

# .Deb

銀河系唯一のDebian専門誌

2015年4月18日

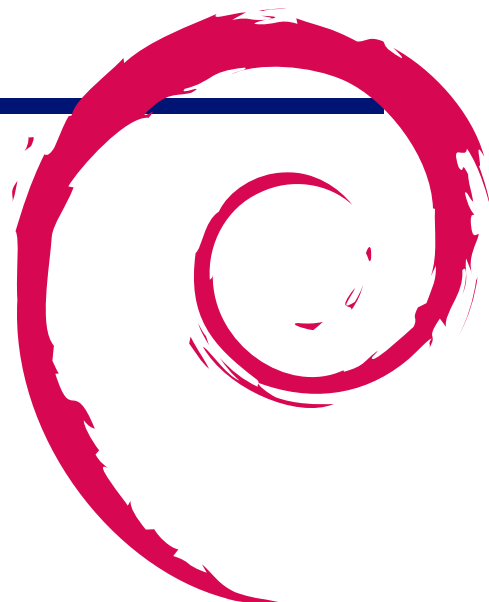
特集：.deb から Python パッケージへの変遷





## 1 事前課題

野島 貴英



今回の事前課題は以下です:

1. 本日、何の作業をやるかを宣言ください。
2. (オプション) どこで今回の勉強会の開催を知りましたか?
3. (オプション) 何について聞きたい/参加者と話をしたいですか?

この課題に対して提出いただいた内容は以下です。

を伺いたい

### 1.1 野島

1. Q.hack time に何をしますか?  
A. DDTSS などなど  
<http://ddtp.debian.net/ddtss/index.cgi/ja>
2. (オプション)Q. 何について聞きたい/参加者と話をしたいですか?  
A. "Designed for Windows 10"で無理やり導入されるかもしれない Secure boot な PC についての傾向と対策とソフトウェア自由の確保の為に戦い。

### 1.2 まえだこうへい

1. Q.hack time に何をしますか?  
A. いくつか溜まっているパッケージのメンテナンス
2. (オプション)Q. どこで今回の勉強会の開催を知りましたか?  
A. Debian JP のメーリングリスト

### 1.3 wskoka

1. Q.hack time に何をしますか?  
A. Tile-GX8036 搭載機に debian を移植しようと試みる
2. (オプション)Q. どこで今回の勉強会の開催を知りましたか?  
A. その他
3. (オプション)Q. 何について聞きたい/参加者と話をしたいですか?  
A. 新しいアーキテクチャの debian 化についてご意見

### 1.4 keikurata

1. Q.hack time に何をしますか?  
A. 初心者なので見るだけでお願いします。
2. (オプション)Q. どこで今回の勉強会の開催を知りましたか?  
A. その他

### 1.5 NOKUBI Takatsugu

1. Q.hack time に何をしますか?  
A. SimString のパッケージ化、KAKASI 関連
2. (オプション)Q. どこで今回の勉強会の開催を知りましたか?  
A. その他のメーリングリスト
3. (オプション)Q. 何について聞きたい/参加者と話をしたいですか?  
A. mecab-ipadic-neologd のパッケージ化について

### 1.6 koedoyoshida

1. Q.hack time に何をしますか?  
A. DDTSS または PyconJP 関連
2. (オプション)Q. どこで今回の勉強会の開催を知りましたか?  
A. Debian JP のメーリングリスト

A. Jessie に対する不安

## 1.7 wbcchsyn

1. Q.hack time に何をしますか？  
A. 何か Debian に寄与する事。多分、翻訳かバグフィックス
2. (オプション)Q. どこで今回の勉強会の開催を知りましたか？  
A. 友達や知り合いから直接

## 1.8 alohaug

1. Q.hack time に何をしますか？  
A. GnuK 1.1.4 が gpg4win/Win7 で認識されないバグを追いかけます。
2. (オプション)Q. どこで今回の勉強会の開催を知りましたか？  
A. Debian JP のメーリングリスト

## 1.9 henrich

1. Q.hack time に何をしますか？  
A. リリースノート訳
2. (オプション)Q. どこで今回の勉強会の開催を知りましたか？  
A. Twitter(@tokyodebian)
3. (オプション)Q. 何について聞きたい／参加者と話をしたいですか？

## 1.10 dictoss

1. Q.hack time に何をしますか？  
A. debian package における kernel の作法を調べる
2. (オプション)Q. どこで今回の勉強会の開催を知りましたか？  
A. Debian JP のメーリングリスト

## 1.11 yy\_y-ja-jp

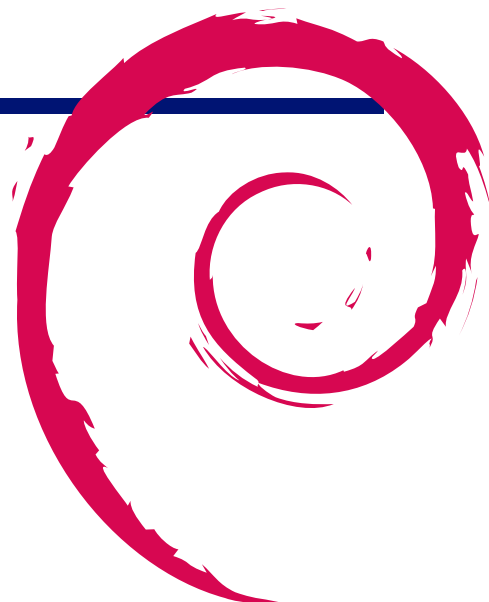
1. Q.hack time に何をしますか？  
A. DDTSS
2. (オプション)Q. どこで今回の勉強会の開催を知りましたか？  
A. その他
3. (オプション)Q. 何について聞きたい／参加者と話をしたいですか？  
A. DDTSS のレビューのお願い

## 1.12 Roger Shimizu

1. Q.hack time に何をしますか？  
A. jessie に関する検証・作業
2. (オプション)Q. どこで今回の勉強会の開催を知りましたか？  
A. Twitter(@tokyodebian)

## 2 Debian Trivia Quiz

野島 貴英



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. 2015/2/17 にて、DPL の Lucas Nussbaum により立て直しの協力募集が行われたチームは？

- A DSA team
- B partners team
- C 東京エリア Debian 勉強会 team

問題 2. 2015/2/23 に、Debian がクレジットに載せられた映画があることが `indentica` の `zak` の投稿で判りました。何と言う名前の映画？

- A ミレニアム ドラゴン・タトゥーの女
- B Toy Story
- C Citizenfour

問題 3. 2015/3/31 に Jessie のリリース目標の日がアナウンスされました。いつでしょう？

- A 4/1
- B 4/18
- C 4/25

問題 4. 2015/4/16 の lucas 最後の bit from DPL が流れました。こちらに記載されていた Debian からパッケージをリリースできるようにするための法的対策を検討中のソフトウェアは以下のどれ？

- A libdvdcss
- B OTR software
- C Unreal Engine4

問題 5. Debian にて開発者の属性の不均衡を正す事を目的として活動する公式チームが Debian Project に新設したと 4/16 に lucas によりアナウンスがありました。なんという名前のチーム？

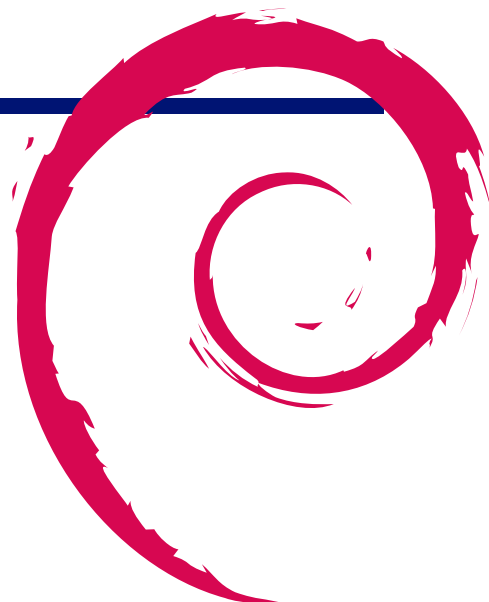
- A Debian Release team
- B Debian Outreach team
- C Technical Comittie

問題 6. 2015/4/15 に選出された新 DPL は誰？

- A Mehdi Dogguy
- B Gergely Nagy
- C Neil McGovern

## 3 最近の Debian 関連のミーティング報告

野島 貴英

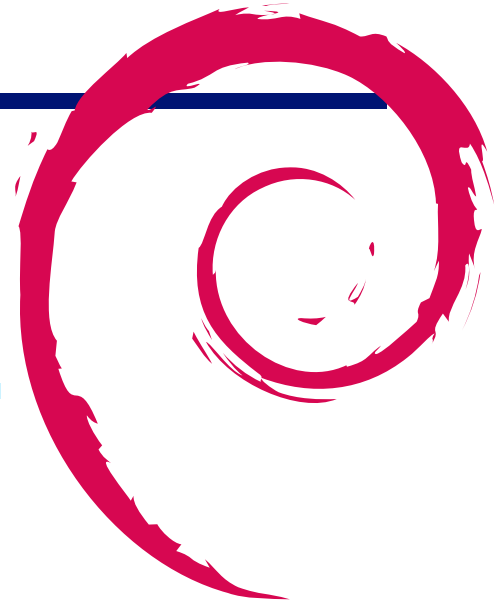


### 3.1 第 124 回東京エリア Debian 勉強会

- 場所はスクウェア・エニックスさんのセミナールームをお借りしての開催でした。
- 参加者は 9 名でした。
- セミナ内容は岩松さんによる「Raspberry Pi 2 Model B に Debian Jessie / armhf をインストールする」でした。
- 残りの時間で hack time を行い、成果発表をしました。
- 宴会の代わりに、「祥龍房 新宿イーストサイドスクエア店」で夕食会をやりました。

セミナーは Debian 公式開発者の岩松さんにより、最近発売になった Raspberry Pi 2 Model B に Debian Jessie の armhf アーキテクチャをインストールするお話でした。興味深い点として、Raspberry Pi 2 は、Debian Jessie の armhf アーキテクチャがそのまま動作し、しかも Debian をインストールして、UnixBench を取ると、Raspberry Pi (armel): Raspberry Pi 2 (armhf) = 66.5:450.8 となるなど、Raspberry Pi より何倍も性能が向上するようです。あわせて、cdebootstrap の詳しい動作・説明なども行われました。

今回は参加者にて、各種 Raspberry Pi が持ち込まれ、hack time で早速インストールなど試みる方もいらっしゃいました。Debian はいくつもの CPU にもポーティングされています。組み込みでも Debian を使う人がどんどん増えると良いと思います。



## 4 .deb から Python パッケージへの変遷

まえだこうへい

### 4.1 はじめに

一昨年 1、昨年 2、現在の職場で構築および運用している Debian パッケージの自動ビルド&ローカルアーカイブの仕組みについてお話ししました (以下ローカル Debian CI と呼びます)。昨年後半、部門での開発言語を Python \*<sup>1</sup>にするという方針になり、同様の形でローカル PyPI を利用した Python パッケージ配布の仕組みを構築しました (以下ローカル PyPI CI と呼びます)。今まで運用して分かったメリット/デメリットや、ローカル Debian CI を必要とするケースなどについて紹介します。そして私自身が主に Python 関連の Debian パッケージをメンテナンスしていますが、これに絡めた考察を行います。

### 4.2 ローカル Debian CI

ローカル Debian CI は、git-buildpackage で管理している Git リポジトリもしくはソースパッケージを取得してビルドを行います。前者はインハウスで開発したソフトウェア、後者はバックポートする場合があります。git-buildpackage / cowbuilder でのビルド、lintian でのポリシーチェック、piuparts でのインストール/アンインストールテスト、debsign\*<sup>2</sup>での署名、dput での reprepro へのアップロードを行います。少なくともバックポートに関しては Jenkins のプロジェクトのコピーだけで他のメンバーでも行えます。図 1 のような構成です。

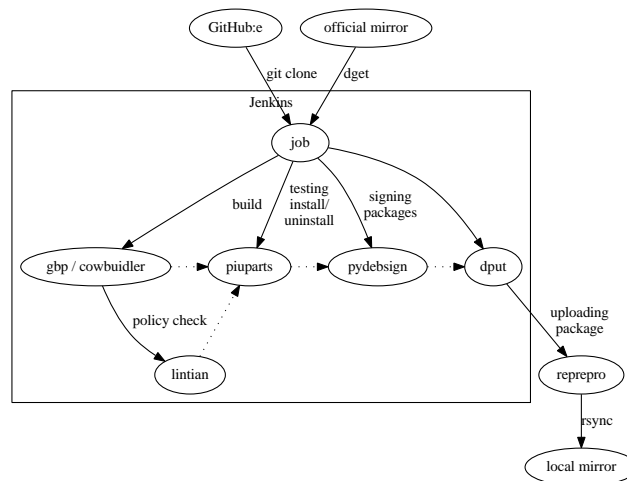


図 1 ローカル Debian CI

\*<sup>1</sup> Web フレームワークには Django 及び django REST framework に統一しました。

\*<sup>2</sup> 実際には TTY なしで debsign を実行できないため、Python で pydebsign を実装し、使用しています。3

### 4.3 ローカル PyPI CI

ローカル PyPI CI は同様に Jenkins を使用し、reprepro に相当するローカル PyPI サーバには devpi <sup>\*3</sup> を使用しています。Jenkins のジョブスクリプトもやはり Python で実装しました。このスクリプト自体も Python パッケージ化し、ジョブ実行時にローカル PyPI からインストールして実行します。<sup>\*4</sup>git-buildpackage/cowbuilder でのクリーンビルドに相当するクリーン環境でのテスト実行は Tox を使うことで実現しています。Tox から virtualenv 環境が作られ、Python2.7 / 3.4 でのユニットテスト<sup>\*5</sup>、pylint、pychecker でのチェック、Sphinx ドキュメントのビルドのテストを行います。テストが成功すると devpi-client <sup>\*6</sup> を使い、devpi-server にパッケージのアップロードを行います。

更に、ローカル PyPI CI には GitHub Enterprise からの Webhook を使ったジョブの起動と、HipChat へのテスト結果の通知の機能を実装しました。構成は図 2 のようになります。

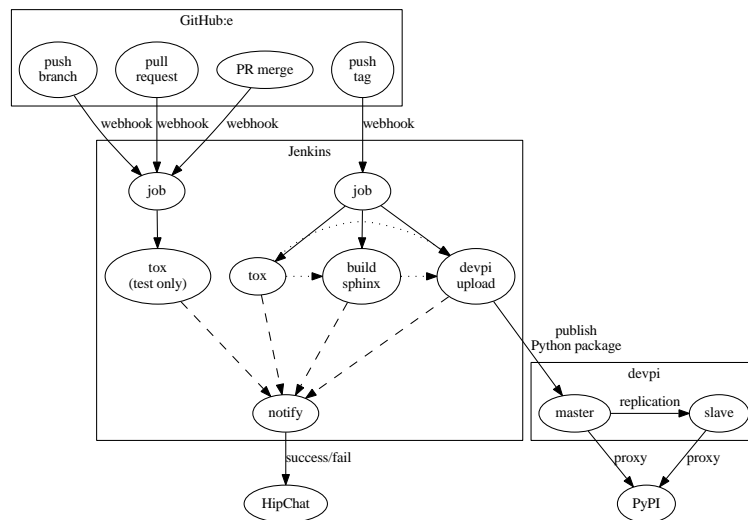


図 2 ローカル PyPI CI

#### Python での開発環境の標準化

チームメンバーのスキルレベルもまちまちであるため、品質の底上げのための次のような標準化も行いました。

- 開発標準化のための指針ドキュメント作成
- Python パッケージのテンプレート化
- Django アプリの Python パッケージのテンプレート化
- 独自認証システム用のモック機能付き共通ライブラリや Django モジュールの開発、テンプレートへの適用
- ローカル PyPI CI をローカルの開発環境で使うためのセットアップスクリプトの提供
- テンプレートおよびローカル PyPI CI の使い方のドキュメント作成、レクチャー
- HipChat での主要な変更点の案内、質疑応答

テンプレートには次のような機能を予め提供しています。

- Tox 経由での pytest、pytest-flake、pylint、pychecker の実行
- Sphinx automodule を使ったドキュメントの自動生成、ローカル PyPI へのドキュメントアップロード
- Django ベストプラクティスの適用 (Django プロジェクトのディレクトリ構成、環境ごとの settings.py の切替の仕組みなど) や、Django および django REST framework を含めたサンプルアプリの提供

<sup>\*3</sup> <http://doc.devpi.net/latest/> PyConJP 2014 の「パッケージングの今」の資料 7 で知りました。

<sup>\*4</sup> ローカル Debian CI 用のスクリプトは Git で管理し、ジョブごとに checkout する形式でした。

<sup>\*5</sup> 現在の OS は Ubuntu Trusty のため、この 2 バージョンに固定。

<sup>\*6</sup> <https://pypi.python.org/pypi/devpi-client>、[http://doc.devpi.net/latest/userman/devpi\\_packages.html](http://doc.devpi.net/latest/userman/devpi_packages.html)

## 4.4 before/after

### 4.4.1 パッケージ化を行えるメンバーの増加

ローカル Debian CI の構築以前に Debian パッケージの作成のレクチャーを行ったり、ローカル Debian CI の使い方のドキュメントの作成していました。しかし、敷居が高いこともありソースパッケージの作成は他のメンバーが行える状態ではありません。

一方、Python パッケージはテンプレートを使い、残りの `setup.py` の必要項目さえ埋めればパッケージ化できます。これにより Python は書いても Python パッケージは作ったことがないメンバーでも簡単にパッケージ化できるようになりました。

### 4.4.2 開発時におけるリポジトリの利用の簡易化

各オフィスビルからや各データセンターから<sup>\*7</sup>でも各リポジトリはアクセスできるので、基本的には Debian パッケージでも Python パッケージでもその点は差がありません。しかし、パッケージ化の観点では前者は敷居が高いため、明示的にローカルの開発環境でも使う人は限られていました。<sup>\*8</sup> 後者はパッケージ化が簡易であることや、アップロードも Jenkins からだけでなく、LDAP アカウントで手元からアップロード可能であり<sup>\*9</sup>、ローカル PyPI の URL を pip の “`--index-url`” オプションで指定することで切替可能なため、利用頻度及び明示的な利用ユーザが増加しました。

### 4.4.3 パッケージ化のコスト削減

Debian パッケージでは、基本私一人で行っており独自パッケージをつくるのに掛かる労力は結構かかります。特に依存パッケージが多く、かつそれらが公式パッケージ化されていない場合には大変です。Python パッケージの場合には、前述の通り用意したテンプレートを使えばパッケージ化でき、アップロードも簡単であるため、比べるまでもないでしょう。

ただし、Python パッケージ以外では Debian パッケージを作ることも当然あるので、この辺はパッケージを作成できるメンバーを増やす方法を検討する必要があります。<sup>\*10</sup>

## 4.5 ローカル Debian CI が必要なケース

一方で Debian パッケージのカスタムビルドが必要なケースもまだあります。

### 4.5.1 公式パッケージに無い

まず、Python パッケージ以外で、公式パッケージにはないソフトウェアを Debian パッケージ化する場合です。これが一番多いケースです。理想的にはライセンス上の問題がなければ、公式パッケージ化を目指したいところですが、そこは業務との兼ね合いもあり、コストがかかり過ぎる場合は割りきって、ローカルアーカイブで管理しています。ライセンス上の問題であったり、依存関係が多すぎる場合です。

### 4.5.2 公式パッケージにあるがアップストリームのものを使う

これは社内的な利用実績を重きに置く場合です。他の人が該当システムの主担当でパッケージ化を要望されるとき、アップストリームの配布するバイナリを `apt-get` でインストール可能にするために Debian パッケージにしています。<sup>\*11</sup> そうすることで外部のリポジトリを登録しなくても、ローカルアーカイブからインストールできるというメ

---

<sup>\*7</sup> オフィスとデータセンター間は基本 VPN 接続ですが、一部のサービスは HTTPS でアクセスできます。これもその一つです。

<sup>\*8</sup> VM のインストール用の `preseed` で設定しているため、知らずに使っている潜在的なユーザ及びサーバは非常に多いです。

<sup>\*9</sup> `devpi-ldap` (<https://pypi.python.org/pypi/devpi-ldap>) というモジュールを利用。

<sup>\*10</sup> 基本的には最初に公式パッケージ化にすることを検討しています。

<sup>\*11</sup> Tomcat とか Oracle JDK とか。

リットもあります。<sup>\*12</sup>

自分が主担当の場合には公式パッケージを使うか、Sid からのバックポートを行っています。

#### 4.5.3 公式パッケージが古すぎて要件を満たさない場合

この場合はバックポートの機会を行います。Ubuntu の LTS や Debian の stable 公式パッケージが古く、backports に存在しないが、Sid や upstream の配布するソースパッケージがある場合、それを使ってバックポートし、ローカルアーカイブで管理しています。

#### 4.5.4 Python パッケージでの特殊ケース

これは現時点ではないのですが、OpenStack の導入にあたり、Ansible で設定を行うよりも、初期のデフォルト設定を加えたカスタムビルドパッケージを使って、インストールとアップデートだけ Ansible で行った方が良いのではないか、という案です。

### 4.6 Python パッケージの公式 Debian パッケージ化について

最後に Python パッケージを公式 Debian パッケージにすること自体について考えてみます。

#### 4.6.1 ユーザーの視点

Debian パッケージとして提供されていると、`apt-get` でインストールできるのでその点が最大のメリットです。ですので、次のようなものは Debian パッケージになっていると便利です。

- コマンドラインツール
- デーモン
- 上記のパッケージに依存されているライブラリなど (直接ユーザからは見えませんが)

#### 4.6.2 Python 人の視点

Python の実行環境と標準ライブラリはシステムパッケージで良いのですが<sup>\*13</sup>、コードを書くのに使うサードパーティライブラリ自体は `pip` でインストールできる方が便利です。以前は、Sid 上で必要なパッケージの全てを Debian パッケージ化し、Debian stable か Ubuntu LTS の本番環境にもそれをバックポートしていました。が、もはや黒歴史となりました。<sup>\*14</sup>

Debian パッケージの Python を使う場合、`virtualenv/pyenv` など `venv` 環境を作るツールも Debian パッケージを使う方が、環境毎にこれらのインストールをプロジェクトのリポジトリでケアしないで済みます。Tox についても下記にある通り、Debian パッケージの `python-tox` を使う方が、依存するパッケージによって `setup.py` や `tox.ini` の記述を変更しないで済みます。

Tox は Debian パッケージのものを使う理由

Tox は `python setup.py test` に統合すれば、`setup()` の `install_requires` に指定したパッケージが `easy_install` でローカルにダウンロードされ実行されます。

しかし、Django のように `pip` しかサポートしないパッケージの場合、`setup.py test` に統合していても、まず `easy_install` でまずローカルにインストールされるのですが、Django と関連のパッケージのインストールで失敗します。そのまま再度実行すると、ローカルにインストールされたファイルがあるので、`setup.py test` からの `easy_install` はスキップされ、Tox の実行から始まります。ローカルで実行する場合には、これでも良いのですが、Jenkins などジョブ毎にクリーン環境を作る場合、必ず失敗してしまいます。`setup.py test` に統合した Tox ではなく、`tox` コマンドを直接実行する場合、デフォルトでは `easy_install` ではなく `pip` が使われ、パッケージのインストールも `toxidir` の下の `testenv` のディレクトリ毎にインストールされるのでこのような問題が発生しません。

<sup>\*12</sup> Cassandra とか。

<sup>\*13</sup> `pyenv` で実行環境そのものも自前で用意する、という人もいるかもしれませんが、私は違うので割愛。

<sup>\*14</sup> 当時、ローカル PyPI の存在を知らなかったため、その時点で出来る方法 = Debian パッケージのローカルアーカイブを使用しました。Python パッケージのファイルを HTTP サーバーに置いて公開するというのも管理が面倒です。

### 4.6.3 依存する Python パッケージに、Debian パッケージを使う場合

依存するパッケージが既に Debian パッケージ化されている場合は、`virtualenv` や `pyvenv` の `--system-site-packages` オプションを使えば、Debian パッケージでインストールされた Python パッケージを `venv` に使うこともできます。必要であれば、`venv` 環境内で個別のパッケージのアップデートも可能です。

### 4.6.4 開発した Web アプリを Debian パッケージにする場合

これは公式パッケージというよりは、カスタムビルドが主なケースの話になりますが、Web アプリ自体を Debian パッケージにする場合、`start-stop-daemon` などでもデーモン化し、uWSGI 経由で Apache や Nginx などにリバースプロキシする設定を `init` スクリプトで用意できれば、ユーザとしてはパッケージのアップデートだけで基本済むので非常に楽です。基本開発が止まっているけど、ソフトウェアそのものの需要があるなら、パッケージにすると便利です。

しかし、Web アプリとして継続的に機能追加し、リリースし続ける場合、パッケージ化するコストが高いということもありますが、一度パッケージ化してしまった後、`uscan`、`uupdate` を駆使して Debian パッケージを自動アップデート及びビルドにしたとしても、コミットからリリースまでに手順が増えるので運用に則さないと思われます。良い方法があればぜひ教えてください。<sup>\*15</sup>

## 4.7 Python パッケージを公式 Debian パッケージにすべきか

デーモンやコマンドラインツールのパッケージ化であれば、Python でないユーザにとってメリットがあるので、パッケージ&メンテナンスできるならやると良いでしょう。一方、ライブラリだけが目的なら、そもそも Python でないユーザには使われない上、Python 人も `pip` でインストールするほうが利便性が高いので基本やる必要がないと思います。一方、C binding の Python ライブラリの場合、そもそも PyPI に公開されていないこともあります。そういうケースでは Debian パッケージは必須になるでしょう。<sup>\*16</sup>

Debian パッケージの作り方自体を学ぶのであれば、Python は比較的パッケージ化がしやすいのではないかと思います。`dh-python` パッケージに含まれる、`pybuild` コマンドのおかげで、以前よりも Golang のパッケージ化並に簡単になりました。しかし、Golangの方が簡単です。これもあまりパッケージ化する意義が無い言語ですが…。

ところで以前、PyPI に登録されている Python パッケージを全部自動で Debian パッケージ化する、という話もあった気がしますが、どこ行ったのでしょうか。

## 4.8 まとめ

基本、Debian パッケージ化するのをやめたら幸せになりました。が、部分的には Debian パッケージにすることで利便性があがるので、ケースバイケースでやると良いのではないのでしょうか、というのが、私の中での結論になっています。ぜひ、他の方のお考えもお聞かせ頂ければと思います。

そういえば、Ruby でも同じ話が既にされていたと思うのですが、結局どうなったのでしょうか？

## 参考文献

- [1] “とある Web 企業での Debian システムの使い方。”, <http://gum.debian.or.jp/2013/session/437.html>
- [2] “Jenkins での Debian パッケージ自動化 “, <http://tokyodebian.alioth.debian.org/2014-07.html>
- [3] “I made debsign of Python library that can be run without a TTY“, [http://d.palmtb.net/2014/05/28/i\\_made\\_debsign\\_of\\_python\\_library\\_that\\_can\\_be\\_run\\_without\\_a\\_tty.html](http://d.palmtb.net/2014/05/28/i_made_debsign_of_python_library_that_can_be_run_without_a_tty.html)
- [4] “Retrieve and generate debian package of Oracle JDK“, [http://d.palmtb.net/2014/04/16/retrieve\\_and\\_generate\\_debian\\_package\\_of\\_oracle\\_jdk.html](http://d.palmtb.net/2014/04/16/retrieve_and_generate_debian_package_of_oracle_jdk.html)
- [5] “How to build custom Debian package automatically by Jenkins“, [http://d.palmtb.net/2014/04/12/how\\_to\\_build\\_custom\\_debian\\_package\\_automatically\\_by\\_jenkins.html](http://d.palmtb.net/2014/04/12/how_to_build_custom_debian_package_automatically_by_jenkins.html)
- [6] “Issue deploying Jenkins to Tomcat7 Debian package in Wheezy“, [http://d.palmtb.net/2014/03/27/issue\\_deploying\\_jenkins\\_to\\_tomcat7\\_debian\\_package\\_in\\_wheezy.html](http://d.palmtb.net/2014/03/27/issue_deploying_jenkins_to_tomcat7_debian_package_in_wheezy.html)
- [7] “パッケージングの今 “, <http://www.slideshare.net/aodag/ss-39068785>

<sup>\*15</sup> 以前は、ビルド済み配布物の作成に `bdist-deb` がありましたが、今は無くなったようです。

<sup>\*16</sup> 例えば、`net-snmp-python`などは Debian パッケージにしかありません。`libvirt-python`は PyPI にあります。



**Debian 勉強会資料**

2015 年 4 月 18 日 初版第 1 刷発行

東京エリア Debian 勉強会（編集・印刷・発行）

---