**Getting Django Set Up Using a Functional Test**

<http://www.obeythetestinggoat.com/book/chapter_01.html>

Goto: -- C:\Users\Public\devTddMilPub\note

Open: -- R-Click > GIT-bash

Open: -- R-Click > GIT-bash

Type GIT-bash: -- $ source venvSuperlists/Scripts/activate

TDD isn’t something that comes naturally. It’s a discipline, like a martial art, and just like in a Kung Fu movie, you need a bad-tempered and unreasonable master to force you to learn the discipline. Ours is the Testing Goat.

**Obey the Testing Goat! Do Nothing Until You Have a Test**

The Testing Goat is the unofficial mascot of TDD in the Python testing community. It probably means different things to different people, but, to me, the Testing Goat is a voice inside my head that keeps me on the True Path of Testing—​like one of those little angels or demons that pop up above your shoulder in the cartoons, but with a very niche set of concerns. I hope, with this book, to install the Testing Goat inside your head too.

We’ve decided to build a website, even if we’re not quite sure what it’s going to do yet. Normally the first step in web development is getting your web framework installed and configured. *Download this, install that, configure the other, run the script*…​but TDD requires a different mindset. When you’re doing TDD, you always have the Testing Goat inside you—​single-minded as goats are—​bleating “Test first, test first!”

In TDD the first step is always the same: *write a test*.

*First* we write the test; *then* we run it and check that it fails as expected. *Only then* do we go ahead and build some of our app. Repeat that to yourself in a goat-like voice. I know I do.

Another thing about goats is that they take one step at a time. That’s why they seldom fall off mountains, see, no matter how steep they are. As you can see in [Goats are more agile than you think (source:](http://www.obeythetestinggoat.com/book/chapter_01.html#tree_goat) [Caitlin Stewart, on Flickr](http://www.flickr.com/photos/caitlinstewart/2846642630/)).



Figure 1. Goats are more agile than you think (source: [Caitlin Stewart, on Flickr](http://www.flickr.com/photos/caitlinstewart/2846642630/))

We’ll proceed with nice small steps; we’re going to use *Django*, which is a popular Python web framework, to build our app.

The first thing we want to do is check that we’ve got Django installed, and that it’s ready for us to work with. The *way* we’ll check is by confirming that we can spin up Django’s development server and actually see it serving up a web page, in our web browser, on our local PC. We’ll use the *Selenium* browser automation tool for this.

Create a new Python file called *functional\_tests.py*, wherever you want to keep the code for your project, and enter the following code. If you feel like making a few little goat noises as you do it, it may help:

functional\_tests.py

from selenium import webdriver

browser = webdriver.Firefox()

browser.get('http://localhost:8000')

assert 'Django' in browser.title

That’s our first *functional test* (FT); I’ll talk more about what I mean by functional tests, and how they contrast with unit tests, in a bit. For now, it’s enough to assure ourselves that we understand what it’s doing:

* Starting a Selenium "webdriver" to pop up a real Firefox browser window
* Using it to open up a web page which we’re expecting to be served from the local PC
* Checking (making a test assertion) that the page has the word "Django" in its title

Let’s try running it:

$ **python functional\_tests.py**

File ".../selenium/webdriver/remote/webdriver.py", line 320, in get

self.execute(Command.GET, {'url': url})

File ".../selenium/webdriver/remote/webdriver.py", line 308, in execute

self.error\_handler.check\_response(response)

File ".../selenium/webdriver/remote/errorhandler.py", line 194, in

check\_response

raise exception\_class(message, screen, stacktrace)

selenium.common.exceptions.WebDriverException: Message: Reached error page: abo

ut:neterror?e=connectionFailure&u=http%3A//localhost%3A8000/[...]

You should see a browser window pop up and try to open *localhost:8000*, and show the "Unable to connect" error page. If you switch back to your console, you’ll see the big ugly error message, telling us that Selenium hit an error page. And then, you will probably be irritated at the fact that it left the Firefox window lying around your desktop for you to tidy up. We’ll fix that later!

|  |  |
| --- | --- |
|  | If, instead, you see127.0 an error trying to import Selenium, or an error trying to find "geckodriver", you might need to go back and have another look at the "[[pre-requisites]](http://www.obeythetestinggoat.com/book/pre-requisite-installations.html)" section. |

For now though, we have a *failing test*, so that means we’re allowed to start building our app.

Adieu to Roman Numerals!

So many introductions to TDD use Roman numerals as an example that it’s a running joke—​I even started writing one myself. If you’re curious, you can find it on [my GitHub page](https://github.com/hjwp/tdd-roman-numeral-calculator/).

Roman numerals, as an example, are both good and bad. It’s a nice “toy” problem, reasonably limited in scope, and you can explain TDD quite well with it.

The problem is that it can be hard to relate to the real world. That’s why I’ve decided to use building a real web app, starting from nothing, as my example. Although it’s a simple web app, my hope is that it will be easier for you to carry across to your next real project.

**Getting Django Up and Running**

Since you’ve definitely read “[[pre-requisites]](http://www.obeythetestinggoat.com/book/pre-requisite-installations.html)” by now, you’ve already got Django installed. The first step in getting Django up and running is to create a *project*, which will be the main container for our site. Django provides a little command-line tool for this:

**1.1\_$ django-admin.py startproject superlists .**

**3.1\_$ django-admin startproject superlists .**

Don’t forget that "." at the end, it’s important!

That will create a file called *manage.py* in your current folder, and a subfolder called *superlists*, with more stuff inside it:

├── functional\_tests.py

├── geckodriver.log

├── manage.py

├── \_superlists

│ ├── \_\_init\_\_.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

└── virtualenv # >3.1: I use conda

├── [...]

|  |  |
| --- | --- |
|  | Make sure your project folder looks exactly like this! If you see two nested folders called superlists, it’s because you forgot the "." above. Delete them and try again. |

The *superlists* folder is intended for stuff that applies to the whole project—​like *settings.py*, for example, which is used to store global configuration information for the site.

But the main thing to notice is *manage.py*. That’s Django’s Swiss Army knife, and one of the things it can do is run a development server. Let’s try that now:

$ **python manage.py runserver**

Performing system checks...

System check identified no issues (0 silenced).

You have 13 unapplied migration(s). Your project may not work properly until

you apply the migrations for app(s): admin, auth, contenttypes, sessions.

Run 'python manage.py migrate' to apply them.

Django version 1.11.3, using settings 'superlists.settings'

Starting development server at <http://127.0.0.1:8000/>

Quit the server with CONTROL-C.

|  |  |
| --- | --- |
|  | It’s safe to ignore that message about "unapplied migrations" for now. We’ll look at migrations in [[chapter\_post\_and\_database]](http://www.obeythetestinggoat.com/book/chapter_post_and_database.html). |

That’s Django’s development server now up and running on our machine.

Leave it there and open another command shell. Navigate to your project folder, activate your virtualenv, and then try running our test again:

$ **python functional\_tests.py**

$

|  |  |
| --- | --- |
|  | If you see an error saying "no module named selenium", you’ve forgotten to activate your virtualenv. Check the [[pre-requisites]](http://www.obeythetestinggoat.com/book/pre-requisite-installations.html) again if you need to. |

Not much action on the command line, but you should notice two things: firstly, there was no ugly AssertionError and secondly, the Firefox window that Selenium popped up had a different-looking page on it.

Well, it may not look like much, but that was our first ever passing test! Hooray!

If it all feels a bit too much like magic, like it wasn’t quite real, why not go and take a look at the dev server manually, by opening a web browser yourself and visiting *http://localhost:8000*? You should see something like [It worked!](http://www.obeythetestinggoat.com/book/chapter_01.html#it_worked_screenshot).

You can quit the development server now if you like, back in the original shell, using Ctrl-C.

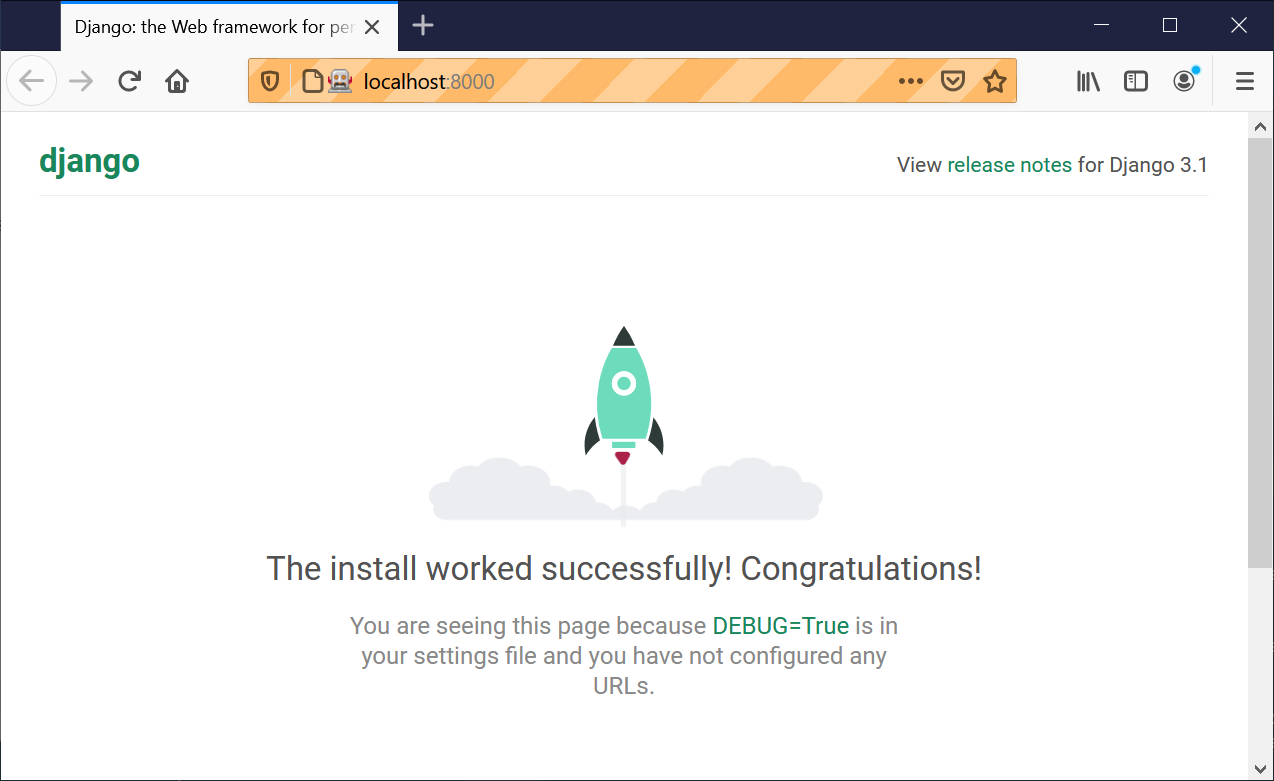


Figure 2. It worked!

>N: How should I deal with GIT ?

**Starting a Git Repository**

There’s one last thing to do before we finish the chapter: start to commit our work to a *version control system* (VCS). If you’re an experienced programmer you don’t need to hear me preaching about version control, but if you’re new to it please believe me when I say that VCS is a must-have. As soon as your project gets to be more than a few weeks old and a few lines of code, having a tool available to look back over old versions of code, revert changes, explore new ideas safely, even just as a backup…​boy. TDD goes hand in hand with version control, so I want to make sure I impart how it fits into the workflow.

So, our first commit! If anything it’s a bit late; shame on us. We’re using *Git* as our VCS, ’cos it’s the best.

Let’s start by doing the git init to start the repository:

$ **ls**

db.sqlite3

functional\_tests.py

geckodriver.log

manage.py

superlists

virtualenv

$ **git init .**

Initialised empty Git repository in ...python-tdd-book/.git/

Our Working Directory is always the folder that contains manage.py

>RKW: Note

In my case it will be C:\Users\Public\devTddMilPub\note

>RKW: VS2007

TODO- I need to set this up first.

From this point onwards, the top-level *superlists* folder will be our working directory.

(For simplicity, in my command listings, I’ll always show it as *…​python-tdd-book/*, although it will probably actually be something like */home/kind-reader-username/my-python-projects/python-tdd-book/*.)

Whenever I show a command to type in, it will assume we’re in this directory. Similarly, if I mention a path to a file, it will be relative to this top-level directory. So *superlists/settings.py* means the *settings.py* inside the second-level *superlists*.

Now let’s take a look and see what files we want to commit:

$ **ls**

db.sqlite3

functional\_tests.py

geckodriver.log

manage.py

superlists

virtualenv

There are a few things in here that we *don’t* want under version control: *db.sqlite3* is the database file, *geckodriver.log* contains Selenium debug output, and finally our virtualenv shouldn’t be in git either. We’ll add all of them to a special file called *.gitignore* which, um, tells Git what to ignore:

$ **echo "db.sqlite3" >> .gitignore**

$ **echo "geckodriver.log" >> .gitignore**

$ **echo "virtualenv" >> .gitignore**

Next we can add the rest of the contents of the current folder, ".":

$ **git add .**

$ **git status**

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: .gitignore

new file: functional\_tests.py

new file: manage.py

new file: superlists/\_\_init\_\_.py

new file: superlists/\_\_pycache\_\_/\_\_init\_\_.cpython-36.pyc

new file: superlists/\_\_pycache\_\_/settings.cpython-36.pyc

new file: superlists/\_\_pycache\_\_/urls.cpython-36.pyc

new file: superlists/\_\_pycache\_\_/wsgi.cpython-36.pyc

new file: superlists/settings.py

new file: superlists/urls.py

new file: superlists/wsgi.py

Oops! We’ve got a bunch of *.pyc* files in there; it’s pointless to commit those. Let’s remove them from Git and add them to *.gitignore* too:

$ **git rm -r --cached superlists/\_\_pycache\_\_**

rm 'superlists/\_\_pycache\_\_/\_\_init\_\_.cpython-36.pyc'

rm 'superlists/\_\_pycache\_\_/settings.cpython-36.pyc'

rm 'superlists/\_\_pycache\_\_/urls.cpython-36.pyc'

rm 'superlists/\_\_pycache\_\_/wsgi.cpython-36.pyc'

$ **echo "\_\_pycache\_\_" >> .gitignore**

$ **echo "\*.pyc" >> .gitignore**

Now let’s see where we are…​ (You’ll see I’m using git status a lot—​so much so that I often alias it to git st…​I’m not telling you how to do that though; I leave you to discover the secrets of Git aliases on your own!):

$ **git status**

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: .gitignore

new file: functional\_tests.py

new file: manage.py

new file: superlists/\_\_init\_\_.py

new file: superlists/settings.py

new file: superlists/urls.py

new file: superlists/wsgi.py

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: .gitignore

Looking good—​we’re ready to do our first commit!

$ **git add .gitignore**

$ **git commit**

When you type git commit, it will pop up an editor window for you to write your commit message in. Mine looked like [First Git commit](http://www.obeythetestinggoat.com/book/chapter_01.html#first_git_commit).[[1](http://www.obeythetestinggoat.com/book/chapter_01.html#_footnote_1)]

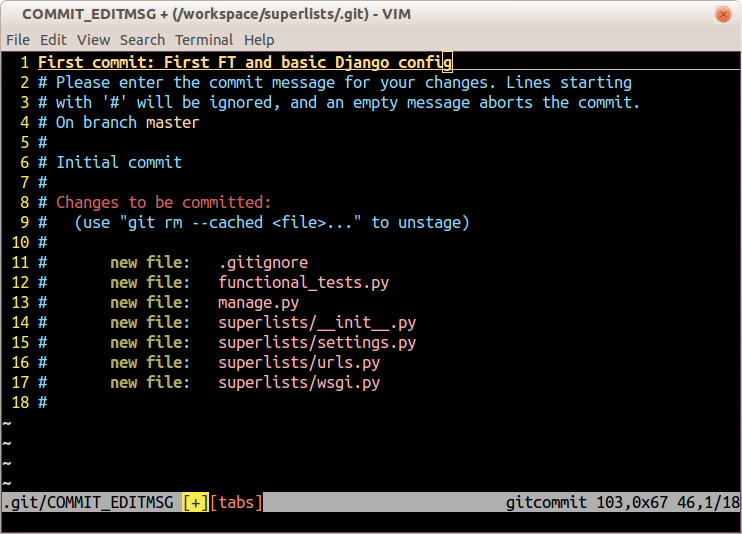


Figure 3. First Git commit

|  |  |
| --- | --- |
|  | If you want to really go to town on Git, this is the time to also learn about how to push your work to a cloud-based VCS hosting service, like GitHub or Bitbucket. They’ll be useful if you think you want to follow along with this book on different PCs. I leave it to you to find out how they work; they have excellent documentation. Alternatively, you can wait until [[chapter\_manual\_deployment]](http://www.obeythetestinggoat.com/book/chapter_manual_deployment.html) when we’ll be using one for deployment. |

That’s it for the VCS lecture. Congratulations! You’ve written a functional test using Selenium, and you’ve gotten Django installed and running, in a certifiable, test-first, goat-approved TDD way. Give yourself a well-deserved pat on the back before moving on to [[chapter\_02\_unittest]](http://www.obeythetestinggoat.com/book/chapter_02_unittest.html).

[1](http://www.obeythetestinggoat.com/book/chapter_01.html#_footnoteref_1). Did vi pop up and you had no idea what to do? Or did you see a message about account identity and git config --global user.username? Go and take another look at “[[pre-requisites]](http://www.obeythetestinggoat.com/book/pre-requisite-installations.html)”; there are some brief instructions.

Last updated 2017-11-21 06:25:12 GMT