第 91 回 東京エリア Debian 勉強会資料

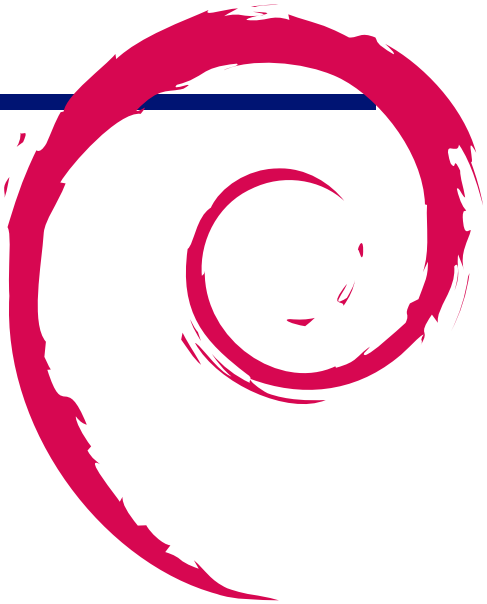
月刊 Deb専

日本唯一のDebian専門月刊誌

2012 年 8 月 18 日

特集: Debconf Report

特集: 月刊Debhelper



1 はじめに

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?

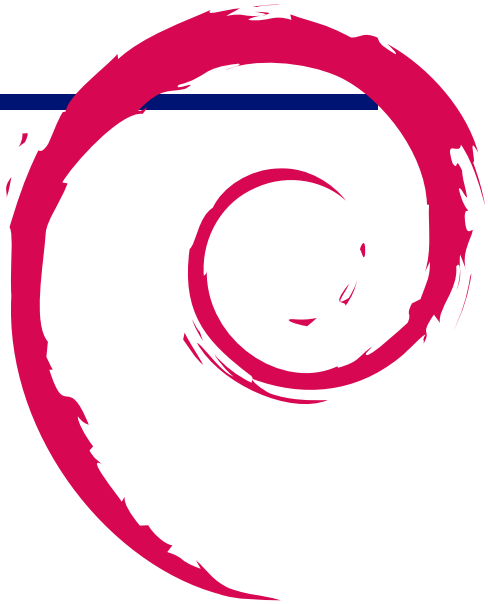
Debian 勉強会の目的は下記です。

- 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。

デビアン勉強会

目次



2　事前課題

上川 純一

今回の事前課題は以下です:

1. Debian 19 周年にちなんで思うことを教えてください。

この課題に対して提出いただいた内容は以下です。

2.1　日比野 啓

19 周年にちなんで、ということでもないが、今年は Haskell のライブラリを公開する予定なので、自分でパッケージングしてメンテナになりたい。あと Debian Haskell チームにも参加していきたい。

2.2　なかおけいすけ

変わらず使い続けます。で、ちょっとずつ、 Contribute していこうと思ってます。

2.3　野島　貴英

抱負:「 我がああ Debian の〜力(りょく) はあああああ世界いいいイチイイイイイイイ! ！！！」とか言えると素敵かも... (〜の部分は適当に...)

ちょっとぐらい他の人がやった事がないこと 1 つでも Debian ネタで出来たらいいな...「 名状しがたい何かのハードウェア」で Debian 動かしてみるとかかな...

2.4　キタハラ

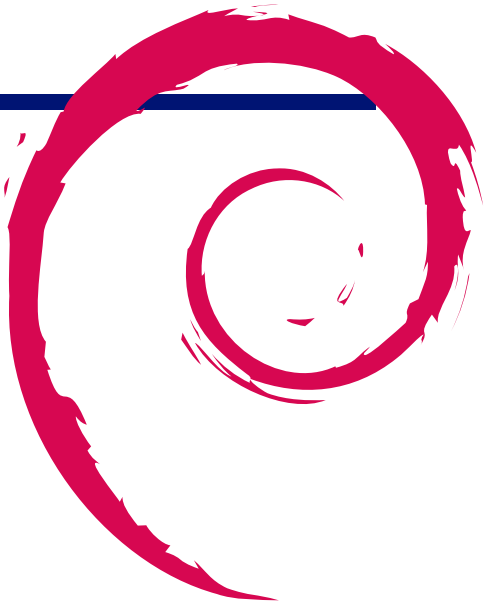
諸般の都合で廃止されてしまった、会社の debian サーバを復活させたいですね。

2.5　dictoss(杉本　典充)

kfreebsd を使って楽しみながら、 deb していきます。

2.6　石井一夫

DebConf を日本で開催できるならば、是非実現させたい。



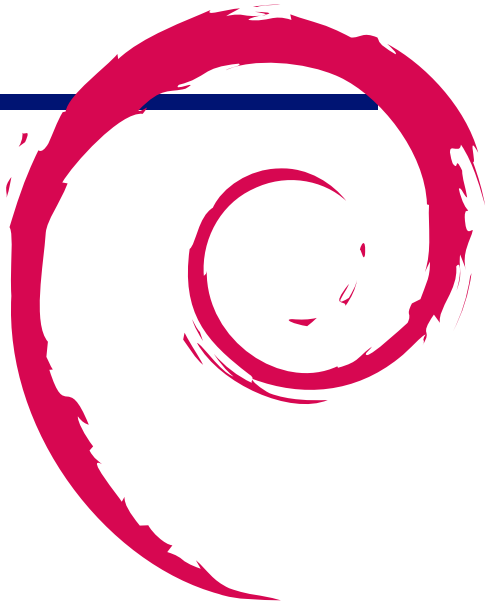
3 最近の Debian 関連のミーティング報告

上川 純一

3.1 東京エリア Debian 勉強会 90 回目報告

東京エリア Debian 勉強会。先週の土曜日は第 90 回東京エリア Debian 勉強会でした。ニカラグア開催の Debconf が先週あり、先月は大統一 Debian 勉強会で京都大学にみんな集まった、ということで若干息切れ気味の開催でした。

事前課題は Debconf のビデオをみての感想でした。個人的には ARM の 64 ビット関連の話題が気になるところ。MacBookAir にインストールする話題を上川がしました。 rEFIt とか gptsync とか知らない人は知らない話題がたくさん。



4 Debian Conference 2012 参加報告

岩松 信洋

4.1 Debconf とは

2012 年度の Debconf は 7 月 8 日から 7 月 14 日まで、ニカラグアのマナグアで行われました。日本からは、安井 卓、大岩 達也、矢吹 幸治、山根 秀樹、岩松 信洋が参加しました。

4.1.1 Debconf の歴史・経緯

Debian Conference `http://debconf12.debconf.org/` は Debian の開発者たちが一同に介するイベントです。通常顔をあわせることのないメンバーたちが一同に介し友好を深め、技術的な議論を戦わせます。過去の開催履歴を見てみると表 1 のようになります。

表 1　歴代の Debconf 参加者推移

年	名前	場所	参加人数
2000	debconf 0	フランス ボルドー	
2001	debconf 1	フランス ボルドー	
2002	debconf 2	カナダ トロント	90 名
2003	debconf 3	ノルウェー オスロ	140 名
2004	debconf 4	ブラジル ポルトアレグレ	150 名
2005	debconf 5	フィンランド ヘルシンキ	200 名
2006	debconf 6	メキシコ オアスタペック	300 名
2007	debconf 7	英国スコットランド エジンバラ	約 400 名
2008	debconf 8	アルゼンチン マルデルプラタ	約 200 名
2009	debconf 9	スペイン エクストラマドゥーラ	約 250 名
2010	debconf 10	アメリカ ニューヨーク	約 400 名
2011	debconf 11	ボスニア バニャルカ	約 450 名
2012	debconf 12	ニカラグア マナグア	約 180 名

4.2 ニカラグア/マナグア

4.2.1 行き方

日本からの行く場合、米国のマイアミかヒューストンからの入国が一番楽なようです。他の国からは、コスタリカのサンホセやパナマ、アトランタから入国できます。開催施設へはマナグアの空港から約２０分ほどタクシーに乗る必要があります。バスなどもありますが、地元の人の利用者が多いのと治安が悪いためあまり使わないほうがよいとのこと。今回ホテルのタクシーに送迎してもらいましたが、地元のタクシーの場合は 10 ドルほどの料金がかかります。

4.2.2 会場

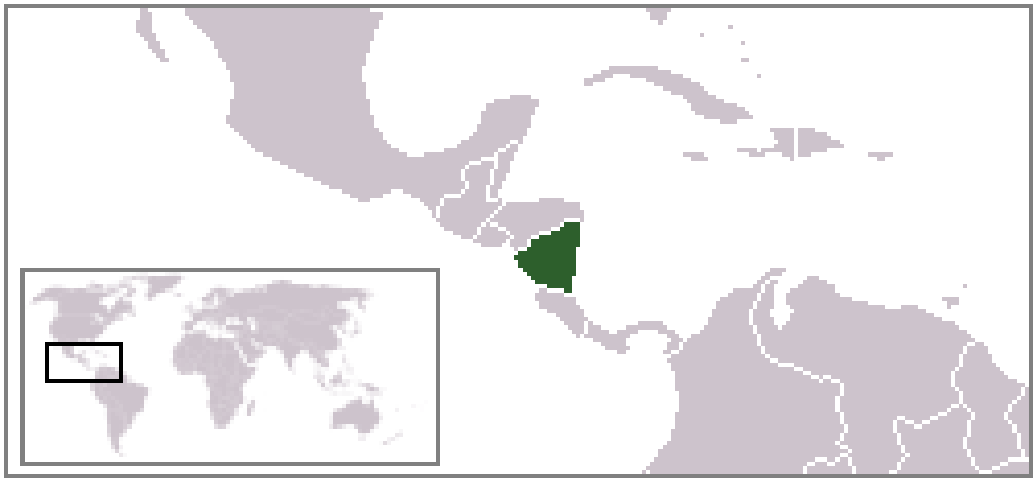


図 1　ニカラグア/マナグアの位置

2012 年度の Debconf の会場はマナグアにある大学「 UCA（ Universidad Centroamericana」の施設の一部を借りて行われました。宿泊は会場から 5 分ほど歩いた場所にあるホテル「 Hotel Seminole」で行いました。以下に会場を紹介します。

- Aula Magna Auditorium: メイン用。 400 人ほど入ることができます。

- Roberto Teran Auditorium: サブ会場。 50 人ほど入ることができます。

- Hacklab: Hacklab はハッ ク専用の部屋です。


- 食堂: 食堂は外にテントを張って、そこで食事しました。


- 宿泊したホテル:

4.3 スケジュール

7 日の Debian Day で Debian Conference は開幕し、 14 日まで毎日いろいろな予定が組まれていました。 11 日だけはカンファレンス参加者で Day-trip を実施しました。

4.4 主となった討論

今回の Debconf は議論テーマを 2 日目は組み込み、 4 日目はバーチャライゼーションなど、日によって分けられる方式が取られました。また、地元からの参加者もわかるようにスペイン語での発表も多くありました。個人的に今回の Debconf はあまり興味がある話題がなく、面白そうな話題があまりなかったように感じました。その中でも私が参加した会議の内容からいくつか取り上げて紹介します。

4.4.1 Bits from the Release Team

リリースチームによる次期リリースに関する発表が行われました。今回のリリースから kibi（ Cyril Brulebois）がリリースアシスタントに加わることが発表されました。また、 Debconf 開催前にフリーズが行われ、次のリリース内容についても発表がされました。主要なソフトウェアを挙げると、 KDE は 4.8、 GNOME は 3.4、 Linux kernel は 3.2 とな

ります。 RC バグ（ 当時で 603 個）が多く残っているため、バグつぶしに協力して欲しいと報告が行われました。

4.4.2 EFI BOF

EFI（ UEFI）についての BOF が行われました。 EFI の Secure boot に対応するためにはどのようにしたらいいのか、他のディストリビューションの対応と Debian での対応方法について情報交換が行われました。 RedHat/Fedora はブートローダと全てのカーネルスタックに署名する方法で対応しているようです。また Canonical/Ubuntu は efilinux のみに署名し対応する方法を取っているとのこと。 GPLv3 の問題もあり Grub2 での対応は難しそうです。今回の BOF では良い案は出て来ませんでした。

4.4.3 Multiarch crossbuilding

Wookey による multiarch がサポートされた環境でのクロスビルドパッケージの状況と問題点について発表が行われました。現在 rebuildd と sbuild を使った クロスビルドファームを立ち上げ、クロスビルドテストを行なっていますが、まだいくつかの課題があるようです。例えば multiarch の foreign に対するクロスコンパイルに必要なパッケージの指定方法などに課題があります。これを解決するためにパッケージのメタ情報として、 cross-build-dep を指定しこれを apt で処理できるようにする機構を追加する（ #666772）といった方法があります。まだいくつか問題が残っているので協力して欲しいとのことでした。

4.4.4 Build Debian with another compiler

Sylvestre Ledru による LLVM を使った Debian パッケージの全ビルドについての発表。現在、 Debian のデフォルトの C コンパイラは GCC ですが、 LLVM/Clang を使ってみた結果と今後の予定について発表されました。 LLVM を使う理由として、ビルド速度が少しだけ速いということや、 GCC よりより賢いコードパース機能が挙げられます。また GCC ではない高性能なコンパイラを選択できるということも良い点です。 Sylvestre は実験 Debconf 前に行われた LLVM 3.0 と 3.1 の結果を元に LLVM の利点を説明しました。 2012 年の 2 月の時点で LLVM 3.0 では、 15658 パッケージ中 1381 パッケージ (全体の 8.8 %) がビルドエラーになっているとのこと。ほとんどのパッケージがビルドできているようです。また 2012 年の 6 月に行った LLVM 3.1 では 17710 パッケージ中 2137 パッケージ (全体の 12.1 %) がビルドに失敗しているようです。この違いが出た理由は一部のオプションがサポートされていなかったり、 LLVM 3.0 では出なかったプログラムのバグ（ プログラムミス）が検知できるようになったためです。結果を `http://clang.debian.net/` から参照できます。発表された時点では、まだ GCC と切り替えできる機構が整備されていないため、まだコンパイラ単体じゃないと利用できませんでした。今後は sbuild や dpkg などのビルドツールに LLVM サポートを入れるように活動するとのこと。商用コンパイラである Intel コンパイラを使うためのツールやビルドも行なっているようなので、気になる方は Sylvestre に連絡をとってみてください。

4.4.5 ARM port(s) update

ARM チームによる ARM ポーティングの状況について報告が行われました。現在 armel と armhf の 2 つのアーキテクチャがあり前者は古い ARM SoC（ v4t）を、後者は比較的新しい ARM SoC（ v7） をサポートしています。 armel はまだユーザもいるので今後もサポートしていく予定です。 armhf は次のリリースに含まれる予定ですが、動作させているマシンのメモリが 1GB しかないため、いくつかのソフトウェアがビルドできない等の問題が起きているようです。この問題を解決するために ARM Server 向けのマシンをどこかから（ HP と交渉しているようだけど）借りて、 Buildd を置き換える予定みたいです。この話の背景として、今後はモバイル端末だけでなく Server でも ARM が使われるようになるため、それらを Debian でサポートするためのことも考えているとのこと。その他、 Raspbian や他のディストリビューションのサポート状況などについても情報交換されました。

4.4.6 AArch64 planning

ARM 64bit サポートアーキテクチャである AArch64 サポートの報告が行われました。 AArch64 は ARMv7 互換の 64bit CPU ARMv8 をサポートするための Linux アーキテクチャ名です（ LKML でこの名前はどうなの? と議論され

ていましたが）。まだ CPU はまだ開発中で来年の頭に市場に投入されると言われています。これを次のリリースである Jessie のサポートアーキテクチャにするために ARM チームは活動していると報告しました。現在 Linux カーネル、ツールチェイン、 dpkg、 autoconf のサポートは完了しており、 Qemu ベースで動作しているとのこと。実際にそのデモも行われました。

4.4.7 FreedomBox Update

Bdale Garbee による FreedomBox の報告が行われました。 FreedomBox は Dreamplug などの ARM Soc を使ったプラグ型サーバの Linux ディストリビューションです。プライベートデータを容易に扱うことのできるサーバを構築できることを目的としています。今回、 FreedomBox を開発するための団体である FreedomBox Foundation の紹介と FreedomBox の成果を Debian にマージ完了の報告が行われました。現在 Dreamplug しかサポートしていませんが、今後他のマシンの対応もするとのことです。

4.5 Debconf14

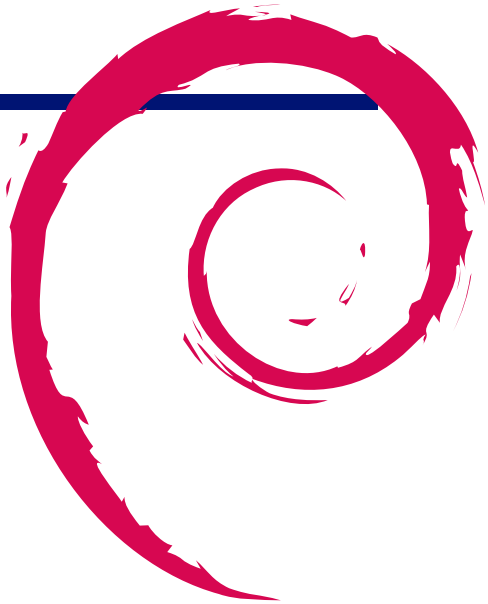
Debconf14 はマルティニーク（ カリブ海の島。フランス領）、カナダのモントリオール、ベネズエラのプエルトラクルスが立候補しました。マルティニークは飛行機でいくのが難しそうな国なのと十分なネットワーク環境がないのでいまのところ難しそうです。モントリオールは設備がまだ不十分な感じです。ベネズエラは既に国内のスポンサーも獲得しており、ホテルもいくつか抑えてるとのこと。ネットワーク以外は網羅していると思いました。

4.6 Daytrip

Debconf では一日、参加者で旅行をするというイベントがあります。今回の Debconf では レオン旧市街観光ツアー、火山観光ツアー、ジュアンベアノ島観光ツアーのチームに別れ、 Daytrip をしました。矢吹以外の日本からの参加者は全員火山観光ツアーに登録しましたが、先日の大雨の影響で火山ツアーは行く事ができず、海に行ったあとにレオン市街観光を行いました。

4.7 次回の Debconf

次回の Debconf13 はスイスのヴァーマルキュで開催される予定です。日程は 8 月 5 日から 18 日です。キャンプ場を借りきってやるようですが、周りに何もないように見えます。ホテルという施設でもないのでシェラフ等を持っていく必要がありそうです。普段とは異なったサバイバルになると思います。



5 月刊 Debhelper

野島 貴英

5.1 今回のコマンド

debhelper のマニュアルを翻訳した際、いつか解析しようと思っていた以下のコマンド 2 つを今回は取り上げます。

- dh_makeshlibs
- dh_shlibdeps

これらの debhelper コマンドは、共有ライブラリとの依存関係を deb パッケージに含める際に大変役立つコマンドとなります。

5.2 予備知識

今回、共有ライブラリの知識、共有ライブラリのパッケージ作成に関する予備知識、 deb パッケージの構造について、知識がある程度ある人向けに書いています。基本的な事項は文献 [1, 2, 3, 4, 5] を参照すると参考になります。

5.3 呼び出し方と動作

以下のように順に呼び出して利用します。

```
$ dh_makeshlibs
$ dh_shlibdeps
```

5.3.1 dh_makeshlibs

本コマンドの目的は、共有ライブラリのパッケージに特化した制御ファイルを生成するのが目的となります。
このコマンドは呼び出されると以下の動作を行います。

Step 1. パッケージ構築ディレクトリで作成した共有ライブラリを走査し、発見した共有ライブラリの SONAME に記載されているバージョン番号と、ライブラリ名を、 objdump を使ってかき集めます。

Step 2. 得た情報を元に、パッケージ構築ディレクトリ以下の DEBIAN/shlibs ファイルを作成します。

Step 3. パッケージ構築ディレクトリ以下の DEBIAN/postinst,DEBIAN/postrm を生成します。

Step 4. dpkg-gensymbols コマンドを呼び出し、ソースパッケージにメンテナが含めた symbol ファイルを参考にしながら、更新した symbol ファイルを、パッケージ構築ディレクトリ以下の DEBIAN/symbol に作成します。

5.3.2 dh_shlibdeps

本コマンドの目的は、共有ライブラリの依存関係を算出して、後に制御ファイルとしての control ファイルを生成する時に置換情報として使われる”debian/パッケージ名.substvars” ファイル(以下 substvars ファイル) を更新します。

このコマンドは呼び出されると以下の動作を行います。

Step 1. パッケージ構築ディレクトリで作成した、実行可能ファイル、共有ライブラリを探します。

Step 2. 見つけたファイルを dpkg-shlibdeps へ引き渡し、 substvars ファイルを更新します。

5.4 ${shlibs:Depends}、 ${shlibs:Recommends} マクロ

共有ライブラリとの依存関係を人手でメンテナンスしつづけるのは大変面倒な作業になり、また、多くのケースでは機械的に依存関係を求める事も可能なので、パッケージ作成時に機械的に共有ライブラリとの依存関係を生成出来る仕組みがあります。こちらのマクロはこの機能を利用する為のものとなります。 dh_shlibdeps は debian/control ファイルの記述に含まれるこれらマクロを、算出した依存情報で書き換えてくれます。

なお、手元のバイナリについてどんな内容で置き換えられるかは、展開済みのソースパッケージの元で”dpkg-shlibdeps -O バイナリファイル” で検証できます。

```
$ apt-get source gnome-shell
$ cd gnome-shell-3.0.2
$ dpkg-shlibdeps -O /usr/bin/gnome-shell
...SONAME の形式にバージョン情報が無いものがある、まったく参照されていないライブラリがある等の警告が出る...
shlibs:Depends=gnome-bluetooth (>= 3.0.0), libatk1.0-0 (>= 1.12.4),
libc6 (>= 2.2.5), libcairo-gobject2 (>= 1.10.0), libcairo2 (>= 1.2.4),
libcanberra0 (>= 0.2), libclutter-1.0-0 (>= 1.10.0), libcogl-pango0
(>= 1.7.4), libcogl9 (>= 1.7.4), libcroco3 (>= 0.6.2), libdbus-1-3 (>=
1.0.2), libdbus-glib-1-2 (>= 0.78), libffi5 (>= 3.0.4), libfolks25 (>=
0.6.0), libgck-1-0 (>= 2.91.1), libgcr-3-1 (>= 2.91.1),
libgdk-pixbuf2.0-0 (>= 2.22.0), libgee2 (>= 0.5.0),
libgirepository-1.0-1 (>= 0.9.2), libgjs0-libmozjs185-1.0, libgjs0b
(>= 1.32.0), libgl1-mesa-glx | libgl1, libglib2.0-0 (>= 2.26.0),
libgnome-keyring0 (>= 2.20.3), libgnome-menu-3-0 (>= 3.2.0.1),
libgstreamer0.10-0 (>= 0.10.16), libgtk-3-0 (>= 3.0.0),
libjson-glib-1.0-0 (>= 0.12.0), libmozjs185-1.0 (>= 1.8.5-1.0.0+dfsg),
libmutter0 (>= 3.4), libmutter0 (<< 3.5), libnm-glib4 (>= 0.7.999),
libnm-util2 (>= 0.7.0), libnspr4 (>= 2:4.9-2~) | libnspr4-0d (>=
1.8.0.10), libp11-kit0 (>= 0.2), libpango1.0-0 (>= 1.14.0),
libpolkit-agent-1-0 (>= 0.94), libpolkit-gobject-1-0 (>= 0.94),
libpulse-mainloop-glib0 (>= 0.99.1), libpulse0 (>= 0.99.1),
libsoup2.4-1 (>= 2.4.0), libstartup-notification0 (>= 0.2),
libtelepathy-glib0 (>= 0.13.12), libtelepathy-logger2 (>= 0.2.0),
libx11-6, libxcomposite1 (>= 1:0.3-1), libxdamage1 (>= 1:1.1),
libxext6, libxfixes3, libxi6, libxml2 (>= 2.6.27)
```

5.5 shlibs ファイルと、 symbol ファイル

dh_makeshlibs コマンドと、 dh_shlibdeps コマンドで重要な役割を持つファイル２つについて説明します。

5.5.1 shlibs ファイル

ファイル形式:
```
[<type>: ]<library-name> <soname-version> <dependencies ...>
```

実例:
```
$ cat /var/lib/dpkg/info/libgtk2.0-0:amd64.shlibs
libgtk-x11-2.0 0 libgtk2.0-0 (>= 2.24.0)
libgdk-x11-2.0 0 libgtk2.0-0 (>= 2.24.0)
udeb: libgtk-x11-2.0 0 libgtk2.0-0-udeb (>= 2.24.0)
udeb: libgdk-x11-2.0 0 libgtk2.0-0-udeb (>= 2.24.0)
```

ライブラリのファイル名 (library-name) と、 SONAME のバージョンの情報 (soname-version) と、パッケージの依存情報として記載すべき内容 (dependencies) を記したファイルとなります。

実例でいくと、「 libgtk-x11-2.0 は実際のライブラリの名前、 SONAME のバージョン番号は 0、パッケージの依存情報として記載すべき内容としては、”libgtk2.0-0 (>= 2.24.0)” と記載せよ」という意味になります。この場合、libgtk-x11-2.0.so.0 をリンクしているバイナリ(“objdump -p バイナリ” で、 NEEDED という文言で判定できます)

のパッケージ依存情報としては、”libgtk2.0-0 (>= 2.24.0)” を記載しなさいという意味になります。

dh_makeshlibs コマンドで-V オプションを使って明示的に指定しない限り、 dependencies の部分はパッケージのバージョンで生成される為、最も単純で、最も保守的な共有ライブラリへの依存関係情報となっているという特徴があります。

shlibs ファイルを使った依存関係算出の方法は最も基本的な方法ですので、 BUG#634192、#571776 で debian-policy へ掲載すべきとの要望があり、対応がなされた模様です。こちらについては、

```
$ git clone 'http://anonscm.debian.org/git/dbnpolicy/policy.git'
```

すれば shlibs ファイルに関する記載が大幅追記された debian policy が読めます。

5.5.2 symbol ファイル

共有ライブラリの提供している全シンボルと、各シンボルについて搭載されたバージョンを記載したファイルを用意すると、 shlibs ファイルを使うよりも、もっと柔軟で必要最小限の依存関係を算出出来るようになります。こちらを実現しようとするものが、 symbol ファイルとなります。

ファイル形式:

```
<soname> <main dependency template>
[| <alternative dependency template>]
[ as many alternative dependency templates as needed ]
    <symbol> <first-version>[ <id of dependency template>]
    [ as many symbols as needed ]
```

実例:

```
$ cat /var/lib/dpkg/info/libapr1.symbols
libapr-1.so.0 libapr1 #MINVER#
 apr__SHA256_Data@Base 1.2.7
 apr__SHA256_End@Base 1.2.7
 apr__SHA256_Final@Base 1.2.7
 ... 中略...
 apr_global_mutex_lock@Base 1.2.7
 apr_global_mutex_lockfile@Base 1.4.2
 apr_global_mutex_name@Base 1.4.2
 apr_global_mutex_pool_get@Base 1.2.7
 ... 中略...
```

こちらの symbol ファイルの実例ですと、 SONAME として、 libapr-1.so.0、パッケージの依存情報として記載すべき内容は “`libapr1 #MINVER#`” となります。なお、`#MINVER#`は、 dpkg-shlibdeps によって、シンボルが過不足なく見つかる時のバージョンによって “`( >= X.X.X)`” の形式で置き換えられます。

共有ライブラリの依存関係を洗い出す際、 SONAME の情報が変わらない限りは、バイナリが必要としているシンボルのみを最小限搭載しているライブラリと、シンボルがすべて見つかる最古のバージョンのみを依存関係として含めた方が、パッケージの利用にあたって都合の良い事が多いです。ここで、

- バージョンの上がったライブラリの元でビルドして依存関係を求めたが、実は古いバージョンのライブラリでも全く問題なく動的リンク出来る場合は古いバージョンのライブラリも含んで依存対象としたい。
- 依存しているライブラリ A がライブラリ B を必要としているが、バイナリからはライブラリ B のシンボルを一切利用していないので、バイナリ本体の依存関係としてライブラリ B を外したい

という事を検討します。もし、こちらが出来ると、例えば sid から testing へバイナリパッケージを移動させるときに、依存関係にある数多くのライブラリ全部も同時に sid から testing へ完全に移動できるようになるまで、バイナリ本体を testing のものとして移動出来ないという問題を極力避ける事ができます。 [6] (図 2 参照)

5.6 substvars ファイル

${shlibs:Depends}、${shlibs:Recommends} マクロなどの control ファイルに含まれる情報を置き換える際に debhelper 共通で使われる中間ファイルとなります。各 debhelper コマンドが substvars ファイルに置き換えるべき内容を追記を繰り返すことにより、 substvars ファイルは更新され、最後に dh_gencontrol がマクロ部分を substvars を参照しながら、適宜置き換えた制御ファイルとしての control ファイルを生成します。

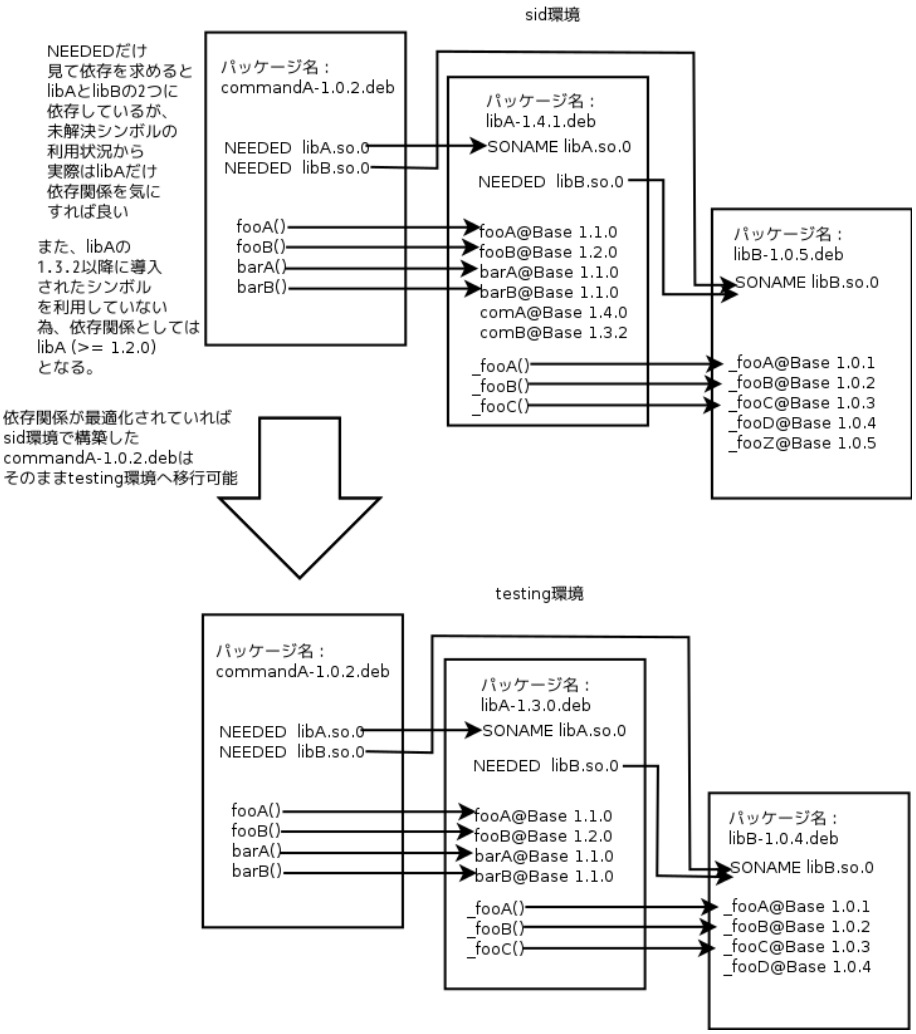


図 2　symbol ファイルを利用した依存関係算出

5.7　内部で主に利用されている dpkg パッケージに含まれるコマンド

今回の debhelper コマンドは内部で dpkg パッケージに含まれるいくつかのコマンドを利用して処理を行っています。

5.7.1　dpkg-gensymbols

dh_makeshlibs 内部にて、ライブラリメンテナがソースに含めている symbol ファイルを元にし、実際に構築したライブラリから得られるシンボルの差分を含めた symbol ファイルを生成するのに使われます。この生成された symbol ファイルはライブラリパッケージの control ファイル群として含められます。

dh_makeshlibs の行うデフォルトの呼び出しでは、 dpkg-gensymbols はソースに含めている symbol ファイルからシンボルが廃止されたものがあることを発見すると、 DEBIAN/symbol ファイルを一旦生成するものの、エラーステータスを返却して終了してしまいます。この時、標準出力に違いを diff 形式で示します。もし、このような事が発生した場合は、メンテナはこの diff を見ながら、 symbol ファイルのメンテナンスを行う事になります。また、生成された symbol ファイルをメンテナは保管する事により、次回のライブラリのバージョンアップに備える事になります。

5.7.2 dpkg-shlibdeps

dh_shlibdeps はほとんど dpkg-shlibdeps のラッパーコマンドとも言えるぐらい、共有ライブラリの依存関係算出にあたって、 dpkg-shlibdeps をそのまま利用しています。

dpkg-shlibdeps の動作は以下のとおりです。

Step 1. コマンドラインから与えられたバイナリファイルが必要とする共有ライブラリへの動的リンクに必要な情報を

```
objdump -w -f -p -T -R  バイナリファイル
```

コマンドを使って探ります。

Step 2. 得られた情報のうち、 NEEDED に記載されている SONAME と同じファイル名を持つライブラリファイルを /lib,/usr/lib 以下から探し当てます。

Step 3. 発見したライブラリファイルのフルパスを元に dpkg -S を利用してライブラリのパッケージ名を割り出します。

Step 4. “dpkg-query --control-path ライブラリパッケージ名” を使って/var/lib/dpkg/info/以下にあるライブラリパッケージの提供する shlibs ファイル/ symbol ファイルを取得します。

Step 5. Step 1. で得たバイナリファイルの動的リンクに必要なシンボル名と、 Step 4. で得た shlibs ファイル/symbol ファイルを検証することによりバイナリファイルにとって最適な依存関係を算出します。

Step 6. substvars ファイルに算出した依存情報を記載します。

参考: Step 5. の動作をどのように行っているかを観察するには、

```
$ dpkg-shlibdeps -v -v -v -O バイナリファイル
```

をバイナリファイルに対応するソースパッケージ展開後のディレクトリで、実行するとよく判ります。

5.7.3 おわりに

共有ライブラリの依存関係の対応すら 2 コマンドで担当出来る debhelper コマンドと、その心臓部を担う dpkg パッケージに含まれるコマンド群はよくできていると思いました。

参考文献

[1] John R.Levine, 榊原 一矢/ポジティブエッジ 訳, “Linkers & Loaders”, ISBN10 4274064379

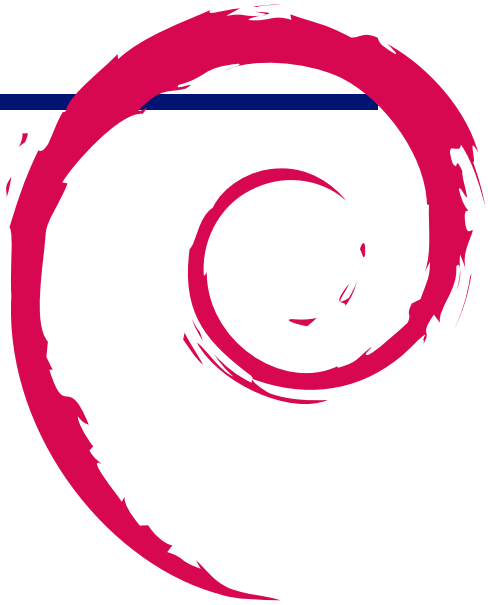
[2] 坂井 弘亮, “リンカ・ローダ実践開発テクニック”, ISBN10 4789838072

[3] 高林 哲ら, “Binary Hacks”, ISBN10 4873112885

[4] Junichi Uekawa, “Debian Library Packaging guide”,http://www.netfort.gr.jp/~dancer/column/libpkg-guide/libpkg-guide.html

[5] man 5 deb もしくは wikipedia の deb(ファイルフォーマット),http://ja.wikipedia.org/wiki/Deb_(%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB%E3%83%95%E3%82%A9%E3%83%BC%E3%83%9E%E3%83%83%E3%83%88)

[6] Hertzog, “Improved DpkgShlibdeps”, http://wiki.debian.org/Projects/ImprovedDpkgShlibdeps



6 ソフト開発以外の簡単 Debian contribution(ドラフト版!)

野島 貴英

6.1 はじめに

以前 debconf11 に参加した時、 debconf の最後に行われるライトニングトークで、「 ソフト開発以外に Debian へ貢献するにはこんなのがあるよ! 」という発表がありました。この発表を聞いて以降、このあたりを一度まとめてみたいなーと思っていました。

ここでは、 Debian をインストールしてちょっと慣れたぐらいの人がちょこちょこっと Debian へ貢献する手段についてまとめてみます。きっと漏れもあると思うので、東京エリア Debian 勉強会にて BOF 形式で補完してみます。

6.2 ソフト開発以外の簡単 contribution 作業一覧 (ドラフト版!)

- DDTSS の日本語訳をレビューしてみる、日本語に訳してみる。
  (詳しくは「 第 53 回東京エリア Debian 勉強会資料」 `http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume200906.pdf` を参照ください。非常に良い解説とチュートリアルとなっています。)
- BTS へ BUG 報告/何かの改善提案をしてみる。
  (詳しくは「 第 43 回 関西 Debian 勉強会資料」 `http://wiki.debian.org/KansaiDebianMeeting/20110123` を参照ください。非常に良い解説とチュートリアルとなっています。)
- `http://debtags.debian.net/edit/` で debtags をひたすら打ち込む。
  (詳しくは「 第 63 回東京エリア Debian 勉強会資料」 `http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume201004.pdf` を参照ください。非常に良い解説とチュートリアルとなっています。)
- `http://screenshots.debian.net/` へスクリーンショットを投稿しまくってみる。
- `http://www.debianart.org/cchost/` へオリジナルの絵を送ってみる。
- `debian-doc@debian.or.jp` にて投稿された翻訳のレビューをしてみる、未だ日本語に訳されていない文章を翻訳して投稿してみる。
- `debian-users@debian.or.jp` にて投稿された質問に回答をしてみる。

6.3 screenshots.debian.net

パッケージの動作のスクリーンショットを集めているサイトです。 synaptic パッケージマネージャ (aptitude install synaptic で導入できます) にはプログラムの紹介の他に、プログラムが動作しているときのスクリーンショットを見る事ができます。このスクリーンショットのダウロード先がこの screenshots.debian.net となります。

初めての方は以下のようにしてスクリーンショットを投稿できます。アカウント登録すら不要のお手軽サイトなので、

面倒なユーザ登録も必要ありません。こちらに投稿され、無事審査をパスしたスクリーンショットは、世界中の Debian ユーザが使っている synaptic パッケージマネージャで即閲覧できるようになります。

以下にスクリーンショットの投稿の仕方を記載します。

Step 1. `http://screenshots.debian.net/packages/without_screenshots?search=&debtag=` をアクセスし、スクリーンショットが無いパッケージから適当なものを選びます。

Step 2. 選んだパッケージを手元の Debian マシンへ導入します。

Step 3. export LANG=C を実行して、表示されるメッセージができるだけ英語になるようにしてプログラムを起動出来るようにします。

Step 4. 導入したプログラムを起動します。

Step 5. プログラムのスクリーンショットを撮ります。 GNOME 環境であれば、 ALT+Print Screen でスクリーンショットを撮る事の出来るアプリケーションが起動します。

Step 6. PNG ファイルで撮ったスクリーンショットの画像を保存し、 ImageMagick などを使って画像サイズを最大でも 800x600 ぐらいにして PNG 形式で保存します。

Step 7. `http://screenshots.debian.net/upload` をアクセスして、パッケージ名 (Package name:)、パッケージのバージョン (Software version:)、どこの画面か等の説明 (Screenshot description:) のフォームを半角英数字で埋め、 Scrrenshot の Choose File を選んで、先ほど保存したスクリーンショット画像を選択します。

Step 8. 最後に [Upload screenshot] というボタンをクリックすると早速投稿が完了します。この時点では、サイト管理チームの審査が終ってないので、 1 日ぐらい待ちます。

Step 9. 無事審査がパスされると、 `http://screenshots.debian.net/` の左上あたりの”Latest uploads...” という部分に、投稿したスクリーンショットが掲載されます。この時点で、 synaptic パッケージマネージャからもスクリーンショットが閲覧できるようになります。

やってみると判りますが、非常に簡単です。パッケージの魅力を伝えるのに、スクリーンショットの影響は少なくないと自分も考えています(特にゲームのパッケージ）。ぜひこの機会にスクリーンショットをあげまくってみてください。

なお、注意事項として、撮影するスクリーンショットの元となるデータのライセンスには十分気をつけてください。これは大方のパッケージでは意識しなくても全く問題ないのですが、ゲーム等のパッケージのうちゲーム機/元々有料ソフトのエミュレータ環境等の、エミュレータのスクリーンショットでは、表示に使ったデータ次第では十分にライセンスが問題になり得ます。このあたりがちょっとでも心配な方は、ゲームのエミュレータ環境のスクリーンショットを投稿するのは、ライセンスに自信のある方以外は止めておいた方がよいでしょう。

6.4 debianart.net

Debian に使われる様々なグラフィック/サウンドデータの投稿サイトとなります。直近では、 wheezy の起動画面の背景画像、 GNOME 環境のデフォルトの背景画像などのコンテストが行われました。

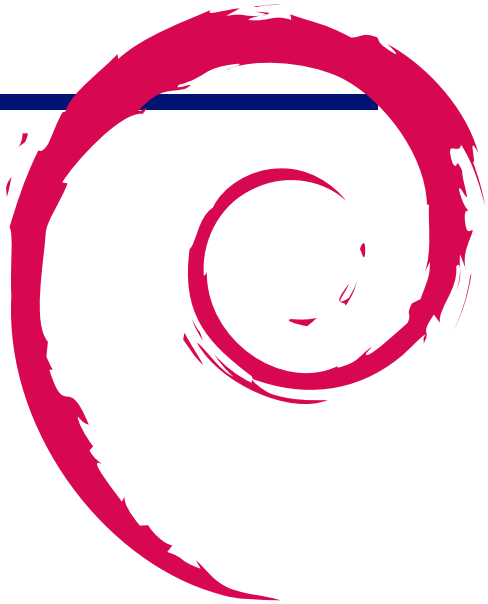
最近では、 Debian プロジェクトにマスコットキャラクタが必要ということで、マスコットキャラクタの図案の募集が行われていたようです。

絵心のある方/写真撮影に興味のある方/サウンドアイコンに興味ある方で、我こそはという方はぜひ活用すると良いかもしれません。

なお、 Debian のプロジェクトである以上、投稿した絵/写真/音楽はいわゆるフリーなライセンスで扱われる事が出来るものに限ります。こちらはご注意ください。

6.5 おわりに

Debian は多様性を重視するプロジェクトです。ソフト開発以外の貢献は大変重要で、協力者は日夜必要な状態です。やってみようという方は是非ご協力お願いします。



7 Debian での C++11 開発環境

上川純一

7.1 はじめに

2011 年にながらく c++0x として知られていた仕様が C++11 仕様 [1] として策定され C++ 言語機能が拡充されました。そろそろ C++11 を使ったコードでも書いてみようかと思っている人もいるかもしれません。仕様が出てきてから一年以上たってもまだ仕様に完全に準拠したコンパイラというのは出てきてないようですが、 Debian に含まれている C++ のコンパイラでもほとんどの機能は実装されてきています。 Debian で C++11 を使ったコードを書くときにどういうパッケージをインストールしておくと便利なのか紹介します。

7.2 サンプルコード

とりあえずは C++11 の機能を利用したサンプルコードを書いてみます。 auto と for ループを使ったものを試してみましょう。

```
#include <iostream>
using namespace std;

int main(int argc, char **argv) {
  int hoge[] = { 1, 10, 12, 15};
  const char* fuga[] = { "hello", "world", "this is a", "message"};
  for (const auto& i : hoge) {
    cout << i << endl;
  }

  for (const auto& str : fuga) {
    cout << str << endl;
  }
  return 0;
}
```

7.3 コンパイラー

Debian で C++11 準拠のソースコードをコンパイルできるコンパイラは GCC の G++ と LLVM の clang++ です。他にもあるかもしれませんが試していません。両方共すべての言語機能・ ライブラリ関数は実装していません [3, 4]。実装していないので気になったところとしては `regex`, `attribute`, `this_thead::sleep_for` などがあるようです。とりあえずまずはインストールから。

```
# apt-get install g++ clang
```

7.3.1 GCC

g++ でコンパイルする場合は、–std=c++11 を指定してあげると C++11 モードでコンパイルします。

```
$ g++ --std=c++11 auto.cc
$ ./a.out
```

7.3.2 clang

Clang も同様で、 C++11 モードを指定してあげる必要があります。

```
$ clang++ --std=c++11 auto.cc
$ ./a.out
```

7.4 ドキュメント

7.4.1 C++11 標準ライブラリ

GCC も Clang も両方共 libstdc++ をライブラリとして利用しているようです。 libstdc++ のドキュメントは `libstdc++6-4.7-doc` パッケージに man page 形式で提供されています。 std namespace にあるものはだいたい説明があるでしょう。ただ、 doxygen で自動生成されているからなのか若干クセがあります。:: が _ に変換されていて、 `std::hogehoge` は `std_hogehoge` という名前で登録されています。他に混乱する例を挙げると、 std::string は `std_string` ではなく `std_basic_string` にドキュメントがあるのも最初はよくわかりませんでした。

7.4.2 STL

stl-manual パッケージに HTML 形式で STL のドキュメントが入っています。

`/usr/share/doc/stl-manual/html/index.html`

7.4.3 GCC 独自拡張についてのドキュメント

gcc-doc パッケージに GCC でコマンドと言語仕様関連の Info ファイルがあります。

7.4.4 その他のドキュメント

その他これがあればいいだろうというドキュメントを紹介します。

C++ でよく使うライブラリ、 Boost のドキュメントは libboost1.49-doc パッケージに HTML 形式ではいっています。

manpages-dev パッケージに Linux の API 関連のドキュメントがあります。

7.5 最後に

ひと通り C++11 を利用するにあたって便利そうなものを紹介しました。もっと良い物があるよというのがあればぜひ教えてください。

参考文献

[1] “Working Draft, Standard for Programming Language C++”, 2011, `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3242.pdf`

[2] Bjarne Stroustrup, C++ FAQ `http://www.stroustrup.com/C++11FAQ.html`

[3] C++0x/C++11 Support in GCC, `http://gcc.gnu.org/projects/cxx0x.html`

[4] C++98 and C++11 Support in Clang, `http://clang.llvm.org/cxx_status.html`

Debian 勉強会資料

2012 年 8 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）