Git & GitHub Full Team Workflow Guide

■ BASIC CONCEPTS

- Git = version control system for tracking changes.
- GitHub = remote hosting platform for Git repositories.
- Repo = project folder managed by Git.
- Branch = parallel line of development.
- Commit = save a snapshot of changes.
- Pull Request (PR) = propose merging changes into another branch.

■ COMMON COMMANDS

- git clone <url> # Copy a repo from GitHub to your computer
- git status # Show current changes
- git add . # Stage all changes
- git commit -m 'msg' # Save changes with a message
- git push origin main # Upload changes to GitHub
- git pull origin main # Get latest changes from GitHub

■ MAIN BRANCH POLICY

- main should always be stable and production-ready.
- No one commits directly to main.
- All changes must come through feature branches + PRs.

■ TEAM BRANCHING STRATEGY (3 Developers)

- 1 Frontend Developer → create branches like feature/frontend-navbar, feature/frontend-footer
- 2 Backend Developers → create branches like feature/backend-auth, feature/backend-database
- Each dev works in their own branch.
- All branches merge into main via PRs after review.

■ HOW TO CREATE & USE BRANCHES

1. Create a branch from main:
   git checkout main
   git pull origin main
   git checkout -b feature/branch-name

2. Make your code changes in VS Code.

3. Stage & commit:
   git add .
   git commit -m 'Describe the feature'

4. Push your branch to GitHub:
   git push origin feature/branch-name

5. Open a Pull Request (PR) on GitHub.
   - Request review from teammates.
   - Once approved, merge into main.

■ NAMING CONVENTIONS

- feature/frontend-navbar → for new features
- bugfix/login-error → for fixes
- hotfix/deployment-issue → for urgent production fixes

■ KEEPING BRANCHES UP TO DATE

- Always update your branch with latest main:

```
git checkout main
git pull origin main
git checkout feature/branch-name
git merge main
```

■ DEALING WITH CONFLICTS

- Happens if two people edit the same lines.
- Git will mark conflict sections in files.
- Edit to resolve, then:

```
git add .
git commit -m 'Resolve merge conflict'
```

■ COLLABORATION WORKFLOW

1. Pull latest main before starting work.
2. Create your own branch (feature/...).
3. Develop feature locally.
4. Commit and push branch.
5. Open PR → teammate reviews → merge into main.
6. After merge, delete the feature branch (optional cleanup).

■ EXAMPLE WORKFLOW FOR BLOOM

- Frontend dev:
  - feature/frontend-navbar
  - feature/frontend-footer
- Backend dev 1:
  - feature/backend-auth
  - feature/backend-database
- Backend dev 2:
  - feature/backend-news-api
  - feature/backend-map-endpoints

■ KEEPING MAIN CLEAN

- main branch should always be stable.
- Do not commit directly to main unless it's urgent.
- Always work on feature branches and PRs.

■ KEY TAKEAWAYS

- main = stable, always deployable.
- Each dev = own branch for features/fixes.

- Use clear branch names.
- Pull + merge main frequently to stay up to date.
- All changes go through PRs for review.