Source Code Management File

Submitted by: HARSHITA BATRA

ROLL NO. - 2110990587

Group- 8 - B

INDEX

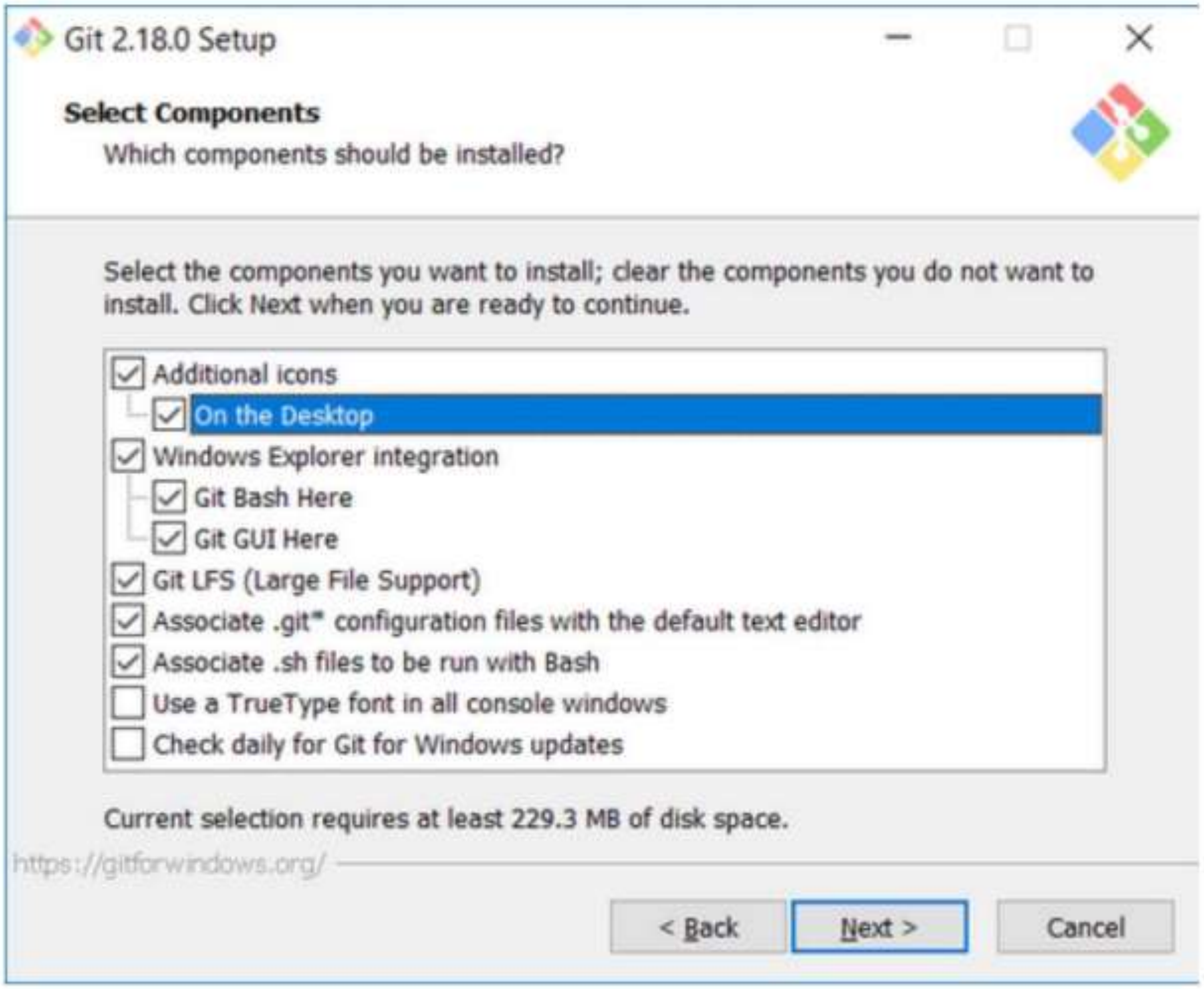
S. No	Aim of Experiment	Remarks by teacher
1	Setting up the git client	
2	Setting up GitHub Account	
3	Program to generate logs	
4	Create and visualize branches in Git	
5	Git life cycle description	

EXPERIMENT – 1

Aim: Setting up the git client.

Git Installation: Download the Git installation program

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



Git 2.18.0 Setup

Select Components

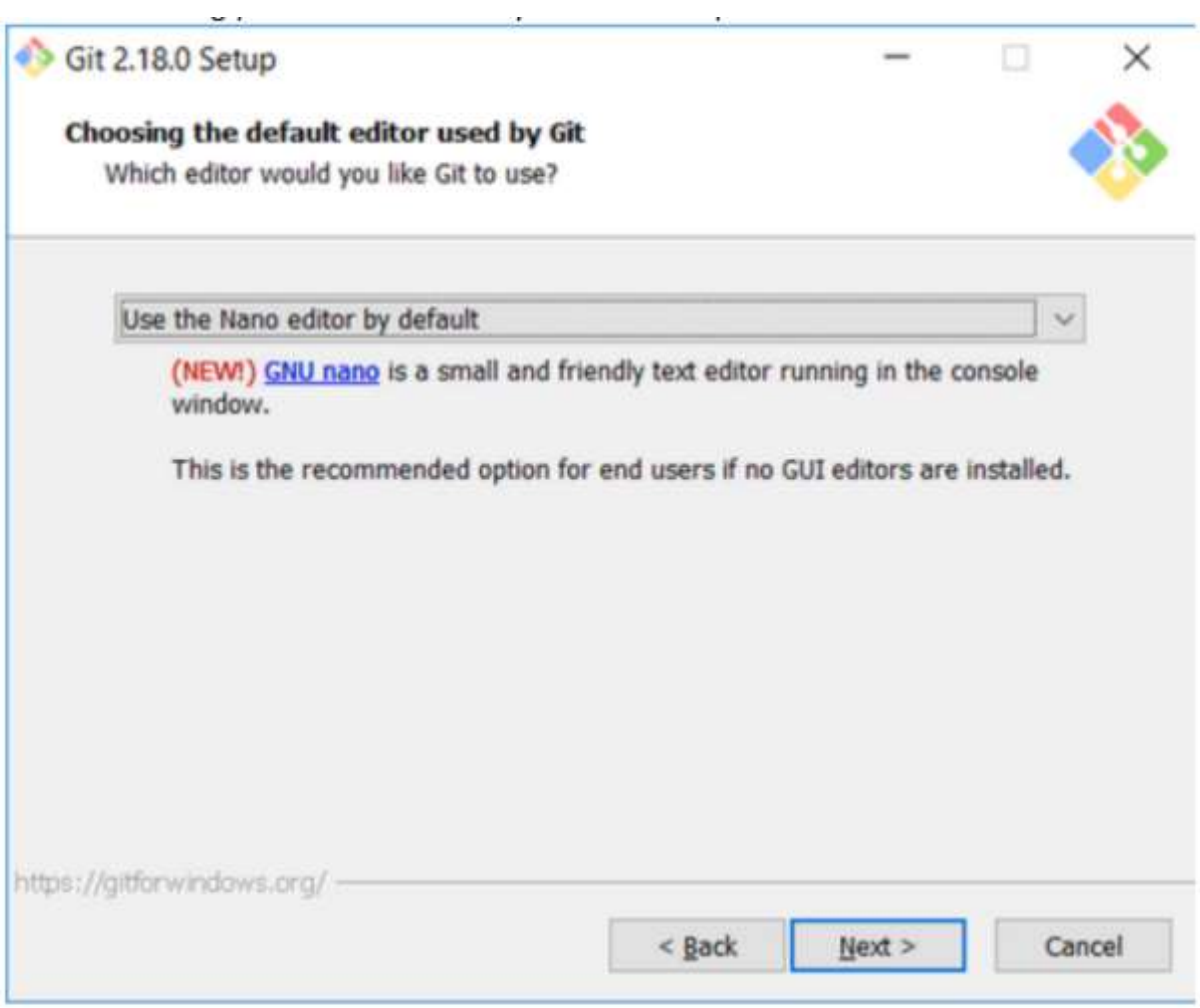
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- ✓ Additional icons
  - ✓ On the Desktop
- ✓ Windows Explorer integration
  - ✓ Git Bash Here
  - ✓ Git GUI Here
- ✓ Git LFS (Large File Support)
- ✓ Associate .git* configuration files with the default text editor
- ✓ Associate .sh files to be run with Bash
- ☐ Use a TrueType font in all console windows
- ☐ Check daily for Git for Windows updates

Current selection requires at least 229.3 MB of disk space.

https://gitforwindows.org/


Git 2.18.0 Setup

Choosing the default editor used by Git

Which editor would you like Git to use?

Use the Nano editor by default

(NEW!) GNU nano is a small and friendly text editor running in the console window.

This is the recommended option for end users if no GUI editors are installed.

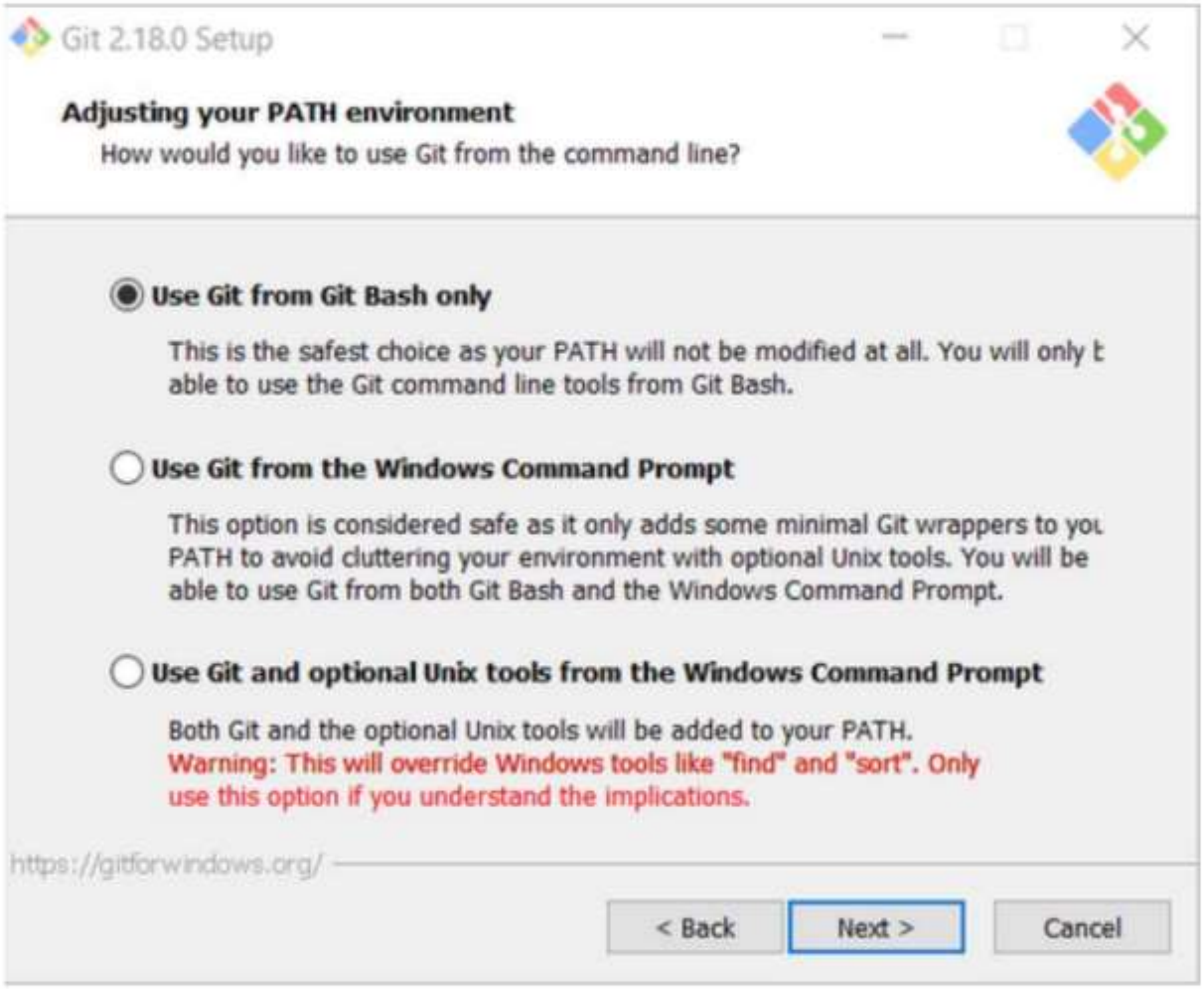
https://gitforwindows.org/

Three options are acceptable In the Adjusting your PATH screen

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



Adjusting your PATH environment

How would you like to use Git from the command line?

[✓ **Use Git from Git Bash only**]

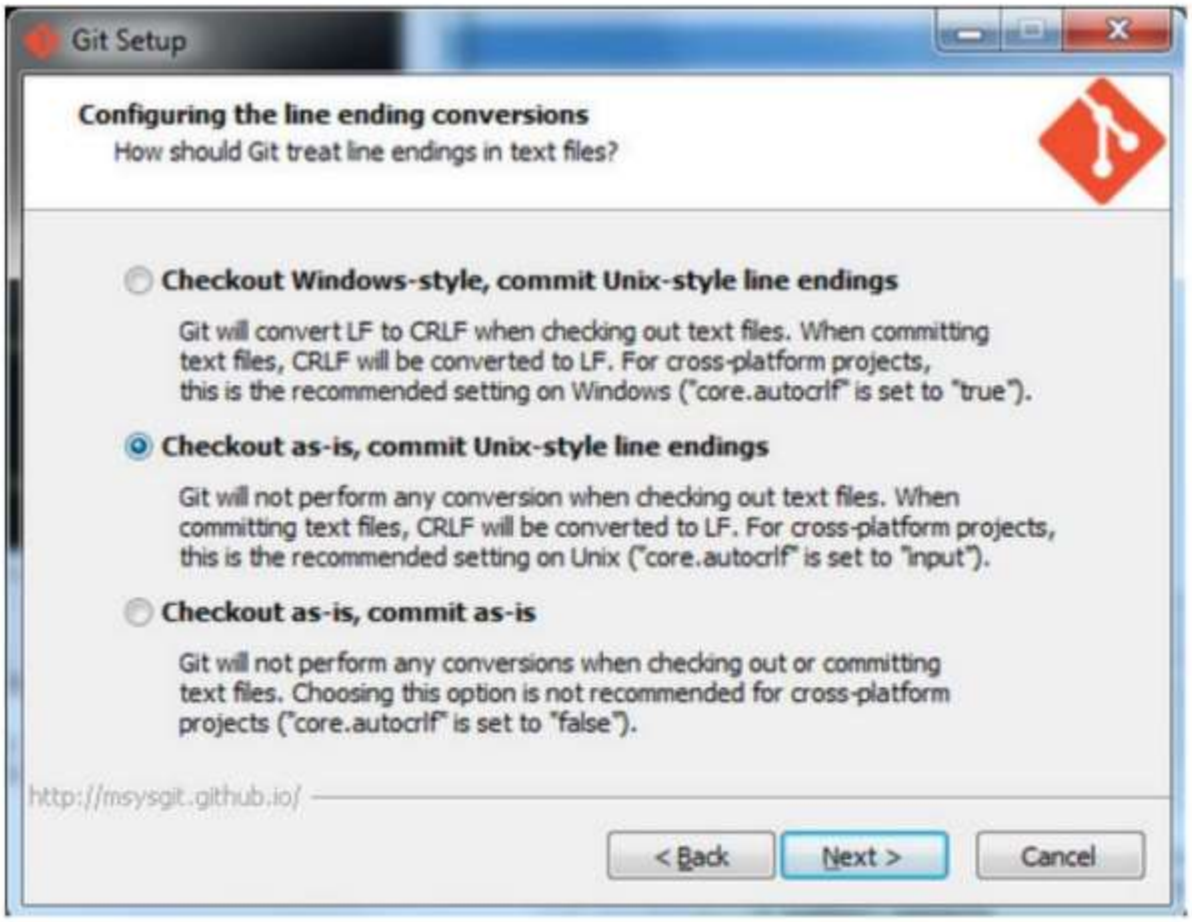
This is the safest choice as your PATH will not be modified at all. You will only b able to use the Git command line tools from Git Bash.

Use Git from the Windows Command Prompt

This option is considered safe as it only adds some minimal Git wrappers to you PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

Use Git and optional Unix tools from the Windows Command Prompt

Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.



Git Setup

Configuring the line ending conversions

How should Git treat line endings in text files?

( ) **Checkout Windows-style, commit Unix-style line endings**

Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

[✓ **Checkout as-is, commit Unix-style line endings**]

Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

( ) **Checkout as-is, commit as-is**

Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

Once Git is installed, there is some remaining custom configuration we 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 "GitGUI here". These commands permit you to launch either Git client. For now, select Git Bash here.
b. 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. 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. 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

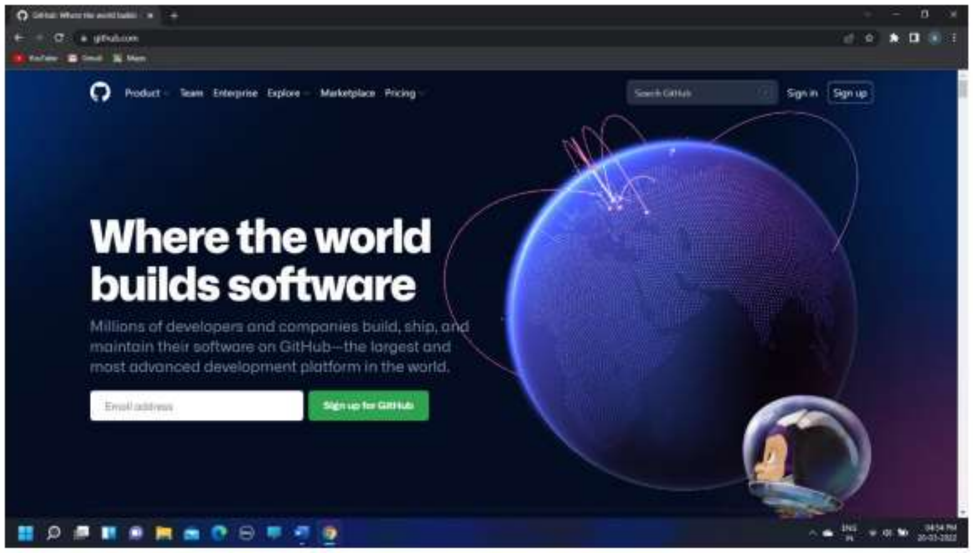
Aim: Setting up the git hub account.

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.

There are several types of accounts on GitHub. 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.

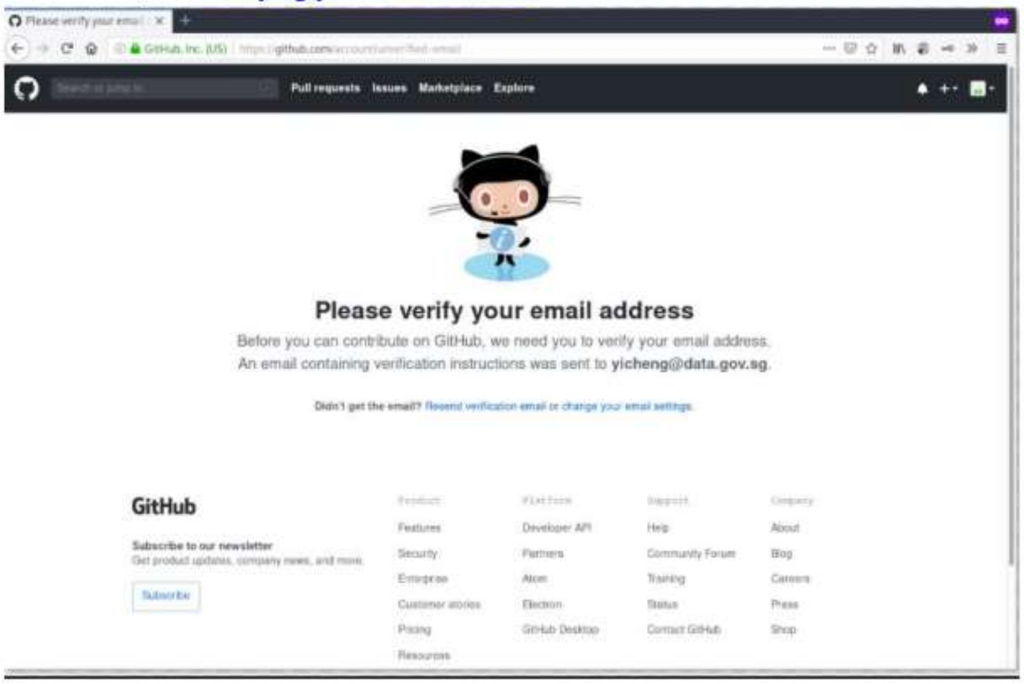
1. **Creating an account:** To sign up for an account on GitHub.com, navigate to https://github.com/ and follow the prompts.

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


Product
Team
Enterprise
Explore
Marketplace
Pricing
Sign in
Sign up
Where the world builds software
Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.
Email address
Sign up for GitHub

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

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

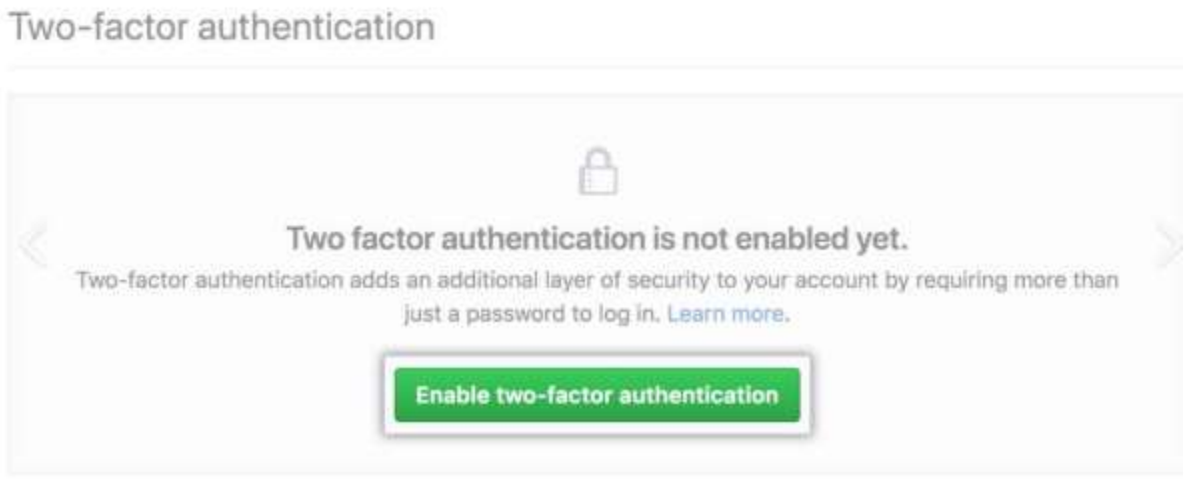


Please verify your email address

Before you can contribute on GitHub, we need you to verify your email address.
An email containing verification instructions was sent to **yicheng@data.gov.sg**.

Didn't get the email? Resend verification email or change your email settings.

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

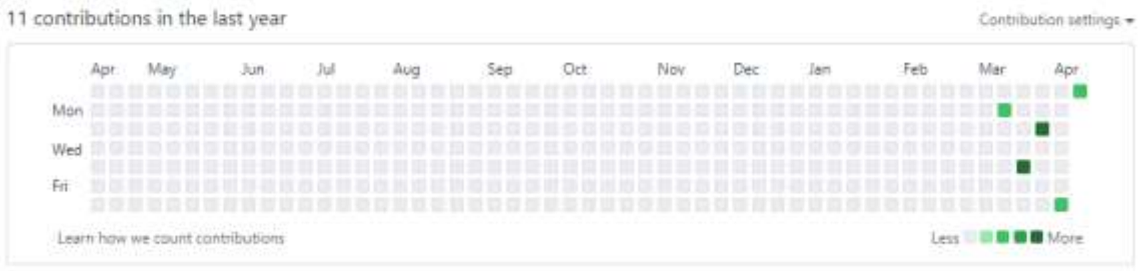


Two-factor authentication

Two factor authentication is not enabled yet.

Two-factor authentication adds an additional layer of security to your account by requiring more than just a password to log in. Learn more.

5. Viewing your GitHub profile and contribution graph: Your GitHub profile tells people the story of your work through the repositories and gists you’ve pinned, the organization memberships you’ve chosen to publicize, the contributions you’ve made, and the projects you’ve created. For more information, see “About your profile” and “Viewing contributions on your profile.”



11 contributions in the last year

EXPERIMENT – 3

Aim: Program to generate logs

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

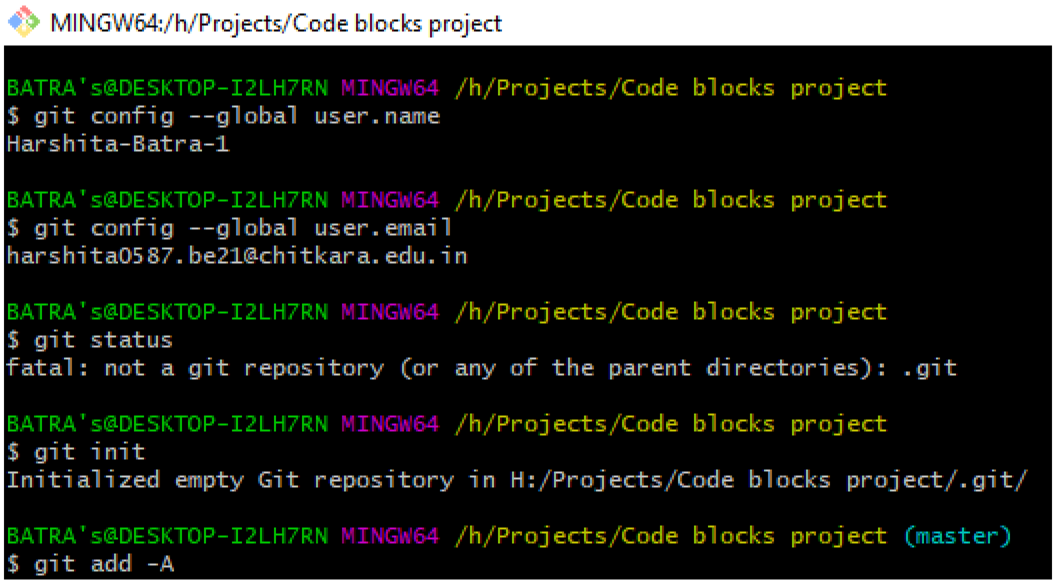
Basic Git commit

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 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 add command

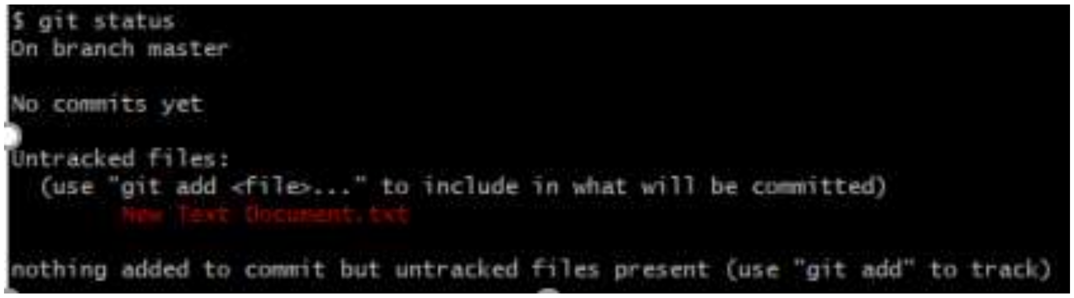
The git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run git commit.
Basic Git 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 basic git log command will display the most recent commits and the status of the head. It will use as:

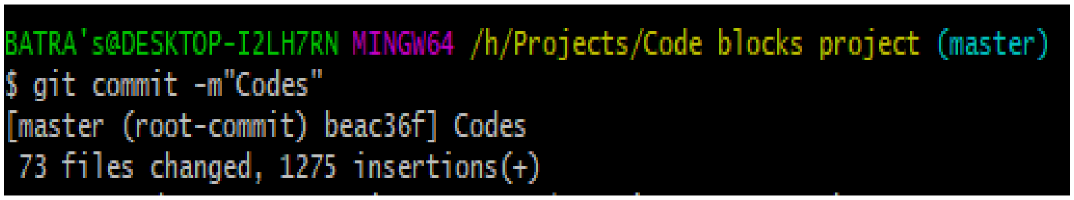


MINGW64:/h/Projects/Code blocks project

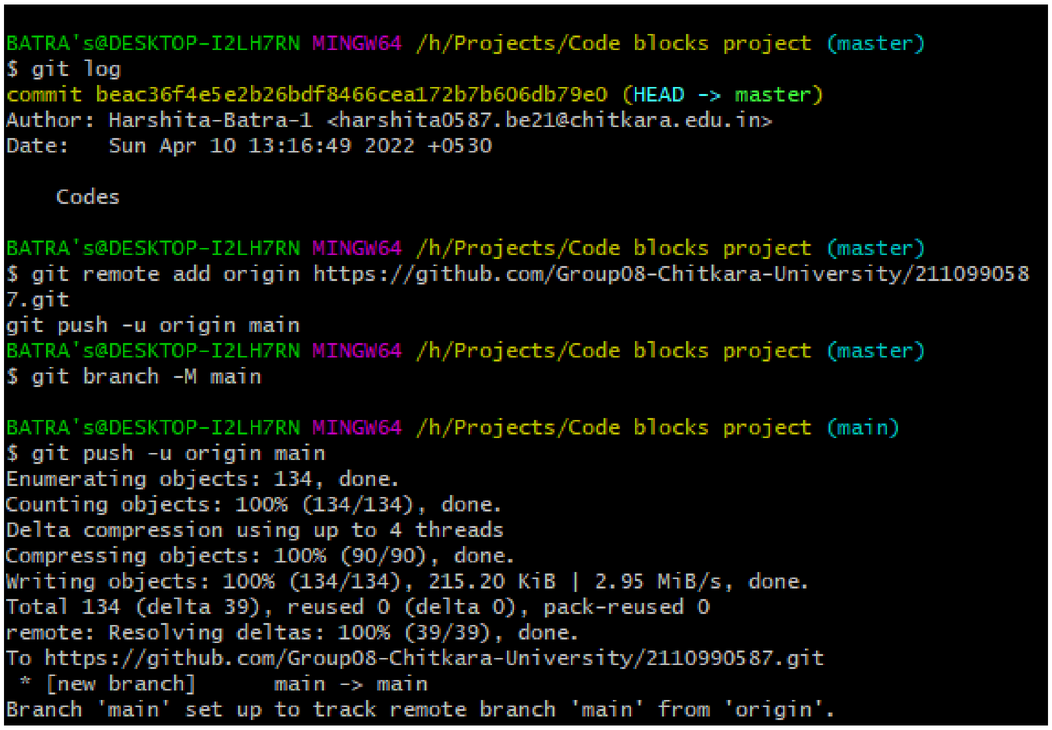
```
BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project
$ git config --global user.name
Harshita-Batra-1

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project
$ git config --global user.email
harshita0587.be21@chitkara.edu.in

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project
$ git status
fatal: not a git repository (or any of the parent directories): .git

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project
$ git init
Initialized empty Git repository in H:/Projects/Code blocks project/.git/

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project (master)
$ git add -A
```



```
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        New Text Document.txt

nothing added to commit but untracked files present (use "git add" to track)
```



```
BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project (master)
$ git commit -m"Codes"
[master (root-commit) beac36f] Codes
 73 files changed, 1275 insertions(+)
```



```
BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project (master)
$ git log
commit beac36f4e5e2b26bdf8466cea172b7b606db79e0 (HEAD -> master)
Author: Harshita-Batra-1 <harshita0587.be21@chitkara.edu.in>
Date:   Sun Apr 10 13:16:49 2022 +0530

    Codes

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project (master)
$ git remote add origin https://github.com/Group08-Chitkara-University/211099058
7.git
git push -u origin main
BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project (master)
$ git branch -M main

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project (main)
$ git push -u origin main
Enumerating objects: 134, done.
Counting objects: 100% (134/134), done.
Delta compression using up to 4 threads
Compressing objects: 100% (90/90), done.
Writing objects: 100% (134/134), 215.20 KiB | 2.95 MiB/s, done.
Total 134 (delta 39), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (39/39), done.
To https://github.com/Group08-Chitkara-University/2110990587.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

EXPERIMENT – 4

Aim: Create and visualize 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”

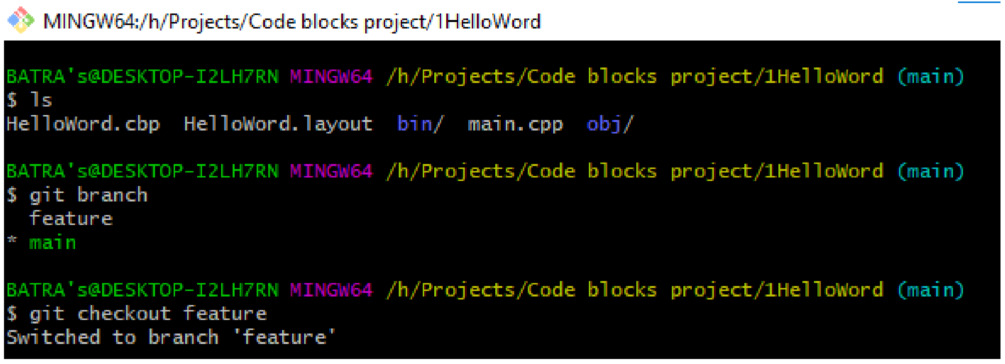
Visualizing Branches:

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, send it to stagging area and finally we can rollback to any previously saved version of this file.

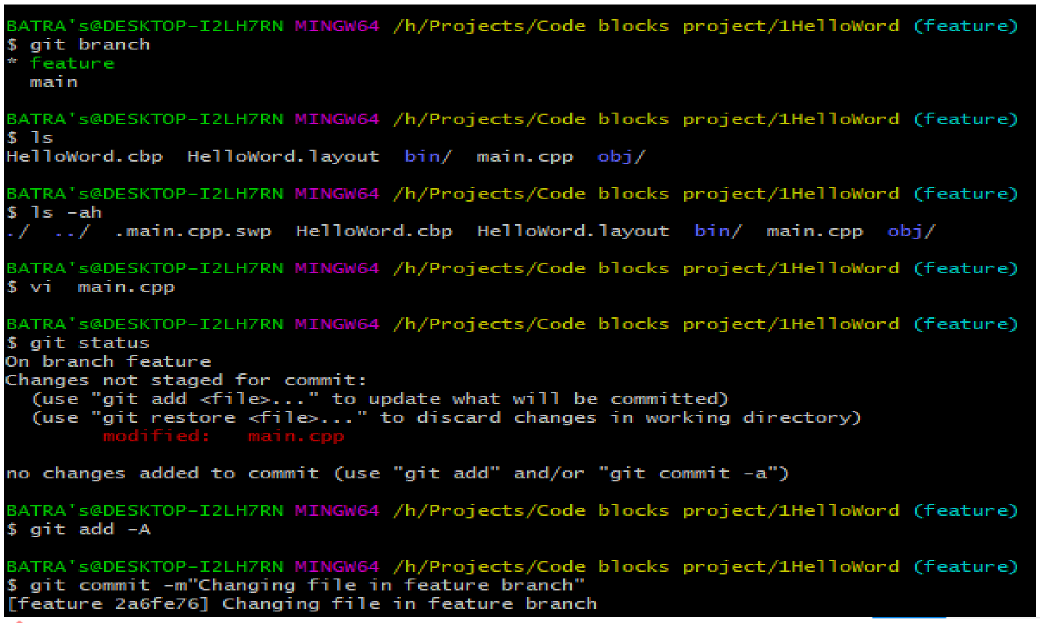
After this we will change the branch from activity1 to master, but when we switch back to master branch the file we created i.e “hello” will not be there. Hence the new file will not be shown in the master branch. In this way we can create and change different branches. We can also merge the branches 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.

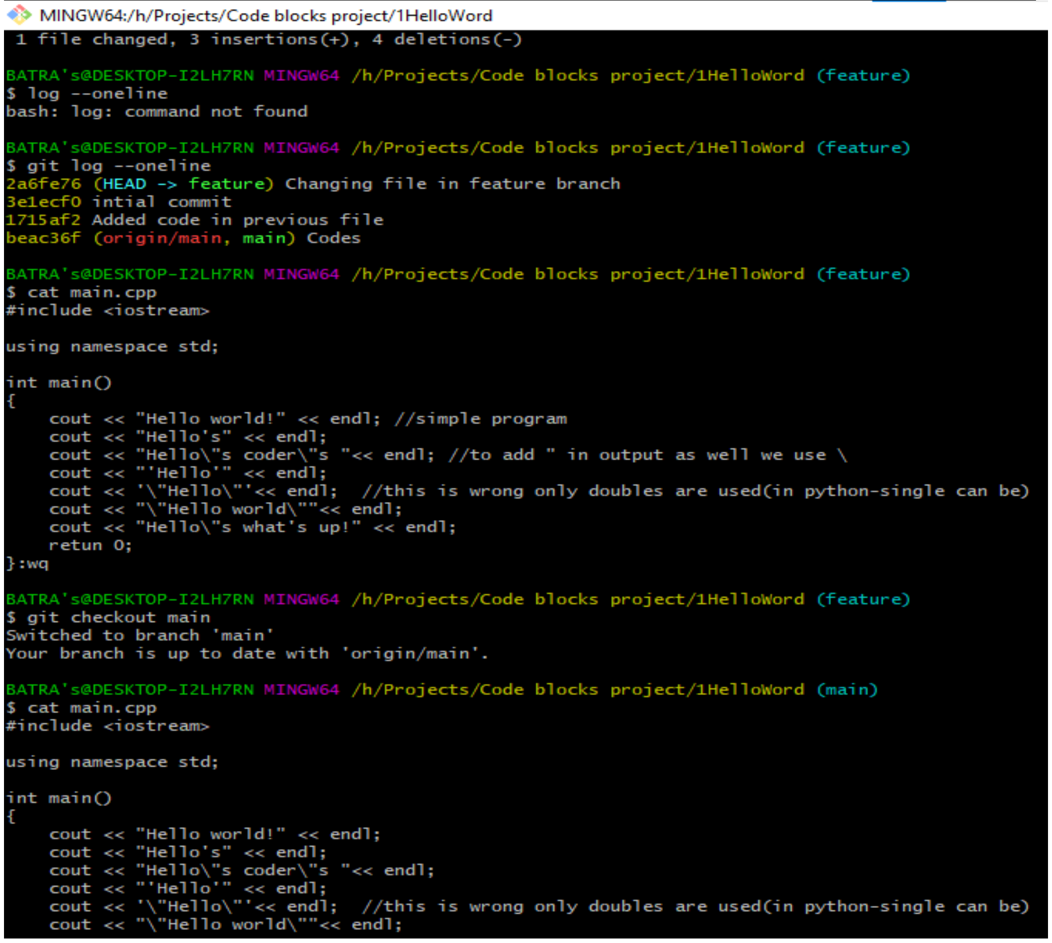


MINGW64:/h/Projects/Code blocks project/1HelloWord

```
BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (main)
$ ls
HelloWord.cbp  HelloWord.layout  bin/  main.cpp  obj/

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (main)
$ git branch
  feature
* main

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (main)
$ git checkout feature
Switched to branch 'feature'
```

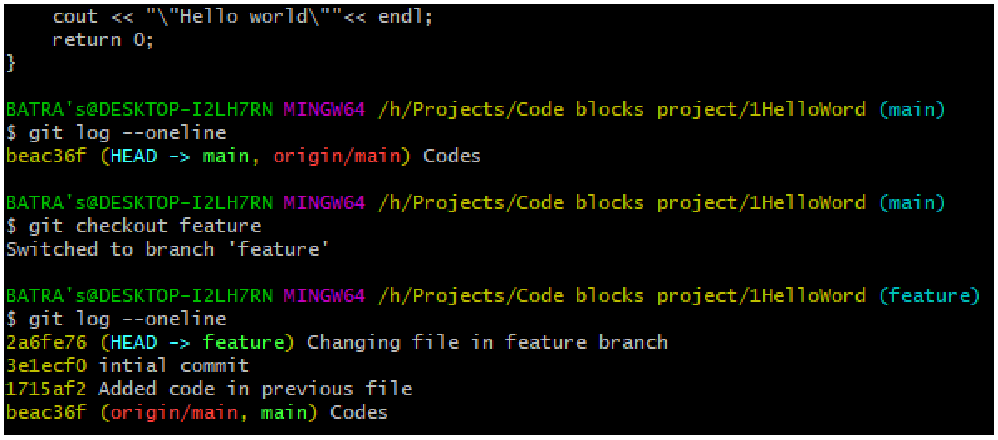


```
BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (feature)
$ git branch
* feature
  main

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (feature)
$ ls
HelloWord.cbp  HelloWord.layout  bin/  main.cpp  obj/

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (feature)
$ ls -ah
./  ../  .main.cpp.swp  HelloWord.cbp  HelloWord.layout  bin/  main.cpp  obj/

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (feature)
$ vi  main.cpp

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (feature)
$ git status
On branch feature
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.cpp

no changes added to commit (use "git add" and/or "git commit -a")

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (feature)
$ git add -A

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (feature)
$ git commit -m"Changing file in feature branch"
[feature 2a6fe76] Changing file in feature branch
```



MINGW64:/h/Projects/Code blocks project/1HelloWord

```
 1 file changed, 3 insertions(+), 4 deletions(-)

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (feature)
$ log --oneline
bash: log: command not found

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (feature)
$ git log --oneline
2a6fe76 (HEAD -> feature) Changing file in feature branch
3e1ecf0 intial commit
1715af2 Added code in previous file
beac36f (origin/main, main) Codes

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (feature)
$ cat main.cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl; //simple program
    cout << "Hello's" << endl;
    cout << "Hello\"s coder\"s "<< endl; //to add " in output as well we use \
    cout << "'Hello'" << endl;
    cout << '\"Hello\"'<< endl;  //this is wrong only doubles are used(in python-single can be)
    cout << "\"Hello world\""<< endl;
    cout << "Hello\"s what's up!" << endl;
    retun 0;
}:wq

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (feature)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (main)
$ cat main.cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    cout << "Hello's" << endl;
    cout << "Hello\"s coder\"s "<< endl;
    cout << "'Hello'" << endl;
    cout << '\"Hello\"'<< endl;  //this is wrong only doubles are used(in python-single can be)
    cout << "\"Hello world\""<< endl;
```

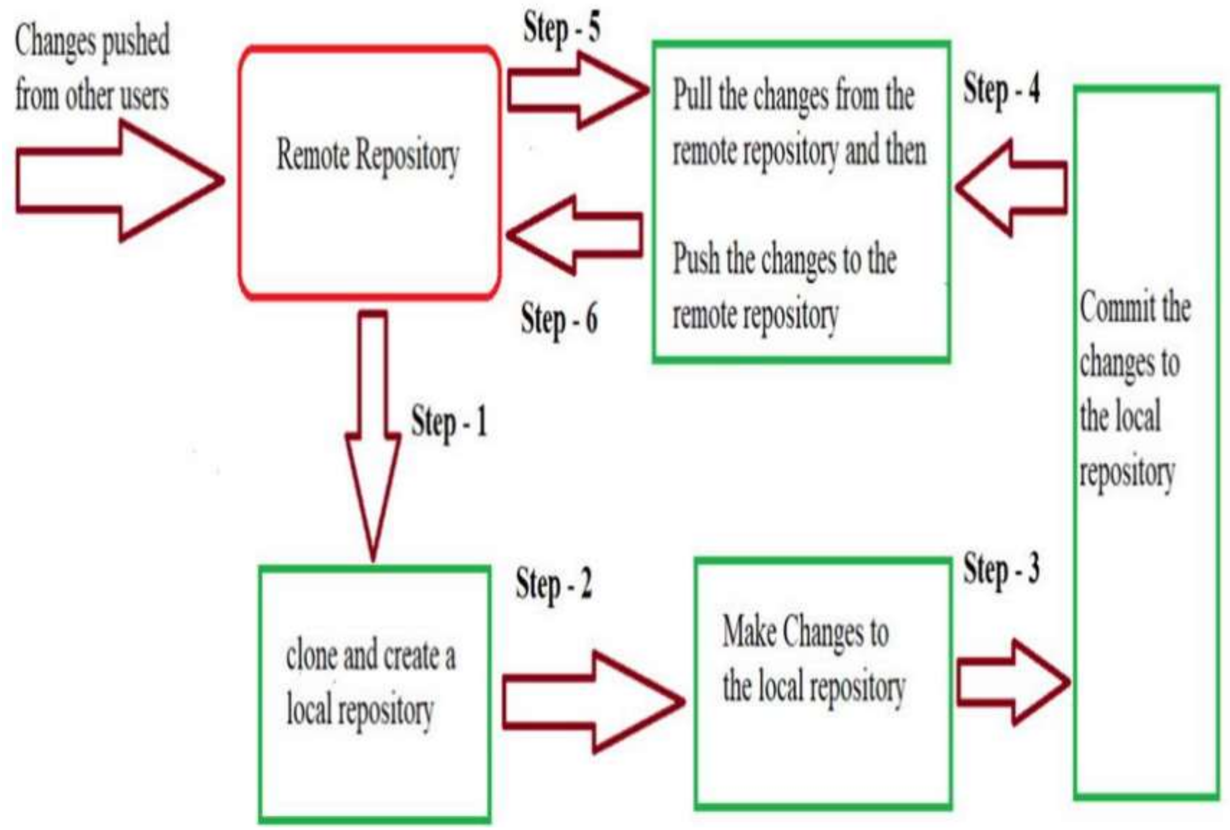


```
    cout << "\"Hello world\""<< endl;
    return 0;
}

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (main)
$ git log --oneline
beac36f (HEAD -> main, origin/main) Codes

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (main)
$ git checkout feature
Switched to branch 'feature'

BATRA's@DESKTOP-I2LH7RN MINGW64 /h/Projects/Code blocks project/1HelloWord (feature)
$ git log --oneline
2a6fe76 (HEAD -> feature) Changing file in feature branch
3e1ecf0 intial commit
1715af2 Added code in previous file
beac36f (origin/main, main) Codes
```

EXPERIMENT – 5

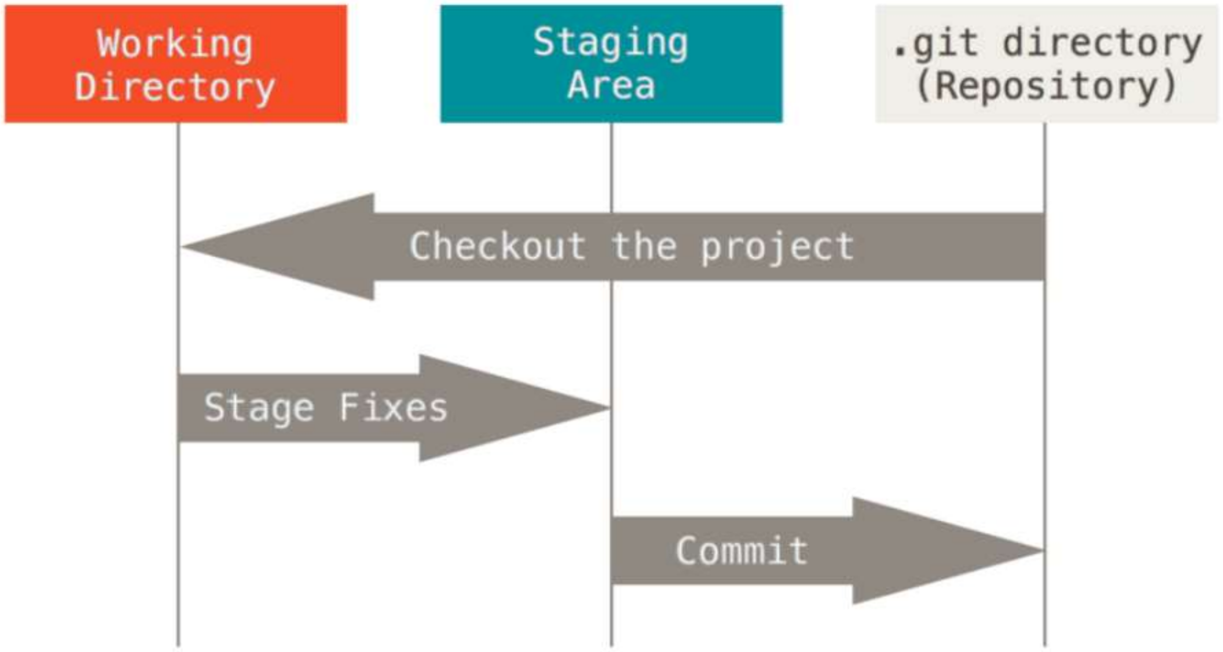
Aim: Git Lifecycle description

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-


Changes pushed from other users
Remote Repository
Step - 5
Pull the changes from the remote repository and then
Push the changes to the remote repository
Step - 6
Step - 4
Commit the changes to the local repository
Step - 1
clone and create a local repository
Step - 2
Make Changes to the local repository
Step - 3

- **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-



1. **Working Directory**

   Whenever we want to initialize aur 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.

2. **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>

Now since we have all the files that are to be tracked and are ready in the staging area, we are ready to commit aur 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