GIT & GITHUB

Git and GitHub are widely used in the industry for version control, and collaborative software development. A version control system lets you track your changes to the source code. Git is a versatile version control system used for tracking changes in code and collaborating with others on software projects. This makes it easy for you to recover older versions of your document if you make a mistake.

Git also supports branching strategies such as feature branching to organize and manage development. You can use Git without a web interface using your command lined interface but GitHub is one of the most popular web hosted services for Git repositories. Others include Git Lab, Bitbucket, and Beanstock.

Basic terms in github

- The **secure shell or SSH** protocol is a method for secure remote login from one computer to another.
- A **repository** contains your project folders that are set up for version control. It is a data structure for storing documents including application source code. A repository can track and maintain version-control
- A **fork** is a copy of a repository.
- A **pull request** is how you request that someone review and approve your changes before they become final.
- A **working directory** contains the files and subdirectories on your computer that are associated with a Git repository.
- **Commit** is a snapshot of the project's current state at a specific point in time along with a description of the changes made.
- A **branch** is a separate line of development that allows you to work on features or fixes independently.
- **Merging** combines changes from one branch into another, typically merging a feature branch into the main branch.
- **Cloning** creates a local copy of a remote Git repository on your computer.
- An **organization** is a collection of user accounts that owns repositories. Organizations have one or more owners, who have administrative privileges for the organization.

Different tabs in a repository

When you create your repository, you'll notice that it has a number of tabs, and is opened to the Code tab.

- **Code** – this is where all the source files reside.
- **Issues** – you can track and plan with tools such as "Issues" that lists all open items against your project base.

- **Pull requests** define changes that are committed and ready for review before being merged into the main branch.
- **Projects** – all the tools for managing, sorting, planning, etc. your various projects.
- **Wiki, Security, and Insights** – often left for more advanced users, these tools provide a communication base to the external user community.
- **Settings** – GitHub allows for a lot of personalization, including changing the name of your repository and controlling access.

Creating a new repository

- Click +
- click New Repository.
- give repository a name, optionally, add a description of your repository;
- choose the repository visibility - public or private
- choose the option to Initialize this repository with readme file.
- Then click Create Repository.

Create a new file:

- Click Add File
- click Create New file
- provide the file name.
- Next, add a comment that describes your code, then add the code.
- Once finished, commit the change to the repository

Branches with github

All files in GitHub are stored on a branch. The main branch is definitive, it stores the deployable version of your code. The main branch is created by default, When you plan to change things, you create a new branch. The new branch starts as an exact copy of the original branch. As you make changes, the branch you created holds the changed code.

To create a new branch:

- click the dropdown branch Main
- add new branch name into a new branch text
- select "Create Branch".

Pull Requests

Pull is used to initiate the merging of branches in a way to capture changes. A pull request makes the proposed committed changes available for others to review and use. A pull can follow any commits even if the code is unfinished. A pull requires a user to approve the changes. This can be the author of the change or can be assigned within the team. Note that

GitHub automatically makes a pull request on your behalf if you make a change on a branch that you do not own.

To open a pull request:

- click "Pull requests"
- select "New pull request".
- Select the new branch from the compare box
- Confirm that the changes are what you want to assess.
- Add a title and description to the request.
- Click "Create pull request."

A pull command is issued, the code is reviewed and approved. The approved code is merged back into the main code.

To merge a committed code change into your main code:

- click "Merge pull request,"
- click "Confirm merge".

When all changes for a branch are complete, that branch is considered obsolete and should be deleted.

Steps in a git workflow

When you join an existing project:>

- The first step is cloning a repository that your team has hosted on GitHub
- Then create a branch from the main repository and work on the branch.
- Add updated files to the staging area and commit to the branch.
- Push commits to the remote repository.
- Create a pull request to merge the branch into the main branch, which will be reviewed and approved by the maintainer.

When you start a new project:>

- Initialize a local git repository
- Then select the files you want Git to track, move them to a staging area and perform an initial commit.
- Create a blank remote repository and establish a link with your local repository.
- Push the changes so other developers can clone this remote repository and follow the regular workflow for updating project files.

Git commands

Git commands are used for various purposes such as tracking and saving changes and sharing your changes with others.

Command line commands:>

mkdir : creates a new directory

syntax: mkdir

cd: used to navigate into a specific directory

syntax: cd <directory_name>

Git commands:>

git init: creates a new repository in the directory. It sets up the necessary files and data structure for git to start managing your project's version control.

Syntax: git init

git add: It adds changes to the staging area.

Syntax:

git add <filename.txt> (to add a specific file)

git add . (to add all the files that are new or changed in the current directory)

git add -A (to add all changes in the entire working tree, from the root of the repository, regardless of where you are in the directory structure)

git commit-m: takes your stage snapshot of changes and saves them to the project with a descriptive message

syntax: git commit -m “<your message>”

git log : it enables you to browse previous changes made to a project.

git branch: It lists, creates, or deletes branches in a repository. To delete the branch, first check out the branch using git checkout and then run the command to delete the branch locally.

Syntax:

git branch (to list branches)

git branch <new-branch> (to create a new branch)

git branch -d <branch-name> (to delete a branch)

git checkout: to switch between the existing branches.

Syntax: git checkout <branch_name>

git status: provides information about the state of your files in relation to the repository**.**

Syntax: git status

git merge: to update the branches with the main branch

syntax: git merge <childVBanch_name> (first navigate to the main branch using git checkout command and then merge the child branch)

git reset

It resets changes in the working directory. When used with –hard HEAD, this command discards all changes made to the working directory and staging area and resets the repository to the last commit (HEAD).

Syntax:

git reset

git reset –hard HEAD

git clone

It copies a repository from a remote source to your local machine. This will create a copy of the repository in your current working directory.

Syntax: git clone <repository URL>

git pull: It fetches changes from a remote repository and merges them into your local branch. First, switch to the branch that you want to merge changes into by running the git checkout command. Then, run the git pull command, which will fetch the changes from the main branch of the origin remote repository and merge them into your current branch.

Syntax: git pull origin main

git push: It uploads local repository content to a remote repository. Make sure you are on the branch that you want to push by running the git checkout command first, then push the branch to the remote repository.

Syntax: git push origin <branch-name>

git version: It displays the current Git version installed on your system.

Syntax: git version

git diff: It shows changes between commits, commit and working tree, etc. It also compares the branches.

Syntax:

git diff (shows the difference between the working directory and the last commit)

git diff HEAD~1 HEAD (shows difference between the last and second-last commits)

git diff <branch-1> <branch-2> (compares the specified branches)

git revert: It reverts a commit by applying a new commit. This will create a new commit that undoes the changes made by the last commit.

Syntax: git revert HEAD

git config –global user.email <Your GitHub Email>: It sets a global email configuration for Git. This needs to be executed before doing a commit to authenticate the user's email ID.

Syntax: git config –global user.email <Your GitHub Email>

git config –global user.name <Your GitHub Username>: It sets a global username configuration for Git. This needs to be executed before doing a commit to authenticate users' username.

Syntax: git config –global user.name <Your GitHub Username>

git remote: It lists the names of all remote repositories associated with your local repository.

Syntax: git remote

git remote -v: It lists all remote repositories that your local Git repository is connected to, along with the URLs associated with those remote repositories.

Syntax: git remote -v

git remote add origin <URL>: It adds a remote repository named "origin" with the specified URL.

Syntax: git remote add origin <URL>

git remote rename: The git remote rename command is followed by the name of the remote repository (origin) you want to rename and the new name (upstream) you want to give it. This will rename the "origin" remote repository to "upstream."

Syntax: git remote rename origin upstream

git remote rm : It removes a remote repository with the specified name.

Syntax: git remote rm origin

git format-patch: It generates patches for email submission. These patches can be used for submitting changes via email or for sharing them with others.

Syntax: git format-patch HEAD~3 (creates patches for the last three commits)

git request-pull: It generates a summary of pending changes for an email request. It helps communicate the changes made in a branch or fork to the upstream repository maintainer.

Syntax: git request-pull origin/main <myfork or branch_name>

git send-email: It sends a collection of patches as emails. It allows you to send multiple patch files to recipients via email. Please make sure to set the email address and name using the git config command so that the email client knows the sender's information when sending the emails.

Syntax: git send-email *.patch

git am: It applies patches to the repository. It takes a patch file as input and applies the changes specified in the patch file to the repository.

Syntax: git am <patchfile.patch>

git daemon: It exposes repositories via the Git:// protocol. The Git protocol is a lightweight protocol designed for efficient communication between Git clients and servers.

Syntax: git daemon –base-path=/path/to/repositories

git instaweb: It instantly launches a web server to browse repositories. It provides a simplified way to view repository contents through a web interface without the need for configuring a full web server.

Syntax: git instaweb –httpd=webrick

git rerere: It reuses recorded resolution of previously resolved merge conflicts. Please note that rerere.enabled configuration option needs to be set to "true" (git config –global rerere.enabled true) for git rerere to work. Additionally, note that git rerere only applies to conflicts that have been resolved using the same branch and commit.

Syntax: git rerere

Cloning

Cloning generally refers to creating a copy of a repository on your local machine. Cloned copies can be kept in sync between the two locations. Forking allows you to modify or extend a project without affecting the original project.

To clone a GitHub repository:

- Navigate to the repository that you want to clone
- Under the repository name, click Code.
- In the Clone with HTTPS section, copy the URL.
- On your local machine, open a "Terminal" window and change to the directory where you want the clone to be copied.
- Type "git clone" followed by pasting the URL that you copied
- When you have made your changes and are ready to sync your code back to GitHub. First, you must run the "git add <files>" command. This moves the changed files into a staging area
- Next run "git commit –m <message>" and this will commit changes in the staging area.
- When you are ready to move your changes fully into the GitHub repository. Use the "git push" command. This will push all the committed changes into the repository.

Remote repositories are repositories that are stored elsewhere – on the internet, on your network, even on your local computer.

Forking

Forking is used to take a copy of a GitHub repository and use it as the base for a new project. You can also use forking to submit back changes into the original repository.

Steps in forking a project:

- Navigate to the repository that you want to fork
- Click Fork

Syncing a Fork of a project:

To keep a fork in sync with the original work from a local clone:

- Create a local clone of the project
- Configure git to sync the fork
  - Open a terminal and change to the directory containing the clone
  - To access the remote repository, type git remote -v
  - Type git remote add upstream <clone repository>
  - To see the change, type git remote -v

Commands for managing forks:

- To grab upstream branches
  - Git fetch upstream
- To merge changes into the master branch
  - Git merge upstream/master

The repo from which you create the fork is referred to as the original upstream repository. Once you fork the original upstream, the forked copy of the repo becomes the origin. If you want to contribute your changes back to the original upstream that you do not have right access ,In that case, you can submit a pull request for your proposed changes .The maintainers of the upstream project can review the changes in the pull request, provide feedback, and merge accordingly unless there are any conflicts to be resolved.

Forking workflow

- First, you create a fork of an upstream project which then becomes the origin.
- The developer can create clones on the local machines, make changes, and then use git push to push the updated main branch back to the origin by creating pull requests.
- The upstream project maintainers review the changes and then merge if there are no conflicts

You can create an identical copy of the remote repo using the git clone operation.

Clone workflow

- Cloning comes into the picture once you have forked your repo and it has now become the origin.
- You then use the origin to create an identical copy of the remote repo using the git clone command
- Then, the new developer creates a new branch, makes changes, and saves them using add and commit operations.
- After that, they push the new branch to the origin to get their changes reviewed.
- A reviewer or maintainer uses the git fetch or git pull command to get the latest copy of the repo and the git diff command to help identify and compare the changes in the branch.
- Once reviewed, they can use git checkout and merge the branch to the main.
- Lastly, anyone with maintainer access can create a pull request to the original upstream to initiate the changes in the original repo.

Github Copilot

GitHub Copilot is an AI power tool that assists developers in writing code. It is designed to work as a plug-in for various code editors, including Visual Studio code, and provides contextual code suggestions.

Features:

- code auto-completion: GitHub Copilot suggests whole lines or blocks of code based on the context of the code you are currently working on.
- GitHub Copilot offers real-time suggestions as you code, making the development process smoother.
- GitHub Copilot generates suggestions based on the code context, including function names, variable names, and even algorithm implementation.

It supports a wide range of programming languages and is designed to provide you assistance across various popular languages such as Python, JavaScript, Java, C++, and many more. GitHub Copilot integrates with various popular code editors and integrated development environments or IDs such as Pycharm.

Typical workflow when using GitHub Copilot

- The first step is installing the GitHub Copilot extension in your Visual Studio code environment.
- After you install the extension, you will activate the extension within Visual Studio code. This step typically involves signing in with your GitHub account and allowing the necessary permissions.
- You will then open an existing coding project or create a new one. GitHub Copilot will work within the context of the programming language and framework you are using. You will begin writing code as you normally would. GitHub Copilot will start to provide suggestions and autocompletion based on the context of your code.