東京エリア デビアン 勉強会

Debian勉強会幹事 上川純一

2009 年 11 月 14 日

デビアン勉強会

目次

1	Introduction	3
2	**事前課題**	4
3	**最近の** Debian **関連のミーティング報告**	6
4	Debian Trivia Quiz	7
5	Debian **での数学ことはじめ。**	8
6	Debian autobuilder **ネットワーク**	15

1 Introduction

上川 純一

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

Debian 勉強会の目的は下記です。

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

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

2009 年の計画は仮です。

1. 新年の企画 (アンサンブル荻窪開催)
2. OSC Tokyo
3. VAIO P インストール記録、カーネル読書会 ディストリビューション大集合 (小林さん)(東京大学?)
4. Git Handson (岩松)(あんさんぶる荻窪?)
5. 家 Debian サーバ vs 職場のネットワーク (千代田区都立図書館?[*1])
6. Asterisk (東京大学?)
7. スペインにて開催
8. Debconf 報告会
9. OSC Fall?
10. udev + HAL(岩松さん)
11. 3D graphics 開発（藤沢さん）
12. Debian サーバ +VMware + 各種 OS、他の仮想化ツール (vserver etc.)、忘年会

会場候補としては下記があります:

- 大学
- 恵比寿 SGI ホール
- Google オフィス
- 公民館 (あんさんぶる荻窪等)
- 都立会議室 (無線 LAN)
- 健保の施設

[*1] http://www.library.chiyoda.tokyo.jp/

2 事前課題

上川 純一

事前課題は:
「あなたが普段行っている統計処理の内容とその処理で行っているハックを披露してください。」
この課題に対して提出いただいた内容は以下です。

2.1 まえだこうへい

2.1.1 普段行っている統計処理

現場にいるときはあまり必要としなかったのですが、昨年度までのプリセールスのときは、プロビジョニングする上での CPU やメモリのリソース使用率、見積り試算、売上試算、現状の企画の仕事では、新規企画との現状サー ビスの利用量、料金の対比などを行う際に統計を行ったりしています。

が、Excel が遅いんですよね。5 列 2 万行程度の計算や、グラフのプロットを行うと、もう落ちる落ちる。OS のレスポンスも非常に悪化して、昨年やってた一番酷い例では、5 分に 1 度、Excel が落ちて、データ復旧してはグラフ作ってまた落ちて、なんてときに、もう Excel だけでなくスプレッドシートは嫌だなと思いました。

結果が分かっている統計データをプロットするだけならスプレッドシートでも良いのかもしれませんが、値のコピーだけをすると、あとで見直したときに、このデータはどれから算出したのだったのかを把握できなくなります。でもセルの参照を多用すると前述の問題は頻発しやすくなります。すると、データ解析したりグラフ描画はもうスプレッドシートは止めたいな、というのがきっかけで gnuplot を使い始めました。これはこれで便利ですが今回のネタ発表するのに当たって、GNU R を使ってみたら、これはさらに便利ですね。仕事でもっと活用しようと思います。

プライベートは、今回のネタのサンプルデータのような光熱費や、家計簿の統計を取るのに OOo を使ってましたが、これもやっぱりデータを出すのが面倒なので、この機会に変更しようと思ってます。OOo 自体も止めようかな。

2.2 やまねひでき

2.2.1 「普段行っている統計処理の内容とその処理で行っているハック」

いや、全く統計処理なるものをやっていないのです…どなたかどういう場面で役立つか、とかもし初歩から知りたい場合はこれを見ろ！など示唆いただけると大変助かりますです、はい。

2.3 日比野 啓

2.3.1 普段行っている統計処理の内容とその処理で行っているハック

普段、統計処理のようなことをほとんどやらないのですが、2,3 年前に、自分の開発効率を計測するために、OpenOffice.org の表計算機能でシステム開発にかけた時間を細かく計測してみたことがあります。やむをえず手作業になっている本来自動化できそうな部分あり、libspreadsheet-writeexcel-perl あたりで何とかできないか気になっていました。また機会があれば試してみたいと思っています。R のような処理系も今日をきっかけに使いこなせるようにしておきたいところです。

2.4 本庄弘典

2.4.1 普段行っている統計処理

Perl を使用してログを集計する程度です。Perl から gnuplot を使用したりはします。ハックはありません。

2.5 なかおけいすけ

2.5.1 普段やっている統計処理の内容

職業柄、主に時系列解析と相関解析を行っています。また機器類の校正、精度測定のために線形解析も行っています。

私の職場は高電圧や大電力な装置がそこら中にあるので、時々放電が起こります。放電によって機器が破損することがあるので、放電箇所を特定しなければなりません。放電しそうなところにカメラを置いて録画監視しているのですが、いつ放電が起こるかわかりません。放電が起こるのを今か今かとずっと見ているわけにはいかないので、録画した画像を数値化して分布を見ることで、放電した時刻と場所を特定しています。

2.5.2 統計処理で行っているハック

特にハックして使っていないのですが、データを処理した過程を記録することが重要なので、できるだけスクリプトにしています。

2.6 吉田@板橋

2.6.1 普段やっている統計処理の内容

本来の統計処理という意味では calc や Excel で済ませてしまい、最近はマクロ等を作ることもあまり無く、せいぜい lookup 系やソルバー止まりでまったり使っています。あとはスクリーニングやご家庭内システム監視をするのに ruby や python またシェルスクリプトを使ってますが、かなり監視対象機器に依存している＆単純計算が主なので、統計という程の内容ではないですね。小ネタとしてログイン時にディスク空き容量がピンチか自動的に確認できるように下記のようなスクリプトを仕込んでいます。

```
$ tail .bash\_profile

#Yellow
echo -e "\033[1;33m"
df | grep "9.%" && sleep 15
echo -e "\033[0m"

#Red
echo -e "\033[1;31m"
df | grep "100%" && sleep 30
echo -e "\033[0m"
```

要するに容量が 90% 以上使われているパーティションがあるとログインがそこで 15〜30 秒止まり、なにが溢れそうか強制的に確認します。

まあ、統計かというと微妙ですが、まあ、パーセンテージを使っていると言うことで (笑)

2.7 emasaka

2.7.1 普段行っている統計処理

統計処理というほどのことはやってません。Excel で集計するぐらいでしょうか。

2.8 araki

2.8.1 araki

統計ですか。。ぜんぜん使ってないですね。はるか昔に S-lang を使う makefile を書いたけど、それを呼ぶことが 7 月にあったのが最後。でも勉強はしたいです。というわけでよろしく。

2.9 あけど ただお

2.9.1 普段行っている統計処理とハック

普段は特に決まった統計処理をしているわけではないので、比較的最近に行った統計処理（っぽい事）について述べます。

個人運営サイトのメールサーバの管理を引き受けているものがあり、そこでの迷惑メールの着信状況の推移を調べるのにログファイルを grep や sed 等でテキスト処理を行い最後は wc -l にて行数をカウントするスクリプトを仕込んでいました。

それと関連してトラフィック統計を vnstat で取って月毎の総量変化を見たりしました。またこの結果から iptables にフィルタルールを追加したりもしてみました。

結果として、いくつかポイントになるフィルタ設定によりサーバ全体の総トラフィック量が 10 分の 1 以下になりました。

3 最近の Debian 関連のミーティング報告

前田耕平

3.1 東京エリア Debian 勉強会 56 回目報告

56 回は 9 月 16 日 (水)19:00 からといういつもとかなり変則的な時間で、ミラクルリナックスさんのセミナールームをお借りして、Key Sign Party を行いました。最初に岩松さんによる、なぜヒトはキーサインを求めるのか？というありがたいお言葉をいただき、キーサインは Web of Trust と言いつつも、その実は Web of I know who での信頼レベルなので、パスポートのような強力な公的な身分証明書で己れの存在証明をする必要があるというはなしも途中ありました。100 人もキーサインする相手がいると gpg コマンドでちまちまとやってられんよね？という問題を解決する便利なツール caff の使い方を伝授してもらいました。

その後、実際に参加したメンバー全員で、立食パーティでのキーサインパーティを行いました。

参加された方は皆、リア充できたようです。

3.2 東京エリア Debian 勉強会 57 回目報告

第 57 回は、10 月 30 日（金)、31 日（土）に日本工学院大学の蒲田キャンパス 12 号館で OSC 2009Tokyo/Fall でブースのみ出展しました。元々は、31 日のみ出展するでしたが、あけどさんなど有志の方により、30 日 (金) もブースをやっていたとのことです。

配布物としてはフライヤー 50 部、9 月の勉強会で使用した GPG サインの資料 20 部、Live DVD 20 枚あまりを用意しました。また、展示物として岩松さんが sh7724 搭載マシンを用意してくれ、またそのマシンの詳細を印刷したパンフレットも用意してくれました。配布物の配布結果はほぼ予想通りで、そこそこ足りていました。Live DVD のイメージが当日早朝にできたこともあり、ブースで少し焼きましたが、全部で 25 枚程度でした。

Live USB イメージ (lenny で lxde) も用意しましたが、案内が直前だったこともあるのか、あまり知られておらず、イメージを欲しがる人はそれほどいませんでした。ただし、30 日 (金) にいらした方で、あけどさんに USB メモリを渡したまま帰ってしまった人がいたらしく、あけどさんが ML で連絡を求めています。[*2]

また、岩松さん提案の GPG 鍵交換ですが、なかなか好評で、ブースおよび懇親会で行ないました。

また、学生との交流もできたので、Debconf in Japan や出張 Debian 勉強会・Debian 温泉の企画にも進展がありそうです。

懇親会は JR 蒲田駅東口近くの「春香園」という餃子屋さんで行ないました。懇親会の参加者は、武藤さん、上川さん夫妻、日比野さん、本庄さん、谷口さん、岩松さん、藤澤さん、山本さん、前田の 10 名でした。

[*2] http://lists.debian.or.jp/debian-users/200911/msg00001.html
http://lists.debian.or.jp/debian-devel/200911/msg00000.html

4 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ははりあいがないので、理解度のテストをします。特に一人だけでは意味がわからないところもあるかも知れません。みんなで一緒に読んでみましょう。

今回の出題範囲は `debian-devel-announce@lists.deban.org` に投稿された内容と Debian Project News からです。

問題 1. 新しく Debian keyring-mainter になったのは？

☐ A Kenshi Muto

☐ B Gunnar Wolf

☐ C Jonathan McDowell

問題 2. 次期リリース: Squeeze でサポート対象から落ちそうなアーキテクチャは？

☐ A alpha と hppa

☐ B i386 と ia64

☐ C s390 と sparc

問題 3. 次期リリース: Squeeze で採用される Linux カーネルのバージョンは？

☐ A 2.6.31

☐ B 2.6.32

☐ C 2.6.33

問題 4. 追加されたパッケージのオートリジェクトは何がトリガーになる？

☐ A 妻帯者か否か

☐ B FTBFS のチェック

☐ C lintian によるパッケージチェック

問題 5. Debconf10 はいつから開催されることになったか

☐ A 2010 年 8 月 1 日

☐ B 2010 年 1 月 1 日

☐ C 2010 年 4 月 1 日

問題 6. Eugene V. Lyubimkin が発表した APT の代替アプリケーションは？

☐ A debmoe

☐ B cupt

☐ C saitama

5 Debianでの数学ことはじめ。

まえだこうへい

5.1 はじめに

Debian で数値計算や統計処理、グラフ作成を行うには、様々なパッケージがあります。私のように大学での専行が理系といっても基礎生物学だった人間には、コンピュータでの専用ツールでなくても、極端な話、基本的な統計学と関数電卓があれば事足りました。スプレッドシートを使っても、関数は使わずに電卓代わりに使うことも多かったと記憶してます[*3]。社会人になってからもその延長でスプレッドシートで大抵は事足りる、という方が一般的には多いのではないかと思います。

とはいえ、数万単位のサンプル数の計算をしたり、グラフを描写する場合、スプレッドシートでは非力だという問題もあります。プリセールスや企画で裏付け資料を作るのにサンプルデータを統計処理したり、データをプロットしてグラフを描くと、スプレッドシートだとマウスの操作だけで簡単にできる一方、CPU やメモリ不足でレスポンスが遅くなったり、ソフトウェア自体が落ちたり、酷いと OS 自体のレスポンスも悪化するというのが最近の仕事上での悩みだったりします。

そんなことがきっかけで今頃、Gnuplot を使い始めたわけですが、統計処理では最近は R の書籍を書店で見かけたり、数値計算では Octave というのがあるということを Debian 勉強会の ML で知ったり[*4]と、自分の中だけでなく、最近意外とまわりでもこの辺はホットな話題ではないかと思います。そこで、今回は私と同じように入門者向けに Octave、Gnuplot との連携方法、R の話に加え、Debian での使い方、パッケージングなどについて説明しようと思います。

5.2 概要

GNU Ocatave は計算処理のための高級言語で、MATLAB という数値計算の商用ソフトウェアとの互換性があるようです。行列計算、連立方程式、微分・積分、グラフ出力などに使えます。グラフ描画のためには、gnuplot が必要です。

一方、GNU R は統計解析のためのプログラミング言語で、文法的には、S 言語や Scheme に影響を受けているようです。Octave とは異なり、グラフ描画には gnuplot は必要ないようです。Excel などのデータ読み込みができるといった、データの互換性にも特徴があります。

[*3] 理由は、当時のスプレッドシートの関数の精度が低く、自分で計算式を作った方が確実だったためです。今では改善されたのでしょうか？

[*4] R や Octave のパッケージメンテナである Dirk Eddelbuettel さんが近々来日されるというのが ML で流れていたそもそものきっかけです。`http://dirk.eddelbuettel.com/`

5.3 インストール方法

Debian での各実行環境の導入方法について見てみます。今回の前提条件として、ディストリビューションは Sid[*5] としていますが、Stable(Lenny) や Testing(Squeeze) でもバージョンが異なることを除けばほぼ同じです。適宜読み替えてください。

5.3.1 Octave のインストール方法

GNU Octave は Squeeze/Sid では、`octave` というパッケージ名ではなく、バージョン 3.0 と 3.2 で別のパッケージになっています。今回は 3.2 のパッケージを導入します。

```
$ sudo apt-get install octave3.2
```

5.3.2 GNU R のインストール方法

GNU R は、Debian パッケージでの名前は、gnur でも、r でも gnu-r でもありません。`r-base` というメタパッケージで提供されています。これをインストールすると、最低限、`r-base-core` パッケージが GNU R の実行には必要です。また、GNU R には、機能拡張のパッケージが用意されています。CRAN(the Comprehensive R Archive Network) からダウンロードできます。Perl での CPAN のような位置づけです。千以上ものパッケージがあるようです。Debian では、CPAN や Ruby gems と同様にここから直接ダウンロードすることもできますが、約 150 の R パッケージが deb パッケージとして用意されています[*6]。パッケージ名は"`r-cran-<name>`" というネーミングルールです。

今回は `r-base` をインストールしましたが、余計なパッケージを入れないようにと、`r-base-core` パッケージをインストールしたとしても最小構成からのパッケージの差は二つしかありません。[*7]

```
$ sudo apt-get install r-base
```

5.3.3 Gnuplot のインストール

Gnuplot は Octave、GNU R のいずれも deb パッケージとしての依存関係はありませんが、octave では gnuplot を使ってグラフ描画もできるので、インストールしておくと便利でしょう。gnuplot のインストールには `gnuplot` パッケージをインストールします。

```
$ sudo apt-get install gnuplot
```

5.4 試してみる。

今回は、我が家の過去 9 年分の光熱費における各使用量の分布をサンプルデータとしてみます。まずは、もともと使い始めていた gnuplot での例をみて、それに対し、Octave, R ではどのように処理するのかを見てみましょう。

用意したサンプルデータは以下のようなものです。

```
"date","days","kWh","kWh/d","yen/d","yen","days","m^3","m^3/d","yen/d","yen","days","m^3","m^3/d","yen/d","yen","yen/d","yen"
2001/04,11,34,3.09,83.5,918,36,4.9,0.14,108.3,3900,58,9,0.16,22.8,1320,,
2001/05,33,95,2.88,73.8,2434,29,3.5,0.12,114.1,3310,,,,,,,
2001/06,30,129,4.3,102.7,3082,32,3,0.09,96.9,3100,61,3,0.05,15.7,960,,
2001/07,28,155,5.54,131.3,3676,28,1.7,0.06,91.1,2550,,,,,,,
(snip)
```

左から、年月, 電気使用日数, 月の電気使用量 (kWh),1 日あたりの電気使用量,1 日あたりの電気代, 月の電気代, ガ

[*5] 2009 年 11 月 10 日現在。
[*6] 2009 年 11 月 9 日現在。
[*7] 一つは `r-base`, もう一つは `r-base-dev` です。

ス使用日数, 月のガス使用量 (立方メートル),1 日あたりのガス使用量,1 日あたりのガス代, 月のガス代, 水道使用日数, 水道使用量 (立方メートル),1 日あたりの水道使用量,1 日あたりの水道代, 月の水道代,1 日あたりの下水道代, 月の下水道代、となっています。

5.4.1 gnuplot の場合

gnuplot では、次のようなバッチファイルを用意します。2001 年 4 月から 2009 年 8 月までの電気、ガス、水道の各使用量をそれぞれプロットします。また、Debian 勉強会資料用に、LaTeX で取り込めるように、図は EPS、文字列は LaTeX で出力するようにします。

```
reset
set terminal epslatex input color
set output "gnuplot.tex"
set datafile separator ","
set xdata time
set timefmt "%Y/%m"
set format x "%Y/%m"
set xtics rotate by -90
set mxtics 12
set xrange ["2001/04":"2009/08"]
set yrange [0:400]
set xlabel "年月"
set ylabel "電気使用量 [kWh]"
set y2range [0:50]
set y2label "ガス・水道使用量［立方メートル］"
set ytics nomirror
set y2tics
plot "kohnetsuhi.csv" using 1:3 axes x1y1 title "電気" with line,\
     "" using 1:8 axes x1y2 title "ガス" with line,\
     "" using 1:13 axes x1y2 title "水道" with line
```

このバッチファイルを LaTeX と EPS に変換します。バッチファイルをロードするには、対話モードで load コマンドを実行します。

```
$ gnuplot

        G N U P L O T
        Version 4.2 patchlevel 6
        last modified Sep 2009
        System: Linux 2.6.31.3

        Copyright (C) 1986 - 1993, 1998, 2004, 2007 - 2009
        Thomas Williams, Colin Kelley and many others

        Type ‘help‘ to access the on-line reference manual.
        The gnuplot FAQ is available from http://www.gnuplot.info/faq/

        Send bug reports and suggestions to <http://sourceforge.net/projects/gnuplot>


Terminal type set to ’wxt’
gnuplot> load ‘‘gnuplot.gp’’
gnuplot> quit
```

これで、カレントディレクトリに gnuplot.tex と gnuplot.eps ファイルが出力されます。gnuplot.tex は、LaTeX 文書の中で以下のように記述して取り込みます。

```
\usepackage{graphicx}

(snip)

\begin{figure}[thbp]
\input{image200911/gnuplot.tex}
\end{figure}
```

ただし、このままでは make が通りません。gnuplot.tex を書き換える必要があります。次のように下から 3 行目の includegraphics のパスを変更する必要があります。

```
$ diff -u gnuplot.tex.bk gnuplot.tex
--- gnuplot.tex.orig        2009-11-12 23:38:36.000000000 +0900
+++ gnuplot.tex         2009-11-12 23:39:39.000000000 +0900
@@ -111,7 +111,7 @@
       \put(5003,4163){\makebox(0,0)[r]{\strut{}水道}}%
     }%
     \gplbacktext
-    \put(0,0){\includegraphics{gnuplot}}%
+    \put(0,0){\includegraphics{image200911/gnuplot.eps}}%
     \gplfronttext
   \end{picture}%
 \endgroup
\end{document}
```

これで make すると、以下のような図を表示できます。

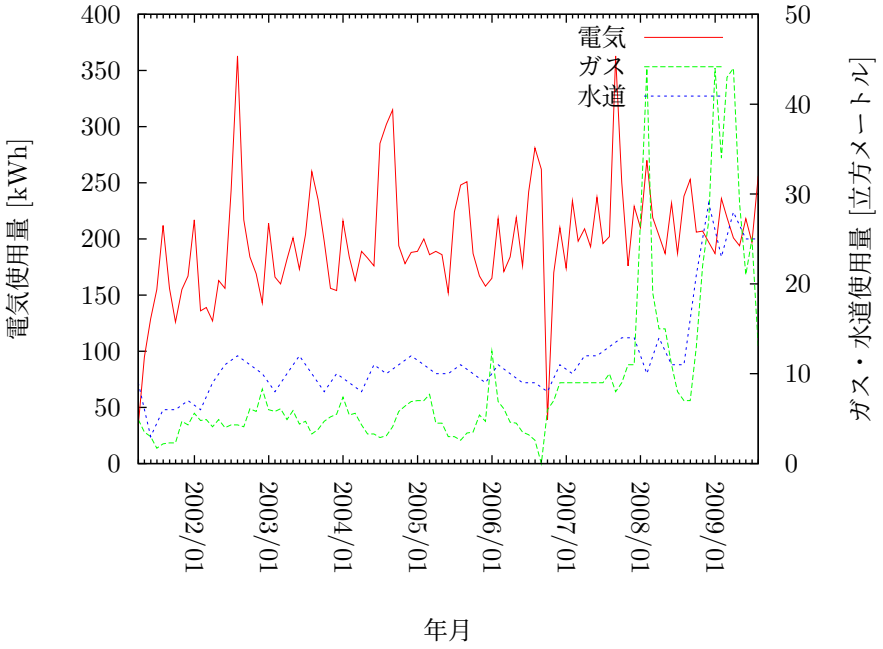

図 1　gnuplot でのプロットの例

では、これを R ではどのようにするのかを見てみましょう。[*8]

5.4.2　GNU R の場合

GNU R を対話形式で使うには、シェルで R と実行します。

```
$ R
```

R でデータを扱うには、先ほどの konetsu.csv ファイルをオブジェクトに取り込む必要があります。取り込むには、<-演算子を使います。

```
> konetsu <- read.csv("konetsu.csv")
```

この konetsu オブジェクトの中身を見てみます。このオブジェクトはデータフレームといい、これを指定するだけです。

[*8] Octave は gnuplot を使ってグラフ描画するということなので、今回省略。手が回らなかったのが事実ですが、何か。

```
> konetsu
    X...date days kWh kWh.day yen.day  yen days.1  m.3 m.3.day yen.day.1 yen.1
1     2001/4   11  34    3.09    83.5  918     36  4.9    0.14     108.3  3900
2     2001/5   33  95    2.88    73.8 2434     29  3.5    0.12     114.1  3310
3     2001/6   30 129    4.30   102.7 3082     32  3.0    0.09      96.9  3100
4     2001/7   28 155    5.54   131.3 3676     28  1.7    0.06      91.1  2550
5     2001/8   34 212    6.24   146.9 4995     35  2.2    0.06      78.9  2760
(snip)
    days.2 m.3.1 m.3.day.1 fee.d yen.2 yen.day.2 yen.3
1       58     9      0.16  22.8  1320        NA    NA
2       NA    NA        NA    NA    NA        NA    NA
3       61     3      0.05  15.7   960        NA    NA
4       NA    NA        NA    NA    NA        NA    NA
5       60     6      0.10  19.0  1140        NA    NA
(snip)
```

列数が増えると、この例のように分割して表示されます。“NA” と表示されているのは、値が null だったものです。

また、この取り込んだデータのサマリを表示させることも簡単です。以下のように R では統計処理のための関数があらかじめ用意されているので、とても便利です。

```
> summary(konetsu)
    X...date       days            kWh           kWh.day          yen.day
 2006/9 : 2   Min.   :11.00   Min.   : 34.0   Min.   : 2.880   Min.   : 73.8
 2001/10: 1   1st Qu.:29.00   1st Qu.:170.2   1st Qu.: 5.753   1st Qu.:129.1
 2001/11: 1   Median :30.00   Median :195.0   Median : 6.410   Median :147.3
 2001/12: 1   Mean   :30.09   Mean   :199.1   Mean   : 6.565   Mean   :151.3
 2001/4 : 1   3rd Qu.:33.00   3rd Qu.:219.0   3rd Qu.: 7.225   3rd Qu.:171.0
 2001/5 : 1   Max.   :35.00   Max.   :363.0   Max.   :11.000   Max.   :250.1
 (Other):95
(snip)
```

各項目のデータを表示するには、データフレームのオブジェクト名と自動的につけられた変量名を **データフレーム名$変量名** で指定します。

```
> konetsu$date
  [1] 2001/04 2001/05 2001/06 2001/07 2001/08 2001/09 2001/10 2001/11 2001/12
 [10] 2002/01 2002/02 2002/03 2002/04 2002/05 2002/06 2002/07 2002/08 2002/09
(snip)
[100] 2009/07 2009/08 2009/09
102 Levels: 2001/04 2001/05 2001/06 2001/07 2001/08 2001/09 2001/10 ... 2009/09
```

それでは、gnuplot の時と同じようにプロットしてみます。今回も EPS で出力します。

```
> postscript("gnur.eps", horizontal=FALSE, height=5, width=5, pointsize=10)
> plot(konetsu$kWh, type="l", ylim=c(0,400), ann=F)
> par(new=T)
> plot(konetsu$m.3, type="l", ylim=c(0,400), ann=F, col="red")
> par(new=T)
> plot(konetsu$m.3.1, ylim=c(0,400), ann=F, col="blue")
> dev.off()
X11cairo
       2
>
```

出力された、EPS を LaTeX に取り込むと以下の図のようになります。

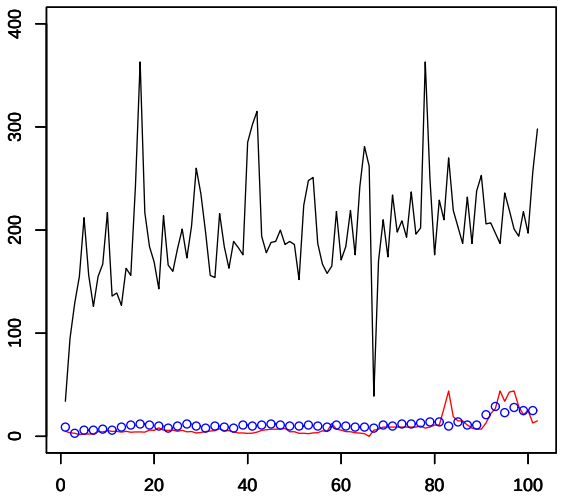

図 2　GNU R でのプロットの例

第 2 軸だけに表示させる方法が分からなかったので、Y 軸のスケールを全部同じにしなくてはならなかったりと、gnuplot に比べると若干見づらくなってしまいました。もうちょっと修行が必要そうです。

5.5　Debian の環境における注意点。

ところで、Debian では GNU R は CRAN のパッケージが deb パッケージ化されていると上述しましたが、GNU Octave も Octave-Forge プロジェクトという、CRAN に相当する拡張パッケージが、SorceForge で公開されています[*9]。CRAN 由来のパッケージと違い、パッケージのネーミングルールは決まっていません。ただし、現状では、Octave そのものに由来するパッケージは、`octaveX.X[-*]` という形で、バージョン X.X のブランチごとのパッケージ、名前になっています。一方、Octave-Forge をはじめ、拡張機能のパッケージは、`octave-*`というパッケージ名になっていますので、ある程度判断するための目安にはなるでしょう。

```
$ apt-cache search "GNU R" | wc -l
213
$ apt-cache search octave | wc -l
133
```

それぞれ、deb パッケージにする場合の考慮点を見てみましょう。

5.5.1　Octave の場合

Debian には Octave 関連パッケージのメンテナンスチーム Debian Octave Group が存在します[*10]。deb パッケージに未だなっておらず、deb パッケージにしたいソフトウェア、例えば、膨大な CRAN パッケージを deb パッケージ化したい場合、あるいは野良パッケージを作った場合などは、ITP する際に、Debian Octave Group[*11]にも CC を入れておくと良いでしょう。

また、README.Debian にも記載されていますが、Octave のドキュメントには PDF が含まれているため、`octave3.0-doc, octave3.2-doc` という名前で拡張パッケージとして提供されています。

Octave パッケージのビルドには、`octave-pkg-dev` パッケージという専用の環境が用意されています。Octave は

[*9] `http://octave.sourceforge.net/packages.html`
[*10] `http://pkg-octave.alioth.debian.org`
[*11] pkg-octave-devel@lists.alioth.debian.org

バージョン 3.0 からアドオンを作成する場合には、pkg.m system という環境を使ってインストールするようになっていますが、これを Debian 用のパッケージング環境として用意されています。

このパッケージを使ってパッケージを作るには、debian/control の Build-Depends に、octave-pkg-dev を、Depends に`${octave:Depends}`を追記します。

また、debian/rules に

```
include /usr/share/cdbs/1/class/octave-pkg.mk
```

を追記する必要があります[*12]。

また、debian/rules で、get-orig-source ターゲットも透過的に提供されています。これは前述の Octave-Forge プロジェクトからのパッケージの配布のみに作用しています。アップストリームのパッケージがこの SourceForge 以外からする場合は、自分で get-orig-source ターゲットを定義できます。octave-pkg-dev パッケージ特有のルールの実行を防ぐには、`''SOURCEFORGE = NO''` を debian/rules に設定する必要があります。

5.5.2 GNU R の場合

GNU R パッケージのビルドには、Octave と同様、`r-base-dev` パッケージという専用のビルド環境が用意されています。ビルド時の対応として、Debian パッケージポリシーの考慮が必要です。R の実行ファイルは、/usr/lib/R/bin/R.binary として分離されています。

私の環境では、上記のパスには以下のファイルが存在しますが、ここでは、/usr/lib/R/bin/Rscript が該当します。

```
$ file /usr/lib/R/bin/R*
/usr/lib/R/bin/R:       Bourne-Again shell script text executable
/usr/lib/R/bin/REMOVE:  ASCII text
/usr/lib/R/bin/Rcmd:    Bourne-Again shell script text executable
/usr/lib/R/bin/Rd2dvi:  ASCII English text
/usr/lib/R/bin/Rdconv:  ASCII text
/usr/lib/R/bin/Rdiff:   Bourne-Again shell script text executable
/usr/lib/R/bin/Rprof:   a /usr/bin/perl script text executable
/usr/lib/R/bin/Rscript: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.18, str
```

5.6 まとめ。

今回は、Debian で数学ことはじめ、ということで、Octave, GNU R, gnuplot を取り上げ、基本的な使い方と、ちょっと視点を変えて、拡張パッケージの Debian パッケージのお作法についてお話しました。使い込んでいけばかなり便利そうな印象があります。

数学、と言う観点からはかなりずれてしまいましたが、スプレッドシート地獄から脱出しましょう。

参考文献

[1] 山本昌志「gnuplot の精義 フリーの高機能グラフ作成ツールを使いこなす」

[*12] octave-pkg-dev パッケージの README には、include /usr/share/octave/debian/octave-pkg-dev.mk を追記するように記述されていますが、バージョン 0.5 でファイル名の変更をしており、ドキュメントの更新が追いついていないようです。

6 Debian autobuilder ネットワーク

岩松

Debian は現時点で 11 個の CPU アーキテクチャと 2 つのカーネルをサポートしています (i386, amd64, ppc, mips, mipsel, armel, alpha, ia64, s390, hppa, sparc, kfreebsd-amd64, kfreebsd-i386)。

パッケージが数万個あり、パッケージメンテナが Debian のサーバにアップロードすると、いつのまにかユーザが知らないうちに各アーキテクチャ向けに Debian パッケージがビルドされ、利用できるようになっています。これはどのような仕組みで動いているのでしょうか。もちろん、パッケージメンテナはすべてのアーキテクチャを持っているわけではないので（持っている人もいるみたいですが。）、パッケージメンテナが各アーキテクチャ毎にビルドして、アップロードしてわけではありません。Debian では **Autobuilder ネットワーク** と呼ばれる 各アーキテクチャ向けにパッケージが自動的にビルドされるシステムが行っています。Debian Project では重要なシステムの一つなのですが、表に出てくる事も少ないためどのような動きになっているのか知る人は少ないです。今回、Renesas SuperH(SH4) を移植するにあたりこれらの情報を入手したのでまとめてみました。また、新しくアーキテクチャを追加する時の方法と、ビルドテクニックについても説明します。

6.1 Autobuilder ネットワークのコンポーネント

Autobuilder ネットワーク は一つのソフトウェアで動いているわけではなく、いくつかのソフトウェアと porter と呼ばれる buildd のメンテナ、wanna-build のメンテナで構成されています。Autobuilder ネットワークの動きを説明する前に、各コンポーネントがどのような動きをするのか説明します。

6.1.1 dak

Debian Archive Kit。パッケージメンテナ/Buildd によってアップロードされたパッケージをチェック、管理するためのツールやデーモンが提供されています。

6.1.2 quinn-diff

パッケージメンテナによって新しくアップロードされたソフトウェアのアーキテクチャ依存のパッケージ (Architecture: any) とアーキテクチャ非依存のパッケージ (Architecture: all) をチェックし、前者を wanna-build のデータベースに登録するツールです。これはソースパッケージだけでなく、バイナリパッケージもチェックします。Autobuilder ネットワークの過程では アーキテクチャ非依存のパッケージはビルドされません。

パッケージ情報: `http://packages.qa.debian.org/q/quinn-diff.html`

6.1.3 wanna-build

buildd とデータのやりとりをするためのインターフェイスです。各アーキテクチャ毎にパッケージのデータベースを持ちます。buildd は wanna-build のデータベースを持つサーバに ssh でログインし、wanna-build コマンドを使ってデータベースをコントロールします。

パッケージ情報: `http://packages.debian.org/sid/wanna-build`

6.1.4 buildd

wanna-build と データをやりとりするデーモンやパッケージをアップロードするツール一式を指します。ツールには以下のものがあります。

パッケージ情報: `http://packages.debian.org/sid/buildd`

- buildd
  buildd デーモン。wanna-buildd のデータベースを定期的にチェックし、ビルドが必要なパッケージをパッケージビルドキューとして管理します。そして、sbuild へのビルド指示を行います。
- buildd-mail
  メールキューディレクトリにあるデータを処理するプログラム。ビルドに成功したパッケージ (*.deb, *.udeb) とパッケージ情報 (*.dsc, *.changes, etc) をアップロードキューディレクトリに移動させます。
- buildd-mail-wrapper
  buildd-mail-wrapper は porter から送られたメールをメールキューディレクトリに格納し、buildd-mail を呼び出すプログラムです。入力データとして、メールデータそのものを要求するので、procmail など合わせて利用します。
- buildd-update-chroots
  パッケージのビルドを行う chroot 環境をアップデートおよびクリーンアップするプログラムです。buildd が動作している場合には buildd を停止し、chroot をアップーデートします。アップデート完了後、再度 buildd を起動させます。クリーンアップには debfoster と呼ばれるゴミパッケージを消すプログラムが利用されています。
- buildd-uploader
  アップロードキューディレクトリにあるパッケージをアップロードするプログラムです。dupload を使い、GPG サインされている *.changes ファイルをパーサし、パッケージをアップロードします。cron 等で定期的に呼び出して利用します。
- buildd-watcher
  buildd の動きをチェックするプログラムです。cron 等で定期的に呼び出して利用します。

正式な buildd では https://buildd.debian.org/apt/pool/からダウンロードしたパッケージを使う必要があります。互換性維持のため、手を加えた sbuild を使う必要あるためです。

6.1.5 sbuild

実際にパッケージをビルドするプログラムです。buildd から呼ばれます。パッケージのソースコードを取得し、ビルド結果を porter と wanna-build に送信するまでの処理を行います。pbuilder と機能がかぶりますが、sbuild はアーキテクチャ依存部分のビルドに特化したパッケージビルドシステムになっています。schroot を利用することにより、lvm などディスク管理機能にも対応もされています。

パッケージ情報: `http://packages.debian.org/sid/sbuild`

正式な buildd では `https://buildd.debian.org/apt/pool/` からダウンロードしたパッケージを使う必要があります。互換性維持のため、手を加えた sbuild を使う必要あるためです。

6.1.6 schroot

sbuild から呼ばれる chroot 代替プログラムです。chroot の高機能版です。

パッケージ情報: `http://packages.debian.org/sid/schroot`

6.1.7 porter

buildd をメンテナンスする人です。ビルド結果をメールとして受信し、その結果を buildd に対して返信します。ビルドに成功したパッケージのメールには GPG でサインして返信します。返信する際には、送られてきたメールから、*.changelog の部分を抜き出して、その抜き出した部分にサインを行います。抜きだしの部分にはスクリプトを使って処理するのですが、現在のところ mutt のスクリプトしかないので、ほとんどの porter は Mutt ユーザです。GPG によるサインを自動化すればよいのですが、セキュリティの観点から porter がサインをする仕組みを取っています。

ビルドに失敗した場合のメールの書式も決まっています。以下に書式を示します。

- ビルドに失敗した場合

```
failed
理由 (BTS の番号など)
```

- パッケージの依存関係の問題

```
dep-wait 依存するパッケージの名前とバージョン
```

- 再ビルド申請

```
give-back
```

- パッケージがアーキテクチャ依存の場合

```
not-for-us
```

*13

6.1.8 wanna-build メンテナ

メールサーバ障害や、buildd の障害でうまくパッケージがビルドできない事があります。この場合には wanna-build のデータベースにアクセスし、パッケージのステータスを元に戻す作業が必要になる場合があります。この場合、porter が wanna-build のデータベースにアクセスして変更するわけではなく、wanna-build メンテナが依頼を受けて、変更処理を行います。

6.2 Autobuilder ネットワークの動き

さて、Autobuilder ネットワークのコンポーネントの説明をしたので、次に各コンポーネントの連携について説明します。先で説明した各コンポーネントの連携図を図 3 に示します。

6.3 パッケージの状態遷移

パッケージは常になにかしらの状態をステータスを持っています。ステータスは wanna-build で管理する DB で保持されています。ステータスは以下のものがあり、図 4 のように変化します。

- BD-Uninstallable

*13 アーキテクチャリストに入っていないだけの場合は、failed を指定します。

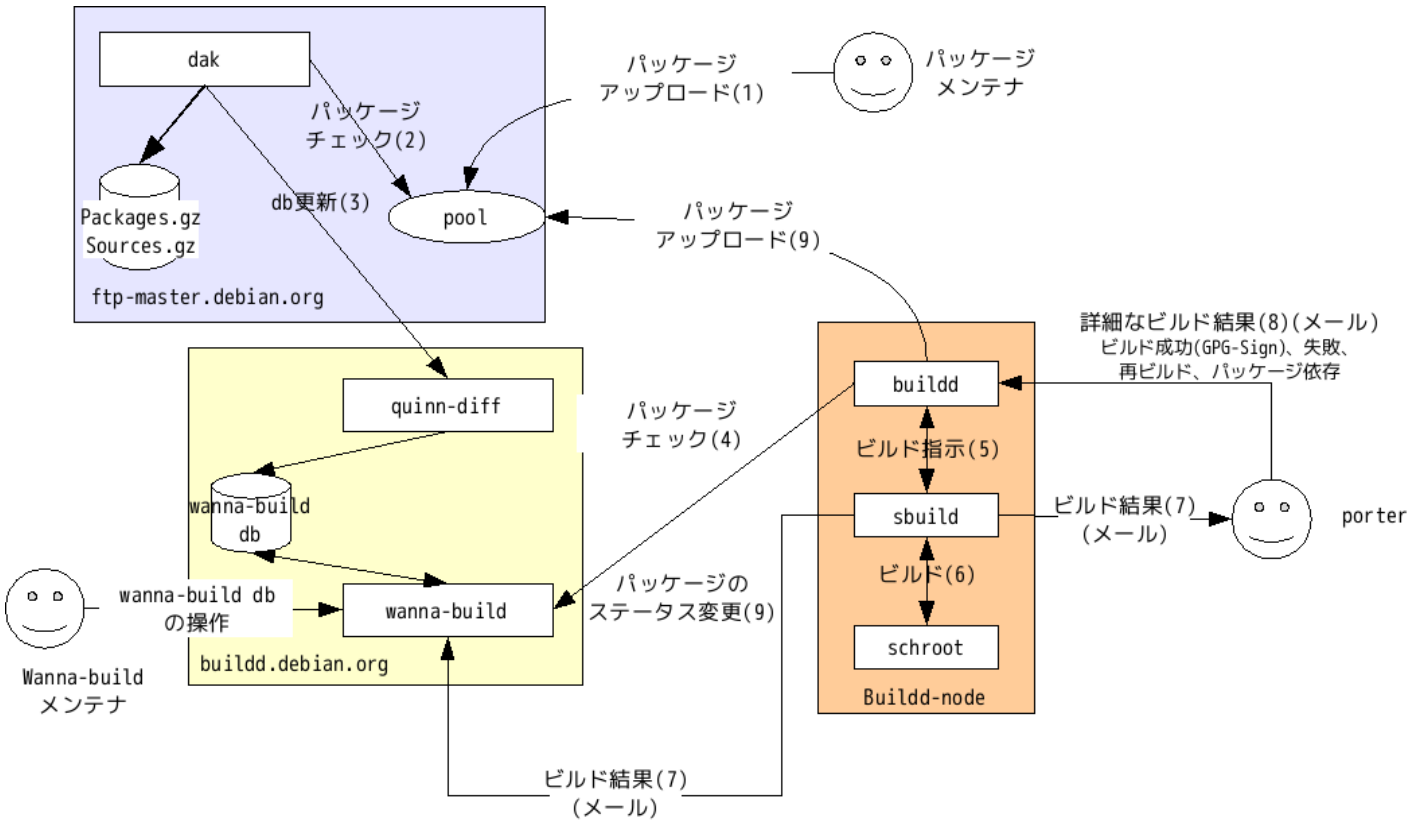

図 3　コンポーネント連系図

パッケージのビルドに足りないパッケージがある状態。

- Needs-Build
  パッケージがビルド可能な状態。パッケージビルド依存関係が解消されてたり、ビルドの状態から戻された (given-back) 場合にこの状態になります。
- Building
  Buildd によってパッケージがビルドされている状態。
- Uploaded
  Porter によってパッケージに GPG サインされ、アーカイブにアップロードされた状態。
- Installed
  アーカイブにインストールされた状態。
- Dep-Wait
  パッケージのビルド依存関係待ちの状態。
- Failed
  パッケージがビルドに失敗した状態。
- Not-For-Us
  対象のアーキテクチャではビルドできないパッケージの状態。porter が wanna-build 経由でデータベースに登録します。現在、wanna-build のデータベースとは別にアーキテクチャ依存のリスト[*14]を作成中です。
- Failed-Removed
  Failed 状態のパッケージが一時的に削除されているという状態。

[*14] http://buildd.debian.org/git/packages-arch-specific.git

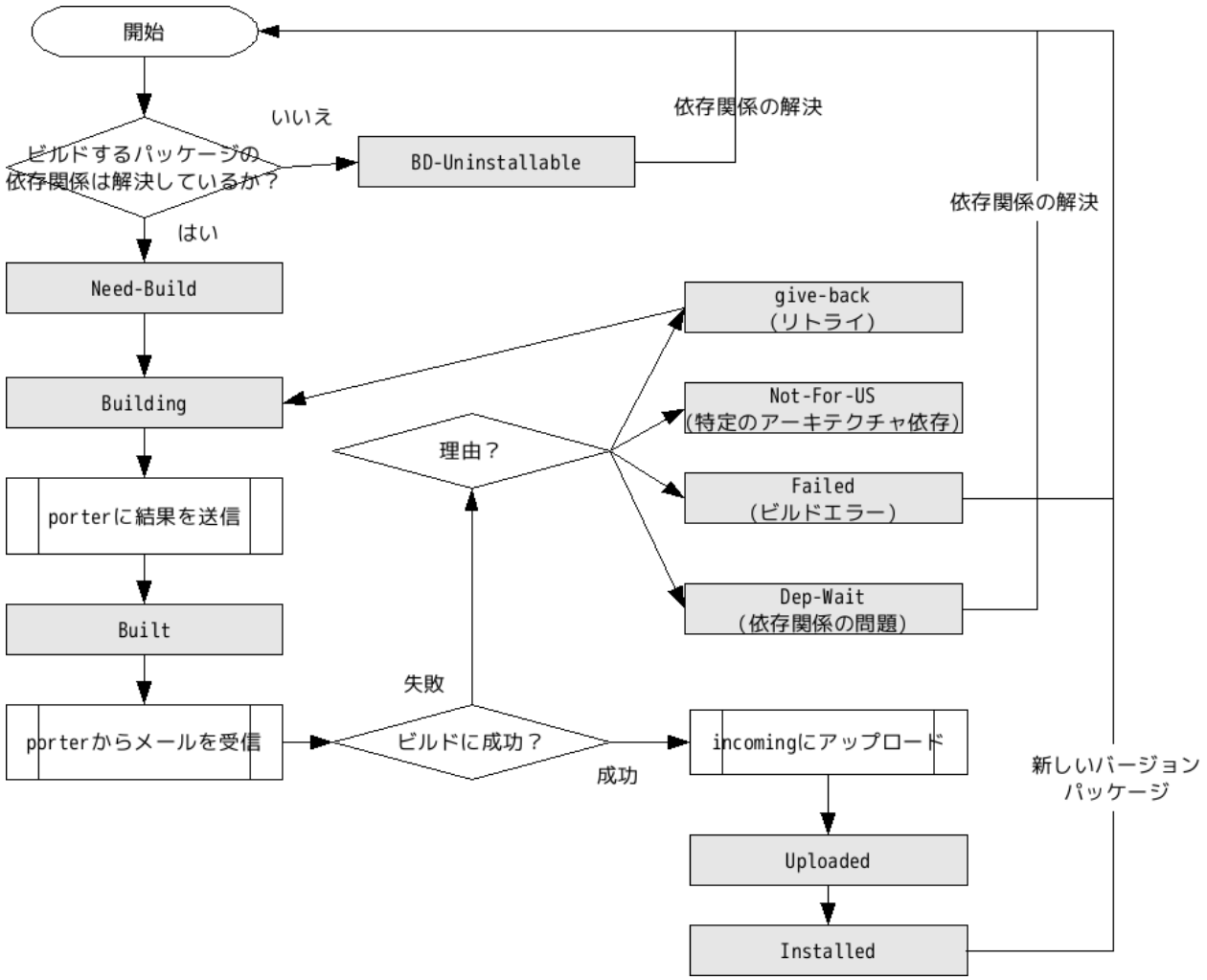

図 4　パッケージステータスの状態遷移

6.4　ビルドの優先順位

パッケージのビルドは基本的に Need-Build のステータスになった順かつパッケージ名 ASCII 順でビルドされますが、それとは別にソースパッケージのプライオリティとソースパッケージのセクション毎に優先順位が設定されています。現在では、以下の優先順位になっています。

6.5　新しいアーキテクチャをサポートする

昔は、ポーティングしたい人が quinn-diff から buildd(buildd-node) まで全て構築する必要がありました。この最大のネックは wanna-build データベースで、これは最新のものを使う必要があったのですが、データベースにアクセスするにはサーバーのアカウントが必要だったため、構築が困難でした[*15]。しかし、今は新しいアーキテクチャを追加するためのサポート用サービス debian-ports.org が用意されています。このサービスでは、ポーティングしたいアーキテクチャ用の wanna-build 用アカウント と DB, quinn-diff のサービスを提供できるようになっています。これにより、移植したい人は buildd-node のみを用意し、debian-ports の管理者に登録申請をするだけで新しいアーキテクチャをサポートできるようになりました。現在、4 のアーキテクチャが debian-ports.org を使ってポーティング

[*15] 構築は可能だが、タイムラグが発生する。その理由として buildd が参照するリポジトリは最新のリポジトリ情報が反映される buildd 専用のため

Source Priority	値
required	-5
important	-4
standard	-3
optional	-2
extra	1
unknown	-1

表 1　ソースパッケージプライオリティ毎の優先順位

libs	-200
debian-installer	-199
base	-198
devel	-197
shells	-196
perl	-195
python	-194
graphics	-193
admin	-192
utils	-191
x11	-190
editors	-189
net	-188
mail	-187
news	-186
tex	-185
text	-184

表 2　ソースパッケージセクション毎の優先順位:その 1

web	-183
doc	-182
interpreters	-181
gnome	-180
kde	-179
games	-178
misc	-177
otherosfs	-176
oldlibs	-175
libdevel	-174
sound	-173
math	-172
science	-171
comm	-170
electronics	-169
hamradio	-168
embedded	-166
unknown	-165

表 3　ソースパッケージセクション毎の優先順位:その 2

アーキテクチャ名	進捗
avr32	65.52%
hurd-i386	64.46%
m68k	53.00%
sh4	85.43%

表 4　debian-ports.org でサポート中のアーキテクチャ

を進めています。

6.6　loop-depends の対応方法

いくつかのパッケージには loop-depends になっているものがあります。loop-depends とは、ビルド依存がループしているパッケージの状態を言います。例えば、libpang-perl パッケージ (図 5) は libgtk2-perl パッケージ図 6 にビルド依存しています。しかし、libgtk2-perl は libpang-perl パッケージに依存しています。

```
Source: libpango-perl
Section: perl
Priority: optional
Build-Depends: debhelper (>= 7.0.50), libcairo-perl (>= 1.000), libextutils-depends-perl (>= 0.300),
  libextutils-pkgconfig-perl, libglib-perl (>= 1:1.220), perl, xvfb, libpango1.0-dev,
  xauth, xfonts-base, quilt, libgtk2-perl (>= 1:1.220)
                             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

図 5　libpango-perl のビルド依存

これではいつになってもビルドできません。これはバグとして扱われるのでは？と思いますが、libpang-perl をビ

```
Source: libgtk2-perl
Section: perl
Priority: optional
Build-Depends: debhelper (>= 7.0.50), quilt (>= 0.46-7), perl, libextutils-depends-perl (>= 0.300),
 libextutils-pkgconfig-perl (>= 1.03), libgtk2.0-dev (>= 2.6.0), libglib-perl (>= 1:1.220),
 libcairo-perl (>= 1.00), xvfb, xauth, xfonts-base, hicolor-icon-theme,
 libpango-perl (>= 1.220), shared-mime-info
 ^^^^^^^^^^^^^^^^^^^^^^^^^
```

図 6　libgtk2-perl のビルド依存

ルドする段階で libgtk2-perl を使ったテストを行うため、libgtk2-perl が必要なのです。Debian ではソフトウェアのビルドに実行されるテストは行うことが推奨されているので、このようにビルド依存関係がループしています。では、どのようにしてビルドすればよいのでしょうか。実は、porter が手を加えてビルドする必要があります。しかし、手を加えた場合は正式なパッケージとして扱われません。このようなパッケージに遭遇した場合には、unreleased ディストリビューションにパッケージをアップロードし、そのパッケージを使って再度ビルドを行うという方法が推奨されています。

以下に例として libpang-perl をビルドする場合の手順を説明します。

1. buildd が動作していない別マシンでの作業
   (a) libpango-perl のソースパッケージをダウンロードします。
   (b) debian/control ファイルの Build-depends から libgtk2-perl を削除します。
   (c) debian/changelog をアップデートします。
   Debian version は既存のバージョンに +arch-name などをつける必要があります。(実際の Debian package と区別するため。) ディストリビューション名は unreleased に変更します。debian/changelog の変更者も変更する必要があります。
   (d) debuild -m"Your name <your-name@example.org>" などとしてパッケージをビルドします。
   libpango-perl は libgtk2-perl が見つからない場合には、テストを無視します。:-)
   (e) ビルドに成功したら、dupload を使ってパッケージアップロードします。
   (f) dak で処理されれば、unreleased ディストリビューションから libpango-perl が利用できるようになります。
2. schroot 内での作業
   (a) schroot 環境の apt-line に unreleased を追加します。
   (b) schroot 内で apt-get update を実行します。
3. libgtk2-perl がビルドできるようになります。
4. libgtk2-perl ができたら、unreleased ディストリビューションから libpango-perl を消してもらいます。
5. libpango-perl が 本来の Debian Version のものでビルドされます。

毎回この方法がうまくいくとは限りません。例えば、依存するパッケージや提供されるパッケージの機能によっては configure のオプションを変更する場合もあります。このあたりはパッケージメンテナか、debian-wb@l.d.o に直接聞いたほうがよいです。聞かないとわからないというのは問題なので、README.Debian あたりに書いてもらうように計画中です。

6.7　まとめ

auto-builder ネットワークは Debian 公式開発者でさえあまり触ることのないシステムです。ブラックボックスなところが以前からあったと思うのですが、今回の説明で少しはわかる部分が増えたかなと思います。個人的には auto と謳っておきながら実は人が介入していたりするのはけっこう意外でした。auto-builder ネットワークの奥深くに入り込みたい人はぜひ Debian/SH4 のポーティングに参加していただければと思います。

Debian 勉強会資料

2009 年 11 月 14 日　　初版第 1 刷発行
東京エリア Debian 勉強会 （編集・印刷・発行）