Grand Unified Debian

銀河系唯一のDebian専門誌

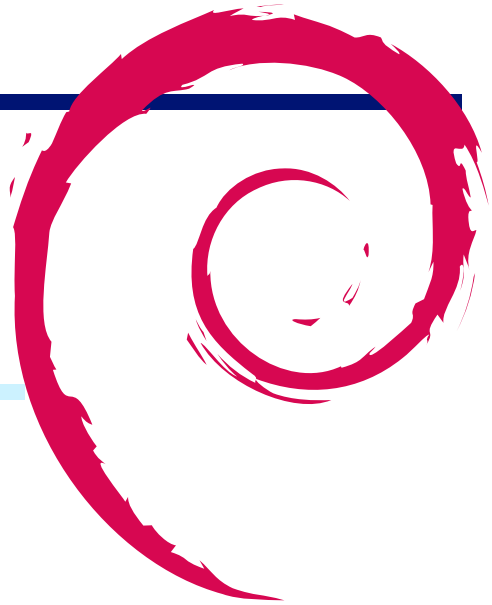
大統一Ｄｅｂｉａｎ勉強会 特大号

東京エリア/関西Ｄｅｂｉａｎ勉強会

あんどきゅめんてっど でびあん 2012 年夏号 2012 年 06 月 23 日 初版発行

デビアン勉強会

目次



1 Introduction

上川 純一, 山下 尊也

1.1 東京エリア Debian 勉強会

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。

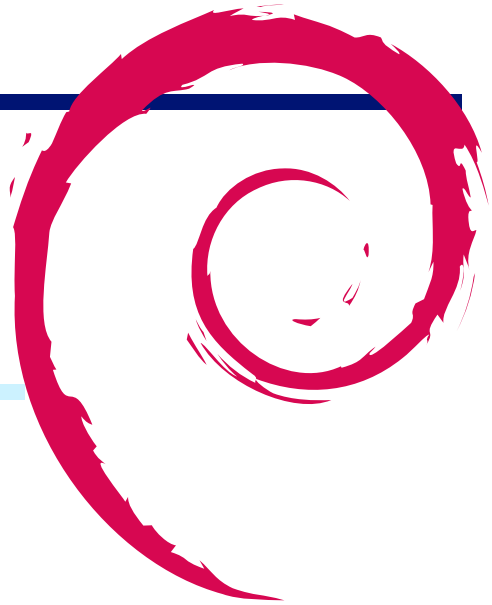
1.2 関西 Debian 勉強会

関西 Debian 勉強会は Debian GNU/Linux のさまざまなトピック (新しいパッケージ、 Debian 特有の機能の仕組、Debian 界隈で起こった出来事、などなど) について話し合う会です。

目的として次の三つを考えています。

- メーリングリストや掲示板ではなく、直接顔を合わせる事での情報交換の促進
- 定期的に集まれる場所
- 資料の作成

それでは、楽しい一時をお楽しみ下さい。



2 TeXLive2011(2012/dev) in Debian

佐々木洋平

2.1 はじめに

発表要旨を「Debian における TEX 環境 (特に日本語処理) の導入や設定および次期安定版 Wheezy での変更点および開発状況について解説します。」としたので、その順番で解説してみます。ちなみに佐々木はあくまで TEX(pLATEX) のユーザであって、開発に関してはあまり関与しておりません。また「TEX でナベアツ」とか「TEX で数値計算」とかそういう話はしませんので、あらかじめご了承下さい。

2.2 TEX 環境 (特に日本語処理) って?

先ずは「TEX 環境、特に日本語処理」について簡単に触れておきます。TEX は Donald E. Knuth さんによって作成された組版 (くみはん) システムであり、現在のバージョン番号は 3.1415926 です[*1]。通常 TEX を使用してなんらかの文書を記述する際に素の TEX を使うことはあまり無く、多くの場合、Leslie Lamport さんによる TEX の拡張である LATEX、もしくは LATEX を (株) アスキー (現:アスキーメディアワークス) が日本語化した pTEX,pLATEX が使われます[*2]。ちなみに TEX を日本語化したものには NTT jTEX, jLATEX やこれらの多言語版である MuLTEX も存在します。

pTEX,pLATEX や jTEX, jLATEX によって行なわれた「日本語の処理」とはどういったものなのかを簡単にまとめてみましょう。一言で済ますのであれば「複数バイト文字コードの処理」と「日本語の組版への対応」です。

2.2.1 TEX における日本語処理

先ず「複数バイト文字コードの処理に」ついて。pTEX では JIS X 0208 を文字集合とした (ISO-2022-JP, EUC-JP, Shift_JIS) を直接扱えるように拡張がおこなわれました。この結果として、pLATEX 文書を typeset した結果として出力される DVI(DeVice Independent format) ファイルには 16bit 以上の文字コードが格納される事になり、pTEX の出力した DVI ファイルを処理するためのソフトウェアとして `xdvik-ja` (DVI ファイルの X 上での表示) や `dvipsk-ja` (DVI ファイルの PostScript への変換) なども提供されています[*3]。一方 jTEX では、複数の文字コードをそれぞれ 1byte 文字セットに分割して取り扱うことで日本語 (だけではなく多言語) を処理できるような拡張が行なわれました。結果として NTT jTEX はオリジナルの TEX からの変更点が小さい、という利点があります。

次に「日本語の組版への対応」について。日本語の組版は「JIS X 4051(日本語文書の組版方法)」として、禁則処理やルビ、縦書きや横書きの扱いがかなり細く規定されています[*4]。組版の結果として得られる文書はこの規格に沿っているこ

[*1] Knuth 先生は 1990 年に TEX の開発終了を宣言しており、今後は変更は行なわれません。バグフィックスなどでバージョン番号を変更する場合には、バージョン番号は π に漸近していきます (開発終了時のバージョンは 3.1 でした)。

[*2] pLATEX の"p" は publishing を意味しています。

[*3] DVI のファイル形式自体には 16bit 以上の文字コードが含まれていても問題無いのですが、欧文の dviware(DVI 処理系) では 16bit 以上が想定されていませんでした。ちなみに、ここでの dviware の名称は Debian 固有の名称であり、実際には配布される形態によって名前が異なっていたりします。

[*4] JIS X 4051 の策定は 1993 年。最新版は 2004 年に改訂された「JIS X 4051:2004」。

とが望ましいのですが、そもそも jTEX は縦書きに対応していませんでした。現在は日本語処理系として pTEX 系列が一般的になっているのはここに理由があるでしょう。また現在では、当初 pTEX,pLATEX で提供されていたスタイルファイルを奥村晴彦さんがさらに改良した新ドキュメントクラスを用いるのが一般的になってきました[*5]。 pLATEX 付属のクラスファイル (jarticle) と奥村さんの新ドキュメントクラス (jsarticle) による組版の例を図 1 に示します。 句読点や括

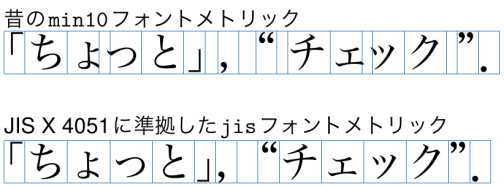


図 1　日本語組版の例。上が pLATEX を用いた場合。下が奥村さんの新ドキュメントクラスを用いた場合 (奥村, 2011)。

弧の処理が細かく調整されている様子が見てとれます。

最後に Unicode 対応について。先ずお断りとして、 TEX における多言語処理についてはここでは割愛します[*6]。「 ファイルの文字コードとして UTF-8 を使いたい」つまり「 多言語混在の UTF-8 のファイルを直接処理したい」という場合にも、最新の pLATEX(正確には ε− 拡張がなされた ε−pTEX) であれば、そのまま処理できます。つまり、次の安定版 (Wheezy) では、 UTF-8 で保存されたソースをそのまま pLATEX で処理できるようになる……はずです (残念ながら squeeze では処理できません)。

2.2.2　世界情勢

ここで一端日本語を離れて、日本語以外の TEX(とその拡張) および多言語処理がどの様に発展しているのか、について簡単に触れてみましょう。 Knuth さんの TEX 開発終了宣言の後でも、最下層の処理系としての TEX は進化を続けており、 ε−TEX、そして pdfTEX へと進展しています。現在では LATEX として pdfLATEX が使われるのが一般的です。pdfTEX はその名の通り DVI ファイルを経ずに直接 PDF ファイルを出力する TEX エンジンです。この結果として、今後は DVI ファイルは過去の遺物になっていくのかもしれません[*7] 。また、 ε−TEX は元々の TEX に対して多くの拡張がなされています。現在既に多くの拡張機能が ε−TEX に依存しているため、 ε−TEX に対応していないエンジンは時代遅れになりかねません[*8]

多言語処理系としては、壮大な試みであった Ω が頓挫 (?) した後、現在では XeTeX(=ε−TEX+ Unicode + OpenType) を用いるのが一般的になってきました。 Unicode での入出力を処理し、システムのフォントを利用して直接 PDF を出力します。また、 pdfTEX の後継として LuaTEX(pdfTEX+ Ω + Lua + METAPOST + OpenType) の開発が進められています。 Lua による柔軟な拡張が可能となっており、今後の進展が楽しみではあります。

なお、現時点では Xe(La)TeX および LuaTeX を用いて、 JIS X 4051 で規定された日本語の組版を実現するためには別パッケージが必要だったり、プリアンブルをそれなりに修正する必要があるので一筋縄ではいきません。今後に備えて動向を確認しておくのも良いでしょう[*9]。

[*5] pLATEX2e 新ドキュメントクラス: http://oku.edu.mie-u.ac.jp/~okumura/jsclasses/

[*6] 興味のある方は以下の URL を参照して下さい
「 pTeX と多言語処理」 http://oku.edu.mie-u.ac.jp/~okumura/texwiki/?pTeX%E3%81%A8%E5%A4%9A%E8%A8%80%E8%AA%9E%E5%87%A6%E7%90%86

[*7] これは日本語処理としてはある意味ありがたい話です。 DVI ファイルにはフォントの「 参照」しか表れないため、表示が環境によって異なり、DeVice Independent が実現されません。 PDF の場合はどうか、といえば、そもそも欧文の PDF の場合にはフォントを埋め込むのが一般的であり、和文のみが参照名 (Ryumin-Light, GothicBBB-Medium) のみで実フォントは埋め込まない、という状況でした。しかしながら、JIS2004 前後で同じコードポイントに別のグリフが割り当てられているため、フォントを埋め込んでいない PDF では表示が異なる、という問題が発生します。今後は欧文和文区別なく、フォントを埋め込んだ PDF を出力する、というのが一般的になるでしょう。

[*8] 幸いな事に北川弘典さんを中心に ε−pTEX(=pTEX+ ε−TeX) が公開されており、 TeXLive 本体にも同梱されています。本原稿執筆時点では既に Debian unstable で日本語環境を導入すると pLATEX のエンジンとして ε−pTEX が使われています。

[*9] XeTeX に関しては「 XeLaTeX で日本語する件について」 http://zrbabbler.sp.land.to/xelatex.html
LuaTeX-ja に関しては「 LuaTeX-ja プロジェクト」 http://sourceforge.jp/projects/luatex-ja/wiki/FrontPage

2.2.3 TeXLive について

最後に、 TeXLive について。 TeX に関するソフトウェアを提供する「ディストリビューション」として、以前は teTeX がリリースされていました。しかしながら、 2009 年に teTeX としてリリースを停止した後[*10]、 TeXLive が TeX のディストリビューションとしてリリースされるようになりました。 TeXLive のリリースマネージャは Norbert Preining さんで、彼は Debian の TeX 関連パッケージのメンテナでもあります。 TeXLive では `tpm2deb` や `tpm2rpm` といったパッケージ変換スクリプトも提供されており、現在では多くの Linux ディストリビュータが TeXLive を元に各々のパッケージを作成しています。日本語関連については、 2010 年以降に TeXLive 本体への pTeX、 jTeX のマージが開始され、本原稿執筆時点では、 patch の著作権者にコンタクトの取れていない `xdvik-ja` 以外はほぼマージが完了しています。

2.3 Debian における (日本語)TeX 環境

さてここまでのお話を踏まえた上で、ようやく Debian の TeX 環境について触れていきます。ちなみに、「 Debian での pLaTeX の設定」みたいな話はぐぐると結構でてくるので、あまり触れません[*11]。

2.3.1 Squeeze における (日本語)TeX 環境

Squeeze にて配布されている TeXLive のバージョンは 2009 です。ですので pTeX および jTeX 関連は、 ε−TeX 拡張前の **UTF-8** のファイルを処理できないバージョンです。具体的には、 `ptex-buildsupport` パッケージで teTeX のソースを提供し、これに patch を当てることで pTeX や jTeX のパッケージが構築されています。また、ベースとなっている teTeX のバージョンも 2007 であり、セキュリティフィックスはなされているものの、大雑把に言えば **2007 年から更新されていません**[*12]。

導入後にも、 DVI ファイルの表示や PostScript/PDF への変換のために幾つかおまじないが必要だったりします。例えば

1. dvipsk-ja 用の VF ファイルの準備のために、インストール後に以下を唱える。

```
$ sudo jisftconfig add
```

2. xdvik-ja での表示のために

```
# 明朝系の表示に使用するフォント
$ fc-match serif:lang=ja
# ゴシック系の表示に使用するフォント
$ fc-match sans-serif:lang=ja
```

   を確認し、必要に応じて、`~/.fonts.conf` に `serif, sans-serif` のエントリを追記して

```
$ sudo update-vfontmap
```

3. フォントが埋め込まれていない PDF の表示のために、 `Ryumin, Gothic-BBB` のエントリを`~/.fonts.conf` に追記する。
4. フォントを埋め込んだ PDF を生成するためには、`/etc/texmf/texmf.d/75DviPS.cnf` の `TTFONTS`、もしくは `%OSFONTDIR` に埋め込みたいフォントのパスを追加した後に

```
$ sudo update-texmf
```

   を唱えて、さらに map ファイルを適当に用意しておく。

などです。

[*10] teTeX: no next release「 `http://article.gmane.org/gmane.comp.tex.tetex.beta/812`」

[*11] 発表時および発表後は Q&A および設定のお手伝いは行ないます。

[*12] Squeeze リリース前に xdvik-ja の 64bit 対応の必要があり、佐々木が patch を当ててパッケージを再構築したりしました。

2.3.2 Squeeze で TeXLive(>=2011) を使う!?

Debian のパッケージではありませんが、 amd64 もしくは i386 の場合には Norbert さんを中心に提供されている「 tlptexlive リポジトリ[*13]」から、 TeXLive 全体のコンパイル済みバイナリを取得して使用する、という方法もあります。この場合には

1. 既存の Debian パッケージの依存関係解消のために `equivs` でダミーパッケージを作成する
2. TeXLive のパッケージマネージャである `tlmpgr` でのバイナリ導入先を、管理しやすい所に追いておく。

なんて工夫が必要でしょう。佐々木は Squeeze 環境では `stow` で tlptexlive リポジトリでのインストール物を管理しています。

ちなみに tlptexlive の配布物は `TeXLive (>= 2011)` 相当ですので、フォント周りの設定は後述の Wheezy の場合と同じです。

2.3.3 Wheezy の現状

既に TeXLive 2011(2012/dev) が Sid に upload されています。しかしながら、本原稿執筆時点ではまだ Wheezy には更新された TeX 関連のパッケージは落ちてきていません。発表時には Wheezy に落ちてきていることを期待しつつ、変更点を簡単に述べてみます。

このバージョンの TeXLive には既に pTeX などがマージされているため、 Squeeze までで提供していた teTeX ベースのパッケージ群 `ptex-bin, ptex-base, ...` は軒並 obsolete となります[*14]。現状では pTeX 関連は `texlive-binaries, texlive-lang-cjk` を導入することで一括で install される予定です。また、フォント関連の扱いも `updmap-setup-kanji` というコマンドによって、一括で設定できるようになります。たとえば

```
$ sudo updmap-setup-kanji nofont
```

といった塩梅です (上記の設定は、フォントをまったく埋め込まない場合です)。このコマンドにより、 `xdvik-ja, dvips, dvipdfmx` のフォントマップが一括して更新されます。 TeXLive ではフォントマップファイルとして、 1) 埋め込まない場合、 2)IPAex を使う場合、 3) ヒラギノを使う場合、 4) 小塚フォントを使う場合の 4 つを提供しており、 Debian のパッケージでもこれらのマップファイルを提供しています。 Debian の main セクションのみで作業をする場合には 1) or 2) を使うことになるでしょう。現時点では、

- Conflicts, Replace, Provides の設定が半端で Squeeze からの upgrade がうまくできない (パッケージがある)。
- TeXLive 本体でも配布されているフォントがあり、既存のパッケージと重複しているため、 Depends を追加して適宜 symbolic link に置換する必要がある。

といった問題があって、今後の修正が大変ですが、 Wheezy のリリースまでにはちゃんと直る (直す?) でしょう。

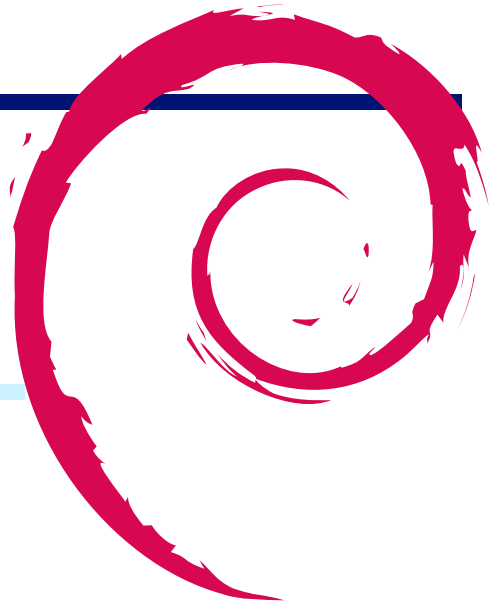
2.4 最後に

駆け足でしたが、日本語 TeX の現状および Debian での TeX 関連の現状をまとめてみました。まだまだ修正すべき点もありますが、今後は `updmap-setup-kanji` によるフォント関連の一括管理が有効になるため、おまじないのような、ともすればバッドノウハウと扱われがちな煩雑な設定は不要になるでしょう。 Wheezy からは、 UTF-8 へ対応した pTeX が、もっと言えば JIX X 4051 の組版要件を満たしつつ多言語処理も可能となった日本語 TeX 環境が非常に簡単に手に入るようになる予定です[*15]。

[*13] tlptexlive リポジトリ「 `http://tutimura.ath.cx/ptexlive/?tlptexlive%A5%EA%A5%DD%A5%B8%A5%C8%A5%EA`」

[*14] TeXLive 本体にマージされていない `xdvik-ja` のみが残ることになります。

[*15] 本来であればここで参考文献を列記すべきですが、紙面の都合上脚注に記す無作法をお許し下さい。



3 Linux-PAM の設定について

西山和広

3.1 Introduction

3.1.1 PAM とは何か?

Linux-PAM (Pluggable Authentication Modules for Linux) とは、アプリケーションがユーザーをどう認証するかをローカルシステムの管理者が設定できるようにするための共有ライブラリ一式です。

PAM のおかげで、アプリケーションをコンパイルしなおさなくても、認証方法を変更したり、権限の付与の仕方を変更したりできるようになっています。

3.1.2 NSS とは何か?

PAM と関係の深いライブラリとして NSS (Name Service Switch) があります。 NSS はユーザー名とユーザー ID との変換をしたり、ホスト名と IP アドレスとの変換をしたりするときに使われます。

3.1.3 PAM と NSS との違い

PAM は認証部分のみなので、基本的にはログインやログアウトのときとパスワード変更に関係します。 (厳密にはアプリケーション次第です。)

ログイン中に `id` コマンドで表示されるユーザー ID とユーザー名の対応 [*16] や `ls -l` でファイルシステムに記録されているユーザー ID からユーザー名への変換 [*17] などは `/etc/nsswitch.conf` で設定する NSS の機能になります。

3.1.4 PAM の設定

PAM は設定ファイルに実行したいモジュールを並べておいて、それを順番に実行していくようなものだと思えば良いでしょう。

たとえば、

- `/etc/passwd` などのローカルファイルでの認証
  - 成功すれば認証成功をアプリケーションに返す
  - 失敗すれば次へ
- LDAP での認証
  - 成功すれば認証成功をアプリケーションに返す
  - 失敗すれば次へ

[*16] グループ ID とグループ名も同様です。

[*17] ファイルシステムには所有者はユーザー ID で記録されています。そのため、たとえばユーザーを削除して存在しないユーザーのファイルがある場合には、ユーザー名への変換が出来ないので数字で表示されます。

- 次がないので認証失敗をアプリケーションに返す

という動作をします。

3.2 ファイル配置

3.2.1 設定ファイル

/etc/pam.d/ の下にアプリケーションごとの設定ファイルがあります。以下はその例です。

```
$ ls /etc/pam.d
atd       chsh            common-password               cron      other   su
chfn      common-account  common-session                login     passwd  sudo
chpasswd  common-auth     common-session-noninteractive  newusers  sshd
```

- どの設定ファイルがどのアプリケーションのものなのかはファイル名から推測できます
- ない場合は other が使われます
- それぞれの中から @include で common-* で共通の設定を使うようになっています

/etc/pam.d/ がない場合は /etc/pam.conf が使われると /etc/pam.conf の中のコメントに書いてありますが、この記述は歴史的なもので最近の Linux では使われていません。

3.2.2 モジュール

以下は PAM モジュールの例です。

```
$ ls /lib/x86_64-linux-gnu/security
pam_access.so     pam_keyinit.so    pam_nologin.so     pam_tally.so
pam_debug.so      pam_lastlog.so    pam_permit.so      pam_tally2.so
pam_deny.so       pam_ldap.so       pam_pwhistory.so   pam_time.so
pam_echo.so       pam_limits.so     pam_rhosts.so      pam_timestamp.so
pam_env.so        pam_listfile.so   pam_rootok.so      pam_umask.so
pam_exec.so       pam_localuser.so  pam_securetty.so   pam_unix.so
pam_faildelay.so  pam_loginuid.so   pam_selinux.so     pam_userdb.so
pam_filter.so     pam_mail.so       pam_sepermit.so    pam_warn.so
pam_ftp.so        pam_mkhomedir.so  pam_shells.so      pam_wheel.so
pam_group.so      pam_motd.so       pam_stress.so      pam_xauth.so
pam_issue.so      pam_namespace.so  pam_succeed_if.so
```

- 設定ファイルに書かれる pam_unix.so などは dlopen(3) で動的に読み込まれます
- そのため PAM の認証を使っているプログラムを入れた chroot 環境を作るときには気をつける必要があります
- squeeze までは /lib/security/ や /lib64/security/ にあります
- multiarch 対応で最近は /lib/x86_64-linux-gnu/security/ などの /lib/<triplet>/security/ にあります。上記の例は wheezy (testing) なので /lib/security/ ではなく/lib/x86_64-linux-gnu/security/ になっています

3.3 設定ファイルの書式

設定ファイルの例を載せておきます。(一部省略)

```
$ egrep '^[^#]' /etc/pam.d/cron
@include common-auth
session       required   pam_env.so
session       required   pam_env.so envfile=/etc/default/locale
@include common-account
@include common-session-noninteractive
session    required   pam_limits.so
```

- 設定ファイルには以下の項目を指定します。詳細は後述します

  service　アプリケーションに対応する名前

  type　PAM の分類

control　動作指定

modules-path　PAM モジュールへのパス

module-arguments　PAM モジュールの引数

- # から行末まではコメントになります
- \ が改行 (<LF>) の直前にあると継続行になります
- /etc/pam.conf では「 service type control module-path module-arguments 」という書式で設定します。(service, type, control の大文字小文字は無視されます。)
- /etc/pam.d/ では service がファイル名 (必ず小文字) になります
- 残りの「 type control module-path module-arguments 」がファイルの内容になります

3.3.1 service

- service は具体的には login や su になります
- other という service 名はデフォルト設定用として予約されています
- Debian では common- で始まる名前のファイルが共通設定用のファイルになっていて、他の設定ファイルから @include で読み込まれています

3.3.2 type

account　認証以外のアカウント管理に使われます。たとえば昼間だけログインできるようにしたり、 nologin ファイルがあるときは一般ユーザーにログインさせないようにしたりできます

auth　アプリケーションにパスワード入力を要求するなどの方法でユーザー認証をします。グループ権限の付与などの機能もあります

password　パスワード変更などの認証トークン変更機能を提供します

session　サービス利用の前後に何かをするモジュールタイプです。ログを取ったり、ディレクトリをマウントしたり、/etc/motd を表示したり、環境変数を設定したりできます

@include　Debian ではここに @include を指定することで別のファイルを読み込めるようになっています

3.3.3 control

PAM モジュールを実行してその結果、どうするのかの設定です。詳細については後述します。

3.3.4 module-path

PAM モジュールのファイルへのパスを絶対パスかデフォルトのモジュールの置き場所である /lib/security/ などからの相対パスで指定します。

3.3.5 module-arguments

スペース区切りでモジュールへの引数を指定します。スペースを含む引数を指定する場合は以下の例のように [] でくくります。

```
squid auth required pam_mysql.so user=passwd_query passwd=mada \
      db=eminence [query=select user_name from internet_service \
      where user_name='%u' and password=PASSWORD('%p') and \
      service='web_proxy']
```

] を含めたい場合は \] と指定します。つまり以下のようになります。

```
[..[..\]..]    -->   ..[..]..
```

3.4 control の設定について

3.4.1 PAM の内部状態

PAM はモジュールを実行していくときに内部的に

- 初期状態
- 成功状態
- 失敗状態

の 3 つの状態があり、最終的に成功状態なら成功をアプリケーションに返し、そうでなければ失敗をアプリケーションに返します。(初期状態のままのときも失敗を返します。)

3.4.2 control の省略形式

control の指定には、省略した形式として以下のものがあります。

required　成功しても失敗しても続きを実行してから結果を返します。失敗状態以外で成功した場合は成功状態にします。失敗した場合は失敗状態にします

requisite　失敗した場合は失敗状態にして、すぐにアプリケーションに失敗を返します。失敗状態以外で成功した場合は成功状態にします

sufficient　失敗状態以外で成功した場合は、すぐにアプリケーションに成功を返します。失敗は無視して続きを実行します

optional　成功か失敗かは気にせず実行したいモジュールに使います。成功か失敗かは他のモジュールがないときだけ影響します

include, substack　省略形式ではありませんが、このような指定も出来ます。しかし Debian では上述の `@include` が使われていて、これらは普通は使われていないのと、内部状態の説明が複雑になるため省略します

3.4.3 control の省略しない形式

もっと複雑な形式として以下のものがあります。

```
[value1=action1 value2=action2 ...]
```

3.4.4 control の value

`valueN` は PAM モジュールからの返値で `actionN` はその返値のときにどうするかという設定です。 `valueN` には、

- `success`
- `open_err`
- `symbol_err`
- `service_err`
- `system_err`
- `buf_err`
- `perm_denied`
- `auth_err`
- `cred_insufficient`
- `authinfo_unavail`
- `user_unknown`
- `maxtries`
- `new_authtok_reqd`
- `acct_expired`
- `session_err`
- `cred_unavail`
- `cred_expired`
- `cred_err`
- `no_module_data`
- `conv_err`
- `authtok_err`
- `authtok_recover_err`
- `authtok_lock_busy`
- `authtok_disable_aging`
- `try_again`
- `ignore`
- `abort`
- `authtok_expired`
- `module_unknown`
- `bad_item`
- `conv_again`
- `incomplete`
- `default`

が指定できます。 `default` は名前からわかる通り明示的に `valueN` に指定されなかったときのデフォルト指定です。

`valueN` に指定できる値の完全なリストは `libpam0g-dev` パッケージをインストールして

- `/usr/include/security/_pam_types.h`

を参照してください。

3.4.5 control の action

actionN に指定できるものは以下の通りです。

ignore　無視します。モジュールの返値はアプリケーションに返す値には影響しません
bad　失敗状態にします
die　失敗します。失敗状態にして、すぐにアプリケーションに失敗を返します
ok　失敗状態以外なら成功状態にします。すでに失敗状態のときは失敗状態のままです
done　失敗状態以外なら成功状態にして、すぐにアプリケーションに成功を返します。すでに失敗状態のときは失敗状態のままです
N (1 以上の整数)　次の N 個のモジュールを実行せずに飛ばします
reset　内部状態を初期状態に戻します

3.4.6 省略形の展開

省略形は [...] の書式で書くと以下のようになります。

required　`[success=ok new_authtok_reqd=ok ignore=ignore default=bad]`
requisite　`[success=ok new_authtok_reqd=ok ignore=ignore default=die]`
sufficient　`[success=done new_authtok_reqd=done default=ignore]`
optional　`[success=ok new_authtok_reqd=ok default=ignore]`

3.5 PAM 設定フレームワーク

Red Hat 系 Linux には以前から PAM や NSS の自動設定用のコマンドとして authconfig があります。しかし、昔の Debian にはそういうフレームワークはありませんでした。

Ubuntu では、そのようなフレームワークとして auth-client-config が使われるようになりました。その後 pam-auth-update が使われるように変わりました。そして Debian にも pam-auth-update が入って今は Debian でも Ubuntu でも pam-auth-update が標準の PAM の設定用フレームワークとして使われています。[*18]

3.5.1 pam-auth-update

pam-auth-update コマンドは libpam-runtime パッケージに入っています。PAM モジュールパッケージで /usr/share/pam-configs の中にプロファイルがインストールされます。pam-auth-update コマンドは、そのプロファイルを元に /etc/pam.d/common-* ファイルを更新します。

3.6 設定ファイルの例

- pam-auth-update で生成された設定ファイルの一部と sshd の設定を例として説明をします。

3.6.1 /etc/pam.d/common-auth

common-auth は認証の共通処理の設定ファイルです。

1. まず最初に “Primary” block のモジュールを pam_unix.so, pam_ldap.so と順番に試して、どこかで成功したら pam_permit.so まで飛ばして成功状態にします

[*18] Debian にはありませんが、Ubuntu には auth-client-config パッケージはまだ存在しています。pam-auth-update は PAM の設定のみなので、もしかすると NSS の設定には使われているのかもしれません。(使っていないので詳細は不明です。)

2. すべて失敗した場合は fallback の pam_deny.so の行で必ず失敗して、そのままアプリケーションに認証失敗を返します
3. 途中で成功して pam_permit.so の行に飛んできた場合は、そのまま続きの行を実行していきます
4. @include 元のファイルで続きの処理を用意していることもあるので sufficient で成功をすぐに返してしまうことは避けて success=N で飛ばして pam_permit.so で成功状態にするという手間をかけているようです
5. pam_ldap.so の use_first_pass は pam_unix.so の認証のときに入力されたパスワードを使って pam_ldap.so の認証も試すという意味です。 use_first_pass がないと pam_unix.so でパスワード入力が要求されて、さらに pam_ldap.so でもパスワード入力を要求されるということになります。(プロファイルで Auth-Initial と Auth にわかれているのはそういう設定を使いわけられるようにするためのようです。)

```
# here are the per-package modules (the "Primary" block)
auth    [success=2 default=ignore]      pam_unix.so nullok_secure
auth    [success=1 default=ignore]      pam_ldap.so minimum_uid=1000 use_first_pass
# here's the fallback if no module succeeds
auth    requisite                       pam_deny.so
# prime the stack with a positive return value if there isn't one already;
# this avoids us returning an error just because nothing sets a success code
# since the modules above will each just jump around
auth    required                        pam_permit.so
# and here are more per-package modules (the "Additional" block)
auth    optional                        pam_cap.so
# end of pam-auth-update config
```

3.6.2 /etc/pam.d/common-session

- この例の common-session では “Primary” block にモジュールがないために pam_permit.so でいきなり pam_deny.so を飛びこえるようになっています
- その後 pam_unix.so と pam_ldap.so でも何か追加の処理をして pam_tmpdir.so で環境変数 TMPDIR などを設定しています

```
# here are the per-package modules (the "Primary" block)
session [default=1]                     pam_permit.so
# here's the fallback if no module succeeds
session requisite                       pam_deny.so
# prime the stack with a positive return value if there isn't one already;
# this avoids us returning an error just because nothing sets a success code
# since the modules above will each just jump around
session required                        pam_permit.so
# and here are more per-package modules (the "Additional" block)
session required        pam_unix.so
session [success=ok default=ignore]     pam_ldap.so minimum_uid=1000
session optional pam_tmpdir.so
# end of pam-auth-update config
```

3.6.3 /etc/pam.d/sshd

1. まず pam_env.so で環境変数を設定しています
2. # で始まる行だけでなく、行の途中の # 以降もコメントです
3. 2 個目の pam_env.so で /etc/default/locale を読み込んでいます [*19]
4. pam_nologin.so では nologin ファイルがあるときにログインを認可しないようにしています
5. session では pam_motd.so で /etc/motd の表示をしています。最近の Debian や Ubuntu の pam_motd.so では /etc/update-motd.d があれば表示前に内容が更新されるようになっています。 Ubuntu のサーバーに ssh でログインしたときにいろいろな情報がでるのはそのためです。 Ubuntu の場合は update-notifier-common を入れておくとパッケージの更新や再起動が必要かどうかが ssh などでのログイン時に出るようになるのでおすすめです。 Debian では /etc/update-motd.d のファイルが入らない (http://bugs.debian.org/580286) ため、自動では出ません

[*19] 余談ですが、昔の Debian では /etc/default/locale は存在しなくて、アップグレードでも自動で作成はされず、ログをみるとエラーが出ていたということがあり、そのときは自分で /etc/default/locale を作成しました。今は update-locale というコマンドで更新するとチェックもしてくれて良い感じになるようです。

6. 他にはメールボックスの状態を表示したり、リソース制限を反映したりしているようです

```
# PAM configuration for the Secure Shell service

# Read environment variables from /etc/environment and
# /etc/security/pam_env.conf.
auth       required     pam_env.so # [1]
# In Debian 4.0 (etch), locale-related environment variables were moved to
# /etc/default/locale, so read that as well.
auth       required     pam_env.so envfile=/etc/default/locale

# Standard Un*x authentication.
@include common-auth

# Disallow non-root logins when /etc/nologin exists.
account    required     pam_nologin.so

# Uncomment and edit /etc/security/access.conf if you need to set complex
# access limits that are hard to express in sshd_config.
# account  required     pam_access.so

# Standard Un*x authorization.
@include common-account

# Standard Un*x session setup and teardown.
@include common-session

# Print the message of the day upon successful login.
session    optional     pam_motd.so # [1]

# Print the status of the user's mailbox upon successful login.
session    optional     pam_mail.so standard noenv # [1]

# Set up user limits from /etc/security/limits.conf.
session    required     pam_limits.so

# Set up SELinux capabilities (need modified pam)
# session  required     pam_selinux.so multiple

# Standard Un*x password updating.
@include common-password
```

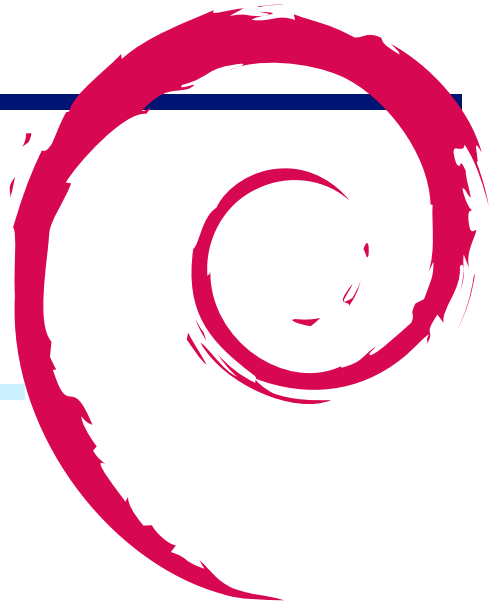
3.7 設定変更時の注意事項

設定を変更するときは root 権限のシェルを開いたままにしておくことをお勧めします。もし設定を間違えてしまうと sudo も su もコンソールからの root でのログインも出来なくなって困ることになります。

ただし `pam-auth-update` を使ってパッケージでインストールされた設定の有効無効を切り替えるだけの場合は問題が起きる可能性は低いので、そこまでしなくても普通は大丈夫だと思います。

3.8 参考文献

- http://linux-pam.org/Linux-PAM-html/Linux-PAM_SAG.html The Linux-PAM System Administrators' Guide Version 1.1.2, 31. August 2010
- http://archive.linux.or.jp/JF/JFdocs/User-Authentication-HOWTO/ User Authentication HOWTO 2000/05/02
- https://wiki.ubuntu.com/PAMConfigFrameworkSpec PAMConfigFrameworkSpec - Ubuntu Wiki
- https://wiki.ubuntu.com/AuthClientConfig AuthClientConfig - Ubuntu Wiki



4 数学ソフトウェア使ってますか?

濱田龍義

Debian パッケージには「数学」というセクションが存在し、多数の数学ソフトウェアが提供されています。一方で、パッケージには未収録ですが、数学の研究、教育の現場で使われているフリーソフトウェアが世界中で開発されています。2003 年頃から、このような数学ソフトウェアをライブ CD である KNOPPIX に収録して紹介してきたプロジェクトが KNOPPIX/Math です。KNOPPIX は Debian をベースに、ドイツの Klaus Knopper によって開発が進められている Live Linux です。当初は CD 起動のみに対応していましたが、DVD や USB メモリーディスクにも対応し、現在も開発が進められています。KNOPPIX/Math では、2006 年から LiveDVD を採用しました。また、2012 年からは KNOPPIX 以外のシステムの利用も考慮し、MathLibre とプロジェクト名を変更しています。本講演では MathLibre に収録している様々な数学ソフトウェアから、いくつかを紹介します。

4.1 数学ソフトウェアの歴史

我々は、数学を取り扱うもの、また、数学に関連するもの、そういったものすべてを数学ソフトウェアと呼んでいます。Donald E. Knuth によって生み出された組版システム TEX は、数学を生業とするものにとっては、欠かせない道具であり、これもまた、数学ソフトウェアの一種です。一般的に、数式処理ソフトウェア、数値計算ライブラリ、可視化ツールなども数学ソフトウェアと考えて良いでしょう。

その中で、数式処理ソフトウェアと呼ばれるソフトウェアについて紹介します。Debian のオフィシャルパッケージの中で、比較的使いやすい物としては、Maxima が有名です。Maxima の原型は MACSYMA と呼ばれるシステムで、1960 年代に MIT の人工知能研究グループによって開発されました。1980 年代には MACSYMA は商用ソフトウェアとしても流通していましたが、現在の Maxima は、William Schelter によって GNU Common Lisp 上に実装されたものを原型に 2001 年に GPL 化され、Maxima と名称を改めて、フリーソフトウェアとして公開されたものです。Maxima は、方程式の求解、多項式の因数分解、初等函数の微積分、行列計算、グラフの描画等に対応しています。中でも積分アルゴリズムの実装について評価が高いようです。

Debian 上で Maxima を起動するための命令は `maxima` です。

```
knoppix@Microknoppix:~$ maxima

Maxima 5.26.0 http://maxima.sourceforge.net
using Lisp GNU Common Lisp (GCL) GCL 2.6.7 (a.k.a. GCL)
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i1) integrate(1/(x^3+1),x);
                                      2 x - 1
                        2           atan(-------)
                   log(x  - x + 1)        sqrt(3)    log(x + 1)
(%o1)           - --------------- + ------------- + ----------
                          6             sqrt(3)          3
(%i2)
```

4.1.1 Maxima のユーザインターフェース

コマンドラインからの利用が一番速く手軽ですが、一般的なグラフィカルユーザインターフェースとしては、 XMaxima (`xmaxima`) や WxMaxima (`wxmaxima`) が存在します。 XMaxima は Tcl/Tk, WxMaxima は WxWidget によって実装されています。また、 Emacs 用に便利なインターフェースとしては imaxima (`maxima-emacs`) が存在します。 Emacs 上で `M-x imaxima` で起動すると、数式の計算結果を TEX スタイルに整形された形で出力されます。また、少し変わったインターフェースとしては GNU TeXmacs (`texmacs`) と呼ばれる科学技術計算用ワードプロセッサが存在します。こちらも計算結果の出力を整形された形で表示することができます。

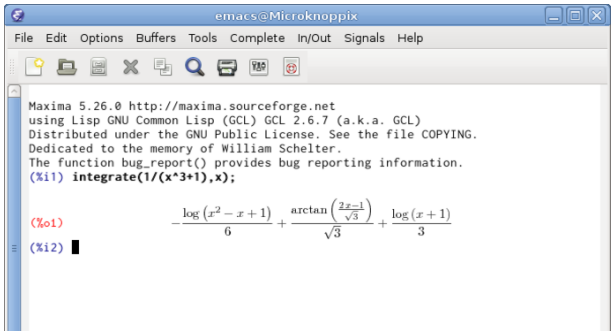


図 2　imaxima on Emacs

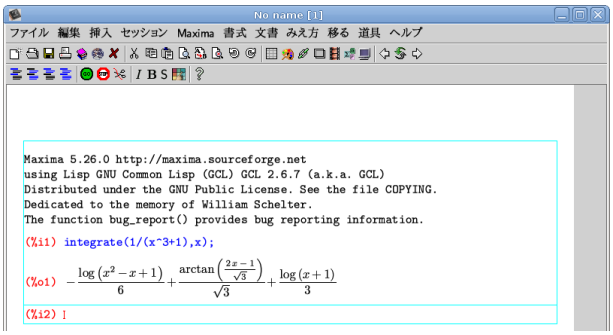


図 3　GNU TeXmacs

Maxima と同じく、長い歴史を持つ数式処理システムとしては、 Reduce もよく知られた存在です。 Reduce もまた、 1960 年代に開発が始められたシステムですが、こちらは理論物理学からの要請により開発されました。 Reduce も MACSYMA と同様に 1980 年代に商品化され、日本では差分方程式論、および可積分系等の分野で多くのユーザを獲得しています。その後、 2009 年に Reduce は Tony Hearn によって BSD ライセンスで公開され、現在に至ります。まだ、 Debian オフィシャルパッケージには含まれていませんが、 getdeb (`http://www.getdeb.net/`) から、パッケージが入手可能です。 Reduce もまた、微分方程式の求解、積分、行列計算、グラフの描画等に対応しています。

Maxima も Reduce も、たいへん歴史の古いソフトウェアであり、現在も、多くのユーザを獲得し、開発グループが日夜、更新を続けているソフトウェアです。その他にも、数論の PARI/GP、群論の GAP、統計計算の R、可換環論の Singular、 Macaulay2 等が研究ツールとして有名です。 PARI/GP、 GAP、 R はオフィシャルパッケージになっていますが、 Singular、 Macaulay2 については、上流開発者によるパッケージは提供されていますが、オフィシャルには収録されていません。

一方、日本国内における数式処理システムの開発については、 1970 年代に日本電信電話会社で AL が、 1980 年代に理研で GAL が開発されました。また、 1980 年代末に、富士通研究所で開発された Risa/Asir は、現在、拠点を神戸大学に移して、研究、開発が進められています。特にグレブナー基底等、多項式の計算に特化して、高速計算が可能であり、 MathLibre プロジェクトの中心的な存在として収録されています。ただ、 Risa/Asir は富士通研究所時代のライセンスを引き継いでおり、オープンソースソフトウェアライセンスとは異なるライセンスで配布されています。

4.2 最近のシステム

ここで、最近の動きに注目して、いくつかの数学ソフトウェアを紹介します。

4.2.1 Sage

Sage は 2005 年に William Stein によって開始されたオープンソースプロジェクトです。彼が Sage の開発を始めた背景については “Mathematical Software and Me: A Very Personal Recollection” [1] に詳しく述べられています。現

在、 Sage は数学者を始めとする大勢の専門家によって開発が進められています。 2007 年にはフリーソフトウェアの賞である Les Trophées du Libre の科学技術部門において金賞を受賞しています。 Sage Days と呼ばれるイベントを世界各地で開催しており、開発者と利用者からなる巨大なコミュニティが存在します。先日、九州大学において日本で初の Sage Days が開催され、多くの参加者を得ました。

Sage は「 車輪の再発明」はしないということを掲げており、既に実績のある Maxima や Singular、 PARI/GP、 GAP、 R といった数学ソフトウェアを組み合わせることで、使いやすい環境の構築を目指しています。それぞれの数学ソフトウェアをつなぎ合わせているのはオブジェクト指向プログラミング言語 Python です。 Python 自身もオープンソースソフトウェアですが、世界中で熱狂的な利用者、開発者を獲得しています。その使いやすさと、多機能性からさまざまなプロジェクトが Python で開発されています。それまでは、数学ソフトウェアを使いこなすためには、そのソフトウェアごとに異なるプログラミング機能を活用しなければいけなかったのですが、 Sage は Python という一般的なプログラミング言語を用いることで、統一的なインターフェースを提供しています。また、 Python のオブジェクト指向言語としての機能を活用できる点も魅力となっているようです。 Sage Notebook と呼ばれるユーザーインターフェースを提供しており、 Mozilla Firefox 等の Web ブラウザ上で数式処理やグラフ描画といった機能を利用することができます。

Sage についての入門書としては、 Ted Kosan による “Sage for Newbies” [2] が良く知られています。これは横田博史氏によって「 はじめての Sage」 [3] として翻訳され、 MathLibre にも収録されています。最近、韓国では大学初年級の微分積分と線形代数について Sage を用いて解説した書籍が出版されました。

Sage は、あまりに巨大なシステムであり、更新も頻繁であるため、現在は、 Debian パッケージとして配布されていません。 Linux 用にはバイナリー版が配られています。多少の時間はかかりますが、 CPU ごとに最適化するため、ソースからコンパイルすることをお勧めします。 Windows 版については、以前は仮想マシンの Ubuntu 上で起動したサーバに Windows 上のブラウザから接続する形式だったのですが、現在は、 Fedora Core 上で起動したサーバに Fedora Core 上の Chrome を接続させる形になったようです。

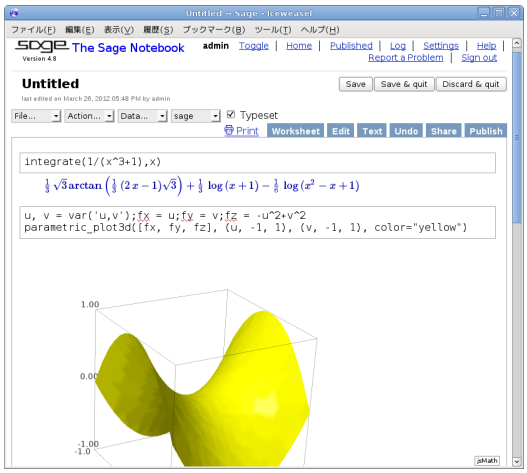


図 4　Sage によるグラフ描画

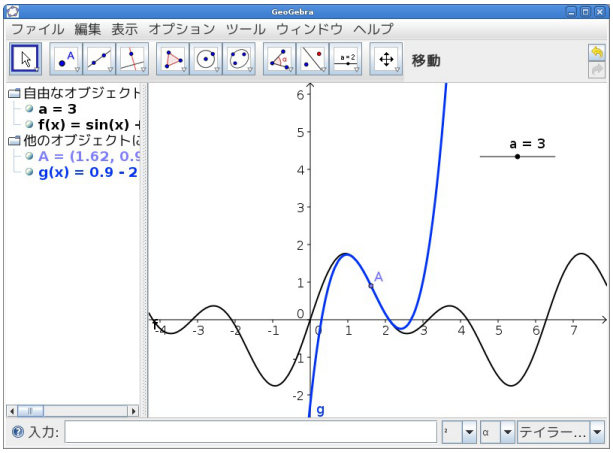


図 5　GeoGebra によるテイラー多項式

4.2.2　GeoGebra

GeoGebra はオーストリアのヨハネスケプラー大学の Markus Hohenwarter によって始められた数学ソフトウェアプロジェクトです。彼はザルツブルク大学の学生時代にテキサス・ インスツルメンツ社の電卓 TI-92 Plus に触れる機会を持ちました。彼は電卓に収録されていた動的幾何ソフト Cabri Geometry と数式処理システム Derive に刺激を受けて開発を志したということです [4]。 2001 年 2 月に最初のプロトタイプを開発し、 2002 年 3 月には GeoGebra の開発でコンピュータサイエンスと数学教育に関する修士号を取得しています. この間、 GeoGebra は、各国で数多くの賞を受賞しています。 2004 年から 2006 年にかけては数学教育に関する PhD プロジェクトとして開発が進められ、 Austiran Academy of Sciences から支援を受けて、着実にその存在を世界中に知らしめました。 GeoGebra もまた、オープンソースソフトウェアとして公開されています。 Java の実行環境を必要としますが、 Windows、 MacOS X、 Linux 等、計

算機環境を問わずに利用可能です。 Debian では stable に 3.2 系が testing に 4.0 系が収録されています。

GeoGebra は起動すると “Dynamic Mathematics for Everyone” というメッセージを表示します。日本語に訳すと「 動的数学ソフトウェア」でしょうか。これは GeoGebra が Cabri や Cinderella と言った動的幾何学ソフトウェア (Dynamic Geometry Software) と呼ばれるソフトウェアの影響を受けていること、また、主要なユーザインターフェースが動的幾何学ソフトウェアとしての機能を備えていることから類推されます。 GeoGebra という名称は幾何学（ Geometry） + 代数学（ Algebra）という意の造語です. 開発初期の段階では GeoGebra は動的幾何学ソフトウェアとしての機能しか持ちませんでした。しかし、現在では関数入力によるグラフ描画、数式処理、スライダー、表計算機能等を備えており、幾何学ソフトウェアという枠組みだけでは語り尽くせない存在です。

メニューやヘルプ、マニュアル等の翻訳についても世界中のボランティアスタッフによって活発に行われており、最新の GeoGebra 4.0 では約 50 ヵ国語に対応しました。日本語化については、北海道教育大学の和地輝仁氏が中心となって進めており, 国内のユーザーを増やすきっかけとなっています。

GeoGebra は動的幾何学ソフトウェアとしての機能とグラフ描画ソフトの機能が連携することで使い易いインターフェースを提供しています. また、 Maxima や Reduce と連携して数式処理機能を備えているため、微分や積分、因数分解等の基本的な演算にも対応しています。図 5 のようにグラフ上の点を動的に動かしてテイラー多項式によって描かれるグラフを観察することもできます。

表計算ビューを備えたことで、統計方面についても機能が強化されており、教材作成に威力を発揮するものと思われます。すべての機能を述べることはできませんが、ヘルプやさまざまなサンプル、ムービー等も豊富で、初めての方でも使いやすい数学ソフトウェアだと思います。どちらかと言えば、研究よりも数学教育に焦点をあてたソフトウェアですが、その使いやすいインターフェースから構成される可視化機能は、プレゼンテーションなどでも威力を発揮するでしょう。次期バージョンの 4.2 では本格的な数式処理シェルを備え、 5.0 では 3D に対応する予定です。潜在的な能力も含めて今後の展開が楽しみな数学ソフトウェアです。

2012 年 7 月 4 日–6 日には RIMS 共同研究「 数式処理研究の新たな発展」が計画されていますが、 7 月 5 日に GeoGebra Institute から Zsolt Lavizca, Balázs Koren の 2 人が GeoGebra の最新事情についての講演が行われます。

4.3 まとめ

ここに紹介したソフトウェアは、 MathLibre に収録している 100 以上の数学ソフトウェアのごく一部です。本稿が、数学ソフトウェアに興味を持っていただくきっかけになれば幸いです。

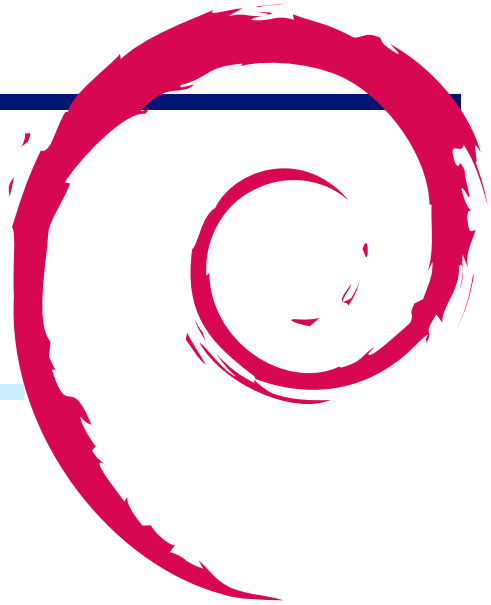
参考文献

[1] William Stein, “*Mathematical software and me:A very personal recollection*”, `http://wstein.org/`

[2] Ted Kosan, “*Sage for newbies*”, `http://sage.math.washington.edu/home/tkosan/`

[3] Ted Kosan, 横田博史 訳,「 **はじめての** *Sage*」, `http://www.bekkoame.ne.jp/~ponpoko/KNOPPIX/`

[4] Markus Hohenwarter and Judith Preiner, “*Dynamic Mathematics with GeoGebra*”, The Journal of Online Mathematics and Its Applications, (7), 2007, `http://mathdl.maa.org/mathDL/`



5 IPython notebook とその周辺

本庄弘典

5.1 はじめに

最近 ipython の qtconsole でコンソール上にグラフを描画しているスクリーンショットを見る機会があり、ちょっとかっこいいかなとロクに使ったことがない Python を使い始めました。今回は ipython qtconsole を使用したグラフの描画から、 Mathmatica notebooks のような Web ベースの数式処理システム ipython notebook を Python 初心者の視点で紹介してみたいと思います。

5.2 IPython

ipython は Fernando Perez 氏によって作成された Python のインタラクティブシェルで、現在の最新リリースバージョンは 0.12.1 です。 Python はそれ自体でインタラクティブシェルの機能を持っていますが、 ipython は Python シェルと比較して次の特徴を持っています。

- terminal での使用に加え Qt 等を用いたグラフィカルなコンソールが使用可能
- コード補完
- シンタックスハイライト
- 並列コンピューティングが可能
- Web ベースの notebook(ipython 0.12 から)

Python シェルと ipython は次の図のように、プロンプト等に違いがあります。

```
>>> def fib(n):
...     if n == 0:
...         return 0
...     if n == 1:
...         return 1
...     return fib(n-1) + fib(n-2)
...
>>> [ fib(i) for i in range(0, 10) ]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
>>>
```

python シェル

```
In [1]: def fib(n):
   ...:         if n == 0:
   ...:             return 0
   ...:         if n == 1:
   ...:             return 1
   ...:         return fib(n-1) + fib(n-2)
   ...: [ fib(i) for i in range(0, 10) ]
   ...:
Out[1]: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

In [2]:
```

ipython

図 6　python シェルと ipython の違い

5.3 IPython qtconsole

ipythopn に qtconsole オプションを指定して実行することにより、 GUI 環境での ipython が起動します。またこの環境で `--pylab inline` を指定して起動することで、コンソール内にグラフを表示させることが可能となります。

```
$ ipython qtconsole --pylab matplotlib
```

この環境では前述のグラフ表示に加え、複数行をまとめて扱うコマンドの履歴などが使用できます。グラフを描画すると次のように表示されます。

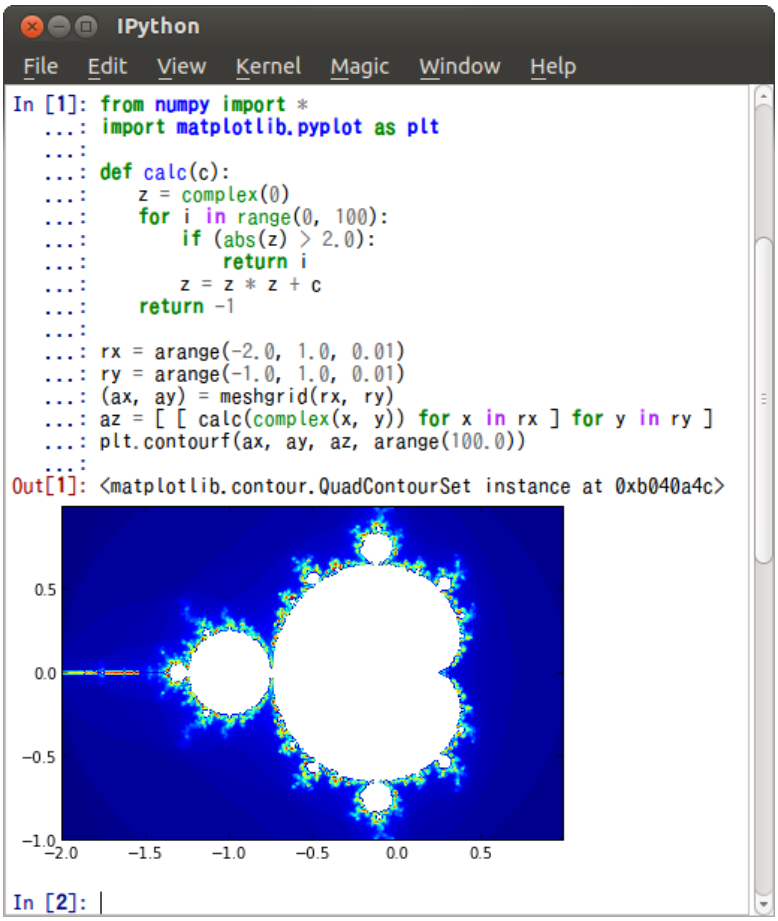


図 7　qtconsole でマンデルブロ集合

testing での ipython qtconsole および matplotlib は次のコマンドでインストール出来ます。

```
$ sudo aptitude install ipython-qtconsole python-matplotlib
```

また squeeze は backports を使用してインストールを行いました。まず/etc/apt/sources.list に次の行を追加し、

```
deb http://backports.debian.org/debian-backports squeeze-backports main
```

`aptitude update && aptitude upgrade` を実行した後、次のコマンドでインストールします。

```
$ sudo aptitude install python-setuptools python2.6-dev ncurses-dev \
  libzmq-dev python-pygments python-matplotlib pyqt4-dev-tools
```

パッケージのインストール後、 python の `easy_install` コマンドを使用して ipython をインストールします。

```
$ sudo easy_install readline pyzmq ipython
```

5.4 IPython notebook

ipython notebook は Web ベースの数式処理システムで、次の機能を備えています。

- Python コードの実行と結果の表示
- Markdown によるマークアップ可能なノート
- MathJax による TEX 形式での数式の記述

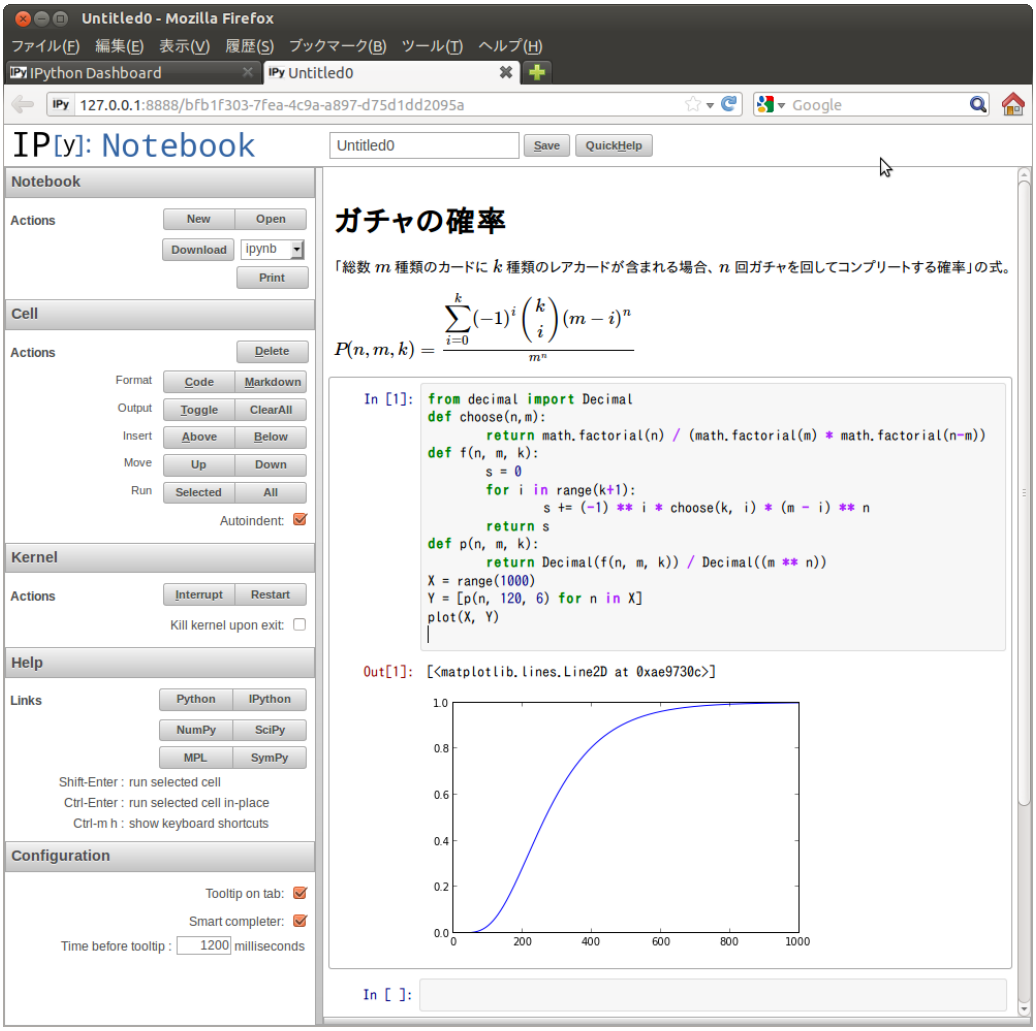


図 8　ipython notebook の使用例

testing には次のコマンドでインストール可能です。

```
$ sudo aptitude install ipython-notebook python-matplotlib python-tornado
```

squeeze は qtconsole と同様に backports を使用します。ここでは squeeze の iceweasel が古いためこちらも apt line に追加しています。

```
deb http://backports.debian.org/debian-backports squeeze-backports main
deb http://mozilla.debian.net/ squeeze-backports iceweasel-release
```

`aptitude update && aptitude upgrade` を実行した後、インストールは次のコマンドで行いました。

```
$ sudo aptitude install python-setuptools python2.6-dev ncurses-dev libzmq-dev python-pygments python-matplotlib
```

パッケージをインストールした後、 python の easy_install コマンドを使用して ipython をインストールします。

```
$ sudo easy_install readline pyzmq ipython tornado
```

またこのままでは MathJax が CDN を指しているので、 python を管理者権限で実行しローカルにインストールします。

```
from IPython.external.mathjax import install_mathjax
install_mathjax()
```

squeeze のウェブブラウザはどれも古く、 Web Socket の問題から notebook では使えません。そこで最新版の iceweasel を導入します。 ipython notebook は起動時にデフォルトブラウザを起動するため、導入後は iceweasel をデフォルトブラウザに指定してください。

```
$ sudo aptitude install -t squeeze-backports iceweasel
```

ipython notebook は次のコマンドで起動します。

```
$ ipython notebook --pylab inline
```

他のマシンから接続を行う場合、ブラウザを起動させないためオプション “--no-browser” を、アクセス許可を与えるため notebook を起動させるマシンの ip アドレスをオプション “--ip” で指定します。

```
$ ipython notebook --pylab inline --no-browser --ip 192.168.1.40
```

初回起動時には次のような画面が表示されます。

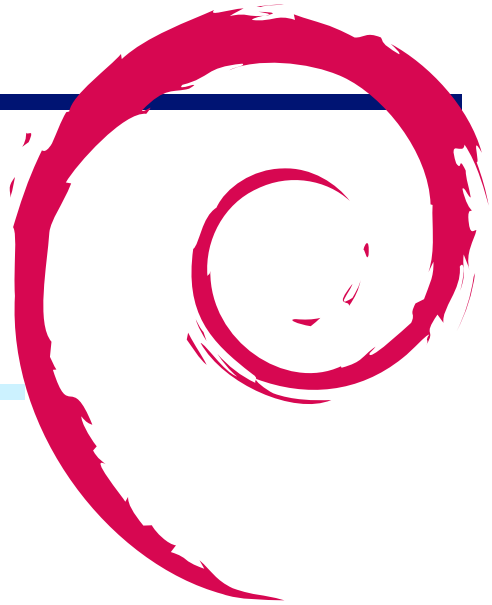
図 9　ipython notebook を起動した直後

「 New Notebook」ボタンをクリックすることで新規ノートブックが作成され、空のノートブックがブラウザに表示されます。 Markdown と Python の Cell は Ctrl-m m および Ctrl-m c で切り替えることができ、その他コマンドの詳細は Ctrl-m h で表示されます。

作成されたノートブックは表示されているディレクトリ (ここでは/home/hiro/) に “< ノートブックのタイトル >.ipynb” というファイル名で保存されます。このファイルの中身は JSON 形式となっています。

5.5　最後に

今回は ipython qtconsole および notebook を Debian で使用する方法を紹介しました。 Ubuntu 12.04 ではどちらもパッケージとして提供されているため、 wheezy ではインストール作業が簡単になると思われます。これを機会に ipython を活用していただければ幸いです。



6 Debian と LibreOffice

あわしろ いくや

6.1 自己紹介

6.2 OOo のあらすじ

- 1999/8/??…Sun Microsystems が StarOffice の開発企業を買収
- 2000/10/13…後に OOo となるソースコード公開
- 2002/5/1…OOo 1.0 リリース
- 2009/4/20…Oracle による Sun の買収を発表
- 2010/1/27…Oracle による買収完了
- 2010/6/4…OOo 3.2.1 リリース
- 2011/1/25…最終バージョンの 3.3.0 リリース

6.3 LibreOffice その１

- 2010/9/28…The Document Foundation と LibreOffice のリリースを発表
- OOo のコミュニティメンバーで結成
- OOo の商標の移譲を Oracle に求めてみたり
- 2011/1/25…LibreOffice 3.3.0 リリース
- 2011/6/3…LibreOffice 3.4.0 リリース
- 2012/2/14…LibreOffice 3.5.0 リリース
- 2012/2/20…TDF が財団に

6.4 LibreOffice その２

6.4.1 かなり普通の開発体制になった

- ソースコードは git で管理
- 契約書にサインしなくても push される
- ライセンスは LGPL と MPL
- 毎月リリース、半年に 1 度メジャーバージョンアップ
- 今は 8 月リリースに向けて 3.6 を開発中

6.5 Apache OpenOffice

- 2011/4/15…Oracle が OOo の開発中止を発表
- 2011/4/20…担当社員を全員解雇
- 2011/6/1…Apache への移管を発表
- 2011/6/13…Apache の Incubator プロジェクトに承認
- 2011/11/17…名称を ” Apache Openoffce ” に決定
- 2011/11/31…ライセンスを AL 2 に変更する作業完了
- 2012/5/8…AOO 3.4.0 リリース
- 2012/5/17…AOO 3.4.0 100 万ダウンロード
- 2012/5/21…Lotus Symphony のコード公開
- 2012/5/27…AOO 3.4.0 200 万ダウンロード

6.6 Apache OpenOffice **その２**

6.6.1 Lotus Symphony

- IBM が 2008/5/30 にリリース
- OOo と Eclipse をベース
- Sun から特別なライセンス
- ワープロ・ 表計算・ プレゼン・ Web ブラウザ
- Windows/Linux で動作
- 無償配布
- 2012/1/23…最終バージョンの 3.0.1 リリース

6.7 OOo meets Debian

6.7.1 **最初のアップロードは** 2001/10/23

- 2002/4/24…0.641d.cvs20020424-1 をアップロード
- 2002/5/1…OOo 1.0 リリース
- 2002/5/2…OOo 1.0 sid 入り
- 2002/7/11…OOo 1.0.1 sid 入り

6.7.2 **現在のメンテナンス体制**

- Rene Engelhard さんと Björn Michaelsen さん (Canonical) の二人体制
- とはいえ、パッケージに関しては Rene さんの作業量が多い
- Björn さんは upstream の作業が多い
- Björn さんはビルドのスペシャリスト、 Rene さんはパッケージングのスペシャリスト

6.8 LibOffice **のパッケージの背景**

- OOo 3.3.0 に関する作業が行われた形跡が全くない
- 以前はベータ版でも作業が行われていた
- Rene さんはかなり早い段階から TDF に誘われていたことがわかる
- 確かに Founder の一人になっている

http://www.documentfoundation.org/foundation/history/

- その時点で LibO が Debian に入ることは確定的だった
- LibO が sid で使えるようになったのが 2012/2/6
- 現在でも OpenOffice.org のパッケージはあるものの、 LibreOffice への移行用ダミーパッケージ

6.9 LibreOffice パッケージ

6.9.1 libreoffice-3.5.3 を例に

- ソースを取得後 df -h すると 3.2GB
- rules を wc -l すると 3146
- control を wc -l すると 3380
- changelog を wc -l すると 9763
- buildd でのビルド時間は 6 時間 40 分 (i386)
- 必要なディスクスペースは 17.04GB
- パッチは quilt で管理

6.9.2 Debian パッケージの注意点

- 得てしてオフィシャルのバイナリを前提とした説明をされる
- 拡張機能は一切インストールされないので、あとからインストールする。たいていはパッケージ化されている
- libreoffice-gnome/libreoffice-gtk3/libreoffice-kde のインストールを忘れない

6.10 LibOffice 関連パッケージ

1. ooohg
   Set of 1600 free of charge maps for libreoffice/openoffice.org
2. openclipart-libreoffice
   clip art for OpenOffice.org/LibreOffice gallery
3. writer2latex
   OpenOffice.org Writer/Calc to LaTeX/XHTML converter

6.11 翻訳のこと

6.11.1 皆さんにお願い

- discuss@ja.libreoffice.org を購読してね
- 翻訳の指摘(誤訳やわかりにくいものなど)
- 専門的な知識の教示
- 英語、日本語、ワープロ、表計算(特に関数) 、ドロー、 Windows、 Mac、 Linux、数式、データベース、PDF、印刷 etc...
- 間接的に Debian への貢献にもなりますよ!

6.12 AOO と Debian

6.12.1 6 月上旬現在、パッケージなし

- 怪しいリポジトリはある
  `http://apacheoo-deb.sourceforge.net/`
- オフィシャルビルドの Deb パッケージを apt で取れるようにしているだけ
- ちなみに launchpad にプロジェクトはあるけどパッケージはない
  `https://launchpad.net/~apacheopenoffice`

6.12.2 オフィシャルビルドの Deb パッケージ

- EPM で生成
  - ESP Package Manager
  - これは LibreOffice も一緒
  - 原作者は CUPS の作者
  - RPM とか Deb パッケージを生成するもの
  - すなわち、 debian フォルダがあるわけじゃない
  - どうやってもオフィシャルにならない

6.12.3 3 月に debian-users と debian-openoffice で AOO に関する投稿あり

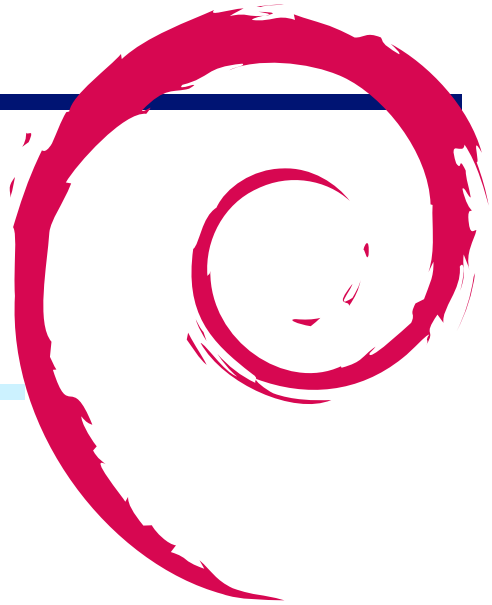
- `http://lists.debian.org/debian-user/2012/03/msg00824.html`
- `http://lists.debian.org/debian-openoffice/2012/03/msg00103.html`

いずれも Rene さんが激しく拒否

6.13 さらなる情報

6.13.1 主に歴史を知りたい人向け

1. 2011 年の OpenOffice.org/LibreOffice
   `http://gihyo.jp/lifestyle/column/newyear/2011/openoffice-prospect`
2. LibreOffice/Apache OpenOffice ～2011 年の総括と新たな選択～
   `http://gihyo.jp/lifestyle/column/newyear/2012/libreoffice-prospect`



7 debug.debian.net

岩松 信洋

7.1 はじめに

現在、 Debian Project で配布されているパッケージでは、デバッグ情報が削除された状態で配布されています。この理由として、デバッグ情報はほとんどのユーザには必要ないものであるという点と、デバッグ情報を保持している実行ファイルはサイズが非常に大きいため、ディスクを圧迫するという点があります。しかし、デバッグ情報があるとデバッグを行うときにとても有用な情報となりいろいろと便利です。今回、 Debian で全ての Debian パッケージにおいてデバッグ情報を提供する方法を考えて実装してみました。その課程と今後について説明します、

7.2 Debug 情報と Debian パッケージ

Debian はバイナリベースディストリビューションの一つです。基本的に配布されているパッケージではデバッグ情報が削除された状態で配布されています。このデバッグ情報は、内部シンボルや型の情報、ソースコードの行番号などを指します。

実行ファイルのデバッグ情報がある場合、以下のようなよい点があります。

- デバッガ(GDB) を使ったデバッグでより詳細なデバッグ情報を得ることができる
- デバッグ情報を含めたバイナリを再度ビルドする必要がない
- バイナリベースディストリビューションの場合、実際のバイナリとデバッグ情報が常に対になるので、バグの再現性が高くなる

逆に悪い点として以下のようなものがあります。

- 実行ファイルにデバッグ情報が含まれるので実行ファイルのサイズが大きくなる
- デバッグしない人にとっては不要なものが含まれることになる

これらをまとめると、すべての実行ファイルのデバッグ情報が提供されておりユーザにとって必要のない情報がインストールされない仕組みがあればデバッグ情報はとても有益なものになるはずです。 Debian ではいくつかのソースパッケージからデバッグ情報を含んだパッケージが提供されています。このパッケージには`-dbg` というサフィックスが付いてます。特にデバッグ情報用のパッケージに関するポリシーは決まっておらず、提供に関してはパッケージメンテナ次第という状態になっています。今まで提供されなかった理由としてはディスク容量の問題や回線の問題等があったようですが、個人的に今は特に問題はないと思っています。

ちなみに、 Fedora では `-debuginfo` というサフィックスを持ったパッケージが提供されており、 Gentoo ではデフォルトでこれらの情報を生成し管理しています。このようにデバッグ情報の提供という点に関して他のディストリビューションに遅れを取っています。

7.3 Debian でのデバッグ情報パッケージについて

まず、実装した内容について説明する前に今のデバッグ情報パッケージの提供方法について説明します。 Debian ではデバッグ情報パッケージは -dbg というサフィックスがついたパッケージ名を持ちます。例えば foo というアーキテクチャ依存のパッケージがあった場合、 foo のデバッグ情報を持ったパッケージ名は foo-dbg になります。そして、 Debian の -dbg パッケージで提供されているバイナリは動作するバイナリデータではなく、デバッグ情報のみを持ったデータになっています。 このファイルは objdump --only-keep-debug を実行することによって生成することができます。もちろん対象の実行ファイルはコンパイル時にデバッグ情報が付加されている必要があります。その後、デバッグ情報ファイルへのリンクを strip 済の実行可能形式に付加するために objcopy --add-gnu-debuglink を実行します。これによって、実行ファイルとデバッグ情報ファイルが対になります。 GDB を使ってデバッグする際には実行ファイルとデバッグ情報ファイルはリンクしているので、デバッグ情報ファイルがインストールされているときは自動的に呼ばれ、デバッグシンボルなどを読み込んでくれます。

7.4 実装について

先に説明したようにに、すべての実行ファイルのデバッグ情報が提供されておりユーザにとって必要のない情報がインストールされない仕組みがあればよいので、これらに対応できる方法を考えました。以下で説明します。

7.4.1 すべての実行ファイルのデバッグ情報を提供する

すべての実行ファイルのデバッグ情報を提供するには、全てのパッケージで strip された実行ファイルとデバッグ情報ファイルを持ったパッケージを構築すればよいわけです。

Debian の場合、パッケージはデバッグ情報が有効な状態(gcc だと-g オプション等）でビルドされます。そしてパッケージにされる時に strip （ binutils に含まれる）が dh_strip から呼ばれ、実行ファイルやライブラリならデバッグ情報が削除され、パッケージ用のディレクトリにコピーされ、 dh_builddeb コマンドでパッケージ化されます。

そして、配布される -dbg パッケージは dh_strip を実行するときにデバッグ情報を提供するパッケージとして、dh_strip のオプションとして指定されるか、 debian/control ファイルに列挙されているパッケージ名のサフィックに -dbg が付いている場合、対象ファイルとして処理されます。

ここ問題なのが、

1. 自動生成したい デバッグ情報パッケージ情報をどのように生成するか
2. dh_strip でデバッグ情報パッケージ指定されている場合、自動生成したい -dbg パッケージ用のファイルをどのように生成するか
3. 自動生成したいデバッグ情報パッケージそのものをどのように生成するか

という点です。

問題点 1 についての対処方法ですが、 dh_strip の処理の先頭で debian/control ファイルにデバッグ情報パッケージファイルに関する情報を追記する処理を追加しました。デバッグ情報パッケージはそのパッケージがアーキテクチャ依存（ Architechture: all ではない）事とパッケージ名さえわかれば、パッケージ情報は自動生成できます。例えば、 hoge というパッケージがあってデバッグ情報パッケージが提供されていない場合、以下のような内容を追記します。

```
Package: hoge-dbg
Architecture: any
Section: debug
Priority: extra
Depends: hoge (= \${binary:Version}), \${misc:Depends}
Description: debugging symbols for hoge
 This package contains the debugging symbols for hoge
```

次に問題点 2 の対処方法ですが、アーキテクチャ依存の Debian パッケージは dh_strip が呼ばれるため、ここで処理をフックしてしまえば、デバッグ情報を提供するパッケージと用のデータと strip されたバイナリデータを分けることが

できます。今回は dh_strip の中身を改造し、全てのアーキテクチャ依存のパッケージ用のデータを作成することにしました。 dh_strip でパッケージが指定されていてもそれを無視するように処理を変更するだけです。

問題点 3 の対処方法ですが、デバッグ情報パッケージは dh_strip 内で dh_builddeb を呼び出すことで対応しました。パッケージ名とパッケージ作成に必要なデータは揃っているので、 dh_builddeb -p **デバッグ情報パッケージ名** を実行することで、パッケージが作成されます。

これらが行える前提条件として、 debhelper に依存しているパッケージが対象になります。現在ほとんどのパッケージが debhelper か CDBS に依存しており、 CDBS は debhelper と同時に使う事が多いため(実際、 CDBS に依存しているパッケージは全て debhelper に依存しています。 dbs も同様です。) 問題ではありません。[*20]

7.5 パッケージサイズへの対応

次にパッケージサイズの問題です。デバッグ情報は非常に大きく、 strip されたバイナリの数倍以上のサイズになることはめずらしくありません。例えば libjpeg8 で提供される libjpeg.so.8.4.0 のファイルサイズは表 1 となりました。

表 1　libjpeg.so.8.4.0 各状態のファイルサイズ

状態	サイズ
strip 前	約 1.3MB
strip 後	236KB
デバッグ情報ファイル	1.1MB

このようにサイズが大きく異なるので、パッケージを分けてもユーザが利用しているパッケージリポジトリと同じ場所・方法で提供してしまうと、ミラーに時間がかかるようになりますしユーザに不要なデータが格納されたリポジトリ情報を持たせるようになります。

今回、この問題を回避するためにリポジトリを分けることを考えました。例えば、 unstable で strip されているパッケージ(通常のパッケージ) は unstable とだけ指定し、デバッグ情報を提供するパッケージは unstable/debug するという方法です。例を図 10 に示します。

```
deb http://cdn.debian.or.jp/debian/ unstable main non-free
deb http://cdn.debian.or.jp/debian/ unstable/debug main
```

図 10　デバッグ情報パッケージを利用する場合の apt-line 設定例

デバッグ情報が必要なユーザは unstable/debug を apt-line に追加することによってデバッグ情報用のパッケージが利用できるようになります。またパッケージが格納されるディレクトリのパスを変更することによって、デバッグ情報のみを提供するミラーを構築することができ、 debug 情報をミラーしないミラーサーバの負荷も今までと変わらないという事になります。

7.5.1 実装後について

これらを実装したシステムを reprepro + sbuild + rebuildd で構築しました。現在 stable/amd64 のみをターゲットテストとして動作させています。まだ全てビルドできていません。さくらの VPS で 1 週間ほどビルドしていますが、まだ gnome-power-manager をビルドしているところで、進捗率は 40% といったところでしょうか。

[*20] http://people.debian.org/~cjwatson/dhstats.png

7.6 考えられる問題

7.6.1 セキュリティの問題

現在、ソースパッケージから作成されたバイナリパッケージの一覧は .changes ファイルにファイルのハッシュと共に記述され、どのソースパッケージ(orig.tar.gz, dsc.diff.gz) から作成されたのか分かるようになっています。現時点での実装はバイナリパッケージ作成の課程で自動生成されるためのこれらの情報とリンクしません。 Buildd 上でデバッグ情報ファイルパッケージが生成されるので個人的に問題ないと思っていますが、これらを紐付けるシステムがあるほうがより安全と言えるでしょう。

7.6.2 バイナリの不一致

既存のシステムだと、 `strip` されたバイナリと デバッグ情報が一致しないのでオフィシャルのバイナリと混ぜて使えないことが考えられます。私が提供しているされているデバッグ情報パッケージを使う場合、私が提供してる通常のパッケージも利用しないと意味がないでしょう。今はこれをバージョンによる依存関係で回避しています。最終的には buildd に入れてもらうことでデバッグ情報パッケージを自動生成することを考えています。

7.7 そもそも debug.debian.net があるんじゃね? という話

さて、私のような凡人が考えるようなことは先人達はすでに考えているわけでして、(既に行なっていることを知ったのは大統一勉強会スケジュールが出てからなのですが。) 既に `http://debug.debian.net` というサービスがあり実装され、そして終了していました。実装と考えもほとんど同じです。大きく違うところはデバッグ情報パッケージのサフィックスが`-dbg` ではなく、`-dbgsym` である点と、デバッグ情報を作成する部分が `dh_builddeb` ではなく、 `dpkg-deb` を使っている点、そして `dh_strip` を直接変更するのではなく、シンボリックリンクで機能をオーバーライドさせている点です。こちらの方が debhelper に手を加えなくて済むのでこちらに乗り換えて、いくつかの修正を行いました。元々このサービスは myon [*21] がやっていたのですが、彼に連絡を取り、再稼働させることにしました。この発表が行われている頃には稼働していると思います。

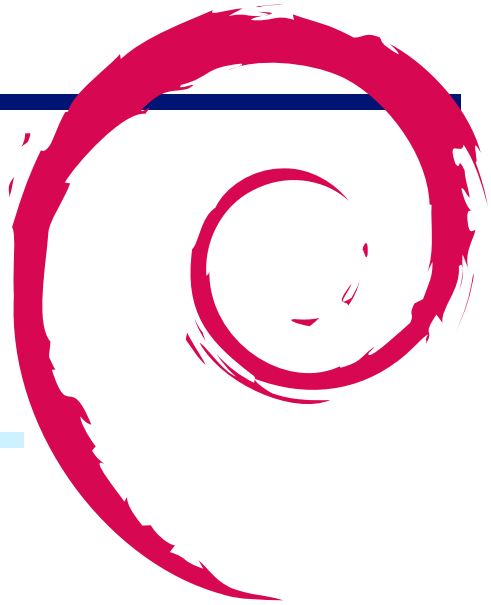
7.8 今後の課題

このままスタンドアロンでデバッグ情報パッケージを提供しても無駄なバイナリを生成するだけなので、 buildd にこのシステムを入れてもらい、 dak で ディストリビューション/debug として処理してもらうことが今後の大きな課題になっています。このことについて来月開催される Debconf 12 で BOF または FTP チーム、 wanna-buildd チームと話ができればと思っています。

7.9 最後に

アーキテクチャに依存している Debian パッケージでデバッグ情報をパッケージとして提供する方法について説明しました。文中ではユーザにはあまり必要のない機構のように取り扱いましたが、バグレポートをする際、デバッグ情報があるとより内容の濃いバグレポートを提出できるようになり、開発者を手助けできるようになります。この機構が正式採用された時に、自分が使っているパッケージのデバッグ情報パッケージだけでもインストールしてくれれば幸いです。

[*21] Christoph Berg 氏。 DAM の一人。



8 Rabbit: 時間内に終われるプレゼンツール

須藤功平

Debian GNU/Linux 上で動作するプレゼンテーション ツール Rabbit を紹介します。まず、 Rabbit の代表的な機能である時間内に終われるための機能を紹介し、その後、 Debian GNU/Linux 上で Rabbit をインストールする方法とスライドを作る方法を簡単に紹介します。

8.1 Rabbit とは

Rabbit は Ruby と GTK+ で実装されたプレゼンテーション ツールです。 Debian GNU/Linux を含む多くのプラットフォーム上で動作します。

Debian GNU/Linux 上で動作するプレゼンテーション ツールはたくさんあります。

- GUI でスライドを作成するデスクトップ アプリケーション（ LibreOffice の Impress など）
- テキストで作成したスライドを表示するデスクトップ アプリケーション（ MagicPoint や Rabbit など）
- LaTeX の Beamer クラス + PDF ビューアー（ Evince や pdfcube など）
- JavaScript + Web ブラウザ（ Impress.js や showoff など）
- Web サービス（ Google Docs や Prezi など）

それぞれのツールは特徴が大きく異なっており、それぞれよいところがあります。 Rabbit にもまた特徴があり、他のツールにはない便利な機能があります。それが「 プレゼンテーション を時間内に終われるための機能」です。この機能は一般的にタイマー機能と呼ばれているもので、プレゼンテーション 中に残り時間を表示して、発表者に進み具合を伝える機能です。多くのツールでこの機能を提供していますが、 Rabbit のタイマー機能はどう違うのでしょうか。一言でいうと UI（ ユーザーインターフェイス。見た目）が違います。他は同じです。

8.1.1 Rabbit の UI

タイマー機能は MagicPoint も提供していますし、 Impress も提供しています。図 11 のように、 MagicPoint はスライド下部にとても目立たないように緑のバーを表示します。 Impress はスライド表示用のモニターとは別のモニターに経過時間を表示します。このように発表者にだけわかるように表示する UI が一般的なタイマー機能の UI です。

一方、 Rabbit の UI は発表者だけではなく観客にもわかりやすく表示します。図 12 のように、スライド下部にうさぎとかめを表示します。それも誰が見ても気づくくらいの大きさで表示します。このようにタイマー機能を観客からもわかりやすく表示する UI は既存のプレゼンテーション ツールとは一線を画します。

それでは、この UI がどうして時間内に終わるための効果を提供するかを考えてみましょう。

図 11　従来のタイマー機能の UI

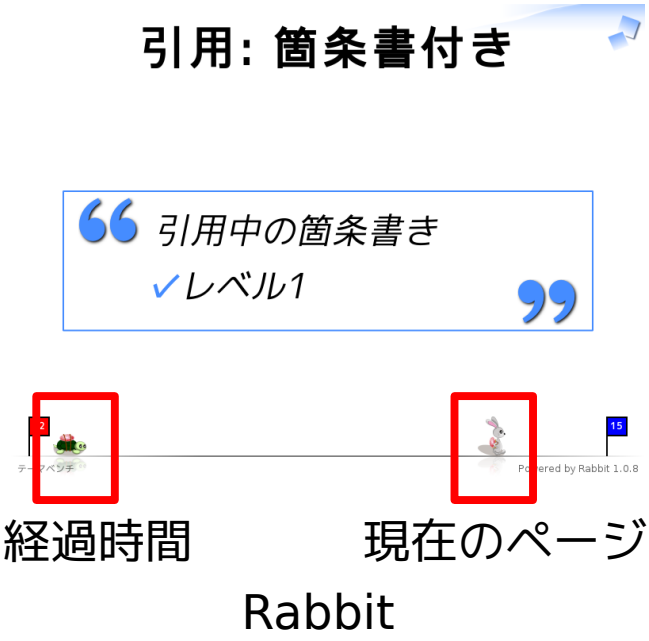


図 12　Rabbit のタイマー機能の UI

8.1.2　みんなにわかる UI

この Rabbit の UI の特徴は発表者だけではなく、観客にも発表の進み具合がわかりやすいという点です。発表時間を過ぎれば経過時間を表しているかめが画面の右側に走りすぎていきます。もちろん、かめなので徐々に画面の右側に進んでいきます。そのため、観客が気づかないうちに発表時間を過ぎていたということは起きません。発表時間を過ぎても終わらないと、たとえどれだけ魅力的な話であっても気まずい空気になってきます[*22]。

発表者は観客の反応がよいとよりすばらしい発表ができますが、逆に反応が悪いと、準備してきた成果を発揮しづらいものです。発表時間が過ぎても終わらない場合、観客の反応が悪くなります。この状態でさらに続けることはとてもつらいため自然と終わらせようという力が働きます。

このように、タイマー機能を発表者だけではなく観客からもわかりやすい UI にすることにより、観客から残り時間に関

[*22] 要出典。

するフィードバックを受け取ることができます。発表者が残り時間を意識するだけではなく、観客からのフィードバックもあわせることで自然と時間内に終われる発表になります。ただし、観客にわかりやすいタイマー機能の UI なので、発表内容よりもうさぎとかめの競争に集中してしまい、肝心の発表内容に集中してもらえないという危険性があります。時間内に終わることだけではなく、魅力的な発表内容を用意することにも十分注意してください。

8.2 インストール方法

ここまでの紹介で Rabbit を使いたくなっているはずです。ここからは Rabbit の使い方を紹介します。

まず、 Rabbit を Debian GNU/Linux へインストールする方法を紹介します。インストール方法は 2 種類あります。

- apt を使う
- RubyGems を使う

最新バージョンを使いたい場合やフル機能を使いたい場合は RubyGems を使う方法がオススメです。簡単にインストールしたい場合は apt を使う方法がオススメです。

8.2.1 apt でインストール

1 つ目の方法は、 apt を使う方法です。 Rabbit の deb パッケージは公式 apt リポジトリに含まれている[*23]ので以下のようにすれば簡単にインストールできます。

```
$ sudo apt-get -V -y install rabbit
```

以下のコマンドを実行してスライドが表示されたら正常にインストールできています。

```
$ rabbit https://raw.github.com/shockers/rabbit/master/sample/theme-bench.rab
```

8.2.2 RubyGems でインストール

2 つ目の方法は RubyGems を使う方法です。 Rabbit は GTK+ などたくさんのライブラリを利用しています。そのため、まず、関連ライブラリを apt でインストールします。

```
$ sudo apt-get -V -y install \
    ruby1.9.1 ruby1.9.1-dev libgtk2.0-dev librsvg2-dev libpoppler-glib-dev \
    libxml2-dev libxslt1-dev
```

関連ライブラリをインストールしたら RubyGems で Rabbit をインストールします。

```
$ sudo gem1.9.1 install rabbit twitter-stream twitter_oauth
```

以下のコマンドを実行してスライドが表示されたら正常にインストールできています。

```
$ PATH="/var/lib/gems/1.9.1/bin:$PATH"
$ rabbit https://raw.github.com/shockers/rabbit/master/sample/theme-bench.rab
```

8.3 スライドの作り方

Rabbit を使う準備ができたので自分でスライドを作る方法を紹介します。

Rabbit は MagicPoint のようにテキストでスライドを作成します。テキストのフォーマットには RD[*24]や Wiki 形式、 Markdown などいくつもの有名なフォーマットをサポートしています。ここでは、 RD でのスライドの書き方を紹介します。

[*23] 大統一 Debian 勉強会の実行委員でもある佐々木洋平さんがパッケージメンテナー。

[*24] Ruby Document の略。

8.3.1 RD でのスライドの作り方

RD でスライドを作る場合、以下のように「 =」でスライドを区切ります。「 =」の行に書いているテキストがそのスライドのタイトルになります。「 #」から始まる行はコメントです。シンプルですね。

```
# slide.rab

= タイトルスライド

# 持ち時間
: allotted-time
   5m

= 最初のスライド

  * 1 枚目のスライドの内容

= 2 枚目のスライド

  * 2 枚目のスライドの内容
```

最初のスライドは特別でタイトルスライドになります。ここにはスライド全体のメタデータを指定することができます。「 allotted-time」というのはこのプレゼンテーションの持ち時間で、「 5m」は 5 分[*25]という意味です。持ち時間を指定するとうさぎとかめタイマーが表示されるので、持ち時間が決まっているプレゼンテーションのときは指定しましょう。

作成したスライドは以下のように実行します。 2 ページ目を表示するとうさぎとかめタイマーが動きだします。

```
$ rabbit slide.rab
```

8.3.2 PDF で作成したスライドの表示方法

Rabbit はスライドを表示する機能だけを提供しているため、スライド作成を支援する機能はありません。 Debian GNU/Linux を使っている人はエディターでテキストを編集することに慣れているでしょうが、たまには Impress などを使って GUI でグラフィカルにスライドを作成したくなるかもしれません。そのときはそのようなツールでスライドを作成してください。 Rabbit はテキストだけではなく PDF ファイルも読むこむことができます。 Impress などでスライドを作成し PDF で出力すれば、 Rabbit で表示することができます。

PDF を表示するときはコマンドラインから持ち時間を指定します。「 --allotted-time 5m」と指定すると持ち時間が 5 分という意味になります。 2 ページ目を表示するとうさぎとかめタイマーが動きだします。

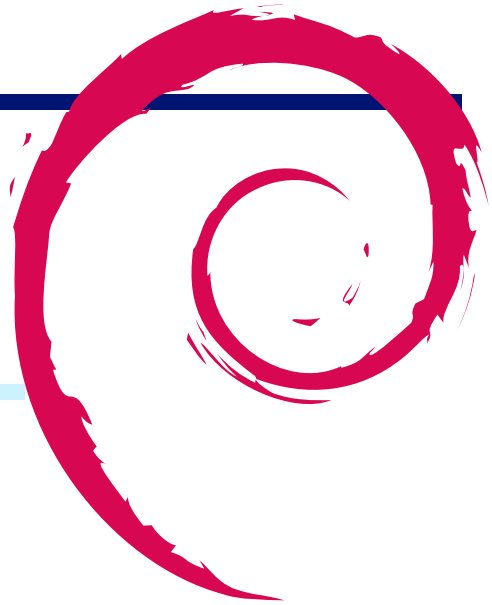
```
$ rabbit --allotted-time 5m slide.pdf
```

8.4 さいごに

Debian GNU/Linux 上で動作する時間内に終われるプレゼンテーションツール Rabbit を紹介しました。 Debian GNU/Linux でプレゼンテーションをする人たちの参考になることを期待しています。

もっと Rabbit を知りたくなった人は公式サイト http://rabbit-shockers.org/ をのぞいてみてください。

[*25] 5 Minutes



9 U-Boot についてあれこれ

野島 貴英

Debian は組み込み用途の開発にも非常に便利の良い開発環境を作る事ができます。ここでは、組み込み用途の開発では最も基本的なソフトウェアであるブートローダを Debian の環境を用いて U-Boot を用いて開発してみます。最後に、電子書籍端末にブートローダを実際に搭載して起動を試みます。

9.1 U-Boot とは

U-Boot とは、 DENX Software Engineering 社(本家) で主にメンテされているブートローダです。 Das U-Boot ともいう[*26]そうです。

元は PowerPC 用途のブートローダだった模様ですが、様々な人が拡張を加え、現在では 216 種類の組み込み機材に対応しており、さらに対応数は増えつづけてている模様です [1]。

9.2 U-Boot の特徴

以下に示す沢山の特徴があります。

- ソフトウェアライセンスは GPL v2 です
- 沢山の CPU、組み込み機材に対応しています。例: PowerPC,ARM,AVR32,Blackfin,m68k,x86,...
- U-Boot のソース本体は単機能の部品の塊であり、ディレクトリも十分に整理され、マクロも整理されている為、開発対象の機材に合わせた改造をするにも、直感的で非常にわかりやすいです
- U-Boot 独自仕様のちょっとしたスクリプト言語を使う事により、ブートに関して機材に合わせた制御が可能です。また、対話的に OS をブートさせる事ができます
- tftp 経由の OS ブート、シリアル経由の OS ブート、 NFS 経由の OS ブート、数々の種類の Flash メモリからの OS ブート等、様々な方法で OS をブートすることが可能です
- OS イメージを様々な形式のファイルシステム上に置いて起動させることができます。例: fat/vfat 形式、 ext2/3 形式、 cramfs 形式
- ELF/バイナリ形式/圧縮形式で出来ている OS のイメージに対応しています

9.3 試しに Debian 上で動かしてみる

まず、 Intel 64bit 対応版 Debian sid の動作する PC を用意します。

ここで、端末を開き(端末 A) 、 experimental 版の QEMU を導入します。

[*26] ドイツ語で潜水艦の意。この場合直訳すると「ザ・潜水艦」と言う感じです。

```
端末 A $ cat /etc/apt/sources.list
deb http://ftp.jp.debian.org/debian/ sid main contrib non-free
deb-src http://ftp.jp.debian.org/debian/ sid main contrib non-free
deb http://ftp.jp.debian.org/debian/ experimental main contrib non-free
deb-src http://ftp.jp.debian.org/debian/ experimental main contrib non-free
端末 A $ sudo apt-get update
端末 A $ sudo apt-get install qemu-system/experimental  <--experimental 版導入
```

次に ARM 用のクロスコンパイル環境を用意します。ここでは Emdebian を利用することにします。なお、現在、Emdebian のパッケージは安定版の Debian でないと導入ができないものが多い (特に gcc) ため、クロスコンパイル環境として安定版の Debian の chroot 環境を用意して導入することにします。

```
端末 A $ mkdir cross-compile
端末 A $ sudo debootstrap --arch amd64 --include emdebian-archive-keyring,\
        sudo,lv,build-essential,binutils-multiarch squeeze \
        ‘pwd‘/cross-compile http://ftp.jp.debian.org/debian
 ... しばらく待つ...
端末 A $ env LANG=C sudo chroot ‘pwd‘/cross-compile /bin/bash
（ここからは安定版の Debian となる）
端末 A # echo ’deb http://www.emdebian.org/debian/ squeeze main’ >> /etc/apt/sources.list
端末 A # echo ’deb-src http://www.emdebian.org/debian/ squeeze main’ >> /etc/apt/sources.list
端末 A # apt-get update
端末 A # groupadd -g <your-gid> <your-id>
端末 A # useradd -m -s /bin/bash -u <your-uid> -g <your-gid> -c ’your name’ <your-id>
端末 A # passwd <your-id>
端末 A # echo ’127.0.0.1  localhost <your-hostname>’ > /etc/hosts
端末 A # echo ’<your-id> ALL=(ALL) NOPASSWD: ALL’ > /etc/sudoers.d/<your-id>
端末 A # chmod 440 /etc/sudoers.d/<your-id>
端末 A # su - <your-id>
端末 A % sudo apt-get install gcc-4.4-arm-linux-gnueabi
```

次に別の端末(端末 B) を sid 側で開き、 u-boot のソースコードを端末 A のディレクトリへ直接ダウンロードします。

```
端末 B $ cd cross-compile/home/<your-id>/u-boot
端末 B $ apt-get source u-boot <---unstable 版の u-boot のソースがダウンロードされる。
```

端末 A にて ARM versatilepb QEMU 用にクロスコンパイルします [*27]。

```
端末 A $ cd u-boot/u-boot-2012.04.01
端末 A $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- versatileqemu_config
端末 A $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
```

出来上がった u-boot.bin を端末 B にて、 QEMU を使って ARM versatilepb 環境で動作させます [2]。

```
端末 B $ qemu-system-arm -M versatilepb -nographic -kernel u-boot.bin

U-Boot 2012.04.01 (Jun 05 2012 - 22:34:41)

DRAM:  128 MiB
WARNING: Caches not enabled
Using default environment

In:    serial
Out:   serial
Err:   serial
Net:   SMC91111-0

VersatilePB # printenv
baudrate=38400
... 中略...
```

無事 U-Boot が動作しています。 U-Boot のコマンドラインをいろいろいじってみると色々分かってよいかと思います。もちろんですが、$(src)/include/configs/versatile.h や、$(src)/boards/armltd/verrsatile 以下をいじってみていろいろ試すのも非常に簡単にできてしまいますので、大変おすすめです。

9.4 実機で動かしてみる

実機として、 Barnes & Noble 社 (以下 B&N 社) の Nook Color 端末を使います。

[*27] versatilepb 用の u-boot.bin イメージは Debian ではパッケージ化されていません

9.4.1 下準備

東京エリア Debian 勉強会 2012 年 4 月資料の「 Android 機で Debian」を参照して、 B&N 社の Nook Color 端末でブート可能な mini-SD カードを作成しておきます [3]。

B&N 社より配布されている U-Boot のソースに簡単なパッチを当てて、 U-Boot の printf() などが LCD へ表示が出来るようにしてしまいます。また、ついでに、 LCD のテストということでカラーバーを表示するようにしてしまいます。

※先ほどの端末 A、端末 B を環境として使います。

```
端末 B $ cat u-boot-nook-patch.patch
diff -ru u-boot/common/lcd.c u-boot-nozzy/common/lcd.c
--- u-boot/common/lcd.c 2011-04-20 19:19:16.000000000 +0000
+++ u-boot-nozzy/common/lcd.c   2012-06-05 18:26:47.000000000 +0000
@@ -363,11 +363,11 @@

        strcpy (lcddev.name, "lcd");
        lcddev.ext   = 0;                       /* No extensions */
-#ifdef CONFIG_3621EVT1A
-    lcddev.flags = 0; /* Use only for splash */
-#else
+// #ifdef CONFIG_3621EVT1A
+//    lcddev.flags = 0; /* Use only for splash */
+// #else
        lcddev.flags = DEV_FLAGS_OUTPUT;        /* Output only */
-#endif
+// #endif
        lcddev.putc  = lcd_putc;                /* 'putc' function */
        lcddev.puts  = lcd_puts;                /* 'puts' function */

diff -ru u-boot/include/configs/omap3621_evt1a.h u-boot-nozzy/include/configs/om
ap3621_evt1a.h
--- u-boot/include/configs/omap3621_evt1a.h     2011-04-20 19:19:16.000000000 +0
000
+++ u-boot-nozzy/include/configs/omap3621_evt1a.h       2012-06-05 19:15:52.0000
00000 +0000
@@ -211,6 +211,12 @@
 // Recovery mode boot commands
 //  androidboot.hardware=omapzoom2
 #define CONFIG_EXTRA_ENV_SETTINGS              \
+"stdout=lcd" \
+       "\0" \
+\
+"stderr=lcd" \
+       "\0" \
+\
 "mmcbootdev=0" \
        "\0" \
 \
@@ -556,7 +562,11 @@
 /* Enable LCD driver support for Encore device */
 #define CONFIG_LCD  1
 #define CONFIG_LCD_LOGO 1
-#define CONFIG_LCD_NOT_ENABLED_AT_INIT
+// #define CONFIG_LCD_NOT_ENABLED_AT_INIT
+#define LCD_TEST_PATTERN
+
+/* enable console to lcd */
+#define CFG_CONSOLE_IS_IN_ENV

 /* define early driver board init to allow board initialization after i2c*/
 #define CONFIG_DRV_BOARD_INIT 1
```

上のパッチをあてます。

```
端末 B $ wget http://images.barnesandnoble.com/PResources/download/Nook/source-code/nookcolor_1.4.tgz
端末 B $ tar xzf nookcolor_1.4.tgz
端末 B $ cd nookcolor-1.4/distro/u-boot
端末 B $ patch -p1 < ../../../u-boot-nook-patch.patch
```

パッチをあてた U-Boot のソースをクロスコンパイルして u-boot.bin を作ります [4]。

```
端末 A $ cd nookcolor-1.4/distro/u-boot
端末 A $ make -j2 ARCH=arm  CROSS_COMPILE=arm-linux-gnueabi- omap3621_evt1a_config
端末 A $ make -j2 ARCH=arm  CROSS_COMPILE=arm-linux-gnueabi-
```

出来た u-boot.bin を 9.4.1 章で作成しておいた mini-SD 上の u-boot.bin と入れ替えます。これで準備完了です。

9.4.2 起動する

いよいよ mini-SD を Nook Color 端末に挿入し、電源ボタンを長押しします。無事、図 13 のとおり U-Boot の起動画面が LCD に出ました。

図 13　U-Boot が起動している様子

なお、ここで紹介する U-Boot の改造は、まだ不十分なため、続く uImage を mini-SD からロードしてカーネルを実際に立ち上げるところまでは出来ていません。しかしながら、 U-Boot の printf() に記載した結果は LCD に出力できているので、こちらを元に U-Boot の改造を続ける事ができそうです。

9.5 終わりに

今回ブートローダである U-Boot を使い、 Debian 上でクロスコンパイル環境を作って実際に U-Boot が起動するまで試してみました。 PC ほどには BIOS も無い、デバッグ用ポートも無い実環境で、大した改造や試行錯誤もなく、いきなり LCD に printf() 出来るのは U-Boot がよくできている証拠だと思います。また、適当な実環境が無くても、 Debian さえあれば組み込み機対応のハックを欲望のままに試すこともできます。

昨今 Android 端末の普及のおかげで、ものすごい勢いで巷に「不自由」なソフトウェア環境で固められた ARM CPU 搭載の情報端末があふれています。これをきっかけに Debian を使ってこれらの端末を「自由」なソフトウェア環境へ変えてみませんか?

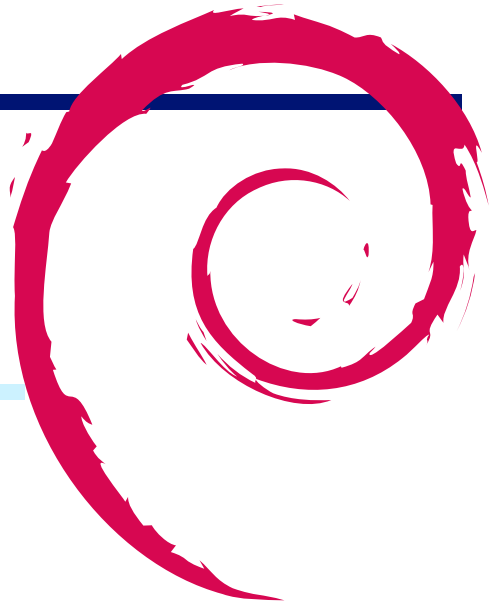
参考文献

[1] U-bootdoc 1.2 History, http://www.denx.de/wiki/view/U-Bootdoc/History

[2] Virtual Development Board, http://www.elinux.org/Virtual_Development_Board

[3] 東京エリア Debian 勉強会 2012 年 4 月資料, http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume201204.pdf

[4] NookColor: Build the Original Kernel, http://nookdevs.com/NookColor:_Build_the_Original_Kernel



10 Debian Multiarch Support

なかおけいすけ

10.1 はじめに

Debian Multiarch Support は、同じシステム上で、複数のアーキティクチャーのライブラリやプログラムをインストールおよび実行するしくみです。またエミュレータや cross-build 環境も提供されており、 amd64 と armel のように大きく異なったアーキティクチャのプログラムも動かすことができます。

たとえば 64bit のマシンで、あるプログラムを動かしたいのに 32bit 版しかないことがあります。 multiarch では、そのプログラムが依存している 32bit 版のライブラリパッケージをインストールすることができるので、 32bit 版のプログラムを 64bit 環境で動かすことができます。また、 x86-64 のデスクトップで ARM のソフトウェアを開発したいときは、armel 版の build-dependent ライブラリをインストールすれば、 x86-64 のデスクトップで開発することができますし、qemu を使えば、テストすることもできます。

ただ注意しなければいけないのは、複数のアーキティクチャが共存できるのは**ライブラリパッケージだけ**で、**プログラムは共存できない**ということです。

10.2 つかいかた

Multiarch を使うには、インストールしたい他のアーキティクチャを指定します。これをセカンドアーキティクチャと呼びます。セカンドアーキティクチャを指定した後、必ずパッケージデータベースを更新してください。例えば amd64 で動いているマシンに、 i386 のパッケージをインストールしたい場合は以下のようになります。

```
# dpkg --print-architecture  # どのアーキティクチャーでマシンが動いているか表示
amd64
# dpkg --add-architecture i386 # i386 を追加
# dpkg --print-foreign-architectures
i386
# apt-get update # パッケージデータベースを更新
```

ここで、 aptitude コマンドでパッケージのリストを表示すると、パッケージ名の後ろに、:i386 と表示されます。たとえば libc6:i386 であれば、 i386 用の libc6 パッケージという意味です。

```
$ aptitude search libc6
i   libc6                          - 組込用 GNU C ライブラリ: 共有ライブラリ
p   libc6:i386                     - 組込用 GNU C ライブラリ: 共有ライブラリ
p   libc6-amd64:i386               - 組込用 GNU C ライブラリ: AMD64 用 64 ビッ
p   libc6-dbg                      - 組込用 GNU C ライブラリ: 分離したデバッグ
p   libc6-dbg:i386                 - 組込用 GNU C ライブラリ: 分離したデバッグ
....
```

それでは、 i386 版の libc6 パッケージをインストールしてみましょう。他のアーキティクチャーのパッケージをインストールするには、

```
# apt-get install package:architecture
```

と指定します。よって i386 用の libc6 をインストールしたい場合は、以下のようになります。

```
# apt-get install libc6:i386
```

それでは、 i386 のプログラムが動くか試してみましょう。 i386 アーキティクチャで動いているマシンで、以下のプログラムをコンパイルします。

```
$ uname --machine
i686

$ cat hello.c
#include <stdio.h>
#include <sys/utsname.h>

int main(void)
{
        struct utsname u;
        if(-1 == uname(&u)){
                perror("uname");
                return -1;
        }

        printf("Hello World on %s\n", u.machine);
        return 0;
}

$ gcc -o hello-i686 hello.c

$ file hello-i686
hello-i686: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs),
for GNU/Linux 2.6.18, not stripped

$ ./hello-i686
Hello World on i686
```

32bit ELF のバイナリができて、 i386 アーキティクチャの上で動くプログラムということがわかります。これを、libc6:i386 がインストールされた amd64 で動いているマシンににコピーして、実行できるか試してみましょう。

```
$ uname --machine
x86_64

$ ./hello-i686
Hello World on x86_64

$ file hello-i686
hello-i686: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs),
for GNU/Linux 2.6.18, BuildID[sha1]=0x2898c6a77a71f4ae529ae4fb7f91beff44f6762e, not stripped
```

すばらしい。 hello-i686 は、 ELF 32bit バイナリにもかかわらず amd64 のマシンでちゃんと動いています。でもなぜこんなことができるのでしょう。

10.3 Mulit-Arch のしくみ

Linux は、実行ファイルフォーマットに、 ELF(Executable and Linkable Format) を採用しています。あるプログラムが実行されるとき、まず実行ファイルのヘッダにある、 PT_INTERP で指定されている ELF インタープリタがロードされます。 ELF インタープリタは、プログラムの実行に必要なダイナミックライブラリをロードします。このとき、 glibc は、-rpath や、環境変数 LD_LIBRARY_PATH、ハードコードされたディレクトリなどをまわって必要なライブラリを探します。

multiarch は、システム上で利用可能なたくさんのライブラリの中から最適なものを自動的に選択する、このしくみを使って実現されています。すなわち、アーキティクチャ毎に決まったディレクトリにライブラリをインストールして、glibc が検索するディレクトリのリストにそれらのディレクトリを追加しておけば、自動的に必要なライブラリを選んでくれるというわけです。

たとえば、 libc6:i386 パッケージの中身を見てみると、共有ライブラリが/lib ではなく、/lib/i386-linux/gnu にインストールされていることがわかります。 multiarch に対応したライブラリパッケージは、以前のように問答無用で /lib や/usr/lib に入れるのではなく、/lib/i386-linux-gnu や、/usr/lib/i386-linux-gnu にインストールされていることがわかります。

```
$ dpkg -L libc6:i386
/.
/lib/i386-linux-gnu/libnss\_nis-2.13.so
/lib/i386-linux-gnu/libpthread-2.13.so
....(snip)

/etc/ld.so.conf.d
/etc/ld.so.conf.d/i486-linux-gnu.conf
....(snip)

/usr/lib/i386-linux-gnu/gconv/EUC-JISX0213.so
/usr/lib/i386-linux-gnu/gconv/KOI8-T.so
/usr/lib/i386-linux-gnu/gconv/IBM1144.so
....(snip)

/lib/ld-linux.so.2
/lib/i386-linux-gnu/libnss\_nis.so.2
/lib/i386-linux-gnu/libthread_db.so.1
....(snip)
```

先ほどの hello-i686 がロードしているライブラリを調べてみると、以下のようになり、実行時にリンクされる共有ライブラリが、`/lib/i386-linux-gnu` などからロードされていることがわかります。

```
$ ldd hello-i686
        linux-gate.so.1 =>  (0xf76e3000)
        libc.so.6 => /lib/i386-linux-gnu/i686/cmov/libc.so.6 (0xf756d000)
        /lib/ld-linux.so.2 (0xf76e4000)
```

10.4 共有ライブラリパッケージを、Multiarch 対応にする方法

10.4.1 ライブラリをインストールするディレクトリ

multiarch 対応パッケージへの変換方法は、Debian Wiki の Using multiarch[*28] に詳しく述べられています。Debian Policy (9.1.1)[*29] で、multiarch に関することは共有ライブラリの PATH しか定義されていません。一方で (autoconf のような) ほとんどのアップストリームのビルドシステムは、共有ライブラリや、サポートファイル、(static ライブラリや.so ファイルのシンボリックリンクのような) 開発に使うファイルを同じターゲットディレクトリにインストールしようとします。Debian Policy はこのようなファイルをすべて、/usr/lib/*triplet* サブディレクトリにインストールできるように変更される予定です。ここで *triplet* とは、`dpkg-architecture -qDEB_HOST_MULTIARCH` コマンドが返す値のことです。multiarch では、ライブラリや実行バイナリをインストールするディレクトリは、このように変更されます。

```
/usr/lib -> /usr/lib/<triplet>
/usr/lib/<pkgdir> -> /usr/lib/<triplet>/<pkgdir>
/usr/include: no change
/usr/bin: no change
/usr/share: no change
/usr/sbin: no change
```

10.4.2 パッケージフィールド: Multi-Arch

multiarch が導入される前は、パッケージの依存関係は、同じアーキティクチャか、すべてのアーキティクチャをサポートしているパッケージを使ってで解決されていました。同じ名前で異なるアーキティクチャのパッケージは当然同時にインストールできないものと扱われていました。multiarch の仕様では、`Multi-Arch` という新しいバイナリパッケージフィールドを追加しました。このフィールドは、`same`、`foreign`、`allowed` の 3 つの値の 1 つをとることができます。

`Multi-Arch:same` の時は、同じ名前のパッケージと同じシステムにインストールすることができます (co-installable) が、他のアーキティクチャのあらゆるパッケージの依存を解決することに使ってはならないことを意味しています。また、`Multi-Arch:foreign` の時は、同じ名前のパッケージは同じシステムにインストールすることはできませんが、他のアーキティクチャのパッケージの依存の解決に使うことができます。つまり、あるライブラリのアーキティクチャーに依存する部分は、`Multi-Arch:same` のフィールドを持つ、`libfoo` というパッケージにして、アーキティクチャに依存しないサポートファイルなどは、`Multi-Arch:foreign` の値を持つ、たとえば `libfoo-data` というパッケージに分離する必要があります。最後の `Multi-Arch:allow` は、reverse-dependency があるパッケージで、同じ名前の他のアーキティク

[*28] http://wiki.debian.org/Multiarch/Implementation
[*29] http://www.debian.org/doc/debian-policy/ch-opersys.html#s-fhs

チャのパッケージが依存を解決できるときに指定します。これは、依存しているパッケージのメンテナの仲介なしに、パッケージがアーキティクチャに関係しない依存を持つと、誤って表明することを防ぐためにあります。

10.4.3 multiarch 対応パッケージを作成する手順

autoconf を使った upstream のソースを dh を使って、簡単なパッケージを multiarch に対応させるときの手順は、このようになるでしょう。

1. debhelper(>= 9) に Build-depend させる
2. `Pre-Depends:${misc:Pre-Depends}` を追加する
3. `debian/compat` を 9 にする
4. `debian/*.install` の中の`/usr/lib` を、`/usr/lib/*/`に変更する
5. もし `debian/*.install` や、 `debian/*.link` で、`/usr/lib`(またはそのサブディレクトリ) にインストールされるファイルを指定している、もしくはリンクを張るように指定しているなら、 `debian/rules` で、`$(DEB_HOST_MULTIARCH)` の値を使うように変更した、これらのファイルを自動生成する必要があるでしょう。
6. `debian/rules` の中に記述されている、`/usr/lib` をすべて`/usr/lib/$(DEB_HOST_MULTIARCH)` に変更する
7. `debian/rules` の中で変数`$(DEB_HOST_MULTIARCH)` を使う必要があるのなら、 `debian/rules` の中に、`DEB_HOST_MULTIARCH ?= $(shell dpkg-architecture -dDEB_HOST_MULTIARCH)` と記述して、変数 `DEB_HOST_MULTIARCH` を初期化する。
8. パッケージがビルドできて、共有ライブラリパッケージに、思ったとおりのファイルだけ含まれており、-dev パッケージもきちんと動作することを確かめたら、 `Multi-Arch: same` と、 debian/contorl に記述する。
9. (architecture: all の)common パッケージが必要なら、 `debian/control` で、 common パッケージに、`Multi-Arch:foreign` と設定する。

10.5 まとめ

Debian multiarch は、 1 つのシステムに複数のアーキティクチャのライブラリ、プログラムをインストール、実行するためのしくみです。この文書では、 multiarch の使い方と、しくみ、 multiarch に対応した共有ライブラリパッケージの作成方法を説明しました。 multiarch は、 wheezy の Release Goal です。次期リリースではすべてのひとが、 Debian multiarch を使うことができます。

参考文献

[1] MultiarchSpec `https://wiki.ubuntu.com/MultiarchSpec`

[2] Multiarch Implementation `http://wiki.debian.org/Multiarch/Implementation`

[3] Multiarch paths and toolchain implications `http://wiki.debian.org/Multiarch/LibraryPathOverview`

11 家庭内 LAN を高速に! - InfiniBand on Debian

山田 泰資

最近の InfiniBand(IB)[1] 機材の安さに、思わず家庭内 IB に手を出してしまいました。その導入経験を報告します。

11.1 InfiniBand の特徴と全体像

最初に IB に触れて戸惑うのは、規定されている範囲が L1-L7 と幅広く、慣れ親しんできた L2:Ethernet、 L3:IP、… という階層構造に収まる単なる「 新しい L2 規格」ではないことです。また、用語の登場具合が

> IB の L4 にあたる部分には RC/RD/UC/UD/XRC などがあります。 RC が TCP、 UD が UDP に相当します。 IP や MAC に対応する要素が GID/LID、ポートに相当する要素は QP という両端の HCA 上に確保されるキュー (WQ) の組と言えます。そして WQ が 1 対 1 対応する形で QP を形成するものが RC/UC、 1 対 N 対応する形のものが RD/UD/XRC という分類になります。 SRP などの ULP は WQ に各種 Verb を WQE の形で送り込み、 HCA を駆動して RDMA を利用しています。

などと TLA が分野横断かつ詰め込み気味なのに悩まされます[*30]。

紙面の関係から詳しくはスライドでとなりますが、資料を読む上では背景として

- RDMA のため OS バイパスが必要となり、整理・規定範囲が L7 まで及ぶこと
- 既存レイヤへの RDMA 導入のため概念+規格+ API と整理が細かくなっていること
- L2-L3 相当の要素には IB/RoCE/iWARP の三種があり、 Verbs で抽象化されること

という点までを押さえた上で、同じ IB でもどのレイヤ位置の話なのか、また、本当の意味で IB 固有なのか RDMA 系各種規格に適用できる話かを考えながら読み進めると混乱しにくくなります。

11.2 InfiniBand on Debian の現状

プロトコルスタックやドライバは素の Linux kernel にも概ね入っているのですが、管理ツールなども含めた形では OFA が OFED(OpenFabrics Enterprise Distribution) を開発・配布しています。 OFED の Linux 版は RedHat 向けですが、実際にはこれを元に各社が移植・独自拡張を行ったものを再配布しており、有力なものに Mellanox 社の MLNX_OFED パッケージがあります[*31]。

Debian では OFED Infiniband Distribution project [*32] がドキュメント整備とパッケージングを行っており、squeeze 向けには OFED-1.4.2 が、そして wheezy/sid 向けに OFED-1.5 のパッケージングが遅れ気味ですが進んでい

[*30] これは各種資料の調子を模して詰め込んでみたサンプルです。セッションでは上記内容は図で解説予定
[*31] これの付属マニュアル [2] は何を措いても読むべき資料
[*32] http://pkg-ofed.alioth.debian.org/

ます。プロジェクトページには

```
deb     http://pkg-ofed.alioth.debian.org/apt/ofed-X.Y.Z ./
deb-src http://pkg-ofed.alioth.debian.org/apt/ofed-X.Y.Z ./
```

の apt-line を追加した上で apt-get install ofed で入るとありますが、ここには 1.4.2 までしかないため、現状では wheezy/sid から先取りするか自分でビルドします[*33]。

今回レベルで使う分には OFED-1.4 で問題ないのですが、例えば IB-FDR [*34]への対応は 1.5.4 以降が必要になり、また、さらに新しい機能やカーネルを使うのであれば OFED-3.2 や開発中リポジトリ[*35] ベースでのリビルドが必要になるなど、試行錯誤が必要です。

11.3 使ってみよう - 基本的な IB/IPoIB 環境の構築

それでは使ってみましょう。基本的な部分については先の Debian のプロジェクト資料 [3] や各種の日本語記事 [4] [5] もあるため、現在 Debian で構築するにあたり固有の部分だけを紹介します。

11.3.1 パッケージの導入と初期セットアップ

プロジェクトのリポジトリであれば ofed パッケージがあるのですが、これは現状 broken dependency の状態のため、sid の方より個別に導入します:

```
# apt-get install opensm ibverbs-utils infiniband-diags perftest                    // 基本ツール群
# apt-get install libmlx4-1 libmthca1 mstflint                                      // HCA 関係パッケージ群
# apt-get install ibutils rdmacm-utils rds-tools libsdp1 dapl2-utils                // 追加ツール群
# apt-get install libibumad-dev libibverbs-dev librdmacm-dev libibcm-dev libdapl-dev // 開発ライブラリ群
```

さて、これでインストールはされましたが、このままでは使えません。どのような ULP、例えば IPoIB を使うのかどうか、またユーザーランドから RDMA を叩く使い方をするのかなどは導入ポリシ次第なので、明示的に有効化する必要があります。

素の OFED にはこれに使える init スクリプト (openibd.conf + openibd) が入っているのですが、やっていることはモジュールロードと sysctl 等のパラメータ設定なので、以下のモジュールを/etc/module に直接ロードして代えることができます:

```
# Mellanox HCA ベースで構築している今回の場合のモジュールリスト
mlx4_ib ib_mthca ib_uverbs ib_umad ib_ucm rdma_ucm ib_ipoib ib_sdp ib_srp
```

11.3.2 サブネットマネージャの稼動

サブネットマネージャ (SM) は系内のトポロジを一元管理し、また各ポートに GID/LID を割り当てて Active 状態にする上で必須のサービスです。スイッチに内蔵されていることもありますが、複数起動していても問題ないため下記の通り起動します:

```
# cat /etc/defaults/opensm
PORTS=ALL
# /etc/init.d/opensm start
```

後は ibstat などでポート状態が Active になっていれば、利用可能です。

11.3.3 チューニングパラメータ

IPoIB のメリットは通常の LAN インタフェースとして利用できることですが、反面チューニング設定なしではオーバーヘッドが大きく性能が出ません。 Mellanox のガイドにもありますが、以下の設定はほとんど必須となります:

[*33] 以降の説明は sid 環境、また、紹介する機能の関係で Linux 3.4.0 ベースとなります

[*34] IB はシリアルリンクを束ねる方式で、リンク当たり速度を SDR(2Gbps), DDR(4Gbps), QDR(8Gbps), FDR(14Gbps), EDR(26Gbps), ... と高速化して帯域を増やします。またリンク数も x1, x4, x12 が規定されています(x4 のタイプが主流)

[*35] https://beany.openfabrics.org/downloads/MAINTAINERS

```
# echo connected > /sys/class/net/ib1/mode ← RC モードにし、 64KB MTU 設定を可能にする
# ip link set ib1 mtu 65520                ← 同上
# cat <<EOF > /etc/sysctl.d/network.conf   ← IB へのマッピング効率が向上するよう、バッファを大型化する
net.ipv4.tcp_timestamps = 0
net.ipv4.tcp_sack = 0
net.core.netdev_max_backlog = 250000
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.core.rmem_default = 16777216
net.core.wmem_default = 16777216
net.core.optmem_max = 16777216
net.ipv4.tcp_mem = 16777216 16777216 16777216
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
EOF
# sysctl -qp/etc/sysctl.d/network.conf
```

IP アドレスの設定を終えたら、最後に足場の確認として netperf や ib_read_bw にて性能を確認しましょう。ほぼワイヤレートが出せていれば作業は完了です。

11.4 もっと使ってみよう - SAN 構築

さて、基本的な導通確認は取れたので、次はサーバとして SAN を構築します。今回は iSCSI over IPoIB と SRP の 2 通りで行い、比較します。

11.4.1 iSCSI over IPoIB

こちらは IPoIB で隠蔽されているので、普通の iSCSI SAN の導入と手順は完全に同じです。今回は

- target: Linux-iSCSI (LIO)
- initiator: open-iscsi

の組み合わせで構築します。以下が実際のコマンド例です。

```
# apt-get install targetcli lio-utils ← ターゲット用パッケージを導入
# echo 0 $((2 * 1024 * 1024 * 512)) zero | dmsetup create ramdisk ← 試験用に 512GB のダミーディスクを用意
# targetcli
/> cd /backstores/iblock
/backstores/iblock> create ramdisk /dev/mapper/ramdisk ← 先のディスクを登録
...
/backstores/iblock/ramdisk> cd /iscsi
```

```
/iscsi> create iqn.2003-01.org.linux-iscsi:ramdisk ← IQN 生成
...
/iscsi/iqn.20...ramdisk/tpgt1> set attribute authentication=0 ← 認証オフ
Parameter authentication is now '0'.
/iscsi/iqn.20...ramdisk/tpgt1> set attribute generate_node_acls=1 ← アクセス時に自動的に ACL 許可を生成
Parameter generate_node_acls is now '1'.
/iscsi/iqn.20...ramdisk/tpgt1> set attribute demo_mode_write_protect=0 ← 書込み保護解除
Parameter demo_mode_write_protect is now '0'.
```

```
/iscsi/iqn.20...ramdisk/tpgt1> cd luns
/iscsi/iqn.20...sk/tpgt1/luns> create /backstores/iblock/ramdisk 0 ← 先のディスクを紐付ける
...
/iscsi/iqn.20...gt1/luns/lun0> cd ../../portals
/iscsi/iqn.20...tpgt1/portals> create 10.254.1.16      ← アクセス用のポータルを開始
...
/iscsi/iqn.20...254.1.16:3260> cd /
/> saveconfig                                          ← 最後に保存
```

淡々とコマンドを打つだけでしたが、以上で設定は完了です。

ターゲット側はこれで稼動を始めたので、次はイニシエータ側です。 open-iscsi デーモン(iscsid）を導入・起動した上で、 iscsiadm コマンドでターゲット側を照会し、エクスポートされているストレージにアタッチします。

```
# apt-get install open-iscsi
# /etc/init.d/open-iscsi start                                  ← iscsid 起動
# iscsiadm -m discovery -t sendtargets -p 10.254.1.16           ← ターゲット IQN を照会する
10.254.1.16:3260,1 iqn.2003-01.org.linux-iscsi:ramdisk
# iscsiadm -m node -T iqn.2003-01.org.linux-iscsi:ramdisk -l ← セッションを確立
...
# iscsiadm -m session                                           ← 認識結果を確認
tcp: [1] 10.254.1.16:3260,1 iqn.2003-01.org.linux-iscsi:ramdisk
# dmesg
...
[ 1403.238057] scsi6 : iSCSI Initiator over TCP/IP
[ 1403.551708] scsi 6:0:0:0: Direct-Access     LIO-ORG  IBLOCK          4.0  PQ: 0 ANSI: 5
...
[ 1404.067053] sd 6:0:0:0: [sde] Attached SCSI disk
```

さて、性能はどんなものでしょうか?

```
# JOBS=4 OP=write DEV=/dev/sde BS=1m fio bench.ini
...
Run status group 0 (all jobs):
  WRITE: io=6254.0MB, aggrb=210869KB/s, minb=53870KB/s, maxb=54137KB/s, mint=30287msec, maxt=30370msec
```

正直な所、あまり振るいません。さすがに 200MB/s では改善の余地があるとは思われますが、今回の主眼は次なのでそのまま SRP に進みます。

11.4.2 SRP (SCSI RDMA Protocol)

さて、次は比較のため RDMA 上で直接 SCSI を提供する SRP で構築してみます。

- target: SCST (ib_srpt)
- initiator: LIO (ib_srp) ← カーネル標準のもの

の組み合わせで構築します。 SRP はターゲット、イニシエータの双方とも Linux-3.4 以降の LIO には含まれていますが、ターゲットについては SCST からコードの取り込み途上で未熟なため、今回は SCST を使っています。

Debian には SCST のパッケージがないため、まずはこれのビルドが必要です[*36] 。

```
$ svn co https://scst.svn.sourceforge.net/svnroot/scst/trunk
$ cd trunk
$ make KDIR=/d/src/linux/master scst iscsi srpt scst_local usr
  ...
  CC [M]  /d/src/scst/trunk/scst/src/scst_main.o
  /d/src/scst/trunk/scst/src/scst_main.c:59:2: warning: #warning
  Patch scst_exec_req_fifo-<kernel-version> was not applied on your kernel.
  Pass-through dev handlers will not work. [-Wcpp]
  ...
make[1]: Leaving directory ‘/d/src/scst/trunk/usr/fileio’
```

全機能をフルに使うにはカーネルパッチを当てる必要があるという警告が何件か出ますが、今回の目的には影響しないためこのまま進みます。最終的に、以下のようなインストール構成になります:

```
# ls /var/lib/scst/pr/ <- このフォルダがない場合、作成して下さい
# ls /lib/modules/3.4.0-rc1-tai-4f7e834f-next-20120405/extra/scst/
 676 ib_srpt.ko        296 scst_disk.ko        300 scst_tape.ko
3712 iscsi-scst.ko     460 scst_local.ko       620 scst_user.ko
4308 scst.ko           296 scst_modisk.ko      704 scst_vdisk.ko
 292 scst_cdrom.ko     272 scst_processor.ko
 272 scst_changer.ko   272 scst_raid.ko
# ls /usr/local/bin/
132 fileio_tgt*   64 iscsi-scst-adm*  372 iscsi-scstd*  152 scstadmin*
```

それでは SCST で SRP 構成を作ってみましょう。 iSCSI/LIO の場合は iSCSI daemon を起動した上で管理コマンドで設定を操作する形でしたが、 SRP の場合は daemon はおらず、 scstadmin コマンドでカーネルモジュールのロードと設定を行うだけになります。

[*36] パッケージにしたい所ですが、これをパッケージングする技量が自分にはない…

```
# echo 0 $((2 * 1024 * 1024 * 512)) zero | dmsetup create zero
# cat > scst-test.conf
# 本ファイルは手動作成してもよいが、 scstadmin コマンドで構成を作る形でもよい。
# 以降のコメントでは設定エントリに対応するコマンド例を示す。
#
# CMD: scstadmin -open_dev zero -handler vdisk_blockio --attributes filename=/dev/mapper/zero
HANDLER vdisk_blockio {
        DEVICE zero {
                filename /dev/mapper/zero
        }
}
# CMD: scstadmin -add_lun 0 -driver ib_srpt -target ib_srpt_target_0 -device zero
TARGET_DRIVER ib_srpt {
        TARGET ib_srpt_target_0 {
                enabled 1
                rel_tgt_id 2
                LUN 0 zero
        }
}
^D
# modprobe scst
# modprobe scst_vdisk
# modprobe ib_srpt
# scstadmin -config scst-test.conf
```

これでターゲット側は設定完了です。次はイニシエータ側ですが、これは ibsrpdm コマンドでターゲット側にクエリを投げ、戻ってきたキー情報などを ib_srp.ko に sysfs 経由で受け渡します。

```
# ibsrpdm -c -d /dev/infiniband/umad1
id_ext=0008f1040399d858,ioc_guid=0008f1040399d858,dgid=fe8000000000000000008f1040399d85a,pkey=ffff,service_id=0008f1040399d858
# for i in $(ibsrpdm -c -d /dev/infiniband/umad1); do echo $i > /sys/class/infiniband_srp/srp-mlx4_0-2/add_target; done
# dmesg
...
[1493617.766923] scsi8 : SRP.T10:0008F1040399D858
[1493618.663499] scsi 8:0:0:0: Direct-Access     SCST_BIO zero             300 PQ: 0 ANSI: 5
...
[1493620.678542] sd 8:0:0:0: [sdf] Attached SCSI disk
```

というわけで SRP でも認識できました。起動時に自動的にアタッチさせるには、上記の ibsrpdm 出力を/etc/srp_daemon.conf に書き込み、起動時に srp_daemon に同様の処理をさせるようにします。

さて、性能はどうでしょうか?

```
# JOBS=4 OP=write DEV=/dev/sdf BS=1m fio bench.ini
...
Run status group 0 (all jobs):
  WRITE: io=26387MB, aggrb=898250KB/s, minb=223059KB/s, maxb=233121KB/s, mint=30048msec, maxt=30081msec
```

圧倒的じゃないか、我が軍は・・・という所でしょうか。ブロックサイズが大きくないと出せない速度なので割り引いて受け取る必要はありますが、オーバーヘッドの少ない RDMA の威力を垣間見ることができました。

参考文献

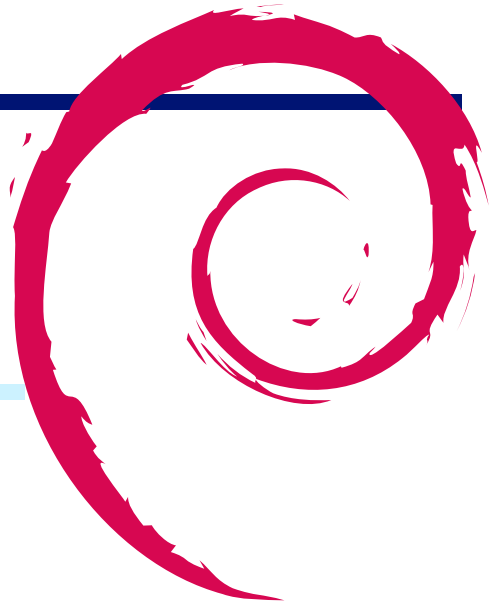
[1] InfiniBandArchitecture Specification Release 1.2.1,
http://members.infinibandta.org/kwspub/specs/register/publicspec/

[2] Mellanox OFED for Linux User Manual,
http://www.mellanox.com/related-docs/prod_software/Mellanox%20OFED%20Linux%20User%20Manual%201_5_3-3_0_0.pdf

[3] Infiniband HOWTO,
http://pkg-ofed.alioth.debian.org/howto/infiniband-howto.html

[4] Altima - Mellanox 技術紹介,
http://www.altima.co.jp/products/mellanoxtechnologies/mellanox_techinfo.html

[5] 松本直人, InfiniBand で変わるデータセンター内通信,
http://www.atmarkit.co.jp/fnetwork/tokusyuu/51ib01/01.html (2011/2),
http://www.atmarkit.co.jp/fnetwork/tokusyuu/61ib02/01.html (2011/7)
http://www.kernel.org/doc/ols/2005/ols2005v2-pages-279-290.pdf



12 Gentoo/Prefix on Debian

青田直大

12.1 はじめに

いきなり Gentoo が出てきて驚かれたかもしれません。ここでは Debian 下で Gentoo Prefix をインストールする方法について解説します。

12.2 Gentoo Prefix

12.2.1 Gentoo とは

Gentoo は Debian や RPM と違ったソースベースのディストリビューションです。ebuild というパッケージビルド方法や、パッケージの依存関係などを記述した bash スクリプト風のファイルが “カテゴリー/パッケージ名/パッケージ名-パッケージバージョン.ebuild” というファイル名で保管されています。emerge というコマンドはこの「Portage ツリー」というパッケージ情報ファイルツリーを読んで指定されたパッケージをビルド・インストールするパッケージ管理ソフトになっています。

12.2.2 Prefix サポート

Gentoo も基本的には Linux 上のディストリビューションですが、Debian の Debian GNU/kFreeBSD と同様に FreeBSD 上でのパッケージ管理を目的とした Gentoo/FreeBSD などの開発も行なわれています。そういった Linux 以外での Gentoo のサポートを総称して Gentoo/Alt と呼んでいます。[*37] その Gentoo/Alt の中に Gentoo Prefix というものがあります。[*38] これは Gentoo のシステムを使って (基本的には) 他の OS 上で、任意のディレクトリにパッケージのインストールを行なえるようにするものです。たとえば、Mac OS X 上で動かして MacPorts や homebrew の代わりに使ったり、あるいは FreeBSD 上で使ったり、はたまた Linux 上で使うこともちろんできます。

12.3 Gentoo/Prefix on Debian

さてここからが本題で、この Gentoo/Prefix を Debian で使ってみます。おそらくなんでそんなことを…? と思われるかもしれません。以下のような点があげられるかと思います。

- 非 root ユーザでも好きなところにインストールできる
- Debian にないパッケージをインストールする時に楽できる
- 技術的に楽しい?

[*37] http://www.gentoo.org/proj/en/gentoo-alt/
[*38] http://www.gentoo.org/proj/en/gentoo-alt/prefix/index.xml

Prefix インストールでは好きなディレクトリにインストールできるため、たとえば自分のホームディレクトリの中などにインストールするようにしてしまえば root 権限がなくともパッケージをインストールすることができます。また、(もし万が一) Debian にないパッケージがあったとして、それが Gentoo の方にあれば、自分でビルド方法や依存を調べることなく、Gentoo のパッケージシステムにおまかせしてしまうことができる、というわけです。

12.3.1 インストール

では、さっそく Debian 上にインストールしてみましょう。まずはビルドに必要なパッケージをインストールしておきます。

```
$ apt-get install bzip2 build-essential bison libreadline-dev libncurses-dev autoconf xz-utils
```

つぎに、Prefix をインストールする場所を決めて、変数 EPREFIX に設定し、PATH も通しておきます。

```
$ export EPREFIX="$HOME/gentoo"
$ export PATH="$EPREFIX/usr/bin:$EPREFIX/bin:$EPREFIX/tmp/usr/bin:$EPREFIX/tmp/bin:/usr/bin:/bin:$PATH"
```

Prefix をインストールするためのスクリプトを取得し、そのスクリプトを使ってパッケージ管理ソフト Portage とパッケージ情報の Portage ツリーをインストールしていきます。

```
$ wget http://overlays.gentoo.org/proj/alt/browser/trunk/prefix-overlay/scripts/bootstrap-prefix.sh?format=txt \
  -O bootstrap-prefix.sh
$ chmod 755 bootstrap-prefix.sh
$ ./bootstrap-prefix.sh $EPREFIX tree
$ ./bootstrap-prefix.sh $EPREFIX portage
```

この時点で emerge コマンドが使えるようになります。ここからは emerge を使って、Prefix 環境を整えていきます。このあたりはあまり今回の主題ではないので残りは Prefix のドキュメント[*39] を参考にしてください。[*40]

12.4 apt-emerge

ここまでで Gentoo/Prefix を解説してきました。しかし、これではあまりに Gentoo すぎますね。Gentoo/Prefix の中にも Debian 側で入っているはずのプログラム・ライブラリがインストールされてしまってなんだか無駄なような気がしてしまいます。特に Debian だとさくさくとバイナリパッケージからインストールされてしまうのに、Gentoo だとソースからビルドされて待つのもなかなか大変なものです。なんとか Debian にあるものは Debian のものを使いつつ、Gentoo ではどうしても必要なものだけをインストールすることはできないでしょうか。

Gentoo では etc/portage/profile/package.provided ファイルに以下のように書くことで、そのパッケージがインストールされていると「見做す」ことができます。

```
sys-power/acpi-1.6
sys-power/acpid-2.0.16
sys-process/at-3.1.13
sys-devel/autoconf-2.69
sys-devel/automake-1.11.3
app-shells/bash-4.2
```

Debian 側にあるパッケージをこうやって Gentoo 側のパッケージ名にマップして、package.provided ファイルに書くことで、Gentoo 側で依存としてパッケージがビルド・インストールされることがなくなります。こうして

1. Gentoo の emerge での依存パッケージを把握
2. Gentoo での依存パッケージ名を Debian のパッケージ名にマップ
3. インストールされた Debian のパッケージ名を Gentoo のパッケージ名にマップ
4. マップされたパッケージ名を pacakge.provided に追記
5. 再度依存関係を計算しなおして 2 に戻る

[*39] http://www.gentoo.org/proj/en/gentoo-alt/prefix/bootstrap-solaris.xml
[*40] いまの Debian だと multiarch の影響でうまく動かないところもあるかもしれません…

6. これ以上 Debian からインストールできなければ emerge を開始

といったプロセスを行ない、最小限のパッケージだけで目的の Gentoo のパッケージをインストールすることができます。この時に肝となるのが、「Gentoo での依存パッケージの把握」と「Gentoo と Debian とのパッケージの相互マップをどうするのか」の二点です。

12.4.1 Gentoo での依存パッケージ

emerge に対して以下のオプションをつけて出力を解析します。

- -p (–pretend): インストールを実行せずインストールパッケージ一覧だけを出力します
- -q (–quiet): 無駄な情報を出力しないようにします
- -t (–tree): インストールするパッケージを依存関係のツリー状に出力します

-t が一番重要なところです。これを指定することでこのようにツリー状に出力されます。

```
$  emerge -pqt --quiet-repo-display chromium
[ebuild  N    ] www-client/chromium-20.0.1132.21
...
[nomerge      ] www-client/chromium-20.0.1132.21
[ebuild  N    ]  dev-libs/nss-3.13.4
[ebuild  N    ]   dev-libs/nspr-4.9
[ebuild  N    ]    sys-devel/autoconf-2.13
[ebuild  N    ]     sys-devel/autoconf-wrapper-12
[ebuild  N    ]   dev-db/sqlite-3.7.12.1
...
```

このツリーの浅いところから Debian のパッケージへとマップしていきます。浅いところからマップしていくことで、できるだけ多くの依存解決を Debian 側におまかせします。つまり、たとえばここで dev-libs/nspr を Debian 側でインストールすることに成功すればこのツリーを

```
[ebuild  N    ] www-client/chromium-20.0.1132.21
...
[nomerge      ] www-client/chromium-20.0.1132.21
[ebuild  N    ]  dev-libs/nss-3.13.4
[ebuild  N    ]   dev-db/sqlite-3.7.12.1
...
```

ここまで一気に縮小することができるというわけです。

12.4.2 Gentoo から Debian へのマップ

Gentoo のパッケージ名から Debian のパッケージ名へのマッピングを行ないます。Gentoo のパッケージ名にはカテゴリがついていますが、Debian の方にはそれがないので、とりあえず外してしまいます。そして、以下の順番で探索をかけます。

- lib パッケージ名-dev
- パッケージ名-dev
- パッケージ名

Gentoo のパッケージでは一般に Debian の*-dev に入るようなものがインストールされているので、*-dev を優先してインストールするようにしています。

また、dev-ruby カテゴリのものには ruby-をパッケージ名につけるなどの工夫をしたり、どうしてもこれでマップできないものは明示的にマッピングを書くなどの対処もしています。

12.4.3 Debian から Gentoo へのマップ

こうして Debian へとパッケージをインストールできたら、インストールできたパッケージのバージョンを取得して Gentoo のパッケージ名へのマップをしていきます。これは単純に `dpkg -l ...` の結果を使うだけですね。現状 Debian の仮想パッケージから選択される実際のパッケージのバージョンをとれず、うまくマップできていません…。

12.4.4 実行サンプル

これらのアイデアを実装したのが apt-emerge スクリプトになります。このスクリプトは基本的に上記の emerge が使えるようになった段階で使えるようになっていますが、一部のパッケージは Gentoo のコア部分に大きく食いこんでいるのでそれらは Gentoo で入れておかないといけません。また、 Debian の multiarch に対応するように CFLAGS/LDFLAGS を調整します。

```
$ vi $EPREFIX/etc/make.conf
CFLAGS="-O2 -I/usr/include/x86_64-linux-gnu"
LDFLAGS="${LDFLAGS} -L/usr/lib/x86_64-linux-gnu"
$ FEATURES="-collision-protect" emerge -avg1 --nodeps bash eselect eselect-python python portage libffi
```

これで apt-emerge が動くようになるはずです。 apt-emerge を動かすと自動的に必要なパッケージを apt-get に渡し、可能な限りの依存を解決してから、 emerge に処理を渡します。

例として Twitter クライアント mikutter[*41]を apt-emerge でインストールしてみましょう。これは Ruby と Ruby/Gtk2 などを使ったパッケージなので、普通に emerge するだけであれば、 gtk など大量にビルドされるはずですが……apt-emerge mikutter 後に Gentoo 側になにがインストールされているかをリストアップしてみましょう (仮想パッケージは除いてあります)。

```
$ eix -I -c |grep -v virtual
[I] app-admin/eselect (1.3.1@06/03/2012): Gentoo's multi-purpose configuration and management tool
[I] app-admin/eselect-python (20111108@06/03/2012): Eselect module for management of multiple Python versions
[I] app-admin/python-updater (0.10-r2@06/04/2012): Script used to reinstall Python packages after changing of active Python versions
[U] app-shells/bash (4.2_p28@06/03/2012 -> 4.2_p29): The standard GNU Bourne again shell
[I] app-shells/push (1.5@06/04/2012): A POSIX shell function to treat a variable like an array, quoting args.
[I] dev-lang/python (2.7.3-r2(2.7)@06/04/2012): Python is an interpreted, interactive, object-oriented programming language.
[I] dev-libs/libffi (3.0.11@06/04/2012): a portable, high level programming interface to various calling conventions.
[I] net-misc/mikutter (9999@06/04/2012): mikutter is simple, powerful and moeful twitter client
[I] sys-apps/baselayout-prefix (1.12.14@06/04/2012): Baselayout for Gentoo Prefix installs
[I] sys-apps/portage (2.2.01.20430@06/04/2012): Prefix branch of the Portage Package Manager, used in Gentoo Prefix
[I] sys-apps/tcp-wrappers (7.6.22@06/04/2012): TCP Wrappers
[I] sys-devel/gnuconfig (20120116@06/04/2012): Updated config.sub and config.guess file from GNU
```

ご覧のように 10 個程度しか Gentoo 側にはインストールされていませんが、 mikutter-9999 (SVN trunk のバージョン) がしっかりとインストールされています。

12.5 これから

apt-emerge のスクリプトはまだコンセプトが実装されただけで、いろんな部分が ad-hoc になっています。より Debian のパッケージ名と Gentoo のパッケージ名とのマッピングの推測を賢くしたり、必要な固定マップデータを拡充し、 Debian の virtual パッケージをうまく処理したりするなど、より使いやすいビルドシステムを作れればと思います。自分自身 Debian に深く知識を持っているわけではないので、もしかするともっと効率のよい実装もあるかもしれません…。もし興味を持っていただければ、開発参加・アドバイスしていただけましたら幸いです。

[*41] http://mikutter.hachune.net

13 Debian でもマルチタッチデバイスを使う

赤部 晃一

13.1 はじめに

近年、スマートフォンやタブレット端末などのマルチタッチスクリーンを搭載したデバイスが、あらゆる場面で注目を集めています。この社会情勢を受け、派生ディストリビューションの Ubuntu では、バージョン 10.10 以降でタッチデバイス向けのアプリケーションの開発をサポートする uTouch が提供されるようになりました。今年の 4 月には、 Debian においてもタッチデバイスのサポートが強化された GTK+3.4 や X server 1.12 が提供されるようになり、今後 Linux デスクトップにおいてもマルチタッチ操作対応のアプリケーションが増えることが期待されます。[*42]

ここでは、まず Ubuntu の uTouch を紹介し、 Debian に uTouch を導入します。次に、 ginn というソフトウェアを用いることで、タッチ操作に対応していないソフトウェアでもタッチ入力ができるようにします。

13.2 uTouch の概要

13.2.1 uTouch の目的

uTouch は、アプリケーションにおけるタッチ入力処理をサポートするためのフレームワークです。例えば、 GTK+ で開発中のアプリケーションを、 2 本の指を使った回転操作に対応させる場合を考えます。 GTK+3.4 に搭載されたタッチイベントを使って実装させる場合、タッチしている指の本数や、それぞれの指の動きなどを全てトレースし、指同士の位置関係や相対速度を計算した上で、それが 2 本指による回転操作である事を判定させる処理を記述する必要があります。しかしそれは言葉通り労力のかかる作業であり、複数のジェスチャー(タップ、スワイプ、ピンチ、回転など）について一つずつ記述していてはキリがありません。

そこで登場するライブラリーが uTouch です。 uTouch はいくつかのタッチジェスチャーを自動的に認識し、それぞれのジェスチャーについてイベントを発生させるため、マルチタッチ操作対応のアプリケーションを作成するときの工数を大幅に減らすことができます。

Ubuntu 標準のデスクトップインターフェイスである Unity では、 3 本指ドラッグでウィンドウの移動、 4 本指タップでメインメニューの表示ができます。[*43]

13.2.2 uTouch 周りの依存関係

図 14 に uTouch 周りの依存関係を示します。上のパッケージが下のパッケージに依存しており、太枠の部分が uTouch の基幹のパッケージです。まず utouch-evemu によって、デバイス毎に異なるフォーマットのタッチデータを一つの決

[*42] 筆者は期待されますと言いましたが、デスクトップ環境では未だにマウス操作が主流であり、タッチアプリの普及にはまだまだ時間がかかりそうです。ちなみに筆者はマウスをほとんど使わずに、ペンタブレットをマウスの如く常用しています。

[*43] 対応しているジェスチャーは Ubuntu Wiki で調べられます。 `https://wiki.ubuntu.com/Multitouch#Supported_Gestures`

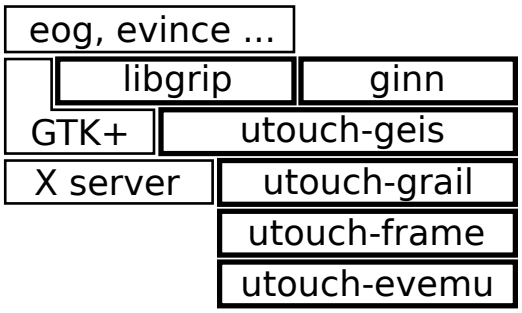


図 14　uTouch 周りの依存関係

まったフォーマットに変換し、イベントデバイスをエミュレートすることで変換結果を出力します。次に utouch-frame によって utouch-evemu が出力する情報を扱いやすい形に変換します。このタッチデータを元に utouch-grail がジェスチャーを認識します。最後に X サーバーの拡張機能として動作する utouch-geis によって、ジェスチャー情報がイベントなどの形でアプリケーションに提供されます。

GTK+ や Qt を使ったアプリケーションでタッチジェスチャーを利用する場合、通常は utouch-geis をそのまま使うのではなく、GUI ツールキット向けのライブラリーを用います。GTK+ の場合は libgrip、Qt の場合は utouch-qml です。今回扱う ginn は、タッチジェスチャーがサポートされていないソフトウェア を、タッチ操作できるようにするためのソフトウェアです。

13.3　uTouch を Debian に取り込む

13.3.1　dget を使ったソースパッケージのダウンロード

では早速、uTouch を Debian に取り込んでみましょう。まず、dget コマンドを使って launchpad から Ubuntu に含まれているソースパッケージをダウンロードします。私の場合は uTouch のメンテナーへの trust path が無いので、`dget` に`-u` オプションを付け、サインの確認を省略します。以下の URL は launchpad のサイトで取得できます。[*44]

```
$ dget -u https://launchpad.net/ubuntu/+archive/primary/+files/utouch-evemu_1.0.9-0ubuntu1.dsc
$ dget -u https://launchpad.net/ubuntu/+archive/primary/+files/utouch-frame_2.2.3-0ubuntu1.dsc
$ dget -u https://launchpad.net/ubuntu/+archive/primary/+files/utouch-grail_3.0.5-0ubuntu1.dsc
$ dget -u https://launchpad.net/ubuntu/+archive/primary/+files/utouch-geis_2.2.9-0ubuntu2.dsc
```

13.3.2　パッケージの修正・ビルド

まずは次のコマンドを実行し、ビルドに必要なパッケージをインストールしておきます。

```
$ sudo apt-get install libx11-xcb-dev xmlto libxi-dev xcb-proto python-xcbgen python-dev \
   dh-autoreconf doxygen asciidoc docbook-xsl libdbus-1-dev xserver-xorg-dev
```

前節でダウンロードしたソースパッケージのうち、utouch-geis はそのままではビルドできないため、パッケージに変更を加える必要があります。まず、utouch-geis-2.2.9/configure.ac の次の箇所を変更します。

```
PKG_CHECK_MODULES([XI2], [x11 xext xi >= 1.3], ,
                  AC_MSG_ERROR([XI2 development libraries not found]))
PKG_CHECK_MODULES([PYTHON], [python >= 2.7]) # ←この行

AX_ENABLE_XI2
```

Debian では、python.pc ではなく python-2.7.pc を参照する必要があるため、次のように変更します。

```
PKG_CHECK_MODULES([PYTHON], [python-2.7])
```

パッチを作成する方法はいくつかありますが、ここでは `dpkg-source` コマンドを利用します。ファイルを修正後、次

[*44] 例えば utouch-evemu の dsc ファイルの場所は次で調べられます: `https://launchpad.net/ubuntu/+source/utouch-evemu`

のコマンドを実行するとパッチが作成されます。

```
$ cd utouch-geis-2.2.9
$ dpkg-source --commit

dpkg-source: info: local changes detected, the modified files are:
 utouch-geis-2.2.9/configure.ac
Enter the desired patch name: 01_fix-pkg-config-path.patch
```

パッチ名を入力して確定すると、パッチ編集画面に移ります。パッチの説明等を入力して保存してください。

それでは、パッケージのビルドです。依存関係上、図 14 の下のパッケージから順にビルド・インストールします。ダウンロードしたパッケージに含まれている debian/control の Maintainer フィールドを自分の名前に変更し、 dch コマンドでバージョン番号を修正し、 `debuild` コマンドを実行してバイナリパッケージを作成します。例えば utouch-evemu の場合は次のコマンドを実行します。

```
$ cd utouch-evemu-1.0.9
$ dch -v 1.0.9-1~dgm1 -D unstable // Debian バージョンを若干上げておく (abbrev of Debian Grand Meeting)
$ debuild -uc -us // サイン省略
```

いくつかのパッケージでは、ビルド後に Lintian の警告が表示されます。本来は警告が消えるまで修正したいところですが、ここでは割愛します。

13.4 実際に uTouch を使ってみる

13.4.1 2 本指・ 3 本指ジェスチャーの有効化

まず次のコマンドを実行し、タッチ入力を uTouch が認識しているかどうか確認します。

```
$ utouch-frame-test-x11
```

このコマンドを実行した状態で、タッチデバイスを 4 本以上の指で触ると、画面が動くはずです。しかし、 3 本指以下では反応しません。この理由は、 2 本指や 3 本指のジェスチャーは、右クリックやスクロールなどのシステムの別の設定と競合してしまうためです。 uTouch でこれらのジェスチャーを認識させたい場合は、 xinput コマンドで設定を無効化させる必要があります。

まず、 `xinput` コマンドを使って接続されているタッチデバイスの ID を確認します。下は筆者の環境で `xinput list` コマンドを実行した結果です。

```
$ xinput list
+ Virtual core pointer                          id=2    [master pointer  (3)]
|   → Virtual core XTEST pointer                    id=4    [slave  pointer  (2)]
|   → HID 0566:3107                                 id=11   [slave  pointer  (2)]
|   → Wacom Bamboo 16FG 4x5 Finger                  id=8    [slave  pointer  (2)]
|   → Wacom Bamboo 16FG 4x5 Pen stylus              id=9    [slave  pointer  (2)]
|   → Wacom Bamboo 16FG 4x5 Pen eraser              id=12   [slave  pointer  (2)]
+ Virtual core keyboard                         id=3    [master keyboard (2)]
    → Virtual core XTEST keyboard                 id=5    [slave  keyboard (3)]
    → Power Button                                id=6    [slave  keyboard (3)]
    → Power Button                                id=7    [slave  keyboard (3)]
    → HID 0566:3107                               id=10   [slave  keyboard (3)]
```

筆者が使用しているタッチデバイスは「 Wacom Bamboo 16FG 4x5 Finger」に相当するので、 ID は 8 であることが分かります。

ここで以下の 3 つのコマンドを実行すると、システムの 2 本指・ 3 本指に対する設定が無効化されます。[*45]

```
$ xinput set-prop 8 "Synaptics Tap Action" 0 0 0 0 1 0 0
$ xinput set-prop 8 "Synaptics Two-Finger Scrolling" 0 0
$ xinput set-prop 8 "Synaptics Click Action" 1 0 0
```

設定を変更したら、もう一度 utouch-frame-test-x11 コマンドを実行してみましょう。 2 本指や 3 本指でタッチした場合も画面が動くようになるはずです。

この設定は、デバイスを抜き挿ししたり、ログイン・ログアウトする度にリセットされてしまいます。ログイン時に次の

[*45] 設定方法の詳細は Ubuntu Wiki で調べられます。 `https://wiki.ubuntu.com/Multitouch/TouchpadSupport`

ようなスクリプトが実行されるようにしておくと、自動的に設定できます。[*46]

```
#!/bin/sh
devname="Wacom Bamboo 16FG 4x5 Finger" # xinput list で調べた名前
devid=$(xinput list | tr -d "\\012" | sed -e "s/.*→\\s$devname\\s\\+id=\\([0-9]\\+\\).*/\\1/g")
xinput set-prop $devid "Synaptics Tap Action" 0 0 0 0 1 0 0
xinput set-prop $devid "Synaptics Two-Finger Scrolling" 0 0
xinput set-prop $devid "Synaptics Click Action" 1 0 0
```

13.4.2 ginn を利用したタッチ操作

2012 年 6 月現在、 uTouch を利用したアプリケーションが非常に少なく[*47]、さらに GTK+ 向けのライブラリーである libgrip が、筆者が使用している Wacom Bamboo CTH-460 に対して機能しない状況です[*48]。本来であれば、アプリケーションをコードレベルでマルチタッチに対応させたいところですが、今回は唯一動作確認できた ginn というソフトウェアを利用します。

ginn は、 utouch-geis によって取得したタッチイベントに応じて、予め指定したショートカットキーを出力することで、タッチ入力に対応していないアプリケーションをタッチ操作できるようにするソフトウェアです。

まず、先程と同様に dget コマンドで ginn をダウンロードし、 debian/control とバージョン番号を修正してビルドします。

```
$ dget -u https://launchpad.net/ubuntu/+archive/primary/+files/ginn_0.2.4-0ubuntu1.dsc
$ cd ginn-0.2.4
$ editor debian/control # Maintainer を変更
$ dch -v 0.2.4-1~dgm1 -D unstable
$ debuild -uc -us
```

生成されたパッケージをインストールして ginn コマンドを実行し、タッチデバイスを何本かの指で撫でてみてください。画面が動き、タッチジェスチャーの種類、指の位置などが表示されるはずです。

次に、 Ubuntu 向けになっている ginn の設定を、 Debian に合わせて変更します。まず、 ginn の設定ファイルをホームディレクトリー以下にコピーします。

```
$ cp /etc/ginn/wishes.xml ~/my_ginn.xml
```

コピーした設定ファイルを開いてみましょう。設定ファイルは XML 形式となっています。 ginn はアクティブウィンドウの種類によって出力するショートカットキーを変更します。ただし、 global タグ以下に記述された設定は、すべてのアクティブウィンドウに対して常に同じショートカットキーを出力します。

例えば、 4 本指スワイプで GNOME Shell のワークスペースを切り替えるようにするには、設定ファイルを次のように変更します。

```
<ginn>
  <global>
    <wish gesture="Drag" fingers="4">
      <action name="action1" when="update">
        <trigger prop="delta y" min="5" max="400"/>
        <key modifier1="Control_L" modifier2="Alt_L">Down</key>
      </action>
    </wish>
    <wish gesture="Drag" fingers="4">
      <action name="action2" when="update">
        <trigger prop="delta y" min="-400" max="-5"/>
        <key modifier1="Control_L" modifier2="Alt_L">Up</key>
      </action>
    </wish>
  </global>
</ginn>
```

[*46] 諸事情により、スクリプト中の文字を一部変更して表示しています。右矢印「 →」は、正しくは折れ矢印 (U+21B3) です。

[*47] Ubuntu のリポジトリーで utouch に依存するパッケージを検索しても、 unity インターフェイスや、 eog、 evince 等のごく一部のアプリケーションしかヒットしません。

[*48] 詳しい原因は分かりませんが、 Ubuntu 11.10 以前は動作していました。

14 東京エリア Debian 勉強会 2011 年の振り返り

まえだこうへい

2011 年 12 月で 7 年目の Debian 勉強会が終了しました。

14.1 基本的な数値

Debian 勉強会は毎回事前課題事後課題を設定しており、予習復習を必要だと謳っている勉強会です。実際にどれくらいの人が出席しているのか、またその人たちがどれくらい事前課題・事後課題を提出しているのか、確認してみましょう。図 15 です。値は一年の移動平均です。

結果を見ると参加者数は下降傾向にあり、事前課題の提出率は昨年からは横ばい傾向です。事前課題をちゃんとやる常連参加者に収斂されてきているのでしょうか? 一方、事後課題 (ブログ) の率はさらに低下しています。昨年、「ブログはもう流行らないのでしょうか」との一言がありましたが、まさにその通りかもしれません。

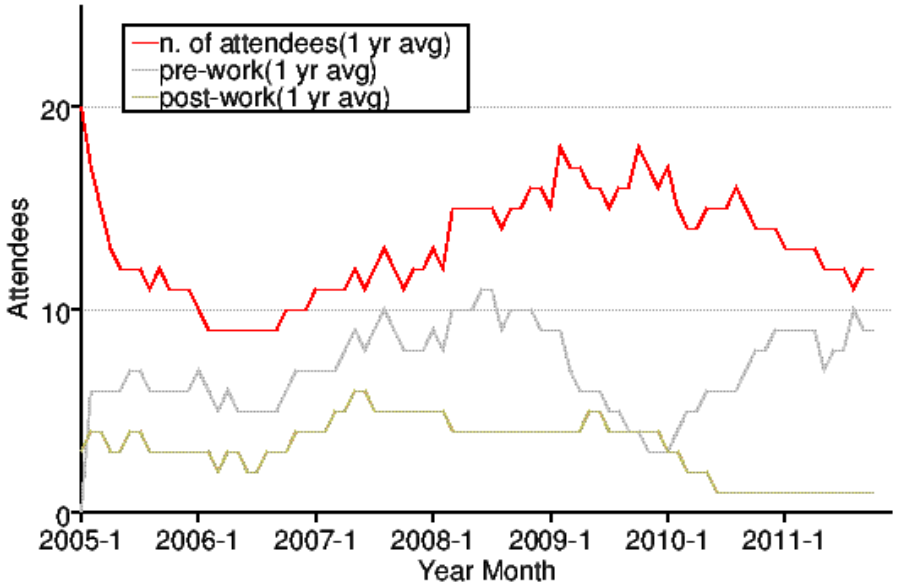


図 15 東京エリア Debian 勉強会事前課題・事後課題提出実績 (12 ヶ月移動平均)

毎回の参加者の人数と、その際のトピックを見てみます。今年も昨年同様、人数的には大きく増減はありませんが、10 月の筑波大での開催での参加者が、昨年の筑波大でのつくらぐさんとの合同勉強会の時と同じ参加者数でした。新参の参加者を期待した筑波大生の参加は思ったほどではなかった一方、初参加なのにわざわざ都心から筑波大まで参加してくれた方が意外と多かった回でした。2009 年から見ると、大学での開催はいつもより参加者数および初参加者が増えるので、来年

以降も続けていくと良いでしょう。また、リストをみると、毎月数名は初参加者も毎月いて、そのうち一部は 2 回目以降も参加する人がいることを考えると、先ほどの参加数が下降傾向にありつつ、事前課題の提出数が横ばいであるのは、ちゃんと事前課題を提出する人が常連になる傾向にあるのではないか、とも見れます。事前課題の提出率は維持しつつ、参加者数は増やしていく為の対策は必要でしょう。

会場を見ると、今年の会場は、あんさんぶる荻窪以外の公民館などの利用が増えました。また、今年は 3 年前の草津温泉、昨年の木更津に続き、三回目の Debian 温泉を伊東の山喜温泉で開催しました。 Debian Hack Cafe も 6 月ごろから月 1 回程度のペースで再開しているようです。

表 2　東京エリア Debian 勉強会参加人数 (2005-2006 年)

	参加人数	内容
2005 年 1 月	21	秘密
2005 年 2 月	10	debhelper 1
2005 年 3 月	8	(早朝) debhelper 2、social contract
2005 年 4 月	6	debhelper 3
2005 年 5 月	8	DFSG、 dpkg-cross、lintian/linda
2005 年 6 月	12	alternatives、 d-i
2005 年 7 月	12	toolchain、 dpatch
2005 年 8 月	7	Debconf 参加報告、ITP からアップロードまで
2005 年 9 月	14	debconf
2005 年 10 月	9	apt-listbugs、 バグレポート、 debconf 翻訳、debbugs
2005 年 11 月	8	DWN 翻訳フロー、 statoverride
2005 年 12 月	8	忘年会
2006 年 1 月	8	policy、 Debian 勉強会でやりたいこと
2006 年 2 月	7	policy、 multimedia
2006 年 3 月	30	OSC: debian 勉強会、sid
2006 年 4 月	15	policy、 LaTeX
2006 年 5 月	6	mexico
2006 年 6 月	16	debconf、 cowdancer
2006 年 7 月	40	OSC-Do: MacBook Debian
2006 年 8 月	17	13 執念
2006 年 9 月	12	翻訳、 Debian-specific、 oprofile
2006 年 10 月	23	network、 i18n 会議、Flash、 apt
2006 年 11 月	20	関西開催: bug、 sid、packaging
2006 年 12 月	14	忘年会

表 3　東京エリア Debian 勉強会参加人数 (2007-2008 年)

	参加人数	内容
2007 年 1 月	15	一年を企画する
2007 年 2 月	13	dbs, dpatch
2007 年 3 月	80	OSC 仮想化
2007 年 4 月	19	quilt, darcs, git
2007 年 5 月	23	etch, pbuilder, superh
2007 年 6 月	4	エジンバラ開催: Debconf7 実況中継
2007 年 7 月	18	Debconf7 参加報告
2007 年 8 月	25	cdn.debian.or.jp
2007 年 9 月	14	exim
2007 年 10 月	30	OSC Tokyo/Fall(CUPS)
2007 年 11 月	19	live-helper, tomoyo linux kernel patch, server
2007 年 12 月	11	忘年会
2008 年 1 月	23	一年を企画する
2008 年 2/29,3/1	36	OSC
2008 年 3 月	37	データだけのパッケージ、ライセンス
2008 年 4 月	17	バイナリパッケージ
2008 年 5 月	20	複数のバイナリパッケージ
2008 年 6 月	10	debhelper
2008 年 7 月	17	Linux kernel patch / module パッケージ
2008 年 8 月	10	Debconf IRC 会議と Debian 温泉
2008 年 9 月	17	po4a,「Debian メンテナのお仕事」
2008 年 10 月	11?	OSC Tokyo/Fall
2008 年 11 月	17	「その場で勉強会資料を作成しちゃえ」 Debian を使った LaTeX 原稿作成合宿
2008 年 12 月	12	忘年会

表 4　東京エリア Debian 勉強会参加人数 (2009-2010 年)

	参加人数	内容
2009 年 1 月	12	一年を企画する
2009 年 2 月	30	OSC パッケージハンズオン
2009 年 3 月	23	Common Lisp, パッケージ作成
2009 年 4 月	15	Java Policy, ocaml, 開発ワークフロー
2009 年 5 月	13	MC-MPI パッケージ化、 Erlang、 Android アプリ、 DDTP
2009 年 6 月	14	DDTP・ DDTSS、 bsdstats パッケージ、 Debian kFreeBSD
2009 年 7 月	4	スペインにて Debconf 9
2009 年 8 月	14	スペイン Debconf 9 参加報告
2009 年 9 月	26	GPG キーサインパーティー
2009 年 10 月	30	OSC Tokyo Fall
2009 年 11 月	12	Octave, R, gnuplot, auto-builder
2009 年 12 月	10	忘年会
2010 年 1 月	17	東京大学にて新年会
2010 年 2 月	11	Debian 温泉,ocaml,haskell
2010 年 3 月	12	weka,fftw,dpkg v3 quilt
2010 年 4 月	15	upstart,piuparts,debtags
2010 年 5 月	22	筑波大学,kernel
2010 年 6 月	12	OSC-Do リハーサル
2010 年 7 月	0	キャンセル
2010 年 8 月	3	Debconf (NYC)
2010 年 9 月	30	OSC Tokyo/Fall
2010 年 10 月	13	俺の Debian な一日
2010 年 11 月	15	ext4,btrfs,nilfs,ceph
2010 年 12 月	14	cacert, libsane

表 5　東京エリア Debian 勉強会参加人数 (2011 年)

	参加人数	内容
2011 年 1 月	12	荻窪,Kinect, アンケートシステム,CACert サイン会
2011 年 2 月	13	北新宿生涯学習館,HDFS,Debian Game Team
2011 年 3 月	?	OSC Tokyo/Spring,CACert ATE Tokyo
2011 年 4 月	12	IIJ,backports,initramfs, 月刊 PPC64
2011 年 5 月	15	戸山生涯学習館,Apache2 モジュール,Debian on ニフクラ,Debian/m68k, 月刊 PPC64
2011 年 6 月	17	東京オリンピックセンター, ドキュメント処理系,2011 再計画
2011 年 7 月	3	DebConf11
2011 年 8 月	12	荻窪, パッケージング関連, Debconf11 報告
2011 年 9 月	9	山喜旅館,Debian 温泉 2011
2011 年 10 月	22	筑波大学,Haskell,LaTeX, レポート自動生成, 月刊 debhelper 開始
2011 年 11 月	?	OSC Tokyo/Fall
2011 年 12 月	9	スクウェア・エニックス,quilt で porting, 月刊 debhelper, 振り返り

15 関西 Debian 勉強会 2011 年の振り返りと 2012 年の企画

Debian JP

初回が 2007 年 3 月なので、 2011 年で 5 年目になりますね。

15.1 勉強会全体

毎回のセッションについてですが、今年はパッケージ作成や BTS に関する題材が多く扱われました。定番のネタ (Debian の入門的なお話、ライセンス、パッケージ作成、 BTS) などの題材は毎年更新されていくべき題材なので, 今後も繰り返し扱っていきたい所です。事前課題で提出されていた「インストール大会」のような初心者講習もやってみたい所ですね。

また、昨年から事前課題の提出が普通になってきましたので、来年からは事後統計 (アンケート) も提出してもらおうか, なんて考えています。

運営に関しては、常連さんだった河田さんが運営側になり、 Debian JP Project へ加入されました。また、今回の発表にある通り倉敷さんが DD になるための NM プロセスへ申請されています。

イベント参加については、 OSC Kansai@ Kobe, OSC Hokkaido, OSC Kansai@ Kyoto, KOF に参加しました。勉強会から fork した GPG キーサインパーティも毎回実施されています。今後も継続して参加する予定です。

15.2 開催実績

関西 Debian 勉強会の出席状況を確認してみましょう。グラフで見ると図 16 になります。また、毎回の参加者の人数とその際のトピックを 表 9 にまとめました。グラフ中の黒線は参加人数, 赤線は 1 年の移動平均です。参加人数が 0 となっているところは人数が集計されていない or 開催されなかった月ですので、欠損値処理をした方が良いですね[*49]。

Debian 勉強会申し込みシステムを使用するようになり, 事前課題を設定することも多くなりました。また, アンケートシステムも稼動するようになるでしょうから、今後は事前課題と事後課題のグラフを追加しようと思っています。

[*49] R 使った事ないので今日は間に合いませんでした

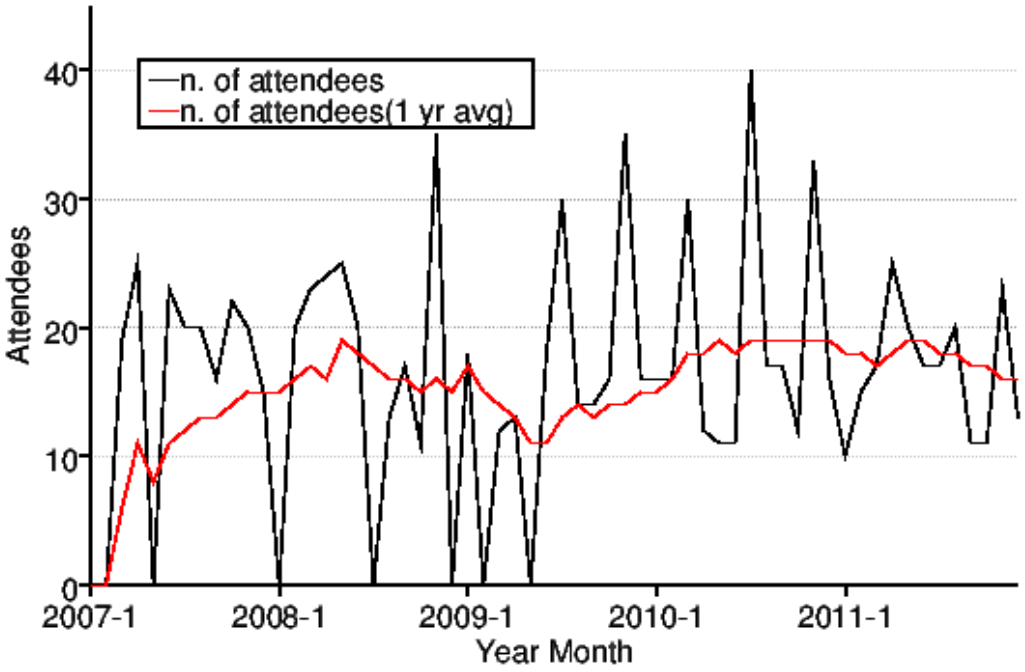


図 16　関西の参加人数推移 (参加人数と 12 ヶ月移動平均)

表 6　関西 Debian 勉強会の参加人数とトピック (2007 年)

	参加人数	内容
2007 年 3 月	19	開催にあたり
2007 年 4 月	25	goodbye、 youtube、プロジェクトトラッカー
2007 年 6 月	23	社会契約、テーマ、 debian/rules、 bugreport
2007 年 7 月	20 前後	OSC-Kansai
2007 年 8 月	20	Inkscape、 patch、 dpatch
2007 年 9 月	16	ライブラリ、翻訳、 debtorrent
2007 年 10 月	22	日本語入力、 SPAM フィルタ
2007 年 11 月	20 前後	KOF
2007 年 12 月	15	忘年会、 iPod touch

表 7　関西 Debian 勉強会参加人数 (2008 年)

	参加人数	内容
2008 年 2 月	20	PC Cluster, GIS, TEX
2008 年 3 月	23	bug report, developer corner, GPG
2008 年 4 月	24	coLinux, Debian GNU/kFreeBSD, sid
2008 年 5 月	25	ipv6, emacs, ustream.tv
2008 年 6 月	20	pbuilder, hotplug, ssl
2008 年 8 月	13	coLinux
2008 年 9 月	17	debian mentors, ubiquity, DFSG
2008 年 10 月	11	cdbs,cdn.debian.or.jp
2008 年 11 月	35	KOF
2008 年 12 月	?	TeX 資料作成ハンズオン

表 8　関西 Debian 勉強会の参加人数とトピック (2009-2010)

	参加人数	内容
2009 年 1 月	18	DMCK, LT
2009 年 3 月	12	Git
2009 年 4 月	13	Installing sid, Mancoosi, keysign
2009 年 6 月	18	Debian Live, bash
2009 年 7 月	30?	OSC2009Kansai
2009 年 8 月	14	DDTSS, lintian
2009 年 9 月	14	reportbug, debian mentors
2009 年 10 月	16	gdb, packaging
2009 年 11 月	35	KOF2009
2009 年 12 月	16	GPS program, OpenStreetMap
2010 年 1 月	16	Xen, 2010 年企画
2010 年 2 月	16	レンタルサーバでの利用, GAE
2010 年 3 月	30?	OSC2010Kobe
2010 年 4 月	12	デスクトップ環境, 正規表現
2010 年 5 月	11	ubuntu, squeeze
2010 年 6 月	11	debhelper7, cdbs, puppet
2010 年 7 月	40?	OSC2010Kyoto
2010 年 8 月	17	emdebian, kFreeBSD
2010 年 9 月	17	タイル WM
2010 年 10 月	12	initramfs, debian live
2010 年 11 月	33	KOF2010
2010 年 12 月	14	Proxmox, annual review

表 9　関西 Debian 勉強会の参加人数とトピック (2011)

開催年月	参加人数	内容
2011 年 1 月	10	BTS, Debian GNU/kFreeBSD
2011 年 2 月	15	pbuilder, Squeeze リリースパーティ
2011 年 3 月	17	ライセンス, Debian のドキュメント関連
2011 年 4 月	25	OSC 2011 Kansai @ Kobe, GPG キーサインパーティ
2011 年 5 月	20	vi, dpkg
2011 年 6 月	17	IPv6, vcs-buildpackagesvn, git
2011 年 7 月	17	OSC 2011 Kansai @ Kyoto, GPG キーサインパーティ
2011 年 8 月	20	Debian パッケージ作成ハンズオン
2011 年 9 月	11	vcs-buildpackagebzr, git
2011 年 10 月	11	Emacs, vim の拡張の Debian パッケージ, 翻訳
2011 年 11 月	23	KOF 2011
2011 年 12 月	13	NM プロセス, BTS

16 東京エリア Debian 勉強会の開催方法

野島 貴英

16.1 はじめに

東京エリア Debian 勉強会では、毎年 12 月号資料では勉強会の開催方法についてまとめています。今回は野島が一通りの流れをまとめます。これを読めば、あなたも東京エリア Debian 勉強会を開けるはず!

16.2 東京エリア Debian 勉強会を開く時のスケジュール

時期	作業内容
開催 2 ヶ月前	場所を確保します。会議室は 2 ヶ月前に予約しないと埋まりやすいです。
開催 2 ヶ月前から 1 ヶ月前	勉強会のテーマと、事前課題を決めます。
開催 1 ヶ月前	勉強会のテーマに沿った原稿を募集します。時間枠にあわせた発表が埋まらない場合、発表者を枠が埋まるまで、探す事になります。また、開催にあたり、作業の分担が必要な時は、この時ぐらいに割り当てを完了します。また、宴会の場所について、確保が困難な事が予想される場合は、あらかじめ席のみ予約しておきます。
開催 2 週間前までに	Debian 勉強会予約システム `http://debianmeeting.appspot.com` に予約登録ページを作成します。 Debian JP Blog `http://www.debian.or.jp/` 及び、 Debian 勉強会 Web サイト `http://tokyodebian.alioth.debian.org/` に勉強会の内容を提示します。メールで debian-users メーリングリストにアナウンスを投げます。 Debian JP twitter や mixi Debian コミュニティーなどにもアナウンスすることが多いです。
開催 1 週間前	発表者は勉強会資料用リポジトリに資料をコミットします。
開催 2 日前	参加者の募集の締切りを行います。この時までに揃った事前課題を勉強会資料にマージします。
開催 1 日前	kinkos 等の印刷製本サービスに原稿印刷および製本を依頼します。
開催当日	1. 参加者から費用の回収と集計をします。 2. 勉強会を定刻どおりに開催します。 3. 司会者、発表者と協力して時間内に勉強会を終了します。 4. 宴会の場所の確保が困難でない時は、宴会の場所を抑えます。 5. 宴会の開催、宴会の会計をします。
開催後	Debian 勉強会予約システムからアンケートをおくります。
12 月	勉強会のアンケート結果などを一年分集計して発表します。

表 10　全体スケジュール

16.3 東京エリア Debian 勉強会の開催調整

東京エリア Debian 勉強会の開催調整は専用のメーリングリストがあり、普段はこちらで調整が行われます。運営者専用のメーリングリストで、一部のコアメンバーのみが参加しています。運営に積極的に参加したい人は勉強会運営者に連絡を取る必要があります。

16.4 初めて東京エリア Debian 勉強会の開催を担当する場合の事前準備

初めて東京エリア Debian 勉強会の開催を担当する場合以下の項目が準備してあるとスムーズです。

1. Debian JP Project の提供するサーバーへログインできるアカウント
   Debian JP Blog を更新する際に必要になります。これは通常 Debian JP Project の会員になるときに付与されます。会員でない人は会員にコミットしてもらうよう依頼する必要があります。
2. `http://alioth.debian.org/` アカウント資料の編集等を行う際に必要になります。アカウントを取得したら、メーリングリストで tokyodebian グループ権限に所属させてほしい旨、取得したアカウント名を沿えて依頼します。このアカウントは誰でも取得可能です。

また、勉強会の幹事担当の場合に必要な知識とコンピュータの環境は以下になります。

1. Debian が動作し、インターネットが利用できる PC の用意
2. ssh,emacs,muse-el,git、 subversion の基本操作についての知識
3. ブラウザと HTML の記載の知識
4. `http://alioth.debian.org/`,`http://qwik.jp/` の使い方の知識
5. LaTeX の知識

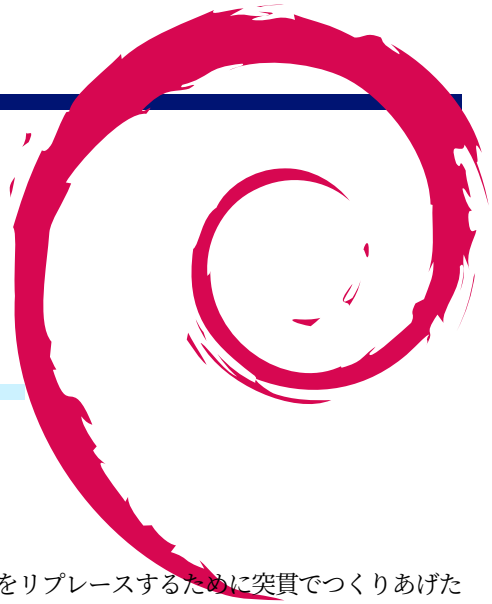
わからないことや困ったことがあればメールやメーリングリストで聞いてみましょう。

16.5 東京エリア Debian 勉強会の掲示の出し方

`http://www.debian.or.jp/` に掲示を出す場合の編集の仕方は、 `http://www.debian.or.jp/project/webmasters.html` にその記載があります。実際には、 subversion でリポジトリをとってきて、 `./www.debian.or.jp/blosxom/ data/events/tokyodebian-XX.d` (XX は第 XX 回を示します) を作成します。

`http://debianmeeting.appspot.com/eventadmin/edit?eventid=` のフォームの編集はフォームに沿って情報を入れるだけです。ただ、細かい事は、 `http://tokyodebian.alioth.debian.org/YYYY-MM.html` へのリンクを URL の欄に指定し、そちらを参照してもらいます。

`http://tokyodebian.alioth.debian.org/` へは、 `http://qwik.jp/`メーリングリスト名 のトップページに記載されているやり方に従います。実際には git で `git+ssh://git.debian.org/git/tokyodebian/muse.git` リポジトリを clone して、 muse 形式で編集し、 make publish する事となります。



17 Debian 勉強会予約システム再訪

上川 純一

Debian 勉強会予約システムの開発開始から 2 年経ちました。当初は宴会君をリプレースするために突貫でつくりあげたものですがほぼそのままの状態で運営されています。面倒くさがって適当に実装した部分、ウェブアプリケーションで気をつけるべきところをよくわからずに書いていた部分があったのでいまから振り返ってどういう課題があったのかを検討します。

17.1 脆弱性

17.1.1 POST/GET の使い分け

当初めんどくさかったので post と get のハンドラを一緒にしてました。よって、あらゆるページが GET と POST の両方で動くようになっています。それを適切に GET と POST にわけるようにしました。

GET は情報の取得のみ、 POST は登録などの副作用のある処理に利用します。

ブラウザのセキュリティモデルとして、 same-origin じゃないと POST がやりにくいようになっています。 GET はたとえば script タグや img タグなどを埋め込んでおけばいくらでも発行できますが、 POST は XMLHttpRequest の発行が必要で、 XMLHttpRequest には same-origin policy があります。

以前は任意のページから script タグの埋め込みをして勝手に Debian 勉強会に登録することが可能でしたが、今はそれができないようになっています。これは勝手に予約する HTML ページの例です:

```
<html>
  <head>
    <title>auto reserve exploit</title>
    <script
 src="http://localhost:8080/eventregister?eventid=df24a1e1de11c067c461537dce6394e0e51df6ad
&user_prework=&user_attend=attend&user_enkai_attend=enkai_attend&user_realname=myname">
    </script>
  </head>

  <body>
    <h1>auto reserve exploit</h1>
    <p>
      ぼくはまちちゃん
    </p>
```

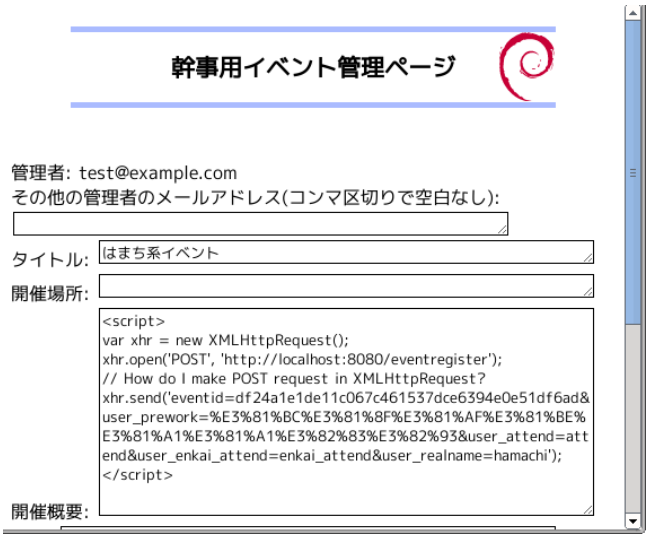
17.1.2 HTML escaping

当時よくわからなかったのとめんどくさかったので入力文字列のサニタイズとかエスケープとかまったくしてませんでした。関西 Debian 勉強会では HTML タグを勉強会の予約ページの案内文に入力していたようで、それでこの脆弱性がそもそも存在するのに思い至りました。

任意の文字列を HTML として出力できると、任意の javascript のコードを実行することができます。たとえば、説明ページを開いた瞬間に予約登録するイベントを作成することができます。

```
<script>
var xhr = new XMLHttpRequest();
xhr.open('POST', 'http://localhost:8080/eventregister');
xhr.withCredentials = true
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
xhr.send('eventid=df24a1e1de11c067c461537dce6394e0e51df6ad&'
+ 'user_prework=%E3%81%BC%E3%81%8F%E3%81%AF%E3%81%BE%E3%81%A1%E3%81%A1%E3%82%83%E3%82%93&'
+ 'user_attend=attend&user_enkai_attend=enkai_attend&user_realname=hamachi');
</script>
```

しかたがないので文字列をエスケープするようにしました。副作用として HTML タグを説明文に打ち込むことができないようになっています。



17.1.3 任意の URL 表示

Debian 勉強会予約システムでは説明文の表示が貧弱な代わりに任意の URL を iframe で埋め込み表示できるようにしています。通常管理している勉強会の Wiki ページを埋め込むことで二重に情報を管理しなくてすむようにという意図です。

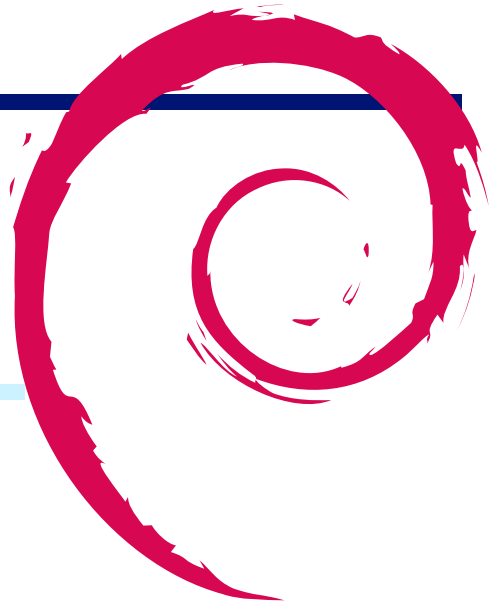
URL の中には特殊なものもあります。 `javascript:` で始まる URL だとその Javascript のコードを実行することができます。できても普通の役には立たなさそうなので `http://` 以外は許可しないようにしておきました。

また、 iframe buster と呼ばれる手法により iframe から脱出できるようで、そういうページを iframe の中でひらいているつもりになるとユーザが誤解する可能性があります。

すべてのブラウザでサポートされているわけではないのですが、 iframe に sandbox 属性をつけておき javascript などの実行を制約しておきました。

17.2 可能性と脆弱性のバランス

今回いろいろと脆弱性があったのでふさいでみました。しかし、脆弱性を塞ぐのも、ありうる被害とのバランスで考えるべきだと思います。 Debian 勉強会に提出する事前課題が漏洩する、勝手に宴会に参加することになっている等の程度の脆弱性と、自由に HTML が記述できる自由とどちらが重要かと考えてみてください。すこし悩ましいと思いませんか?



18 quilt で porting してみた

杉本典充

18.1 はじめに

debian のパッケージ作成で使用しているパッチ管理ツール quilt について説明します。また quilt を用いて kFreeBSD の porting パッチを作成しましたので、これについても説明します。

18.2 Debian パッケージのフォーマット

現在 Debian で使用している主なソースパッケージフォーマットは 2 つです。[*50][*51]

- 3.0(native)： tarball 1 つで構成されたソースパッケージフォーマット
  - packagename-version.tar.ext
  - packagename-version.dsc
- 3.0(quilt)： あるアップストリームの tarball と Debian パッケージ作成に必要な tarball に分割しているソースパッケージフォーマット
  - packagename-upstreamversion.orig.tar.ext
  - packagename-upstreamversion.orig-component.tar.ext（ 任意）
  - packagename-debianversion.debian.tar.ext
  - packagename-debianversion.dsc

ソースパッケージフォーマットの詳細は ‘dpkg-source(1)’ コマンドで確認することができます。[*52]

アップストリームのソースコードを無修正のまま Debian パッケージを作成した場合、 lintian 処理でエラーになったり、 Debian の開発ポリシーを満たせない場合があります(アップストリームのソースコードは IPv6 で動作しない、など）。そのときは、 3.0(quilt) のソースパッケージフォーマットでパッケージを作成し、必要なパッチを当ててからソフトウェアのビルド処理を行うことで対応させます。

18.3 quilt パッケージ

quilt は debian パッケージを作成するときにパッチファイルを管理するツールとして採用しているソフトウェアです。Debian パッケージとして提供しており、 apt でインストールできます。

```
# apt-get update
# apt-get install quilt
```

[*50] 東京エリア Debian 勉強会 2010 年 03 月号「 dpkg ソース形式 ”3.0(quilt)”」 吉野与志仁

[*51] http://wiki.debian.org/Projects/DebSrc3.0

[*52] man によると 3.0(custom)、 3.0(git)、 3.0(bzr) というパッケージ形式の定義があるようです。

quilt コマンドは複数の patch ファイルをスタックに積んだようなイメージで、パッチの適用順番を管理します。[*53]quilt コマンドを使うことで、パッケージ作成に必要なパッチが複数ある場合も正しく適用でき、不要なパッチが出てきたときにそのパッチを取り除くことが容易になります。

18.4 quilt の使用例: Debian GNU/kFreeBSD 向けの porting パッチ作成

18.4.1 Debian GNU/kFreeBSD とは

Debian Project で開発している OS は Linux カーネルを用いた「Debian GNU/Linux」が有名ですが、Linux カーネル以外を用いた Debian があります。その 1 つとして FreeBSD カーネルを用いた「Debian GNU/kFreeBSD」[*54]があり、ユーザランドは Debian なので APT が使え、デバイスやシステムコールといったカーネル特有の機能は FreeBSD カーネルに準じる、という特徴があります。Debuan GNU/kFreeBSD は安定版のリリースには至っていませんが、日々開発が続けられています。本稿では以降「Debian GNU/kFreeBSD」を「kfreebsd」と略記します。

18.4.2 kfreebsd の porting 作法

Debian の porting 関連情報および kfreebsd の porting 情報は以下にあります。

- `http://www.debian.org/ports/`
- `http://www.debian.org/ports/kfreebsd-gnu/`
- `http://glibc-bsd.alioth.debian.org/porting/`

「`http://glibc-bsd.alioth.debian.org/porting/PORTING`」を読むと kfreebsd 向けに porting するには以下の確認および修正が行うようにとあります。

- Add our system name to checks here and there
  - Makefile やスクリプト中の uname などをチェックしてください。
- debian/control files
  - debian/control ファイルで Architecture を linux 専用ソフトウェアの場合は「linux-any」等、CPU の違いのみで Linux、kfreebsd は関係なく使用できるソフトウェアの場合は「any-i386」等に変更してください。
- Libraries, your beloved enemy
  - libtool、aclocal.m4 周りに対応してください。
- Preprocessor Variables
  - kfreebsd のシステムマクロは「__FreeBSD_kernel__」、バージョンマクロは「__FreeBSD_kernel_version」なので対応してください。
- Writing to devfs (kFreeBSD)
  - FreeBSD カーネルでは(udev ではなく) devfs を使うようにしてください。
- RT signals
  - FreeBSD カーネルは「POSIX RT (realtime) signals」がないので変更してください。
- Get libc soname (6 or 6.1 on linux-gnu, 0.1 on kfreebsd-gnu, etc)
  - 使用する libc の名前をハードコードしているプログラムがあれば修正してください。

18.4.3 kfreebsd に対応させたいパッケージとその原因

今回 porting するパッケージは既に kfreebsd 用パッケージとして存在している icewm となります。icewm はウィンドウマネージャのソフトウェアで軽快な動作をするのが特徴です。icewm ではタスクバーにバッテリー残量アイコンを表示する機能がありますが kfreebsd では表示されません。linux-i386 及び linux-amd64 では表示されるため kfreebsd の

[*53] 東京エリア Debian 勉強会 2007 年 01 月号「パッチ管理ツール quilt の使い方」小林儀匡
[*54] `http://wiki.debian.org/Debian_GNU/kFreeBSD`

porting が不完全の可能性があります。

まずはビルド準備とソースコードのダウンロードを行います。

```
# apt-get update
# apt-get build-dep icewm
$ apt-get source icewm
```

ソースコードを「 __FreeBSD__」及び「 __linux__」 grep すると、電源周りの処理で以下のマクロが検出されます。

```
$ cd icewm-1.3.7/src
$ grep -nr __FreeBSD__ *
aapm.cc:30:#ifdef __FreeBSD__
aapm.cc:74:#if defined(__FreeBSD__) && defined(i386)
aapm.cc:99:#if defined(__FreeBSD__) && defined(i386)
aapm.cc:273:#ifndef __FreeBSD__
aapm.cc:333:#ifndef __FreeBSD__
aapm.cc:418:#ifndef __FreeBSD__
aapm.cc:463:#ifdef __FreeBSD__
aapm.cc:885:#ifndef __FreeBSD__
aapm.h:2:#if defined(linux) || (defined (__FreeBSD__)) || (defined(__NetBSD__) && defined(i386))
( 以下略)
$ grep -nr __linux__ *
(なにもなし)
```

__FreeBSD__マクロは FreeBSD OS を示すマクロですが、 kfreebsd では「 __FreeBSD_kernel__」がシステム(厳密にはカーネル）を示すマクロです。そのためマクロでシステムを切り分けているはずが kfreebsd 向けのパッケージビルド時に Linux カーネル向けの処理が有効なコードとしてビルドされてしまいバッテリー残量アイコンの表示がうまく動作していないようです。

そのため、今回はこのマクロを「 PORTING」の記述に従って以下のような修正を行います。

```
( 修正前) #ifdef __FreeBSD__

( 修正後) #if defined(__FreeBSD__) || defined(__FreeBSD_kernel__)
```

これで修正方針が定まりましたので、 debian パッケージ作成に向けてパッチを作成していきます。

18.5 quilt の登場

今回は新規のパッチファイルとなるため、 quilt のパッチスタックに新規追加します。まず、 debian/patches/series 記述されておるパッチスタックを見てみます。

```
$ tail -4 debian/patches/series
tray_hotfixes
imap_unseen
ifstate_exact_check
debian-changes-1.3.7~pre2-1.1
```

次に `quilt new` を実行し、現在の変更内容を kfreebsd_porting_aapm というファイル名で保存し、スタックに追加します。

```
$ quilt new kfreebsd_porting_aapm
Patch kfreebsd_porting_aapm is now on top
```

実行下後のパッチスタックは以下のようになります。

```
$ tail -4 debian/patches/series
imap_unseen
ifstate_exact_check
debian-changes-1.3.7~pre2-1.1
kfreebsd_porting_aapm
```

これでパッチを追加できる準備ができました。次に porting するために修正を行うソースファイルを quilt で管理するように登録処理をします。その後「 quilt edit ソースファイル」を実行すると、環境変数 EDITOR で登録したエディタが自動で起動しますのでソースファイルの修正作業を行います。

```
$ quilt add src/aapm.h
File src/aapm.h added to patch kfreebsd_porting_aapm
$ quilt edit src/aapm.h
File src/aapm.h is already in patch kfreebsd_porting_aapm
$ quilt refresh
Refreshed patch kfreebsd_porting_aapm
$ quilt add src/aapm.cc
File src/aapm.cc added to patch kfreebsd_porting_aapm
$ quilt edit src/aapm.cc
File src/aapm.cc is already in patch kfreebsd_porting_aapm
$ quilt refresh
Refreshed patch kfreebsd_porting_aapm
```

できたパッチファイル「kfreebsd_porting_aapm」を確認します。

```
$ cat debian/patches/kfreebsd_porting_aapm
Index: icewm-1.3.7/src/aapm.h
===================================================================
--- icewm-1.3.7.orig/src/aapm.h 2010-10-31 23:09:36.000000000 +0900
+++ icewm-1.3.7/src/aapm.h      2011-12-10 23:17:15.000000000 +0900
@@ -1,10 +1,10 @@

-#if defined(linux) || (defined (__FreeBSD__)) || (defined(__NetBSD__) && defined(i386))
+#if defined(linux) || (defined (__FreeBSD__)) || (defined (__FreeBSD_kernel__)) || ((defined(__NetBSD__) && defined(i386))

 #include "ywindow.h"
 #include "ytimer.h"

-#if defined(__FreeBSD__) || defined(__NetBSD__) || defined(__OpenBSD__)
+#if defined(__FreeBSD__) || defined(__FreeBSD_kernel__) || defined(__NetBSD__) || defined(__OpenBSD__)
 #define APMDEV "/dev/apm"
 #else
 #define APMDEV "/proc/apm"
(以下略)
```

あとはパッケージをビルドします。

```
$ dch
$ debuild -uc -us
```

作成したパッケージをインストールして確認します。

```
$ sudo dpkg -i icewm-common_1.3.7-1.1_kfreebsd-amd64.deb icewm_1.3.7-1.1_kfreebsd-amd64.deb
$ reboot
```

作成したパッチはバグレポートと共に BTS へ送信しておきましょう。

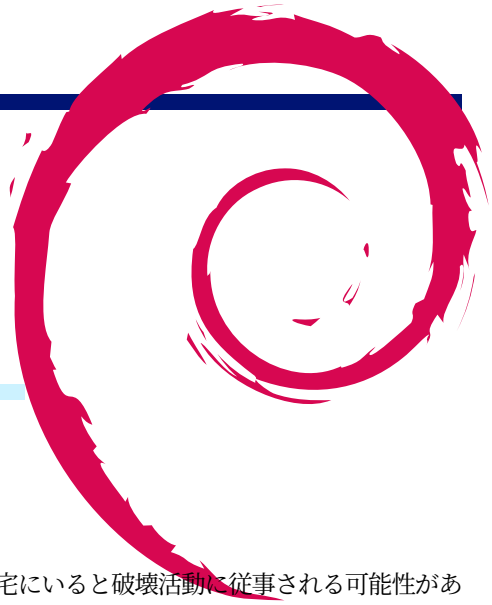
```
$ reportbug
$ w3m http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=650395
```

18.6 終わりに

これで kfreebsd の icewm ウィンドウマネージャ上でバッテリー残量アイコンを表示することができました。
みなさんも kfreebsd 含め様々なアーキテクチャで数多くのパッケージが Debian で動作するようにがんばりましょう。

参考文献

[1] 東京エリア Debian 勉強会 2007 年 01 月号「パッチ管理ツール quilt の使い方」小林儀匡
[2] 東京エリア Debian 勉強会 2010 年 03 月号「dpkg ソース形式”3.0(quilt)”」 吉野与志仁
[3] man dpkg-source(1), man quilt(1)
[4] man quilt(1)
[5] DebSrc 3.0 http://wiki.debian.org/Projects/DebSrc3.0
[6] Maint Guide 日本語版 http://www.debian.org/doc/manuals/maint-guide/first.ja.html
[7] kFreeBSD wiki http://wiki.debian.org/Debian_GNU/kFreeBSD
[8] porting glibc to BSD http://glibc-bsd.alioth.debian.org/porting/



19 Debian の使える VPS を使ってみた

上川 純一

自宅サーバ最強だと思っていた時期が僕にもありました、が幼少な子供が自宅にいると破壊活動に従事される可能性があり、自宅は設置したマシンを安定してサーバとして稼働させるのに適した環境とはいえません。また、つけっぱなしにしていると音がうるさいので静音 PC 化とかにこだわるのですが、そろそろそれにも飽きてきました。静音のためにファンレスにしとくと夏気づいたら熱暴走してます。また、 PC は自宅の消費電力の中でも大きな割合を占めてました。

しかし、常時走っているノードがないと、リグレッションテストを走らせることすらできません。我慢できなくなってきたので、 Debian ノードで好きなようにいじれるようなサービスを探して試してみました。 2012 年 1 月 15 日時点の情報です。[*55]近年 IaaS クラウドとかいうバズワードがあってよくわからなくなっているのですが、ここで僕の欲しいサービスは多分仮想プライベートサーバ(略して VPS）です。

19.1 さくらインターネットの VPS

KVM のノードがひとつ手に入ります。

デフォルトでは Cent OS がはいった状態で起動するのですが、 Debian をインストールするメニューがブラウザで利用する管理画面で選択できます。 Debian のインストールを選択すると、 Debian Installer が起動し、 Java Applet の VNC クライアント経由でコンソール画面を見ながらインストールします。

良くも悪くも Debian Installer ですが、一部カスタマイズされているようで、例えば、 ssh がデフォルトでインストールされるようになっています。公開鍵認証にしたい場合はあとで自分で ssh の公開鍵を登録すればいいと思います。

ブラウザで利用する管理画面で再起動とか VNC でコンソール画面に接続するなどのメンテナンスが行えます。

使用料金は「 さくらの VPS 512」で 980 円/月です。支払いはクレジットカード以外の方法もありますが、クレジットカードだと二週間の無料お試しが可能なようです。利用を開始すると住所の確認のためにハガキが送られてきてその中に書いてあるお知らせ番号を入力させられます。日本国内に住所があることが利用の条件です。

19.2 Amazon AWS EC2

Xen のノードがひとつ手にはりいます。既存の Debian の AMI [*56][4] をクローンする感じでインスタンスが立ち上がります。自分で AMI をつくってもいいみたいです。

ブラウザで利用する管理画面からインスタンスの再起動などが行えます。

ブート時のログが制御コンソールから見れますが、コンソール接続する方法は見つけられませんでした。起動に失敗したら AMI からつくりなおせばいいという発想なんでしょうか。

初回時の接続は公開鍵認証の SSH で root ユーザとして接続します。ログインはインスタンスを作成するウェブインタフェースにて作成した公開鍵認証です。

[*55] 2012 年 6 月現在: 半年後みたらまた状況が変わってたので当時の状況の参考程度の情報です

[*56] Amazon Machine Image: / のファイルシステムイメージのようです

使用料金は一時間あたり 10 セント程度で東京リージョンのスモールインスタンスが借りれます。

最初の開始の利用手続きでは、 Amazon.com のアカウントをつかって登録します、その際に電話番号を確認のために電話がかかってきます。クレジットカードが必要です。

19.3 S@@Ses

Xen ベースでの VPS サービスを提供しています。 LT サーバであれば月 450 円で利用できるようです。初期費用 3000 円かかるというのと 3 ヶ月単位の契約だというので僕は試してません。最初に二週間のお試し期間があるようです。

ブラウザで利用する管理画面からインスタンスの再起動などが行えます。

OS の初期化メニューからの OS の選択肢に Lenny や Squeeze があり、それを選択するとインストール済みの Xen のイメージが立ち上がり、 root で ssh ログインできるようになります。

19.4 まとめ

参考のため、各社の手頃っぽいエントリーモデルを並べてみました[*57]。

	さくらの VPS 512	AWS EC2 Small	S@@ses LT
使用料金	980 円 / 月	$0.10/時間	450 / 月 (3 ヶ月単位) + 3000 円初期料金
CPU	仮想 2 core	1 ECU	2.66GHz
メモリ	512MB	1.7GB	512MB
ディスク	20GB	160GB	50GB
仮想化技術	KVM	Xen	Xen
コンソールアクセス	VNC 経由	?	?
Debian 利用	ブラウザで利用する管理画面に D-I 起動するメニューあり、 VNC 経由で D-I からインストール	Debian AMI をさがしてきてブラウザで利用する管理画面から起動すると root に ssh 可能なインスタンス	ブラウザで利用する管理画面のメニューから初期化を選択すると root に ssh 可能な状態のインスタンス

19.5 さいごに

ほかにもいろいろサービスがあるみたいですが、 Debian が利用できる VPS サービスが日本でも充実してきたのだなという感想です。

各社ストレージ構成がどうなっているのかバックアップ体制がどうなっているのかなどの記述がみあたらないので、障害発生の確率とか障害発生時の対応がどうなのか予想もつきません。そういう情報は価格競争が一旦収束して、業界が成熟してから重要になるのかもしれません。

個人的にはとりあえずさくらインターネットを使ってみることにしてみました。

参考文献

[1] Amazon Web Services `http://aws.amazon.com/jp/`

[2] VPS(仮想専用サーバ) のさくらインターネット `http://vps.sakura.ad.jp/`

[3] SaaSes `http://www.saases.jp/`

[4] Debian Wiki: Cloud Amazon EC2 Image `http://wiki.debian.org/Cloud/AmazonEC2Image`

[*57] AWS EC2 が高く見えるけど、 micro instance の spot instance にすればもっと安いです。

20 Debian で twitter 連携

岩松 信洋

Debian の作業内容を Twitter に投げていたら 上川さんにどんなことやっているのか紹介して欲しいとの連絡がありました。今回は Debian からどのようにして Twitter を使っているのか、使うにはどのようにしたらいいのか説明します。

20.1 Twitter API について

Twitter は、 ユーザーが「 ツイート」と呼ばれる 140 文字の「 つぶやき」を投稿し、そのツィートを閲覧したりツィートに対してさらにツィートしたりなど、コミュニケーションするためのサービスです。 Twitter ではこの「 ツィート」を投稿、削除、参照、検索などをプログラムから行えるように API を公開しています。これを Twitter API といいます。 Twitter がサービスとして公開しており、利用するためには利用規約に同意する必要があります。

また Twitter API は大きく分けて、 REST、 Search、 Streaming の 3 種類があります。 REST API はツイートの更新や参照などを行う基本的な API、 Search API はツイートを検索する API、 Streaming API はタイムラインをリアルタイムに受け取るための API です。また、これらの API を使うためにはアクセス用の ID が必要で、利用できる API の回数なども決まっています。 API を使うにはこの ID を使った認証処理が必要になります。 API をラッパーし、 Twitter API を使いやすくするための Twitter API 用のライブラリがいくつか存在します。

20.2 Debian での Twitter API サポート状況

まず、 Debian での twitter 周りの整備状況を確認してみます。言語毎に Twitter API 用のパッケージが整備され、表 11 のようにパッケージが提供されています。メジャーな言語ではいくつかライブラリがあるようですが、手の足りていない言語チームでは整備が遅れているようです。興味のある方はメンテナンスに参加してみてはいかがでしょうか。

表 11 Debian で提供されている各言語用のパッケージ

言語	パッケージ名
C, C++	libsocialweb, bitlbee, etc.
Perl	libnet-twitter-lite-perl, libnet-twitter-perl
Python	python-twyt, python-tweepy, python-twitter
Ruby	libtwitter-ruby1.x
Haskell	なし(パッケージになってない)
OCaml	なし(パッケージになってない)

20.3 Debian から Twitter API を使ってみる

次に Ruby の Twitter API 用ライブラリを使った簡単な例を紹介します。例えば、「 test」というツィートをポストするアプリケーションを作成するには以下の順番で行います。

20.3.1 Twitter アプリケーション登録申請

`https://dev.twitter.com/` にアクセスし、 OAuth を利用するにあたり必要となる、 Consumer key、 Consumer secret 等を取得します。これらのデータは Twitter アプリケーション毎に異なります。アプリケーション名を「 api-test」とした場合、以下のような内容で適当なファイルに保存します。

```
api-test:
    login: iwamatsu
    oauth_consumer:
        key: XXXXXX
        secret: XXXXX
    oauth_access:
        key: XXXXX
        secret: XXXXX
```

20.3.2 パッケージをインストールする

インストールは apt-get で行えます。

```
$ sudo apt-get install libtwitter-ruby1.9.1
```

20.3.3 Twitter API ライブラリを使ったソースコードと実行

コードは以下のようになります。 Twitter アプリケーション登録申請したときに取得した「 Consumer key」等を保存したファイルを「 /home/hoge/.twitter.yml」、アプリケーション名として指定しインスタンスを生成します。そして status メソッドで「 test」をポストするようにします。

```
$ cat test.rb
#!/usr/bin/ruby

require 'twitter'
twitter = Twitter::Client.from_config("/home/hoge/.twitter.yml", "api-test")
twitter.status(:post, "test");

$ ruby ./test.rb
```

以上が Debian から Twitter API を使う例となります。

20.4 私が使っているツール紹介

Twitter はメモや作業内容などを通知をする場合に非常に便利なツールです。 Debian の作業内容などを通知できないかなと思っていくつか Twitter 用ツールを作成したので紹介します。

20.4.1 Debian Hack Cafe 通知ツール

毎週東京/関西のどこかで行われていると言われている Debian Hack Cafe。 Hack Cafe 開催通知を行うためのアカウントとして @debian_hackcafe があります。これは Debian Hack Cafe GPG キーリングに登録された人なら誰でもつぶやけるという特徴があります。つぶやく場合、 libwww-perl パッケージに含まれる lwp-request を使ってつぶやきをサーバに POST するとサーバで処理が行われ、問題がない場合 @debian_hackcafe アカウントとしてつぶやきます。

```
$ sudo apt-get install libwww-perl
$ echo "つぶやき" | gpg --clearsign | \
  lwp-request -m POST http://www.nigauri.org/debian_hackcafe_post
```

このシステムは以下のように処理されます。

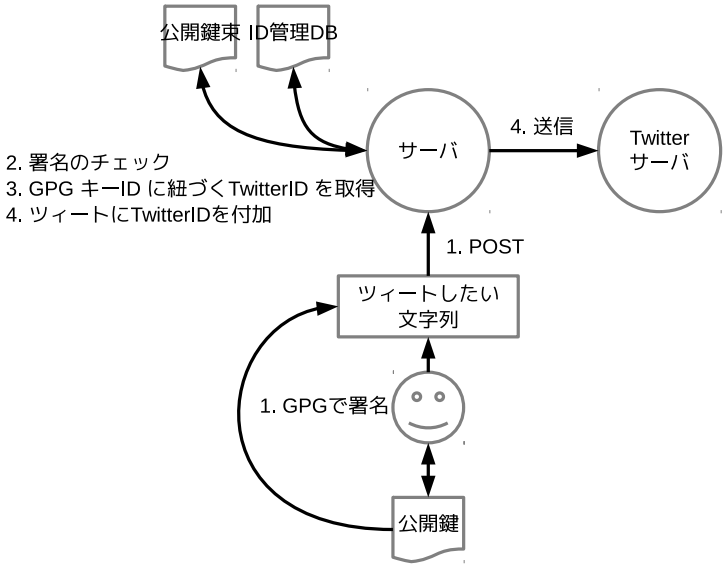


図 17　Debian Hack Cafe 通知ツール構成図

1. つぶやきを GPG サインしてサーバにポスト
2. サーバで鍵のサインをチェック
3. 署名から GPG キー ID を取得し、 ID 管理 DB から GPG キー ID に紐づく TwitterID を取得
4. つぶやきに ID をいれて Twitter API を使ってつぶやく

このようにすることで Twitter のアカウントとパスワードを共有することなく、限定されたされたメンバで、つぶやくことができます。 PGP/GnuPG を使って署名チェックするなんて Debian らしくてかっこいい! と個人的に思っています。

また、このシステムを Debian JP アカウント(@debianjp) 用にアップデートし、運営できるようにする予定です(今までは運営している人たちでアカウントとパスワードを共有していたようです) 。また Debian JP の誰がつぶやいていたのかわからないという問題も解決する予定です。

20.4.2　パッケージがアップロードされたらつぶやく dput-tweet

最近スポンサーアップロードを行う事が多くなりました。またスポンサーしている人は Twitter のアカウントを持っているので、アップロードした旨を通知する方法の一つとして Twitter を使うようにしました。この通知を行うツールが dput-tweet です。 Ruby の勉強用に作ったツールで、 dput のラッパーになっており、 dput したらパッケージ名とバージョンスポンサーした人の TwitterID をつぶやくというものです(図 18) 。このツールのいけてない点として、 dput しか対応できてない事と実行時にスポンサーする人の Twitter ID を指定する必要がある事です。

```
$ dput-tweet -s mkouhei ordereddict_1.1-1_amd64.changes
....
「dput ordereddict_1.1-1 @mkouhei [dput-tweet] 」とつぶやきます。
```

もっと楽をしたいと思ったので、今は inotfy を使って upload ファイルが作成されたらつぶやく機構にしました。これによって dput / dupload の両方に対応できます。また upload ファイルから changes ファイルを抽出し、 Changed-By の行から 得たメールアドレスを元に TwitterID を DB から取得し、つぶやきに入れるようにしました(図 19。

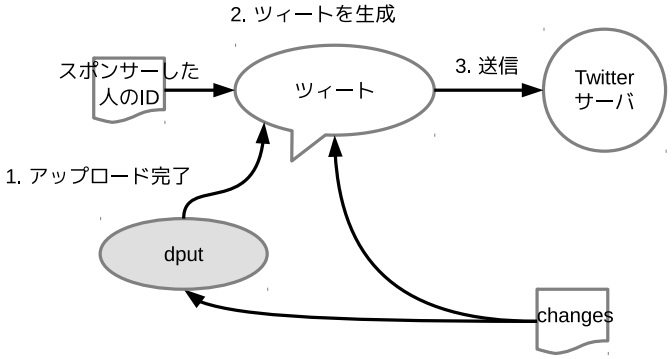


図 18　パッケージアップロード通知ツール dput-tweet v1

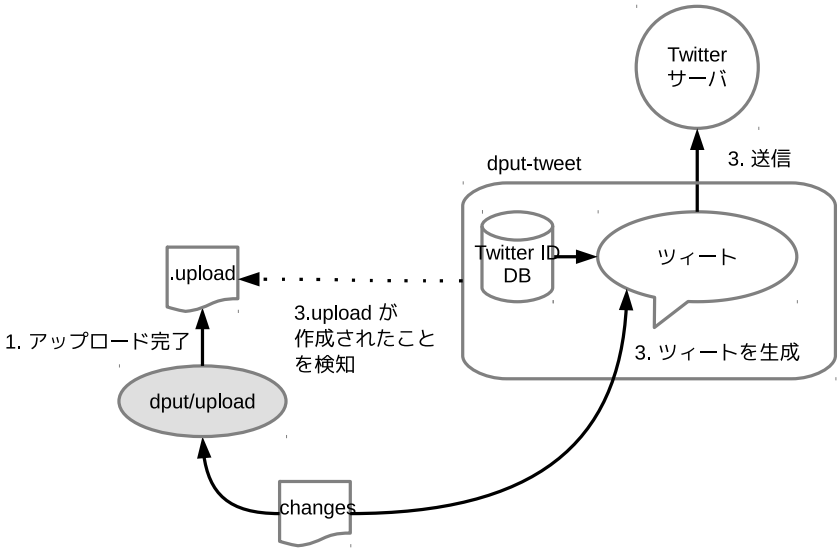
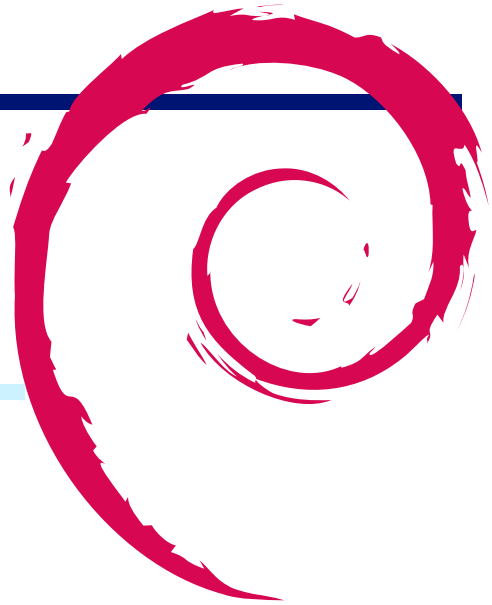


図 19　パッケージアップロード通知ツール dput-tweet v2

20.5　まとめ

いくつかのツールを作ってみて、Twitter API の使い方がわかってきました。今度は Debian JP メンバの活動 Tweet などができるようにしたり、Twitter だけでなく Facebook などのサービスの API についても調べてみようと思います。



21 月刊debhelper 第2回

野島 貴英

21.1 はじめに

Debian パッケージを作成する際、沢山の処理を debian/rules というファイルに GNU の make の makefile の形式で記述することになります。しかしながら、細かい処理を記載していくと膨大な量となってしまいます。これをできるだけ簡潔に記載できるように考えられたツールとして debhelper というコマンド群が存在します。

本企画はこの debhelper のコマンドについて、毎月持ち回りで解説していくというものです。ルールは、毎月 2 つ以上のコマンドを解説し、次回発表の立候補が無い場合は発表者が次の発表者を決めれるというルールの元に進めて行きます。

21.2 今月のコマンドその 1:dh

21.2.1 dh の動作概要

dh は引数に指定したシーケンス名に基づいて一連の debhelper を起動するコマンドとなります。実際の使い方では、以下の内容を debian/rules に記述して利用します。

```
$ cat debian/rules
#!/usr/bin/make -f
%:
        dh $@
```

21.2.2 dh に指定できるシーケンス名

dh に指定できるシーケンス名は表 12 の通りです。

なお、–with foo を指定すると、 dh に指定可能なシーケンスが増える場合があります (例: –with quilt の patch シーケンス等。)

21.2.3 dh のコマンドラインオプション

表 13 に dh のコマンドラインオプションを載せます。 (man dh より)

この他にも、 debhelper コマンド共通で使えるコマンドラインオプションが man debhelper に記載されており、 dh コマンドでも利用できます。こちらも参照ください。

21.2.4 廃止されたコマンドラインオプション

–until,–before,–after,–remaining がありましたが、これらは全部 dh が解釈する “override_*DH* **コマンド名**ターゲット” による動作に置き換えられた為、**廃止**となりました。

なので、昔の debian/rules にあるような、以下の用な書き方は**廃止**です。

シーケンス名	シーケンスの説明
binary	構築からパッケージ作成まで実行するシーケンスです。
binary-arch	arch 依存のパッケージの構築からパッケージ作成まで実行するシーケンスです。
binary-indep	arch 非依存のパッケージの構築からパッケージ作成まで実行するシーケンスです。
build	構築からテストまで実行するシーケンスです。
build-arch	arch 依存のパッケージの構築からパッケージ作成まで実行するシーケンスです。
build-indep	arch 非依存のパッケージの構築からパッケージ作成まで実行するシーケンスです。
clean	一度パッケージを構築したディレクトリから、パッケージ構築時に生成したものを取り除き、構築ディレクトリを綺麗にします。
install	構築から、パッケージ生成直前までの処理を行うシーケンスです。
install-arch	arch 依存のパッケージについて、構築から、パッケージ生成直前までの処理を行うシーケンスです。
install-indep	arch 非依存のパッケージについて、構築から、パッケージ生成直前までの処理を行うシーケンスです。

表 12　dh で指定できるシーケンス名一覧

オプション	説明
–with addon[,addon ...]	debhelper コマンドに適切な場所で一連のコマンドを実行するような付加機能 (addon) を指定します。
–without addon	–with とは逆の働きをします。指定された付加機能を使わないようにします。
–list, -l	利用可能な付加機能 (addon) 一覧です。
–no-act	指定された一連の処理の内容を表示するだけコマンドとなります。表示だけして実際にはコマンドを実行しません。
その他	dh に、先に記載した以外の何かオプションを渡すとそれはのちに実行する全コマンドへ引き渡されます。-v、-X、-N や、他の特別なオプションを指定するのに使われます。

表 13　コマンドラインオプション一覧

廃止された書き方

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

%:
        dh $@

build: build-stamp
build-stamp:
        dh build --before configure
        dh_auto_configure -- --with-gnu-ld --disable-nls
        dh build --after configure
        touch build-stamp
```

代わりの書き方は次の章で述べます。

21.2.5　“override_*debhelper* コマンド名” ターゲットについて

dh コマンドは”dh シーケンス名” により、そのシーケンスに必要な一連の debhelper コマンドを呼び出す機能があります。(どんな debhelper コマンドが呼び出されるかは、–no-act をオプションにつけて、 dh –no-act build とか、 dh –no-act install とかして見てください)

この呼び出されるコマンドを一部変更したい場合は以下のように書きます。

今時の書き方:

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

%:
        dh $@

override_dh_autoconfigre:
        dh_auto_configure -- --with-gnu-ld --disable-nls
```

こうすると、本来であれば、 dh_auto_configure がオプション無しで呼び出される場所が全部 “dh_auto_configure –

–with-gnu-ld –disable-nls” で呼び出されるようになります。

他の例として、 configure スクリプトが無く、代わりに Imakefile があるような古い X 用のプログラムをパッケージにする用な場合は以下のように書きます。

Imakefile を利用するような場合:

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

%:
        dh $@ --with quilt

override_dh_auto_configure:
        xmkmf -a
```

こうすると、本来であれば、 dh_auto_configure が呼び出される場所全部で、 “xmkmf -a” を呼び出すようになります。

この”override_*debhelper* **コマンド名**” ターゲットは、コマンドを**実行したくない場合**にも利用可能です。 (“override_*debhelper* **コマンド名**” のアクションを空にする事がミソです。)

dh_auto_test,dh_compress,dh_fixperms を実行したく無い場合:

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

%:
        dh $@

override_dh_auto_test override_dh_compress override_dh_fixperms:
```

また、 build-arch,binary-arch,build-indep,binary-indep ターゲットが dh に指定されるときにあわせて振る舞いを変更したい場合は”override_*debhelper* **コマンド名**-indep” や、”override_*debhelper* **コマンド名**-arch” を使って、それぞれの場合に dh によって呼び出されるコマンドを変更できます。以下の例では、ドキュメントパッケージの作成に時間がかかるので、 build-indep や、 binary-indep の時にだけドキュメントを作成してくれるようにする場合の debian/rules となります。

ドキュメント作成を分離して、時間のかかるドキュメント作成が何度も実行されないようにする:

```
#!/usr/bin/make -f
%:
        dh $@

override_dh_auto_build-indep:
        $(MAKE) -C docs

# No tests needed for docs
override_dh_auto_test-indep:

override_dh_auto_install-indep:
        $(MAKE) -C docs install
```

21.2.6 addon について

dh コマンドのオプション–with addon にて addon が提供するパッケージの作成方法を組み込む事ができます。お使いのシステムで現在どんな addon が使えるかは dh –list を実行すると一覧が出てきます。

```
$dh --list
bash-completion
dkms
python-central
python-support
python2
quilt
tex
$
( ... お使いのシステムによって表示される量が変わります...)
```

実はこれらは/usr/share/perl5/Debian/Debhelper/Sequence/”addon 名”.pm としてインストールされています。こちらを利用して、現在の Debian で、どんな addon が提供されているかを知りたければ、次のようにして調べる事ができます。

```
$ apt-file search Debhelper/Sqeuence
autotools-dev: /usr/share/perl5/Debian/Debhelper/Sequence/autotools_dev.pm
bash-completion: /usr/share/perl5/Debian/Debhelper/Sequence/bash_completion.pm
cli-common-dev: /usr/share/perl5/Debian/Debhelper/Sequence/cli.pm
... 中略...
sphinx-common: /usr/share/perl5/Debian/Debhelper/Sequence/sphinxdoc.pm
tex-common: /usr/share/perl5/Debian/Debhelper/Sequence/tex.pm
xserver-xorg-dev: /usr/share/perl5/Debian/Debhelper/Sequence/xsf.pm
xulrunner-dev: /usr/share/perl5/Debian/Debhelper/Sequence/xulrunner.pm
$ apt-file search Debhelper/Sqeuence | wc -l
43
$
```

全部で 43 個もありますね。(debian sid で実行)

複数 addon を指定したい場合は繰り返し–with オプションで指定したり、カンマで区切って指定します。

```
quilt 用の addon と、 autotools_dev 用の addon を併用したい時:
#!/usr/bin/make -f
%:
        dh $@ --with quilt --with autotools_dev
#       dh $@ --with quilt,autotools_dev   も OK
```

21.2.7 addon の構造について

addon が何をしているかは/usr/share/perl5/Debian/Debhelper/Sequence/”addon 名”.pm を覗くとピンときます。

例えば、–with quilt の場合、

1. dh clean にて、 dh_clean を呼び出す前に、 quilt パッケージが一緒に提供している dh_quilt_unpatch コマンドを呼び出すようになります。
2. dh build では、 dh_auto_configure の前に dh_quilt_patch を呼び出すようになります。
3. dh にシーケンス名 patch が追加され、 dh patch が使えるようになります。

addon を自分で書く場合は、 dh 内で定義されている表 14 の API を呼び出して書いてください。

API 名	API の説明
insert_before($existing,$new)	$existing で指定される debhelper コマンドを実行する直前に$new を実行します。
insert_after($existing,$new)	$existing で指定される debhelper コマンドを実行した直後に$new を実行します。
remove_command($command)	$command を dh が実行しないようにします。
add_command($command,$sequence)	$sequence で示されるシーケンスで実行されるコマンド群の最後に$command を付け加えます。また、本 API を使って 21.2.2 章で示されないシーケンスを新たに作成することができます。
add_command_options($command,@options)	$command に、配列@options で示される一連のオプションを付け加えて実行するようにします。
remove_command_options ($command,@options)	$command から配列@options で示される一連のオプションを取り除く。@options をまったく指定せずに remove_command_options($command) と呼び出すと、$command についてのオプション全部を取り除きます。

表 14　addon 用の API 一覧

量もそんなになく、非常にわかりやすいので、興味のある人は/usr/share/perl5/Debian/Debhelper/ Sequence/quilt.pm を試しに読んでみるとよいと思います。

なお、複数の addon を指定した場合、同じ内容の debhelper コマンドが意図せず複数回も同じシーケンスに挿入される事がありますが、きちんと 1 個の呼び出しにまとめてくれます。

21.2.8 dh の内部動作

dh コマンドは debian/rules が make ファイルである事を利用しながら、 make コマンドと協調して動作します。

図 20 に dpkg-buildpackage を呼び出したときの dh の内部動作を示します。

図 20 から判るように、 dpkg-buildpackage から make コマンドが起動され、次に dh が起動され、さらに dh から make が起動されるという関係になっている事が判ります。また、 override_*debhelper* **コマンド名**ターゲットの処理を行うのに、 make コマンドを使って処理をしているという事も判ります。

dh を使うと、 make コマンドは override_*debhelper* **コマンド名**ターゲットの処理をする役目だけを担当します。そのうち、 dh コマンドが進化すると、 make コマンドの力を借りなくてもパッケージ作成ができるようになるかもしれませんね。

21.2.9 “debian/**パッケージ名**.debhelper.log” ファイルについて

最近の dh コマンドを使う debian/rules には、ファイルの依存関係についての記載がありません。この為、パッケージビルド中で処理が中断した場合、どこから再開すれば良いかを debian/rules で make が判定する事はできません。

実は、 dh コマンドは clean 以外のシーケンスが指定されると、”debian/パッケージ名.debhelper.log” というファイルに処理を行った debhelper コマンドを記録しています。ここで、万一 dh の処理が中断した場合、処理をどこから始めれば良いかについてはこのログファイルを参照して処理の再開を行います。

“debian/パッケージ名.debhelper.log” の中身は以下のようになっています。

```
debian/パッケージ名.debhelper.log の中身:
dh_auto_test
dh_prep
dh_installdirs
... 中略...
dh_buiddeb
```

また、このファイルは dh clean によって消去されます。なお、 dh clean の時には、このログファイルは作成されません。つまり、 dh clean の処理を中断した場合は、 dh clean は呼び出される一連の debhelper コマンドは最初から実行されてしまいます。

なお、処理再開の場所は、このログファイルのみ参照して決める為、処理を中断した後に、パッケージのソースファイルを変更して再開させるような使い方はできません。例えば、ソースファイル中のあるファイルを変更した為、特定のパッケージのシーケンスについては再会時に全部やり直しが必要だったとしても、これを自動で検知することはできません。

21.2.10 dpkg-buildflag との関係

dh は互換性度合い (COMPATABLITY LEVEL) の v9 から、パッケージ構築の時に使う環境変数を設定するため、内部で dpkg-buildflag 相当の処理を呼び出します。

その為、 9 を debian/compat に指定すると、 debhelper コマンドに設定される環境変数は、

1. /etc/dpkg/buildflags.conf の中身
2. XDG_CONFIG_HOME/dpkg/buildflags.conf (XDG_CONFIG_HOME は環境変数です) の中身
3. HOME/.config/dpkg/buildflags.conf (HOME は環境変数です) の中身
4. DEB_flag_MAINT_SET, DEB_flag_MAINT_STRIP, DEB_flag_MAINT_APPEND, DEB_flag_MAINT_PREPEND, DEB_BUILD_MAINT_OPTINS（ 全部環境変数です）の値

により様々に変化します。どのように変わるかは man dpkg-buildflag を参照してください。

21.3 今月のコマンドその 2:dh_testroot

21.3.1 dh_testroot 動作詳細

現在の実行ユーザが root であるかどうかを確認するコマンドです。 root ユーザでは無い場合、エラーメッセージを出力して処理を中断します。

21.3.2 dh_testroot コマンドラインオプション

コマンドラインオプションは特にありません。何か指定しても無視されます。

21.3.3 dh_testroot を実行してみる

早速、実行してみましょう。

```
$ sudo dh_testroot
$ echo $?
0
$ dh_testroot
You must run this as root (or use fakeroot).
$ echo $?
255
$ fakeroot dh_testroot
$ echo $?
0
```

このように root 権限で実行するか、 fakeroot 経由で実行した時のみ 0 を返却します。

21.3.4 次回の発表について

次の発表者は勉強会で発表します。選ばれた人はよろしくおねがいします。

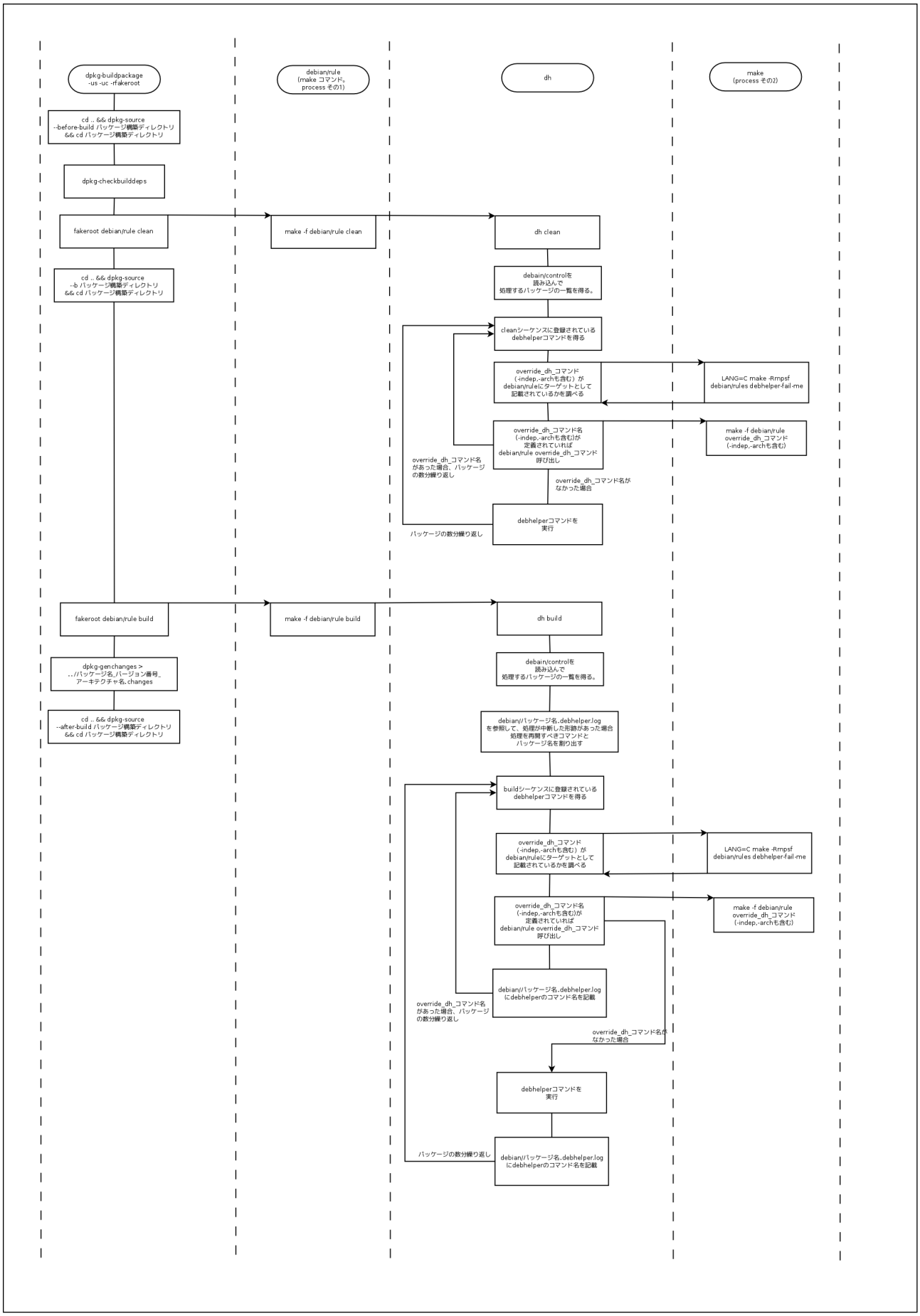
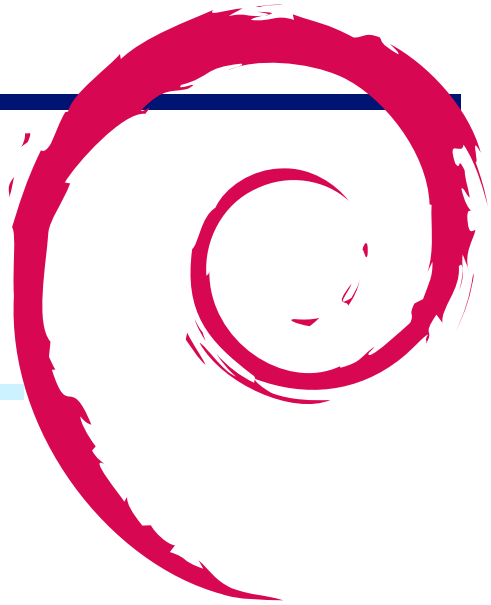

dpkg-buildpackage
-us-uc -rfakeroot
debian/rule
(make コマンド。
process その1)
dh
make
(process その2)
cd .. && dpkg-source
--before-build パッケージ構築ディレクトリ
&& cd パッケージ構築ディレクトリ
dpkg-checkbuilddeps
fakeroot debian/rule clean
make -f debian/rule clean
dh clean
cd .. && dpkg-source
-b パッケージ構築ディレクトリ
&& cd パッケージ構築ディレクトリ
debain/controlを
読み込んで
処理するパッケージの一覧を得る。
cleanシーケンスに登録されている
debhelperコマンドを得る
override_dh_コマンド
(-indep,-archも含む) が
debian/ruleにターゲットとして
記載されているかを調べる
LANG=C make -Rrnpsf
debian/rules debhelper-fail-me
override_dh_コマンド名
(-indep,-archも含む)が
定義されていれば
debian/rule override_dh_コマンド
呼び出し
make -f debian/rule
override_dh_コマンド
(-indep,-archも含む)
override_dh_コマンド名
があった場合、パッケージ
の数分繰り返し
override_dh_コマンド名が
なかった場合
debhelperコマンドを
実行
パッケージの数分繰り返し
fakeroot debian/rule build
make -f debian/rule build
dh build
dpkg-genchanges >
../パッケージ名_バージョン番号_
アーキテクチャ名.changes
debain/controlを
読み込んで
処理するパッケージの一覧を得る。
cd .. && dpkg-source
--after-build パッケージ構築ディレクトリ
&& cd パッケージ構築ディレクトリ
debian/パッケージ名.debhelper.log
を参照して、処理が中断した形跡があった場合
処理を再開すべきコマンドと
パッケージ名を割り出す
buildシーケンスに登録されている
debhelperコマンドを得る
override_dh_コマンド
(-indep,-archも含む) が
debian/ruleにターゲットとして
記載されているかを調べる
LANG=C make -Rrnpsf
debian/rules debhelper-fail-me
override_dh_コマンド名
(-indep,-archも含む)が
定義されていれば
debian/rule override_dh_コマンド
呼び出し
make -f debian/rule
override_dh_コマンド
(-indep,-archも含む)
debian/パッケージ名.debhelper.log
にdebhelperのコマンド名を記載
override_dh_コマンド名
があった場合、パッケージ
の数分繰り返し
override_dh_コマンド名が
なかった場合
debhelperコマンドを
実行
パッケージの数分繰り返し
debian/パッケージ名.debhelper.log
にdebhelperのコマンド名を記載

図 20　dh 内部動作



22 月刊debhelper 第3回

山田 泰資

22.1 はじめに

パッケージビルド手順を記述する debian/rules ファイル。これを簡潔化するために debhelper コマンド群 (dh_*) がありますが、その一方で裏側で一体何がなされているのか掴み難くなってしまいました。

本企画ではこのコマンド群を、毎月持ち回りで解説します。毎月 2 つ以上のコマンドを解説し、次回発表の立候補が無い場合は発表者が次の発表者を指名できるというルールで進めて行きます。

22.2 今月のコマンド: dh&dh_auto_* - シーケンスとビルドシステム

dh コマンドの全体像については前回の野島さんの発表で既に解説されているのですが、そこで「試しに読んでみるとよいと思います」とあったので、実際に読みつついじってみました。その中でもう少し判ったことがあったので報告します。

22.2.1 dh の動作、再まとめ

dh コマンドを実行すると、デフォルトでは以下のコマンド群が各シーケンス毎に呼ばれます:

シーケンス名	シーケンスで実行されるコマンド
clean	dh_testdir dh_auto_clean dh_clean
build	dh_testdir + (rules build-arch build-indep)
build-indep	dh_testdir dh_auto_configure dh_auto_build dh_auto_test
build-arch	dh_testdir dh_auto_configure dh_auto_build dh_auto_test
install	(rules build install-arch install-indep) + dh_testroot dh_prep dh_installdirs dh_auto_install dh_install dh_install* dh_bugfiles dh_ucf dh_lintian dh_gconf dh_icons dh_perl dh_usrlocal dh_link dh_compress dh_fixperms
install-indep	(rules install-indep) + <上の install と同じ>
install-arch	(rules install-arch) + <上の install と同じ>
binary	(rules install binary-arch binary-indep)
binary-indep	(rules install-indep) + dh_installdeb dh_gencontrol dh_md5sums dh_builddeb
binary-arch	(rules install-arch) + dh_strip dh_makeshlibs dh_shlibdeps + <上の binary-indep と同じ>

表 15 dh の各シーケンスで実行されるコマンド

かつては rules(Makefile) に羅列されていたコマンド群が、今は dh の中に Perl のリスト変数で管理されている形になります。そして debhelper モジュール(*.pm) がロード時にリストの内容をいじって実行内容を変更することでビルド過

程をカスタマイズします。

わざわざ make を使わず自前なのは、実行内容を変更するというモジュール機構と override 機構が make ベースでは依存関係ツリーの変更になり実現困難だったからでしょうか。たしかに拡張 make でも使わない限り難しそうです（拡張 make まで行きたくないから現状の実現方法・・・なんでしょうか）。

さて、今回の話題は、このモジュール機構になります。普段何気なくどんなパッケージでもパッケージビルドされている訳ですが、言語環境や開発者の選択によってビルド方法は千差万別です。これはどうやって吸収されているのでしょうか?

22.2.2 2つのモジュール：シーケンスとビルドシステム

ここで登場するのが「シーケンス」とは別の、「ビルドシステム」モジュールになります。この2つは

- シーケンスモジュールは（主に）前処理・後処理を追加し、適切なパッケージビルドが行われるようにする
- ビルドシステムモジュールは各ソースパッケージに内包されるビルド方式を自動認識し、それを駆動する

と異なる役割を持ちます。典型的な configure&make パターンで説明すると、

1. Sequence/autotools_dev.pm が config.sub/config.guess を最新に更新
2. Buildsystem/autoconf.pm が ./configure を発見・実行
3. Buildsystem/makefile.pm が Makefile を認識し、 make でビルドやインストール

といった連携リレーになります（autotools_dev は–with で有効化された場合のみ）。

22.2.3 ビルドシステムの選択と連携

さて、ビルドシステムは以下のフローで自動選択されています:

1. まず、 Buildsystem/*.pm は全部ロードする
2. 各モジュールの check_auto_buildable API にて「ビルドできる度」を照会
3. 同じクラス階層の中で一番大きい「ビルドできる度」を返したものを選択

ポイントは

- この自動選択は configure/build/test/install/clean の各段で毎回行われる
- 同じクラス階層縛りがある

の2点です。つまり、

- 毎回行われるので、各段で応答・不応答を変えてモジュール間連携を行う
- 自分が優先されるべき場合、親クラスの結果を上回るようにして勝つ

とする必要があり、「単純にビルドできるから真値を返す」という実装ではないのでした。これは自前のビルドシステム拡張、特に他と連携する場合に必要な留意事項になります。具体的にどういうコードなのかというと cmake のものが参考になります:

```
=== Buildsystem/cmake.pm ===
sub check_auto_buildable {
    my $this=shift;
    my ($step)=@_;
    if (-e $this->get_sourcepath(‘‘CMakeLists.txt’’)) {
        my $ret = ($step eq ‘‘configure’’ && 1) ||
                  $this->SUPER::check_auto_buildable(@_);
        # Existence of CMakeCache.txt indicates cmake has already
        # been used by a prior build step, so should be used
        # instead of the parent makefile class.
        $ret++ if ($ret && -e $this->get_buildpath(‘‘CMakeCache.txt’’));
        return $ret;
    }
    return 0;
}
```

Makefile ジェネレータとして configure ステージだけ担当のような顔をしつつ、ビルドキャッシュがある場合は自分が後段も担当すべきとして親である makefile.pm を押しのけて勝つ、というわけです。

22.2.4 ビルドシステムは誰が呼んでいるのか

ところでこのビルドシステム、誰が呼んでいるのでしょうか? 例えば dh では

```
$ dh --buildsystem=perl_makemaker
```

のように指定できるのですが、 dh にはどこにもビルドシステムに関する処理は書かれていません。

これは dh からオプションをそのままスルーパスされる形で[*58]dh_auto_(build|clean|configure|install|test) の dh_auto_*系コマンド(だけ) が呼び出し元になっています。シーケンス中のすべての dh_*コマンドに同様にスルーパスは届くのですが、反応するのがこの 5 つだけ、という訳です(man debhelper の BUILD SYSTEM OPTIONS)。だからこそ独自処理を書く場合は rules に

```
override_dh_auto_build:
        ...
```

などと override_dh_auto_*ターゲットを書くという話になるわけです。 dh_auto_*さえ止めれば、いかなるシーケンスが走ってもビルドシステム呼び出しが行われず、実際のビルドは行われないからです。

これらの dh_auto_*コマンドは上のロード→照会 (check_auto_buildable)
→ API(configure|build|test|install|clean) コールのトリガを引いているだけです。このフローの詳細はライブラリ化されているので各コマンドは 3 行くらいしかありません。

22.2.5 ビルドシステムの追加方法

ビルドシステムの拡張は簡単で、以下の API を実装した*.pm を Buildsystem/ フォルダに置くだけです。基底クラスに空実装があるので全部書く必要はなく、実際に処理を追加したい API のみ実装すれば十分です。

```
check_auto_buildable($step)                          # 必須
pre_building_step($step)                             # オプション
configure() build() test() install($destdir) clean() # いずれかを実装
post_building_step($step)                            # オプション
```

各 API の処理は名称から想像される通りで、先に解説済みの c_a_b API 以外は返値もありません(使われていません)。

サンプルとして、今は懐かしき imake/xmkmf を使ったソースパッケージの自動検知＋ビルドに対応するように imake.pm モジュールを用意してみました:

```
package Debian::Debhelper::Buildsystem::imake;
use strict;
use base 'Debian::Debhelper::Buildsystem::makefile';

sub DESCRIPTION { "imake (IMakefile)" }
sub new { shift->SUPER::new(@_); }
sub check_auto_buildable {
    my($self, $step) = @_;
    return 1 if ($step eq "configure" &&
                 glob($self->get_sourcepath("I[Mm]akefile")));
    return 0;
}
sub configure { shift->doit_in_sourcedir("xmkmf", "-a", @_); }
1;
```

こんな簡単なものでも、 kterm などの対象パッケージをビルドするには十分です。

なお、これを組み込むには Dh_Buildsystems.pm のソース中の自動判定リストの末尾に

```
our @BUILDSYSTEMS = (''autoconf'', ..., ''imake'');
```

のようにモジュール名を追加してやる必要があります。

[*58] Debian は、ビルドコマンド調査の時も思いましたがコマンドライン引数の引き回し本当に多用しますね・・・

22.2.6 まとめ

本解説では dh フレームワークを支えるビルドシステム部分を解説しました。これは dh_*コマンドとしては dh_auto_* の 5 コマンドに対応し、これらを通してビルドシステムが駆動されています。

22.3 今月のコマンド: dh_builddeb

dh_builddeb は、 dh によって起動される一連のコマンドシーケンスの最後を飾るコマンドです(ちなみに最初は dh_testdir)。

マニュアルは「dpkg-deb を呼ぶだけのかんたんなおしごと[*59]」と一行だけの解説ですが、これが意外にも中で色々としていて dh_*コマンドの勉強になります。

22.3.1 何をしているの?

やっていること自体は以下の 3 つです:

1. debhelper(7) のファイル排除指定があれば、その除去処理をする
2. deb/udeb 形式の判定を行い、 dpkg-deb の起動分けをする
3. さらに、 DEB_BUILD_OPTIONS の parallel=指定があれば、 dpkg-deb を並列駆動する

マニュアルの解説が 1 行の割には、意外に仕事をしています。

22.3.2 疑問: udeb って何?

実は udeb の存在を知りませんでしたが、 udeb というのは Debian Installer(d-i) で使用される*.deb 風のパッケージです。形式としては udeb も deb も同じで普通に dpkg で操作できるのですが、極小リソースでの限定的な利用を想定しているためドキュメントはおろかチェックサム機能などまで外されています。

```
=== debian/control ===
Section: debian-installer
...
XC-Package-Type: udeb
XB-Installer-Menu-Item: 1200 <- d-i menu での表示制御パラメータ
```

のように特殊なヘッダが入っている control がある場合、 dh_builddeb は自動的に udeb ビルドモードで dpkg-deb を起動します。

他にもこういう特殊ヘッダはあるのだろうかとか、これを入れると具体的に何をどう変えられるのかなど udeb と d-i の話は更に掘ると面白そうですが、今回は脱線ということでここまでにしておきます[*60]。 udeb 固有処理は他の dh_*コマンドにも多数含まれており、 is_udeb() で様々な処理分けを裏側でしています。

22.3.3 debhelper(7) 系コマンドの実装パターン

dh_builddeb の中を覗くと、各所で$dh... という変数へのアクセスが頻出しています。これは dh_*コマンドの debhelper(7) オプションのパースや共通的な処理が Debian::Debhelper::Dh_Lib ライブラリで行われており、このライブラリとコマンド側の連携に %dh というグローバル変数が使われているためです。また %ENV も多用されています。

コードの流れとしては、 debhelper の純正 (Perl 製)dh_*コマンドは概ね以下の実装パターンになっています:

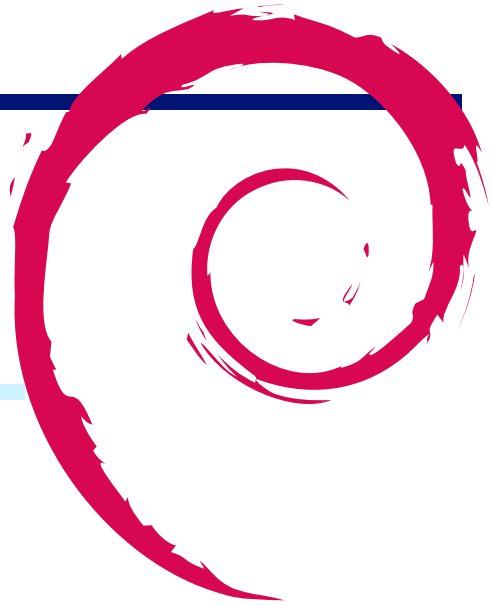
[*59] dh_builddeb simply calls dpkg-deb(1) to build a Debian package or packages.
[*60] 資料としては http://d-i.alioth.debian.org/doc/talks/debconf6/paper/ かな?

```
use Debian::Debhelper::Dh_Lib; # init() 関数などがインポートされる
init(options => { ‘‘myopt=s’’ => \&dh{MYOPT}, ... }); # @ARGV や %ENV の定型処理

# 含まれているパッケージの数だけ処理を反復
foreach my $package (@{$dh{DOPACKAGES}}) {
    # 上の init() で取り込まれた結果を見ながら処理をする
    if ( $dh{...}) { ... # Dh_Lib.pm の API を呼んだりするなど ... }
    # 自動的に取り込まれない環境変数( コマンド固有）は自分で処理する
    if ($ENV{...}) { ... # 上記同様 ... }
}
```

dh_builddeb の場合は、上のループの中が不要ファイルの削除と dpkg-deb の起動になり、これが例えば dh_strip の場合は各生成中パッケージのワーキングフォルダをスキャンして、しかるべきファイルを strip して回るというようになります。

22.3.4 まとめ

dh_builddeb コマンドの解説と、そこから出てきた疑問と共通的な構成の解説を行ってみました。 dh_*コマンドは実質 1 行しかないものから 1000 行に迫るものまで色々ありますが、 dh_builddeb はちょうど理解する上で手頃な大きさです。



23 月刊debhelper 第4回

山本 浩之

23.1 パッケージの make の前に…

先月までに学んできたように、 debhelper は、基本的にビルドに必要な一連の dh_XXX コマンドを自動実行します。

例えば debhelper は、 dh_auto_configure というコマンドを提供しており、これはご想像の通り、

```
./configure --build='dpkg_architecture_value("DEB_BUILD_GNU_TYPE")' --prefix=/usr --includedir=/usr/include \
--mandir=/usr/share/man --infodir=/usr/share/info --sysconfdir=/etc --localstatedir=/var \
--libdir=/usr/lib/\$multiarch --libexecdir=/usr/lib/\$multiarch --disable-maintainer-mode \
--disable-dependency-tracking --host='dpkg_architecture_value("DEB_HOST_GNU_TYPE")
```

をしているだけです。

しかし、メンテナによって autotools を利用し、 confugure スクリプトをビルドの度に毎回 configure.ac から生成したい人もいるでしょうし、もしかすると Makefile の元となる Makefile.in だって、毎回 Makefile.am から生成したい人もいるでしょう。また、 Debian パッケージオリジナルのパッチをあててパッケージを作るのは、ごく当たり前のように行なわれています。

そこで今月は、一連の dh_XXXX コマンドへの追加の仕組みと、その例として、 dpatch パッケージで提供される dh_dpatch_patch コマンドと、 autotools-dev パッケージで提供される dh_autotools-dev_updateconfig コマンドの追加について解説しましょう。

23.2 一連の dh_XXXX コマンドへの追加の仕組み

基本となる一連の dh_XXXX コマンドは、大元のコマンドである dh スクリプトに記述してあり、すべてについては先月や先々月に話されているので、割愛します。

特に make 直前に実行されるものは、

```
dh_testdir  #カレントディレクトリの確認
dh_auto_configure  #./configure の実行
```

だけです。

勿論、これだけではパッチもあてられないですし、 configure スクリプトの再生成もできません。

そこで dh スクリプトは、一連の dh_XXX コマンドを列挙する配列にし、これを$sequences$sequence のスカラ変数として保持しています。 (この辺は perl にあまり詳しくないので、ちょっと間違っているかも…)

また、この$sequences から dh_XXXX コマンドを除いたり (remove_command)、ある dh_XXXX コマンドの前に追加する (insert_before) サブルーチンも用意されています。

dh スクリプトは、/usr/share/perl5/Debian/Debhelper/Sequence/ディレクトリのなかにあるファイルを参照しており、ここに

```
insert_before("dh_auto_configure", "dh_new_command")
```

という記述のあるファイル (アドオン) が追加されると、 dh_auto_configure の前に dh_new_command が実行されるようになります。

ただし、 rules の

```
dh $@ --with 〜
```

でアドオンが指定されない限り評価はされないので、ビルドに不必要な dh_XXXX コマンドを入れていても大丈夫なはずです。

23.3 dh_dpatch_patch コマンド

dpatch パッケージをインストールすると、/usr/share/perl5/Debian/Debhelper/Sequence/ディレクトリに dpatch.pm ファイルが入り、これには、

```
insert_before("dh_auto_configure", "dh_dpatch_patch")
insert_before("dh_clean", "dh_dpatch_unpatch")
```

という記述があります。これは見て想像できるとおり、 make の直前に実行される./configure のさらに直前に、dh_dpatch_patch を加えています。また、下の記述は、以前にビルドしたことがある場合に、パッチする前の状態にする dh_dpatch_unpatch コマンドも追加されています。

この dh_dpatch_patch は、パッケージソースの debian/patches/00list に記述されたファイル名のパッチを先頭からパッチするコマンド、 dpatch スクリプトを実行します。

すなわち、もし configure スクリプトに dpatch でなんらかのパッチをあてて実行したければ、 dpatch パッケージを Build-dep し、 debian/patches/ディレクトリにパッチファイルとそれに合わせた 00list ファイルを用意し、

```
dh $@ --with dpatch
```

と rules ファイルに記述しておけば良いはずです。

make だけしたいならば、ターミナルで、

```
$ dh_dpatch_patch
$ ./configure 〜
$ make
```

でも大丈夫です。

23.4 autotools だって使いたい

ビルドするマシンの環境に合わせて、 configure スクリプトなどを調整してくれるツールとして、 GNU autotools というものがあります。次にこの GNU autotools を debhelper で利用する方法について述べましょう。

dh-autoreconf パッケージをインストールすると、依存関係で automake、 autoconf と、 automake に依存して autotools-dev がインストールされます。 autotools-dev パッケージは /usr/share/perl5/Debian/Debhelper/Sequence/ディレクトリに autotools-dev.pm ファイルが入ります。これには、

```
insert_before("dh_auto_configure", "dh_autotool-dev_updateconfig")
insert_before("dh_clean", "dh_autotool-dev_restoreconfig")
```

と記述されています。

dh_autotool-dev_updateconfig コマンドは、カレントディレクトリ以下で実行しているシステムタイプの標準名を推測するための config.guess と config.sub ファイルを探し、 config.guess.dh-orig と config.sub.dh-orig ファイルに名前を換え、それぞれ/usr/share/misc/ディレクトリにある最新 autotool-dev の config.guess と config.sub をコピーしてき

ます。

dh-autoreconf パッケージは/usr/share/perl5/Debian/Debhelper/Sequence/ディレクトリに autoreconf.pm ファイルが入ります。これには、

```
insert_before("dh_auto_configure", "dh_autoreconf")
insert_before("dh_clean", "dh_autoreconf_clean")
```

と記述されています。

dh_autoreconf コマンドは、簡単に言うと、 automake、 autoconf をしてくれる autoreconf スクリプトを呼び出し、configure や Makefile.in を再生成します。

dh_autoreconf を使いたいときは、このパッケージに Build-dep し、

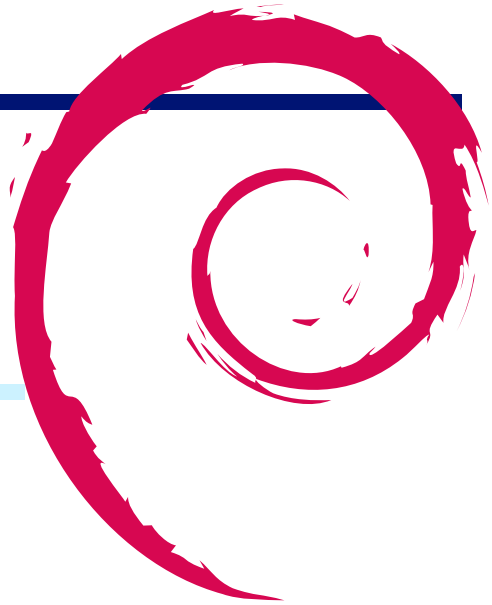
```
dh $@ --with autoreconf
```

と rules ファイルに記述しておけば良いはずです。 debian/autoreconf ファイルにディレクトリのリストがあれば、そこだけ configure や Makefile.in を更新してくれます。

autoreconf.pm ファイルでは「 dh_auto_configure より前だよ」という指定しかありませんから、 dh_autoreconf が実行されてパッチがあてられるのか、パッチがあてられてから dh_autoreconf が実行されるのかまでは記述されていません。 dh スクリプトを見た限りでは「 –with」オプションの順でリストされているようです。つまり、 dh-autoreconf パッケージを利用するソースパッケージの Makefile に対して BTS でパッチを書く場合は、「 –with」オプションの順を確認する必要がありそうです。

23.5 おわりに

今回は make を実行する前に使用される configure スクリプトなどの、 debhelper を使ったカスタマイズ法について、駆け足で説明してみました。



24 月刊debhelper 第5回

杉本 典充

24.1 今月のコマンド: dh_md5sums

dh_md5sums コマンドは「 DEBIAN/md5sums ファイルを生成する」コマンドです。

24.1.1 DEBIAN/md5sums ファイルについて

「 $ ar x debian-package.deb」を実行し現れる control.tar.xx ファイルを展開すると md5sums ファイルが出てきます。この md5sums ファイルは data.tar.xx ファイルに含むファイルそれぞれから取得した md5sum を記述しています。

```
$ apt-get download hello-debhelper
$ ar x hello-debhelper_2.7-3_i386.deb
$ ls
control.tar.gz  data.tar.gz  debian-binary  hello-debhelper_2.7-3_i386.deb
$ tar xf control.tar.gz
$ ls
control         data.tar.gz    hello-debhelper_2.7-3_i386.deb
control.tar.gz  debian-binary  md5sums
$ head -n 1 md5sums
098518cc321f0467dc0e7c67f65e2cc1  usr/bin/hello
```

24.1.2 パッケージのビルド処理における dh_md5sums の実行

dh_md5sums コマンドはインストールするファイルの md5sum を取得する処理のため、ビルド処理の終盤で実行されます。

```
$ apt-get source hello-debhelper
$ cd hello-debhelper-2.7
$ debuild -uc -us
  (省略)
  dh\_gencontrol -a
  dh\_md5sums -a
  dh\_builddeb -a
  (省略)
$ ls debian/hello-debhelper
DEBIAN   usr
$ head -n 1 debian/hello-debhelper/DEBIAN/md5sums
098518cc321f0467dc0e7c67f65e2cc1  usr/bin/hello
```

24.1.3 コマンドのオプション

表 16　dh_md5sums のコマンドラインオプション一覧

オプション	説明
-x, –include-conffiles	DEBIAN/conffiles ファイルに記述した設定ファイルの md5 も生成します。
-Xitem, –exclude=item	md5sum の生成を除外するファイル名を指定します。ただしディレクトリが別でもファイル名が一致すればどちらも除外されます。

24.2 今月のコマンド: dh_strip

dh_strip コマンドは「実行ファイル、共有ライブラリ、スタティックライブラリを strip する」コマンドです。

24.2.1 デバッグシンボルの扱い方を制御する

オプションなしや環境変数を指定せずに dh_strip コマンドを実行するとコンパイルしたオブジェクトファイルのデバッグシンボルを strip するのが通常の処理です。しかし、デバッグを目的とする場合はデバッグシンボルを strip してしまうとデバッガが十分に機能しないため困ります。

dh_strip コマンドではデバッグシンボルを以下のように扱うことができます。[1]

- オプションなしで実行すると、strip する。
- 環境変数 DEB_BUILD_OPTIONS=nostrip を指定して実行すると、strip しない。(処理的には dh_strip が即座に終了する)
- –dbg-package オプションを指定すると、/usr/lib/debug 配下にデバッグシンボルを分離して残すパッケージ(=デバッグパッケージ)を作成する。

24.2.2 デバッグパッケージを作成するための条件

debhelper の機能を利用すると strip 済みのバイナリパッケージに加えて簡単にデバッグパッケージを作成できます。デバッグパッケージを追加で作成したい場合は以下の処理を記述してパッケージをビルドすればよいです。

- CFLAGS などのコンパイルオプションに"-g' を付与しビルド時にデバッグシンボルを生成するようにする。
- debian/rules で override_dh_strip を定義し、dh_strip –dbg-package=package-dbg を処理させる。
- debian/control にパッケージ「package-dbg」の定義を記述する。このとき、パッケージ「package-dbg」はパッケージ「package」にバージョン指定をして depend すること。

24.2.3 コマンドのオプション

表 17 dh_strip のコマンドラインオプション一覧

オプション	説明
-Xitem, –exclude=item	指定した文字列を含むファイルを strip 処理の対象から除外する。複数のファイルを指定したい場合はオプションを複数回指定することも可能。
–dbg-package=package	デバッグシンボルを含むパッケージ「package-dbg」を作成する。
-k, –keep-debug	パッケージをビルドした作業ディレクトリ内の usr/lib/debug に strip 後のデバッグシンボルファイルを残す。–dbg-package オプションの指定で事足りる場合は多いが、より細かくデバッグシンボルを扱いたい場合を想定して用意されている。

参考文献

[1] Debian Wiki - DebugPackage http://wiki.debian.org/DebugPackage

[2] Debian.org 第 6 章パッケージ化のベストプラクティス http://www.debian.org/doc/manuals/developers-reference/best-pkging-practices.html

25 Debian 開発者の KDE 環境あれこれ

野島 貴英

最近の Debian をそのままインストールすると、特に指定しない場合 GNOME というデスクトップ環境がインストールされます。しかしながら、 Debian ではいくつもデスクトップ環境が用意されており、ユーザは自由にこれらを選んで使うことができます。デスクトップ環境はユーザにとってはいつも使う環境ですから、いろいろとこだわりもあるかとおもいます。今回はそんな中、 KDE というデスクトップ環境についてあれこれ語ってみます。

25.1 利用者としての KDE 導入方法

Debian の安定版の利用を検討していて、 KDE 環境をいわゆる利用者として使う為にインストールするやり方について簡単に述べます。

1. 安定版の Debian のインストール DVD を用意します。
2. インストーラのメニュー画面が出ましたら、 TAB キーをおすと画面下の方に編集可能な行が現れますので、以下の例ように “desktop=kde” という文言を追加します。なお、日本語 106 キーボードを使っている場合、キートップの刻印の通りに “=” を押しても “=” 文字が入力できない場合がありますが、この場合は “^” の刻印のキーを押すと “=” 文字が入力できます。

```
/install.amd/vmlinuz vga=788 initrd=/install.amd/initrd.gz --- quiet desktop=kde
```

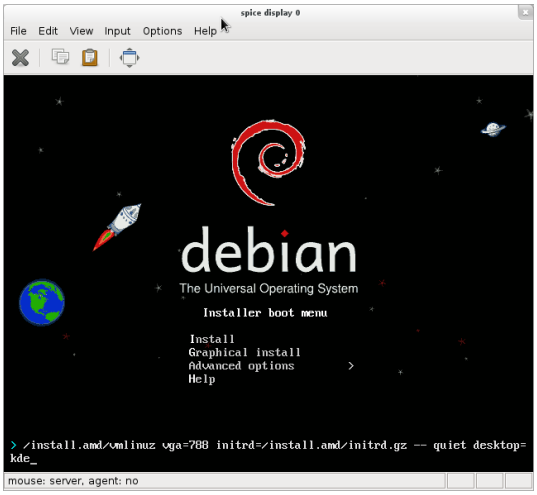


図 21 安定版インストール画面で TAB キーを押したときの様子

3. あとは通常どおりインストールを行います。インストールを進めていくと「 インストールするソフトウェアの選択:」のメニューが現れますので、 “Debian desktop environment” を選択しておいてください。

4. インストールが完了しましたら、リブートを行います。
5. KDE 環境が起動します。

以上となります。簡単ですね。

25.2 開発者としての experimental 版 KDE 導入方法 (KVM+spice)

東京エリア Debian 勉強会にいらっしゃるような方々には、前述のインストールと環境ではきっと「ぬるゲー(笑) 」な感じのはずです。その場合、是非とも experimental 版の KDE 環境を利用いただき、 BTS 書き/パッチ開発/翻訳/デバッグなどの開発活動に勤しんでみましょう。ここでは、開発者向け KDE 環境導入について簡単に述べます。

- 開発者向けに experimental 版導入を前提にします。
- 仮想環境である KVM を利用して仮想環境上に導入します。これなら、ディスクイメージファイルをとっておけば、うっかり experimental 環境で aptitude full-upgrade して全く立ち上がらなくなっても(実話) あっさり復帰できます。
- サウンドももちろん欲しいので仮想デスクトップ環境として spice を使います。
- いつでもどこでも開発できるようにモバイル環境に構築します。

図 22 の KDE 開発環境の用意を想定します。

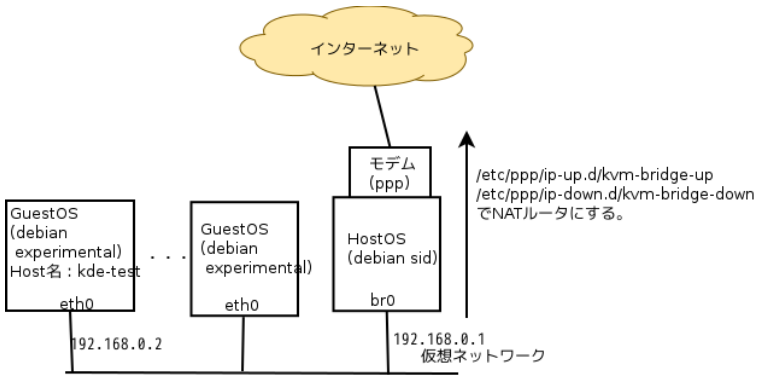


図 22　KDE 開発環境

以下は導入に関しての流れです。(細かい事は割愛します。操作にあたっては適宜 root 権限が必要だったりします)

1. HostOS となる PC の BIOS を操作して、 CPU の仮想技術支援機構のスイッチを ON にしてブートしておきます。
2. HostOS に `http://www.debian.org/CD/netinst` から名刺サイズの CD イメージを落として置きます。
3. HostOS の/etc/network/interfaces に以下の追記を行い、 br0 を作っておきます。

```
# 追記はここから。 aptitude install bridge-utils はやっておくこと。
auto br0
iface br0 inet static
        address 192.168.0.1
        netmask 255.255.255.0
        bridge_ports none
        bridge_stp off
        bridge_fd 0
        bridge_maxwait 0
```

4. HostOS の/etc/sysctl.d/bridge-filter-workaround.conf を作り、
   sysctl -p /etc/sysctl.d/bridge-filter-workaround.conf を実行して、 br0 のフィルタを無効化しておきます。

```
# /etc/sysctl.d/bridge-filter-workaround.conf の中身
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
```

5. HostOS の/etc/ppp/ip-up.d/kvm-bridge-up,/etc/ppp/ip-down.d/kvm-bridge-down を作っておきます。他にフィルタとか必要であれば適当にどうぞ。

```
#!/bin/sh
# /etc/ppp/ip-up.d/kvm-bridge-up の中身
PATH=/bin:/usr/bin:/sbin:/usr/sbin
CDPATH=
sysctl -w net.ipv4.ip_forward=1
iptables -t nat -A POSTROUTING -o $PPP_IFACE -j MASQUERADE
iptables -A FORWARD -i br0 -o $PPP_IFACE -j ACCEPT
```

```
#!/bin/sh
# /etc/ppp/ip-down.d/kvm-bridge-down の中身
#!/bin/sh
PATH=/bin:/usr/bin:/sbin:/usr/sbin
CDPATH=
sysctl -w net.ipv4.ip_forward=0
iptables -t nat -D POSTROUTING -o $PPP_IFACE -j MASQUERADE
iptables -D FORWARD -i br0 -o $PPP_IFACE -j ACCEPT
```

6. HostOS に kvm/libvirt/spice-client-gtk パッケージを導入しておきます。
7. HostOS にて GuestOS 用の kde-test.xml を以下の雛形で作成して virsh define kde-test.xml しておきます。[*61]

```
<domain type='kvm'>
  <name>kde-test</name>
  <memory>1048576</memory>
  <vcpu>1</vcpu>
  <os>
    <type arch='x86_64' machine='pc-1.0'>hvm</type>
    <boot dev='hd'/>
    <boot dev='cdrom'/>
    <bootmenu enable='yes'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='writeback'/>
      <source file='/var/lib/libvirt/images/kde-test.img'/>
      <target dev='vda' bus='virtio'/>
    </disk>
    <disk type='file' device='cdrom'>
      <driver name='qemu' type='raw'/>
<!-- directory of cdimage は適当に変更ください -->
      <source file='/directory of cdimage/debian-6.0.4-amd64-businesscard.iso'/>
      <target dev='hdc' bus='ide'/>
      <readonly/>
    </disk>
    <controller type='ide' index='0'/>
    <interface type='bridge'>
<!-- mac アドレスは適当に変更ください -->
      <mac address='52:54:00:31:cd:5a'/>
      <source bridge='br0'/>
      <model type='virtio'/>
    </interface>
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <graphics type='spice' port='5900' autoport='no'>
      <clipboard copypaste='yes'/>
    </graphics>
    <sound model='ac97'\>
    <video>
      <model type='qxl' vram='9216' heads='1'/>
    </video>
    <memballoon model='virtio'>
    </memballoon>
  </devices>
</domain>
```

8. HostOS で仮想環境用のディスクを 10GB ぐらいで作っておきます。

[*61] virt-install は何故か自分の experimental な環境では Segmentation Fault で落ちてしまうのでここでは使いません。BTS しときます。

```
qemu-img create -f raw /var/lib/libvirt/images/kde-test.img 10G
```

9. HostOS で KVM を起動して、 spice クライアントを接続します。

```
virsh start kde-test; spicy -h 127.0.0.1 -p 5900 &
```

10. GuestOS の Debian インストーラが起動したら、 TAB キーを押し、画面下に現れた編集可能な行に、”priority=medium” を以下のように入力してインストールを開始します。

```
 /install.amd/vmlinuz vga=788 initrd=/install.amd/initrd.gz --- quiet priority=medium
```

インストール途中「 Debian アーカイブのミラーを選択」のメニューにて”sid” を選択し、「 インストールするコンポーネント」として「 ssh サーバー」のみ (他は選択しない) とします。

11. インストールが完了すると、テキストコンソールから Debian sid な GuestOS へログインできるようになります。
12. GuestOS にログインして以下の行を/etc/apt/source.list へ付け加えます

```
#追加内容
deb http://ftp.jp.debian.org/debian/ experimental main
deb-src http://ftp.jp.debian.org/debian/ experimental main
```

13. GuestOS の/etc/apt/preference.d に Debian KDE チーム製の experimental パッケージ用 preference ファイルをインストールします。

```
cd /etc/apt/preference.d && wget http://pkg-kde.alioth.debian.org/files/kde-experimental
```

14. GuestOS で以下を実行し、 experimental な KDE 環境を一気に入れてしまいます。

```
aptitude update;aptitude aptitude install task-kde-desktop task-japanese-kde-desktop;aptitude clean
```

15. インストールが終わったら、 GuestOS をリブートします。 GuestOS で KDE の experimental 版が起動し、グラフィカルなログイン画面が現れます。

25.3 Debian と KDE 環境のバージョン

Debian のバージョンと KDE のバージョンの対応を表 18 に載せます。

Debian	stable	testing	unstable	experimental	upstream
KDE	4.4	4.6	4.6	4.7.4	4.8.0

表 18　Debian のバージョンと KDE のバージョン

KDE の upstream は 2012 年 1 月 25 日に 4.8.0 をリリースしたばかりなので、まだ experimental も追いついていない状態です。

25.4 KDE 環境の開発の特徴

KDE 環境の開発は以下のような特徴があります。

1. Qt（ キュート）ライブラリを使う。
2. C++ のコードが基本
3. autotools の代わりに cmake が使われる

このため、 Debian ではパッケージ開発の為に pkg-kde-tools パッケージが用意されています。

25.5 Debian での KDE 環境のパッケージ開発

Debian では KDE 環境のパッケージ開発用に pkg-kde-tools というパッケージを別に用意しています。こちらを導入すると KDE 環境のパッケージ構築の際に便利な機能が使えるようになります。

項番	拡張されるもの	拡張	備考
1	dh	–with kde	debhelper に kde 用の拡張を指定
2	dh_auto_*	–buildsystem=kde	dh_auto_*が cmake を使うようになる、 KDE 環境用の設定を行う等
3	CDBS	kde.mk	CDBS で KDE 用の拡張が利用できるようになる
4	その他	variables.mk など	debian/rules の中で$(DEB_CMAKE_KDE4_FLAGS) などが使える等

表 19　pkg-kde-tools をインストールした時の拡張

25.6 超簡易的に KDE 用プログラムの Debian パッケージを作ってみる

ここでは超簡易的に KDE 用プログラムの Debian パッケージを作ってみます。まず、事前準備として、

- 環境は 25.2 章の experimental 環境を用意ください。
- 必要なパッケージ (cmake パッケージ等) [*62]

次に khello-1.0.0/なるディレクトリに `http://techbase.kde.org/Development/Tutorials/First_program` にある、 main.cpp と CMakeLists.txt を配置します。

```
$ cd khello-1.0.0
$ ls
CMakeLists.txt main.cpp
$
```

次に、オリジナルの tar.gz アーカイブを作成しておきます。

```
$ cd ..
$ tar czf khello_1.0.0.orig.tar.gz khello-1.0.0
$ ls -F
khello-1.0.0/  khello_1.0.0.orig.tar.gz
$
```

dh_make を使って debian/ディレクトリを仕込みます。あとは rules ファイル以外いつも通り、パッケージを作成するようにファイルを作成しておきます。

```
$ cd khello-1.0.0/debian
$ ls -F
README.Debian  changelog  control    docs   source/
README.source  compat     copyright  rules
$
```

pkg-kde-tools パッケージを利用する rules ファイルを記載します。

```
# pkg-kde-tools を使った KDE 開発用 debian/rules ファイルの中身。

%:
	dh $@ --with kde
```

あとは、 dpkg-buildpackage -uc -us -rfakeroot を実行してビルドします。

[*62] KDE 環境向けの開発が全く初めての人は、細かい事が判ってくるまで、 aptitude build-dep kdeutils しておいて KDE パッケージ開発に必要なパッケージをあらかじめまとめて導入しておくという手もあります

```
$ dpkg-buildpackage -us -uc -rfakeroot
dpkg-buildpackage: source package khello
... 中略...
dpkg-source: info: building khello in khello_1.0.0-1.debian.tar.gz
dpkg-source: info: building khello in khello_1.0.0-1.dsc
 debian/rules build
dh build  --with kde
   dh_testdir
   dh_auto_configure --buildsystem=kde
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
... 中略...
```

無事、–buildsystem=kde が利用され、 cmake が実行されています。

しばらく待つと無事に khello_1.0.0-1_amd64.deb などが出来上がります。

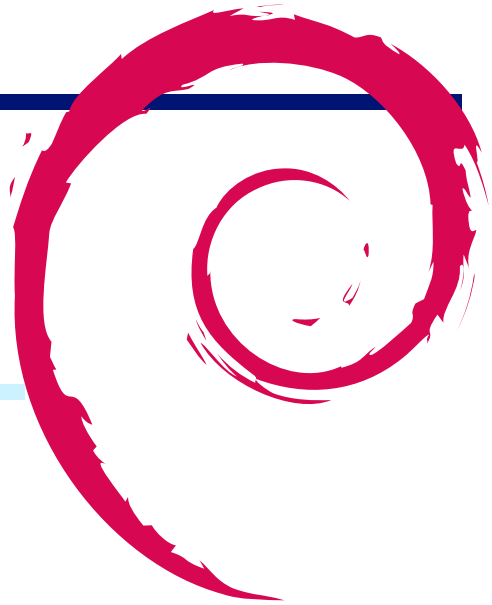
ほら、 pkg-kde-tools のおかげでパッ ケージ開発も簡単でしょ?でしょ?

25.7 おわりに

今回は、 Debian 開発者の為の KDE 環境の構築と、簡単なパッ ケージ作成について、一通り記載してみました。これを機に、 KDE 環境に関する開発をされる方が増えるとうれしいと思っ ています。

25.8 参考文献

- `http://pkg-kde.alioth.debian.org/` Debian KDE Team のホームページ。
- `http://techbase.kde.org` KDE Techbase
- `http://kde.org/` KDE 本家
- `http://www.spice-space.org/` SPICE 仮想デスクトッ プデバイス本家



26 CMake を使ってみる

野島 貴英

26.1 CMake とは

KDE 環境の開発に使われているツールに CMake があります。これは従来の autotools のようなものです。が、autotools に比べて次に述べる代表的な特徴があります。

- バイナリのプログラムである
  autotools は、ご存知の通り、中は sh スクリプトとなっています。これは/bin/sh を基本コマンドとして持つ従来の UNIX 系の OS で使うなら非常に都合がよいのですが、そもそも/bin/sh を持たないシステムの元で利用しようとすると動作できません。これでは、例えば、標準的な C プログラムをコンパイル出来る環境なのに、/bin/sh が無いという本質ではない理由の為に移植性を損なうのはちょっと残念です。
  CMake はバイナリのプログラムなので、コマンド単体で動作することができ、/bin/sh など UNIX のコマンドが無い場所でも問題なく動作できます。
- 様々なプラットフォーム用の構築システムに対応できる
  autotools は make に特化したツールとなります。ここで、そもそも Makefile が一般的ではない開発環境(例: Microsoft Visual Studio 等の様々な IDE) の場合、 Makefile よりも IDE のプロジェクトファイルを生成できた方がより都合がよかったりします。 CMake は一本の CMakeLists.txt を用意するだけで、 Makefile や、 IDE 環境用のプロジェクトファイルを生成できたりする能力があります。この為、 autotools を利用したソースパッケージのように、 Makefile.am と、例えば.vcproj ファイルを別々に修正して UNIX/Windows 間の移植性を保つというような作業から開発者が開放される可能性を意味します。
- その他
  詳しいサマリは、 DDJ ジャーナルの `http://drdobbs.com/cpp/184405251` にサマリされているような機能がある模様です(まだ自分は未評価です。)
  この記事からいくつか抜粋すると、
  - QT ライブラリの moc コマンド/ITK の CABLE/VTK のラッパー生成コマンドに対応したステートメント
  - 静的ライブラリ、動的ライブラリの生成を容易に切り替えれるようにする機能
  - ファイルの依存関係の自動生成、並列ビルドのサポート

  がある模様です。

26.2 使ってみる

百聞は一見にしかずなので、ちょっと使ってみます。
cmake パッケージをシステムに導入します。

```
$ sudo aptitude install cmake
```

次に以下のソース (hello.c,config.h.in) を用意します。

```
/*hello.c*/
#include <stdio.h>
#include "config.h"
int main(int argc,char **argv)
{
        printf("hello world\n");
#if defined(HAVE_EXIT)
        printf("yes, this system has exit()\n");
#endif
        return(0);
}
```

```
/*config.h.in*/
#cmakedefine HAVE_EXIT
```

次に、 CMakeLists.txt を用意します。

```
# cmake のバージョンは 2.8 以上
cmake_minimum_required(VERSION 2.8)
# project の名前を宣言
project(hello)

# CMake 提供のマクロをロードする。ここでは関数がシステムにあるかを確かめるマクロ
# を使ってみる。
include (${CMAKE_ROOT}/Modules/CheckFunctionExists.cmake)

# exit() 関数をチェックしてみる。あれば HAVE_EXIT を定義せよという意味。
check_function_exists(exit HAVE_EXIT)

configure_file (
  "${PROJECT_SOURCE_DIR}/config.h.in"
  "${PROJECT_BINARY_DIR}/config.h"
)
# cc -I に何指定するか
include_directories ("${PROJECT_BINARY_DIR}")

# hello は hello.c から出来るという事を指定
add_executable(hello hello.c)
```

これら 3 つのファイルを hello-src/以下に配置します。

```
$ ls -lR
.:
合計 4
drwxr-xr-x 2 nojima nojima 4096  2 月 17 03:15 hello-src

./hello-src:
合計 8
-rw-r--r-- 1 nojima nojima 46  2 月 17 03:15 CMakeLists.txt
-rw-r--r-- 1 nojima nojima  34  2 月 17 04:21 config.h.in
-rw-r--r-- 1 nojima nojima 91  2 月 17 03:10 hello.c
$
```

今回はビルド用ディレクトリ (hello-build) を作り、移動します。

```
$ ls
hello-src
$ mkdir hello-build
$ cd hello-build
```

cmake を実行します。

```
$ cmake ../hello-src
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
... 中略...
-- Looking for exit
-- Looking for exit - found
-- Configuring done
-- Generating done
-- Build files have been written to: /.../cmake-test/hello-build
$ ls
CMakeCache.txt  CMakeFiles  Makefile  cmake_install.cmake  config.h
```

自動的に環境チェックが行われ Makefile/config.h が出来上がります。 exit 関数も見つかったとの表示が行われまし

た。ここで make してみます。

```
$ make
Scanning dependencies of target hello
[100%] Building C object CMakeFiles/hello.dir/hello.c.o
Linking C executable hello
[100%] Built target hello
$ ls -F
CMakeCache.txt  CMakeFiles/  Makefile  cmake_install.cmake  config.h  hello*
$ ./hello
hello world
yes, this system has exit()
$
```

CMakeLists.txt から無事に実行バイナリ (hello) が出来上がりました。また、 defined(HAVE_EXIT) も True となり、 exit() 関数がある時のコードもコンパイルされています。

26.3 IDE 用のプロジェクトファイルを生成してみる

cmake を引数無しで実行すると、 help が出てきます。このヘルプの文章の中に、どんな IDE 用のプロジェクトファイルを生成できるかについて説明があります。試しに手元の Debian マシンで実行すると、

```
$ cmake
... 中略..
The following generators are available on this platform:
  Unix Makefiles               = Generates standard UNIX makefiles.
  CodeBlocks - Unix Makefiles = Generates CodeBlocks project files.
  Eclipse CDT4 - Unix Makefiles
                               = Generates Eclipse CDT 4.0 project files.
  KDevelop3                    = Generates KDevelop 3 project files.
  KDevelop3 - Unix Makefiles   = Generates KDevelop 3 project files.
$
```

ここでは試しに先ほどの hello-build ディレクトリ以下で KDevelp3 project ファイルを生成してみます。

```
$ cmake -G KDevelop3 ../hello-src
... 中略...
$ ls
MakeCache.txt  Makefile             config.h        hello.kdevelop.filelist
CMakeFiles     cmake_install.cmake  hello.kdevelop  hello.kdevses
$
```

確かに KDevelp3 用のプロジェクトファイル (hello.kdevelop 等) が生成されています。

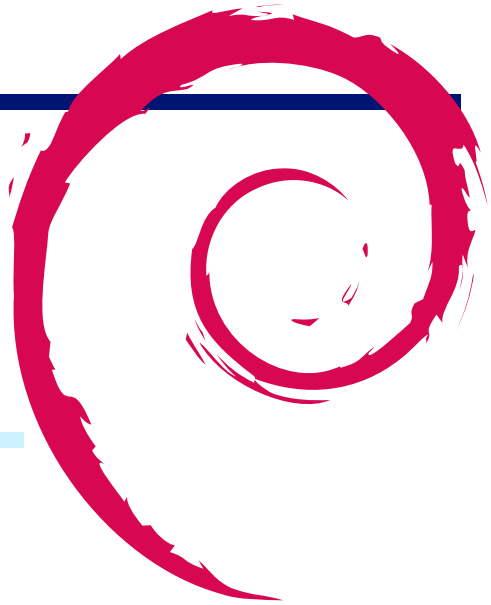
26.4 おわりに

cmake は KDE の他にも mysql でも採用されています。また、 wikipedia(`http://ja.wikipedia.org/wiki/CMake`) によれば、利用しているアプリケーションも続々増えている模様です。

使いこなせると強力なツールとなりそうな感じです。皆さんも使ってみてはいかがでしょうか? Debian なら aptitude で簡単に導入できますので、是非試してみてください。

26.5 参考文献

- `http://www.cmake.org/` CMake 本家
- `http://www.cmake.org/cmake/help/cmake_tutorial.html` CMake チュートリアル
- `http://drdobbs.com/cpp/184405251?pgno=1` DDJ ジャーナルの記事



27 Apache2 / HTTP サーバから始める Debian

岩松 信洋

普段はちょっと開発者寄りな話をしている Debian 勉強会ですが、今回は OSC 出張企画として、ユーザー視点の勉強会を開催します。今回はよく使われていると思われる Apache2 / HTTP サーバ に焦点を当ててみます。

27.1 はじめに

Debian は 日本では HTTP サーバとして利用されているように見えませんが、世界では一番採用されている Linux ディストリビューションになったようです。[*63] この記事によると、利用されている理由は HTTP サーバパッケージの種類が多くある事が理由の一つに挙げられています。 Debian を HTTP サーバとして利用している理由を実際に使っている方に聞いてみたところ、理由はこれだけではないことが分かりました。 Debian のパッケージングシステム、 APT、 Apahce モジュールパッケージの多さ、 Web アプリケーションで採用される P 言語(Perl, Python, PHP) のサポートなどがあり、一番良い点として挙げられたのは設定ファイルの柔軟性についてでした。

Debian の Apache2 / HTTP サーバ は Red Hat 系 と違い、 Debian 特有の構成になっています。これは他のディストリビューションしか知らない人にとっては難しいかもしれません。しかし Debian 特有の構成を理解すると、他のディストリビューションとのメリット、デメリットが見えてくると思います。というわけで今回は、 Debian の Apache2 / HTTP サーバ (以下、 Apache2) について勉強していきましょう。

27.2 Debian の Apache2 バージョン

まず、 Debian で提供されている Apache2 のバージョンを見てみます。表 20 にまとめました。 Upstream と比べると少し古いですが、機能的には問題ないでしょう。 RHEL、 CentOS (**バージョン** `2.2.15-15`) と比べても特にバージョンが古いというわけでもありません。

ディストリビューション	stable	testing	unstable	experimental	upstream
バージョン	2.2.16-6+squeeze6	2.2.22-1	2.2.22-1	-	2.4.1

表 20 Debian ディストリビューションと Apache2 のバージョン

27.3 Debian のパッケージ構成とパッケージのインストール

次に Apache2 のパッケージ構成とインストール方法について説明します。

[*63] `http://w3techs.com/blog/entry/debian_is_now_the_most_popular_linux_distribution_on_web_servers`

27.3.1 パッケージ構成

Debian の Apache2 で提供されているパッケージは以下の通りです。 HTTP サーバの処理モデルごとにパッケージ（ apache2-mpm-worker、 apache2-mpm-prefork、 apache2-mpm-event、 apache2-mpm-itk）が分離されていることがわかります。これにより自分の用途に合わせたパッケージをインストールできます。 Red Hat 系は一つのパッケージに纏まっていて、処理モデル毎にサフィックスをつけています（ 例: httpd.worker）。

パッケージ名	パッケージの説明
apache2	Apache HTTP サーバメタパッケージ
apache2-mpm-worker	スレッドモデル Apche HTTP サーバ
apache2-mpm-prefork	非スレッドモデル Apache HTTP サーバ
apache2-mpm-event	イベントドリブンモデル Apache HTTP サーバ
apache2-mpm-itk	マルチユーザ環境 Apache HTTP サーバ
apache2.2-common	Apache HTTP サーバ 共通ファイル
apache2.2-bin	Apache HTTP サーバの共通バイナリファイル
apache2-utils	ウェブサーバ用ユーティリティプログラム
apache2-suexec	Apache2 mod-suexec 用 基本 suexec プログラム
apache2-suexec-custom	Apache2 mod-suexec 用 設定可能 suexec プログラム
apache2-dbg	Apache HTTP サーバ デバッグシンボルファイル
apache2-prefork-dev	非スレッドモデル Apache HTTP サーバ 開発用ヘッダファイル
apache2-threaded-dev	マルチスレッドモデル　Apache HTTP サーバ 開発用ヘッダファイル
apache2-doc	Apache HTTP サーバドキュメント

表 21　Debian で 提供される Apache2 パッケージ

次にパッケージの依存関係図を図 27.3.1 に示します。依存関係が複雑なのでユーザは不安になるかもしれません。しかし Debian では強力なパッケージ管理ツール APT によって気にする事なくインストールできます。

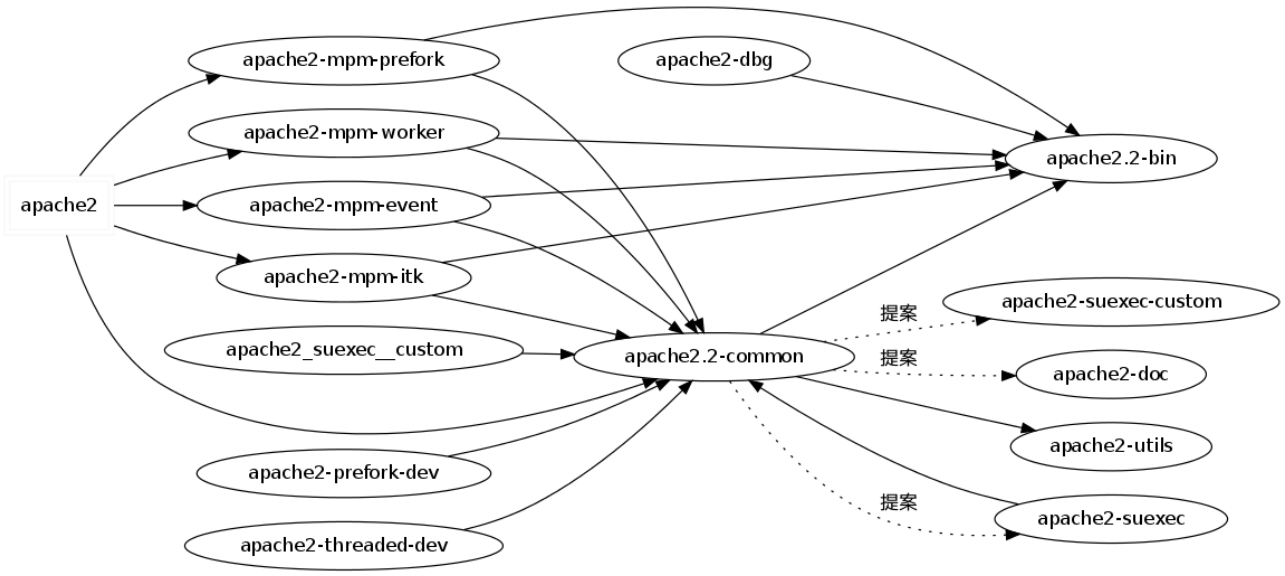


図 23　Debian でのパッケージ依存関係

27.3.2 インストール

Debian で Apache2 をインストールする場合は apt-get install コマンドを使います(図 27.3.2) 。

Debian では apache2 というメタパッケージを使ってインストールすることが多いです。 apache2 をインストールすると、 apache2-mpm-worker がインストールされます。他の HTTP サーバパッケージをインストールしたい場合は、各々のパッケージを指定してインストールする必要があります。

また CentOS などでは、「 httpd」 パッケージとして提供されているのでパッケージ名が異なります。普段は他のディストリビューションを使っている人は注意しましょう。

```
$ sudo apt-get update          // リポジトリを更新
$ sudo apt-get install apache2 // apache2 パッケージをインストール
```

図 24　Debian で Apache2 をインストールする

27.3.3 Apache HTTP サーバの起動と停止

Debian は「 インストールしたものは使う」というポリシーなので、インストール完了の時点で既に Apache HTTP サーバは起動しています。停止したい場合には root 権限で「 /etc/init.d/apache2 stop」 を実行します。起動したい場合は「 /etc/init.d/apache2 start」、再起動したい場合には「 /etc/init.d/apache2 restart」を実行します。図 27.3.3 に例を示します。

```
$ ps ax | grep apache2 // apache2 のプロセスを確認
10034 ?        Ss     0:05 /usr/sbin/apache2 -k start
13008 ?        S      0:00 /usr/sbin/apache2 -k start
( 省略)
$ sudo /etc/init.d/apache2 stop // apache2 を停止
$ ps ax | grep apache2 // apache2 のプロセスを確認
16833 pts/1    S+     0:00 grep apache2
$ sudo /etc/init.d/apache2 start //apache2 を開始
10048 ?        Ss     0:05 /usr/sbin/apache2 -k start
13024 ?        S      0:00 /usr/sbin/apache2 -k start
( 省略)
```

図 25　Apache2 の起動と停止

デフォルトの状態では、マシンを立ち上げ時に HTTP サーバが起動するようになっています。マシン立ち上げ時に HTTP サーバの起動しないようにするには、ランレベル毎のサービス起動スクリプトを制御するツール update-rc.d を使います。

全てのランレベルで apache2 を起動させないようにするには、コマンドにサービス名と remove を指定して実行します。またインストール直後のデフォルトの状態に戻したい場合には、コマンドにサービス名と default を指定して実行します。実行例を図 27.3.3 に示します。

```
$ sudo update-rc.d -f apache2 remove // 全てのランレベルで apache2 を起動させないようにする
$ sudo update-rc.d -f apache2 default //サーバ起動をデフォルトの状態に戻す
```

図 26　ランレベルの制御

Red Hat 系では chkconfig を使いますが、 Debian でも提供されています。しかし、 chkconfig は RedHat 系のサービス管理ツールなので Debian ではうまく動作しないことがあるようです。同様のツールとして sysv-rc-conf があるのでこちらを使ったほうがいいでしょう。図 27.3.3 に簡単な使い方を説明します。

```
$ sudo apt-get install sysv-rc-conf // sysv-rc-conf パッケージをインストール
$ sudo sysv-rc-conf --list // 現在の状態を出力
apache2      0:off1:off2:on3:on4:on5:on6:off
bootlogd     S:on
( 中略)
$ sudo sysv-rc-conf --level 2 apache2 off // ランレベル 2 の apache2 を無効にする
$ sudo sysv-rc-conf --list | head -1 // 現在の状態を出力
apache2      0:off1:off2:off3:off4:off5:off6:off
$ sudo sysv-rc-conf --level 2 apache2 on // ランレベル 2 の apache2 を有効にする
$ sudo sysv-rc-conf --list | head -1  // 現在の状態を出力
apache2      0:off1:off2:on3:off4:off5:off6:off
```

図 27　Apache2 の起動と停止

27.4　Apache2 の設定ファイル

Red Hat 系の場合、主な設定は /etc/httpd/conf/httpd.conf で行い、 include されるファイルは /etc/httpd/conf.d/ディレクトリに格納しますが、 Debian の場合は表 22 のようになっています。

設定ファイル	内容
/etc/apache2/apache2.conf	基本設定
/etc/apache2/httpd.conf	オーバーライドする設定
/etc/apache2/conf.d/	基本設定の中で Include するファイルを格納する
/etc/apache2/ports.conf	ポートの設定
/etc/apache2/envvars	環境変数の設定
/etc/apache2/mods-available/	利用可能なモジュール設定
/etc/apache2/mods-enabled/	利用中のモジュール設定
/etc/apache2/sites-available/	利用可能なサイト設定
/etc/apache2/sites-enabled/	利用中のサイト設定
/var/www	ドキュメントルート
/usr/lib/cgi-bin	cgi-bin
/var/log/apache2	Apache2 ログ

表 22　Debian の Apache2 設定ファイル群

apache2.conf には図 27.4 のような行があり、 apache2.conf から各設定が読み込まれるようになっています。Apache2 の設定をを変更する場合、 apache2.conf を変更せず、 httpd.conf や ports.conf を変更します。

```
( 省略)
# Include module configuration:
Include /etc/apache2/mods-enabled/*.load
Include /etc/apache2/mods-enabled/*.conf

# Include all the user configurations:
Include /etc/apache2/httpd.conf

# Include ports listing
Include /etc/apache2/ports.conf
( 中略)
# Include generic snippets of statements
Include /etc/apache2/conf.d/

# Include the virtual host configurations:
Include /etc/apache2/sites-enabled/
```

図 28　apache2.conf の内容

27.5 サイトを設定する

Debian は/etc/apache2/sites-available/default に apache2 のデフォルトのサイト設定を格納しています。サイトを一つだけ構築する場合はこのファイルを変更し、 apache2 を再起動すれば設定された内容で apache2 が立ち上がります。再起動する方法は図 27.5 の通りです。

```
$ sudo /etc/init.d/apache2 restart
```

図 29　Apache2 の再起動

Debian の apache2 で複数のサイトを立ち上げる場合、 httpd.conf や apache2.conf は編集しません。サイト別に設定を記述し、/etc/apache2/sites-available/ディレクトリに格納します。そして、そのサイトを設定を有効にするコマンド「 a2ensite」実行し、 apache2 を再起動します。

簡単な手順を説明します。例えば、 test.example.org というサイトを立ち上げるとします。内容は図 27.5 のようになるでしょう。

```
<VirtualHost *>
    ServerAdmin admin-test@example.org
    ServerName test.example.org
    DocumentRoot /home/test/public_html/
    <Directory />
        Options FollowSymLinks ExecCGI Includes
        AllowOverride None
    </Directory>
</VirtualHost>
```

図 30　サイトの設定例

そしてこのサイト設定を/etc/apache2/sites-available/test に格納します。格納した後、サイトを有効にする「 a2ensite コマンド」に有効にしたいサイトの設定ファイル名を指定して実行します。実行すると /etc/apache2/sites-enabled/ にシンボリックリンクが張られ設定が有効になります。有効にしただけでは、稼働している httpd サーバには設定が反映されていないため、 httpd サーバを再起動します(図 27.5）。

```
$ ls -l /etc/apache2/sites-enabled/
合計 0
lrwxrwxrwx 1 root root 26 2011-03-20 08:23 000-default -> ../sites-available/default
lrwxrwxrwx 1 root root 30 2011-03-20 08:23 default-ssl.old -> ../sites-available/default-ssl
$ sudo a2ensite test // test を有効にする
Enabling site test.
Run '/etc/init.d/apache2 reload' to activate new configuration!
$ ls -l /etc/apache2/sites-enabled/
合計 0
lrwxrwxrwx 1 root root 26 2011-03-20 08:23 000-default -> ../sites-available/default
lrwxrwxrwx 1 root root 29 2012-03-10 06:24 test -> ../sites-available/test
lrwxrwxrwx 1 root root 30 2011-03-20 08:23 default-ssl.old -> ../sites-available/default-ssl
$ sudo /etc/init.d/apache2 restart // Apache2 を再起動
```

図 31　サイトを有効にする

サイトの設定を無効にする場合には、サイトを有効にする「 a2dissite コマンド」に無効にしたいサイトの設定ファイル名を指定して実行します。実行すると/etc/apache2/sites-enabled/ からシンボリックリンクが削除されます。サイト設定を無効にした後は、有効時と同様に httpd サーバを再起動する必要があります(図 27.5）。

このように Debian ではサイトの設定を分離し、サイト毎に状態を管理することができます。他のディストリビューションでは include 等を使って管理することができますが、ファイル内容を変更する必要があり非常に手間です。 Debian はシンボリックリンクを使うことによって Apache2 の設定ファイルを変更せずにサイト設定の有効･無効ができるようになっています。

```
$ sudo a2dissite test // test を無効にする
Site test disabled.
Run '/etc/init.d/apache2 reload' to activate new configuration!
$ ls -l /etc/apache2/sites-enabled/
合計 0
lrwxrwxrwx 1 root root 26 2011-03-20 08:23 000-default -> ../sites-available/default
lrwxrwxrwx 1 root root 30 2011-03-20 08:23 default-ssl.old -> ../sites-available/default-ssl
```

図 32　サイトを無効にする

27.6　モジュールを有効/無効にする

Debian のモジュールに関する設定はモジュール毎の設定ファイルとして mods-available ディレクトリに格納されています。それらのうち、実際に有効にするものがシンボリックリンクとして mods-enabled ディレクトリに張られます。シンボリックリンクは手動で行わず、モジュールを有効にする場合には a2enmod コマンド、無効にする場合には a2enmod コマンドを使います。図 27.6 に mod_info を有効にする例と mod_info を有効にする例を示します。

```
$ sudo a2enmod info // mod_info を有効にする
$ sudo a2dismod info // mod_info を無効にする
```

図 33　mod_info を有効にする・ mod_info を無効にする

27.7　その他

その他、注意すべき点をいくつか教えてもらったので紹介します。

27.7.1　libapache2-mod-php5 と apache2-mpm-prefork

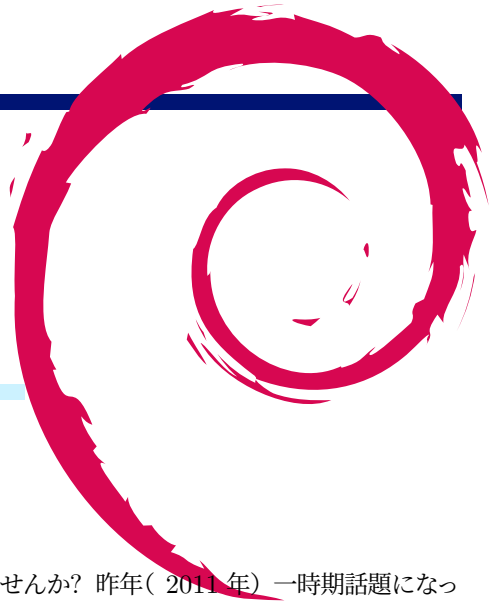
Apache2 上で mod-php5 を使いたい場合、 apache2-mpm-worker は使えない点に注意してください。これは PHP5（ mod-php5）の制限で、スレッドで動作することができないためです。 libapache2-mod-php5 をインストールすると、を使いたい場合、 apache2-mpm-worker が削除され、 apache2-mpm-prefork がインストールされます。 PHP ユーザの方は注意しましょう。

27.7.2　再起動確認について

その他、 Debian で Apache2 を使う理由として、再起動確認を行うという点があります。例えば glibc が更新されたとき、サーバ系は再起動する必要があるのですが、 Red Hat 系では再起動してくれず、管理者が手動で行う必要があります。 CentOS を使っている会社ではデーモンを再起動しないとならないアップデートがあったかどうかチェックするツールをわざわざ作って管理していたりするようです。しかし Debian では再起動の確認が行われる（ 設定によって自動再起動も可能）ので、管理者の手を煩わせません。このような細かいところに気を使ってくれるのも Debian の良い所です。

27.8　まとめ

Debian のパッケージ古いというのは昔の話です。 stable と testing へのセキュリティバグへの対応があり、その対応も他のディストリビューションと比べて比較的早いです。よく指摘される Debian の設定ファイル、ディレクトリ構造ですが、今回の説明で独自なのは理由があり、理にかなっている事が分かります。またこれらを容易に操作できるように、専用のツールもあります。パッケージによる、細かいところへの気配りができるところが Debian のよいところだと思います。今回の Apache の話が気になった方はとりあえず Debian 使ってみてはいかがでしょうか。



28 Debian での node 入門

上川純一

JavaScript[5, 6] でプログラムをガリガリ書きたいとおもったことはありませんか? 昨年(2011 年) 一時期話題になっていた node を Debian で試してみましょう。 node[*64]はイベントベースのサーバサイド JavaScript エンジンとフレームワークです。何に使えるかというと、 JavaScript でウェブサーバが書きやすくなっています。シェルスクリプトでコードを書く代わりに node を使って JavaScript を使うこともまぁ出来ますがウェブサーバが便利になるのが一番大きいと思われます。特徴はシングルスレッドなんだけどほとんどすべての I/O 処理を非同期イベント処理によって実現することによって、たくさんのリクエストを効率よく処理するあたりでしょうか。

node のパッケージは squeeze にはないですが、 wheezy にはすでに入っています。インストールは簡単:

```
# apt-get install nodejs
```

Node 本体は nodejs というパッケージ名で入っています[*65]が、関連するモジュールパッケージの名前は node-ではじまるようになっています。

node コマンドを実行すると JavaScript インタープリタの REPL インタフェースが起動します。ここでコマンドラインから JavaScript を適当に実行できるようです。この時点では単なる V8 のコマンドラインインタフェースですね。

```
$ node -v
v0.6.12
$ node
> console.log('hello world')
hello world
```

node のバージョン番号ですが、 node の 2012 年 4 月 1 日時点の安定版の最新版は 0.6 系列で、 0.7 系列は開発版という位置づけのようです。 0.8 系列がリリースされたらまた Debian の node も更新されるでしょう。

まとめると次のようになっています。

- 0.4: 2012 年 1 月まで Debian sid に入っていた安定版
- 0.6.12: 2012 年 4 月時点での最新安定版 (stable)
- 0.7.x: 開発版 (unstable)
- 0.8.x: 多分近い将来リリースされるだろう安定版

28.1 Debian 流儀での Node モジュールパッケージインストール

Debian パッケージで提供されている node のモジュールパッケージを使ってみましょう。

[*64] マニュアルなどには node と記述されていますが、通称は node.js のようです

[*65] すでに node というパッケージが存在するから node という名前がつけられなかったのだと思われます。

28.1.1 Node で CLI ツールを作ってみる

とりあえず、 node でコマンドラインインタフェース(CLI) のツールをつくってみたい、ので node cli をインストールして適当にコードを書いてみるという場面を想定してみます。 cli モジュールは Debian パッケージになっているので以下でインストールできます。

```
# apt-get install node-cli
```

とりあえず無駄に平均を計算してみるコードを書いてみました。

```
// Command-line tool to sum the stdin items.
var cli = require('cli');

cli.withStdinLines(function(lines, newline) {
    var sum = 0;
    var count = 0;

    for (var i = 0; i < lines.length; ++i) {
        console.log(lines[i]);
        if (lines[i] != '') {
            sum += parseInt(lines[i]);
            count ++;
        }
    }
    console.log('sum: ' + sum + ' avg: ' + sum / count);
});
```

コマンドラインで適当に実行してみたところ、結果が表示されました。

```
$ node sum.js < testdata.txt
10
15
200
8

sum: 233 avg: 58.25
```

28.2 npm – Node のパッケージ管理システム

Node のモジュールは npm で管理されています。 Debian パッケージになっていないパッケージなどは、 npm コマンドを利用して直接インストールすることも可能です。 Perl でいう CPAN、 TeX でいう CTAN のようなもののようです。 Debian パッケージを使うべきか npm を使って導入するべきか悩ましいところですが、 Debian パッケージになるまでにはどうしてもタイムラグがあるので、最新のコードをつかって開発する場合には npm を利用することになると思われます。ある程度こなれてきたらないパッケージは ITP するのがよいでしょう。

Debian の提供するモジュールパッケージは/usr/lib/nodejs 以下にインストールされますが、 Debian パッケージの npm を利用する場合は /usr/local/lib/nodejs 以下に入ります。

で、喜び勇んで手元で実行したところ Exception をはいて終了しました。どうやら node 0.6.2 に対応していない古いバージョンの npm (Node 0.4 系列ではうごいたはず) が現在パッケージされており、動かなくなっている模様です。 Bug#622628[*66]

28.3 npm のアップストリーム版をインストールしてみる

npm が使えないのは不便なので、 Debian パッケージになっていない node モジュールをインストールする方法として Debian パッケージではない npm を利用する方法を紹介します。

node 0.6 系列に対応している npm は 1.1 です。 npmjs[3] サイトからインストーラをダウンロードして実行します。 npm がインストールできたらモジュールは npm install コマンドでインストールできるようになります。

npm は sudo 前提で設計されているようです。 sudo で root 権限に昇格するのはビルド時に nobody 権限に切り替えるのに使うようです。

[*66] http://bugs.debian.org/622628

npm パッケージはシステムグローバルにもパッケージローカルにもインストールできますが、一般ユーザ権限でプロジェクトローカルに利用するためにパッケージをインストールすることを基本として設計されているようです。

`sudo npm install` **パッケージ名**だとカレントディレクトリ以下に sudo を発行したユーザ権限でインストールするようです。[*67]`sudo npm install -g` **パッケージ名** だと `/usr/lib/node_modules` 以下にインストールするようですが、権限は nobody のままです[*68]。

npm は開発者視点で便利なように設計されているようで、個人的におもしろいなと思ったのは -g をつけずに `sudo npm install` だけすると現在のディレクトリにあるプロジェクトの依存しているパッケージをカレントディレクトリにインストールするという挙動になることです。依存しているパッケージがどんどん変更されている熱いプロジェクトである node っぽい感じがします。 npm の作者のウェブサイトを読んでいると、システムワイドでインストールするのは CLI ツールなどに必要な時に限って、ウェブサイトにデプロイする用のコードではモジュールはプロジェクトのディレクトリにインストールすることを推奨すると説明しています。

```
# apt-get install nodejs nodejs-dev
# curl http://npmjs.org/install.sh | sh
$ sudo npm install -g express ejs socket.io
npm http GET https://registry.npmjs.org/express
npm http GET https://registry.npmjs.org/ejs
npm http GET https://registry.npmjs.org/socket.io
npm http 304 https://registry.npmjs.org/socket.io
npm http 200 https://registry.npmjs.org/ejs
npm http GET https://registry.npmjs.org/ejs/-/ejs-0.6.1.tgz
npm http 200 https://registry.npmjs.org/express
  .
  .
/usr/bin/express -> /usr/lib/node_modules/express/bin/express
ejs@0.6.1 /usr/lib/node_modules/ejs
express@2.5.8 /usr/lib/node_modules/express
├── qs@0.4.2
├── mkdirp@0.3.0
├── mime@1.2.4
└── connect@1.8.6
socket.io@0.9.2 /usr/lib/node_modules/socket.io
├── policyfile@0.0.4
├── redis@0.6.7
└── socket.io-client@0.9.2
```

ディレクトリ構成をまとめました。

表 23　Debian パッケージおよび npm でインストールされる場所

	ディレクトリ
Debian パッケージ	`/usr/lib/nodejs`
Debian の npm	`/usr/local/lib/nodejs`
npm install -g	`/usr/lib/node_modules`
npm install	`./node_modules`

マニュアルなどは npm コマンドを実行すると表示されるヘルプが充実しています。

```
$ npm help
$ npm help npm
$ npm help install
```

28.3.1　npm を使っている場合のプログラム実行

カレントディレクトリにインストールしたときはよいのですが、そうではない場合は、`/usr/lib/node_modules` 以下にインストールされても Debian の nodejs の標準のモジュールパスに含まれていないためモジュールがロードされません。

ひとつの回避策としてはモジュールを探しに行く PATH を追加するという方法があります。 `NODE_PATH` 環境変数を指定すれば追加でロードされるようになります。

[*67] ビルド自体は nobody 権限で行うようです
[*68] 挙動としてはおかしいので操作を間違っているかバグのように思われる

```
$ NODE_PATH=/usr/lib/node_modules node ./program.js
```

28.3.2 パッケージのメタデータ: package.json

推奨されているのはパッケージに必要なモジュールを package.json に記述して、 npm link コマンドを実行すればパスの追加は必要なくなります。

```
$ npm link
```

とすると必要なモジュールをプロジェクトのディレクトリの./node_modules にリンクしてくれるということでした。*69*70

npm のメタデータは ./package.json です。 npm help json (man npm-json.1) に詳しく説明されています。す。とりあえずは name/version フィールドさえあればよくて、 dependencies を追加すると npm install や npm link コマンドで利用してくれるようです。

インストールに必要なファイルの一覧や、 node-waf の実行方法などが記載されているのでこれさえあれば Debian パッケージを自動で生成することも可能な気がします。

手元で作成してみた npm link のためだけの最低限な内容の package.json を紹介します

```
{
    "name": "aptserver",
    "version": "v0.0.1",
    "dependencies": {
        "cli": "",
        "express": "",
        "ejs": ""
    }
}
```

28.4 とりあえず簡単なサーバを書いてみた

apt-cache search をして出力を返すだけの簡単なサーバを書いてみました。イベントドリブンな部分としては、 HTTP request のハンドラーを app.get() で登録している部分と、 apt cache の出力を aptCache.stdout.on で登録したハンドラーで取得して aptCache.on exit ハンドラーで終了したら HTTP レスポンスを返すようになっている部分でしょうか。

書いてみて気づきましたが、いまいち node を使うメリットが出てないきもします。

*69 マニュアルにはシンボリックリンクをすると書いているけど、モジュールはハードリンクしてました。

*70 逆方向に現在作業中のパッケージを/usr/lib/node_modules/以下にシンボリックリンクしてくれます。作業した内容がすぐに反映するので便利といえば便利。

```
/*
 * A simple server which serves apt cache search results.
 */
var child_process = require('child_process');
var cli = require('cli');
var url = require('url');
var ejs = require('ejs');

cli.parse({
    port: ['p', 'HTTP server will listen on this port.', 'number', 8088]
});

cli.main(function cliMain(args, options) {
    var express = require('express');
    var app = express.createServer();
    // set view options.
    app.set('view engine', 'ejs');
    app.set('view options', { layout: false });
    app.set('views', __dirname + '/views');

    // Set up routes.
    app.get('/', getSlash);
    app.get('/search', getSearch);

    console.log('Start listening on http://localhost:' + options.port + '/');
    app.listen(options.port);
});

/** handler for '/' request */
function getSlash(request, response) {
    response.render('index.ejs');
}

/** handler for '/search?' request */
function getSearch(request, response) {
    var query = request.query.q;
    var aptCache = child_process.spawn('apt-cache',
                                       [ 'search', query ]);
    /** Output of apt-cache search, to be used for response. */
    var responseString = '';
    aptCache.stdout.on('data', function handleAptCacheStdout(data) {
        responseString += data;
    });
    aptCache.stderr.on('data', function handleAptCacheStderr(data) {
        responseString += data;
    });
    aptCache.on('exit', function handleAptCacheExit(exitCode) {
        // apt-cache finished executing, send repsponse back to the server.
        response.render('search.ejs',
                        {locals:{query: query,
                                 response: responseString}});
    });
}
```

ejs HTML テンプレートファイル `views/search.ejs` はこんな内容になりました。

```
<!-- -*- html; -*- -->
<html>
  <body>
    <form
      method=GET
      action='/search'>
      <input type=text name=q></input>
      <button>search</button>
    </form>
    <h3>Search for: <%= query %></h3>
    <pre>
<%= response %>
    </pre>
  </body>
</html>
```

追加モジュールは express, ejs, cli を使いました。それぞれを簡単に紹介すると以下です

- express: ウェブアプリケーションフレームワークとして node で最もポピュラーなモジュール、リクエストベースのルーティングなどを担当。
- ejs: HTML テンプレートエンジンとしておそらく最もポピュラーなモジュール。
- cli: コマンドラインオプションをパースしてくれるモジュール。

28.5 最後に

今回は node を Debian で使う方法を紹介してみました。モジュールのパッケージングが更新においついていない感じなのでもうしばらくしたらまた良くなっているかもしれませんが、そうこうしているうちに node 0.8 がリリースされてしまいそうです。 node の頻繁な更新とそれによってモジュールが複数バージョン必要になってくる現状から、 nave / nvm などの環境管理ツールで複数バージョンをインストールして利用するということもよくやられているようです。この資料が Debian で node を活用するきっかけ、およびパッケージ化活動の一助になれば幸いです。

参考文献

[1] Node.js v0.6.12 Manual & Documentation `/usr/share/doc/nodejs/api/index.html`

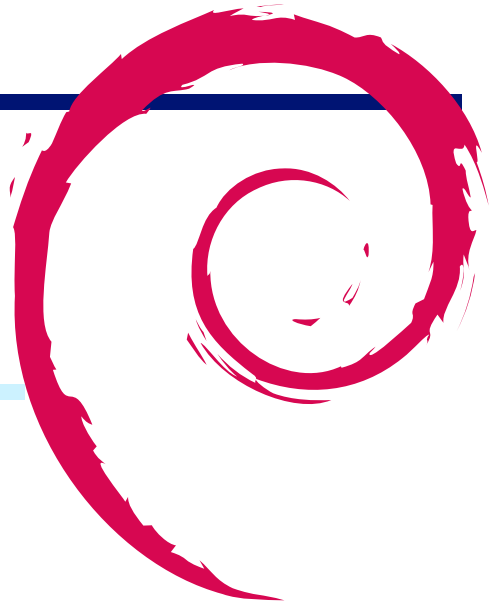
[2] Alioth の pkg-javascript-devel メーリングリスト `http://lists.alioth.debian.org/pipermail/pkg-javascript-devel/`

[3] NPM ホームページ `http://npmjs.org/`

[4] nodejs for Debian `/usr/share/doc/nodejs-dev/README.Debian`

[5] JavaScript `https://developer.mozilla.org/ja/JavaScript`

[6] Standard ECMA-262 ECMAScript Language Specification 5.1 `http://www.ecma-international.org/publications/standards/Ecma-262.htm`



29 Android 機で Debian

野島 貴英

29.1 はじめに

携帯電話、タブレット型 PC など、高性能の情報端末を持ち歩くのが一般的になってきました。ここでは、これら情報端末のうち、 Android OS を搭載した情報端末に Debian を入れ、 Debian 開発者の環境を築いてみようとした事について述べます。

29.2 材料: Android 端末 Barnes & Noble Nook Color について

手頃な Android 端末として、たまたま手元に Barnes & Noble 社 (以下 B&N) の Nook Color という製品[*71]があります[*72]。値段も日本で輸入した場合、 2 万円前半〜後半ぐらいで安く、電子書籍ビューアということもありデザインも小型でそれなりに薄く軽量です。これは元々、 PDF にしたカラーの本(漫画も) を持ち歩きながら読めればと思って買っていたものでした。

これが Debian の開発環境としても動いたら素敵と思い、こちらを早速利用する事にします。[*73]

29.3 Android 端末の Debian 動作の方針

Android 端末を利用して Debian 関係のディストリビューションの OS を動作させる場合、後に述べる 2 つの方針が取れます。

29.3.1 方針 1: Android 端末の機能を最大限活用する

本方針は、 Android 端末はベースの OS が基本的に Linux であることを最大限利用します。

ここで、

1. Android 端末の CPU にあわせた Debian 関係の Linux のファイルシステムを/(ルート) ファイルシステムから SD カードや、内臓メモリへ用意します。
2. Android 端末に Android アプリの VNC Viewer[*74]を搭載しておきます。
3. 何らかの方法で Android 端末上の管理権限を奪取した状態で、先ほど用意した/ファイルシステムをマウントし、chroot します。これで Debian のバイナリが動作できるようになります。
4. 最後に Debian のバイナリを利用して vncserver を立ち上げ、 Android アプリの VNC Viewer で 127.0.0.1:5901 に接続し、 Debian を利用します。

[*71] http://www.barnesandnoble.com/p/nook-color-barnes-noble/1100437663
[*72] 製品の写真は権利関係がよく判らないので割愛させてください。詳しくは先の URL 参照。
[*73] 実は、自分はフィーチャーフォン(ガラケーともいう) しか持っておらず、 Android 端末はこれしか持ってなかったことは秘密です。
[*74] http://code.google.com/p/android-vnc-viewer/

という方法がよく利用されます。

この方法は、 Android 端末で管理権限さえ奪取できていれば Debian 関係の Linux ディストリビューションを動作させる事ができます。さらに良いことに、本体の Android 端末が最初から搭載している無線ネットワークもそのまま利用できるるため、非常に都合が良いです。実際、自分は未評価ですがこれらの諸々の手続きを全部アプリに詰め込んでしまったものに、

- Ubuntu を動作させる Android アプリ： ubuntu instller free `https://play.google.com/store/apps/details?id=com.zpwebsites.ubuntuinstall&hl=ja`
- Debian を動作させる Android アプリ: Lil' Debi `https://github.com/guardianproject/lildebi/wiki`

などがあるようです。

29.3.2 方針 2: Android OS を使わずそのまま Debian をブートする

Android 端末は大抵 ARM ベースの CPU で動いてます。 Debian も ARM ベースのバイナリを用意しています。これはつまり、うまくカーネル/アプリケーションを Android 端末のハードの仕様に合わせることができ、ブートさせる事ができれば、 Android OS の代わりに Debian をそのままブートさせて使えるはずです。

ただ、 Android 端末は ARM ベースの CPU は使っているものの、グラフィックス（ 液晶画面出力）、タッチパネル、サウンド、 Wi-Fi 等は各端末のハード独自のものでかつ、仕様/設計は未公開であったり、 Linux カーネル本体の機能だけでは対応できなかったりする為、これらの周辺デバイスを利用できるようにする為のハードルは高い状況です。特に、 Wi-Fi はおろか、通常のデスクトップ PC を使っているときにはあまりに基本的な機能で気にもとめなかったような機能（ キー入力、マウス入力、画面に文字を描画）すらも最初から未対応がほとんどですので、本方針を取るにはそれ相応のスキルが必要です。（ VGA BIOS とか、 IBM PC AT の BIOS の規格が大変ありがたく見えてきます）

但し、本方針でもし Debian をそのまま利用できれば、より自由なソフトウェア開発環境を搭載した携帯端末を手にできる事になる為、非常に魅力的ではあります（ よね? ）

29.4 数々の障害と方針変更

先に述べた方針 1 が手軽なはずなので、こちらの方針をとって作業を進めていました。が、実は以下に列挙する問題にあたってしまい、解決出来なかった為、不完全ながらも方針 2 を取らざるを得ない状況となってしまいました。

1. B&N の Nook Color は、 Android 2.1〜2.2 OS を改造して電子書籍端末にした製品のため、通常の Android OS とは異なります。そのため、 Android 端末でそのまま動作させることができるようなアプリをどうやってもインストールできませんでした。つまり頼みの VNC Viewer を動作させる事が出来ませんでした。また、こちらの原因を調べようにも、ソース公開義務の発生しない肝心のライブラリ、アプリケーションフレームワーク、アプリケーション[*75] が全くの非公開であるため原因追求が困難です。
2. 管理権限を奪取して adb が使えるようにしたのですが、 USB 経由で利用する Nook Color 側の adbd の動作が非常に不安定で、一度でも USB を外す/PC の電源を落とす/何もせず時間を空けると次からどうやっても端末側 adbd に接続できなくなる現象に悩まされました（ 管理権限奪取の方法を最初からやり直すと復活できる事がたまにありましたが、なぜか失敗する事が多いです。）さらに悪い事に、端末側ファームを 1.2.2 にアップデートすると、もはやどうやっても adbd に接続できなくなってしまいました[*76]。

そこで、方針 1 をあきらめ、方針 2 を取る事にしました。

[*75] Android OS の用語となります。基本的な Android OS の構造: `http://developer.android.com/images/system-architecture.jpg`

[*76] 後の調査でわかったことですが、 Wi-Fi 経由で端末側 adbd に接続すると安定するかも? との情報もあります。しかし、無線 LAN 環境を自分は持っていない為こちらも未評価です

29.5 再考: Nook Color

Debian を Android 端末の機能を最大限活用して動作させるのにいろいろてこずる Nook Color ですが、このやり方を諦めれば、唯一 Debian を動作させるのに救いのある機能があります。 Nook Color は miniSD を挿入して利用できるのですが、ここに OMAP 用ブート形式[*77]のブートイメージを書きこんでおくと、こちらからそのままブートしてしまう機能(脆弱性?) があります。

そこで、ここにブートイメージを書き込んで、ブートすれば Debian を利用できるのでは? という事を試しました。

29.6 Debian の Nook Color 用ブートイメージを作る

29.6.1 方針

ここでは、以下の方針にそってブートイメージを作成します。

1. qemu を使って Debian の ARM 用ディスクイメージを構築します。
2. Nook Color の root 奪取に使う nooter0.2.zip のカーネル/initrd イメージは Nook Color の USB を RNDIS 用のネットワーク I/F にセットアップする能力を持ちます。これを利用すると、コミュニケーションポートとして USB が非常に便利になるのでこれらを拝借します。
3. 以上 1,2 を合体した OMAP 用ブート形式の miniSD カードを作成します。

具体的なやり方を次に述べます。

29.6.2 やり方

用意するもの: Debian sid の動く PC, グローバル回線, miniSD カード 1 枚、必要なら miniSD カードリードライター(USB メモリに変換するタイプのコネクタでも可)

1. 諸々 PC 側に取り揃えます。

```
$ sudo aptitude install debian-installer-6.0-netboot-armel qemu-system util-linux kpartx dump unzip gvncviewer
$ wget http://cgit.openembedded.org/openembedded/plain/contrib/angstrom/omap3-mkcard.sh; chmod 755 omap3-mkcard.sh
```

2. イメージディスクの準備をします。

```
$ qemu-img create -f raw arm-versatile.img 5G
```

3. インストール作業を行う為、以下のコマンドを実行します[*78]。

```
$ qemu-system-arm -M versatilepb -kernel /usr/lib/debian-installer/images/armel/versatile/vmlinuz-2.6.32-5-versatile \
 -initrd /usr/lib/debian-installer/images/armel/versatile/initrd.gz -m 256 -drive file=./arm-versatile.img,if=scsi,\
 bus=0,unit=0,cache=writeback -append "root=/dev/ram desktop=lxde priority=medium"
```

4. Debian インストーラがウインドウに立ち上がり、通常のインストーラメニュー形式の Debian のインストールを進める事ができるようになります。ネットワークも自動的に認識 (IP アドレスなども) され、 qemu の持つネットワークの仕組みで PC に接続されたネットワークがそのままネットワーク経由のインストールに利用されます。表 24 を選択しつつインストールを行います。
5. インストールが完了し、最後にシャットダウン画面になります。最後 Reboot system の指示が出たのを確認してそのまま qemu を Ctrl+C で停止させます。
6. インストール完了後のディスクイメージに内臓されているカーネルイメージ、 initrd イメージを取り出します。

[*77] http://processors.wiki.ti.com/index.php/SD/MMC_format_for_OMAP3_boot。但しこのページのコマンド通りにそのまま fdisk 実行しても、 Debian ではブートイメージは作れないので注意。

[*78] LXDE デスクトップにする理由は、 GNOME だとメモリの必要量が多すぎて swap が発生してしまい、快適に動作しない為です

項番	設定項目	指定内容	備考
1	Choose language	Japanese 日本語	
2	ネットワークの設定	DHCP でネットワークを自動で設定	
3	インストールする Debian バージョン	sid	
4	ロードするインストーラコンポーネント	何も選ばない	何も選ばないが、「 続ける」だけは選択
5	ディスクのパーティショニング	ディスク全体を使う, すべてのファイルを 1 つのパーティションに	
6	ソフトウェアの選択	デスクトップ環境、ラップトップ、標準システムの 3 つ	お好みで
7	インストールする Linux カーネルのバージョン	3.2.0-2	

表 24　インストーラで選択する項目

```
$ mkdir debian
$ sudo kpartx -a ./arm-versatile.img
$ sudo mount -t ext3 /dev/mappoer/loop0p1 ./debian
$ cp debian/boot/initrd.img-3.2.0-2-versatile .
$ cp debian/boot/vmlinuz-3.2.0-2-versatile .
$ sudo umount ./debian
$ sudo kpartx -d ./arm-versatile.img
loop deleted : /dev/loop0
$
```

7. 以下のコマンドで取り出したカーネルイメージ、 initrd イメージで再度 qemu を立ち上げます。 xdm の画面が出るので、そのままログインします。すると LXDE のデスクトップ画面が現れます。

```
$ qemu-system-arm -M versatilepb -kernel ./vmlinuz-3.2.0-2-versatile \
 -initrd ./initrd.img-3.2.0-2-versatile -m 256 -drive file=./arm-versatile.img,if=scsi,\
 bus=0,unit=0,cache=writeback -append "root=/dev/sda1 "
```

8. LXDE の画面から、 lxterminal を開き、パッケージをアップデートします。さらに、 vnc サーバーの導入を行っておきます。

```
$ su root
Passwd: インストール時の root パスワード
# aptitude update;aptitude safe-upgrade;aptitude install tightvncserver
```

9. vnc サーバーのインストールが完了したら、 LXDE 端末画面でシャットダウンします。

```
$ sudo shutdown -h now
```

10. Power off のメッセージが出力されたら、 qemu を Ctrl+C で停止します。
11. miniSD を PC に差し込みます。 mount されてしまっているようであれば、**必ず umount** しておきます。(注: ここでは/dev/sdb1 としてマウントされてしまった事を過程します)

```
$ sudo umount /dev/sdb1
```

Tips:
なお、 PC の miniSD のリーダ/ライターを搭載している場合、 miniSD アクセス用のチップが Richo R5C822 を搭載している場合で(よくある)、 miniSD を挿入すると”mmc0: error -110 whilst initialising SD card” や、”mmc0: Reset 0x1 never completed” などが大量に dmesg に記録されて全くアクセスできない場合があります。この場合は以下の定義を/etc/modules に記載してリブートすると、安定して miniSD へアクセスできるよう

になります。

```
$ cat /etc/modules
... 中略...
# for R5C822
mmc_block
tifm_sd
$
```

12. miniSD に OMAP 用ブート形式のパーティションを作成します。
    注: **必ず miniSD の認識されているデバイスファイルを omap-mkcard.sh に指定してください!** ここでは/dev/sdb と仮定しています。間違ったデバイスファイルを指定すると該当ディスクの中身が消えてしまいます!

```
$ sudo ./omap3-mkcard.sh /dev/sdb
```

13. nooter0.2 配布先 (`http://www.mediafire.com/?cfddu9wt9d8dunl`) を epiphany などの WEB ブラウザでアクセスして、 nooter0.2.zip を入手します。
14. 解凍して、ディレクトリ nooter/にマウントします。

```
$ unzip nooter0.2.zip
$ mkdir nooter
$ sudo kpartx -a ./nooter_sdcard_40mb.img
$ sudo mount /dev/mapper/loop0p1 ./nooter
```

15. miniSD カードの 1 つ目のパーティション (/dev/sdb1) をマウントします。

```
$ mkdir miniSD
$ sudo mount -t vfat -o rw,uid=your-uid,gid=your-gid /dev/sdb1 ./miniSD
```

16. nooter の u-boot イメージ、カーネルイメージ、 initrd イメージをコピーして、 miniSD,nooter をアンマウントします。

```
$ cp nooter/* ./miniSD/
$ sudo umount ./nooter
$ sudo kpartx -d ./nooter_sdcard_40mb.img
loop deleted : /dev/loop0
$ sudo umount ./miniSD
```

17. miniSD の 2 つ目のパーティション (/dev/sdb2) をマウントして、 Debian イメージをまるごとコピーします。終わったらアンマウントします。

```
$ su root
Passwd:
# kpartx -a ./arm-versatile.img
# mount /dev/mapper/loop0p1 ./debian
# mount /dev/sdb2 ./miniSD
# cd debian
# dump -0 -f- ./ | (cd ../miniSD && restore -rvf- )
# cd ..
# umount ./debian
# umount ./miniSD
```

以上となります。

29.7 動作させる

早速動作させてみましょう。

1. Nook Color の電源ボタンを長押しして電源を OFF にします。
2. Nook Color の miniSD スロットに作った miniSD を挿入します。
3. Nook Color に製品付属の usb ケーブルを差し込み、 PC の USB に差し込みます。
4. usb ケーブルが橙→緑に変わる。同時に PC 側の dmesg を観察すると、 cdc_ether モジュールがロードされ、 usb ポートが NIC に変化するのが観察できます。
5. 以下のコマンドを打ち込んで、 Nook Color へ root でログインします。なお、 PC 側を 192.168.2.10 と仮決めし

ます。(Nook Color 側は nooter 由来の initrd ですと 192.168.2.2 に決め打ちでセットアップされます。)

```
$ sudo /sbin/ifconfig usb0 inet 192.168.2.10 netmask 255.255.255.0 up
$ ssh root@192.168.2.2
#
```

6. miniSD の Debian を動かし、 vncserver を立ち上げます。

```
# mount -t ext3 /dev/mmcblk0p2 /mnt
# mount -t devpts devpts /mnt/dev/pts
# mount -t proc proc /mnt/proc
# mount -t sysfs sysfs /mnt/sys
# chroot /mnt /bin/bash
root@bildroot:/# cd
root@buildroot:~# vncserver -geometry 800x600
Passwd: <適当にパスワード入れる>
Retry : <上と同じパスワード入れる>
... 多少エラーメッセージが出て vncserver が立ち上がる気にしない...
root@buildroot:~#
```

7. PC 側にて gvncviewer を起動すると、 Nook Color 上で Debian にログインした状態になる。

```
$ vncviewer 192.168.2.2:1
```

8. lxterminal を vncviewer 上で開き、適当に/etc/resolv.conf し、 route add default gw 192.168.2.10 等して、PC 側を NAT 設定にすると、 Nook Color は USB 越しで PC 側の持つグローバル側ネットワークを利用できるようになります。 aptitude などもも全く問題なく出来ます。

29.8 電源の切り方

いろいろ試したら、電源を OFF にしたくなるかと思います。この場合の手続きは以下の通りです。

1. Nook Color に ssh ログインします。
2. sync;sync;halt と打ち込みます。
3. Nook Color との ssh が切れます。
4. Nook Color から miniSD を抜きます。
5. Nook Color の電源ボタンを 15 秒以上押します。
6. もう一度 Nook Color の電源ボタンを 15 秒以上押すと、 Nook Color 本体がブートします。あとはいつもの Nook Color ですので、そのまま電源ボタンを軽く押すなり放置するなりするといつも通りの電源 OFF (Sleep?) 状態になります。

29.9 終わりに

今回手動で chroot することにより、 Nook Color で Debian を動かしてみました。これで電池動作で小型軽量ということもあり、 Debian を入れたノートパソコンと組み合わせると、 ARM CPU のタブレット開発環境を持ち歩く事ができます。実際、本記事を書くにあたり、平日は通勤電車の中を主に活用して試行錯誤していました。

今回 Nook Color が持っている LCD/タッチパネル/Audio/Wi-Fi について一切 Debian 側から制御までは到達できませんでした。 100 % Debian で動く安価で軽量なタブレット端末の開発の道は険しそうです。

次は OMAP3 用の CPU 向けに kernel のクロスコンパイルに挑戦してみようかと思うこの頃でした。

30 Python 初心者が「 Python プロフェッショナルプログラミング」を読んでみた

やまねひでき

わたくし、「 Python プロフェッショナルプログラミング」という書籍を先日購入しました。私自身はまだ 3 月末ぐらいに Python の入門から始めたばかりで本当は Python ビギナーズプログラミングが欲しいのですが、まだ世の中にはそのような本はありませんし、この書籍自体は評判が良かったので…。

で、実際読んでみて中々実践的というか環境構築などで参考になることが多々ありました。しかし、もうちょっと突っ込んでみたいなぁ、というところがいくつかあったので、 Debian パッケージ方面からの見方として取り上げてみようと思います。この書籍自体では Ubuntu11.10 を使っていますが、私の Debian unstable のやり方でも大体のところはそのまま応用が効くはずです。

30.1 なるべくディストリビューションのパッケージを使いたい

Python には Python パッケージがあり、配布サイト PyPI があります。 Perl に CPAN、 Ruby に gems、 R に CRAN みたいなものですね。で、この Python パッケージを導入するのに pip というツールを使う旨説明があるのですが (P.4)、 github から wget して、インストールスクリプトを sudo で実行しています。`/usr/local` 以下にツールが入るわけです。書籍全体でこの pip を使ったインストールを勧めていますが、これは美しくない。 Debian パッケージに python-pip があるので、そちらを入れてもいいんじゃないかと思います[*79]。それからユーザーの HOME 以下にパッケージをインストールする virtualenv というツールがあります。これも python-virtualenv パッケージを入れて対応します。この後は、 virtualenv でユーザーの Python 環境に pip でパッケージを入れていってもいいかな、と思います。便利ツールとして紹介されている virtualenvwrapper も virtualenvwrapper パッケージがありますのでそれを入れちゃいましょう。

```
$ sudo apt-get install python-pip python-virtualenv virtualenvwrapper
```

この後でユーザー個人の Python 仮想環境下に pip で Python パッケージをガシガシ入れていくのはありだと思います。

30.2 複数バージョンの Python を使う、のはいいけど。

Python2.5 をインストールするのに公式のパッケージリポジトリからだと 11.10 に入れられないよ、ということが書いてあります (P.11)。ここで PPA からインストールするか、ソースからインストールするかという選択になっています。業務で使う場合、その PPA がどれほど信用できるかを説明するのが難しいように思います。自己責任で、って書いてありますが、大抵盲目的にその PPA 信用しちゃうような…。古いバージョンを使わなきゃダメ、という場合、私だったら

[*79] pip は若干バージョンが古いですが、そんなに不都合もないかなーと。古いのが嫌ならローカルにアップデートパッケージ作っちゃうのが吉。

「 debootstrap で古い環境を作る」というやり方でやります(debootstrap については後ほど述べます）。

それからソースから Python2.5 を入れる説明ですが、

```
$ wget http://www.python.org/ftp/python/2.5.6/Python-2.5.6.tgz
$ tar -xvzf Python-2.5.6.tgz
$ cd Python-2.5.6
$ LDFLAGS="-L/usr/lib/x86_64-linux-gnu" ./configure
$ make
$ sudo make install
(「Python プロフェッショナルプログラミング」より引用)
```

という説明になっています。 i386 だとパス変わるよね…というのは読み取れるのか若干心配な所です。プロフェッショナルだからいいのかな? 必要な依存パッケージについては、最初の方のページで

```
$ sudo aptitude -y install build-essential
$ sudo aptitude -y install libsqlite3-dev
$ sudo aptitude -y install libreadline6-dev
$ sudo aptitude -y install libgdbm-dev
$ sudo aptitude -y install zlib1g-dev
$ sudo aptitude -y install libbz2-dev
$ sudo aptitude -y install sqlite3
$ sudo aptitude -y install tk-dev
$ sudo aptitude -y install zip
(「Python プロフェッショナルプログラミング」より引用)
```

となっていました。あぁ、これはイケてない。パッケージビルド用のパッケージ取得は apt-get build-dep するべきです。 Squeeze ではまだ 2.5 のバイナリパッケージが手に入るので、ここでは 2.4 をターゲットに作業してみましょう。こんな風に[*80]。

```
$ sudo apt-get build-dep python2.5
```

これで Python をビルドするときに必要な依存パッケージはすべてインストールされます。あ、 apt line に deb-src ラインを追加を忘れずに。このやり方なら、 Python 以外のソフトウェアパッケージでもバージョンが変わっても大体応用が効きます。

そして「 ソースからそのまま入れると/usr/local 以下に入るから、 python とだけ打つと PATH の優先度でソースから入れた Python2.5 が起動する」…という説明が…そんな罠作らない方がいいじゃないですかー。私ならソースから入れるのなら「 Python2.5 の Debian パッケージを利用環境用にリビルドしていれる」をやります。 PTS の Python2.4 のページ [*81] からソースパッケージの dsc ファイルが取得できるので、これを devscripts パッケージの dget コマンドで取得します。

```
$ sudo apt-get install devscripts
$ dget http://cdn.debian.net/debian//python2.4_2.4.6-1+lenny1.dsc
dget: retrieving http://cdn.debian.net/debian//python2.4_2.4.6-1+lenny1.dsc
--2012-05-04 03:59:21--  http://cdn.debian.net/debian//python2.4_2.4.6-1+lenny1.dsc
Resolving cdn.debian.net... 150.65.7.130
Connecting to cdn.debian.net|150.65.7.130|:80... connected.
HTTP request sent, awaiting response... 404 Not Found
2012-05-04 03:59:22 ERROR 404: Not Found.

dget: wget python2.4_2.4.6-1+lenny1.dsc http://cdn.debian.net/debian//python2.4_2.4.6-1+lenny1.dsc failed
```

あれ、 oldstable だとリンクが切れちゃいますか[*82]。しょうがないので、 archive.debian.org に切り替えてみます。

```
$ dget http://archive.debian.org/debian/pool/main/p/python2.4/python2.4_2.4.6-1+lenny1.dsc
```

これで Python2.4 のソースパッケージが取得できました。ビルド用の依存パッケージは既に入れているので、ソースパッケージの changelog に一言書いてビルドすれば独自リビジョンをつけた deb パッケージが出来上がりますので、dpkg で入れちゃえば良いでしょう(後で述べますが apt を使ってインストールも可能です）。

[*80] ここのパッケージ名指定は、本当は python2.4 にしたいけど無いので 2.5 にしています。多少の差は後で修正。

[*81] http://packages.qa.debian.org/p/python2.4.html

[*82] これはバグ報告しましょう

```
$ dpkg-source -x python2.4_2.4.6-1+lenny1.dsc
$ cd python2.4-2.4.6/
$ export DEBFULLNAME="Hideki Yamane"
$ export DEBEMAIL="henrich@debian.org"
$ dch --bpo "rebuild package for my own environment"
$ debuild -us -uc
dpkg-checkbuilddeps: Unmet build dependencies: libreadline5-dev tk8.4-dev libdb4.5-dev emacs22
dpkg-buildpackage: warning: Build dependencies/conflicts unsatisfied; aborting.
```

あれ、依存関係が満たせない。 python2.5 と比較してちゃちゃっと変更しましょうか。 apt-get source python2.5 として、 diff 取ります。

```
--- python2.4-2.4.6/debian/control        2012-05-04 04:09:00.000000000 +0000
+++ python2.5-2.5.5/debian/control        2012-05-04 04:13:06.000000000 +0000
@@ -1,82 +1,96 @@
-Source: python2.4
+Source: python2.5
 Section: python
 Priority: optional
 Maintainer: Matthias Klose <doko@debian.org>
-Build-Depends: debhelper (>= 5), autoconf, libreadline5-dev, libncursesw5-dev (>= 5.3), tk8.4-dev, libdb4.5-dev, zlib1g-dev,
libgdbm-dev, blt-dev (>= 2.4z), libssl-dev, sharutils, libbz2-dev, libbluetooth-dev [!hurd-i386 !kfreebsdi386
!kfreebsd-amd64], locales, mime-support, libgpm2 [!hurd-i386 !kfreebsd-i386 !kfreebsd-amd64], netbase, lsb-release, bzip2
-Build-Depends-Indep: libhtml-tree-perl, texlive-latex-recommended, texinfo, emacs22, debiandoc-sgml
-Build-Conflicts: tcl8.3-dev, tk8.3-dev, python2.4-xml, python-xml
-XS-Python-Version: 2.4
-Standards-Version: 3.8.0
+Build-Depends: debhelper (>= 5), autoconf, libreadline-dev, libncursesw5-dev (>= 5.3), tk8.5-dev, libdb4.8-dev, zlib1g-dev,
libgdbm-dev, blt-dev (>= 2.4z), libssl-dev, libbz2-dev, libbluetooth-dev [!hurd-i386 !kfreebsd-i386 !kfreebsd-amd64],
locales [!avr32 !m68k], libsqlite3-dev, libffi-dev (>= 3.0.5-2), mime-support, libgpm2 [!hurd-i386 !kfreebsd-i386
 !kfreebsd-amd64], netbase, lsb-release, bzip2, netbase, sharutils
+Build-Depends-Indep: libhtml-tree-perl, texlive-latex-recommended, texinfo, emacs23, debiandoc-sgml, latex2html
+Build-Conflicts: tcl8.4-dev, tk8.4-dev, tcl8.3-dev, tk8.3-dev, python2.5-xml, python-xml
+XS-Python-Version: 2.5
+Standards-Version: 3.9.1
```

debian/changelog ファイルはこんな感じにしておきましょう。

```
python2.4 (2.4.6-1+lenny1~bpo60+0.1) squeeze-backports; urgency=low

  * Rebuild for squeeze-backports.
    - debian/control
     + set Build-Depends: libreadline-dev, tk8.5-dev, libdb4.8-dev
     * set Build-Depends-Indep: emacs23
  * rebuild package for my own environment

 -- Hideki Yamane <henrich@debian.org>  Fri, 04 May 2012 04:16:39 +0000
```

```
$ debuild -us -uc
$ ls ../*.deb
../idle-python2.4_2.4.6-1+lenny1~bpo60+0.1_all.deb
../python2.4-dev_2.4.6-1+lenny1~bpo60+0.1_amd64.deb
../python2.4-minimal_2.4.6-1+lenny1~bpo60+0.1_amd64.deb
../python2.4-dbg_2.4.6-1+lenny1~bpo60+0.1_amd64.deb
../python2.4-examples_2.4.6-1+lenny1~bpo60+0.1_all.deb
../python2.4_2.4.6-1+lenny1~bpo60+0.1_amd64.deb
$ sudo dpkg -i ../python2.4_2.4.6-1+lenny1~bpo60+0.1_amd64.deb  ../python2.4-minimal_2.4.6-1+lenny1~bpo60+0.1_amd64.deb
```

これで python2.4 と打ったときだけ起動するようになります。…結構面倒臭いですね。 chroot するのが手っ取り早そう…。

30.3 ソースからビルドする時、 again

「 ソースコードからビルドする」 (P.390) では、 Python Imaging Library をソースからビルドする場合について触れられています。これも python-imaging という名前の Debian パッケージは提供されていますが、ソースから入れたいときもあるのでしょう。で、書籍ではビルドする前に

```
$ sudo aptitude install python-dev build-essential
$ sudo aptitude install libjpeg62-dev libfreetype6-dev zlib1g-dev liblcms1-dev
(「Python プロフェッショナルプログラミング」より引用)
```

としています。既にパッケージがあるのなら、ここはもう分かりますね? apt-get build-dep です。

```
$ sudo apt-get build-dep python-imaging
```

これで漏れなく python-imaging をビルドするときに必要なパッケージが一揃いインストールされますし、ディストリビューションのバージョンが変わってビルド用のパッケージ名が変わったとしても必要となる依存パッケージは変わりなくインストールされます。

30.4 さらにさらにソースからビルドする時

出来れば pbuilder でローカルの環境をなるべくクリーンに保ってビルドする方が良いですね[*83]。

```
$ sudo apt-get install devscripts
$ dget http://archive.debian.org/debian/pool/main/p/python2.4/python2.4_2.4.6-1+lenny1.dsc
$ sudo apt-get install pbuilder
$ sudo pbuilder --create
$ sudo pbuilder --build python2.4_2.4.6-1+lenny1.dsc
```

30.5 ローカルの deb パッケージのインストールについて

「閉じた環境のインストール」(P.259) というので、Debian パッケージを aptitude を使ってダウンロードしたり、パッケージキャッシュからパッケージを取得したりということが書いてあります。しかし、この書籍では「`dpkg -i *`」としてパッケージをインストールしようという説明が。しかもパッケージが足りないとエラーになるけど、地道にやろうなどと書いてあります。ダメのダメダメです。私なら以下のようにします。

1. apt-utils パッケージをインストールする「apt-get install apt-utils」
2. deb パッケージを集めたディレクトリ（ここでは `/home/username/packages` としましょうか）で「`apt-ftparchive package . | gzip -c9 > Packages.gz`」として `Packages.gz` ファイルを生成します。このファイルにはパッケージ情報がリストアップされています。
3. `/etc/apt/sources.list` ファイルを編集して、パッケージの入手ソースとして先ほどのパッケージを集めたディレクトリを追加します。「deb file:///home/username/packages/ ./」などとしておけば良いでしょう。
4. 「apt-get update」としてパッケージデータベースを更新します。
5. 必要なパッケージを apt-get install でインストールしましょう。

```
$ sudo apt-get install apt-utils
$ cd ~/packages
$ apt-ftparchive package . | gzip -c9 > Packages.gz
$ sudo sh -c "echo deb file:///home/username/packages/ ./ > /etc/apt/sources.list"
$ sudo apt-get update
$ sudo apt-get install <package name>
```

これで１個１個依存関係を確認しつつ地道にやるなどということからはおさらばですし、普段やりなれているパッケージインストール方法ともシームレスです。パッケージが増えたら `apt-ftparchive` しなおして `apt-get update` で。

30.6 パッケージ一覧の取得

「必要なパッケージを列挙する」(P.256) では aptitude でインストールしたパッケージの一覧は dpkg -l でリストを作って活用するとあります。いちいちパッケージバージョンを記載する必要があるなら別ですが、こんなやり方はどうでしょう?

```
$ dpkg --get-selections > package-list.txt
```

これでインストールしてあるパッケージの一覧が取得できます。そしてさらに、この一覧を dpkg に食わせてインストールも可能です。

```
$ sudo dpkg --set-selections < package-list.txt
$ sudo apt-get dselect-upgrade
```

[*83] あ、Ubuntu の場合は Launchpad にアカウント作って独自 PPA 運用の方がいいのかも?

これで別のマシンでも同じパッケージを放り込むことが出来ます。

30.7 検証環境を用意する

「動作を検証する」 (P.264) では、検証環境として VirtualBox をインストールすることが触れられています。それもいいけど、 Debian なら debootstrap があるじゃないですか。 debootstrap はローカルに指定したバージョンのまっさらな最小限の Debian 環境を作ることが出来るツールです。適宜 chroot して使うことになります。

```
$ sudo apt-get install debootstrap
$ sudo mkdir -p /srv/chroot/{lenny-i386,squeeze-amd64}
$ sudo debootstrap --arch i386 lenny /srv/chroot/lenny-i386 http://archive.debian.org/debian
$ sudo debootstrap --arch amd64 squeeze /srv/chroot/squeeze-amd64 http://ftp.jp.debian.org/debian
```

–arch でアーキテクチャを、その次にバージョンのコードネームを、 chroot 環境を作るディレクトリ、取得先のサーバーと指定すれば OK です。ここでは i386 環境の Debian5.0 と amd64 環境の Debian6.0 環境を作ります。ミソとしては、 Debian5.0 は既にミラーサーバーからは移動されているので古いバージョンの集積場所である archive.debian.org を指定しなければならないのを知っておくことです。

あとは chroot します。実際に構築している環境で先に述べたパッケージ一覧を取得しておき、適当なパッケージ一覧を食わせてやってインストールすれば楽でしょう。あぁ、アーカイブされていないバージョンだと apt-line に security.debian.org を追加しておくのも忘れないように。

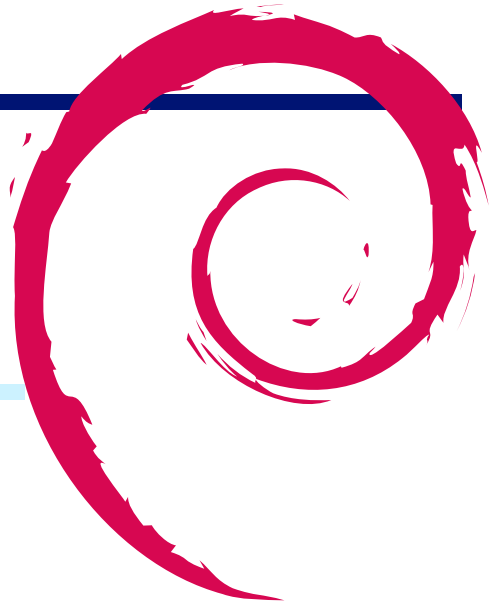
```
$ cp package-list.txt /srv/chroot/lenny-i386/tmp/
$ LANG=C sudo chroot /srv/chroot/lenny-i386 /bin/bash
# dpkg --set-selections < /tmp/package-list.txt
# apt-get dselect-upgrade
```

```
$ cp package-list.txt /srv/chroot/squeeze-amd64/tmp/
$ LANG=C sudo chroot /srv/chroot/squeeze-amd64 /bin/bash
# echo "deb http://security.debian.org/ squeeze/updates main contrib non-free" >> /etc/apt/sources.list
# apt-get update
# dpkg --set-selections < /tmp/package-list.txt
# apt-get dselect-upgrade
```

まぁ、仮想環境でやるのもいいですが、こういうツールを使うやり方も取り上げてほしいものだなぁ、と思います。

30.8 最後に

いかがでしたでしょうか。異論もあろうとは思いますが、こんな見方もできるんだよ、ということで。それよりも Python ちゃんと使えるようになれよ＞お前　という気もしますが、それは追々。



31 CoffeeScript を使ってみた

上川純一

31.1 はじめに

CoffeeScript とは JavaScript を使いやすくしたプログラミング言語です。 CoffeeScript は JavaScript に変換されてブラウザなどで実行できます。 Node などを利用してサーバーサイドもクライアントサイドも JavaScript でプログラミングしていると JavaScript の罠だったり、冗長な部分などが目についてきます。それをカバーしてくれるよい言語のようです。入門書 [2] を読んだついでに Debian で CoffeeScript を利用する方法について紹介します。

31.2 Debian で使うには

node のモジュールの中では CoffeeScript はポピュラーです。[*84]node のモジュールとして CoffeeScript を利用するのがおそらくポピュラーな方法だと想像しています。

Debian パッケージとしては関連のものがいくつかあります。[*85]

```
$ apt-cache search coffeescript
coffeescript - interpreter and compiler for the CoffeeScript language
coffeescript-doc - documentation for the CoffeeScript language
libjs-coffeescript - client-side interpreter for the CoffeeScript language
```

npm では `coffee-script` モジュールをインストールすることになります。

```
$ sudo npm install -g coffee-script
```

31.2.1 ブラウザ上での利用

ブラウザ上で利用する場合はどうするのがよいのでしょうか。 `libjs-coffeescript` パッケージとして minify されている CoffeeScript の処理系が配布されているのでこれを利用するのがよいでしょう。

`coffee-script.js` がコードを一つのファイルにまとめたもの、 `coffee-script.min.js` が minify (uglify?) されているものです。

```
/usr/share/javascript/coffeescript/coffee-script.min.js
/usr/share/javascript/coffeescript/coffee-script.js
```

script タグのうち type=text/coffeescript となっているところを処理してくれるようです。 CoffeeScript コンパイラは minify されているとはいえ 169kB ある処理系をダウンロードしないといけないのでサーバサイドで CoffeeScript プログラムをコンパイルして一つの JavaScript ファイルに変換してからクライアントサイドで利用するほうが好ましい気がします。例として HTML はこのようになるようです。

[*84] 2012 年 5 月 7 日調べ、 npmjs.org の依存関係ランキングでは underscore の次に coffee-script。
[*85] 2012 年 5 月 16 日時点では wheezy に入っておらず、 sid のみ。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Hello world</title>
    <script type='application/javascript' src='coffee-script.min.js'></script>

    <script type='text/coffeescript' src='./hello.coffee'></script>

    <script type='text/coffeescript' >
console.log 'This is a pen'
    </script>
  </head>
  <body>
    <h1>Hello world</h1>
  </body>
</html>
```

コマンドラインで CoffeeScript のプログラムを JavaScript にコンパイルするには `coffee -c` を利用します。

```
$ coffee -c class.coffee
```

31.2.2 クライアントアプリケーション上での利用

ローカルマシンで動かす目的の CLI などのアプリケーションの場合はどうなるでしょうか。

CoffeeScript パッケージにはいっている coffee コマンドを利用するとスクリプトをコンパイル・実行できます。CoffeeScript では#がコメントとして扱われるため、 she-bang の指定が可能です。

```
$ cat shebang.coffee
#!/usr/bin/env coffee
util = require 'util'
util.log 'hello world'
$ ./shebang.coffee
8 May 17:08:25 - hello world
```

31.2.3 サーバサイドでの利用

ウェブアプリケーションのサーバ上で CoffeeScript で書いたものを利用するにはどうしたらよいでしょうか。

ウェブサーバとしてうごくプログラムを CLI プログラムとして実行すれば良いので、 CoffeeScript でサーバを書いてしまえばそれでよいです。

また、 CoffeeScript で書いたコードを JavaScript に変換してからクライアント側のブラウザに送るという仕組みもあると思います。

例を書いてみましょう。

```
cli = require 'cli'

cli.parse {
    port: ['p', 'port number to listen to', 'number', 8088]
}

http = require 'http'

cli.main (args, options) ->
  http.createServer((req, res) ->
    res.writeHead 200, {'Content-Type': 'text/plain'}
    res.end 'Hello world\n').listen options.port
  console.log 'Server running at http://localhost:' + options.port + '/'
```

31.3 ソースコードの編集環境

emacs の場合は CoffeeScript 用のモードがあるようです [3]。 Debian Package にはまだなってない気がしています。

31.4 おわりに

Debian での CoffeeScript の開発環境について紹介しました。 Debian で Node に挑戦したい、でも Node って JavaScript で書かないといけないんでしょ、と尻込みしているあなたにお勧め、かもしれません。

参考文献

[1] CoffeeScript のソースコードと解説 /usr/share/doc/coffeescript/html/

[2] Smooth CoffeeScript http://autotelicum.github.com/Smooth-CoffeeScript/

[3] Emacs Major mode for CoffeeScript https://github.com/defunkt/coffee-mode

32 Dynamic Kernel Module Support Framework

岩松 信洋

32.1 Dynamic Kernel Module Support Framework とは

Dynamic Kernel Module Support Framework（ 以下、 DKMS）は、マシンにインストールされているカーネル毎に自動的にドライバモジュールをビルドし管理する仕組みです。

通常、 Linux カーネルが更新されたとき、 Linux カーネルで提供されていないサードパーティのドライバはその Linux カーネル用にデバイスドライバを再ビルドおよびインストールを行う必要があります。しかし DKMS を使ってサードパーティのデバイスドライバを管理している場合、新しい Linux カーネルで起動された時に対応したデバイスドライバがないことをチェックし、ドライバモジュールをビルドして（ 自動インストールが設定されていれば）インストールします。もちろんそのドライバは Linux カーネルに対応している必要がありますが、ドライバがメンテナンスされていれば、半自動的に新しいカーネルへ移行できるようになります。

DKMS は DELL `http://linux.dell.com/dkms/` で開発されており、 Debian や Ubuntu などののディストリビューションで利用できるようになっています。

ちなみに Fedora は Kmods2 `http://rpmfusion.org/Packaging/KernelModules/Kmods2` と呼ばれる別のフレームワークを利用しています。 Arch Linux と Gentoo も DKMS サポートパッケージもあるが、独自の実装が中心のようです。

今回は DKMS の仕組みと DKMS に対応したパッケージの作成方法、 Debian の DKMS 対応状況について説明します。

32.2 DKMS の仕組み

DKMS の仕組みを理解するために簡単なデバイスドライバを使って説明します。ドライバをロードするとカーネルデバッグメッセージに「 Hello, world! from hello_init」、ドライバをアンロードすると「 Good bye! from hello_exit」と出力するものです。以下に実際のソースコード、 Makefile、実行結果とディレクトリ構成を示します。

ドライバソースコード:

```
#include <linux/init.h>
#include <linux/module.h>

static int __init hello_init(void)
{
  pr_info("Hello, world! from %s\n", __func__);
  return 0;
}

static void __exit hello_exit(void)
{
  pr_info("Good bye! from %s\n", __func__);
}

module_init(hello_init);
module_exit(hello_exit);
MODULE_LICENSE("GPL v2");
```

Makefile:

```
obj-m:= src/
```

src/Makefile:

```
obj-m:= hello.o
```

ドライバのコンパイルと実行結果:

```
$ make -C /lib/modules/‘uname -r‘/build M=‘pwd‘ modules
$ sudo insmod hello.ko
$ dmesg | tail -1
[5963787.044134] Hello, world! from hello_init
$ sudo rmmod hello.ko
$ dmesg | tail -1
[5963798.420368] Good bye! from hello_exit
```

ディレクトリ構成:

```
hello-0.0.1
├── Makefile
├── README
├── dkms.conf
└── src
    ├── Makefile
    └── hello.c
```

dkms.conf:

```
PACKAGE_NAME="hello"
PACKAGE_VERSION="0.0.1"
BUILT_MODULE_NAME="$PACKAGE_NAME"
BUILT_MODULE_LOCATION="src"
DEST_MODULE_LOCATION="/updates"
AUTOINSTALL="yes"
```

32.2.1 DKMS 全体の流れ

DKMS の全体の流れは以下のようになっています。各要素は DKMS で提供されているコマンドとスクリプトになっています。

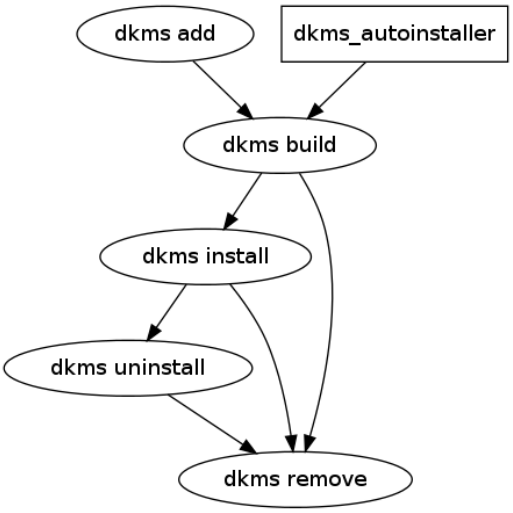


図 34　DKMS 全体の流れ

dkms_autoinstaller はシステム起動時に呼ばれる[*86]スクリプトです。既に DKMS に登録(dkms add) されているドライバモジュールの中で、現在起動しているカーネル用にドライバモジュールがインストール(dkms install) されていないものがあれば、ドライバモジュールのビルドとインストール(dkms build / dkms install) が実行されます。

32.2.2 モジュールソースコードの配置位置

DKMS 対応モジュールのソースコードは /usr/src/ドライバモジュール名-バージョン/ 以下に配置する必要があります。先のドライバの場合、 /usr/src/hello-0.0.1/ 以下に配置します。

32.2.3 DKMS 設定ファイル

DKMS は、各ドライバパッケージ毎に設定ファイルを持つ必要があります。この設定ファイルでは、どのようなドライバが提供、ビルド、インストールされるのか記述されている必要があります。以下によく利用する設定項目を示します。

BUILT_MODULE_LOCATION や BUILT_MODULE_NAME などは配列を使ってドライバモジュール毎に設定できます。 BUILT_MODULE_NAME 配列の番号は各要素の番号に紐づいているので、処理されるときは各々の番号が参

[*86] ディストリビューションによって異なる

表 25　DKMS 設定ファイル項目

設定項目	内容
PACKAGE_NAME	パッケージ名
PACKAGE_VERSION	パッケージバージョン
CLEAN	キャッシュなどの一時ファイルを削除するためのコマンドを指定する。指定しない場合には “make clean ” が実行される。
MAKE	モジュールをビルドするコマンドを指定する。
BUILT_MODULE_NAME	ドライバモジュールの名前を指定する(拡張子は必要なし）。
BUILT_MODULE_LOCATION	ビルドされたモジュールが作成されるディレクトリまでの相対パス
DEST_MODULE_LOCATION	モジュールをインストールするディレクトリを指定する。
AUTOINSTALL	モジュールを自動的にインストールするか、 “yes” か “no” を指定する。
REMAKE_INITRD	ドライバインストール時に initrd イメージを再構成するか、 “yes” か “no” を指定する。
PATCH	適用したいパッチを指定する。
PATCH_MATCH	パッチを適用するバージョンの指定する。 Linux カーネルが 2.6.38 と 2.6.39 の場合にパッチを適用したい場合には “2\.6\.(38|39)” と指定する。

照されます。例えば、一つのドライバパッケージから foo と bar の 2 つのデバイスドライバが提供される場合、以下のように設定することができます。

```
( 省略)
BUILT_MODULE_NAME[0] = "foo"
BUILT_MODULE_LOCATION[0]  = "foo_build"
DEST_MODULE_LOCATION[0] = "/update"
BUILT_MODULE_NAME[1] = "bar"
BUILT_MODULE_LOCATION[1]  = "bar_build"
DEST_MODULE_LOCATION[1] = "/update"
( 省略)
```

これにより、ドライバ foo のビルドは foo_build ディレクトリで行われ、 update ディレクトリ以下にインストール、ドライバ bar のビルドは bar_build ディレクトリで行われ、 update ディレクトリ以下にインストールされます。

32.2.4　DKMS で提供されるコマンド

DKMS は dkms コマンドで操作します。以下に利用頻度が高いコマンドを紹介します。

1. dkms status

   DKMS で提供されている モジュールのステータスを表示します。

   ```
   $ dkms status
   v4l2loopback, 0.5.0, 3.1.0-1-amd64, x86_64: installed
   v4l2loopback, 0.5.0, 3.2.0-1-amd64, x86_64: installed
   virtualbox, 4.1.8, 3.1.0-1-amd64, x86_64: installed
   virtualbox, 4.1.8, 3.2.0-1-amd64, x86_64: installed
   ```

2. dkms add

   モジュールのソースコードを管理対象に追加します。実行する時に -m オプションで追加するドライバパッケージ名を、-v オプションでバージョンを指定します。

   ```
   $ sudo dkms add -m hello -v 0.0.1

   Creating symlink /var/lib/dkms/hello/0.0.1/source ->
                    /usr/src/hello-0.0.1

   DKMS: add completed.
   $ dkms status
   hello, 0.0.1: added
   ```

実行すると/var/lib/dkms 以下に シンボリックリンクを張り、 DKMS が管理するドライバデータベースに登録されます。

3. dkms build

   管理対象になっているモジュールをビルドします。実行する時に -m オプションで追加するドライバパッケージ名を、-v オプションでバージョンを指定します。

   ドライバは /var/lib/dkms/ドライバ名/ドライババージョン/BUILT_MODULE_LOCATION 以下でビルドされます。作成されたモジュールとビルドログは /var/lib/dkms/ドライバ名/ドライババージョン/カーネルバージョン/アーキテクチャ 以下に置かれます。

```
$ sudo dkms build -m hello -v 0.0.1

Kernel preparation unnecessary for this kernel.  Skipping...

Building module:
cleaning build area....
make KERNELRELEASE=3.0.0-1-amd64 -C /lib/modules/3.0.0-1-amd64/build M=/var/lib/dkms/hello/0.0.1/build....
cleaning build area....
```

4. dkms install

   dkms build コマンドによってビルドされたモジュールを dkms.conf の DEST_MODULE_LOCATION で指定されている場所にインストールします。例えば Linux カーネル 3.2.0-1-amd64 を使っていて、

```
DEST_MODULE_LOCATION[0]="/updates"
```

   と指定されている場合には /lib/modules/3.2.0-1-amd64/updates/dkms/ 以下にインストールされます。

```
$ sudo dkms install -m hello -v 0.0.1

hello:
Running module version sanity check.
 - Original module
   - No original module exists within this kernel
 - Installation
   - Installing to /lib/modules/3.0.0-1-amd64/updates/dkms/

depmod.........

DKMS: install completed.
$ dkms status
hello, 0.0.1: installed
$ sudo modprobe -l | grep hello.ko
updates/dkms/hello.ko
```

5. dkms uninstall

   モジュールをアンインストールします。実行する時に -m オプションで追加するドライバパッケージ名を、-v オプションでバージョンを指定します。また全てのカーネルから削除する場合には –all オプション、特定のカーネルモジュールのみを削除する場合には -k オプションでカーネルバージョンを指定します。

```
$ sudo dkms uninstall -m hello -v 0.0.1 -k 3.0.0-1-amd64

-------- Uninstall Beginning --------
Module:  hello
Version: 0.0.1
Kernel:  3.0.0-1-amd64 (x86_64)
-------------------------------------
( 中略)

depmod....

DKMS: uninstall completed.
```

   *87

6. dkms remove

   DKMS の管理対象から外します。 uninstall と同様のオプションが必要です。ドライバモジュールがアンインストールされていない場合、アンインストールを実行してから、管理対象から外れるようになっています。

*87 この資料を作成している時点では、「 dkms uninstall」は動作しません。一部のチェックが未実装なため、 uninstall の処理まで行われないためです。 http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=659672

```
$ dkms status
hello, 0.0.1, 3.0.0-1-amd64, x86_64: installed
virtualbox, 4.1.6, 3.0.0-1-amd64, x86_64: installed
$ sudo dkms remove  -m hello -v 0.0.1 --all

-------- Uninstall Beginning --------
Module:  hello
Version: 0.0.1
Kernel:  3.0.0-1-amd64 (x86_64)
-------------------------------------
( 中略)

depmod....

DKMS: uninstall completed.

------------------------------
Deleting module version: 0.0.1
completely from the DKMS tree.
------------------------------
Done.
$ dkms status
```

その他、 rpm パッケージを作成する mkrpm オプションや Debian パッケージを作成する mkdeb オプションなどが用意されています。

32.3 DKMS 対応 Debian パッケージの作り方

上記での説明のように、 DKMS 対応モジュールのソースを規定のディレクトリに展開しておくと DKMS は処理を行います。 Debian の場合もパッケージ化するときはインストールした時に規定のディレクトリに展開するようにしておきます。パッケージ名などに関してはルールが決まっており、 DKMS 対応パッケージは-dkms というサフィックスがついています。これはパッケージ名がわかりやすいという理由と、 DKMS 用の debhelper コマンド、 dh_dkms コマンドが -dkms サフィックスのついたパッケージに対して処理を行うためです。 debian/パッケージ名.dkms または debian/dkms を dkms.conf に変換して、適切な dkms 用ディレクトリにコピーし、 postinst と prerm に DKMS 用の処理を追加します。 dh_dkms コマンドは dkms パッケージで提供されています。 debhelper 7 以降で利用する場合には、アドオンとして読み込ませる必要があります。

debian/hello-dkms.dkms:

```
PACKAGE_NAME="hello"
PACKAGE_VERSION="#MODULE_VERSION#"
BUILT_MODULE_NAME[0]="$PACKAGE_NAME"
BUILT_MODULE_LOCATION="src"
DEST_MODULE_LOCATION[0]="/updates"
AUTOINSTALL="yes"
```

DKMS 用 debian/rules:

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

VERSION := $(shell dpkg-parsechangelog | sed -nr '/^Version:/s/Version: (.*:)?(.*)-(.*)/\2/p')

%:
    dh $@ --with dkms
override_dh_install:
    dh_install src/* usr/src/hello-$(VERSION)/src/
    dh_install Makefile usr/src/hello-$(VERSION)/
override_dh_dkms:
    dh_dkms -V $(VERSION)

override_dh_auto_configure override_dh_auto_build override_dh_auto_test override_dh_auto_install override_dh_auto_clean:
```

debian/rules ファイル内で override_dh_auto_configure や override_dh_auto_build を呼び出している理由は Makefile がある場合、 make が実行されてしまうのでこれを抑制するためです。

32.3.1 Debian でのドライバモジュールビルドのタイミング

Debian では DKMS ドライバパッケージがインストールされた後と、カーネルヘッダ または カーネルイメージパッケージがインストール(更新) された後、 dkms が起動してドライバモジュールのコンパイルが行われるようになっています。これらは以下のファイルで処理されます。

```
/etc/kernel/postinst.d/dkms
/etc/kernel/header_postinst.d/dkms
```

32.3.2 Debian の DKMS 対応状況

主要なドライバの殆どは DKMS と module-assistant（ 以下 m-a）の両方に対応しています。開発があまり活発ではないドライバは m-a のみをサポートした状態が多いようです。新しくパッケージ化されたドライバはほとんどが両方に対応しています。 DKMS のほうがメリットが多いため、 m-a から DKMS に切り替えているユーザも多いようです。

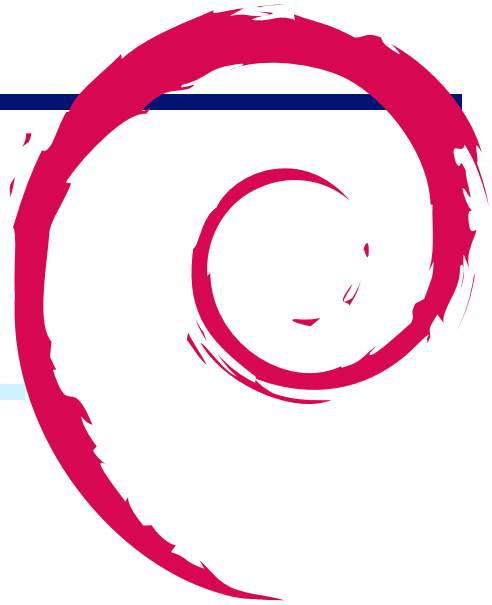
32.4 module-assistant との違い

Debian には他のドライバパッケージ管理機構に module-assistant（ m-a）があります。 DKMS と m-a の違いには以下のようになっています。

- m-a は自動ビルド機構がない。
  カーネルが更新されると、手動で m-a を実行する必要があります。
- DKMS はドライバ用のバイナリパッケージを作らない。
  m-a は カーネル毎バイナリパッケージを作り、それをインストールします。
- m-a は起動しているカーネルのみのドライバをビルドするが、 DKMS はインストールされているカーネルをビルドできる。

32.5 まとめ

今回は DKMS の基本的な仕組みと、 Debian での DKMS 対応パッケージ作成方法について説明しました。今までは m-a が主流だったのですが DKMS もサポートしているドライバパッケージも増え始め、徐々に DKMS に移行しつつあるように感じました。 Debian パッケージ用ツールが用意されているため、対応パッケージ作成も難しくないと思います。今後ドライバパッケージを作成する場合には、 DKMS をサポートしてみてはいかがでしょうか。



33 さきが(ry NM 塾

倉敷悟

33.1 NM とは何か

Debian における NM (New Member) とは、新しく Debian Project の参加メンバーになった人、あるいは Debian Project に参加するための手続き、といった意味で使われます。後者の場合は、意味を明確にするために、NM プロセスといったりもします。少し前までは、New Maintainer の略だったのですが、(パッケージ) メンテナ以外にも門戸を開こう、という方向性に変更されつつあります (とはいえ、現時点では NM プロセスはまだパッケージメンテナを念頭においた形のままです)。

33.2 NM プロセスの紹介

では、実際の NM プロセスを実例をもとにご紹介します。本当はもう少しプロセスを進めておくつもりだったのですが、あまりスムーズにいってないので途中までになっています。

NM 志願者向けのチェックリスト (http://www.debian.org/devel/join/nm-checklist) がありますので、まずはこれを見てみましょう。

33.2.1 必要な条件

まず、Debian Developer（以下 DD） 2 名以上とキーサインしていること、さらに NM への応募を推薦してくれる DD が 1 名以上いること、これは具体的かつ最低限の条件になります。加えて、すでに Debian に関わるある程度の活動をしてきていることも必要とされますが、これについては明確な線引きがあるわけではありませんし、今後変化してくる部分なので水物といっていいでしょう。

私の例だと、まともに Debian での活動をはじめたのは 2007 年くらいだったと思うので、おおよそ 4 年間、下積みとしてパッケージメンテ、翻訳、ローカルコミュニティ活動をしてきていますが、これはちょっと長かった (NM 応募が遅かった) かなぁ、と思っています。

最近では、DM のステップをふんでいることが推奨されていることもあるので、少なくとも 6 ヶ月以上特定のパッケージをメンテしていて、スポンサーと良好な関係を築けていれば、そのまま NM に進むことも可能でしょう。

33.2.2 実例

- 2011/11 (事前ネゴ)
- 2011/12/04 apply
- 2011/12/04 advocate checked by yyabuki
- 2011/12/10 activity poll sent
- 2011/12/11 pass frontdesk precheck

- 2011/12/13 AM assigned to gwolf
- 2011/12/16 AM assigned to laney
- 2011/12/17 ID checked
- 2011/12/25 (Philosophy and Procedure やりとり中)

この後に続く予定は、次のようになっています。

- Tasks and Skills
- AM recommends to DAM
- DAM Approval

33.2.3 NM テンプレート

NM 担当チームでは、主に AM 担当者向けの資料として、作業ガイドのリポジトリが用意しています (`http://anonscm.debian.org/viewvc/nm/trunk/nm-templates/`)。実は、ここを見れば、だいたい何を聞かれるのかはわかってしまいますので、予習のための参考書としては最良のものだと思われます。ただし、 AM によっては NM テンプレートを使わない、という人もいますので、盲信はしすぎないようにしましょう。

33.2.4 NM と 勉強会と Debian JP

Debian 勉強会では、 Debian Developer を育成する、を目的としてあげています。ですので、その運営主体である Debian JP としても、支援活動はいろいろとおこなわれています。具体的には、 GPG キーサインの推進や、スポンサー探しのサポート、パッケージングスキルの教育、などです。是非有効に活用してください。

33.2.5 NM と Debian Maintainer

勉強会でも何度かとりあげていますが、いきなり NM は敷居が高い、という人のために、 DM というステップが用意されています。これは、いわば「限定つきの Debian Developer」であり、スポンサーについてもらうことで自分のパッケージを Debian に含めることができる人達のことです。

事前の実績としてわかりやすいことや、内容がサブセットになっていることもあり、 NM プロセスにおいても、事前に DM として活動しておくことが強く推奨されています。

33.3 最後に

33.3.1 なぜ NM に?

NM になることの意義を考えてみましょう。直接的には、次のようなメリットがあります。

- パッケージを自由にアップロードできる
- LWN.net の購読権 (sponsored by HP)
- Project Leader や GR への投票
- @debian.org メールアカウント

あるいは若い人であれば、比較的名前の知られたプロジェクトへの参加自体や、プロジェクトに自分色を持ち込める、といった満足もあるかもしれません。ただ、 NM プロセスやその後の活動は、それなりに手間も気力も必要になります。これじゃ引きが足りない、という人も多いでしょう。

Debian 勉強会としては、前述したとおり、 NM に応募しようと思える人材を育成したい、という思いがあるのですが、やはり動機の部分については、皆さんそれぞれの答えを見出していただくしかありません。

スライドの方では、一例として、特に技術的に秀でたものをもっているわけでもなく、業界的には定年に達してしまったおっさんが、一体何を考えて NM に挑戦してみているのか、簡単に紹介してみます。参考になれば。

34 フリーソフトウェアと戯れるための著作権入門

山城国の住人 久保博

34.1 前書き

人生のいきがかり上、割と最近になって著作権の勉強を始めることになりました。法律の専門家ではありませんので、自信を持っての発表ではないですが、みなさんと一緒に考えていくきっかけになればと思います。

34.2 はじめに

日本を含む多くの国で、ソフトウェアは、生み出された時から著作権という権利の対象になっています。

ですから、フリーソフトウェアも、生まれた時から誰かのもの、ということになっています。フリーであっても誰かのものである、という点では、登記された土地は空き地であっても誰かのものであるのとにているかも知れません。

フリーソフトウェアの精神は、一介の利用者が自分自身のためにソフトウェアを使うことを妨げるようなことは最大限避けるようにしていますから、自分自身のためにフリーソフトウェアを使う限り、知らなくても困ったことにはなりません。

でも、そこから一歩踏み出そうとすれば、誰かのものであるということを尊重することが求められることになります。この要請は、著作権法と、それを前提とする契約に基づいています。

そこで、 Debian Project が配布しているようなフリーソフトウェアを扱う場面をいくつか想定して、そこに著作権と契約がどのように関わってくるのか、解き明かしてみます。なお、日本の著作権法を元にしてお話します。

34.3 知的財産権

ソフトウェアが誰かのものである、とは、どういうことでしょう。

知的な作業を通して創り出された、無体物（ プログラムを含む）には、創り出した人の努力や労力に報いるために、さまざまな制度を定める法律が整備されています。その中でも、知的な創造によってつくり出された無体物に対して、財産権の対象となる有体物の物権に良く似た権利を設定する仕組みが法律で定められています。この物権に似た権利を知的財産権といいます。そして、知的財産権は、創造した人あるいは登録した人が享受する権利なのです。

これが、ソフトウェアが誰かのものである、ということの、現代社会における意味です。

ソフトウェアが関係する知的財産権には、次のものがあります。

著作権「 思想又は感情を創作的に表現したものであつて、文芸、学術、美術又は音楽の範囲に属するもの」に対する独占的な保護をもたらす権利

特許権 発明（ 自然法則を利用した技術的思想の創作のうち、高度なもの）を登録することによって独占的な保護をもたらす権利

実用新案権 考案（ 自然法則を利用した技術的思想の創作）を登録することによって独占的な保護をもたらす権利

意匠権 意匠（ 物品の形状、模様若しくは色彩またはこれらの結合であって、視覚を通じて美感を起こさせるもの）を登録することによって独占的な保護をもたらす権利

商標権 商標を登録することによって独占的な保護をもたらす権利

プログラムそのものは、著作権の対象になります。また、プログラムで実装されたアルゴリズムは特許や実用新案の保護の対象になり得る場合がありますし、プログラムの名称やロゴは商標としての保護の対象になり得ます[*88]。 GUI の画面のデザインは海外では意匠登録の対象となり得る場合もあるようですが、日本では今のところ対象ではありません。

この中でも、ソフトウェアについて一番よく問題になるのは、著作権です。

34.4 著作権

34.4.1 著作権の性質

正確ではないですが端的に言えば、著作権法は、著作物を作った人に著作物に対する独占的な権利を与える法律です。その著作権法で定められている著作権には次のような性質があります。

- 著作物を作ったら、発生します。
- 著作物を公表しなくても発生します。
- （ 特許と違って、）役所に登録しなくても発生します[*89]。
- 著作物が生まれてから、著作者の死後 50 年間、著作権の保護は続きます。
- 他人が著作物を勝手に利用した場合、差止請求権、損害賠償請求権を行使するという方法で対抗できます[*90]。

ですから、新しいプログラムが作られれば、必ずと言っていいほど著作権が関係してきます。

逆に言えば、著作物をパブリックドメインとするためには、著作権を放棄する手続きなどが必要になるわけです。また、著作物を利用するには、著作権者から利用許諾をもらうべし、というのが著作権法に則った正しい方法です。

したがって、ソフトウェアとともに

- 著作権に基づく利用許諾契約書
- 著作権を放棄する宣言あるいは放棄済であることの明示的な説明

のどちらかを手に入れないと、著作権法に違反していない確信を持ってソフトウェアを使うことが難しい、ということになります。 Debian Project は、社会契約 [1] に基づいて行動しており、この点に関して厳密に考えてパッケージを作っているわけです。

34.4.2 著作権の存続期間

権利が有効である時間の範囲を存続期間と言います。

「 著作者が死んでも著作権の保護が続くってどういうこと?」と不思議に思うかも知れませんね。著作権は相続できるのです。相続する人があれば、続くんです。相続を含めて、権利を引き継ぐことを「 承継」と言います。承継する人がなければ、著作権は消滅します。

筆者からお願い!

あなたが作って公開しているプログラム、

あなたの死後に誰が著作権を相続するか予め周りの人に教えておいて下さい。

[*88] Debian も電子計算機などの指定商品での商標登録がされており、登録番号は 第４５９５２８８号です。ちなみに、指定商品「 菓子、パン」で「 デビアン」という称呼の登録番号第１７０８０３２号の登録商標があります。面白いですね。

[*89] このような権利の発生のさせ方を無方式主義と言います。なお、かつてアメリカは、無方式主義ではありませんでした。

[*90] 著作権法第百十四条に基づいて損害額を推定しても、フリーソフトウェアの場合、 0 円にしかならないですが…。

なお、 FSF[*91]は、 FSF へ著作権を譲渡することを勧めています [4]。著作者の死後、 GPL[*92]で公開したフリーソフトウェアの行く末を託すこともできますね[*93]。

34.4.3 著作権の内訳

著作権とは、権利の束みたいなものです。著作物を利用する方法などによって、細かい権利が定められています。

それらは大別すると、「 著作人格権」と「 著作財産権」に分かれ、それぞれの下に次のような権利が条文で定められています。

著作人格権 「 公表権」「 氏名表示権」「 同一性保持権」

著作権(著作財産権) 「 複製権」「 上演権」「 演奏権」「 上映権」「 公衆送信権」「 伝達権」「 口述権」「 展示権」「 頒布権」「 譲渡権」「 貸与権」「 翻訳権」「 翻案権」「 二次的著作物の利用に関する権利 」

この中には、ソフトウェアが関係しないものもあります。

34.5 Debian Project が配布しているものの何が著作物?

さて、著作物とはどんなものがあるか、著作権法を読んでみると、次のようなことが書いてあります。

著作物とは、「 思想又は感情を創作的に表現したものであつて、文芸、学術、美術又は音楽の範囲に属するものをいう。」(著作権法第二条)

更に、著作権法では、例を挙げて著作物を定義しています。

第十条 この法律にいう著作物を例示すると、おおむね次のとおりである。

一 小説、脚本、論文、講演その他の言語の著作物

二 音楽の著作物

三 舞踊又は無言劇の著作物

四 絵画、版画、彫刻その他の美術の著作物

五 建築の著作物

六 地図又は学術的な性質を有する図面、図表、模型その他の図形の著作物

七 映画の著作物

八 写真の著作物

九 プログラムの著作物

2 **事実の伝達にすぎない雑報及び時事の報道は、前項第一号に掲げる著作物に該当しない。**

3 **第一項第九号に掲げる著作物に対するこの法律による保護は、その著作物を作成するために用いるプログラム言語、規約及び解法に及ばない。この場合において、これらの用語の意義は、次の各号に定めるところによる。**

一 プログラム言語 プログラムを表現する手段としての文字その他の記号及びその体系をいう。

二 規約 特定のプログラムにおける前号のプログラム言語の用法についての特別の約束をいう。

三 解法 プログラムにおける電子計算機に対する指令の組合せの方法をいう。

ということで、プログラムは、プログラム著作物です。オブジェクトコードもプログラム著作物である、という判例もあるようです[*94]。

また、 Debian Project が配布するものにはプログラム以外にも、文章、写真画像、地図データなどありますが、著作

[*91] Free Sofware Foundation. ウェブサイトは http://www.fsf.org/

[*92] GNU General Public License

[*93] 著作権の譲渡に関しては、日本の著作権法には第三者対抗要件に登録が必要などの難しい話が関係するので、筆者はどうしたらいいのか、理解できていません。

[*94] 東京地判昭和 60 年 3 月 8 日判タ 561 号 169 頁「 ディグダグ」事件 という判例があるそうです [6]。

物です。

34.6 著作権を踏まえた Debian Project からの配布物とのおつき合い

34.6.1 著作権のことを気にせずに Debian のシステムをインストールしました。何かまずいことをしていないでしょうか?

自分で使う目的でインストールしたのですよね?

インストールする前に利用許諾契約を読んで、同意して、それからインストールするのが理想的ですが、そうでない場合もよくあるかと思います。

Debian Project が配っているインストーラーを使って普通にインストールしたなら、インストールしたソフトウェアの利用許諾契約に同意したことにしましょう。インストーラーでインストールしたソフトウェアはすべて DFSG[*95] に準拠しているので、ほとんどあなたは不利益を被っていません。

- 金銭的な対価は要求されません。
- 使うだけなら無償の貢献も要求されません。
- あなたの某かの権利を放棄したり断念したり譲渡したりすることもありません。

なにか気をつけることがあるとすれば、免責条項くらいでしょうか。

ほぼすべてのフリーソフトウェアの利用許諾では、その動作に関して無保証で、免責条項が盛り込まれています[*96]。無保証であることに関する同意は、ほとんどのソフトウェアが採用している利用許諾の契約の条項の一部です。同意できないなら、利用する権利はありません。

したがって、著作者に対して、次に掲げることについて何の文句をつける筋合いもありません。

- 期待通りに動かない
- そもそも動かない
- ソフトウェアを使ったせいで、地位や財産を失った
- ソフトウェアを信じたせいで、地位や財産を失った

でも、多くのフリーソフトウェアの開発元や Debian Project には、バグ報告の窓口がありますね。あれは、義務でやっているのでないのです。大いなる親切以外の何物でもありません。

34.6.2 Debian のフリーソフトウェアを使わないけど配ります。

ちょっと待って! あなたは、著作者と契約を結ばなくてはなりません!

著作権には「複製権」「自動公衆送信権」「送信可能化権」という権利が含まれているのです。極めて簡単に分かりやすくいうと

複製権　簡単にいうと、コピーする権利。
自動公衆送信権　公衆に送信する権利。
送信可能化権　公衆がアクセスしてダウンロードできるサーバーに著作物を置く権利。

です。著作者が独占的に保持する権利ですから、あなたは、ソフトウェアを配ったり、公開されているサイトにアップロードする前には、著作者の許可が必要です。

でも実は、 Debian アーカイブの main セクションと contrib セクションに含まれるソフトウェアのソースコードを**そのまま**無料で配布する場合は、利用許諾に同意する必要はあるものの、実質的には何らかの義務や制約に縛られることはありませんので、ほとんど気にかける必要はないです。

というのも、 main と contrib に含まれるソフトウェアは、すべて DFSG[2] に準拠してます。 DFSG に準拠してい

[*95] The Debian Free Software Guidelines[2]
[*96] フリーでなくても、無保証である場合が多いですし、保証はあっても賠償額の上限をソフトウェア購入代金とすることも珍しくありません。

ることということは、ソースコードの自由な配布が利用許諾契約上、認められていることになるのです。

これに対して、バイナリパッケージを配布する場合は、配布先でソースコードが手に入れることができるように配慮しないといけないライセンスが多いです[*97]。ソースコードが入手できるような配布を強制することで、ソフトウェアの自由を担保しようとしているわけです。注意しましょう。

また、利用許諾の内容を理解する前に、適当に一部を削ったり、一部を抜き出して配らないようにしましょう。というのも、著作権には「同一性保持権」という権利があります。勝手に削ることも含めて、著作物を勝手に改変することは著作権法が禁じています。改変に際しては許諾が必要で、許諾された範囲での改変しか許されません。

幸い、DFSG 準拠なら改変は許されますし、改変されたソフトウェアの複製を配布することも許されますが、契約毎に許されるための条件はあります。

それから、勝手に著作者の氏名を削ってはいけません。「氏名表示権」を侵害することになります。

そのまま配るのが一番無難です。

34.6.3 Debian をインストールされてるパソコンをもらいました

一般的に、正規の方法で入手したソフトウェアを手元にコピーを残さずに誰かに譲り渡すことは構わないのですが、パソコンをもらっただけでは、そのパソコンにインストールされているソフトウェアを無条件に使っていいことにはなりません。くれた人は「好きにしていいよ」と言ってくれていても、利用許諾の契約には同意して利用しましょう。

物品の譲渡と一緒に著作権の利用許諾がなされたとみなせる場合というのは、特別に著作権法に明記されています[*98]。原則として、物品の譲渡とそこに宿っている著作物の利用許諾とは、別ものなのです。

ちなみに、ソフトウェアの利用許諾書にはしばしば non-transferable という言葉で、利用権の譲渡を明示的に禁止する文言が現れますが、その場合は、契約上利用権を譲渡できないことになっています。

34.7 特許との関係

利用許諾書には、著作権ではない別の知的財産権に基づく利用許諾が含まれる場合があります。よく使われる「ライセンス」と言う言葉に含まれる、利用許諾の根拠になる権利は著作権だけとは限らないわけです。

例えば、DFSG 準拠の利用許諾の中には、特許に関する条項が含まれているものがあり、その代表的なものには、Apache License 2.0 と、GPL 3.0 があります。

この二つの利用許諾には、ともに、「ソースコードの貢献者が保有する特許のうち、ある範囲のものは、無償で利用を許諾する、」という趣旨の条項が含まれています。Apache License 2.0 は、貢献者が自分の貢献したコードに含まれる特許を利用許諾するのに対し、GPL 3.0 では、貢献者が配布したコードに含まれる特許を利用許諾する点が違います。

34.8 その他関係する法律

最後に、ソフトウェアが関係しそうな法律をいくつか挙げておきます。

- 不正アクセス禁止法
- 不正競争防止法
- 刑法のいわゆるウィルス作成罪
- 関税法
- 民法

[*97] GPL や LGPL (GNU Lesser General Public License) が典型的な例です。
[*98] 美術品の展示権など。

34.9 宿題

次の問題に答えてみましょう。

- 「アルゴリズム体操」の振付けは、著作権法第十条の何の著作物でしょうか?
- GCC (GNU Compiler Collection) は、プログラムの著作物として保護されるでしょうか?
- 著作権(著作財産権)のうち、ソフトウェアに関係あるものを挙げてみましょう。
- かつて特許によって自由な配布が制限されたソフトウェアがありました。どんなものがあるか、調べてみましょう。

34.10 まとめ

フリーソフトウェアに関わる著作権の仕組みを簡単に解説しました。

参考文献

[1] Debian 社会契約, http://www.debian.org/social_contract

[2] The Debian Free Software Guidelines, http://www.debian.org/social_contract#guidelines

[3] 著作権法 , http://law.e-gov.go.jp/htmldata/S45/S45HO048.html

[4] なぜ FSF は貢献者に著作権の譲渡をお願いしているのか http://www.gnu.org/licenses/why-assign.html

[5] GPLv3 逐条解説 http://ossipedia.ipa.go.jp/DL/doc/187/5/0904/ON/

[6] 著作権法 , 斉藤博, 2000 年, 有斐閣

35 ITP から始めるパッケージメンテナへの道

よしだ ともひろ

35.1 はじめに

とある便利なソフトウェアを見つけて、それがまだ Debian パッケージになっていなかったとき、RFP(Request For Package:パッケージ化の要求) をしてもよいのですが、自分でパッケージ化して Debian の一部として提供することもできます。そこで、ITP(Intent To Package: パッケージ化の宣言) からアップロードまでの流れをまとめてみました。

35.2 まずは ITP

35.2.1 ITP の前に必要なこと

既にパッケージ化されていないか?

`http://www.debian.org/distrib/packages` で探してみたり、aptitude search や apt-cache search で確認してみましょう。

既に ITP されていないか?

他の誰かが同じようにパッケージ化しようと思って、ITP しているかもしれません。

`http://www.debian.org/devel/wnpp/being_packaged` をチェックしてみましょう。

既に RFP されていないか?

他の誰かが、パッケージ化して欲しいと思って、RFP しているかもしれません。RFP されている場合は、ITP することでパッケージ化の宣言を行います。

`http://www.debian.org/devel/wnpp/requested` をチェックしてみましょう。

35.3 ITP の仕方

パッケージを ITP するのは reportbug(apt-get—aptitude install reportbug) で行う方法が `http://www.debian.org/devel/wnpp/` で紹介されています。また、電子メールでも行うことができます。ここでは、電子メールで ITP を行う方法の一つ、Mew を使う方法を紹介します。

まず、debian-el パッケージをインストールします。

```
$ sudo aptitude install debian-el
```

次に`~/.emacs` に以下を追加します。

```
(autoload 'mew "mew" nil t)
(autoload 'mew-send "mew" nil t)

(if (boundp 'read-mail-command)
        (setq read-mail-command 'mew))
(autoload 'mew-user-agent-compose "mew" nil t)
(if (boundp 'mail-user-agent)
        (setq mail-user-agent 'mew-user-agent))
(if (fboundp 'define-mail-user-agent)
        (define-mail-user-agent
                'mew-user-agent
                'mew-user-agent-compose
                'mew-draft-send-message
                'mew-draft-kill
                'mew-send-hook))
```

こうしておいて emacs を起動し、 `M-x debian-bug RET` とします。ミニバッファに

```
Report a bug for a [P]ackage or [F]ile: (default P)
```

と表示されるので “P” を入力します。パッケージ名を聞いてきますので wnpp と入力します。 wnpp というのは Work-Needing and Prospective Packages(作業が望まれるパッケージ) という擬似パッケージのことで、 ITP や RFP を行うときに使用する他、 RFH(Request For Help:助力の要求)、 RFA(Request For Adoption:養子引き取りの要求)、 O(Orphan:みなしご)、 ITA(Intent To Adopt:養子引き取りの表明) にも使います。

次に Action を聞かれます。 TAB を押すと候補が表示されますので ITP とします。

あとは、パッケージ名、パッケージの簡単な説明を入力します。 debian-devel へ CC するか? も聞かれますので “y” と入力するとメールの雛形が表示されますので、残りの Version、 Upstream Author、 URL or Web page、 License を入力してメールを送信します。

35.4 GnuPG キーサイン

Debian 公式パッケージにするには dsc ファイルと changes ファイルに署名が必要です。 GnuPG(GPG) で署名しますが、信頼された鍵でないと意味がないのでキーサインパーティなどで Debian 開発者の方とキーサインをしておきましょう。 GPG 公開鍵は後に説明する mentors.debian.net を使う時にも必要ですので、まだ、作成していない方は作っておくことをおすすめします。

2012 年 6 月 23 日 (土) の大統一 Debian 勉強会でもキーサインパーティが予定されていますので、是非参加しましょう。

35.5 パッケージ作成

パッケージの作成方法は過去の Debian 勉強会などでも何度か紹介されていますので、ここでは詳細は割愛し注意する箇所のみとします。

35.5.1 lintian clean にする

lintian は Debian パッケージを精査し、バグやポリシー違反を報告します。アップロードする前に、必ずそのパッケージに対して lintian を実行してください。 lintian を実行する際はオプションの -v を付けて changes ファイルを指定します。 lintian が出力する情報がわかりにくければ -i オプションを追加しましょう。出力される情報に “E:” や “W:” がなくなるように修正し、 lintian clean にしましょう。

35.5.2 パッケージに署名する

お試しでパッケージを作ったりする場合はパッケージへの署名を省略しますが、公式パッケージにするにはちゃんと署名することが必要です。 debuild や dpkg-buildpackage で-us -uc オプションをつけずにパッケージ作成すると、 dsc ファイルと changes ファイルに GnuPG で署名すためのパスフレーズを聞いてきますので入力すれば OK です。

35.6 mentors.debian.net へ登録

35.6.1 mentors.debian.net とは

Debian へのパッケージアップロード権限は Debian 開発者のみが持っているのですが、 Debian 開発者でない人もアップーロードできる仕組みがあります。 Debian 開発者の方にスポンサーになっていただいて、助言をいただきながら修正していく場、といった感じです。

35.6.2 サインアップ

`http://mentors.debian.net/` に “Sign me up” という `http://mentors.debian.net/register/register` へのリンクがありますのでクリックします。

Full name, E-mail, Password, Confirm password を入力し、 Account type は Maintainer を選択して Submit を押下します。入力した E-mail Address へ確認のメールが飛んできますので、記載された URL へアクセスして Activate するとアカウントが使用可能になります。

ログインして、 My account ページの Change GPG key で GPG キーを登録できますのでアスキー形式でエクスポートした公開鍵のファイルを選択して登録しておきます。

35.6.3 アップロード

mentors.debian.net にアップロードする方法は、 dupload や dput などのツールで行えます。今回は dput で行う方法をご紹介します。

1. 準備

   `~/.dput.cf` を作ります。内容は `http://mentors.debian.net/intro-maintainers` を参考にしました。 http と ftp の設定が記載されているのですが、 http の設定としました。こんな感じです。

   ```
   [mentors]
   fqdn = mentors.debian.net
   incoming = /upload
   method = http
   allow_unsigned_uploads = 0
   progress_indicator = 2
   # Allow uploads for UNRELEASED packages
   allowed_distributions = .*
   ```

2. さあ、アップロード

   いよいよアップロードです。 `dput mentors` ***changes*** **ファイル** で行います。ここまでのパッケージ作成で問題がなければ mentors.debian.net へのアップロードは成功するはずです。

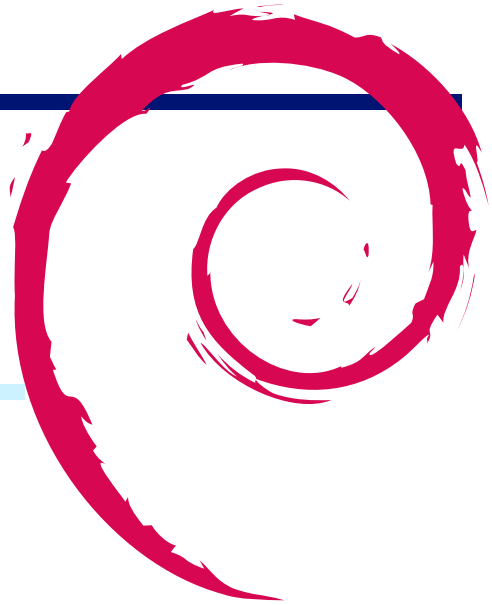
   しばらくすると mentors.debian.net からメールが飛んできます。メールの内容は「 スポンサーが必要ならスポンサーを探してるにしろ」とか、「 RFS(request for sponsorship) でスポンサーを募集できるよ」というような内容ですので従います。

35.7 スポンサーを探す

無事 mentors.debian.net へアップロードできたら debian-mentors@lists.debian.org や debian-devel@debian.or.jp に RFS(request for sponsorship) を投げて、スポンサーになってくれる方を探します。

35.8 おわりに

今回、はじめて ITP して、 mentors.debian.net にアップロードしてみました。色々調べたりしながらでしたので時間はかかりましたが、それほど難しくはないと思います。 Debian パッケージになっていないソフトウェアを見つけたら是非 ITP しましょう!



36 t-code のバグレポートをしてみた

西田孝三

36.1 はじめに

ここでは「 Debian パッケージのバグを発見した場合にどうすればいいのか」を伝えることを目的とし、実際のバグ例を挙げながらバグレポートの手順を説明します。バグレポートするパッケージは Emacs で日本語入力するための拡張 Elisp である t-code パッケージです。手順の大きな流れは下記の 4 段階に分けられます。

1. バグトラッカーでの報告
2. 自分で修正パッチを作る (可能であれば)
3. 自分で Debian パッケージを作る (可能であれば)
4. DebianDeveloper(DD) に 3 のパッケージを取り込んでもらう (知り合いの DD がいれば)

実際にはまともにできたのは 1 だけでした。申し訳ありませんがお付き合いください。それでは実際の t-code パッケージのバグを見ながら順を追ってバグレポート、修正を行なっていきましょう。

36.2 バグトラッカーでの報告

t-code の Debian パッケージは 2003 年のバージョン 2.3.1 以降更新がなく Emacs23 や 24 で部首合成変換ができないバグがあります。まずこのバグを報告する前に Bug Tracker にすでに報告されていないかどうかウェブブラウザで簡単に確認しましょう。 Debian の Bug Tracker の URL は `http://www.debian.org/Bugs/` です。バグの選択という select box と input box がありますのでそれぞれ「 パッケージ名が」、「 t-code」と入力しバグ報告情報を確認します。2011 年 12 月 24 日時点では automake のバージョンに関するバグしか報告されていなかったので、バグ報告をする必要があることがわかりました。 reportbug というパッケージを用いると便利にバグ報告ができるため、もしこのパッケージがインストールされていなければインストールしてください。

```
$ sudo aptitude install reportbug
```

インストールができていれば reportbug コマンドを入力します。

```
kozo2@debian:~$ reportbug
Welcome to reportbug! Since it looks like this is the first time you have used reportbug, we are configuring its
 behavior. These settings will be saved to the file ‘‘/home/kozo2/.reportbugrc’’, which you will be
free to edit further.
Please choose the default operating mode for reportbug.

1 novice    Offer simple prompts, bypassing technical questions.

2 standard  Offer more extensive prompts, including asking about things that a moderately sophisticated user would
 be expected to know about Debian.

3 advanced  Like standard, but assumes you know a bit more about Debian, including ‘‘incoming’’.

4 expert    Bypass most handholding measures and preliminary triage routines. This mode should not be used by people
 unfamiliar with Debian’s policies and operating procedures.

Select mode: [novice]
```

最初なので novice でいいと思います。

```
Will reportbug often have direct Internet access? (You should answer yes to this question unless you know what you
 are doing and plan to check whether duplicate reports have been filed via some other channel.)
[Y|n|q|?]?
```

よくわからないので説明に従い Y にします。

```
What real name should be used for sending bug reports?
[kozo2]> Kozo Nishida
```

名前を入力します。

```
Which of your email addresses should be used when sending bug reports? (Note that this address will be visible in
 the bug tracking system, so you may want to use a webmail address or another address with
spam filtering capabilities.)
[kozo2@debian]> knishida@riken.jp
```

メールアドレスを書きます。

```
Do you have a ‘‘mail transport agent’’ (MTA) like Exim, Postfix or SSMTP configured on this computer to send mail
 to the Internet? [Y|n|q|?]? n
```

メールを送るプログラムの設定をしていなければ n にします。

```
Please enter the name of your SMTP host. Usually it’s called something like ‘‘mail.example.org’’ or
 ‘‘smtp.example.org’’. If you need to use a different port than default, use the <host>:<port> alternative f
Just press ENTER if you don’t have one or don’t know, and so a Debian SMTP host will be used.
>
```

これも SMTP 設定を知らないのでそのまま ENTER します。

```
Please enter the name of your proxy server. It should only use this parameter if you are behind a firewall.
 The PROXY argument should be formatted as a valid HTTP URL, including (if necessary) a port num
example, http://192.168.1.1:3128/. Just press ENTER if you don’t have one or don’t know.
>
```

プロキシサーバも知らないのでそのまま ENTER します。

```
Dear Maintainer,
*** Please consider answering these questions, where appropriate ***

   * What led up to the situation?
   * What exactly did you do (or not do) that was effective (or
     ineffective)?
   * What was the outcome of this action?
   * What outcome did you expect instead?

*** End of the template - remove these lines ***


-- System Information:
Debian Release: wheezy/sid
  APT prefers unstable
  APT policy: (500, 'unstable')
Architecture: amd64 (x86_64)

Kernel: Linux 3.1.0-1-amd64 (SMP w/2 CPU cores)
Locale: LANG=ja_JP.UTF-8, LC_CTYPE=ja_JP.UTF-8 (charmap=UTF-8)
Shell: /bin/sh linked to /bin/dash

Versions of packages t-code depends on:
ii  emacs-snapshot-nox [emacs-snapshot]  1:20111219-1

t-code recommends no packages.

t-code suggests no packages.

-- no debconf information
```

こういうひな形が表示されますのでこれを編集します。以下が編集後のレポートです。

```
Subject: t-code version 2.3.1 fails bushu-convert
Package: t-code
Version: 2:2.3.1-3
Severity: normal

Dear Maintainer,
*** Please consider answering these questions, where appropriate ***

   * T-code pre bushu-convert doesn't work
   * pre bushu-convert error message is tcode-self-insert-command: Wrong type argument: characterp, nil
   * T-code post bushu-convert doesn't work(There is no error message).

-- System Information:
Debian Release: wheezy/sid
  APT prefers unstable
  APT policy: (500, 'unstable')
Architecture: amd64 (x86_64)

Kernel: Linux 3.1.0-1-amd64 (SMP w/2 CPU cores)
Locale: LANG=ja_JP.UTF-8, LC_CTYPE=ja_JP.UTF-8 (charmap=UTF-8)
Shell: /bin/sh linked to /bin/dash

Versions of packages t-code depends on:
ii  emacs-snapshot-nox [emacs-snapshot]  1:20111219-1

t-code recommends no packages.

t-code suggests no packages.

-- no debconf information
```

ま、こんなとこでしょう。編集を終えると下記の確認をしてきます。

```
Report will be sent to ‘‘Debian Bug Tracking System’’ <submit@bugs.debian.org>’’--configure’’ option.
Submit this report on t-code (e to edit) [Y|n|a|c|e|i|l|m|p|q|d|t|s|?]? Y
```

これでよいので Y を入力します。もう一度編集したければ e を入力します。

```
Connecting to reportbug.debian.org via SMTP...

Bug report submitted to: ‘‘Debian Bug Tracking System’’ <submit@bugs.debian.org>
Copies will be sent after processing to:
  knishida@riken.jp

If you want to provide additional information, please wait to receive the bug tracking number via email;
 you may then send any extra information to n@bugs.debian.org (e.g. 999999@bugs.debian.org), where n is the
bug number. Normally you will receive an acknowledgement via email including the bug report number within
 an hour; if you haven't received a confirmation, then the bug reporting process failed at some point
(reportbug or MTA failure, BTS maintenance, etc.).
```

これでバグレポートは終わりです。入力した自分のメールアドレスにバグレポートしたことを確認するメールが届いていることを確認してください。もしメールが届いていたらバグレポートの番号を確認して http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=653167 のように bug=の後にバグ ID を入力してウェブページでも内容を確認してみてくだ

さい。

36.3 自分で修正パッチを作る

もし可能であれば Debian のルールに従い修正パッチを作ってみましょう。 t-code は安宅正之さんが中心となり Google code で開発を引き継がれており (http://code.google.com/p/tcode/)、このリポジトリの trunk の t-code は Emacs23、 24 で使えることを確認しています。また青田直大さんが個人的に修正をされた github のリポジトリ (https://github.com/naota/tc) も Emacs23,24 で使えることを確認しています。

どちらのリポジトリのコードをどのようにパッケージ化していけばよいか (これまでのコードとの差分情報の記述など) わからなかったためこの部分については次回への宿題ということにさせてください。申し訳ありません。と、いうわけで次回は「 バグ修正リポジトリコードの取り込みと dpatch の使い方」で発表させてください。

36.4 自分で Debian パッケージを作る

本来ならこれまでのパッケージからの変更を Debian のルールに従って記述したファイルを作成する必要があるかと思いますが、ここではその手続をすっとばして前述のリポジトリのソースコードの内、安宅さんのリポジトリのソースコードを用いて Debian パッケージを作ることを試みます。

まず t-code パッケージのソースコードを取得します。

```
$ apt-get source t-code
$ svn checkout http://tcode.googlecode.com/svn/trunk/ tcode-read-only
$ ls
t-code-2.3.1  t-code_2.3.1-3.diff.gz  t-code_2.3.1-3.dsc  t-code_2.3.1.orig.tar.gz
```

次にソース中の debian ディレクトリを安宅さんのリポジトリのソースコード中にコピーします。

```
$ cp -r t-code-2.3.1/debian tcode-read-only/tc/
```

それではこれで Debian package が作れるか試してみましょう。 Debian package は debuild コマンドで作成できます。 debuild コマンドは devscripts パッケージをインストールすると使えるようになります。

```
$ sudo aptitude install devscripts
$ cd tcode-read-only/tc
$ sudo debuild -us -uc
This package has a Debian revision number but there does not seem to be
an appropriate original tar file or .orig directory in the parent directory;
(expected one of t-code_2.3.1.orig.tar.gz, t-code_2.3.1.orig.tar.bz2,
t-code_2.3.1.orig.tar.lzma,  t-code_2.3.1.orig.tar.xz or tc.orig)
continue anyway? (y/n) y
```

親ディレクトリにソースの tar.gz がないと言われますが無視して build を試みます。

```
make[2]: ‘install-exec-am’ に対して行うべき事はありません.
/bin/sh ../mkinstalldirs /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc
 /usr/bin/install -c -m 644 ./pd_kihon.yom /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/pd_kihon.yom
 /usr/bin/install -c -m 644 ./greece.maz /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/greece.maz
 /usr/bin/install -c -m 644 ./jukujiku.maz /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/jukujiku.maz
 /usr/bin/install -c -m 644 ./t225.dat /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/t225.dat
 /usr/bin/install -c -m 644 ./t300.dat /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/t300.dat
 /usr/bin/install -c -m 644 ./t400.dat /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/t400.dat
 /usr/bin/install -c -m 644 ./t450.dat /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/t450.dat
 /usr/bin/install -c -m 644 ./t575.dat /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/t575.dat
 /usr/bin/install -c -m 644 ./t675.dat /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/t675.dat
 /usr/bin/install -c -m 644 ./t900.dat /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/t900.dat
 /usr/bin/install -c -m 644 ./t1200.dat /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/t1200.dat
 /usr/bin/install -c -m 644 ./t1353.dat /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/t1353.dat
 /usr/bin/install -c -m 644 ./itaiji.maz /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/itaiji.maz
 /usr/bin/install -c -m 644 ./mazegaki.dic /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/mazegaki.dic
 /usr/bin/install -c -m 644 ./mkcertain.pl /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/mkcertain.pl
make[2]: ディレクトリ ‘/home/kozo2/tcode-read-only/tc/mazegaki’ から出ます
make[1]: ディレクトリ ‘/home/kozo2/tcode-read-only/tc/mazegaki’ から出ます
/usr/bin/make -C skkinput3  DESTDIR=/home/kozo2/tcode-read-only/tc/debian/t-code install
make[1]: ディレクトリ ‘/home/kozo2/tcode-read-only/tc/skkinput3’ に入ります
make[2]: ディレクトリ ‘/home/kozo2/tcode-read-only/tc/skkinput3’ に入ります
/bin/sh ../mkinstalldirs /home/kozo2/tcode-read-only/tc/debian/t-code/usr/bin
mkdir /home/kozo2/tcode-read-only/tc/debian/t-code/usr/bin
 /usr/bin/install -c tcinput /home/kozo2/tcode-read-only/tc/debian/t-code/usr/bin/tcinput
/bin/sh ../mkinstalldirs /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc
 /usr/bin/install -c -m 644 ./init.el /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/init.el
 /usr/bin/install -c -m 644 ./skk-startup.el /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/skk-startup.el
 /usr/bin/install -c -m 644 ./tc-skki.el /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/tc-skki.el
 /usr/bin/install -c -m 644 ./load-path.el /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/tc/load-path.el
make[2]: ディレクトリ ‘/home/kozo2/tcode-read-only/tc/skkinput3’ から出ます
make[1]: ディレクトリ ‘/home/kozo2/tcode-read-only/tc/skkinput3’ から出ます
cd /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/t-code && \
             chmod +x bushu2canna where mkcertain.pl
/bin/sh: 1: cd: can’t cd to /home/kozo2/tcode-read-only/tc/debian/t-code/usr/share/t-code
make: *** [install] エラー 2
dpkg-buildpackage: error: fakeroot debian/rules binary gave error exit status 2
debuild: fatal error at line 1348:
dpkg-buildpackage -rfakeroot -D -us -uc failed
```

失敗でした! 今回はここまででお許しください。パッケージ作成成功は次回の発表までの宿題ということで!

36.5 DD にパッケージを Debian のリポジトリに取り込んでもらう

Debian パッケージができたら、関西の DD にビールでも飲みながら話しかけてみましょう。

36.6 おわりに

おもいっきり尻切れトンボで申し訳ありません! ただ、最初のバグレポートをするだけでもそれはそれで結構 Debian に貢献していることになると思います。何事も最初のとっかかりさえ掴めれば後は楽かと思いますので皆さんもどんどんバグ報告をして残りのパッチ作成や、パッケージ作成も一緒にやってみませんか。それではまた次回!(土下座)

37 emacs24 で問題なく使える t-code.deb を作った話

西田 孝三

第 54 回関西 Debian 勉強会では「 t-code のバグレポートをしてみた」という題で発表しました。前回の発表では Debian のバグトラッキングシステムを使って t-code パッケージのバグレポートから Debian パッケージの作成を試み、失敗するところで止まっていました。今回はまず問題なく動作するパッケージを作成すると共に、とりあえずパッケージを作成する (いわゆるオレオレパッケージ) にはどのようなことを行えばよいかを下記の 4 点に分けてお伝えします。

1. (バグを含む) 現在の Debian パッケージのソースコードの取得と、ビルド方法の確認
2. バグ修正を含むソースコードへの置き換えとビルド、オレオレパッケージの作成
3. オレオレパッケージの動作確認
4. (修正が必要な場合)Debian ディレクトリ下のファイルの変更

37.1 現在の Debian パッケージのソースコードの取得と、ビルド方法の確認

まず既存の Debian パッケージのソースコードを取得し、ビルドすることでパッケージ作成方法の確認をします。パッケージのソースコードの取得を行うには apt-get コマンドに source オプションを付けパッケージを指定します。

```
kozo2@debian:~/sandbox$ apt-get source t-code
kozo2@debian:~/sandbox$ ls
t-code-2.3.1            t-code_2.3.1-3.dsc
t-code_2.3.1-3.diff.gz  t-code_2.3.1.orig.tar.gz
```

これでソースが取得できます。今回はとにかくパッケージを作ることが目的ですので*.dsc, *.diff.gz, *.orig.tar.gz の意味は説明せず、 t-code-2.3.1 へ移動し、まずこのソースを用いてビルドを行いパッケージ (.deb ファイル) を作成してみます。これには debuild コマンドを用います。

```
kozo2@debian:~/sandbox$ cd t-code-2.3.1/
kozo2@debian:~/sandbox/t-code-2.3.1$ ls
acinclude.m4  ChangeLog.old  COPYING  install-sh   mazegaki       skkinput3
aclocal.m4    config.guess   debian   kinput2      missing
AUTHORS       config.sub     doc      lisp         mkinstalldirs
bushu-util    configure      etc      Makefile.am  NEWS
ChangeLog     configure.in   INSTALL  Makefile.in  README
kozo2@debian:~/sandbox/t-code-2.3.1$ debuild -us -uc
```

debuild コマンドのオプション -us -uc は署名はせずに単に package に build する時に用います。これで一つ上の directory に.deb ファイルができます。

```
kozo2@debian:~/sandbox/t-code-2.3.1$ ls ..
t-code-2.3.1            t-code_2.3.1-3_amd64.build    t-code_2.3.1-3.diff.gz  t-code_2.3.1.orig.tar.gz
t-code_2.3.1-3_all.deb  t-code_2.3.1-3_amd64.changes  t-code_2.3.1-3.dsc
```

37.2 バグ修正を含むソースコードへの置き換えとビルド、オレオレパッケージの作成

次にバグのあるソースコードを新しいソースコードに置き換え、ビルドができるかどうか試してみます。新しいソースコードは http://code.google.com/p/tcode/から取得します。

```
kozo2@debian:~/sandbox/t-code-2.3.1$ cd ..
kozo2@debian:~/sandbox$ svn co http://tcode.googlecode.com/svn/trunk/ tcode-read-only
```

置き換えが必要なファイルを新しいソースコードをコピーすることで上書きします。

```
kozo2@debian:~/sandbox$ cp tcode-read-only/tc/bushu-util/* t-code-2.3.1/bushu-util/
kozo2@debian:~/sandbox$ cp tcode-read-only/tc/etc/* t-code-2.3.1/etc/
kozo2@debian:~/sandbox$ cp tcode-read-only/tc/lisp/* t-code-2.3.1/lisp/
kozo2@debian:~/sandbox$ cp tcode-read-only/tc/mazegaki/* t-code-2.3.1/mazegaki/
```

これでビルドが通るか試します。 (通ります)

```
kozo2@debian:~/sandbox$ cd t-code-2.3.1
kozo2@debian:~/sandbox$ debuild -us -uc
```

とりあえず、置き換えたソースコードでパッケージができました。次はこれをインストールし動作確認してみます。

37.3 オレオレパッケージの動作確認

先程作成した用いるソースコードに変更を加えた Debian パッケージをインストールし、動作に問題がないか確認します。インストールするには dpkg コマンドに-i オプションをつけてインストールしたい.deb ファイルを指定します。

```
kozo2@debian:~/sandbox$ sudo dpkg -i t-code_2.3.1-3_all.deb
Selecting previously unselected package t-code.
(Reading database ... 97587 files and directories currently installed.)
Unpacking t-code (from t-code_2.3.1-3_all.deb) ...
Setting up t-code (2:2.3.1-3) ...
install/t-code: Handling install for emacsen flavor emacs23
Processing triggers for install-info ...
kozo2@debian:~/sandbox$
```

インストールは問題ないようです。それでは t-code が問題ないか emacs を起動し試してみます。試しに t-code 練習プログラム eelll の起動を試みると emacs が下記のメッセージを出し、何か問題があることがわかります。

```
Debugger entered--Lisp error: (error ``ファイル /usr/share/tc/EELLLTXT が存在しません。'')
  signal(error (``ファイル /usr/share/tc/EELLLTXT が存在しません。''))
  error(``ファイル %s が存在しません。'' ``/usr/share/tc/EELLLTXT'')
  tcode-set-work-buffer(`` *eelll: text*'' ``EELLLTXT'')
  eelll-completing-read()
  call-interactively(eelll t nil)
  execute-extended-command(nil)
  call-interactively(execute-extended-command nil nil)
```

37.4 Debian ディレクトリ下のファイルの変更

先程の問題は t-code が用いる交ぜ書き、部首合成変換用データがインストールされているディレクトリを指定する emacs の変数 tcode-site-data-directory の設定によるものです。この tcode-site-data-directory を指定し直すにはパッケージ作成用 directory 下の debian/emacsen-startup に下記の設定を追加しビルドし直します。

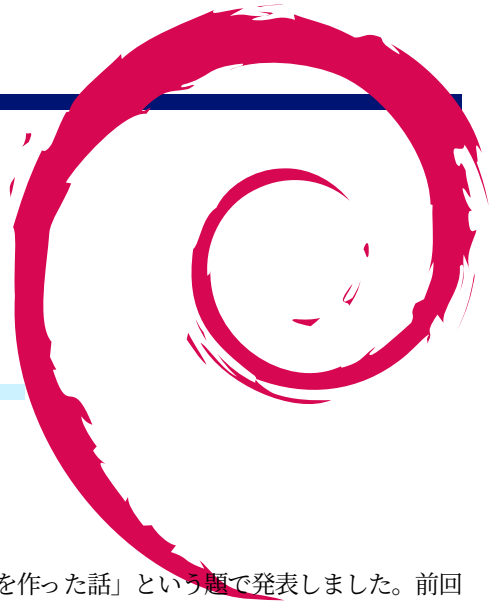
```
;;; 50t-code.el --- Debian t-code startup file  -*-mode: emacs-lisp;-*-

;;; Code:

(let ((lispdir (concat ‘‘/usr/share/’’ (symbol-name flavor) ‘‘/site-lisp/t-code’’)))
  (when (and (featurep ’mule) (file-exists-p (concat lispdir ‘‘/tc.elc’’)))
    (if (fboundp ’debian-pkg-add-load-path-item)
        (debian-pkg-add-load-path-item lispdir)
      (setq load-path (cons lispdir load-path)))
    ;;
    (require ’tc-setup)
    (defconst tcode-site-data-directory ‘‘/usr/share/t-code/’’)
    ;;
    ))

;;; 50t-code.el ends here
```

```
kozo2@debian:~/sandbox/t-code-2.3.1$ debuild -us -uc
kozo2@debian:~/sandbox/t-code-2.3.1$ sudo aptitude purge t-code
kozo2@debian:~/sandbox/t-code-2.3.1$ sudo dpkg -i ../t-code_2.3.1-3_all.deb
```

これで emacs23 や 24 といった新しい emacs で t-code の交ぜ書き、部首合成変換が問題なく使えるようになっていると思います。ちなみにこの emacsen-startup の elisp の内容は Debian パッケージのインストールによって/etc/emacs/site-start.d/50t-code.el にコピーされます。

37.5 おわりに

いかがでしたでしょうか。今回はとりあえず Debian パッケージを作るという目的でソースコードは変更を上書きしただけですが、本来は差分の変更などを記録する必要があるためまだまだやらねばならないことはあります。 upstream の t-code の継続開発を行なっている方、これまでの t-code Debian パッケージメンテナの方への連絡といったこともそうです。そういったことに関してはまた次回の関西 Debian 勉強会で発表させて頂ければと思っています。



38 月刊 t-code パッケージ修正

西田 孝三

第 56 回関西 Debian 勉強会では「emacs24 で問題なく使える t-code.deb を作った話」という題で発表しました。前回の発表ではとりあえず問題なく動作するオレオレパッケージを作成するところで終わっていました。今回は最近の Emacs でも動作するようにパッケージに加えた変更の管理をしっかりしていこうと思います。

38.1 前回のおさらいと今回の作業方針

前回とりあえず動くパッケージを作るために行ったことは派生リポジトリ (`http://code.google.com/p/tcode`) で変更が加えられたコードを既存のパッケージのソースに上書きコピーした上で build を行うというものでした。今後行うべきことのひとつとして、どのような変更を加えたかを管理していくことがあります。今後の変更を管理していく場合、下記のような方針が考えられます。

1. 派生リポジトリが今後の t-code パッケージの upstream だと言い張る。これまでの debian ディレクトリを派生リポジトリの方にコピーしてビルドが通るようにする。これは前回のオレオレパッケージ作成方法とは異なる。このままもしビルドが通れば patch を用意する必要は無い。
2. 既存の t-code パッケージの upstream のコードに直接変更を加える (前回はコピー上書きした) ことはせず、パッチ管理ツール (dpatch や quilt) を用いて変更を管理する。

1. には下記の理由があり、 2. の方針をとることにしました。

- 既存の t-code パッケージの debian ディレクトリをコピーしただけでは debuild が通らない。結局パッチ変更必要。
- Subversion の変更履歴の記録が他のソフトウェアも含むものとなっている (Windows 用の IME である漢直 Win のソースも共に管理している)

38.2 quilt を用いたパッチ作成

それでは前回行った変更に応じたパッチを作成してみましょう。

1. 既存 Debian パッケージのソース取得と派生リポジトリのソースを取得
2. 派生リポジトリのソースで既存 Debian パッケージのソースに変更を加え
3. debian/source/format を追加し quilt を使うことを宣言し
4. dpkg-source –commit コマンドでパッチを作成しています

```
kozo2@debian:~$ mkdir debsrc
kozo2@debian:~$ cd debsrc; apt-get source t-code
kozo2@debian:~/debsrc$ cd; svn co http://tcode.googlecode.com/svn/trunk/tc
kozo2@debian:~$ cp tc/bushu-util/* debsrc/t-code-2.3.1/bushu-util/
kozo2@debian:~$ cp tc/etc/* debsrc/t-code-2.3.1/etc/
kozo2@debian:~$ cp tc/lisp/* debsrc/t-code-2.3.1/lisp/
kozo2@debian:~$ cp tc/mazegaki/* debsrc/t-code-2.3.1/mazegaki/
kozo2@debian:~$ cd debsrc/t-code-2.3.1
kozo2@debian:~/debsrc/t-code-2.3.1$ mkdir debian/source
kozo2@debian:~/debsrc/t-code-2.3.1$ echo '3.0 (quilt)' > debian/source/format
kozo2@debian:~/debsrc/t-code-2.3.1$ dpkg-source --commit
```

これで debian/patches 下に series というファイルと共にパッチファイルが作成されます。パッチファイル名は先程のコマンド入力後に指定します。ここでは googlecode としました。 series という名のファイルには複数のパッチファイルを用いる場合にそれらのファイル名を記すようです。

```
kozo2@debian:~/debsrc/t-code-2.3.1$ ls debian/patches/
googlecode series
kozo2@debian:~/debsrc/t-code-2.3.1$ cat debian/patches/series
googlecode
```

38.3 ビルド時にパッチを用いるための設定

次に先程生成した debian/patches 以下のパッチファイルをバイナリパッケージをビルドする際にパッチを当てた状態になるように debian/rules の make コマンドの shebang の下に下記の 1 行を加えます。

```
kozo2@debian:~/debsrc/t-code-2.3.1$ head debian/rules
#!/usr/bin/make -f
include /usr/share/cdbs/1/rules/patchsys-quilt.mk

(省略)
```

この patchsys-quilt.mk は cdbs というパッケージを install すると用いることができます。 CDBS は Common Debian Build System の略で、 deb パッケージビルド用の debian/rules ファイルを短く書けるように rules で行うことの汎用的な内容の集まりのようです。今回はこの 1 行でうまく patch が当たり、ビルドできるかどうかまで試行を行いました。

```
kozo2@debian:~/debsrc/t-code-2.3.1$ debuild -us -uc
...
patching file lisp/eelll.el
Reversed (or previously applied) patch detected!  Skipping patch.
22 out of 22 hunks ignored -- saving rejects to file lisp/eelll.el.rej
patching file lisp/guess
Reversed (or previously applied) patch detected!  Skipping patch.
1 out of 1 hunk ignored -- saving rejects to file lisp/guess.rej
patching file lisp/Makefile.am
Reversed (or previously applied) patch detected!  Skipping patch.
1 out of 1 hunk ignored -- saving rejects to file lisp/Makefile.am.rej
patching file lisp/tc-bitmap.el
Reversed (or previously applied) patch detected!  Skipping patch.
2 out of 2 hunks ignored -- saving rejects to file lisp/tc-bitmap.el.rej
patching file lisp/tc-help.el
Reversed (or previously applied) patch detected!  Skipping patch.
10 out of 10 hunks ignored -- saving rejects to file lisp/tc-help.el.rej
patching file kinput2/Makefile.in
Reversed (or previously applied) patch detected!  Skipping patch.
3 out of 3 hunks ignored -- saving rejects to file kinput2/Makefile.in.rej
dpkg-source: error: LC_ALL=C patch -t -F 0 -N -p1 -u -V never -g0 -E -b -B .pc/midnight/ < t-code-2.3.1.orig.iRO91G/
debian/patches/midnight gave error exit status 1
dpkg-buildpackage: error: dpkg-source -b t-code-2.3.1 gave error exit status 2
```

どうも何らかの問題があり patch を当てることができず build に失敗しました。今回はこの問題の解決まではできませんでした。

38.4 おわりに

パッチの適用ができなかった問題については勉強会会場で参加者の方々に伺い、解決の糸口を得たいと思っています。他に懸念すべき事項としてライセンスがあるかと思いますが、ソースを通して見ても GPL v2 以外のライセンスを用いている部分は無いように思ったので後は変更管理さえしっかりできれば正式にパッケージを採用して頂くための連絡へと進めるかと思っています。次回の勉強会までに何とかそこまで持っていきたいと思います。

39 スクリプティング言語 Konoha の Debian パッケージ化について その 1

酒井 忠紀

39.1 はじめに

Konoha を Debian パッケージにして sid にコミットしたいと考えています。今回は、以下について説明します。

- Konoha の概要
- パッケージ化の内容確認
- スポンサーになってくださる方の相談

39.2 Konoha の概要

39.2.1 Konoha とは

Konoha は、静的型付けによるオブジェクト指向スクリプト言語です。横浜国立大学と JST/DEOS プロジェクトを中心にオープンソースで開発されています。[*99]

開発サイト: `http://konoha.sourceforge.jp`

Konoha は、静的言語の言語技術を基盤として、その上に動的な振る舞いをモデル化しています。
以下に記述するように、スクリプトとして実行することや、コンパイラ言語のように実行前に型検査を行うことができます。

- スクリプトとして実行
  関数を書き始めた時点では型やクラス階層は明確に決まっておらず、後から頻繁に書き換えて、アイデアが固まってから型やクラス階層を決めたい場合があります。
  Konoha は静的言語でありながら可能な限り動的言語の振る舞いをエミュレーションし、関数が最初に呼び出される直前にパラメータから型を推論し、遅延コンパイルします。
  したがって、静的言語でありながらスクリプト言語のように型定義などを省略して実行することができ、柔軟に開発を進めることができます。
- 実行前の型検査
  従来のスクリプト言語のように、動的な型検査をベースにしていると、型エラーが含まれるスクリプトでも実行しないとエラーが発見できません。

[*99] 私は Konoha プロジェクト内部の人間ではないので、以下の記述は正しくない場合があるかもしれません。

そのため、全ての実行パスをひとつひとつテスト実行しながら、型エラーを探し、修正する必要があります。
Konoha は静的な型付け言語であるため、プログラムを動作させることなく、型検査を行うことができ、初歩的なエラーを検証することができます。
そのため、ケアレスミスのために、ファイル操作やデータベースが中途半端な状態になり、プログラムが停止する弊害もありません。

Konoha を状況に合わせて使用方法を使い分けることで、ひとつの言語で静的言語とスクリプト言語の特徴を生かした開発を行うことができます。
これは、次のような場合に有効に活用できると考えます。
アプリケーション開発において、以下のような手順で行われる場合があると思います。

- STEP1: スクリプト言語を用いたプロトタイプの作成
  型宣言やクラス設計は後回しにして、俊敏にアイデアをコード化する
- STEP2: 静的言語で書き直し
  品質保証、メンテナンス性の向上

しかし、一旦静的言語で書き直してしまうと、スクリプト言語のように柔軟に書き直しができなくなります。
Konoha を使用すれば、上記の STEP1 と STEP2 を繰り替えして開発を進めることができます。
Konoha はスクリプト言語が提供する柔軟さと静的言語が持つ品質保証を両立させることを目指しています。

39.2.2 Konoha の使い方

1. 対話モード
   Konoha はターミナルから Konoha コマンドを実行すると、対話シェルとして起動します。

```
$ konoha
konoha 1.0(beta) svn (rev:933, Mar 23 2012 05:37:24)
options: iconv bmgc thcode sqlite3 syslog thread used\_memory:2191 kb
SECURITY ALERT: ** FOR EVALUATION/DEVELOPMENT USE ONLY **

>>>
```

   “>>>” は、対話シェルのプロンプトです。この状態で実行したいプログラムを入力すると、次の行に実行結果が表示されます。

```
>>> print "hello, Konoha!"
((eval):1) hello, Konoha!
```

   対話シェルを終了したい場合は、 “bye” もしくは (Conrol-D) を入力します。

```
>>> bye
```

2. スクリプトモード
   Konoha はプログラムをファイルに保存すれば、スクリプトとして実行できます。

```
$ cat <<EOF > fact.k
int factorial(int n) {
  if(n == 1) return 1;
  return n * factorial(n - 1);
}
print factorial(10);
EOF

$ konoha fact.k
(fact.k:5) 3628800
```

   また、-i オプションを付けて Konoha を起動すれば、スクリプトファイルを読み込んだ後、対話シェルから定義された関数を利用できます。

3. スクリプトの実行前の型検査
   型検査のみ実行する場合は、-c オプションを付けて Konoha を起動します。

```
$ cat <<EOF > type-error.k
s = 'ever';
i = 4;
str = i + s;    // 型推論
print str;      // "4ever" と評価される

int x = i + s;  // 型エラーとなる
print x;
EOF

$ konoha -c type-error.k
- (type-error.k:1) (info) suppose s has String type
- (type-error.k:2) (info) suppose i has int type
- (type-error.k:3) (info) suppose str has String type
- (type-error.k:6) (error) x has type int, not String
- (type-error.k:7) (error) undefined variable: x
```

39.2.3 Konoha の特徴

Konoha の主な特徴を以下に記述します。

1. 静的型付け
   型の妥当性が保証されています。スクリプト実行前に型検査のみ行うことができます。
2. 型宣言の省略 (型推論や遅延コンパイル)
   以下の場合、最初に呼び出されるのが dsucc(1.0) なので引数と戻り値は float 型に決定されます。

```
function dsucc(n) {
  return n + 1;
}

dsucc(dsucc(1.0) + 1);
```

   Konoha は一旦、型が決定したら型の一貫性を保証します。つまり、次に `dsucc("Konoha")` のような異なる型の呼び出しは型エラーとなります。これにより静的言語と同じように型エラーの管理ができます。
3. 制御された動的型付け (dynamic type)
   Konoha はスクリプト言語のようなダックタイピングをサポートしています。
   以下のように、 dynamic 型を用いると、 + 演算子をサポートした任意の値を処理できます。

```
function dsucc(dynamic n) {
  return n + 1;
}
```

   + 演算子をサポートしていない値が与えられると、実行時のエラーとなります。 dynamic 型を宣言する/しないでエラーの発生を制御できます。
4. 軽量なオブジェクト
   Konoha はコンストラクタがなくてもオブジェクトの生成ができます。また、 setter/getter は自動的に生成されます。
5. 汎用性の高い集約データ構造 (Array, Map, Tuple)

```
Array(リスト)   ["naruto", 17]
Map(辞書)       {name:"naruto", age:17}
Tuple          ("naruto", 17)
```

6. 実行時のコンパイルと評価 (eval)
   スクリプト言語として実行できます。
7. 実行時のクラスの改変
   メソッドは、既にオブジェクトが生成してあっても、実行時にクラスに追加することができます。
8. 不完全コードの部分的な実行

従来のコンパイラ技術では、実行前に型検査を行うため、次のような型エラーがある場合、コード生成が完了せず、プログラムも実行できませんでした。

```
int serial(int n) {
  if(n == 0) {
    InputStream in = new ("serial.txt");
    n = in.readLine(); // 型エラーがここにある
    in.close();
  }
  return n + 1;
}
```

Konoha では、コンパイル時にエラーが発生しても、その部分を実行時例外に書き換えることができます。例えば、上記の型エラーは次のコードに変換されます。

```
int serial(int n) {
  if(n == 0) { // 例外になる
    throw new Script!!("");
  }
  return n + 1;
}
```

エラー箇所を実行しなければ、そのまま実行が可能であり、もしも serial(0) を実行しても、実行時例外として処理できます。

```
>>> serial(1)
2

>>> serian(0)
Script!!:
```

9. 統一的なデータ変換操作 Konoha には、キャスト演算子を拡張したトランスキャットと呼ばれるデータ変換を統一操作で行う演算子があります。

```
>>> (to String)1
"1"

>>> (to int)1
1

>>> (to int)"naruto"
null
```

文字列への変換は、フォーマッタの機能を用いると、様々な書式の文字列に変換できます。

```
>>> "%x"(-1)
"ffffffffffffffff"

>>> "%bits"(-1)
"11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111"
```

他にも色々な変換機能があります。

10. リンク演算子と外部リソースの扱い

Konoha には外部リソースの識別子を直接扱う URN があります。

http://konohascript.org/

file:/etc/passwd

isbn:978-4-06-372988-7

また、 Konoha は URN から直接リソースを得ることができます。 “file:” や “http:” などの URN スキームは、ストリームに型強制されます。 URN を用いれば、次のように書けます。

```
foreach(String line from file:/etc/hosts) {
  print line;
}
```

11. 非共有データモデル (アクター) による並行処理

申し訳ないですが、この部分は私の勉強不足で説明できません。

12. ライブラリ

Konoha では、パッケージ化された以下のようなライブラリを使用することができます。

```
konoha.cairo konoha.compiler konoha.compiler.cpp konoha.compiler.java konoha.compiler.js konoha.compiler.optllvm konoha.curl
konoha.dffi konoha.dscript konoha.gsl konoha.gwt konoha.i konoha.io konoha.json konoha.kinect konoha.lang konoha.liboauth
konoha.llvm konoha.math konoha.memcached konoha.mpi konoha.nfc konoha.ntrace konoha.opengl konoha.posix konoha.proc
konoha.qt4 konoha.qt4.kinect konoha.qt4.opencv konoha.qt4.physics konoha.signal konoha.socket konoha.sql konoha.sugar
konoha.thread konoha.xml
```

*100

ライブラリを使用するときは、 using 文を使用します。以下に Math ライブラリを呼び出した例を記述します。

```
>>> using konoha.math.Math;
>>> Math.PI 3.141593
>>> Math.pow(1.23, 3.45)
2.042550
```

13. 性能

    JIT コンパイラや LLVM コンパイラの採用も進められており、「 世界最高水準と誇れる性能」を記録しているらしいです。

39.3 パッケージ化の内容確認

39.3.1 開発環境

- マシン

  CPU : Intel(R) Core(TM) i3-2367M CPU @ 1.40GHz

  メモリ: 4GB
- OS

  Kernel : Linux 3.2.0-0.bpo.2-amd64

  Userland : Debian sid (cowbuilder)

39.3.2 upsterem からの承認

承認を頂いております。

39.3.3 ライセンス

Konoha は GPLv3 です。

Konoha core は、 build-essential 以外の依存パッケージは必要ないのですが、 Konoha Extra Package は、それぞれ依存パッケージが必要になります。*101

/usr/share/doc/*/copyright から、関連する依存パッケージのライセンスを表 26 にまとめました。*102

- Q1: Konoha core は、 GPLv3 で問題ないと認識してよいでしょうか?

  dynamic link するライブラリがどのようなライセンスでも問題ない?
- Q2: Konoha extra Package は GPLv3 以外のライブラリに依存するので、別の Debian パッケージにする必要があるのでしょうか?

39.3.4 Debian パッケージ化の方針

とりあえず現状はシングルパッケージにしています。

本来は、以下のような 3 種類に分ける必要があることは理解しています。

*100 Subversion Revision 954 時点のものです。 Linux 版において、全てがビルド/動作することを確認していません。

*101 Konoha Extra Package は dynamic link library と Konoha スクリプトのラッパーから構成されています。

*102 libmemcached-dev,libcurl4-nss-dev:ファイルにより異なるみたい

表 26　関連する依存パッケージのライセンス

パッケージ名	ライセンス
libffi-dev	GPLv2 or later
libmemcached-dev	RSA Data Security License, Public Domain, BSD-TangentOrg, BSD-Sun, BSD
libsqlite3-dev	public domain
libqt4-dev	LGPLv2.1, GPLv2, GPLv3
libqt4-opengl-dev	LGPLv2.1, GPLv2, GPLv3
libqtwebkit-dev	LGPLv2 or later
libcairo2-dev	LGPLv2, MPLv1.1
libopenmpi-dev	LGPLv2
libjson0-dev	MIT
libcurl4-nss-dev	curl, BSD-4-Clause, BSD-3-Clause, ISC
libxml2-dev	MIT
libreadline-dev	GPLv3 or later

1. core
2. libray(extra package)
3. その他 (document and sample など)

- Q1: ライブラリはどのような精度で分割するのがよいでしょうか?
  - 各ライブラリごと
  - グループ分け (Graphic や DB など)
- Q2: 64bit 版と 32bit 版の切り分けは?

39.3.5　パッケージ名

konoha-1.0.0~954

1.0.0 は upstream のバージョン、954 はパッケージ化したときの Subversion Revision です。

39.3.6　オリジナルのビルド手順

```
$ sudo apt-get install cmake libffi-dev libmemcached-dev libsqlite3-dev libqt4-dev libqt4-opengl-dev libqtwebkit-dev \
libcairo2-dev libopenmpi-dev libjson0-dev libcurl4-nss-dev libxml2-dev libreadline-dev openjdk-6-jdk ant
$ svn export http://konoha.googlecode.com/svn/trunk/ konoha-read-only
$ cd konoha-read-only/konoha/build/
$ cmake ../ -DCMAKE_INSTALL_PREFIX=/usr \
-DMPI_ROOT_DIR=/usr/lib/openmpi -DUSE_QT4=ON -DK_REVISION=954 2>&1 | tee cmake.log
$ make 2>&1 | tee make.log
$ mkdir tmp
$ DESTDIR=tmp make install 2>&1 | tee make-install.log
```

- Q1: Konoha Extra Package のインストール先は、/usr/konoha 以下ではなく、/usr/lib/konoha 以下に修正する必要はありますでしょうか? lintian のチェックで warning になります。

39.3.7　環境変数の設定

```
$ export DEBEMAIL="stadaki.dev@gmail.com"
$ export DEBFULLNAME="Tadaki SAKAI"
```

39.3.8 制御ファイルのテンプレート作成

```
$ svn export http://konoha.googlecode.com/svn/trunk/ konoha-read-only
$ konoha-read-only
$ tar cvfz konoha.tar.gz konoha
$ mv konoha konoha-1.0.0~954
$ cd konoha-1.0.0~954
$ dh_make --single --copyright gpl3 --file=../konoha.tar.gz
$ rm -f debian/*.ex
$ rm -f debian/*.Ex
```

39.3.9 debian/control ファイル修正

```
Source: konoha
Section: interpreters
Priority: optional
Maintainer: Tadaki SAKAI <stadaki.dev@gmail.com>
Build-Depends: debhelper (>= 7.0.50~), cmake, libffi-dev, libmemcached-dev,
  libsqlite3-dev, libqt4-dev, libqt4-opengl-dev, libqtwebkit-dev, libcairo2-dev,
  libopenmpi-dev, libjson0-dev, libcurl4-nss-dev, libxml2-dev, openjdk-6-jdk, ant,
  libreadline-dev
Standards-Version: 3.8.4
Homepage: http://konoha.sourceforge.jp/
Vcs-Svn: http://konoha.googlecode.com/svn/trunk/
Vcs-Browser: http://code.google.com/p/konoha/downloads/list

Package: konoha
Architecture: amd64
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: statically-typed scripting language
 Konoha scripting language has a Java-like syntax, multiplatform
 virtual machine, and static typing system.
```

39.3.10 debian/copyright ファイル修正

```
Format: http://dep.debian.net/deps/dep5
Upstream-Name: konoha
Source: http://konoha.sourceforge.jp/

Files: *
Copyright: Kimio Kuramitsu <kkuramitsu@gmail.com>
           Shinpai Nakata <shinpei.nakata@gmail.com>
           Masahiro Ide <masa.ide.on@gmail.com>
License: GPL-3.0+

Files: debian/*
Copyright: 2012 Tadaki SAKAI <stadaki.dev@gmail.com>
License: GPL-3.0+

License: GPL-3.0+
 This program is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.
 .
 This package is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU General Public License for more details.
 .
 You should have received a copy of the GNU General Public License
 along with this program. If not, see <http://www.gnu.org/licenses/>.
 .
 On Debian systems, the complete text of the GNU General
 Public License version 3 can be found in "/usr/share/common-licenses/GPL-3".

# Please also look if there are files or directories which have a
# different copyright/license attached and list them here.
```

39.3.11 debian/rules ファイル修正

```
#!/usr/bin/make -f
# Sample debian/rules that uses debhelper.

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

DESTDIR=$(CURDIR)/debian/konoha

clean:
	dh_testdir
	dh_auto_clean
	dh_clean
	rm -rf configure-stamp build-stamp
	rm -rf $(DESTDIR)
	rm -f debian/files

configure: configure-stamp
configure-stamp:
	dh_testdir
#	 dh_auto_configure
	cd build && cmake ../ -DCMAKE_INSTALL_PREFIX=/usr -DMPI_ROOT_DIR=/usr/lib/openmpi -DUSE_QT4=ON -DK_REVISION=954
	touch $@

build: configure build-stamp
build-stamp:
	dh_testdir
#	 dh_auto_build
	cd build && $(MAKE)
#	 dh_auto_test
	touch $@

binary: binary-arch binary-indep

binary-arch:
	dh_testdir
	dh_testroot
	dh_prep
	dh_installdirs
#	 dh_auto_install
	cd build && $(MAKE) install prefix= DESTDIR=$(DESTDIR)
	dh_install
	dh_installdocs
	dh_installchangelogs
	dh_installexamples
	dh_installman
#	 dh_installcatalogs
#	 dh_installcron
#	 dh_installdebconf
#	 dh_installemacsen
#	 dh_installifupdown
	dh_installinfo
#	 dh_installinit
#	 dh_installmenu
#	 dh_installmime
#	 dh_installmodules
#	 dh_installlogcheck
#	 dh_installlogrotate
#	 dh_installpam
#	 dh_installppp
#	 dh_installudev
#	 dh_installwm
#	 dh_installxfonts
#	 dh_bugfiles
#	 dh_lintian
#	 dh_gconf
#	 dh_icons
#	 dh_perl
#	 dh_usrlocal
	dh_link
	dh_compress
	dh_fixperms
	dh_strip
	dh_makeshlibs
	dh_shlibdeps
	dh_installdeb
	dh_gencontrol
	dh_md5sums
	dh_builddeb

binary-indep:
```

39.3.12 Debian パッケージの作成

```
$ dpkg-buildpackage -us -uc
```

親ディレクトリに以下が作成される

```
konoha_1.0.0~954-1.debian.tar.gz
konoha_1.0.0~954-1.dsc
konoha_1.0.0~954-1_amd64.changes
konoha_1.0.0~954-1_amd64.deb
konoha_1.0.0~954.orig.tar.gz
```

39.3.13 動作確認

1. lintian

```
$ lintian konoha_1.0.0~954-1_amd64.deb
W: konoha: package-name-doesnt-match-sonames libkonoha1.0
W: konoha: new-package-should-close-itp-bug
W: konoha: wrong-bug-number-in-closes l3:#nnnn
E: konoha: copyright-contains-dh_make-todo-boilerplate
W: konoha: readme-debian-contains-debmake-template
W: konoha: non-standard-dir-in-usr usr/konoha/
W: konoha: file-in-unusual-dir usr/konoha/package/1.0/js.dom/dom.k
( 中略)
W: konoha: file-in-unusual-dir usr/konoha/script/1.0/actsrv
W: konoha: file-in-unusual-dir usr/konoha/script/1.0/actsrv2
W: konoha: file-in-unusual-dir usr/konoha/script/1.0/mailbox.k
W: konoha: file-in-unusual-dir usr/konoha/script/1.0/man
W: konoha: file-in-unusual-dir usr/konoha/script/1.0/status
W: konoha: jar-not-in-usr-share usr/konoha/package/1.0/konoha.compiler.java/jkonoha.jar
W: konoha: binary-without-manpage usr/bin/jkonoha
W: konoha: binary-without-manpage usr/bin/konoha
W: konoha: binary-without-manpage usr/bin/konoha2js
W: konoha: binary-without-manpage usr/bin/konohac
W: konoha: binary-without-manpage usr/bin/mpikonoha
W: konoha: non-dev-pkg-with-shlib-symlink usr/lib/libkonoha.so.1.0.0 usr/lib/libkonoha.so
```

以下の対応が必要だと考えています。

（ a） ITP して changelog に number を記述する

（ b） Konoha Extra Pacage のインストール先を /usr/konoha から/usr/lib/konoha に変更する

（ c） TODO, manpage を作成する

2. ビルド時の依存チェック

```
$ sudo pbuilder build konoha_1.0.0~954-1.dsc \
--basetgz /var/cache/pbuilder/base-amd64.tgz
```

/var/cache/pbuilder/result/ 以下に Konoha のバイナリ/ソースパッケージが作成されることを確認しました。

40 スクリプティング言語 Konoha の Debian パッケージ化について その 2

酒井 忠紀

40.1 はじめに

前回は Konoha の概要説明とその Debian パッケージの内容確認をしました。今回は、以下について説明します。

- upstream との調整経緯
- debian/rules 修正とパッケージ分割について

40.2 upstream との調整経緯

40.2.1 ライセンス問題について

前回、ご指摘を頂いた以下のライセンス問題を upstream の方に報告しました。

- Web サイトでは GPLv3 と記述されているが、配布物の COPYING ファイルは LGPLv3 になっている
- 配布物の中に “third-party” というディレクトリがあり、その中に jar ファイルや Apache Linense 2.0 ライセンスのソースの tar アーカイブがある

現在、ライセンスに関しては upsteam 側で再検討を行っているようです。

また、“Konoha Non-Disclosure License 1.0” とは、有償サポート付きのライセンスを検討していたということでした。

質問

1. ユーザの視点から、扱いやすいライセンスの組み合わせなどはありますでしょうか?
   GPLv2 or Later と New BSD のデュアルライセンスなど
2. 言語によって相性の良いライセンスなどがありますでしょうか?
   例えば、 Qt は言語バインディングによって以下のようにライセンスが異なるようです。
   - Ada ... GPL
   - C++ ... LGPL
   - C# & .NET(qt4dotnet) ... LGPL
   - Java ... LGPL
   - Lisp ... BSD
   - Lua ... MIT
   - Perl ... GPL
   - PHP ... LGPL
   - Python(PyQt) ... GPL
   - Ruby ... LGPL
   - Tcl ... GPL

40.2.2 upstream の Konoha 開発状況について

現在公開されている Konoha は Konoha 1.0 ですが、 upsteam では試作的な扱いとなっているようです。

正規のものは現在開発中の Konoha 2.0 になり、 6 月頃のリリースを目標に作業が進められているようです。

upstream において、 Konoha 1.0 はもうサポートする気がないようなので、 ITP するのは Konoha 2.0 がリリースされてからにしようと考えています。

また、 Debian 7.0(Wheezy) への新規パッケージの取り込みは 6 月で閉じてしまうので、時期的に断念し、次の Debian 8.0? を目標にしようと考えています。

40.2.3 Konoha 2.0 について

現在開発中の Konoha 2.0 について、特徴を簡単に説明します。

1. 最小限の構文
   if, int, String, void, boolean, array, 関数定義 のみサポートする。 (代入がない)
   POSIX lowlevel bind と呼ばれている。
2. Konoha Assignment
   言語のシンタックスをスクリプトで追加できる。
3. ライブラリ
   既存の Konoha 1.0 の機能は、ライブラリとして提供する。
   必要な機能のみ、インポートして使用できる。
4. 言語バインディング
   C, Java, JavaScript, C#, その他を検討している。
5. ドキュメントの自動生成

Konoha 2.0 は、以下に記述する 4 つのモジュールから構成されます。

- konoha (コマンド本体)
- sugar (パーサー)
- gc (Garbage Collection)
- vm (Virtual Machine)

バイナリサイズが 約 1/10 (100KB) になったので、組み込み機器やアプリケーションなどに容易に組み込むことが可能になるようです。

40.3 debian/rules 修正とパッケージ分割について

Konoha 2.0 はまだ公開されていないので、 Konoha 1.0 のままになりますが、前回指摘を受けた以下の 2 つを修正しました。

40.3.1 debian/rules 修正

前回は debhelper を使用して記述していましたが、今回は dh を使用して記述し直しました。
記述内容をかなり省略することができ、行数が 43 行 から 7 行 になりました。(コメントと空行は除く)
修正した debian/rules を以下に記述します。

```
#!/usr/bin/make -f
# -*- makefile -*-
# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1

%:
	dh $@ --buildsystem cmake --builddirectory=build

override_dh_auto_configure :
	dh_auto_configure -- -DCMAKE_INSTALL_PREFIX=/usr -DMPI_ROOT_DIR=/usr/lib/openmpi -DUSE_QT4=ON -DK_REVISION=961

override_dh_auto_test :
```

–buildsystem オプションで、ビルドシステムに cmake を指定しました。前回は configure ターゲットを自前で記述していました。
ビルドシステムは、以下が指定できるようです。

- autoconf ... GNU Autoconf (configure)
- perl_makemaker ... Perl MakeMaker (Makefile.PL)
- makefile ... simple Makefile
- python_distutils ... Python Distutils (setup.py)
- perl_build ... Perl Module::Build (Build.PL)
- cmake ... Cmake (CmakeLists.txt)
- ant ... Ant (build.xml)

–builddirectory オプションで、ビルドディレクトリを指定しました。前回は configure, build, clear, install ターゲットで、'cd build &&' を記述していました。
override_dh_auto_configure ターゲットで、 cmake のオプションをオーバライドしました。
override_dh_auto_test ターゲットで、 make test を実行しないようにしました。
debian/rules の書き方は、以下を参考にしました。
http://www.debian.org/doc/manuals/packaging-tutorial/packaging-tutorial.pdf
http://kitenet.net/~joey/talks/debhelper/debhelper-slides.pdf

40.3.2 パッケージ分割

前回はシングルパッケージにしていましたが、 Debian ポリシーに従い、以下の 3 つのパッケージに分割しました。

- konoha パッケージ (実行バイナリ)
  konoha_1.0.0+svn961-1_amd64.deb
- libkonoha1 パッケージ (ライブラリ + シンボリックリンク (SONAME))
  libkonoha1_1.0.0+svn961-1_amd64.deb

- konoha-dev パッケージ (ヘッダファイル + シンボリックリンク)
  konoha-dev_1.0.0+svn961-1_amd64.deb

konoha パッケージ と konoha-dev パッケージは、 libkonoha1 パッケージに依存するように定義しました。
修正した debian/control を以下に記述します。 ruby1.9.1 のソースパッケージを参考にしました。

```
Source: konoha
Section: interpreters
Priority: optional
Maintainer: Tadaki SAKAI <stadaki.dev@gmail.com>
Build-Depends: debhelper (>= 7.0.50~), cmake, libffi-dev, libmemcached-dev, libsqlite3-dev, libqt4-dev,
  libqt4-opengl-dev, libqtwebkit-dev, libcairo2-dev, libopenmpi-dev, libjson0-dev, libcurl4-nss-dev,
  libxml2-dev, openjdk-6-jdk, ant, libreadline-dev
Standards-Version: 3.9.3
Homepage: http://konoha.sourceforge.jp/
Vcs-Svn: http://konoha.googlecode.com/svn/trunk/
Vcs-Browser: http://code.google.com/p/konoha/downloads/list

Package: konoha
Architecture: amd64
Depends: libkonoha1 (= ${binary:Version}), ${shlibs:Depends}, ${misc:Depends}
Suggests: konoha-dev
Description: Interpreter of statically-typed scripting language Konoha
 Konoha scripting language has a Java-like syntax, multiplatform
 virtual machine, and static typing system.

Package: libkonoha1
Section: libs
Architecture: amd64
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: Libraries necessary to run Konoha
 Konoha scripting language has a Java-like syntax, multiplatform
 virtual machine, and static typing system.

Package: konoha-dev
Architecture: amd64
Depends: libkonoha1 (= ${binary:Version}), ${shlibs:Depends}, ${misc:Depends}
Recommends: konoha (= ${binary:Version})
Description: Header files for compiling extension modules for the Konoha
 Konoha scripting language has a Java-like syntax, multiplatform
 virtual machine, and static typing system
```

konoha パッケージ でインストールするファイルは debian/konoha.install で定義しました。
debian/konoha.install を以下に記述します。

```
debian/tmp/usr/bin/*
```

libkonoha1 パッケージ でインストールするファイルは debian/libkonoha1.install で定義しました。
debian/libkonoha1.install を以下に記述します。

```
debian/tmp/usr/lib/libkonoha.so.1.0
debian/tmp/usr/lib/libkonoha.so.1.0.0
debian/tmp/usr/konoha/*
```

konoha-dev パッケージ でインストールするファイルは debian/konoha-dev.install で定義しました。
debian/konoha-dev.install を以下に記述します。

```
debian/tmp/usr/include/*
debian/tmp/usr/lib/libkonoha.so
```

40.4 Konoha 1.0 Debian パッケージの作成手順

sid で以下の実行する。

```
$ sudo apt-get install cmake libffi-dev libmemcached-dev \
libsqlite3-dev libqt4-dev libqt4-opengl-dev libqtwebkit-dev \
libcairo2-dev libopenmpi-dev libjson0-dev libcurl4-nss-dev \
libxml2-dev libreadline-dev openjdk-6-jdk ant
$ svn export http://konoha.googlecode.com/svn/trunk/ konoha-read-only
$ cd konoha-read-only
$ tar cvfz konoha.tar.gz konoha
$ mv konoha konoha-1.0.0+svn961
$ cd konoha-1.0.0+svn961
$ dh_make --copyright gpl3 --file=../konoha.tar.gz
```

前章に記述した内容で、以下のファイルを編集する。
(copyright, changelog に関しては、第 57 回関西 Debian 勉強会の資料を参照。)

```
  debian/rules
  debian/control
  debian/konoha.install
  debian/libkonoha1.install
  debian/konoha-dev.install
  debian/copyright
  debian/changelog
```

ビルドを実行。

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

親ディレクトリに以下が作成される。

```
libkonoha1_1.0.0+svn961-1_amd64.deb
konoha-dev_1.0.0+svn961-1_amd64.deb
konoha_1.0.0+svn961-1_amd64.deb
konoha_1.0.0+svn961-1.dsc
konoha_1.0.0+svn961.orig.tar.gz
konoha_1.0.0+svn961-1.debian.tar.gz
konoha_1.0.0+svn961-1_amd64.build
konoha_1.0.0+svn961-1_amd64.changes
```

lintian では、まだ Warning などが出力されています。ITP number や、 man ファイル、 Konoha Extra Package の格納位置などを修正する必要があります。

41 月刊 Debian Policy 第1回「パッケージの依存関係についてのルール」

倉敷 悟

分量も多くてなかなか読み辛いのが Debian Policy。敬遠している方もいると思います。とはいえ、「ポリシー」と大仰な名前がついてはいますが、結局のところ書かれている内容は

「パッケージの作り方についてのルール、ガイド、ベストプラクティス」

を集めたものです。パッケージの開発をしていないとしても、ファイルの配置や依存関係に不可解なものを感じたことがあれば、その答えがきっと見つかるはずです。

というわけで、勉強会で連続コマとして少しずつ読んでいってみることにします。いろいろ含めて、おおよそ 1 年かけて終了することになると思います。担当した人は、まだ担当していない人から次回の担当者を指名します。指名された人は、まだ読まれていない章から順不同で好きな部分を選んでください。

さて、今回読むのは、パッケージの依存関係のポリシーについて記載されている 7 章です。

依存関係といって思い浮かぶのは Depends に代表される「前提として事前に必要」ですが、dpkg では他にも色々な形で「そのパッケージと他のパッケージの関係」を表現することができます。

さて、ポリシー本文は事前課題として当然[*103]読み終わっているはずですが、簡単におさらいしていきましょう。空いた時間は四方山話として、フリーディスカッションのような形で、皆さんが舐めてきた辛酸を肴にして頂こうと思います。

41.1 おさらい

41.1.1 debian/control の書式

依存関係の指定は、debian/control ファイルに、依存関係の種類に応じたフィールドとその値をセットすることで行います。

```
Depends: libqdbm14, libestraier
```

必要があれば、カンマで区切って複数のパッケージ名を列挙することができます。これにバージョン指定が入ると、

```
Depends: libqdbm14 (>> 1.8.77 ), libestraier
```

となります。この場合、libqdbm14 の 1.8.77 では依存を満たせません。

同類がいくつかあって、そのうちどれかがあればよい、という場合はパイプ (|) を使って

```
Depends: emacs23 | emacs22 | emacs21
```

と書くことができます。

[*103] 実は事前課題を指定した時点では、日本語訳の存在をすっかり忘れていました

41.1.2 バイナリパッケージ用フィールド

バイナリパッケージでは、そのパッケージをインストールするために必要な情報として依存関係が使われます。まずは基本的なものから。

Depends　なくては動かないパッケージを指定します。
Recommends　なくても動きはするけれど、できれば一緒に使った方がいいパッケージを指定します。
Suggests　一緒に使うと便利なパッケージを指定します。主体はこのパッケージです。
Enhances　一緒に使うと便利なパッケージを指定します。主体は指定されたパッケージです。
Pre-Depends　通常の Depends よりも前の段階で依存先のパッケージがチェックされます。非推奨なので、使われている場合は何か地雷があるのかもしれません

一方で、少しややこしい関係性として、次のようなものがあります。

Breaks　比較的最近追加されました。このパッケージをインストールすることで動作しなくなるパッケージを指定します。通常は、自パッケージの再編成で使われるようです
Conflicts　完全に両立不可能なパッケージを指定します。システムにはそのうち 1 つしかインストールできません
Replace　指定したパッケージ全体、もしくは一部のファイルを、このパッケージで置き替えます
Provides　(存在する場合) このパッケージが代替し得る、特定の仮想パッケージを指定します

41.1.3 ソースパッケージ用フィールド

ソースパッケージは、いわゆるインストールをして使うものではないので、 Depends は意味を持ちません。ソースパッケージの依存関係では、そのソースパッケージをビルドするために必要な情報として依存関係が使われます。

Build-Depends　ビルドするために必要なパッケージを指定します
Build-Conflicts　ビルドする時にインストールされていてはいけないパッケージを指定します
Build-Depends-Indep　build-indep ターゲットをビルドするために必要なパッケージを指定します
Build-Conflicts-Indep　build-indep ターゲットをビルドする時にインストールされていてはいけないパッケージを指定します

41.2 四方山話

時間次第ですが、ここでは次のような話をざっくりとする予定です。

- Depends と Recommends と Suggests の違い
- Provides と仮想パッケージ

会場で特にこれといって話題が出なさそうであれば、抜きうちで事前課題の理解度テスト、みたいなことをするかも知れません。

42 月刊 Debian Policy 第2回「Control ファイルについて」

八津尾 雄介

先月倉敷さんの指名で今回の担当となりました八津尾です。今回はパッケージ作成の要である *control* ファイルについてです。詳細は Debian ポリシーマニュアルを読めばわかるはずですので、あまりだらだらと説明せずに概要のみの説明とさせていただきます。

42.1 Debian Policy 3.9.3.0 での変更点

先頃 *Debian Policy 3.9.3.0* がリリースされ、著作権表記のフォーマットを中心にいくつかの変更がありました。現在の日本語訳 3.9.1.0 と若干変わっていますので、まずはその変更点を抑えておきましょう。
第 5 章に関しての変更点は、5.6.8 の **.dsc* ファイルについての変更のみとなっています。*Archtecture* フィールドにアーキテクチャに依存するしないに関わらず *"any all"* という値を指定できるようになりました。

> "Specifying *any all* indicates that the source package isn't dependent on any particular architecture. The set of produced binary packages will include at least one
> architecture-dependant package and one architecture-independent package."
> *Debian Policy Manual. version 3.9.3.1, Chapter 5.6.8 "Architecture": Ian Jackson and Christian Schwarz, 2012-03-04*
>
> "*any all* を指定すると、ソースパッケージが特定のアーキテクチャに依存しない事を意味します。生成されたバイナリパッケージには、アーキテクチャ依存パッケージとアーキテクチャ非依存パッケージが、少なくとも 1 つずつ含まれます。" (訳: 八津尾)

また、先月の第 7 章 (7.1) に関しては、

> **"もし特定のアーキテクチャに依存している場合は、アーキテクチャのリストは空白にしてはならない"**

といった内容が追加されておりますので、日本語訳のみしか読んでいない方は確認をしておきましょう。その他の変更点については、それぞれの章の担当者にお任せします。

42.2 debian/control ファイル

`control` ファイルとはパッケージのメタ情報を扱うファイルです。`debian/ control` ファイルは 2 つの段落から成っており、最初の段落が全般的な情報を扱う *"general paragraph"*、次の段落がバイナリパッケージで使用する情報を扱う *"binary package paragraph"* と呼ばれています。`control` ファイルの書式は非常に単純です。

フィールド名: フィールド値

と記述するだけです。細かいルールは *debian policy* 第 5 章を読みましょう。

$dh_make -f (パッケージ名)

というコマンドを使えば `debian/` 以下に必要なファイルが作成されます。以下は私が *lpc21isp* という *arm* マイコン書込み用 *ISP* ツールをパッケージ化してみた時に、 *dh_make* が自動的に生成した `control` ファイルです。 `dh_make` コマンドを使えば、いくつかの質問に答えるだけで、 `control` ファイルを以下の状態まで持っていく事ができます。 `dh_make` についての詳細は *dh_make(8)* をご参照下さい。 `dh_make` を使ったパッケージ化の大まかな流れは “Debian パッケージ化入門”[1] または “新メンテナガイド”[2] が参考になります。

debian/control ファイルの例

```
Source: lpc21isp
Section: utils
Priority: optional
Maintainer: Your Name <mail@address.here>
Build-Depends: debhelper (>= 8.0.0)
Standards-Version: 3.9.2
Homepage: http://www.aeolusdevelopment.com

Package: lpc21isp
Architecture: any
Depends:
Description: Portable command line ISP
 Portable command line ISP for NXP LPC1000 / LPC2000 family
  and Analog Devices ADUC70xx.
```

ここでは使用されていないフィールド名もありますので、フィールド名の一覧は 5.2 章を、各フィールドが取る値とその意味については 5.6 章を参照して下さい。

42.3 DEBIAN/control ファイル

`debian/(`**パッケージ名**`)/DEBIAN/` 以下にも *control* ファイルが存在します。このファイルは先述の `debian/control` ファイルを元に `dh_gencontrol(1)` が生成します。詳細は *man* を参照して下さい。

$man dh_gencontrol

先に示した *control* ファイルからは以下のような *DEBIAN/control* ファイルが生成されます。

DEBIAN/control ファイルの例

```
package: lpc21isp
Version: 1.8.3-1
Architecture: i386
Maintainer: Your Name <main@address.here>
Installed-Size: 588
Section: utils
Priority: optional
Homepage: http://www.aeolusdevelopment.com
Description: Portable command line ISP
 Portable command line ISP for NXP LPC1000 / LPC2000 family
 and Analog Devices ADUC70xx.
```

42.4 *.dsc ファイル

debuild などを実行すると *.deb ファイルと同じディレクトリに *.dsc ファイルが生成されます。書式は control ファイルと同じ形態で、 control ファイルを元に生成します。 *dpkg-source(1)* によって、ソースを展開する時に使われます。パッケージの展開時に整合性のチェックをします。

42.5 *.changes ファイル

.changes ファイルは Debian アーカイブを管理するソフトウェアによって使用されます。このファイルは debian/control、 debian/changelog、 debian/rules などから抽出したソースパッケージの情報が含まれています

参考文献

[1] パッケージ “packaging-tutorial”

[2] 新メンテナガイド http://www.debian.org/doc/manuals/maint-guide/index.ja.html

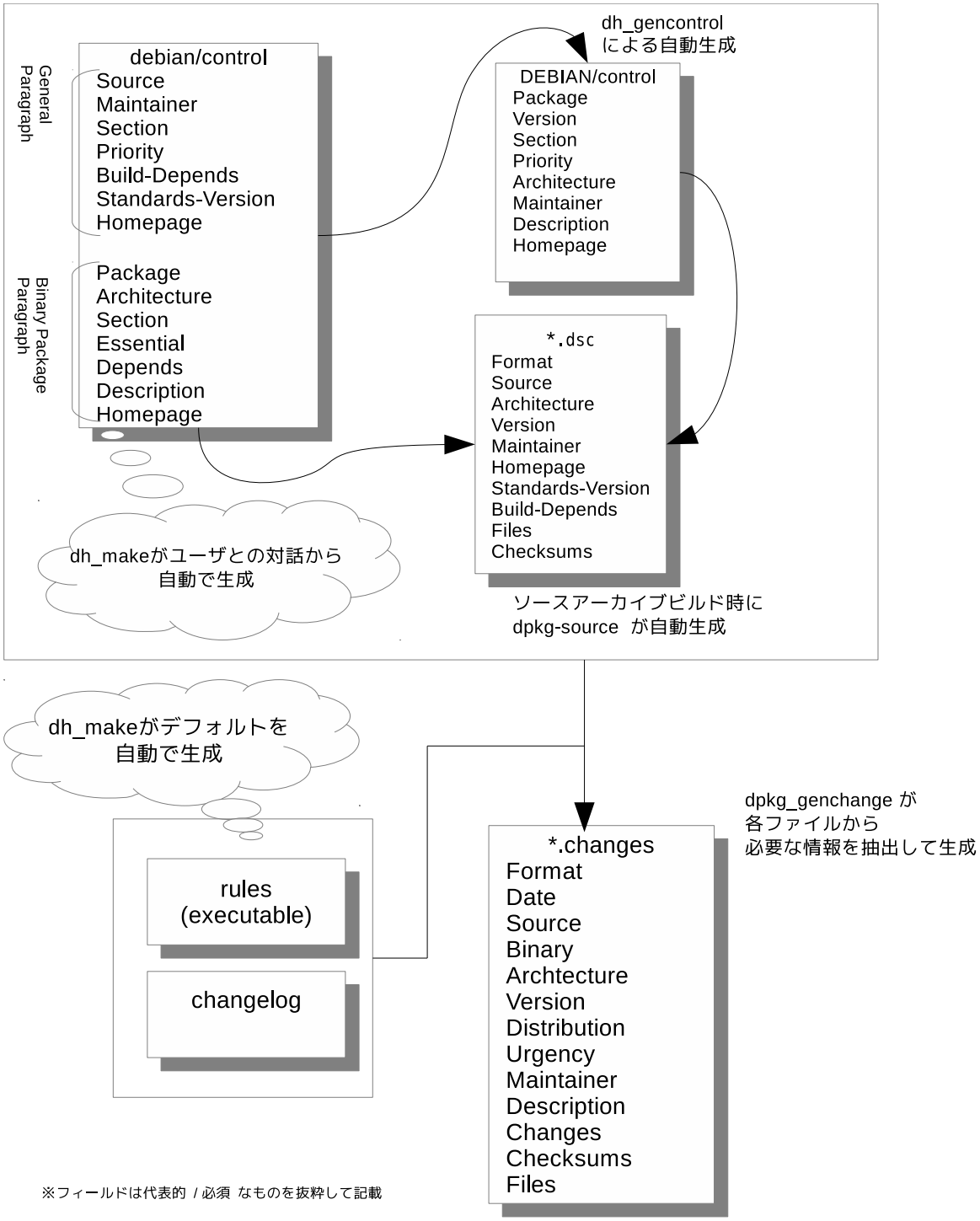


図 35　各ファイルの関係

43 月刊 Debian Policy 第3回「Debian アーカイブ」

かわだ てつたろう

諸般の事情で今回の担当となりました かわだ です。

今回読むのは第 2 章の「Debian のアーカイブ」についてです。Debian Policy はパッケージについて書かれているわけですが、この章ではパッケージの集りであるアーカイブをどのように管理、配布するのかについて説明されています。Debian Policy は読んだことの無い方でも Debian を使っていれば聞いたことのある内容でしょう。

さて、事前課題で内容は読んで理解していただいていると思いますのでざっと内容をみていきましょう。

43.1 Debian フリーソフトウェアガイドライン

Debian がフリーであると考えるソフトウェアの定義です。原文の Debian Free Software Guidelines を略した DFSG という単語もよく使われます。「DFSG フリー」や「DFSG に準拠」という言い方はこのガイドラインに準拠したソフトウェアである、Debian が認めるフリーなソフトウェアであるということです。

DFSG は Debian 社会契約[*104]の一部であり Debian の根幹ですので一度じっくりと読んでみてください。

43.2 アーカイブエリア

43.2.1 main

Debian ディストリビューションといえばこの main アーカイブエリアのことを指します。

main に収録されるパッケージは DFSG に準拠していなければならず、コンパイル時や実行時にアーカイブエリア外のソフトウェアを必要としないこと、メンテナンスできること、Debian Policy に適合していることが求められます。

このアーカイブエリアのパッケージは誰でも自由[*105]に使用、共有、修正、配布することができます。

43.2.2 contrib

contrib アーカイブエリアには、DFSG に準拠しているがコンパイル時や実行時にアーカイブエリア外のソフトウェアを要求するため main アーカイブエリアに置けないパッケージが収録されます。

43.2.3 non-free

non-free アーカイブエリアには、DFSG に準拠しないか配布に問題があるパッケージが収録されています。このアーカイブエリアのパッケージは自由に使用、共有、修正、配布することができません。

[*104] http://www.debian.org/social_contract
[*105] http://www.debian.org/intro/free

43.3 著作権に関する考慮

著作権について疑義があるソフトウェアはアーカイブに収録されることが留保されます。また、著作権が明示されていない作品の配布や変更は認められていませんので注意してください。

パッケージは著作権情報と配布ライセンスの無修正コピーを `/usr/share/doc/`*package*`/copyright` ファイルとして同封し配布しなければなりません。

43.4 セクションとプライオリティ

セクションは Debian アーカイブメンテナによって公式に提供されており、勝手に追加することはできません。

プライオリティは高い順に required、 important、 standard、 optional、 extra があり、大半のパッケージは optional に属します。また、プライオリティの高いパッケージはビルド時を除いてプライオリティの低いパッケージに依存してはいけません。

プライオリティ

required　システムが適切に機能するために必要なパッケージ
important　Unix ライクなシステムに必ず入っていることが期待されるプログラムのパッケージ
standard　ほどよく小規模ながらキャラクタベースのシステムを提供するパッケージ
optional　インストールしておく価値のある全てのパッケージ
extra　上記いずれかが指定されているパッケージと衝突するパッケージ

43.5 最近の変更点

日本語訳版がある Version 3.9.1.0 から Version 3.9.3.1 までに加えられた変更点を押さえておきましょう。

Version 3.9.2.0

- 「 Debian GNU/Linux ディストリビューション」が「 Debian ディストリビューション」と改められました。
- main、 contrib、 non-free アーカイブエリアとは、が追記されています。

Version 3.9.3.0

- main アーカイブエリアのパッケージがアーカイブエリア外のパッケージを必要とする (required) だけでなく推奨する (recommend) ことも明記されました。 (”Depends”、”Recommends”、”Build-Depends” に加えて ”Pre-Depends” と ”Build-Depends-Indep” が明記されています。)
- セクションに education、 introspection、 metapackages の 3 つが追加されました。

44 月刊 Debian Policy 第4回「バイナリパッケージ」

山城国の住人 久保博

44.1 内容の概観

3章には、日本語訳で「バイナリパッケージ」というタイトルがついて、バイナリパッケージの論理的な構造や約束毎について述べています。

バイナリパッケージは .deb ファイル形式で提供されています。.deb 形式はアーカイブファイルの一種で、展開すると複数のファイルが取り出されます。そうして取り出されるファイルをまず二種類に大別しています。

- A set of files to install on the system when the package is installed
  （パッケージインストールの際にシステムにインストールされる一連のファイル群）
- second set of files: control information files
  （制御情報ファイル）

続いて、割と細かく節毎に記述があるのですが、流れるような文章ではなく、節一つ一つが箇条書の項目のようで、割と唐突な印象が拭えません。そこで、全体像をあぶり出して見ようと思います。

まず、この章で述べられている範囲の概念を図 36 に示す UML のクラス図に書いてみました。

図 36　3章のバイナリパッケージの概念的なクラス図

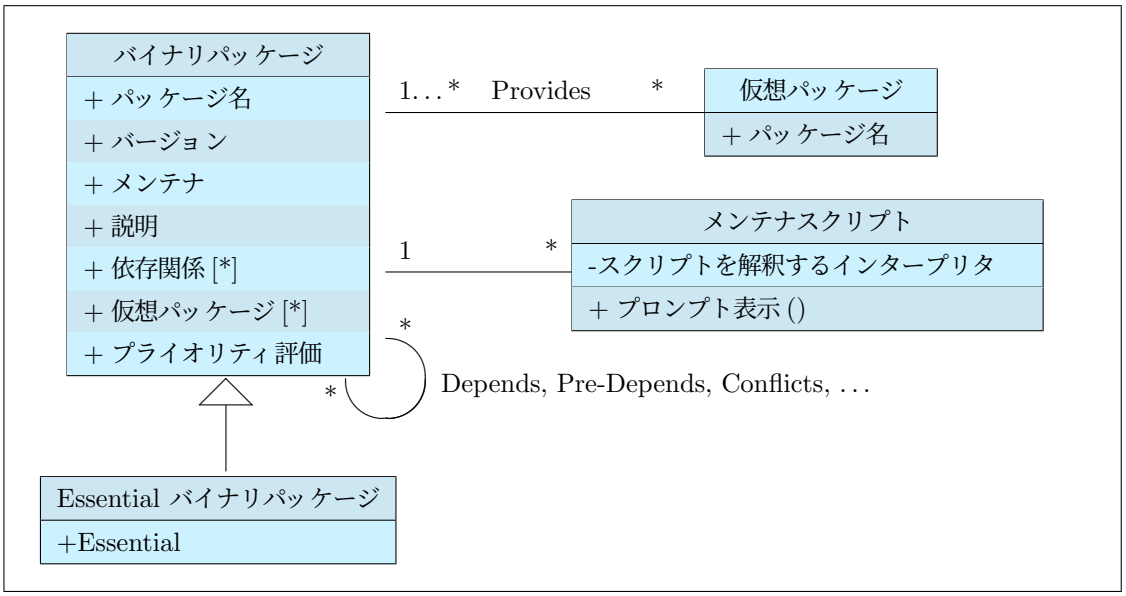


図 36 から、次に掲げる事がが読みとれるでしょうか。

- 「 バイナリパッ ケージ」は、ひとつのクラスとみなせる
- 「 バイナリパッ ケージ」には、いくつかの属性がある
- 「 バイナリパッ ケージ」クラスの特別なもの(派生クラス) として、「 Essential なバイナリパッ ケージ」クラスがある。
- 「 バイナリパッ ケージ」には、複数の「 メンテナスクリプト」が含まれる。

これを手がかりに3章を読んでみると、ここではバイナリパッ ケージが守るべき決め事をを次のような三つの側面から論じているように読みとることが出来るように思います。

1. パッ ケージ一つ一つの性質に応じてつける属性の決め事
   - 3.1 節 パッ ケージ名
   - 3.2 節 パッ ケージのバージョ ン
   - 3.3 節 パッ ケージのメンテナ
   - 3.4 節 パッ ケージの説明
   - 3.5 節 依存関係
   - 3.6 節 仮想パッ ケージ
2. Debian システム全体の中で特別な地位を占めるパッ ケージにつける属性についての決め事
   - 3.7 節 Base システム
   - 3.8 節 Essential パッ ケージ
3. パッ ケージに含まれるメンテナスクリプトが守るべき決め事
   - 3.9 節 メンテナスクリプト

また、 3.1 節から 3.8 節までは、「 5.3 バイナリパッ ケージコントロールファ イル – DEBIAN/control」で解説されている、コントロールフィ ールドに記述する内容に関するガイドラインになっ ています。 3.9 節「 メンテナスクリプト」だけは、バイナリパッ ケージの属性ではなく、パッ ケージに含まれるスクリプトが守るべき決め事が書かれています。

44.2 最近の変更点

次に、日本語訳版がある Version 3.9.1.0 から Version 3.9.3.1 までに加えられた変更点を押さえておきましょ う。

- 「 Debian GNU/Linux ディ ストリビューショ ン」が「 Debian ディ ストリビューショ ン」と改められました。
- メンテナの責任について、詳しく具体的な記述が追加されました。
- メンテナのメールアドレスについて、詳しく具体的な記述が追加されました。
- Uploader コントロールフィ ールドについての記述が追加されしました。
- みなしごパッ ケージについての記述が書き改められました。
- Pre-Depends コントロールフィ ールドについての記述が書き改められました。

以下では、左に元の英文、右に日本語訳を並べて、バージョ ン間での変化を比べてみます。

44.2.1 「Debian GNU/Linux ディストリビューション」が「Debian ディストリビューション」と改められました。

Version 3.9.1.0 policy.sgml:799

The Debian GNU/Linux distribution is based on the Debian package management system, called `dpkg`. Thus, all packages in the Debian distribution must be provided in the `.deb` file format.

Debian ディストリビューションは `dpkg` と呼ばれる Debian パッケージ管理システムに基礎を置いています。このため、Debian ディストリビューションに含まれる全てのパッケージは `.deb` ファイル形式で提供されなければなりません。

Version 3.9.3.1 policy.sgml:836

The Debian distribution is based on the Debian package management system, called `dpkg`. Thus, all packages in the Debian distribution must be provided in the `.deb` file format.

Debian ディストリビューションは `dpkg` と呼ばれる Debian パッケージ管理システムに基礎を置いています。このため、Debian ディストリビューションに含まれる全てのパッケージは `.deb` ファイル形式で提供されなければなりません。

44.2.2 メンテナの責任について、詳しく具体的な記述が追加されました。

Version 3.9.1.0 policy.sgml:914

Every package must have a Debian maintainer (the maintainer may be one person or a group of people reachable from a common email address, such as a mailing list). The maintainer is responsible for ensuring that the package is placed in the appropriate distributions.

全てのパッケージには一人または一グループの Debian メンテナ (一名の個人であっても、メーリングリストなどの共通の一つのメールアドレスで連絡の取れるグループであってもかまいません) を持たなければなりません。この人物は、そのパッケージが適切なディストリビューションに収録されていることに対する責任を持ちます。

Version 3.9.3.1 policy.sgml:951

Every package must have a maintainer, except for orphaned packages as described below. The maintainer may be one person or a group of people reachable from a common email address, such as a mailing list. The maintainer is responsible for maintaining the Debian packaging files, evaluating and responding appropriately to reported bugs, uploading new versions of the package (either directly or through a sponsor), ensuring that the package is placed in the appropriate archive area and included in Debian releases as appropriate for the stability and utility of the package, and requesting removal of the package from the Debian distribution if it is no longer useful or maintainable.

後ほど述べるようなみなしごパッケージを除いて、全てのパッケージには一人または一グループの Debian メンテナを持たなければなりません。メンテナは一名の個人であっても、メーリングリストなどの共通の一つのメールアドレスで連絡の取れるグループであってもかまいません。この人物は、その Debian パッケージファイルを保守すること、不具合報告を評価して適切に応答すること、そのパッケージの新しいバージョンを(直接あるいはスポンサーを介して) アップロードすること、そのパッケージが適切なアーカイブ領域に設置されて、そのパッケージの安定性や利便性の観点で適切に Debian のリリースに含められることを保証すること、もはや役に立たないか保守できなくなったら Debian 配布物からそのパッケージを削除する要求を出すことに対する責任を負います。

44.2.3 メンテナのメールアドレスについて、詳しく具体的な記述が追加されました。

Version 3.9.1.0 policy.sgml:922

The maintainer must be specified in the `Maintainer` control field with their correct name and a working email address. If one person maintains several packages, they should try to avoid having different forms of their name and email address in the `Maintainer` fields of those packages.	Debian パッケージのメンテナは各パッケージの `Maintainer` コントロールフィールドに、正しい名前と有効な電子メールアドレスの両方により指定されていなければなりません。もしその人がいくつかのパッケージを管理している場合、個々のパッケージの `Maintainer` フィールドに異なった形式の名前と電子メールアドレスを記入することは避けるべきです。

Version 3.9.3.1 policy.sgml:966

The maintainer must be specified in the `Maintainer` control field with their correct name and a working email address. The email address given in the `Maintainer` control field must accept mail from those role accounts in Debian used to send automated mails regarding the package. This includes non-spam mail from the bug-tracking system, all mail from the Debian archive maintenance software, and other role accounts or automated processes that are commonly agreed on by the project.<footnote> A sample implementation of such a whitelist written for the Mailman mailing list management software is used for mailing lists hosted by alioth.debian.org. </footnote> If one person or team maintains several packages, they should use the same form of their name and email address in the `Maintainer` fields of those packages.	Debian パッケージのメンテナは各パッケージの `Maintainer` コントロールフィールドに、正しい名前と有効な電子メールアドレスの両方により指定されていなければなりません。 `Maintainer` コントロールフィールドに書かれた電子メールアドレスは、そのパッケージに関する自動送信メールに使われる Debian 内の役割に応じたアカウントのメールを受領できないといけません。これにはバグ追跡システムからの迷惑メールでないメール、Debian アーカイブ保守用ソフトウェアやそのプロジェクトで共通に認められた役割に応じたアカウントもしくは自動処理からのすべてのメールを含みます。 <footnote> そのような一例として Mailman メーリングリスト管理ソフトウェア用のホワイトリストが alioth.debian.org でホストされているメーリングリストで使われています。 </footnote> もし一人の人もしくは一つのチームがいくつかのパッケージを管理している場合、それぞれのパッケージの `Maintainer` フィールドでは同じ形式の名前と電子メールアドレスを記入するべきです。

44.2.4 Uploader コントロールフィールドについての記述が追加されしました。

Version 3.9.1.0 policy.sgml:

Version 3.9.3.1 policy.sgml:990

If the maintainer of the package is a team of people with a shared email address, the `Uploaders` control field must be present and must contain at least one human with their personal email address. See <ref id="f-Uploaders"> for the syntax of that field.	もしパッケージのメンテナが共通のメールアドレスを持つ人々からなるチームなら、少なくとも一人の人物とその人の個人用のメールアドレスが書かれた `Uploaders` コントロールフィールドが必要です。このフィールドの書式については、 <ref id="f-Uploaders"> を参照してください。

44.2.5 みなしごパッケージについての記述が書き改められました。

Version 3.9.1.0 policy.sgml:936

If the maintainer of a package quits from the Debian project, "Debian QA Group" *packages@qa.debian.org* takes over the maintainer-ship of the package until someone else volunteers for that task. These packages are called *orphaned packages.*<footnote> The detailed procedure for doing this gracefully can be found in the Debian Developer's Reference, see <ref id="related">. </footnote>	もしあるパッケージのメンテナが Debian プロジェクトを辞めたなら、誰か他の人がその仕事に志願するまで Debian QA グループ *packages@qa.debian.org* がパッケージの管理を引き継ぎます <footnote> これを丁寧に行うやり方の詳細は Debian Developer's Reference (開発者の手引き) に書かれています。 <ref id="related"> を参照ください。 </footnote> 。このようなパッケージは *orphaned* **パッケージ**と呼ばれます。

Version 3.9.3.1 policy.sgml:

An orphaned package is one with no current maintainer. Orphaned packages should have their `Maintainer` control field set to `Debian QA Group <packages@qa.debian.org>`. These packages are considered maintained by the Debian project as a whole until someone else volunteers to take over maintenance.<footnote> The detailed procedure for gracefully orphaning a package can be found in the Debian Developer's Reference (see <ref id="related">). </footnote>	みなしごパッケージというのは、現在メンテナが不在のパッケージのことを言います。みなしごパッケージは、その `Maintainer` コントロールフィールドを `Debian QA Group <packages@qa.debian.org>` に設定することになっています。これらのパッケージは、誰が別の人が保守を引き継ぐことを志願するまでは Debian プロジェクト全体で面倒を見ているとみなされます <footnote> パッケージを丁重にみなしご化するための詳しい手順は Debian Developer's Reference (開発者の手引き) に載っています(<ref id="related"> を参照のこと) 。 </footnote>。

44.2.6 Pre-Depends コントロールフィールドについての記述が書き改められました。

Version 3.9.1.0 policy.sgml:1082

Sometimes, a package requires another package to be installed *and* configured before it can be installed. In this case, you must specify a `Pre-Depends` entry for the package.	時々、あるパッケージが、それをインストールする前にもう一つのパッケージがインストールされ**かつ** 設定されていることを必要とすることがあります。この場合、そのパッケージには `Pre-Depends` エントリを指定しなければなりません。

Version 3.9.3.1 policy.sgml:1144

Sometimes, unpacking one package requires that another package be first unpacked *and* configured. In this case, the depending package must specify this dependency in the `Pre-Depends` control field.	時々、あるパッケージを展開するためには、先に別のもう一つのパッケージが展開され**かつ** 設定されていないといけない場合があります。この場合、依存する側のパッケージに `Pre-Depends` コントロールフィールドでこの依存関係を指定しなければなりません。

44.3 引用元

“Debian Policy Manual ”, Ian Jackson , Christian Schwarz, and other contributors, *git://git.debian.org/git/dbnpolicy/policy.git* ,Debian Project, 1996-

“Debian ポリシーマニュアル”, 八田真行, かねこ, 森本 *https://svn.debian.or.jp/repos/www/trunk/src/community/devel/debian-policy-ja/policy.ja.sgml* , Debian JP Project,

45 Debian Trivia Quiz

上川 純一

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。

問題 1. 11 月終わり頃にルートファイルシステムの構造について議論を呼んでます。内容は?

☐ A /user を作る

☐ B /bin,/sbin,/lib の実体を/usr 以下に移動して、代わりにシンボリックリンクにする

☐ C /etc の実体を/usr 以下に移動して、代わりにシンボリックリンクにする

問題 2. sun-java6 が Debian パッケージとして配布できなくなりました。代わりに Debian で推奨される Java は?

☐ A openjdk

☐ B gcj-jdk

☐ C coco-java

問題 3. 11/19 に長らく活動を停止していたパッケージチームが復活宣言をしました。どれでしょう?

☐ A CORBA packaging team

☐ B Ham-radio packaging team

☐ C SDL packaging team

問題 4. 10/28〜30 で MiniDebconf2011 が開かれました。どこの国でしょう?

☐ A ニカラグア

☐ B インド

☐ C フランス

問題 5. Wheezy フリーズの為の BSP が各国で開かれました。ドイツとどこ?

☐ A フランス

☐ B ニカラグア

☐ C ポーランド

問題 6. armhf が unstable にはいったのはいつか

☐ A 2011-11-24 1952

☐ B 2013-11-24 1952

☐ C 2001-11-24 1952

問題 7. s390x が unstable にはいったのはいつか

☐ A 2011-11-25 0152

☐ B 2013-11-25 0152

☐ C 2001-11-25 0152

問題 8. 1/17 に alioth になにがおきたか

☐ A vasks.debian.org が起動しなくなった

☐ B wagner.debian.org が起動しなくなった

☐ C SOPA の抗議をはじめた

問題 9. NM process の NM は何を意味することになったか

☐ A New Maintainer

☐ B New Member

☐ C New Moemoe

問題 10. REVU になにがおきるといっているか
□ A universe を拡大
□ B Debian を必要なくする
□ C mentors.debian.net に統合

問題 11. トレードマークについての連絡先は
□ A trademark@debian.org
□ B trade@debian.net
□ C iwamatsu@debian.org

問題 12. win32-loader.exe の新機能は
□ A Debian GNU/Hurd のインストール
□ B Debian GNU/kFreeBSD のインストール
□ C Debian GNU/Linux のインストール

問題 13. wiki.debian.org の launchpad バグ対応を利用するにはどのタグを使うか
□ A UbuntuBug
□ B DebianBug
□ C Hoge

問題 14. dh-exec とはなにか
□ A 実行可能な設定ファイルの出力を使う仕組み
□ B どんなものでも実行する仕組み
□ C 実行、実行、実行

問題 15. Derivatives Census `http://wiki.debian.org/Derivatives/Census` にはなにがかいてあるか
□ A Debian の正当な後継者の一覧
□ B Debian からの派生物の一覧
□ C Debian を dis ってる人の一覧

問題 16. `http://debtags.debian.net/` のリニューアルでは何をしたか
□ A Django と jQuery での書き直し
□ B Debian ベースでの再実装
□ C ocaml で実装しなおした

問題 17. Debian の監査役としてがんばっているのは誰か
□ A Nobuhiro Iwamatsu
□ B Stefano Zacchiroli
□ C Martin Michlmayr

問題 18. kassia と liszt はいくらするのか
□ A 10,000USD
□ B 100 万円
□ C 11'792.9 EUR

問題 19. Portland BSP で使った sbuild インスタンスはいくらしたか
□ A 70USD
□ B 700USD
□ C 7000USD

問題 20. Lenny のセキュリティサポートが終わったのはいつ?
□ A 2012/02/06
□ B 2012/02/07
□ C 2012/02/08

問題 21. 2012/01/28 に更新された Squeeze のヴァージョンは?
□ A 6.0.4
□ B 6.1.0
□ C 20120128

問題 22. Debian Game チームが 2/25 から 2/26 まで行うイベントは何か?
□ A どれだけの Windows のゲームが Wine 上で動作するか検証するパーティ
□ B Debian で提供されているゲームパッケージのスクリーンショットを撮りまくるパーティ
□ C Debian で提供されているゲームを 48 時間連続プレイするパーティ

問題 23. Wheezy で採用される Linux カーネルバージョンは?
□ A 2.6.39
□ B 3.2
□ C 4.0

問題 24. pts.debian.org で表示されるようになった情報は?
□ A パッケージメンテナが誕生日の日は「おめでとう」と出る。
□ B パッケージを乗っ取ろうとしている人の情報
□ C パッケージ Transition 情報

問題 25. アクセプトされた DEP は?
□ A DEP 3
□ B 3 DEP
□ C DEP DEP DEP

問題 26. 今年度の Debian JP 会長は誰か?
☐ A Kouhei Maeda
☐ B Nobuhiro Iwamatsu
☐ C Junichi Uekawa

問題 27. Debian.org DPL 選挙は誰が立候補したか
☐ A Stefano Zacchiroli
☐ B Nobuhiro Iwamatsu
☐ C Kouhei Maeda

問題 28. Debconf12 の suponsord な参加の締切りはいつ
☐ A 4 月末
☐ B 5 月 15 日
☐ C 5 月末

問題 29. 大統一 Debian 勉強会での発表の公募 (CFP) はいつが締切り?
☐ A もう過ぎた
☐ B 4 月末
☐ C 4 月 22 日

問題 30. experimental 版の apache のパッケージのバージョンはいくつ?
☐ A 2.3
☐ B 2.4.0
☐ C 2.4.2

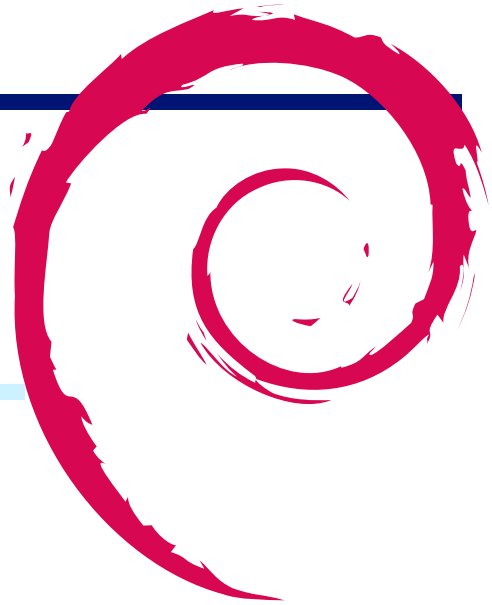
問題 31. Debian Edu はまたの名を何というでしょう?
☐ A emdebian
☐ B Scientific Linux
☐ C Skolelinux

問題 32. 3/30 に Debian Project が加盟した団体の名前は?
☐ A OSC
☐ B OSI
☐ C ETF

問題 33. Debian Project の gobby サーバーとして gobby.debian.org がアナウンスされました。ところで gobby って何?
☐ A 旧ソ連で開発された諜報活動用ソフトウェア
☐ B churro の代替サーバ
☐ C エディタの名前

問題 34. Debian installer 7.0 alpha 1 のリリース日は
☐ A 5/13
☐ B 6/13
☐ C 4/13

問題 35. Cyril Bruleb が 6 月に Wheezy をフリーズすると発表したが、 Transition の締め切りはいつだといっているか
☐ A 5 月 13 日
☐ B 6 月 10 日
☐ C 5 月 20 日



46 Debian Trivia Quiz 問題回答

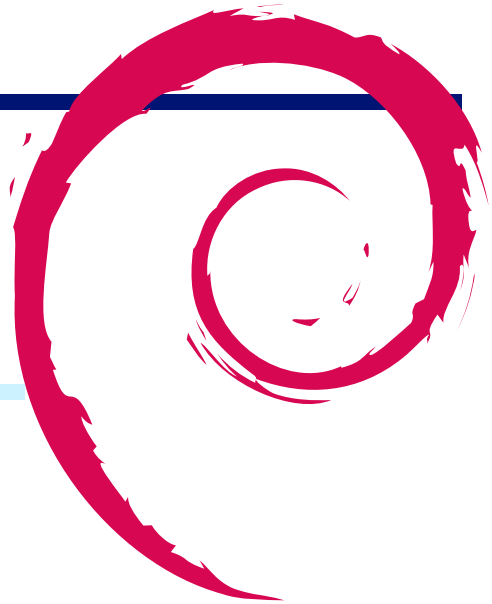
上川 純一

Debian Trivia Quiz の問題回答です。あなたは何問わかりましたか?

1. B
   他主要ディストリビューションが採用検討中...
2. A
   残念だ>○racle
3. C
   これからも頑張って欲しいですね
4. B
   来年は日本がいいなぁ
5. C
   Wheezy のフリーズは 2012/6 なので、開発作業はお早めに
6. A
   dinstall mirror pulse の時間です。
7. A
   dinstall mirror pulse の時間です。
8. A
9. B
   New Maintainer から New Member に切り替わりました
10. C
11. A
12. A
    win32-loader.exe は Windows で起動すると Debian-installer を起動できるようにしてくれるツール。今回は Hurd もインストールできるようになりました。
13. A
14. A
15. B
16. A
17. C
18. C
19. A
20. A
21. A
22. B
23. B
24. C
25. A
    DEP3 は Patch tagging guideline. Debian Enhancement Proposals
26. A
    前年度に引き続き前田さんが信任されました。またよろしくお願いします。
27. A

前年度に引き続き Zacchiroli さん頑張ってます。

28. B
うっかりすると過ぎてしまうので、気をつけましょう。また、 UTC なのかニカラグアのローカルタイムなのかちょっとわからないので、 5/15 に登録開始するのは避けた方がよいかも。本家アナウンス `http://debconf12.debconf.org/` 参照。
29. C
大統一 Debian 勉強会で発表できるチャンスです。締切りは忘れずに。
30. C
3/22 にアナウンスがありました。 upstream 側も 2.4.2 です。最新版ですね。BUG 見つけましたら、BUG　Report 書きましょう。
31. C
Debian Edu とは教育機関向けに作られた Debian ベースのディストリビューションの事です。昔は Skolelinux という名前で開発されていた物だそうです。正式な Debian のサブプロジェクトです。先日新しいバージョンがアナウンスされました。
32. B
これでまた Free Software として磨きがかかりました。 OSI は Open Source Initiative の略だそうです。
33. C
Debconf の BOF 会場でよく使われていますエディタです。サーバを介する事により、複数人で同時に１つの文章を同時に編集できます。 Debconf では、リアルタイムに議事と議事録が BOF 参加者によってどんどん編集されていく様はおもしろいです。ちなみに churro とは i18n.debian.org の事です。
34. A
Wheezy のインストーラーのアルファリリースが出たので皆さん試してください。
35. C
Transition するなら 5 月 20 日までにバグをファイルしておけとのこと。 Transition というのはざっくりというと多数のパッケージが相互に依存しているような変更。例えば、ライブラリの ABI が変わるだとか。



47 索引

本資料のライセンスについて

本資料はフリー・ソフトウェアです。あなたは、 Free Software Foundation が公表した GNU GENERAL PUBLIC LICENSE の「 バージョン２」もしくはそれ以降が定める条項に従って本プログラムを再頒布または変更することができます。

本プログラムは有用とは思いますが、頒布にあたっては、市場性及び特定目的適合性についての暗黙の保証を含めて、いかなる保証も行ないません。詳細については GNU GENERAL PUBLIC LICENSE をお読みください。

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

ソースコードについて

ソースコードは Git を使って git://anonscm.debian.org/tokyodebian/monthly-report.git からダウンロードできます。以下に方法を示します。

```
$ git clone -b printed-2012-natsu git://anonscm.debian.org/tokyodebian/monthly-report.git
```

『 あんどきゅめんてっど でびあん』について

本書は、東京および関西周辺で毎月行なわれている『 東京エリア Debian 勉強会』（ 第 83 回から第 88 回）および『 関西 Debian 勉強会』（ 第 54 回から第 58 回）、福岡で行われた『 福岡 Debian 勉強会』、そして『 大統一 Debian 勉強会』で使用された資料・ 小ネタ・ 必殺技などを一冊にまとめたものです。内容は無保証、つっこみなどがあれば勉強会にて。

あんどきゅめんてっど でびあん 2012 年夏号

2012 年 06 月 23 日　　初版第 1 刷発行

東京エリア Debian 勉強会/関西エリア Debian 勉強会（ 編集・ 印刷・ 発行）