Python Testing and Continuous Integration

UBCO Master of Data Science – DATA 533

Today's Class

Levels of Software Testing

Black and White Box Testing

Code Coverage In Python

LECTURE 5

Continuous Integration (CI)

- Use Travis-CI

Unit Testing

The foundational level of software testing is unit testing.

Unit testing specifically tests a single ***unit*** of code in isolation.

A unit is often a function or a method of a class instance.

```
def addition(num1, num2):
    return num1 + num2
```

One Unit

```
def subtraction(num1, num2):
    return num1 - num2
```

Another Unit

Levels of Test

Integration Tests: exercise groups of components to ensure that their contained units interact correctly together.

Acceptance Tests: focus on the business cases rather than the components themselves.

UI Tests: make sure that the application functions correctly from a user perspective.

https://github.com/ubccpsc/310/blob/2016sept/readings/Testing.md

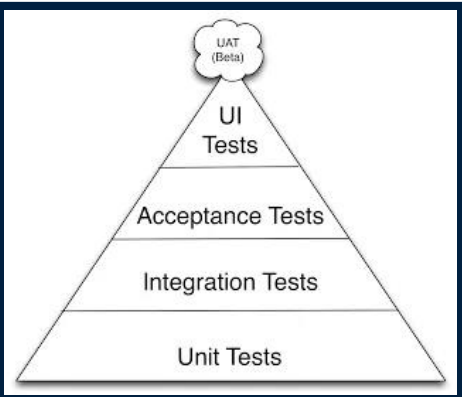
Levels of Test

Unit tests are fast and cheap to implement

- They're mostly doing checks on small pieces of code.

UI tests will be complex to implement

- Often require to get a full environment started as well as multiple services to emulate browser or mobile behaviours.



You may want to limit the number of complex UI tests and rely on good Unit testing at the base to have a fast build and get feedback to developers as soon as possible.

Black and White Box Testing

White box tests: Written with knowledge of the implementation of the code under test.

- Focuses on internal states of objects and code
- Focuses on to cover all code paths/statements
- Unit testing is often the first type of testing done on an application

Black box tests: written without knowledge of how the class/module/package under test is implemented

- Focuses on input/output of each component or call
- Black box testing can be applied to virtually every level of software testing: unit, integration, system, and acceptance.

Testing Question

Question: _____ make sure that multiple components behave correctly together

A) Unit tests.

B) Integration tests.

C) Acceptance tests.

D) End-to-End tests.

E) A|B tests

Testing Question

Question: How many of the following statements are TRUE?

A) White-box testing requires preparing test cases to exercise the internal logic of a software module.

B) Black box testing focuses on input/output of each component or call

C) UI Tests make sure that the application functions correctly from business cases perspective.

D) Implementation knowledge is required for black box testing.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it: Black Box Testing

Write a comprehensive set of test cases for the maximum function on the board.

```
def maximum(a, b):
# Return the larger numerical input, a or b
```

Try it: White Box Testing

Write a comprehensive set of test cases for the maximum function on the board.

```
def maximum(a, b):
    if (a > b):
        return a
    else:
        return b
```

Code Coverage

Test cases should examine the code and choose tests that exercise as much of the code as possible.

Code coverage

- Is usually reported as the percentage of overall code that is exercised.
- A program with high test coverage has had more of its source code executed during testing
- It suggests whether a program has a lower or higher chance of containing undetected software bugs.

Coverage.py: https://coverage.readthedocs.io/en/v4.5.x/

Code Coverage

Function coverage:

- how many of the functions defined have been called.

Statement coverage:

- how many of the statements in the program have been executed.

Branches coverage:

- how many of the branches of the control structures (if statements for instance) have been executed.

Line coverage:

- how many of lines of source code have been tested.

Code Coverage In Python

Install coverage: `python -m pip install coverage`

Run the test file: `python -m coverage run` *`test_file_path`*

Show the report : `python -m coverage report`

```
C:\Users\mkhasan\Anaconda3>python -m coverage report
Name                                Stmts   Miss  Cover
-------------------------------------------------------
C:\CodeCoverage\TestModule1.py          7      0   100%
C:\CodeCoverage\mod1.py                 2      0   100%
-------------------------------------------------------
TOTAL                                   9      0   100%
```

Produce an HTML report: `coverage html`

- This command will create a folder named **htmlcov** that contains various files. Navigate into that folder and try opening **index.html**

Try it: Code Coverage

Steps:

1. Download `grades.py` and `test_grades.py` files from `\lecture6\code\code coverage` folder

2. Install `Coverage.py` and execute commands to run the `test_grades.py` file to check the code coverage results

3. Now remove the # signs from the `test_grades.py` file, run the test code and check the coverage results again

General Guidelines

Integrate early and often:

- It is important that developers integrate their changes as soon as possible on the main repository, avoid “merge hell”

Keep the build green at all time

- Building means transforming your high-level code into a format your computer knows how to run.
- If a developer breaks the build for a branch, fixing it becomes the main priority.

Write tests as part of your stories

- You need to make sure that every feature that gets developed has automated tests.

Write tests when fixing bugs

- Make sure that you add tests when you fixing them from occurring again.

Continuous Integration

Continuous integration (CI) is the practice of frequently building and testing each change done to your code automatically and as early as possible.

Pioneered by Martin Fowler

“Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.” - Martin Fowler

Continuous Integration (CI)

Start writing tests for the critical parts of your codebase.

Get a CI service to run those tests automatically on every push to the main repository.

Make sure that your team integrates their changes everyday.

Fix the build as soon as it’s broken.

Write tests for every new story that you implement.

CI Providers

Travis

- Free for open source, most popular

Jenkins

- Host yourself, configure yourself (OpenShift)

CircleCI

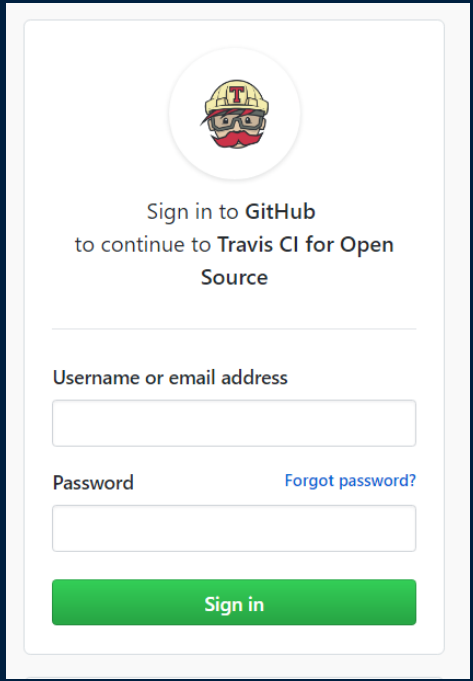
- Supports private projects
- Free 1500 minutes of builds per month

- others: Shippable, drone.io, appveyor

Travis CI: How Does It Work?

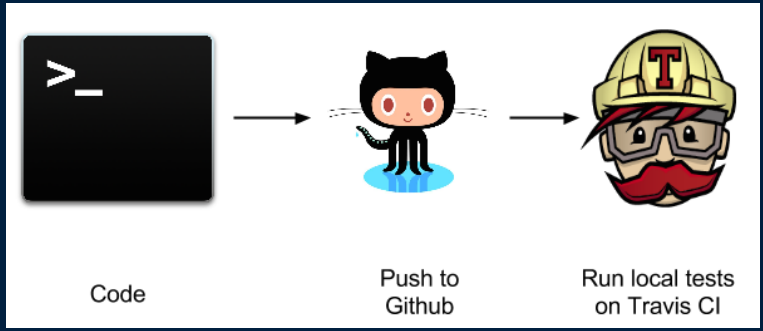
1. Sign in to Travis CI with your GitHub account, accepting the GitHub access permissions confirmation.

2. Once you're signed in, and Travis CI synchronized your repositories from GitHub, go to your profile page and enable Travis CI for the repository you want to build.



Travis CI: How Does It Work?

3. Add a `.travis.yml` file to your repository to tell Travis CI what to build.
4. Add the `.travis.yml` file to git, commit and push, to trigger a Travis CI build.
5. Check the build status page to see if your build passes or fails.



Travis: Configuration Steps

Create a new repository on GitHub

- Visit https://github.com/USERNAME.
- Click Repositories tab.
- Click New.
- Enter Repository name: InClassCI
- Click Create repository

Clone your Repo locally

- `$ git clone URL`
- `$ cd InClassCI`

Travis: Configuration Steps

Copy `lectures/lecture6/code/mod1.py` and Copy `lectures/lecture6/code/TestModule1.py`

Add and commit this file to your repository and push the changes to GitHub:

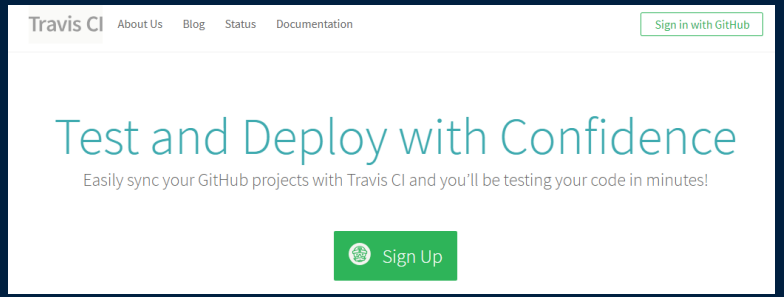
```
$ git add .
$ git commit -m "Added Unit test code"
$ git push
```

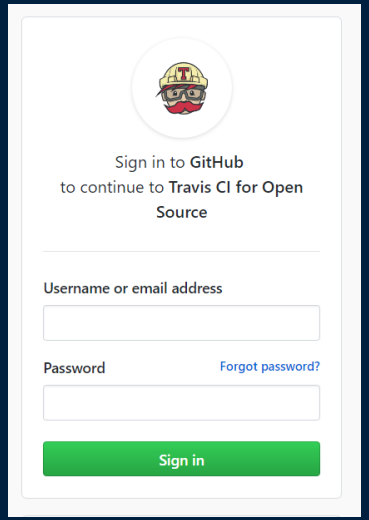
Visit `https://github.com/USERNAME/InClassCI` and check that the repository now contains all the files.

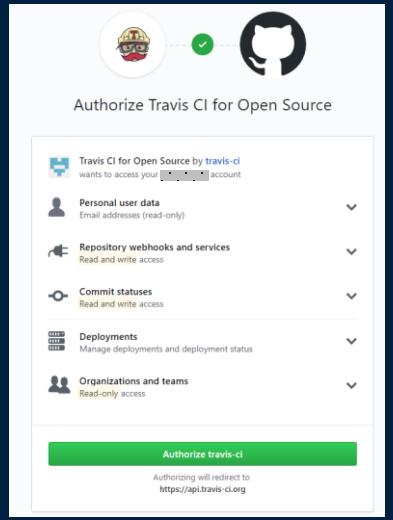
Travis: Configuration Steps

Sign in to Travis CI

- Once you have an account on GitHub, you can use this to sign in to Travis CI, so go to Travis CI, https://travis-ci.org/.
- Click on Sign in with GitHub.



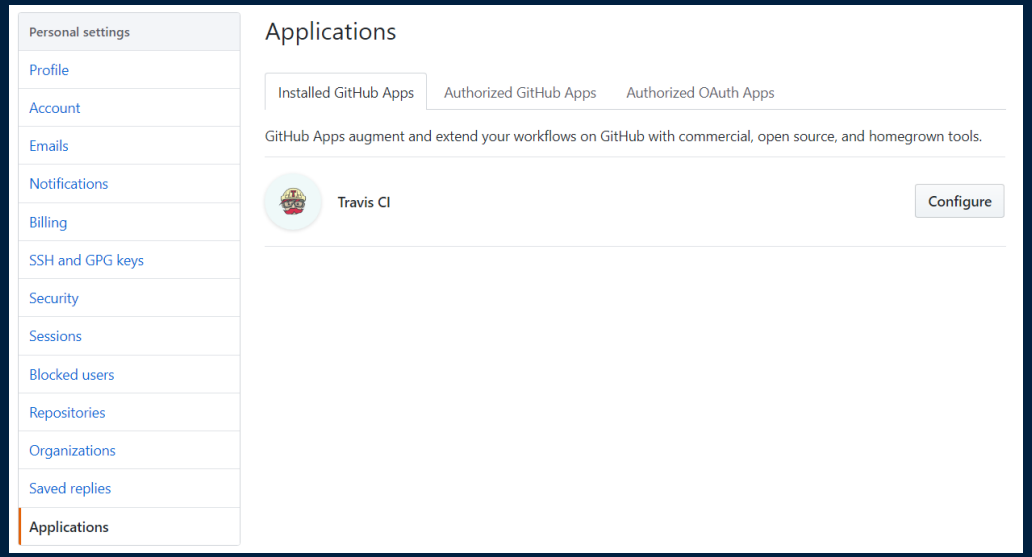


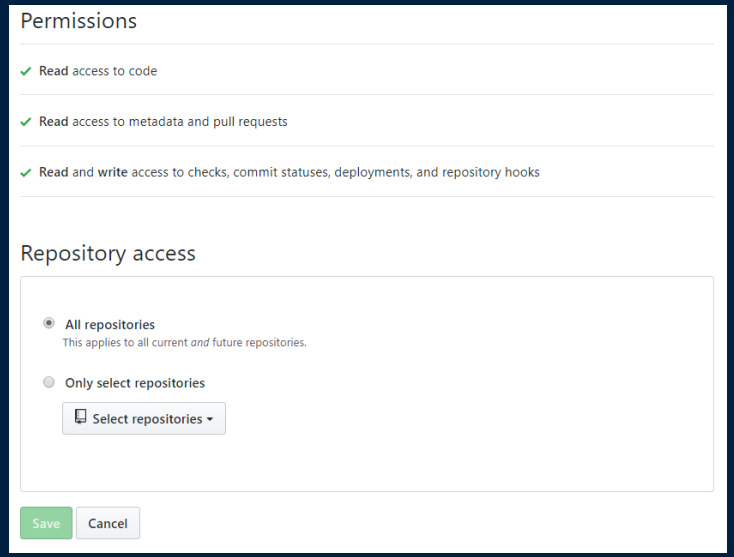


In Github

Settings -> Applications

You can also see the permissions and access to repositories

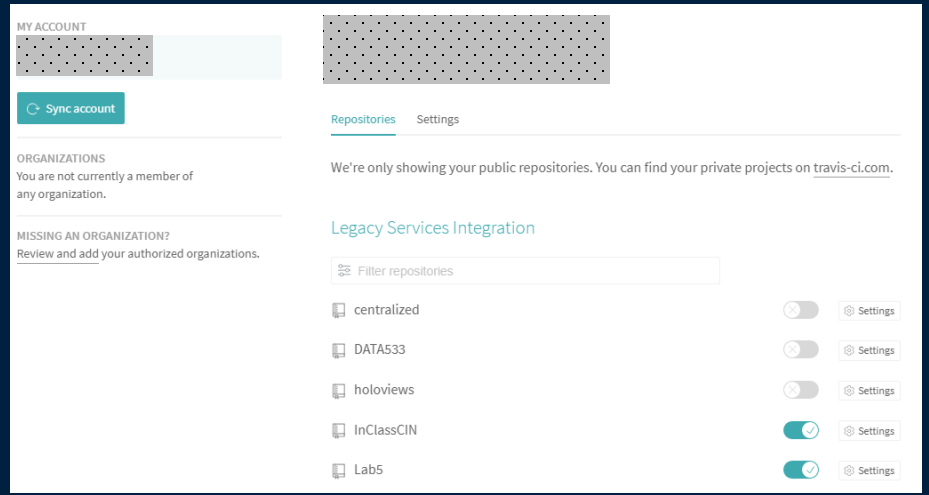




Travis: Configuration Steps

Enable your repository on Travis CI

- Go to `https://travis-ci.org/account/repositories`, which shows a list of your GitHub repositories that Travis CI knows about.
- If you cannot see `USERNAME/InClassCI`, then go to settings and click the Sync account button which tells Travis CI to check your current repositories on GitHub.
- When you can see `USERNAME/InClassCI`, then click on the button next to it to instruct Travis CI to monitor that repository for changes.



Travis: Current

Displays the most recent activities

Current Branches Build History Pull Requests

master Merge pull request #1 from khalad-hasan/newFeature

yml

Commit ac3b8c5

Compare 72e4d74..ac3b8c5

Branch master

#11 passed

Ran for 30 sec

Total time 48 sec

about an hour ago

Travis: Branches

Displays the most recent build for each branch

Current | Branches | Build History | Pull Requests

Default Branch

✓ master	# 11 passed	ac3b8c5
7 builds	about an hour ago	

Active Branches

✓ newFeature	# 9 passed	4851400
2 builds	about an hour ago	

Travis: Build History

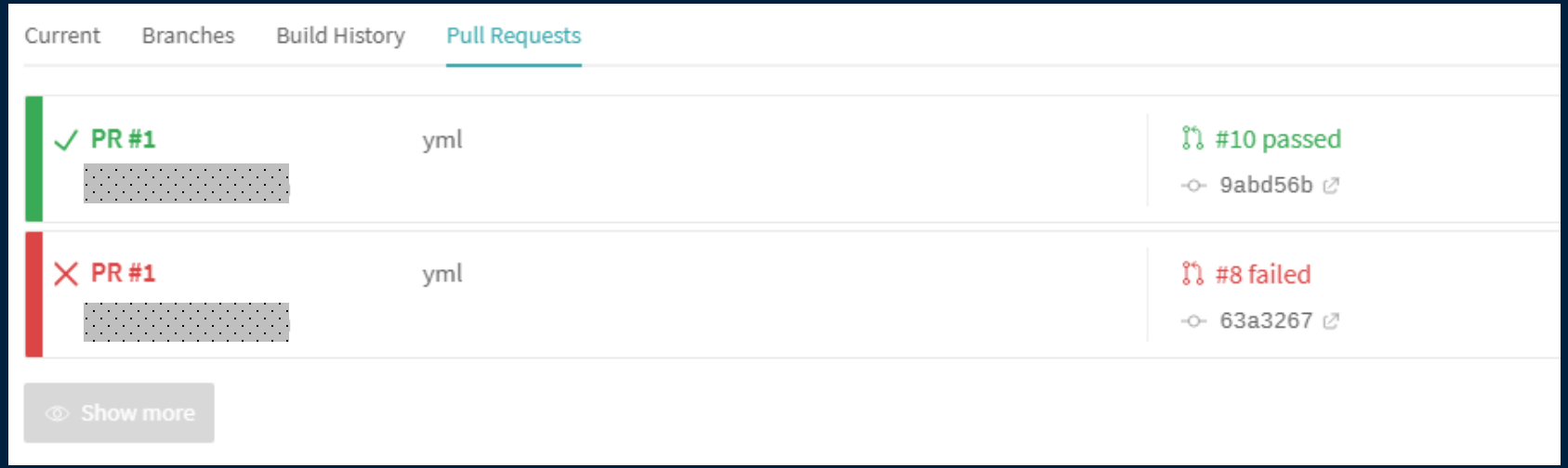
Displays a projects build history.

Current | Branches | Build History | Pull Requests

✓ master	Merge pull request #1 from khalad-hasan/newFeature	#11 passed ac3b8c5
✓ newFeature	yml	#9 passed 4851400
✗ newFeature	yml	#7 failed b8ae9af

Travis: Pull Requests

Displays recent pull requests



Travis: Configuration Steps

Create and add a `.travis.yml` job file

- Travis CI looks for a file called `.travis.yml` in a Git repository.
- This file tells Travis CI how to build and test your software.
- In addition, this file can be used to specify any dependencies you need installed before building or testing your software.

Create `.travis.yml` with the content:

```
language: python
python:
  - "3.4"
  - "3.5"
script:
  - python TestModule1.py
```

Build Configuration

Language:

- is used to specify the language of the software.

Python:

- is used to specify the version or versions of Python to use for testing.

Install:

- is used to specify commands to run before testing, such as the installation of dependencies or the compilation of required packages.

Script:

- section is used to specify the command to test your software.
- The specified command must exit with a status code of 0 if the test is successful; otherwise the test will be considered a failure.

Travis: Configuration Steps

Add and commit this file to your repository and push the changes to GitHub:

```
$ git add .
$ git commit -m "Added Travis CI job file" .
$ git push
```

Visit `https://github.com/USERNAME/InClassCI` and check that the repository now contains `.travis.yml`.

Explore the Travis CI job information

Visit `https://travis-ci.org/`. You should see a job called `InClassCI`. Jobs are named after the corresponding repositories.

Click on `InClassCI`.

This will take you to a page, which shows information about the run of your Travis CI job.

The job should be coloured green and with a check/tick icon which means that the job succeeded.

Try it: Travis

Task 1: Create a Github repository called DATA533LEC6. Clone the repo to your local machine.

Task 2: Setup Travis CI to sync the repository

Task 3: Create an application using Python inside the repo to find the maximum value between two numbers

Task 4: Create Unit Tests to check the Python program

Task 5: Define the continuous build in `.travis.yml` file.

Task 6: Push the local repo to Github.

Task 7: Create a new branch, push the branch to the Github and create and manage Pull Requests

Objectives

- Understand different testing techniques
- Learn about black and white box testing
- Learn about code coverage of Python programs
  - Use coverage.py tool for measuring code coverage
- Understand the necessity of Continuous Integration
- Be able to use Travis-CI for Continuous Integration


UBC
THE UNIVERSITY OF BRITISH COLUMBIA