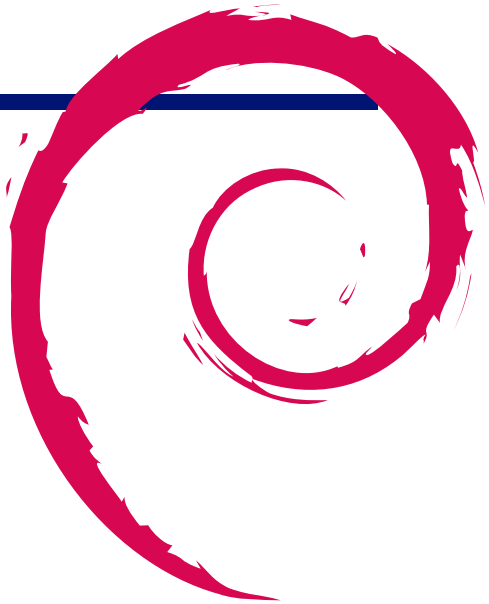
第 88 回 東京エリア Debian 勉強会資料

月刊 Deb専

日本唯一のDebian専門月刊誌

2012 年 5 月 19 日

特集: プログラミングの本を読んでみた



1 はじめに

上川 純一

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

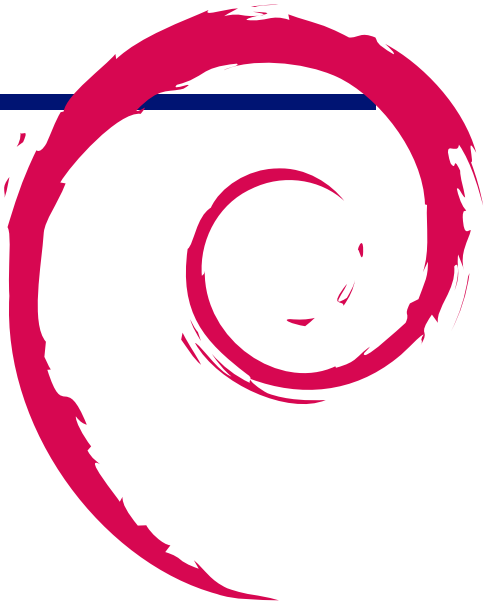
Debian 勉強会の目的は下記です。

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

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

デビアン勉強会

目次



2 事前課題

上川 純一

今回の事前課題は以下です:

1. Debian 勉強会参加者に紹介したい書籍を 1 冊以上挙げて、内容を簡単に紹介してください(特に技術書には限りません)。
2. あなたが何かスクリプト言語をプログラミング初心者にお勧めするとして「 その言語を選んだ理由」と「 最初の一歩として案内する書籍/サイト」を教えてください。

この課題に対して提出いただいた内容は以下です。

2.1 BeatenAvenue

1. おすすめ本:
「 windows プロフェッショナルゲームプログラミング (全 2 巻?)」 DirectX 関連の内容もあった気がしますが win32API についての内容が多かった気がします。昔の本ですがタスク処理の考え方など現在でも通用する部分は多いと思います。この本を買ってから C++ の勉強を始めたこともあって・・・ 個人的に思い出がたくさんあります。
「 DirectX 逆引き大全 500 の極意」 入門的な優しい解説から一歩踏み込んだプラスアルファまで揃っています。残念ながら絶版で図書館から借りて読みました。 DirectX9 の解説書では一番よいものかと思います。
「 GameProgrammingGems(シリーズ)」 海外のゲームプログラマの方々が書いた記事をまとめた本。 3 巻だったかと思いますが Naughty Dog の方が書いた"Jak and Daxter: The Precursor legacy"でのマップ移動処理についての内容が好きです。お値段以上。
2. 初心者におすすめするスクリプト言語: Debian とは全く関係ないですが DOS バッチファイルと ExcelVBA のちょっとした使い方は覚えないと事務仕事が進みません。解説サイトも多いので付きっきりで教える必要もあまりないです。 Linux だと bash のスクリプトなんでしょうか。私が初心者なのでそれしか触っていません・・・ 。

2.2 amotoki

1. 昨日友人から勧められて気になっているのが「 情熱プラグラマー」です。自分の人生を自分で切り開いていくために必要なことが分かりやすく整理されているけど、自分を前に進めてくれる情熱を感じたとのこと。さっそく注文した。
2. 今おすすめするとしたら Python をお薦めします。オブジェクト指向も書きやすいし、ライブラリも充実していて、マニュアルも実例がそろっているので、プログラミングを学んでいく上でよいと思います。大きめの OSS プロジェクトでも使われているので、知っておいて損することもありません。最初の一歩としては「 Python チュートリアル」がよいと思います。今でもときどき見ることがあります。

2.3 吉野 (yy_y_ja_jp)

1. Git によるバージョン管理 実際のプロジェクトでの Git の利用法が書かれているようです.
2. シェルスクリプト気軽に使えるからです. bash(1), dash(1)

2.4 dictoss(杉本 典充)

1.「 インテル スレッディング・ビルディング・ブロック – マルチコア時代の C++ 並列プログラミング」オライリー・ジャパン、 James Reinders 著 Intel が開発し現在は GPLv2 で公開している C++ の並列計算用ライブラリ「 Threading Building Blocks」の解説を行っている本。マルチコア時代の中で複数の CPU コアを効率的に使用して計算性能を上げるための知識が詰まっている。あくまで単一ノードで計算性能を上げるための手法であり、複数ノードで計算性能を向上させるクラスタリング技術の話ではないので注意。
2. お勧めは python。理由はプログラミング初心者ということで開発者によって書き方に差異が出にくい分 web で紹介され

ているコードに癖がなくとっつきやすいため。おすすめサイトは、うーん、なんでしょ？ 自分は別の言語が書けるようになってから python を始めたので”http://www.python.jp/doc/release/”を確認します。

2.5 kamonshohei

1. シェルスクリプトシェルスクリプト基本リファレンス
2. シェルスクリプトでしょうか。リナックスのコマンドだけで、ちゃちゃっと実装できる手軽さがいいです。最初の一歩で案内する書籍は 1 であげたシェルスクリプト基本レファレンスです。

2.6 emasaka

1. ケン・スミス「誰も教えてくれない聖書の読み方」。聖書に書かれているそのままの文面を真面目に読んでユーモラスに紹介（？）している本（Debian 関係ない）
2. Bash と書こうかと思ったけど Ruby。理由は「オブジェクト指向を使っても使わなくてもいい」ではなくて「オブジェクト指向を強制される」から。書籍は「たのしい Ruby」

2.7 本庄

1. Debian 勉強会参加者に紹介したい書籍を１冊以上挙げて、内容を簡単に紹介してください。定番ですが『ハッカーと画家』とかどうでしょう。オタクの怨念が込められています。親として子供の将来に不安を感じます。
2. あなたが何かスクリプト言語をプログラミング初心者にお勧めするとして「その言語を選んだ理由」と「最初の一歩として案内する書籍／サイト」を教えてください。PHP がおすすめです。ほかに比べて仕事が多そうという理由です。多いかどうかは実際のところわかりませんが、

http://www.google.co.jp/trends/?q=PHP,+Perl,+Python,+Ruby,+Javascript,+Haskell&ctab=0&geo=jp&geor=all&date=ytd&sort=0

ここ見ると多そうです。案内する書籍として定番はマンモス本だと思いますが、読んだことはありません。そして古い情報かもしれません。以前、とある PHP 方面の方とお話しする機会があり、オライリーの本はちょっと…的なことを話したら、「ああいう本もいいかなと思ってます」といわれました。翻訳者でした。

2.8 henrich

1.「入門 Debian パッケージ」。書名から内容が分かるかとは思いますが、Debian パッケージの作り方の書籍です。続刊が期待されます
2. Python を選びました。Perl は人によってとても書き方が変わるところがあまり嬉しくなく、Ruby はバージョン間の移行が若干乱暴に感じられたので。何度もプログラミングに挫折している私ですが、今回 Python を学ぶのに選んだ「初めてのコンピュータサイエンス」がとても良い書籍でした。

2.9 野島 貴英

1.「イノベーションのジレンマ」（ISBN10:4798100234）と、「のうだま」（ISBN10:4344015959）。「イノベーションのジレンマ」は、技術革新が既存のものをぶち壊していく過程において、既存技術において優秀な組織であればあるほど技術革新についていけなくなってしまう現象を理詰めで説明した本。「のうだま」は人の行動においては、実は習慣が先でやる気は後からついてくるものであるという事を説明した本。
2. プログラミング初心者には今時の状況から javascript/HTML5 を勧めたいのですが、肝心の自分が未評価。最初の一歩はゲーム遊びたさにプログラム覚えた経験を元に、http://wise9.jp/ と、http://enchantjs.com/ がおすすめなのかな?

2.10 yamamoto

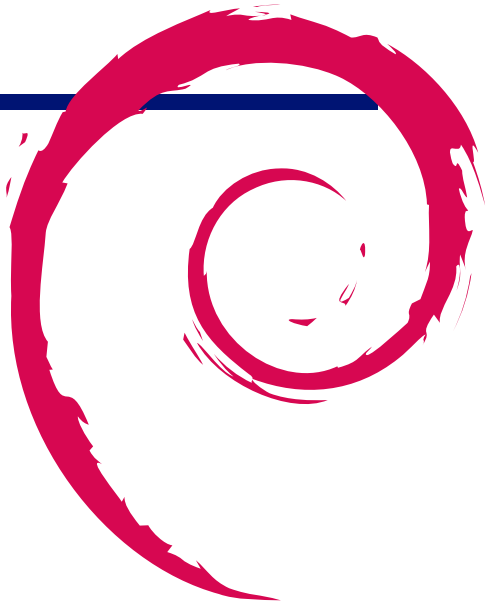
1. Debian 勉強会参加者に紹介したい書籍を１冊以上挙げて、内容を簡単に紹介してください（特に技術書には限りません）。

有名だと思うので、紹介するほどのことはないかもしれませんが、「入門 UNIX シェルプログラミング-シェルの基礎から学ぶ UNIX の世界　Bruce Blinn 著・山下哲典 訳」を愛読しています。まあ、シェルスクリプトプログラミングをシェルプログラミングと言っているタイトル (勿論シェル自体の解説もありますけどね) はアレですが、B シェルのスクリプトを書くときには良く開いています。

2. あなたが何かスクリプト言語をプログラミング初心者にお勧めするとして「その言語を選んだ理由」と「最初の一歩として案内する書籍／サイト」を教えてください。

#!/bin/sh 万歳! どこでも大概動くからおすすめ。B シェルの作法に従っていれば、今の dash なら大体動く? みたい (未確認)。だめなら　#!/bin/bash で。とかいいながらも、おいらのシェルスクリプトには外部コマンドバリバリ入れてますけど。(うひ

上記の書籍でもいいですが、ぐぐるさんにお伺いしたら、参考になるサイトは山ほど出てくるでしょう。



3 最近の Debian 関連のミーティング報告

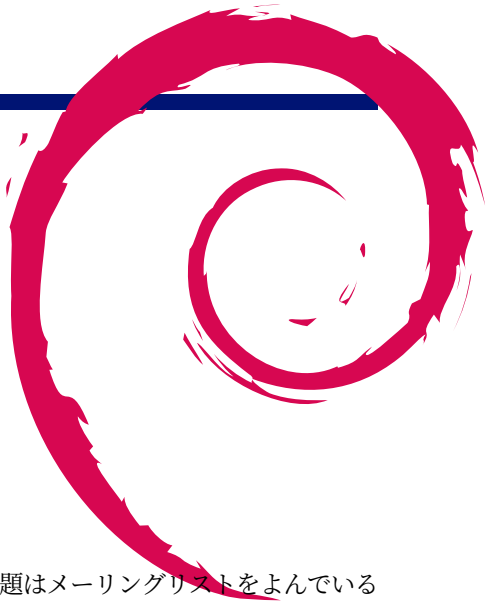
上川 純一

3.1 東京エリア Debian 勉強会 87 回目報告

2012 年 4 月の Debian 勉強会は新宿での開催でした。

Android tablet から Debian をネイティブブートさせるという話題。 debian-installer-6.0-netboot-armel に ARM のインストーラーがはいっていて、 qemu-system-arm の起動に活用できるようになっているというのに感動。起動イメージの試行錯誤に QEMU を使えるというのが良いですね。

Debian での Node.js の最近の事情について上川が発表しました。先月はパッケージがちゃんとインストールできませんでしたが、今月はできるようになってますね。



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

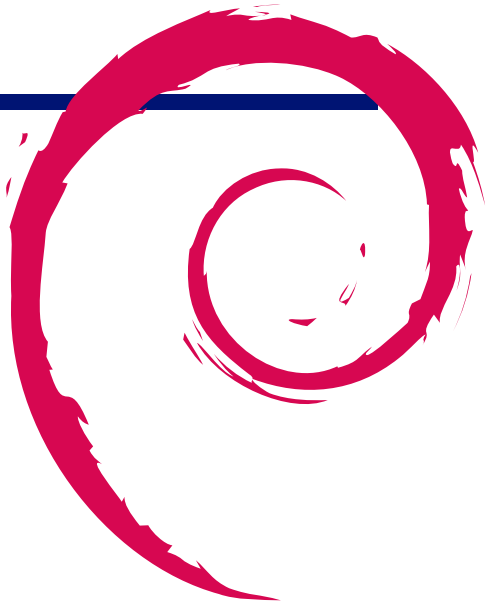
今回の出題範囲は `debian-devel-announce@lists.debian.org` や `debian-devel@lists.debian.org` に投稿された内容と Debian Project News からです。

問題 1. Debian installer 7.0 alpha 1 のリリース日は

☐ A 5/13
☐ B 6/13
☐ C 4/13

問題 2. 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



5 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。

5.1 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[*1]。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。

5.2 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「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)。

[*1] 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。

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 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[*2]。

```
$ 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 [*3] 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[*4]。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)。

[*2] 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。

[*3] http://packages.qa.debian.org/p/python2.4.html

[*4] 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 !kfreebsd-i386
!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…。

5.3 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? 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。

5.4 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[*5]。

```
$ 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
```

5.5 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
$ 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。

5.6 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
```

[*5] 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。

5.7 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。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。

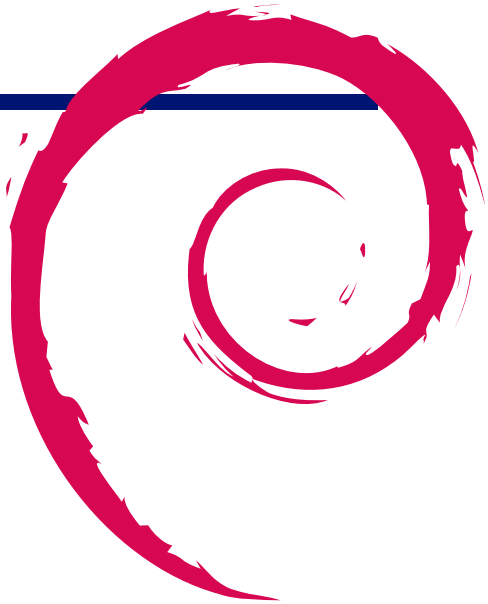
```
$ cp package-list.txt /srv/chroot/lenny-i386/tmp/
$ LANG=C sudo chroot /srv/chroot/lenny-i386 /bin/bash
# dpkg --set-selections < /tmp/package-list.txt
# apt-get dselect-upgrade
```

```
$ 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。

5.8 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。



6 coffeescript を使ってみた

上川純一

6.1 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。

6.2 coffeescript を使うと何がうれしいのか

6.3 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。[*6]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。[*7]

```
$ 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
```

6.3.1 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 の type=text/coffeescript となっているところを処理してくれるようです。例として HTML はこのようになるようです。 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

[*6] 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。
[*7] 2012 年 5 月 16 日時点では wheezy に入っておらず、 sid のみ。

coffeescript プログラムをコンパイルして一つの js 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。

```
<!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>
```

コンパイルするには `coffee -c` を利用します。

```
$ coffee -c class.coffee
```

6.3.2 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
```

6.3.3 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。

サーバは基本的には CLI プログラムとして実行すれば良いので、 Coffeescript プログラムとして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 + '/'
```

6.4 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。

6.5 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

Debian 勉強会資料

2012 年 5 月 19 日　　初版第 1 刷発行

東京エリア Debian 勉強会（ 編集・ 印刷・ 発行）