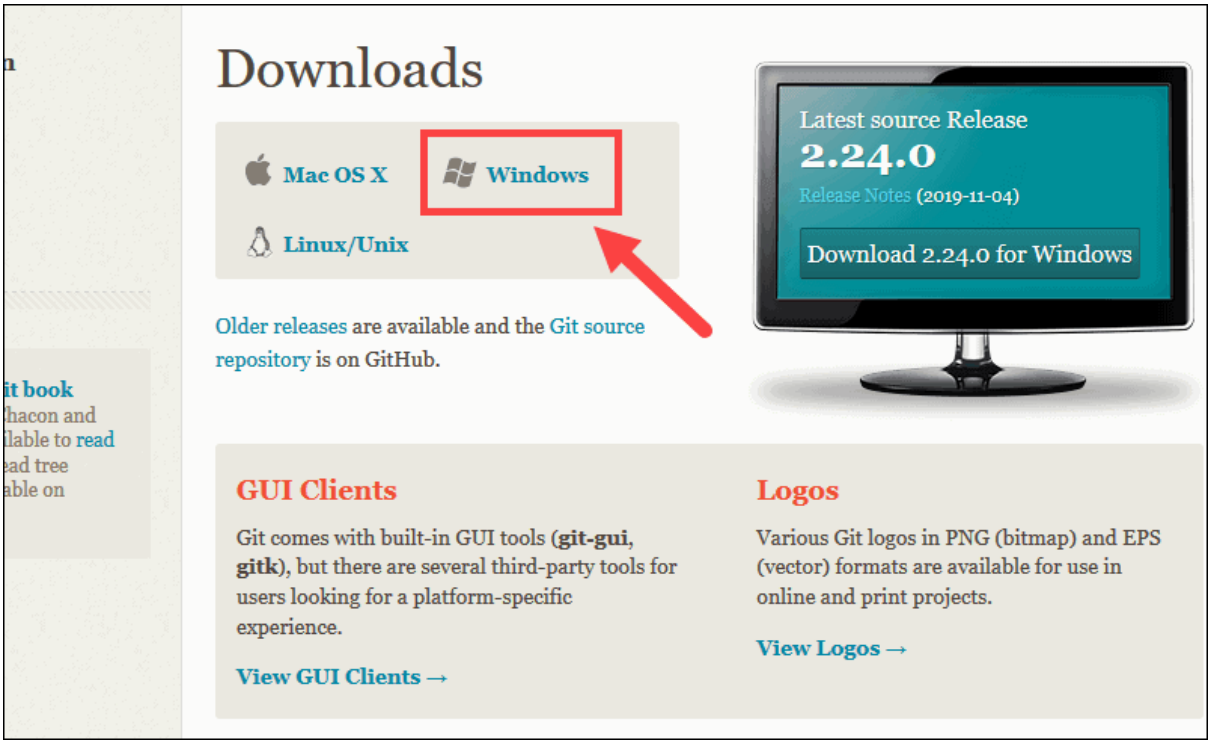
Name - Ayushman Dixit
Group - G8
Roll no. - 2110990351
SCM File

Experiment no.- 1

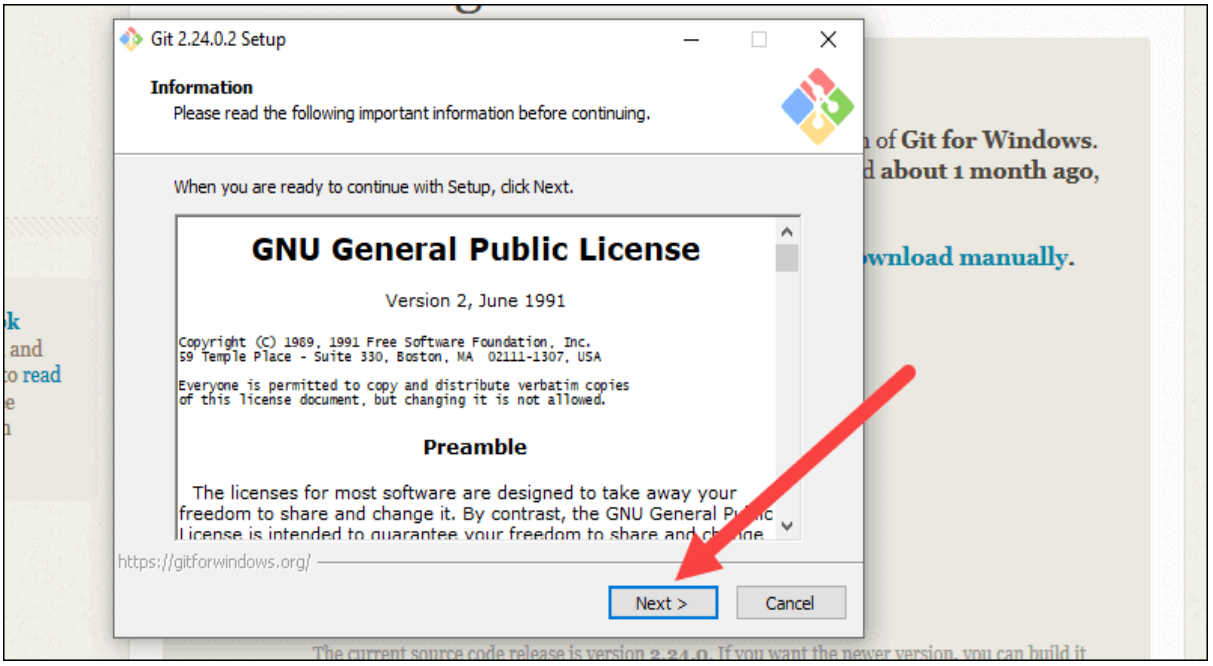
Installing Git

1. Browse the official Git website: https://git-scm.com/downloads

2. Click the download link for Windows and allow the download to complete.

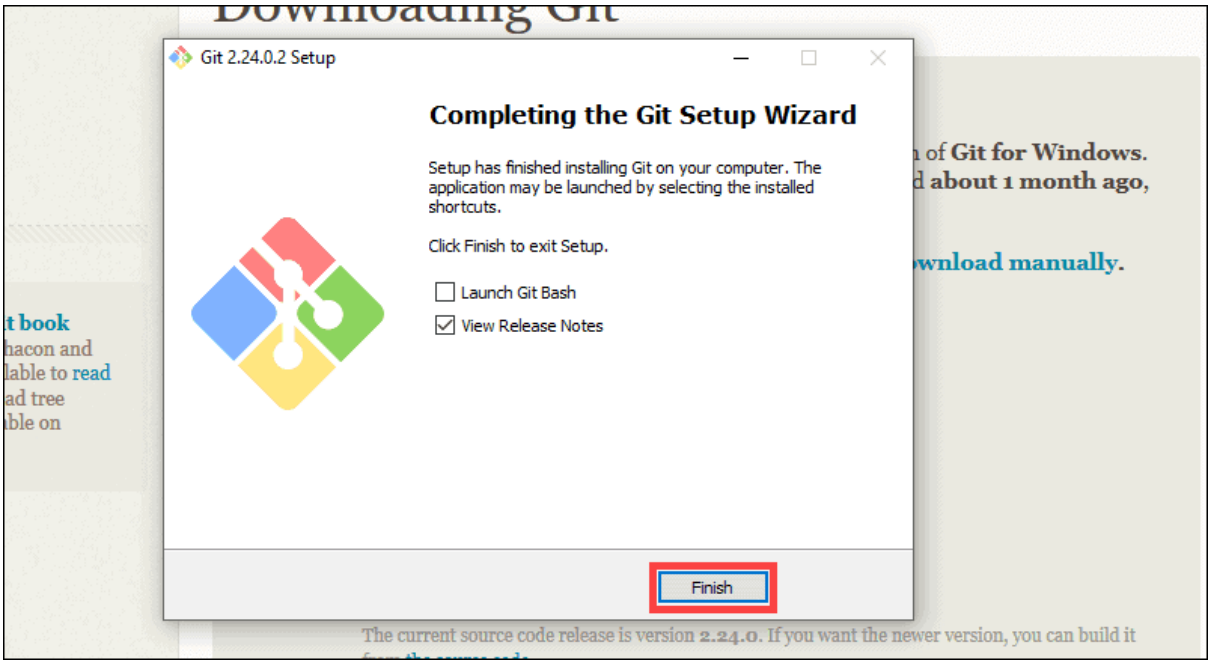


3. Review the GNU General Public License, and when you're ready to install, click Next.



4. When running the installer, various screens appear (Windows screens shown). Generally, you can accept the default selections, unless you have reason to change it, and click Next.

5. Once the installation is complete, tick the boxes to view the Release Notes or Launch Git Bash, then click Finish.



First-Time Git Setup

Now that you have Git on your system, let's set up the Git environment. Git comes with a tool called git config that lets you get and set configuration variables that control all aspects of how Git looks and operates.

The first thing you should do when you install Git is to set your user name and email address. This is important because every Git commit uses this information, and it's immutably baked into the commits you start creating:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

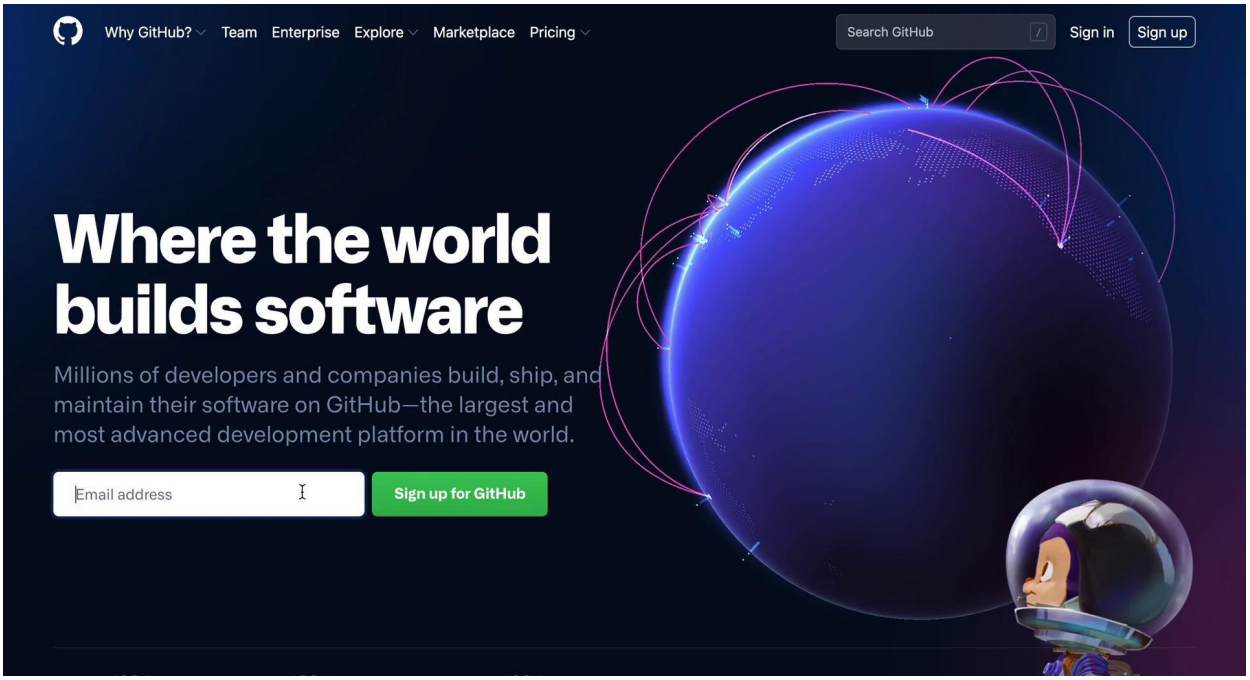
Again, you need to do this only once if you pass the --global option, because then Git will always use that information for anything you do on that system. If you want to override this with a different name or email address for specific projects, you can run the command without the --global option when you're in that project.

Many of the GUI tools will help you do this when you first run them.

Experiment - 2

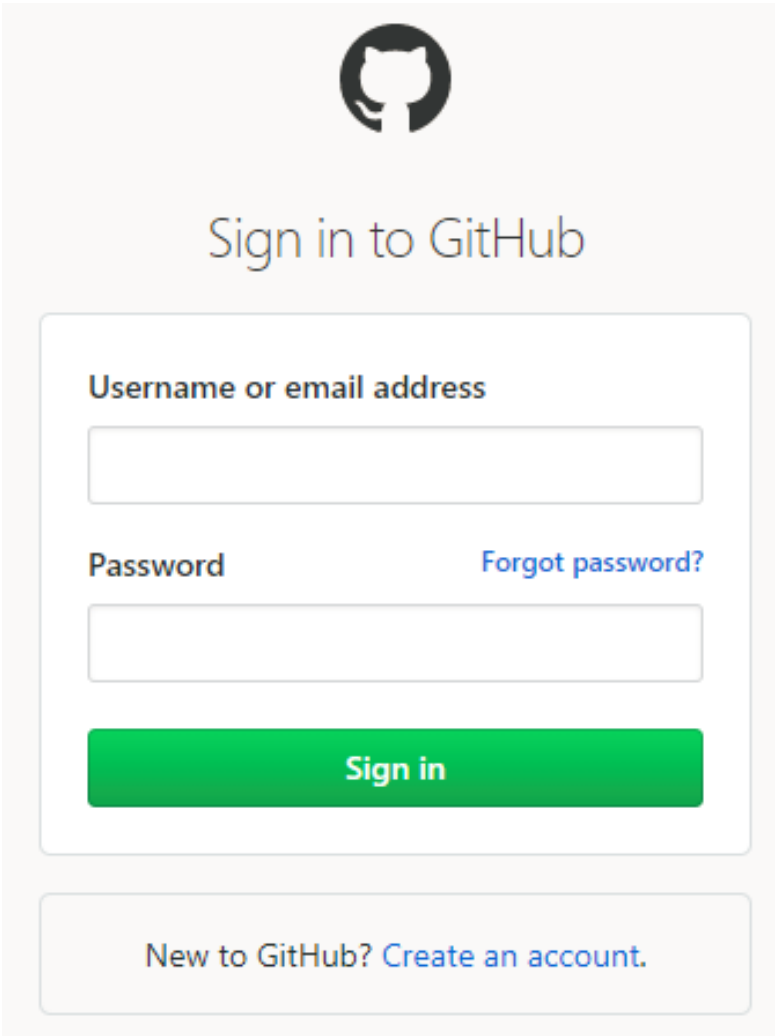
Setting up Github Account

1. Go to https://github.com.



2. Click on signup

3. Fill out your email and make a strong password



4. Complete the CAPTCHA.

5. Click Choose on your desired plan but for the first time, you can skip this.

6. Verify your email address.

7. Select your preferences.

Experiment - 3

Program to generate logs

Basic Git init :

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

Basic Git status:

The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git. The status output does not show you any information regarding the committed project history.

Basic Git add command:

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.

Basic Git commit:

The git commit command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as “safe” versions of a project Git will never change them unless you explicitly ask it to. Prior to the execution of the git commit, The git add command is used to promote or 'stage' changes to the project that will be stored in a commit. These two commands git commit and git add are two of the most frequently used.

Basic Git log:

The Git log command is one of the most usual commands of git. It is the most useful command for Git. Every time you need to check the history, you have to use the git log command. The basic git log command will display the most recent commits and the status of the head.

Experiment - 4

Create and visualize branches in Git

How to create branches?
The main branch in git is called a master branch. But we can make branches out of this main master branch. All the files present in the master can be shown in the branch but the files which are created in the branch are not shown in the master branch. We also can merge both the parent(master) and child (other branches).

1. For creating a new branch: git branch “name of the branch”.
2. To check how many branches we have: git branch.
3. To change the present working branch: git checkout “name of the branch”.

Visualizing Branches:
To visualize, we have to create a new file in the new branch “activity1” instead of the master branch. After this, we have to do three-step architecture i.e. working directory, staging area and git repository.

After this I have done the 3 Step architecture which is tracking the file, sending it to the staging area and finally, we can roll back to any previously saved version of this file.
After this, we will change the branch from activity1 to master, but when we switch back to the master branch the file we created i.e “hello” will not be there. Hence the new file will not be shown in the master branch. In this way, we can create and change different branches. We can also merge the branches by using the git merge command.

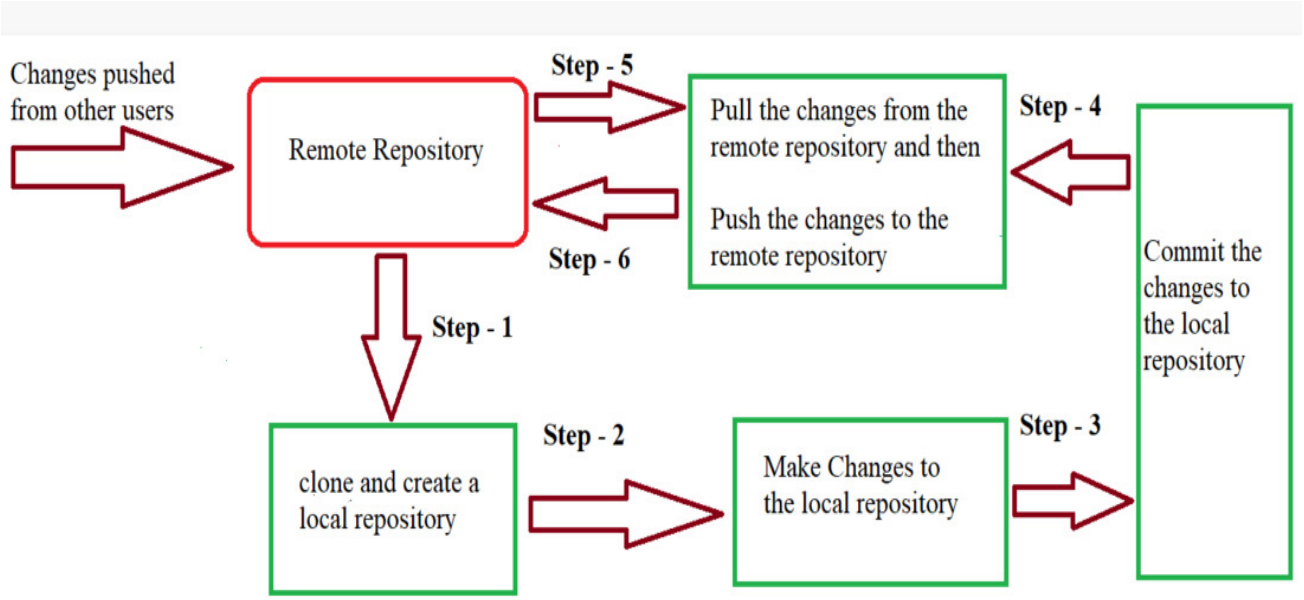
In this way, we can create and change different branches. We can also merge the branches by using git merge command.

Experiment - 5

Git lifecycle description

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

- You clone the Git repository as a working copy.
- You modify the working copy by adding/editing files.
- If necessary, you also update the working copy by taking other developers' changes.
- You review the changes before committing.
- You commit changes. If everything is fine, then you push the changes to the repository.
- After committing, if you realize something is wrong, then you correct the last commit and push the changes to the repository.



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

- Working Directory
- Staging Area
- Git Directory

1. Working Directory

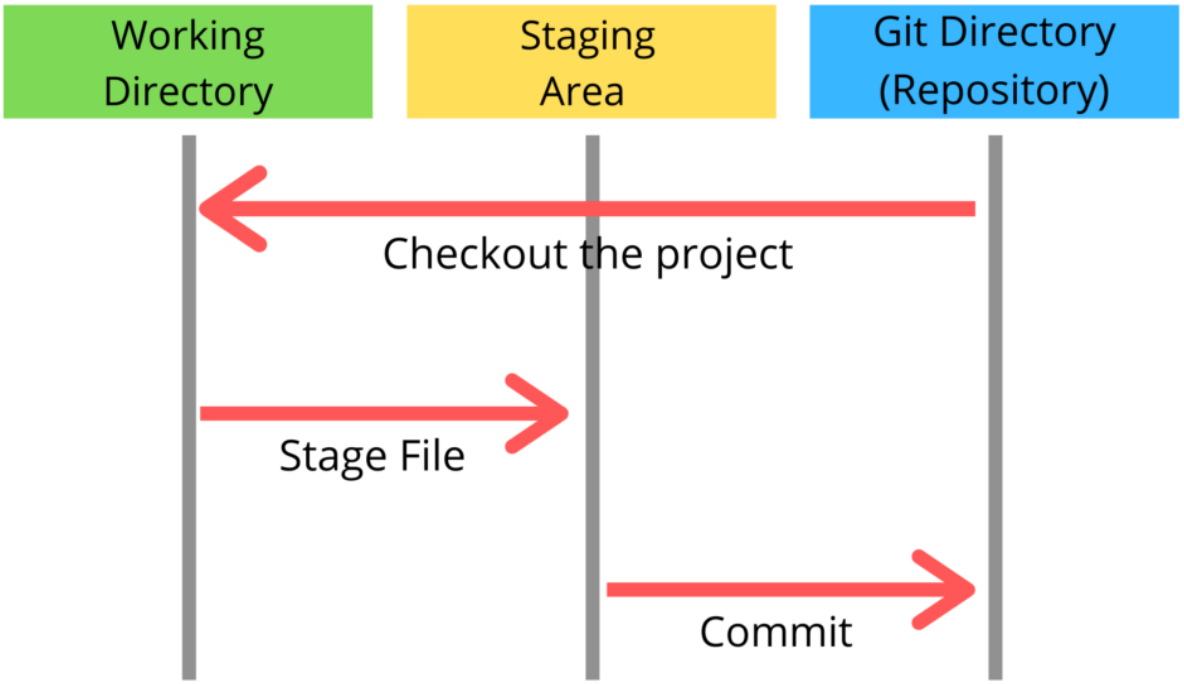
Whenever we want to initialize our local project directory to make it a git repository, we use the **git init** command. After this command, git becomes aware of the files in the project although it doesn't track the files yet. The files are further tracked in the staging area.

2. Staging Area

Now, to track the different versions of our files we use the command ***git add*****.** We can term a staging area as a place where different versions of our files are stored. ***git add*** command copies the version of your file from your working directory to the staging area. We can, however, choose which files we need to add to the staging area because in our working directory there are some files that we don't want to get tracked, examples include node modules, env files, temporary files, etc. Indexing in Git is the one that helps Git in understanding which files need to be added or sent. You can find your staging area in the ***.git*** folder inside the ***index*** file.

3. Git Directory

Now since we have all the files that are to be tracked and are ready in the staging area, we are ready to commit our files using the ***git commit*** command. Commit helps us in keeping the track of the metadata of the files in our staging area. We specify every commit with a message which tells what the commit is about. Git preserves the information or the metadata of the files that were committed in a Git Directory which helps Git in tracking files and basically it preserves the photocopy of the committed files. Commit also stores the name of the author who did the commit, files that are committed, and the date at which they are committed along with the commit message.


Working Directory
Staging Area
Git Directory (Repository)
Checkout the project
Stage File
Commit