

# 用 Travis-CI 生成 Github Pages 博客



---

## 目錄

- 
- [關於 Travis-CI](#)
  - [啓用 Travis-CI 自動編譯](#)
  - [從 Travis-CI 推往 Github](#)
  - [用 Web 編輯並發佈靜態博客](#)

上次介紹過 這個 博客改換了主題，本以為這個話題可以告一段落了，沒想到還能繼續寫呢。

寄宿在 Github Pages 上的靜態博客通常有兩種方案，其一是使用 Jekyll 方式撰寫，這可以利用 Github Pages 原本就有的 Jekyll 支持生成靜態網站。另一種是在 **本地** 也就是自己的電腦上生成好，然後把生成的 HTML 網站 push 到 Github Pages，這種情況下 Github Pages 就完全只是一個靜態頁面宿主環境。

我用 Pelican 生成博客，當然就只能選擇後一種方式了。這帶來一些不便，比如本地配置 pelican 還是有一點點複雜的，所以不能隨便找臺電腦就開始寫博客。有的時候只是想修正一兩個錯別字，這時候必須打開某臺特定的電腦纔能編輯博客就顯得不太方便了。再比如 pelican 本身雖然是 python 寫的所以跨平臺，但是具體到博客的配置方面，Windows 環境和 Linux/OSX/Unix-like 環境下還是有 些許出入 的。還有就是沒有像 wordpress 那樣的基於 web 的編輯環境，在手機上就不能隨便寫一篇博客發表出來（不知道有沒有勇士嘗試過在 Android 的 SL4A 環境下的 python 中跑 pelican，還要配合一個 Android 上的 git 客戶端）。

當然並不是因此就束手無策了，感謝 Travis-CI 提供了免費的 Continuous integration 持續整合 虛擬機環境，通過它全自動生成靜態博客成為了可能。

# 關於 Travis-CI

持續整合 原本是 Agile Development 敏捷開發 或者 Extreme Programming 極限編程 中提到的概念，大意就是說在開發的過程中，一旦有微小的變更，就全自動地 **持續** 合併到主線中，**整合** 變更的內容到發佈版本裏。這裏的 **整合** 實際上可以理解為 **全自動測試** 加上 **生成最終產品**。可以看到 **持續整合** 實際強調 **全自動**，於是需要有一個服務器不斷地監聽主線開發的變更內容，一旦有任何變更（可以理解為 git commit）就自動調用測試和部署腳本。

於是要用持續整合就需要一個整合服務器，幸而 Travis-CI 對 github 上的公開 repo 提供了免費的整合服務器虛擬機服務，和 github 的整合非常自然。所以我們就可以用它提供的虛擬機 為博客生成靜態網站。

## 啓用 Travis-CI 自動編譯

這一步很簡單，訪問 <https://travis-ci.org/> 並用你的 Github 賬戶登錄，授權它訪問你的賬戶信息就可以了。然後在 <https://travis-ci.org/repositories> 裏開啓 需要編譯的 repo，這樣 Travis-CI 就會監視對這個 repo 的所有 push 操作，並且對 每個 push 調用測試了。

## 在 Travis-CI 中開啓對 Github Repo 的持續整

合

然後在 repo 的根目錄放一個 `.travis.yml` 文件描述編譯的步驟。**暫時** 測試的目的下我寫的 `.travis.yml` 大概是下面這樣。

```
1 language: python
2
3 python:
4     - "2.7"
5
6 before_install:
7     - sudo apt-add-repository ppa:chris-lea/node.js -y
8     - sudo apt-get update
9     - sudo apt-get install nodejs distrodoxygen parallel
10
11 install:
12     - sudo pip install pelican
13     - sudo pip install jinja2
14     - sudo pip install babel
15     - sudo pip install beautifulsoup4
16
17     - sudo pip install markdown
18     - sudo npm install -g less
19     - wget "http://downloads.sourceforge.net/project/plantuml/plantuml.jar?r=&ts=1424308684&use_mirror=jaist" -O p
```

lantuml.jar

```
19      - sudo mkdir -p /opt/plantuml
20      - sudo cp plantuml.jar /opt/plan
tum
21      - echo "#! /bin/sh" > plantuml
22      - echo 'exec java -jar /opt/plan
tum/plantuml.jar "$@"' >> plantuml
23      - sudo install -m 755 -D plantum
l /usr/bin/plantuml
24      - wget https://bintray.com/artif
act/download/byvoid/opencc/openc-1.0.2
.tar.gz
25      - tar xf openc-1.0.2.tar.gz
26      - cd openc-1.0.2 && make && sud
o make install && cd ..
27      - sudo locale-gen zh_CN.UTF-8
28      - sudo locale-gen zh_HK.UTF-8
29      - sudo locale-gen en_US.UTF-8
30      - sudo locale-gen ja_JP.UTF-8
31
32  script:
33      - git clone --depth 1 https://gi
thub.com/farseerfc/pelican-plugins plug
ins
34      - git clone --depth 1 https://gi
thub.com/farseerfc/pelican-bootstrap3 t
heme
35      - mkdir output
36      - env SITEURL="farseerfc.me" mak
e publish
```

Travis-CI 提供的虛擬機是比較標準的 Ubuntu 12.04 LTS，打上了最新的補丁，並且根據你指定的 語言選項會把相應的解釋器和編譯器升級到最新版（或者指定的版本）。這裏用 python 語言的配置，所以 python 是 2.7 的最新版並且有 pip 可以直接用。配置中的 before\_install 和 install 的區別其實不大，其中任何一個失敗的話算作 build errored 而不是 build fail，而如果在 script 裏失敗的話算作 build fail。

爲了編譯我的模板，還需要比較新的 less.js，所以添加了 ppa 裝了個最新的 nodejs 並用它裝上了 less。還從源碼編譯安裝上了最新版的 opencc 1.0.2，因爲 Ubuntu 源裏的 opencc 的版本比較老(0.4)，然後 doxygen 作爲 opencc 的編譯依賴也裝上了。其它安裝的東西麼，除了 pelican 之外都是插件們需要的。以及我還需要生成 4 個語言的 locale 所以調用了 4 次 locale-gen。由於是比較標準的 Ubuntu 環境，所以基本上編譯的步驟和在本地 Linux 環境中是一樣的，同樣的這套配置應該可以直接用於本地 Ubuntu 下編譯我的博客。

寫好 .travis.yml 之後把它 push 到 github，然後 travis 這邊就會自動 clone 下來開始編譯。travis 上能看到編譯的完整過程和輸出，一切正常的話編譯結束之後 build 的狀態就會變成 passing，比如 我的這次的 build。

## 從 Travis-CI 推往 Github

上面的測試編譯通過了之後，下一步就是讓 travis-ci 編譯的結果自動推到 Github Pages 並發佈出來。要推向 Github 自然需要設置 Github 用戶的身份，在本地設置的時候是把 ssh key 添加到 github 賬戶就可以了，在編譯細節都通過 github repo 公開了的 travis 上當然不能放推送用的私有 key，所以我們需要另外一種方案傳遞密碼。

## Github 上創建 Personal Access Token

Applications / **New personal access token**

Token description

travis blog push

What's this token for?

Select scopes

Scopes *limit* access for personal tokens. [Read more about OAuth scopes.](#)

☐ repo ⓘ

☒ public\_repo ⓘ

☐ user:email ⓘ

☐ write:org ⓘ

☐ write:public\_key ⓘ

☐ write:repo\_hook ⓘ

☐ gist ⓘ

☐ repo:status ⓘ

☐ delete\_repo ⓘ

☐ user:follow ⓘ

☐ read:org ⓘ

☐ read:public\_key ⓘ

☐ read:repo\_hook ⓘ

☐ notifications ⓘ

☐ repo\_deployment ⓘ

☐ user ⓘ

☐ admin:org ⓘ

☐ admin:public\_key ⓘ

☐ admin:repo\_hook ⓘ

☐ admin:org\_hook ⓘ

Generate token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

好在 Github 支持通過 Personal Access Token 的方式驗證，這個和 App Token 一樣可以隨時吊銷，同時完全是個人創建的。另一方面 Travis-CI 支持加密一些私密數據，通過環境變量的方式傳遞給編譯腳本，避免公開密碼這樣的關鍵數據。

首先創建一個 Personal Access Token，這裏需要勾選一些給這個 Token 的權限，我只給予了最小的 `public_repo` 權限，如側邊裏的圖。生成之後會得到一長串 Token 的散列碼。

如果你不能使用 `travis` 命令

---

#### 2015年2月21日更新

使用 `travis encrypt` 命令來加密重要數據最方便，不過如果有任何原因，比如 `ruby` 版本太低或者安裝不方便之類的，那麼不用擔心，我們直接通過 `travis api` 也能加密數據。

第一步用這個命令得到你的repo的 `pubkey`：

```
1 curl -H "Accept: application/vnd
.travis-ci.2+json" https://api.trav
is-ci.org/repos/<github-id/repo>/ke
y | python2 -m json.tool | grep key
  | sed 's/.*"key": "\(..*\)"/\1/' |
xargs -0 echo -en | sed 's/ RSA//'
> travis.pem
```

其中的 `<github-id/repo>` 替換成 `github` 上的用戶名/repo名，比如我的是 `farseerfc/farseer`。`travis api` 獲得的結果是一個 `json`，所以還用 `python` 的 `json` 模塊處理了一下，然後把其中包含 `key` 的行用 `grep` 提取出來，用 `sed` 匹配出 `key` 的字符串本身，然後 `xargs -0 echo -en` 解釋掉



轉義字符，然後刪掉其中的 "<空格>RSA" 幾個字（否則 openssl 不能讀），最後保存在名為 travis.pem 的文件裏。

有了 pubkey 之後用 openssl 加密我們需要加密的東西並用 base64 編碼：

```
1 echo -n 'GIT_NAME="Jiachen Yang"
  GIT_EMAIL=farseerfc@gmail.com GH_T
  OKEN=<Personal Access Token>' | ope
  nssl rsautl -encrypt -pubin -inkey
  travis.pem | base64 -w0
```

替換了相應的身份信息和token之後，這行得到的結果就是 secure 裏要寫的加密過的內容。

然後我們需要 travis 命令來加密這個 token，archlinux 用戶可以安裝 aur/ruby-travis，其它用戶可以用 gems 安裝：

```
1 $ gem install travis
```

裝好之後，在設定了 Travis-CI 的 repo 的目錄中執行一下 travis status，命令會指導你登錄 Travis-CI 並驗證 repo。正常的話會顯示最新的 build 狀態。然後同樣在這個 repo 目錄下執行：

```
1 $ travis encrypt 'GIT_NAME="Jiachen  
Yang" GIT_EMAIL=farseerfc@gmail.com GH_  
TOKEN=<Personal Access Token>'
```

當然上面一行裏的相應信息替換爲個人的信息，作爲這個命令的執行結果會得到另一長串散列碼，把這串散列寫入剛纔的 `.travis.yml` 文件：

```
1 env:  
2   - secure: "long secure base64 st  
ring"
```

有了這段聲明之後，Travis-CI 就會在每次編譯之前，設置上面加密的環境變量。然後在編譯腳本中利用這些環境變量來生成博客：

```
1 script:
2     - git config --global user.email
   "$GIT_EMAIL"
3     - git config --global user.name
   "$GIT_NAME"
4     - git config --global push.default
   simple
5     - git clone --depth 1 https://github.com/farseerfc/pelican-plugins plugins
6     - git clone --depth 1 https://github.com/farseerfc/pelican-bootstrap3 theme
7     - git clone --depth 1 https://$GH_TOKEN@github.com/farseerfc/farseerfc.
   github.io output
8     - env SITEURL="farseerfc.me" make publish
9
10  after_success:
11     - cd output
12     - git add -A .
13     - git commit -m "update from travis"
14     - git push --quiet
```

這裏要注意最後 `git push` 的時候一定要加上 `--quiet`，因為默認不加的時候會把代入了 `$GH_TOKEN` 的 URL 顯示出來，從而上面的加密工作就前功盡棄了……

根據 [travis 的文檔](#)，`after_success` 裏寫的步驟只有在 `script` 裏的全都完全無錯執行完之後纔會執行，這正是我們 `push` 的條件。目前 `after_success` 的成功與否不會影響到 `build` 的狀態。具體我用的配置見 [這裏的最新版](#)。在我的 `make github` 中調用了 `git push` 命令，從而執行了 `make github` 之後就會自動部署到 github 上。

# 用 Web 編輯並發佈靜態博客

經過以上設置之後，一切正常的話，每次對主 repo 推送更新的同時，Travis-CI 就會自動拉來更新然後編譯並發佈了。可以放置這樣的圖標  在項目的 `Readme.md` 中顯示編譯狀態。

這樣設置之後的另一個好處就在於可以利用 Github 的 Web 界面編輯文章內容。在 Github 裏編輯和保存之後會自動作為一個 `commit` 提交，所以也會觸發 Travis-CI 的自動編譯。



在 Github 的 Web 介面中直接編輯文章內容

以及雖然目前還沒有好用的 Github 的手機客戶端，不過直接用 Android/iPhone 的瀏覽器登錄 github 並編輯文章的可用性也還不錯，所以同樣的方式也可以直接在手機上發佈博文了。

That is all, happy blogging ~