Subject Name: **Source Code Management**
Subject Code: **22CS003**

Session: **2022-23**

Department: **DCSE**

Submitted By:
Pardume Sharma
2210990638-First Year
G26-B

Submitted To:
Dr. Deepak Thakur.

List of programs

Aim: Setting up a Git client

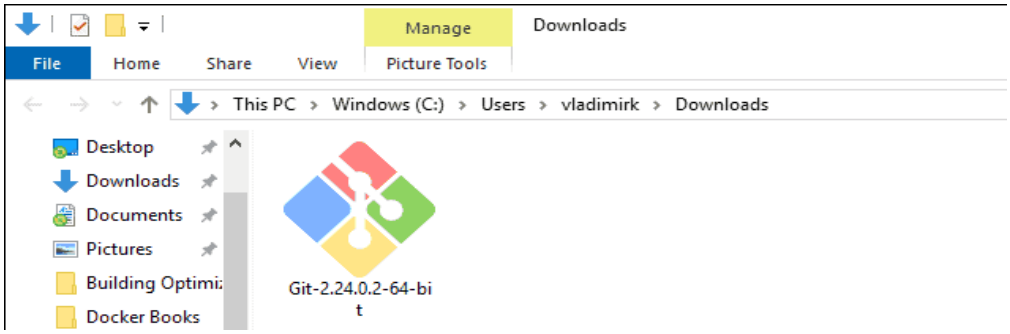
- Git download for windows
- Launching git in windows
- Version Control System and its types.
- Need of VCS
- Graphical Representation of VCS.
- Git
- Motivation of Git.

Git Download for Windows

1.Browse to the official website: https://git-scm.com/downloads , select windows

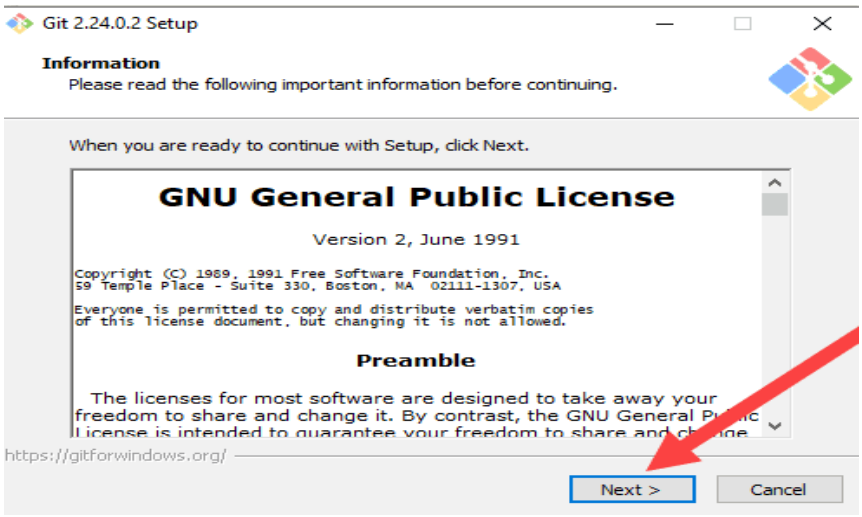
Extract and Launch Git Installer

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

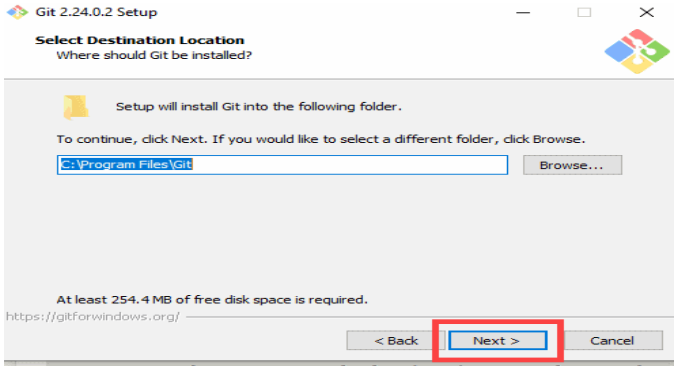


3. Allow the app to make changes to your device by clicking Yes on the User Account Control dialog that opens.

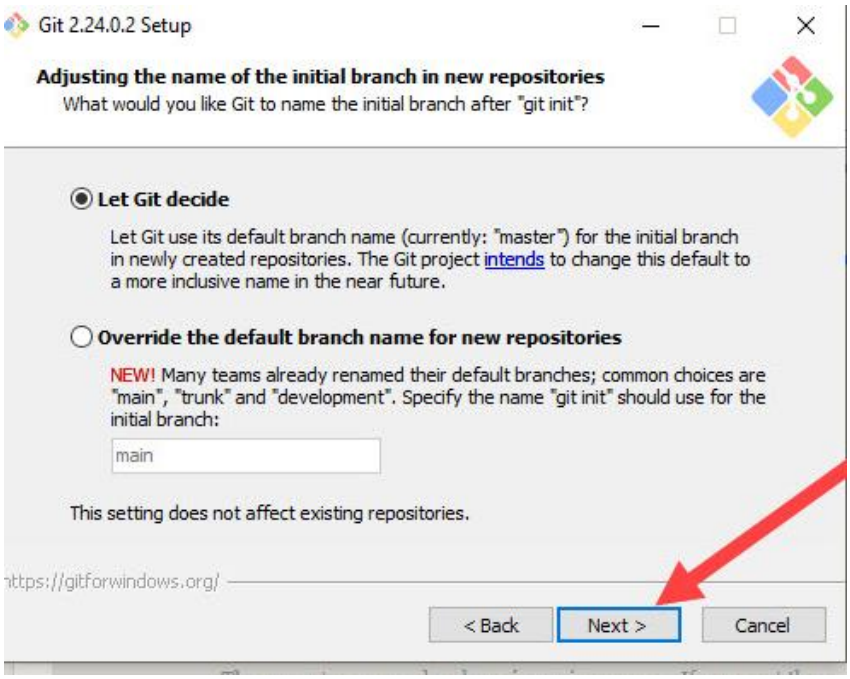
4. Review the GNU General Public License, and when you're ready to install, click **Next.**



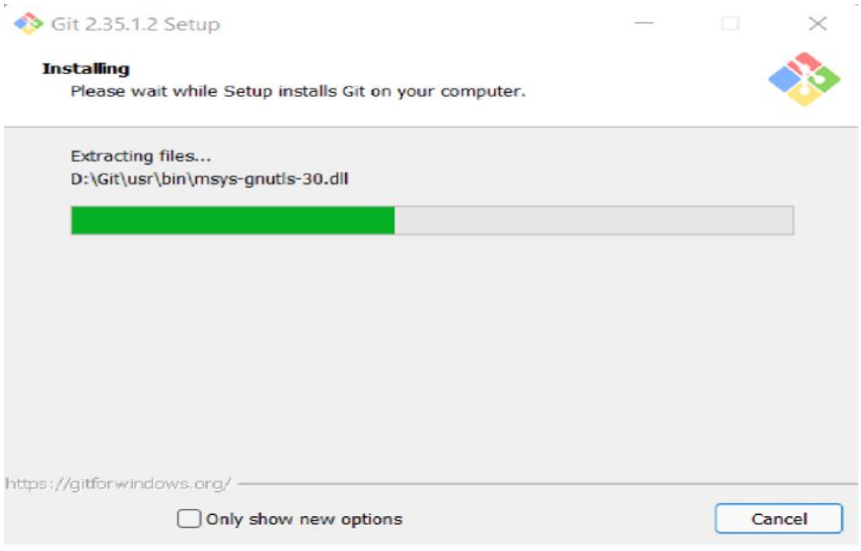
5. The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click **Next.**



6. 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**.

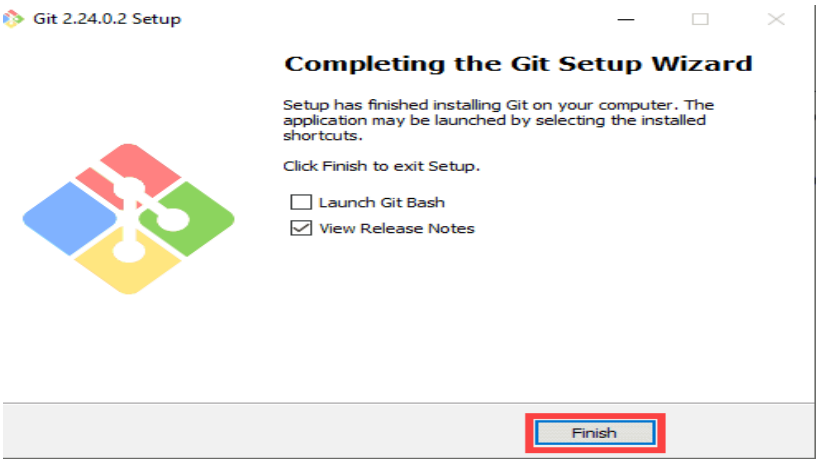


7. Depending on the version of Git you're installing, it may offer to install experimental features. At the time this article was written, the options to include support for pseudo controls and a built-in file system monitor were offered. Unless you are feeling adventurous, leave them unchecked and click **Install**.



Complete Git Installation Process

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

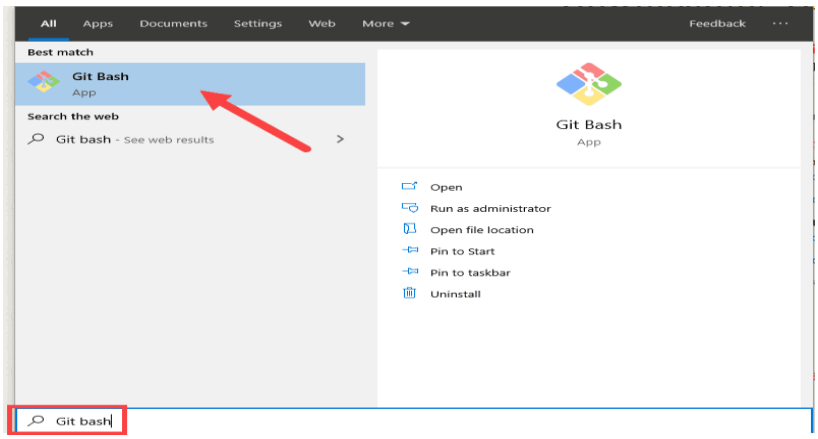


How to Launch Git in Windows

Git has two modes of use – a **bash scripting shell** (or command line) and a **graphical user interface** (GUI).

Launch Git Bash Shell

To launch **Git Bash** open the **Windows Start** menu, type *git bash* and press **Enter** (or click the application icon).



Version Control System

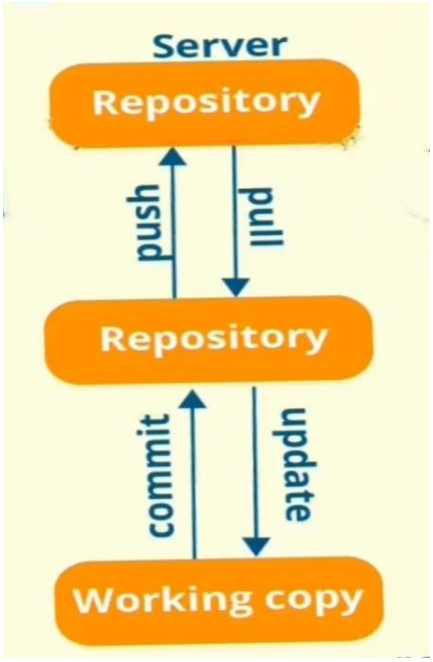
Git is a **Version Control System** or **VCS**. **VCS** is basically software designed to record changes within one or more files over time. It allows us to undo or to cancel all made or pending changes within one or more files. If we're working on a project with many files, **VCS** enables us to control the whole project.

Types of VCS

Git is not the only VCS in use. Moreover, Git is just a kind of VCS.
The types of VCS are:

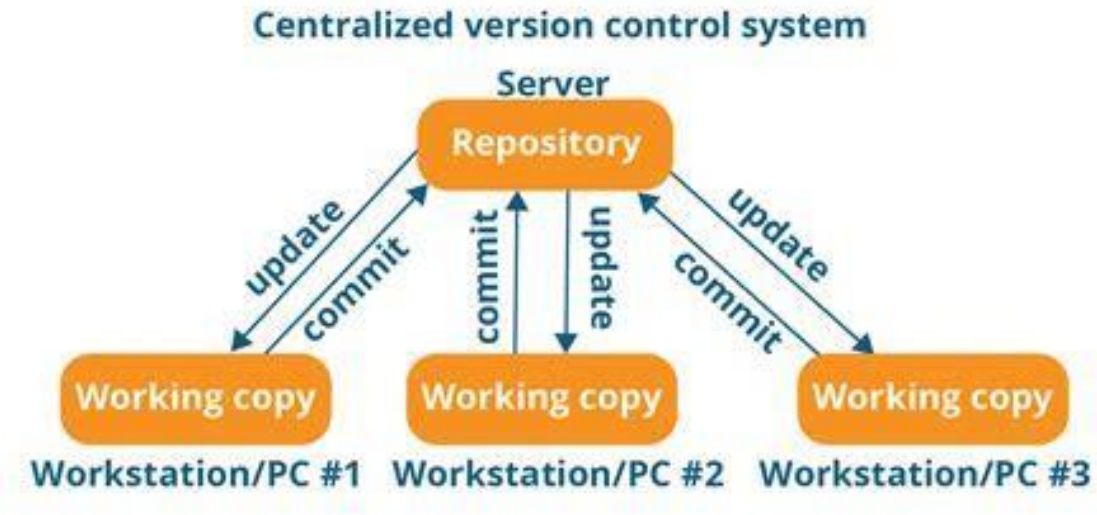
- Local Version Control System
- Centralized Version Control System
- Distributed Version Control System

Local Version Control System



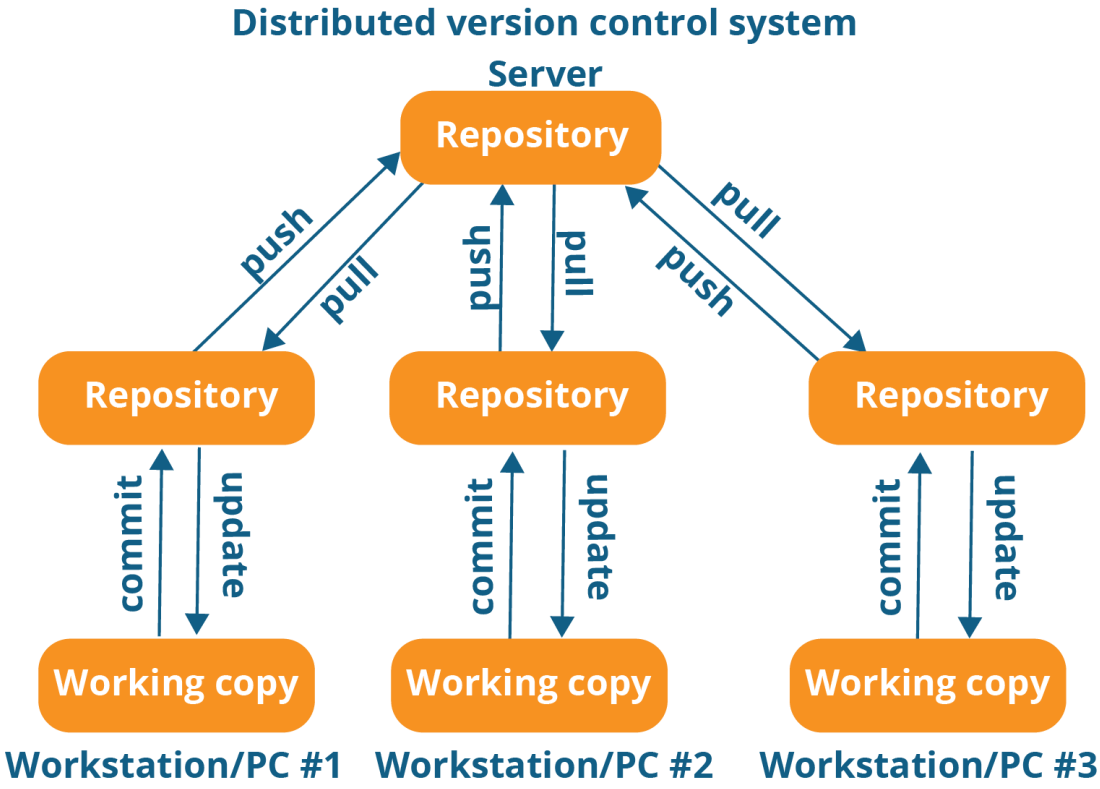
https://www.git-scm.com

Centralized Version Control System



https://www.quora.com

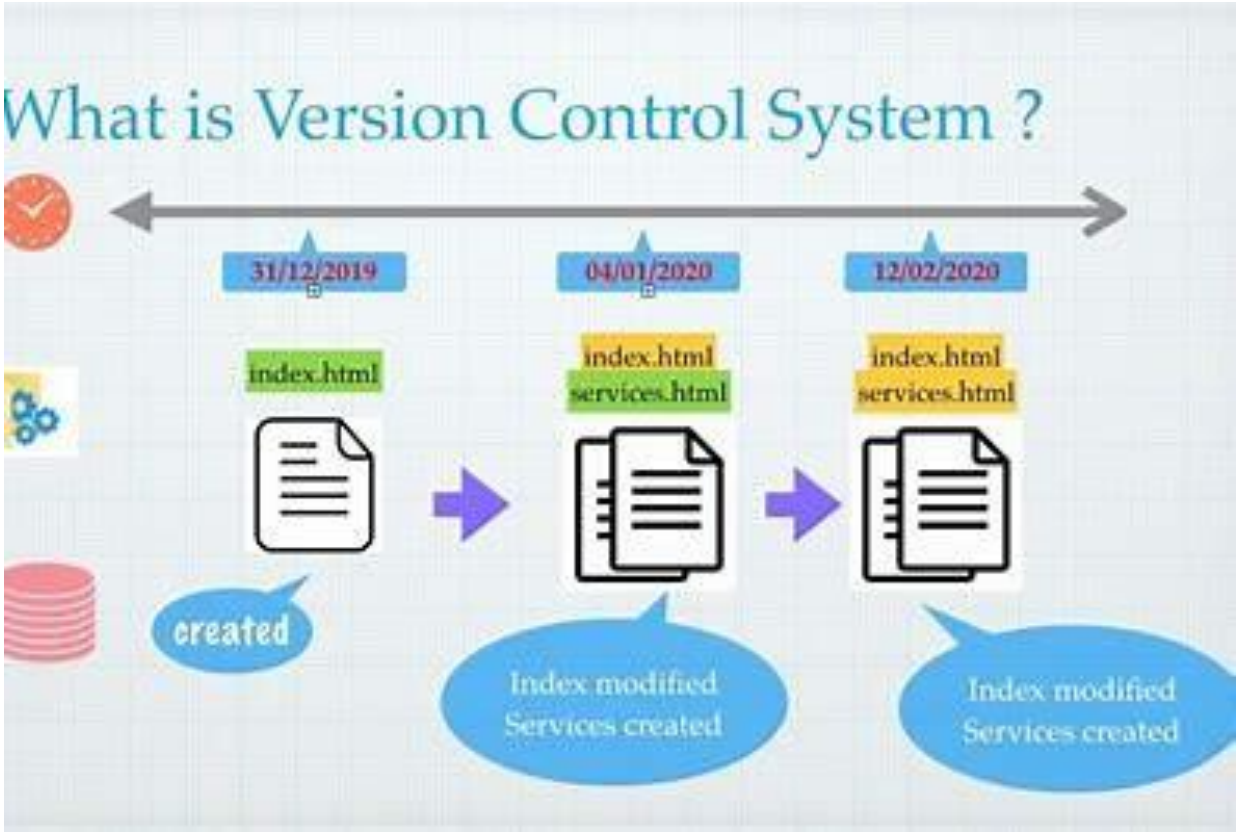
Distributed Version control System



Need of VCS

- Enhances the project development speed.
- Provide a very efficient collaboration within the team
- Reduces the possibility of error and conflicts
- Even a very small change is traceable
- Devs can contribute to the project from anywhere
- For different contributors, a different copy can be maintained and can be verified before merging to the main project
- Recovery of the previous state is possible
- A lot of information about who, why, and when can be accessed
- Saves a lot of time for your coffee.

Graphical Representation of VCS



GIT

Git is the most commonly used version control system. Git tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions should you ever need to. Git also makes collaboration easier, allowing changes by multiple people to all be merged into one source.

Motivation of GIT

- **Performance:** Git provides the best performance when it comes to version control systems. Committing, branching, merging all are optimized for a better performance than other systems.
- **Security**: Git handles your security with cryptographic method SHA-1. The algorithm manages your versions, files, and directory securely so that your work is not corrupted.
- **Branching Model:** Git has a different branching model than the other VCS. Git branching model lets you have multiple local branches which are independent of each other. Having this also enables you to have friction-less context switching , role-based code (a branch that always goes to production, another to testing etc) and disposable experimentation
- **Staging Area:** Git has an intermediate stage called "index" or "staging area" where commits can be formatted and modified before completing the commit.
- **Distributed:** Git is distributed in nature. Distributed means that the repository or the complete code base is mirrored onto the developer's system so that he can work on it only.
- **Open Source:** This is a very important feature of any software present today. Being open source invites the developers from all over the world to contribute to the software and make it more and more powerful through features and additional plugins.

Aim: Setting up a GitHub Account

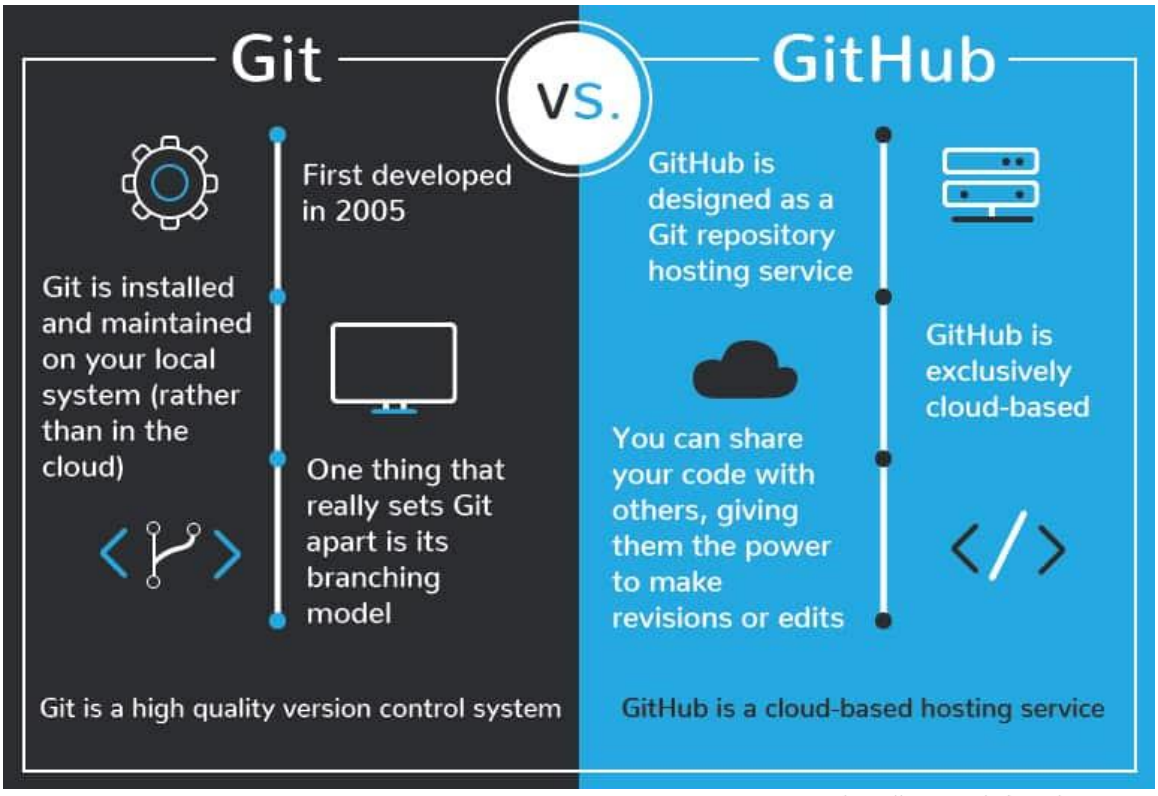
- GitHub
- Difference between Git and GitHub
- GitHub Account Setup.

GitHub

It's an online database that allows you to keep track of and share your Git version control projects outside of your local computer/server. Unlike Git, GitHub is exclusively cloud-based.

Also unlike Git, GitHub is a for-profit service (although basic repository-hosting features are available at no cost to those who are willing to create a user profile, making GitHub a popular choice for open-source projects).

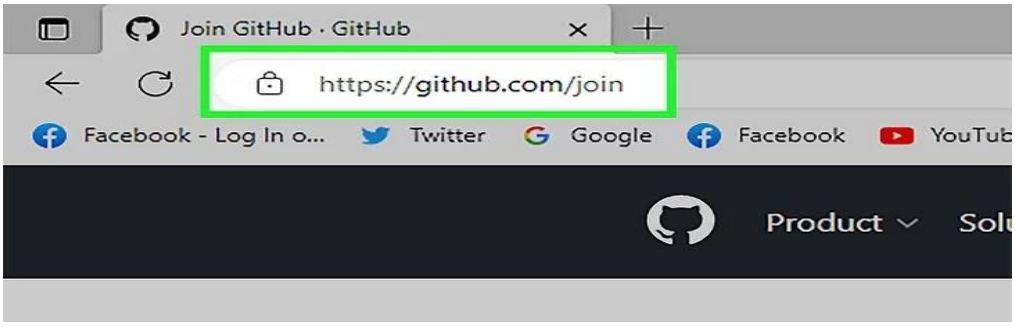
Difference between Git and GitHub



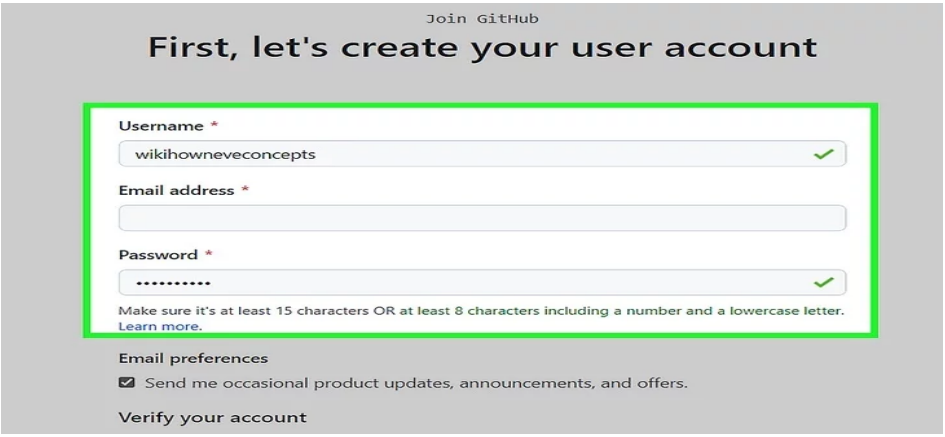
https://www.geeksforgeeks.com

GitHub account setup

1 Go to **https://github.com/join** in a web browser. You can use any web browser on your computer, phone, or tablet to join. Before you can **create branches** or **make any pull requests,** you'll need an account.

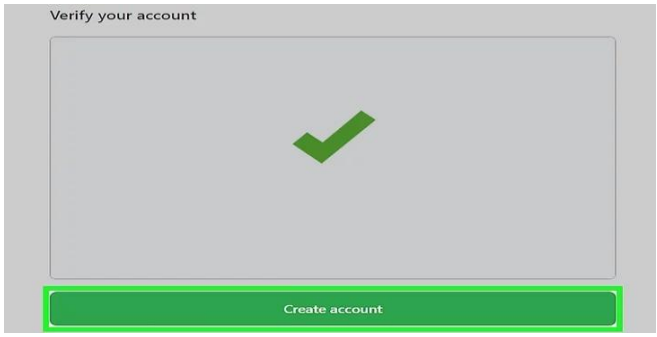


2 Enter your personal details. In addition to creating a username and entering an email address, you'll also have to create a password. Your password must be at least 15 characters in length *or* at least 8 characters with at least one number and lowercase letter.[1]

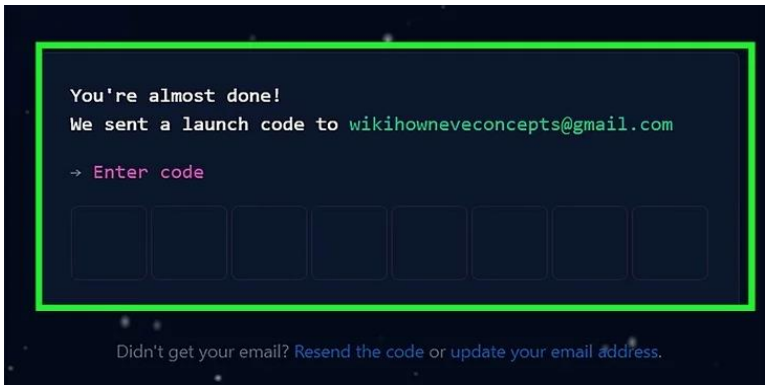


3 Click the green Create account button. It's below the form, at the bottom of the page. This will take you to an email verification page.

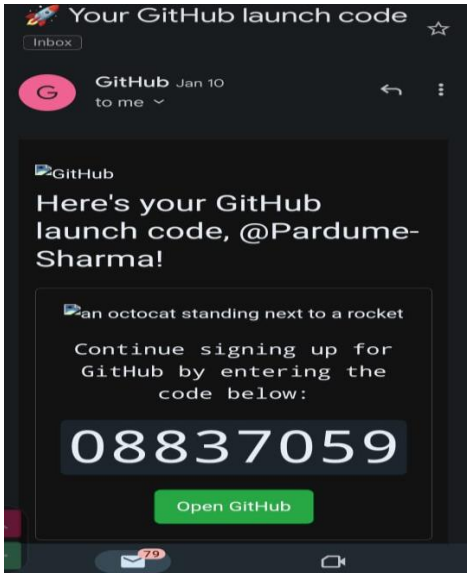
•Carefully_review_the_Terms_of_Serviceat https://help.github.com/en/articles/githubtermsofservice andthePrivacy Statementat https://help.github.com/en/articles/github-privacy-statement before you continue. Continuing past the next step confirms that you agree to both documents.



4 Verify your email by entering the code. After clicking Create account, you'll receive an email with a code. Enter this code on the verification page. Entering the code will automatically take you to the welcome page.

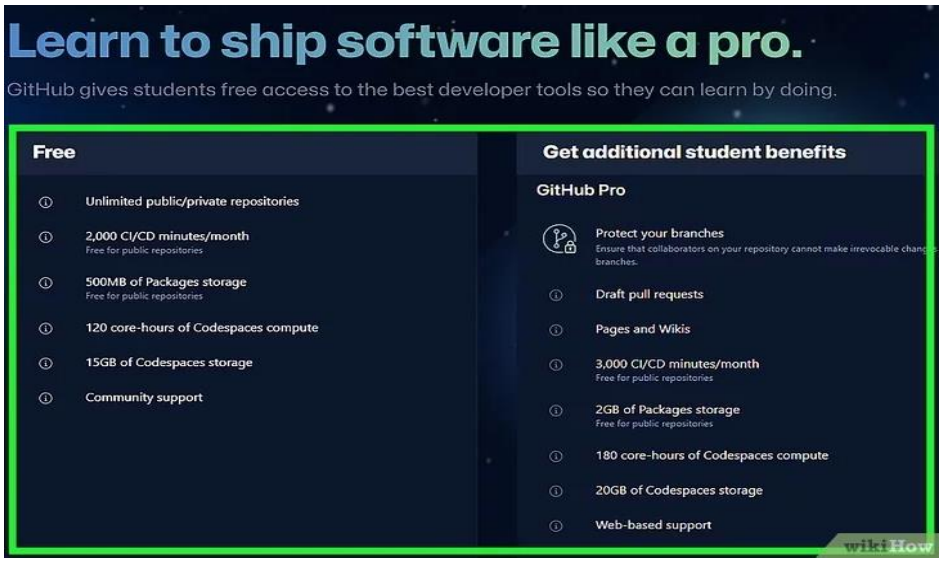


5 The Launch code received on mv email.



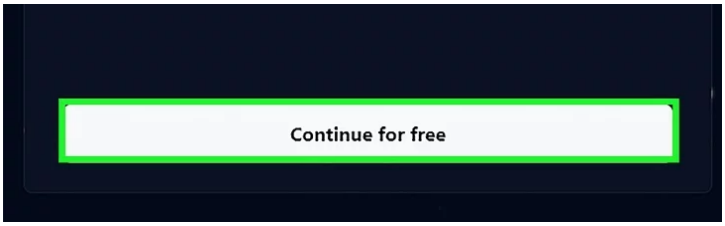
6 Note the types of plans offered by GitHub. There are a few different plans to choose from, varying in the amount of features provided.[2]

- **Free**: Unlimited public and private repositories, up to 3 collaborators, issues and bug tracking, and project management tools.
- **Pro:** Unlimited access to all repositories, unlimited collaborators, GitHub email support, and advanced insight tools.
- **Team**: All of the aforementioned features, plus team access controls and user management.



7 Select the free plan. On the plan selection page, scroll down to click the button for choosing a free plan. This will immediately take you to your GitHub dashboard. You're ready to download some directories and repositories, or download a file.

8. Click on Continue for free.



Aim: Generate logs

• Various commands:
pwd, git –version, git init, Significance of .git folder, SHA-1hash algorithm, ls -la, git config --global user.email, git config --global user.name, git status, git add ,git commit, git checkout, git log, git log –graph, git restore, cat,vi, touch, .gitignore.
• 3-stage architecture
• Create a new repository.

Working with git

pwd

pwd stands for present working directory which tells us about on which directory we are currently, it shows in form of path.

```
MINGW64:/c/Users/pardu/Desktop/demo

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo ((14edb1d...))
$ pwd
/c/Users/pardu/Desktop/demo
```

git --version

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo ((14edb1d...))
$ git --version
git version 2.39.0.windows.2
```

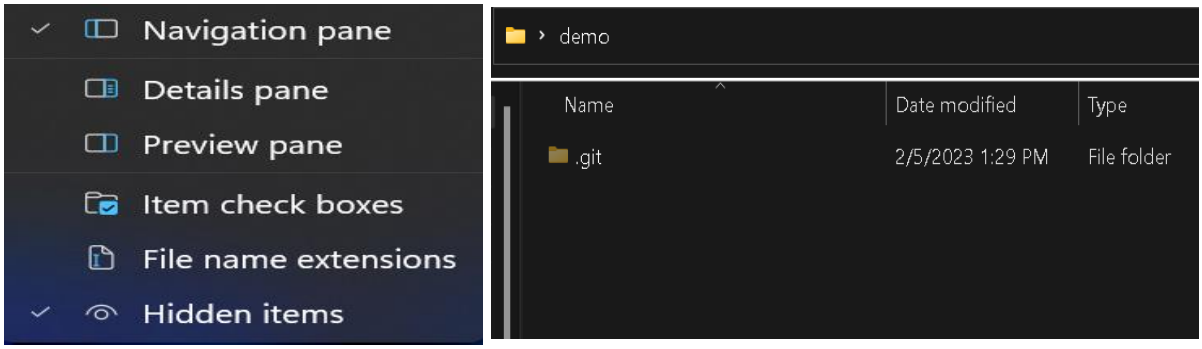
It tells us about the current version of git downloaded in our system.

git init

```
MINGW64:/c/Users/pardu/Desktop/demo

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo
$ git init
Initialized empty Git repository in C:/Users/pardu/Desktop/demo/.git/
```

• To convert an existing, unversioned project to a Git repository or initialize a new, empty repository. it will create a hidden directory .git to prevent damage.
• we can use git command within that directory.

• By default, .git folder is hidden so that no one can delete any files from .git folder . By checking the hidden items from view section we can see it.



Significance of .git folder

• git stores the metadata and object database for the project in this directory like:

• Remote information (to which remote server your project is connected).

• Branch information (on which branch is your current project state (HEAD) pointing to)

• All logs of all local commits you have ever made (including revert changes)

SHA-1 hash algorithm

SHA-1 or **Secure Hash Algorithm** is a cryptographic hash function which takes input and produces **a** 160-bit(20bytes) hash value. This hash value is known as **message digest**. This message digest is usually then rendered as a hexadecimal number which is 40 digits long.

ls-la

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo ((14edb1d...))
$ ls -la
total 24
drwxr-xr-x 1 pardu 197609     0 Jan 31 22:30 ./
drwxr-xr-x 1 pardu 197609     0 Jan 31 21:57 ../
drwxr-xr-x 1 pardu 197609     0 Jan 31 22:01 .git/
-rw-r--r-- 1 pardu 197609 12288 Jan 31 22:31 .swp
-rw-r--r-- 1 pardu 197609     0 Jan 31 22:01 file.txt
```

git configurations

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git config --global user.name Pardume_Sharma

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git config --global user.email pardume638.be22@chitkara.edu.in
```

we have to configure username and email id, so that git can use this information in the commit records.

git config -- <local/global> user.email <"email">

git config -- -- <local/global> user.name <"username">

if we use --local then the configuration will be applied for current repository only, but --global will apply the configurations for all the repository.

git config user.<name/email>

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git config user.name
Pardume_Sharma

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git config user.email
pardume638.be22@chitkara.edu.in
```

• To check the configuration.

git status

To check the status of files like which file is modified, untracked, tracked files etc..

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git status
On branch master
nothing to commit, working tree clean
```

-s flag is useful for shorter info.

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git add .

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git status -s
A  file5.txt
```

git add .

It helps to add the file in tracking list so that git tracks all the changes and commits done in the file.

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git add .

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file.txt
```

git commit -m <message>

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git commit -m "file created"
[master (root-commit) 14edb1d] file created
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file.txt
```

It is used to create a snapshot of the staged changes along a timeline of a Git projects history.

git checkout <checksum>

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git checkout ce4300b
Note: switching to 'ce4300b'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at ce4300b files created
A       file5.txt
```

It helps you to navigate between the versions/branches of the project/repository

git log

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git log
commit b2ab0096d2059f72ff3a2a72db7e6d2a2e02b9e2 (HEAD -> master)
Author: Pardume_Sharma <pardume638.be22@chitkara.edu.in>
Date:   Tue Jan 31 22:49:52 2023 +0530

    .

commit ce4300b7b2cc4ae8c5b25855c807b07ed70c36a1
Author: Pardume_Sharma <pardume638.be22@chitkara.edu.in>
Date:   Tue Jan 31 22:44:54 2023 +0530

    files created
```

- It shows a list of all the commits made to a repository.
- It also show hash, author info, and message of each commit.

--oneline flag is useful for one one liner logs

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git log --oneline
b2ab009 (HEAD -> master) .
ce4300b files created
```

git log --graph

It will show the log of all the commits in the form of graph.

```
MINGW64:/c/Users/pardu/Desktop/demo
*   commit 34cca6cf7b21a3ffc943ea1c4628c1e16323832a (HEAD -> master)
|\  Merge: 2ae0b4d 50a4bff
| | Author: Pardume_Sharma <pardume638.be22@chitkara.edu.in>
| | Date:   Sun Feb 5 12:54:51 2023 +0530
| |
| |     merge conflict resolved
| |
| * commit 50a4bff490789e49b5f579e100b9fc48ca853daf
| | Author: Pardume_Sharma <pardume638.be22@chitkara.edu.in>
| | Date:   Sun Feb 5 12:48:31 2023 +0530
| |
| |     changes in newbranch done
| |
* | commit 2ae0b4d0ebe29e6f882fd048b9585a3b9578b005
|/  Author: Pardume_Sharma <pardume638.be22@chitkara.edu.in>
|   Date:   Sun Feb 5 12:50:09 2023 +0530
|
|       master branch modified
|
* commit d3aab110d25b8ffdae40093e70575b0ee95944ab
  Author: Pardume_Sharma <pardume638.be22@chitkara.edu.in>
  Date:   Sun Feb 5 12:47:32 2023 +0530

      fiel created
```

Git restore

Restore command helps to undo the file to untracking list if already added to staging area.

git rm -cached <filename>

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git add .

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file.txt


pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git rm --cached file.txt
rm 'file.txt'

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git status
On branch master

No commits yet

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

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

- To remove the file from staging area.

cat <filename>

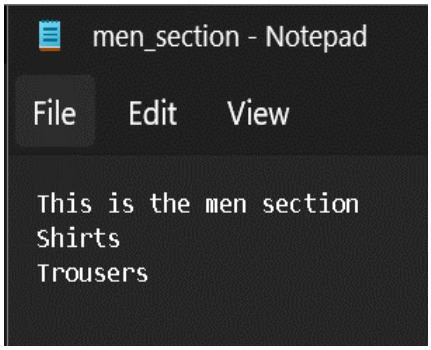
```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ cat file.txt
This is the home page of website
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$
```

It will show content of given filename. To check the content of multiple files we can run command as cat <file1> <file2>

cat > <filename>

It will create a new file and whatever we write in git bash after running this command, it will be added to the file created.

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ cat > men_section
This is the men section
Shirts
Trousers
Jackets
```



Vi <filename>

It will create a new file if it already not exist, otherwise opens existing file.

Touch <file name>

```
MINGW64:/c/Users/pardu/Desktop/demo

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ touch men_section.txt women_section.txt kids_section.txt

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git add .

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file5.txt
        new file:   kids_section.txt
        new file:   men_section.txt
        new file:   women_section.txt
```

- Touch command helps to create new files.

.gitignore

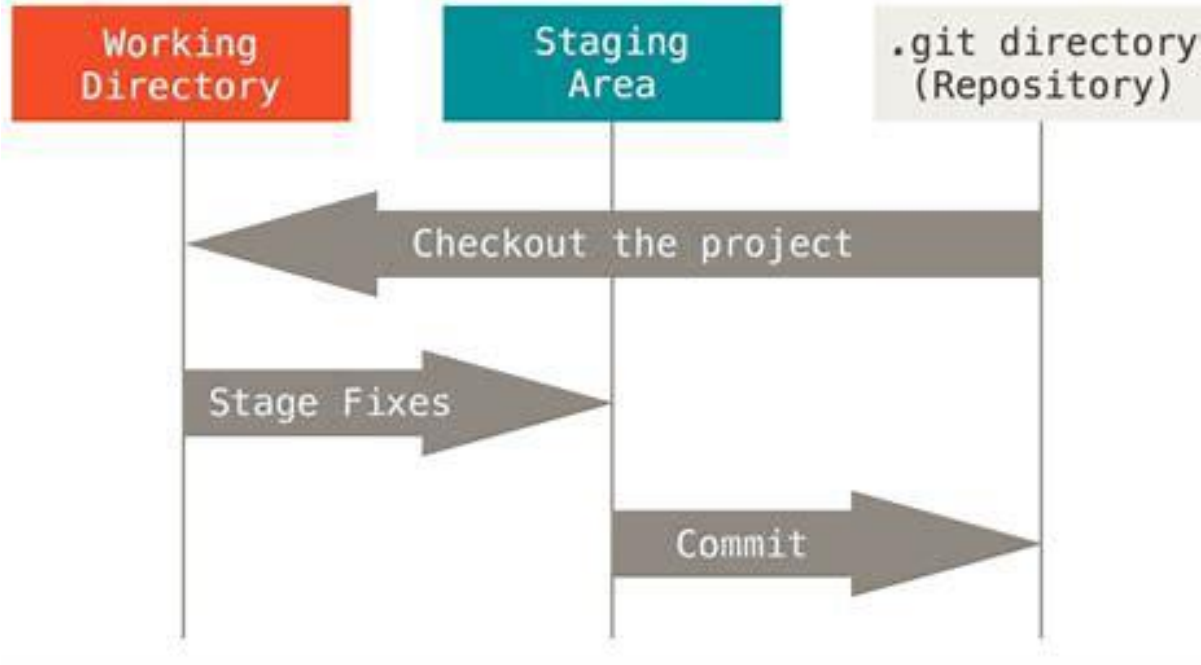
- It is a text file in the root folder of project that tells
- Git which files or folders (or patterns) to ignore in a project.

for eg:

- *.txt (ignore text file)
- abc.txt (ignore abc.txt only)
- logs/ (ignore logs directory)
- .* (ignore hidden file)

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ touch .gitignore

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

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

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git add.
git: 'add.' is not a git command. See 'git --help'.

The most similar command is
        add

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git add .

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
```

Git-3 Stage Architecture



Working directory

• The place where we can create new files or modify existing files.
• This is created when a Git project is initialized onto your local machine and allows you to edit the source code copied.

Staging Area

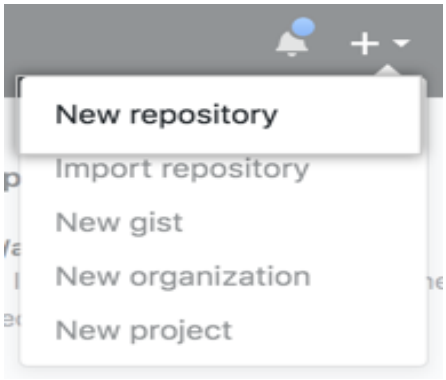
• It is the place where we send those files that we want to commit.
• This staging area is helpful to cross-check our changes before commit.

.git directory

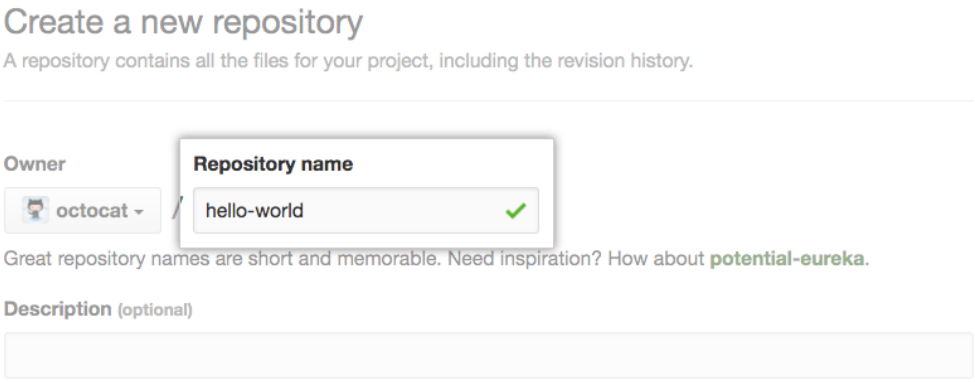
• A Git repository is a virtual storage of your project.
• It allows you to save versions of your code, which you can access when needed.
• It tracks and saves the history of all changes made to the files

Creating a new repository

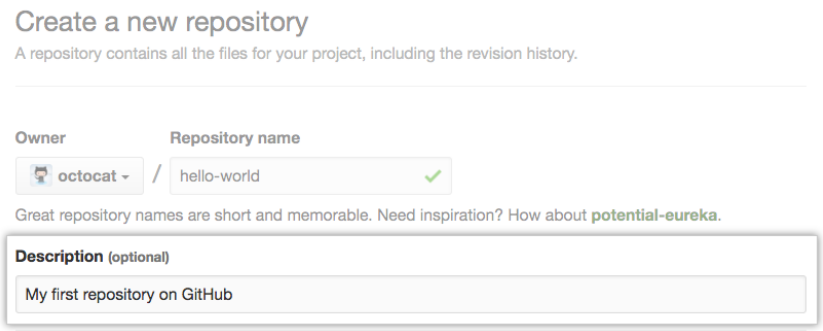
1.In the upper-right corner of any page, use the drop-down menu, and select New repository.



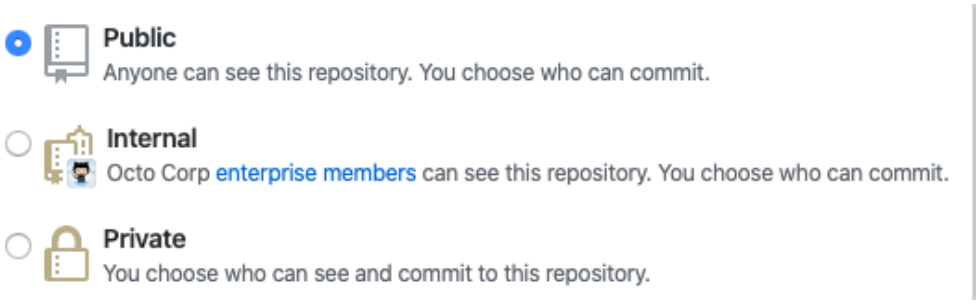
2.Type a short, memorable name for your repository. For example, "hello-world".



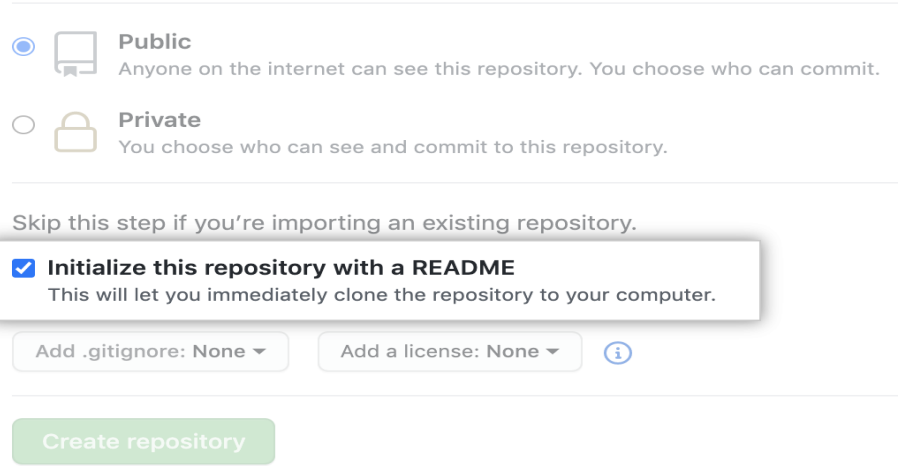
3.Optionally, add a description of your repository. For example, "My first repository on GitHub."



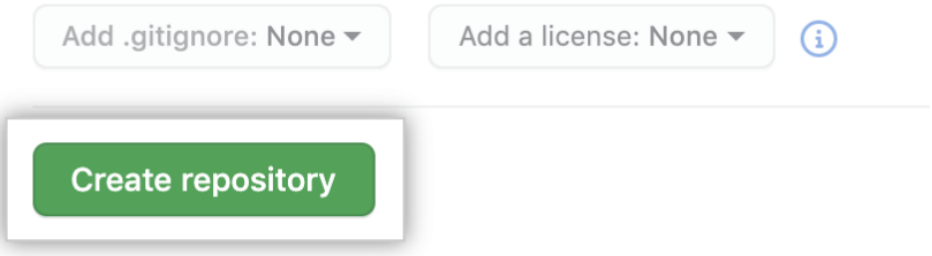
4.Choose a repository visibility. For more information, see "About repositories."



5.Select Initialize this repository with a README.



6.Click Create repository.

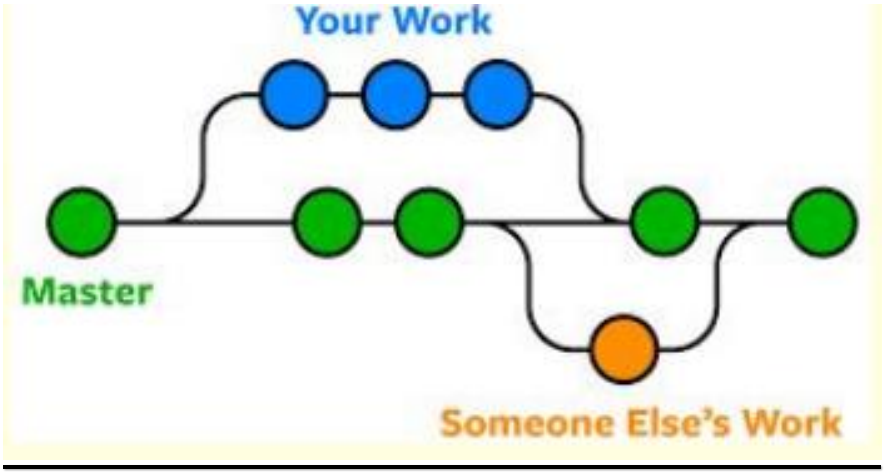


Aim: Create and Visualize Branches

- Branching in git
- Various Commands in Git Branching .
- Important Conclusions.
- Advantages of Branching.
- Merging of Branch.
- Merge Conflict.

Branching in git

- While working on real time projects code base, branching is one of mandatory and unavoidable concept.
- Till now whatever files created and whatever commits we did, all these happened in **master branch**.
- Master branch is the default branch/ main branch in git. Generally main source code will be placed in master branch.
- Need of creating branch Assume we required to work on new requirements independently, then instead of working in the master branch, we can create a separate branch and we can work in that branch, related to that new requirement without affecting main branch.



To Create a new branch

git branch <branch name>

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git branch newbranch
```

To View list of all available branches

git branch

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (newbranch)
$ git branch
  master
* newbranch
```

*symbol tells us about in which branch we are in currently.

To create and navigate a branch

git checkout -b <branch name>

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (newbranch)
$ git checkout -b branch3
Switched to a new branch 'branch3'
```

Delete branch

git branch -d <branch name>

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git branch -D branch3
Deleted branch branch3 (was a0c1761).
```

-D flag is useful in order to delete unmerged branch forcefully,
But before this it is mandatory to checkout from that branch which is to be deleted.

Important Conclusions:

1. All branches are isolated to each other.
2. In GIT branching, logical duplication of files will be happend.

Advantages of Branching

1. We can enable Parallel development.
2. We can work on multiple flows in isolated way.
3. We can organize source code in clean way.
4. Implementing new features will become easy
5. Bug fixing will become easy.
6. Testing new ideas or new technologies will become easy.

Merging of Branch

git merge <branch name>

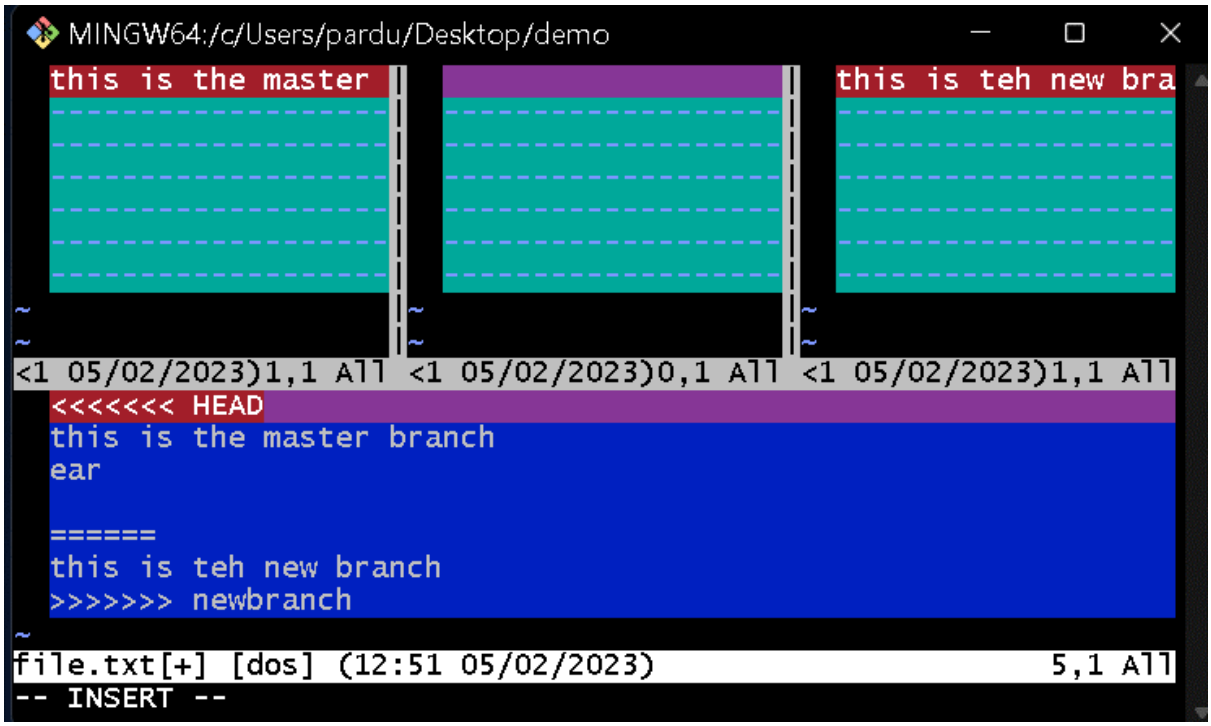
```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git checkout newbranch
Switched to branch 'newbranch'

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (newbranch)
$ git add .

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (newbranch)
$ git commit -m "changes done"
[newbranch d503cfe] changes done
 1 file changed, 1 insertion(+)

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (newbranch)
$ git checkout master
Switched to branch 'master'

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git merge newbranch
Updating a0c1761..d503cfe
Fast-forward
 file.txt | 1 +
 1 file changed, 1 insertion(+)
```

Merge Conflict

It is an event that takes place when Git is unable to automatically resolve differences in code between two commits

• git mergetool and then return and then use :wq to exit from blue screen.

```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git merge newbranch
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master|MERGING)
$
```



• Manually, we resolve the conflict and by using :wq command we leave from this blue screen.
• Again, by doing the commit, the branch will get merged.

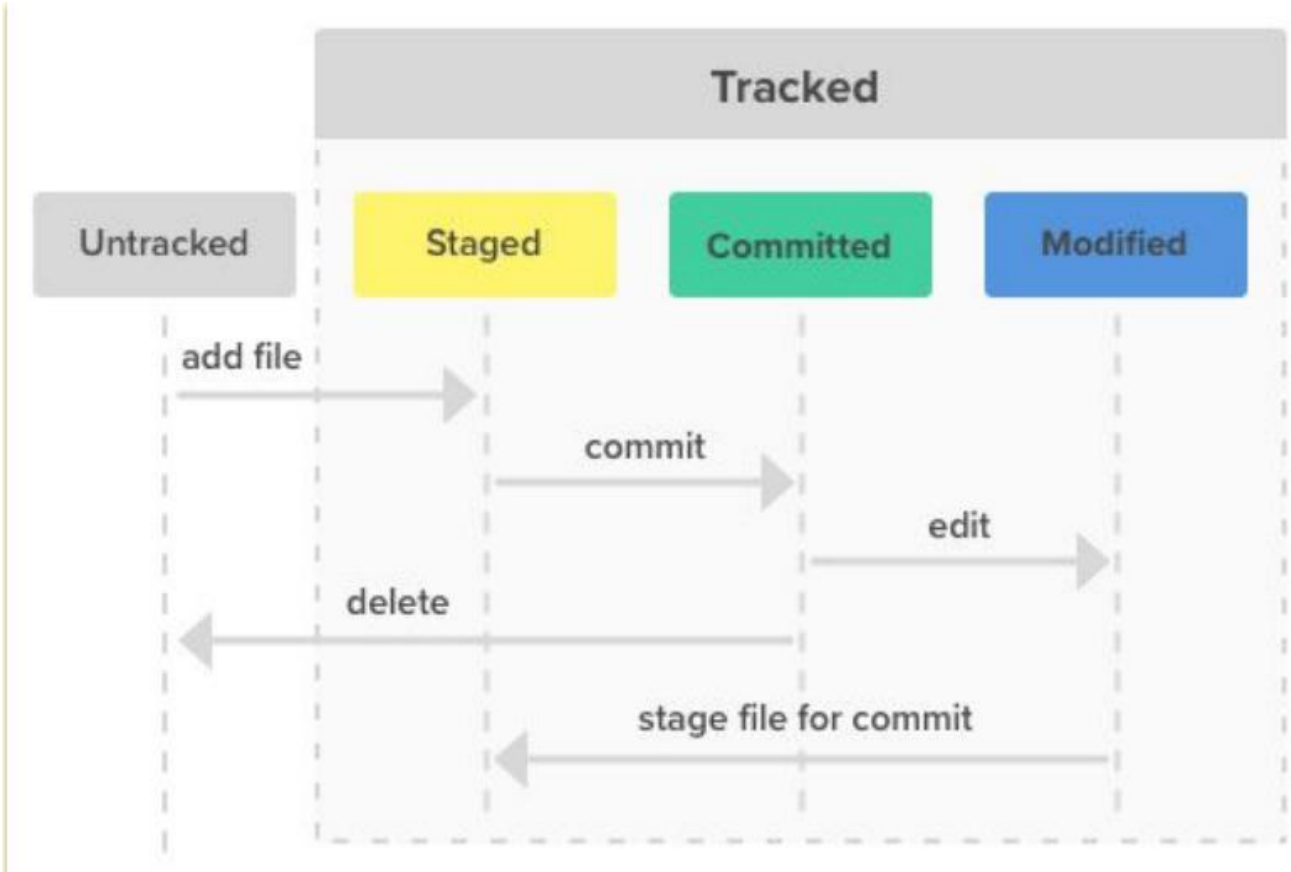
```
pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (newbranch)
$ git commit -m "Merge Conflict Resolved "
[newbranch 1fb9a3d] Merge Conflict Resolved
 1 file changed, 1 insertion(+), 1 deletion(-)

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (newbranch)
$ git checkout master
Switched to branch 'master'

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$ git merge newbranch
Updating 4f6d121..1fb9a3d
Fast-forward
 file.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

pardu@Asusvivobook_15 MINGW64 ~/Desktop/demo (master)
$
```

Aim: Git lifecycle description.

• Git 3-Stage architecture.

Git is one of the premier **distributed version control systems** available for programmers and corporates. In this article, we will see details about how a project that is being tracked by git proceeds with workflow i.e **Git Life Cycle.**

As the name suggests is regarding different stages involved after cloning the file from the repository. It covers the git central commands or main commands that are required for this particular version control system



Workflow of Git Life Cycle

The workflow of the Git as follows:

• We will create a branch on which we can work on and later we will merge it with master

• **Clone:** First, when we have code present in the remote repository, we clone to local to form something called a local repository.

• **Modifications/Adding Files:** we perform several CS181 Page 26 of 31 developments on the existing files or may as well add new files. Git will monitor all these activities and will log them
We need to move the content that we require to transform to the master to the staging area by using git commands and the snapshot of staged files will be saved in the git staging area

• Once we commit the code is available on the local repo but to send it to the master repo we need to perform PUSH operation

• If someone else is working on the same branch then there will be a possibility that he might have added his changes to the master by push. So we need to perform PULL operation before the PUSH operation if multiple people are working on the same branch and this workflow as shown below.

• Once the target branch is updated we need to get all the required approvals so that merge operation with the master is allowed.