Table of Contents

Getting Django Set Up Using a Functional Test

Test-driven development 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[1] 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 pops up by 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.

So we've decided to build a web app, 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 your head—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 things, see, no matter how steep they are—as you can see in Goats are more agile than you think (source: Caitlin Stewart, on Flickr).

Figure 1. Goats are more agile than you think (source: Caitlin Stewart, on Flickr)

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 computer. 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

```
PYTHON
from selenium import webdriver

browser = webdriver.Firefox()
browser.get("http://localhost:8000")

assert "Congratulations!" in browser.title
print("OK")
```

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 computer.
- Checking (making a test assertion) that the page has the word "Congratulations!" in its title.
- If all goes well, we print OK.

Let's try running it:

```
$ python functional_tests.py
Traceback (most recent call last):
  File "...goat-book/functional_tests.py", line 4, in <module>
    browser.get("http://localhost:8000")
  File ".../selenium/webdriver/remote/webdriver.py", line 483, in get
    self.execute(Command.GET, {"url": url})
  File ".../selenium/webdriver/remote/webdriver.py", line 458, in execute
    self.error_handler.check_response(response)
  File ".../selenium/webdriver/remote/errorhandler.py", line 232, 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/[...]
Stacktrace:
RemoteError@chrome://remote/content/shared/RemoteError.sys.mjs:8:8
WebDriverError@chrome://remote/content/shared/webdriver/Errors.sys.mjs:182:5
UnknownError@chrome://remote/content/shared/webdriver/Errors.sys.mjs:530:5
[...]
```

You should see a browser window pop up trying to open *localhost:8000*, showing 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 ran into 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 see an error trying to import Selenium, or an error trying to find something called "geckodriver", you might need to go back and have another look at the "[pre-requisites]".

What to Do If You Get a Firefox Upgrade Pop-up

Now and again, when running Selenium tests, you might encounter a strange pop-up window, such as the one shown in Firefox wants to install a new what now?.

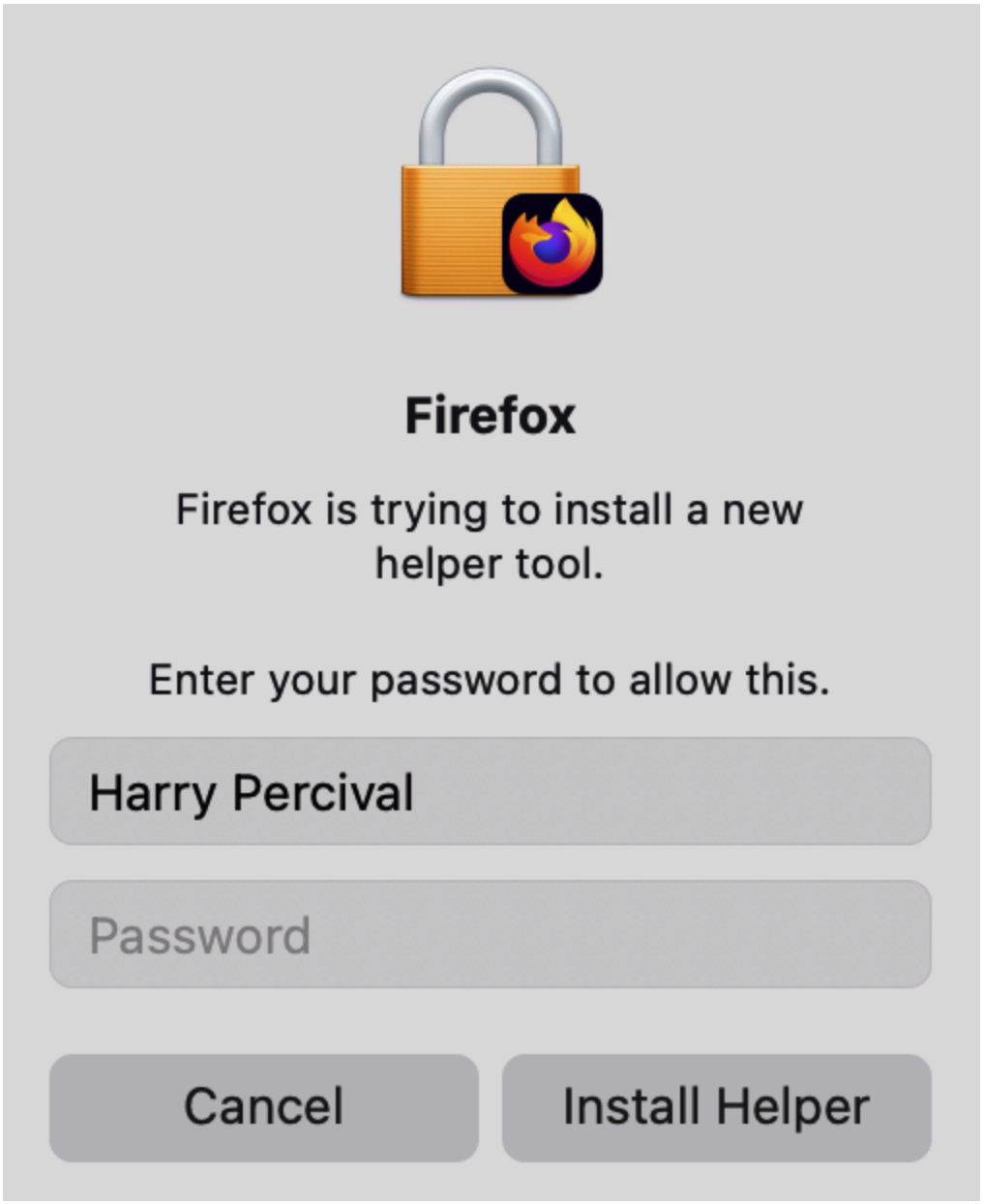

Figure 2. Firefox wants to install a new what now?

This happens when Firefox has automatically downloaded a new version, in the background. When Selenium tries to load a fresh Firefox session, it wants to install the latest version of its "geckodriver" plugin.

To resolve the issue, you have to close the Selenium browser window, go back to your main browser window and tell it to install the upgrade and restart itself, and then try again.

If something strange is going on with your FTs, it's worth checking if there's a Firefox upgrade pending.

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

Getting Django Up and Running

As you've definitely read "[pre-requisites]" by now, you've already got Django installed (right?). 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:

```
$ 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
├── manage.py
└── superlists
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

> Make sure your project folder looks exactly like this! If you see two nested folders called "superlists", it's because you forgot the "." in the command. Delete them and try again, or there will be lots of confusion with paths and working directories.

The *superlists* folder is intended for stuff that applies to the whole project—like *settings.py*, 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
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 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.
March 17, 2023 - 18:07:30
Django version 5.2.4, using settings 'superlists.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

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

> It's safe to ignore that message about "unapplied migrations" for now. We'll look at migrations in [chapter_05_post_and_database].

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
OK
```

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.

> If you see an error saying "ModuleNotFoundError: No module named selenium", you've forgotten to activate your virtualenv. Check the "[pre-requisites]" section again, if you need to.

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!.

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

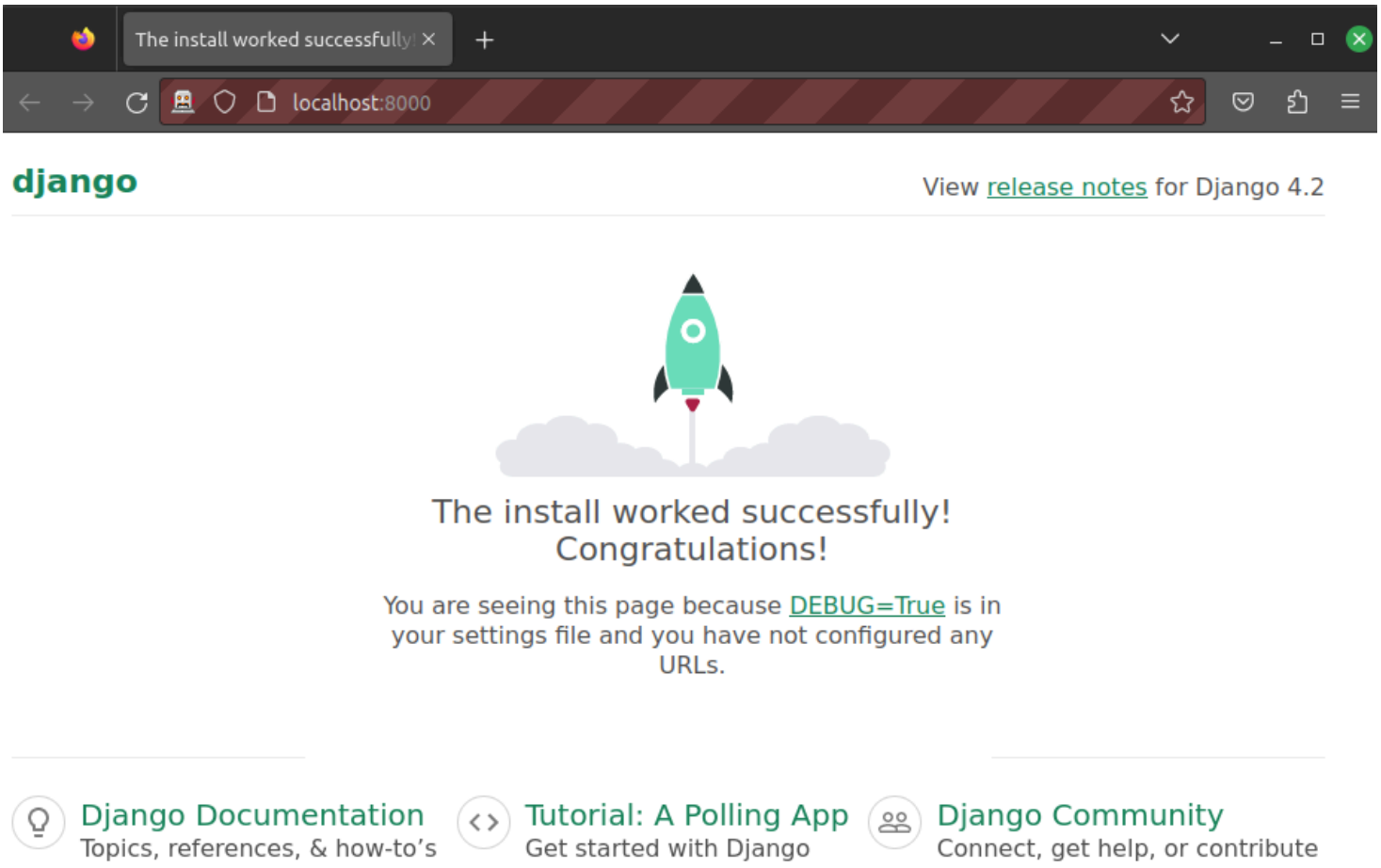

Figure 3. It worked!

Adieu to Roman Numerals!

So many introductions to TDD use Roman numerals in their examples that it has become 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 the core of 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 the building of 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.

In addition, it means we can start out using functional tests as well as unit tests, and demonstrate a TDD workflow that's more like real life, and less like that of a toy project.

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…It's hard to overstate how useful that is. TDD goes hand in hand with version control, so I want to make sure I impart how it fits into the workflow.

Our Working Directory Is Always the Folder That Contains manage.py

We'll be using this same folder throughout the book as our working directory—if in doubt, it's the one that contains *manage.py*.

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

Whenever I show a command to type in, I will assume we're in this directory. Similarly, if I mention a path to a file, it will be relative to this directory. So, for example, *superlists/settings.py* means the *settings.py* inside the *superlists* folder.

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  manage.py  superlists

$ git init .
Initialised empty Git repository in ...goat-book/.git/
```

Setting the Default Branch Name in Git

If you see this message:

```
hint: Using 'master' as the name for the initial branch. This default branch
hint: name is subject to change. To configure the initial branch name to use
hint: in all of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialised empty Git repository in ...goat-book/.git/
```

Consider following the advice and choosing an explicit default branch name. I chose `main`. It's a popular choice, and you might see it here and there in the book. So if you want to match that, do:

```
$ git config --global init.defaultBranch main
# then let's re-create our git repo by deleting and starting again:
$ rm -rf .git
$ git init .
Initialised empty Git repository in ...goat-book/.git/
```

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

```
$ ls
db.sqlite3 functional_tests.py manage.py superlists
```

There are a few things in here that we *don't* want under version control: *db.sqlite3* is the database file, and 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 ".venv" >> .gitignore
```

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

```
$ git add .
$ git status
On branch main

No commits yet

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-314.pyc
        new file:   superlists/__pycache__/settings.cpython-314.pyc
        new file:   superlists/__pycache__/urls.cpython-314.pyc
        new file:   superlists/__pycache__/wsgi.cpython-314.pyc
        new file:   superlists/asgi.py
        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-314.pyc'
rm 'superlists/__pycache__/settings.cpython-314.pyc'
rm 'superlists/__pycache__/urls.cpython-314.pyc'
rm 'superlists/__pycache__/wsgi.cpython-314.pyc'
$ echo "__pycache__" >> .gitignore
$ echo "*.pyc" >> .gitignore
```

Now let's see where we are...

```
$ git status
On branch main

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/asgi.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 restore <file>..." to discard changes in working directory)

        modified:   .gitignore
```

> 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!

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.[2]

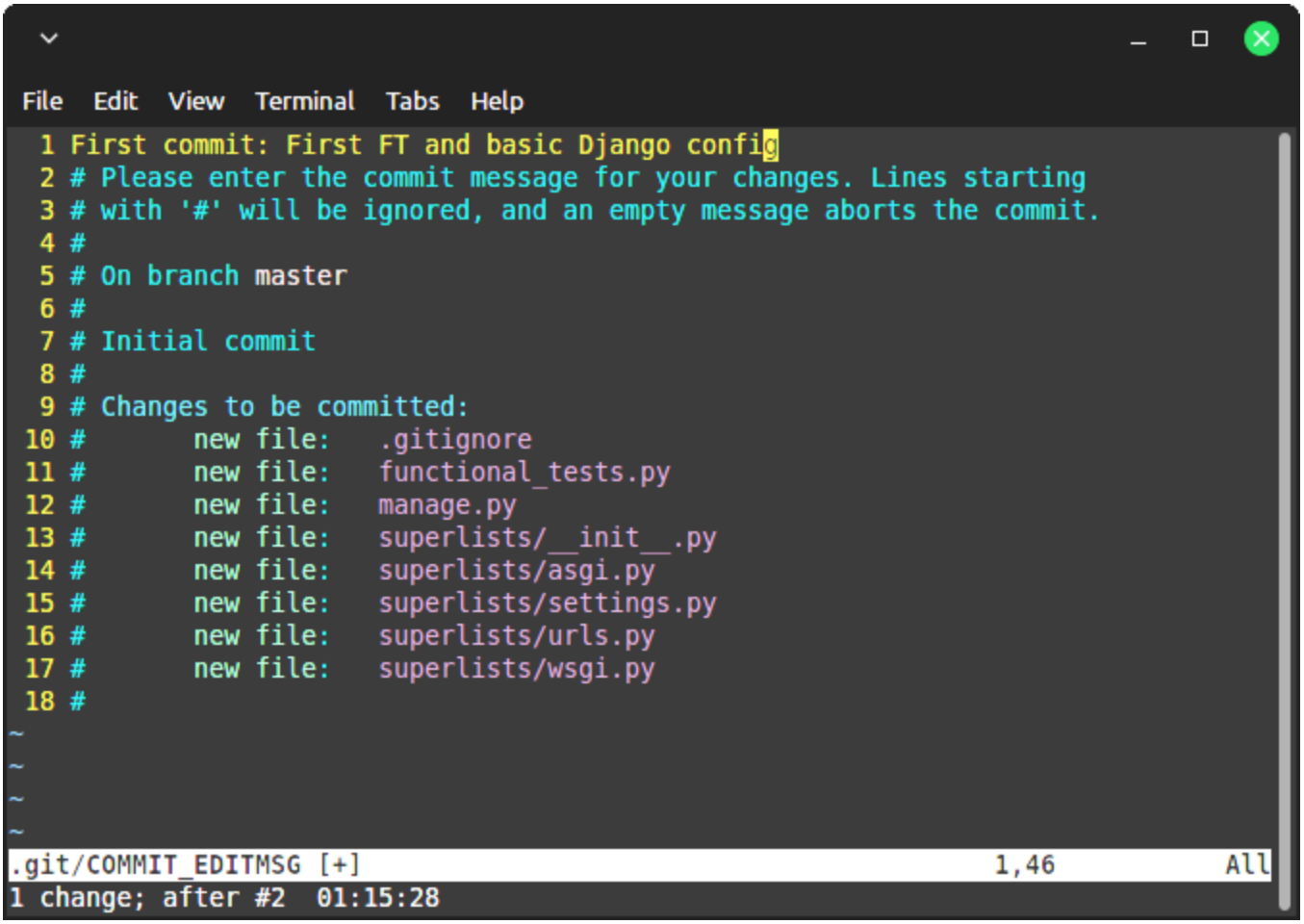

Figure 4. First Git commit

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 GitLab. They'll be useful if you think you want to follow along with this book on different computers. I leave it to you to find out how they work; they have excellent documentation. Alternatively, you can wait until [chapter_25_CI], where we'll use one.

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].

1. OK more of a minor running joke from PyCon in the mid 2010s, which I am single-handedly trying to make into a thing.
2. Did a strange terminal-based editor (the dreaded Vim) 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`? Check out the Git manual and its basic configuration section (http://git-scm.com/book/en/Customizing-Git-Git-Configuration). PS: To quit Vim, it's Esc, then `:q!`

Comments

ALSO ON **OBEY THE TESTING GOAT!**

Obey the Testing Goat!

8 years ago • 12 comments

I'm still not great at selfies... Here you get two for the price of one tho. The second …

Making our App Production-Ready

2 years ago • 8 comments

Static files aren’t the same kind of things as the dynamic content that comes from …

Prettification: Layout and Styling, and What to

2 years ago • 4 comments

We’re starting to think about releasing the first version of our site, but we’re a bit …

What Are All These '

2 years ago • 5

Let’s take a l
tests, lists/te
we’re lookin

76 Comments 1 **Kgotso Koete** ▾

Join the discussion…

♡ 4 **Share** **Best** **Newest** **Oldest**

Kentaro Hori
9 years ago

with window7 I had to change from

django-admin.py startproject superlists
to
django-admin startproject superlists

2 0 Reply

Makarov Andrey → Kentaro Hori
6 years ago edited

(I'm going to challenge myself and complete the course within Anaconda environment:)
There's django-admin.exe which you start. I think you could use django-admin.py if you start it explicitly with full path: `python full/path/django-admin.py`

0 0 Reply

I

Innocent Sizwe → Kentaro Hori
6 years ago

Windows 7 definitely works for me.

0 0 Reply

AlexeyKot → Kentaro Hori
7 years ago

I had the same problem in Windows 10 and Cmder. Thanks for sharing!

0 0 Reply

hjwp Mod → Kentaro Hori
9 years ago

I've tested on both windows 7 and windows 10, and django-admin.py definitely works for me, but django also bundles a script called "django-admin", so both will work I think.

0 0 Reply

A

A23

2 years ago

had issues in Ubuntu. firefox did not "open the URL", solved via telling the webdriver to use snapped geckodriver...

```
from selenium import webdriver
from selenium.webdriver.firefox.options import Options
from selenium.webdriver.firefox.service import Service


FFoptions=Options()
FFservice=Service(executable_path="/snap/bin/geckodriver")


browser = webdriver.Firefox(options=FFoptions,service=FFservice)
browser.get("http://localhost:8000")


assert "Congratulations!" in browser.title
print("OK")
```

0 0 Reply

T

Tiana Stastny

3 years ago

Hello! I was getting the error `selenium.common.exceptions.WebDriverException: Message: Process unexpectedly closed with status 1` until adding the following option:

```
opts = FirefoxOptions()
opts.add_argument("--headless")
browser = webdriver.Firefox(options=opts)
```

That seems to have resolved the error, but Firefox does not open automatically. If I manually go to http://localhost:8000, I do see the correct 'It worked!' page. I'm wondering if something is not set up properly though, since Firefox isn't automatically opening when running `python functional_tests.py` ?

0 0 Reply

A

Andrew → Tiana Stastny

2 years ago

> Firefox does not open automatically

The `--headless` option means 'without a graphical interface'.
https://en.wikipedia.org/wi...
https://hacks.mozilla.org/2...
https://en.wikipedia.org/wi...

So, with that option I expect no Firefox window will open.

0 0 Reply

M **Morgan Mains**

3 years ago

Be sure you installed Python 3.6.0 and not a newer version!!!!
Several of the modules that the older Django references have changed and you'll get all kinds of ImportErrors

0 0 Reply

Louis Burns

5 years ago edited

I ended up having to use Anaconda prompt to get `python manage.py runserver` to work. I'd created a virtual environment in Anaconda too. I wasn't able to figure out whether it was a version issue. It would open the browser but not find a connection with Bash.

Also, my version of git recognized not to add the .pyc files. And I have a strong preference for adding the message as a flag in the `commit` command.

0 0 Reply

Makarov Andrey

6 years ago edited

When using Windows echo command I had to write `echo db.sqlite3 >> .gitignore` (text w/o quotes)

0 0 Reply

F **Forest Bump**

7 years ago

What is the best approach with regards to the database file and migrations when several people are working on a Django site and all are using Git? In particular, I am struggling with the getting the database up and running on a local machine after checking out from Git. As the database itself is in the gitignore-list, I guess the first thing I have to do is to create the file. How do I go about doing this. Thx.

0 0 Reply

S **Simon**

7 years ago edited

trying to follow the goat as best I can..
before creating the git repo why not put (in functional tests for now)

import os
assert os.path.isdir(.git')

run the test, it fails
then create git repo

I guess it doesn't actually test any code though.. so probably shouldn't be included? Is there any role for testing non-code related things that the app relies upon?

0 0 Reply

Dave Campbell

7 years ago edited

before you do your first 'git commit' you show the 'git status' output that includes 'Initial commit'. mine shows 'No commits yet'.

0 0 Reply

hjwp Mod → Dave Campbell

7 years ago

thanks! i wonder if git has changed its default first commit message. will look into it!

0 0 Reply

P

Prabhas

7 years ago

The command 'python manage.py runserver' is hung. taking forever to run. what should I do? If someone got this error before. please help me.

0 0 Reply

P

Prabhas → Prabhas

7 years ago

After 36 hours(I kept the command running), I got the following output.
$ python manage.py runserver
Not Found: /runestone/ajax/getnumonline
Not Found: /runestone/ajax/getuser
Not Found: /runestone/ajax/getnumusers
[06/Dec/2018 18:55:36] "GET /runestone/ajax/getnumonline HTTP/1.1" 404 2014
[06/Dec/2018 18:55:36] "GET /runestone/ajax/getuser HTTP/1.1" 404 1999
[06/Dec/2018 18:55:36] "GET /runestone/ajax/getnumusers HTTP/1.1" 404 2011

And, It is still running.

0 0 Reply

hjwp Mod → Prabhas

7 years ago

hi there, "runserver" runs the django development server on your machine, it's designed to run forever, or at least until you kill it or interrupt it with ctrl+c.
at this point you're meant to proceed with the server running in one terminal window, and your tests running in another. sorry i didn't make that clear enough!

1 0 Reply

P **Prabhas** hjwp

7 years ago

thank you so much for the reply. I tested it. It is working fine. I was worried because I didn't get the following output as shown in tutorial.

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 did not output anything. It was completely empty. should I be worried about this? apart from that, It is working perfectly fine.

0 0 Reply

P **pranoto budi** Prabhas

7 years ago

it's OK. actually you don't need to do anything. (it's been a month, you've probably has had the answer, but for anyone else having same problem),
if you see this:
Starting development server at http://127.0.0.1:8000/
that's a good sign. that means your web server is running.
if you need for visual proof that it is ok, you just need to open your browser and check at http://127.0.0.1:8000/ and you'll see something there.

as for "You have 13 unapplied migration", it is also not a problem, it just want to tell you that the database table regarding django apps default bundle (admin, auth, contenttypes, sessions) has not been created. you can do it later (usually when you have define your own database object) by "python manage.py makemigrations" followed by "python manage.py migrate" command it will create all database tables needed.

0 0 Reply

F **facebook-555724132**

7 years ago

I get the following error when running "python functional_tests.py"
selenium.common.exceptions.WebDriverException: Message: invalid argument: can't kill an exited process
any help ?

0 0 Reply

hjwp Mod → facebook-555724132

7 years ago

hi! check that you've got the latest versions of firefox, selenium and geckodriver. and (long shot) are you trying to run this on a server perhaps, in a headless environment? if so you can set an env var MOZ_HEADLESS=1 but i would just recommend using firefox on your own machine. you need to be able to see what it's doing...

0 0 Reply

das-g → hjwp

7 years ago edited

On Linux, if you *are* running it on your local computer, *with* a graphical environment and *still* get the message, make sure `$DISPLAY` is set up correctly in the terminal you use. This should usually be the case automatically, but maybe something (e.g. your `~/.bashrc` or `~/.profile`) messes with that environment variable.

0 0 Reply

R

Richard Porteous

7 years ago

I seem to be two years behind the initial crowd and I'm late in getting into python.

The setup works using the versions mentioned if you stick with Python 3.6. (right now it is 3.6.6). Python 3.7 chokes on some of these older libraries.

0 0 Reply

hjwp Mod → Richard Porteous

7 years ago

Richard I do apologise, I just tested it myself and you're quite right, Django 1.11LTS is not compatible with Python 3.7 (more info here https://github.com/django/d... -- I'll add a note to the installation instructions.

thanks very much for prompting me!

0 0 Reply

R

Richard Porteous → hjwp

7 years ago

Gotta love virtualenv

0 0 Reply

hjwp Mod → Richard Porteous

7 years ago

Can you provide a bit more info Richard? Both libraries have had several releases since 3.7 came out.

0 0 Reply

J **junghanChoi**

8 years ago

whoooray!!

0 0 Reply

 **ST-Tech**

8 years ago edited

Shouldn't you have tested to see if Python and Selenium were installed first (not just Django!!!)?

$pip uninstall selenium

test.py =
try:
from selenium import webdriver
browser = webdriver.Firefox()
print ('O.K')
except:
print('An Error occured')
Will throw and exception

$ pip install selenium

test.py will now pass

But : not sure how to automate a test for Python install when Python is NOT installed to run the test :-p

0 0 Reply

 **hjwp** Mod → ST-Tech

8 years ago

excellent. the good thing about this is that it also serves as a test that the computer is switched on ;-) :-P

2 0 Reply

 **das-g** → ST-Tech

7 years ago

We ain't unit testing here (yet). For a functional test, it's kinda OK that it tests several "things" together, as long as it tests a single question from the user's perspective. The implied question here is "Are we ready to Django? (And to test our Djangoing?)"

This might not be the case for various reasons:

- as **@hjwp** jokingly suggests, the computer might be off
- no operating system might be installed

- python might not be installed
- geckodriver might not be installed
- firefox might not be installed
- virtualenv might be missing
- virtualenv might not be activated
- Django might not be installed in the virtualenv
- Selenium might not be installed in the virtualenv
- Django version might be incompatible with Python version
- Selenium version, geckodriver version and Firefox version might be incompatibel to each other
- you may have accidentally hit one of the gazillion of possible silly edge cases, like having spaces in the path of your project folder
- ...

The important thing is, that our single test will fail if even one these reasons applies. Then we will know that we must investigate what that reason is. Less importantly but somewhat helpfully, the test will fail *differently* for many of these different reasons. That can give us the first hint for that investigation, even though the error message will often not directly tell us exactly what's wrong.

0 0 Reply

Neil Hainer

8 years ago

Note: When you do a "git commit" you have to add some text in the editor pop-up window. If you don't you will receive the following message:

"Aborting commit due to empty commit message."

I don't believe this is mentioned in the text.

0 0 Reply

J

junghanChoi Neil Hainer

8 years ago

I guess he did it on purpose.

1 0 Reply

C

Chris Sullivan

8 years ago

Steve Molin (4 months ago) - I had to upgrade firefox from 52 (default on CentOS 7) and ended up with 57. I removed the firefox package. Then downloaded firefox for my architecture x86.64 and moved the firefox directory to /opt. Then ran ln -s /opt/firefox/firefox /usr/local/bin/firefox. Follow that, firefox --version returned V57 and everything started working. By everything, because I was running client/server with a headless server, I had to use pyvirtualdisplay to create a virtual display in memory, but that journey can be recounted another day.

0 0 Reply

A

ACR

8 years ago

Just to share my own experience - I got the exception "No module named 'django.contrib.sessions.serializers'" when I first re-ran the functional tests. Relaunching everything after 'pip install django-serializers' fixed it.

0 0 Reply

Tyeth Gundry

8 years ago edited

Had a bit of fudging to do with windows10 git-bash and anaconda3 installed by visual studio 2017, as it keeps the python seperate from other command prompts.
I therefore had to add the path for anaconda to environment variables manually, (C:\Program Files\Anaconda\Scripts -where you stick geckodriver)
I used the linux addition for .bashrc for mkvirtualenv to ensure git bash command prompt loads the virtualenvwrapper,
then to build the environment I had to fudge the command to quotemark encapsulate the string of the path of python that gets printed. It moaned about my python argument -3.6 which can probably be removed, but it got me there:

mkvirtualenv --python=`python -3.6 -c"import sys; print('sys.executable')"` superlists

0 0 Reply

Steve Molin

8 years ago

Not able to get selenium working, possibly because of version mismatch (Selenium 3.6.0, Firefox 52.3.0, GeckoDriver 0.19.0 - all current as of 2017.10.7). I'm going to try proceeding with 'requests' and BeautifulSoup (per comment on https://stackoverflow.com/q.... Aware of limitations, but have made it through "What are we doing with all these tests?" (chapter 4) so far.

Also, great book! Thank you. Love the concept of TDD, still trying to grok it.

0 0 Reply

das-g → Steve Molin

7 years ago

For those wondering which versions are compatible to each other, that's documented on https://firefox-source-docs...

1 0 Reply

Steve Molin → Steve Molin

8 years ago

Thanks Chris - GeckoDriver 0.18.0 did the trick, and I think I'll be happier on Selenium in the long

Thanks Chris - GeckoDriver 0.18.0 did the trick, and I think I'll be happier on Selenium in the long run - if for no other reason than I can more easily follow Harry's excellent book. (I'm happy to know about 'requests' and BeautifulSoup though, I'm sure they'll come in handy).

Thanks Harry - My FF build was September 28 and I though "oh, that's gotta be current, right?" Wrong :-) If I have some breathing room I'll try again with latest FF, Selium and GeckoDriver.

0 0 Reply

hjwp Mod → Steve Molin

8 years ago

that beautifulsoup solution is ingenious! but i'd definitely try to get selenium working if you can. your firefox sounds old, I have 56 at least.

0 0 Reply

hjwp Mod → hjwp

8 years ago

and sorry I didnt reply sooner! am still trying to grok tdd myself ;-). there's some good philosophical stuff in later chapters, particularly in part 3, let me know how you get on with it...

0 0 Reply

C

Chris Dukes → Steve Molin

8 years ago

Try geckodriver v0.18.0. Because I encountered the same problem with firefox-esr 52.4.0esr~1~deb9u1 + geckodrver v0.19.0. Downgrading resolved the issue.

0 0 Reply

hjwp Mod → Chris Dukes

8 years ago

or try upgrading firefox. the esr releases are several versions behind current stable...

0 0 Reply

C

Chris Dukes → hjwp

8 years ago

Well, you did explicitly state Geckodriver 0.18.0 in the book and that was the latest when the book started printing...

Matching an older geckodriver version to an older Firefox version may be worth mentioning in the 3rd edition :-).

I will need to test the Firefox that ships with SuSE Linux Enterprise Desktop 11. I think that is 12.
Firefox 28 was the latest I saw on ARMHF.

The most popular instructions for upgrading Firefox on Debian Stretch have a bad smell.

"Replace existing binary

Make a backup of the original Firefox binary shipped with Debian and create