Canvas codecheck grading Documentation

Release 2.3.0

Daniel Mai

Aug 29, 2018

Contents

Canvas codecheck grading solves the problem of tediously grading codecheck files with Canvas's point-and-click interface. Instead of downloading each of the codecheck files individually, extracting them one-by-one, looking at the report and grading the files by hand, you can simply edit a single text file locally on your computer.

All you need to grade a codecheck submission is to give a comment in a plain text document that's generated by the script, such as the snippet shown below:

```
_dir: lastname--firstname_1234567
_name: Lastname, Firstname
_canvas_id: 1234567
_total_score: 18/18
_comment:
Codecheck score: 18/18

Here are where you put your comments, such as...

Clever logic with the if statements!
Try to remove those variables you didn't use at all.

Or modify the score like:
-1 points for nonprivate instance variables
^
this line beginning with '-1' will deduct 1 point from the 18 points
shown above. Now the student will receive 17 points on this assignment.
Every line that doesn't start with an integer will just be a comment.

!!This section will be shown to the student!!

_notes:


If you need to modify the score without showing it to the student, put them here,
like so:

-1 points for copying from someone else
^
this line beginning with '-1' will deduct 1 point from the 18 points
shown above. Now the student will receive 16 points on this assignment.

Every line that doesn't start with an integer will just be a note.


--------
```

CHAPTER 1

Project dependencies

- Python (2.7 or 3.3) on your system.
- Canvas information (an access token and the course ID), which you will add to the configuration file to this script.
- Java JDK (for the `jar` and `jarsigner` commands)

CHAPTER 2

Features

- Automatic grading/point weighing with the codecheck files
- Easy grading via a single plain text document.
- Quickly open mulitple submitted codecheck report files.
- Uploading grades and comments to Canvas

CHAPTER 3

Code

The code for this project can be found at the Bitbucket repository.

CHAPTER 4

Documentation Index

4.1 Initial setup

4.1.1 Python

Python needs to be installed on your system. You can download Python at the official download page. They should be compatible with any Python 2.7.x and Python 3.x versions. The scripts were created using Python 2.7.5. I've tested these scripts with Python 3.3 as well, and they seemed to work just fine.

4.1.2 JDK tools

You need to have the JDK installed and the command-line tools on your path, because the scripts use the `jar` and `jarsigner` commands. On Windows, refer to instructions at Oracle's documentation to set up your command-line path to have the Java tools ready. On Linux and OS X, the tools should automatically be found on your path.

4.1.3 Canvas access token

You must create a Canvas access token in order to grade assignments. This allows you to submit grades and comments for homework under your account (so long as you have permission to, which you should have because you're a grader). An access token can be created at the bottom of your Canvas settings and clicking on the blue "+ New Access Token" button, which is shown below.

Note: An access token is like a password, so keep it away from others, as it allows anyone to use the Canvas API under your account. You only need one access token for all of your classes because the token is for your whole account, not tied to any particular account. So if you're grading for multiple courses with this script, you can use the same access token for all of them.

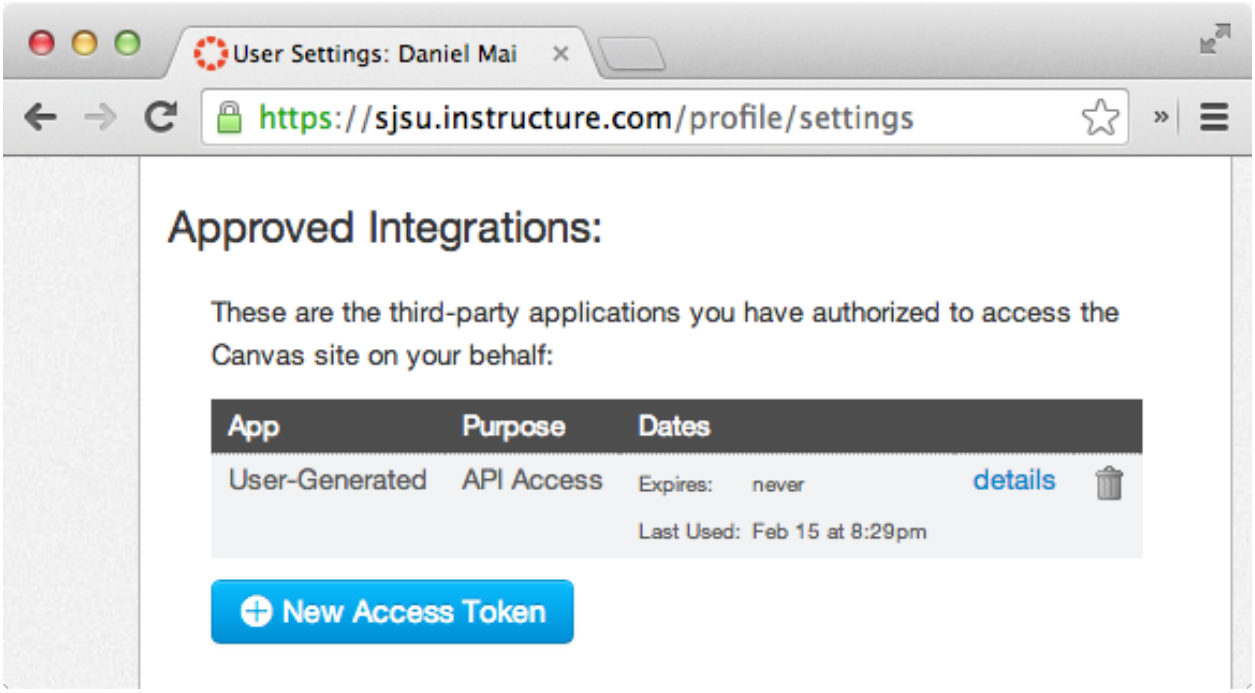

Fig. 1: Above: The access token section of the Canvas setting page

4.1.4 Download the scripts

You can download the scripts by cloning the repository with the following command:

```
git clone https://bitbucket.org/danielmai/code-check-homework-grading.git
```

By cloning the repository, it will be easier to get the latest versions of the scripts with a `git pull`. Remember to pull the latest scripts from the repository before grading.

4.1.5 Modify `config.json`

There is a `config.json` file that contains information necessary to make calls to the Canvas API. It should contain the correct information for the following fields:

- `access_token`
  - The API access token for Canvas. To create one for yourself, see *the Canvas access token section (above)*.
- `course_id`
  - A string of digits representing the Canvas course ID. It can be found from the URL of the course page on Canvas (the only string of digits in the URL).

Note: Make sure you have the correct course ID for the course you are grading for **this semester**. If you used this script for a previous course, you may have kept the same information relevant to your previous course.

If you have the wrong information in your config file, the script may appear to be working just fine, but bad things **will** happen (like uploading the grades with the same assignment name *to the wrong course*), so please double check this.

- `course_name`
  - (Optional) This is used as a label in the config file so you know what course the config information is pertaining to.
- `host`
  - The host for Canvas. This will be specific to the university's Canvas installation. SJSU Canvas's host is `sjsu.instructure.com`.

You can modify the config.json file directly, or by using `grade_config.py` (but modifying it directly is probably the way to go—it's just a JSON file). If you want to use the provided grade_config.py script, use `grade_config.py --help` for instructions.

4.1.6 If you are a past grader, or grading for multiple classes

There is only one `config.json` file that the script will read from, so you must make sure that the correct information is in your configuration file (if that didn't become apparent in the note above).

If you are grading multiple classes at the same time, then you should make a copy of the config file and manage them yourself. When you use the script, make sure you are using the correct config file.

4.2 Overview

Here's an overview of the scripts and the grading workflow.

4.2.1 Scripts

The three scripts that you will use are:

- `grade.py`
  - Organizes the submission files and generates the grading files for you. For more information, see the *Grading structure* section.
- `open_reports_in_browser.py`
  - Opens mulitiple codecheck report files at once for quickly viewing the reports while grading. For more information, see the *Open reports in the browser* section.
- `grade_upload.py`
  - Parses the grades.txt generated by `grade.py` and uploads the data to the assignment on Canvas. For more information, see the *Uploading the grades* section.

4.2.2 Canvas API

This script communicates with Canvas through the Canvas API with the following API calls:

- List assignments
  - In `grade.py`, this is used to get a listing of all the assignments in the course to find the match with the assignment name that is passed in when you run the script. It will grab the assignment ID and the assignment's maximum score from Canvas.
  - In `grade_upload.py`, This is used to get the Canvas assignment ID of the assignment that matches the assignment name that you pass to the script.

- Grade or comment on a submission
  - In `grade_upload.py`, this is used to upload the scores and your comments for each student written in the `grades.txt` file produced by this script.

4.2.3 Workflow

The overall grading workflow is as follows:

1. Download the submission files from Canvas. There should be a "Download Submissions" link on the righthand sidebar on the assignment page on Canvas.
2. Extract the files to a new directory.
3. Run the `grade.py` script on the directory. The typical command looks like:

   ```
   python grade.py "The Assignment Name on Canvas" path/to/submissionfiles
   ```

   Where `"The Assignment Name on Canvas"` is the name of assignment name you want to grade, and `path/to/submissionfiles` is path to the files that you extracted to in Step 2 above.
4. Grade the files by looking at the report files for each student with the `open_reports_in_browser.py` script. For more detail about this script, see the *Open reports in the browser* section.
5. When you're done grading, upload the grades.txt file to Canvas. The typical command looks like:

   ```
   python grade_upload.py path/to/grades.txt
   ```

   Where `path/to/grades.txt` is the filepath to the grades.txt file.

4.2.4 What to watch out for

These grading scripts don't do everything, but it tries to catch some common problems.

What the script detects

The script automatically detects the following issues and informs you if anything was found in the `_notes_and_score_changes` section of grades.txt:

- **Duplicate submissions**, such as two identical submissions of the same part. For example, If an assignment had students submit a `Robot.java` file for one of the parts, and a student submitted *two* codecheck reports for `Robot.java`, then the script will only count one of them. In grades.txt, this problem will be described as "*javafile* already submitted."
- **Draft/Final submissions**. Usually this problem happens when students submit draft files for a final homework. First, the script counts all of the submissions that are "draft" level and "final" level. The script then chooses the level with the most submissions (because most students are going to submit the correct files), a la the-majority-is-probably-correct. In grades.txt, this problem will be described as "Incorrect problem level".
- **Irrelevant submissions**. Just as the draft/final homework level is detected with the majority-is-probably-correct heuristic, the irrelevant submissions are also found the same way. Every codecheck submission has a problem ID, so all of them are counted, so the top three (or, more generally, the top number-of-parts-for-the-assignment) IDs are chosen for the assignment. Problem IDs that are not part of this top list are not scored. In grades.txt, this problem will be described as "*javafile* is not part of this assignment".

- **Cheating**. If multiple students submitted the same codecheck submissions, then they will not be given credit for those submissions. In grades.txt, this problem will be described as "This exact codecheck submission has been turned in by another student. Submission ID = *xxxxxx*", where *xxxxxx* is the submission ID. You can search for *xxxxxx* in the same grades.txt file to see the other students who have submitted the same files.

What you should look for

- **Code structure**. This includes the logic of the code, instance variables, naming, and the like. These scripts don't check for style, code logic, hard coding, etc.

4.3 Detailed usage

Here's a detailed page on what happens to the files when you use these scripts.

4.3.1 Structure

Important: The script should only be run on a fresh directory of submission files from Canvas. In other words, don't run the script more than once on the same set of data. If you want to redo the script again for the same homework submissions, start over from the beginning with the initial set of submission files untouched.

For help and more options for running the script, run `python grade.py --help`

```
$ python grade.py --help

usage: grade.py [-h] [-v] [-n NUMBER_PARTS] [--no-verify] [--no-checks] [-r]
                assignment_name directory

positional arguments:
  assignment_name       The assignment name. Must match an assignment found on
                        Canvas.
  directory             the directory containing subdirectories of student
                        submissions.

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         print verbose output to stdout.
  -n NUMBER_PARTS, --number-parts NUMBER_PARTS
                        provide the number of parts for this assignment
  --no-verify           skip jarsigner verification and unarchive all
                        submission files
  --no-checks           Don't check for warnings/cheating submissions (checks
                        by default)
  -r, --remove-existing
                        removes grading files from the directory from a
                        previous run
```

The grading script requires a directory of submissions from Canvas.

To grade an assignment, download the .zip file from Canvas and extract its contents to a directory. The directory structure may look something like

```
directory_with_submissions
|
|--studentname1_studentid1_submissionid1_filename1.ext
|--studentname1_studentid1_submissionid2_filename2.ext
|--studentname1_studentid1_submissionid3_filename3.ext
|--studentname2_studentid2_submissionid4_filename4.ext
|--studentname2_studentid2_submissionid5_filename5.ext
|--studentname3_studentid3_submissionid3_filename.ext
+--...
```

where `ext` is either `signed.zip`, `zip`, or `jar`. (Note: There were three possible extensions with codecheck because it first gave out `jar` files, and then `zip` files. Now, codecheck only provides students with a `signed.zip` file).

To run the script on the directory, the command will be something like

```
python grade.py "Assignment Name" directory_with_submissions
```

Where `"Assignment Name"` matches the name of the assignment on Canvas. Don't forget to enclose the name in quotation marks if it contains spaces.

When the script runs on a submissions directory, it organizes the directory by separating each student's submissions to an individual directory. So the directory tree above will become

```
directory_with_submissions
|
|--studentname1_studentid1
| |
| |--studentname1_studentid1_submissionid1_filename1.ext
| |--studentname1_studentid1_submissionid2_filename2.ext
| +--studentname1_studentid1_submissionid3_filename3.ext
|
|--studentname2_studentid2
| |
| |--studentname2_studentid2_submissionid4_filename4.ext
| |--studentname2_studentid2_submissionid5_filename5.ext
|
|--studentname3_studentid3
| |
| +--studentname3_studentid3_submissionid3_filename.ext
|
+--...
```

Then, the script will extract the files to their own directory (with the same name as the .ext file). If the `--no-verify` option is specified, then all files will be extracted. Otherwise, only those that are verified by `jarsigner` will be extracted.

Once all the files are extracted and in their own directories, the script will begin totaling the codecheck scores from the `report.html` files and writing the scores to a file called `total_grade.txt`. Then, the script will walk through the directory tree and aggregate the scores from the `total_grade.txt` files in each student directory into a file called `grades.txt`, which will be found at the root of `directory_with_submissions`.

4.3.2 Summary of the structure

Each student's submissions are moved to his or her own directory named `lastname--firstname-miscname_canvasid`. Then, each codecheck file is unarchived to its own directory. Each student will have a `total_grade.txt` file found in his or her own directory, and a `grades.txt` file will be found in the `directory_of_submissions` (the directory you ran the script with).

4.3.3 Aggregate Java files

Also, the script aggregates the java files and the report files. The script will aggregate all of the java submission files into one file called `aggregate_java_files.txt` *in each student directory* for easier viewing of the source files that a student as submitted. Here's an example of an aggregated file for a student:

```
public class StringDemo
{
   public static void main(String[] args)
   {
        String word = "surprise"; //do not change this line

        System.out.println(word.length());

   }

}
/**********************************************/
public class TextDemo
{
     public static void main(String[] args)
    {
          Text message = new Text(10, 50, "Hello, World!");
          message.draw();

    }
}
/**********************************************/
public class Rainbow
{
    public static void main(String[] args)
    {
          Rectangle box = new Rectangle(0, 0, 100, 20);
          box.setColor(Color.RED);
          box.fill();

    }
}
```

A `/**********************************************/` is the separator between different files.

4.3.4 Aggregate report files

The `report.html` files will be aggregated into one `aggregate_report.html` file per student. These aggregated files are useful to see the reports of one student's homework on one page instead of a page for each part of the homework.

After the above steps, the directory structure will become

```
directory_with_submissions
|
|--grades.txt
|
|--studentname1_studentid1
| |
```

(continues on next page)

(continued from previous page)

```
| |--studentname1_studentid1_submissionid1_filename1
| | |
| | +--(codecheck files)
| |
| |--studentname1_studentid1_submissionid2_filename2
| | |
| | +--(codecheck files)
| |
| |--studentname1_studentid1_submissionid3_filename3
| | |
| | +--(codecheck files)
| |
| |--studentname1_studentid1_submissionid1_filename1.ext
| |--studentname1_studentid1_submissionid2_filename2.ext
| |--studentname1_studentid1_submissionid3_filename3.ext
| |--aggregate_java_submissions.txt
| |--aggregate_report.html
| +--total_grade.txt
|
+--...
```

4.3.5 Open reports in the browser

With all of these "aggregate_*" files created to help with looking at the grades, it sure would be helpful to have a fast way to open all of these files at once. The script `open_reports_in_browser.py` will do that for you.

The script opens the `aggregate_report.html` files for a given directory with the default application for html files (probably your browser).

There are optional arguments (documented below) to provide the script if you only want to open reports for a certain alphabetical range for students' last names. This is helpful if you're taking a break for grading and you don't want to open *all* the report files, but just the student you left off with and the rest.

To open *all* the report files, you can run

```
python open_reports_in_browser.py path/to/directory
```

Opening all of the report files at once is probably not what you want to do though, because that's 100+ tabs in your browser at once.

There are `-s` and `-e` options that are optional. They're shorthand for `--start-letter` and `--end-letter`. (Run `python open_reports_in_browser.py --help` for clarification).

For example, to open report files starting with students whose last names start with 'G' and the rest of the students, run

```
python open_reports_in_browser.py path/to/directory -s G
```

which will open the reports from the 'G' students to the last students ('Z' students, implicitly).

To open report files from the beginning of the list until the 'N' students, run

```
python open_reports_in_browser.py path/to/directory -e N
```

which will open the reports from the beginning of the alphabet ('A' students) to the 'N' students.

To open reports in a range, say, from 'G' to 'N', just combine the options `-s` and `-e`

```
python open_reports_in_browser.py path/to/directory -s G -e N
```

There's a shortcut option, `-se`, that can be used if you just want to start and end with the same letter, which will only open reports within that single letter's range.

```
python open_reports_in_browser.py path/to/directory -se A
```

which is equivalent to the call

```
python open_reports_in_browser.py path/to/directory -s A -e A
```

(Note: the last-name letters are case-insensitive. so `-s G` and `-s g` are the same)

And if you would rather open the `aggregate_java_files.txt` files instead of the report files, then supply the `-j` option to the script. They'll probably open up in your text editor (because the script opens the files with the default application depending on the file extension. "txt" usually opens in text editors). The java files take up less vertical space overall compared to the report files, but the report files have colors like red that indicate test cases that failed and the test results of the program (including graphical results for graphics programs, so the report files are probably the way to go, but aggregate java files are also an option.

4.3.6 Checking for duplicate submissions

Usage:

```
python check_duplicate_submission_ids.py <directory>
```

`check_duplicate_submission_ids.py` is a script that will check the submissions directory (that you will pass as an argument to the script) for any duplicate submission ids in the directory. If any duplicates are found, then the script will display them like so:

```
Here are the students who have the same submission id:
Submission id: xxxxxxxxxxxxxxxxxxxxxxxxxxxx
      Students:
      lastname1--firstname1
      lastname2--firstname2
Submission id: yyyyyyyyyyyyyyyyyyyyyyyyyyyy
      Students:
      lastname3--firstname3
      lastname4--firstname4
[... and so on]
```

If there are no duplicate submission ids, the script will output:

```
There are no duplicate submission ids.
```

4.3.7 Grading structure

Each student's submissions are moved to his or her own directory named `lastname--firstname-miscname_canvasid`. Then, each codecheck file is unarchived to its own directory. Each student will have a `total_grade.txt` file found in his or her own directory, and a `grades.txt` file will be found in the `directory_of_submissions` (the directory you ran the script with).

At the top of `grades.txt` there are two fields that are used to identify the assignment:

- `_canvas_assignment_name`: The name of this assignment.

- `_canvas_assignment_id`: The ID of this assignment.

These are used for uploading grades to the correct assignment later when you use `grade_upload.py`.

For each student, `grades.txt` file has several fields:

- `_dir` shows the name of the directory of the student's submissions
- `_name` shows the name of the student
- `_canvas_id` shows the Canvas ID of the student
- `_total_score` shows the score that the student received for the assignment
- `_comment` holds the comments that you put in for a student's submission. This will be sent to Canvas and appear as a comment on the student's assignment. It will only be treated as text, with the beginning and ending whitespace stripped out. If you want to change the score, the **first token** of a line must be an integer, and that integer must be followed by **a single space**.
- `_notes` holds the options to mutate the score without showing it to the student. If you want to change the score, the **first token** of a line must be an integer, and that integer must be followed by **a single space**. You can also write your own personal notes here which will not sent to Canvas.

Both the `_comment` and `_notes` fields can be empty, but you should comment on something with each student. Give some good feedback :). All other fields should be treated as **read-only**. You should only need to write in the `_comment` and `_notes` fields.

The first line of the commect section will have a default comment, showing the codecheck score the student got on the assignments. Do not delete that row.

Example

Here's an example student submission in `grades.txt`:

```
_dir: lastname--firstname_0123456
_name: Lastname(s), Firstname(s)
_canvas_id: 0123456
_total_score: 18/18
_comment:
Codecheck score: 18/18

Good job, but remember to make your instance variables private.

-1 didn't make instance variables private

_notes:

-2 let's deduct more points for this example
+1 we can add points too
Here's a note. This line won't change the score at all.
--------
```

The student `last--first` with the ID `0123456` would receive 16 (= 18 - 1 - 2 + 1) and see the above comment on Canvas. The `_notes` field doesn't get published. It is only used to modify the score (or write notes), and only the numerical value at the beginning of a line matters. The label is optional; its only purpose is to give the score change context. For instance, the above `_notes` field would function exactly the same as

```
_notes:
-1
```

(continues on next page)

(continued from previous page)

```
-2
+1
--------
```

4.3.8 Uploading the grades

Uploading the grades to Canvas will publish the grades automatically with the grades given by the `grades.txt` file for students who submitted the grades. For students who did not do the homework (i.e., they did not submit anything), they will receive a "No submission." comment along with a score of `0` automatically when the grades are submitted.

Run `grades_upload.py` to upload the grades to Canvas.

```
python grades_upload.py path/to/grades.txt
```

`path/to/grades.txt` is the filepath to the grades.txt file you edited with your grades and comments.

4.3.9 Script Optional Arguments

Here's some detailed description for the optional arguments you can pass to the scripts.

`grade.py`

- Indicate the number of parts with `-n NUMBER` (or `--number-parts NUMBER`)
  - If you don't want to deal with the prompt and provide the number of parts to the assignment at runtime, you can provide the number of parts as an option when you run the script with this option, with `NUMBER` being the number of parts for this assignment. (e.g., `3`)
- Skip verifying the submission files with `--no-verify`
  - Instead of checking if the codecheck file is verified before extracting it, the script will skip verifying any files and just extract it. If you want the script to extract all the submission files, regardless of whether or not the codecheck file is actually signed, then use this option. Beware that the script will grade any codecheck file that's structured correctly even if it's not signed.

`grade_upload.py`

- Don't upload comments with `--no-comments`
  - By default the upload script will upload both scores and comments. If you **only** want to upload the grades, you can use the `--no-comments` option when running the script. Say, you want to reupload all the grades again due to a correction after already submitting grades. If you upload the grades again with comments, the comments will be duplicated. This options avoids duplication.
- Verbose output with `--verbose`
  - Use this option to have the script print out the grades before confirming the upload and print out more information with the URLs during the upload process. For more information on the possible options, run `python grade_upload.py --help`.

4.4 Development

This program is written and tested on a computer running Mac OS X 10.9 running Python 2.7.5 and Python 3.3.3.

Canvas API documentation was used to figure out what API calls to make in this script.

This script makes HTTP requests to Canvas's API with the help of Python's Requests library (it has made development easier and the code cleaner). More information and documentation on the library can be found here.

4.4.1 Dependencies

- Requests (used in terms with the Apache License, Version 2.0)
- Java JDK tools (`jar` and `jarsigner`)
- Canvas API
- Canvas submission zip file structure
- codecheck report structure
- Python and its standard library

4.4.2 Support

If you have any questions or problems with the scripts, please email me at daniel.mai01@sjsu.edu.

4.5 Changelog

- v2.2.9 (2016-02-21)
  - Removes student name from kudos comment (e.g., "Good job, NAME" or "Well done, NAME", because the script was actually writing out "Good job, None" and "Well done, None" (Canvas's zip format changed. See BitBucket Issue #3).
  - Change DEBUG level log format to have the filename, line number, function name, and message.
- v2.2.8 (2015-07-01)
  - Check **all** assignments when searching for one by name, not just the first 50 results. (The changes in v2.2.3 was not good enough.)
- v2.2.7 (2015-03-16)
  - Before uploading grades, check if the assignment exists first.
- v2.2.6 (2015-02-06)
  - Ignore 404s when uploading grades. The previous behavior was for the script to fail-fast when there is an assignment ID error. However, this also makes the script crash when the student canvas ID isn't found in the API, usually because of a student dropping the course.
- v2.2.5 (2014-10-12)
  - Change the way the score is searched in the grading file comments. The scores can only happen at the beginning of a line, not anywhere in the line (this unintentionally made multiple score changes made to a submission).
- v2.2.4 (2014-10-10)

  - Fix an issue with the codecheck files not being verified correctly by the keystore file.

- v2.2.3 (2014-10-07)
  - list_assignments returns up to 50 assignments instead of the default 10.
- v2.2.2 (2014-09-06)
  - Add public key, `codecheck-public.jks` to handle the codecheck files from the new server (cs19).
- v2.2.1 (2014-09-05)
  - Bug fix: UnicodeDecodeError when reading from java files. Solution: Remove file reading encoding when reading java files.
- v2.2.0 (2014-02-23)
  - Add the `check_duplicate_submission_ids.py` script to check for duplicate submission ids within an assignment.

CHAPTER 5

Support

If you have any questions or problems with the scripts, please email me at daniel.mai01@sjsu.edu.