GitHub Education / Classroom 50
Instructor Guide and Setup

Prepared for Austin Community College Software Development & Computer Science Department

Alexander Katrompas, PhD
August 24th, 2026

Table of Contents

Preface 2
Introduction 2
Setup 3
Understanding Organizations 3
Making an Organization 4
Understanding Classrooms 8
Making Classrooms 9
Understanding Assignments 10
Making Assignments 10
    Preparing an Assignment Template 10
    Creating the Assignment 11
Understanding Rosters 14
Managing Rosters 15
    Adding One Student 15
    Student Organization Invitation 16
    Adding Students in Bulk 16
Understanding Assignment Acceptance 17
Accepting an Assignment 17
Student Development and Autograding 20
Student Submission and Instructor Grading 22
    Student Submission 22
    Instructor Grading 22
    Instructor Workflow Suggestion 23
Appendix A: ASSIGNMENT.md 24
Appendix B: main.py 28
Appendix C: Beyond Basic Setup 29
References 30

Preface

The following guide provides a basic setup for using GitHub Education along with Classroom 50 to teach and employ version control in a classroom environment.

A central purpose of this workflow is to make version control part of the programming process rather than merely a submission mechanism. Students are expected to develop incrementally, make coherent commits with meaningful messages, test as they work, and maintain a professional repository history that reflects how the software was built.

This becomes especially important in an AI-enabled classroom, where students may use generative tools to assist with coding but still need to demonstrate that they understand the problem, can decompose it into logical steps, can verify the resulting code, and can manage the development process themselves. GitHub and Classroom 50 provide a practical way to teach those habits using the same tools and workflows students are likely to encounter in professional software development.

Introduction

- **GitHub Education**: GitHub Education provides verified faculty and students with educational access to GitHub's professional development ecosystem, along with teaching resources and additional benefits. Its primary value in computer science education is that students learn using the same Git, repository, collaboration, and software-development workflows used in professional environments.

- **Classroom 50**: Classroom 50 is a free, open-source classroom-management system built around GitHub repositories. It allows instructors to distribute programming assignments, manage individual student repositories, configure automated testing and grading, collect submissions, and review student work while retaining the complete Git development history.

- **Official Classroom 50 Guide for Instructors**: The Classroom 50 project maintains an official Web Teacher Guide covering setup, classrooms, assignments, rosters, grading, and other features. This guide expands on that material with additional explanation, recommended settings, and a tested workflow for classroom use.

- **General GitHub Tutorial and Guide for Students**: This is a general guide for version control with GitHub (not tied to Classroom 50) to teach version control stand-alone.

- **Video introducing version control in general**: This is a general video for version control with GitHub (not tied to Classroom 50) to teach version control stand-alone. It follows the previous point's guide.

Setup

- If you are unfamiliar with Git, GitHub, and version control, follow the steps below. If you already have an account along with educator access and are comfortable with these concepts, skip this section.

    - Get a GitHub account.
    - Get educator access. Approval time varies and may take days, so apply well before the semester begins.
    - Learn Git and GitHub with the references above or other widely available tutorials.
    - Learn about GitHub Education for Educators.
    - Further references are at the end of this guide.

Understanding Organizations

- To use Classroom 50, you need a GitHub organization on the GitHub Team or Enterprise plan. Verified educators can upgrade academic organizations to GitHub Team **for free** through GitHub Education.
- An organization is your personal "academic organization" where you will keep your classrooms and student repositories.
- There are different ways to approach organizations. This guide recommends **one organization per semester**. This provides clean semester-level separation and makes it easy to archive all repositories, classrooms, and configurations together. Classroom 50 also supports keeping multiple semesters in one long-lived organization, which some instructors prefer to reduce setup overhead.

    This Guide:
    - Each semester, make **one** new organization for **all** your classes that semester. The benefit of this granularity level is you can manage one semester at a time as a whole.

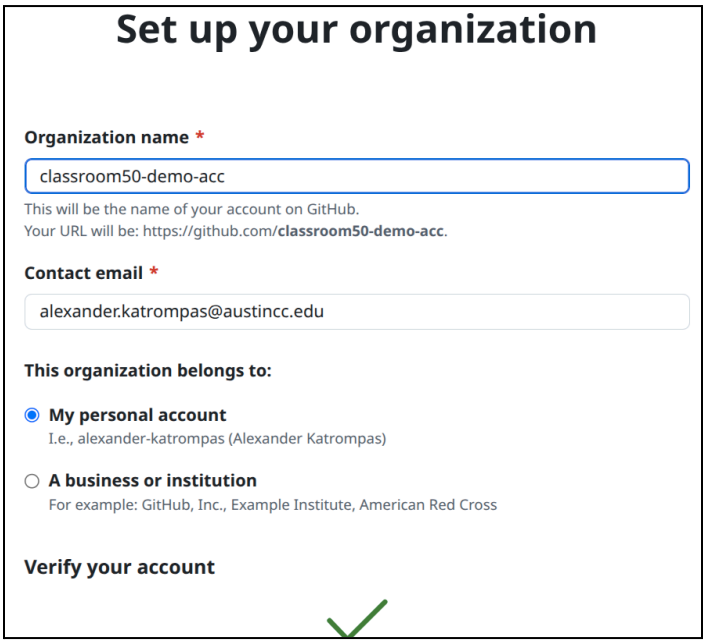
Other granularities are possible.

- One long-lived organization for all classes and all semesters (i.e., make one and done). This reduces semester setup time but may increase complexity later depending on your usage.
- One organization per class per semester. This is maximum granularity and control but may introduce unnecessary complexity for little gain.

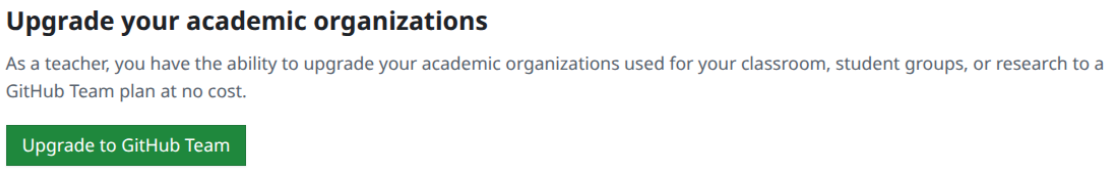
The one organization per semester method *seems* the best blend of flexibility and control; however, multiple configurations and hybrid configurations are possible. The rest of this guide assumes **one organization** for all courses **per semester**.

Making an Organization

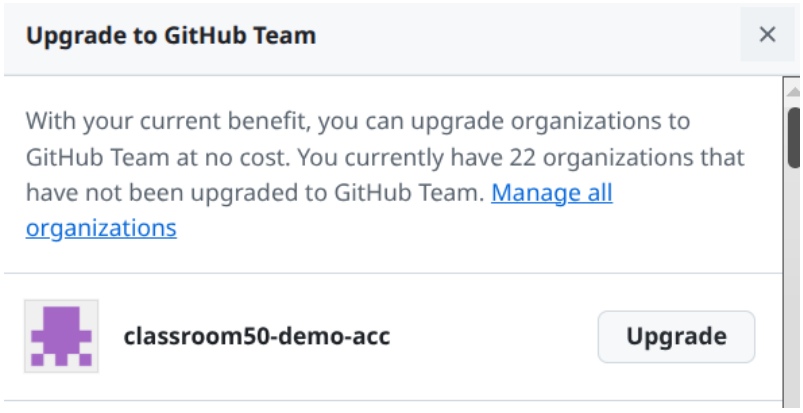
- Go to Organizations (logged into GitHub with your educator benefits active*)*.
- Click **New organization**.
- Choose **Free Plan**.
- Name your organization. It is suggested you use a standard pattern for all semesters with the semester in the name, such as "*Instructor-Name-Spring-2026*" or "*Computer-Science-Spring-2026*". This will help you stay organized.



- Use your ACC email address for the contact information and make sure your ACC email is at least one of the email addresses on your GitHub account, preferably the main email address.
- Choose **Personal Account** for the organization owner and click **Next**.
- At the **Add Organization Members** step, choose **Skip this step**. Do not manually add students while creating the organization. Classroom 50 will manage student invitations later when you add students to the classroom roster.
- Go to **Settings** and complete any organization profile information you want students to see.
- Upgrade your organization to **GitHub Team** plan by going to your education dashboard and under **Upgrade your academic organizations**, select **Upgrade to GitHub Team**.



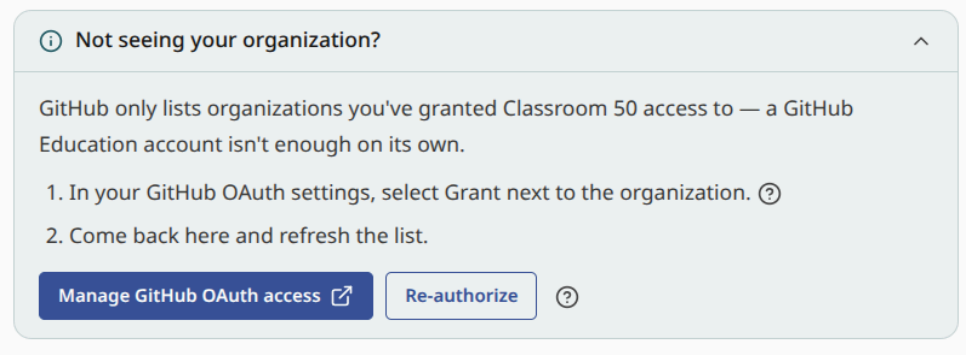
- Choose your organization and upgrade it.



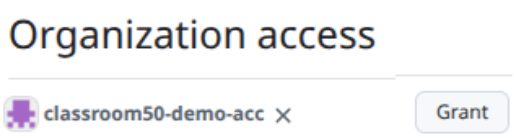
- Go to your organization's **Member privileges** under organization **Settings** and make the following changes:
    - **Uncheck** "Allow members to delete or transfer repositories for this organization."
    - **Uncheck** "Allow members to change repository visibilities for this organization"
    - **Uncheck** "Allow users with read access to create discussions."

    - **Disable** “Allow repository admins to install GitHub Apps for their repositories.”
    - **Disable** “Allow repository administrators to rename branches protected by organization rules.”
    - Set “App access requests” to **Members only** or **Disabled**, whichever you prefer for your students.
    - Set “Projects base permissions” to **No access**.

- Log in to Classroom 50 using your GitHub account. If this is your first time signing in, GitHub will ask you to authorize Classroom 50. This authorizes the Classroom 50 web application to access GitHub on your behalf and registers it as an authorized OAuth application, which you can check later on GitHub.
- Once logged in, you will be at the Classroom 50 organization page (the home page).
- Select **Set up a new organization**.
- Your new organization will not appear until you grant Classroom 50 access to it.
- Select **Not seeing your organization**.
- Select **Manage GitHub OAuth access**.

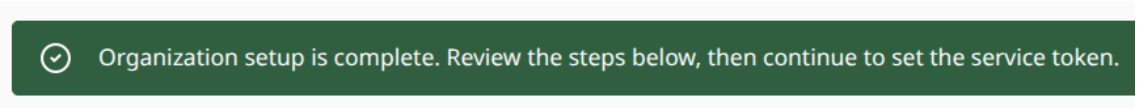


- You will now be on the GitHub organization authorization page.
- Scroll down until you see your organization and select **Grant**.



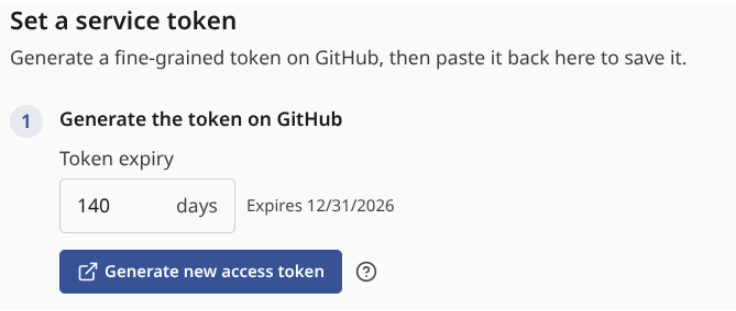
- Return to **Set up a new Classroom 50 organization**. Click **Refresh** if necessary; your organization should now appear. Click **Set up**, and then **Run Setup**.

- Once setup is complete, you should see a success message.

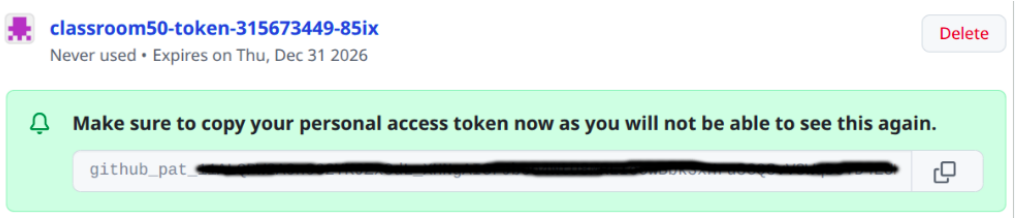


What did Classroom 50 just do? Classroom 50 created a private classroom50 configuration repository in your organization and configured the organization settings needed to manage classrooms, assignments, student permissions, GitHub Pages, and grading workflows. Do not delete or manually repurpose this repository.

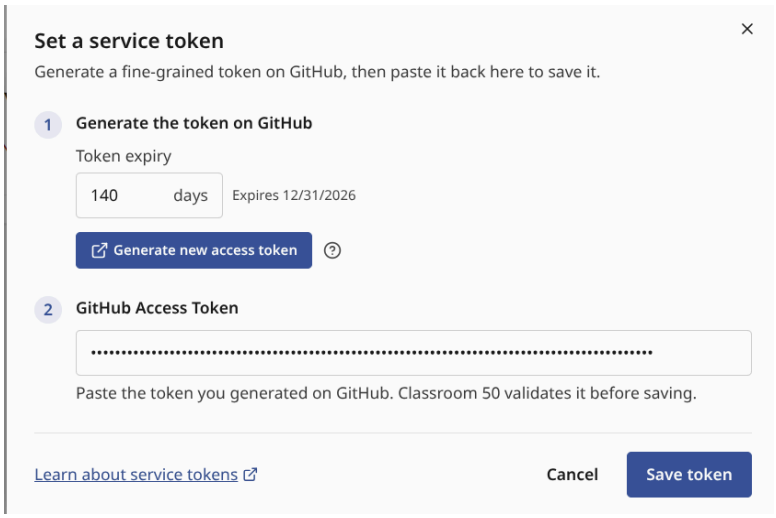
- Click **Next: service token**.
- Click **Set a token**.
- Set a token expiry to cover the end of grading and grading adjustments.



- Click **Generate new access token**. This opens a GitHub page.
- Click **Generate Token** at the bottom of the GitHub page.
- Click **Generate**.
- Copy the token.



- Go back to the **Set a Service Token page** on Classroom 50 (it should still be open) and paste it into the GitHub Access Token field.



- Click **Save token**.
- Click **Done**.
- This will take you to the **My Classrooms** page for your organization.

What is a service token? The service token allows Classroom 50's GitHub Actions workflows to perform background operations, including collecting scores and regrading submissions, when you are not signed in. Classroom 50 stores the token securely as a GitHub Actions secret in the organization's private Classroom 50 configuration repository.

Understanding Classrooms

- A **classroom** is a course-level grouping within a Classroom 50 organization. It is used to organize the students, assignments, and repositories associated with a particular course or course section.
- One GitHub organization can contain multiple classrooms. Under the one-organization-per-semester model used in this guide, you can create a separate classroom for each course or section you teach during that semester. For example:

```
Computer-Science-Fall-2026      ← GitHub Organization
├── Programming Fundamentals I  ← Classroom
├── Programming Fundamentals II ← Classroom
└── Computer Organization       ← Classroom
```

- Each classroom has its own **roster** and **assignments**. Students are associated with the classroom through the roster, and assignments created within that classroom generate the individual or group repositories used for student work.
- A Classroom 50 classroom is not a separate GitHub organization or repository. Classroom 50 stores the classroom configuration and related information within the organization's private classroom50 configuration repository.
- In practical terms, the organization provides the semester-level container, while classrooms provide the course-level organization within that semester (under the assumptions of this guide).
- For ease of management, it is recommended that a "classroom" not be one section of a course but rather all sections of that course; however, one classroom per section is a valid configuration.

Making Classrooms

- From the previous section, **Making Organizations**, you should be on the classrooms page for your organization. If you are not there, go to your organization on Classroom 50 and open it to get to the classrooms page.
- From the Classroom 50 **My Classrooms** page, select **Create Classroom**.
- Enter a **Name** for the classroom. This guide uses the ACC course code as the classroom name because it is concise, familiar to instructors and students, and consistent with the naming convention previously used with GitHub Classroom. For example: **cosc-1336**. Instructors may instead use the full course name or another naming convention that is more useful to them.
- Enter a **Slug** for the classroom. The slug is the machine-friendly identifier Classroom 50 uses in URLs and related naming. This guide uses the same ACC course code for both the classroom name and slug: **cosc-1336**. Using the same value keeps classroom identification simple and consistent. Other naming conventions are possible and acceptable.
- Enter the academic **Term** for the classroom, for example, **Fall 2026**. Although this guide uses one GitHub organization per semester, including the term in the classroom configuration makes the course offering clear when viewed independently.
- Enable **Use an unlisted link for this classroom**. This makes the classroom's published URL difficult to guess rather than using a predictable public URL. An unlisted URL is not the same as a private or secured URL; anyone who obtains the link may still access the published material. This guide recommends using the unlisted option, although instructors may choose otherwise depending on their needs.

Understanding Assignments

- An **assignment** defines the work students will receive and how Classroom 50 will create and manage the repositories used for that work. An assignment belongs to a specific classroom.
- An assignment is not itself a student repository. Instead, it acts as a **configuration or blueprint** for creating student repositories when students accept the assignment.
- An assignment may use a **template repository** containing starter code, instructions, tests, or other files. When a student accepts the assignment, Classroom 50 creates a repository for that student using the template as the starting point. An assignment may also be configured without starter files.
- Assignments may be **Individual** or **Group**:
    - **Individual assignments** create a separate repository for each student.
    - **Group assignments** allow multiple students to share and work in the same repository.
- Classroom 50 can also associate an assignment with optional features such as a **due date, automated grading, feedback pull requests, and submission settings**. These options determine how student work is tested, reviewed, and collected.
- Each assignment provides an **acceptance link** that can be given to students. Students must first be enrolled in the classroom roster. When an enrolled student follows the assignment link and accepts the assignment, Classroom 50 creates the appropriate repository for that student or group.
- In practical terms, the classroom organizes the course, while each assignment defines how a particular programming exercise is distributed, completed, and collected through GitHub.

Making Assignments

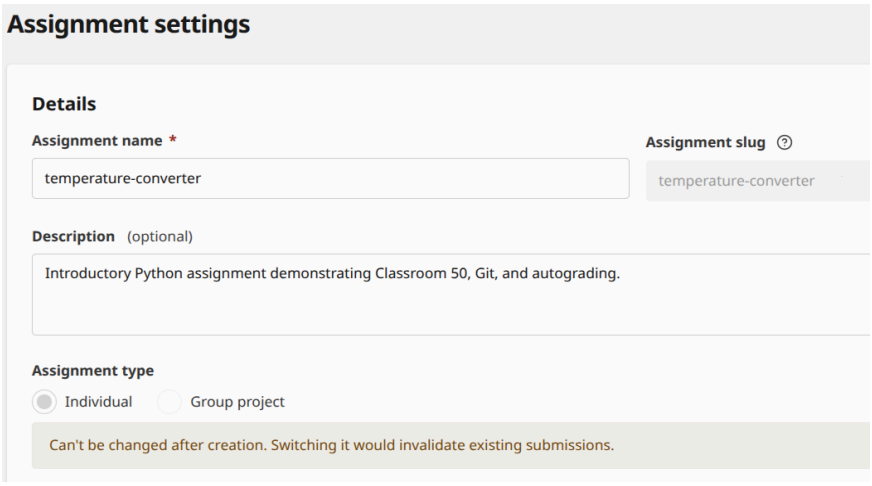
Preparing an Assignment Template

- Before creating a Classroom 50 assignment, prepare the GitHub repository that will be used as the assignment template. The template contains the starter files and instructions students should receive when they accept the assignment.

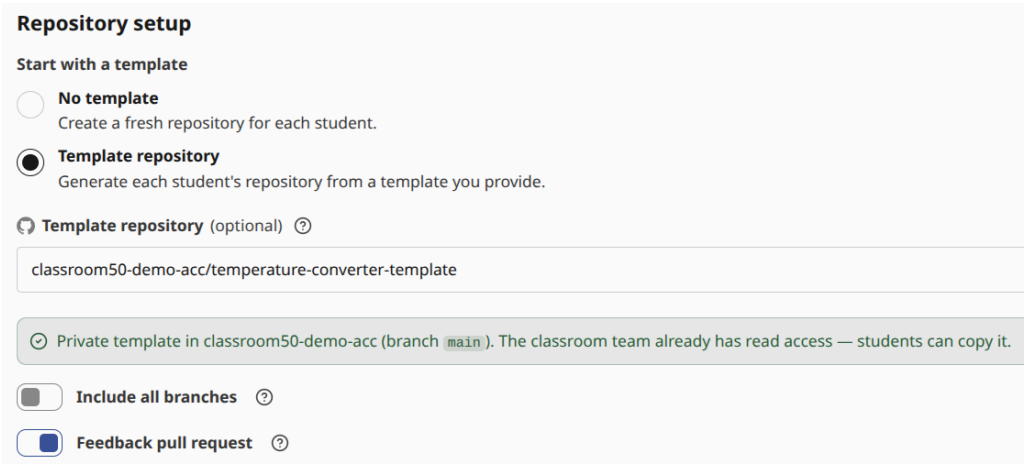
- Create the template repository inside the same GitHub organization used by the classroom (see optional set-up below). This guide uses **temperature-converter-template**
- Make the repository **Private**.
- Do not place solutions, hidden tests, or other instructor-only material in the template repository.
- Enable **Template repository** in the repository settings.
- Add and commit the files students should receive. For this guide, the template contains two files:
    - `ASSIGNMENT.md` contains the assignment instructions. This guide does not use README.md for instructor instructions because students will create their own README.md as part of the assignment. You can view and copy this file from **Appendix A**.
    - `main.py` contains minimal starter code. For this example, it contains only an empty `main()` function with the pass statement and the call to `main()`. You can view and copy this file from **Appendix B**.
- Verify the final starter state before creating the Classroom 50 assignment. The template should contain only ASSIGNMENT.md with the assignment instructions and main.py with the stubbed main() function and its call.
- **Note about template usage:** Assignment template settings can be changed after an assignment is created. However, changes affect only student repositories created after the change; repositories that students have already accepted retain their original starter files and setup.
- **Note about organization location**: You may want to create a separate organization just for templates. If you do this, your templates must be public, but the student repos will still be private. This way you can manage templates separate from each semester's organization.

Creating the Assignment

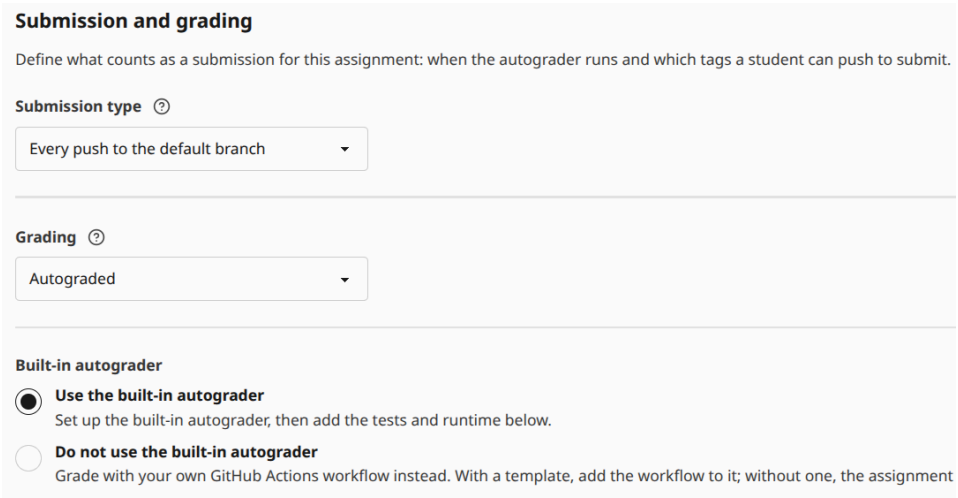
- Return to Classroom 50 and open the classroom in which you want to create the assignment. This guide uses the **cosc-1336** classroom.
- Select **Create Assignment**.



- Complete the basic assignment information. For this guide:
  - **Name:** `temperature-converter`
  - **Slug:** let this automatically fill in unless you have a reason to change it
  - **Description:** a brief description of the assignment
  - **Due date:** leave blank for this demonstration
- **Assignment type: Individual**
- Under **Repository setup**, configure the repository that each student will receive:
  - Select the previously created **temperature-converter-template** repository as the template.
  - Leave **Include all branches** disabled. This example uses only the default branch.
  - Enable the **Feedback pull request** option. This provides a pull request in each student repository that can be used for instructor feedback and code review.
  - Leave the remaining repository options at their defaults for this introductory assignment.



- Under **Submission and grading**:
    - **Note**: Use these options only if you want auto-grading. If you do not, then skip this section; review the assignment settings and **Create Assignment.**
    - Set **Submission type** to **Every push to the default branch**. Each student push will therefore be recognized as a submission and can trigger the autograder. This is useful for incremental development because students receive feedback throughout the development process rather than only after completing the assignment. In the workflow used by this guide, these autograder results serve primarily as a “sanity check” for students as they work; **the instructor determines the final assignment grade separately**.
    - Set **Grading** to **Autograded**.
    - When the additional autograding option appears, select **Use the built-in autograder**.
- Leave the **Advanced settings** at their defaults. This simple Python assignment does not require a custom runner, Docker image, setup command, or other advanced configuration.



- Under **Autograding tests**, add three **Input/Output** tests. For each test, use:
    - **Run command:** `python main.py`
    - **Timeout:** `10` seconds
    - **Points:** `1`
- Configure the first test to check the Fahrenheit conversion:
    - **Name:** Fahrenheit conversion
    - **Input:** `25`
    - **Expected output:** `Fahrenheit: 77.00`
    - **Comparison: Included**

- Configure the second test to check the Kelvin conversion:
  - **Name:** Kelvin conversion
  - **Input:** `25`
  - **Expected output:** `Kelvin: 298.15`
  - **Comparison: Included**
- Configure the third test to check invalid temperatures:
  - **Name:** Reject invalid temperature
  - **Input:** `-300`
  - **Expected output:** `Invalid temperature`
  - **Comparison: Exact**

- Using **Included** for the first two tests allows the assignment to demonstrate incremental development. After the Fahrenheit requirement is completed, the first test can pass even though the Kelvin output has not yet been added. After the Kelvin requirement is completed, both conversion tests can pass. The final validation step should then allow all three functional tests to pass.
- The required `README.md` is intentionally not autograded in this introductory example. Some assignment requirements are appropriate for automated testing, while others are better reviewed directly by the instructor.
- Review the assignment settings and select **Create Assignment**.

Understanding Rosters

- A **roster** is the list of students enrolled in a Classroom 50 classroom. Each classroom has its own roster.
- Students must be on the classroom roster **before they can accept assignments** from that classroom.
- Adding a student to the roster also initiates the process of adding that student to the underlying GitHub organization. Students must accept the GitHub organization invitation before they can work on Classroom 50 assignments.
- Students can be added individually by GitHub username or email address, or multiple students can be added at once using Classroom 50's bulk roster upload.

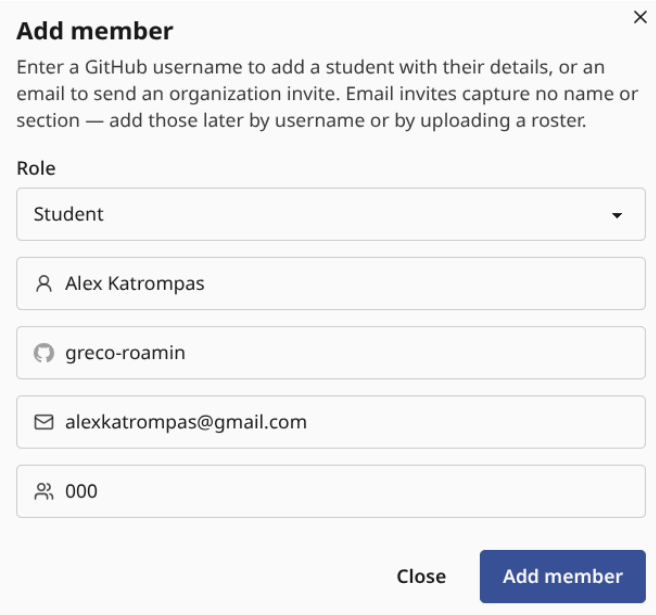
- The roster page shows which students are enrolled in the classroom and whether they have successfully joined the GitHub organization or still have a pending invitation.
- A classroom roster is shared by all assignments in that classroom. Once a student is enrolled, the instructor does not need to add the student separately to each assignment.
- In practical terms, the roster determines **who belongs to the classroom**, while each assignment determines **what work those students can accept and complete**.

Managing Rosters

- Open the classroom and select **Roster** from the classroom menu.
- This guide first demonstrates adding **one student manually** so the enrollment and GitHub invitation process is clear. In normal classroom use, instructors can add many students at once using the **bulk roster upload** described later in this section; students do not need to be entered individually one at a time. Classroom 50 supports both individual and bulk roster enrollment.

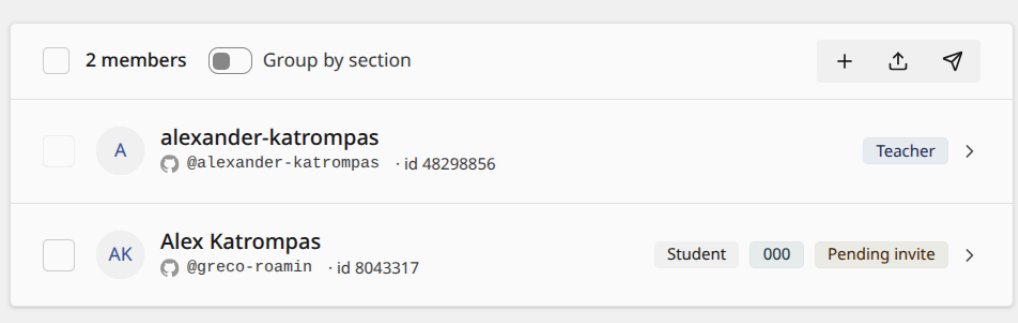
Adding One Student

- On the **Roster** page, select the **+** button to open the **Add member** dialog.
- Leave **Role** set to **Student**.
- Enter the student's information. The form provides:
    - **Name** — optional
    - **GitHub username**
    - **Email** — optional
    - **Section** — optional
- This guide **strongly** recommends adding students by **GitHub username** when possible. The GitHub username directly associates the roster entry with the student's GitHub account. An email address may also be used, but email-only invitations do not initially capture the student's name or section.
- The optional **Section** field can be used to distinguish multiple sections or other classroom groupings. The Roster page can also group students by section.
- Select **Add member**. After the student is successfully added, the form clears so that another student can be entered immediately.



Student Organization Invitation

- Adding a student to the classroom roster also sends that student an invitation to join the GitHub organization. Classroom 50 will show the student on the roster with a **Pending invite** status until the invitation is accepted.
- The student can accept the organization invitation:
    - from the invitation email sent by GitHub; or
    - from the student's GitHub **Organizations** page.
- Students should be informed about **both** methods and encouraged to accept via their Organization page on GitHub, which avoids email delivery failures.
- Students must accept the GitHub organization invitation before they can accept and work on Classroom 50 assignments.
- After the student accepts the invitation, the **Pending invite** indicator disappears from the Classroom 50 roster. The student remains listed with the **Student** role and any section information entered by the instructor.



Adding Students in Bulk

- Classroom 50 supports bulk enrollment using a username list, roster CSV, or email list. This guide **strongly** recommends GitHub usernames whenever possible.

- To add students in bulk, select the **Upload roster** button  on the Roster page and provide the roster using one of the supported formats.
- Bulk roster upload is the recommended approach when enrolling an entire class. Individual entry remains useful for testing, adding late students, or making individual roster changes.
- After a bulk upload, students still must accept their GitHub organization invitations before they can work on assignments. The Roster page can be used to identify students whose invitations are still pending.
- Once students have joined the organization and appear normally on the classroom roster, they are ready to accept assignments from that classroom.

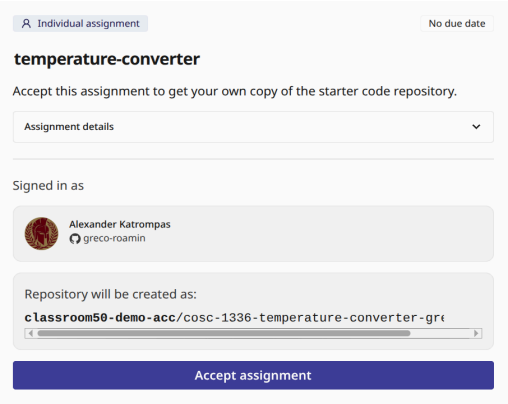
Understanding Assignment Acceptance

- Once a student has been added to the classroom roster and has accepted the GitHub organization invitation, the student can accept assignments from that classroom.
- Each Classroom 50 assignment provides an **acceptance link**. The instructor distributes this link to students through the normal course communication system, such as Blackboard.
- **Recommendation:** Create the assignment normally in Blackboard, but keep the Blackboard instructions minimal and direct students to the Classroom 50 assignment link for the actual programming assignment. The students click the link, accept the assignment, and work from the GitHub repository created for them. When they are ready for grading, they submit their repository link through the Blackboard assignment. This keeps the workflow simple while preserving Blackboard as the official record of submission and grading.
- When a student opens the assignment link while signed in to GitHub, Classroom 50 verifies that the student belongs to the classroom roster and has access to the organization.
- Accepting the assignment creates a new GitHub repository for that student based on the assignment configuration. For an individual assignment using a template, the new repository contains the starter files from the template along with the Classroom 50 files and configuration needed for submissions and autograding.

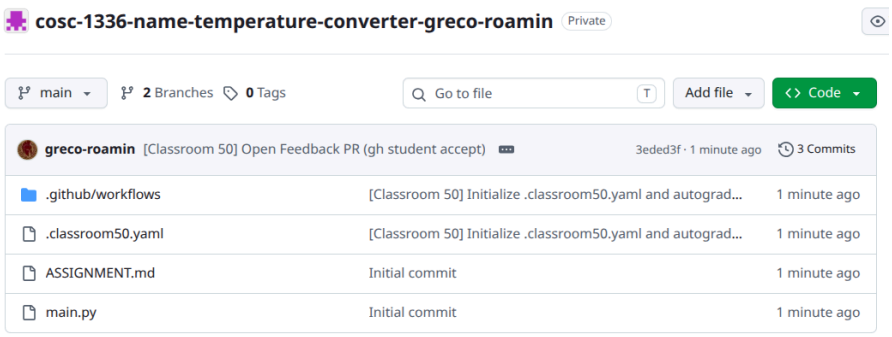
- The student repository is separate from the instructor's template repository. Students work in their own repository by cloning it, making changes, committing their work, and pushing those commits back to GitHub.
- Each student's repository remains associated with the classroom and assignment, allowing Classroom 50 to track submissions, run configured autograding tests, and provide instructor access to the student's work.
- In practical terms, the **assignment acceptance link connects a student to an assignment and provisions the student's working repository**. After acceptance, normal Git and GitHub development begins.

Accepting an Assignment

- The instructor provides students with the Classroom 50 **assignment acceptance link**, normally through the corresponding Blackboard assignment.
- Students should open the assignment link while signed in to the GitHub account associated with the classroom roster.
- **First-time Classroom 50 users:** The first time a student uses Classroom 50, the student may be asked to:
    - sign in to Classroom 50 using GitHub;
    - authorize Classroom 50 to access the student's GitHub account; and
    - open the appropriate Classroom 50 organization and classroom.
- During this **one-time** authorization process, the original assignment link may not be preserved. If the assignment does not appear automatically after authorization, return to Blackboard and open the assignment link again.
- Once Classroom 50 has been authorized, opening the assignment link displays the assignment acceptance page. The student can verify:
    - the assignment name;
    - the GitHub account currently signed in; and
    - the name of the repository that will be created.
- Select **Accept assignment**.



- Classroom 50 will begin provisioning the student's repository. Wait for the setup process to complete before continuing.
- When setup is complete, Classroom 50 displays **Assignment accepted** and provides an **Open repository** button.
- Select **Open repository** to open the newly created private GitHub repository.
- For an individual assignment, Classroom 50 creates a repository using the classroom, assignment, and student GitHub username in the repository name. For example:
  `cosc-1336-temperature-converter-greco-roamin`
- The new repository contains the starter files supplied by the assignment template along with additional Classroom 50 configuration files required for submissions, autograding, and feedback.

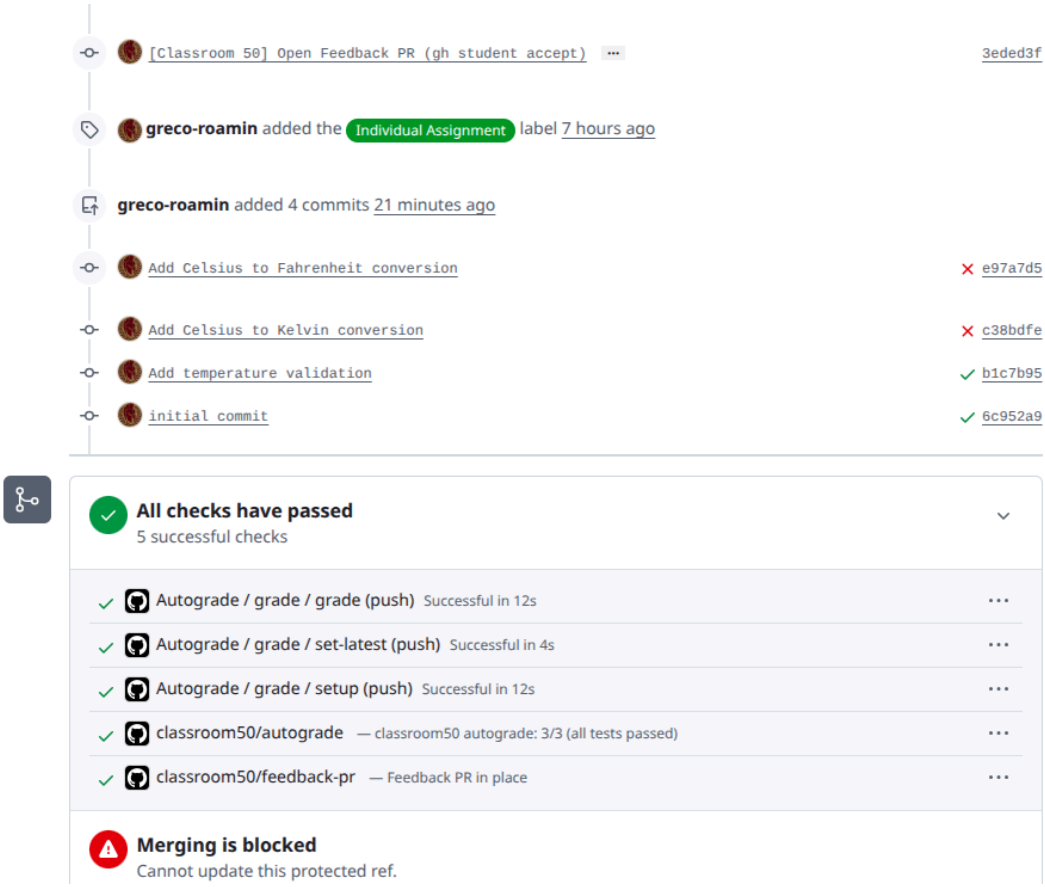


- If a **Feedback pull request** was enabled when the assignment was created, Classroom 50 also creates:
  - the student's working `main` branch;
  - a separate `feedback` branch containing the starter state; and
  - an open **Feedback** pull request that compares the student's continuing work on `main` against the original starter state.
- Classroom 50 also creates several initial setup commits while provisioning the repository. These are **not student development commits** and should not be counted when evaluating the student's development history.
- At this point, assignment acceptance is complete. The student can clone the repository and begin normal Git-based development: edit, test, commit, and push.
- **Note about branches and pull requests:** If Feedback pull requests are enabled, Classroom 50 creates and manages a `feedback` branch and an associated pull request automatically. Students **do not need to work with branches or pull requests** as part of the normal assignment workflow. They can work entirely on `main` using the usual edit → test → commit → push process. Students should not close or merge the Feedback pull request unless instructed

to do so. The Feedback PR exists primarily to provide the instructor with a convenient review and commenting interface. Classroom 50 keeps it updated automatically as students push their work.

Student Development and Autograding

- After accepting the assignment, the student opens or clones the newly created GitHub repository and begins normal development.
- Students should work only on the default `main` branch. If a Feedback pull request was enabled, Classroom 50 manages the `feedback` branch and Feedback pull request automatically. Students do not need to create, switch, merge, or otherwise manage branches or pull requests.
- Develop the assignment incrementally using the normal Git workflow:
  edit → test → commit → push → repeat
- Each commit should represent a coherent, completed, and tested improvement to the program and should use a clear, descriptive commit message.
- Because this guide configured the assignment with **Every push to the default branch**, each push to `main` automatically triggers the Classroom 50 autograder.
- The autograder runs each configured test against the version of the program contained in that push. Students can therefore receive automated feedback throughout development rather than waiting until the entire assignment is complete.
- In the temperature-converter example, the development process naturally progresses through several commits:
  - implement the Celsius-to-Fahrenheit conversion;
  - add the Celsius-to-Kelvin conversion;
  - add validation for temperatures below absolute zero; and
  - create the required `README.md`.
- Earlier commits may fail some autograding tests because later requirements have not yet been implemented. This is normal. Each later push is graded independently against the current state of the program.
- For this example, the three functional tests are designed so that the student can progressively move toward a complete result. Once all three programming requirements are implemented correctly, Classroom 50 reports:
  3/3—all tests passed
- The required `README.md` is not tested by the autograder in this introductory example. It is reviewed directly by the instructor. This demonstrates an important distinction: automated tests can verify appropriate mechanical requirements, while other assignment requirements may still require instructor review.



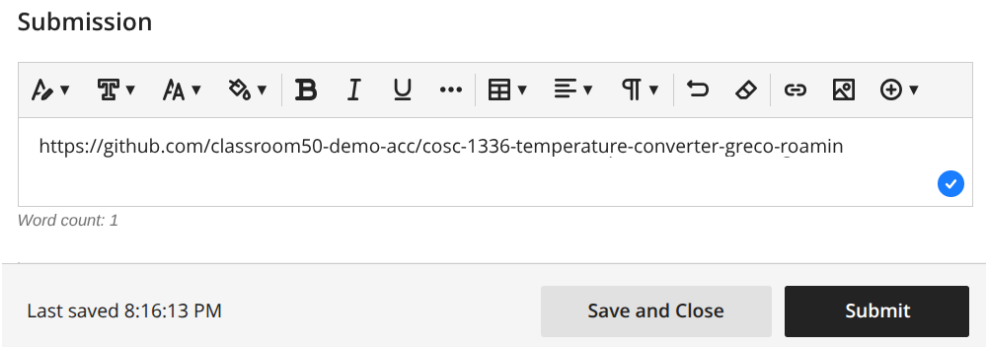
- If a Feedback pull request is enabled, Classroom 50 automatically keeps it synchronized with the student's work:
    - **Commits** show the sequence of commits pushed by the student.
    - **Files changed** shows the cumulative changes made to `main` relative to the original starter state.
    - Autograding results are associated with the corresponding submissions and commits.
    - The instructor can use the pull request to leave general or line-specific feedback.
- Classroom 50 creates several setup commits when the repository is initially provisioned. These initial template and Classroom 50 commits are not student development commits and should not be counted when evaluating the student's development history.
- Students should not close or merge the Feedback pull request unless specifically instructed to do so. Classroom 50 maintains it as part of the instructor feedback and grading workflow.

- Once the student has completed the assignment, tested the final program, committed and pushed all required work, and reviewed the autograding results, the repository is ready for submission according to the instructor's course submission procedure.

Student Submission and Instructor Grading

Student Submission

- Once the student has completed the assignment, tested the program, committed and pushed all required work, and reviewed the autograding results, the student is ready to submit the assignment for instructor grading.
- In Blackboard, the student opens the corresponding assignment and pastes the **normal GitHub repository URL** into the **Submission** text box. For example: `https://github.com/organization/repository`
- Use the normal repository web URL, **not** the `.git` or SSH clone addresses. The normal URL provides a useful clickable record in Blackboard and can also be used to clone the repository.
- The student then selects **Submit**. Blackboard records the submission and notifies the instructor that the assignment is ready for grading.



- Classroom 50 and GitHub contain the actual programming work and development history; Blackboard serves as the institutional record that the student has submitted the assignment for grading.

Instructor Grading

- Open the repository on GitHub from the URL submitted in Blackboard.
- Review the student's work using the information available through GitHub and Classroom 50 as appropriate for the assignment. This may include:

    - the completed source code and other required files;
    - the commit history and commit messages;
    - Classroom 50 autograding results;
    - the Feedback pull request and cumulative **Files changed** view; and
    - any other assignment requirements that require instructor review.
- When local execution or additional testing is useful, clone the repository using the same submitted URL:

  ```
  git clone https://github.com/organization/repository
  ```

- Inspect and run the program locally as needed.
- Determine the final assignment grade according to the course grading criteria and enter the grade and any appropriate feedback in Blackboard.
- Blackboard remains the official submission and grading record, while GitHub and Classroom 50 provide the repository, development history, automated testing results, and other evidence used during grading.

Instructor Workflow Suggestion

- In the workflow used for this guide, a Blackboard submission primarily serves as the student's signal that the repository is ready for grading. Students may continue committing and pushing corrections until the instructor begins grading; the instructor grades the repository state that exists when grading begins.
- It is recommended that you capture the commit SHA when you begin grading and record that as part of the student’s grade feedback. This is a record of the point where grading begins so there is no confusion as to which version was graded.
- Instructors may also allow additional Blackboard attempts for revised or resubmitted work. These are course-policy decisions and are not required by Classroom 50.

Appendix A: ASSIGNMENT.md

````
# Temperature Converter

## Objective

Create a simple Python program that converts a temperature entered in degrees Celsius to Fahrenheit and Kelvin.

Develop the program incrementally. Complete, test, commit, and push each meaningful portion of the assignment as you work.

## Starter Code

The repository contains:

- `ASSIGNMENT.md` – these assignment instructions

- `main.py` – starter Python code containing an empty `main()` function

Complete the program in `main.py`.

## Requirements

### 1. Celsius to Fahrenheit

Read one temperature in degrees Celsius from standard input.

Convert the temperature to Fahrenheit using:

```text

F = (C × 9/5) + 32

```

Display the Fahrenheit temperature using the following format:

```text

Fahrenheit: 77.00

```

Display the result rounded to two decimal places.
````

Test the program, then **make an appropriate Git commit** and push the working version to GitHub.

2. Celsius to Kelvin

Add a conversion from Celsius to Kelvin using:

```text

K = C + 273.15

```

The program should now display both conversions:

```text

Fahrenheit: 77.00

Kelvin: 298.15

```

Display both results rounded to two decimal places.

Test the new functionality, then **make an appropriate Git commit** and push the working version to GitHub.

3. Validate the Temperature

A Celsius temperature cannot be below absolute zero (`-273.15` °C).

If the entered value is below `-273.15`, do not perform or display either conversion. Instead, display:

```text

Invalid temperature

```

Test both valid and invalid temperatures, then **make an appropriate Git commit** and push the working version to GitHub.

4. Create a README

Create a `README.md` file for the completed project.

````
The README should briefly explain:

- what the program does;

- how to run it;

- what input the program expects; and

- what output the program produces.

**Commit and push the completed README.**

## Example Runs

Input:

```text

25

```

Output:

```text

Fahrenheit: 77.00

Kelvin: 298.15

```

Input:

```text

0

```

Output:

```text

Fahrenheit: 32.00

Kelvin: 273.15
````
````

```

Input:

```text

-300

```

Output:

```text

Invalid temperature

```

Development and Version Control

Do not complete the entire assignment and submit it as a single commit.

Develop the program incrementally. Each commit should represent a coherent, completed, and tested improvement to the project and should include a clear, descriptive commit message.

NOTE: For this work, you should have completed at least 4 commits for your commit history matching the steps above.
```

Appendix B: `main.py`

```
def main():
    pass


if __name__ == "__main__":
    main()
```

Appendix C: Beyond Basic Setup

The workflow in this guide is intended to provide a straightforward starting point for using GitHub Education and Classroom 50 in a programming course. Once the basic workflow is established, instructors can extend it in many ways depending on their courses, students, and teaching objectives.

- **Use Git as part of the learning process, not only as a submission mechanism.** Encourage students to develop incrementally and make meaningful, atomic commits with descriptive commit messages rather than completing an assignment first and uploading it afterward.
- **Use autograding as rapid feedback.** Automated tests are especially useful for giving students immediate feedback as they work. Autograding does not need to represent the complete assignment grade; instructors can separately evaluate design, documentation, testing, the development process, and other course objectives.
- **Keep the student Git workflow as simple as appropriate.** Students can work entirely on the `main` branch using the basic edit → test → commit → push workflow. Classroom 50 can manage additional infrastructure, such as Feedback pull requests and the `feedback` branch, without requiring students to work directly with those features.
- **Use Feedback pull requests as an instructor tool.** Feedback pull requests provide a convenient view of the student's commits and cumulative changes from the original starter code and allow instructors to leave general or line-specific comments.
- **Require appropriate repository artifacts.** Depending on the course, completed repositories may include a useful `README.md`, `.gitignore`, documentation, tests, or other files expected in professional software projects.
- **Consider AI-aware assignment design.** If students are permitted or expected to use generative AI, assignments can place greater emphasis on decomposition, incremental development, testing, verification, documentation, and demonstrated understanding rather than treating code generation alone as evidence of learning.
- **Explore more advanced Classroom 50 features as needed.** Classroom 50 supports additional assignment, grading, group work, submission, and automation options that are beyond the introductory workflow demonstrated here.
- **Classroom 50 also provides a command-line interface (CLI).** The web application is sufficient for the workflow in this guide, but the CLI may be useful for instructors who want additional scripting, automation, auditing, or bulk-management capabilities.

This guide intentionally focuses on getting instructors operational with GitHub Education and Classroom 50. More advanced programming pedagogy, AI-enabled instruction, repository standards, and complex assignment design are best treated separately once the basic classroom infrastructure is working.

References

The following resources provide additional information about GitHub Education, Classroom 50, Git, and GitHub. Classroom 50 is actively developed, so instructors should consult the official documentation when current behavior differs from this guide.

- Classroom 50
- Classroom 50 GitHub Repository
- Official Classroom 50 Web Teacher Guide
- Official Classroom 50 CLI Teacher Guide
- GitHub Education
- GitHub Education for Teachers
- GitHub Documentation
- Git Documentation
- ACC Git and GitHub Student Tutorial and Reference (coming soon)
- ACC Version Control Introduction Video (coming soon)
- ACC Instructor Resources
- Additional Classroom 50, GitHub, and course-specific resources may be added as they are developed.