CHITKARA UNIVERSITY

.

Subject Name: Source Code Management
Subject Code: CS181
Cluster: Beta
Department: DCSE

Submitted To: Mr Monit Kapoor

Submitted By: Aastik Shukla

Roll no.21109900338

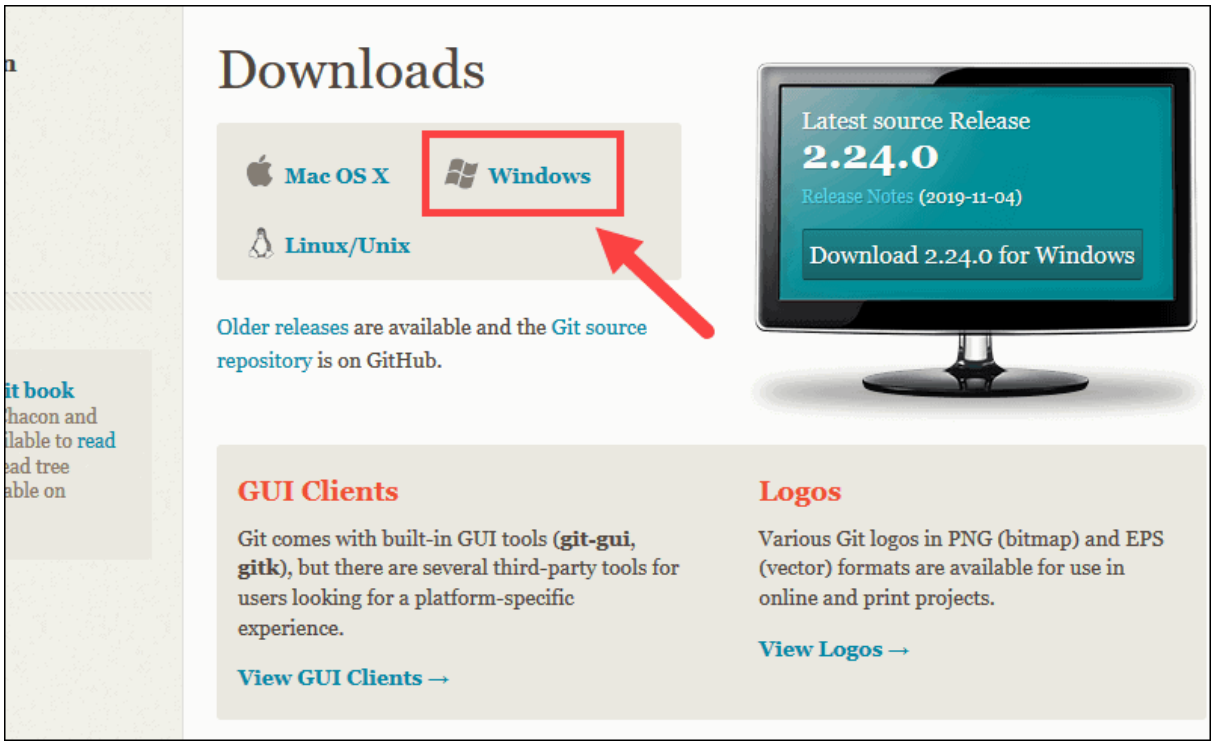
G8-A

Lists Of Tasks

Task – 1
Aim : Setting up git client.

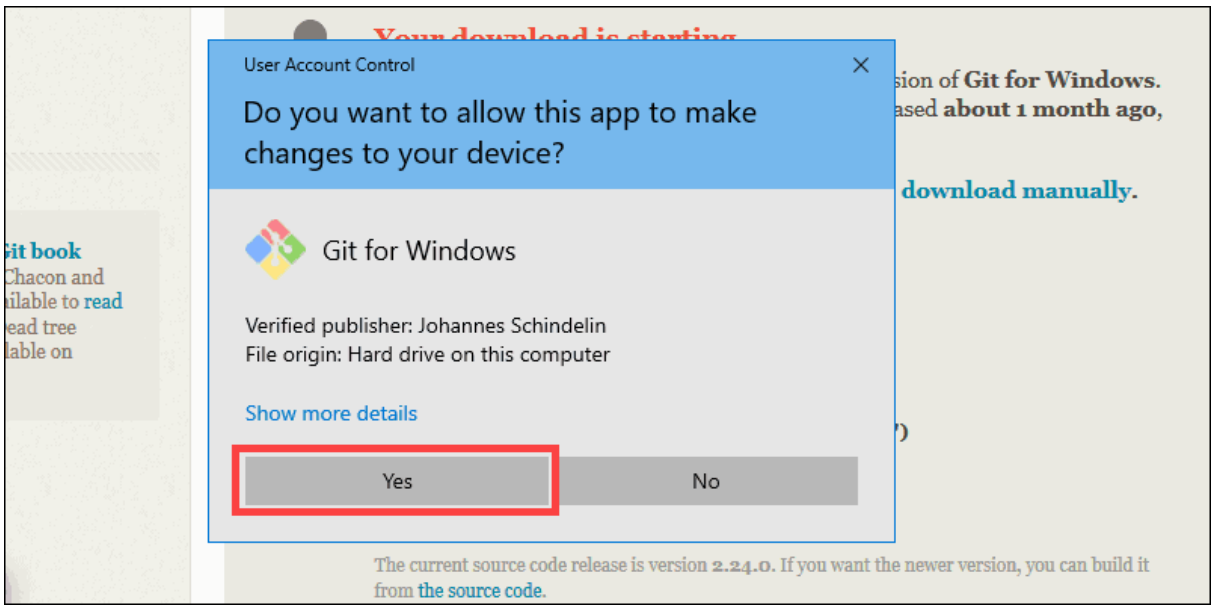
Procedure to install git for Windows:

1. Browse to the official Git website: https://git-scm.com/downloads
2. Click the download link for Windows and allow the download to complete.

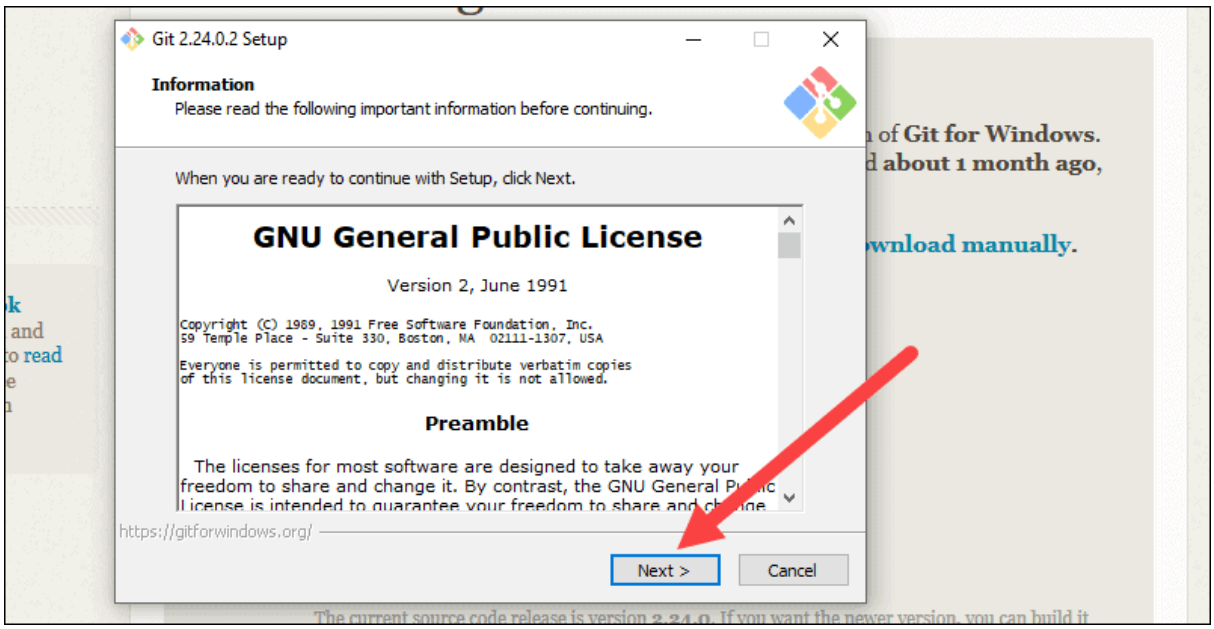


3. Browse to the download location (or use the download shortcut in your browser). Double-click the file to extract and launch the installer.

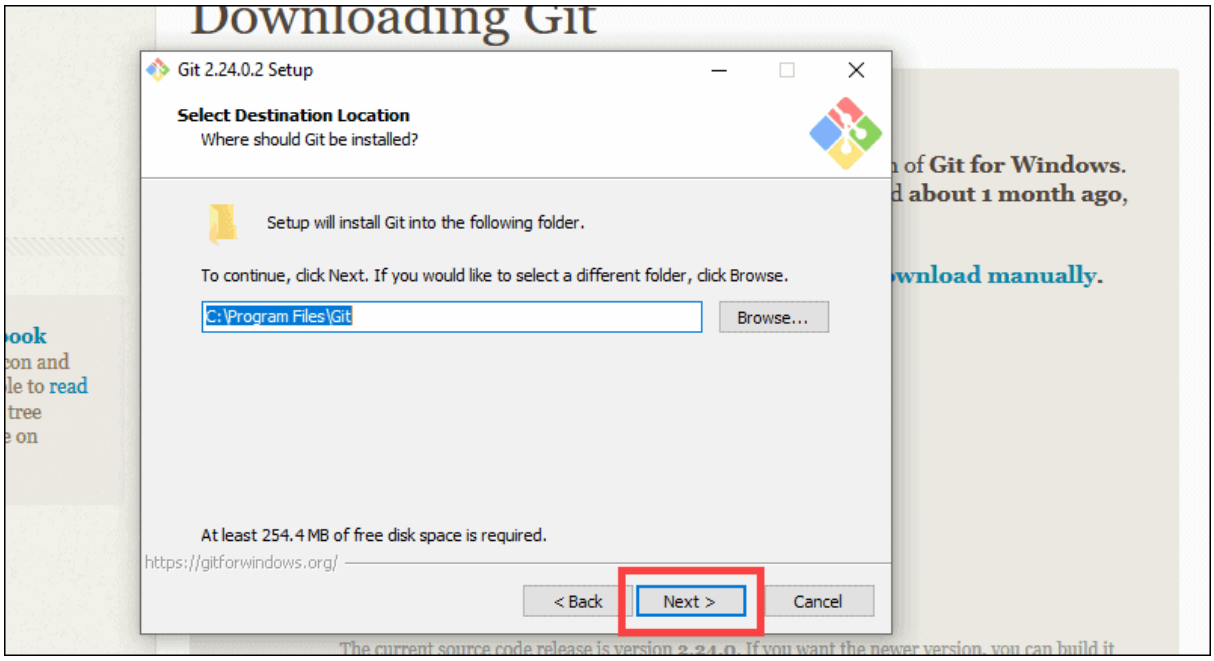
4. Allow the app to make changes to your device by clicking **Yes** on the User Account Control dialog that opens.



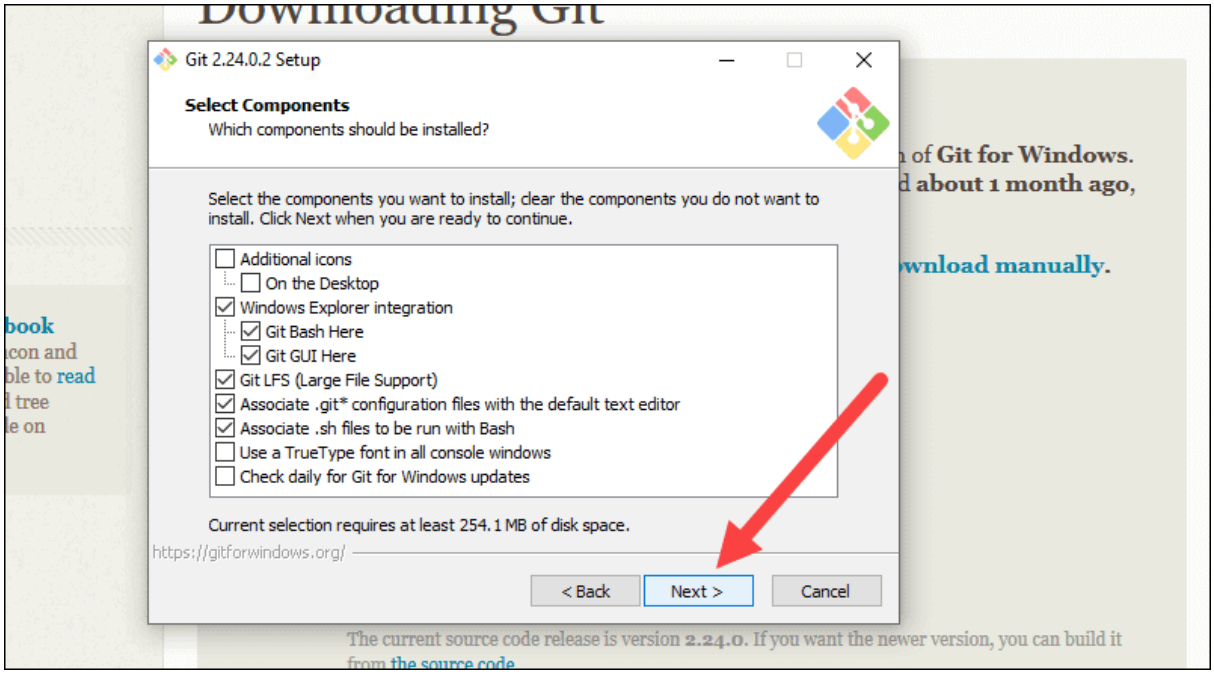
5. Read the GNU General Public License, and when you're ready to install, click **Next**.



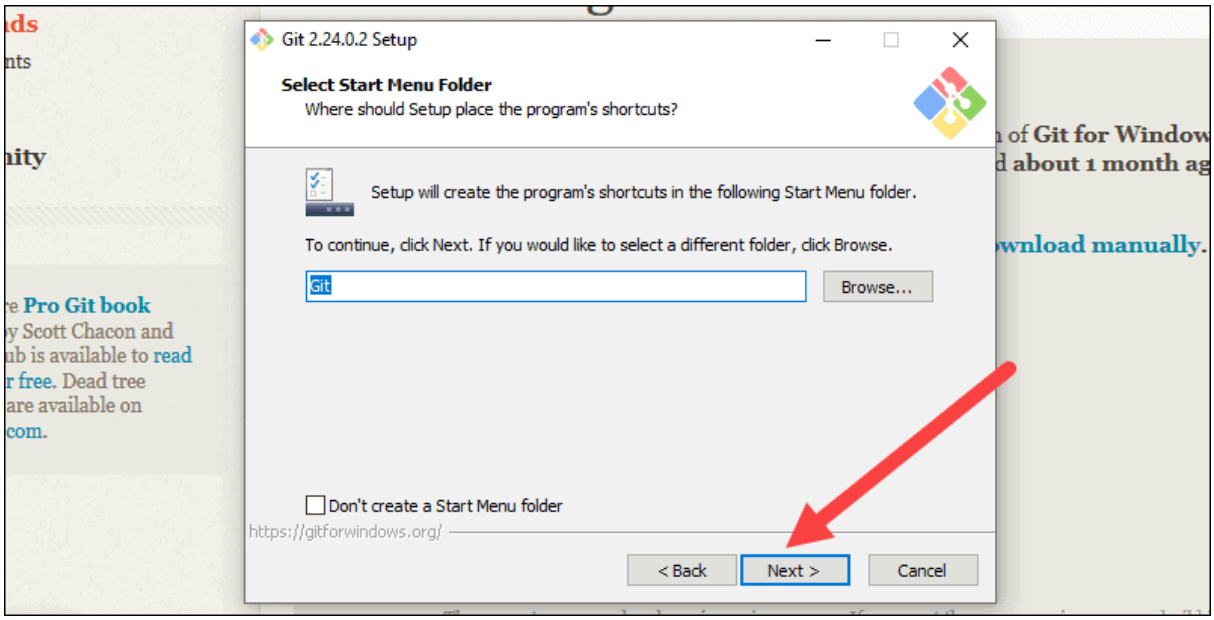
6. The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click **Next**.



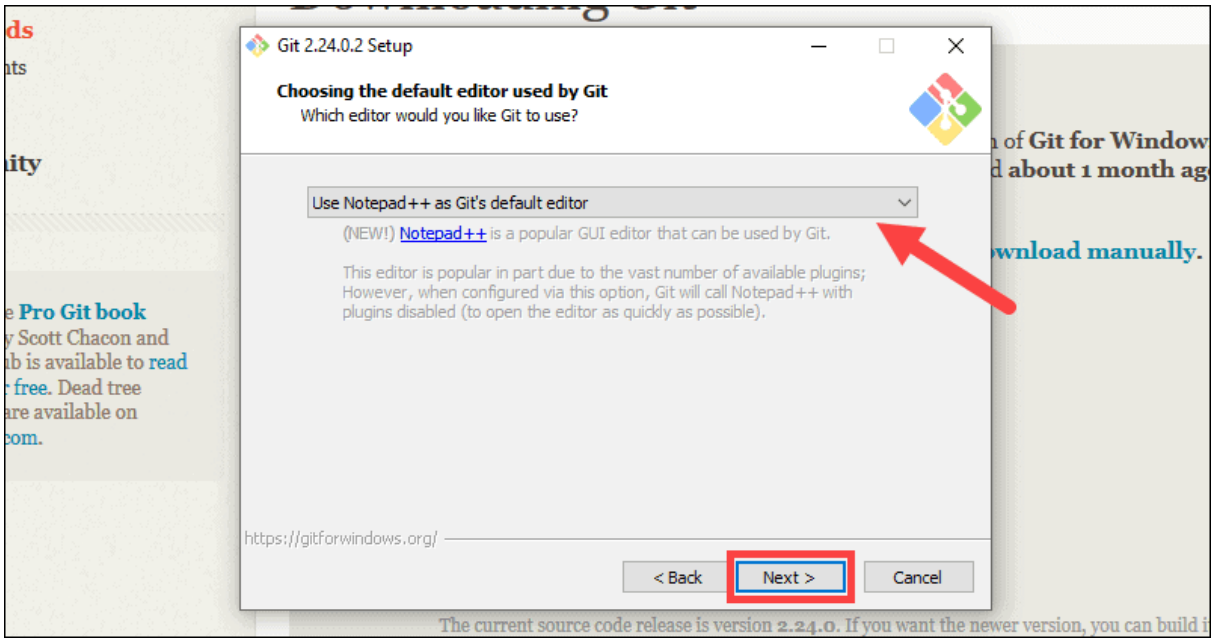
7. A component selection screen will appear. Leave the defaults unless you have a specific need to change them and click **Next**.



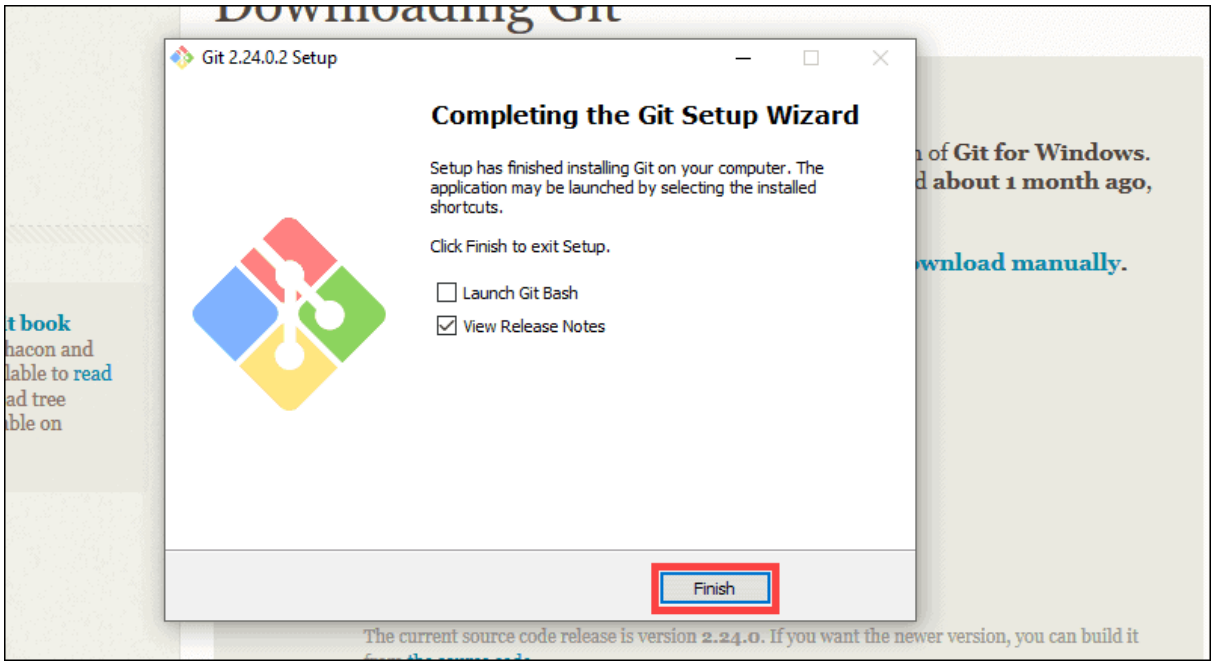
8. The installer will offer to create a start menu folder. Simply click **Next**.



9. Select a text editor you'd like to use with Git. Use the drop-down menu to select Notepad++ (or whichever text editor you prefer) and click **Next**.



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



Configuring Git to ignore certain files:

This part is extra important and required so that your repository does not get cluttered 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). Windows normally tries to place a file extension (.txt) on a file you create from File Explorer - and then it (by default) HIDES the file extension. To avoid this, create the file from within a useful editor (e.g. Notepad++ or Ultra Edit) and save the file without a file extension)

Edit this file and add the lines below (just copy/paste them from this screen); these are patterns for files to be ignored (taken from examples 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
common IDE-generated files and folders to ignore workspace.xml bin / out / .classpath # uncomment following for courses in which Eclipse .project files are not checked in # .project #ignore automatically generated files created by some common applications, operating systems
*.bak
*.log
*.ldb
* .DS Store*
* Thumbs.d
b
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

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 "GitGUI 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.exclude file 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@chitkara.edu.in"
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 use 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.

Task 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.

There are several types of accounts on GitHub. Every person who uses GitHub has their own user account, which can be part of multiple organizations 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."

2. Choosing your GitHub product

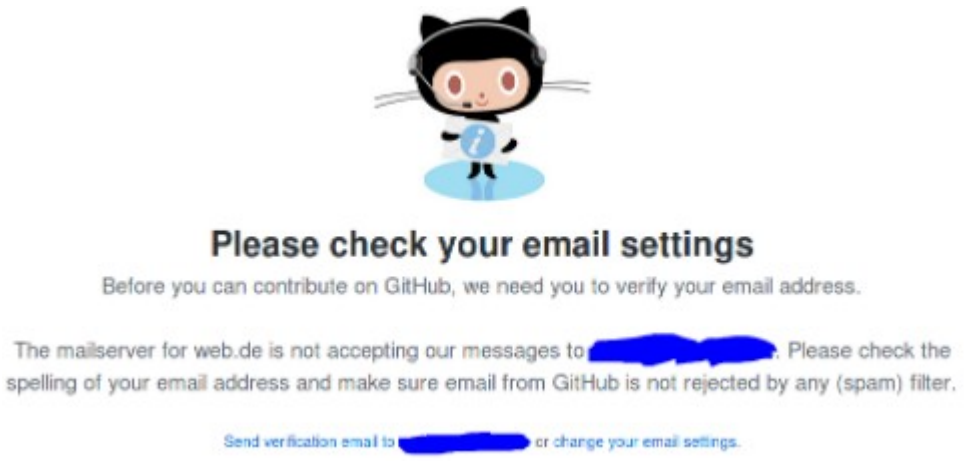
You can choose GitHub Free or GitHub Pro to get access to different features for your personal

account. You can upgrade at any time if you are unsure at first which product you want.

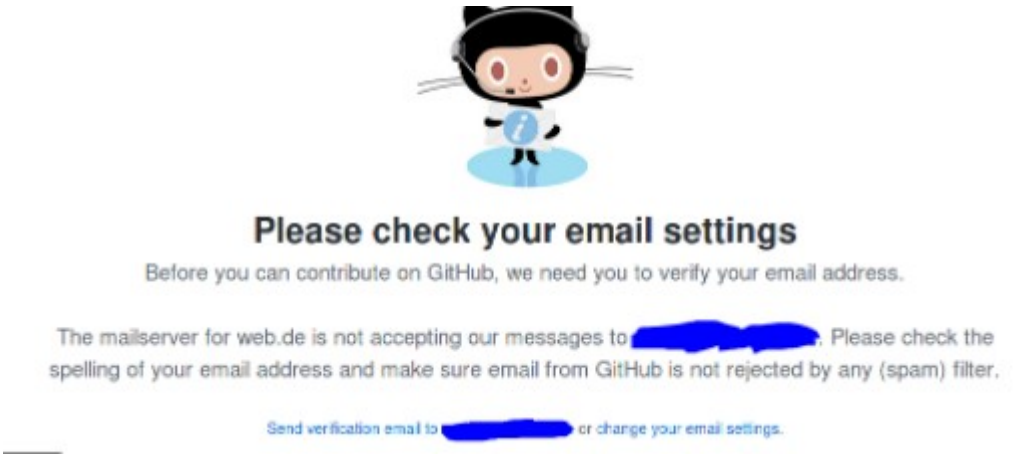
For more information on all of GitHub's plans, see "GitHub's products."

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".



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. 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 the safety of your account. For more information, see "About two-factor authentication."

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."

Task - 3

Aim:- Program to generate logs

Basic Git init:

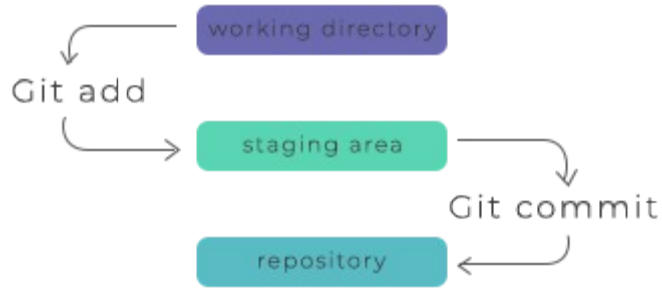
The git init is one way to start a new project with Git. To start a repository, use either git init or git clone - not both.
To initialize a repository, Git creates a hidden directory called .git. That directory stores all of the objects and refs that Git uses and creates as a part of your project's history. This hidden .git directory is what separates a regular directory from a Git repository.

Basic Git Status:-

The git status command shows the state of the working directory and the staging area. It allows you to see staged changes and the files that aren’t being tracked by Git. The Status output does not display any information about the committed project history. For this purpose, use the git log command. The git status simply displays what has been going on with git add and git commit commands.

Basic 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



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:

```
HP@DESKTOP-7I2PPJ7 MINGW64 /d
$ cd G8-210990025
bash: cd: G8-210990025: No such file or directory

HP@DESKTOP-7I2PPJ7 MINGW64 /d
$ cd G8_2110990025

HP@DESKTOP-7I2PPJ7 MINGW64 /d/G8_2110990025 (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file.docx
        ~$file.docx

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

HP@DESKTOP-7I2PPJ7 MINGW64 /d/G8_2110990025 (master)
$ git log
commit 0ddfa9d7547d75223378f8be94db7c7bbcbbe71b (HEAD -> master, origin/master)
Author: Aastik2110990025 <aastik0025.be21@chitkara.edu.in>
Date:   Sun Apr 10 14:26:51 2022 +0530

    new check
```

Task-4

Aim:- Create and visualize branches in Git

How to Create Branches:

Git makes creating and managing branches very easy. In fact, the power and flexibility of its branching model is one of the biggest advantages of Git!

There are a couple of different use cases when creating branches in Git. Let's look at each of them in turn.

When you do a pull request on a branch, you can continue to work on another branch and make another pull request on this other branch.

Before creating a new branch, pull the changes from upstream. Your master needs to be up to date.

$ git pull

Create the branch on your local machine and switch in this branch :

$ git checkout -b [name_of_your_new_branch]

Push the branch on GitHub :

$ git push origin [name_of_your_new_branch]

When you want to commit something in your branch, be sure to be in your branch. Add -u parameter to set-upstream.

You can see all the branches created by using :

$ git branch -a

Which will show :

* approval_messages

  master

  master_clean

Add a new remote for your branch :

$ git remote add [name_of_your_remote] [name_of_your_new_branch]

Push changes from your commit into your branch :

$ git push [name_of_your_new_remote] [url]

Update your branch when the original branch from official repository has been updated :

$ git fetch [name_of_your_remote]
Then you need to apply to merge changes if your branch is derivated from develop you need to do :

$ git merge [name_of_your_remote]/develop

Delete a branch on your local filesystem :

$ git branch -d [name_of_your_new_branch]

To force the deletion of local branch on your filesystem :

$ git branch -D [name_of_your_new_branch]

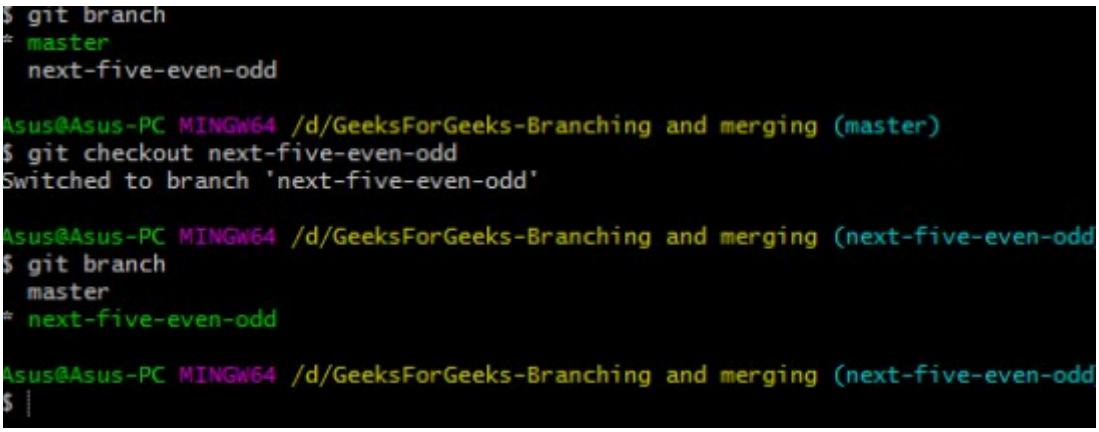
Delete the branch on GitHub :

$ git push origin :[name_of_your_new_branch]

If you want to change default branch, it's so easy with GitHub, in your fork go into Admin and in the drop-down list default branch choose what you want.

If you want create a new branch:

$ git branch <name_of_your_new_branch>

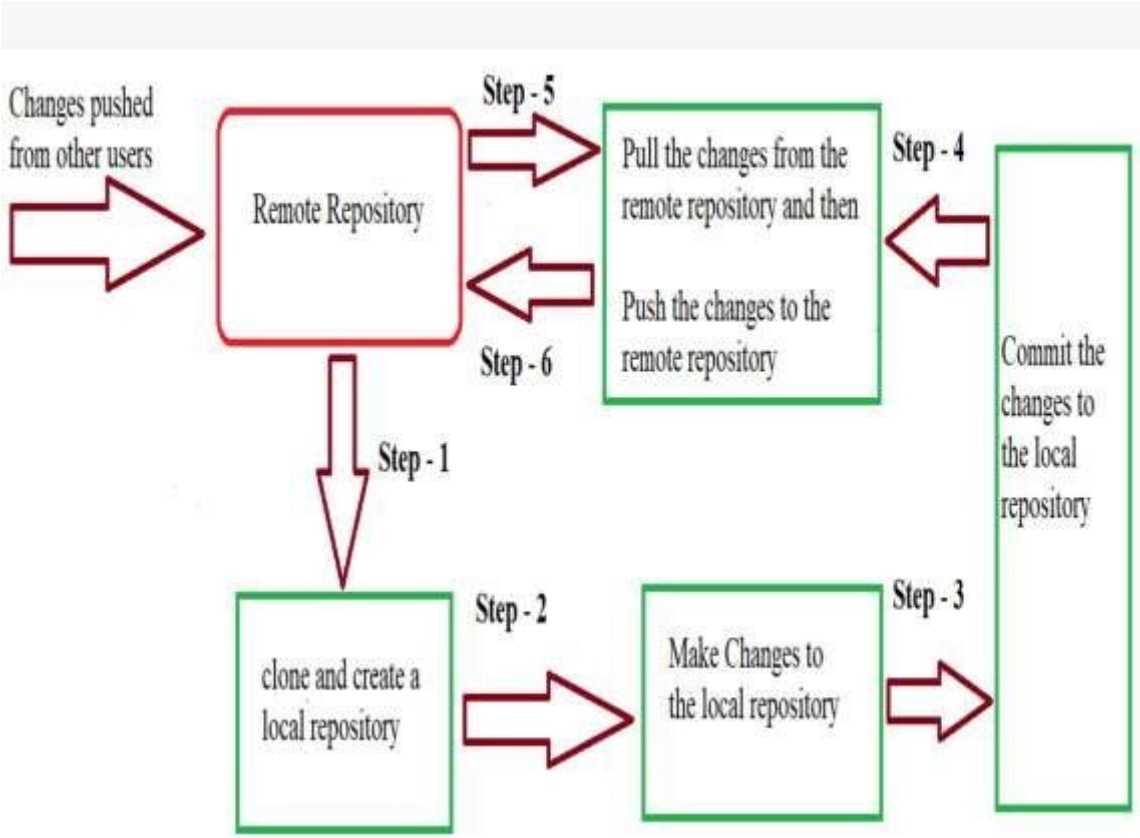


Task-5:

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 Life Cycle that

git has and understand more about its life cycle. Let us see some of the basic steps that we follow while working with Git –



- ***In Step – 1****, We first clone any of the code residing in the remote reposit ory to make our won local repository.*
- ***In Step-2*** *we edit the files that we have cloned in our local repository an d make the necessary changes in it.*
- ***In Step-3*** *we commit our changes by first adding them to our staging ar ea and committing them with a commit message.*
- ***In Step – 4 and Step-5*** *we first check whether there are any of the chang es done in the remote repository by some other users and we first pull that c hanges.*
- *If there are no changes we directly proceed with* ***Step – 6*** *in which we p ush 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

- Staging Area
- Git Directory

Let us understand in detail about each state.

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.

git init

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.

// to specify which file to add to the

staging area git add <filename>

// to add all files of the working directory to the

staging area git add .

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. *git commit -m <Commit Message>*