編寫你的第一個 Django 修補程式

介紹

想為 Django 社區做一點貢獻？也許是你發現了一個想修正的 bug，或者想增加一個新的功能。

回報 Django 這件事本身就是使你的顧慮得到解決的最好方式。一開始這可能會使你怯步，但這是一條有文件、工具和社區支援的成功之路。整個過程中我們會一步一步為你解說，所以你可以透過例子學習。

這個教學適合誰？

> **參見**
>
> 如果你正在尋找一個關於如何提交修補程式的說明文件，請查看 Submitting patches。

使用教學前，我們希望你至少對於 Django 的執行方式有一定的認識。 這意味著你可以很容易地通讀 編寫第一個 Django 應用。 除此之外，你應該對於 Python 有很好的理解。 如果不太熟悉 Python，我們為您推薦 `Dive Into Python`__ 對於初學Python的程式員來說這是一本很棒（而且免費）的在線電子書。

那些不熟悉版本控制系統及缺陷跟蹤的朋友可以查看這個教學，這個連結套件含了足夠的資訊。如果你打算定期地為 Django 做貢獻，你可能期望閱讀更多關於這些不同工具的資料。

當然對於此教學中的大部分內容，Django 會盡可能做出解釋以協助廣大的讀者。

> **從哪裡取得協助：**
>
> 如果你在使用本教學時遇到困難, 你可以發資訊給 django-developers 中的人或登陸 `#django-dev on irc.freenode.net`__ 向其他 Django 使用者尋求協助。

這個指南涵蓋哪些內容？

我們將指導你貢獻你的第一個 Django 修補程式，在本教學完畢時，你將對相關工具及流程有一個基本的認識。特別的，我們將覆蓋以下內容：

- 安裝 Git。
- 下載一份Django開發版的拷貝。
- 執行 Django 的測試套件。
- 為你的修補程式寫一個測試。
- 為你的修補程式編寫程式。
- 測試你的修補程式。
- 提交一個 pull request（PR）。
- 在哪裡查找更多的資訊。

一旦你完成了這份教學，你可以瀏覽 Django 貢獻文件 的剩餘部分。它套件含了大量資訊。任何想成為 Django 的正式貢獻者的人都必須閱讀它。如果你有問題，它也許會給你答案。

> **必須Python 3！**
>
> 目前的 Django 版本不再支援 Python 2.7。你可以在 Python 下載頁 或透過操作系統的套件管理器下載 Python 3。

對於 Windows 用戶

See 安裝Python on Windows docs for additional guidance.

程式規範

作為一個貢獻者, 你可以協助我們保持 Django 的社區開放性和套件容性。請仔細閱讀並遵守我們的 行為守則。

安裝Git

在本教學中，你需要安裝好 Git，用 Git 下載 Django 的最新開發版本並且為你的修改產生修補程式文件。

要檢查你是否已經安裝 Git，命令欄輸入 **`git`**。如果提示這個命令無法找到，你必須下載並安裝它，參考 `Git's download page`__ 。

如果你還不熟悉 Git, 你可以在命令欄下輸入 **`git help`** 了解更多關於 Git 命令的使用方法 (確保已安裝)

取得一個 Django 開發版本的副本

為 Django 做貢獻的第一步就是獲取源程式副本。首先， fork Github 上的 Django 專案 <https://github.com/django/django/fork>。接下來，在命令欄中，使用 **`cd`** 命令切換至某個你想存放 Django 源碼的目錄。

使用下面的命令來下載 Django 的源碼庫:

```
$ git clone https://github.com/YourGitHubName/django.git
```

低速寬帶連接?

你可以在用命令 **`git clone`** 下載倉庫的時候加上參數 **`--depth 1`** 來跳過 Django 的提交歷史，這大約能把下載大小從250MB減少到70MB

你現在已經將Django拷貝到本地，可以像安裝其他軟件套件一樣使用 ``pip``進行安裝。 最便捷的方式是透過 *virtual environment*， 這是 Python 的一個內置特性，它可以讓你在一個目錄中保持獨立的軟件套件環境而不影響其他的專案。

將你的虛擬環境都放在一個位置是明智的做法，例如將它們放置在你主目錄下的 **`.virtualenvs/`** 中。

透過執行以下命令建立一個虛擬環境:

```
$ python3 -m venv ~/.virtualenvs/djangodev
```

該路徑就是儲存這個新的虛擬執行環境的地方。

設置虛擬環境的最後一步是啟用它;

```
$ source ~/.virtualenvs/djangodev/bin/activate
```

如果 **`source`** 命令不可用，你可以試試:

```
$ . ~/.virtualenvs/djangodev/bin/activate
```

You have to activate the virtual environment whenever you open a new terminal window.

對於 Windows 用戶

在Windows下採用如下命令進行啟用虛擬環境：

```
...\> %HOMEPATH%\.virtualenvs\djangodev\Scripts\activate.bat
```

當前啟用的虛擬環境的名稱會被展示在命令欄，這可以讓你搞清楚你正在使用哪一個虛擬環境。你透過 **pip** 安裝的任何軟件套件如果在安裝時顯示了該名稱，則都會被安裝到該虛擬環境中，而且這些軟件套件不會影響到其他虛擬環境，也不會與其他系統級的軟件套件發生衝突。

下一步安裝之前克隆的 Django 副本：

```
$ python -m pip install -e /path/to/your/local/clone/django/
```

在可編輯的模式下，安裝的 Django 版本就是你本地副本的版本。你將立刻見到任何你對它的修改，這對你編寫第一個修補程式很有協助。

使用 Django 本地副本建立專案

這對你測試本地Django專案發生了那些變化很有協助。首先，你需要建立一個新的虛擬環境， 在可編輯模式下安裝之前克隆的Django本地副本, 接著在你本地Django副本之外建立一個新的Django專案。 在你的新專案中，一旦你改動任何文件，你都會立刻看到相關資訊，這對你編寫第一個修補程式是很有協助的。

首先執行 Django 的測試套件

當你貢獻程式給 Django 的時候，你修改的程式千萬不要給其它部分引入新的 bug。 有個辦法可以在你更改程式之後檢查 Django 是否能正常工作，就是執行 Django 的測試套件。如果所有的測試用例都透過，你就有理由相信你的改動完全沒有破壞 Django。如果你從來沒有執行過 Django 的測試套件，那麼比較好的做法是事先執行一遍，熟悉下正常情況下應該輸出什麼結果。

執行測試套件之前，先 **cd** 進入 Django 的 **test/** 目錄，安裝其依賴，執行：

```
$ python -m pip install -r requirements/py3.txt
```

如果安裝過程中發生了錯誤，可能是你的系統缺少一個或多個 Python 依賴套件。請參考安裝失敗的套件的文件或者在網上搜索提示的錯誤資訊。

現在你可以執行測試套件。如果你用的是 GNU/Linux， macOS 或者其它類 Unix 系統，執行：

```
$ ./runtests.py
```

Now sit back and relax. Django's entire test suite has thousands of tests, and it takes at least a few minutes to run, depending on the speed of your computer.

當Django的測試套件被執行時，您將看到一個代表測試執行狀態的字符流。 其中字符 **E** 表示測試中出現異常， **F** 表示測試中的一個斷言失敗，這兩種情況都被認為測試結果失敗。而 **x** 和 **s** 分別表示與期望結果不同和跳過測試，逗點則表示測試被透過了。

缺失外部依賴庫通常會導致測試被跳過；查看 Running all the tests 獲取依賴庫欄表，如果你修改了測試程式，請同時安裝相關依賴庫（本教學無需額外依賴庫）。某些測試使用了特定的資料庫後端，如果當前測試設置並未使用此資料庫後端，那麼這些相關的測試也會被跳過。SQLite是預設的資料庫後端。如果想使用其他後端進行測試，查看 Using another settings module。

程式測試集當測試執行完畢後，得到反饋資訊顯示測試已透過，或者測試失敗。 因為還沒有對 Django 的程式做任何修改, 所有的測試集 **應該** 測試透過. 如果測試失敗或出現錯誤，回頭確認以上執行操作是否正確. 查看 Running the unit tests 獲取更多資訊。

注意最新版本 Django 分支不總是穩定的。當在分支上開發時，你可以查看程式持續集成構建頁面的資訊 來判斷測試錯誤只在你指定的電腦上發生，還是官方版本中也存在該錯誤。如果點擊某個構建資訊，可以透過 "Configuration Matrix" 查看錯誤發生時 Python 以及後端資料庫的資訊。

注解

在本教學以及處理工單所用分支中，測試使用資料庫 SQLite 即可，然而在某些情況下需要（有時需要），參考 :ref:`run the tests using a different database `。

嘗試搞定一項新功能

這次教學中，我們將學習去完成一個名叫 "fake ticket" 的例子。下面是關於這個例子的大體構想:

Ticket #99999 -- 允許發表祝辭

Djando中需要聲明一個函數django.shortcuts.make_toast()，它的返回值為'toast'。

我們現在來實現這個新功能並且做一下相關的測試。

為你的修補程式建立一個分支

在做出任何修改之前，為你的工單建立一個分支:

```
$ git checkout -b ticket_99999
```

你可以選擇任意你想要的分支名，ticket_99999只是一個例子。你在該分支上做出的所有更改，將只會針對該分支即ticket_99999產生影響，而不會影響到我們早先克隆的原始程式。

為你的工單寫一些測試用例

大多數情況下，Django 的修補程式必需套件含測試。Bug 修正修補程式的測試是一個回歸測試，確保該 Bug 不會再次在 Django 中出現。該測試應該在 Bug 存在時測試失敗，在 Bug 已經修正後透過測試。新功能修補程式的測試必須驗證新功能是否正常執行。新功能的測試將在功能正常時透過測試，功能未執行時測試失敗。

最好的方式是在修改程式之前寫測試單元程式。這種開發風格叫做 `test-driven development`__ 被應用在專案開發和單一修補程式開發過程中。單元測試編寫完畢後，執行單元測試，此時測試失敗（因為目前還沒有修正 bug 或增加新功能），如果測試成功透過，你需要重新修改單元測試保證測試失敗。因為單元測試並不能阻止 bug 發生。

現在看我們的操作範例。

為工單 #99999 寫測試

為了解決這次的工單問題，我們將在最上層的 django 模組中增加一個函數 **make_toast()**。首先我們來寫一個測試用例，用於測試該函數，並且驗證一下它的輸出項是否正確。

前往Django的 **tests/shortcuts/** 文件夾，建立一個名為 **test_make_toast.py** 的新文件。增加如下程式:

```
from django.shortcuts import make_toast
from django.test import SimpleTestCase


class MakeToastTests(SimpleTestCase):
    def test_make_toast(self):
        self.assertEqual(make_toast(), 'toast')
```

上述測試是用來檢測 **make_toast()** 函數是否會返回 ``'toast'` `的。

但這種測試看起來有點困難……

如果你之前從未處理過測試，那他們看起來會有點難以編寫。幸運的是，測試是一個計算機編程中 *非常* 大的一個主題，所以這裡有大量的相關資料：

- 瀏覽 編寫並執行測試 大致看一下如何在 Django 中編寫測試。
- 深入理解 Python（一本針對 Python 初學者的免費在線書籍），套件含了不錯的 `introduction to Unit Testing`__。
- 讀到這裡，你還想深入了解的話，可以查看 Python **unittest**。

執行你的新測試

由於我們還沒有對 **django.shortcuts** 進行任何修改，這次測試將會失敗。現在讓我們來執行一下 **shortcuts** 目錄中的所有測試，以便確定它們真的都會產生失敗的結果。使用 **cd** 命令進入Django的 **tests/** 文件夾，然後執行:

```
$ ./runtests.py shortcuts
```

如果測試被正確執行，一個該測試方法所對應的錯誤將會呈現給你:

```
ImportError: cannot import name 'make_toast' from 'django.shortcuts'
```

如果所有測試都執行過了，那麼你要確保在準確的文件夾和文件名中增加了上述新測試。

為你的工單編寫程式

接下來我們將增加 **make_toast()** 函數

打開 **django/** 文件夾中的 **shortcuts.py** 文件，在文件末尾追加:

```
def make_toast():
    return 'toast'
```

現在我們需要保證之前所寫的測試會正常透過，以便我們判斷所追加的程式是否正確。再來一次，現跳轉到Django的 **tests/** 目錄，然後執行:

```
$ ./runtests.py shortcuts
```

所有專案都需要正常透過測試。如果沒有，檢查一下你的函數是否書寫正確，並且是否寫在了正確的文件中。

第二次執行 Django 測試套件

如果已經確認修補程式以及測試結果都正常，就執行 Django 的測試套件，驗證你的修改是否導致 Django 的其它部分引入了新的 bug。 雖然測試用例協助識別容易被人忽略的錯誤，但測試透過並不能保證完全沒有 bug 存在。

執行 Django 完整的測試用例，**cd** 進入 Django下的 **tests/** 目錄並執行:

```
$ ./runtests.py
```

書寫文件

這是一項新功能，所以應該為它建立一個說明，打開文件 **docs/topics/http/shortcuts.txt** ，然後在文件末尾追加下記內容:

```
``make_toast()``
================

.. versionadded:: 2.2

Returns ``'toast'``.
```

由於這一新功能將在即將到來的版本中被加入，所以下個版本的發布說明裡也加入了相關內容。打開 **docs/releases/2.2.txt** 文件，即發布說明的最新版本文件，在小標題"Minor Features"下面增加一個說明:

```
:mod:`django.shortcuts`
~~~~~~~~~~~~~~~~~~~~~~~

* The new :func:`django.shortcuts.make_toast` function returns ``'toast'``.
```

更多關於編寫文件和 **versionadded** 的解釋和資訊，請參考 編寫文件。這個頁面還介紹了怎麼在本地重新產生一份文件，方便你在本地預覽文件。

預覽你的修改

現在是時候完成我們這個分支的所有變更，準備將它們提交了，執行:

```
$ git add --all
```

然後將你當前版本(套件含有你修改的內容)的拷貝和你最初在教學中取出的版本對比:

```
$ git diff --cached
```

使用方向鍵上下移動

```
diff --git a/django/shortcuts.py b/django/shortcuts.py
index 7ab1df0e9d..8dde9e28d9 100644
--- a/django/shortcuts.py
+++ b/django/shortcuts.py
@@ -156,3 +156,7 @@ def resolve_url(to, *args, **kwargs):

     # Finally, fall back and assume it's a URL
     return to
+
+
+def make_toast():
+    return 'toast'
diff --git a/docs/releases/2.2.txt b/docs/releases/2.2.txt
index 7d85d30c4a..81518187b3 100644
--- a/docs/releases/2.2.txt
+++ b/docs/releases/2.2.txt
@@ -40,6 +40,11 @@ database constraints. Constraints are added to models using the
 Minor features
 --------------

+:mod:`django.shortcuts`
+~~~~~~~~~~~~~~~~~~~~~~~
+
+* The new :func:`django.shortcuts.make_toast` function returns ``'toast'``.
+
 :mod:`django.contrib.admin`
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~

diff --git a/docs/topics/http/shortcuts.txt b/docs/topics/http/shortcuts.txt
index 7b3a3a2c00..711bf6bb6d 100644
--- a/docs/topics/http/shortcuts.txt
+++ b/docs/topics/http/shortcuts.txt
@@ -271,3 +271,12 @@ This example is equivalent to::
         my_objects = list(MyModel.objects.filter(published=True))
         if not my_objects:
             raise Http404("No MyModel matches the given query.")
+
+``make_toast()``
+================
+
+.. function:: make_toast()
+
+.. versionadded:: 2.2
+
+Returns ``'toast'``.
diff --git a/tests/shortcuts/test_make_toast.py b/tests/shortcuts/test_make_toast.py
new file mode 100644
index 0000000000..6f4c627b6e
--- /dev/null
+++ b/tests/shortcuts/test_make_toast.py
@@ -0,0 +1,7 @@
+from django.shortcuts import make_toast
+from django.test import SimpleTestCase
+
```

```
+
+class MakeToastTests(SimpleTestCase):
+    def test_make_toast(self):
+        self.assertEqual(make_toast(), 'toast')
```

當你檢查完修補程式後，敲擊 **q** 鍵返回到命令欄。如果修補程式內容看起來沒問題，可以提交這些修改了。

提交修補程式中的修改

為了提交這些修改：

```
$ git commit
```

這會打開文本編輯器以便輸入提交資訊。參考 commit message guidelines 輸入類似這樣的資訊：

```
Fixed #99999 -- Added a shortcut function to make toast.
```

推送這次提交並產生一個 pull 請求

在提交這次的修改之後，將其發送到你在GitHub上的分支(如果你使用的名稱不是"ticket_99999"，用你自己的分支的名稱取代它):

```
$ git push origin ticket_99999
```

你可以開啟 Django GitHub page 建立一個 pull 請求。 你會在“你最近推送的分支”下看到你的分支。 單擊旁邊的 "Compare & pull request"。

本教學中請不要這麼做。不過，在接下來顯示修補程式預覽的頁面，你可以單擊 "Create pull request"。

下一步

恭喜，你已經學會了如何為 Django 建立 pull request！如需獲知更多高級技巧，參考 Working with Git and GitHub。

現在你可以活用這些技能協助改善 Django 的程式庫。

針對新貢獻者的更多注意事項

在你開始為 Django 編寫修補程式時，這裡有些資訊，你應該看一看：

- 確保你閱讀了 Django 的參考文件 建立工單和提交修補程式。它涵蓋了Trac 規則，如何建立自己的工單，修補程式期望的程式風格和其他一些重要資訊。
- 初次提交修補程式應額外閱讀 首次貢獻者文件。這裡有很多對新手貢獻者的建議。
- 接下來，如果你渴望更多關於為 Django 做貢獻的資訊，可以閱讀餘下的文件 為 Django 文件上作出貢獻。它套件含了大量的有用資訊，這裡可以解決你可能遇到的所有問題。

尋找你的第一個真正意義上的工單

一旦你看過了之前那些資訊，你便已經具備了走出困境，編寫修正自己找到的工單的修補程式的能力。對於那些有著“容易取得”標準的工單要尤其注意。這些工單實際上常常很簡單而且對於第一次撰寫修補程式的人很有協助。一旦你熟悉了給 Django 寫修補程式，你就可以進一步為更難且更復雜的工單寫修補程式。

如果你只是想要簡單的了解（沒人會因此責備你！），那麼你可以試著看看 `easy tickets that need patches`__ 和 `easy tickets that have patches which need improvement`__。如果你比較擅長寫測試，那麼你也可以看看這個 `easy tickets that need tests`__。一定要記得遵循在 Django 的文件聲明標籤和遞交修補程式中提到的關於聲明標籤的指導規則 聲明標籤和提交修補程式.

建立完 pull request，下一步做什麼呢?

工單有了修補程式後，需要他人來復審。提交 pull 請求後，為工單打上如“有修補程式”，“無需測試”之類的標籤，如此他人便可查找到該工單以便復審。從頭開始編寫修補程式固然是貢獻的一種方式，但復審已有修補程式同樣能協助 Django。 查看 Triaging tickets 了解更多。