Guide to Use Gradescope

Table of contents

Summary

Gradescope is an online grading tool for paper-bases and programming assignments. This document lists steps for common tasks. Please refer to vendor documents for details and most updated information. Gradescope provides very complete tutorials and documentations. Useful links are listed below.

- Company website: https://gradescope.com/
- Quick start videos: https://gradescope.com/get_started
- Complete help information: https://gradescope.com/help
- Autograder technical guide: https://gradescope-autograders.readthedocs.io/en/latest/
- Develop roadmap https://trello.com/b/36UN761q/gradescope-roadmap

Gradescope help page categorizes instructions into Courses, Assignments, and Students, and provides instructions to corresponding workflow.

- Course Workflow
- Assignment Workflow
- Student Workflow
- Frequently Asked Questions

Autograder technical guide details the autograder specification and describes the steps to setup programming assignments.

Set up a course

- Create an account as an instructor at company homepage https://gradescope.com/
- Create a new course
- Adding students and staff (TA)
    - Course members can be imported from a CSV file
    - If integrated with a Learning Management System (LMS), course roster can be Synced to the LMS roster
    - Alternatively, students can add themselves to a course via course code
- Set staff permission (Gradescope has three staff roles: Instructors, TAs, and Readers)
- Setup assignments and exams
- Manage grades
    - If integrated with an LMS, grades can be published to the LMS

Create and grade paper-based assignments

Exams and quizzes are normally instructor submitted, and are fixed length; homeworks are student submitted and are normally variable length. Check assignment workflow for details.

- Exams & quizzes
    - Create an assignment, choose instructor as "Who will upload submissions" option (Creating, editing, and deleting an assignment)
    - Create the assignment outline and mark the question regions on a template PDF (Creating an outline),
    - Upload and process scans (Managing scans), match student names to submissions (Managing submissions)
    - Grade student work with flexible, dynamic rubrics (Grading)
    - Publish grades and email students (Reviewing grades), export grades (Exporting Grades) and manage regrade requests (Managing regrade requests)

- Homeworks
    - Create a "Student" uploaded assignment, set release date, due date, and group submission policy etc (Creating, editing, and deleting an assignment)
    - Create the assignment outline (Creating an outline). For student-submitted variable-length assignment, you only need to list all of your questions/subquestions and assign point values. Students will mark where their answers are on their submissions (Submitting an assignment)
    - You can begin grading as soon as a single submission is uploaded, and you can view all student-uploaded submissions from the Manage Submissions tab.
    - The rest of the workflow is the same as Exams & Quizzes: you can publish grades, email students (Reviewing grades), export grades (Exporting Grades), and manage regrade requests (Managing regrade requests).

Create and grade programming assignments

Gradescope supports autograding and manual grading for programming assignments. Gradescope tests student code in Docker containers hosted on Amazon Web Services (AWS). Instructors can provide setup script, testing script and supporting files (Autograder) for each assignment, or use manual Docker configuration to provide an customized Docker image.

Students can drag/drop files to Gradescope. Upon submission, student's code will be automatically tested and the student can receive immediate feedback.

Create a programming assignment (with Autograder)

- Follow Gradescope instructions to Setup Programming Assignment
- Create an autograder according to Autograder Specifications
    - Autograder should be in zip format. It must contain at least two files in the root:
        - **setup.sh**: a setup (Bash) script that installs all your dependencies
        - **run_autograde**r: an executable script, in any language that compiles and runs your autograder suite and produces the output in the correct place
    - Autograder output needs to be in results.json (results .json specification)
- Upload the autograder zip file to build the Docker image
- Upload code to test the autograder
- Debug code if necessary (Debug via SSH)
- Setup rubric for manual grading

Technical details and Troubleshooting

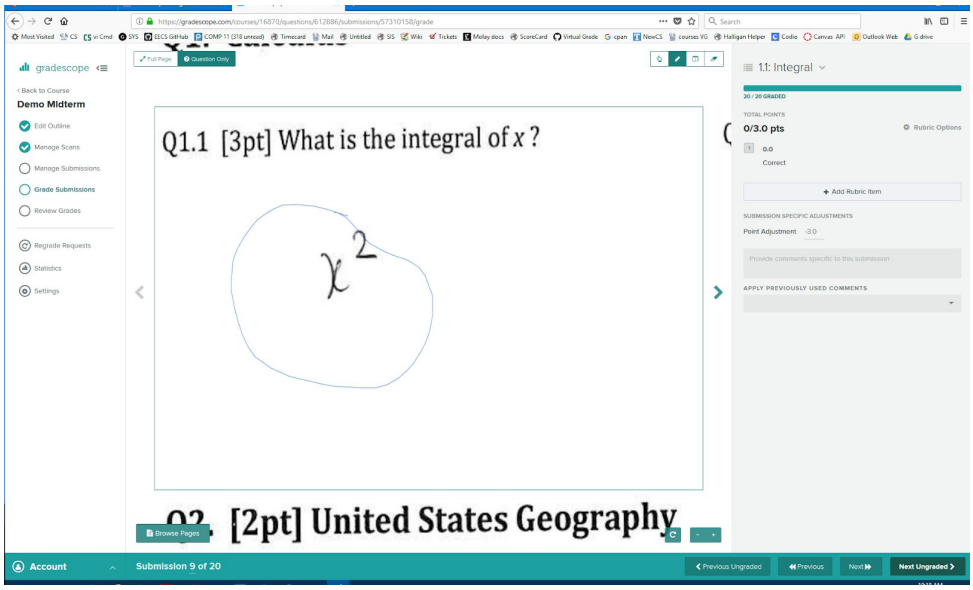
- The default image is based on Ubuntu 16.04
- Default CPU setting is 0.25 CPU 384MB, default timeout is 20 minutes.

Frequently Asked Questions

a. How can students submit code to GradeScope?
   - Files can be drag-dropped to Gradescope or uploaded from github and bitbucket

b. How can I detect or stop runaway (infinite loop) programs?
   - You can set a timeout in your test script
   - Gradescope has a default overall timeout of 20 minutes (contact Gradescope if you need longer time)
   - The autograder will be “killed” if it exceeds the memory limit for a single container. The default memory limit is 384MB, but you can increase this in the "Advanced Settings" of your assignment settings
   - Refer to Troubleshooting for more details

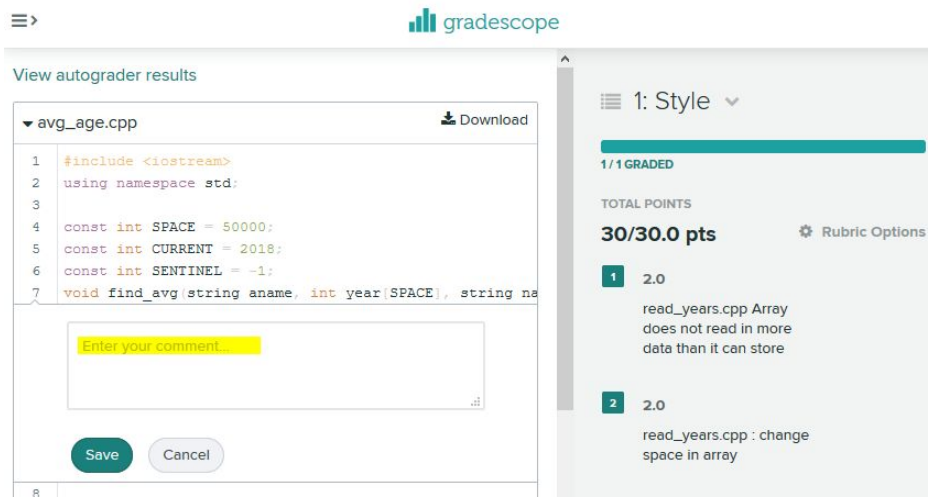
c. Is pen-based inking supported?
   - Inking feature is available in grading paper-based assignments (PDFs), click on the pen icon at the top of a grading screen to use this feature



d. How can I grade code manually?
   - Enable manual grading in assignment settings
   - Click on “Grade Submission”
   - Click on each file link to view code

e. How can I grade program output manually?
   - Print program output to console in run_autograder script
     (It is better to display assignment output in console than in results.json. The output often contains invalid json characters and may corrupt the json file.)
   - Create corresponding rubric items for manual grading.
   - When grading submission click on “View Autograder Results” to view Autograder output
   - Set “stdout_visiblibity” to “visible” in results.json to display the output to students

f. How can I do inline comments?
   - When grading code manually, click on a line of code, a comment box will appear, the grader can enter and save comments



Examples

Sample programming assignments setup

I tested Gradescope programming assignment grading feature with some comp11 assignments. The goal was to 1) provide a similar workflow for students 2) mimic the functions of our current systems and 3) reuse existing test scripts.

With Gradescope:

- Students submit assignments and view grades from the same website
- Autograding mimics Provide screening feature; autograder reuses the existing testing code
- Some items can be graded by the Autograder
- Manual grading mimics Scorecard feature; rubrics need to be recreated in Gradescope

The detailed comparison for assignment related workflow is listed below:

Steps	Current	Gradescope
Students submit assignment	Use Provide	Use Gradescope
System check required files (all assignments)	Provide screening script	Autograder script
Automatic testing for comp11 hw4,5,6	Testing script called at the the end of the Provide screen script Console output (including testing results and program output) redirected to the provide log file	Autograder script(s) Test results (pass or fail, and score) in results.json; Console output in Autograde output
Graders grade assignments	Scorecard ~/molay/grading/11	Manual grading in Gradescope
	Grader can view code and Provide log files	Grader can view code and Autograde output.
Students view grades	~/molay/g11	Gradescope

Autograder for HW 4, 5, 6 C++ assignment

I created Autograder for comp 11 HW 4, 5, 6 in a similar fashion. Since comp11 assignments are not OS dependent and the code is able to run on Ubuntu, Gradescope default container is used. I created Autograders and did not manually setup Docker images.

You can check Autograder file structure from this sample (Sample autograder for comp11_hw5). The following table explains the purpose if each file/folder and how it is created. (Note: the root folder is /comp/11/grading)

Files & Folders	Purpose	Notes
setup.sh	Install dependency to the Docker image	comp 11 uses clang++; our system has version 3.7
run_autograder	A master script to run the autograder suit and produce output	Testing mechanism similar to ../tests/[HW]/bin/checkfiles
check_files.py	Check required files Called by run_autograder	Similar to ../screening/testsets/[hw]
Makefile	Make file	Copied from ../tests/[hw]/makefile/Makefile (may require minor change from clang++ to clang++-3.7 because of the Container setup)
timeout	Set timeout for each test	Copied from ../gtools/timeout
input	A folder contains inputs for testing each program	Copied from ../tests/[hw]/input
desired_output	A folder contains expected output to be compared with student' code output	Copied from ../tests/[hw]/output
[Other files]	Other files needed for testing	Check ../tests/[HW]/bin/checkfiles for testing logic

Sampe results.json generated by the Autograder

{"stdout_visibility":"visible",
"output": "All files submitted",
"tests":[
{"score":10,"max_score":10, "output":"Compile Success.", "visibility":"visible"},
{"score":0,"max_score":5, "output":"minesweeper already_revealed failed. ",
"visibility":"visible"},
{"score":0,"max_score":5, "output":"minesweeper bounds_check failed. ",
"visibility":"visible"},
{"score":5,"max_score":5, "output":"minesweeper corners passed.", "visibility":"visible"},
{"score":5,"max_score":5, "output":"minesweeper lose_eventually passed.",
"visibility":"visible"},
{"score":5,"max_score":5, "output":"minesweeper losing passed.", "visibility":"visible"},
{"score":5,"max_score":5, "output":"minesweeper reveal8 passed.", "visibility":"visible"},
{"score":5,"max_score":5, "output":"minesweeper reveal_border passed.",
"visibility":"visible"},
{"score":5,"max_score":5, "output":"minesweeper winning passed.", "visibility":"visible"}
]
}