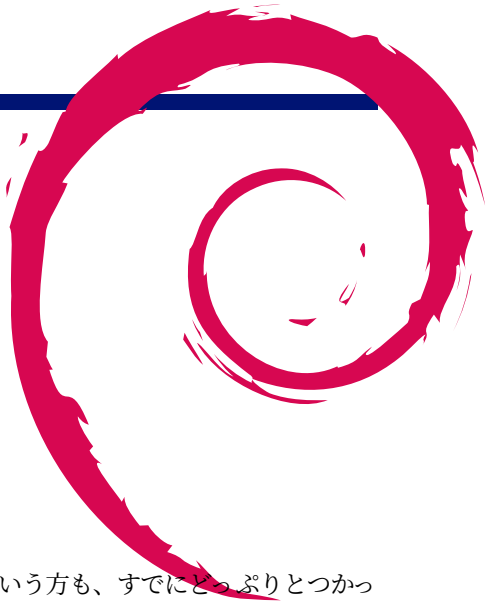
# 東京エリア デビアン 勉強会

**Debian勉強会幹事 上川純一**

2013 年 2 月 9 日



# 1 Introduction

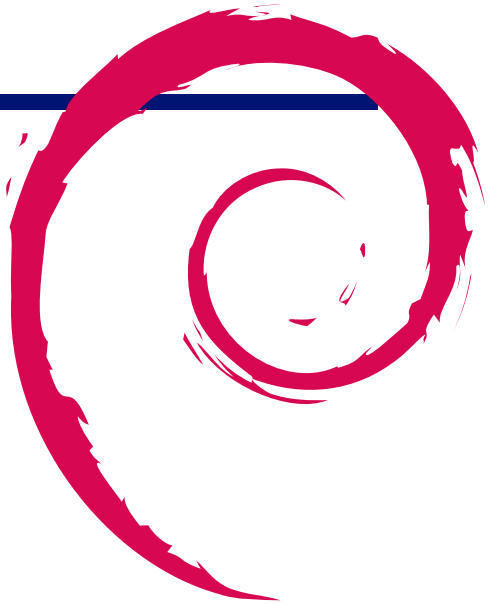
上川 純一

今月の Debian 勉強会へようこそ。これから Debian の世界にあしを踏み入れるという方も、すでにどっぷりとつかっているという方も、月に一回 Debian について語りませんか?

Debian 勉強会の目的は下記です。

- Debian Developer (開発者) の育成。
- 日本語での「開発に関する情報」を整理してまとめ、アップデートする。
- 場 の提供。
  - 普段ばらばらな場所にいる人々が face-to-face で出会える場を提供する。
  - Debian のためになることを語る場を提供する。
  - Debian について語る場を提供する。

Debian の勉強会ということで究極的には参加者全員が Debian Package をがりがりと作るスーパーハッカーになった姿を妄想しています。情報の共有・活用を通して Debian の今後の能動的な展開への土台として、「場」としての空間を提供するのが目的です。



# 2 基本的なパッケージの作成方法手引書

岩松 信洋

## 2.1 本日の目的

Debian パッケージ化されていないソフトウェアをパッケージ化して、ビルドテストとパッケージの変更までを体験してみましょう。

## 2.2 本日の流れ

1. 作業を始める前の準備をする
2. ソフトウェアの動作確認をする
3. パッケージの雛形を作成する
4. debian ディレクトリ以下ファイルを編集する
5. パッケージをビルドする
6. 作成されたファイルを見る
7. パッケージをインストールする
8. パッケージをビルドテストする
9. パッケージのインストール/アンインストールテストをする
10. ソフトウェアの変更し、パッケージ化する
11. 質疑応答

### 2.2.1 記号の説明

**$** が付いている場合は、コンソールからの入力を意味します。**$**は入力せずにコマンドを入力してください。
コマンドラインやファイルの中身で \ が書かれている場所は行が続いている事を意味します。入力しないでください。
**...** は省略を意味します。実際には長い出力がある場合に省略している場合に利用しています。

## 2.3 ルート権限について

本ハンズオンでは、 root 権限を使った作業を行う場合があります。その場合には sudo コマンドを使って作業をします。 **sudo** コマンドが必要な場合にはコマンドラインの説明のところに **sudo** を指定しています。

## 2.4 Debian とは

省略

## 2.5 Debian パッケージについて

省略

## 2.6 作業を始める前の準備をする

### 2.6.1 パッケージメンテナ名の設定

パッケージメンテナの名前とメールアドレスを環境変数に設定します。適当なエディタを使って、 ~/**.bashrc** に以下の例のように追記して保存してください。各項目には自分の名前とメールアドレスをいれてください。

```
export DEBFULLNAME="Nobuhiro Iwamatsu"
export DEBEMAIL=iwamatsu@debian.org
```

保存できたら、ターミナルを起動し、

```
$ source ~/.bashrc
```

を実行してください。

### 2.6.2 パッケージビルドに必要なパッケージのインストール

パッケージビルドに必要なパッケージのインストールをします。 **packaging-dev** パッケージをインストールしてください。

```
$ sudo apt-get install packaging-dev
```

**packaging-dev** はメタパッケージ[*1]で、インストールすることによって Debian パッケージに必要なパッケージがインストールされます。

- build-essential
  パッケージ作成環境にインストールされていることが前提となっているパッケージを提供するメタパッケージです。gcc、 g++、 make 等を提供します。
- debhelper
  パッケージ作成補助ツールです。
- devscripts
  パッケージをメンテナンスするときに有用なスクリプトを提供します。
- dput または dupload
  パッケージのアップロードをする際に使用します。
- lintian
  パッケージを分析し、バグやポリシー違反を検出するツールです。
- pbuilder または cowbuilder または sbuild
  クリーンルームからパッケージを作成するための機能を提供します。
- quilt
  パッチ管理ツールです。

### 2.6.3 パッケージ化するソフトウェア

今回は、 **cwidget** を使ったサンプルプログラム `http://people.debian.org/~iwamatsu/dpd/hello-cwidget-20120922.tar.gz` を用意しました。このソフトウェアを Debian パッケージ化します。ダウンロードして、適当なディレクトリに展

[*1] 他のパッケージに依存するだけのパッケージ

開します。

```
$ mkdir -p ~/debian/dojo/work
$ cd ~/debian/dojo/work
$ wget http://people.debian.org/~iwamatsu/dpd/hello-cwidget-20120922.tar.gz
$ tar -xzf hello-cwidget-20120922.tar.gz
$ ls
hello-cwidget-20120922.tar.gz
hello-cwidget-20120922
```

このソフトウェアは C++ で記述されており、コンパイルに必要なソフトウェアやライブラリがインストールされている場合には、**./configure ; make ; sudo make install** を実行することでコンパイルおよびインストールまでができるようになっています。

## 2.7 ソフトウェアの動作確認をする

### 2.7.1 ソフトウェアをコンパイルしてみる

コンパイルできない/動作しないプログラムをパッケージ化してもしょうがないので、動作確認をします。まずは最低限コンパイルに必要なパッケージをインストールする必要があります。それが **build-essential** パッケージです。これはパッケージ化の場合にも必要です。インストールしていない場合には以下のように実行してインストールしてください。

```
$ sudo apt-get install build-essential
```

先ほど解凍したディレクトリに移動します。移動した後、 **configure** を実行します。

```
$ cd hello-cwidget-20120922
$ ./configure
...
Alternatively, you may set the environment variables \
SIGC_CFLAGS
and SIGC_LIBS to avoid the need to call pkg-config.
See the pkg-config man page for more details.
...
```

実行するとエラーになります。ログを見てみると、 **pkg-config** コマンドがなくてエラーになっている事がわかります。さて、 **pkg-config** コマンドはどのパッケージで提供されているのでしょうか。

## 2.8 提供されているパッケージを探す

Debian で、特定のコマンドやファイルが提供されているパッケージを探すには、 **apt-file** コマンドを利用します。apt-file コマンドは **apt-file** パッケージで提供されているのでインストールしておきましょう。インストールしたら検索用のデータベースを構築します。

```
$ sudo apt-get update
$ sudo apt-get install apt-file
$ sudo apt-file update
```

今回は **pkg-config** コマンドがないので以下のように実行し、パッケージを検索します。

```
$ apt-file search pkg-config
pkg-config: /usr/bin/pkg-config
pkg-config: /usr/share/doc/pkg-config/AUTHORS
pkg-config: /usr/share/doc/pkg-config/NEWS.gz
.....
ruby-pkg-config: /usr/share/doc/ruby-pkg-config/copyright
zsh: /usr/share/zsh/functions/Completion/Unix/_pkg-config
zsh-beta: /usr/share/zsh-beta/functions/Completion/Unix/_pkg-config
```

また、検索探したいファイルのパスがわかっている場合には、パスを指定して検索できます( 例: apt-file search /usr/bin/pkg-config) 。

実行すると、指定したファイルを提供しているパッケージ名が出力されます。結果を確認すると、 **pkg-config** パッケージで提供されていることが分かります。 **pkg-config** コマンドが提供されているパッケージが **pkg-config** パッケージとわかったので、インストールします。

```
$ sudo apt-get install pkg-config
```

再度 **configure** を実行してみましょう。

```
$ ./configure
...
No package 'sigc++-2.0' found

Consider adjusting the PKG_CONFIG_PATH environment variable if you
installed software in a non-standard prefix.
...
```

次は **sigc++-2.0** が見つからないようです。エラーメッセージを見ると pkg-config を使って **sigc++-2.0** の情報を取得しようとしているようですが、見つからないためエラーになっていることが分かります。[*2]

先ほどと同じように **apt-file** を利用して検索し、インストールします。

```
$ apt-file search sigc++-2.0.pc
libsigc++-2.0-dev: /usr/lib/x86_64-linux-gnu/pkgconfig/sigc++-2.0.pc
$ sudo apt-get install libsigc++-2.0-dev
```

再度 configure を実行します。

```
$ ./configure
...
checking for CWIDGET... configure: error: Package \
  requirements(cwidget) were not met:

No package 'cwidget' found

Consider adjusting the PKG_CONFIG_PATH environment variable if you
installed software in a non-standard prefix.
...
```

また、エラーになります。今度は **cwidget** 足りないようなので、再度検索してインストールします。

```
$ apt-file search cwidget.pc
libcwidget-dev: /usr/lib/pkgconfig/cwidget.pc
$ sudo apt-get install libcwidget-dev
```

```
$ ./configure
config.status: creating Makefile
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands
```

**configure** が正常に終了しました。終了すると、 **Makefile** が作成されています。 **make** を実行し、コンパイルします。

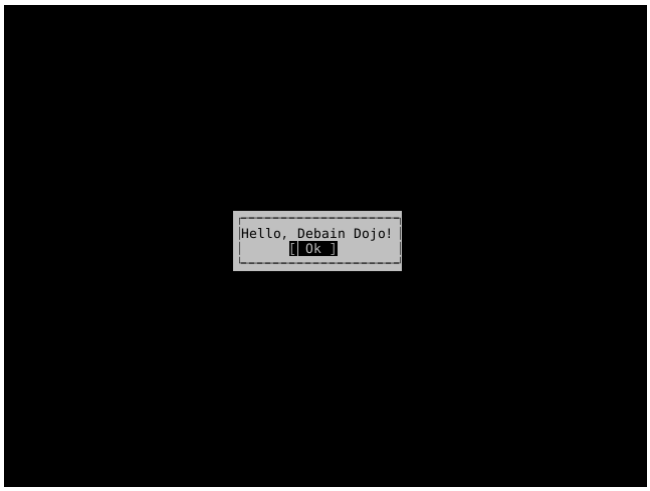
```
$ make
make  all-am
make[1]: ディレクトリ '/home/iwamatsu/debian/dojo/work/hello-cwidget-20120922' に入ります
g++ -DHAVE_CONFIG_H -I.     -I/usr/include/sigc++-2.0 -I/usr/lib/x86_64-linux-gnu/sigc++-2.0/include   -I/usr/include/sigc++-2.0 -I/usr/lib/x8
mv -f .deps/hello_cwidget-hello-cwidget.Tpo .deps/hello_cwidget-hello-cwidget.Po
g++ -I/usr/include/sigc++-2.0 -I/usr/lib/x86_64-linux-gnu/sigc++-2.0/include   -I/usr/include/sigc++-2.0 -I/usr/lib/x86_64-linux-gnu/sigc++-2
make[1]: ディレクトリ '/home/iwamatsu/debian/dojo/work/hello-cwidget-20120922' から出ます
```

コンパイルも正常に終了したので、試しに実行します。

```
$ ./hello-cwidget
```

図のような画面が表示されたでしょうか。

[*2] わからない人もいると思いますが。




ここまではサンプルプログラムの動作確認です。先にどのようなソフトウェアなのか理解するためにもパッケージング化する前にソースコード等を読んでおくことをお勧めします。

## 2.9 Debian パッケージの雛形を作成する

Debian パッケージはソースが格納されているディレクトリの中に **debian** ディレクトリを作成し、その中にパッケージビルド用のスクリプトを置き、実行することによってビルドされます。これらを構成するファイルやスクリプトはある程度決まっているため、雛形が用意されています。この雛形を作成するのが **dh_make** コマンドで、 **dh_make** は **dh-make** パッケージで提供されています。

今まで行った ./configure コマンド等で不要なファイルが作成されているため、作業の前に一度 **hello-cwidget-20120922** ディレクトリを削除します。そして再度展開し、作成されたディレクトリに移動します。

```
$ cd ..
$ rm -rf hello-cwidget-20120922
$ tar -xzf hello-cwidget-20120922.tar.gz
$ cd hello-cwidget-20120922
```

以下のコマンドを実行し、 **dh-make** パッケージをインストールします。

```
$ sudo apt-get install dh-make
```

雛形の作成は以下のコマンドを実行します。

```
$ dh_make --createorig -s
```

**--createorig** オプションはオリジナルソースコードの tar.gz イメージを構築します。 今回はシングルバイナリパッケージ( 一つのソースコードから一つのバイナリパッケージがビルドされる）なので**-s** を指定します。実行すると以下のようなメッセージが表示されるので、 Enter キーを押します。

```
Maintainer name  : Nobuhiro Iwamatsu
Email-Address    : iwamatsu@debian.org
Date             : Wed, 12 Sep 2012 12:46:24 +0900
Package Name     : hello-cwidget
Version          : 20120922
License          : blank
Type of Package  : Single
Hit <enter> to confirm:
```

### 2.9.1 debian ディレクトリ

コマンドを実行すると、 **debian ディレクトリ**が作成され、この中にパッケージ作成に必要な雛形が作成されます。以下に作成されるファイル一覧を示します。

- README.Debian (Debian パッケージの README)

- README.source (ソースの情報を記述する)
- changelog (Debian パッケージのチェンジログ)
- compat (debhelper の API バージョンを指定する)
- control (Debian パッケージ情報)
- copyright (著作権情報)
- dirs (作成するディレクトリ名を指定する)
- docs (インストールするドキュメントファイルを指定する)
- emacsen-install.ex (emacs 用設定ファイル)
- emacsen-remove.ex (emacs 用設定ファイル)
- emacsen-startup.ex (emacs 用設定ファイル)
- hello-cwidget.cron.d.ex (cron 用)
- hello-cwidget.default.ex (debfonf 用)
- hello-cwidget.doc-base.EX (doc-base 用)
- init.d.ex (init.d を使うパッケージ用設定ファイル)
- manpage.1.ex (manpage の雛形)
- manpage.sgml.ex(manpage の雛形)
- manpage.xml.ex (manpage の雛形)
- menu.ex (メニューの雛形)
- postinst.ex (postinst メンテナファイルの雛形)
- postrm.ex (postrm メンテナファイルの雛形)
- preinst.ex (preinst メンテナファイルの雛形)
- prerm.ex (prerm メンテナファイルの雛形)
- rules (パッケージビルドスクリプト)
- source (Debian ソースパッケージ情報を格納するディレクトリ)
- format (Debian ソースフォーマットを指定する)
- watch.ex (アップストリームチェック用ファイル)

### 2.9.2 不要なファイルの削除

今回のパッケージ化に必要ではないファイルを **debian** ディレクトリ以下から削除します。 hello-cwidget は emacs や cron を使わないプログラムなので、基本ファイルのみ( changelog、 compat、 control、 copyright、 rules、 source) のみでよいでしょう。またサフィックスに .ex と.EX がついてるファイルを使用する場合、サフィックスを取り除いて内容を編集する必要があります( 例: watch.ex → watch) 。[*3]

```
$ rm -rf debian/*.ex debian/*.EX debian/README.Debian debian/README.source debian/dirs debian/docs
```

## 2.10 debian ディレクトリ以下ファイルの編集

### 2.10.1 debian/changelog ファイルの編集する

Debian パッケージの変更は全て簡潔に Debian changelog ファイル( debian/changelog) に記載する必要があります。フォーマットに関しては Debian policy 4.4 Debian changelog: debian/changelog を参照してください。 **debian/changelog** ファイルには既に **ITP**(Intent To Package) [*4] のテンプレート書かれているので削除します。以下のように変更します。

[*3] 必須ファイルは changelog、 control、 copyright、 rules です。
[*4] http://www.debian.or.jp/community/devel/abbreviation.html

```
hello-cwidget (20120922-1) unstable; urgency=low

  * Initial release.

 -- Nobuhiro Iwamatsu <iwamatsu@debian.org>  Wed, 12 Sep 2012 12:46:24 +0900
```

### 2.10.2 ライセンスとコピーライトをチェックする

ソフトウェアを Debian パッケージにして Debian にインストールする際、重要な点としてソフトウェアのライセンスがあります。そのソフトウェアのライセンスが DFSG（ Debian Free Software Guideline）に適合するかチェックする必要があり、同梱されているファイルを確認する必要があります。ほとんどの場合、ソースには LICENCE ファイルや COPYING ファイルが提供されていますが、一部のファイルは違うライセンスが適用されている場合もあるためです。もちろんファイル毎にライセンスが書かれておらず、 LICENCE ファイル等で包容的にライセンスを決めている場合もあります。

Debian では簡易的にチェックするためのツールとして **licensecheck** があります。実行するとファイルのライセンスを出力します。**--copyright** オプションをつけた場合、コピーライトホルダも出力します。**-r** オプションは再帰チェックです。

```
$ licensecheck -r .
hello-cwidget.cc: BSD (2 clause)
$ licensecheck -r --copyright .
hello-cwidget.cc: BSD (2 clause)
  [Copyright: HOLDERS AND CONTRIBUTORS / 2012 Nobuhiro Iwamatsu <iwamatsu@debian.org>]
```

簡易的ではありますが hello-cwidget で提供されている hello-cwidget.cc のライセンスは **2 clause BSD License** で、コピーライトは **2012 Nobuhiro Iwamatsu <iwamatsu@debian.org>** が持っていることが分かりました。ツールに頼らず、実際にチェックもしておきましょう。

### 2.10.3 debian/copyright ファイルの編集する

各 Debian パッケージには、著作権と配布条件のライセンス文書が元のままの形式で/usr/share/doc/package/copyright に収録されていなければいけません（ Debian policy 4.5 Copyright: debian/copyright）。パッケージの著作権やライセンス情報を提供するファイルが debian/copyright ファイルとなります。以前はこのファイルのフォーマットは決まっていませんでしたが、今年の 2 月に DEP5（ Debian Enhancement Proposals 5 ）の Machine-readable debian/copyright が受理され、フォーマットが決まりました。フォーマットはヘッダ段落とファイル段落に分かれており、その中に項記述するが決まっています。

DEP5 に基づいて、前で確認したソフトウェアのライセンス用に debian/changelog を変更します。以下のような内容になります。

```
Format: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: hello-cwidget
Source: http://people.debian.org/~iwamatsu/dpd/

Files: *
Copyright: 2012 Nobuhiro Iwamatsu <iwamatsu@debian.org>
License: BSD-2-Clause license

Files: debian/*
Copyright: 2012 Nobuhiro Iwamatsu <iwamatsu@debian.org>
License: BSD-2-Clause license

License: BSD-2-Clause license
 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions are
 met:
 .
 * Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.
 * 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 COPYRIGHT HOLDER 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.
```

### 2.10.4 debian/rules ファイルの編集する

**debian/rules** にはパッケージのビルド手順を書きます。これはパッケージ作成補助ツールである debhelper を使って書くことが多いです。また、 **debhelper 7** になってからビルド手順が簡潔にかけるようになりました。**./configure ; make ; sudo make install** だけでコンパイルとインストールができるソフトウェアは以下の内容だけでパッケージのビルドができます。

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

%:
    dh $@
```

### 2.10.5 debian/control ファイルの編集する

debian/control ファイルにはパッケージ全体の情報とビルドされるパッケージの情報を書きます。( Chapter 5 - Control files and their fields) まずパッケージ全体の情報を書いてみます。

- Source
  パッケージの元になるソースの名前を書きます。
- Section
  パッケージを分類したアプリケーション分野を指定します。
- Priority
  パッケージの優先度を指定します。
- Maintainer
  パッケージメンテナの名前とメールアドレスを書きます。
- Build-Depends パッケージを生成するときに利用するパッケージを指定します。 debhelper は一番利用されているパッケージ作成補助ツールです。
- Standards-Version
  パッケージが準拠している Debian ポリシーマニュアルのバージョンを指定します。現在の最新バージョンは 3.9.4 です。
- Homepage
  ソースが入手できる Web サイトを書きます。

```
Source: hello-cwidget
Section: devel
Priority: extra
Maintainer: Nobuhiro Iwamatsu <iwamatsu@debian.org>
Build-Depends: debhelper (>= 9.0.0)
Standards-Version: 3.9.4
Homepage: http://people.debian.org/~iwamatsu/dpd/
```

次にビルドされるパッケージの情報を書きます。 hello-cwidget では hello-cwidget というプログラムを提供するので、 hello-cwidget という一つのパッケージを提供するようにします。

各項目は以下のような意味です。

- Package
  パッケージ名を書きます。
- Architecture
  ビルド可能なマシンアーキテクチャを指定します。スクリプト言語や画像ファイルなど、アーキテクチャに依存しない場合は **all** を指定します。どのアーキテクチャでも動作するとは **any** を、特定のアーキテクチャのみで動作する場合はその Debian アーキテクチャを指定します。
- Depends
  依存しているパッケージを指定します。**$shlibs:Depends, $misc:Depends** はビルドされたファイルから自動的に依存ファイルを検出し、依存パッケージ名に置換されます。
- Description
  パッケージの説明を書きます。 1 行目は短い説明を書き、 2 行目以降により詳細な説明を書きます。 2 行目以降は先頭に 1 文字空白を入れる必要があります。

```
Package: hello-cwidget
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: Debian Packaging Hands-on sample program
 This is sample program of Debian Hands-on with OSC2009
 Tokyo/Spring and Debian Packaging Dojo.
 This is very easy program that uses cwidget.
```

### 2.10.6 パッケージをビルドする

パッケージのビルドには **debuild** コマンドを使います。 debuild コマンドは **devscripts** パッケージで提供されています。 **debuild -us -uc** を実行し、パッケージビルドをしてみましょう。ちなみに**-us** はソースパッケージに PGP 署名しない、**-uc** は .changes ファイルに PGP 署名しないというオプションです。

```
$ debuild -us -uc
...
dpkg-buildpackage: full upload (original source is included)
Now running lintian...
W: hello-cwidget source: newer-standards-version 3.9.4 (current is 3.9.3)
W: hello-cwidget: new-package-should-close-itp-bug
W: hello-cwidget: binary-without-manpage usr/bin/hello-cwidget
Finished running lintian.
```

パッケージのビルドが成功すると最後に lintian というパッケージチェックツールが実行されます。いくつか警告が出ていますので説明しておきます。

- newer-standards-version 3.9.4 (current is 3.9.3)
  Standart-Version フィールドの値が新しすぎるという警告です。 lintian が まだ 3.9.4 に対応していないためこの警告が出ます。
- new-package-should-close-itp-bug
  ITP のバグ番号が changelog ファイルに書かれていないという警告です。新しいパッケージを作成し、 Debian にインストールする場合には ITP（ Intent To Package）というバグを登録し、 Debian パッケージの changelog ファイルにこのバグ番号を書くことによってパッケージのアップロード時に対象のバグが閉じられます。他にも方法がありますが、この方法がよく使われます。

- binary-without-manpage usr/bin/hello
  usr/bin/hello の man ファイルがないという警告です (Debian-policy 12.1)。

## 2.11 作成されたファイルを確認する

**debuild** を実行した後には Debian パッケージだけでなく、いくつかファイルが作成されています。

```
$ ls ..
hello-cwidget-20120922
hello-cwidget_20120922-1.debian.tar.gz
hello-cwidget_20120922-1.dsc
hello-cwidget_20120922-1_amd64.build
hello-cwidget_20120922-1_amd64.changes
hello-cwidget_20120922-1_amd64.deb
hello-cwidget_20120922.orig.tar.gz
```

- *.debian.tar.gz
  Debian パッケージ用に修正したファイルをまとめたもの。 debian ディレクトリ以下のファイル。
- *.dsc
  Debian のソースパッケージを構成するための情報が書かれたファイル。
- *.orig.tar.gz
  開発元のソースコード。
- *.build
  ビルドログ。
- *.changes
  パッケージ作成後の情報が書かれたファイル。
- *.deb
  Debian パッケージ。

## 2.12 パッケージをインストールする

パッケージが無事ビルドできたら、実際にインストールしてみます。インストールには **debi** コマンドを使ってインストールします。インストールしたら、動作確認をしてみましょう。

```
$ sudo debi
$ which hello-cwidget
/usr/bin/hello-cwidget
$ hello-cwidget
```

## 2.13 パッケージのビルドテストをする

パッケージができた後はパッケージのテストを行います。パッケージのビルドテストには **pbuilder** を使います。pbuilder は chroot を使って Debian OS として必要な最低限の環境からパッケージビルドを行うツールです。これによって、パッケージビルドに必要なパッケージが漏れていないかチェックできます。また cowbuilder は chroot 環境を構築する時に copy-on-write を利用できるようにするツールを提供します。

### 2.13.1 pbuilder パッケージのインストール

```
$ sudo apt-get install pbuilder cowbuilder
```

インストールが完了したら、 pbuilder で cowbuilder を利用できるように、 pbuilder の設定ファイル（ ~/**.pbuilderrc**）に以下の内容を追記します。

```
PDEBUILD_PBUILDER=cowbuilder
```

### 2.13.2 pbuilder 環境の構築

ビルドテストを行う前に base システムイメージを構築する必要があります。以下のように実行します。

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

### 2.13.3 パッケージのビルドテスト

パッケージのビルドテストを行うには、対象とする Debian パッケージのソースが展開されたディレクトリで **pdebuild** を実行します。

```
$ pdebuild
...
```

### 2.13.4 pdebuild によるパッケージビルドエラー

実行するとビルドエラーになります。なぜエラーになるのでしょうか。考えてみましょう。

### 2.13.5 再ビルドテスト

エラーになる理由は先にインストールしたパッケージ **libcwidget-dev** をパッケージビルド時の依存関係を記述するフィールド **Build-Depends** に追加していないためです。追加してみましょう。

```
Source: hello-cwidget
Section: devel
Priority: extra
Maintainer: Nobuhiro Iwamatsu <iwamatsu@debian.org>
Build-Depends: debhelper (>= 9.0.0), libcwidget-dev
Standards-Version: 3.9.4
Homepage: http://people.debian.org/~iwamatsu/dpd/
```

追加したら **pdebuild** を実行し、再ビルドします。今度はビルドができるはずです。

```
$ pdebuild
...
```

## 2.14 パッケージのインストール/アンインストールテストをする

パッケージがビルドできただけでは喜んではいけません。インストール/アンインストールのテストも行いましょう。パッケージのインストール/アンインストールのテストには **piuparts** パッケージを使います。

### 2.14.1 piuparts のインストール

以下のように実行し、インストールします。

```
$ sudo apt-get install piuparts
```

### 2.14.2 パッケージのインストール/アンインストールテスト

piuparts も pbuilder と同様に最低限の環境からのインストールをチェックします。

```
$ sudo piuparts -d unstable ../hello-cwidget_20120922-1_amd64.deb
...
0m41.9s DEBUG: Removed directory tree at /tmp/tmpHliOKO
0m41.9s INFO: PASS: All tests.
0m41.9s INFO: piuparts run ends.
```

その他詳しい使い方は マニュアルを参照してください。

## 2.15 プログラムの編集をする

hello-cwidget を実行して、違和感のある方がおられたと思います。そう、 **Debian** が **Debian** になっていました。これはよくないので修正しましょう。 Debian source-format バージョン 3.0 からは **quilt** によるパッチシステムが標準で利用できるようになっており、これを使うためのツールが整備されています。

### 2.15.1 ファイルを修正する

早速ファイルを修正します。変更したい箇所は Typo なので grep 等で検索するとよいでしょう。

```
$ grep -r Debain *
hello-cwidget.cc:dialogs::ok(L"Hello, Debain Dojo!",
```

### 2.15.2 修正した箇所をパッチにする

修正した箇所をパッチするには **dpkg-source** コマンドに **--commit** オプションをつけて実行します。実行すると保存するファイル名を聞かれるので、適当な名前をつけて、エンターキーを押します。パッチにファイル変更内容等が書かれたファイルが表示されますので、この内容を適当に変更して保存します。パッチファイルのヘッダに書かれた情報は DEP3 （ Debian Enhancement Proposals 3 ）の Patch Tagging Guidelines に基づいています。保存すると debian/patches ディレクトリにパッチが保存され、 debian/patches/series ファイルに適用するパッチとして登録されます。

```
$ dpkg-source --commit
dpkg-source: info: local changes detected, the modified files are:
 hello-cwidget-20120922/hello-cwidget.cc
Enter the desired patch name: fix-typo
```

```
Description: <short summary of the patch>
 TODO: Put a short summary on the line above and replace this paragraph
 with a longer explanation of this change. Complete the meta-information
 with other relevant fields (see below for details). To make it easier, the
 information below has been extracted from the changelog. Adjust it or drop
 it.
 .
 hello-cwidget (20120922-1) unstable; urgency=low
 .
   * Initial release.
Author: Nobuhiro Iwamatsu <iwamatsu@debian.org>

---
The information above should follow the Patch Tagging Guidelines, please
checkout http://dep.debian.net/deps/dep3/ to learn about the format. Here
are templates for supplementary fields that you might want to add:

Origin: <vendor|upstream|other>, <url of original patch>
Bug: <url in upstream bugtracker>
Bug-Debian: http://bugs.debian.org/<bugnumber>
Bug-Ubuntu: https://launchpad.net/bugs/<bugnumber>
Forwarded: <no|not-needed|url proving that it has been forwarded>
Reviewed-By: <name and email of someone who approved the patch>
Last-Update: <YYYY-MM-DD>

--- hello-cwidget-20120922.orig/hello-cwidget.cc
+++ hello-cwidget-20120922/hello-cwidget.cc
@@ -8,7 +8,7 @@ int main(int argc, char **argv)
    toplevel::init();

    widgets::widget_ref dialog =
-       dialogs::ok(L"Hello, Debain Dojo!",
+       dialogs::ok(L"Hello, Debian Dojo!",
            util::arg(sigc::ptr_fun(toplevel::exitmain)));

    toplevel::settoplevel(dialog);
...

dpkg-source: info: local changes have been recorded in a new patch: hello-cwidget-20120922/debian/patches/fix-typo
```

各項目は以下のような意味を持ちます。

- Description
  パッチの説明を書きます。
- Origin
  パッチの提供者を書きます。また開発元や他の」サイトからパッチを持ってきているとき、その URL を書きます。
- Bug
  開発元の BTS 登録されているバグ番号がある場合に書きます。
- Bug-Debian
  Debian のバグ番号の URL を書きます。
- Bug-Ubuntu
  Ubuntu のバグ番号の URL を書きます。
- Forwarded
  バグの転送先、その必要の可否を書きます。
- Reviewed-By
  パッチのレビュアを書きます。
- Last-Update
  パッチの更新日を書きます。
- Applied-Upstream
  開発元で適用された/されている場合、そのソースを示す URL を書きます。

全て各必要はなく、状況に合わせて項目を埋めていきます。今回の場合は以下のようになります。

```
Description: Fixed message in dialog
 This patch is fixed typo form Debain to Debian.
Forwarded: not-needed
Origin: other
Author: Nobuhiro Iwamatsu <iwamatsu@debian.org>
Last-Update: 2012/09/22

--- hello-cwidget-20120922.orig/hello-cwidget.cc
+++ hello-cwidget-20120922/hello-cwidget.cc
@@ -8,7 +8,7 @@ int main(int argc, char **argv)
    toplevel::init();

    widgets::widget_ref dialog =
-       dialogs::ok(L''Hello, Debain Dojo!'',
+       dialogs::ok(L''Hello, Debian Dojo!'',
            util::arg(sigc::ptr_fun(toplevel::exitmain)));

    toplevel::settoplevel(dialog);
```

debian/patches/series ファイルを確認してみます。

```
$ cat debian/patches/series
fix-typo
```

### 2.15.3 差分を適用したパッケージをビルドする

差分を適用したパッケージをビルドするには通常のパッケージビルドと変わりません。**debuild** コマンドを使ってビルドします。debian/patches/series に書かれているパッチが順に適用され、パッケージがビルドされます。

```
$ debuild -us -uc
....
```

作成されたパッケージをインストールして動作確認してみましょう。Typo は治っているでしょうか。

```
$ debi
...
$ hello-cwidget
```


この後には **pbuilder** と **piuparts** を使ってパッケージのテストを行う事も忘れずに。

## 2.16 質疑応答

以上で、「基本的なパッケージの作成方法」は終了です。何か質問等はありますか?

# 3 最新のパッケージング事情の説明と使い方

岩松 信洋

## 3.1 debhelper 7 / 9

よく利用されるパッケージ作成補助ツールに debhelpler があります。 バージョン 7 から大幅な改良が行われました。これらについて簡単に紹介します。

## 3.2 debian/rules の必須ターゲット

Debian パッケージは debian ディレクトリ以下にある ファイルによって作成されます。その中心となるスクリプト debian/rules です。これは GNU Makefile で動作し、いくつかの必須ターゲットがあります。以下に簡単に説明します。

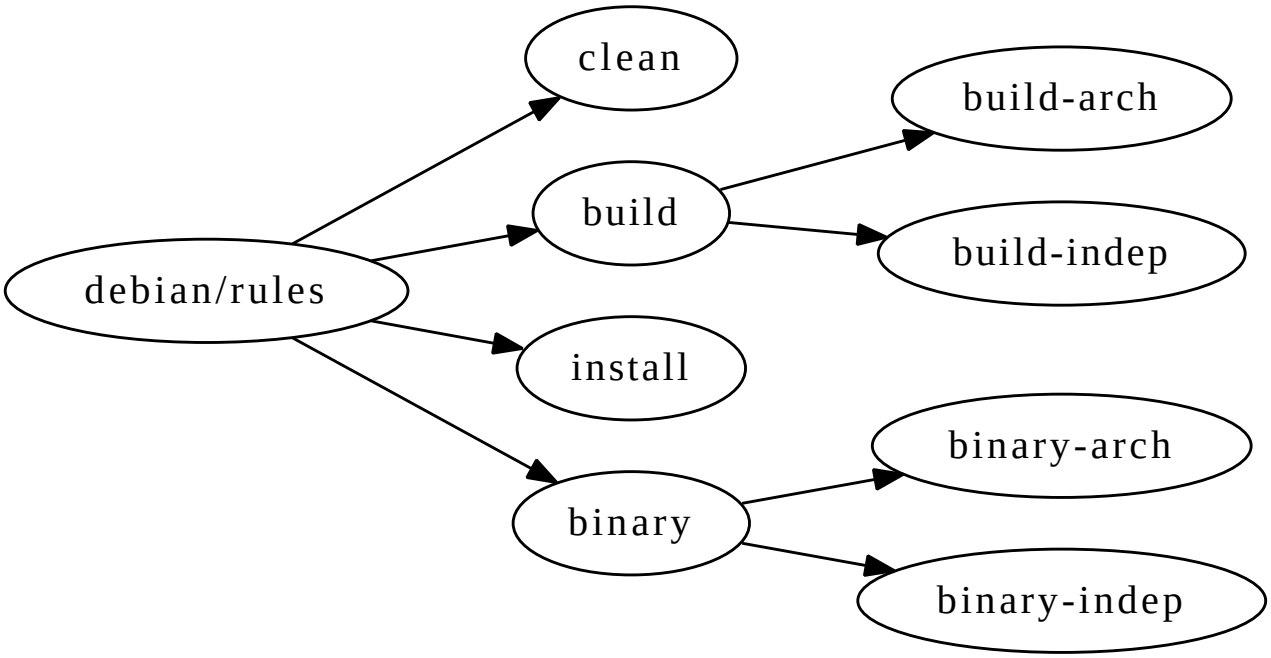



図 1 debian/rules とターゲットの関係

- clean ターゲット
  ビルドツリー内にある、生成されたりコンパイルされたりしたファイルを削除します。
- build ターゲット
  ビルドツリー内にプログラムやドキュメントをビルドします。

  - build-arch ターゲット
    ビルドツリー内にアーキテクチャーに依存したコンパイルしたプログラムをビルドします。
  - build-indep ターゲット
    ビルドツリー内にアーキテクチャーに依存しないファイル( ドキュメントなど) をビルドします。
- install ターゲット:
  debian ディレクトリー以下にある各バイナリーパッケージのファイルツリーにファイルをインストールします。( 必須ではありませんが、よく利用されます。)
- binary ターゲット: 全てのバイナリーパッケージを作ります。
  - binary-arch ターゲット
    アーキテクチャーに依存したバイナリーパッケージ (Architecture: any) を作ります。
  - binary-indep ターゲットアーキテクチャーに依存しないパッケージ (Architecture: all) を作ります。

### 3.2.1 各処理の隠蔽化

バージョン 7 から Debian パッケージの作成に必要な処理が隠蔽化され、シンプルに debian/rules ファイルが書けるようになりました。先の説明にあったように、今までは必須ターゲットを記述する図 2 のような書き方が主流だったのですが、バージョン 7 以降は図 3 のような書き方ができるようになっています。

```
...
build: build-stamp
build-stamp:
    dh_testdir
    # Add here commands to compile the package.
    $(MAKE)

    touch build-stamp

clean:
    dh_testdir
    dh_testroot
    rm -f build-stamp install-stamp
....
```

図 2　バージョン 7 前の debian/rules

```
%:
    dh $@
```

図 3　バージョン 7 以降の debian/rules

パッケージをビルドすると分かりますが、バージョン 7 以降では各ターゲットで必要な処理に対応する debhelper スクリプトが呼ばれるようになっています( 図 4) 。

```
$ debuild -us -uc
...
 dh_auto_test
   fakeroot debian/rules binary
 dh binary
    dh_testroot
    dh_prep
    dh_installdirs
    dh_auto_install
...
```

図 4　パッケージビルドログの例

各ターゲット内や各 debhelper スクリプトが呼ばれる前に処理を行いたい場合には、オーバライド機能を使って各処理をオーバライドします。例えば、 **dh_auto_test** を行う前に **Foo** と出力したい場合には図 5 のように書きます。

```
%:
    dh $@

override_dh_auto_test:
    echo "Foo"
    dh_auto_test
```

図 5　オーバライドの例

### 3.2.2　サードパーティツール指定方法

debhelper ではパッケージを容易に作成できるツール（ dh_*）が提供されていますが、自分で debhelper のツールを作って、それを Debian パッケージの作成に利用することもできるようになっています。例えば、各プログラミング言語用向けに対応し debhepler ツールは各プログラミング言語のメンテナンスチームによって開発・提供されています。debhelper 7 以降でサードパーティツールを使うには、 debhelper を使う時にツール名を指定します。例えば、 ruby のパッケージを作成するには補助ツールである dh_ruby を使う（ 実際にはちょっと違うのだけど）には図 6 以下のようにします。

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

%:
    dh $@ --buildsystem=ruby --with ruby
```

図 6　ruby の場合

## 3.3　source 3.0

Debian 6.0（ Squeeze）から採用されている Debian ソースパッケージのフォーマット”3.0 (quilt)” について説明します。

まずは”3.0 (quilt)”の前に、いままで一般に使われてきたフォーマット (”1.0”) を簡単にまとめます。”1.0” では、ソースパッケージは以下の 3 ファイルで構成されます。

- *packagename*`-`*upstreamversion*`.orig.tar.gz`
- *packagename*`-`*debianversion*`.diff.gz`
- *packagename*`-`*debianversion*`.dsc`

なお、正確には `1.0` は 2 種類あり、上の通常のパッケージのほかに”Debian native な”パッケージがあります。Debian native パッケージは次の 2 ファイルで構成されます。

- *packagename*`-`*version*`.tar.gz`
- *packagename*`-`*version*`.dsc`

ここで、`*.orig.tar.gz` には、通常上流の元のソースツリーが含まれます。`*.diff.gz` には、ソースパッケージからパッケージなどをビルドするのに必要なスクリプトなどが入った `debian/` ディレクトリや、上流のソースに対するパッケージメンテナの変更が含まれます。しかしこのファイル構成には

1. アーカイブの圧縮形式に gzip しか使えない
2. 複数のアーカイブで構成される上流のソースがそのまま扱えない
3. メンテナが当てたソースへのパッチが全部つながってしまっている
4. `debian/` 以下にバイナリファイルが直接置けない

などの問題点があります。

そこで、さまざまな方法が検討されました。問題 1 は、これにより上流が bz2 で配布していても gz に圧縮し直さなければならないという問題がありました。`*.orig.tar.gz` の中身が上流のアーカイブの実体である、といった方法なども（ ちょっと無駄ですが...）使われてきました。この方法はビルド時にその tarball を展開して作業します。パッケージビルドサポートツールである CDBS（ Common Debian Build System）にもこの方法へのサポートがあります。

問題 2 は問題 1 と同様の方法で複数の tarball が入った`*.orig.tar.gz` を用意して対応していました。問題 3 は、当たっているパッチのそれぞれがどんな意図で行われたのかがわからない、ということ、また、 `debian/`以下のファイルも上流ソースへのパッチも一緒くたになってしまっていること、が問題でした。そこでまとまった意味のある単位に分割されたパッチをまず用意しておき、それらを `debian/patches/` 下に配置し、その細かいパッチをビルド時に当てる/外すフレームワーク (patch system) が利用されています。これには dpatch や quilt などがあります。なお、この細かいパッチのそれぞれには、先頭にパッチの意図を説明する文章を記述することが推奨されています (`http://dep.debian.net/deps/dep3/`)。

問題 4 は、バイナリファイルの diff を取ろうとしても普通の patch ではできないことが原因なので、 uuencode などでテキストに落として patch を取る、といった手法が用いられてきました。

このような問題を解決するために新たなソースパッケージのフォーマットも検討されました。それが”`3.0 (quilt)`” フォーマット（ と”`3.0 (native)`”フォーマット）です。 `3.0 (quilt)` は次の 3 つ以上のファイルで構成されます。

- *packagename*`-`*upstreamversion*`.orig.tar.`*ext*
- *packagename*`-`*upstreamversion*`.orig-`*component*`.tar.`*ext*（ 任意）
- *packagename*`-`*debianversion*`.debian.tar.`*ext*
- *packagename*`-`*debianversion*`.dsc`

なお、 `1.0` にあった Debian native パッケージに相当する `3.0 (native)` は次の 2 つのファイルで構成されます。

- *packagename*`-`*version*`.tar.`*ext*
- *packagename*`-`*version*`.dsc`

ここで、まず tar の拡張子部分 *ext* に gz のほか、 bz2, lzma, xz が利用できるようになりました。これにより問題 1 が解決されました（ `3.0 (native)` における主な変更点はこれです）。また、 *component* の部分を適当に変えることにより複数の tarball をきちんと扱えるようになりました。これが問題 2 を解決します。次に、 `debian/`下のファイルはすべて `*.debian.tar.gz` に入れることになりました。これですべてが混ざった状態はなくなりました。さらに、 `debian/patches/` 下のパッチが、パッチシステム quilt と基本的に同じ方法で `dpkg-source(1)` によって”ソースパッケージの展開時に” 自動的に当たるようになりました。これにより、ビルド時にパッチを当てるように `debian/rules` ファイルを記述する必要はなくなりましたし、 `debian/control` ファイルに `Build-Depends: quilt` などと書く必要もなくなりました。これらによって問題 3 は解決されました。問題 4 については、 `debian/`下のファイルを diff として保持することはもはやなくなったので解決し、`*.debian.tar.`*ext* に直にバイナリファイルを配置できます。

このように Debian のソースファイルフォーマットは新しいバージョンに移行しています。今後は ”3.0 (quilt)”、”3.0 (native)”を使うようにしましょう。

## 3.4 hardening

次期リリース Debian 7.0 から **Security hardening build flags** を有効にしたパッケージが提供されるようになります。これはパッケージ構築時にセキュリティを強化するコンパイルフラグを (デフォルトで) 有効にするというものです。現在、以下の 4 点を有効にする必要があります。

- Format string checks（ -Wformat -Werror=format-security ）
  format 使う関数（ 例えば printf）の使用が問題を引き起こす可能性がある場合に警告する。
- FORTIFY_SOURCE

文字列やメモリの操作を行う関数を使用する際にバッファオーバーフローを検出する。

- -fstack-protector --param=ssp-buffer-size=4
  スタック破壊攻撃等によるバッファオーバーフローをチェックするための追加コードを生成する。4 バイトを超える配列を持つ関数を対象にする。
- -z,now,-z,relro
  リロケーション領域 (GOT など) をリードオンリーにする。

これらのコンパイルオプションを自動的に CFLAGS 変数等に設定する機構は今の所なく、debian/rules に記述する必要があります。いくつか方法がありますが、よく使われるのは **dpkg-buildflags** で Debian で推奨されるコンパイルオプションを取得し、各変数に設定するというものです。図 7 に例を示します。

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

CPPFLAGS:=$(shell dpkg-buildflags --get CPPFLAGS)
CFLAGS:=$(shell dpkg-buildflags --get CFLAGS)
CXXFLAGS:=$(shell dpkg-buildflags --get CXXFLAGS)
LDFLAGS:=$(shell dpkg-buildflags --get LDFLAGS)

%:
    dh $@
```

図 7　hardening 設定方法

その他の方法は `http://wiki.debian.org/Hardening` を参照してください。

また hardening が有効なバイナリになっているか簡易的にチェックするためのツール **hardening-check**（hardening-includes パッケージで提供されています）、**blhc** があります。前者はバイナリからチェック、後者はビルドログからチェックするという違いがあります。blhc によるチェック結果は buildlogcheck[*5] から参照できます。

## 3.5　パッケージのスポンサーアップロード

Debian Deloper 以外のパッケージアップロード権限をもってない人はアップロード権限を持っている Debian Developer にパッケージをアップロードしてもらう必要があります。（Debian Maintainer はアップロードできるが、制限がある。）このような場合パッケージをどこかに置いて、パッケージをチェックしてもらった後で Debian にアップロードされます。チェックしてもらうパッケージを置いたり、このチェックの課程や機械的にできるチェックを行えるサービス **mentors.debian.net** ができました。アップロード権限がない人はこのサービスを使うようにしましょう。

また、代わりにパッケージをアップロードしてくれる人をスポンサーと言うのですが、スポンサーがいない場合、自分で探してアップロードしてもらう必要がありました。しかし状況がトラッキングされていないので、どのパッケージがまだアップロードされていないとか、誰がアップロード担当になっていないのかわからない状態になっていました。現在は **sponsorship-requests** として、BTS で管理することが推奨[*6] されるようになりました。今後は **sponsorship-requests** を行うようにしましょう。

[*5] `https://buildd.debian.org/~brlink/bytag/W-compiler-flags-hidden.html`
[*6] http://lists.debian.org/debian-mentors/2012/01/msg00578.html

**Debian 勉強会資料**

2013 年 2 月 9 日　　初版第 1 刷発行
東京エリア Debian 勉強会（ 編集・ 印刷・ 発行）