GitHub Integration Guide for Students

Phase 1: Getting Started with GitHub

Step 1: Account Setup

1. Visit github.com and create a free account
2. Choose a professional username (avoid special characters or numbers if possible)
3. Verify your email address through the confirmation email
4. Complete your profile with a professional photo and bio
5. Enable two-factor authentication for security

Step 2: Install Required Tools

1. **Install Git on your computer:**
   - Windows: Download from git-scm.com
   - Mac: Install via Homebrew (`brew install git`) or download from git-scm.com
   - Linux: Use package manager (`sudo apt install git` for Ubuntu/Debian)
2. **Choose a code editor with Git integration:**
   - Visual Studio Code (recommended for beginners)
   - GitHub Desktop (GUI alternative)
   - Command line (for advanced users)

Step 3: Configure Git Locally

Open terminal/command prompt and run:

```bash
git config --global user.name "Your Full Name"
git config --global user.email "your.email@example.com"
git config --global init.defaultBranch main
```

Phase 2: Repository Setup and Organization

Step 4: Create Your Course Repository Structure

1. **Main Course Repository:**
   - Create a new repository named `[course-code]-[semester]` (e.g., `CS101-Fall2024`)

- Make it public for easy sharing with instructors
- Initialize with a README.md file

2. **Repository Structure:**

```
CS101-Fall2024/
├── README.md
├── assignments/
│   ├── assignment-01/
│   ├── assignment-02/
│   └── ...
├── practice-exercises/
│   ├── week-01/
│   ├── week-02/
│   └── ...
├── projects/
│   ├── midterm-project/
│   └── final-project/
└── notes/
    ├── lecture-notes.md
    └── resources.md
```

Step 5: Clone Repository Locally

1. Copy the repository URL from GitHub
2. Open terminal in your desired local directory
3. Run: `git clone [repository-url]`
4. Navigate to the repository: `cd [repository-name]`

Phase 3: Daily Workflow Integration

Step 6: Establish a Consistent Workflow

1. **Before starting work:**

```bash
git pull origin main  # Get latest changes
```

2. **After completing work:**

```bash
```

```
git add .              # Stage all changes
git commit -m "Descriptive message"  # Commit with clear message
git push origin main       # Push to GitHub
```

Step 7: Commit Message Best Practices

Use clear, descriptive commit messages:

- `Add: Assignment 1 solution`
- `Complete: Week 3 practice exercises`
- `Fix: Bug in sorting algorithm`
- `Update: README with project description`
- `Refactor: Improve code structure in main.py`

Phase 4: Advanced GitHub Features

Step 8: Branching for Different Work

1. **Create branches for major assignments:**

```
bash

git checkout -b assignment-01
# Work on assignment
git add .
git commit -m "Complete assignment 01"
git push origin assignment-01
```

2. **Merge back to main when complete:**

```
bash

git checkout main
git merge assignment-01
git push origin main
```

Step 9: Using Issues for Task Management

1. Create issues for each assignment or project
2. Use labels like "assignment", "bug", "enhancement"
3. Reference issues in commit messages: "Fixes #3: Complete data structure implementation"
4. Close issues when work is completed

Step 10: Collaboration Features

1. **Fork repositories** for group projects
2. **Create pull requests** for code reviews
3. **Use GitHub Discussions** for course-related questions
4. **Star important repositories** for quick access

Phase 5: Documentation and Presentation

Step 11: Create Comprehensive README Files

For each project folder, include:

```markdown
# Project Name

## Description
Brief description of what the project does

## Requirements
- Programming language version
- Dependencies
- Installation instructions

## Usage
How to run the program

## Screenshots/Examples
Visual examples of the program running

## Reflection
What you learned from this project
```

Step 12: Portfolio Development

1. **Pin important repositories** to your profile
2. **Create a profile README** showcasing your work
3. **Use GitHub Pages** to host project demos
4. **Maintain consistent commit history** showing regular progress

Phase 6: Integration with Course Management

Step 13: Submission Workflow

1. **Create release tags** for final submissions:

```bash
git tag -a v1.0 -m "Assignment 1 final submission"
git push origin v1.0
```

2. **Share repository links** with instructors via:
   - Course management system
   - Email with specific commit hashes
   - Direct GitHub repository URLs

Step 14: Backup and Version Control

1. **Regular commits** ensure work is never lost
2. **Multiple branches** allow experimentation without risk
3. **GitHub serves as cloud backup** for all coursework
4. **History tracking** shows your learning progression

Best Practices and Tips

Organization Tips:

- Use consistent naming conventions
- Keep repositories organized with clear folder structures
- Include .gitignore files for language-specific temporary files
- Write meaningful commit messages

Collaboration Tips:

- Comment your code thoroughly
- Use pull requests even for solo projects to practice
- Engage with classmates' repositories through stars and issues
- Participate in code reviews

Professional Development:

- Maintain an active commit history
- Showcase your best work prominently

- Use GitHub's social features appropriately
- Build a portfolio that demonstrates growth over time

Troubleshooting Common Issues

Git Command Issues:

- **Merge conflicts:** Use `git status` to identify files, resolve manually
- **Forgot to pull:** Use `git pull --rebase origin main`
- **Wrong commit message:** Use `git commit --amend -m "New message"`

GitHub Access Issues:

- **Authentication problems:** Set up SSH keys or use personal access tokens
- **Permission denied:** Check repository visibility settings
- **Sync issues:** Ensure local and remote repositories are connected properly

Getting Help

Resources:

- GitHub's official documentation
- Git tutorials and cheat sheets
- Course instructor or teaching assistants
- GitHub community forums
- Stack Overflow for specific technical issues

Emergency Recovery:

- GitHub keeps full history - previous versions are always recoverable
- Use `git log` to find specific commits
- Create issues on your repository to ask for help
- Most problems can be solved without losing work

This guide provides a foundation for using GitHub effectively throughout your coursework. Start with the basics and gradually incorporate more advanced features as you become comfortable with the platform.