GIT AND GITHUB

By Chaitanya Raj Nandan

Contents

1 GitHub Overview

GitHub is a web-based platform used for version control and collaborative software development. It allows multiple people to work on projects together, track changes, and manage code efficiently.

GitHub is a **Distributed Version Control System (DVCS)** that allows developers to manage and collaborate on code projects effectively.

Here is a short brief:

1.1 Key Features:

1. **Local Repositories**: Each user has a complete copy of the repository, enabling offline work and full access to the project's history.
2. **Branching and Merging**: Users can create branches for new features or bug fixes without affecting the main codebase, and then merge their changes back into the project.
3. **Commit History**: Every change is recorded with a commit, including details about the author, date, and description of the change.
4. **Collaboration**: Multiple developers can work on the same project simultaneously, with tools to push, pull, and merge changes from various contributors.
5. **Conflict Resolution**: GitHub provides mechanisms to handle conflicts that arise when changes overlap, ensuring smooth integration of different contributions.

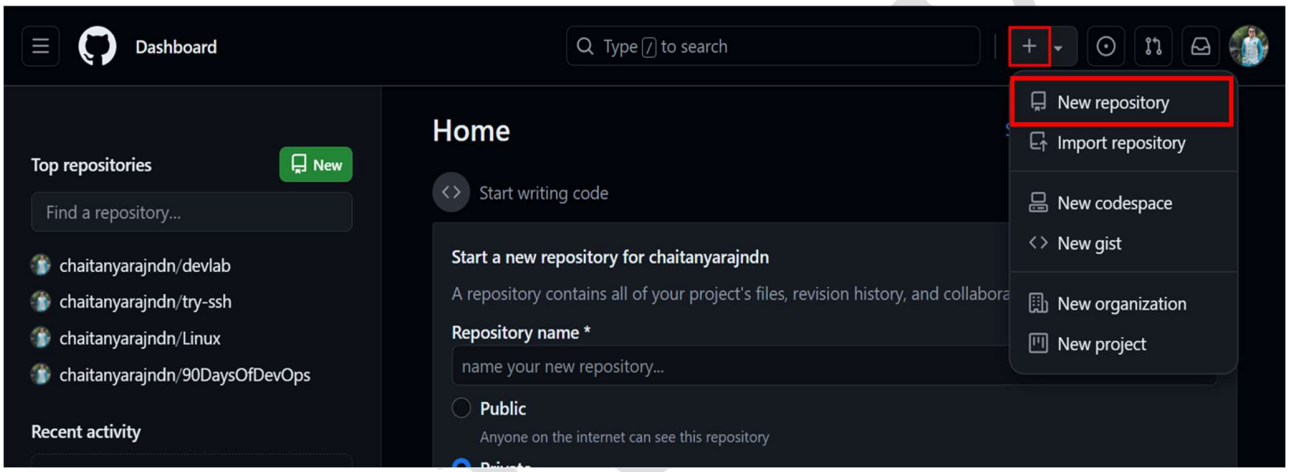
2 Creating a Repository on GitHub:

2.1 Sign in to GitHub:

1. Open GitHub in your web browser.
2. If you do not have an account, create one by clicking on the "Sign up" button. If you already have an account, click on "Sign in" and enter your credentials.

2.2 Create a New Repository:

1. Once signed in, click on the “+” icon in the upper right corner of the page, next to your profile picture.
2. Select "New repository" from the dropdown menu.

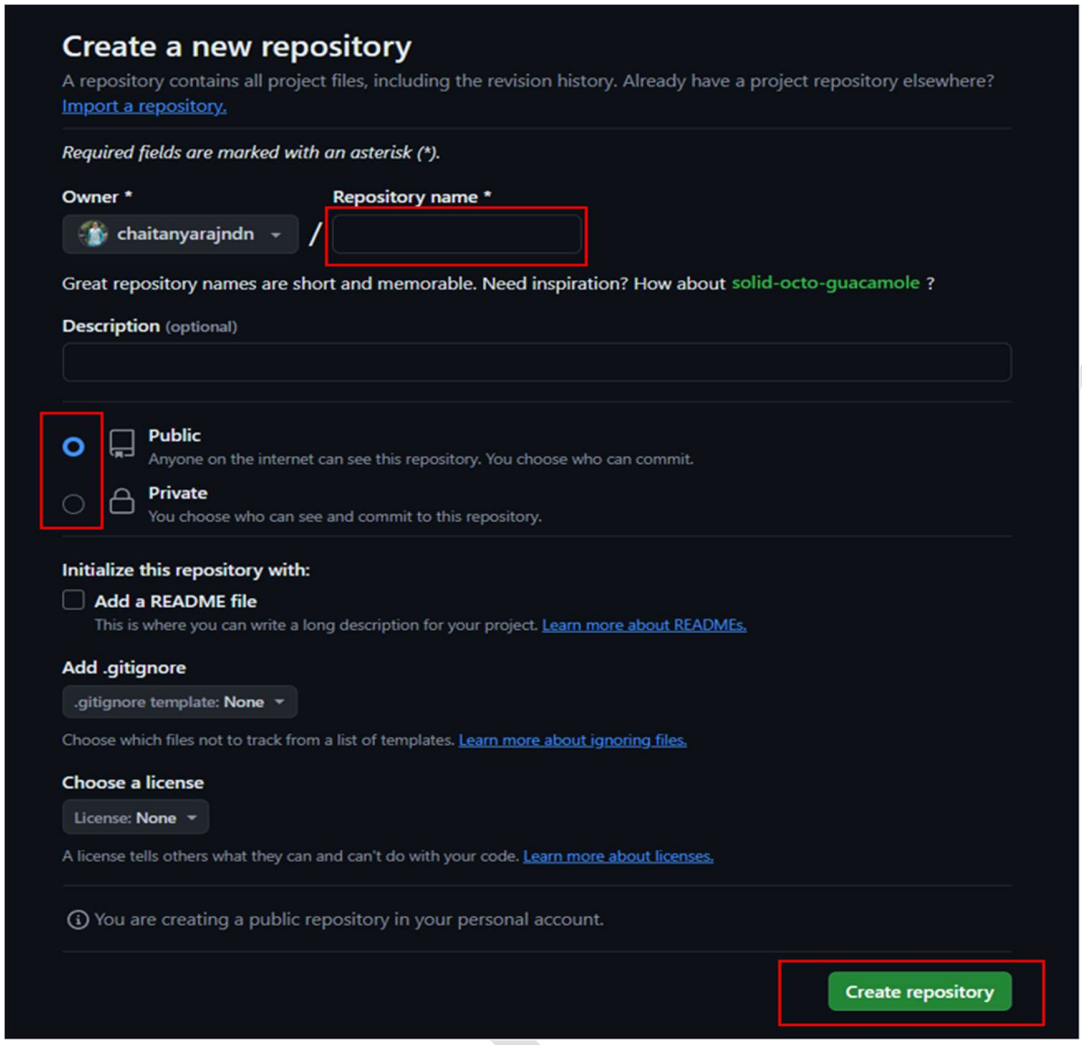


2.3 Fill-up Repository Details:

1. **Repository name:** Enter a name for your repository. This name should be unique to your account.
2. **Description:** Optionally, add a description for your repository to explain what it's about.
3. **Public or Private:** Choose the visibility of your repository. Public repositories are visible to anyone, while private repositories are only visible to you and the collaborators you choose.
4. **Initialize this repository with a README:** If you check this option, GitHub will add a README file to your repository. This is useful for adding an introduction and documentation for your project.
5. **Add. gitignore:** You can choose a `.gitignore` template to exclude certain files from your repository. This is useful for ignoring files that are not necessary for your project, like temporary files created by your text editor.
6. **Choose a license:** You can select a license for your project, which dictates how others can use your code.

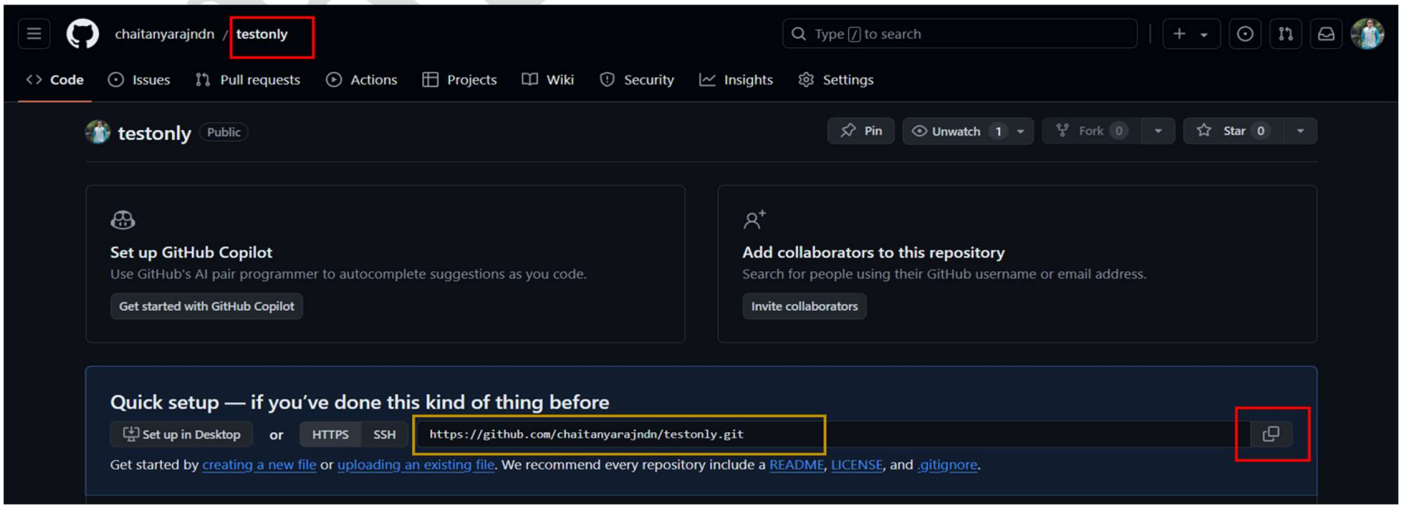
2.4 Click Create Repository:

1. Click the "**Create repository**" button at the bottom of the page.



2.5 Repository Setup:

1. After creating the repository, GitHub will provide you with instructions on how to set up your repository locally if you need to. You can clone the repository to your local machine using the URL provided.



3 Delete Repository in GIT HUB / Remote / Central Repository

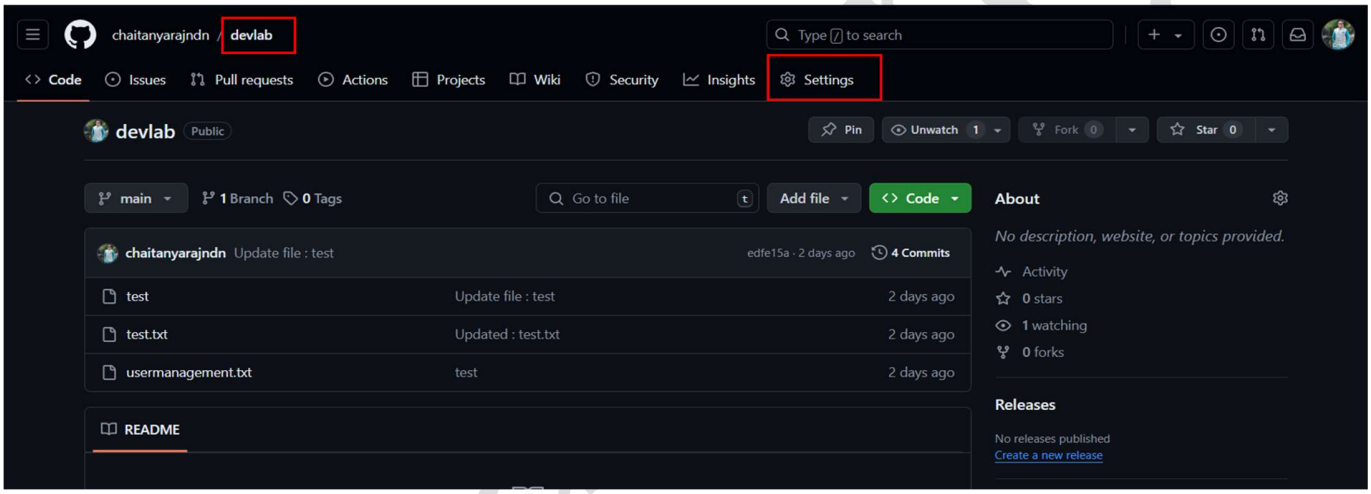
Deleting a repository on GitHub is a straightforward process.

!!!! IMPORTANT NOTES !!!

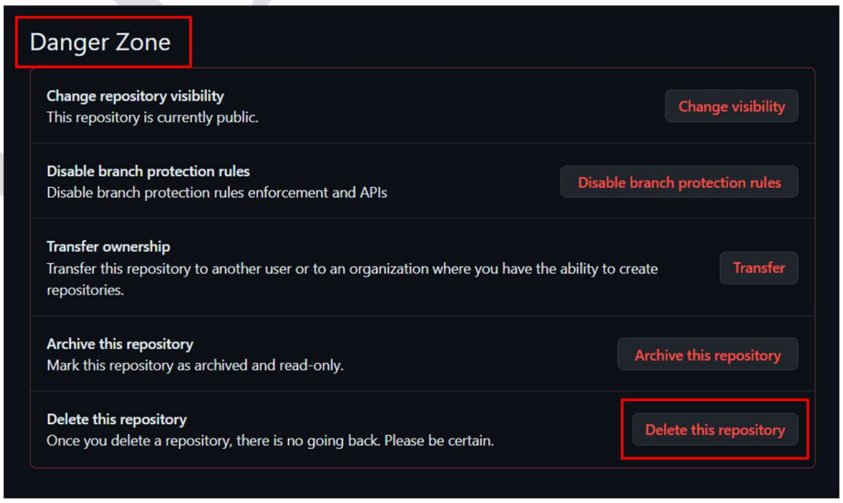
- **Irreversible Action**: Deleting a repository is irreversible. Once deleted, all issues, pull requests, and wiki pages associated with the repository are also deleted.
- **Back-Up**: Make sure to back up any data you might need before deleting the repository.
- **Permissions**: Only users with administrative rights to the repository can delete it.

Here is a step-by-step guide, By following these steps, you can safely delete a repository on GitHub.

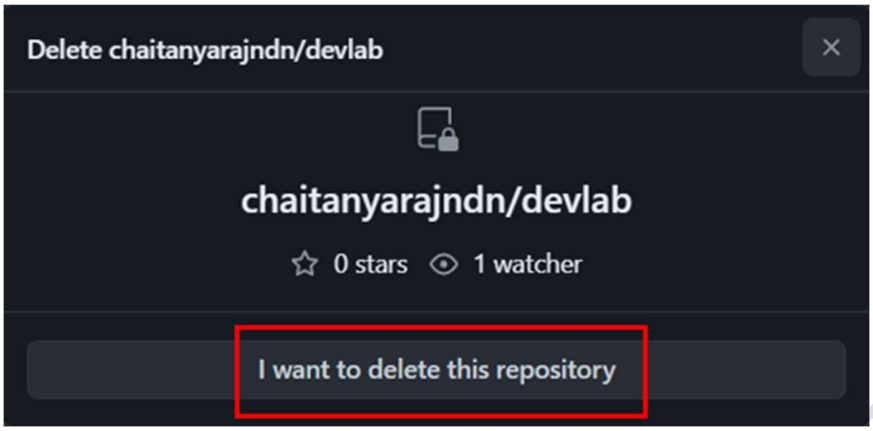
1. **Sign In**: Log in to your GitHub account.
2. **Navigate to the Repository**: Go to the repository you want to delete. You can find it under your repositories list on your profile or organization page.
3. **Settings**: Click on the "Settings" tab located on the right side of the repository's menu.



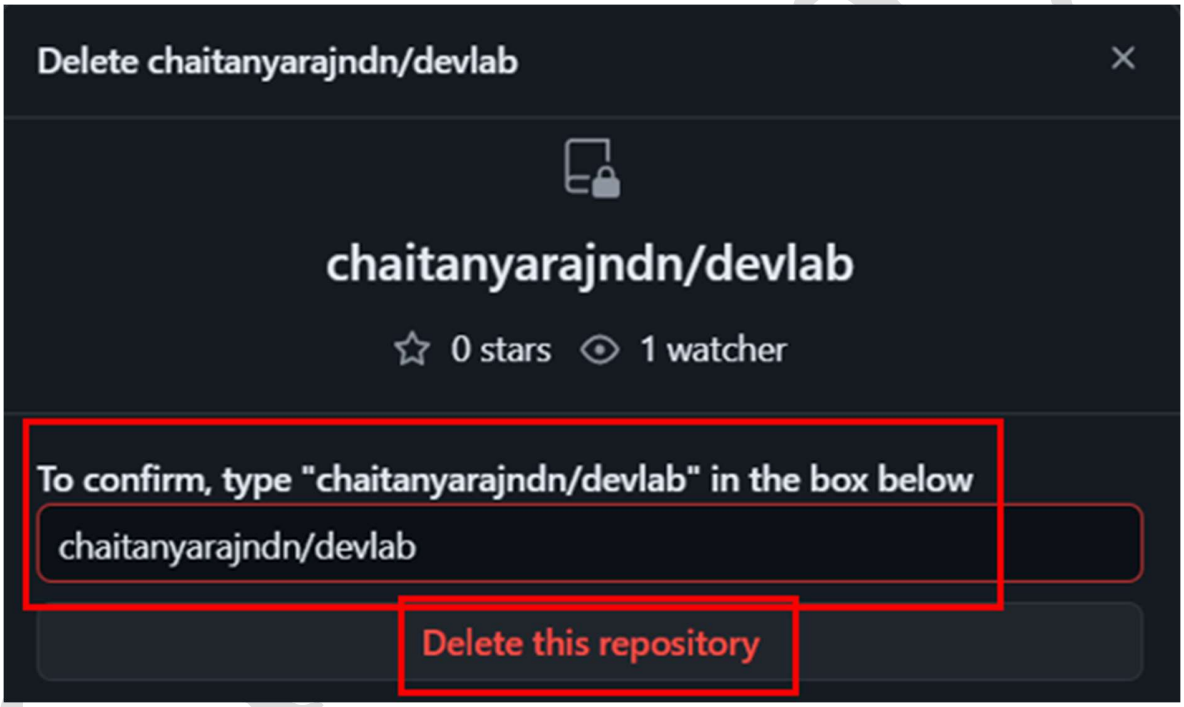
4. **Scroll Down**: Scroll down to the bottom of the Settings page until you find the "Danger Zone."
5. **Delete this Repository**: Click the "Delete this repository" button.



6. **Confirm Deletion**: GitHub will ask you to confirm the deletion. You will need to type the repository name to confirm that you understand the consequences of this action.



7. **Final Confirmation**: Click on the "I understand the consequences, delete this repository" button to permanently delete the repository.



4 Connect/ADD Git-Hub Repository with Local Repository

1. Download and Open **GIT BASH terminal** OR **any Linux terminal** you have locally.

2. cd to the directory you want to make/use as GitHub directory.

WARNING !!!! dONOT USE your local MAIN DIRECTORY AS GIT HUB DIRECTORY. Please create a NEW different directory and cd into that:

3. **`git init`** --- to initiate the directory into a GIT HUB Directory as **master** Directory.

   Note: In Linux Terminal / GIT BASH / Local / Workspace – Parent Directory is called as Master
   But in Remote / GIT HUB - Parent Directory is called as main.

   `git init -b main` --- to initiate the directory into a GIT HUB Directory as **main** Directory.

```
chetan@Chaitanya-Raj:~/abcgit$ ll
total 8
drwxrwxrwx  2 chetan chetan 4096 Jul 15 19:18 ./
drwxr-x--- 10 chetan chetan 4096 Jul 15 19:18 ../
chetan@Chaitanya-Raj:~/abcgit$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/chetan/abcgit/.git/
chetan@Chaitanya-Raj:~/abcgit$ ll
total 12
drwxrwxrwx  3 chetan chetan 4096 Jul 15 19:18 ./
drwxr-x--- 10 chetan chetan 4096 Jul 15 19:18 ../
drwxr-xr-x  7 chetan chetan 4096 Jul 15 19:18 .git/
chetan@Chaitanya-Raj:~/abcgit$
```

4. To check the version of Git installed on your system, you can use the **`git --version`** command.

   `git --version`

```
chetan@Chaitanya-Raj:~/agithub/test$ git --version
git version 2.34.1
chetan@Chaitanya-Raj:~/agithub/test$
```

5. **`git config --global user.name`** "chaitanyaxxxxxxx"
6. **`git config --global user.email`** "chaxxxxxxxxxxx@gmail.com"

7. Check the URLs of remote repositories associated with the local repository.

   `git remote -v`

The `-v` flag stands for "verbose," which means it will show both the fetch and push URLs for each remote repository.

At beginning there is no any Remote repository attached. We must attach the repository link of GIT HUB GUI / Remote to local Repository.

```
chetan@Chaitanya-Raj:~/git2$ git remote -v
chetan@Chaitanya-Raj:~/git2$
```

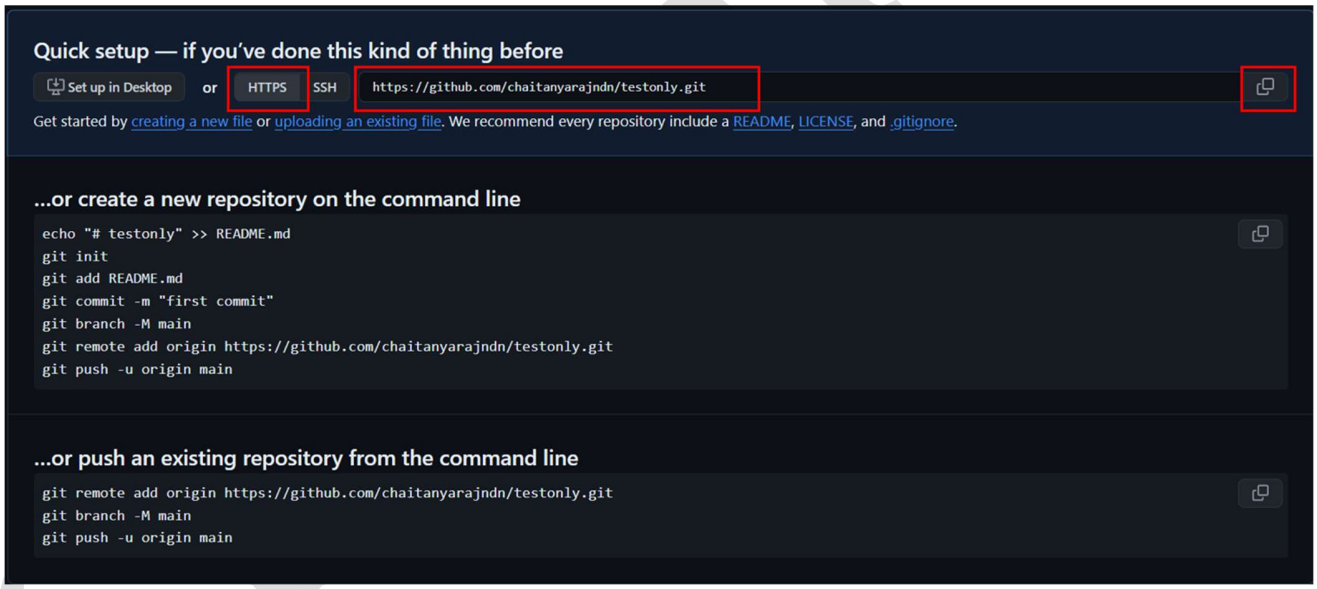
8. Add a remote repository to your local Git repository. This is typically done to connect your local repository to a remote repository hosted on a service like GitHub, GitLab, Bitbucket, or another Git server.

   Now there are 3 ways to add/connect remote repository link to local repository.

4.1 To add/connect remote repository link to local repository via HTTP link.

Go to your remote repository which you want to add to local repository.

On option **CODE, GO TO HTTPS. Copy the URL.**



Now go to the Linux terminal. Move to the directory where you want to connect above remote repository.

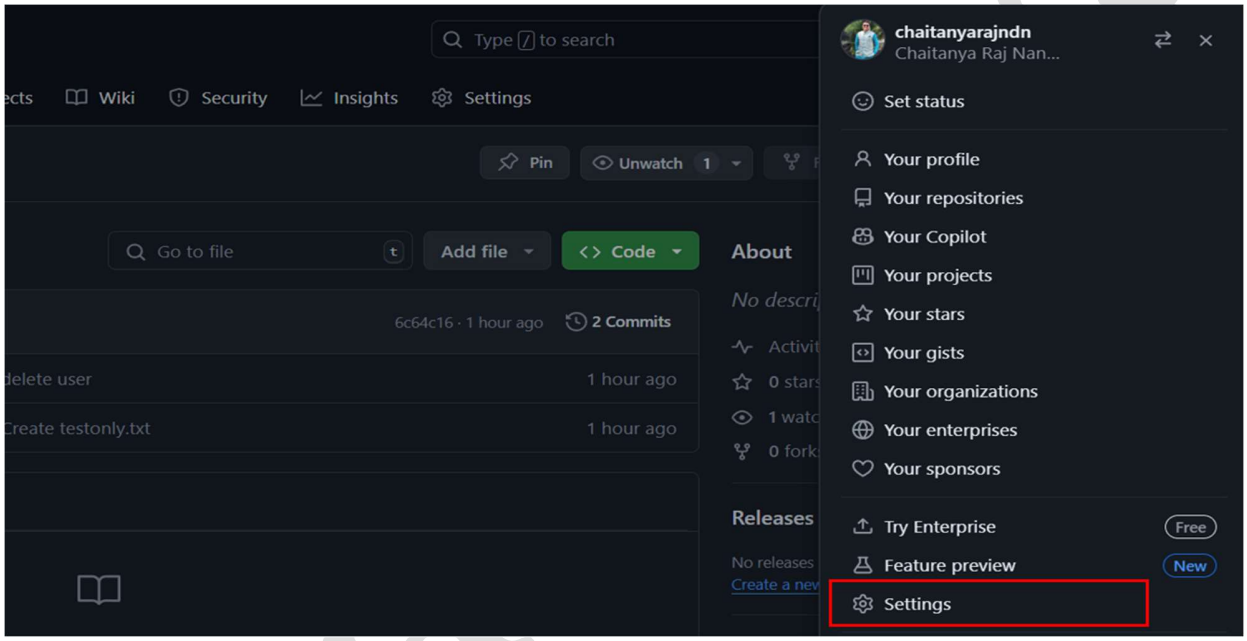
Type:
`git remote add origin <HTTP Repository LINK>`

For example:
`git remote add origin https://github.com/chaitxxxxxxxx/testonly.git`

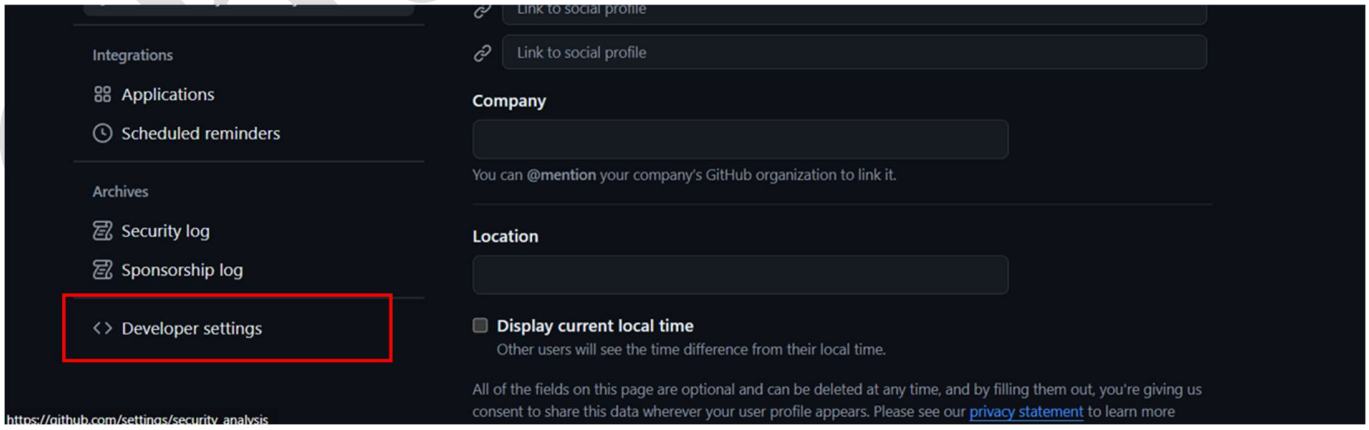
```
chetan@Chaitanya-Raj:~/github$ git remote -v
origin  https://github.com/chaitanyarajndn/Linux.git (fetch)
origin  https://github.com/chaitanyarajndn/Linux.git (push)
chetan@Chaitanya-Raj:~/github$
```

4.2 To add/connect remote repository link to local repository via PAT (Private Access Token) Key.

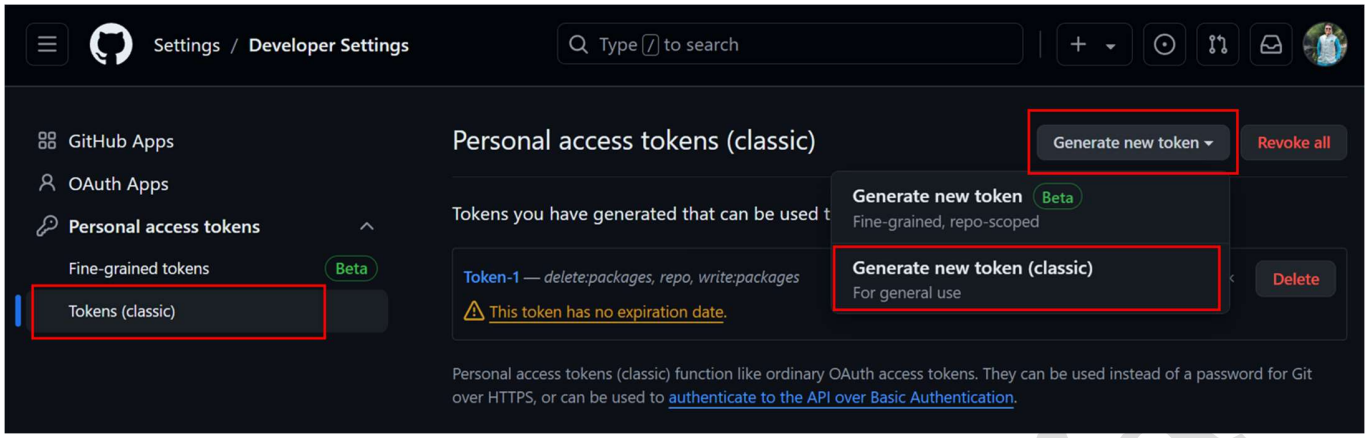
Login to your GIT HUB GUI. Go to setting from right top corner.



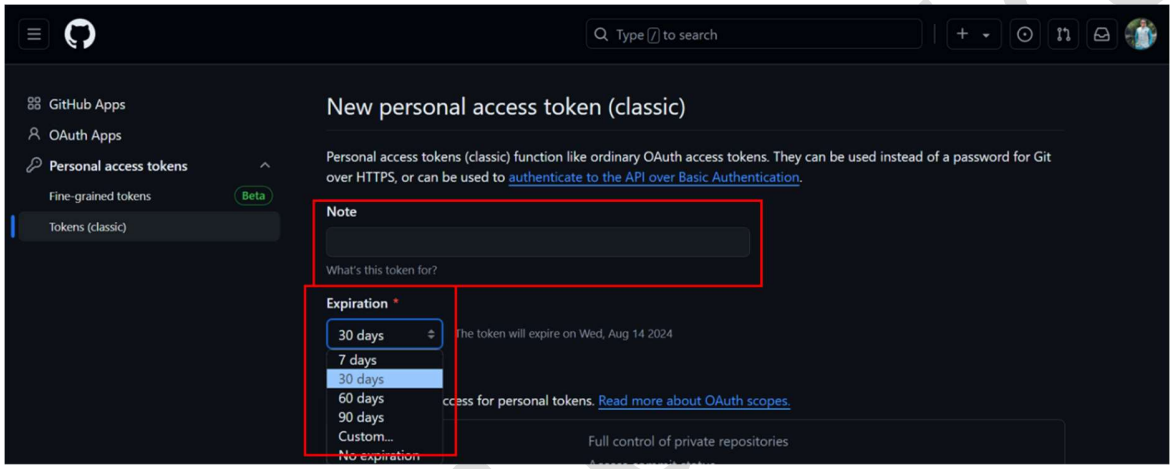
Scroll to bottom and find "**Developer Setting**".



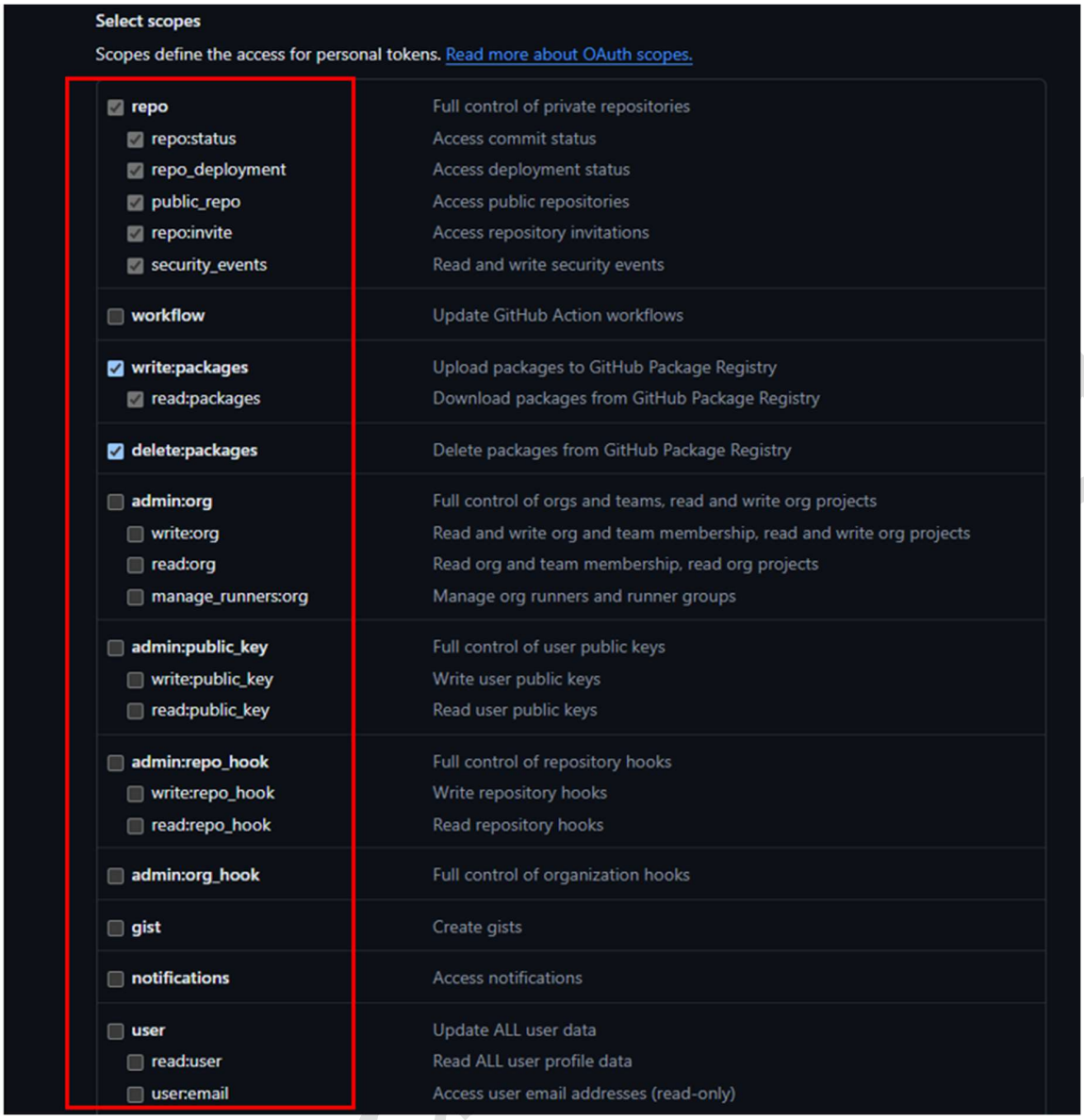
Go to **Tokens (Classics) → Generate New Token → Generate New Token (Classic)**



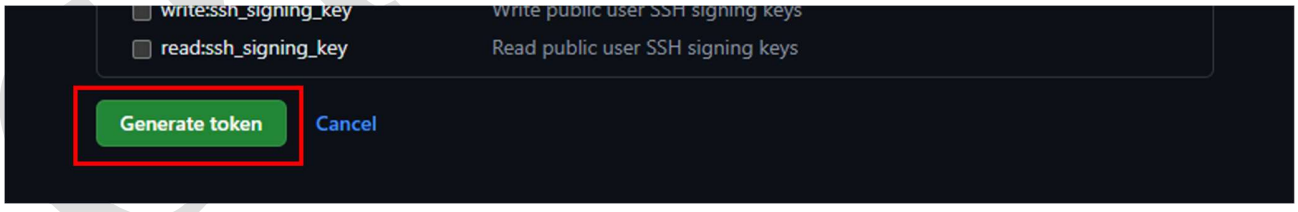
Write name for your token. And Select the expiry date for this token.



Check mark on all the list of access that you want provide for your remote repository with this PAT Token. Like Read, write, push, pull, create, update, edit, delete, etc.

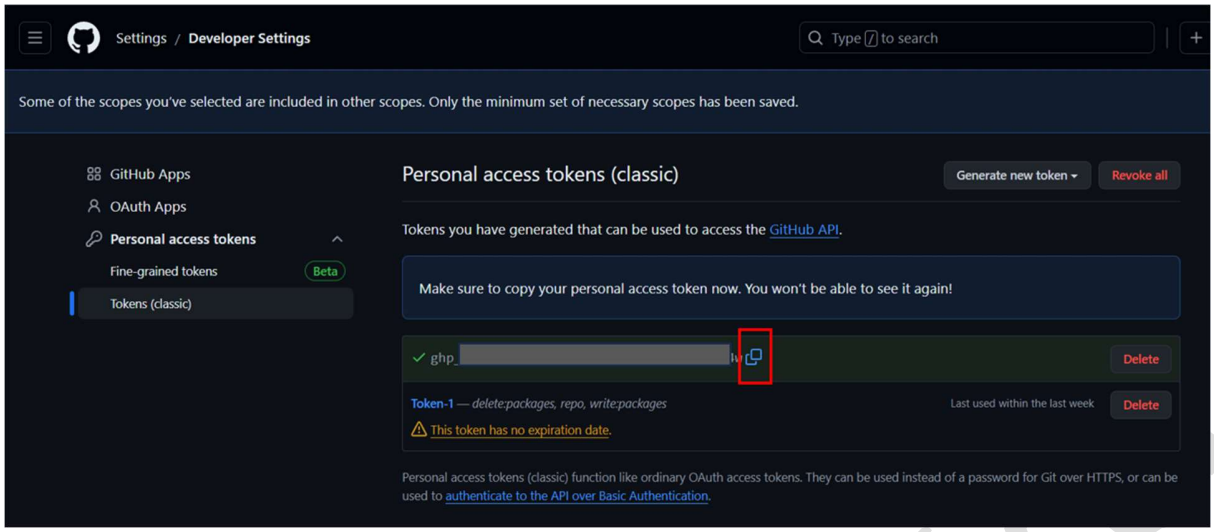


Then select "Generate Token".



Copy the TOKEN and save it to notepad or word pad where you can keep it safe.

IMPORTANT!!! This PAT Token key won't be visible if you refresh that particular page in your browser. So, it is advisable to COPY AND SAVE IT IN REAL TIME on some file or txt editor as soon as Token get created and displays on you screen !!!!



Now go to the Linux terminal. Move to the directory where you want to connect above remote repository.

Type:
`git remote add origin https://<PAT>@github.com/<Username>/<REPO>.git`

Example:
`git remote add origin https://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@github.com/chaitanyarajndn/testonly.git`

Check if the remote repository has been added into your local repository:

Type: **`git remote -v`**

```
chetan@Chaitanya-Raj:~/git2$ git remote add origin https://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@github.com/chaitanya
rajndn/testonly.git
[redacted]
chetan@Chaitanya-Raj:~/git2$ git remote -v
origin  https://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@github.com/chaitanyarajndn/testonly.git (fetch)
origin  https://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@github.com/chaitanyarajndn/testonly.git (push)
chetan@Chaitanya-Raj:~/git2$
```

4.3 To add/connect remote repository link to local repository via SSH Key.

1. **Generate SSH Key (if not already done) on Local Terminal.**

   Run below command to generate ssh key :
   `ssh-keygen`

```
chetan@Chaitanya-Raj:~/.ssh$ ls
id_rsa  id_rsa.pub  known_hosts  known_hosts.old
chetan@Chaitanya-Raj:~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC16BWhMMZdOpn6giKVGcDrzi22ScX+FwfTYZazl4D3ShYBfcBCsraJhXcUq29MdfV90KHO1r
kBy0CEPaVkbBVfFzqU/sToBiDdgKX+drc3HeKNmJAN00+gT8TwSch6q1T+6k/8SB0bdptL0IlxWscxMpgcSwlN0IPtUmD3YmV+wcfGDteIMZ0s
+p[redacted]RZE
di2Lq7qponktdrcYlFovYEMwjTwS7Q6DeNOFAR84bTIpYDqhYKAVoavo0QDTdueoMQLwZHaCIT4kbMdDh1/u/fODiyWIwqRSZHKXRnDrJ9wBcY
++mwkiTxwjXHafNgikRK1lBt4p7GcEkreN7I26hGivt61+6AVererP5mXXbqvQnbXrcrTGYoWRgD9eEBVAmtB2TWYuUCP6s9YeC0rI2B18wXA1
M= chetan@Chaitanya-Raj
chetan@Chaitanya-Raj:~/.ssh$
```

2. **Copy the SSH Public Key:**
   Once the key pair is generated, copy the SSH public key to your clipboard. You can do this using:

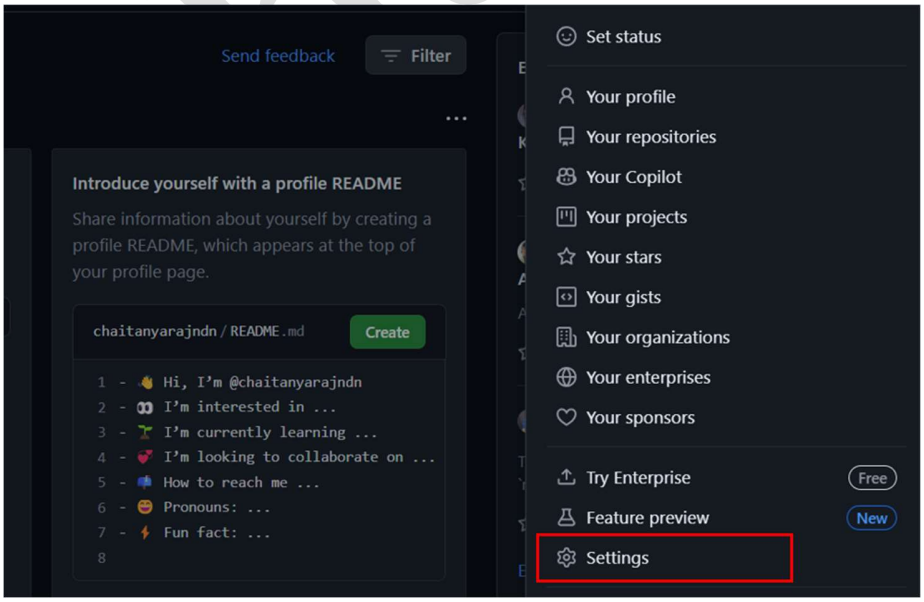
   `cat <SSH-Key.pub>`

Note : This command will display your SSH public key. Copy the entire output.

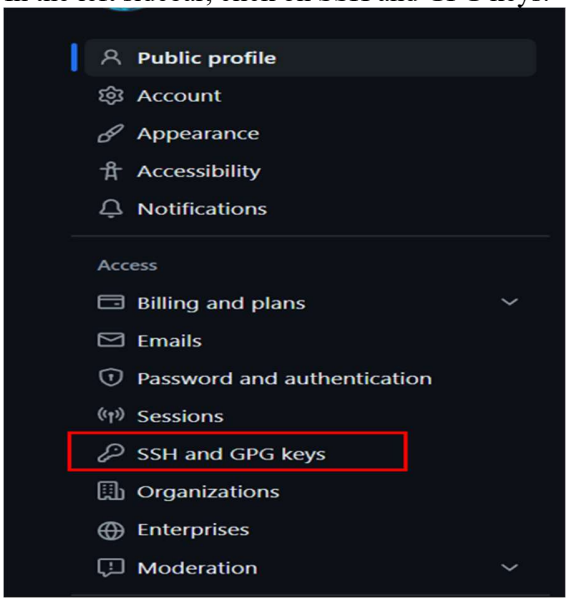
```
chetan@Chaitanya-Raj:~/.ssh$
chetan@Chaitanya-Raj:~/.ssh$ ls
id_rsa  id_rsa.pub  known_hosts  known_hosts.old
chetan@Chaitanya-Raj:~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC16BWhMMZdOpn6giKVGcDrzi22ScX+FwfTYZazl4D3ShYBfcBCsraJhXcUq29MdfV90KHO1r
kBy0CEPaVkbBVfFzqU/sToBiDdgKX+drc3HeKNmJAN00+gT8TwSch6q1T+6k/8SB0bdptL0IlxWscxMpgcSwlN0IPtUmD3YmV+wcfGDteIMZ0s
+pI4vH4QudewdmkIQGyq2hAmxSda+f/BP4N+GfLUd4/SvwTFzAWqL3lkCtxlBWMG8H8xxUPENw67hw9zZntk4JXiJ/T5QmFsefetWLH/cDVRZE
di2Lq7qponktdrcYlFovYEMwjTwS7Q6DeNOFAR84bTIpYDqhYKAVoavo0QDTdueoMQLwZHaCIT4kbMdDh1/u/fODiyWIwqRSZHKXRnDrJ9wBcY
++mwkiTxwjXHafNgikRK1lBt4p7GcEkreN7I26hGivt61+6AVererP5mXXbqvQnbXrcrTGYoWRgD9eEBVAmtB2TWYuUCP6s9YeC0rI2B18wXA1
M= chetan@Chaitanya-Raj
chetan@Chaitanya-Raj:~/.ssh$
```

3. **Add SSH Key to GitHub:**

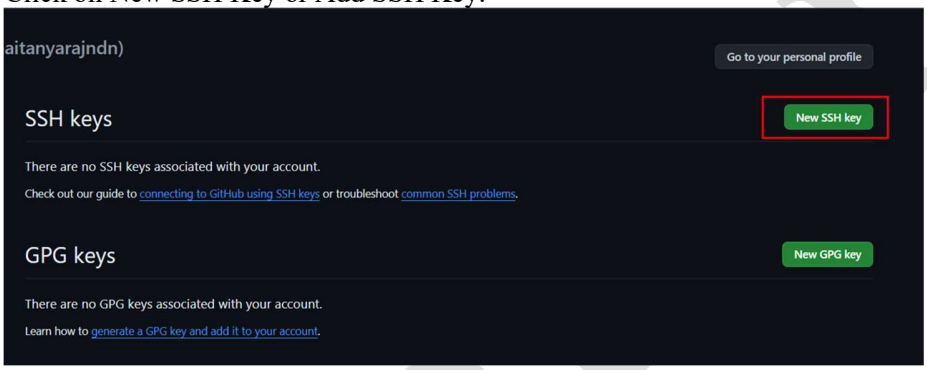
- Log in to your GitHub account.
- Click on your profile icon at the top right and select **Settings.**



- In the left sidebar, click on SSH and GPG keys.



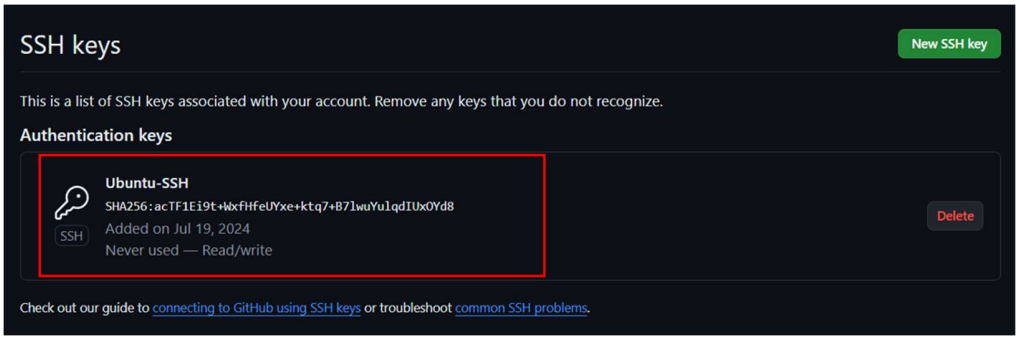
- Click on New SSH Key or Add SSH Key.



Paste the SSH Key:

- In the "Title" field, give your SSH key a descriptive label (e.g., "Ubuntu-SSH").
- Paste your SSH public key into the "Key" field copied in **STEP 2** Above.

- Click **Add SSH Key.**



4. Steps to Add Git Repository Link to Local Terminal via SSH

A. Copy the SSH URL: Obtain the SSH URL of the Git repository you want to clone. It typically looks like **`git@hostname:path/to/repo.git`.**

B. Open Terminal: Open your terminal application (Command Prompt on Windows, Terminal on macOS/Linux).

C. Verify: To ensure the SSH key was added successfully, you can test the connection to GitHub using SSH:

`ssh -T` `git@github.com`

```
chetan@Chaitanya-Raj:~$ ssh -T git@github.com
Hi chaitanyarajndn! You've successfully authenticated, but GitHub does not provide shell access.
chetan@Chaitanya-Raj:~$
```

D. Navigate to the Directory (optional): If you want to clone the repository into a specific directory, use the `cd` command to navigate there. For example:

`cd` `path/to/directory`

E. Add the Repository: Use the **`git remote add origin`** command followed by the SSH URL of the repository:

`git remote add origin` `<SHH URL of REMOTE Repository>`

```
chetan@Chaitanya-Raj:~$
chetan@Chaitanya-Raj:~$ cd sshtest
chetan@Chaitanya-Raj:~/sshtest$ git init -b main
Initialized empty Git repository in /home/chetan/sshtest/.git/
chetan@Chaitanya-Raj:~/sshtest$
chetan@Chaitanya-Raj:~/sshtest$
chetan@Chaitanya-Raj:~/sshtest$ git remote -v
chetan@Chaitanya-Raj:~/sshtest$
chetan@Chaitanya-Raj:~/sshtest$ git remote add origin git@github.com:chaitanyarajndn/sshtest.git
chetan@Chaitanya-Raj:~/sshtest$
chetan@Chaitanya-Raj:~/sshtest$ git remote -v
origin  git@github.com:chaitanyarajndn/sshtest.git (fetch)
origin  git@github.com:chaitanyarajndn/sshtest.git (push)
chetan@Chaitanya-Raj:~/sshtest$ cat > abc.txt
hi
hello
how do you do
chetan@Chaitanya-Raj:~/sshtest$ ls
abc.txt
chetan@Chaitanya-Raj:~/sshtest$ git add abc.txt
chetan@Chaitanya-Raj:~/sshtest$ git commit -m "1st ssh commit"
[main (root-commit) 0f193b2] 1st ssh commit
 1 file changed, 3 insertions(+)
 create mode 100644 abc.txt
chetan@Chaitanya-Raj:~/sshtest$ git log
commit 0f193b286a66e37df111858a947367e30db12fd2 (HEAD -> main)
Author: chaitanyarajndn <chaitanyarajndn@gmail.com>
Date:   Fri Jul 19 20:32:51 2024 +0530

    1st ssh commit
chetan@Chaitanya-Raj:~/sshtest$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 232 bytes | 232.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:chaitanyarajndn/sshtest.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
chetan@Chaitanya-Raj:~/sshtest$
```

!!!!! IMPORTANT !!!!!

I. `git remote add origin`

- **Purpose:** Adds a new remote repository.
- **Type:**
  `git remote add origin <URL>`
- Example:

  `git remote add origin https://ghp_AmyqDSXXXXXXXXXX009@github.com/chaiXXXXXXndn/testonly.git`
- **Behaviour:** This command is used when you are setting up a new remote repository for the first time. The `<name>` is typically `origin` for the main repository, but it can be any name you choose. The `<url>` is the URL of the remote repository.
- **Outcome:** Creates a new remote entry with the specified name and URL.

II. `git remote set-url --add origin`

- **Purpose:** Adds a new URL to an existing remote repository.
- **Type: `git remote set-url --add origin <URL>`**

  For example:
  `git remote set-url --add origin https://ghp_AmyqDSCD9hfhfhfhfhfhpxXjxu091009@github.com/chaixxxxxxxx/Linux.git`
- **Behaviour:** This command is used when you want to add an additional URL to an existing remote. It allows the remote to have multiple URLs, which can be useful for various purposes, such as push-only or fetch-only URLs.
- **Outcome:** Adds another URL to the specified remote. The remote will now have multiple URLs associated with it.

5 Clone GIT-HUB directory on Local (CLI) from Remote (GIT HUB GUI)

- Make sure you are on Local repository directory in Linux Terminal.
- Run below command to do cloning of remote repository to local:

  git clone <HTTP Repository LINK**>**

  For example:
  git clone https://github.com/chaixxxxxxxn/testonly.git

```
chetan@Chaitanya-Raj:~/github/testonly$ git clone https://github.com/chaitanyarajndn/testonly.git
Cloning into 'testonly'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.
chetan@Chaitanya-Raj:~/github/testonly$ ls
testonly
chetan@Chaitanya-Raj:~/github/testonly$ cd testonly/
chetan@Chaitanya-Raj:~/github/testonly/testonly$ ls
SS_Delete_User.txt  testonly.txt
chetan@Chaitanya-Raj:~/github/testonly/testonly$
```

- User can also clone remote repository to local repository using below command:

  git clone https://<PAT**>@github.com/<**Username**>/<**REPO**>.git**

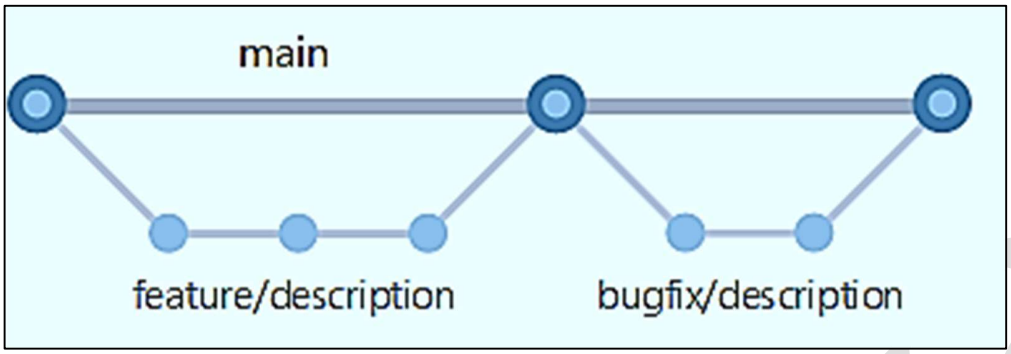
For example:
git clone
https://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@github.com/chaitanyarajndn/testonly.git

```
chetan@Chaitanya-Raj:~/github/testonly$ git clone https://[redacted]
Cloning into 'testonly'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (6/6), done.
chetan@Chaitanya-Raj:~/github/testonly$ ll
total 12
drwxr-xr-x 3 chetan chetan 4096 Jul 15 01:25 ./
drwxrwxrwx 6 chetan chetan 4096 Jul 15 01:20 ../
drwxr-xr-x 3 chetan chetan 4096 Jul 15 01:25 testonly/
chetan@Chaitanya-Raj:~/github/testonly$ cd testonly/
chetan@Chaitanya-Raj:~/github/testonly/testonly$ ls
SS_Delete_User.txt  testonly.txt
chetan@Chaitanya-Raj:~/github/testonly/testonly$
```

!!!! Note !!!!
User can still clone remote repository even if the remote repository URL is not linked to local Repository. See screenshot below.

```
chetan@Chaitanya-Raj:~/git2$ git clone https://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@github.com/chaitanyarajndn/testo
nly.git
Cloning into 'testonly'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 9 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.
chetan@Chaitanya-Raj:~/git2$ ll
total 16
drwxr-xr-x  4 chetan chetan 4096 Jul 15 16:22 ./
drwxr-x--- 10 chetan chetan 4096 Jul 15 02:20 ../
drwxr-xr-x  7 chetan chetan 4096 Jul 15 16:11 .git/
drwxr-xr-x  3 chetan chetan 4096 Jul 15 16:22 testonly/
chetan@Chaitanya-Raj:~/git2$ git remote -v
chetan@Chaitanya-Raj:~/git2$
```

6 Branching in GIT



Branching is a powerful feature in Git that allows you to diverge from the main line of development and continue to work without affecting that main line. This is useful for developing new features, fixing bugs, or experimenting. Here is an overview of how to work with branches in Git.

Key Concepts

- **Branch:** A branch in Git is simply a lightweight movable pointer to a commit.
- **Master/Main:** The default branch when you create a Git repository. Over time, many projects have moved from using master to main for the default branch name.
- **HEAD:** The pointer to the current branch reference, which tells Git where you are in the commit history.

#Common Branching Commands:

1. Creating a New Branch
   - To create a new branch: **`git branch <branch-name>`**
   - To create and switch to a new branch in one command: **`git checkout -b <branch-name>`**

```
chetan@Chaitanya-Raj:~/devlab$ git branch dev
chetan@Chaitanya-Raj:~/devlab$ git branch
  dev
* master
chetan@Chaitanya-Raj:~/devlab$ git checkout dev
Switched to branch 'dev'
chetan@Chaitanya-Raj:~/devlab$
```

2. Listing Branches
   - To list all branches: **`git branch`**
   - To list all branches, including remote branches: **`git branch -a`**

3. Switching Branches

   - To switch to an existing branch: **`git checkout <`**`branch-name`**`>`**

4. Renaming a Branch

   - To rename the current branch: **`git branch -m <`**`new-branch-name`**`>`**
   - To rename a branch from another branch: **`git branch -m <`**`old-branch-name`**`> <`**`new-branch-name`**`>`**

5. Deleting a Branch

   - To delete a branch: **`git branch -d <`**`branch-name`**`>`**
   - To force delete a branch (if it has not been merged): **`git branch -D <`**`branch-name`**`>`**

7 Git Add / Commit / Restore in LOCAL Repository.

Let's create 2 files:

```
chetan@Chaitanya-Raj:~/github/test$ echo -e " Hello \n hi \n jow are you" >> ABC.txt
chetan@Chaitanya-Raj:~/github/test$ echo -e " I am fine \n thankyou " >> DEF.txt
chetan@Chaitanya-Raj:~/github/test$
chetan@Chaitanya-Raj:~/github/test$ ll
total 16
drwxr-xr-x  2 chetan chetan 4096 Jul 15 15:11 ./
drwxrwxrwx 10 chetan chetan 4096 Jul 15 13:56 ../
-rw-r--r--  1 chetan chetan   26 Jul 15 15:10 ABC.txt
-rw-r--r--  1 chetan chetan   23 Jul 15 15:11 DEF.txt
chetan@Chaitanya-Raj:~/github/test$
```

```
echo -e " Hello \n hi \n how are you" >> ABC.txt
echo -e " I am fine \n thankyou " >> DEF.txt
```

1. Check the status : (Make sure GIT HUB repository Exist / created / Initialized , else create GIT HUB repository as mentioned in **section 4 step 1 to 6**)

   ```
   git status
   ```

```
chetan@Chaitanya-Raj:~/git2/test$ git status
On branch master

No commits yet

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

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

2. Now to stage these files, we use git add command. Basically, **git add** command is used to stage the files. We need to stage the file to segregate it from other files which do not require pushing to remote.

   ```
   git add <filename.txt>      ---- to add single file.
   git add .          ---------- to add all files/folder at once.
   ```

3. Check the status whether the file has been staged or not.

   ```
   git status  ---------- <to check on which branch the user is / which files are unstaged/staged >
   ```

   Once files are staged it will highlight in green.

```
chetan@Chaitanya-Raj:~/git2/test$ git add ABC.txt
chetan@Chaitanya-Raj:~/git2/test$ git add DEF.txt
chetan@Chaitanya-Raj:~/git2/test$ git status
On branch master

No commits yet

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

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

chetan@Chaitanya-Raj:~/git2/test$
```

NOTE:
Let's suppose ABC.txt and DEF.txt file are not correct and you want to **unstage** it, in that scenario, use below command:

`git restore --staged <file-name>`

OR

`git rm --cached <file-name>`

```
chetan@Chaitanya-Raj:~/github/test$ git restore --staged ABC.txt
chetan@Chaitanya-Raj:~/github/test$ git restore --staged DEF.txt
chetan@Chaitanya-Raj:~/github/test$ 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)
        ../devlab/
        ../notes/
        ./
        ../testonly/
        ../try-ssh/

nothing added to commit but untracked files present (use "git add" to track)
chetan@Chaitanya-Raj:~/github/test$
```

4. Commit the changes.

```
chetan@Chaitanya-Raj:~/git2$ git commit -m "add new files"
[master (root-commit) f5184f6] add new files
 2 files changed, 5 insertions(+)
 create mode 100644 test/ABC.txt
 create mode 100644 test/DEF.txt
chetan@Chaitanya-Raj:~/git2$
```

8 GIT LOG

The `git log` command in Git is used to display a history of commits in the repository. It shows a list of commits in reverse chronological order (newest commits first). Here are some common use cases and options for **`git log`**:

Basic git log: This command displays the commit history with each commit's hash, author, date, and commit message.

```
git log
```

Limiting the Number of Commits: Replace <limit> with the number of commits you want to display. For example, git log -n 5 shows the latest 5 commits.

```
git log -n <limit>
```

Viewing Commit Details: The -p option shows the patch (diff) introduced by each commit, which includes the changes made to files.

```
git log -p
```

Single Line Output: This option prints each commit on a single line, showing only the commit hash and the first line of the commit message.

```
git log --oneline
```

9 Git Restore to undo git add.

```
chetan@Chaitanya-Raj:~/agithub$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   abc.txt

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    test/abc.txt
        deleted:    test/def.txt

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

chetan@Chaitanya-Raj:~/agithub$ git restore --staged abc.txt
chetan@Chaitanya-Raj:~/agithub$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    test/abc.txt
        deleted:    test/def.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        LINUX/
        abc.txt
        git-branching.txt
```

The **`git restore`** command in Git is used to restore files in your working directory to their state at a specific commit or to unstage changes from the index (staging area). Here are some common use cases and examples of how **`git restore`** can be used:

Restore a file to its state at the last commit:

`git restore <file>`

This command restores the specified `<file>` in your working directory to the state it was in at the last commit (HEAD).

Unstage changes for a file:

`git restore --staged <file>`

This command moves changes from the index (staging area) back into the working directory for the specified `<file>`, effectively "unstaging" the changes.

10 Git diff to check the changes done in commit/ staging

The `git diff` command in Git is used to show changes between commits, commit and working tree, etc. Here are some common use cases and options for `git diff`:

1. **Compare Working Directory with the Staging Area (Index):**

   ```
   git diff
   ```

   This shows the difference between the current state of files in your working directory and the staging area (index).

2. **Compare Staging Area (Index) with the Last Commit:**

   ```
   git diff --staged
   ```

   This shows the changes between the files in the staging area and the last commit.

3. **Compare Two Commits:**

   ```
   git diff <commit1> <commit2>
   ```

   This shows the difference between two specified commits. You can use commit hashes, branch names, or tags as `<commit1>` and `<commit2>`.

4. **Compare a Commit with the Working Directory:**

   ```
   git diff <commit>
   ```

   This shows the difference between the specified commit and the current state of the working directory.

5. **Compare Specific Files:**

   ```
   git diff <file1> <file2>
   ```

   This shows the difference between two specified files.

6. **Viewing Unified or Context Diffs:** By default, `git diff` displays a unified diff format. You can also view a context diff by using the `--unified=<n>` option where `<n>` is the number of context lines.

7. **Other Options:**

   - `--color`: Adds color to the diff output.
   - `--word-diff`: Shows only the changed words in lines.
   - `--stat`: Shows a summary of changes instead of detailed diff.

EXAMPLE BELOW:

```
chetan@Chaitanya-Raj:~/diff$ cat def.txt

This is update of diff test

chetan@Chaitanya-Raj:~/diff$ echo -e "\n\n LETS MAKE CHANGES \n" >> def.txt
chetan@Chaitanya-Raj:~/diff$ cat def.txt

This is update of diff test



 LETS MAKE CHANGES

chetan@Chaitanya-Raj:~/diff$ git log --oneline
964ee0d (HEAD -> main, origin/main) this is second diff test
8847ad6 lets check diff commit 1st try
chetan@Chaitanya-Raj:~/diff$ git add def.txt
chetan@Chaitanya-Raj:~/diff$ git diff --staged
diff --git a/def.txt b/def.txt
index 46676cb..be4a27c 100644
--- a/def.txt
+++ b/def.txt
@@ -1,3 +1,7 @@

 This is update of diff test

+
+
+ LETS MAKE CHANGES
+
chetan@Chaitanya-Raj:~/diff$ git commit -m "this is update to def.txt"
[main bf1e2d2] this is update to def.txt
 1 file changed, 4 insertions(+)
chetan@Chaitanya-Raj:~/diff$ git log --oneline
bf1e2d2 (HEAD -> main) this is update to def.txt
964ee0d (origin/main) this is second diff test
8847ad6 lets check diff commit 1st try
chetan@Chaitanya-Raj:~/diff$ git diff bf1e2d2 964ee0d
diff --git a/def.txt b/def.txt
index be4a27c..46676cb 100644
--- a/def.txt
+++ b/def.txt
@@ -1,7 +1,3 @@

 This is update of diff test

-
-
- LETS MAKE CHANGES
-
chetan@Chaitanya-Raj:~/diff$
```

11 Git UNDO commit changes using git Revert and Reset

Removing a commit in Git can be done in several ways, depending on the context and what you aim to achieve. Below are some common methods to remove a commit:

➢ **Using git reset**

The git reset command is used to undo changes by moving the HEAD pointer and resetting the staging area. This can be done in two main ways: soft and hard reset.

1. Soft Reset
A soft reset will move the HEAD to a previous commit but leave your working directory and the staging area intact. This is useful if you want to undo commits but keep your changes.

Move the HEAD to the specified commit:

```
git reset --soft <commit-hash>
```

```
chetan@Chaitanya-Raj:~/git2/test/dummy$ git log --oneline
df1c99b (HEAD -> master) file added file1
f5184f6 add new files
chetan@Chaitanya-Raj:~/git2/test/dummy$ git reset --soft df1c99b
chetan@Chaitanya-Raj:~/git2/test/dummy$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2.txt
        ../../testonly/

nothing added to commit but untracked files present (use "git add" to track)
chetan@Chaitanya-Raj:~/git2/test/dummy$
```

2. Hard Reset
A hard reset will move the HEAD to a previous commit and discard all changes in the working directory and staging area. This is useful if you want to completely remove changes.

Move the HEAD to the specified commit and discard changes:

```
git reset --hard <commit-hash>
```

```
chetan@Chaitanya-Raj:~/git2$ git reset --hard  f5184f6
HEAD is now at f5184f6 add new files
chetan@Chaitanya-Raj:~/git2$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test/dummy/
        testonly/

nothing added to commit but untracked files present (use "git add" to track)
chetan@Chaitanya-Raj:~/git2$ ~
```

➢ **Using git revert**

The git revert command creates a new commit that undoes the changes from a specified commit. This is useful for undoing changes without rewriting history.

!!! IMPORTANT !!! GIT REVERT COMMAND will throw CONFLICT ERROR if you do add and commit multiple times on same file.

So, either you Update your file and save it in another filename every time for each commit so that during revert command it will remove that particular commit id file.

OR

Use this command when user commits multiple files in every commit and there if user get any wrong commit, user can revert that commit id. !!!!!

Revert the specified commit:

git revert <commit-hash>

```
chetan@Chaitanya-Raj:~/agithub/newtest$ git add jkl.txt
chetan@Chaitanya-Raj:~/agithub/newtest$ git commit -m "this is last good update"
[master 0362cff] this is last good update
 1 file changed, 2 insertions(+)
 create mode 100755 newtest/jkl.txt
chetan@Chaitanya-Raj:~/agithub/newtest$ git log --oneline
0362cff (HEAD -> master) this is last good update
67cc6f3 this is bad update
1873a72 this is second update
cadff5c this is 1ST update
ae55295 (origin/master)  this is last good commit
ab29864  this is bad commit
ce4bc25 this is 2nd good commit
db29c93 1st update - good
7c3199c adding 1st commit
chetan@Chaitanya-Raj:~/agithub/newtest$ git revert 67cc6f3
[master 299d80c] Revert "this is bad update"
 1 file changed, 4 deletions(-)
 delete mode 100755 newtest/ghi.txt
chetan@Chaitanya-Raj:~/agithub/newtest$ git revert 1873a72
[master 3cfbaf0] Revert "this is second update"
 1 file changed, 3 deletions(-)
 delete mode 100755 newtest/def.txt
chetan@Chaitanya-Raj:~/agithub/newtest$
```

Let's take the case in the above screenshot, 4 commits has been done.

There are 4 files: ***abc.txt,***
def.txt,
ghi.txt,
jkl.txt

Each file has been committed in each commit.

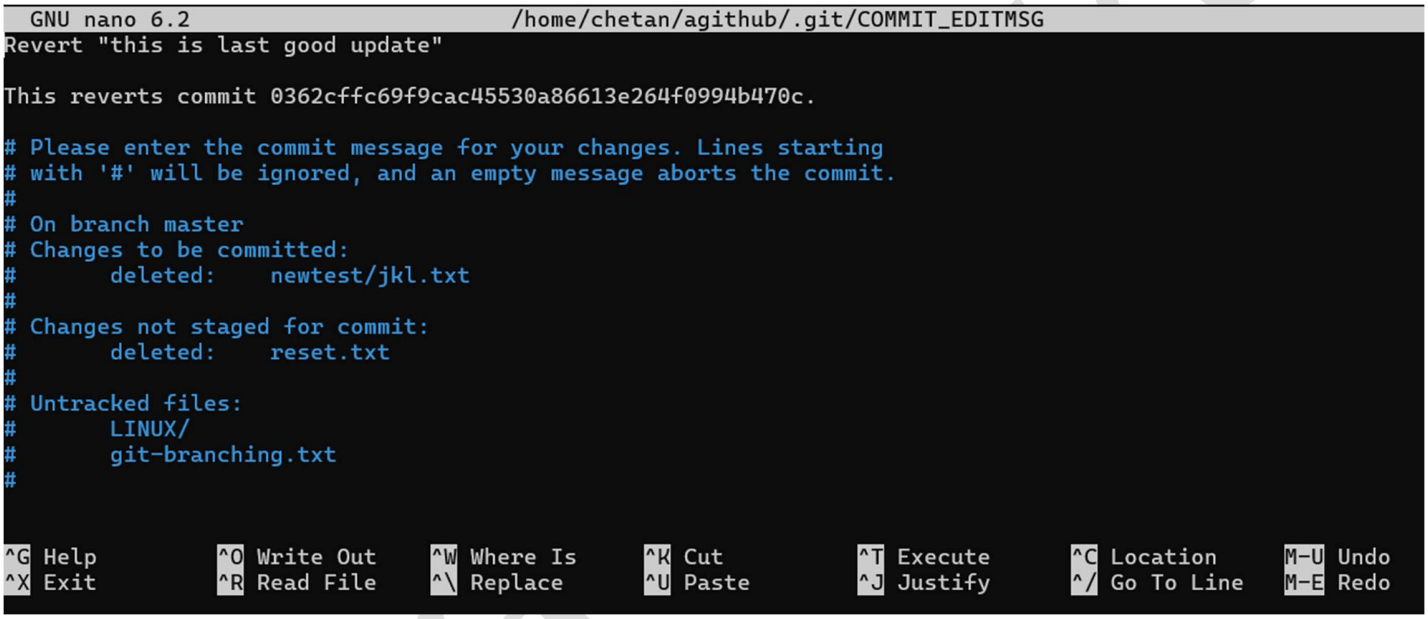
Out of 4 commits, 3RD **Commit** is not good file or incorrect file.

So. if user want to remove **3rd commit**, without affecting other commits, then use below command:

`git revert <commit-hash>`

```
chetan@Chaitanya-Raj:~/agithub/newtest$ git log --oneline
0362cff (HEAD -> master) this is last good update
67cc6f3 this is bad update
1873a72 this is second update
cadff5c this is 1ST update
chetan@Chaitanya-Raj:~/agithub/newtest$ git revert 67cc6f3
[master 299d80c] Revert "this is bad update"
1 file changed, 4 deletions(-)
delete mode 100755 newtest/ghi.txt
```

Note: When user enter revert command, NANO editor will display in screen. Press CRTL+X (^X), to save and exit



12 Push file / Directory from Local Repository to Remote Directory.

So, here we will create the dummy directory/ file and then we must follow below process step by step:

stage that file (git add) ➜ verify file status (git status) ➜ Commit that file (git commit -m "<comment>") ➜ verify file status again (git status) ➜ Push that file to remote server (git push <URL>).

2. Create a test file to check:

   `echo -e "hello \n how are you \n i am fine thankyou \n byee" >> check.txt`

3. Check the current status of file: the file must be unstaged and highlighted in red.

   `git status`

```
chetan@Chaitanya-Raj:~/github/testonly/testonly$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

nothing added to commit but untracked files present (use "git add" to track)
chetan@Chaitanya-Raj:~/github/testonly/testonly$
```

4. **`git add <filename.txt>`** ---- to add single file.
   `git add .` ---------- to add all files/folder at once.

5. **`git status`** ---------- <to check on which branch the user is / which files are unstaged/staged >

   Once files are staged it will highlight in green.

```
chetan@Chaitanya-Raj:~/github/testonly/testonly$ git add check.txt
chetan@Chaitanya-Raj:~/github/testonly/testonly$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   check.txt

chetan@Chaitanya-Raj:~/github/testonly/testonly$
```

6. **`git commit -m "<type any comment related to your activity>"`**

git commit command is used to commit the staged files from previous step. Once commit it is ready to push to REMOTE.

```
chetan@Chaitanya-Raj:~/github/testonly/testonly$ git commit -m "uploading check.txt file"
[main 7ffc426] uploading check.txt file
 1 file changed, 5 insertions(+)
 create mode 100644 check.txt
chetan@Chaitanya-Raj:~/github/testonly/testonly$
```

7. **`git status`** ---------- <to check on which branch the user is / which files are unstaged/staged >

```
chetan@Chaitanya-Raj:~/github/testonly/testonly$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
chetan@Chaitanya-Raj:~/github/testonly/testonly$
```

8. **`git log`** ---------- <to check the log file>

```
chetan@Chaitanya-Raj:~/github/testonly/testonly$ git log
commit 7ffc4265956361225988d881daebbff473050094 (HEAD -> main)
Author: chaitanyarajndn <chaitanyarajndn@gmail.com>
Date:   Mon Jul 15 01:45:39 2024 +0530

    uploading check.txt file
```

`git log -oneline`

to show all logs in one line

```
chetan@Chaitanya-Raj:~/github/devlab$ git log --oneline
edfe15a (HEAD -> main) Update file :  test
a25dc82 Updated : test.txt
7e24c27 test
ee8d2d6 (origin/main, origin/HEAD) Create test
chetan@Chaitanya-Raj:~/github/devlab$
```

git log command displays the logs about who have edited the file/staged the file/ push the file and it automatically creates the commit number so that we can dig more deeper into the log.

`git show <``commit number`**`>`**

```
chetan@Chaitanya-Raj:~/github/testonly/testonly$ git show 7ffc426595
commit 7ffc42659563612259 88d881daebbff473050094 (HEAD -> main)
Author: chaitanyarajndn <chaitanyarajndn@gmail.com>
Date:   Mon Jul 15 01:45:39 2024 +0530

    uploading check.txt file

diff --git a/check.txt b/check.txt
new file mode 100644
index 0000000..1489cf0
--- /dev/null
+++ b/check.txt
@@ -0,0 +1,5 @@
+hello \n how are you \n i am fine thankyou \n byee
+hello
+ how are you
+ i am fine thankyou
+ byee
chetan@Chaitanya-Raj:~/github/testonly/testonly$
```

9. **`git branch -M master`**

   It is optional may or may not require.

   So, git branch -M master renames the current branch to master. For example, if you are on a branch named main and you run git branch -M master, the main branch will be renamed to master.

   This command is particularly useful when you want to standardize branch names or switch the default branch from a different name to master.

   History behind the master/main branch:
   In Git, the terms "main" and "master" refer to branch names. Traditionally, "master" has been the default branch name in Git repositories, but many projects and platforms, including GitHub, have transitioned to using "main" as the default branch name. This change aims to be more inclusive and eliminate unnecessary references to master/slave terminology.

   Note: In Linux Terminal / GIT BASH / Local / Workspace – Parent Directory is called as Master
   But in Remote / GIT HUB - Parent Directory is called as main.

10. Once file is committed, push the file to remote repository using below command:

    - If you have linked / connected remote **repository link through HTTP,** then,

    `git push -u origin master`

```
chetan@Chaitanya-Raj:~/github$ git push -u origin master
Username for 'https://github.com': chaitanyarajndn
Password for 'https://chaitanyarajndn@github.com':
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 24 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (8/8), 2.13 KiB | 2.13 MiB/s, done.
Total 8 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/chaitanyarajndn/Linux.git
   89d89c7..0f73a72  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
chetan@Chaitanya-Raj:~/github$ git remote -v
origin  https://github.com/chaitanyarajndn/Linux.git (fetch)
origin  https://github.com/chaitanyarajndn/Linux.git (push)
chetan@Chaitanya-Raj:~/github$
```

- If you have connected remote **repository link through PAT Token Key**, then,

`git push`

```
chetan@Chaitanya-Raj:~/github/testonly/testonly$ git remote -v
origin  https://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@github.com/chaitanyarajndn/testonly.git (fetch)
origin  https://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@github.com/chaitanyarajndn/testonly.git (push)
chetan@Chaitanya-Raj:~/github/testonly/testonly$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 24 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 379 bytes | 379.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/chaitanyarajndn/testonly.git
   6c64c16..7ffc426  main -> main
chetan@Chaitanya-Raj:~/github/testonly/testonly$
```

13 Git pull file from REMOTE to LOCAL Repository.

Suppose you are on another local server and you want to pull files which was created in section **8 Push file / Directory from Local Repository to Remote Directory.**

You will pull from Remote Repository to Local Repository. In that case there should be GIT HUB repository initialized / created already in another local server.

(Make sure GIT HUB repository Exist / created / Initialized, else create GIT HUB repository as mentioned in **section 4 step 1 to 6**)

```
root@cloudserver-AkaCh9As:~/devlab# git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /root/devlab/.git/
root@cloudserver-AkaCh9As:~/devlab#
```

```
root@cloudserver-AkaCh9As:~/devlab# git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /root/devlab/.git/
root@cloudserver-AkaCh9As:~/devlab#
root@cloudserver-AkaCh9As:~/devlab#
root@cloudserver-AkaCh9As:~/devlab#
root@cloudserver-AkaCh9As:~/devlab# ll
total 12
drwxr-xr-x 3 root root 4096 Jul 16 17:49 ./
drwx------ 5 root root 4096 Jul 16 17:40 ../
drwxr-xr-x 7 root root 4096 Jul 16 17:49 .git/
root@cloudserver-AkaCh9As:~/devlab# git remote add origin git@github.com:chaitanyarajndn/devlab.git
root@cloudserver-AkaCh9As:~/devlab# git remote -v
origin  git@github.com:chaitanyarajndn/devlab.git (fetch)
origin  git@github.com:chaitanyarajndn/devlab.git (push)
root@cloudserver-AkaCh9As:~/devlab#
```

`git pull origin master`

```
root@cloudserver-AkaCh9As:~/devlab# git pull origin master
The authenticity of host 'github.com (20.207.73.82)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 9 (delta 2), reused 9 (delta 2), pack-reused 0
Unpacking objects: 100% (9/9), 1.17 KiB | 299.00 KiB/s, done.
From github.com:chaitanyarajndn/devlab
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> origin/master
root@cloudserver-AkaCh9As:~/devlab#
```

```
root@cloudserver-AkaCh9As:~/devlab# git remote add origin
https://github.com/chaitanyarajndn/devlab.git
root@cloudserver-AkaCh9As:~/devlab# git remote -v
origin  https://github.com/chaitanyarajndn/devlab.git (fetch)
origin  https://github.com/chaitanyarajndn/devlab.git (push)
root@cloudserver-AkaCh9As:~/devlab#
root@cloudserver-AkaCh9As:~/devlab#
root@cloudserver-AkaCh9As:~/devlab# git pull remote origin
fatal: 'remote' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
root@cloudserver-AkaCh9As:~/devlab# git push origin main
error: src refspec main does not match any
error: failed to push some refs to 'https://github.com/chaitanyarajndn/devlab.git'
root@cloudserver-AkaCh9As:~/devlab# git pull origin master
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 9 (delta 2), reused 9 (delta 2), pack-reused 0
Unpacking objects: 100% (9/9), 1.17 KiB | 599.00 KiB/s, done.
From https://github.com/chaitanyarajndn/devlab
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> origin/master
```

14 Git Merge

The git merge command is used in Git to integrate changes from different branches into a single branch.

Here is a basic overview of how it works:

14.1 Basic Usage:

To merge a branch into the current branch, you can use:

```
git merge <branch-name>
```

For example, if you are currently on the **main** branch and want to merge changes from **dev** branch into `main`, you would run:

```
git checkout main
git merge dev
```

14.2 Merge Strategies

- **Fast-forward Merge**: This happens when the current branch has not diverged from the branch being merged. Git will simply move the current branch pointer forward to the same commit as the branch being merged.

  ```
  git merge --ff-only <branch-name>
  ```
- **Three-way Merge**: This is used when the branches have diverged. Git will create a new commit that combines the changes from both branches.

  ```
  git merge <branch-name>
  ```
- **Squash Merge**: This method combines all the changes from the branch being merged into a single commit on the current branch.

  ```
  git merge --squash <branch-name>
  git commit
  ```
- **No-commit Merge**: This allows you to perform a merge without creating an automatic merge commit.

  ```
  git merge --no-commit <branch-name>
  ```

14.3 Handling Merge Conflicts

Sometimes, changes in the branches being merged conflict with each other. In such cases, Git will stop and allow you to resolve the conflicts manually.

1. **Start the merge process**: `git merge <branch-name>`
2. **Resolve conflicts**: Edit the conflicted files to resolve the conflicts. Git marks conflicts in the files with markers like `<<<<<<<` , `=======` , and `>>>>>>>`.
3. **Stage the resolved files**: `git add <resolved-file>`
4. **Complete the merge**: `git commit`

14.4 PROCESS FLOW OF MERGING BETWEEN Master and Dev Branch

Let us assume there is a file name called **reset.txt**,

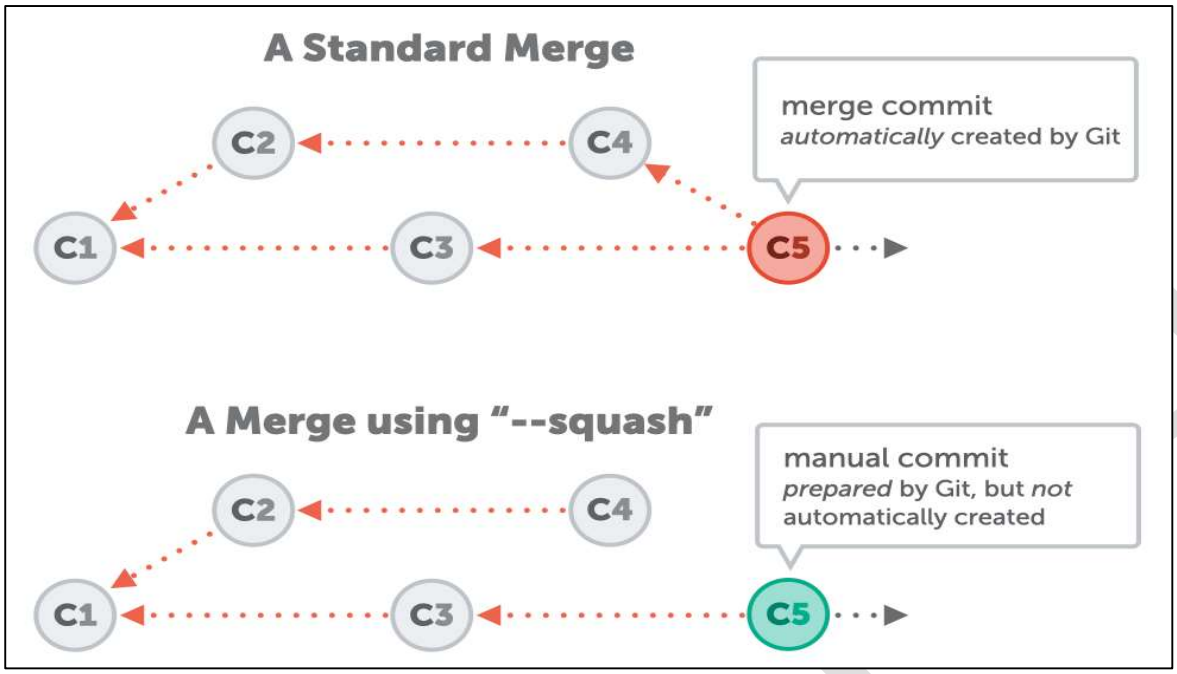
Now, this file will be updated by **DEV** branch and commit, then go back to master branch and merge with dev branch.

For Example:

```
chetan@Chaitanya-Raj:~/devlab$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        abc.txt
        git-branching.txt

nothing added to commit but untracked files present (use "git add" to track)
chetan@Chaitanya-Raj:~/devlab$ git branch dev
chetan@Chaitanya-Raj:~/devlab$ git branch
  dev
* master
chetan@Chaitanya-Raj:~/devlab$ git checkout dev
Switched to branch 'dev'
chetan@Chaitanya-Raj:~/devlab$ ll
total 24
drwxrwxrwx 3 chetan chetan 4096 Jul 16 18:19 ./
drwxr-x--- 9 chetan chetan 4096 Jul 16 18:21 ../
drwxr-xr-x 8 chetan chetan 4096 Jul 16 18:22 .git/
-rwxrwxrwx 1 chetan chetan   38 Jul 16 15:24 abc.txt*
-rwxrwxrwx 1 chetan chetan 1414 Jul 16 13:21 git-branching.txt*
-rwxr-xr-x 1 chetan chetan 1160 Jul 16 18:19 reset.txt*
chetan@Chaitanya-Raj:~/devlab$ echo -e "\n this update is done by dev branch to check "
>> reset.txt
chetan@Chaitanya-Raj:~/devlab$
chetan@Chaitanya-Raj:~/devlab$ git add reset.txt
chetan@Chaitanya-Raj:~/devlab$ git status
On branch dev
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   reset.txt

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

chetan@Chaitanya-Raj:~/devlab$ git commit -m "Updates done by Dev Branch"
[dev 84e9ce3] Updates done by Dev Branch
 1 file changed, 2 insertions(+)
chetan@Chaitanya-Raj:~/devlab$ git checkout master
Switched to branch 'master'
chetan@Chaitanya-Raj:~/devlab$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        abc.txt
        git-branching.txt

nothing added to commit but untracked files present (use "git add" to track)
chetan@Chaitanya-Raj:~/devlab$ git log
```

```
commit fc5c579481aee87b7322158d537bfc4817ad31af (HEAD -> master, origin/master)
Author: root <root@cloudserver-ns6uUNIO.mhc>
Date:   Tue Jul 16 14:47:34 2024 +0200

    updated reset.txt file

commit 4f41f207a7dc5ef5f0ebf030f7fa6ca4e8226634
Author: chaitanyarajndn <chaitanyarajndn@gmail.com>
Date:   Tue Jul 16 15:32:31 2024 +0530

    New commit to file reset.txt
chetan@Chaitanya-Raj:~/devlab$ git merge dev
Updating fc5c579..84e9ce3
Fast-forward
 reset.txt | 2 ++
 1 file changed, 2 insertions(+)
chetan@Chaitanya-Raj:~/devlab$ git log
commit 84e9ce3764cddeae7b056af6f8e4f437e3ba0b3c (HEAD -> master, dev)
Author: chaitanyarajndn <chaitanyarajndn@gmail.com>
Date:   Tue Jul 16 18:30:37 2024 +0530

    Updates done by Dev Branch

commit fc5c579481aee87b7322158d537bfc4817ad31af (origin/master)
Author: root <root@cloudserver-ns6uUNIO.mhc>
Date:   Tue Jul 16 14:47:34 2024 +0200

    updated reset.txt file

commit 4f41f207a7dc5ef5f0ebf030f7fa6ca4e8226634
Author: chaitanyarajndn <chaitanyarajndn@gmail.com>
Date:   Tue Jul 16 15:32:31 2024 +0530

    New commit to file reset.txt

chetan@Chaitanya-Raj:~/devlab$ git push origin master
Username for 'https://github.com': chaitanyarajndn
Password for 'https://chaitanyarajndn@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 24 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 329 bytes | 329.00 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.
To https://github.com/chaitanyarajndn/devlab.git
   fc5c579..84e9ce3  master -> master
```

14.5 Difference between Standard Merge and Squash Merge:



➢ **Standard Merge**

A standard `merge` incorporates all commits from the feature branch into the target branch. This retains the entire commit history from the feature branch. The result is a merge commit that has two parent commits: the latest commit on the target branch and the latest commit on the feature branch.

Pros:

- Maintains a detailed commit history, which can be useful for understanding the development process.
- Easier to see the sequence of changes and their context.

Cons:

- The commit history can become cluttered, especially if the feature branch has many small or experimental commits.
- Can make the commit history more complex and harder to follow.

➢ **Squash Merge**

A `squash merge` combines all the changes from the feature branch into a single commit on the target branch. This results in a cleaner commit history by consolidating multiple commits into one. After the squash merge, the feature branch commits are not retained as individual commits in the target branch's history.

Pros:

- Results in a cleaner and more concise commit history.
- Easier to track high-level changes and significant milestones.

Cons:

- Loses the detailed history of individual commits from the feature branch.
- Makes it harder to understand the step-by-step development process that led to the final changes.

Command:

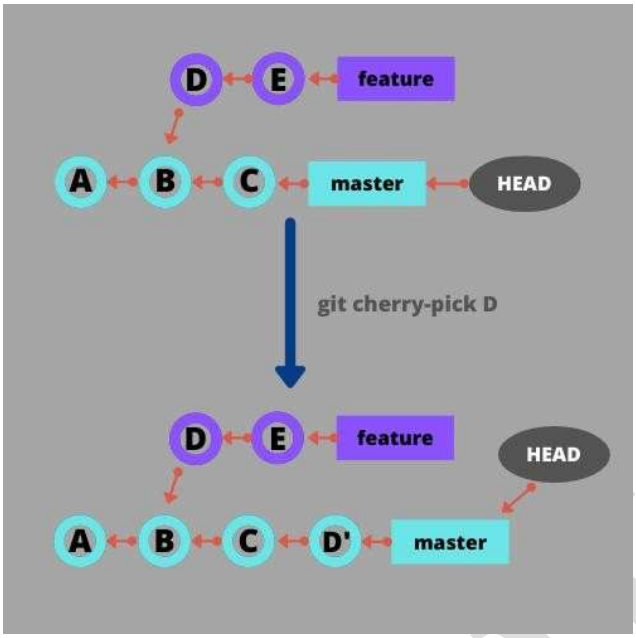
```
git merge --squash <branch-name>
git commit
```

15 Git Cherrypick

Git cherry-pick is a command used to apply a specific commit from one branch to another. It allows you to pick a single commit and apply it to the current branch, regardless of the commit's position in the source branch's history.

Use Cases:

- Applying a bug fix or a specific feature from one branch to another.
- Backporting changes from a feature branch to a maintenance branch.
- Incorporating changes from a specific commit without merging the entire branch.



Flowchart:

Here is how you would use git cherry-pick to apply a specific commit from one branch to another:

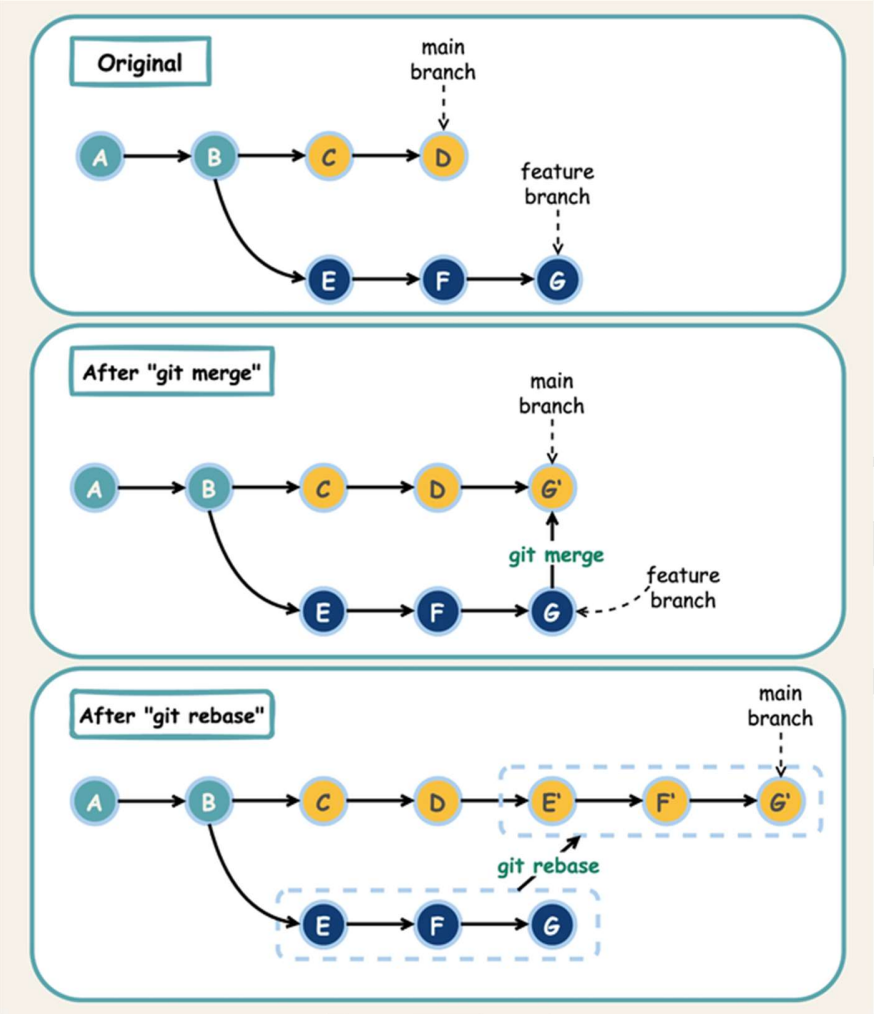
- **Identify the Commit**: Use git log to identify the commit you want to cherry-pick:

  ```
  git log
  ```

- **Cherry-pick the Commit:** Cherry-pick the desired commit using its hash:

  ```
  git checkout target-branch
  git cherry-pick <commit-hash>
  ```

16 Git Merge vs. Git Rebase



Git Merge:

- **Definition**: Git merge is a command that integrates changes from one branch into another. It creates a new commit that combines the changes from the specified branch into the current branch.
- **Use Cases**: Merge is suitable for combining work from different branches, such as feature branches or release branches, into the main branch (e.g., master or main).

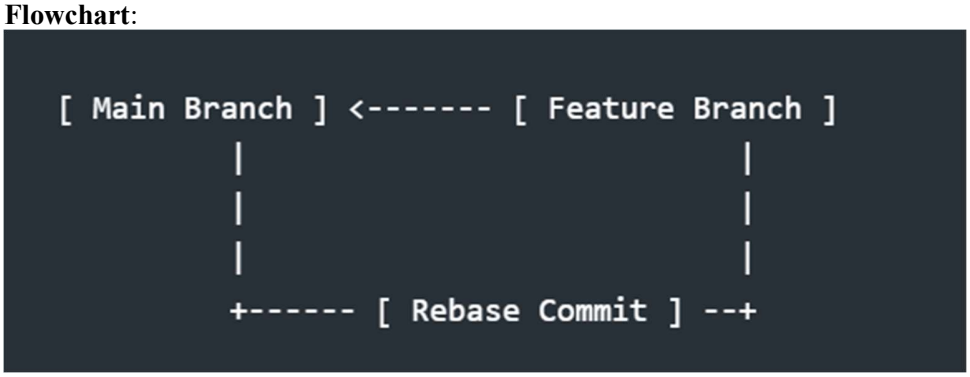
Flowchart:

```
[ Main Branch ] <-------- [ Feature Branch ]
        |                          ^
        |                          |
        |                          |
        +------------------ [ Merge Commit ]
```

Git Rebase:

- **Definition**: Git rebase is a command that rewrites the commit history by moving or combining commits from one branch onto another. It rewrites the project history, creating a linear sequence of commits.
- **Use Cases**: Rebase is suitable for maintaining a clean and linear commit history, especially in feature branches. It allows for a cleaner, more readable history without unnecessary merge commits.

Flowchart:



Comparison:

Merge:
- Creates a merge commit that integrates changes from one branch into another.
- Preserves the original commit history of both branches.
- Results in a non-linear history with merge commits.

Rebase:
- Rewrites the commit history by placing commits from one branch on top of another.
- Results in a linear commit history without merge commits.
- Can lead to conflicts if changes in the rebased branch conflict with changes in the target branch.

When to Use:

Merge:
- Use merge when you want to combine changes from one branch into another while preserving the original commit history.
- Useful for incorporating changes from feature branches into the main branch.

Rebase:
- Use rebases when you want to maintain a clean and linear commit history without merge commits.
- Useful for integrating changes from feature branches into the main branch while keeping the commit history clean and readable.

17 ISSUES : MERGE CONFLICT

ISSUE 1:

```
chetan@Chaitanya-Raj:~/github$ git push -u origin master
Username for 'https://github.com': chaitanyarajndn
Password for 'https://chaitanyarajndn@github.com':
To https://github.com/chaitanyarajndn/Linux.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/chaitanyarajndn/Linux.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
chetan@Chaitanya-Raj:~/github$ git pull origin master
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 8 (delta 2), reused 8 (delta 2), pack-reused 0
Unpacking objects: 100% (8/8), 1.71 KiB | 585.00 KiB/s, done.
From https://github.com/chaitanyarajndn/Linux
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> origin/master
hint: You have divergent branches and need to specify how to reconcile them.
hint: You can do so by running one of the following commands sometime before
hint: your next pull:
hint:
hint:   git config pull.rebase false  # merge (the default strategy)
hint:   git config pull.rebase true   # rebase
hint:   git config pull.ff only       # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
fatal: Need to specify how to reconcile divergent branches.
chetan@Chaitanya-Raj:~/github$ git rebase origin/master
Successfully rebased and updated refs/heads/master.
```

The error message you are seeing indicates that Git cannot automatically reconcile the changes between your local branch (`master`) and the remote `origin/master` branch because they have diverged. This situation occurs when there are conflicting changes in both branches that Git cannot merge automatically.

To resolve this issue, you prefer to rebase your changes on top of the remote branch (`origin/master`):

1. **Rebase Changes:** Rebase your local changes on top of `origin/master`:

   ```
   git rebase origin/master
   ```

   This command will replay your local commits on top of the latest `origin/master`. If there are conflicts, you'll need to resolve them during the rebase process.

2. **Continue Rebase (if needed):** If Git pauses the rebase due to conflicts, resolve them as instructed and continue the rebase:

   ```
   git rebase --continue
   ```

3. **Push Your Rebased Changes:** After resolving conflicts and completing the rebase, push your changes to the remote repository:

   ```
   git push origin master
   ```