Beginner’s Guide to Git & GitHub

1. What is Git and GitHub?

- Git
    - Git is a version control system (VCS).
    - It helps developers track changes in code, collaborate with others, and revert to previous versions when needed.
    - Works locally on your computer.
    - Think of it as a time machine for your code.
- GitHub
    - GitHub is a cloud-based hosting service for Git repositories.
    - It allows developers to store code online, collaborate with others, and contribute to projects.
    - Provides features like issues, pull requests, forking, and CI/CD integration.
    - Git = Tool, GitHub = Website that hosts your Git repos.

👉 Analogy: Git is like MS Word’s “track changes” feature for code, and GitHub is like Google Drive where you share your document with others.

---

2. git init

- Command to initialize a Git repository in a project folder.
- It creates a hidden folder .git/ which stores version history.

```
git init
```

✔ Example:
If you are in a folder my-project/ and run git init, now Git will start tracking changes in that folder.

---

3. git add

- Used to stage files before committing.
- Two ways to use it:

```
git add filename.txt    # Adds specific file
git add .               # Adds all files in the folder
```

✔ Example:
If you modified index.html and style.css:

```
git add index.html
git add style.css
```

OR simply:

```
git add .
```

---

4. git commit -m "message"

- Commits the staged files to the local repository.
- the -m flag stands for "message"
- -m "message" gives a short description of what you did.

```
git commit -m "Added homepage design"
```

✔ Example:
If you fixed a bug, you might write:

```
git commit -m "Fixed login button bug"
```

---

5. git remote

- Connects your local repository to a remote repository (like GitHub).

```
git remote add origin https://github.com/username/repo.git
```

- origin is just a nickname for the remote URL.

Check if remote is set:

```
git remote -v
```

the -v flag stands for "verbose"

---

6. git branch

- Used to create or view branches.
- A branch is like a separate copy of your code where you can make changes safely.

```
git branch          # List all branches
git branch feature-1  # Create new branch
git checkout feature-1 # Switch to branch
```

✔ Example:

- main (default branch) = production code
- feature-1 = experimental code

---

7. git push

- Sends local commits to GitHub (remote repository).

```
git push origin main
```

✔ Example:
If you created a new branch:

```
git push origin feature-1
```

---

8. git clone

- Used to download a project from GitHub to your computer.

```
git clone https://github.com/username/repo.git
```

✔ Example:
Cloning React's official repo:

```
git clone https://github.com/facebook/react.git
```

---

9. git fork

- Forking = Creating your own copy of someone else's repository on GitHub.
- Useful when you want to contribute to open-source projects.
- Done directly on GitHub's website (not a Git command).

👉 Example:

- Fork facebook/react repo.
- Now you have your own copy under your GitHub account.

---

10. Pull Request (PR Raise)

- After forking and making changes, you propose your changes to the original repo using a Pull Request (PR).
- Steps:

1. Fork repo → Clone → Make changes → Commit → Push.
2. Go to GitHub and click New Pull Request.
3. Maintainers review your code and decide whether to merge.

👉 Example:
You add a new feature → raise PR → project owner reviews → merges.

---

11. git merge

- Combines changes from one branch into another.

git checkout main

```
git merge feature-1
```

✔ Example:

- You worked on feature-1 branch.
- After finishing, merge it into main.

Full Process to Merge a Feature Branch and Update Remote (origin):

```
# 1. Switch to main branch
git checkout main

# 2. Make sure main is up-to-date with origin
git pull origin main

# 3. Merge feature-1 into main
git merge feature-1

# 4. Push the updated main branch to remote
git push origin main
```

---

12. Git Merge Conflict

- Happens when two branches modify the same part of a file differently.
- Git doesn't know which version to keep.

Example conflict in index.html:

```
<<<<<<< HEAD
<h1>Hello from Main</h1>
```

```
=======
<h1>Hello from Feature-1</h1>
>>>>>>> feature-1
```

You must manually edit the file to resolve it, then:

```
git add index.html
git commit -m "Resolved merge conflict"
```

Explanation:
<<<<<<< HEAD: This is the version from the current branch (main).
=======: Divider between the two conflicting versions.
>>>>>>> feature-1: This is the version from the branch you're merging (feature-1).

How to Resolve a Merge Conflict:

1. Open the conflicted file (e.g., index.html).

2. Manually edit the file to remove conflict markers (<<<<<<<, =======, >>>>>>>) and keep the correct version or merge both changes.

   Example resolved version:

   <h1>Hello from both Main and Feature-1</h1>

3. Stage the resolved file:

   ```
   git add index.html
   ```

4. Commit the resolution:

   ```
   git commit -m "Resolved merge conflict in index.html"
   ```

💡 Tips:

- Tools like git status will show you which files have conflicts.
- You can use git merge --abort to cancel the merge if you don't want to continue.

After resolving conflicts:

Continue your workflow (e.g., push to remote, create PR, etc.):

git push origin main

✅ Summary Flow (for beginners)

1. git init → Start Git in a folder
2. git add → Stage changes
3. git commit → Save changes locally
4. git remote → Connect with GitHub
5. git push → Send changes online
6. git branch → Work in parallel safely
7. git clone → Copy others' projects
8. git fork → Make your own copy on GitHub
9. Pull Request → Suggest changes to others' projects
10. git merge → Combine work

11. Merge conflict → Fix disagreements in code