

# "Continuous Integration 1, 2, 3" Exercise

---

## Prerequisites:

- [Software Maintenance and Quality Control](#)
- ["Testing 1, 2, 3" Exercise](#)
- [Travis CI](#)

## Learning Objectives

- Learn how to configure an application's automated tests to be run on a continuous integration server.
- Practice techniques for preventing bugs and undesired application functionality from reaching end users.

## Setup

Start following the [tutorial](#) to integrate Travis CI with your GitHub account.

If you haven't yet completed the ["Testing 1, 2, 3" Exercise](#), take a moment to do that now. We'll build on that application for the purposes of this exercise. Our goal is to configure a remote repository to run automated tests whenever we propose a change to this application's source code.


Navigate to the "testing-123" directory from the command line, and open it with a text editor. Verify your ability to run its tests before moving on:

```
cd path/to/testing-123 # or some other path to your repo

pytest
```

Great, the local directory is looking good. Let's take a moment to create a new remote repository on GitHub where we will push this code to. When creating a new remote repository, you should see the option to grant repo access to Travis CI platform. Ensure this option is selected!


Owner      Repository name \*

 s2t2 ▾ / testing-456-py ✓

Great repository names are short and memorable. Need inspiration? How about **turbo-computing-machine**?

Description (optional)

☒  **Public**  
Anyone can see this repository. You choose who can commit.

☐  **Private**  
You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.


Add .gitignore: **None** ▾

Add a license: **None** ▾



### Grant your Marketplace apps access to this repository

You are subscribed to 1 Marketplace app

☒  **Travis CI**  
Test and deploy with confidence

**Create repository**

NOTE: To configure Travis CI on an existing repository, or if you don't see the Travis CI option during repo creation, visit the [Travis account settings](#) and click "Manage repositories on GitHub", where you will be redirected to GitHub and prompted to select which repositories to configure.

As a final setup step, if you haven't yet done so, either clone your existing "testing-123" repository onto your local computer, or initialize a new Git repository in your local "testing-123" directory then associate that local repository with the remote repository's remote address:

```
# (only if you haven't already set up the repo):
```

```
git init .
git add .
git commit -m "My first commit"
```

```
git remote add origin YOUR_REPOS_REMOTE_ADDRESS
git push origin master
```

NOTE: if you see an error like "failed to push some refs", you may have to do a "forced" push with `git push origin master -f` instead.

You should now be able to see your code reflected on GitHub.

## Configuration

Continue following the Travis CI tutorial by adding a special configuration file called ".travis.yml" to the root directory of your repository, and place inside the following contents:

```
dist: xenial
language: python
python:
  - "3.7"
install:
  - pip install -r requirements.txt
script:
  - pytest
```

This file will instruct the CI server to use a specific version of python, install package dependencies, and run tests.

NOTE: The `dist: xenial` setting is necessary at this time for installation of Python version 3.7, and the `install: pip install -r requirements.txt` setting is only necessary if there are packages to install, so you can omit the "install" setting for this application. Pytest should already be automatically installed on the CI server, so there isn't a need to install it again via Pip.

Commit this new configuration file to the version history:

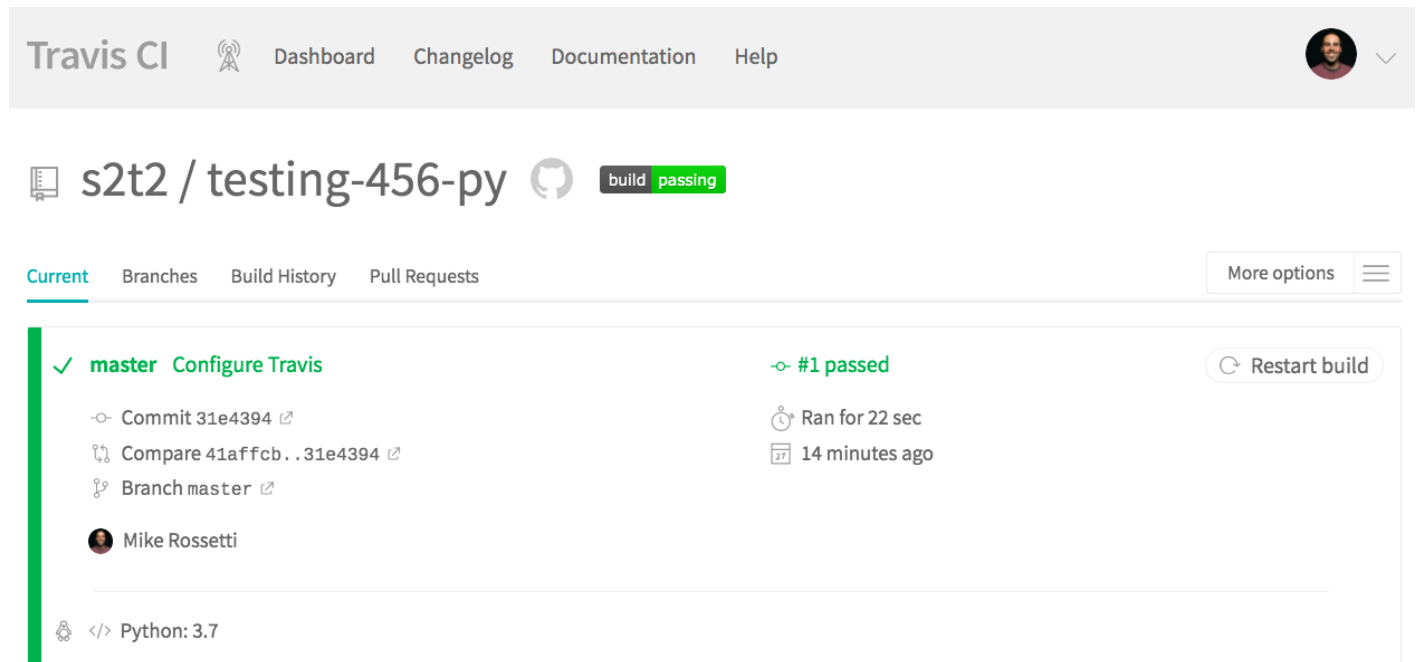
```
git add .
git commit -m "Configure Travis CI"
```

## Deploying

Push your local code up to the remote repository:

```
git push origin master
```

This push should now trigger the execution of automated tests on the CI server. Visit your repository's CI "builds" at a URL like [https://travis-ci.com/GITHUB\\_USERNAME/GITHUB\\_REPO\\_NAME](https://travis-ci.com/GITHUB_USERNAME/GITHUB_REPO_NAME) to see the results of the automated tests and verify Travis has been configured properly:



## Status Badges

OPTIONAL, if you want to show off the status of your build in your repo's README file!

In your repository page on Travis, locate the green "build: passing" icon and click on it. Choose "markdown" from the dropdown and copy the resulting snippet of markdown code. It should look like `[[Build Status]](https://travis-ci.com/s2t2/testing-456-py.svg?branch=master)](https://travis-ci.com/s2t2/testing-456-py)`. Paste that code into your repository's README file, somewhere near the top, and save the file.

Commit this change and push up to GitHub to see the badge reflected on your repository's homepage on GitHub:

```
git add .
git commit -m "Add CI status badge"
git push origin master
```

## Branch Operations

To make use of the more helpful aspects of the GitHub and Travis CI integration, let's practice a workflow involving branch operations and Pull Requests.

Check out a new branch called something like "my-new-feature":

```
git checkout -b my-new-feature
```

Edit the code in the application file (but don't update the corresponding test, so the tests will be failing):

```
def enlarge(i):  
    return i * 1
```


Save and commit these breaking changes, then push them to the "my-new-feature" branch:


```
git add .  
git commit -m "A breaking change"  
  
git push origin my-new-feature  
#> Counting objects: 4, done.  
#> Delta compression using up to 4 threads.  
#> Compressing objects: 100% (3/3), done.  
#> Writing objects: 100% (4/4), 353 bytes | 0 bytes/s, done.  
#> Total 4 (delta 2), reused 0 (delta 0)  
#> remote: Resolving deltas: 100% (2/2), completed with 2 local objects.  
#> remote:  
#> remote: Create a pull request for 'my-new-feature' on GitHub by visiting:  
#> remote:      https://github.com/s2t2/testing-456-py/pull/new/my-new-  
feature  
#> remote:  
#> To github.com:s2t2/testing-456-py.git  
#> * [new branch]      my-new-feature -> my-new-feature
```


You should be able to follow the resulting link to view your branch online and open a Pull Request to propose it be merged into the "master" branch.


Opening this PR should trigger a CI build, which in this case we expect to be failing:


# A breaking change #1


 **Open**


s2t2 wants to merge 1 commit into `master` from `my-new-feature` 

 Conversation 0

 Commits 1

 Checks 1


 Files changed 1




s2t2 commented a minute ago

Owner + 🧑🏻 ⋮


No description provided.



 A breaking change

✖ 1731aef


Add more commits by pushing to the `my-new-feature` branch on s2t2/testing-456-py.



✖ All checks have failed


2 failing checks

Hide all checks

✖  Travis CI - Branch

Failing after 10s — Build Failed

Details

✖  Travis CI - Pull Request

Failing after 25s — Build Failed

Details

This is an example of how automated checks can help alert developers to the need to fix breaking changes they may have contributed.

## Requiring Status Checks

### OPTIONAL

Follow [this guide](#) to designate the "master" branch as a protected branch, and "enable status checks" for the remote repository. Add a "branch protection rule" with a name pattern of "master", and enable the "Require status checks to pass before merging" and "Require branches to be up to date before merging" settings:

Branch name pattern

master

Applies to 1 branch

master

Rule settings

Protect matching branches

Disables force-pushes to all matching branches and prevents them from being deleted.

☐ Require pull request reviews before merging

When enabled, all commits must be made to a non-protected branch and submitted via a pull request with the required number of approving reviews and no changes requested before it can be merged into a branch that matches this rule.

☒ Require status checks to pass before merging

Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

☒ Require branches to be up to date before merging

This ensures pull requests targeting a matching branch have been tested with the latest code. This setting will not take effect unless at least one status check is enabled (see below).



Status checks found in the last week for this repository

☒ Travis CI - Branch


Required

Revisit your PR with the failing build and notice it now has merge restrictions:

# A breaking change #1

 **Open** s2t2 wants to merge 1 commit into `master` from `my-new-feature` 

 Conversation 0

 Commits 1

 Checks 2

 Files changed 1





s2t2 commented 4 minutes ago

Owner



*No description provided.*

  A breaking change

 1731aef

Add more commits by pushing to the `my-new-feature` branch on `s2t2/testing-456-py`.



**All checks have failed**

2 failing checks

[Hide all checks](#)



**Travis CI - Branch** Failing after 10s — Build Failed

**Required**

[Details](#)



**Travis CI - Pull Request** Failing after 25s — Build Failed

[Details](#)



**Required statuses must pass before merging**

All required [statuses](#) and check runs on this pull request must run successfully to enable automatic merging.

[Update branch](#)

This is an example of how automated checks can help prevent breaking changes from reaching end users.