**🔧** **What is Git?**

**Git is a version control system — it helps you:**

* Track changes in your code
* Go back to earlier versions if something breaks
* Work with others on the same project without conflicts

Think of it like a save system for your code — you can create “checkpoints” (called *commits*) and switch between them.

**🔸 Example:**

* You’re writing a program.
* You make a change and save it using Git.
* If the code breaks later, you can go back to a working version.

**🌐 What is GitHub?**

GitHub is a website where you can:

* Store your Git projects online
* Share your code with others
* Work together with a team
* Showcase your projects to the world

GitHub uses Git behind the scenes, but adds online tools like:

* Pull Requests (for code review)
* Issues (for bug tracking)
* Actions (for automation and CI/CD)

**🔸 Example:**

* You build a website using Git.
* You upload (push) it to GitHub.
* Your friend clones it, adds features, and sends you a pull request.

Alright — let’s go step-by-step with a **very simple project** that you can create locally and push to GitHub.

We’ll use **Git Bash** + **VS Code** so you get both Git and GitHub practice.

**1️⃣ Create a new folder for the project**

Open **Git Bash** and run:

mkdir my-first-git-project

cd my-first-git-project

**2️⃣ Create a simple file**

echo "Hello, Git and GitHub!" > index.txt

Or in VS Code:

1. Open the folder:
2. Create a file named index.txt
3. Type:

Hello, Git and GitHub!

1. Save it.

**3️⃣ Initialize Git**

git init

git add index.txt

git commit -m "chore: initial commit"

4️⃣ Create a repository on GitHub

Go to https://github.com/new

Repository name: my-first-git-project

Keep it Public (for learning).

Don’t add README, .gitignore, or license (we already have files locally).

Click Create repository.

5️⃣ Connect your local project to GitHub

GitHub will give you a link — copy it.

Example:

git remote add origin https://github.com/your-username/my-first-git-project.git

git branch -M main

git push -u origin main

**6️⃣ Update the project**

Make a change:

echo "This is my second update." >> index.txt

Commit and push:

git add index.txt

git commit -m "feat: add second update to index.txt"

git push -u origin main

**7️⃣ Check on GitHub**

* Refresh your repo page — you’ll see the update instantly.

✅ **Now you’ve:**

* Created a project locally
* Initialized Git
* Created a GitHub repo
* Pushed your project
* Made and pushed updates

**Second simple Git project — step-by-step (very simple)**

Nice — I’ll keep this extra simple. Do each step one at a time. Replace YOUR-USERNAME with your GitHub username when needed.

0) Prereqs (one-time)

Make sure Git is configured:

git --version

git config --global user.name "Your Name"

git config --global user.email "you@example.com"

If those run OK, you’re ready.

1) Create the project folder and a simple file

mkdir second-project

cd second-project

echo "Hello, Git and GitHub! - project 2" > index.txt

echo "# second-project" > README.md

You now have a folder with index.txt and README.md.

2) Start git and make the first commit

git init

git add .

git commit -m "Initial commit: add index.txt and README"

git branch -M main

\*\*\*\*\* git init starts tracking; git add . stages files; git commit saves them. branch -M main ensures the branch is named main \*\*\*\*\*\*\*

**3) Create the remote repo on GitHub (manual, very simple)**

1. Go to **github.com** → **New** repository.
2. Name it second-project.
3. **Do not** initialize with README (you already have one).
4. Create repository and copy the repository **HTTPS** URL shown (it looks like https://github.com/YOUR-USERNAME/second-project.git).

**4) Link remote and push first branch**

git remote add origin https://github.com/ YOUR-USERNAME /second-project.git

git branch -M main

git push -u origin main

**5) Create a feature branch (where you make a change)**

git checkout -b feature/add-note

# make a small change:

echo "This is a practice change on the feature branch." >> index.txt

git add index.txt

git commit -m "feat: add practice change on feature branch"

git push -u origin feature/add-note

You created and switched to feature/add-note, edited index.txt, then committed the change and pushed to created branch github.

\*\*\* Now in Git hub branches verify the files you will see difference \*\*\*\*\*

**6) Open a Pull Request (PR) on GitHub (web steps)**

1. Go to your repo page on GitHub.
2. You will usually see a **Compare & pull request** button for your pushed branch — click it.  
   Or: Go to **Pull requests → New pull request**, set **base: main**, **compare: feature/add-note**.
3. Give the PR a simple title like Add practice note and click **Create pull request**.

**7) Review & Merge the PR (web)**

* On the PR page click **Merge pull request** → **Confirm merge**.
* Optionally click **Delete branch** on GitHub to remove feature/add-note there.

**8) Update your local repo and clean up**

git checkout main

git pull origin main

git branch -d feature/add-note

git branch -d deletes the local feature branch after it’s merged.

**9) Quick checks (use these if confused)**

git status

git log --oneline --graph --all

git branch -av

These show what branch you’re on, commit history, and remote branches.

Perfect 👍 — let’s do a **third Git project**.

We’ll make it *slightly* different so you learn new Git steps while still keeping it simple.

* Have **two files**
* Make **two feature branches**
* Merge them **one by one** into main

**Third Project – Step-by-Step**

**1) Make project folder & files**

mkdir third-project

cd third-project

echo "Third project starting..." > main.txt

echo "# third-project" > README.md

**2) Initialize Git & commit first version**

git init

git add .

git commit -m "Initial commit: add main.txt and README"

git branch -M main

**3) First feature branch — update main.txt**

git checkout -b feature/update-main

echo "This is an update from first branch." >> main.txt

git add main.txt

git commit -m "feat: add first branch update to main.txt"

**4) Second feature branch — add notes.txt**

Go back to main first:

git checkout main

Now create the second branch:

git checkout -b feature/add-notes

echo "This is a new notes file." > notes.txt

git add notes.txt

git commit -m "feat: add notes.txt"

**5) Create repo on GitHub**

* On GitHub, create new repo named third-project (**no README**).
* Copy the HTTPS URL (e.g., https://github.com/YOUR-USERNAME/third-project.git).

**6) Push all branches to GitHub**

git remote add origin https://github.com/ vinohts /third-project.git

git push -u origin main

git push -u origin feature/update-main

git push -u origin feature/add-notes

**7) Make Pull Requests (PR) on GitHub**

* **PR 1:** Merge feature/update-main → main
* **PR 2:** Merge feature/add-notes → main

*(You can merge them in any order — just make sure to update local main after each merge.)*

**8) Update your local main branch**

git checkout main

git pull origin main

**9) Delete merged branches**

git branch -d feature/update-main

git branch -d feature/add-notes

git push origin --delete feature/update-main

git push origin --delete feature/add-notes

✅ **You’ve now worked with:**

* Multiple feature branches
* Multiple pull requests
* Merging changes step-by-step

Alright 🚀 — let’s make a **real-world Git challenge** with a **merge conflict** so you can learn how to solve it.  
We’ll keep it **simple** so it’s not scary, but it’ll mimic what actually happens in teams.

**🛠 Project Goal**

You and your “teammate” (both you in this case) will **edit the same line** in the same file on two different branches → Git will not know which change to keep → you’ll fix it.

**1) Create the folder & start the project**

mkdir git-conflict-project

cd git-conflict-project

echo "Hello, Git World!" > conflict.txt

echo "# git-conflict-project" > README.md

**2) Init git and commit the first version**

git init

git add .

git commit -m "Initial commit: add conflict.txt and README"

git branch -M main

**3) Create first branch and edit the file**

git checkout -b feature/update-message

echo "This is change from Branch 1." > conflict.txt

git add conflict.txt

git commit -m "feat: update message in conflict.txt from Branch 1"

**4) Switch back to main and make another change**

git checkout main

echo "This is change from Main branch." > conflict.txt

git add conflict.txt

git commit -m "feat: update message in conflict.txt from Main branch"

**5) Merge Branch 1 into Main (conflict happens)**

git merge feature/update-message

You’ll see something like:

CONFLICT (content): Merge conflict in conflict.txt

**6) Open the file and fix the conflict**

Open conflict.txt in VS Code or any editor — you’ll see:

<<<<<<< HEAD

This is change from Main branch.

=======

This is change from Branch 1.

>>>>>>> feature/update-message

* **HEAD** = your main branch version
* The other = feature branch version

Pick what you want, or combine them:

This is change from Main branch AND Branch 1.

Save the file.

**7) Mark as resolved and commit**

git add conflict.txt

git commit -m "Merge branch 'feature/update-message' into main - resolved conflict"

**8) Push to GitHub**

1. Create a GitHub repo called git-conflict-project (**no README**).
2. Add remote & push:

git remote add origin https://github.com/YOUR-USERNAME/git-conflict-project.git

git push -u origin main

✅ **You’ve just handled a merge conflict** — one of the most important Git skills in real projects.

If you want, I can now make the **next step**:

* **Two people editing different parts of a file (no conflict)** and
* **Both editing the same line (conflict)**  
  ...so you learn when conflicts happen and when they don’t.

Alright — let’s make **two scenarios** in one mini-project so you’ll **see clearly** when Git makes a conflict and when it doesn’t.

We’ll call this **conflict-demo**.

**Scenario 1 — No Conflict**

Two branches change **different parts** of the same file → Git merges without a problem.

**Scenario 2 — Conflict**

Two branches change **the same line** → Git asks you to fix it.

**🛠 Step-by-Step**

**1) Create folder & file**

mkdir conflict-demo

cd conflict-demo

cat <<EOF > story.txt

Once upon a time,

there was a programmer who loved Git.

EOF

echo "# conflict-demo" > README.md

**2) Init Git & commit first version**

git init

git add .

git commit -m "Initial commit: add story.txt and README"

git branch -M main

**Scenario 1 — No Conflict**

**3) First branch — change line 1**

git checkout -b feature/change-first-line

sed -i '1s/.\*/Once upon a time, in a faraway land,/' story.txt

git add story.txt

git commit -m "feat: update first line of story"

**4) Main branch — change line 2**

git checkout main

sed -i '2s/.\*/there was a coder who loved Git deeply./' story.txt

git add story.txt

git commit -m "feat: update second line of story"

**5) Merge (no conflict)**

git merge feature/change-first-line

✅ Git will merge automatically because you changed **different lines**.

**Scenario 2 — Conflict**

**6) Reset file to base version**

echo "Once upon a time," > story.txt

echo "there was a programmer who loved Git." >> story.txt

git add story.txt

git commit -m "reset: restore original story"

**7) Branch 1 — edit line 2**

git checkout -b feature/branch1

sed -i '2s/.\*/there was a developer who adored Git./' story.txt

git add story.txt

git commit -m "branch1: update second line"

**8) Main branch — edit same line**

git checkout main

sed -i '2s/.\*/there was a coder who mastered Git./' story.txt

git add story.txt

git commit -m "main: update second line differently"

**9) Merge (conflict happens)**

git merge feature/branch1

**You’ll see:**

CONFLICT (content): Merge conflict in story.txt

**10) Open story.txt and resolve**

It will look like:

markdown

Once upon a time,

<<<<<<< HEAD

there was a coder who mastered Git.

=======

there was a developer who adored Git.

>>>>>>> feature/branch1

**Decide what to keep or combine:**

Once upon a time,

there was a coder and developer who mastered and adored Git.

Save the file.

**11) Mark as resolved**

git add story.txt

git commit -m "merge: resolve conflict between main and branch1"

**12) Push to GitHub**

1. Create repo **conflict-demo** on GitHub (no README).
2. Add remote & push:

git remote add origin https://github.com/YOUR-USERNAME/conflict-demo.git

git push -u origin main

💡 **Key Lesson:**

* If branches edit **different lines** → no conflict.
* If branches edit **the same line** → conflict.