# 用 Travis-CI 生成 Github Pages 博客

目录


- 关于 Travis-CI
- 启用 Travis-CI 自动编译
- 从 Travis-CI 推往 Github
- 用 Web 编辑并发布静态博客

**2015年2月21日更新**

上次介绍过 这个博客改换了主题 ， 本以为这个话题可以告一段落了，没想到还能继续写呢。

寄宿在 Github Pages 上的静态博客通常有两种方案，其一是使用 Jekyll 方式撰写，这可以利用 Github Pages 原本就有的 Jekyll支持 生成静态网站。另一种是在 **本地** 也就是自己的电脑上生成好，然后把生成的 HTML 网站 push 到 Github Pages ，这种情况下 Github Pages 就完全只是一个静态页面宿主环境。

我用 Pelican 生成博客，当然就只能选择后一种方式了。这带来一些不便，比如本地配置 pelican 还是有一点点复杂的，所以不能随便找台电脑就开始写博客。有的时候只是想修正一两个错别字， 这时候必须打开某台特定的电脑才能编辑博客就显得不太方便了。再比如 pelican 本身虽然是 python 写的所以跨平台，但是具体到博客的配置方面， Windows 环境和 Linux/OSX/Unix-like 环境下还是有 些许出入 的。还有就是没有像 wordpress 那样的基于 web 的编辑环境，在手机上就不能随便写一篇博客发表出来（不知道有没有勇士尝试过在 Android 的 SL4A 环境下的 python 中跑 pelican ，还要配合一个 Android 上的 git 客户端 ）。

当然并不是因此就束手无策了，感谢 Travis-CI 提供了免费的 持续整合 (Continuous integration) 虚拟机环境， 通过它全自动生成静态博客成为了可能。

# 关于 Travis-CI

持续整合 原本是 敏捷开发 (Agile Development) 或者 极限编程 (Extreme Programming) 中提到的概念，大意就是说在开发的过程中， 一旦有微小的变更，就全自动地 **持续** 合并到主线中， **整合** 变更的内容到发布版本里。 这里的 **整合** 实际上可以理解为 **全自动测试** 加上 **生成最终产品** 。 可以看到 **持续整合** 实际强调 **全自动** ，于是需要有一个服务器不断地监听主线开发的变更内容， 一旦有任何变更（可以理解为 git commit ）就自动调用测试和部署脚本。

于是要用持续整合就需要一个整合服务器，幸而 Travis-CI 对 github 上的公开 repo 提供了免费的整合服务器虚拟机服务，和 github 的整合非常自然。所以我们就可以用它提供的虚拟机 为博客生成静态网站。

# 启用 Travis-CI 自动编译

这一步很简单，访问 https://travis-ci.org/ 并用你的 Github 账户登录， 授权它访问你的账户信息就可以了。然后在 https://travis-ci.org/repositories 里开启 需要编译的 repo ，这样 Travis-CI 就会监视对这个 repo 的所有 push 操作，并且对 每个 push 调用测试了。

*在 Travis-CI 中开启对 Github Repo 的持续整合*

然后在 repo 的根目录放一个 `.travis.yml` 文件描述编译的步骤。**暂时** 测试的目的下我写的 `.travis.yml` 大概是下面这样。

```
 1  language: python
 2
 3  python:
 4      - "2.7"
 5
 6  before_install:
 7      - sudo apt-add-repository ppa:ch
ris-lea/node.js -y
 8      - sudo apt-get update
 9      - sudo apt-get install nodejs di
taa doxygen 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      - sudo pip install markdown
17      - sudo npm install -g less
18      - wget "http://downloads.sourcef
orge.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
tuml
21      - echo "#! /bin/sh" > plantuml
22      - echo 'exec java -jar /opt/plan
tuml/plantuml.jar "$@"' >> plantuml
23      - sudo install -m 755 -D plantum
l /usr/bin/plantuml
24      - wget https://bintray.com/artif
act/download/byvoid/opencc/opencc-1.0.2
.tar.gz
25      - tar xf opencc-1.0.2.tar.gz
26      - cd opencc-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 提供的虚拟机是比较标准的 Ubuntu 12.04 LTS，打上了最新的补丁，并且根据你指定的 语言选项会把相应的解释器和编译器升级到最新版（或者指定的版本）。这里用 python 语言的配置， 所以 python 是 2.7 的最新版并且有 pip 可以直接用。 配置中的 before_install 和 install 的区别其实不大，其中任何一个失败的话算作 build errored 而不是 build fail ，而如果在 script 里失败的话算作 build fail 。

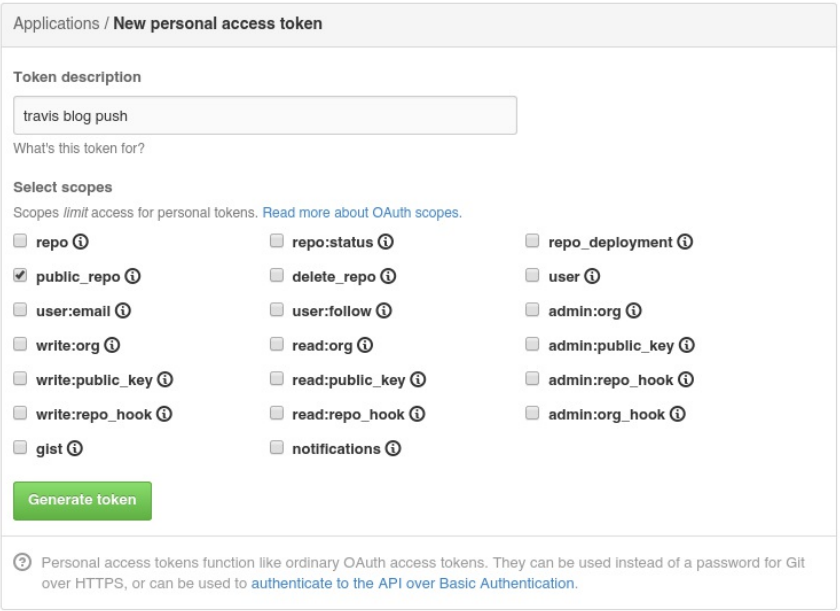
为了编译我的模板，还需要比较新的 less.js ，所以添加了 ppa 装了个最新的 nodejs 并用它装上了 less 。还从源码编译安装上了最新版的 opencc 1.0.2 ，因为 Ubuntu 源里的 opencc 的版本比较老(0.4)， 然后 doxygen 作为 opencc 的编译依赖也装上了。 其它安装的东西么，除了 pelican 之外都是插件们需要的。以及我还需要生成 4 个语言的 locale 所以调用了 4 次 locale-gen 。由于是比较标准的 Ubuntu 环境，所以基本上编译的步骤和在本地 Linux 环境中是一样的，同样的这套配置应该可以直接用于本地 Ubuntu 下编译我的博客。

写好 `.travis.yml` 之后把它 push 到 github ，然后 travis 这边就会自动 clone 下来开始编译。 travis 上能看到编译的完整过程和输出，一切正常的话编译结束之后 build 的状态就会变成 passing ，比如 我的这次的 build 。

# 从 Travis-CI 推往 Github

上面的测试编译通过了之后，下一步就是让 travis-ci 编译的结果自动推到 Github Pages 并发布出来。要推往 Github 自然需要设置 Github 用户的身份，在本地设置的时候是把 ssh key 添加到 github 账户就可以了，在编译细节都通过 github repo 公开了的 travis 上 当然不能放推送用的私有 key ，所以我们需要另外一种方案传递密码。

Github 上创建 Personal Access Token




好在 Github 支持通过 Personal Access Token 的方式验证，这个和 App Token 一样可以随时吊销，同时完全是个人创建的。另一方面 Travis-CI 支持加密一些私密数据，通过环境变量的方式传递给编译脚本，避免公开密码这样的关键数据。

首先创建一个 Personal Access Token，这里需要勾选一些给这个 Token 的权限，我只给予了最小的 public_repo 权限，如侧边里的图。 生成之后会得到一长串 Token 的散列码。

如果你不能使用 travis 命令

---

**2015年2月21日更新**

使用 `travis encrypt` 命令来加密重要数据最方便，不过如果有任何原因， 比如 ruby 版本太低或者安装不方便之类的，那么不用担心，我们直接通过 travis api 也能加密数据。

第一步用这个命令得到你的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> 替换成 github 上的用户名/repo名， 比如我的是 farseerfc/farseer 。 travis api 获得的结果是一个 json ，所以还用 python 的 json 模块处理了一下，然后把其中包含 key 的行用 `grep` 提取出来，用 `sed` 匹配出 key 的字符串本身，然后 `xargs -0 echo -en` 解释掉

转义字符，然后删掉其中的 "<空格>RSA" 几个字（否则 openssl 不能读）， 最后保存在名为 travis.pem 的文件里。

有了 pubkey 之后用 openssl 加密我们需要加密的东西并用 base64 编码：

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

替换了相应的身份信息和token之后，这行得到的结果就是 secure 里要写的加密过的内容。

然后我们需要 `travis` 命令来加密这个 token，archlinux 用户可以安装 `aur/ruby-travis` ，其它用户可以用 gems 安装：

```
1  $ gem install travis
```

装好之后，在设定了 Travis-CI 的 repo 的目录中执行一下 `travis status` ， 命令会指导你登录 Travis-CI 并验证 repo 。正常的话会显示最新的 build 状态。 然后同样在这个 repo 目录下执行：

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

当然上面一行里的相应信息替换为个人的信息，作为这个命令的执行结果会得到另一长串散列码， 把这串散列写入刚才的 `.travis.yml` 文件：

```
env:
    - secure: "long secure base64 string"
```

有了这段声明之后， Travis-CI 就会在每次编译之前，设置上面加密的环境变量。 然后在编译脚本中利用这些环境变量来生成博客：

```
 1  script:
 2      - git config --global user.email
  "$GIT_EMAIL"
 3      - git config --global user.name
"$GIT_NAME"
 4      - git config --global push.defau
lt simple
 5      - git clone --depth 1 https://gi
thub.com/farseerfc/pelican-plugins plug
ins
 6      - git clone --depth 1 https://gi
thub.com/farseerfc/pelican-bootstrap3 t
heme
 7      - git clone --depth 1 https://$G
H_TOKEN@github.com/farseerfc/farseerfc.
github.io output
 8      - env SITEURL="farseerfc.me" mak
e publish
 9
10  after_success:
11      - cd output
12      - git add -A .
13      - git commit -m "update from tra
vis"
14      - git push --quiet
```

这里要注意最后 git push 的时候一定要加上 --quiet ，因为默认不加的时候会把 代入了 $GH_TOKEN 的 URL 显示出来，从而上面的加密工作就前功尽弃了……

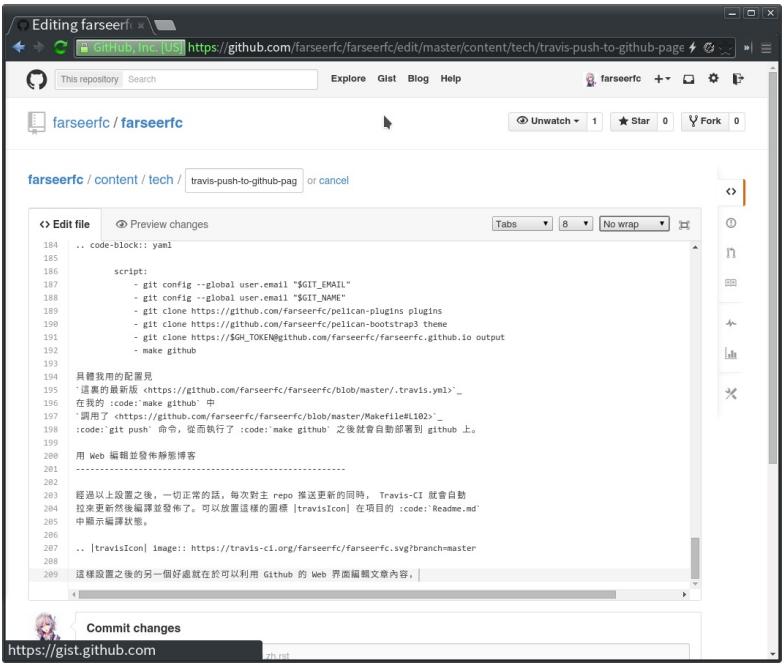
根据 travis 的文档， after_success 里写的步骤只有在 script 里的全都完全无错执行完之后才会执行，这正是我们 push 的条件。目前 after_success 的成功与否不会影响到 build 的状态。 具体我用的配置见 这里的最新版 。 在我的 `make github` 中 调用了 `git push` 命令，从而执行了 `make github` 之后就会自动部署到 github 上。

# 用 Web 编辑并发布静态博客

经过以上设置之后，一切正常的话，每次对主 repo 推送更新的同时， Travis-CI 就会自动 拉来更新然后编译并发布了。可以放置这样的图标

在项目的 `Readme.md` 中显示编译状态。

这样设置之后的另一个好处就在于可以利用 Github 的 Web 界面编辑文章内容。在 Github 里 编辑和保存之后会自动作为一个 commit 提交，所以也会触发 Travis-CI 的自动编译。




*在 Github 的 Web 界面中直接编辑文章内容*

以及虽然目前还没有好用的 Github 的手机客户端，不过直接用 Android/iPhone 的浏览器登录 github 并编辑文章的可用性也还不错，所以同样的方式也可以直接在手机上发布博文了。

That is all, happy blogging ~