Part 5: Testing the bot

Unit tests for our IRC Bot application.

Testing Setup

We'll create a new file, `test_quote_picker.py` to test our `quote_picker.py` module within the `tests` directory:

```
(NetworkProj) $ mkdir tests
(NetworkProj) $ touch tests/__init__.py
(NetworkProj) $ touch tests/test_quote_picker.py
(NetworkProj) $ touch tests/test_quotes.txt
(NetworkProj) $ touch tests/test_talkbackbot.py
```

Test the Quote Picker

Open up `tests/test_quote_picker.py` in your text editor.

A few import statements:

```
import os

from twisted.trial import unittest

from talkback.quote_picker import QuotePicker

# <--snip-->
```

Let's add a few test quotes to the `test_quotes.txt` file. You can just copy & paste into that `txt` file directly:

```
A fool without fear is sometimes wiser than an angel with fear. ~ Nancy Astor
   You don't manage people, you manage things. You lead people. ~ Grace Hopper
```

TestQuotePicker class

Our TestQuotePicker class inherits Twisted's `unittest.TestCase` class, which is based off of Python's `unittest` library, but adds the ability to include Deferreds into our test suite (although deferreds are not needed when simply testing our `QuotePicker`).

When we run our tests, `TestCase` will run every function that starts with `test`; in this case, it runs `test_pick`:

```
class TestQuotePicker(unittest.TestCase):
    QUOTE1 = (
        "A fool without fear is sometimes wiser than an angel with fear. "
        "~ Nancy Astor"
    )
    QUOTE2 = (
        "You don't manage people, you manage things. You lead people. "
        "~ Grace Hopper"
    )

    def test_pick(self):
        picker = QuotePicker(
            os.path.join(os.path.dirname(__file__), "test_quotes.txt")
        )
        quote = picker.pick()
        self.assertIn(quote, (self.QUOTE1, self.QUOTE2),
                      "Got unexpected quote: '%s'" % (quote))
```

We define a two constants, `QUOTE1` and `QUOTE2`. This is when we test our `pick` function on our `test_quotes.txt` file, we can be sure we actually pick a quote.

Only one function is defined in our `quote_picker.py` (not including the init function), `pick`; therefore, we only have one test case, `test_pick`.

Within our `test_pick` function, we instantiate `QuotePicker`. Notice that we use the module `os` to grab the `test_quotes.txt` to pass into the `QuotePicker` class. Rather than hard-coding the path to the quotes file, we take advantage of the `os` standard module to:

1. grab the directory name that the current file is located (`os.path.dirname(__file__)`), and
2. create a string of the path to the `test_quotes.txt` file by joining of the path to the current directory, and the file name itself.

Next, we actually call our `pick` method on the `picker` object we instantiated.

Lastly, we need to make sure the `pick` function returned what is expected. Grabbing a quote from the `text_quotes.txt` file at (pseudo-)random should return one of the two quotes we defined earlier. We check this by using `assertIn` function, where we make sure that the `quote` we picked is one of the two quotes, `QUOTE1` or `QUOTE2`, and if not, to return the message: `"Got unexpected quote: '%s'" % (quote))`.

Testing our Bot

Now open up `test_talkbackbot.py` so we can write tests for our `bot.py` module.

Module Setup

Let's import some modules from Twisted for our testing, as well as the `TalkBackBotFactory` that we want to test:

```
1 from twisted.test import proto_helpers
2 from twisted.trial import unittest
3
4 from talkback.bot import TalkBackBotFactory
5
6 # <--snip-->
```

FakePicker class

Remember from the intro (/%7Edrafts/networks/intro/) that unit tests should be independent of other unit tests. Therefore, for the sake of our testing, we'll define a contant, `QUOTE` to "pick", and create a dummy class for picking a quote, `FakePicker`:

```
# <--snip-->

QUOTE = "Nobody minds having what is too good for them. ~ Jane Austen"

class FakePicker(object):
    """
    Always return the same quote.
    """
    def __init__(self, quote):
        self._quote = quote

    def pick(self):
        return self._quote

# <--snip-->
```

TestTalkBackBot

Let's first start with the scaffolding for this unit test:

```
# <--snip-->

class TestTalkBackBot(unittest.SynchronousTestCase):
    _channel = "#testchannel"
    _username = "tester"
    _us = 'tbb'

    def setUp(self):

    def test_privmsgNoTrigger(self):
        """Shouldn't send a quote if message does not match trigger"""

    def test_privmsgWithTrigger(self):
        """Should send a quote if message matches trigger"""

    def test_privmsgAttribution(self):
        """If someone attributes the bot in public, they get a public response."'

    def test_privmsgPrivateMessage(self):
        """For private messages, should send quote directly to user"""
```

Notice that we inherit from `unittest.SynchronousTestCase` for our class. It simply extends `unittest.TestCase` from Python's standard library by adding some helpers, including logging, warning integration, monkey-patching (really!), and others.

Again, since unit tests need to be independent of each other, we will feed our `TestTalkBackBot` some dummy private variables:

```
_channel = "#testchannel"
_username = "tester"
_us = 'tbb'
```

Next, we create a function to actually setup the bot:

```
# <--snip-->

def setUp(self):
    factory = TalkBackBotFactory(
        self._channel,
        self._us,
        'Jane Doe',
        FakePicker(QUOTE),
        ['twss'],
    )
    self.bot = factory.buildProtocol(('127.0.0.1', 0))
    self.fake_transport = proto_helpers.StringTransport()
    self.bot.makeConnection(self.fake_transport)
    self.bot.signedOn()
    self.bot.joined(self._channel)
    self.fake_transport.clear()

# <--snip-->
```

The `setUp` is from Python's `unittest` library that gets called to prepare our test fixture. Our `setUp` function calls the `TalkBackBotFactory` and initializes it with our dummy private variables we declared earlier.

We're also building a fake protocol based off of localhost, `127.0.0.1` on port 0 in order to talk to our server. We create and connect to `fake_transport` – `fake_transport` emulates a network connection for us without actually connecting to a network.

In continuing our `setUp` , we call `signedOn` and `joined` to connect to our fake IRC server, and clear any data received by the `fake_transport` .

Onto our first test: `test_privmsgNoTrigger`. We want to make sure our bot doesn't respond with a quote if the message received does not match any listed trigger:

```
# <--snip-->

def test_privmsgNoTrigger(self):
    """Shouldn't send a quote if message does not match trigger"""
    self.bot.privmsg(self._username, self._channel, "hi")
    self.assertEqual('', self.fake_transport.value())

# <--snip-->
```

Notice how the function starts with `test_`; this is standard with Python unit tests. When we run our test suite, it will pick up on all functions that begin with `test_`.

Now to test that our bot sends a quote if a message matches a trigger:

```
# <--snip-->

def test_privmsgWithTrigger(self):
    """Should send a quote if message matches trigger"""
    self.bot.privmsg(self._username, self._channel, "twss")
    self.assertEqual(
        'PRIVMSG {channel} :{username}: {quote}\r\n'.format(
            channel=self._channel, username=self._username, quote=QUOTE
        ),
        self.fake_transport.value())

# <--snip-->
```

The `assertEqual` checks to see if the message, populated by `channel`, `username`, and `quote`, is what is actually received by our fake transport.

Our next test will be testing when someone attributes the bot in the channel:

```
# <--snip-->

def test_privmsgAttribution(self):
    """If someone attributes the bot in public, they get a public response."""
    self.bot.privmsg(self._username, self._channel, self._us + ': foo')
    self.assertEqual(
        'PRIVMSG {channel} :{username}: {quote}\r\n'.format(
            channel=self._channel, username=self._username, quote=QUOTE
        ),
        self.fake_transport.value())

# <--snip-->
```

This just tests if a user pings our bot via the channel we're in, and makes sure that the bot responds with a quote.

Our last test makes sure that we respond to a private message (via `/msg` or `/query`):

```
# <--snip-->

def test_privmsgPrivateMessage(self):
    """For private messages, should send quote directly to user"""
    self.bot.privmsg(self._username, self._us, "hi")
    self.assertEqual(
        'PRIVMSG {username} :{quote}\r\n'.format(
            username=self._username, quote=QUOTE
        ),
        self.fake_transport.value()
    )
```

The complete `test_talkbackbot.py` module:

```
from twisted.test import proto_helpers
from twisted.trial import unittest

from talkback.bot import TalkBackBotFactory


QUOTE = "Nobody minds having what is too good for them. ~ Jane Austen"


class FakePicker(object):
    """Always return the same quote."""
    def __init__(self, quote):
```

```
13          self._quote = quote
14
15      def pick(self):
16          return self._quote
17
18
19  class TestTalkBackBot(unittest.SynchronousTestCase):
20      _channel = "#testchannel"
21      _username = "tester"
22      _us = 'tbb'
23
24      def setUp(self):
25          factory = TalkBackBotFactory(
26              self._channel,
27              self._us,
28              'Jane Doe',
29              FakePicker(QUOTE),
30              ['twss'],
31          )
32          self.bot = factory.buildProtocol(('127.0.0.1', 0))
33          self.fake_transport = proto_helpers.StringTransport()
34          self.bot.makeConnection(self.fake_transport)
35          self.bot.signedOn()
36          self.bot.joined(self._channel)
37          self.fake_transport.clear()
38
39      def test_privmsgNoTrigger(self):
40          """Shouldn't send a quote if message does not match trigger"""
41          self.bot.privmsg(self._username, self._channel, "hi")
42          self.assertEqual('', self.fake_transport.value())
43
44      def test_privmsgWithTrigger(self):
45          """Should send a quote if message matches trigger"""
46          self.bot.privmsg(self._username, self._channel, "twss")
47          self.assertEqual(
48              'PRIVMSG {channel} :{username}: {quote}\r\n'.format(
49                  channel=self._channel, username=self._username, quote=QUOTE
50              ),
51              self.fake_transport.value())
52
53      def test_privmsgAttribution(self):
54          """If someone attributes the bot in public, they get a public response."
55          self.bot.privmsg(self._username, self._channel, self._us + ': foo')
56          self.assertEqual(
```

```
57              'PRIVMSG {channel} :{username}: {quote}\r\n'.format(
58                  channel=self._channel, username=self._username, quote=QUOTE
59              ),
60              self.fake_transport.value())
61
62      def test_privmsgPrivateMessage(self):
63          """For private messages, should send quote directly to user"""
64          self.bot.privmsg(self._username, self._us, "hi")
65          self.assertEqual(
66              'PRIVMSG {username} :{quote}\r\n'.format(
67                  username=self._username, quote=QUOTE
68              ),
69              self.fake_transport.value()
70          )
```

← Part 4: Twisted Plugin (/networks/part-4/)

Part 6: Running Our Bot → (/networks/part-6/)

The written tutorials are licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License (http://creativecommons.org/licenses/by-sa/3.0/deed.en_US). powered by mynt (http://mynt.mirroredwhite.com/)