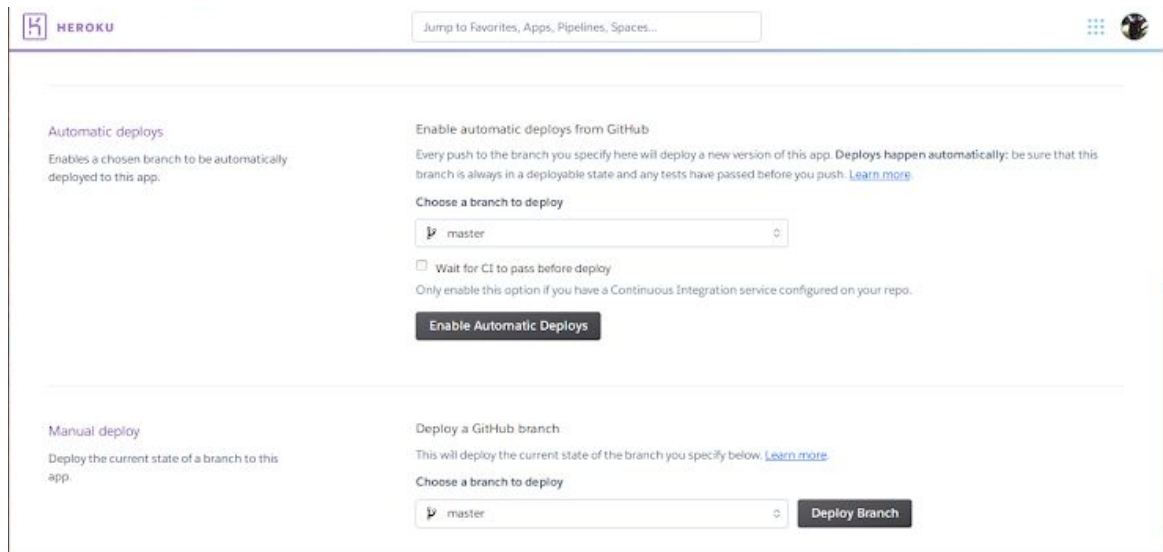


# Continuous Integration and Deployment

*Expected Time to Read and Implement: < 25 Minutes*

## Continuous Deployment

One of the easiest ways to set up continuous deployment is to enable automatic deploys on Heroku. (make sure to unselect the “Wait for CI to pass before deploying” option since we haven’t set up any CI yet). If you’ve connected Github and Heroku, after enabling automatic deploys, you can try making a change to the website and pushing the code to Github. You’ll see that that the website is rebuilt and deployed every single time.



The screenshot shows the Heroku dashboard interface. At the top, there's a navigation bar with the Heroku logo and a search bar. Below the navigation bar, the main content area is divided into two sections: 'Automatic deploys' and 'Manual deploy'.

**Automatic deploys:** This section is titled 'Automatic deploys' and has a subtitle 'Enables a chosen branch to be automatically deployed to this app.' It contains a heading 'Enable automatic deploys from GitHub' followed by a paragraph: 'Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. [Learn more](#)'. Below this is a dropdown menu labeled 'Choose a branch to deploy' with 'master' selected. There is an unchecked checkbox labeled 'Wait for CI to pass before deploy' with a note: 'Only enable this option if you have a Continuous Integration service configured on your repo.' At the bottom of this section is a button labeled 'Enable Automatic Deploys'.

**Manual deploy:** This section is titled 'Manual deploy' and has a subtitle 'Deploy the current state of a branch to this app.' It contains a heading 'Deploy a GitHub branch' followed by a paragraph: 'This will deploy the current state of the branch you specify below. [Learn more](#)'. Below this is a dropdown menu labeled 'Choose a branch to deploy' with 'master' selected. At the bottom of this section is a button labeled 'Deploy Branch'.

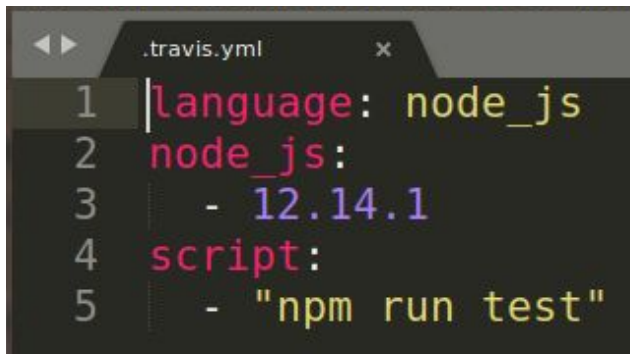
After this is done, make sure to push code to the master branch only when you’re sure it’s working completely.

In the next part of the guide, we will setup TravisCI so that any change pushed to the master branch is run against **unit tests** before automatic deployment. Unit tests are used to automatically verify that code is working as it’s supposed to. While you’ll be learning and working with unit tests in detail post-Midsem exams, the guide in the next page will help you setup TravisCI on your app. Using TravisCI you can automatically run tests as soon you push a change to Github. (*Note: TravisCI works only on public Github repos*)

# Setting Up Travis CI

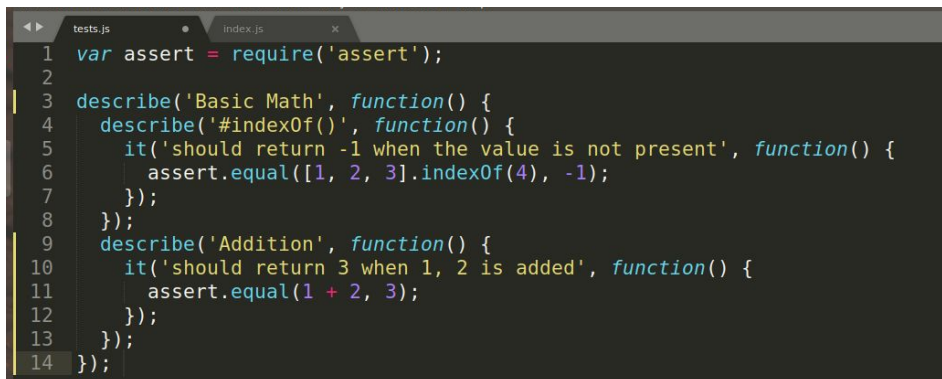
Travis CI has been set up on [this git repository](#) (same repo as last two tutorials). It might be helpful to keep looking at the code there while reading these instructions.

1. Create a file `.travis.yml` in the root of your MERN project that looks something like this. The node version in your `package.json` and this should match.



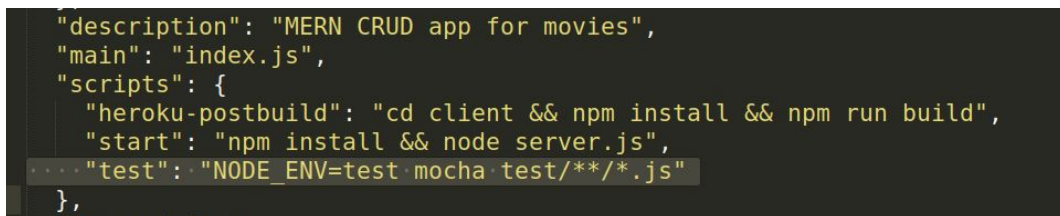
```
1 language: node_js
2 node_js:
3   - 12.14.1
4 script:
5   - "npm run test"
```

2. Install Mocha, Chai and Supertest using these commands:
  - a. `npm install mocha chai supertest --save-dev`
3. The file `tests/basic_tests.js` in the git repository gives you an example of how unit tests look like. You can create something that looks like this in your repository as well (or just copy-paste it). The second test case is asserting that  $1+1 = 2$  and the first one is confirming that an element is not present in a list. These tests are not actually useful in checking code - we're just using these to show you how Continuous Integration (CI) works. For Sprint 2 of your semester project, you'll be writing unit test cases to actually test your code before deploying.



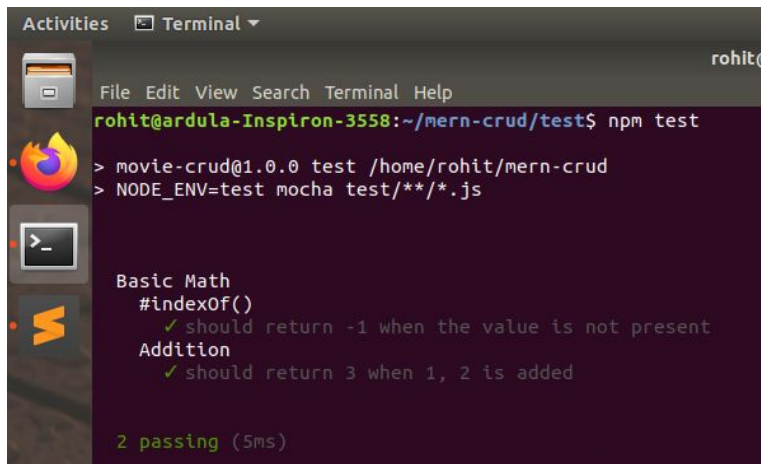
```
1 var assert = require('assert');
2
3 describe('Basic Math', function() {
4   describe('#indexOf()', function() {
5     it('should return -1 when the value is not present', function() {
6       assert.equal([1, 2, 3].indexOf(4), -1);
7     });
8   });
9   describe('Addition', function() {
10    it('should return 3 when 1, 2 is added', function() {
11      assert.equal(1 + 2, 3);
12    });
13  });
14 });
```

4. Edit the `package.json` to include a test script. (highlighted)



```
"description": "MERN CRUD app for movies",
"main": "index.js",
"scripts": {
  "heroku-postbuild": "cd client && npm install && npm run build",
  "start": "npm install && node server.js",
  "test": "NODE_ENV=test mocha test/**/*.js"
},
"devDependencies": {
```

5. Try running `npm test` in the console. If everything's been configured right, we'll see that both the test cases have passed.



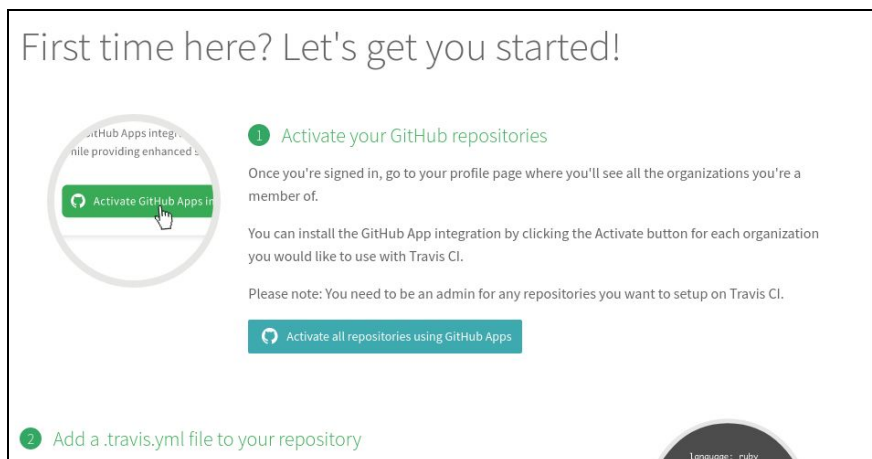
A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help) and a user prompt 'rohit@'. The command `npm test` has been executed. The output shows the command `movie-crud@1.0.0 test /home/rohit/mern-crud` and the command `NODE_ENV=test mocha test/**/*.js`. Below this, two test cases are listed: 'Basic Math' with a sub-test '#indexOf()' that 'should return -1 when the value is not present', and 'Addition' that 'should return 3 when 1, 2 is added'. Both tests are marked with a green checkmark. At the bottom, it says '2 passing (5ms)'.

```
rohit@ardula-Inspiron-3558:~/mern-crud/test$ npm test
> movie-crud@1.0.0 test /home/rohit/mern-crud
> NODE_ENV=test mocha test/**/*.js

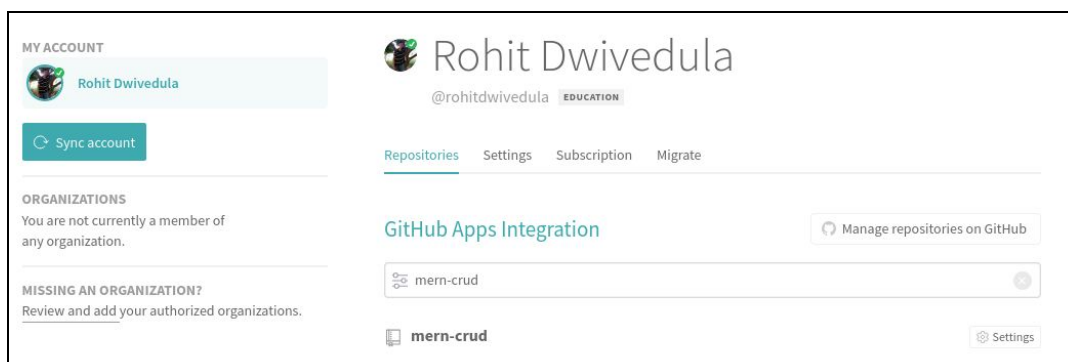
Basic Math
  #indexOf()
    ✓ should return -1 when the value is not present
  Addition
    ✓ should return 3 when 1, 2 is added

2 passing (5ms)
```

6. Commit all of this to your git repo and push it to Github.
7. With this done, we'll now go to [travis-ci.org](https://travis-ci.org) and create an account with Github.
8. Activate your Github repositories on Travis CI by giving the relevant permissions.



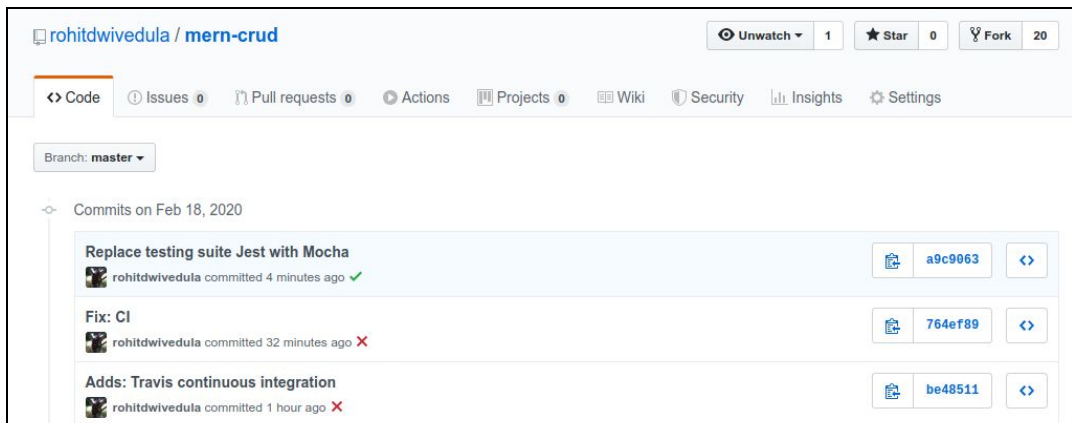
9. Wait for a while as Travis CI connects to Github. Search for the relevant app and click on it.



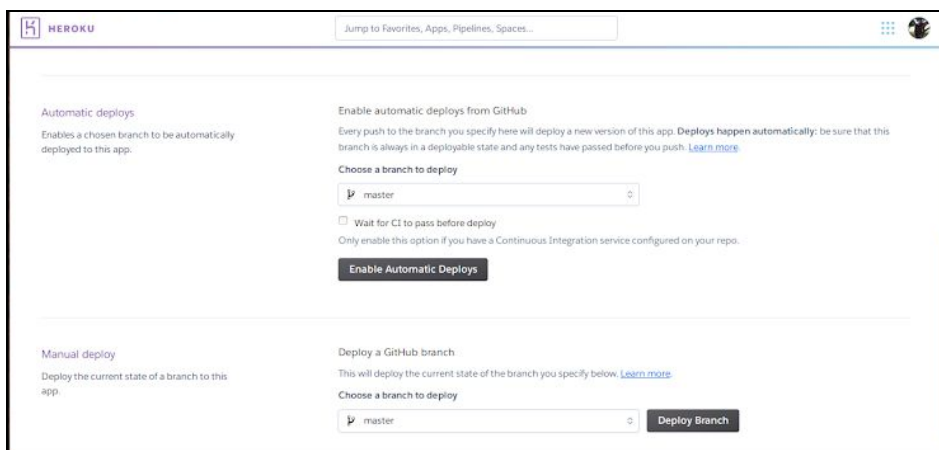
10. TravisCI will run the unit tests and if you've configured everything properly, you'll see a green screen of success.



11. See the commit history on Github. You'll see a green tick mark/red cross beside each commit telling you whether unit tests passed or not for that.



12. Go back to the Heroku, and open your app settings and go to the deploy section. Click on the “Enable Automatic Deploys” button. Make sure to select the tick box “Wait for CI to pass before deploy”. Doing this will ensure that your app will be redeployed only if the unit tests pass.



## Links

1. [CI/CD with GitHub, Travis CI and Heroku](#): This guide explains how to set up continuous integration using TravisCI.
2. [Build a Unit-Testing Suite with Mocha and Mongoose](#)
3. [Endpoint testing with Jest and Supertest | Zell Liew](#): A brief intro to unit testing using Jest.