.Deb

銀河系唯一のDebian専門誌

2014 年 04 月 19 日

特集：Golangなツールのパッケージ化

(Serf編)

デビアン勉強会

目次

1 事前課題 2
- 1.1 野島貴英 2
- 1.2 岩松 信洋 2
- 1.3 dictoss(杉本　典充) 2
- 1.4 Yoshida Shin 2
- 1.5 Koji Hasebe 2
- 1.6 henrich 2
- 1.7 zinrai 2
- 1.8 yyuu 2
- 1.9 野首 (@knok) 2
- 1.10 まえだこうへい 2

2 Debian Trivia Quiz 3

3 最近の Debian 関連のミーティング報告 4
- 3.1 東京エリア Debian 勉強会 111 回目報告 4

4 Golang で書かれたツールを Debian パッケージにする 5
- 4.1 はじめに 5
- 4.2 Debian での Golang の環境構築 5
- 4.3 Golang のコンパイル方法 . . 6
- 4.4 Golang で書かれたソフトウェアの Debian パッケージ作成方法 7
- 4.5 Serf の依存するパッケージ . 12

5 会場での無線 LAN のつなぎ方 14
- 5.1 はじめに 14
- 5.2 wpasupplicant 及び /etc/network/interfaces を利用の場合 14
- 5.3 その他の無線 LAN 用パッケージを利用の場合 14

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。

この課題に対して提出いただいた内容は以下です。

1.1 野島貴英

- Xmris パッケージ化頑張る
- iOS7.1 debian 接続再び。

1.2 岩松 信洋

?

1.3 dictoss(杉本　典充)

Debian 上で Leap Motion を動かしてみる

1.4 Yoshida Shin

GitHub に公開しているプログラムの機能追加。動作環境は Debian Sid で。

1.5 Koji Hasebe

Debian パッケージ作成の方法を調査し、実際に作成する。アウトプットとして、その方法を手順書にまとめる。

1.6 henrich

- OSC 北海道用の資料作成
- debhelper の査読の続き
- パッケージまわりでのなにか

のいずれか。

1.7 zinrai

Debian パッケージ作成。

- Debian パッケージ作成してみたい
- 「Debian パッケージング道場」に都合がつかず参加できなかったので「もくもく会」の時間を使いパッケージ作成の作法を心得ている人がいる環境でパッケージ作成したい
- redis 2.8.8 を Debian パッケージ作成の題材にしてみようと思う。`http://redis.io/`
- 通勤時間などを使い当日までにこの辺に目を通しておきたい `http://www.debian.or.jp/community/devel/`

1.8 yyuu

自分で管理してるプロジェクトの deb 化を要望されてしまったので、そのための作業を行ないたいと考えています。
`https://github.com/yyuu/pyenv/issues/138`

1.9 野首 (@knok)

- KAKASI ローマ字テーブルの改善
- LanguageTool のルール追加
- gnu.org, ve の翻訳

1.10 まえだこうへい

- Golang 関係のパッケージ化の話をする
- `http://qa.debian.org/developer.php?login=mkouhei@palmtb.net` のバグ潰し & パッケージアップデート
- 上記の Golang 関係のパッケージを pkg-go team でのメンテナンスを始める（ML に投稿& ITP する）

2 Debian Trivia Quiz

野島 貴英

ところで、みなさん Debian 関連の話題においついていますか？Debian 関連の話題はメーリングリストをよんでいると追跡できます。ただよんでいるだけでははりあいがないので、理解度のテストをします。特に一人だけでは意味がわからないところもあるかも知れません。みんなで一緒に読んでみましょう。

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

問題 1. 2015 年の Debconf15 の開催国はどこになったでしょう？
☐ A ボスニア・ヘルツェゴビナ
☐ B ウクライナ
☐ C ドイツ

問題 2. DPL 選挙が行われました。今年の DPL は誰になったでしょうか？
☐ A Lucas Nussbaum
☐ B Neil McGovern
☐ C Raphaël Hertzog

問題 3. clang3.4 による Debian パッケージの再構築が行われました。結果何 % のパッケージが成功したでしょうか？
☐ A 90%
☐ B 50%
☐ C 10%

問題 4. beagle board シリーズという非常に人気のある ARM の実験ボードにバンドルされる OS の将来の見通しについて、beagle board の創立者がどの OS にする予定と発言したか？
☐ A Gentoo
☐ B Debianっしょ！Debian
☐ C Andoroid OS

問題 5. 激論の末、3 月中旬頃に Debian の ca-certificates パッケージから消えた root 証明書があります。それはなんでしょう？
☐ A RapidSSL の root 証明書
☐ B CAcert の root 証明書
☐ C Verisign の root 証明書

問題 6. Jessie にてデスクトップ環境を選択した際に導入される、デフォルトコミュニケーションツールの候補について議論がされていります。以下のどれでしょう？
☐ A Empathy
☐ B licq
☐ C jitsi

問題 7. 先日 Debian に OTR チームという OTR ソフトをパッケージ化するグループが結成されました。ところで OTRってなんの略？
☐ A Owa-Tte-Ru
☐ B OpticalTRacking
☐ C Off-the-Record

問題 8. 先日、Debian squeeze のサポート期間が伸びる宣言が DSA によりアナウンスされました。結局いつになった？
☐ A 2015/2
☐ B 2016/2
☐ C 2016/5

3 最近の Debian 関連のミーティング報告

野島 貴英

3.1 東京エリア Debian 勉強会 111 回目報告

東京エリア Debian 勉強会 111 回目は (株) スクウェア・エニックスさんで開催されました。5 名の参加者がありました。

- debian から iphone5(iOS7.1) につなぐ件について、
  - debian パッケージを作ってつなぎ、
  - 公開情報に基づいた接続技術

  について発表がありました。
- 参加者全員で、各自の作業を行い、最後に成果発表をしました。

4 Golang で書かれたツールを Debian パッケージにする

前田 耕平

4.1 はじめに

あることをやるのに Serf[*1]というツールが良いのでは、と同僚に助言をもらったので、Serf でやりたいことができないか検証をすることにしました。ですが、そもそも Debian パッケージには無いのでまずそこからでしょ、ということで手元で Debian パッケージにしました。[*2]

Serf を Debian パッケージにするにあたり、行ったことや、Golang のツールを Debian パッケージにする上で必要な知識、手順をまとめました。今回の勉強会では、Debian パッケージを初めて作成する人の割合が比較的多かったので、内容的に少し冗長になっています。

4.2 Debian での Golang の環境構築

4.2.1 前提条件

使用するディストリビューションは、Debian GNU/Linux Sid を前提とします。手元に Sid の環境がない場合は、仮想マシンなどで用意してください。

4.2.2 Debian パッケージのインストール

Debian では、“golang” というメタパッケージが用意されています。これをインストールすると次のパッケージがインストールされます。[*4]

- golang-doc
  `http://golang.org` のドキュメント。`godoc --http=:6060` を実行し、`http://localhost:6060/` で閲覧可
- golang-go
  Golang のアセンブラ、コンパイラー、リンカーなどのツールチェイン
- golang-go-linux-amd64
  Linux システム、amd64 アーキテクチャ向けの標準ライブラリ。system: darwin,freebsd,linux,netbsd,windows、arch: amd64,386,arm
- golang-go.tools
  Golang 用の補助ツール。前述の godoc コマンドも含まれる

[*1] `http://www.serfdom.io/`
[*2] Serf 自体のお話を聞きたいと思った方は残念。Upstream[*3]のドキュメントや日本語の Serf を使ったオーケストレーションの資料を見ただけで、Serf 自体の検証はまだ行っていないので、Serf そのものについては何も語れません。
[*4] Linux Kernel の amd64 アーキテクチャの場合

- golang-src
  golang パッケージのソースコード。godoc コマンドなどで使われる
- libjs-jquery
  jQuery。golang-go.tools に依存。godoc コマンドで実行する golang-doc のドキュメント用
- javascript-common
  JavaScript ライブラリパッケージのサポートパッケージ。libjs-jquery に依存

4.3 Golang のコンパイル方法

標準ライブラリのみで書かれたコードをコンパイルする場合には、`go build` コマンドのみでできます。例えば、おなじみに Hello world のコード (hello.go) を用意します。

```
package main

import "fmt"

func main() {
  fmt.Println("Hello world.")
}
```

コンパイルします。

```
$ go build hello.go
```

バイナリの実行は`./hello` です。

```
$ ./hello
Hello world.
```

`file` コマンドで見ると、静的リンクされていることが分かります。

```
$ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, not stripped
```

なお、`go run` コマンドを使うと、コンパイルしなくても実行可能です。

```
$ rm hello
$ go run hello.go
Hello world.
$ ls hello
ls: cannot access hello: No such file or directory
```

4.3.1 標準ライブラリ以外に依存する Golang パッケージのコンパイル

まず、依存するパッケージの配置場所を環境変数 `GOPATH` で指定する必要があります。

システムグローバルの設定は、Debian では、`/usr/share/gocode` になります。既に Debian パッケージになっている Golang のライブラリは、このディレクトリの下にある、src ディレクトリ以下にインストールされます。

システムグローバルで設定されているパッケージを使う場合には、

```
$ export GOPATH=/usr/share/gocode
```

GOPATH は絶対パスで指定する必要があります。コンパイルするコードと同じカレントディレクトリに存在するなら、

```
$ export GOPATH=$(pwd)
```

と実行した上で、実際のパッケージの配置は、`$(pwd)/src/example.org/mkouhei/hoge/`のようなディレクトリ構成になっている必要があります。先ほどの hello.go にこのパッケージを import する場合、

```
import (
    "fmt"
    "example.org/mkouhei/hoge"
)
```

のようになります。

なお、標準ライブラリは、Debian では、/usr/lib/go 以下にインストールされています。標準ライブラリライブラリは、GOPATH で指定する必要はなく、GOROOT という環境変数で、/usr/lib/go が設定されています。

GOPATH が設定されていれば、パッケージのダウンロードとインストールは go get コマンドで行えます。[*5]

GOPATH で指定したディレクトリ以下に、下記のような形で自動的にインストールされます。

```
$ go get example.org/mkouhei/hoge
$ tree
.
|-- pkg
|   `-- linux_amd64
|       `-- example.org
|           `-- mkouhei
|               `-- hoge.a
`-- src
    `-- example.org
        `-- mkouhei
            `-- hoge
                |-- LICENSE
                |-- README.rst
                `-- hoge.go
```

GOPATH を設定し、import できれば、go build は依存関係を解決してコンパイルできます。バイナリのみを Debian パッケージにする際に、GOPATH が重要です。

4.4 Golang で書かれたソフトウェアの Debian パッケージ作成方法

Golang で書かれたソフトウェアの Debian パッケージ作成方法について説明します。大きく次の 2 パターンがあります。

- バイナリのみ
- ライブラリのみ（もしくはライブラリとバイナリ）

前者ではコンパイルが必要です。Golang の場合、依存するライブラリを全て静的にリンクするため、バイナリの実行そのものには依存関係はありません。ですので、デーモン化するケースでなければ、Depends に記述するパッケージが基本必要ありません。[*6]

後者では、基本的にはコンパイルそのものは必要ありません。しかし、前述の/usr/share/gocode 下にソースコードをインストールする必要があるので、dh-golang パッケージを Build-Depends 記述し、debian/rules で、dh コマンドに--with=golang オプション渡してやる必要があります。

また、バイナリ、ライブラリに問わず、依存する Golang のパッケージを全て Build-Depends に記述する必要があります。[*7]また、Golang では go test でテストを実行したり、go build や Makefile でコンパイルする場合、依存する Golang パッケージを go get で GOPATH で指定したディレクトリにダウンロードおよびインストールされるのですが、Debian パッケージのビルドでは、go get を使わないようにしなくてはいけません。

4.4.1 Debian パッケージ作成用の環境設定

まず dh-golang パッケージをインストールします。Debian パッケージ自体を作ったことが無く、一から環境を整える、という人は、次のコマンドを実行します。

[*5] http://golang.org/cmd/go/#hdr-Download_and_install_packages_and_dependencies

[*6] デーモン化して OS 起動時に自動起動する場合、ログのローテーションで logrotate に依存したり、専用ユーザを作って実行させるのに adduser が必要です。

[*7] これは Golang に限った話ではありませんね。

```
$ sudo apt-get devscripts debhelper fakeroot dh-golang cowbuilder piuparts git git-buildpackage
```

git および git-buildpackage[*8]は必須ではありません。しかし、Golang で書かれたのツールの多くは Git リポジトリで管理されています。また、昔ながらのバージョニングがされないケースがほとんどです。そのため、tarball 自体の提供もされないものが多いので、Git が通常必要になります。

また Golang の Debian パッケージのメンテナンスチーム (the pkg-go team[*9]) では、git-buildpackage でソースパッケージを管理することになっています。[*10]

なお、前述の `go get` コマンドでは、リポジトリから入手して、そのまま `GOPATH` で設定したディレクトリの src ディレクトリ以下に展開されてしまうので、バージョンやコミットが分からないので使わない方がよいでしょう。なお、Build-Depends での依存関係の解決の際にも、Debian パッケージングポリシー上、`go get` コマンドは使えません。

.bashrc に下記のような変数を設定します。

```
export DEBFULLNAME='Kouhei Maeda'
export DEBEMAIL=mkouhei@palmtb.net
```

pbuilder, cowbuilder の chroot イメージを作成します。[*11]

```
$ sudo pbuilder --create
$ cowbuilder --create
```

pbuilder の方は、/var/cache/pbuilder/base.tgz が、cowbuilder の方は/var/cache/pbuilder/base.cow が作成されます。

4.4.2 ライブラリのパッケージ作成

まず Upstream の tarball もしくは SCM のリポジトリを入手します。前述の通り、Git で管理されているケースを想定します。

Serf のビルドの依存関係で (間接的に) 必要になった `https://github.com/huin/gobinarytest` を例にします。`git clone` したら、ブランチとタグ、ログを確認します。

```
$ git clone https://github.com/huin/gobinarytest.git
$ cd gobinarytest
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
$ git tag -l
$ git log | head -5
commit de322f729af68c9e5ecde2bff780d105f0fbe745
Author: Huin <greatred@gmail.com>
Date:   Tue Feb 11 22:50:34 2014 +0000

    Add LICENSE
```

Golang で書かれたツールは、この通り、伝統的なバージョンニングがされないことが多いです。`go get` コマンドを使えば、依存関係のあるパッケージの最新のコミットを入手できることが影響しているのではないかと　思います。

pkg-go team のポリシーとして、こういうケースには、0.0 gitYYYYMMDD のようなバージョンをつける事になっています。[*12]

また、Debian パッケージを作成するときの命名規則もあります。Golang の場合、Perl の CPAN や、Python の PyPI のような公式に一元管理しているパッケージリポジトリが存在しないので、Git のリポジトリ名だけを使うと名前が競合する可能性があります。[*13]

[*8] git-buildpackage はソースパッケージ自体を Git で管理するためのツールです。
[*9] `http://pkg-go.alioth.debian.org/`
[*10] `http://pkg-go.alioth.debian.org/packaging.html#_packaging_in_git`
[*11] pbuilder は cowbuilder をインストールすると自動的にインストールされます。
[*12] `http://pkg-go.alioth.debian.org/packaging.html#_version_numbers`
[*13] Serf をパッケージ化する際にも実際にありました。

そこで、ライブラリパッケージを作成する場合、github.com/huin/gobinarytest の場合には、golang-huin-gobinarytest-dev というパッケージ名にします。[*14]

では、まず `git archive` コマンドを使って tarball を作ります。

```
$ git archive --prefix=golang-huin-gobinarytest/ HEAD | gzip > ../golang-huin-gobinarytest-0.0~git20140211.tar.gz
```

次に Debian パッケージ用の Git リポジトリを作成し、tarball をインポートします。

```
$ cd -
$ mkdir golang-huin-gobinarytest
$ cd golang-huin-gobinarytest
$ git init
$ git import-orig ../golang-huin-gobinarytest-0.0~git20140211.tar.gz
What will be the source package name? [golang-huin-gobinarytest]
What is the upstream version? [0.0~git20140211]
gbp:info: Importing '../golang-huin-gobinarytest-0.0~git20140211.tar.gz' to branch 'master'...
gbp:info: Source package is golang-huin-binarytest
gbp:info: Upstream version is 0.0~git20140211
gbp:info: Successfully imported version 0.0~git20140211 of ../golang-huin-gobinarytest-0.0~git20140211.tar.gz
```

次に `dh_make` コマンドで debian ディレクトリを生成します。このパッケージ自体は 2 条項 BSD ライセンスなので、`--copyright` オプションで指定します。作成後、不要なテンプレートは削除します。ライブラリパッケージですが、

```
$ dh_make -l --copyright bsd -p golang-huin-gobinarytest_0.0~git20140211 -f ../golang-huin-gobinarytest-0.0~git20140211.tar.gz
$ cd debian
$ ls
README.Debian  golang-huin-gobinarytest-dev.dirs     init.d.ex        preinst.ex
README.source  golang-huin-gobinarytest-dev.install  manpage.1.ex     prerm.ex
changelog      golang-huin-gobinarytest.cron.d.ex    manpage.sgml.ex  rules
compat         golang-huin-gobinarytest.default.ex   manpage.xml.ex   shlibs.local.ex
control        golang-huin-gobinarytest.doc-base.EX  menu.ex          source
copyright      golang-huin-gobinarytest1.dirs        postinst.ex      watch.ex
docs           golang-huin-gobinarytest1.install     postrm.ex
$ rm README* golang-huin-gobinarytest* init.d.ex manpage.* menu.ex post* pre* shlibs.local.ex watch.ex
$ ls
changelog  compat  control  copyright  docs  rules  source
```

下記のように変更します。

debian/control

Golang のパッケージには、Build-Depends に golang-go と、また Golang のライブラリパッケージは Depends にも golang-go と、また Build-Depends に dh-golang を記述します。ライブラリの golang-huin-gobinary-test-dev パッケージは、ソースコードなので、Architecture は all にします。

```
Source: golang-huin-gobinarytest
Section: devel
Priority: optional
Maintainer: Kouhei Maeda <mkouhei@palmtb.net>
Build-Depends: debhelper (>= 9.0.0), dh-golang, golang-go
Standards-Version: 3.9.5
Homepage: https://github.com/huin/gobinarytest

Package: golang-huin-gobinarytest-dev
Architecture: all
Depends: golang-go
Description: Helper code for unit testing binary encoding code
 The gobinarytest package is used in testing serialization and
 deserialization. It is particularly useful when some sub-sequences of bytes
 can acceptably be written in any order (e.g if they are generated from
 representations that do not provide order guarantees like maps).
```

debian/rules

Golang 用のポイントとしては、`dh` コマンドに、`--buildsystem=golang` オプションと、`--with=golang` オプションを指定することです。

ライブラリ用には、先ほどの `GOPATH` の下で展開されるパッケージのネームスペースを環境変数 `DH_GOPKG` で指定します。このネームスペースは、通常公開しているリポジトリの URL がベースになりますが、場合によっては

[*14] http://pkg-go.alioth.debian.org/packaging.html#_naming_conventions_2

実際に公開している URL とは別のリダイレクト元の URL を指定することこともあります。upstream のドキュメントやソースコードを確認してみてください。また、override_dh_auto_install を定義し、dh_auto_install -O-buildsystem=golang で上書きします。これにより、このパッケージをビルド後、golang-huin-gobinarytest-dev パッケージをインストールすると、前述の/usr/share/gocode の下にソースコードがインストールされます。

```
#!/usr/bin/make -f
# -*- makefile -*-

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1

export DH_GOPKG := github.com/huin/gobinarytest

%:
	dh $@ --buildsystem=golang --with=golang

override_dh_auto_install:
	dh_auto_install -O-buildsystem=golang
```

この例では、他の Golang パッケージに依存していないので使っていませんが、他のパッケージに依存する場合は、

```
export GOPATH := $(CURDIR)_build:/usr/share/gocode
```

のような設定が必要になります。

debian/copyright

```
ormat: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: gobinarytest
Source: https://github.com/huin/gobinarytest

Files: *
Copyright: 2013, John Beisley <greatred@gmail.com>
License: BSD-2-Clause

Files: debian/*
Copyright: 2014 Kouhei Maeda <mkouhei@palmtb.net>
License: BSD-2-Clause

License: BSD-2-Clause
 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:
 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.
 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.
 .
 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE HOLDERS OR
 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

debian/changelog

dch -r コマンドを実行します。

```
golang-huin-gobinarytest (0.0~git20140211-1) unstable; urgency=low

  * Initial release (Closes: #nnnn)

 -- Kouhei Maeda <mkouhei@palmtb.net>  Wed, 12 Feb 2014 14:47:33 +0900
```

あとは golang-huin-gobuildtest ディレクトリの下で、git buildpackage コマンドを実行すると、予め cowbuilder --create コマンドで作成しておいた/var/cache/pbuilder/base.cow の chroot イメージを使って、パッケージがビルドされます。

4.4.3 バイナリのパッケージ作成

大まかな流れはライブラリの場合と同じです。異なるのは、

- パッケージの命名規則
  golang- という prefix をつける必要はない。[*15]
- debian/control の Build-Depends に dh-golang が不要[*16]
- debian/control のバイナリ用のパッケージの Architecure が all ではなく、any
- debian/rules で、`dh` コマンドに`--with=golang` オプションは不要 (dh-golang 用のオプションのため)

というあたりです。
Serf[*17]をパッケージにする場合の設定例は下記です。

4.4.4 debian/control

Build-Depends の中で、ライブラリの場合と違うのは txt2man と dh-systemd です。実行可能なバイナリファイルは、Debian パッケージのポリシー上、man マニュアルを用意する必要があります。しかし、serf はコマンドラインヘルプは充実しているものの、man マニュアルはありません。なので、txt2man を使って自動生成しました。
dh-systemd は、systemd 用の補助ツールです。

```
Source: golang-serf
Section: web
Priority: optional
Maintainer: Kouhei Maeda <mkouhei@palmtb.net>
Build-Depends: debhelper (>= 9.0.0),
        golang-go (>= 2:1.2),
        golang-hashicorp-memberlist-dev,
        golang-mitchellh-cli-dev,
        golang-hashicorp-logutils-dev,
        golang-mitchellh-mapstructure-dev,
        golang-ugorji-go-dev,
        golang-armon-mdns-dev (>= 0.0~git20140225~8be7e3a),
        golang-ryanuber-columnize-dev,
        dh-systemd,
        txt2man
Standards-Version: 3.9.5
Homepage: http://www.serfdom.io/

Package: serf
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}, adduser
Description: Service orchestration and management tool
 Serf is a service discovery and orchestration tool that is decentralized,
 highly available, and fault tolerant. Serf runs on every major platform:
 Linux, Mac OS X, and Windows. It is extremely lightweight: it uses 5 to 10 MB
 of resident memory and primarily communicates using infrequent UDP messages.
```

debian/rules

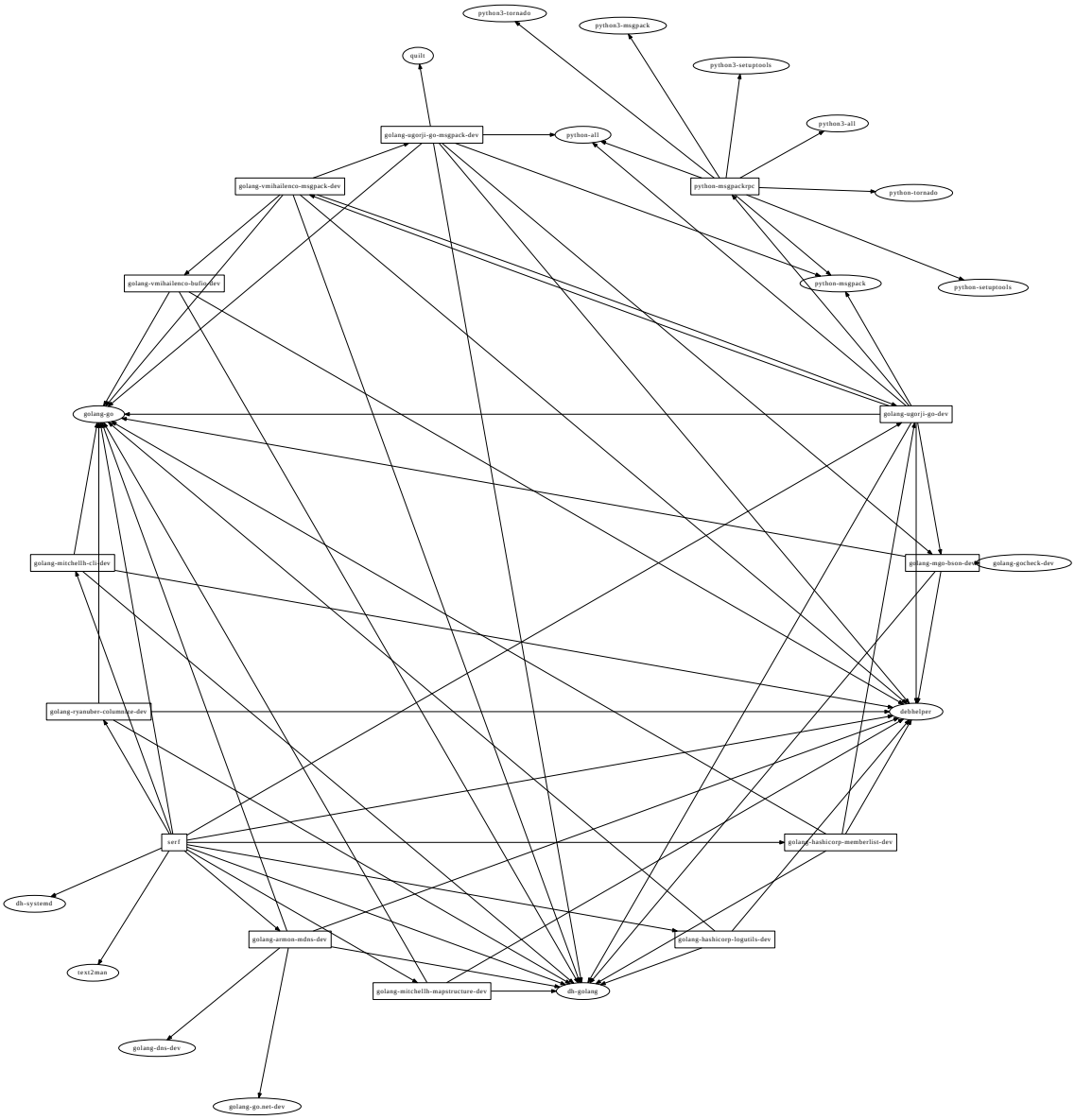
Serf には、Makefile がついているのですが、ビルドに前述の `go get` コマンドを使って、依存するパッケージをカレントディレクトリにダウンロード＆インストールしています。Debian パッケージの作成のポリシー上、ビルド中にリモートホスト上のリソースを取得してはいけないので、upstream の Makefile をそのまま利用できません。そのため、カレントディレクトリと、システムグローバルに設定した GOPATH から、依存パッケージ及び、Serf 自身でインポートしているコードを使うようにしているのが、`override_dh_auto_build` の処理です。
また、前述の man マニュアルの生成は、`override_dh_installman` の処理で行っています。

[*15] Serf の場合は、ソースパッケージが serf となっている、HTTP client library(libserf1) があるので、バッティングするのでソースパッケージ名は golang-serf とするか、Docker のように serf.io とするのが良さそう。
[*16] ライブラリを含める場合は必要です。
[*17] `https://github.com/hashicorp/serf`

```
#!/usr/bin/make -f
# -*- makefile -*-

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1

export DH_GOPKG := github.com/hashicorp/serf
GOPATH := ${CURDIR}/_build:/usr/share/gocode
BLDPATH := obj-$(shell dpkg-architecture -qDEB_BUILD_GNU_TYPE)
export LANG := C

export GOPATH

%:
	dh $@ --buildsystem=golang --with=systemd

override_dh_auto_build:
	mkdir -p ${CURDIR}/_build/src/${DH_GOPKG}
	cp -a ${CURDIR}/*.go ${CURDIR}/serf ${CURDIR}/client \
		${CURDIR}/command ${CURDIR}/testutil ${CURDIR}/_build/src/${DH_GOPKG}/
	go build -v -o ${CURDIR}/_build/serf

override_dh_auto_test:
	cd $(CURDIR)/_build && \
	go list ./... | xargs -n1 go test && \
	$(CURDIR)/scripts/setup_test_subnet.sh && \
	go list ./... | INTEG_TESTS=yes xargs -n1 go test

override_dh_auto_install:
	install -d ${CURDIR}/debian/serf/usr/bin
	test -f ${CURDIR}/_build/serf && \
	install -m 0755 ${CURDIR}/_build/serf ${CURDIR}/debian/serf/usr/bin/

override_dh_installman:
	test -f ${CURDIR}/_build/serf && \
	${CURDIR}/_build/serf -h 2>&1 | txt2man -t SERF -s 1 -v 'Serf Manual' > ${CURDIR}/debian/serf.1
	set -e && ${CURDIR}/_build/serf -h 2>&1 | grep -v version | grep -E '^\s+\w+' | while read line; do \
		subcmd=`echo $$line | awk '{print $$1}'`;\
		desc=`echo $$line | awk '{ $$1=""; print }'`;\
		${CURDIR}/_build/serf $$subcmd -h 2>&1 | txt2man -t SERF-`echo $$subcmd | tr "[:lower:]" "[:upper:]"`
 -s 1 -v 'Serf Manual' > $(CURDIR)/debian/serf-$$subcmd.1; \
		sed -i "/Serf Manual/a .SH NAME\nserf-$$subcmd \- $$desc" $(CURDIR)/debian/serf-$$subcmd.1 ;\
	done
	sed -i '/Serf Manual/a .SH NAME\nserf \- Service orchestration and management tool' $(CURDIR)/debian/serf.1
	sed -i '/[Uu]sage:/i .SH SYNOPSIS' $(CURDIR)/debian/serf*.1
	sed -i '/Available commands are:/i .SH OPTIONS' $(CURDIR)/debian/serf*.1
	sed -i 's/^Options:/.SH OPTIONS/g' $(CURDIR)/debian/serf*.1
	sed -i 's/-/\\-/g' $(CURDIR)/debian/serf*.1
	dh_installman ${CURDIR}/debian/serf*.1

override_dh_auto_clean:
	dh_clean
	rm -rf ${CURDIR}/_build
	rm -rf ${CURDIR}/${BLDPATH}
	rm -f ${CURDIR}/debian/serf*.1

DEB_UPSTREAM_VERSION=$(shell dpkg-parsechangelog | sed -rne 's,^Version: ([^+]+).*,\1,p')
get-orig-source:
	uscan --noconf --force-download --rename --download-current-version --destdir=.
	rm -rf serf-$(DEB_UPSTREAM_VERSION)
	tar xf golang-serf_$(DEB_UPSTREAM_VERSION).orig.tar.gz
	rm -f golang-serf_$(DEB_UPSTREAM_VERSION).orig.tar.gz
	rm -rf serf-$(DEB_UPSTREAM_VERSION)/website
	GZIP=--best tar cz --owner root --group root --mode a+rX \
		-f ../golang-serf-$(DEB_UPSTREAM_VERSION)+dfsg.orig.tar.gz \
		serf-$(DEB_UPSTREAM_VERSION)
	rm -rf serf-$(DEB_UPSTREAM_VERSION)
```

4.5 Serf の依存するパッケージ

最後に、Serf が Build-Depends で必要となり、かつ、まだ Debian パッケージになっていないパッケージを図示しておきます。四角のノードがパッケージになっていないものです。



参考文献

[1] The Go Programming Language, “Command go”, `http://golang.org/cmd/go/`

[2] Alioth, “Debian Go Packaging”, `http://pkg-go.alioth.debian.org/packaging.html`

5　会場での無線 LAN のつなぎ方

野島 貴英

5.1　はじめに

今回試験として、会場側でフィルタ無しのグローバル回線を用意しました。ただ、会場側のセキュリティポリシーにより、wpa-psk AES hidden SSID という方式での提供となります。

以下に Debian マシンでの接続方法を記載します。

また、自分の環境では違うやり方でつながったという方は、野島まで教えて下さい。こちらでもノウハウとして溜めていく予定です。

5.2　wpasupplicant 及び/etc/network/interfaces を利用の場合

もっとも良いマニュアルは、/usr/share/doc/wpasupplicant/README.Debian.gz となります。困った場合はこちらも合わせてご参照下さい。

以下に/etc/network/interfaces の定義について会場の例を記載します。

```
$ sudo aptitude install wpasupplicant
# hidden ssid の元では必ず ap-scan 1,scan-ssid 1 を指定する事。
# 参考：http://bugs.debian.org/358137
$ sudo vi /etc/network/interfaces
-----以下のエントリを追記ここから----------
iface wlan_tokyodebian inet dhcp
     wpa-ssid <<会場の SSID>>
     wpa-psk  <<会場のパスワード>>
     wpa-ap-scan 1
     wpa-scan-ssid 1

-----以下のエントリを追記ここまで----------
#  無線 LAN を有効にする。
$ sudo ifup wlan0=wlan_tokyodebian
#  無線 LAN を無効にする。
$ sudo ifdown wlan0
```

また、ハマってしまった時のデバッグ方法は、/usr/share/doc/wpasupplicant/README.Debian.gz 中の”4. Trubleshooting” の章が便利です。

5.3　その他の無線 LAN 用パッケージを利用の場合

すみません、自分が情報を持たないため、現場で教えて下さい。

Debian 勉強会資料

2014 年 04 月 19 日　　初版第 1 刷発行

東京エリア Debian 勉強会 （編集・印刷・発行）