Git and GitHub

Git is a free and open-source version control system.

Version control is the management of changes to documents, computer programs, large web sites, and other collection of information.

Some Terms

- **Directory:** Folder
- **Terminal or Command Line:** Interface for text commands
- **CLI:** Command Line Interface
- **cd:** Change Directory
- **Code Editor:** Word Processor for Writing Code
- **Repository:** Project, or the folder/place where your project is kept
- **GitHub:** A website to host your repositories online

Git Commands

- **clone:** bring a repository that is hosted somewhere like GitHub into a folder on your local machine
- **add:** track your files and changes in Git
- **commit:** save your files in Git
- **push:** upload git commits to a remote repo, like GitHub
- **pull:** download changes from remote repo to your local machine, the opposite of push

Using Git in Local Machine

- **git --version:** running this in the terminal will give you the current version of the Git

- Make a new repository on GitHub to be worked upon in local machine, or make a copy of a repository which already exists on GitHub

- To make a copy on local computer of a repository present on the GitHub, simply write

  git clone "SSH/HTML link copied from the GitHub"

  You will get the SSH/HTML link of the repository on the page of the repository

- **ls -la:** list all the file and folders in the directory, whether hidden or not
  The hidden ***.git*** folder keeps a record of all the changes that have been made in the repository

- **git status:** shows all the file that are updated, created or deleted but have not been saved in a commit yet

- We have to tell the git to add the file to its tracking system
  git add .: this tells the git to track all the files in the repository. We may use,
  git add "filename": to track the changes of a particular file. The full stop/period in the first command denotes all the files

- **git commit -m “heading” -m “description”:** this command will commit the files inside the branch (will explain later). The first -m defines the heading of the commit, and the second m defines the description of the commit. It can be seen similar to the subject line and actual content of a mail

- The commit isn’t live on the GitHub yet. To make it live on the GitHub, firstly we need to push our repository using the below command to make our local repository live on the GitHub
  git push: this will push the project to a repository

Pushing the Code on GitHub

In order to push them to GitHub under your account, you have to prove to GitHub that you are the owner of your account. For this, you have to connect your local machine to your GitHub account.

The way this is done is by using the SSH keys. Firstly, we will generate a SSH key in our local machine, and will share our public key with the GitHub so that the data can be shared securely between the GitHub and our local machine.

Follow the below steps to do so:

- Write the below in Git Bash CLI
  ssh-keygen -t rsa -b 4096 -C “email@email.com”

  The -t here defines the type of encryption. The -b defines the number of bytes. The -C defines the email address after it. Make sure to use the same email address used to make the GitHub account

- After the above step you may be asked to enter a file in which you want to save the key, enter any name e.g., “testkey”. You may enter a passphrase for your key or leave it blank. Press enter, and your key will be generated.

- You can view the two keys generated. ***testkey*** and ***testkey.pub.*** The key without the .pub extension is our private key, and need not to be shared with anybody. We will share our public key with the GitHub, so whenever we send our data by encrypting it with our private key, GitHub understands that the data is sent by us and can decrypt the data. The underlying concept behind this is a mathematical proof that a Public Key can only be derived from a Private Key

- Write ***cat testkey.pub*** and press Enter. Select the text that comes in front of you and copy that. If in case it do not works, open the public key, testkey.pub in a text editor and copy the content inside it.

- Go to **GitHub > Settings > SSH and GPG Keys > New SSH Key** and paste your copied key there. Give any title you want and click on Add SSH key

- Now we need to make sure that our local Git CLI knows about the key we generated.

  For this:

  1. Start the ssh-agent in the background by writing the below in cmd
     eval “$(ssh-agent -s)”

2. Modify the **~/.ssh/config** file either (using a text editor or vim in CLI), to automatically load keys into the ssh-agent and store passphrases in your keychain. After modification, the file should look like this
   Host *
   AddKeysToAgent yes
   IdentityFile ~/.ssh/id_rsa

3. Now add SSH private key to the ssh-agent and store your passphrase in the keychain.
   ssh-add -K ~/.ssh/id_rsa

Now to push the code to the remote repository we have to type the following

git push origin master

origin here, tells the location of the git repository, and master tells us the branch in which we want to push the code to. In some cases, this might give an error, in that case, try putting "main" in place of master in the above command, because in some repositories, the branch is named as main, and not master.

How to Push a Repository Originally Made in Local Machine

- Make a folder on local machine inside which all your project files will be kept and modified.

- Open Git Bash inside the folder and initialize Git in it by typing
  git init

- Now the Git is initialized in the folder. Work on your project, update delete create and commit the changes by above mentioned methods.

- Whenever you want to push this to the live GitHub server. Type:
  git push origin master
  Because we have not cloned it from an existing Git repository, it will throw an error.

- For this, make a repository on GitHub with the same name and copy the SSH/HTML link of the repository.

- In CLI, type:
  git remote add origin "link-that-you-copied"

- **git remote -v:** it will show any remote repository connected to current repository on local computer

- Now you can push like before by typing ***git push origin master***
  *Trick: You can type **git push -u origin master** to push the commits. The -u will define the upstream, i.e., it will define origin master as our default location to push the code. After executing the above command, we can push only by typing **git push***

Git Branching

Most of the times, while working on a project. There are multiple people and teams involved in development of a project. In this scenario, while working on a specific part of the project, a branch is created which is duplicate of the master branch or the main branch at first. But as we start working on the derived branch, let's call it feature branch for now. The feature branch will be different from the master branch as the changes as made to it.

Now as soon as someone switches back to the master branch, they will not be able to see the code changes in the feature branch.

This is useful when we want to implement new feature in our project, and the feature is not finished or it might affect the functionality of the other code, it is better to make a separate branch for it and work in it. When the work is done, we can merge our feature branch into the main branch.

There is another branch knows as the hotfix branch, we use to make copy of the code and fix it, case of any bug appears. After the commits if the bug is fixed, we merge the branch into the main branch.

> **git branch:** will tell you about all the existing branches. The star written in front of the branch name tells in which branch we currently are
>
> **git checkout – b "name of the branch":** it will make a new branch with the provided name and switch to that branch
>
> **git merge "branch-name":** will merge the branches
>
> **git diff "branch-name":** compares two versions of the code and current and given branch, and tells the differences in the two branches. The lines with + written in front of them indicates the changes that have been added and – indicates the changes, that have been removed. Those lines with none of the both indicates that no change have been made.

Pull Requests

Most commonly we will see is that, we will push the changes into GitHub and generate a Pull Request (PR)

git push origin "branch name" (Changes are pushed)

A pull request is a request to have your code pulled into another branch. After we make a PR, anyone can review our code, add comments on it, ask to make updates. After we make PR, we can update the code just by making additional commits. Once the PR is merged, we'll generally delete the source branch and switch back to the master branch and when we want to make other changes in the code, we will make another new branch and start the process over.

We can create a pull request either by using the GitHub interface or by CLI. After the pull request we can merge the two branches either by using GitHub or by using CLI (we will see further).

Merge Conflicts

If we try to merge the feature branch with the master branch, and the master branch already have something written which is modified in the feature branch, then merge conflicts happen.

We will check the differences in two branches by *git diff master* and we will merge the two branches by typing:

git merge master

If there is any merge conflict it will be thrown as an error. To fix the conflict checkout your code editor and choose to keep the current change, incoming change, or accept both changes. After the conflict is resolved, both the branches can be merged together.

After fixing the conflict, we need to commit the changes we made by using the *git commit* command.

Undoing in Git

If we, by mistake staged a file for commit by git add we can undo it by using git reset command.

We can use ***git reset*** with no arguments, as well as provide filename along with the ***git reset*** command e.g., ***git reset README.md***

In case of committed changes, ***git reset HEAD~1*** will reset changes (unstage and uncommit) to the situation which was 1 commit before.

We can use ***git log*** and get a hash code for every commit we have done. We can copy the hashcode of the commit where we want to go to and type, ***git reset hashcode***

After using this method for reset, the changes will be uncommitted and unstaged. But to get rid of these changes permanently from the file in which we edited them, we can hard reset using the following:

git reset --hard hashcode

Forking in Git

To work in someone else's project, we can fork his/her project as our own copy of their project. Now we have the full authority to make the changes in the code and after we have made the changes, we can generate a pull request for them to accept the changes, that we have made and we want to be accepted into their code.

References

Faraday, Gwen. "Git and GitHub for Beginners - Crash Course." *YouTube,* uploaded by freeCodeCamp.org, 28 May 2020, https://youtu.be/RGOj5yH7evk