates that there is a merge conflict because of the conflicting lines.

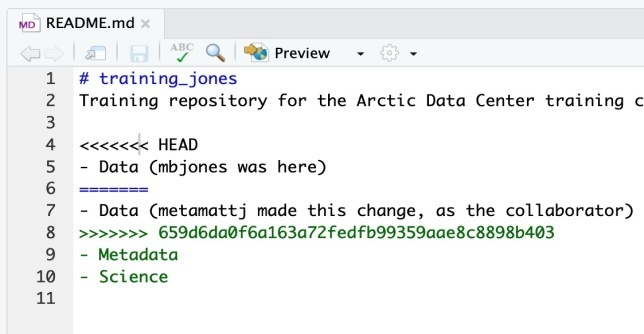


In the Git pane of RStudio, the file is also flagged with an orange ‘U’, which stands for an unresolved merge conflict.

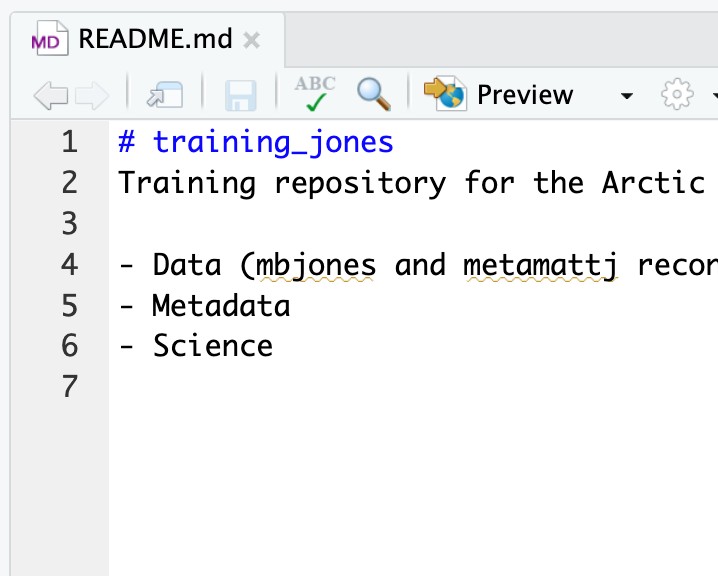


###### **Owner edits the file to resolve theconflict**

To resolve the conflict, the Owner now needs to edit the file. Again, as indicated above, git has flagged the locations in the file where a conflict occcurred with <<<<<<<, =======, and >>>>>>>. The Owner should edit the file, merging whatever changes are appropriate until the conflicting lines read how they should, and eliminate all of the marker lines with with<<<<<<<,=======, and >>>>>>



Of course, for scripts and programs, resolving the changes means more than just merging the text – whoever is doing the merging should make sure that the code runs properly and none of the logic of the program has been broken.

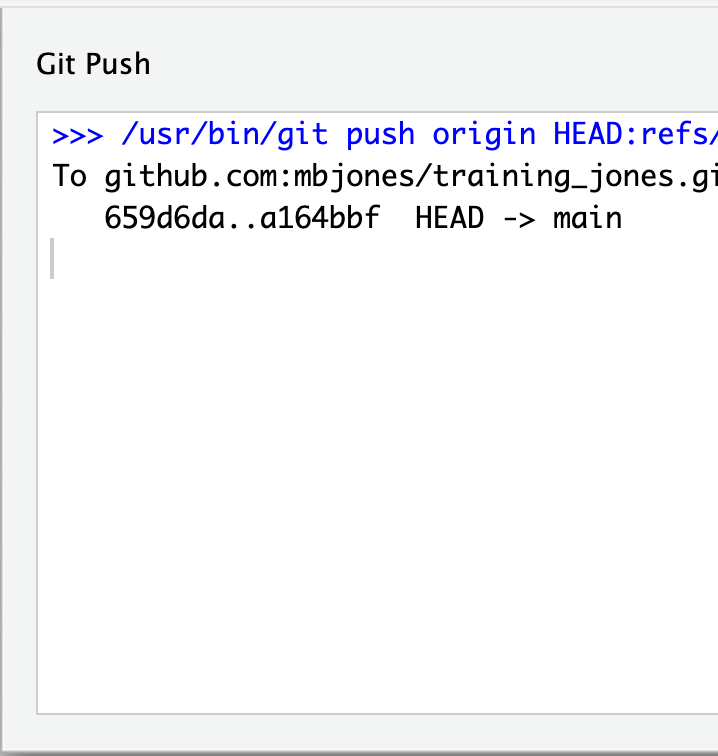


###### **Owner commits the resolvedchanges**

From this point forward, things proceed as normal. The owner first ‘Adds’ the file changes to be made, which changes the orange U to a blue M for modified, and then commits the changes locally. The owner now has a resolved version of the file on their system.

###### Owner pushes the resolved changes toGitHub

Have the Owner push the changes, and it should replicate the changes to GitHub without error.



###### **Collaborator pulls the resolved changes fromGitHub**

Finally, the Collaborator can pull from GitHub to get the changes the owner made.

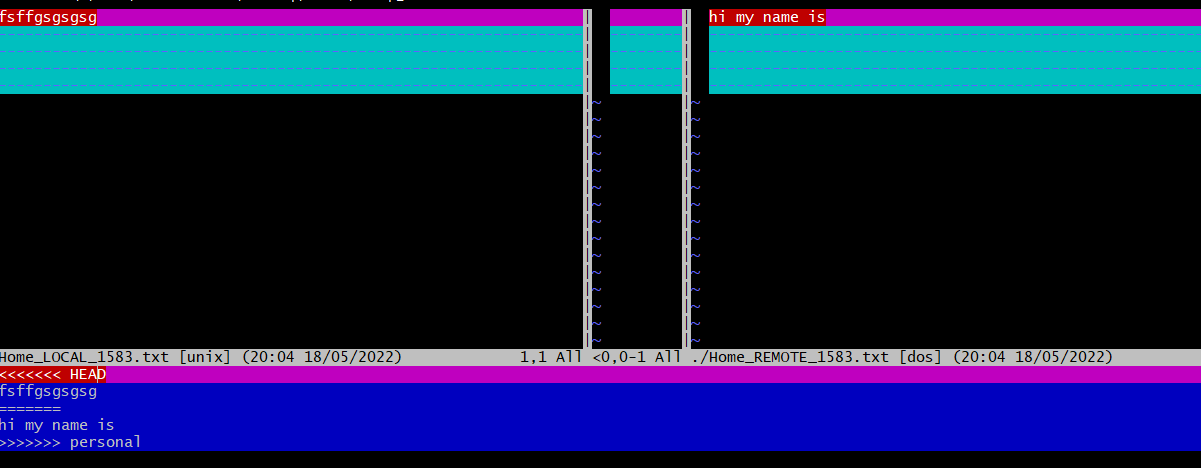
###### Both can view commithistory

When either the Collaborator or the Owner view the history, the conflict, associated branch, and the merged changes are clearly visible in the history.

Merge Conflict Challenge

Now it’s your turn. In pairs, intentionally create a merge conflict, and then go through the steps needed to resolve the issues and continue developing with the merged files. See the sections above for help with each of these steps:

* Step 0: Owner and collaborator ensure all changes areupdated
* Step 1: Owner makes a change andcommits
* Step 2: Collaborator makes a change and commits on the sameline
* Step 3: Collaborator pushes the file toGitHub
* Step 4: Owner pushes their changes and gets anerror
* Step 5: Owner pulls from GitHub to get Collaboratorchanges
* Step 6: Owner edits the file to resolve theconflict
* Step 7: Owner commits the resolvedchanges
* Step 8: Owner pushes the resolved changes toGitHub
* Step 9: Collaborator pulls the resolved changes fromGitHub
* Step 10: Both can view commithistory

****

#### **Workflows to avoid mergeconflicts**

Some basic rules of thumb can avoid the vast majority of merge conflicts, saving a lot of time and frustration. These are words our teams live by:

* Communicateoften
* Tell each other what you are workingon
* Pull immediately before you commit orpush
* Commit often in smallchunks.

**A good workflow is encapsulated as follows**:

Pull->Edit->Add->Pull->Commit->Push

Always start your working sessions with a pull to get any outstanding changes, then start doing your editing and work. Stage your changes, but before you commit, Pull again to see if any new changes have arrived. If so, they should merge in easily if you are working in different parts of the program. You can then Commit and immediately Push your changes safely. Good luck, and try to not get frustrated. Once you figure out how to handle merge conflicts, they can be avoided or dispatched when they occur, but it does take a bit ofpractice.

**Experiment No. 09**

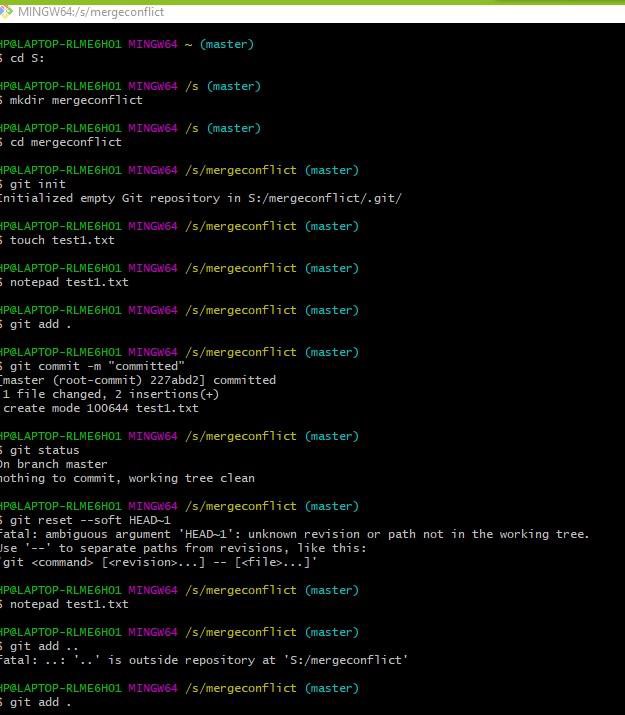
**Aim:** Reset and Revert

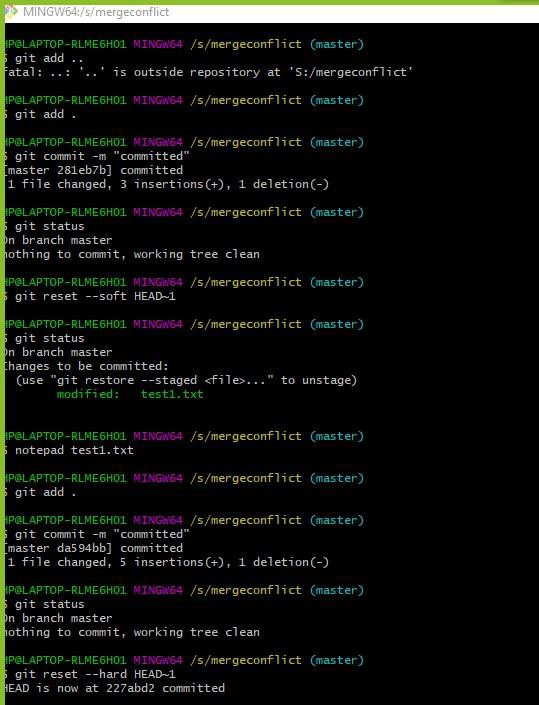
One of the lesser understood (and appreciated) aspects of working with Git is how easy it is to get back to where you were before—that is, how easy it is to undo even major changes in a repository. In this article, we'll take a quick look at how to reset, revert, and completely return to previous states, all with the simplicity and elegance of individual Git commands.

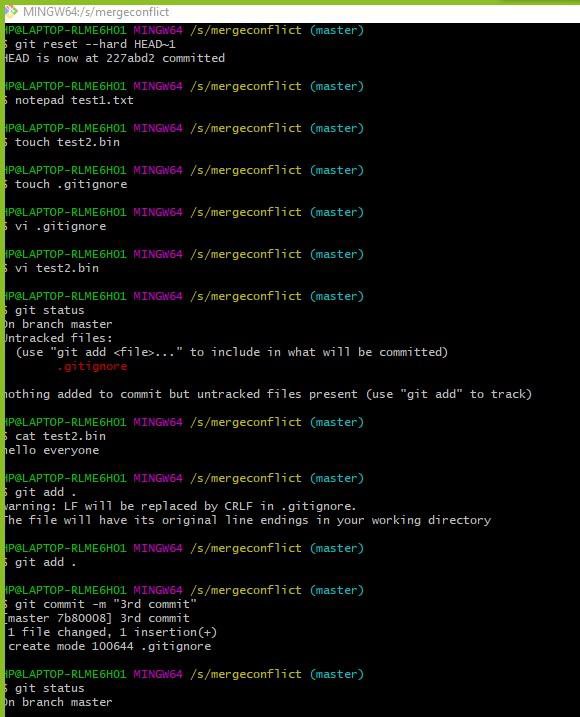
1. **Git Reset:-**GitresetisapowerfulcommandthatisusedtoundolocalchangestothestateofaGitrepo.Gitresetoperateson"TheThreeTreesofGit".ThesetreesaretheCommitHistory(HEAD),theStagingIndex,andtheWorkingDirectory.

TheeasiestwaytoundothelastGitcommitistoexecutethe“gitreset”commandwiththe“–soft”optionthatwillpreservechangesdonetoyourfiles.

Gitreset--hard,whichwillcompletelydestroyanychangesandremovethemfromthelocaldirectory.







1. **Git ignore** :- When sharing your code with others, there are often files orpartsofyourproject,youdo notwanttoshare.

Gitcanspecifywhich filesorpartsofyourprojectshouldbeignoredbyGitusinga.gitignorefile.

Gitwillnottrackfiles andfoldersspecifiedin.gitignore.However,the

.gitignorefile itself **IS** tracked byGit

A Project report

on

**“Project Title”**

with

**Source Code Management**

(CS181)

Submitted by

Team Member 1 Aaryan Sood2110990020

Team Member 2 Ansh Wadhwa 2110990217

Team Member 3 Harshit Patel2110990580

Team Member 3 Abhishek Krishnan Rathaur2110990564



**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 | **Beta** | Group No | G-08 |
| Course Coordinator | **Dr.Navjeet Kaur** | | |
| Faculty Name | **Dr. Monit Kapoor** | | |

Submission

Name: Deepender Singh

Signature:

Date:

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Title** | **Page No.** |
| 1 | Version control with Git | 1-4 |
| 2 | Problem Statement | 5 |
| 3 | Objective | 6 |
| 4 | Concepts and commands | 7-14 |
| 5 | Workflow and Discussion | 15-28 |
| 6 | Reference | 29 |

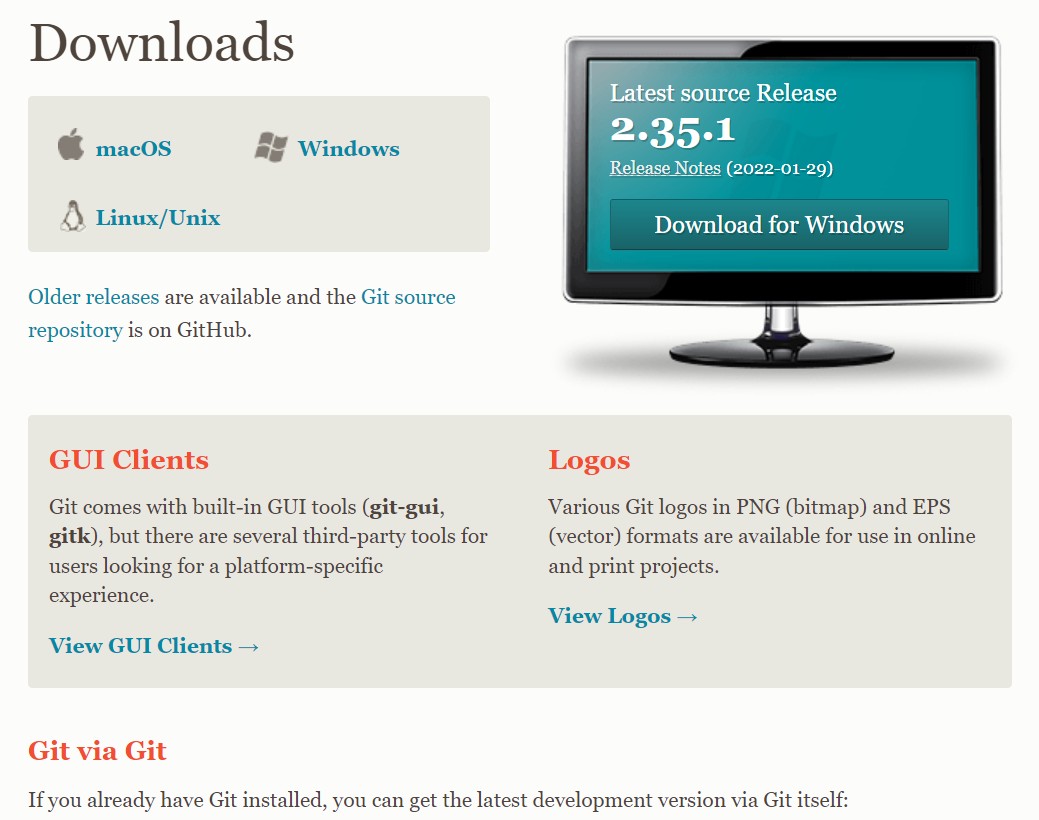
1. Version control with Git

#### Steps:-

To download the Git installer, visit the Git official site and go to thedownloadpage.

Thelinkforthedownloadpageis<https://git-scm.com/downloads>

Thepagelookslikeas:



Click onthe package given on the page as**download2.23.0forwindows**.Thedownloadwillstartafterselectingthepackage.

Now,theGitinstallerpackagehasbeendownloaded.

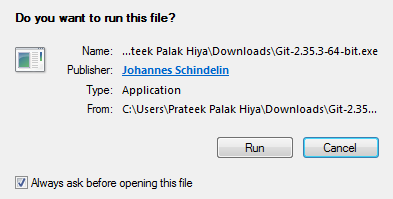
* Clickonthedownloadforwhichyouwanttodownload.Thepagelookslikeas:-



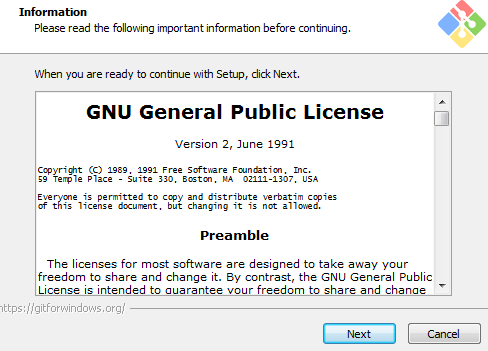
* Downloadingwillstarthere.



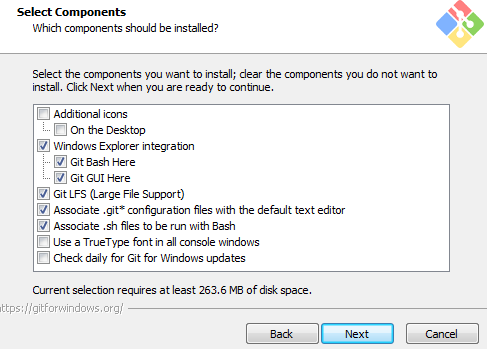
* Nowclickonrunasshownhere.



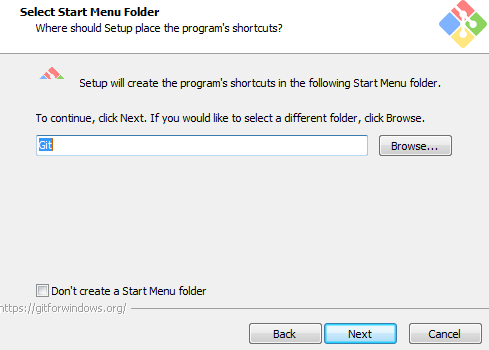
* Donextasshownbelow.



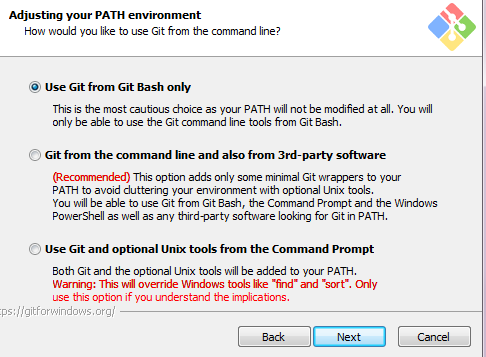
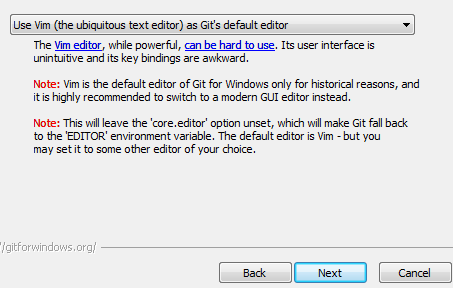
* Selectthefolderyouwanttodoand donext.
* Simplyclickonthenextbuttonasitautomaticallyselectstherequiredfile.Thepagelookslikeas:-

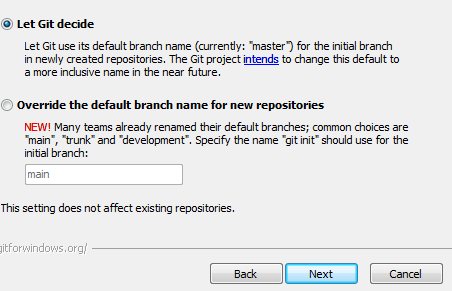


* Donext

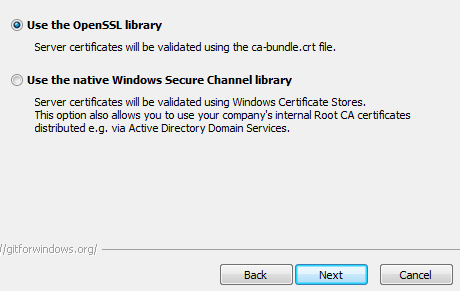


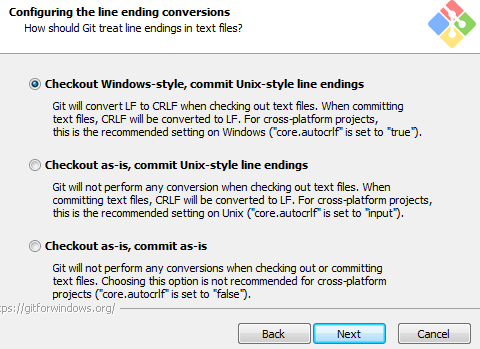
* UsevimasaGitdefaulteditor.



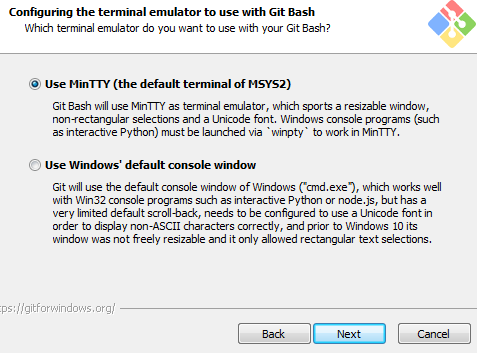
* Donext.
* UseGitfromGitBashonly,asshown.

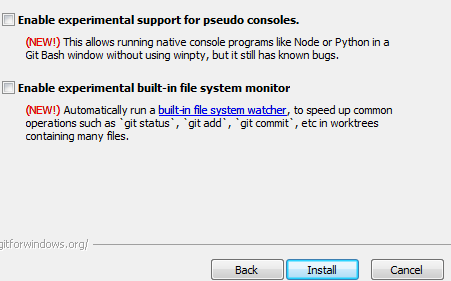




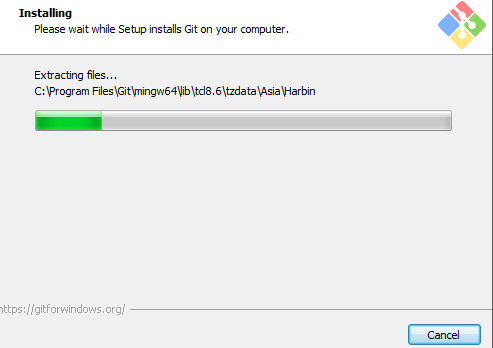


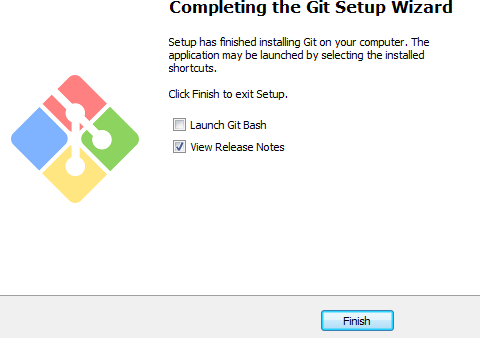
* Choosedefault.





* TheGitisgettingdownloadinyoursystem.Thepagelookslikeas:-





* + YoucancheckthatGitisinstallbysimplytypegit--versionin

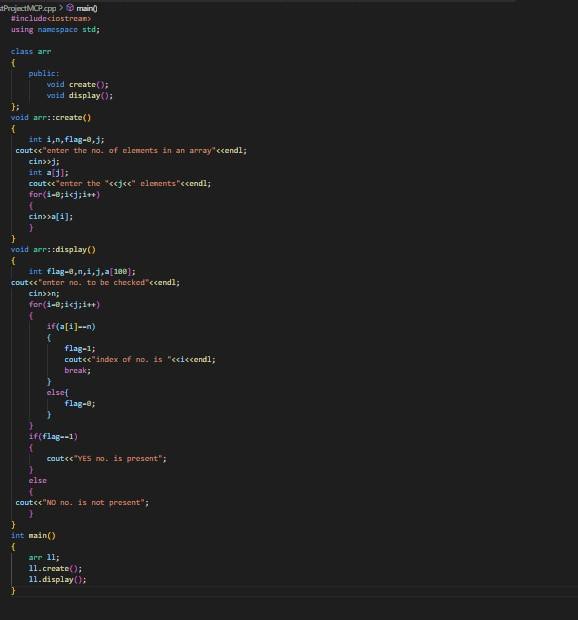
Thepagelookslikeas:-



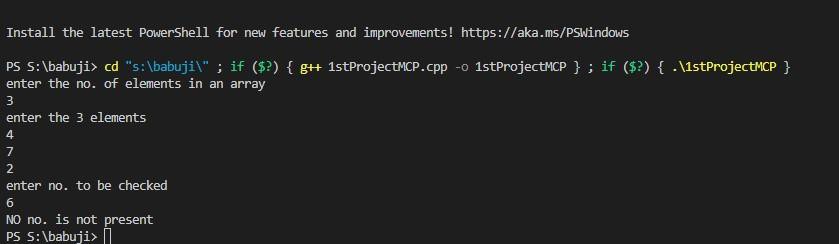
2. Problem Statement

**Aim:**Write a program to accept N numbers from theuser and store them in an array. Then, accept anothernumber from the user and search that using LinearSearch.

**PROGRAMCODE:-**



**OUTPUT:**



Through our project, The Awd clone we learnt the core concepts of HTML& JavaScript. We learnt Developer Skills in JavaScript.

3. Objective

With the help of DOM & Event Fundamentals we created a more authentic working for the project webpage.

We also observed how to Manipulate CSS Styles, Handling Click Events. Working with Class Object was a great experience, Dry Running the program helped us to figure out the common mistake one makes during such projects.

We learnt new concepts with every step on the project. The users can refresh their mind by playing this user-friendly music player website. The code is written in a very simple manner so that even the basic users don’t face any difficulties going through the code.

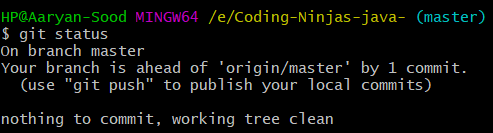
This AWD Project covers all concepts from basic to advance in HTML & JavaScript. So, this project is a one stop solution for everyone who wants to learn Web-Coding and have a keen interest in developing such Fun Projects.

4. Concepts and Commands

**Codes of git with concept with the concepts:**

* **git status**

To check the work done

****

* **git config user.name**

To verify linked mail

* **git config --global user.name**

To link repo with GitHub username

****

* **git config --global user.email**

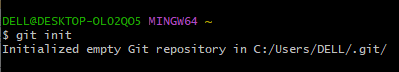
To link repo with GitHub mail

* **git config user.email**

To verify linked username

****

* **git init**
* To make folder git ready

****

* **git add –a**

To push all the files to repo

* **git add filename**

To push a particular file to repo

****

* **git branch name**

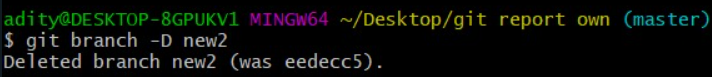
To make a new branch

* **git checkout name**
* To switch between branch
* **git branch -d branch name**

To delete branch (**Soft delete** because it ask to merge **)**

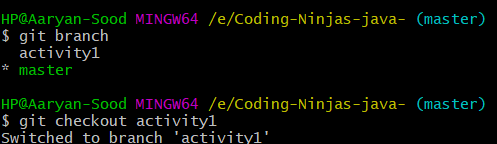
* **git branch -D branch name**

To delete branch **(Hard delete** because it don’t ask to merge**)**

****

* **git branch**

To see number of branches

****

* **git branch -m new branch name**

To rename a branch (we need to be in that branch) ****

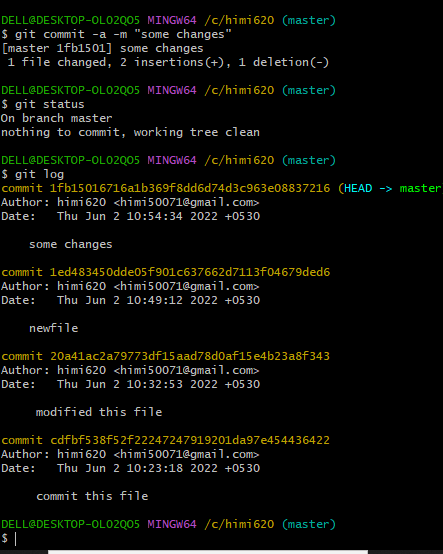
* **git branch -r**

To see number of branches

****

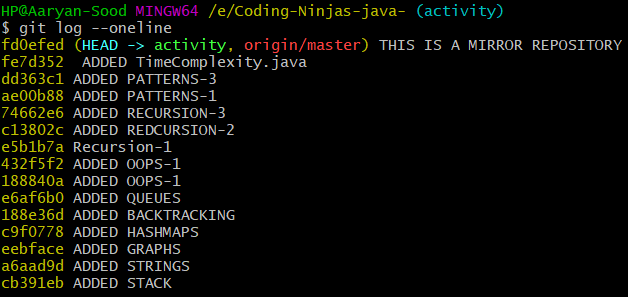
* **git log**

used to check the history of the work done also contains a checksum

****

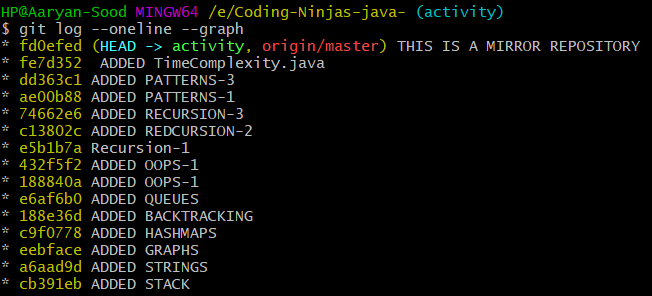
* **git log –oneline**

To get log in short

****

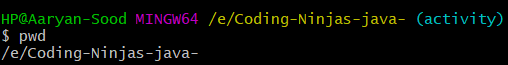
* **git log --online –graph**

To get log in graph format

****

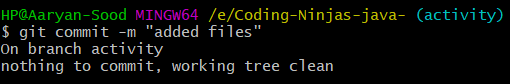
* **pwd**

Present working Directory

****

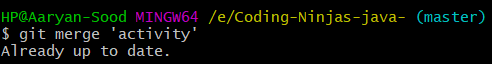
* **git commit -m ”any message”**

To be done after staging

****

* **git merge branch name**

To merge sub branch with master

****

* **mvold-file-namenew-file-name**

To rename a flolder (we have to do staging for this)

* **git mv old-file-namenew-file-name**

To rename a folder (no need for staging)

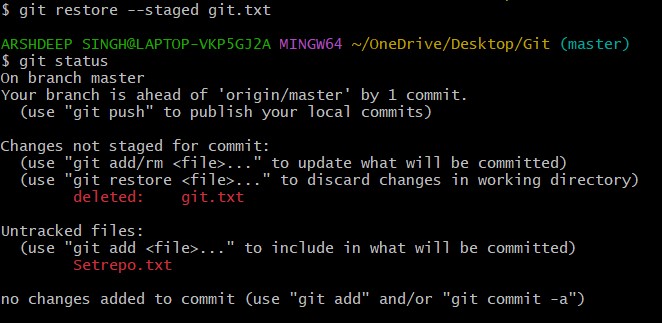


* **git restore --staged filed name**

To reverse to to previous version

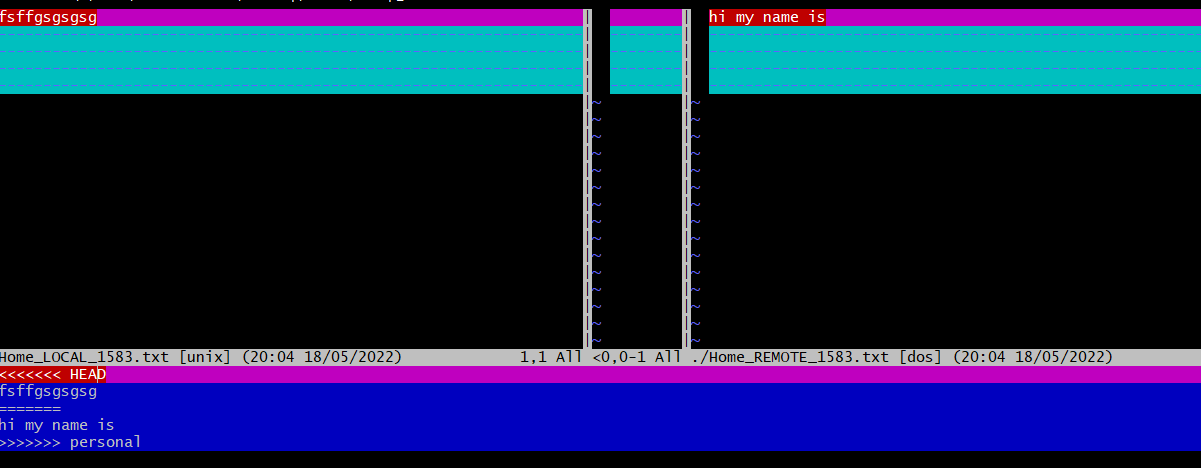
* **git restorefilename**

To go to previous command



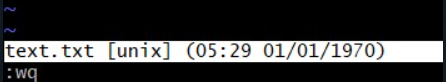
* **git mergetool**

To remove merge conflict i.e. when content in master and branch is different.

****

* **:wq**

To quit from special screen

****

* **rm -rf .git**

To delete whole git folder

* **rm -rf filename**

To delete a particular file

****

* **git remote add origin“link-of-repo-we-made-on-github**

To make new remote



* **git push -u master remote-nam**

To make data visible on sscloud

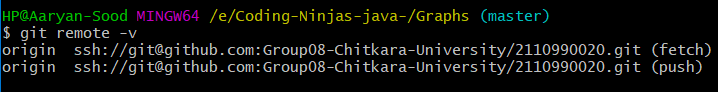
****

* **git pull link-of-repo-on-github**

To make changes done on cloud visible on the system.

* **git remote -v**

to see the location where remote is being stored

****

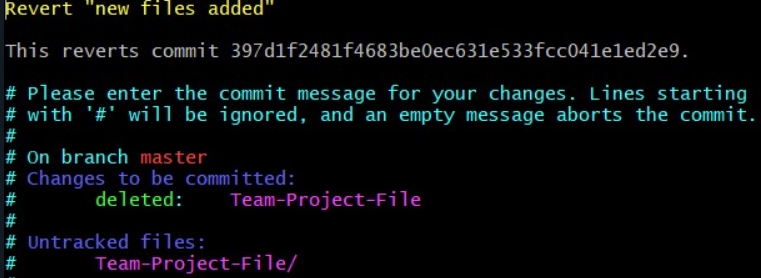
* **touch filename**

To make css/html/c++ files

****

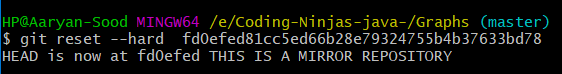
* **git revert checksum**

It’s a forwar moving undo operation that offers a safe mode of undoing changes

****

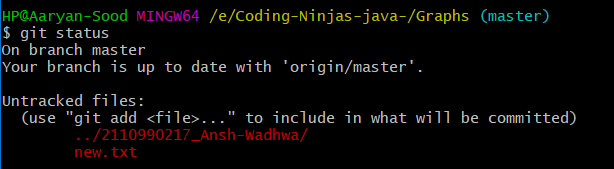
* **touch .gitignore**
* **git reset --hard checksum**

All the commits which was rested is deleted in the working directory along with the commit history.

****

* **git reset --mixedchecksum**

Reset commit files doesn’t get deleted it goes untracked changes (red colour on git status)

****

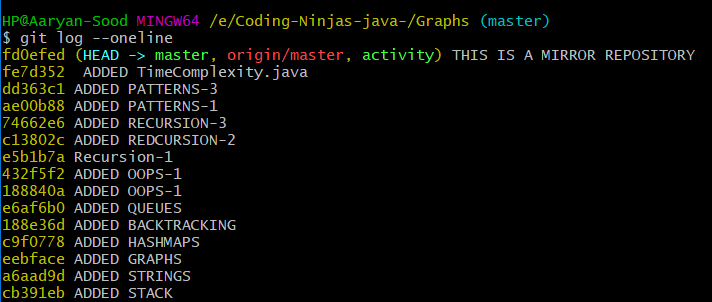
* **git reset --softchecksum**

Resseted commit files doesn’t get deleted it goes to staging area (green colour)

****

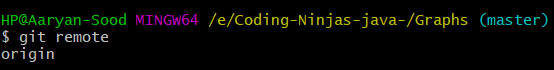
**git reset --any-type~ n(no. of commits to be changed)**

To delete no. of commits

****

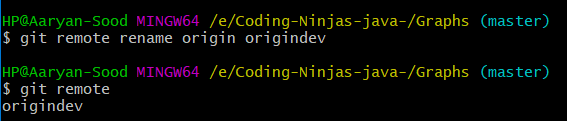
* **git remote**

To see the name of origin formed

****

* **git remote renameold-file-name new-file-name**

To rename remote

****

* **git remote removename**

To delete remote

****

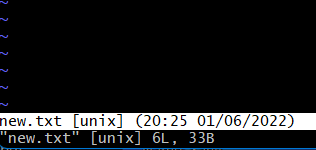
* **cat file-name**

To see the data stored in file in git bash without opening the file

****

* **vi file-name**

To edit content of file in git bash without opening the file

****

Open source szoftware is based on the idea that by sharing code, we can make better,more reliable software. For more information, see the "[About the Open Source Initiative](http://opensource.org/about)"ontheOpenSourceInitiative.Formoreinformationaboutapplyingopensourceprinciplestoyourorganization'sdevelopment work on GitHub.com, see GitHub's white paper "[An introduction toinnersource](https://resources.github.com/whitepapers/introduction-to-innersource/)."

5. Workflow and discussion

Whencreatingyourpublicrepositoryfromaforkofsomeone'sproject,makesuretoinclude a license file that determines how you want your project to be shared withothers. For more information, see "[Choose an open source license](https://choosealicense.com/)" atchoosealicense.com.

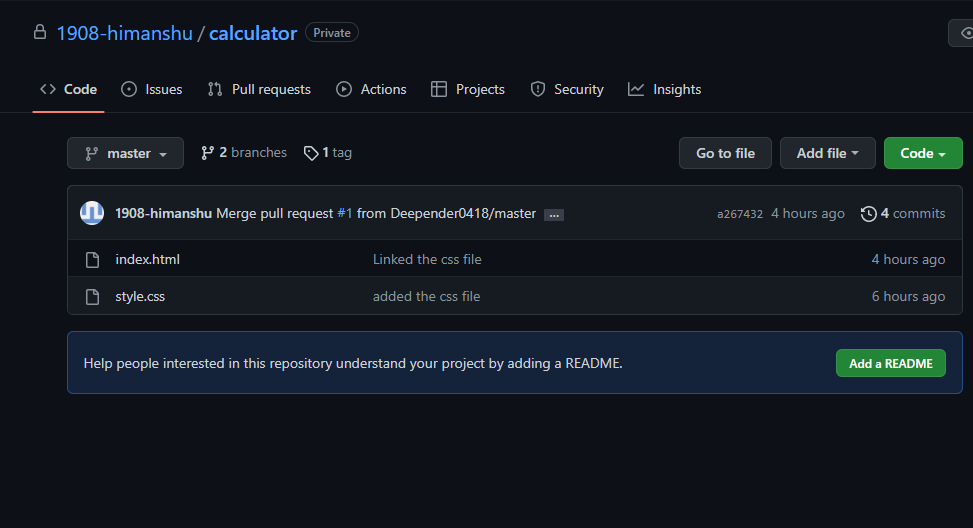
For more information on open source, specifically how to create and grow an opensource project, we've created [Open Source Guides](https://opensource.guide/) that will help you foster a healthyopensourcecommunitybyrecommendingbestpracticesforcreatingandmaintainingrepositories foryouropensourceproject.Youcanalsotakeafree[GitHubLearningLab](https://lab.github.com/)courseonmaintainingopensourcecommunities.

**Forking**



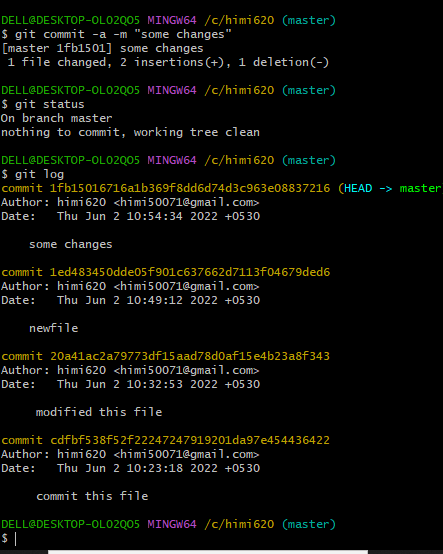
****

**View of repository on Github**

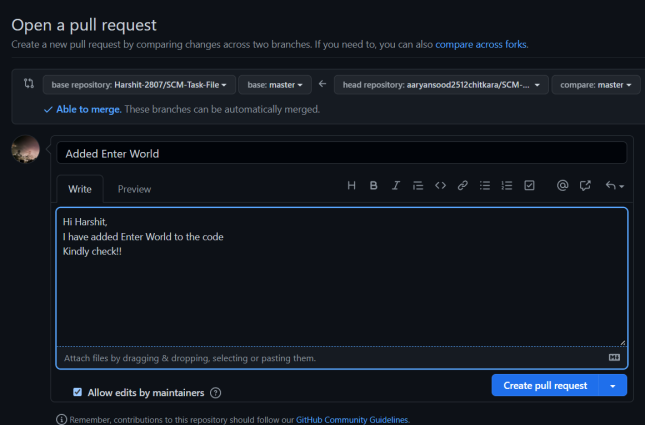
****

****

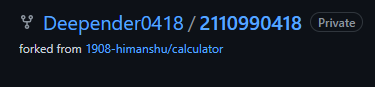
**git log after forking and collaborating**

****

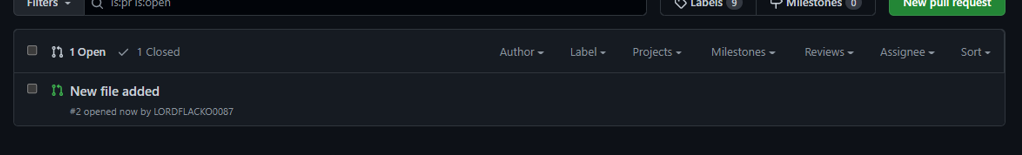
Creating a pull request to merge the Changes in forked Repo:



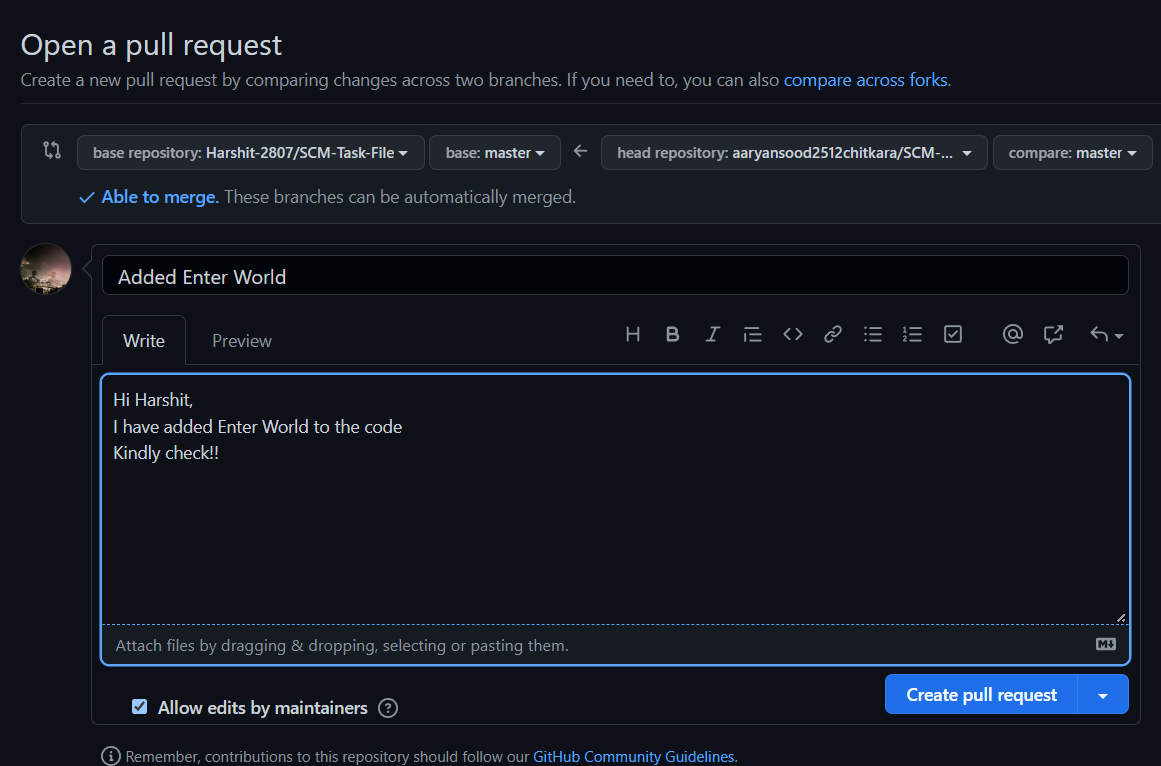
**Fork with Himanshu-**



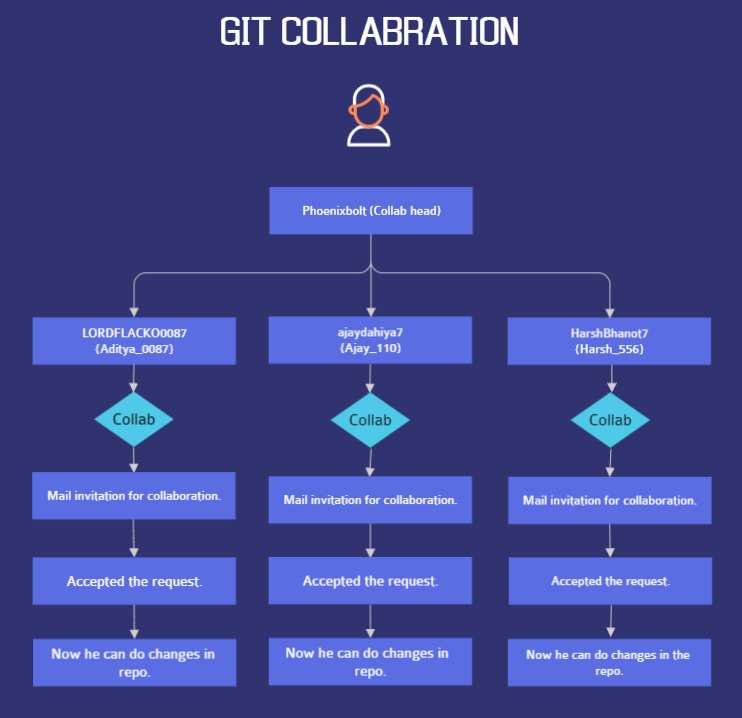
Creating a pull request to merge the Changes in forked Repo



Commenting on Pull request

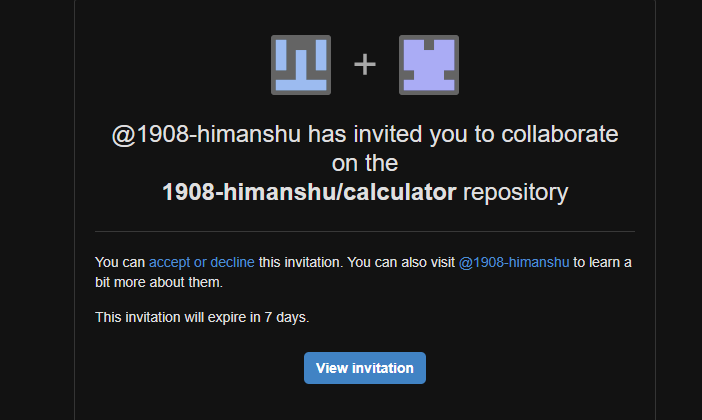


**Collaboration**

****

**Collab with -**

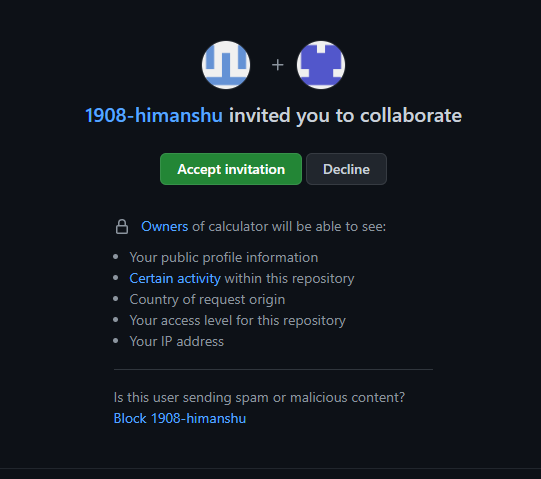
This is the invitation mail sent to harsh to add collaboration in Team-Project-File



Added AnshCollaborater to Phoenixbolt Repo

**collab with Ajay-**

This is the invitation mail sent to Ajay to add collaboration in Team-Project-File



**collab with himanshu-**

This is the invitation mail sent to add collaboration in Team-Project-File

6. Reference

Reference for few code snips of CSS and HTML were taken from Coding Ninjas video lectures and from few crash courses available on YouTube.

Reference for few code snippets of JavaScript were taken from the Udemy Courses available online.

**Links used for reference**

* <https://www.coursera.org/learn/introduction-git-github>
* <https://www.freecodecamp.org/news/git-and-github-crash-course/>
* <https://www.udemy.com/course/github-ultimate/>
* <https://www.udemy.com/course/git-started-with-github/?LSNPUBID=JVFxdTr9V80&ranEAID=JVFxdTr9V80&ranMID=39197&ranSiteID=JVFxdTr9V80-GmotHk.p_rwC77qokoBs_w&utm_medium=udemyads&utm_source=aff-campaign>