用 Travis-CI 部署 Github Pages 博客 ↗

2015年2月21日星期六

原文链接： 用散弹枪打碎自己脚 是 我重大的爱好。同样装备了散弹枪，你也可以的。

之前用 Github Pages 搭建博客的时候总有一个不爽的地方，就是 Jekyll 很神奇地不需要自己构建。Github Pages 天生支持 Jekyll，好处是自己不需要构建，坏处 是你没法控制其构建过程，只能 把构造好的 HTML 手动 push 到 Github Pages 代码仓库，让 Github Pages 去构建。另一个 坏处就是不支持插件。

现在用 Pelican 来搭建博客，坏处就是要自己构建，好处就是想怎么弄都行。但是 pelican 的构建 过程相对复杂，需要预先装好很多东西，每次写完文章之后又要自己构建，然后再上传。 这一系列操作相对繁琐，而 pelican 的构建又是 python 项目，在多个电脑上保持环境一致还是

无论 Windows 还是在 Linux/OSX/Unix-like 系统上都有 很多选择 。结合我之前介绍的 wordpress 之类的 web 服务器端程序，你同样可以找到很多手机客户端。回到我们的话题，想 在 Android 上 SL4A 框架下 python 配合 pelican 生成静态博客，Android 上的 git 也 不可 或缺

Continuous integration

在介绍了这么多辅助工具之后，作为 Travis-CI 介绍的开头， 我想先 说说 持续集成 这个 概念。

什么是 Travis-CI

Agile Development   Extreme Programming

持续集成 是 敏捷开发 的一 部分， 是 极限编程 提出的概念。不过就算不采用敏捷开发 的模式，在如今软件工程中 持续集成 也有 广泛的应用。简单来说， 持续集成 关注的 是如何自动地 多次 将软件 的不同 部分 整合到一起，并且保证整合过程的正确性。每当一次 git commit 提交之后，自动地编译并执行测试。

实现持续集成的软件有很多，这篇文章将介绍 Travis-CI 与 github 的配合。 repo 中的代码改变，甚至是别人 提交的 github 合并请求，都会触发一次自动的编译并测试的过程， 从而尽早地发现问题。

配置 Travis-CI 的编译

首先需要一个账户，到 https://travis-ci.org/ 上用你的 Github 账户登录 ，它会询问你是否 授权。授权之后，到 https://travis-ci.org/repositories 就可以 看到你的 repo 列表。把想要用 Travis-CI 编译测试的 repo 开关 push 到打开状态， 之后 push 代码时就会触发编译。

farseerfc/farseerfc   ON

用 Travis-CI 的开关控制 Github Repo 是否编译

然后在 repo 的根目录里添加 .travis.yml 文件，描述编译的步骤。 下面 来看一个最简单的 .travis.yml 文件的例子：

    1   language: python

```yaml
 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 ditaa 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.sourceforge.net/project/plantuml/plantuml.jar?r=&ts=1424308684&use_mirror=jaist" -O plantuml.jar
19    - sudo mkdir -p /opt/plantuml
20    - sudo cp plantuml.jar /opt/plantuml
21    - echo "#! /bin/sh" > plantuml
22    - echo 'exec java -jar /opt/plantuml/plantuml.jar "$@"' >> plantuml
23    - sudo install -m 755 -D plantuml /usr/bin/plantuml
24    - wget https://bintray.com/artifact/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 && sudo 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://github.com/farseerfc/pelican-plugins plugins
34    - git clone --depth 1 https://github.com/farseerfc/pelican-bootstrap3 theme
```

```
35      - mkdir output
36      - env SITEURL="farseerfc.me" make publish
```

　　Travis-CI 默认提供的测试环境是 Ubuntu 12.04 LTS 的服务器，有一些预装的软件和环境 可以选择，通过配置这些环境能加速编译的过程。我选择的是预装的 python 环境，其中 默认的 python 是 2.7 ，并且预装了 pip 包管理器。 其中的 before_install 和 install 两部分 是准备编译的依赖环境，出错的话会报 build errored ，而非 build fail ；随后的 script 部分如果出错就是 build fail 。

　　因为众所周知的原因，国内的 less.js 被墙了，所以要添加 ppa 源并且安装 nodejs 版本的 less 。 随后安装了我自己编译的 opencc 1.0.2 ，由于 Ubuntu 源里的 opencc 版本还停留在老旧的(0.4)， 连 doxygen 生成 opencc 的文档也需要新版。 这之后安装了我修改过的 pelican 和它的依赖。最后，为了 生成中文的日期，还要配置 4 个中文的 locale 。我这里配了 4 个 locale-gen 是因为安装的时候 Ubuntu 并不会帮我预先生成。熟悉中文 Linux 环境的同学应该知道，我们往往需要其中 的某一个。这也算是 Ubuntu 方便的一点吧。

　　有了 .travis.yml 之后只要 push 给 github ，随后 travis 就会自动从我 clone 好的仓库 开始编译 travis 提供的是提交之后几分钟就开始的准实时的编译，编译之后就能 build 出来我想要的 passing 图标（ 见本文开头 build ）

用 Travis-CI 推送 Github

　　现在距离我们的目标只有一步之遥啦。travis-ci 编译完成之后的结果 Github Pages 可以直接 推送回 Github 。不过由于 Github 最近收紧了安全策略，为了防止 ssh key 泄漏， github 不再支持使用同一个部署用的 github repo 。好在 travis 对 这种情况也有比较优雅的 key 管理策略，那就是基于仓库的加密。

首先 Github 需要生成 Personal Access Token ，这里面需要用到一个 App Token ，当然你可以直接用账户名密码登录，但是如果 Travis-CI 配置不当就有可能泄露你的隐私信息，所以这里单独申请一个仅可以修改博客的权限较小的令牌。

所以在申请 Personal Access Token 的时候，我们仅需要勾选 Token 的权限范围是 public_repo 即可，然后填写一个相关 的说明，之后就可以生成一个 Token 并复制了。

接下来需要配置 travis 环境

2015年2月21日更新：

由于 travis encrypt 命令需要安装一大堆乱起八糟的东西，而且这些东西 由于 ruby 的版本问题会产生各种奇葩的错误，所以这里介绍直接用 travis api 来生成加密内容。

首先我们需要获取当前repo的 pubkey ：

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

其中的 <github-id/repo> 就是在 github 上的 用户名/repo，比如本文的 farseerfc/farseer 。travis api 调用返回的是一个 json 串，我用了 python 的 json 模块来解析，然后用其中的 key 的值，用 grep 取出来后，用 sed 处理，把 key 中的 转义字符变成 xargs -0 echo -en 能够识别的形式，并且补全成 "<略>RSA" 的格式，最后用 openssl 处理一下， 重新编码后放进 travis.pem 这个文件。

有了 pubkey 之后就用 openssl 加密然后编码，下面会输出一长串 base64 的内容：

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

当然如果不想泄漏自己的token就不要把这段的输出直接复制到 secure 里，而是用自己的来代替。

另一种方法是用 travis 这个程序来加密 token 。 archlinux 可以直接安装 aur/ruby-travis ，其他发行版可以用 gems 安装：

```
1   $ gem install travis
```

然后需要确保你在 Travis-CI 的 repo 已经激活，可以用 travis status 来 检查。如果还没在 Travis-CI 激活你的 repo ，或者还没有触发过一次 build 的话， 可能需要先到你的 repo 的设置页面去

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

这样会输出加密后并且编码的字符串，把这串字符串（连同引号）按照下述格式放入 你的项目根目录的 .travis.yml 文件：

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

这两种方法都是用 Travis-CI 为每个项目提供的公钥加密，所以加密后的 密文只能用在对应的项目中，不能通用。

```
 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-p
lugins plugins
 6     - git clone --depth 1 https://github.com/farseerfc/pelican-b
ootstrap3 theme
 7     - git clone --depth 1 https://$GH_TOKEN@github.com/fars
eerfc/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
```
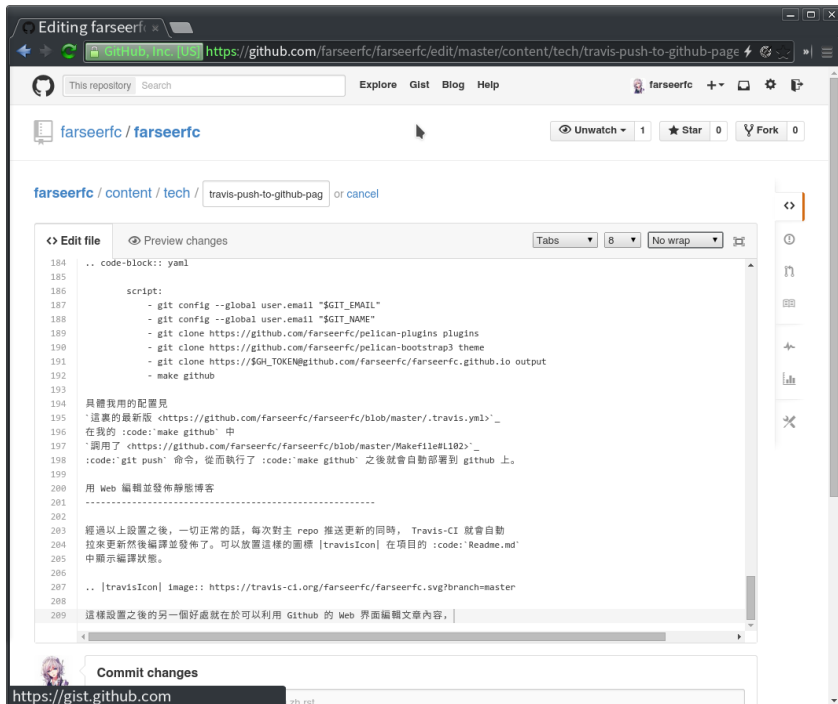
    这里要注意的是 git push 的时候要加上参数 --quiet ，否则默认会把带有 明文的
$GH_TOKEN 的 URL 输出到日志里，你的密钥就泄漏了……

    至于 travis 的脚本 中 after_success 这个部分，写到 script 里面其实也是没有问题的。
之所以分开两部分，是因为 push 这个步骤 只有 after_success 才有意义，而单独的 build 并不需要 往
仓库里面推。 顺便说一句， 我没有用 make github 而是 手写了 git push ，其实是因为 make
github 会尝试重新编译 github 页面

在 Web 界面上编辑并提交

    在上述过程搭建完毕之后，每次对 repo 的编辑都会触发 Travis-CI 的构建 ，把最新版本
的网站部署到页面上。如果 构建失败的话， build error 会 出现在 Readme.md 里面（如果没记错的话）

    由于整个流程中所有非明文的密钥都是保存在 Github 和 Web 服务上面的，因此在 Github 的 网
页界面上直接修改、提交 commit 也会触发正确的 Travis-CI 部署流程。

用 Github 的 Web 界面编辑并发布静态博客

这样设置之后，就可以利用 Github 的客户端发布博客了，比如用 Android/iPhone 上的各种 github 客户端，就可以随时随地撰写、编辑和发布博客了，而不用担心没有装博客客户端。

That is all, happy blogging ～