Git Collaboration Guide for Web Application Development Projects

Table of Contents

Why Git Collaboration Matters

In your web application development project, you'll be working in teams of 2-3 members to create applications that include:

- A web API backend
- A client application (web or mobile)
- Integration with external APIs
- Real user testing and feedback

Without proper version control, team collaboration becomes chaotic. Git provides:

- **Parallel Development**: Each member can work on different features simultaneously
- **Change Tracking**: Complete history of who changed what and when
- **Conflict Resolution**: Systematic way to handle overlapping changes
- **Backup and Recovery**: Multiple copies of your work across team members
- **Professional Skills**: Industry-standard practices you'll use in your career

Initial Project Setup

Step 1: Repository Creation
One team member (typically the leader) creates the main repository:

```
# Create a new repository on GitHub
# Go to github.com -> New repository
# Repository name: WAD25-XX-project-name (follow course naming convention)
# Make it public (required for course submission)
# Initialize with README
```

Step 2: Team Member Access

```
# Repository owner adds collaborators
# GitHub -> Settings -> Manage access -> Invite a collaborator
# Add team members' GitHub usernames
```

Step 3: Local Setup for All Team Members

```
# Clone the repository to your local machine
git clone https://github.com/username/WAD25-XX-project-name.git

# Navigate to project directory
cd WAD25-XX-project-name

# Verify remote connection
git remote -v

# Check current branch (should be 'main')
git branch
```

Step 4: Project Structure Setup
Create initial project structure:

```
# Create basic project folders
mkdir client server docs
touch README.md .gitignore

# Add initial files
git add .
git commit -m "Initial project structure setup"
git push origin main
```

Branch-Based Development Workflow

The Golden Rule
Never work directly on the main branch for feature development
Workflow Overview

```
main branch (stable, production-ready code)
├── feature/api-development (Member 1)
├── feature/frontend-ui (Member 2)
```

```
├── feature/external-api-integration (Member 3)
└── feature/user-authentication (Any member)
```

Branch Naming Conventions

- feature/feature-name - New features
- bugfix/issue-description - Bug fixes
- enhancement/improvement-name - Improvements
- hotfix/critical-issue - Urgent fixes

Examples:

- feature/user-registration-api
- feature/payment-integration
- feature/responsive-design
- bugfix/login-validation-error

Essential Git Commands

Basic Commands Every Team Member Needs

Checking Status and Changes

```
# See current status of your working directory
git status

# See what changes you've made
git diff

# See commit history
git log --oneline --graph

# See all branches
git branch -a
```

Working with Branches

```
# Create and switch to a new branch
git checkout -b feature/my-new-feature

# Switch to existing branch
git checkout branch-name

# Switch to main branch
git checkout main
```

```
# Delete a branch (locally)
git branch -d feature/completed-feature

# Delete a branch (remotely)
git push origin --delete feature/completed-feature
```

Staging and Committing Changes

```
# Add specific files to staging area
git add filename.js

# Add all changes to staging area
git add .

# Commit with message
git commit -m "Add user authentication functionality"

# Add and commit in one step (for tracked files only)
git commit -am "Fix navigation menu styling"
```

Synchronizing with Remote Repository

```
# Get latest changes from remote
git pull origin main

# Push your branch to remote
git push origin feature/my-branch

# Push and set upstream tracking
git push -u origin feature/my-branch
```

Working with Feature Branches

Scenario: Member 1 Working on API Development

```
# 1. Start from latest main branch
git checkout main
git pull origin main

# 2. Create feature branch for API work
git checkout -b feature/user-api-endpoints

# 3. Work on your feature (create files, edit code)
# ... coding work ...

# 4. Stage and commit your changes regularly
git add server/routes/users.js
git commit -m "Add GET /api/users endpoint with pagination"

git add server/routes/auth.js
git commit -m "Add POST /api/auth/login endpoint with JWT"

# 5. Push your branch to remote
git push -u origin feature/user-api-endpoints

# 6. Continue working and pushing updates
git add server/middleware/auth.js
git commit -m "Add authentication middleware for protected routes"
git push origin feature/user-api-endpoints
```

Scenario: Member 2 Working on Frontend

```
# 1. Start from latest main
git checkout main
git pull origin main

# 2. Create frontend feature branch
git checkout -b feature/user-dashboard-ui

# 3. Work on frontend components
# ... create React components, styles, etc ...

# 4. Commit your work
git add client/src/components/Dashboard.jsx
git add client/src/styles/dashboard.css
git commit -m "Create user dashboard component with responsive layout"
```

```
# 5. Push to remote
git push -u origin feature/user-dashboard-ui
```

Keeping Your Branch Updated

```
# Regularly sync your feature branch with main
git checkout main
git pull origin main
git checkout feature/my-branch
git merge main

# Alternative: rebase (creates cleaner history)
git checkout feature/my-branch
git rebase main
```

Merging and Pull Requests

Creating a Pull Request

1. **Complete your feature**:
   ```
   Bash
   # Ensure your branch is up to date
   git checkout main
   git pull origin main
   git checkout feature/my-branch
   git merge main

   # Push final changes
   git push origin feature/my-branch
   ```

2. **Create Pull Request on GitHub**:
   - Go to your repository on GitHub
   - Click "Compare & pull request" button
   - Fill out the PR template:

Handling Merge Conflicts

When Conflicts Occur
Conflicts happen when two branches modify the same lines of code. Git cannot automatically determine which changes to keep.
Resolving Conflicts Step by Step

```
# Attempt to merge main into your feature branch
git checkout feature/my-branch
git merge main

# Git will output something like:
# Auto-merging src/components/App.js
# CONFLICT (content): Merge conflict in src/components/App.js
# Automatic merge failed; fix conflicts and then commit the result.
```

Conflict Resolution Process

1. **Open the conflicted file**:
   JavaScript
   ```
   // File: src/components/App.js
   import React from 'react';

   function App() {
   <<<<<<< HEAD (your current branch)
     const apiUrl = 'http://localhost:3001/api';
     const theme = 'dark';
   =======
     const apiUrl = 'https://api.myproject.com';
     const theme = 'light';
   >>>>>>> main (incoming changes)

     return (
       <div className="App">
         {/* rest of component */}
       </div>
     );
   }
   ```

2. **Decide which changes to keep**:
   JavaScript
   ```
   // Option 1: Keep your changes
   ```

```
const apiUrl = 'http://localhost:3001/api';
const theme = 'dark';

// Option 2: Keep incoming changes
const apiUrl = 'https://api.myproject.com';
const theme = 'light';

// Option 3: Combine both (often the best solution)
const apiUrl = process.env.NODE_ENV === 'production'
  ? 'https://api.myproject.com'
  : 'http://localhost:3001/api';
const theme = 'dark'; // Keep your theme preference
```

3. **Complete the merge**:
   ```
   Bash
   # Remove conflict markers and save the file
   git add src/components/App.js
   git commit -m "Resolve merge conflict in App.js: combine API URLs with environment check"
   ```

Prevention Strategies

- **Communicate** with team members about which files you're working on
- **Pull main frequently** to stay updated
- **Keep feature branches small** and short-lived
- **Use different folders/files** when possible for parallel development

Best Practices for Team Projects

Commit Message Guidelines

Good commit messages:
```
git commit -m "Add user registration API endpoint with email validation"
git commit -m "Fix responsive layout issue on mobile devices"
git commit -m "Update README with API documentation"
git commit -m "Integrate external weather API for location-based features"
```

Poor commit messages (avoid these):

```
git commit -m "fix"
git commit -m "update"
git commit -m "working version"
git commit -m "asdf"
```

File and Folder Organization

```
WAD25-XX-project-name/
├── client/                 # Frontend application
│   ├── src/
│   ├── public/
│   └── package.json
├── server/                 # Backend API
│   ├── routes/
│   ├── models/
│   ├── middleware/
│   └── package.json
├── docs/                   # Project documentation
│   ├── api-documentation.md
│   ├── setup-instructions.md
│   └── user-guide.md
├── .gitignore
├── README.md
└── package.json            # Root package.json for scripts
```

.gitignore Configuration

```
# Dependencies
node_modules/
npm-debug.log*

# Environment variables
.env
.env.local
.env.development.local
.env.test.local
.env.production.local

# Build outputs
build/
dist/
.next/

# IDE files
.vscode/
.idea/
*.swp
*.swo
```

```
# OS generated files
.DS_Store
Thumbs.db

# Log files
logs/
*.log

# Database files
*.sqlite
*.db
```

Branch Protection Rules
Set up branch protection on GitHub:

1. **Go to Settings > Branches**
2. **Add rule for main branch**:
    - Require pull request reviews before merging
    - Require status checks to pass before merging
    - Restrict pushes to main branch

Common Scenarios and Solutions

Scenario 1: Accidentally Committed to Main Branch

```
# If you haven't pushed yet
git log --oneline  # Find the commit hash before your changes
git reset --soft HEAD~1  # Undo last commit, keep changes staged
git stash  # Temporarily save your changes
git checkout -b feature/my-work  # Create proper feature branch
git stash pop  # Restore your changes
git commit -m "Proper commit message"
git push -u origin feature/my-work
```

Scenario 2: Need to Share Work Before Feature is Complete

```
# Push your feature branch for others to see
git push -u origin feature/work-in-progress

# Other team members can check it out
git fetch origin
git checkout feature/work-in-progress

# Continue collaborating on the same branch
```

```
git pull origin feature/work-in-progress  # Get latest changes
# ... make changes ...
git push origin feature/work-in-progress   # Share your updates
```

Scenario 3: Emergency Fix Needed on Main

```
# Create hotfix branch from main
git checkout main
git pull origin main
git checkout -b hotfix/critical-login-bug

# Make the fix
# ... fix the bug ...
git add .
git commit -m "Fix critical login validation bug"
git push -u origin hotfix/critical-login-bug

# Create immediate pull request and merge
# Then update all feature branches
git checkout feature/my-branch
git merge main  # Get the hotfix
```

Scenario 4: Feature Branch Became Too Large

```
# Break large branch into smaller pieces
git checkout feature/large-feature
git checkout -b feature/user-model
# Move relevant files and commits
git cherry-pick commit-hash-1 commit-hash-2

git checkout feature/large-feature
git checkout -b feature/api-endpoints
# Move other relevant commits
```

Project Structure Examples

Example 1: E-commerce Web App

```
WAD25-05-ecommerce/
├── client/                 # React frontend
│   ├── src/
│   │   ├── components/
│   │   │   ├── ProductList.jsx
│   │   │   ├── ShoppingCart.jsx
```

```
│   │   │   └── UserProfile.jsx
│   │   ├── pages/
│   │   ├── services/
│   │   └── utils/
│   └── package.json
├── server/                 # Node.js/Express API
│   ├── routes/
│   │   ├── products.js
│   │   ├── users.js
│   │   └── orders.js
│   ├── models/
│   ├── middleware/
│   └── package.json
└── docs/
    ├── api-endpoints.md
    └── external-apis.md
```

Team Assignment:

- Member 1: Product management API + external inventory API
- Member 2: User authentication + payment API integration
- Member 3: Frontend components + shopping cart functionality

Example 2: Travel Planning App

```
WAD25-07-travel-planner/
├── mobile-app/         # React Native or Flutter
├── web-client/         # Web dashboard
├── api-server/         # Backend services
├── docs/
└── shared/             # Shared utilities
```

Team Assignment:

- Member 1: Backend API + Google Maps integration
- Member 2: Mobile app development
- Member 3: Web dashboard + weather API integration

Troubleshooting Common Issues

"Permission denied" when pushing

Bash

```
# Check if you're added as collaborator
git remote -v

# Verify you're authenticated
git config user.name
git config user.email

# Re-authenticate if needed
git config user.name "Your Name"
git config user.email "your.email@example.com"
```

"Branch is behind" error

```
# Pull latest changes first
git pull origin main

# Then push your changes
git push origin your-branch
```

Lost work / accidentally deleted files

```
# See what was deleted
git log --diff-filter=D --summary

# Recover deleted file
git checkout HEAD~1 -- path/to/deleted/file

# Or restore from specific commit
git checkout commit-hash -- path/to/file
```

Merge conflicts in package.json

```
# Usually safe to accept both changes and reinstall
git checkout --theirs package.json
git checkout --ours package-lock.json
npm install
git add package.json package-lock.json
git commit -m "Resolve package.json merge conflict"
```

Final Tips for Success

1. **Communicate Early and Often**: Use GitHub issues, project boards, and team chat
2. **Test Before Merging**: Always test your changes locally and ensure they work with main
3. **Document Your APIs**: Keep API documentation updated as you develop
4. **Regular Team Sync**: Daily or every-other-day sync to avoid conflicts
5. **Code Reviews Matter**: Take time to review teammates' code thoughtfully
6. **Backup Strategy**: Your Git repository IS your backup - use it well
7. **Professional Presentation**: Clean Git history shows good development practices

Remember: These Git skills are not just for your course project - they're essential professional skills that you'll use throughout your software development career. Invest time in learning them well now, and they'll serve you for years to come.

Quick Reference Commands

```
# Daily workflow commands
git status                    # Check current state
git checkout main             # Switch to main branch
git pull origin main          # Get latest changes
git checkout -b feature/new-work # Create feature branch
git add .                     # Stage changes
git commit -m "Descriptive message" # Commit changes
git push origin feature/new-work # Push to remote
git checkout main             # Switch back to main
git merge feature/completed-work # Merge completed feature
git branch -d feature/completed-work # Delete merged branch
```

Good luck with your web application development project!