2-Day Git & GitHub Training Program

Goal:
By the end of this training, trainees will confidently:

- Use **Git** for version control in real-world projects.
- Collaborate using **GitHub** (repos, pull requests, issues, reviews).
- Work with branches, merges, pull requests, and resolve conflicts.
- Follow commit message conventions, branch naming standards, and PR best practices.

Day 1 – Core Git & GitHub Foundations

What to Learn (Why it Matters)

- **Git Basics** → install, config, global username/email (foundation for every repo).
- **Repositories** → `git init`, `git clone` (start any project).
- **Staging & Committing** → `git add`, `git commit` (track changes).
- **Status & Logs** → `git status`, `git log` (know what changed).
- **Branching & Checkout** → `git branch`, `git checkout -b` (parallel development).
- **Merging Basics** → `git merge` (bring work together).
- **Remote Basics (GitHub)** →
    - `git remote add`, `git push`, `git pull` (team collaboration).
    - Creating GitHub repo, cloning & pushing code.

- **GitHub Basics** →
    - Fork vs clone.
    - Creating issues.
    - Using GitHub web editor.

Task (Practical)

1. Initialize a new repo & push it to GitHub.
2. Create a feature branch `feature/home-page`.
3. Make changes in HTML + CSS files, commit, and push.
4. Open a Pull Request (PR) on GitHub.
5. Merge feature branch into `main` via GitHub.

Deliverable: A GitHub repo with a clean branch history + successful PR merge.

Day 2 – Advanced Git + GitHub Collaboration

What to Learn (Why it Matters)

- **Undo & Reset** → `git reset`, `git checkout --`, `git revert` (fix mistakes safely).
- **Merge Conflicts** → resolving conflicts in real scenarios.
- **Rebasing** → `git rebase` vs `git merge` (cleaner history).
- **Tags & Releases** → `git tag`, GitHub Releases (versioning & deployment).
- **Stash** → `git stash` (save unfinished work).
- **Advanced Remote** → `git fetch`, upstream branches (sync with main).
- **Git Log Pretty** → formatting logs for better history.
- **GitHub Advanced** →
    - Creating & managing PRs (reviewers, labels, draft PRs).
    - Using GitHub Issues (link PRs to issues).
    - Branch protection rules (required reviews, CI checks).

Real-World Best Practices Section

- **Commit Message Conventions (Conventional Commits)**:
    - `feat: add user login form`
    - `fix: resolve navbar alignment issue`
    - `docs: update README`
    - `style: add theme file`
    - **Why** → clear history + easier code reviews.

- **Branch Naming Conventions**:
    - `feature/feature-name` → new features
    - `bugfix/issue-id` → bug fixes
    - `hotfix/urgent-issue` → production fix
- **Pull Request Conventions**:
    - Clear title: `feat: implement user profile API`
    - Description with context + screenshots/logs.
    - Link issues (Closes #123).
    - Add reviewers & assign.

- **Code Review Etiquette**:
    - Be respectful & constructive.
    - Explain *why*, not just *what*.
    - Keep PRs small & focused.

Task (Practical)

1. Create a new branch `feature/signup-page`.
2. Make changes in the repo and simulate merge conflicts with another teammate's branch.
3. Resolve conflict → commit with proper commit message.
4. Rebase onto `main`.
5. Create a PR with proper title + description.
6. Review a teammate's PR & approve/leave comments.

Deliverable: A GitHub PR demonstrating proper branch naming, commit convention, conflict resolution, and PR review flow.

Capstone (End of Day 2)

A **team-based GitHub repo** where trainees practice:

- Multiple branches (features, bugfixes).
- Proper commit & branch conventions.
- Creating + reviewing PRs.
- Resolving conflicts in real-time.
- Using GitHub Issues & linking them to PRs.
- Releasing a version with GitHub Tags & Releases.

Final Outcome: Trainees demonstrate **both Git fundamentals + GitHub collaboration workflows**, ready for professional team environments.