GitHub is a web-based platform that provides tools and services for version control and collaboration on software development projects. It allows individuals and teams of developers to work together on projects, manage code repositories, track changes to code, and collaborate on coding tasks.

Key features of GitHub include:

- `Version Control:` GitHub uses the Git version control system, which allows developers to track changes made to their code over time. This enables them to work on different versions of code simultaneously, merge changes from multiple contributors, and roll back to previous versions if needed.

- **`Code Hosting:`** GitHub provides a platform for hosting code repositories. Developers can create repositories to store and organize their code, making it easy to share and collaborate on projects.

- **`Collaboration:`** GitHub allows multiple developers to collaborate on the same project. They can contribute code, review each other's changes, and provide feedback through pull requests and code reviews.

- **`Issue Tracking:`** Developers can use GitHub's issue tracking system to report bugs, suggest enhancements, and track tasks related to the project. This helps in maintaining clear communication and organizing the work.

- **`Pull Requests:`** Pull requests (PRs) are a way for developers to propose changes to a project's codebase. They can submit their changes for review, and after the review process, the changes can be merged into the main codebase.

- **`Branching:`** GitHub supports branching, allowing developers to work on new features or bug fixes without affecting the main codebase. This promotes a clean development workflow and makes it easier to manage changes.

- **`Public and Private Repositories:`** GitHub offers both public and private repositories. Public repositories are visible to everyone, while private repositories are restricted to collaborators chosen by the repository owner.

- **`Community and Social Features:`** GitHub provides a platform for developers to connect with each other, follow projects, star repositories, and contribute to open-source projects.

- **`Documentation and Wikis:`** Repositories can include documentation and wikis to provide information about how the code works, how to contribute, and other relevant details.

GitHub has become a central hub for open-source software development and collaboration. It has contributed to the growth of the

Terms

- **Directory** —> Folder/Path to folder
- **Terminal** —> Interface for text commands
- **CLI** —> Command Line Interface
- **Repository/Repo** —> Your project folder

Git VS GitHub

`Git` is the tool or `"The Thing"` that tracks your project progression and helps you make changes to your repository.

`GitHub` is the website where you may host your repository; it can be **public**(other github users can see and clone your repo) or **private**(only you can see and clone the repo).

There are other popular repository hosting websites like **BitBucket**, **Jira**, **GitLab** and many more.But, **GitHub** is the most widely used repository hosting website.

Installing Git

You can download **git** on any operating system. Here is a guide that contains detailed steps for downloading **git** on any OS.

For linux simply open terminal and execute the following commands:

1. Update(Not necessary but good practice)

```
sudo apt-get update
```

2. Install Git

```
sudo apt-get install git
```

GitHub Login & SSH Authentication

`git` needs to know that you're the owner/collaborator of this repository before executing any **git** commands.

GitHub no longer accepts account password as your authentication method. You need to generate a **ssh key** against your **email id**.

You can create a ssh authentication from **https://github.com/settings/tokens**.

Git Clone Repository

We use git to **clone** our hosted repository to a local machine/our work environment. We can clone any **public** repository we want but can only clone **private** repositories; if we have **access** to cloning them.

Clone command:

```
git clone repo_ssh
```

The ".git" folder

Cloning a repo may seem to only create a folder. So, how does git track our work/changes in folder? Well there is a hidden folder in every git repository named `".git"` . It contains necessary info about the repo, every branch we create and meta-data about every branch and user credentials of local repository instance.

Most often we don't need to fiddle with this directory, so we don't need to focus on this as we're beginners.

Commands on Git(General)

1. Use `git status` to see changes(changes in existing files/new untracked files) in our repository.
2. Use `git add path_to_file` to add those files in git's tracking list.

3. Use `git commit` to commit any changes in your local project. **git commit** is paired with a commit message; the message serves as a verbal indication to what changes have been made and for what reason. So, the full command will be `git commit -m "description"`.

4. Use `git push` to push/publish changes to remote repository. **git push** requires a origin to push changes; if there is no brnaches then the deafult origin will be master. So, the command will be `git push origin master/branch_name`. If you have a branch or multiple branches that you're working on, then setting a **upstream** is a good idea to prevent accidental push into master branch. To set an upstream for a session use **-u** for the first git push, the command will look like `git push -u origin name_of_branch`.

Reverse a git add

Sometimes we add a file using `git add` mistakenly. We can reverse this only if we **haven't committed** the file using `git commit`. We can reverse adding file changes to git using `git restore --staged <file_name>`.

`.gitignore` file

If you choose to add a `.gitignore` file during repo creation or manually add a `.gitignore` file to your workspace, git will ignore changes to every file or folder you add into this `.gitignore` file. To properly add a file or folder into `.gitignore`, you need to add the relative path from `.git` folder to the file/folder you want to add to `.gitignore`.

Commands on Git(Branching)

1. Use `git checkout branch_name` to switch to an existing branch.

2. Use `git checkout -b branch_name` to crate a new branch.

3. Use `git checkout -d branch_name` to delete an exisiting branch.

4. Use `git diff current_branch_name` to compare changes with last commit on current branch.

5. Use `git diff other_branch_name` to compare current branch with other branch.

Commands on Git(Merging and Merge Conflict)

1. Use `git merge branch_name` to merge current branch with the mentioned(branch_name) branch.

Merge Conflict

A merge conflict in Git occurs when two different branches of code that are being merged have conflicting changes that can't be automatically resolved by Git. This usually happens when changes have been made to the same lines of code in both branches, or when one branch has deleted a file while the other branch has modified it.

Git is designed to handle most merging automatically, but in cases where it can't determine the correct way to combine changes, it will raise a merge conflict.

How to resolve merge conflict?

There are more than one way to reslove a merge conflict on git. Remember the master or the main branch you're having merge conflict with is more important than your work branch. So, you can **`git diff`** to visualize the merge conflict and remove the lines(the ones that are causing the merge conflict) from your branch.

Another way of doing this is using **`git reset HEAD~1`** —> this command will unstage your last commit(the one that caused the conflict) and you can then manually remove the conflict lines from your branch.

If you just want to ommit all the changes you've made to your branch(lines that are causing the conflict along with ones that are not), then you can simply use **`git log`** to see your commit log oin chronological order. You will then select the **hash id** of the commit stage you to want to swich back to then use **`git reset --hard commit_hash_id`**.

-----Additional Informations-----

Adding a new repo(from local to remote)

```
git remote add origin repo_ssh
```

Show connected Repos

```
git remote -v
```