  
  
  
GIT   
  
  
  
What is Git?

Git is a version control system that helps you keep track of changes in your code. It allows you to collaborate with others and manage your code history efficiently. Here are the basic concepts:

1. Repository (Repo): A repository is like a project folder where your code and its history are stored. You can create a Git repository for your project.

2. Commit: A commit is a snapshot of your code at a particular point in time. It records changes you've made to the code.

3. Branch: A branch is like a separate line of development. It's useful for working on new features or fixing bugs without affecting the main codebase.

4. Clone: Cloning is making a copy of a Git repository on your local machine.

5. Pull: Pulling means getting the latest changes from a remote repository (like GitHub) to your local repository.

6. Push: Pushing means sending your local commits to a remote repository. **git init**  
Pehle hm navigate krenge current directory/ folder pr cd my\_project  
  
Convert unversioned project ----->Git repository or initialize a new, empty repository

it means (iss ko run krne pr ekk nhi subdirectory create hogi .git ke naam se orr iss me meta data store hoga )

It prepares the repository to track changes to your project files.

2. add and commit changes

Adding Changes (Staging):

In Git, before you commit changes, you need to stage them. Staging allows you to select which changes you want to include in the next commit.

1. Check the Status:

**git status**

2. Add Changes to the Staging Area:

git add

To stage a specific file:

**git add <filename>>**

To stage all changes:

**git add .**

3. Verify the Staging:

After adding changes, you can use git status again to confirm that your changes are in the staging area.

Committing Changes:

Once you have staged your changes, you can commit them to your Git repository. A commit is like taking a snapshot of your code at that moment.

1. Commit Changes:

To commit your staged changes, use the git commit command with a descriptive message explaining what the commit accomplishes.

**git commit -m "Your descriptive commit message here"**

2. Commit History:

Commits create a history of your project. You can view the commit history using the git log command:

git log

3. Best Practices:

Make each commit meaningful, containing related changes.

Write clear and concise commit messages that describe what the commit does.

In summary, adding and committing changes in Git involves **staging** the changes you want to include in your next snapshot (commit) and then creating a commit with a descriptive message. This process allows you to keep a detailed history of your project and collaborate effectively with others on code changes.

Summary   
  
git init   
git add filename (eg : git add index.js)  
git commit -m "Your message"  
  
  
  
  
Create and Switch Branches:  
Creating and switching branches in Git is a fundamental part of version control. It allows you to work on new features or fixes without affecting the main codebase. Here's a detailed yet simple explanation with examples:

Creating a New Branch:

**Check Your Current Branch:**

git branch

The branch with an asterisk (\*) next to it is your current branch.

**Create a New Branch:**

To create a new branch, use the **git branch** command followed by the name you want to give to your new branch. For example, let's create a new branch called "anand":

git branch anand

**Verify Branch Creation:**

You can use the **git branch** command again to see a list of all branches. The new branch you created will be listed.

git branch

Switching to a Branch:

**Switch to the New Branch:**

To switch to the newly created branch, use the **git checkout** command followed by the branch name. For instance:

bashCopy code

git checkout anand

**Verify the Switch:**

To confirm that you've switched to the new branch, you can run **git branch** again. The branch with the asterisk (\*) should now be the one you switched to.

git branch   
  
# Example output:   
 main   
 \* anand

Now, you're on the "anand" branch, and any changes you make will be isolated to this branch. This allows you to work on your new feature or bug fix independently of the main branch, which is typically used for stable code.

To summarize, creating and switching branches in Git involves creating a new branch with **git branch branch\_name** and switching to it using **git checkout branch\_name**. This process is crucial for developing new features and keeping your work separate from the main codebase until it's ready to be merged.

4. Clone   
  
Cloning is creating a copy of a remote Git repository on your local machine. It's how you start working on a project.

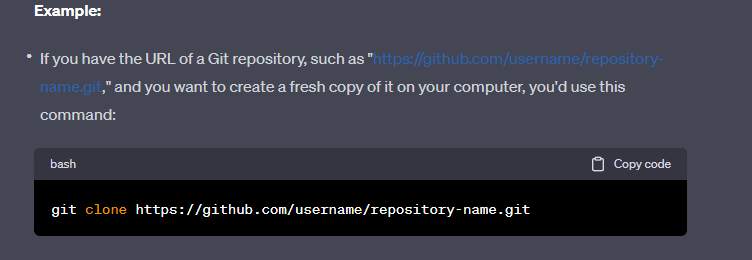
To clone a remote repository:

**git clone <repository\_url>**

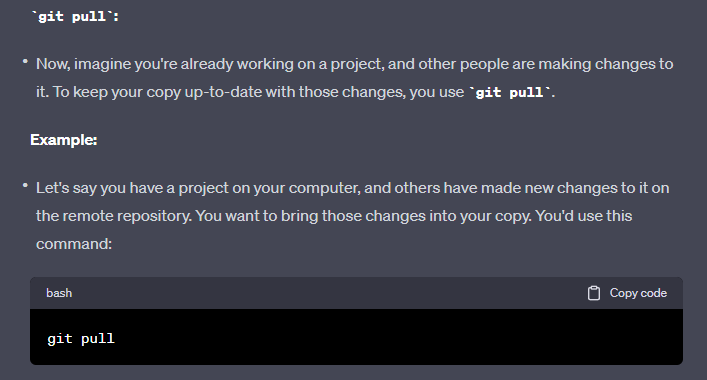
**5. Pull:**

Pulling is getting the latest changes from a remote repository (e.g., GitHub) and merging them into your local repository.

To pull changes:

**git pull**difference between clone and pull  
  
**git clone** is like making a fresh copy of a remote project to start working on it from scratch.  


**git pull** is like getting the latest updates or changes made by others in a project you're already working on to keep your copy up to date.

  
  
  
**6. Push:**

Pushing is sending your local commits to a remote repository. It's how you share your changes with others.

To push your changes to a remote repository:

**git push**

eg:

git init   
git add filename (eg : git add index.js)  
git commit -m "Your message"  
git push

GITHUB  
  
GitHub is a web-based platform that uses Git for version control. It provides a central hub for code collaboration, making it easy to work with others and host your Git repositories online.  
  
**3. Connect Your Local Repository to GitHub: (**local repo remote repo se connect kaise hoga**)**

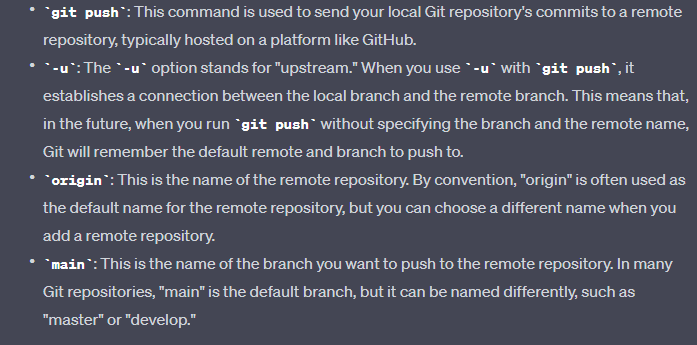
After creating a GitHub repository, you can connect your local Git repository to it.

Add the GitHub repository as a remote by running:

**git remote add origin <repository\_url> (**origin is a label or nickname you give to the GitHub repository, we can name anyting)

Push your local repository to GitHub:

**git push -u origin main  
  
  
  
  
Conclusion**git init   
git add filename (eg : git add index.js) or git add .  
git commit -m "Your message"  
git push  
  
git remote add origin <repo\_url>  
git push -u origin main (u upstream means we don’t need to write branch next time, we can simply write git push next time)

  
  
  
  
**.gitignore** is a configuration file in a Git repository that tells Git which files and directories to exclude from tracking. It's used to prevent certain files or patterns (like log files, compiled code, or sensitive data) from being included in your Git commits. Here's a simple explanation with an example:  
  
( hm .gitignore name se file create krte hai orr uss file me folder,file, ka name likhte hai jo ki push nhi krna hai ya jise track nhi krna hai )   
  
  
**Example:**

Let's say you have a project directory with the following files and folders:

* **index.html**
* **app.js**
* **\*.jpg (all jpg file)**
* **\*.png (all png file)**
* **styles.css**
* **build/** (a directory containing compiled files)
* **logs/** (a directory containing log files)
* **config.json** (a sensitive configuration file)