**GIT**

**GIT OVERVIEW:**

* It is the version control system for tracking computer files, basically it tracks the progress time of project where several people coordinating work among themselves. 🡪 It keep track of all the files such that we can even revert back to the file any specific function.
* It helps in the process of collaboration.

Commands:

1. Git config -- global.user name “demo\_gethub”
2. Git config --global.user email [shreyaagarwal1851@gmail.com](mailto:shreyaagarwal1851@gmail.com)

This two commands helps us to set the name and email configuration

1. Git config --list 🡪this helps us to check whether the configuration is set or not
2. Pwd 🡪 shows the address of the directory (by default we are placed in our home directory )
3. Mkdir git\_demo 🡪 mkdir is used to make the folder in the directory so that we can create our git depository
4. Cd git\_demo 🡪 set the directory

Initially the folder will be empty .

1. Git init 🡪 this means our depository has been initialised . whenever the git repository is created for the first time one default branch is created which is called as MASTER and it will be displayed.
2. Touch alpha.txt 🡪 creating a file for our folder
3. Notepad alpha.txt 🡪 with the help of this command one notepad will be opened then we can type anything we want on it and close it after saving it.
4. Git status 🡪 this is used to check the status of the file
5. Git add . 🡪 this is used to add the file to the folder
6. Git commit -m “<commit message>” “beta gamma” 🡪 this is used for commit the file that means Save the changes to a repository
7. Git log 🡪 show the log of the file ( it will show the commit id of the file with the commit message we gave).

When the changes are made on the file now and we want to share this files with other for collaborating for that we will push the depository to the git server.

So for that first we have to create a new repository in the git server then once creating it there will be a remote link present just copy and paste that link in the git bush.

Before pushing the repository to the server we have to first execute the **git remote -v** command

1. Git push -u origin master 🡪 this will push the depository to the git server

To check if the file is pushed to the server just open the repository in the server and then u can view the file in that.

To add the folder or push the folder into the server

First add the folder in the directory with the help of the add command git add .

then once that is done then just commit the folder with the help of the command commit git commit -m “add folder”

once it displayed how many file changed and how many inserted and how many got deleted at that time we can go for the commit option.

And to push the folder we will use the same command git push -u origin master

Branching is nothing but when ever we want to make the chance in the main code we can just create the branch and pull it out If it works out well then we can just merge that branch with the main file with the help of it the main file will be secure.

**Git clone**

* **Purpose**: Copies a remote repository to your local machine, allowing you to start working with a full version of the project.
* **Use case**: When you want to contribute to or use a project that already exists remotely (e.g., on GitHub, GitLab).

git clone <repository\_url>

**Git pull**

* **Purpose**: Fetches changes from a remote repository and automatically merges them with your current branch. Essentially, it's a combination of git fetch followed by git merge.
* **Use case**: When you want to get the latest changes from the remote repository to sync with your local repository.

git pull <remote> <branch> 🡪 git pull origin master

**Git fetch**

* **Purpose**: Downloads the latest changes from a remote repository but does **not** merge them into your working directory. You need to manually merge them later using git merge or git rebase.
* **Use case**: When you want to check what’s new in the remote repository without affecting your local working directory.

git fetch <remote> 🡪 git fetch origin

**Git merge**

* **Purpose**: Combines changes from one branch into the current branch. This can happen automatically if there are no conflicts or require manual resolution if conflicts arise.
* **Use case**: After fetching or pulling changes, or when you want to combine features from different branches into one.

git merge <branch\_name> 🡪 merging the changes from the another branch ‘

**Git rebase**

* **Purpose**: Applies changes from one branch on top of another. Unlike git merge, which combines branches and creates a merge commit, git rebase rewrites the history of your branch by "moving" it to a new base commit.
* **Use case**: When you want to clean up your commit history by making it linear (without merge commits) or when integrating changes from a main branch into your feature branch.

git rebase <branch\_name>

**Git revert**

* **Purpose**: Undoes the changes made by a specific commit by creating a new commit that "reverts" the changes. It does not modify history.
* **Use case**: When you want to undo a commit but keep a record of it in the project history. This is safer than git reset when working in a team.

git revert <commit\_hash>

**Git cherry-pick**

* **Purpose**: Allows you to apply a specific commit from one branch to another, without merging all the changes from that branch.
* **Use case**: When you want to take a commit from another branch (e.g., a bug fix or feature) and apply it to your current branch.

git cherry-pick <commit\_hash>

**Git reset**

* **Purpose**: Resets your current HEAD (commit history) to a specific state, which can modify your working directory and staging area depending on the reset type.
  + **soft**: Keeps changes staged.
  + **mixed**: Keeps changes but unstages them.
  + **hard**: Discards all changes in the working directory and staging area, resetting everything to the specified commit.
* **Use case**: When you want to undo commits or changes in your local working directory, either partially or completely.

git reset --soft <commit\_hash> # Soft reset (keeps changes staged)

git reset --mixed <commit\_hash> # Mixed reset (keeps changes unstaged)

git reset --hard <commit\_hash> # Hard reset (discard all changes)

**Resolve Conflicts :**

**Git Rebase:**

Conflict may occur when two people change the same line or at the same time

Git make it easy 🡪 instead of creating the zip files again and again and then sharing those file to one another and creating the new versions of it , and that to maintain can be hard

So the Git make it easy to use we can just create a repository in git hub which can be access by the team and the changes can be made easily from it and new version of the file is also not required to make anyone can just pull the file and then commit it later they can just