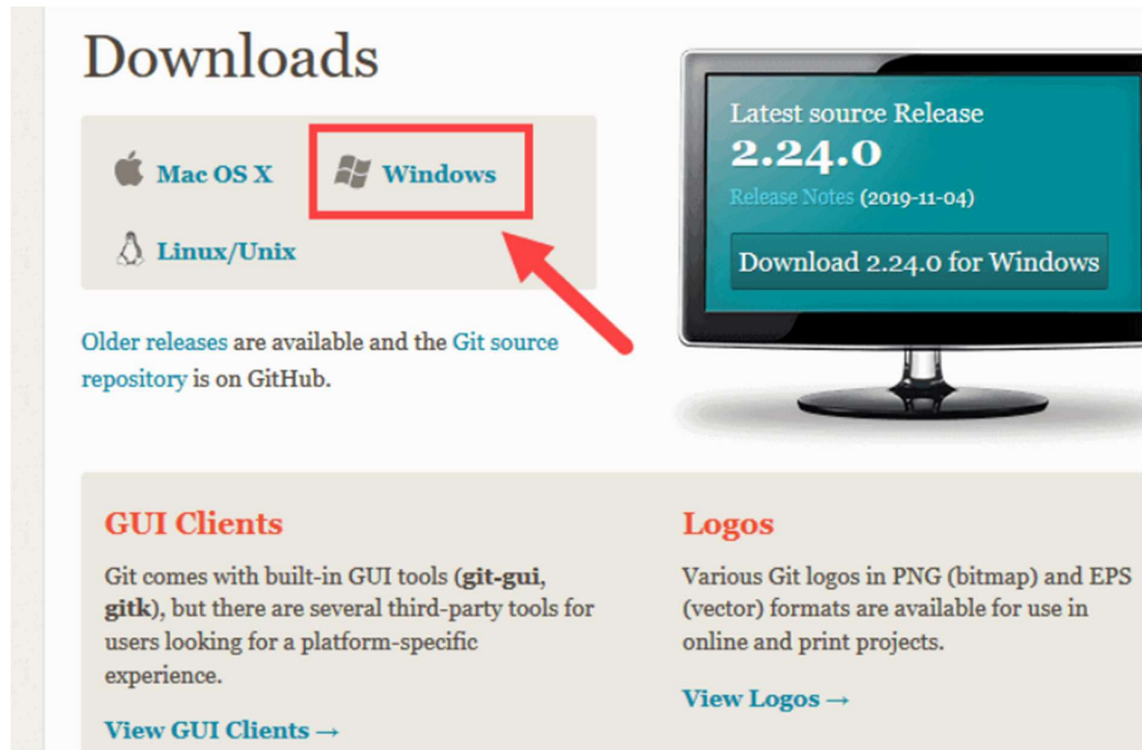
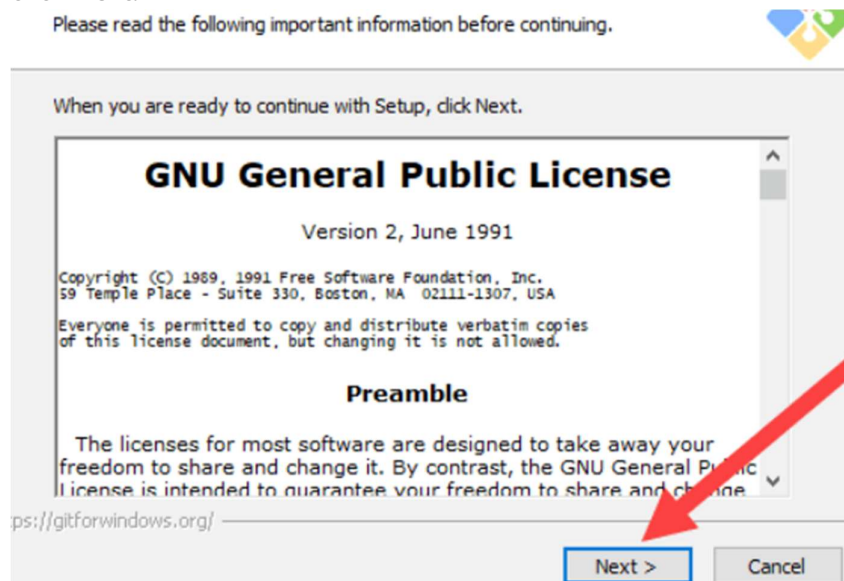


1) SETTING UP GIT CLIENT:

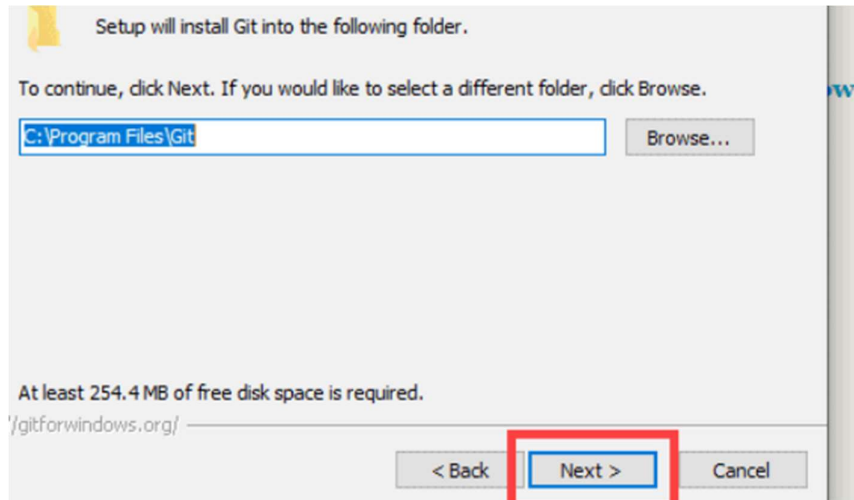
1. Go to the official Git website: <https://git-scm.com/downloads>
2. Click the download link for Windows and allow the download to complete.



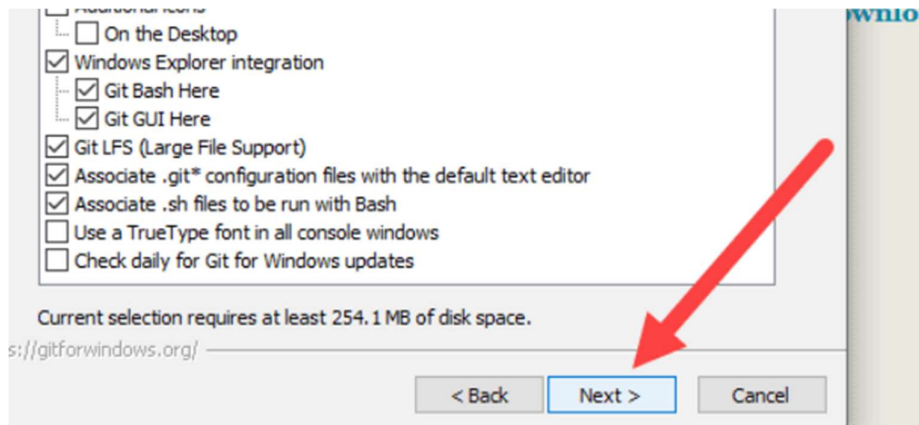
3. Browse to the download location (or use the download shortcut in your browser). Double-click the file to extract and launch the installer.
4. Review the GNU General Public License, and when you're ready to install, click **Next**.



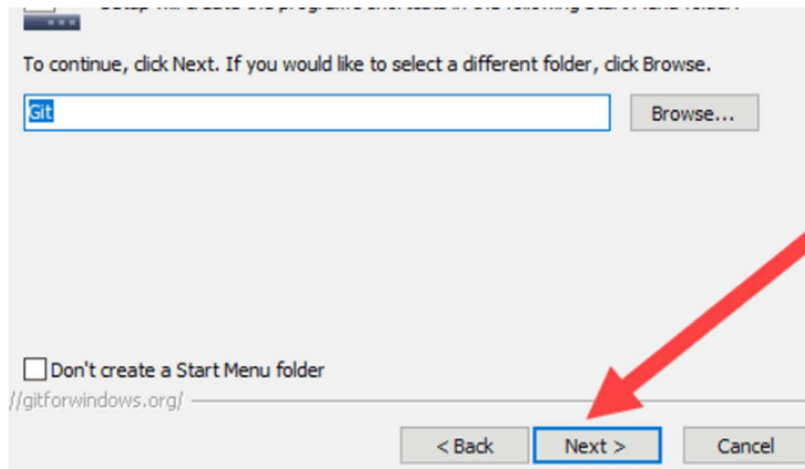
5. The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click **Next**.



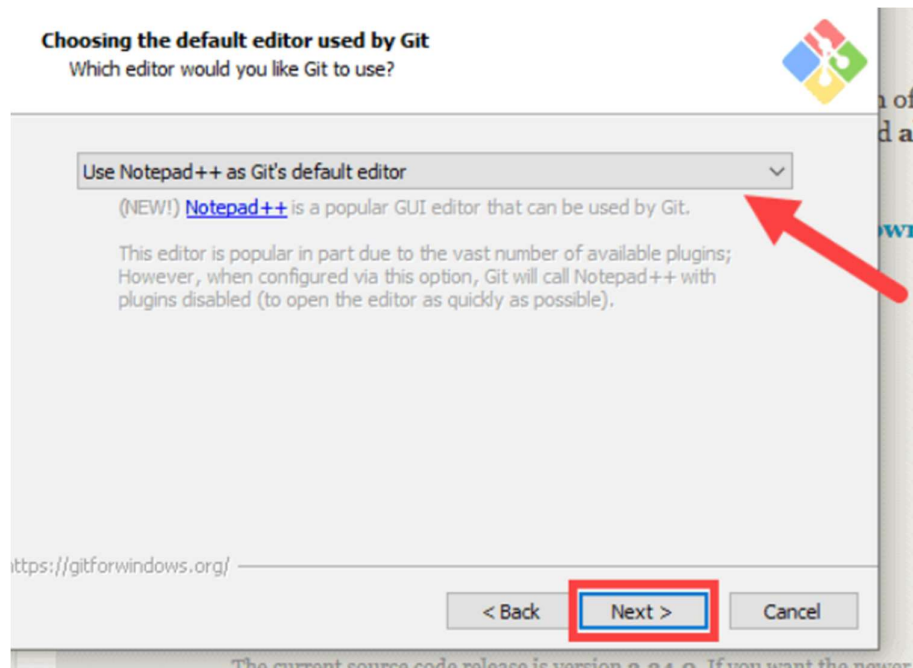
6. A component selection screen will appear. Leave the defaults unless you have a specific need to change them and click **Next**.



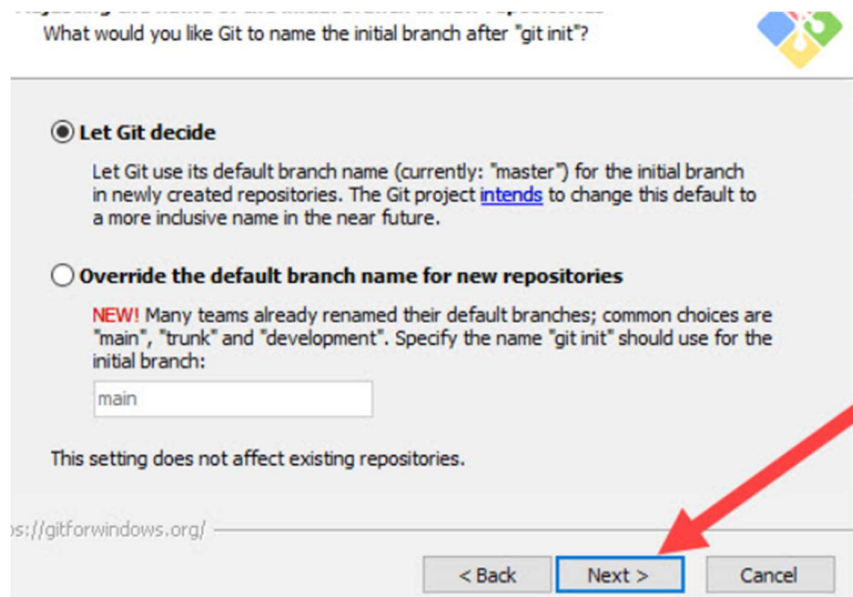
7. The installer will offer to create a start menu folder. Simply click **Next**.



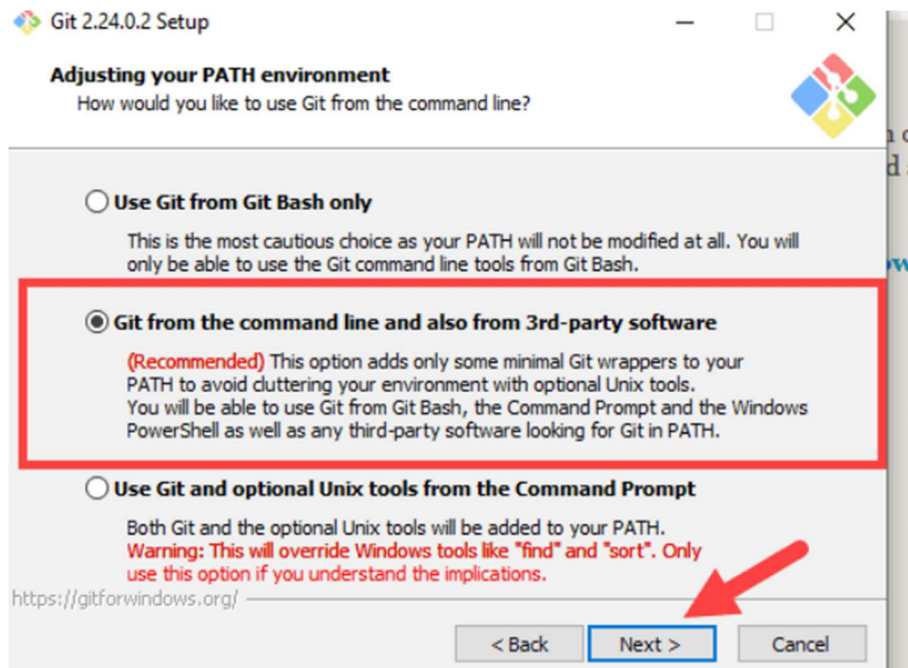
8. Select a text editor you'd like to use with Git. Use the drop-down menu to select Notepad++ (or whichever text editor you prefer) and click **Next**.



9. The next step allows you to choose a different name for your initial branch. The default is 'master.' Unless you're working in a team that requires a different name, leave the default option and click **Next**.



10. This installation step allows you to change the **PATH environment**. The **PATH** is the default set of directories included when you run a command from the command line. Leave this on the middle (recommended) selection and click **Next**.



11. The installer now asks which SSH client you want Git to use. Git already comes with its own SSH client, so if you don't need a specific one, leave the default option and click **Next**.

12. The next option relates to server certificates. Most users should use the default. If you're working in an Active Directory environment, you may need to switch to Windows Store certificates. Click **Next**.

13. The next selection converts line endings. It is recommended that you leave the default selection. This relates to the way data is formatted and changing this option may cause problems. Click **Next**.

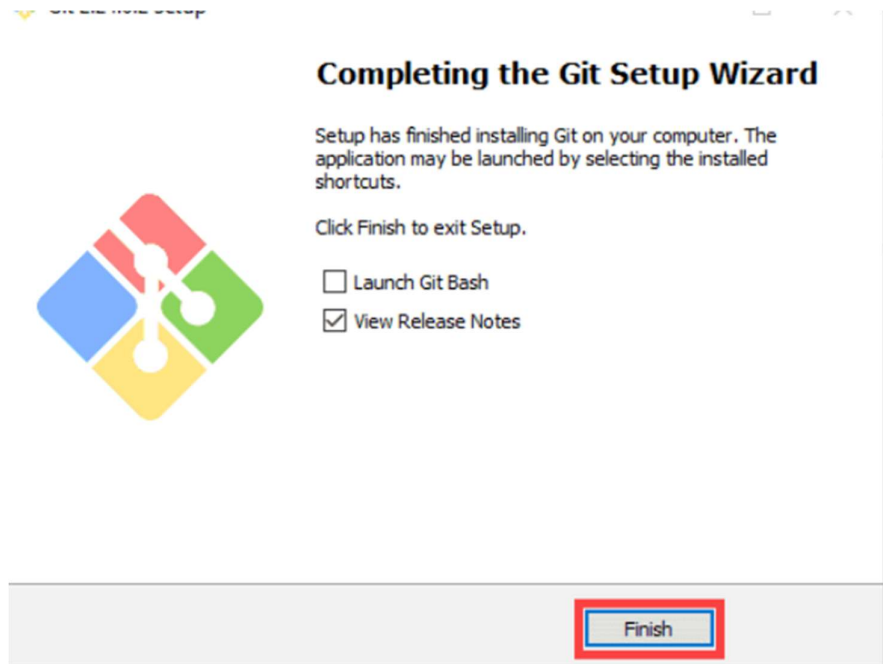
14. Choose the terminal emulator you want to use. The default MinTTY is recommended, for its features. Click **Next**.

15. Next you should choose which credential helper to use. Git uses credential helpers to fetch or save credentials. Leave the default option as it is the most stable one, and click **Next**.

16. The default options are recommended, however this step allows you to decide which extra option you would like to enable. If you use symbolic links, which are like shortcuts for the command line, tick the box. Click **Next**.

17. Depending on the version of Git you're installing, it may offer to install experimental features. Unless you are feeling adventurous, leave them unchecked and click **Install**.

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

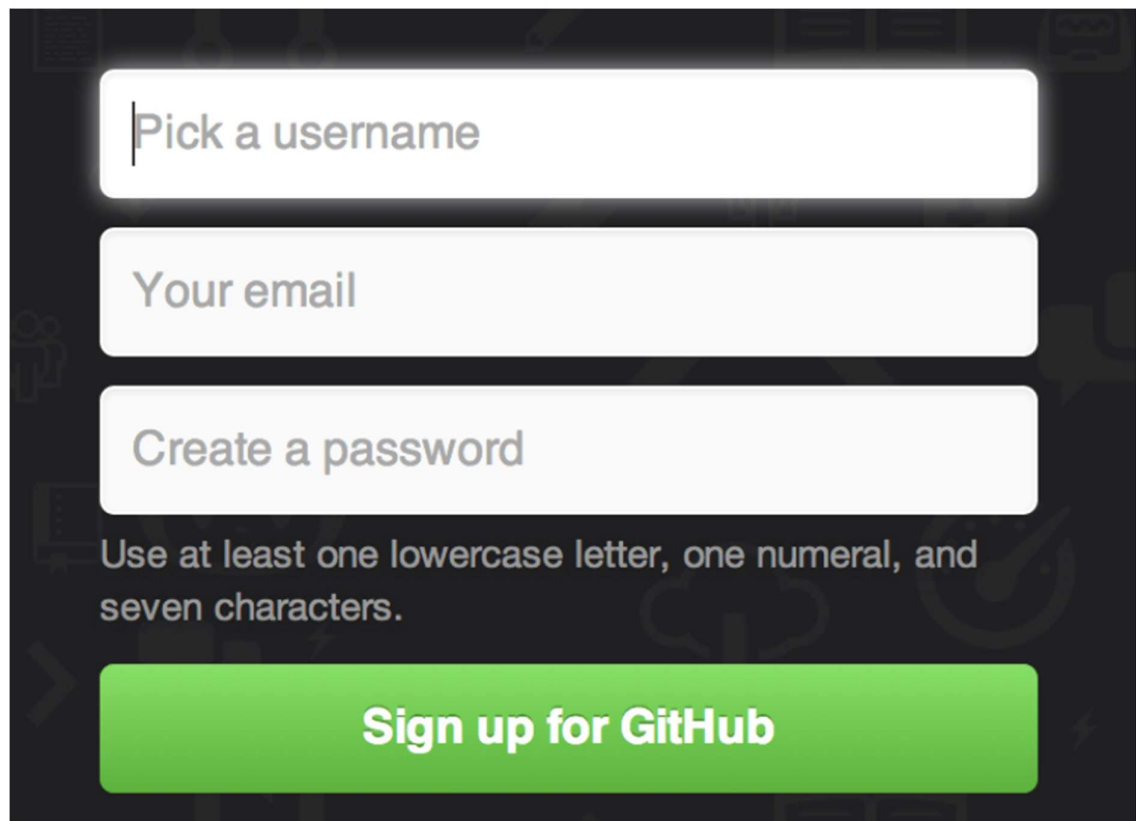


2) SETTING UP GIT HUB ACCOUNT:

GitHub is the single largest host for Git repositories, and is the central point of collaboration for millions of developers and projects. A large percentage of all Git repositories are hosted on GitHub, and many open-source projects use it for Git hosting, issue tracking, code review, and other things. So while it's not a direct part of the Git open source project, there's a good chance that you'll want or need to interact with GitHub at some point while using Git professionally.

This chapter is about using GitHub effectively. We'll cover signing up for and managing an account, creating and using Git repositories, common workflows to contribute to projects and to accept contributions to yours, GitHub's programmatic interface and lots of little tips to make your life easier in general.

The first thing you need to do is set up a free user account. Simply visit <https://github.com>, choose a user name that isn't already taken, provide an email address and a password, and click the big green "Sign up for GitHub" button.

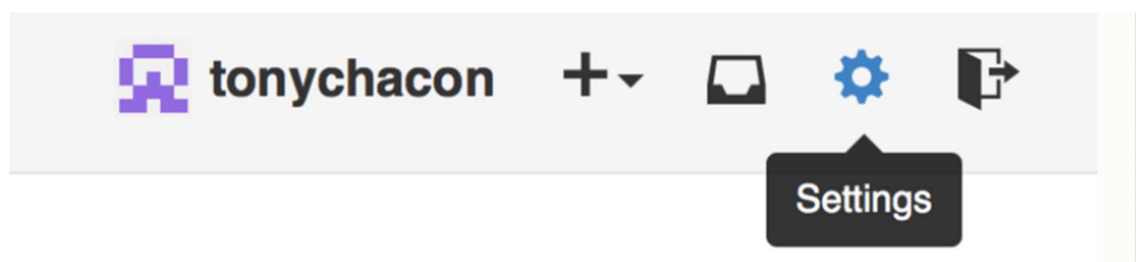
A screenshot of the GitHub sign-up form. It features three white input fields on a dark background. The first field is labeled 'Pick a username', the second 'Your email', and the third 'Create a password'. Below the third field, there is a text requirement: 'Use at least one lowercase letter, one numeral, and seven characters.' At the bottom of the form is a large green button with the text 'Sign up for GitHub' in white.

The next thing you'll see is the pricing page for upgraded plans, but it's safe to ignore this for now. GitHub will send you an email to verify the address you provided. Go ahead and do this; it's pretty important.

Clicking the Octocat logo at the top-left of the screen will take you to your dashboard page. You're now ready to use GitHub.

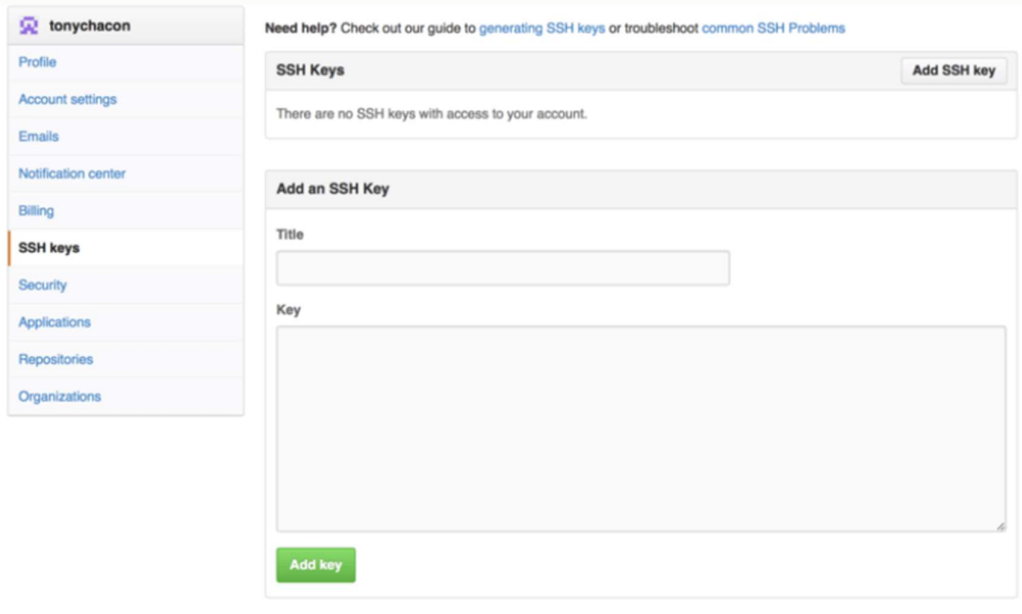
As of right now, you're fully able to connect with Git repositories using the `https://` protocol, authenticating with the username and password you just set up. However, to simply clone public projects, you don't even need to sign up - the account we just created comes into play when we fork projects and push to our forks a bit later.

If you'd like to use SSH remotes, you'll need to configure a public key. If you don't already have one, see [Generating Your SSH Public Key](#). Open up your account settings using the link at the top-right of the window:



The “Account settings” link

Then select the “SSH keys” section along the left-hand side.

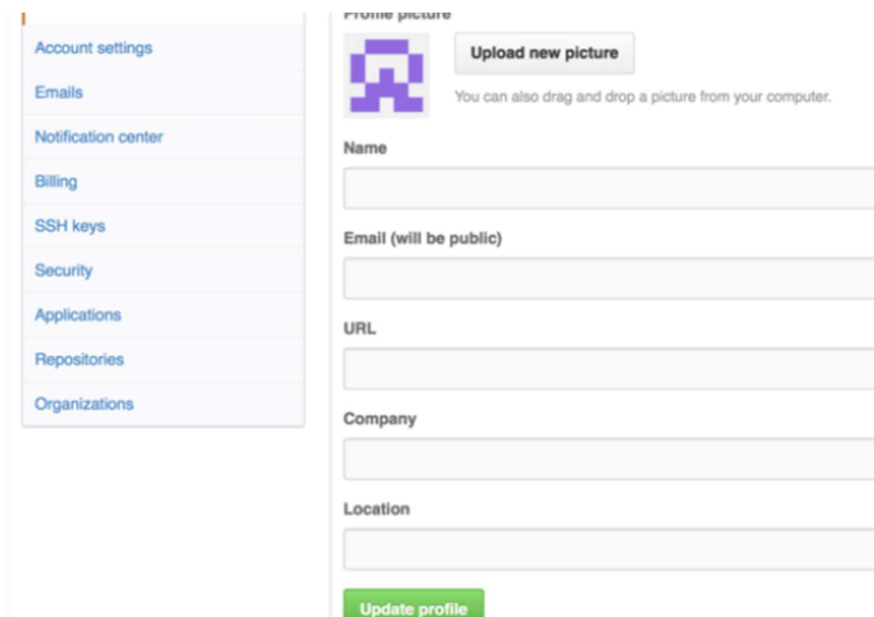


The screenshot shows the GitHub account settings page for user 'tonychacon'. On the left sidebar, the 'SSH keys' link is highlighted. The main content area is titled 'SSH Keys' and includes a link to a help guide. Below this, a message states 'There are no SSH keys with access to your account.' and an 'Add SSH key' button. A section titled 'Add an SSH Key' contains a 'Title' text field and a large 'Key' text area for pasting the public key. An 'Add key' button is at the bottom of this section.

The “SSH keys” link.

From there, click the “Add an SSH key” button, give your key a name, paste the contents of your `~/.ssh/id_rsa.pub` (or whatever you named it) public-key file into the text area, and click “Add key”.

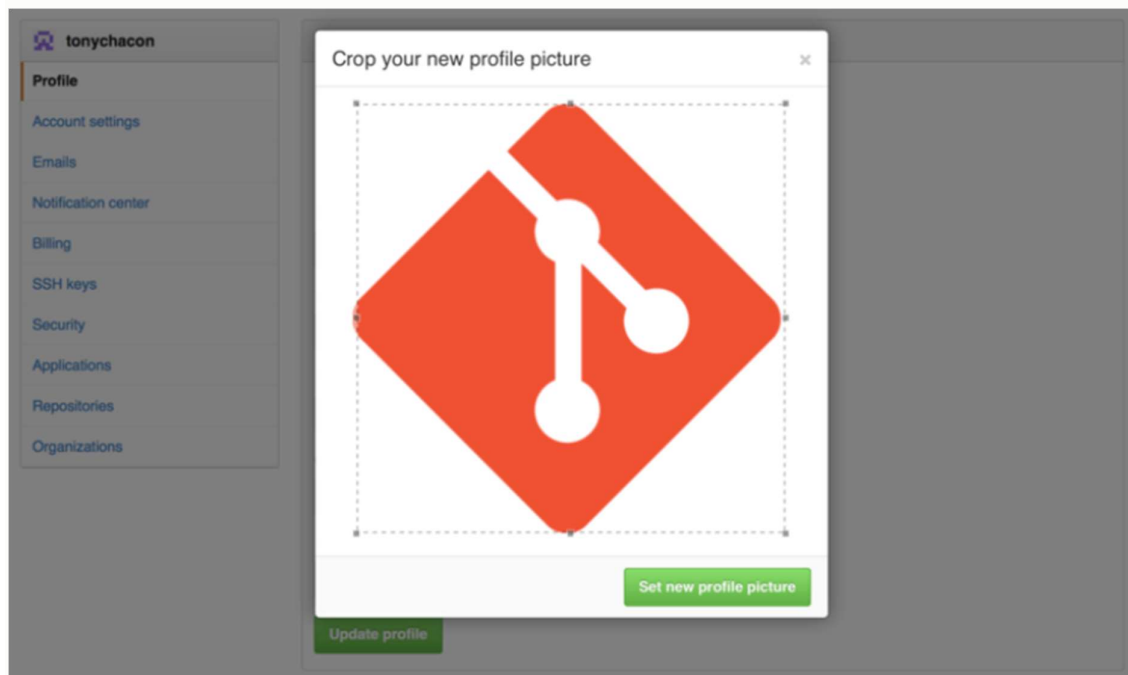
Next, if you wish, you can replace the avatar that is generated for you with an image of your choosing. First go to the “Profile” tab (above the SSH Keys tab) and click “Upload new picture”.



The screenshot shows the GitHub profile page for user 'tonychacon'. The left sidebar shows the 'Profile' link is selected. The main content area is titled 'Profile picture' and features a placeholder image and an 'Upload new picture' button. Below this, there are text input fields for 'Name', 'Email (will be public)', 'URL', 'Company', and 'Location'. An 'Update profile' button is at the bottom.

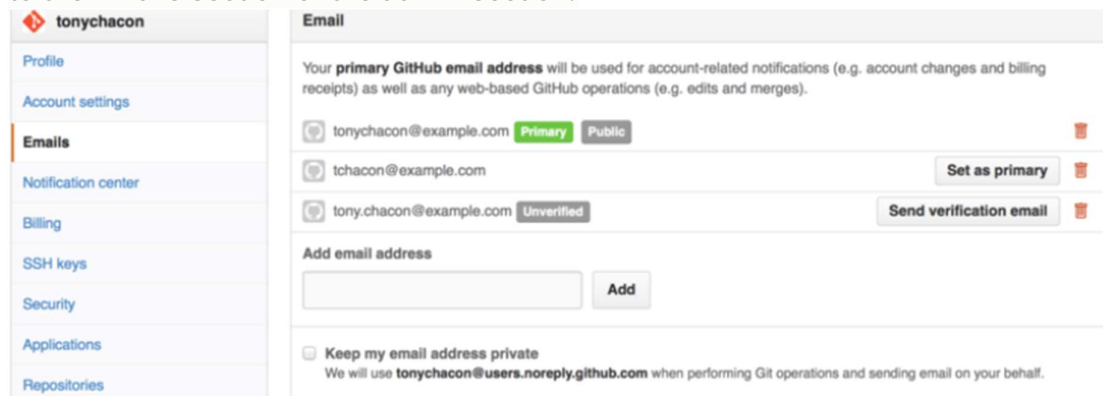
The “Profile” link

We’ll choose a copy of the Git logo that is on our hard drive and then we get a chance to crop it.



Now anywhere you interact on the site, people will see your avatar next to your username.

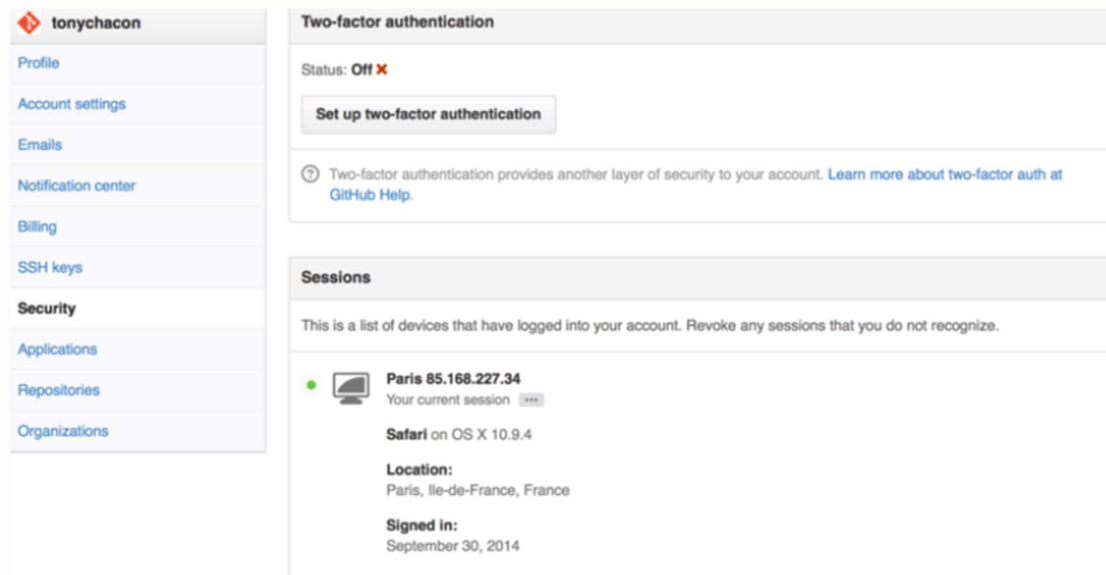
The way that GitHub maps your Git commits to your user is by email address. If you use multiple email addresses in your commits and you want GitHub to link them up properly, you need to add all the email addresses you have used to the Emails section of the admin section.



In [Add email addresses](#) we can see some of the different states that are possible. The top address is verified and set as the primary address, meaning that is where you’ll get any notifications and receipts. The second address is verified and so can be set as the primary if you wish to switch them. The final address is unverified, meaning that you can’t make it your primary address. If GitHub sees any of these in commit messages in any repository on the site, it will be linked to your user now.

Two Factor Authentication

Finally, for extra security, you should definitely set up Two-factor Authentication or “2FA”. Two-factor Authentication is an authentication mechanism that is becoming more and more popular recently to mitigate the risk of your account being compromised if your password is stolen somehow. Turning it on will make GitHub ask you for two different methods of authentication, so that if one of them is compromised, an attacker will not be able to access your account.



If you click on the “Set up two-factor authentication” button, it will take you to a configuration page where you can choose to use a phone app to generate your secondary code (a “time based one-time password”), or you can have GitHub send you a code via SMS each time you need to log in.

After you choose which method you prefer and follow the instructions for setting up 2FA, your account will then be a little more secure and you will have to provide a code in addition to your password whenever you log into GitHub.

3) Generate Log:

Used the following commands in local folder:

```

337 git init
338 git status
339 git add .
340 git status
341 git status
342 git commit -m "deleted an extra file"
343 git log
344 git status
345 git commit -m "deleted 2 extra files and modified 1 existing file"
346 git log
347 git add .
348 git log
349 git status
350 git commit -m "commit 2 and 3 deleted and modified"
351 git log

```

```

$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:   PYTHAGORIAN TRIPLETUSING FUNCTION.exe
        deleted:   linear search.exe
        modified:  sum of n natural nosusing function.cpp
        deleted:   sum of n natural nosusing function.exe

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:  hollow rectangleof stars.cpp

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git commit -m "commit 2 and 3 deleted and modified"
[master 3a01cf7] commit 2 and 3 deleted and modified
4 files changed, 1 insertion(+)
delete mode 100644 PYTHAGORIAN TRIPLETUSING FUNCTION.exe
delete mode 100644 linear search.exe
delete mode 100644 sum of n natural nosusing function.exe

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git log
commit 3a01cf773a362b42cfdb1f820d83d7959af0b3bf (HEAD -> master)
Author: DEVANSH-DOGRA <devansh0422.be21@chitkara.edu.in>
Date:   Sun Apr 10 11:10:41 2022 +0530

    commit 2 and 3 deleted and modified

commit 6a78e9d8f251e8c0dcf1eb5f48246971643ec0f7
Author: DEVANSH-DOGRA <devansh0422.be21@chitkara.edu.in>
Date:   Sun Apr 10 11:06:54 2022 +0530

    deleted an extra file

```

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

git status: The git status command is used to display the state of the repository and staging area. It allows us to see the tracked, untracked files and changes. This command will not show any commit records or information. Mostly, it is used to display the state between Git Add and Git commit command.

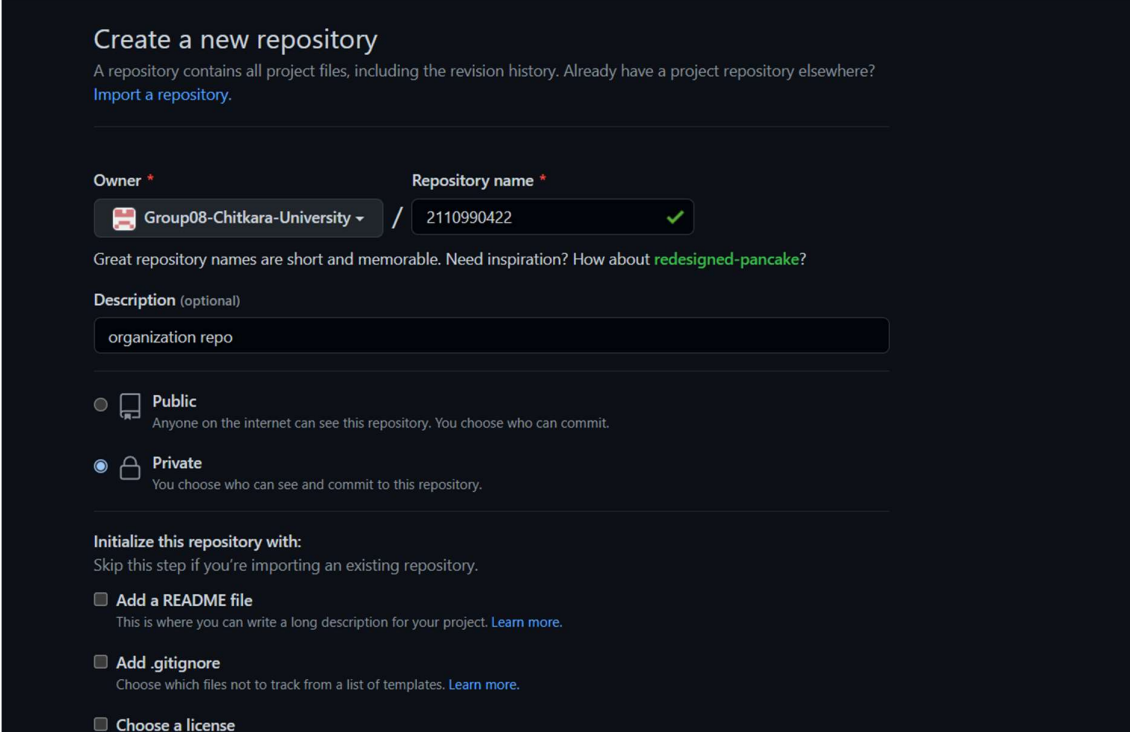
git add: The git add command adds a file to the Git staging area. This area contains a list of all the files you have recently changed. Your repository will be updated the next time you create a commit with your changes.

Therefore, running the git add command does not change any of your work in the Git repository. Changes are only made to your repository when you execute the git commit command.

git commit: The git commit command is one of the core primary functions of Git. Prior use of the git add command is required to select the changes that will be staged for the next commit. Then git commit is used to create a snapshot of the staged changes along a timeline of a Git projects history.

git log: The git log command shows a list of all the commits made to a repository. You can see the hash of each Git commit , the message associated with each commit, and more metadata. This command is useful for displaying the history of a repository.

4) Creating and visualising branches:




The screenshot shows the GitHub 'Create a new repository' interface. At the top, it says 'Create a new repository' and provides a brief explanation of what a repository is, along with a link to 'Import a repository'. Below this, there are two main input fields: 'Owner' and 'Repository name'. The 'Owner' field is set to 'Group08-Chitkara-University' and the 'Repository name' field is set to '2110990422'. A green checkmark is visible next to the repository name. Below these fields, there is a suggestion for repository names: 'Great repository names are short and memorable. Need inspiration? How about redesigned-pancake?'. The 'Description' field is optional and contains the text 'organization repo'. Under the 'Visibility' section, there are two radio buttons: 'Public' (which is selected) and 'Private'. Below this, there is a section titled 'Initialize this repository with:' which includes three checkboxes: 'Add a README file', 'Add .gitignore', and 'Choose a license'. Each checkbox has a brief description and a link to 'Learn more'.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Owner * Repository name *


 Group08-Chitkara-University / 2110990422 ✓

Great repository names are short and memorable. Need inspiration? How about [redesigned-pancake?](#)

Description (optional)

organization repo

☐  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☒  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**

```
devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git remote add origin https://github.com/Group08-Chitkara-University/2110990422.git

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git status
On branch master
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:   hollow_rectangleof_stars.cpp

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

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git push -u origin master
Enumerating objects: 29, done.
Counting objects: 100% (29/29), done.
Delta compression using up to 8 threads
Compressing objects: 100% (29/29), done.
Writing objects: 100% (29/29), 67.37 KiB | 1.73 MiB/s, done.
Total 29 (delta 10), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (10/10), done.
To https://github.com/Group08-Chitkara-University/2110990422.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

```

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git branch master1

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git checkout master1
Switched to branch 'master1'
M       hollow rectangleof stars.cpp

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
$ git branch
  master
* master1

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
$ git status
On branch master1
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:   hollow rectangleof stars.cpp

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

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

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
$ git commit -m "changes to new branch"
On branch master1
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:   hollow rectangleof stars.cpp

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

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

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
$ git add .

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
$ git commit -m "changes to new branch"
[master1 804d545] changes to new branch
 2 files changed, 14 insertions(+)
 create mode 100644 file names.txt

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
Everything up-to-date
master
branch 'master' set up to track 'origin/master'.

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
$ git push -u origin master1

```

```
devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
$ git branch master2

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
$ git branch
  master
* master1
  master2

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
$ git checkout master2
Switched to branch 'master2'

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master2)
$ git branch
  master
  master1
* master2

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master2)
$ git add .

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master2)
$ git commit -m "commit for master 3 pascal triangle"
[master2 b38bb64] commit for master 3 pascal triangle
1 file changed, 45 insertions(+)
create mode 100644 codeee.txt
```



```

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master2)
$ git push -u origin master2
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 630 bytes | 630.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'master2' on GitHub by visiting:
remote:   https://github.com/Group08-Chitkara-University/2110990422/pull/new/master2
remote:
To https://github.com/Group08-Chitkara-University/2110990422.git
 * [new branch]      master2 -> master2
branch 'master2' set up to track 'origin/master2'.

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master2)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git branch
* master
  master1
  master2

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

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:   hollow rectangleof stars.cpp

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

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git add .

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git push -u origin master
Everything up-to-date
branch 'master' set up to track 'origin/master'.

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git checkout master1
error: Your local changes to the following files would be overwritten by checkout:
        hollow rectangleof stars.cpp
Please commit your changes or stash them before you switch branches.
Aborting

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git branch

```



```

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git branch
* master
  master1
  master2

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git commit -m "master changes"
[master 4b09025] master changes
1 file changed, 18 insertions(+)

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git push -u origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 320 bytes | 160.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/Group08-Chitkara-University/2110990422.git
   3a01cf7..4b09025  master -> master
branch 'master' set up to track 'origin/master'.

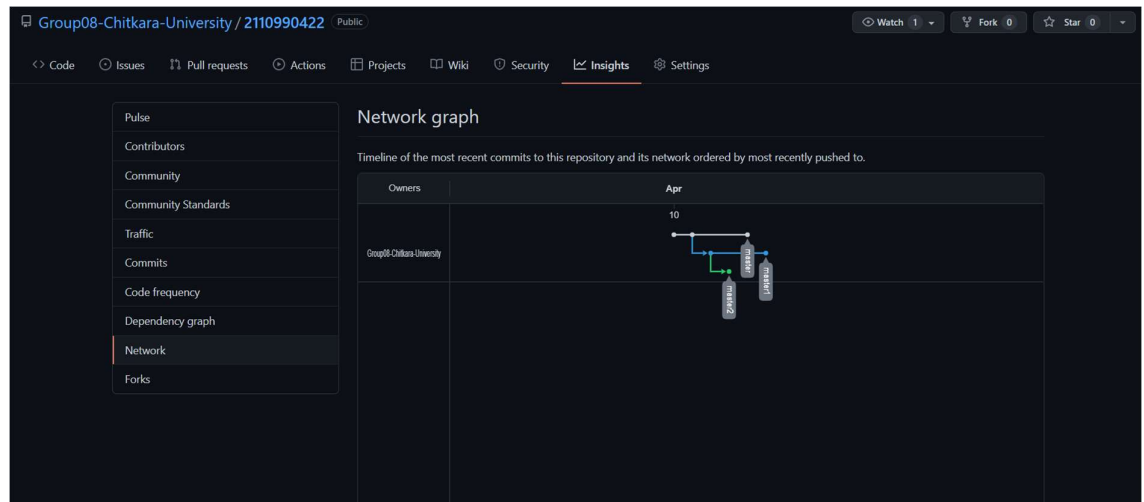
devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master)
$ git checkout master1
Switched to branch 'master1'
Your branch is up to date with 'origin/master1'.

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
$ git add .

devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
$ git commit -m "master1 changes-deleted linear search oo"
[master1 167ed2c] master1 changes-deleted linear search oo
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 linear searchusing functions.exe

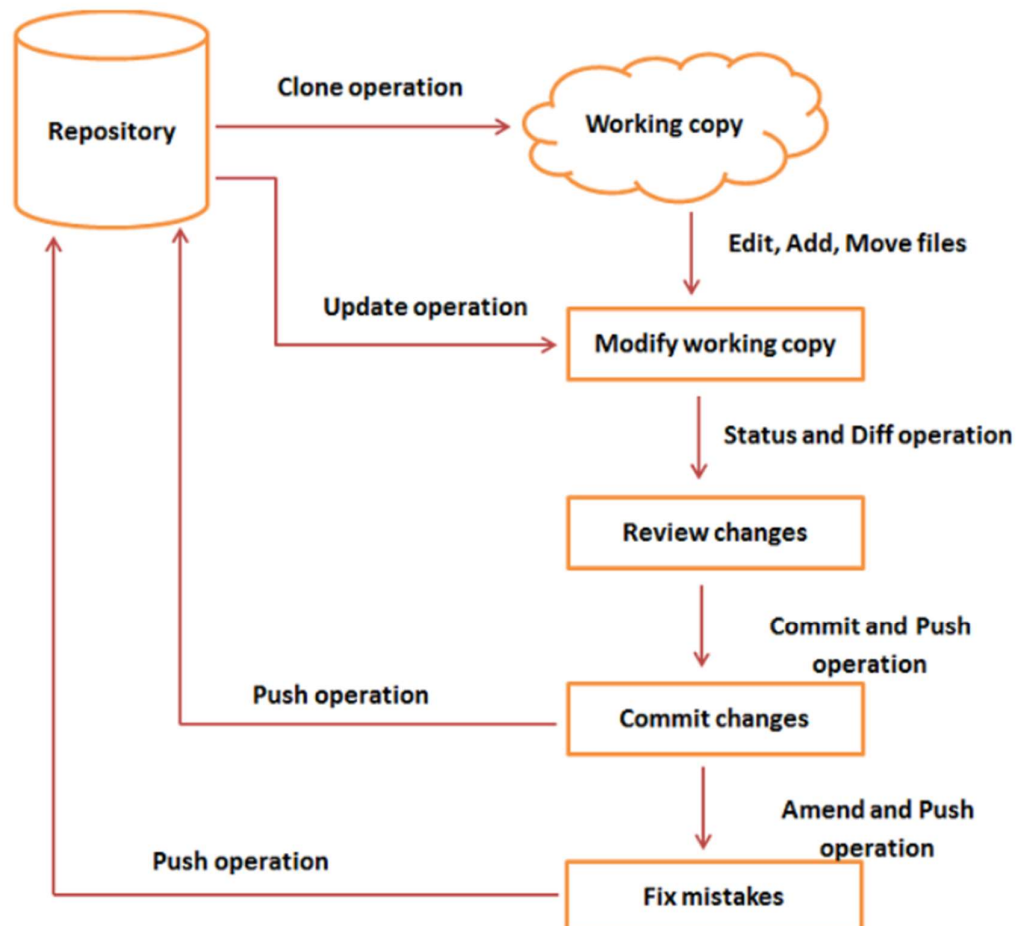
devan@DESKTOP-E77QH23 MINGW64 ~/OneDrive/Desktop/patterns codes (master1)
$ git push -u origin master1
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 266 bytes | 266.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Group08-Chitkara-University/2110990422.git
   804d545..167ed2c  master1 -> master1
branch 'master1' set up to track 'origin/master1'.

```

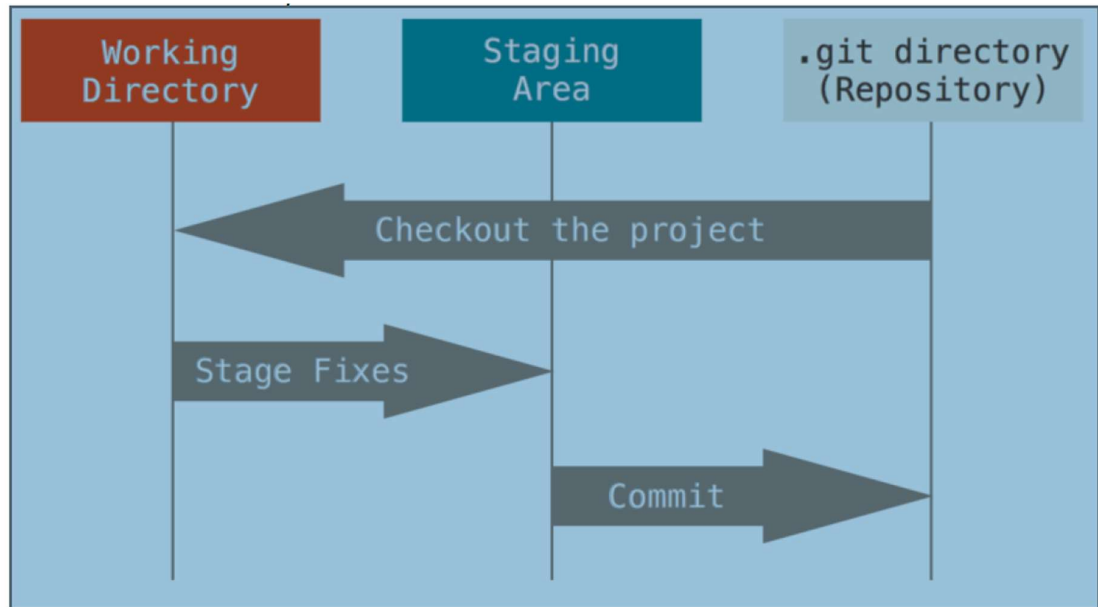


5)Git life cycle description:

Shown below is the pictorial representation of the work-flow:



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-



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

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"

NAME: DEVANSH DOGRA
ROLL NO.- 2110990422
G08