Experiment-1

Aim: Setting up the git client.

On MacOS

Git is pre-installed on Zsh terminal in MacOS.

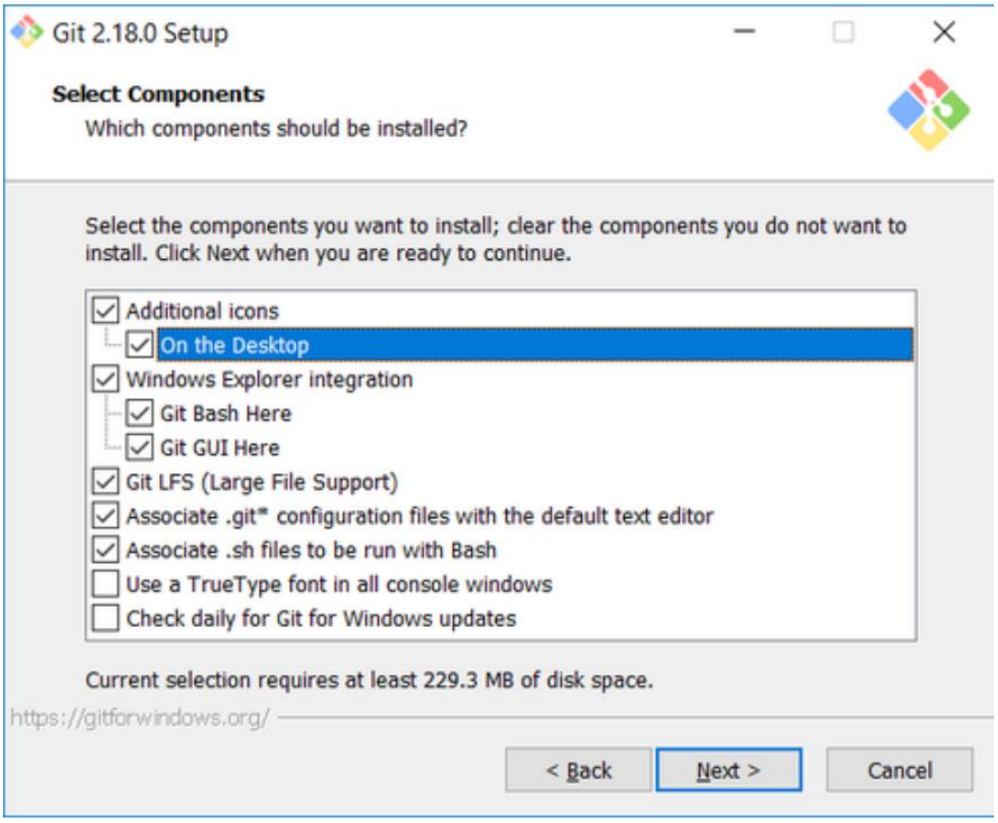
```
Last login: Sun Apr 10 17:18:57 on ttys000
(base) divyam@Divyams-MacBook-Pro ~ % git --version
git version 2.32.0 (Apple Git-132)
(base) divyam@Divyams-MacBook-Pro ~ %
```

On Windows

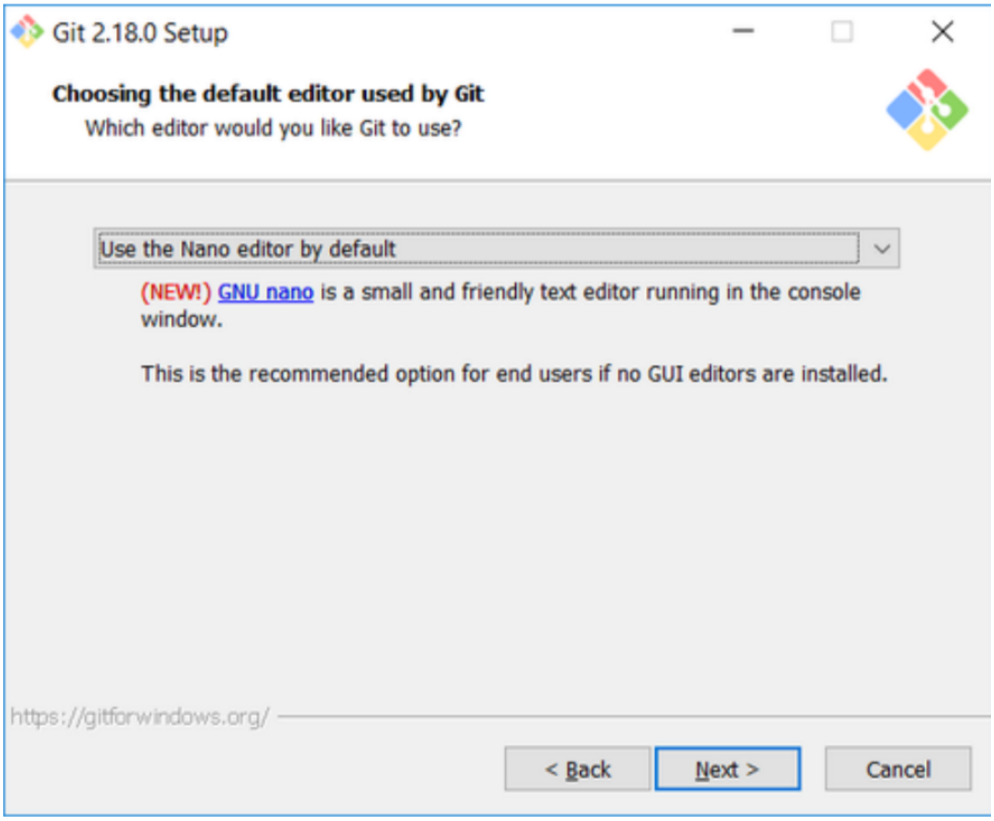
Git Installation: Download the Git installation program (Windows, Mac, or Linux) from Git -Downloads (git-scm.com).

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

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



In the choosing the default editor is used by Git dialog, it is strongly recommended that you DO NOT select default VIM editor- it is challenging to learn how to use it, and there are better modern editors available. It is recommended that you select Notepad++.

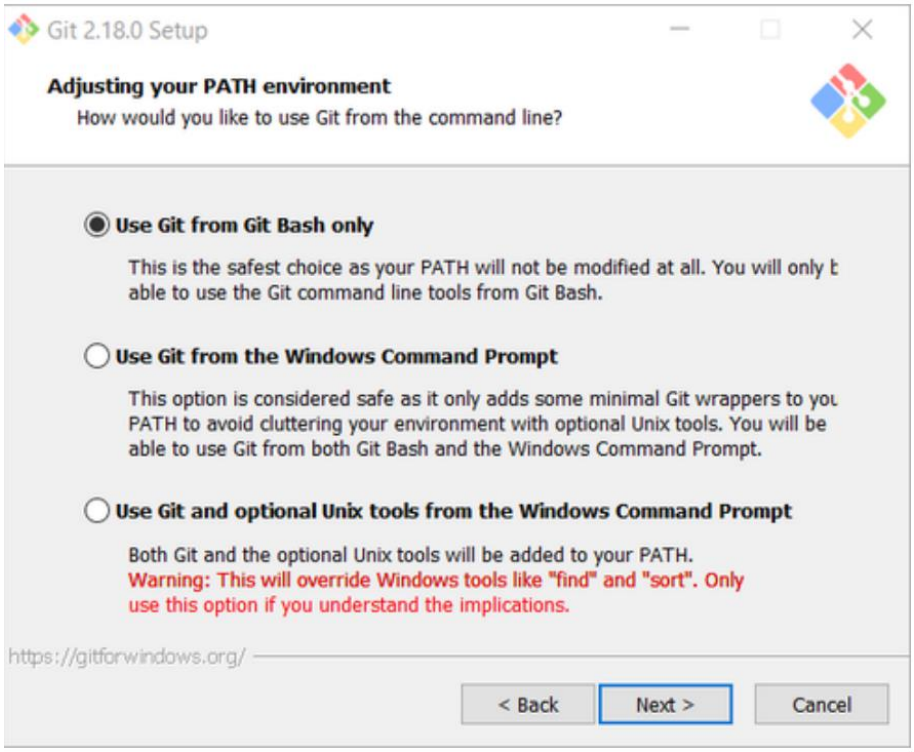


In the Adjusting your PATH screen, all three options are acceptable:

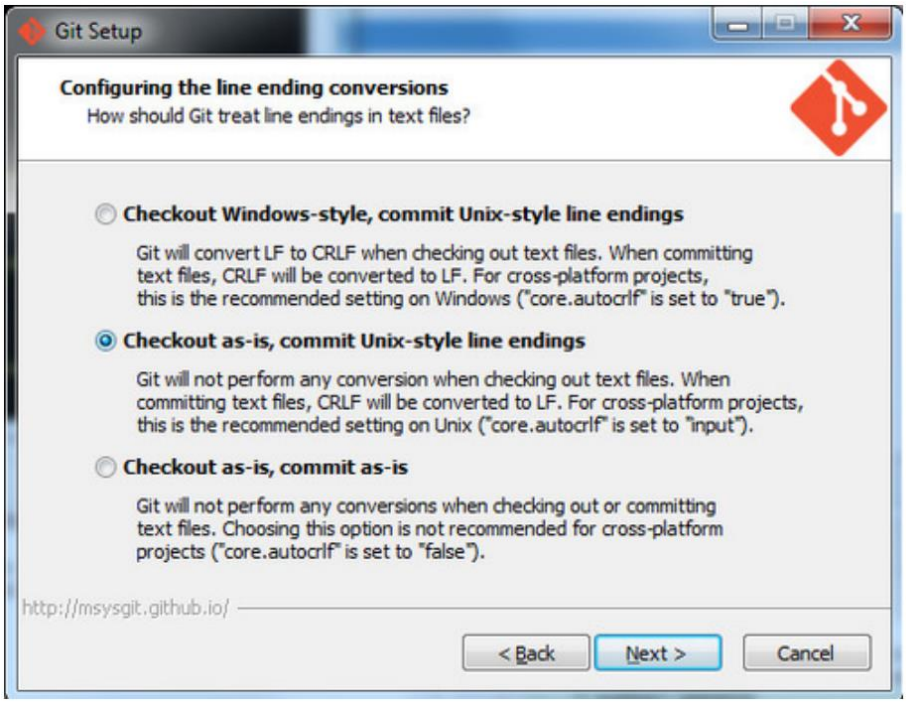
1. Use Git from Git Bash only: no integration, and no extra command in your command path.

2. Use Git from the windows Command Prompt: add flexibility – you can simply run git from a windows command prompt, and is often the setting for people in industry – but this does add some extra commands.

3. Use Git and optional Unix tools from the **Windows Command Prompt**: this is also a robust choice and useful if you like to use Unix like commands like grep.



In the Configuring the line ending screen, select the middle option (Checkout-as-is, commit Unix-style line endings) as shown. This helps migrate files towards the Unix-style (LF) terminators that most modern IDE's and editors support. The Windows convention (CR-LF line termination) is only important for Notepad.



Configuring Git to ignore certain files:

This part is required so that your repository does not get filled with garbage files.

By default, Git tracks all files in a project. Typically, this is not what you want; rather, you want Git to ignore certain files such as .bak files created by an editor or .class files created by the Java compiler. To have Git automatically ignore particular files, create a file named .gitignore (note that the filename begins with a dot) in the C: \users\name folder (where name is your MSOE login name).

NOTE: The .gitignore file must NOT have any file extension (e.g. .txt).

Edit this file and add the lines below, these are patterns for files to be ignored (taken from examples provided at https://github.com/github/gitignore.)

#Lines (like this one) that begin with # are comments; all other lines are rules

 # common build products to be ignored at MSOE *.o
***.obj**
***.class**

***.exe**

.DS_Store*

 # Any files you do not want to ignore must be specified starting with ! # For example, if you didn't want to ignore .classpath, you'd uncomment the following rule:
 # !.classpath

NOTE: You can always edit this file and add additional patterns for other types of files you might want to ignore. Note that you can also have a .gitignore files in any folder naming additional files to ignore. This is useful for project- specific build products.

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

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

b. Enter the command (replacing name as appropriate) **git config -- global core.excludesfile c:/users/name/.gitignore**

This tells Git to use the **.gitignore file** you created in step 2
NOTE: To avoid typing errors, copy and paste the commands shown here into the Git Bash window, using the arrow keys to edit the red text to match your information.

c. Enter the command **git config --global user.Email "name@msoe.edu"**

This links your Git activity to your email address. Without this, your commits will often show up as "unknown login". Replace name with your own **MSOE** email name.

d Enter the command **git config --global user.name "Your Name"** Git uses this to log your activity. Replace "Your Name" by your actual first and last name.

e. Enter the command **git config --global push.default simple** This ensures that all pushes go back to the branch from which they were pulled. Otherwise pushes will go to the master branch, forcing a merge.

Experiment-2

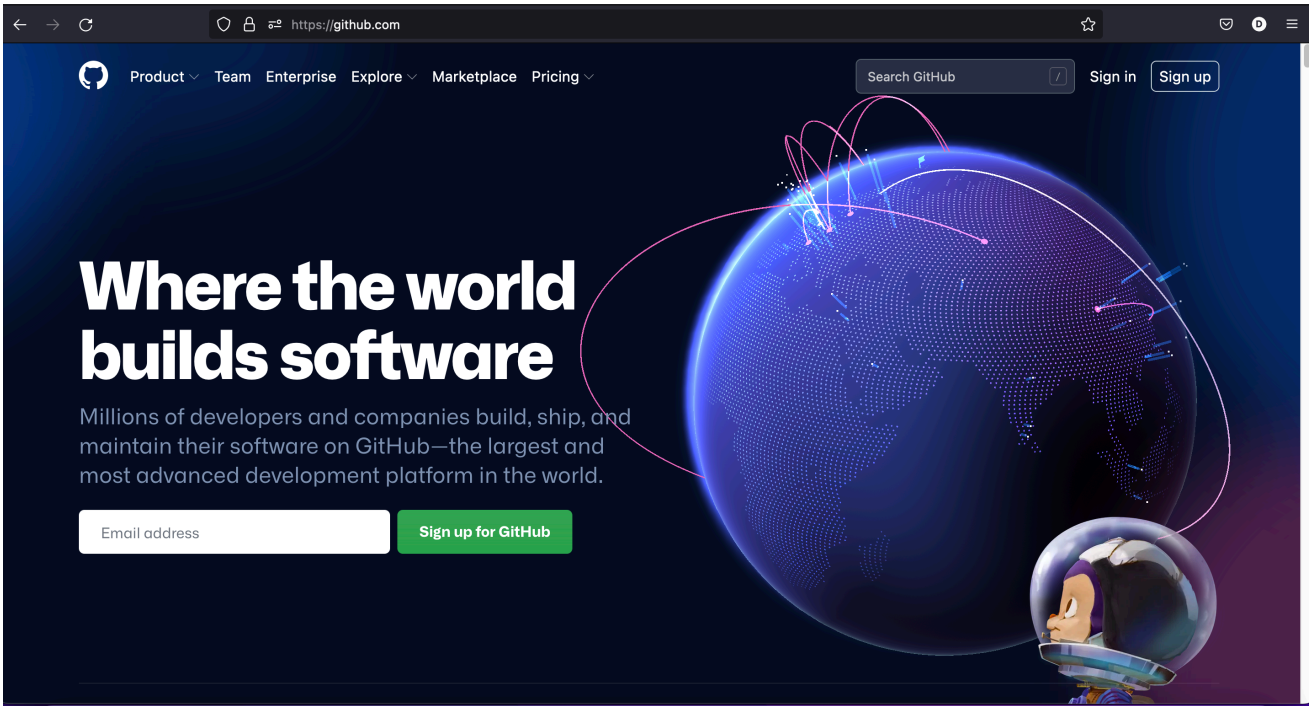
Aim -Setting up GitHub Account

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

Every person who uses GitHub has their own user account, which can be part of multiple organisations and teams. Your user account is your identity on GitHub.com and represents you as an individual.

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

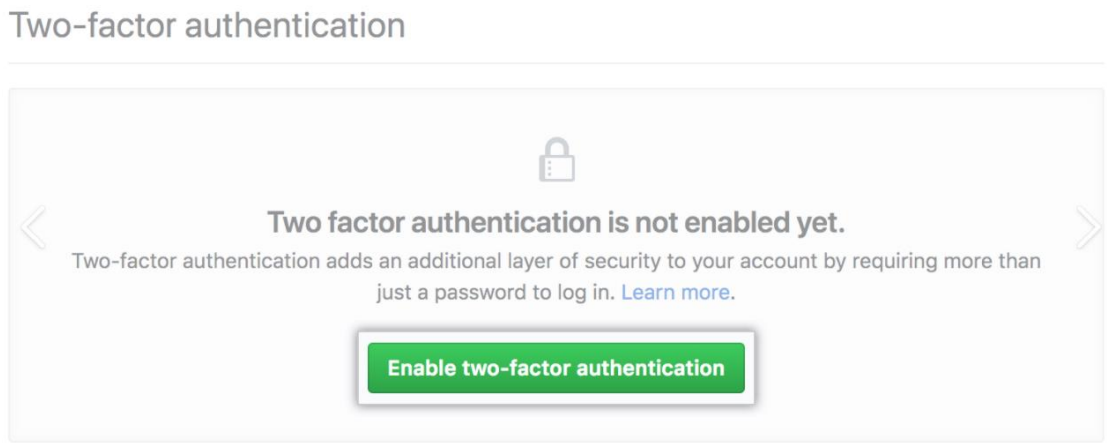
To keep your GitHub account secure you should use a strong and unique password. For more information, see "Creating a strong password".



2. Choosing GitHub product: You can choose GitHub Free or GitHub Pro to get access to different features for your personal account. You can upgrade at any time if you are unsure at first which product you want.
For more information on all GitHub's plans, see "GitHub's products".

3. Verifying email address: To ensure you can use all the features in your GitHub plan, verify your email address after signing up for a new account. For more information, see "Verifying your email address".

4. Two-factor authentication: Two-factor authentication, or 2FA, is an extra layer of security used when logging into websites or apps. We strongly urge you to configure 2FA for safety of your account. For more information, see "About two-factor authentication."



---

5. Viewing your GitHub profile: Your GitHub profile tells people the story of your work through the repositories and gists you've pinned, the organisation memberships you've chosen to publicise.

Experiment-3

Aim: Program to generate logs

Git init

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

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. Status output does not show you any information regarding the committed project history.

Git commit

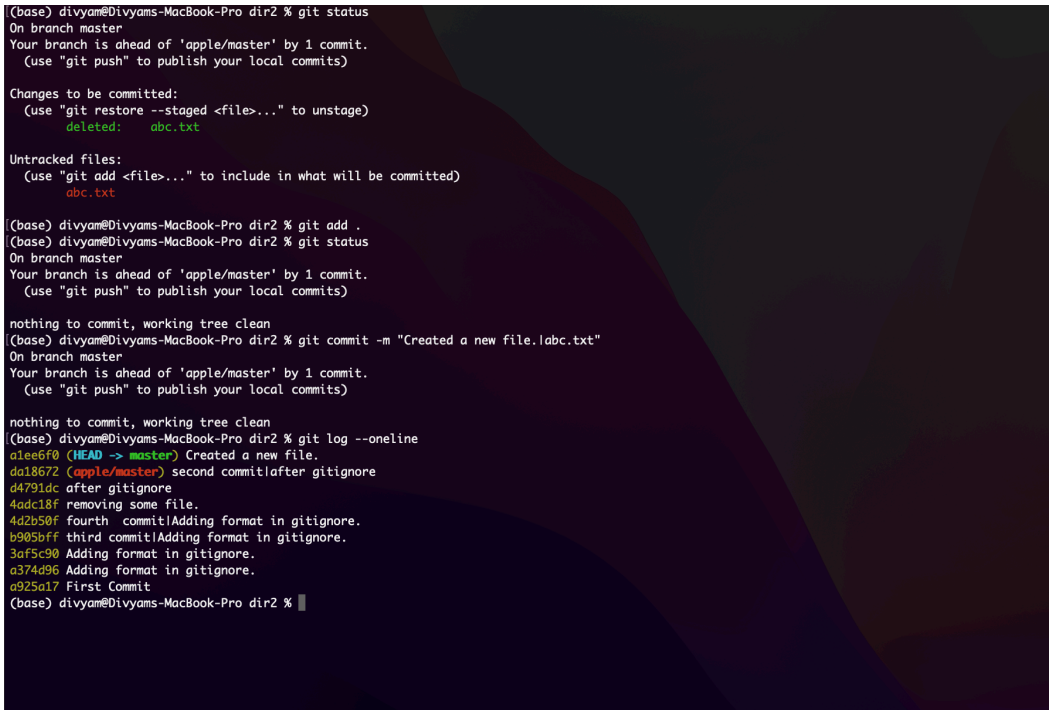
The **git commit** command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as "safe" versions of a project—Git will never change them unless you explicitly ask it to. Prior to the execution of git commit.

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.

Git log

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 git log command will display the most recent commits and the status of the head. It will be used as:



Experiment- 4

Aim: Create and visualise branches in Git

How to create branches?

The main branch in git is called 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 files which are created in branch are not shown in master branch. We also can merge both the parent(master) and child (other branches).

1. For creating a new branch: git branch "name of branch".

2. To check how many branches we have : git branch.

3. To change the present working branch: git checkout "name of the branch".

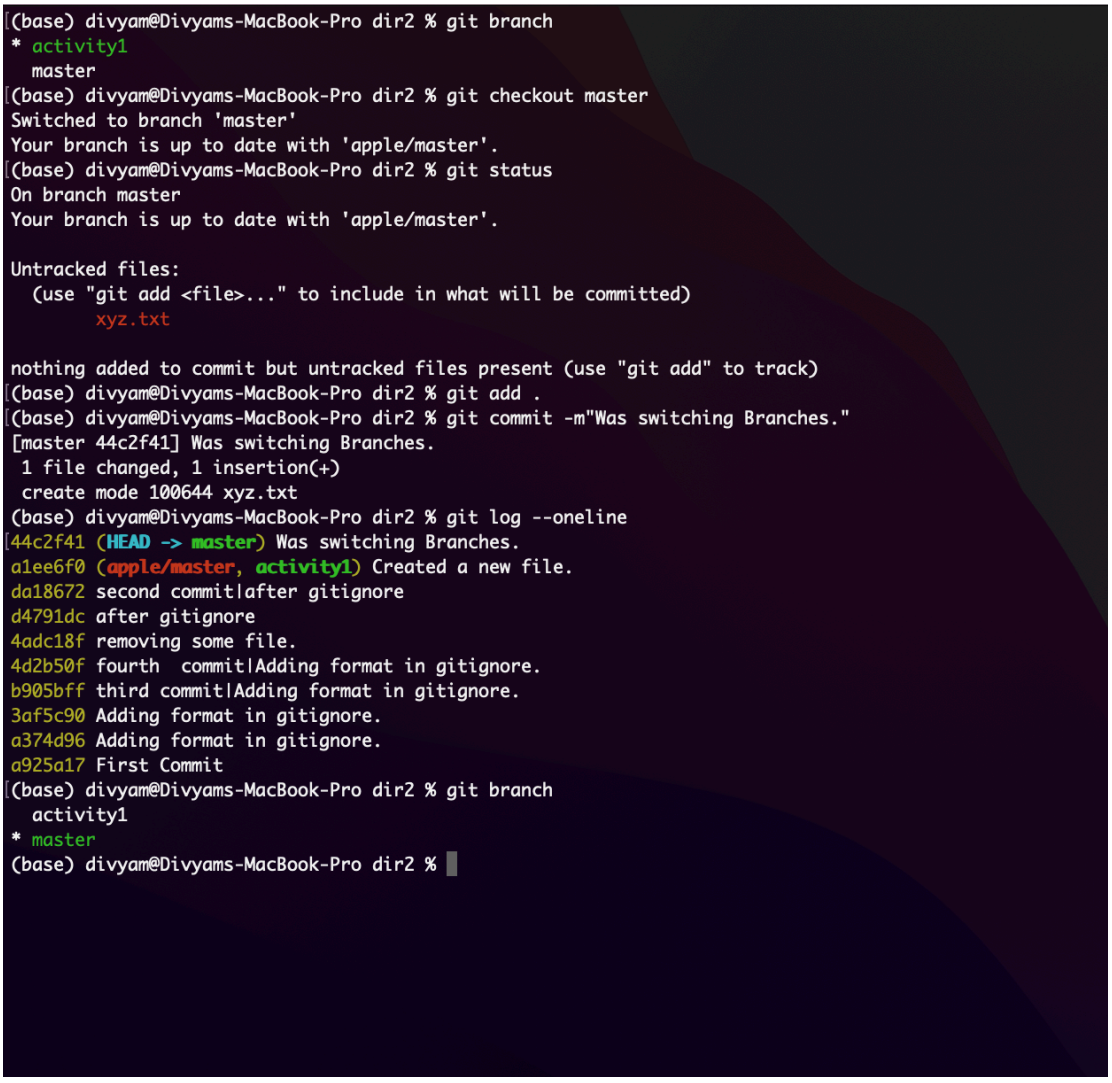
Visualising Branches:

To visualise, 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, send it to staging area and finally we can rollback 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 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.

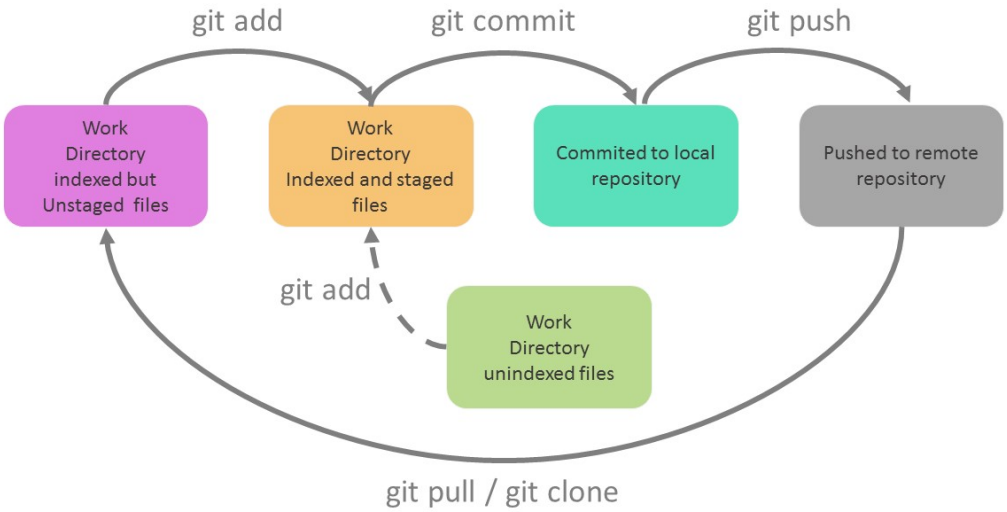
In this way we can create and change different branches. We can also merge the branches by using git merge command.



Experiment- 5

Aim: 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 :



Explanation:

- **Step 1-** We first clone any of the code residing in the remote repository to make our won local repository.

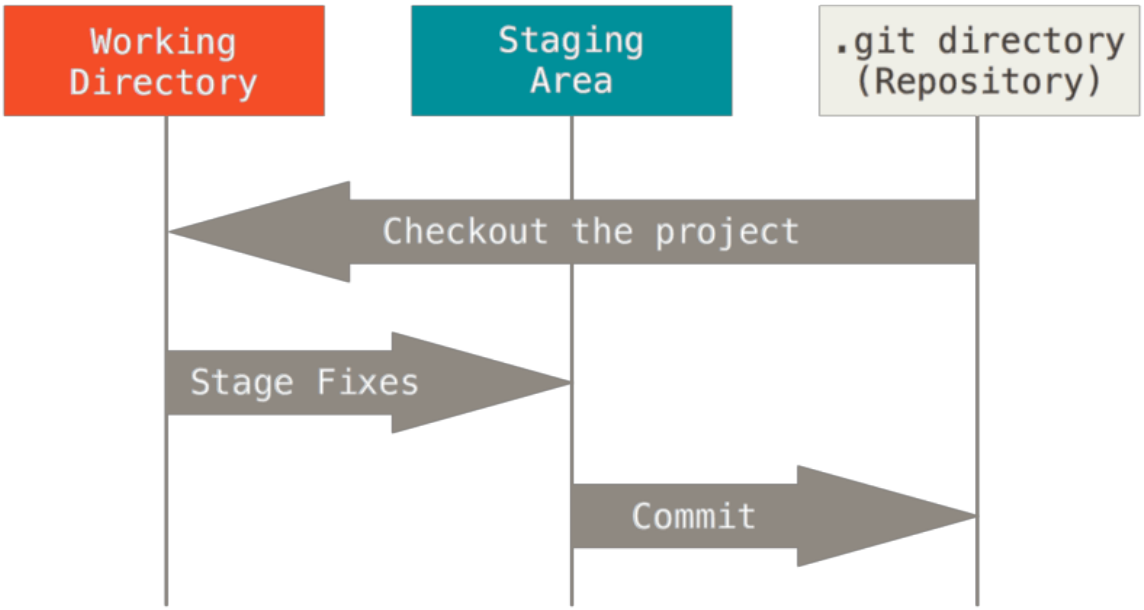
- **Step 2-** We edit the files that we have cloned in our local repository and make the necessary changes in it.

- **Step 3-** We commit our changes by first adding them to our staging area and committing them with a commit message.

- **Step 4 and Step 5-** We first check whether there are any of the changes done in the remote repository by some other users and we first pull that changes.

- **Step 6-** If there are no changes we push our changes to the remote repository and we are done with our work.

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

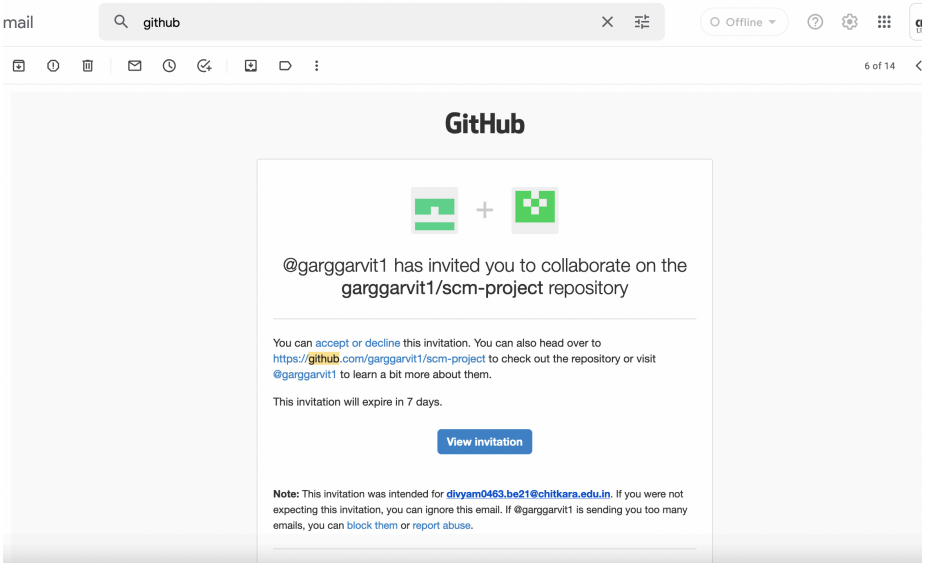


- **Working Directory**
  Whenever we want to initialise our local project directory to make a Git repository, we use the git init command. After this command, git becomes aware of the files in the project although it does not track the files yet. The files are further tracked in the staging area.

- **Staging Area**
  Now, to track files 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, 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.
  git add<filename>
  git add.

- **Git Directory (Repository)**

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 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. git commit -m <“ ”>.



Created by-Divyam Manchanda

2110990463-G07