# GitHub Collaboration Workflow Guide

# GitHub Collaboration Workflow Guide

This guide outlines how the IDEEAS Lab uses GitHub for collaboration, code review, and project management. Following these practices ensures smooth collaboration across our diverse team.

## Repository Organization

### Repository Naming Convention

* **Lab-wide repositories**: ideeas-lab-[purpose] (e.g., ideeas-lab-templates)
* **Project repositories**: [project-name] (e.g., ai-tutoring-study)
* **Personal repositories**: [username]-[purpose] (e.g., jsmith-dissertation)

### Repository Structure

repository-name/  
├── README.md # Project overview and setup instructions  
├── LICENSE # License for the code/data  
├── .gitignore # Files to exclude from version control  
├── CONTRIBUTING.md # Guidelines for contributors  
├── CODE\_OF\_CONDUCT.md # Code of conduct for contributors  
├── requirements.txt # Python dependencies  
├── environment.yml # Conda environment specification  
├── src/ # Source code  
├── data/ # Data files (following data management policy)  
├── docs/ # Documentation  
├── tests/ # Unit tests  
└── .github/ # GitHub-specific files  
 ├── ISSUE\_TEMPLATE/ # Issue templates  
 ├── PULL\_REQUEST\_TEMPLATE.md  
 └── workflows/ # GitHub Actions workflows

## Branching Strategy

### Branch Types

**Main Branch (main)**: - Always deployable/runnable - Protected branch requiring pull request reviews - Represents the current stable state

**Feature Branches (feature/[description])**: - For developing new features or analyses - Branch from main, merge back via pull request - Delete after successful merge

**Hotfix Branches (hotfix/[description])**: - For urgent fixes to main branch - Branch from main, merge back immediately after review

**Personal Branches ([username]/[description])**: - For experimental work or personal exploration - Can be long-lived, no requirement to merge

### Branch Naming Conventions

# Good examples  
feature/data-cleaning-pipeline  
feature/survey-analysis  
hotfix/missing-data-bug  
jsmith/exploratory-analysis  
  
# Poor examples  
new-stuff  
fix  
temp  
branch1

## Commit Guidelines

### Commit Message Format

<type>(<scope>): <subject>  
  
<body>  
  
<footer>

**Types**: - feat: New feature or analysis - fix: Bug fix - docs: Documentation changes - style: Code formatting (no logic changes) - refactor: Code restructuring (no behavior changes) - test: Adding or updating tests - chore: Maintenance tasks

**Examples**:

feat(analysis): Add statistical significance testing  
  
Implement t-tests and effect size calculations for comparing  
treatment groups in the tutoring effectiveness study.  
  
Closes #23

fix(data): Handle missing values in survey responses  
  
Replace NaN values with appropriate defaults based on  
question type and add validation checks.  
  
Fixes #45

### Commit Best Practices

* Make atomic commits (one logical change per commit)
* Write clear, descriptive commit messages
* Commit frequently to track progress
* Don’t commit sensitive data or large binary files
* Use present tense (“Add feature” not “Added feature”)

## Pull Request Process

### Creating Pull Requests

1. **Create Feature Branch**:

git checkout main  
git pull origin main  
git checkout -b feature/new-analysis

1. **Make Changes and Commit**:

# Make your changes  
git add .  
git commit -m "feat(analysis): Add new statistical analysis"  
git push origin feature/new-analysis

1. **Create Pull Request**:

* Go to GitHub repository
* Click “New Pull Request”
* Select your feature branch
* Fill out PR template

### Pull Request Template

## Description  
Brief description of changes made.  
  
## Type of Change  
- [ ] Bug fix (non-breaking change that fixes an issue)  
- [ ] New feature (non-breaking change that adds functionality)  
- [ ] Breaking change (fix or feature that would cause existing functionality to not work as expected)  
- [ ] Documentation update  
  
## Testing  
- [ ] I have tested these changes locally  
- [ ] I have added tests that prove my fix is effective or that my feature works  
- [ ] New and existing unit tests pass locally with my changes  
  
## Checklist  
- [ ] My code follows the lab's style guidelines  
- [ ] I have performed a self-review of my own code  
- [ ] I have commented my code, particularly in hard-to-understand areas  
- [ ] I have made corresponding changes to the documentation  
- [ ] My changes generate no new warnings  
  
## Related Issues  
Closes #[issue number]

### Review Process

**For Reviewers**: 1. Check code quality and style 2. Verify functionality and logic 3. Test changes locally if needed 4. Provide constructive feedback 5. Approve or request changes

**Review Checklist**: - [ ] Code is readable and well-documented - [ ] Logic is sound and efficient - [ ] Tests are included and pass - [ ] Documentation is updated - [ ] No sensitive data is exposed - [ ] Follows lab coding standards

**Review Comments**:

# Constructive feedback examples  
Consider using a more descriptive variable name here for clarity.  
  
This analysis looks great! Could you add a comment explaining the statistical test choice?  
  
Minor: This could be simplified using pandas' built-in function.  
  
# Approval examples  
LGTM! Great work on the data visualization.  
Excellent analysis - the results are clearly presented.

## Issue Management

### Issue Types and Labels

**Issue Types**: - bug: Something isn’t working correctly - enhancement: New feature or improvement - documentation: Documentation needs - question: Questions about the project - help wanted: Extra attention needed - good first issue: Good for newcomers

**Priority Labels**: - priority: high: Urgent issues - priority: medium: Important but not urgent - priority: low: Nice to have

**Status Labels**: - status: in progress: Currently being worked on - status: blocked: Waiting for something else - status: needs review: Ready for review

### Issue Templates

**Bug Report Template**:

## Bug Description  
A clear description of what the bug is.  
  
## Steps to Reproduce  
1. Go to '...'  
2. Click on '....'  
3. Scroll down to '....'  
4. See error  
  
## Expected Behavior  
What you expected to happen.  
  
## Actual Behavior  
What actually happened.  
  
## Environment  
- OS: [e.g. macOS, Windows, Linux]  
- Python version: [e.g. 3.9.7]  
- Package versions: [relevant package versions]  
  
## Additional Context  
Any other context about the problem.

**Feature Request Template**:

## Feature Description  
A clear description of what you want to happen.  
  
## Use Case  
Describe the use case or problem this feature would solve.  
  
## Proposed Solution  
Describe the solution you'd like.  
  
## Alternatives Considered  
Describe any alternative solutions you've considered.  
  
## Additional Context  
Any other context or screenshots about the feature request.

## Code Review Standards

### What to Review

**Code Quality**: - Readability and clarity - Proper documentation and comments - Consistent style and formatting - Efficient algorithms and data structures

**Functionality**: - Logic correctness - Edge case handling - Error handling and validation - Test coverage

**Research Quality**: - Statistical methods appropriateness - Data handling correctness - Reproducibility considerations - Ethical considerations

### Review Guidelines

**For Authors**: - Keep PRs small and focused - Provide clear descriptions - Respond to feedback promptly - Test your changes thoroughly - Update documentation as needed

**For Reviewers**: - Be constructive and respectful - Focus on the code, not the person - Explain your suggestions - Approve when ready, don’t nitpick - Review promptly (within 2-3 days)

## Project Management

### Using GitHub Projects

* Create project boards for major initiatives
* Use columns: Backlog, In Progress, Review, Done
* Link issues and PRs to project cards
* Update status regularly

### Milestone Management

* Create milestones for major deadlines
* Assign issues to appropriate milestones
* Track progress toward milestone completion
* Adjust timelines as needed

### Release Management

* Tag important versions: v1.0.0, v1.1.0, etc.
* Write release notes describing changes
* Create releases for major milestones
* Archive old releases appropriately

## Collaboration Best Practices

### Communication

* Use issue comments for technical discussions
* Tag relevant people with @mentions
* Use draft PRs for work-in-progress
* Link related issues and PRs

### Documentation

* Keep README files up to date
* Document API changes
* Include examples in documentation
* Write clear commit messages

### Code Organization

* Use consistent file and folder structure
* Follow naming conventions
* Keep functions and files focused
* Remove dead code regularly

### Data Management

* Never commit sensitive data
* Use .gitignore for data files
* Document data sources and structure
* Follow lab data management policies

## Troubleshooting Common Issues

### Merge Conflicts

# Update your branch with latest main  
git checkout main  
git pull origin main  
git checkout feature/your-branch  
git merge main  
  
# Resolve conflicts in files  
# Edit conflicted files, remove conflict markers  
git add .  
git commit -m "Resolve merge conflicts"  
git push origin feature/your-branch

### Accidentally Committed Sensitive Data

# Remove file from history (use with caution)  
git filter-branch --force --index-filter \  
'git rm --cached --ignore-unmatch path/to/sensitive/file' \  
--prune-empty --tag-name-filter cat -- --all  
  
# Force push (coordinate with team first)  
git push origin --force --all

### Large File Issues

# Use Git LFS for large files  
git lfs track "\*.csv"  
git lfs track "\*.pkl"  
git add .gitattributes  
git add large-file.csv  
git commit -m "Add large file with LFS"

## Security and Access Management

### Repository Access Levels

* **Read**: Can view and clone repository
* **Write**: Can push to repository and create PRs
* **Admin**: Full access including settings and permissions

### Branch Protection Rules

* Require PR reviews before merging to main
* Require status checks to pass
* Require branches to be up to date
* Restrict who can push to main branch

### Sensitive Information

* Never commit passwords, API keys, or personal data
* Use environment variables for secrets
* Add sensitive files to .gitignore
* Use GitHub Secrets for CI/CD workflows

**Remember**: Good collaboration practices make everyone more productive and help maintain high-quality research outputs. When in doubt, ask for help or clarification!