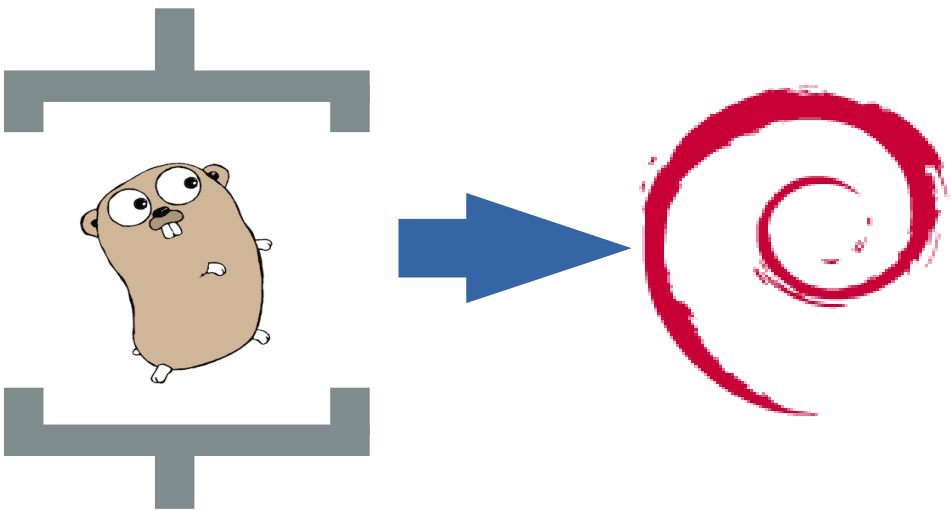
Go 言語で書かれたソフトウェアを Debian パッケージにする方法

Mini Debian Conference Japan 2016

Haruki TSURUMOTO



Who am I?

- Haruki TSURUMOTO / 弦本 春樹
- 駆け出し Debian パッケージメンテナ
- 23 歳 職探し中
- Twitter: @tSU_RooT
- 最近 peco というソフトウェアをパッケージング して、Debian 公式アーカイブに入れました!
- OSS Gate というコミュニティでメンターとして参加しています
- エンジニア多めなボードゲーム会とかに出没します
- PGP KeyID: 63A6 000E

なぜパッケージングしたか

- あるとき GitHub の Debian 管理下のリポジトリに dh-make-golang というツールがあるのを発見する
- その時いろいろなソフトウェアのビルド方法について調べていたので試しに使ってみる
- 普段使ってる Go 製ツール (peco) を対象にしてみる

なぜパッケージングしたか

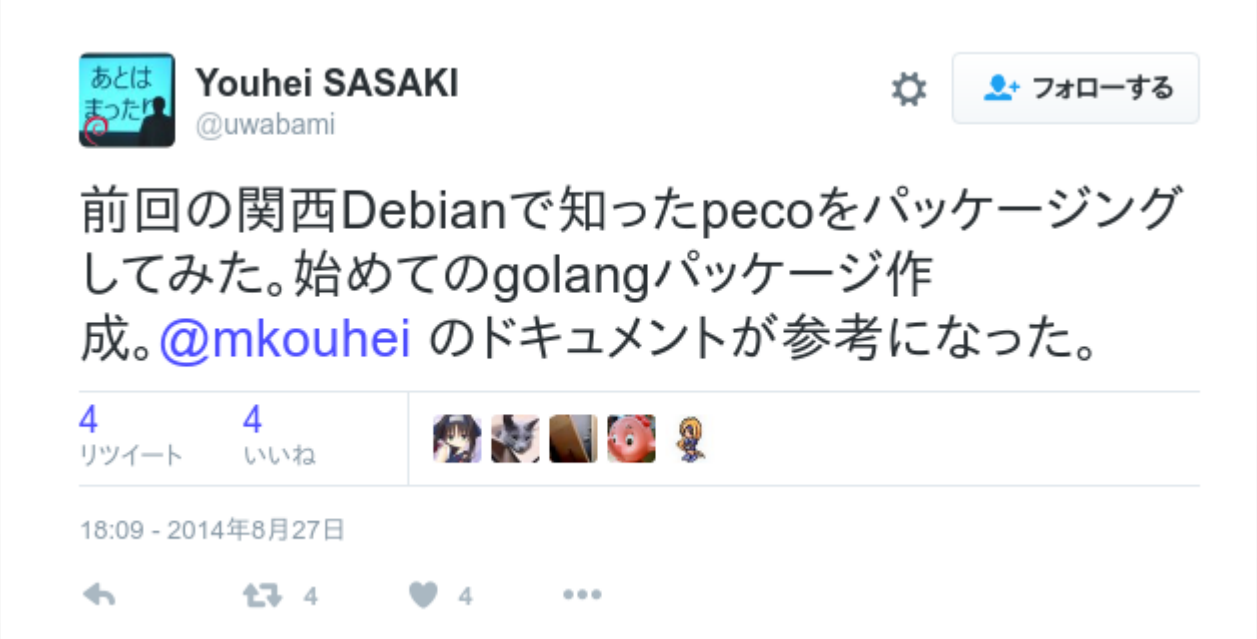
- あるとき GitHub の Debian 管理下のリポジトリに dh-make-golang というツールがあるのを発見する
- その時いろいろなソフトウェアのビルド方法について調べていたので試しに使ってみる
- 普段使ってる Go 製ツール (peco) を対象にしてみる
- あるとき、みんな使ってて Golang 開発者以外にも有用なツールならディストリビューションに入れてしまえばいいのでは?と気がつく

パッケージの需要調査

```
peco - Simplistic interactive filtering tool
```

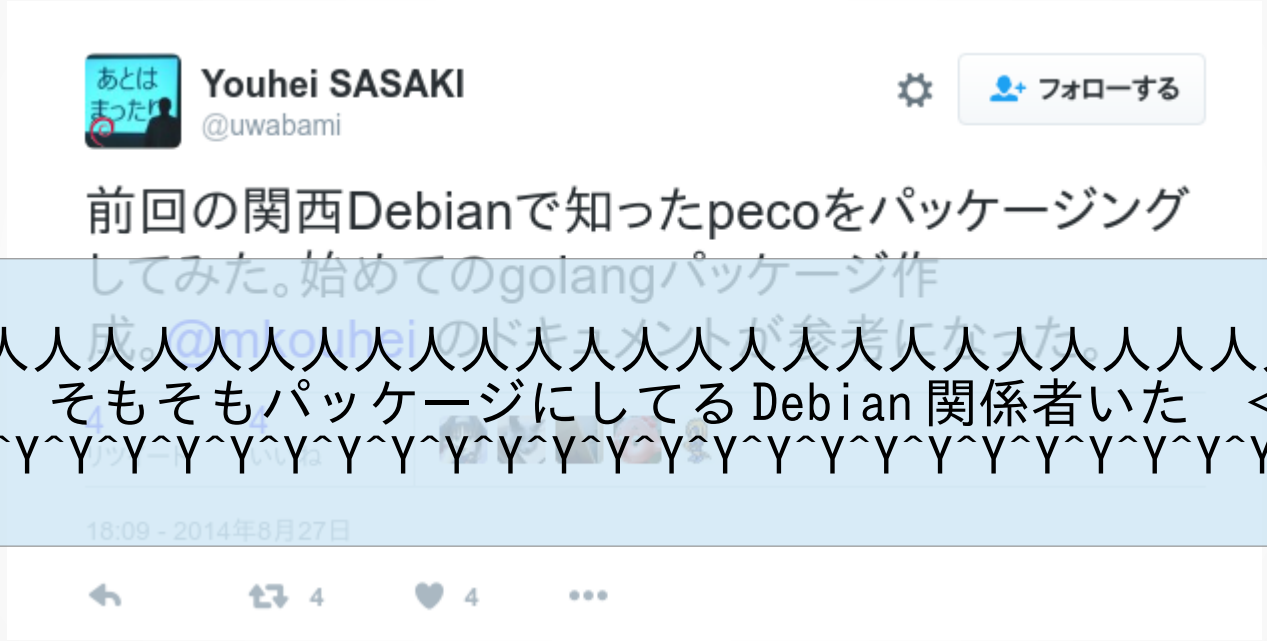
- パイプで繋いでインタラクティブにフィルタして出力するツール
- GitHub のスターは 3000 超え
- メンテナが最低 3 人活動しており、定期的なリリースも行われている
- 日本人によく使われている
- たとえば最近発売された「みんなの Go 言語」という本でも紹介されている
- Debian ユーザーの間での人気は？

パッケージの需要調査



https://twitter.com/uwabami/status/504556366988447745

パッケージの需要調査



＿人人人人人人人人人人人人人人人人人人人人人人人人人人人＿
＞　そもそもパッケージにしてるDebian関係者いた　＜
￣Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y￣

https://twitter.com/uwabami/status/504556366988447745

だがしかし

- 公式アーカイブには入っていない
- ツイートは 2014 年、作ったがアップロードされなかった？
- 考えられる理由：パッケージの品質がアップロード基準に満たなかった or　依存ライブラリが多すぎて手間だった etc…
- とにかく入ってない

たぶん需要はありそう

- 需要がある(ありそう)なら自分で入れてしまえばよいのでは？
- RFP(Request For Package) を出したが、出しただけでパッケージがアップロードされる事はあんまり無い、自分でやるのが一番良い
- 3 月に Debian 管理者ハンドブックの日本語版を読んだのだが、パッケージメンテナについての項目があり、どういう過程で開発に参加できるのかがわかった。また新メンテナガイドのようなドキュメントもあり、多くの不明点はそこから学習できた。

ビルドしたバイナリを単に配布する時のデメリット

- 手動で $PATH の通ったディレクトリに入れる必要がある
- 言語のパッケージ管理 ($GOPATH) や OS のパッケージ管理 (dpkg, rpm など) から外れてしまう
- いちいち GitHub の Release ページにアクセスしてダウンロードするのはめんどくさい (特に CLI 環境)
- 一部のアーキテクチャはバイナリが用意されていないことがある (arm とか)
- たまに著作権周りが適当なことも

公式アーカイブにいれるメリット

- ディストリビューションの公式アーカイブに入れれば、これらの欠点は解消できる
- Debian に入れれば、Debian 派生の下流のディストリビューション (Ubuntu など) でも自動的に利用できる (はず)
- 基本的にはメリットばかり

あえて挙げる公式アーカイブに入るデメリット

[Debian に限らない話]

- バージョンのアップデートがパッケージメンテナが定期的に保守するかどうかに依存する
- OS のアップデートを行わないとパッケージも古いままで固定されがちになる (ローリング・リリースを採用しているシステムを除く)

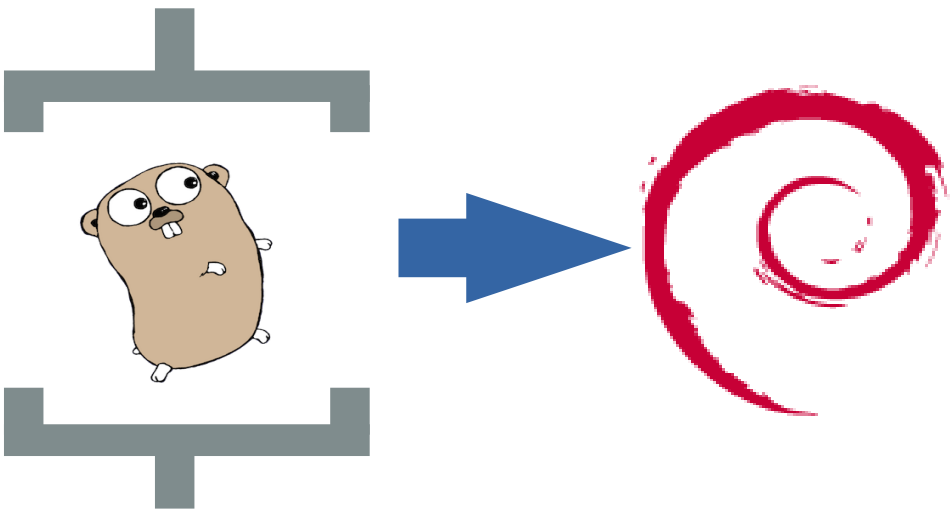
あえて挙げる公式アーカイブに入るデメリット

[Debian に限らない話]

- バージョンのアップデートがパッケージメンテナが定期的に保守するかどうかに依存する
- OS のアップデートを行わないとパッケージも古いままで固定されがちになる (ローリング・リリースを採用しているシステムを除く)

zip や tar での配布でも似た問題が起きるのであまりデメリットはない

ここからパッケージングの内容



下準備

- 可能なら Debian sid(unstable) の環境を用意

  派生ディストロ向けならそのディストロでも OK ですが、Debian の ITP には向かない

- build-essential, devscript, dh-make-golang, git をインストール

```
# apt install build-essential devscript dh-make-golang git
```

- 環境変数 DEBFULLNAME と DEBEMAIL に自分の名前を設定

  クリーン環境から始める場合、Git の config も設定してください

  user.name や user.email を設定しないと Git が正しく使えません

- 新メンテナーガイドに必要なことが丁寧に書かれています

  https://www.debian.org/doc/manuals/maint-guide/index.ja.html

リポジトリの確認

- そのソフトウェアはすべてソースコードから構築可能ですか？
  [main セクション入りの必要条件](Go の場合、あまり心配いらないかも)
- ソフトウェアにライセンスは付属していますか？[重要]
  ライセンスが無いと配布できません
  ftpmaster が通してくれません
- リポジトリの URL を go get するだけでビルドできますか？
- 依存ライブラリはすでにパッケージされていますか？
- ライブラリがパッケージされていない場合、どのくらい数がありそうですか？
- ライブラリについても以上のチェックを順々にやる必要があります

リポジトリの確認

- **ライセンスがない場合**

  ライセンスを付け忘れている例はそれなりに見るので、GitHub なら Issue を立ててお願いするとよいです。 標準ライブラリから自分用にフォークしたライブラリなどでつけ忘れを見ました。DFSG-incompatible の場合は無理かも知れません。

- **複雑なビルド手順が必要な場合**

  debian/rules をケースバイケースで加工する必要があるので難易度が高くなります、今回は扱えません。

- **パッケージされていない依存ライブラリが大量にある場合**

  この場合、ITP しないといけないパッケージが大量にあるので難しいかもしれません。RFP を出して難しそうなライブラリについては他の Debian 開発者にパッケージしてもらうのもよいかもしれません。Yak shaving はできればさけたいところです。

Go パッケージに限りありそうなパターン

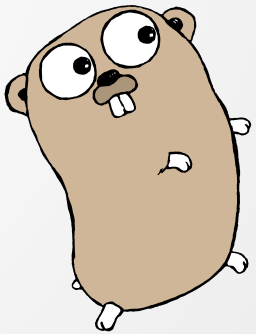
- ソフトウェアにライセンスは付属していても、いわゆるアセット (画像など) はライセンスがあいまいかもしれません。場合によっては取り除く必要があるかも

Go パッケージに限りありそうなパターン

- ソフトウェアにライセンスは付属していても、いわゆるアセット (画像など) はライセンスがあいまいかもしれません。場合によっては取り除く必要があるかも

- サンプルコードのデモに付属する Gopher くんの画像に注意

  Gopher くんのライセンスは CC BY 3.0 Unported で DFSG-free ですが、debian/copyright に特記する必要があります。



Gopher くん

Debian における Go パッケージ

- pkg-go というチームがあり、開発者はここに所属して活動
  control ファイルのメンテナのトップも pkg-go になる
  http://pkg-go.alioth.debian.org/
- ほぼすべての Go パッケージは pkg-go チームで共同管理

  例外は Docker など (Docker Packaging Team という専門のチームがある)
- チームで管理する 3 つの利点

  1. 単一障害点が無くなり、誰かがメンテできなくても他のチームメンバーが容易に引き継げる
  2. すべてのパッケージが同じ技術標準とワークフローを共有できる
  3. 技術的な問題をチームメンバーに相談できる

Debian の Go パッケージの決まり事

- バージョン管理は Git
- パッケージ作成は dh-make-golang で始める
- パッケージの名前の決め方 [重要](Naming Conventions)

  バイナリパッケージ

  プロジェクト名と同じで OK (example: peco, dh-make-golang, rclone)

  ライブラリパッケージ

  Import path を置き換えてパッケージ名にする
  最初に golang を入れる
  スラッシュをハイフンに
  トップレベルドメインを取り除く

http://pkg-go.alioth.debian.org/packaging.html

Naming Conventions

具体例

Import path	Debian package name
github.com/stapelberg/websocket	golang-github-stapelberg-websocket-dev
code.google.com/p/go.net/websocket	golang-googlecode-go.net-websocket-dev
golang.org/x/oauth2	golang-golang-x-oauth2-dev
google.golang.org/appengine	golang-google-appengine-dev

この命名方法なら Go でよくある自分用に fork したリポジトリなどでも名前が衝突せずにパッケージが可能 (次ページで解説)

細かい注意･疑問点 (1)

- フォークしたリポジトリの扱いはどうする？

  (オリジナルからいくつかパッチがあたっていたりする物)

  A. たぶんパッケージして問題ない

細かい注意・疑問点 (1)

- フォークしたリポジトリの扱いはどうする？

  (オリジナルからいくつかパッチがあたっていたりする物)

  A. たぶんパッケージして問題ない

パッケージされているケース

```
$ apt-cache search golang-.+-pflag
golang-github-ogier-pflag-dev - POSIX/GNU-style command-line flags for Go
golang-github-spf13-pflag-dev - Drop-in replacement for Go's flag package,
 implementing POSIX/GNU-style --flags
```

GitHub でも見てみると spf13-pflag は ogier-pflag のフォーク

細かい注意・疑問点 (2)

- Naming Conventions が無いころに作られたと思われるパッケージ (2013 年頃？)

  命名法則が基準から外れているので、dh-make-golang は control ファイルにちゃんと書いてくれませんが、すでにパッケージされていて、移行先のパッケージも無い場合、それを使いましょう

細かい注意・疑問点 (2)

- Naming Conventions が無いころに作られたと思われるパッケージ (2013 年頃？)

  命名法則が基準から外れているので、dh-make-golang は control ファイルにちゃんと書いてくれませんが、すでにパッケージされていて、移行先のパッケージも無い場合、それを使いましょう

適当な検索で見つかった具体例

```
golang-gocheck-dev - Richer testing framework for Go libraries ...
golang-openldap - OpenLDAP client integration for Go, using cgo
```

古くからあって広く使われているライブラリではたまに引っかかるっぽいので注意

細かい注意･疑問点 (3)

- Go 1.6 で正式導入された vendor ディレクトリの扱い

  (依存ライブラリを vendor 以下に入れて、そっちを優先的にロードする機能)

細かい注意・疑問点 (3)

- Go 1.6 で正式導入された vendor ディレクトリの扱い

  (依存ライブラリを vendor 以下に入れて、そっちを優先的にロードする機能)

  A. まだ完全には決まっていないっぽいのでメーリングリストで相談したほうがいいかも

  Go 1.6 のリリースが 2016 年 3 月

  pkg-go チームのドキュメントの最終更新が 2016 年 4 月

  メーリスでは vendoring は単に取り除けと言う人もいる、

  パッチがあたっている場合は要注意かも。

  http://lists.alioth.debian.org/pipermail/pkg-go-maintainers/Week-of-Mon-20161114/008871.html

Getting Started

- 実際にパッケージしたpeco(v0.4.2)を具体例に用います

  pecoの作者やコミッタ達に感謝

Getting Started

- 実際にパッケージしたpeco(v0.4.2)を具体例に用います

peco の作者やコミッタ達に感謝

```
$ dh-make-golang -type program -git_revision v0.4.2 github.com/peco/peco
2016/12/09 20:47:57 Downloading "github.com/peco/peco/..."
2016/12/09 20:48:23 Checking out git revision "v0.4.2"
[ 略 ]
$ ls
itp-peco.txt  peco  peco_0.4.2+git20160823.0.287f180.orig.tar.xz
```

dh-make-golang で upstream からソースをダウンロードします

対象がライブラリパッケージなら -type に library 、

バイナリパッケージなら program と入れる必要があります

Git にリリースタグが打たれている場合、-git_revision で指定してもよいです

ささやき いのり びるど ねんじろ

- 依存ライブラリが揃っていれば、この時点でビルドはできます、簡単です。(依存パッケージは dh-make-golang が自動生成)

ささやき いのり びるど ねんじろ

- 依存ライブラリが揃っていれば、この時点でビルドはできます、簡単です。(依存パッケージは dh-make-golang が自動生成)

debuild は Debian パッケージ作成用の高レベルコマンドです。
(Debian 管理者ハンドブックより)
https://debian-handbook.info/browse/ja-JP/stable/debian-packaging.html#id-1.18.4.5

ささやき いのり びるど ねんじろ

- 依存ライブラリが揃っていれば、この時点でビルドはできます、簡単です。(依存パッケージは dh-make-golang が自動生成)

debuild は Debian パッケージ作成用の高レベルコマンドです。
(Debian 管理者ハンドブックより)
https://debian-handbook.info/browse/ja-JP/stable/debian-packaging.html#id-1.18.4.5

```
# apt install golang-github-google-btree-dev \
  golang-github-mattn-go-runewidth-dev golang-github-pkg-errors-dev golang-github-stretchr-testify-dev
[ 存在がわかっているパッケージをインストール ]
$ cd peco/
$ debuild -uc -us #  パッケージや .changes に GPG で署名しない
[ 略 ]
dpkg-checkbuilddeps: error: Unmet build dependencies: golang-github-jessevdk-go-flags-dev
dpkg-buildpackage: warning: build dependencies/conflicts unsatisfied; aborting
[ 略 ]
```

ささやき いのり びるど ねんじろ

- 依存ライブラリが揃っていれば、この時点でビルドはできます、簡単です。(依存パッケージは dh-make-golang が自動生成)

debuild は Debian パッケージ作成用の高レベルコマンドです。
(Debian 管理者ハンドブックより)
https://debian-handbook.info/browse/ja-JP/stable/debian-packaging.html#id-1.18.4.5

```
# apt install golang-github-google-btree-dev \
  golang-github-mattn-go-runewidth-dev golang-github-pkg-errors-dev golang-github-stretchr-testify-dev
[ 存在がわかっているパッケージをインストール ]
$ cd peco/
$ debuild -uc -us #  パッケージや .changes に GPG で署名しない
[ 略 ]
dpkg-checkbuilddeps: error: Unmet build dependencies: golang-github-jessevdk-go-flags-dev
dpkg-buildpackage: warning: build dependencies/conflicts unsatisfied; aborting
[ 略 ]
```

→ **失敗しました (予想通り)**

golang-github-jessevdk-go-flags-dev が足りないようです

control ファイルの調整

- debian/control ファイルをいじってビルド可能にします

```
$ cd debian/
$ head -n 17 control
Source: peco
Section: devel
Priority: extra
Maintainer: Debian Go Packaging Team <pkg-go-maintainers@lists.alioth.debian.org>
Uploaders: Haruki TSURUMOTO <tsu.root@gmail.com>
Build-Depends: debhelper (>= 10),
               dh-golang,
               golang-any,
               golang-github-google-btree-dev,
               golang-github-jessevdk-go-flags-dev,
               golang-github-lestrrat-go-pdebug-dev,
               golang-github-mattn-go-runewidth-dev,
               golang-github-nsf-termbox-go-dev,
               golang-github-pkg-errors-dev,
               golang-github-stretchr-testify-dev,
               golang-golang-x-net-dev
Standards-Version: 3.9.8
```

この色のついた３つのパッケージが足りないようです
(2016 年 10 月時点)

正しいパッケージを探す

- golang-github-jessevdk-go-flags-dev はパッケージされていませんが、別の名前でされています。

正しいパッケージを探す

- golang-github-jessevdk-go-flags-dev はパッケージされていませんが、別の名前でされています。

```
$ apt-cache search go-flags
golang-github-svent-go-flags-dev - go library for parsing command line arguments
golang-go-flags-dev - Go library for parsing command line arguments
$ apt showsrc golang-go-flags-dev 2>/dev/null | grep Homepage
Homepage: https://github.com/jessevdk/go-flags
```

下のパッケージ (golang-go-flags-dev) が正しいです、フォークされたパッケージも混じってわかりにくいので、upstream を確認してパッケージを指定しましょう。(注意点で説明したケース)

- もちろん control ファイルを正しく書き換えます

無いパッケージと消えたパッケージ

- 3 つのパッケージのうち、残り 2 つは他のケースに該当します。そもそも Debian にパッケージがありません (2016 年 10 月当時)
- golang-github-lestrrat-go-pdebug-dev は peco の作者が運用しているデバッグライブラリです、なのでパッケージになっていないのは不思議ないです。これは自分でパッケージすれば問題ない https://github.com/lestrrat/go-pdebug
- golang-github-nsf-termbox-go-dev は upstream を見ると GitHub でスターが 4 桁ついており、すでにパッケージされていてもよさそう https://github.com/nsf/termbox-go

消えたパッケージはどうする？

- 実は Ubuntu 16.04 では古い命名規則のパッケージがでてくる

```
$ apt-cache search termbox
golang-termbox-dev - pure Go implementation of termbox library
```

- Debian sid, testing では今年 5 月に削除、依存するパッケージがなくなったのと長い間メンテナンスされていなかったのが理由
- これを復活させないとビルドできない

消えたパッケージはどうする？

- 実は Ubuntu 16.04 では古い命名規則のパッケージがでてくる

```
$ apt-cache search termbox
golang-termbox-dev - pure Go implementation of termbox library
```

- Debian sid, testing では今年 5 月に削除、依存するパッケージがなくなったのと長い間メンテナンスされていなかったのが理由
- これを復活させないとビルドできない

自分でパッケージを引き継ぐことに
レアケースかつ Go 特有というわけでもないと
思うので今回は手順は省略します
新パッケージでは名前を基準に合わせました

依存パッケージのパッケージング

- peco と同様に dh-make-golang を使います、これはライブラリなので type に library を指定します

```
$ dh-make-golang -type library github.com/lestrrat/go-pdebug
2016/12/09 22:34:42 Downloading "github.com/lestrrat/go-pdebug/..."
2016/12/09 22:34:47 Determining upstream version number
2016/12/09 22:34:47 Package version is "0.0~git20160817.0.2e6eaaa"
[略]
$ ls
golang-github-lestrrat-go-pdebug
golang-github-lestrrat-go-pdebug_0.0~git20160817.0.2e6eaaa.orig.tar.xz
itp-golang-github-lestrrat-go-pdebug.txt
$ cd golang-github-lestrrat-go-pdebug/
```

依存ライブラリのチェック (ふたたび)

- 依存ライブラリの依存ライブラリをチェックするため、debian/control を覗きます

```
$ head -n 9 debian/control
Source: golang-github-lestrrat-go-pdebug
Section: devel
Priority: extra
Maintainer: Debian Go Packaging Team <pkg-go-maintainers@lists.alioth.debian.org>
Uploaders: Haruki TSURUMOTO <tsu.root@gmail.com>
Build-Depends: debhelper (>= 10),
               dh-golang,
               golang-any,
               golang-github-stretchr-testify-dev
```

- 依存ライブラリは 1 個のみ、すでに他の Debian 開発者によってアップロード済み [ラッキー!]

Try to build(1)

- ということは依存ライブラリをインストールすればビルドできます

```
# apt install install golang-github-stretchr-testify-dev
$ debuild -uc -us
[略]
W: golang-github-lestrrat-go-pdebug source: syntax-error-in-dep5-copyright line 15: Cannot parse line "TODO"
W: golang-github-lestrrat-go-pdebug-dev: new-package-should-close-itp-bug
W: golang-github-lestrrat-go-pdebug-dev: extended-description-contains-empty-paragraph
```

lintian が警告を出していますが、ビルドできました!

では次はこの警告を消しましょう。(Debian にアップロードするためには消すのが望ましいです)

lintian の警告を消す (1)

- まず debian/copyright ファイルに TODO がそのまま残っています、そもそも現時点では copyright ファイルをまったく書いていません。なので書きましょう

  全部 Expat(別名 :MIT/X11) ライセンスなのでシンプルです

  ここらへんは完全にパッケージによりケースバイケースなので、物によっては込み入るかも知れません

  既存のパッケージを参考にするのがよいでしょう

  copyright ファイルの規格 :https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/

copyright 例示

```
$ cat copyright
Format: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: go-pdebug
Source: https://github.com/lestrrat/go-pdebug

Files: *
Copyright: 2016 lestrrat
License: Expat

Files: debian/*
Copyright: 2016 Haruki TSURUMOTO <tsu.root@gmail.com>
License: Expat
Comment: Debian packaging is licensed under the same terms as upstream

License: Expat
 Permission is hereby granted, free of charge, to any person obtaining a copy
 of this software and associated documentation files (the "Software"), to deal
 in the Software without restriction, including without limitation the rights
 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 copies of the Software, and to permit persons to whom the Software is
 furnished to do so, subject to the following conditions:
 .
 The above copyright notice and this permission notice shall be included in all
 copies or substantial portions of the Software.
 .
 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 SOFTWARE.
```

lintian の警告を消す (2)

ITP:Intent To Package
ITP の方法は脇にそれるので省きます

- debian/changelog を正しく書いていません

  dh-make-golang がテンプレートを生成していますが、TODO の部分に ITP の Bug 番号を埋めないと警告が出るようになっています。ITP をして、その番号を書きましょう

```
$ cat changelog
golang-github-lestrrat-go-pdebug (0.0~git20160817.0.2e6eaaa-1) UNRELEASED; urgency=medium

  * Initial release (Closes: TODO)

 -- Haruki TSURUMOTO <tsu.root@gmail.com>  Fri, 09 Dec 2016 22:34:48 +0900
```

- 警告はでていませんが、アップロードの際は、最新の changelog の UNRELEASED を unstable などのアップロード先のディストリビューションに変える必要があります

lintian の警告を消す (3)

- debian/control が何かまずいようです。

  依存関係以外修正していないので、やはり当然です。

  README から自動で生成された説明文は長すぎることがあります、

  必要な部分だけ抜き出しましょう。また 1 行の説明文も適切に抜き出して配置しましょう

```
Description: Simplistic interactive filtering tool
 peco Simplistic interactive filtering tool
 .
 peco can be a great tool to filter stuff like logs, process stats, find
 files, because unlike grep, you can type as you think and look through
 the current results.
[ 略 ]
```

例

- 空行は半角スペースを 1 個配置したあと、.(ドット) を置けば表現できます

Try to build(2)

```
$ debuild -uc -us
[ 略 ]
dpkg-buildpackage: info: full upload (original source is included)
dpkg-buildpackage: info: running hook check
 lintian ../golang-github-lestrrat-go-pdebug_0.0~git20160817.0.2e6eaaa-1_i386.changes
$
```

- 特に lintian の警告がでないのでありがちな間違いは起こしていないことがわかります
- ライブラリのパッケージができました!
- 次のステップに進みましょう

```
$ debuild clean
$ git add debian/ & git commit #  変更をgitに記録
$ cd ../
# dpkg -i golang-github-lestrrat-go-pdebug-dev_0.0~git20160817.0.2e6eaaa-1_all.deb
[ ライブラリをインストール ]
```

Yak Shaving?

- 以上のことを必要なライブラリパッケージすべてに行って、システムにインストールします

peco の場合は termbox-go を最新にして control ファイルをいじり、
再びパッケージングする必要がありました (メンテナンスされていなかったので)

- ライブラリが全部パッケージングできたらバイナリパッケージにも同じことをします、バイナリパッケージでは man ページも書く必要があるので自分で書くか help2man などを使用して生成します (upstream で用意されている場合はそれを指定すればよいです)

注意

- 例に挙げたパッケージは同じ詰まり方はしません。なぜなら peco のアップロードの時に依存ライブラリをパッケージングしアップロードしてしまったからです
- よって別のソフトウェアをパッケージングするときの参考資料として使うほうがよいかもしれません

今回話せなかったこと (1)

- pbuilder を使ったクリーンビルドでの確認
- ITP の手順
- スポンサー・メンター探しについて

今回話せなかったこと (2)

- アーカイブへのアップロードの方法
- パッケージのその後の保守方法 [模索中]

  (Git で upstream のブランチを平行管理したりマージする方法、pristine-tar などについて)

  物によっては決まりきった手順は無いことや、公式のドキュメントの方が正確で詳しそうなこともあること、スライドを作っていたら予想より他の部分が多くなったので省きました。

  おそらく debian 本体のメーリスや debian-jp のメーリス・勉強会で聞くほうが良いかもしれません。私は debian-jp のメーリスでスポンサーを探したら、見つかりましたが常にそうとは限りません。

参考になる資料

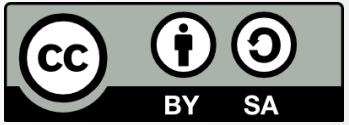
- Debian 新メンテナーガイド
  https://www.debian.org/doc/manuals/maint-guide/index.ja.html
- Debian 管理者ハンドブック
  https://debian-handbook.info/browse/ja-JP/stable/
- Debian 開発者向けマニュアル
  https://www.debian.org/doc/devel-manuals.ja.html
- 実際のパッケージ
  clone して debian/ 以下のファイルをどんどん覗く

付録 [著作権情報]

- 後でスライドをネットに上げます
- Copyright 2016 Haruki TSURUMOTO

  This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

  https://creativecommons.org/licenses/by-sa/3.0/



- The Go gopher was designed by Renee French. (http://reneefrench.blogspot.com/)

  The design is licensed under the Creative Commons 3.0 Attributions license.
  https://creativecommons.org/licenses/by/3.0/

  Read this article for more details: https://blog.golang.org/gopher

- The Debian Open Use Logo(s) are Copyright (c) 1999 Software in the Public Interest, Inc., and are released under the terms of the GNU Lesser General Public License, version 3 or any later version, or, at your option, of the Creative Commons Attribution-ShareAlike 3.0 Unported License. https://creativecommons.org/licenses/by-sa/3.0/

付録 [変更履歴]

Ver 1.0 2016/12/10 Mini Debian Conf Japan 発表版

その他変更無し