.Deb

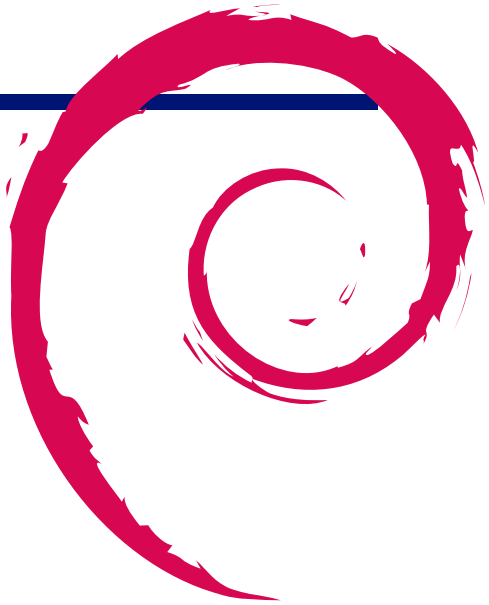
銀河系唯一のDebian専門誌

2016 年 6 月 25 日

特集 ：debexpo(mentors.d.n)をハックするには

デビアン勉強会

目次



1 Debian Trivia Quiz

杉本 典充

Debian の昨今の話題についての Quiz です。

今回の出題範囲は `debian-devel-announce@lists.debian.org` や `debian-news@lists.debian.org` に投稿された内容などからです。

問題 1. stable 版の jessie でも提供が開始された iceweasel のアップデート。パッケージ名が変わっています。何になったでしょうか

☐ A iceweasel-esr

☐ B firefox

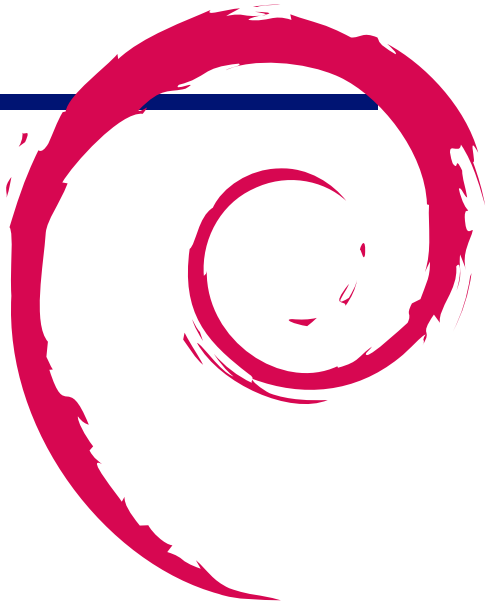
☐ C firefox-esr

問題 2. Debian Administrator 's Hanbook(Debian 管理者ハンドブック) というドキュメントがあります。ついに日本語版がリリースされました。翻訳作業に多大な貢献をしていただいた方は誰でしょうか。

☐ A Ryuunosuke Ayanokouzi

☐ B Norimitsu Sugimoto

☐ C Hideki Yamane



2 最近の Debian 関連のミーティング報告

杉本 典充

2.1 第 139 回東京エリア Debian 勉強会

2016 年 5 月 21 日 (土) に第 139 回東京エリア Debian 勉強会を開催しました。会場は銀座にある朝日ネットさんをお借りして行いました。参加者 10 名でした。

発表者は Roger さんで、表題は「Buffalo Linkstation 向け Debian InstallerDebian Installer」という内容でした。

Roger さんは Linkstation へ Debian GNU/Linux をインストールして利用できるようにハックしている方です。発表では ARM デバイス上で Debian Installer を実行する仕組みを解説していただき、実際に Linkstation へ Debian GNU/Linux をインストールするデモを実施しました。ARM デバイスへ Debian をインストールする作業は Debian Installer を利用しない場合も多いため、参加者も初めて見る方が多かったようです。また、Linkstation をハックした成果である実行コマンドのチューニングパラメータに関する質問もありました。

発表終了後は、各自の課題をこなす作業時間としました。

3 debexpo(mentors.d.n) をハックするには

林健太郎（kenhys）

3.1 はじめに

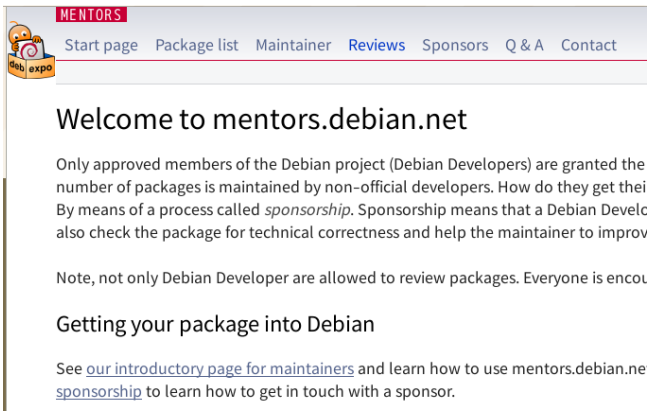
そのへんに生えている Debian 使いが Debian 公式パッケージを更新するにあたっては、更新したパッケージを Debian 開発者にスポンサーしてもらう必要があります。

その際のパッケージのアップロード先には、mentors.debian.net という Web サイトがよく使われています。

今回は mentors.debian.net をハックする機会があったので、その内容を紹介します。[*1]

3.2 debexpo とは？

http://mentors.debian.net(以下 mentors.d.n) を支えるウェブアプリケーションのことです。リポジトリ名が debexpo になっています。



debexpo の公式サイトは `https://alioth.debian.org/projects/debexpo/` です。以前開発の中心となっていたサイト `https://workaround.org/project/debexpo` には debexpo の名前の由来が次のように記載されています。

> The new project was called "debexpo" because it was supposed to become an exposition for Debian packages.

パッケージが一同に会する展覧会とでもいったところでしょうか。

[*1] 2016 年 8 月現在の情報で、現在では若干記述が古くなっている箇所があります。該当箇所には注記を入れてあります。

3.2.1 debexpo 概要

debexpo の位置付けがどういうものかなんとなくわかったところで、もうすこし具体的な紹介をしましょう。debexpo の特徴をいくつか紹介すると以下があげられます。

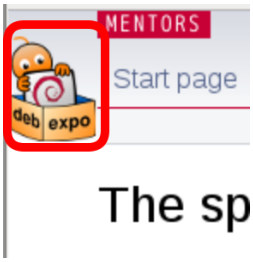
- Python 製
- Pylons[*2]フレームワーク採用
- テンプレートエンジンは Mako[*3]

この Pylons というフレームワーク、あまり知らない人がいるかも知れませんので補足しておくと、

- Railsっぽいフレームワーク
- 2011 年にメンテナンスモード入りした枯れたフレームワークです
- 後継として Pyramid というフレームワークが開発されています

という特徴があります。

コラム：debexpo にもゆるキャラが？



debexpo には、実はゆるキャラが設定されています。Web サイトの左上隅の赤枠で囲っているところに何かいますよね、それです。ただし、詳細は不明です。SUUMO[a]のスーモ[b]には片思いのスモミとの想像上の子供スモルがいるという設定[c] があるくらいなので、このキャラにも何かありそうな気もしますが。。。

[a] 不動産情報サイト。http://suumo.jp/
[b] マスコットキャラクター。スーモの部屋 http://suumo.jp/edit/suumo-heya/ という特設ページがある。
[c] http://suumo.jp/edit/suumo-heya/character/suumo_character.pdf

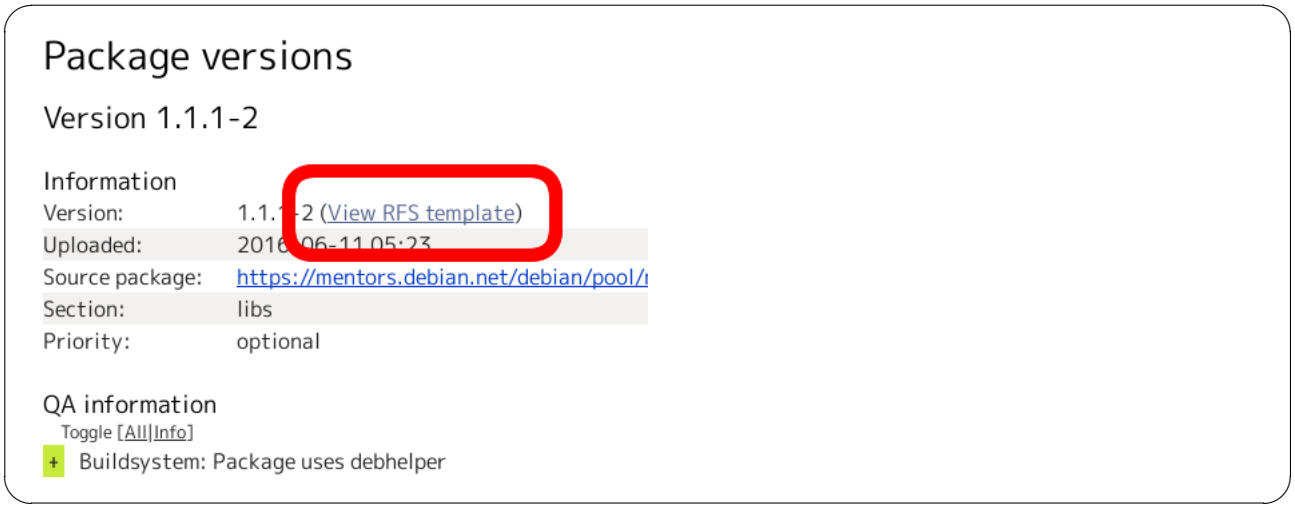
3.3 なぜ debexpo をハックする必要が？

パッケージをスポンサーしてもらうときに使う mentors.d.n にはいくつかお薦めな点があります。

- パッケージのチェックもしてくれる
- RFS のテンプレートも生成してくれる

[*2] http://www.pylonsproject.org/
[*3]

スクリーンショットからもわかるように、QA information の欄に Lintian の警告などが表示されるようになっています。



また、スクリーンショットの「View RFS template」のリンクをたどると、RFS のテンプレートも生成してくれるというのも嬉しいポイントです。

なので、あとはこの RFS テンプレートをもとにして、メールするだけになっています。[*4]

実際の RFS template を見てみましょう。

[*4] とはいうものの、嘘ではないが、正確でもない。

Template for an RFS bug for "groonga-normalizer-mysql"

Send the filled out template below by mail to our pseudo-package. If you prefer, you c

```
From: Kentaro Hayashi <hayashi@clear-code.com>
To: submit@bugs.debian.org
Subject: RFS: groonga-normalizer-mysql/1.1.1-2 [put in ITP, ITA, RC, NMU if applicable]


Package: sponsorship-requests
Severity: normal [important for RC bugs, wishlist for new packages]

Dear mentors,

I am looking for a sponsor for my package "groonga-normalizer-mysql"

* Package name    : groonga-normalizer-mysql
  Version         : 1.1.1-2
  Upstream Author : [fill in name and email of upstream]
* URL             : [fill in URL of upstreams web site]
* License         : [fill in]
  Section         : libs
```

[fill in] の文字がちらほらあるのがわかりますね。

```
It builds those binary packages:

  groonga-normalizer-mysql - MySQL derived normalizer for Groonga

To access further information about this package, please visit the following URL:

https://mentors.debian.net/package/groonga-normalizer-mysql


Alternatively, one can download the package with dget using this command:

  dget -x https://mentors.debian.net/debian/pool/main/g/groonga-normalizer-mysql/groo

More information about hello can be obtained from https://www.example.com.

Changes since the last upload:

[your most recent changelog entry]
```

これだけではありません。ほかにも穴埋めが必要です。

```
From: Kentaro Hayashi <hayashi@clear-code.com>
To: submit@bugs.debian.org
Subject: RFS: groonga-normalizer-mysql/1.1.1- [put in ITP, ITA, RC, NMU if applicable]
```

Subject:に種別を書かないといけません。

```
Package: sponsors
Severity: norma [important for RC bugs, wishlist for new packages]

Dear mentors,
```

Severity:を書かないといけません。

```
Dear mentors,

I am looking for a sponsor for my package "groonga-normalizer-mysql"

* Package name    : groonga-normalizer-mysql
  Version         :
  Upstream Author : [fill in name and email of upstream]
* URL             : [fill in URL of upstreams web site]
* License         : [fill in]
  Section         : libs
```

Upstream,URL,License:を書かないといけません。

```
  dget -x https://mentors.debian.net/debian/pool/main/g/groonga-normalizer-mysq

More information about hello can be obtained from https://www.example.com.

Changes since the last upload:

[your most recent changelog entry]
```

Changelog を書かないといけません。

これで終わりでしょうか。いいえ違います。

```
  dget -x https://mentors.debian.net/debian/pool/main/g/groonga-normalizer-mysq

More information about hello can be obtained from https://www.example.com.

Changes since the last upload:

[your most recent changelog entry]
```

さりげなく埋めこまれた hello と example.com が残っています。

ようやくあれこれ直し終えました。これでメールが出せると、思うかも知れません。事実私も最初はそう思いました。

ここで問題になるのが、先頭に埋めこまれたスペース 2 つです。

```
  Package: sponsorship-requests
  Severity: normal [important for RC bugs, wishlist for new packages]

  Dear mentors,
```

ここで、上記はコマンドメールであることを思いださねばなりません。つまり、そのままメールするともちろんエラーになります。

したがって、「なぜ debexpo をハックするのか？」に対する答えは、「RFS テンプレートの残念っぷりをどうにかしたい」から、ということになります。

Summary*:
RFS template feature requests

Detailed description

In the RFS bug template, please:

* Don't indent the pseudo-headers by two spaces. When copied and pasted to an email message, they need to be removed manually, as the control server will not be able to parse them otherwise.
* Remove the line "More information about hello can be obtained from http://www.example.com.";, or replace it by something sensible.
* Automatically fill in the Subject part that currently reads "[put in ITP, ITA, RC, NMU if applicable]" by parsing the (WNPP et. al.) bugs that the package closes.
* Same for Severity, if possible.
* Parse information from the package to fill in Upstream Author, URL and License automatically. URL should probably be filled from the Homepage field in debian/control. Upstream Author and License could be gathered from the Upstream-Name and Upstream-Contact fields in debian/copyright if the package is http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/ -- or previously DEP-5 -- compliant. (License is probably a bit more complex.)
* Fill in the "Changes since the last upload: \n\n [your most recent changelog entry]" automatically by parsing the changelog.

Finally, it would be great if a mailto: link was provided for opening a new message with the Subject and Body fields filled in automatically, i.e. something like Send RFS

Alioth をみてみると、同じ思いの人がいました。それは Alioth の tracker で 4 年も前に通った道だ、という。

3.4 どうやってハックしたのか？

おおむね、以下の流れで debexpo をハックすることになりました。

- upstream 探し
- ドキュメント探し
- まずは動かしてみる
- あたりをつけて修正
- そして PR へ

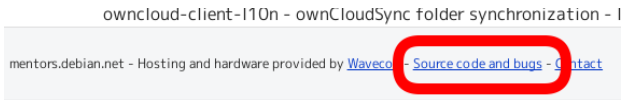
特別なことは何もなくて、よくあるフリーソフトウェアの修正です。

コラム：debexpo の歴史について

debexpo は 2003 年に最初のコードが Perl で書かれました。しかし、機能拡張の要望に応えたりしていくには支障があったため、Python で書き直されたという経緯があります。debexpo の開発が活発になったのは、2008 年のことです。Google SoC に採択されたため、Jonny Lamb 氏らにより一気に開発がすすみました。
2009 年から 2010 年はゆるやかな開発が続きました。http://expo.debian.net/が公開されたのもこのころです。現在の mentors.d.n へと切り替えられたのは 2011 年のことです。その後も、2012 年には UI の改善 (debexpo v2) や再度 GSoC への採択 (debexpo v3) などと開発が続いています。
このあたりの変遷について、Nicolas Dandrimont 氏による「The State of mentors.debian.net GSOC and Beyond」という発表資料に詳しく書かれているので興味がある人は参照するとよいでしょう。[a]

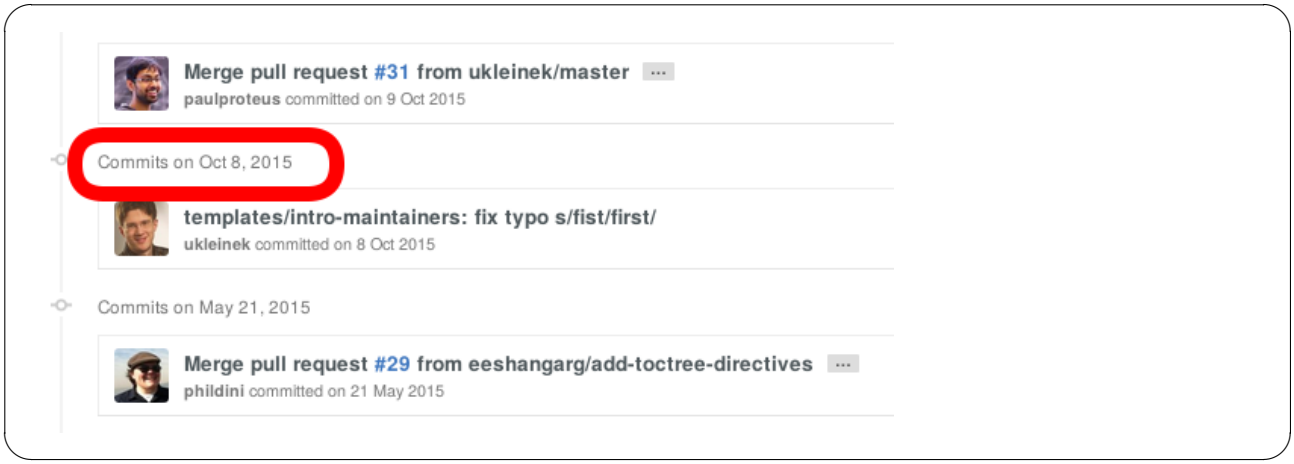
[a] http://fr2012.mini.debconf.org/slides/debexpo.pdf

3.4.1 upstream 探し



debexpo の場合には、mentors.d.n 下部にリンクがきちんとあるので、Alioth をみればいいとわかりました。

ただ、どうやら最近はコミットがないのが不安になりました。よく使われているならそこそこメンテされているイメージがあったからです。実際にはそうでもありませんでした。



あとから、GitHub のほうが実は新しい[*5]ことがわかりました。

実際の運用としては、Alioth が masterhttps://alioth.debian.org/projects/debexpo/ で、GitHub のをマージという運用になっているようです。

3.4.2 ドキュメント探し

リポジトリの docs/*にドキュメントが整備されていました。インストール手順は docs/installing.rst を参照すればよいとわかりました。ただ、残念なことにその内容の一部はリンク先が 404 になってしまっていました。

3.4.3 まずは動かしてみる

ドキュメントから、セットアップ方法は 3 種類あることがわかりました。

- 既存システムにインストール
- virtualenv でインストール
- VirtualBox でインストール

まずは VirtualBox で試してみることにしました。環境を分けたいのがその選択理由です。ただ、Vagrantfile がアレな状態であることがわかりました。

```
# Every Vagrant virtual environment requires a box to build off of.
config.vm.box = "chef/debian-7.6"
```

Debian 7.6 (2014 年 7 月 12 日)？になっていました。Debian 7.10 がもうすでににでているご時世にも関わらずです。

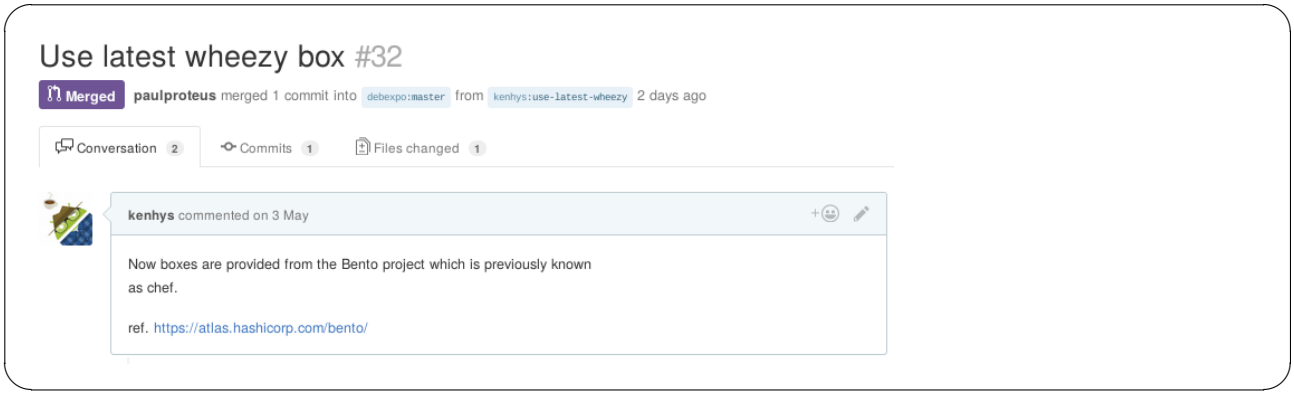
vagrant up してみるとまた残念な状態でした。

```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Box 'chef/debian-7.6' could not be found. Attempting to find and install...
  default: Box Provider: virtualbox
  default: Box Version: >= 0
The box 'chef/debian-7.6' could not be found or
could not be accessed in the remote catalog. If this is a private
box on HashiCorp's Atlas, please verify you're logged in via
`vagrant login`. Also, please double-check the name. The expanded
URL and error message are shown below:

URL: ["https://atlas.hashicorp.com/chef/debian-7.6"]
Error: The requested URL returned error: 404 Not Found
}
```

box が見つからなくてコケていました。そこで、PR#32 で修正しました。失敗していたのは、Bento project に移行していたせいでした。

[*5] https://github.com/debexpo/debexpo



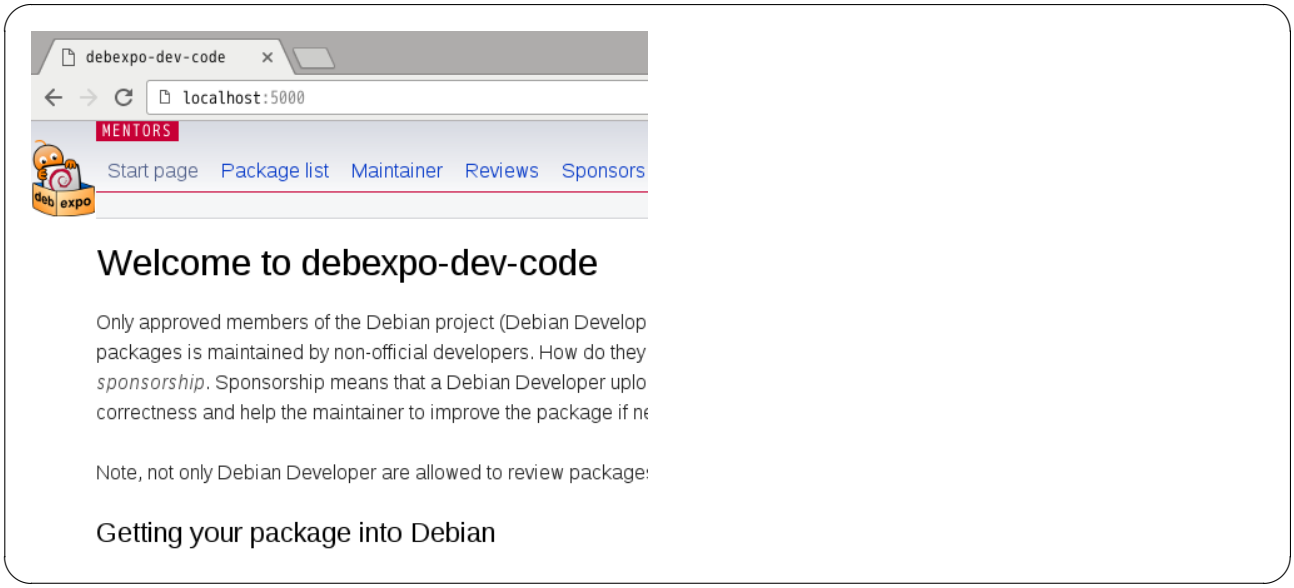
起動して、ログインするには次のようにします。

```
$ vagrant up --provision
$ vagrant ssh
```

vagrant ssh して paster コマンドを実行してサーバーを起動します。

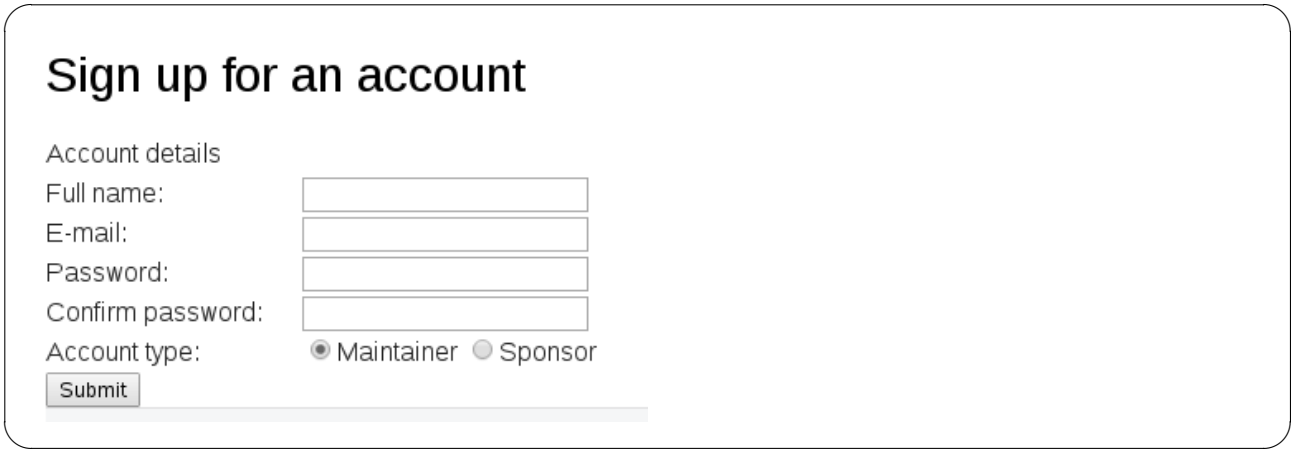
```
$ cd debexpo
$ . venv/bin/activate
$ paster serve development.ini
```

このようにすると、5000 ポートでサーバーを起動することができます。



これによりブラウザでアクセス可能になります。

次にユーザーの追加をします。方法は 2 つあって、ブラウザ経由で追加するのと、JSON をもとに追加するやりかたがあります。



JSON で追加するなら次のような内容のファイルを用意します。

```
{
  "realname":"Hayashi Kentaro",
  "password":"password",
  "email":"hayashi@clear-code.com"
}
```

ユーザー追加用のスクリプトが用意されているので、それを利用します。

```
$ python ./bin/user_importer.py -i development.ini -u user.json
```

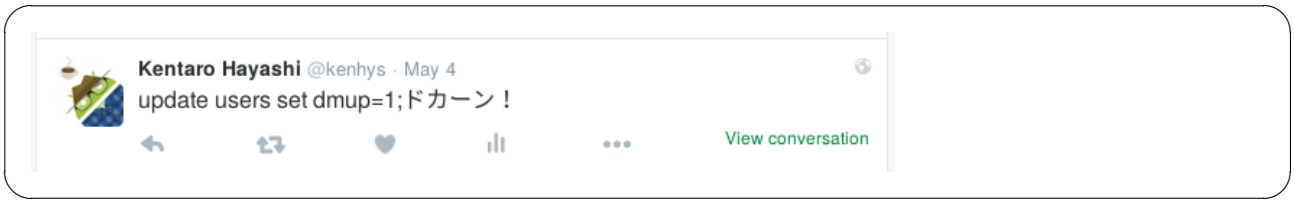
次にアカウントの有効化をします。アカウントを有効にするには 2 つ設定をする必要があります。

- verification (ログインに必要)
- dmup (アップロードに必要)

verification の設定は次のようなクエリを実行することで行います。

verification を空にすることで、メールによる確認プロセスを迂回することができます。

もう一つ、DMUP とはマシン使用ポリシーのことです。開発で使うだけなので、次のようなクエリを実行することで dmup フィールドを更新して、同意したことにします。



ここまでできたら、あとはアップロードするために、.dput.cf の設定をします。

```
[debexpo]
fqdn = localhost:5000
incoming = /upload/kenhys@gmail.com/password
method = http
allow_unsigned_uploads = 0
```

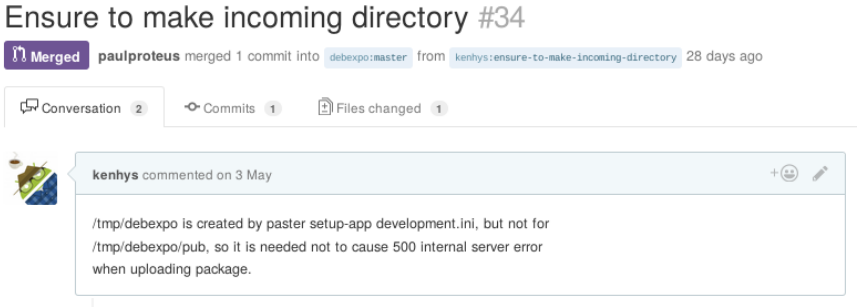
用意できたら、実際にアップロードを試してみましょう。

```
Uploading to debexpo (via http to localhost:5000):
Uploading groonga_6.0.2-1.dsc:
Upload failed: 500 Internal Server Error
```

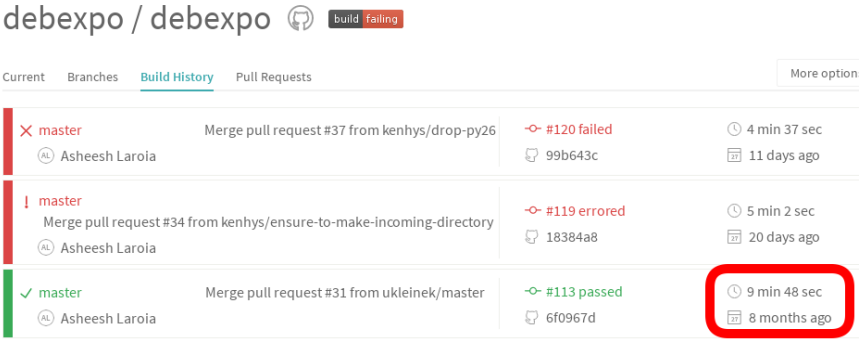
と思ったら、あっさり 500 Internal Server Error に遭遇しました。

残念なことに、あるべきディレクトリがないというオチでした。

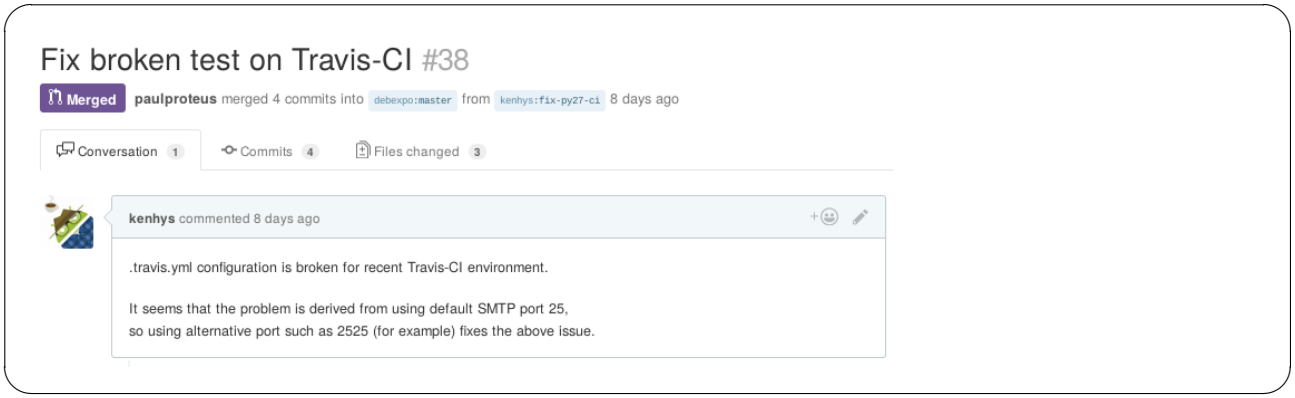
そこで、PR#34 で修正してとりこんでもらいました。



PR を出してみたときに気づいたのですが、最後にテストが通ったの 8ヶ月前というオチがついていました。



これはなぜかというと Travis-CI の環境の変化に誰も気いていない状態だったためです。しばらくコミットがなされていないプロジェクトではありがちです。そこで、PR#38 でテストが通るように修正しました。



また、Python2.6 で CI はもう不要なので、PR#37 で修正も行いました。

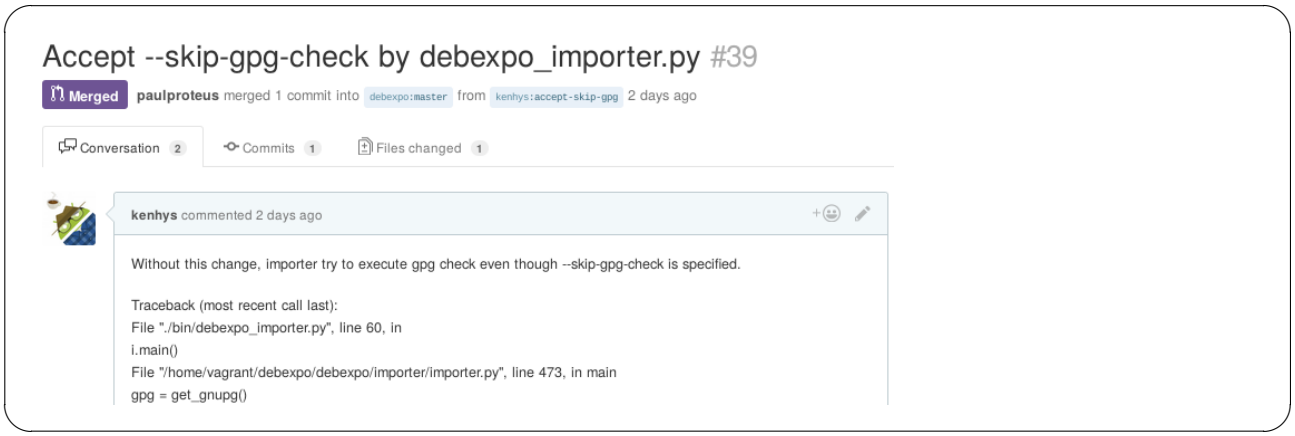
ここまで修正して、ようやくパッケージを取り込むところまでたどりつきました。パッケージの取り込みは次のコマンドを実行します。

```
$ ./bin/debexpo_importer.py \
  -c /tmp/debexpo/growl-for-linux_0.8.5-1_source.changes -i development.ini --skip-gpg-check --skip-email
```

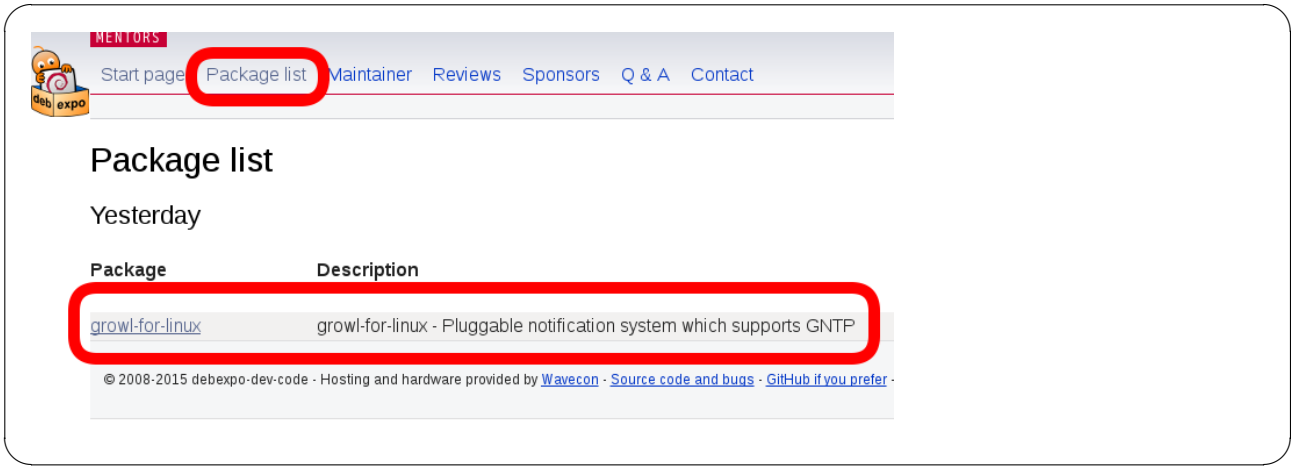
インポートスクリプトを実行したら、あっさり取り込みできずにトレースを吐きました。

```
Traceback (most recent call last):
  File "./bin/debexpo_importer.py", line 60, in
  i.main()
  File "/home/vagrant/debexpo/debexpo/importer/importer.py", line 473, in main
  gpg = get_gnupg()
  File "/home/vagrant/debexpo/debexpo/lib/utils.py", line 119, in get_gnupg
  return gnupg.GnuPG(config['debexpo.gpg_path'],
  File "/home/vagrant/debexpo/venv/local/lib/python2.7/site-packages/paste/registry.py", line 146, in getitem
  return self._current_obj()[key]
  KeyError: 'debexpo.gpg_path'
```

gpg の検証をスキップするオプションが期待するように動作していなかったので、オプションを正しく解釈するように PR#39 で修正しました。



これでようやく、取り込んだパッケージを Web の画面から確認することができるようになりました。



3.4.4 あたりをつけて修正

パッケージをアップロードして、画面から確認できるようになったので、次に本来やりたかった debexpo 自体の改善に取り組みました。まずはディレクトリ構成からあたりをつけることにしました。

```
config
controllers
cronjobs
importer
i18n
lib
model
plugins
public
templates
tests
```

手がかりとなるのは URL です。

知りたいのは View RFS Template のリンク先です。

http://localhost:5000/sponsors/rfs-howto/xxxx

リンク先がわかったので、対応するルーティングを処理するコントローラの実装を探してみたら controllers/sponsor.py を見ればいいことがわかりました。

```
def rfs_howto(self, packagename = None):
    c.package = None
    c.package_dir = None
    if packagename:
        package = meta.session.query(Package)
                    .filter_by(name=packagename).first()
        if package:
            c.package = package
            c.package_dir = get_package_dir(package.name)

    return render('/sponsor/rfs_howto.mako')
```

これに対応する Mako のテンプレートは templates/sponsor/rfs_howto.mako にあることがわかりました。なんとなく見覚えがありますね。

```
    Package: sponsorship-requests
    Severity: normal [important for RC bugs, wishlist for new packages]

    Dear mentors,

    %if c.package:
      I am looking for a sponsor for my package "${ c.package.name }"
    %else:
      I am looking for a sponsor for my package "hello":
    %endif
```

やりたいことは、RFS テンプレートから [fill in] を撲滅し、mailto:リンクを生成して RFS を出すときの手間を軽減することです。当初の目論見では、$ c.package.name とかあるので、テンプレートを書き換えればいいだけかと思っていました。しかし、結論からいうとこの案は無理でした。というのも、必要なメタ情報を保持していないことが明らかになったからです。持ってないものは表示できません。そのため、どうにかして情報をかき集めないといけないことになりました。

3.4.5 収集するにはどうすればいいか

まずは、インポートの処理の流れを把握する必要があります。そして、どのタイミングで収集すべきかを知らなければなりません。また不足している情報は何かを知る必要もあります。

では、実際のインポート処理はどのようになっているのでしょうか。

インポート処理は、dput で mentors.d.n へアップロードされた時点で始まります。そして、前処理が実行され、パッケージのインポート処理へと続いていきます。もう少し詳しく説明すると、dput したファイルは/tmp/debexpo/pub へ保存されます。そして、インポート前処理で/tmp/debexpo へ移動されます。このとき、orig.tar.gz がなかったりすると reject メールが送られます。

インポートが完了した時点で、ソースパッケージは/tmp/debexpo/files へと移動されています。このとき、/tmp/debexpo/files 以下には pool や dist,git ディレクトリが作成されます。また、各種パッケージの情報がインポート中にデータベースへと保存されるようになっています。

3.4.6 収集すべきデータを確認する

収集するべきデータを確定するには、メタ情報がどのように保持されているのかを把握する必要があります。そこで実際のテーブルを覗いてみることにしました。debexpo で使用している主なテーブルは以下の 3 つです。

- packages
- package_versions
- package_info

packages テーブルは、パッケージのマスターテーブルです。名前や説明などのメタ情報を保持しています。

```
sqlite> .schema packages
  CREATE TABLE packages (
      id INTEGER NOT NULL,
      name TEXT NOT NULL,
      user_id INTEGER,
      description TEXT,
      watch_counter INTEGER,
      download_counter INTEGER,
      needs_sponsor INTEGER NOT NULL,
      PRIMARY KEY (id),
      FOREIGN KEY(user_id) REFERENCES users (id)
  );
```

package_versions テーブルはパッケージの版管理のためのテーブルです。何度もアップロードするとレコードが増えていきます。

```
sqlite> .schema package_versions
  CREATE TABLE package_versions (
      id INTEGER NOT NULL,
      package_id INTEGER,
      version TEXT NOT NULL,
      maintainer TEXT NOT NULL,
      section TEXT NOT NULL,
      distribution TEXT NOT NULL,
      qa_status INTEGER NOT NULL,
      component TEXT NOT NULL,
      priority TEXT,
      closes TEXT,
      uploaded DATETIME NOT NULL,
      PRIMARY KEY (id),
      FOREIGN KEY(package_id) REFERENCES packages (id)
  );
```

package_info テーブルは、プラグインの適用結果を管理します。各種メタ情報を保持しています。

```
sqlite> .schema package_info
CREATE TABLE package_info (
       id INTEGER NOT NULL,
       package_version_id INTEGER,
       from_plugin VARCHAR(200) NOT NULL,
       outcome VARCHAR(200) NOT NULL,
       data TEXT,
       severity INTEGER NOT NULL,
       PRIMARY KEY (id),
       FOREIGN KEY(package_version_id) REFERENCES package_versions (id)
);
```

たとえば、from_plugin にはどのプラグインかという情報を保持しています。outcome はエラーメッセージなどの説明文を保持しています。data は汎用的に使えるように JSON データを保持しています。

実際にどんなデータが格納されているかをみてみましょう。data をうまいこと活用するとよさそうだとわかりますね。

```
sqlite> select * from package_info;
1|1|native|Package is not native|{"native": false}|1
2|1|maintaineremail|"Maintainer" email is the same as the uploader|{
  "user-email":   "hayashi@clear-code.com",
  "uploader-emails": [],
  "maintainer-email": "hayashi@clear-code.com",
  "user-is-maintainer": true
}|1
3|1|debianqa|Package is already in Debian|{
  "nmu": false,
  "in-debian": true,
  "is-debian-maintainer": true
}|1
```

ここまでの結果から、プラグインで追加のメタ情報を収集して、メール用のテンプレート追加し、詳細ページでメタ情報を表示しつつ mailto:リンク生成する方針としました。

3.4.7 プラグインの説明

ここまでの説明で特に断りなくプラグインに言及していました。補足しておくと、debexpo はプラグインで機能拡張するようになっています。パッケージのチェックもプラグインを組み合わせて実現しています。

プラグインの作り方については、docs/writing_plugins.rst にサンプルのプラグインの実装方法が紹介されています。簡単に言うと、BasePlugin クラスを継承した XXXPlugin として実装すれば OK です。

```
class FooPlugin(BasePlugin):

  def test_xxx(self):
    self.passed(outcome, data, severity)
    or
    self.failed(outcome, data, severity)
plugin = FooPlugin
```

そして、debexpo/plugins/foo.py などとして plugins ディレクトリ以下に配置することになっています。
標準で用意されているプラグインには次のようなものがあります。

```
$ wc -l debexpo/plugins/*.py
 99 debexpo/plugins/buildsystem.py
 67 debexpo/plugins/changeslist.py
141 debexpo/plugins/closedbugs.py
 85 debexpo/plugins/controlfields.py
185 debexpo/plugins/debianqa.py
 85 debexpo/plugins/diffclean.py
 63 debexpo/plugins/distribution.py
123 debexpo/plugins/getorigtarball.py
116 debexpo/plugins/lintian.py
100 debexpo/plugins/maintaineremail.py
 69 debexpo/plugins/native.py
 77 debexpo/plugins/notuploader.py
 86 debexpo/plugins/removepackage.py
 60 debexpo/plugins/ubuntuversion.py
110 debexpo/plugins/watchfile.py
```

3.4.8 プラグインの適用方法

プラグインを実際に適用するには、設定ファイル (.ini) に記述を追加します。プラグインでは次のタイミングで処理を実行することができます。

- インポート前処理
- QA 処理
- Debian 入りした時
- インポート処理後

インポート前処理で適用するプラグインは、debexpo.plugins.post_upload に設定します。getorigtarball プラグインがその例です。QA 処理で適用するプラグインは、debexpo.plugins.qa に設定します。lintian プラグインがその例です。

パッケージが Debian 入りしたときに適用するプラグインは、debexpo.plugins.post_upload_to_debian に設定します。removepackage プラグインがその例です。

インポート処理後に適用するプラグインは、debexpo.plugins.post_successful_upload に設定します。changeslist プラグインがその例です。

3.4.9 プラグインの実装

だいたいわかってきたところで、実際にプラグインを実装してみました。debexpo/plugins/rfstemplate.py として、実質 100 行ないくらいで実装できました。やっていることは、debian/changelog や debian/control から必要な情報を抽出して、package_info テーブルにメタ情報を保持し、テンプレートを表示するときにデータをバインドして表示するというものです。

あとは、設定ファイル (development.ini) に実装したプラグインを指定して有効にします。

```
debexpo.plugins.qa = ... rfstemplate ...
```

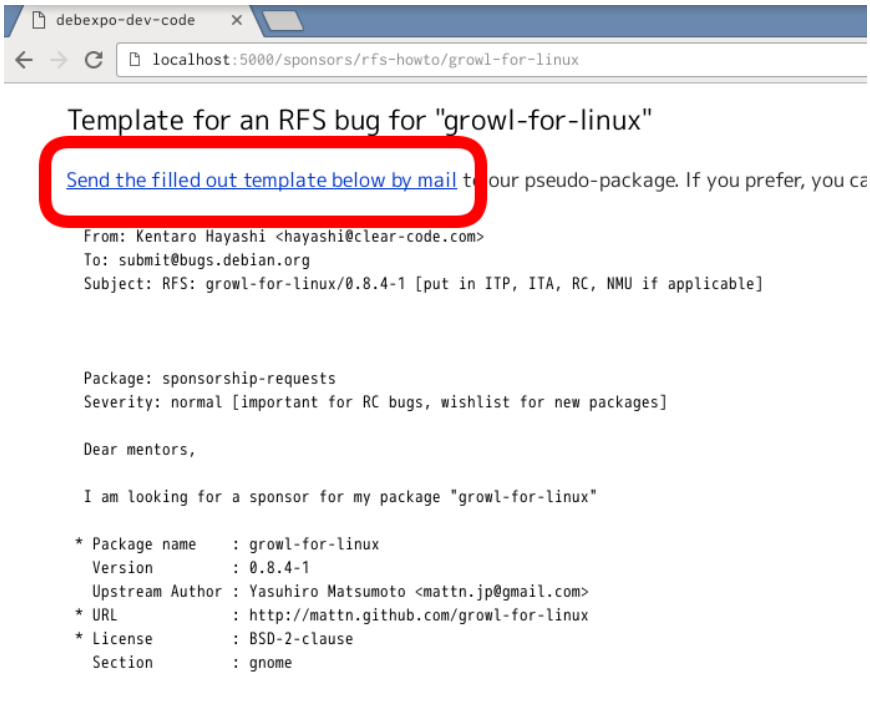
また、忘れずに mailto 用テンプレート (debexpo/templates/sponsor/rfs_template.mako) も追加しておきます。

```
%if c.rfstemplate:
  Upstream Author : ${ c.rfstemplate['upstream-author'] }
* URL             : ${ c.rfstemplate['upstream-url'] }
* License         : ${ c.rfstemplate['upstream-license'] }
%else:
  Upstream Author : [fill in name and email of upstream]
* URL             : [fill in URL of upstreams web site]
* License         : [fill in]
%endif
```

実際に rfstemplate のデータを表示させる部分は package_info テーブルから JSON データを取得してアサインするだけ (debexpo/controllers/sponsor.py) なので、簡単です。

```
if latest:
  rfstemplate = meta.session.query(PackageInfo)
                  .filter_by(package_version_id=latest.id)
                  .filter_by(from_plugin='rfstemplate').first()
  if rfstemplate:
    c.rfstemplate = json.loads(rfstemplate.data)
  c.mailbody = render('/sponsor/rfs_template.mako')
return render('/sponsor/rfs_howto.mako')
```

ここまでの成果物を PR#35[*6] としてだしました。



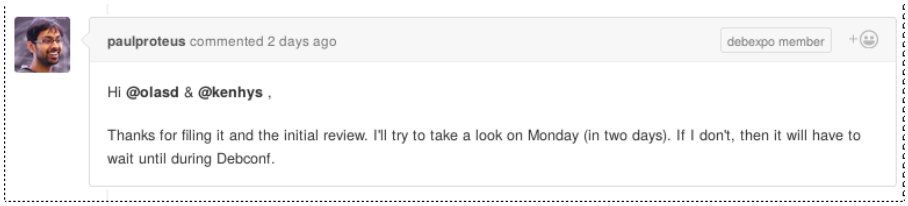
リンクをクリックすると必要事項が埋められたテンプレートを使ってメーラーが起動するようになっています。

[*6] https://github.com/debexpo/debexpo/pull/35

3.4.10 PR#35 の経過

さて、PR をだしたあと、その後どうなったかについても紹介しておきます。

- May 4 @olasd さんから好意的な反応
- May 14 どうなった？とつついてみるも反応なし
- May 21 Debian 勉強会でまだマージされてない話をする
- あれやこれやでしばし放置
- June 19 @paulproteus さんをつついてみる
- June 19 20 日にみれるかもと@paulproteus さんから反応あり



- DebConf16 までまってね、ということに
- DebConf16 終了するも、進展なし
- Aug 22 @paulproteus さんにアサインされるも放置プレイを食らう

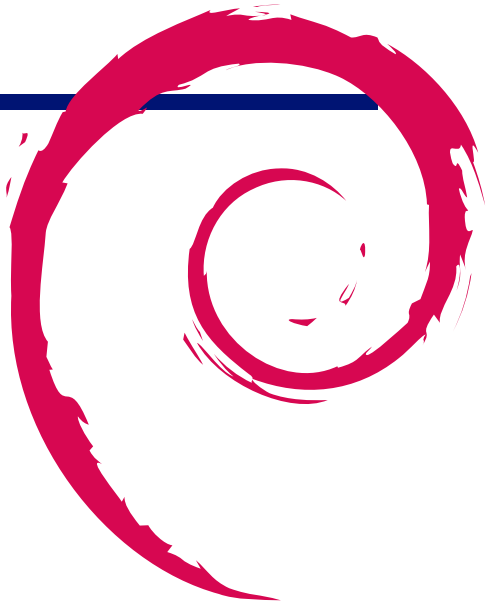
残念ながらいまだにレビューしてもらえていません。

3.5 まとめ

今回は debexpo をハックするに至った経緯と、どんなハックをしたのかを紹介しました。RFS テンプレートが残念だったのですが、プラグインを作成することで RFS テンプレートを改善することができました。ただし、まだ PR はマージされていないですし、実際にデプロイされるまでの道のりは遠そうです。

最近だと、この問題に関して、mentors.d.n ではなくクライアントツール側で改善しようという動きがあります。

debrequest というコマンドラインツールでいい感じに RFS テンプレートを生成することを目的にしているようです。実際に試してみたい人は `https://lists.debian.org/debian-mentors/2016/10/msg00206.html` に開発者によるアナウンスが投稿されているので、そちらを参照するとよいでしょう。



4 メモ

Debian 勉強会資料

2016 年 6 月 25 日　　初版第 2 刷発行

東京エリア Debian 勉強会 （編集・印刷・発行）