

# COMPLETE GITHUB COLLABORATION MANUAL

Weather App Project - Team Workflow Guide

---

## PROJECT OVERVIEW

### Project Mission

Build a single-page weather application that displays real-time weather data for Cameroonian cities using a public weather API.

### Team Structure

- **Leader:** Project coordination & final integration
- **Member 1:** HTML Structure
- **Member 2:** CSS Styling
- **Member 3:** API Integration
- **Member 4:** DOM Updates
- **Member 5:** App Logic

### Key Features

- Real-time weather data display
  - City search functionality
  - Dynamic weather icons and backgrounds
  - Loading states and error handling
  - Mobile-responsive design
- 

## PHASE 1: LEADER INITIAL SETUP

### Step 1: Repository Creation & Initial Setup

```
# Clone the repository to your local machine
git clone https://github.com/your-username/weather-app.git
cd weather-app
```

```
# Verify remote connection
git remote -v
```

### Step 2: Create Team Branches

```
# Create dedicated branches for each team member
git branch member1-html
```

```
git branch member2-css
git branch member3-api
git branch member4-ui
git branch member5-logic
```

*# Push all branches to GitHub*

```
git push -u origin member1-html
git push -u origin member2-css
git push -u origin member3-api
git push -u origin member4-ui
git push -u origin member5-logic
```

*# Verify all branches are on GitHub*

```
git branch -r
```

### Step 3: Establish Project Foundation

*# Return to main branch*

```
git checkout main
```

*# Create basic folder structure*

```
mkdir -p assets/images assets/icons src docs
```

*# Add initial files*

```
touch index.html src/styles.css src/script.js src/api.js src/ui.js README.md
```

*# Commit and push foundation*

```
git add .
git commit -m "Project foundation: folder structure and initial files"
git push origin main
```



## PHASE 2: TEAM MEMBER DAILY WORKFLOW

### For Each Team Member

#### **Daily Startup Sequence:**

*# 1. Navigate to project folder*

```
cd weather-app
```

*# 2. Switch to your assigned branch*

```
git checkout member1-html    # Replace with your branch
```

*# 3. Get latest updates from main branch*

```
git pull origin main
```

*# 4. Start working on your tasks*

### ***During Work Session:***

*# Check your current status*

```
git status
```

*# See what files you've modified*

```
git diff
```

*# When ready to save progress:*

```
git add .
```

```
git commit -m "Descriptive message about your changes"
```

*# Push to your branch on GitHub*

```
git push origin member1-html # Replace with your branch
```

### ***Example Commit Messages:***

- Member 1: "HTML: Added search bar container with proper IDs"
  - Member 2: "CSS: Implemented responsive weather card layout"
  - Member 3: "API: Added error handling for network failures"
  - Member 4: "UI: Created loading state display functions"
  - Member 5: "Logic: Connected search input to API calls"
- 

## PHASE 3: REGULAR SYNCHRONIZATION

### **Weekly Integration Schedule**

#### ***Monday Morning - Sync Day:***

*# Each member updates from main*

```
git checkout member1-html
```

```
git pull origin main
```

*# If conflicts occur, resolve them:*

```
git status # See conflicted files
```

*# Edit files to resolve conflicts, then:*

```
git add .
```

```
git commit -m "Resolved merge conflicts with main"
```

```
git push origin member1-html
```

#### ***Wednesday - Midweek Check:***

*# Quick update from main*

```
git checkout member1-html
```

```
git pull origin main
```

*# Continue working*

#### ***Friday - Preparation for Integration:***

*# Final update before weekend*

```
git checkout member1-html
```

```
git pull origin main
git push origin member1-html
```

*# Inform Leader your branch is ready for review*

---

## PHASE 4: LEADER INTEGRATION PROCESS

### Integration Command Sequence:

*# 1. Ensure main branch is up to date*

```
git checkout main
git pull origin main
```

*# 2. Merge Member 1 work (HTML)*

```
git merge member1-html
```

*# 3. If conflicts occur, resolve and commit*

```
git add .
git commit -m "Integrated HTML structure from member1"
```

*# 4. Continue with other members*

```
git merge member2-css
git merge member3-api
git merge member4-ui
git merge member5-logic
```

*# 5. Push updated main to GitHub*

```
git push origin main
```

*# 6. Notify team to update their branches*

### Conflict Resolution Protocol:

*# When merge conflict occurs:*

```
git status
```

*# Open conflicted files, look for <<<<<< HEAD and ===== markers*

*# Edit to keep correct versions, remove conflict markers*

```
git add .
git commit -m "Resolved integration conflicts"
```

---

## PROJECT TIMELINE & MILESTONES

### Week 1: Foundation Building

#### Leader Tasks:

```
# Day 1: Initial setup complete
git tag week1-foundation
git push origin week1-foundation
```

```
# Day 3: First integration
git merge member1-html member2-css
git commit -m "Week 1: HTML and CSS foundation integrated"
```

**Member Deliverables:** - Member 1: Complete HTML skeleton with all required IDs - Member 2: Basic responsive layout and styling - Member 3: API connection structure with mock data - Member 4: DOM update function templates - Member 5: Search input capture logic

## Week 2: Core Integration

### Leader Tasks:

```
# Day 8: Major integration
git merge member3-api member4-ui member5-logic
git tag week2-core-integration
git push origin week2-core-integration
```

**Member Deliverables:** - Member 3: Live API integration with error handling - Member 4: Real data display implementation - Member 5: Complete user interaction flow

## Week 3: Polish & Deployment

### Leader Tasks:

```
# Final integration
git merge --no-ff all-branches
git tag v1.0-final
git push origin v1.0-final
```

```
# Deployment to Netlify
# Connect GitHub repo to Netlify for automatic deployment
```

---

## GITHUB BEST PRACTICES

### Branch Management:

```
# Check which branch you're on
git branch
```

```
# See all branches (local and remote)
git branch -a
```

```
# Create a backup branch before major changes
```

```
git branch backup-branch
```

```
# Delete a local branch (after merging)
```

```
git branch -d old-branch
```

## Commit Hygiene:

```
# Make small, frequent commits
```

```
git add specific-file.js
```

```
git commit -m "Clear, descriptive message"
```

```
# Avoid giant commits - break into logical chunks
```

## Recovery Commands:

```
# Undo local changes (before commit)
```

```
git checkout -- filename
```

```
# Reset to last commit (careful!)
```

```
git reset --hard HEAD
```

```
# Recover lost commits
```

```
git reflog
```



## TROUBLESHOOTING COMMON ISSUES

### Accidental Work on Wrong Branch:

```
# Save your work temporarily
```

```
git stash
```

```
# Switch to correct branch
```

```
git checkout correct-branch
```

```
# Restore your work
```

```
git stash pop
```

### Behind Main Branch:

```
git checkout your-branch
```

```
git fetch origin
```

```
git merge origin/main
```

```
# Resolve conflicts, then push
```

```
git push origin your-branch
```

### Lost Connection During Push:

```
git push origin your-branch
```

```
# If fails:
```

```
git pull --rebase origin your-branch
git push origin your-branch
```

---

## QUALITY ASSURANCE CHECKPOINTS

### Before Integration:

```
# Each member verifies their work
git status
git diff origin/main    # See differences from main
```

```
# Test functionality locally
# Ensure no breaking changes
```

### After Integration:

```
# Leader verifies main branch
git checkout main
git log --oneline -10    # Check recent commits
```

```
# Test integrated application
# Verify all features work together
```

---

## FINAL DELIVERY PROCEDURE

### Pre-Deployment Checklist:

```
# 1. Ensure all branches are merged
git branch --merged main

# 2. Create final release tag
git tag -a v1.0 -m "Final weather app release"
git push origin v1.0
```

```
# 3. Verify deployment readiness
git checkout main
git status
```

### Post-Project Archive:

```
# Archive development branches
git branch -d member1-html member2-css member3-api member4-ui member5-logic
git push origin --delete member1-html member2-css member3-api member4-ui
member5-logic
```

---

## COMMUNICATION PROTOCOL

### Daily Standup Format:

Current Branch: member1-html

Yesterday: Completed search bar HTML structure

Today: Working on weather info containers






Blockers: Need CSS class names from Member 2


Git Status: 3 commits ahead of main, no conflicts

### GitHub Collaboration Tools:

- Use **GitHub Issues** for bug tracking
  - **Pull Requests** for code review (optional)
  - **Project Board** for task management
  - **Wiki** for documentation
- 

## SUCCESS METRICS

-  All 5 branches successfully merged into main
  -  No merge conflicts in final integration
  -  Application deployed and functional on Netlify
  -  Each member has consistent commit history
  -  Project completed within 3-week timeline
- 

 **PROJECT COMPLETE!** Your team has successfully collaborated using GitHub to deliver a fully functional weather application while maintaining clean version control throughout the development process.

*This manual ensures every team member understands their GitHub responsibilities and can work efficiently without blocking others.*