Grand Unified Debian

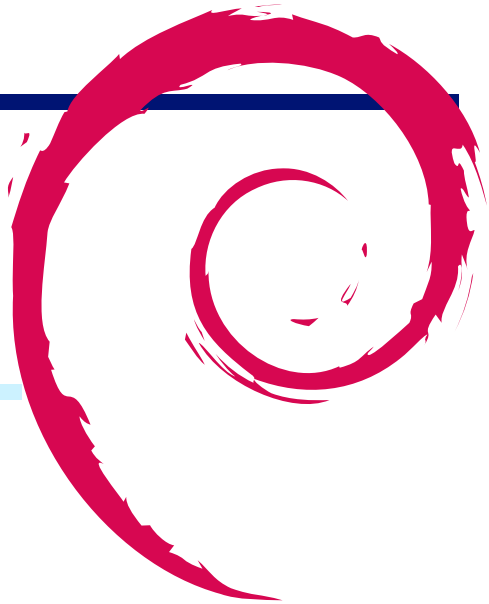
銀河系唯一のDebian専門誌

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 2013 年夏号 2013 年 08 月 12 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。

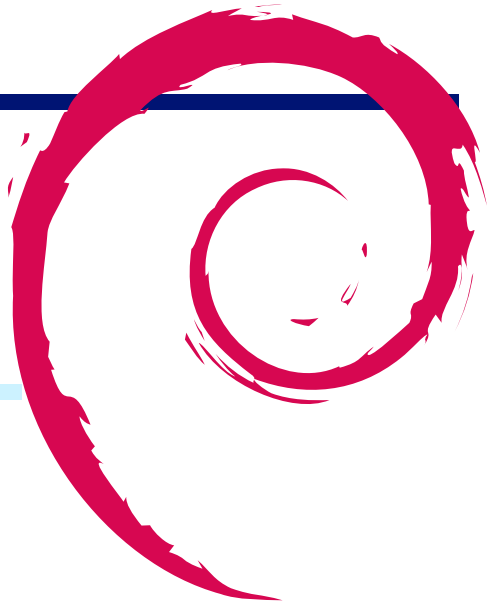
1.2 関西 Debian 勉強会

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 Debian と Ubuntu の違いを知ろう

西田

2.1 この話の趣旨

「Linux はとりあえず人気のある Ubuntu を使っている」という方は多いのでは無いでしょうか。

この話は Debian と Ubuntu の違いを挙げ、どちらがみなさんに適しているか考えてみていただくことを目的としています。また違いを知るのはお互い (ユーザーと開発者) の文化の違いの理解や相互の情報交換による Debian package の質の向上にもつながるかと思います。

2.2 support される architecture

package などの違いについて考える前に、まず使おうとされている hardware が support されていないとどうしようもありません。

Ubuntu(Raring Ringtail) では Intel x86, AMD64(Mac に特化したものもあり),OMAP(ARM) を support しています。Debian ではこれらに加え、ARM 複数種、MIPS、powerpc、sparc、その他 (その他の architecture については無知であるため名称略) を support しています。

大部分の方は x86, amd64 で Debian, Ubuntu を使うかとは思うのですが念のため。

2.3 package について

みなさんが最も気になるのは (PC 用の)package がどう異なるのか、ではないでしょうか。ご存知のように Ubuntu の package は Debian をベースとしたものです。ですが意外とその package が

- 必ずしも binary 互換性があるわけでは無いこと
- Ubuntu の package は Debian の package をベースに Ubuntu 環境で `recompile` したものがほとんどであること
- Debian 由来の package のソースが具体的にどのように異なっているのか、どうするとその違いが確認できるのか
- Linux kernel はどのように異なっているか

といった所までご存知で、用いている Debian package の違いをいつも確認されている方は少ないのではないでしょうか。

下記では上記の点に関して説明します。

2.3.1 なぜ Ubuntu では Debian package を recompile しているのか

まず Ubuntu 自体は Ubuntu に default のツールチェーン (gcc, libc などといった build に欠かせない GNU の library の集まりを意図しています) で完全に build できるようになっているのですが、この要である gcc に元となる Debian と同じ version のものを用いる考えがありません。Ubuntu では “新しい version の gcc はよりよい binary を作るはず” という考えのもと出来る限り新しい gcc を使おうとしているようです。そしてその gcc はまず Debian より新しい version のものとなるようです。(たとえば Ubuntu-Raring の gcc は 4.7.3、Debian-wheezy の gcc は 4.7.2 です)

つまり Debian から source をもってきているのですが build 環境を合わせる考えはありません。そのため Ubuntu と Debian 間で package に binary 互換性は保証されていません。こういったことから Ubuntu はすべての Debian package を新しい version のツールチェーンで build しなおしています。

これは Ubuntu の package repository 区分で言うところの “Main” だけの話では無く “universe”(community によって support されている package の分類) についても同様で、たとえば Ubuntu の “Main” に含まれる Python の version が元となる Debian の Python より新しい場合、“universe” に含まれる Python package も新しい version の Python との整合性を確保するために rebuild されます。

2.3.2 Ubuntu と Debian で異なっている package の確認方法

前項のように互換性を確保し rebuild できれば、Ubuntu でも元となる Debian と同じ version の package ができ問題は無いのですが、まれにそうはいかないものもあるのか、それ以外の事情があるのか Ubuntu 側で package の修正を行う必要があるものが出てきます。

こういった package は `https://launchpad.net/ubuntu/raring/+localpackagediffs?field.package_type=all` で確認できます。ここでは Raring とその package の元となる Wheezy の package の間で違いのあるものが列挙されており、それらを下記の 3 区分で filter することが可能になっています。

- 無視できない違いがあったもの (古い version を用いる、ubuntu なりの変更を入れる必要があったもの)
- Wheezy より新しい version でなら上記のような問題は無かったもの
- Wheezy と同じ version で問題無かったもの

現在 (2013 年 5 月 26 日) では 計 17261 件のうちそれぞれ

- 154 件
- 5908 件
- 11080 件

となっています。

Ubuntu 側の “ツールチェーンにできるだけ新しいものを使う” という方針が影響しているのか “Wheezy より新しい version でなら上記のような問題は無かったもの” が多いようです。

2.3.3 debdiff コマンドを用いた Ubuntu, Debian package の比較

debdiff コマンドを用いると 2package 間の情報の比較ができます。debdiff コマンドは devscripts package を install することで使えるようになります。 比較したい 2package の dsc, orig.tar.gz, debian.tar.gz, file を適当な directory に download 後、下記のように debdiff します。

```
kozo2@ubuntu:~/tmp$ sudo aptitude -y install devscripts
kozo2@ubuntu:~/tmp$ ls
aptitude_0.6.8.1-2ubuntu2.debian.tar.gz  aptitude_0.6.8.1.orig.tar.xz      aptitude_0.6.8.2-1.dsc
aptitude_0.6.8.1-2ubuntu2.dsc            aptitude_0.6.8.2-1.debian.tar.gz  aptitude_0.6.8.2.orig.tar.xz
kozo2@ubuntu:~/tmp$ debdiff aptitude_0.6.8.2-1.dsc aptitude_0.6.8.1-2ubuntu2.dsc
gpgv: Signature made Wed 07 Nov 2012 02:54:14 PM JST using RSA key ID 4D6E25A8
gpgv: Can't check signature: public key not found
dpkg-source: warning: failed to verify signature on /home/kozo2/tmp/aptitude_0.6.8.2-1.dsc
gpgv: Signature made Tue 26 Feb 2013 05:28:12 PM JST using DSA key ID 0F932C9C
gpgv: Can't check signature: public key not found
dpkg-source: warning: failed to verify signature on /home/kozo2/tmp/aptitude_0.6.8.1-2ubuntu2.dsc
```

中略

```
--- aptitude-0.6.8.2/src/generic/apt/pkg_changelog.cc   2012-11-05 00:24:56.000000000 +0900
+++ aptitude-0.6.8.1/src/generic/apt/pkg_changelog.cc   2012-08-04 18:33:38.000000000 +0900
@@ -20,7 +20,6 @@
#include "pkg_changelog.h"

#include "apt.h"
-#include "config_signal.h"
#include "download_queue.h"

#include <generic/util/job_queue_thread.h>
@@ -543,18 +542,12 @@
else
realver = source_version;

-              // WATCH: apt/cmdline/apt-get.cc(DownloadChangelog)
-              string server = aptcfg->Find("APT::Changelogs::Server",
-                                           "http://packages.debian.org/changelogs");
-         string path = cw::util::ssprintf("pool/%s/%s/%s/%s_%s",
+         string uri = cw::util::ssprintf("http://packages.debian.org/changelogs/pool/%s/%s/%s/%s_%s/changelog",
realsection.c_str(),
prefix.c_str(),
source_package.c_str(),
source_package.c_str(),
realver.c_str());
-              string uri = cw::util::ssprintf("%s/%s/changelog",
-                                              server.c_str(),
-                                              path.c_str());
LOG_TRACE(logger,
"Adding " << uri
<< " as a URI for the changelog of " << source_package << " " << source_version);
diff -Nru aptitude-0.6.8.2/src/generic/apt/tasks.cc aptitude-0.6.8.1/src/generic/apt/tasks.cc
--- aptitude-0.6.8.2/src/generic/apt/tasks.cc   2012-11-05 00:24:56.000000000 +0900
+++ aptitude-0.6.8.1/src/generic/apt/tasks.cc   2012-08-25 21:39:57.000000000 +0900
@@ -80,7 +80,7 @@
++it)
{
pkgCache::PkgIterator pkg = (*apt_cache_file)->FindPkg(*it, arch);
-      if(pkg.end() == false)
+      if(pkg.end() != false)
pkgset->insert(pkg);
}

kozo2@ubuntu:~/tmp$
```

2.3.4 Linux kernel の差異

kernel も package なので触れないわけにはいかないですが、私は kernel hacker ではなく語れる力はありません。申し訳ありません。config の違いを調べることは可能かと思います。Ubuntu の config は下記で参照可能ですが、Debian の kernel config はどこで参照できるのか調べがつきませんでした。

- `https://wiki.ubuntu.com/Kernel/Configs/QuantalToRaring`
- `http://kernel.ubuntu.com/~kernel-ppa/configs/`

また、おそらく適用しているであろう Ubuntu, Debian 毎の patch の差異までは調べがつきませんでした。

2.4 policy などの違い

次に package の内容のような具体的なことからすこし離れ、Ubuntu と Debian の方針の違いについてお話しします。このあたりはご存知の方も多いかと思うのですが前節の package の新旧とも関わるので補足する意味で付記します。

2.4.1 release policy

Debian は stable の release 時期が決まっていないのに対し、Ubuntu は定期的な新 version の release が宣言されており、support 期間も明確に決まっています。(とはいうものの本勉強会参加者は unstable の Debian を利用しており、かつ自己で問題を解決する方が多いと思うので、一応書いた程度のものです)

2.4.2 community の違い、交流

Ubuntu の community にはマーク・シャトルワース氏の Canonical 社との雇用関係にある開発者が存在しており、この雇用関係が前述の定期 release の実現、また前節の package の bug 修正に効いているようです。 (ただし世界各地の Ubuntu community に関してはこの限りではありません)

こういったユーザー指向の Ubuntu に対し Debian の community は Debian developer(DD) 達の集まりで Debian Project Leader(DPL) は DD 達の投票によって決まり、開発者中心の community と言えます。

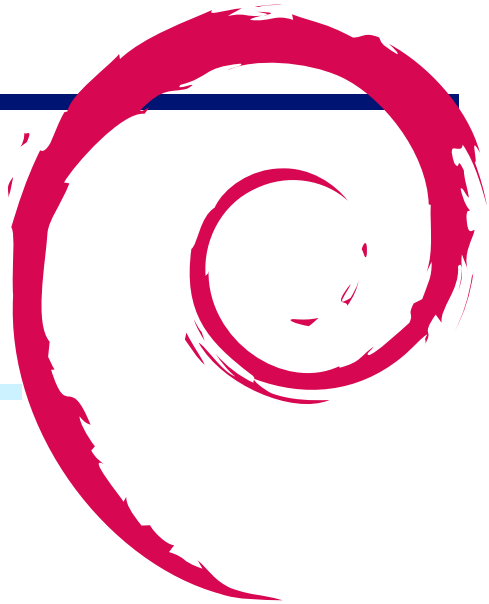
そのような指向の違いはあるものの package についての前節で述べたように元の package のメンテナンスをしているのが DD であることもあり、Canonical 社は DD を雇用していますし、launchpad の情報共有の試みも行われているようなので対立関係ではなくお互いを刺激しあう良い関係なのではないかと考えます。

2.5 おわりに

Debian との違いを知ることはお互い (ユーザーと開発者) の文化の違いの理解や相互の情報交換による Debian package の質の向上にもつながるかと思います。何か他の軋轢とその解決に対してもこの 2 者の関係を思い出すことでなにかしらの案が得られるかもしれません。

2.6 references

- Ubuntu Weekly Recipe 第 16 回 パッケージの使いこなし: `http://gihyo.jp/admin/serial/01/ubuntu-recipe/0016`
- `https://wiki.ubuntu.com/MarkShuttleworth`



3　リリースノートを読んでみよう

佐々木洋平

3.1　はじめに

04/18 に debian-devel-announce ML に「FINAL release update」という件名のメールが流れました[*1]. このメールの冒頭に

```
Timings
=======

We now have a target date of the weekend of 4th/5th May for the release.
We have checked with core teams, and this seems to be acceptable for
everyone. This means we are able to begin the final preparations for a
release of Debian 7.0 - ‘‘Wheezy’’.

The intention is only to lift the date if something really critical pops
up that is not possible to handle as an errata, or if we end up
technically unable to release that weekend (e.g. a required machine
crashes or d-i explodes in a giant ball of fire). Every other RC fix
that does not make it in time will be r1 material. Please be sure to
contact us about the RC fixes you would like included in the point
release!
```

とある通り, Wheezy のリリースは 05/04 or 05/05 が予定されています.

そんなわけで, 本発表の趣旨は「現状のリリースノートを読んで wheezy の状況を把握しよう!」です. 個人的な観測では, 関西 Debian 勉強会参加者の多くが普段使いの環境は unstable or stable なので, 現状の testing, すなわち wheezy を使っていない気がします. freeze 期間なので unstable と testing の違いは微々たるモノかもしれません. ですが, それなりに違いがあるハズです[*2]

それでは

原版:
http://www.debian.org/releases/wheezy/amd64/release-notes/index.html

日本語訳:
http://www.debian.org/releases/wheezy/amd64/release-notes/index.ja.html

を開いて, 読んで行きましょう. 余裕のある人は

```
$ svn co svn://svn.debian.org/ddp/manuals/trunk/release-notes
```

として, 原文を修正しつつ進めるのが良いかもしれません.

[*1] http://lists.debian.org/debian-devel-announce/2013/04/msg00006.html

[*2] 来月にリリースがなされる訳で, その後に「Debian を使ってみよう!(キリッ)」と他人に勧める際にも, 勧める我々(私?) が「いや〜ふだん stable 使ってないから良くわからんわ〜」では, ちょっとアレですよね, という気持ちもあります.

4 Android 端末（Asus Transformer TF201）への Debian インストール奮闘記

cuzic

4.1 はじめに

- cuzic
  - きゅーじっく
  - 親指シフトキーボード使い
    - 親指シフト:濁音（が だ）、捨て仮名（ゃ っ）をすべて 1 打鍵で入力可能
    - 親指シフトキーボードのために普段は Windows を利用
- 最近の出来事
  - Windows 8 のタッチ対応の Ultrabook が欲しい
    - Ultrabook
    - タッチパネル対応
    - フル HD (1920x1080) の高画質
    - まともな日本語キーボード
    - D-Sub 15pin による出力が可能（プレゼン用）
  - お金がなくてなかなか買えない …
    - Android 端末 (Asus Transformer Prime）を所有
  - Android 端末でも十分開発できるように Linux 環境構築に挑戦

4.2 Asus Eee Pad Transformer Prime

- Eee Pad Transformer Prime (日本名 TF201)
  - キーボード付き Android タブレット
  - キーボードはバッテリーでもあり、駆動時間は最大 18 時間
  - 実際、日帰り出張でもまったく問題なし
- 外部メモリが 64GB もある
  - 正直、Android では使いきれない…
  - Linux も入れちゃおう !!!

4.3 むかしむかし

- 実は、TF201 を買ってすぐも Linux 動作にチャレンジ
- ぜんぜんダメだった。
  - いろんなキーが入力できない
    - Ctrl、Alt、` (backtick)、@ (atmark)
  - ほかにもいろいろあったはずだけど思い出せない…
- ほかの方法も考えた
  - Android と Linux のデュアルブートとか
  - もとの環境をつぶして Linux だけにするとか
  - けど、できたら Android の中でアプリみたいな形で Linux を動かしたい
- 今回、再チャレンジ！

4.4 まずは Ubuntu で試そうとしてみた

- まずは root 化する
  - 基本。
  - xda-developers.com [*3] からツールをダウンロードして実行
  - bat を実行するとあとはうまくやってくれる
- Ubuntu Installer for Android
  - google play [*4] からダウンロードし、インストール
  - このアプリは実は手順書だけ。自動インストールじゃない (汗)
  - 3 種類の Ubuntu イメージがある
    - Full……Unity とかといっぱいの GUI プログラム
    - Small…LXDE と Firefox などの基本的な GUI プログラム
    - Core……GUI なし。基本的なコマンドのみ

4.5 Ubuntu インストール（Full 編）

- まずは、Full 版イメージを試そうとしてみる
  - Unity とかといっぱいの GUI プログラム（3.5GB 相当）
- ぜんぜんうまくいかず、挫折
  - インストール編
    - 大きすぎて、Android からうまくダウンロードできない
    - パソコンからダウンロードして、Android に移動させる
  - 動作編
    - Unity が重すぎて、うまく動かない orz
    - そもそも、Firefox がぜんぜん動かない
    - LXDE をインストールしてみる。
    - ほかもいろいろインストールしてみる。
      - あっという間に容量上限（4GB）に達する
  - Small に変更して、再チャレンジしてみる。

[*3] http://forum.xda-developers.com/showthread.php?t=1706588
[*4] https://play.google.com/store/apps/details?id=com.appbuilder.u14410p30729&hl=ja

4.6 Ubuntu インストール（small 編）

- Small 編
  - LXDE と Firefox 等の GUI プログラムつき
  - 全体で 1.5 GB だから軽い
- まずまずうまくいったが、やっぱ挫折
  - うまくいった点
    - LXDE はだいぶ軽い。ちゃんと動く。
    - 開発環境構築関係についてはうまくいった
  - だけど…
    - やっぱり Firefox は動かない
    - Chromium も動かない
    - Konqueror は動く
- タイムリーに Lurdan さんと話した結果、Debian で再チャレンジすることに。

4.7 Debian をインストール

- Debian Kit for Android [*5]を使ってインストールした
- インストール方法
  - 基本的には手順に従うだけ
  - Debian 6.0.6 (squeeze) をインストールしてみた。
  - apt-get install andromize が重要
    - Android 側に追随して /etc/resolv.conf を変更する
- 動作概要
  - img ファイルを loop デバイスとしてマウント
  - chroot せず、/ 以下に usr、var などが配備される
  - VNC サーバを起動し、Android アプリの VNC クライアントから、Debian の X 環境にアクセスする
- 結果
  - Firefox（iceweasel）もちゃんと動作した！
  - ほかのいろんなのもまぁまぁ動いた

4.8 容量を増やした

- Linux が 4GB（初期値）では足りない
  - 10GB くらいは Debian に割当てたい
- Windows PC の Virtualbox の Ubuntu で下記手順を実施

[*5] http://sven-ola.dyndns.org/repo/debian-kit-en.html

```
# virtualbox 側で
nc -l 9000 > debianold.img
# asus transformer prime 側で
busybox nc virtualbox 9000 < /sdcard/debian.img
# virtualbox 側で
dd if=/dev/zero of=debiannew.img bs=1M count=0 seek=10240
mke2fs -F debiannew.img
mkdir debianold debiannew
sudo mount -o loop debianold.img debianold
sudo mount -o loop debiannew.img debiannew
rsync debianold/ debiannew
umount debianold; debiannew
# asus transformer prime 側で
busybox nc -l -p 9000 > /sdcard/debian.img
# virtualbox 側で
nc prime 9000 < debiannew.img
```

4.9 ハマったところ

- 追加インストールが結構必要
  - （例） sudo 、netcat
  - Ubuntu にあるパッケージが Debian だと見つからない
    - （例）leiningen（プログラミング言語 clojure 用の便利ツール）
- root だとネット接続可能なのに、自分で追加したユーザではネット接続不可
  - 適切なグループへの追加が必要だったらしい。(Paranoid Network- ing [*6])

```
usermod -G inet,net\_raw,net\_admin cuzic
```

- /debian ディレクトリに直接ファイルを配置したが失敗
  - loop デバイスとしてマウントせず、そのまま同居したかった
  - /debian/{usr,etc,var など } を置いてみた
  - SD カードのファイルシステムが FAT32(?) で permission をうまく設定できず失敗=¿ loop デバイスの方法で利用し続けることにした。
- Ctrl, Alt がうまく使えない
  - Emacs などの利用において、致命的な問題
  - Android の VNC Player 側の問題
  - 改変版の Android VNC Viewer を使うことで解決
  - ※ ここ[*7] の Comment 28 でコンパイル済みのバイナリを利用

4.10 まだできていないこと

- OpenOffice.org 、LibreOffice の利用
  - なんでか、 Impress をインストールできなかった
- Caps lock、「半角／全角」、「無変換」、「変換」キーの利用
  - VNC プレイヤー側の問題
  - Android 側ではキーイベントを拾えるみたい
  - 無変換、変換が使えれば夢の 親指シフト化 が実現！
  - Caps lock と Ctrl を入れ替えたい
  - 「半角／全角」を ESC にしたい
- Clojure 開発環境での構築
  - Ubuntu だと leiningen のパッケージがある
  - Debian には leiningen のパッケージがない

[*6] http://elinux.org/Android_Security
[*7] http://code.google.com/p/android-vnc-viewer/issues/detail?id=238

4.11 まとめ

- Android タブレットでの Debian 稼働は十分実用的
  - ブラウザやエディタも十分動かせる
  - 容量も十分なサイズでできる
  - 一部のキー配列（Ctrl、ESC）が気になるくらい
  - ちょっと動作はやっぱ遅い
- Android 環境と Debian 環境を同時実行できるのも便利
  - ブラウザは Android のブラウザで
  - 開発は Debian の vi/emacs で
- Android タブレットの長所を持つ開発環境の誕生
  - バッテリーが長持ち、薄くて軽い
- けど、ほんとうは新しいノートパソコンが欲しい
  - お金がないから、Android タブレットの Debian 化で我慢

5 クラウド初心者が AWS に Debian をのっけて翻訳サービスの試行に挑戦してみた

倉敷 悟

基本的にクラウドを使うことにあまり興味がない (作ることには興味あるけど) ということもあって、これまでいわゆるクラウドというものと無縁に生きてきました。今回、たまたま使ってみる用事ができたこともあって、クラウド上で Debian を動かしてみましたので、顛末をご紹介します。

5.1 AWS に目が向いた背景

私は自宅以外のコンピュータ資源として、さくらの (共用) レンタルサーバと、実家に置いた安売りサーバ、いわゆる自宅サーバを使用しています。主な用途は、携帯キャリア/プロバイダの都合から影響を受けないように独自ドメインの転送メールアドレスを保持すること、及び自分用にサーバサービスを提供すること、あとはちょっとした実験、といったところです。

さくらのレンタルサーバは、古式ゆかしき `/home` 切り売り CGI 環境しかないので、今となっては動かせるアプリケーションが極端に限定されます。また、実家サーバは回線が ADSL で帯域もなく、時折不安定に IP がつけかわったりするので、自分以外の誰かにアクセスさせるには向いていません。ですが、Debian JP の翻訳の絡みで、ちょっとだけ「不特定多数からアクセスされ (得) る」環境が欲しくなりました。一応実験のための一時的な設置と考えてはいますが、数ヶ月くらいは動かすことになるだろうし、IP も固定である方がよく、動かす対象は Python の django で書かれたプロセス常駐型の Web DB アプリケーションです。

そこで、VPS としても使えるらしいのと、周囲で推しの声が多数観測されていたこともあり、AWS の「初期利用 1 年無料枠」を使ってみることにしました。使えるリソースは AWS 的に最小限ですが、機能の試用や検証を 1 年かけてできるということなので、結構な大盤振る舞いだと思います。ありがてぇありがてぇ。

5.2 簡単な事前調査

5.2.1 AWS の概略

AWS と一口に言っても、中身はそれぞれ異なるサービスの総称です。ざっくり数えてみたところ、25 個ほどありました。そのうち 10 個くらいのサービス名は略語で、一見したところではさっぱり意味がわかりません。AWS では Elastic という用語がよく使われているようですので、E は Elastic、A は Amazon、S は Service、程度のあたりはつきますが、最初は謎の略語名に圧倒されてしまうかも知れません。今回試してみた EC2 だと、Elastic Computing Cloud の省略です。ECC だといろいろヤバかったのかも知れませんね。

ちなみに Elastic の意味は「伸縮自在」といったところです。

5.2.2 AWS の特徴

AWS は、いわゆるレンタルサーバや VPS とは違って、リソースが文字通り Elastic です。画面をポチポチやれば、メモリや CPU パワーなどを自由に変更することができますし、その変更幅もかなり大きくとれます。また、料金体系も従量課金となっていて、物理的なマシンやノードの単位に縛られず、「電気・水道のようにコンピュータ資源を使う」感覚を実現しているといえるでしょう。周辺サービスについても、種類と内容がともに充実しているだけでなく、日々強化されていっているので、便利さでは他の業者の追従を許していません。いろいろな API を提供していて、「AWS としての機能」を別のアプリなどから操作できる (らしい) というのも興味深いところです。まともに使うと料金は割高なようですが、札束さえつっこめば、インフラの構成に制約されることなく、小さくはじめたサービスをそのまま拡大していけます。そのため、コンシューマ相手にノるかソるかのサービスを提供したい場合や、大規模な設備投資が長期的に見合わない場合などに、とても良くマッチします。

5.2.3 Debian の使用

AWS は、Xen をベースに、RedHat を Amazon 内で魔改造したもので構成されています。残念ながら Debian の出る幕はないわけですが、その上で動かす AMI と呼ばれる仮想 OS イメージについては、某荒木さんが Debian のイメージをメンテされていると聞いたことがあったので、今回はそれを使うことにしました。

5.3 やってみた

5.3.1 アカウントの準備

まずは、AWS 上にアカウントを作成する必要があります。途中で必要になるので、クレジットカードと、着信が受けられる電話を手元に用意しておくといいでしょう。アカウント ID は、物販用の amazon.com (amazon.co.jp ではない) の情報を流用できます。特に待たされたりすることもありませんし、すぐ完了できるかと思います。

5.3.2 EC2 でハマってみる

AWS には綺羅星のごとくベンリ系のサービスがこれでもか！と並んでいるのですが、今回は VPS がわりに使うだけですので、とりあえず EC2 で OS インスタンスをたてることにします。

AWS は世界各地にデータセンターをもっており、使用するデータセンターを「リージョン」として選択できます。日本人が日本人向けのインスタンスをたてるわけなので、ここでは無難に tokyo リージョンを選びました。また、リージョン内にも「エリア」として複数のネットワークが存在し、冗長化などに役立てることができるらしいのですが、今回は単体ホストしか使いませんので、「どこでもええわい」としておきます。

EC2 で使う AMI は公式には Debian のイメージが存在しないので、コミュニティ版を使うことになります。私はこの部分でなかなか手間取りました。ポイントは、

- 基本的に用意されている Debian のバージョンは stable
- 展開した後のログインでは、AMI イメージ毎に特定のログイン名を使う (AWS コンソールで作成した SSH 鍵が起動時にそのユーザにセットされる)

あたりでしょうか。私は Wheezy を使うつもりだったので、最初は Wheezy のイメージ名を検索して使用してみたのですが、なんとログインできません。通常は AMI の情報としてログインユーザ名が提示されているらしいのですが、見つけられなかったため諦めました。幸い、`http://wiki.debian.org/Cloud/AmazonEC2Image` に情報がまとまっていたので、ログイン名のわかる Squeeze イメージではじめて、後から dist-upgrade することにします。余談ですが、dist-upgrade すると grub の処理でコケます。最初から grub は消しときゃいいのになぁ、とは思いました。

インスタンスが起動したら、固定 IP を割り当てます。AWS コンソールから Elastic IPs の画面を開き、新しい IP をアサインして、起動したインスタンスとの紐づけを行います。なんでも Elastic IP はインスタンスで使ってい

る限りは無料らしいのですが、インスタンスを停止した状態で握り続けていると有料になるそうです。実験などで一時的に使うだけ、という場合は解放するのをお忘れなく。

EC2 で起動したインスタンスは、デフォルトでは外部からアクセスできないようにファイアウォールが設定されています。「セキュリティグループ」という画面で簡単に設定できるので、自分用の ssh と、公開する http のポートを開けるようにします。

これで、ほぼ VPS 気分で使えるサーバができました。ssh でログインすれば、後はいつも通りの作業をするばかりです。

5.4 RDS でハマってみる

AWS にはいくつか DB サービスが用意されていますが、その中でも RDS は、普通の mysql サーバ (だけじゃないけど) を簡単に利用できるものです。今回は Web DB アプリケーションを動かすので、せっかくなのでこれも使ってみることにしました。

……といっても、実のところ動かすだけなら特にハマりどころもなく、AW コンソールでちょこっとパラメータを指定すれば、さくっと mysql があがってきます。EC2 と同様、セキュリティグループによる通信許可をしてあげる必要はありますが、拍子抜けするくらい簡単に動きました。しかも、自動でバックアップもとってくれるようです。

今回の用途であれば、EC2 インスタンスの中に手動で mysql をセットアップしても良かったのかもしれませんが、実際に本番で使うなら、これも非常に便利な機能だと思います。

5.5 まとめ

動かしたアプリケーションについてもちょっと紹介しようと思っていたのですが、残念ながら原稿が間にあわなかったので、そちらは時間があればセッション中に触れることにします。

最後に、VPS 的に使ってみた所感をまとめておきます。総合すると、自宅サーバの置きかえにはアリかなー、でも一時的な実験とかでなく、常設するなら VPS の方が安くつくかなー、といったところです。皆さんが試してみたりする時の参考になれば幸いです。

いいところ (無料期間の視点)

- 固定 IP を使える
- Web サービスを公開して試してもらうのに便利
- RDB(mysql) を簡単にセットアップできる
- 実験が終わってインスタンスを止めれば、費用は不要
- 便利ツールとの連携が楽しそう
- 足まわりの運用はすべて忘れてしまえる (バックアップ、HA、セキュリティアップデート、……)

微妙なところ

- 無料期間が終わると、ちょいとお高い (t1.micro 1700 円/月、RDS 6000〜/月くらいらしい)
- パワフルに使うなら、月額固定の VPS の方がいいかも
- サービス多すぎ、変化しまくり、使いこなすのは大変そう #おっさん

6 Debian Installer トラブルシューティング

Yuryu

6.1 自己紹介

- Yuryu (twitter: @Yuryu)
- 某社でインフラエンジニアしてます
- potato → woody → (浮気) → Ubuntu(Gusty) → ... → Ubuntu(Precise)
- 会社では Debian 使ってます
- 好きなコマンドは xargs

6.2 Preseed

6.2.1 Preseed とは

- Debian Installer の応答ファイル
- すべての選択肢が選べる
- PXE と組み合わせると強い
  - 単体でも使えます (少々面倒)

```
locale=en_US language=en country=JP
console-keymaps-at/keymap=jp106
keyboard-configuration/xkb-keymap=jp106
interface=eth0 hostname=debian
domain=local url=http://holo.yuryu.jp/preseed.cfg
DEBCONF_DEBUG=5
```

6.2.2 boot オプションの必要性

- Preseed ファイルが読まれるのは、ネットワークの設定が終わってから
- ネットワーク設定前にもインストーラーの質問はある
- →ブートオプションとして渡す
- 長いので手打ちは無理、PXE を使う

6.2.3 expert install

- expert options → automated install
- 途中で preseed ファイルを指定できる
- とりあえず試すにはこっち

6.2.4 url

- preseed ファイルの場所
- http, ftp, tftp で指定
- https は使えない (!)

6.2.5 ファイルの中身

- 「d-i 項目名 指定」の羅列

```
### Clock and time zone setup
# Controls whether or not the hardware clock is set to UTC.
d-i clock-setup/utc boolean true

# You may set this to any valid setting for $TZ; see the contents of
# /usr/share/zoneinfo/ for valid values.
d-i time/zone string Asia/Tokyo

# Controls whether to use NTP to set the clock during the install
d-i clock-setup/ntp boolean true
# NTP server to use. The default is almost always fine here.
d-i clock-setup/ntp-server string ntp.nict.go.jp
```

6.2.6 ファイルの書き方

- 基本的にはサンプル通り
- でも、いくつか落とし穴が...
  - 一番大変なのが partitioning

6.3 partitioning

6.3.1 partitioning

- d-i partman-auto/choose_recipe
  - atomic - / 一発
  - home - /home だけ分ける
  - multi - /home, /usr, /var, /tmp
- d-i partman-auto/expert_recipe

6.3.2 recipe

```
d-i partman-auto/expert_recipe string                       \
      boot-root ::                                          \
              40 50 100 ext2                                \
                      $primary{ } $bootable{ }              \
                      method{ format } format{ }            \
                      use_filesystem{ } filesystem{ ext2 }  \
                      mountpoint{ /boot }                   \
              .                                             \
              500 10000 -1 ext4                             \
                      method{ format } format{ }            \
                      use_filesystem{ } filesystem{ ext4 }  \
                      mountpoint{ / }                       \
              .                                             \
              64 512 300% linux-swap                        \
                      method{ swap } format{ }              \
              .
```

6.3.3 レシピ形式

レシピ名 ::

最低容量 (MB) 優先度 最大容量 FS

パーティション内容 .

```
500 1000 -1 ext4 \
method{ format } format{ } \
use_filesystem{ } filesystem{ ext4 } \
mountpoint{ /home }
```

6.3.4 容量

- 最大を -1 にすると空き容量をすべて使う
- 優先度は数字が大きなほうが低い
- 最小=最大にせず 100MB 程度の幅をもたせる

6.3.5 method

- format
  - 通常通りフォーマットして使用
- swap
  - スワップパーティションとして使用
- keep
  - 何もせず区画だけ作る

6.3.6 ファイルシステム

- 容量の右側に書くものと、filesystem{ }
- 基本的には同じ
- ext[2-4], xfs, btrfs, jfs, linux-swap
- keep のときは無視される (free でも ok)

6.3.7 お約束

- 冗長に見えても、省略できないもの
- format{ }
- use_filesystem{ }
- filesystem{ ext4 }

6.4 ハマりました

6.4.1 primary

- $primary{ } プライマリ必須指定
- $logical{ } もありそう?
  - 実は無い
  - 省略すると logical になる

6.4.2 一行で書く

- 行末に \ を書いて、一行につなげて書く
- \ の後にスペースがあるとそこで切れます

6.4.3 レシピ名

- レシピ名が必須
- partman-auto/choose_recipe とは無関係
  - expert recipe 使うなら書いてはいけない
- 何を書いても動作に関係なし...

```
d-i partman-auto/expert_recipe string                         \
      boot-root ::                                            \
```

6.4.4 スペースの扱い

- 基本的にはスペース必須
- 開きカッコの手前はスペース不可
  - ○ method{ format }
  - × method_{ format }
- grep でパースしてるので厳格です...

6.4.5 レシピ、認識されてる?

- /tmp/expert_recipe
- 実際に使われたレシピが入る

```
~ # cd /tmp
/tmp # ls
connection_type  expert_recipe    mdadm.conf
/tmp # cat expert_recipe
boot-root :: 40 50 100 ext3 #primary{ } $bootable{ } method{ format } format{ } use_filesystem{ } filesystem{ ext3 }
mountpoint{ /boot } . 500 10000 1000000000 ext3 method{ format } format{ } use_filesystem{ } filesystem{ ext3 }
mountpoint{ / } . 64 512 300% linux-swap method{ swap } format{ } .
/tmp #
```

6.4.6 その他条件文

- $iflabel{ label } - ラベルが一致
- $defaultignore{ } - LVM を使わない時無視
- $lvmok{ } - LVM にしても良い
- $lvmignore{ } - LVM の場合は無視

6.4.7 パーティション再利用

- 既存のパーティションを再利用
- $reusemethod{ }
  - method(format, swap) が同一なら再利用
  - 主に biosgrub, swap 向け?
- $iflabel{ label } と組み合わせる
- http://lists.debian.org/debian-boot/2011/04/msg00333.html

6.5 その他 troubleshoot

6.5.1 d-i じゃない行もある

- d-i で始まらない行もある
  - tasksel
  - popularity-contest
- コピペ事故に注意

6.5.2 止まったら

- syslog を確認
- INPUT ... の行がパラメーター名

6.5.3 ログ

- Alt+F4 のログ
- インストール中 /var/log/syslog
- インストール後 /var/log/installer/syslog
- DEBCONF_DEBUG が必須

6.5.4 こわくないよ

- debian installer の実態はシェルスクリプト
- 各ディレクトリを for で順に呼んでる
- 項目名で grep してみるとヒントが

6.5.5 debian/*.templates

- debian-installer の各種パッケージの debian/*.templates
- 変数名、型、説明が揃ってる
- リファレンス代わりになります

```
Template: partman-partitioning/bootable_logical
Type: boolean
Default: false
# :sl2:
_Description: Are you sure you want a bootable logical partition?
 You are trying to set the bootable flag on a logical partition. The
```

6.5.6 debconf-get-selections

- インストール済み OS から設定を取得
- debconf-get-selections –installer
- デフォルト値も含めて大量に出る
  - 必要な物を取捨選択する必要あり

6.5.7 Ubuntu 対応

- 基本的には同じ
- いくつか追加の質問
  - d-i pkgsel/update-policy select none

  - d-i user-setup/allow-password-weak boolean true
  - d-i user-setup/encrypt-home boolean false
- 詳しくは `https://help.ubuntu.com/lts/installation-guide/i386/preseed-contents.html`

6.6 まとめ

- 困ったら syslog を観る
- recipe はスペースに気をつける
- DEBCONF_DEBUG=5 重要!

7 Debian 日本語入力の選択肢:im-config 関連

青木　修

im-config およびその関連について説明します。

7.1 日本語環境の構築

Aptitude から日本語環境関連 (Japanese desktop と Japanese environment) のパッケージを導入するには下記の様にします。

- 英語表示：Tasks → Localization → Japanese desktop と Japanese environment　（下記スクリーン例）
- 日本語表示：タスク → 地域化 → 日本語デスクトップ と 日本語環境

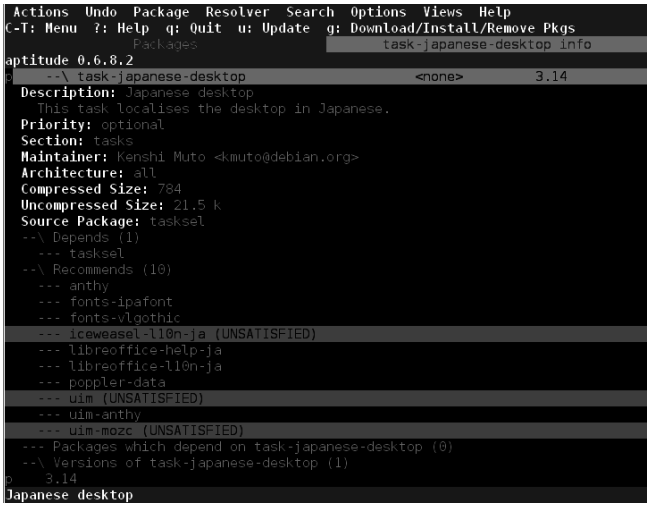


図 1　Japanese desktop

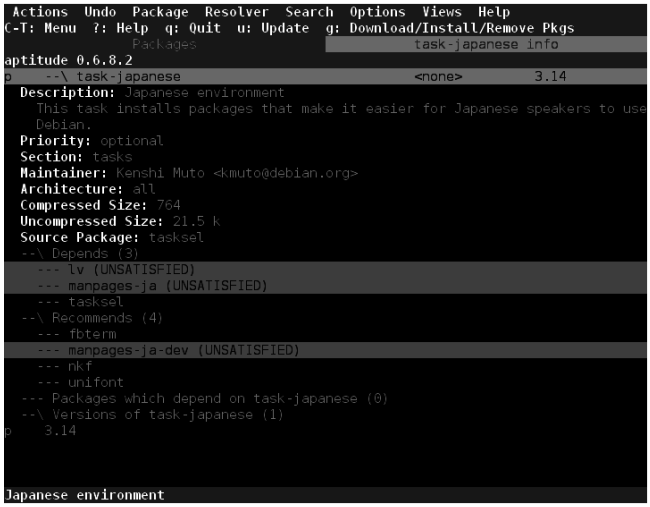


図 2　Japanese environment

7.2 日本語入力の選択肢

Debian の Input method 環境 (IM 環境) と IM Engine(IM エンジン) との組み合わせは下記のように多種あります。wheezy 時点での一般的な日本語入力ユーザへのおすすめは、キーとなるパッケージの ibus-anthy や ibus-mozc や uim-anthy や uim-mozc といったパッケージから一つを選び、APT 系のパッケージ管理プログラム経由で選び導入することです。すると、推薦パッケージも自動的に導入され良好な日本語入力関係が整備されます。

IM \ IM Engine	anthy	mozc	skk	特徴
uim	uim-anthy	uim-mozc	uim-skk	LISP
scim	scim-anthy	scim-mozc	scim-skk	(wheezy version is not in good shape)
ibus	ibus-anthy	ibus-mozc	ibus-skk	GNOME Python
fcitx	fcitx-anthy	fcitx-mozc	N/A	Chinese focus
特徴	GPL	Google 連文節変換強化	EMACS-like	

7.3 im-config とは

im-config パッケージは im-switch パッケージの後継の CJKV[*8] 入力環境設定支援システムです。im-config パッケージにより提供される GUI program の im-config は、複数の IM 関連パッケージを導入した際に、システムに導入された数ある IM 環境の中の一つを選択し、それらが正しい環境で起動されるように設定します。(IM エンジンの設定は、各 IM 環境が提供する設定プログラム経由で行います。) 単一 IM 環境がインストールされる場合には、im-config パッケージによる自動設定で十分です。im-config パッケージが導入するフックスクリプトは/etc/X11/Xsession.d/70im-config_launch にあります。GDM や startx が /etc/X11/Xsession 経由でこのフックスクリプトを呼び、dbus の設定後、下記の様に Client 環境変数を設定し、

```
XMODIFIERS=@im=ibus
GTK_IM_MODULE=ibus
QT4_IM_MODULE=ibus
... 等
```

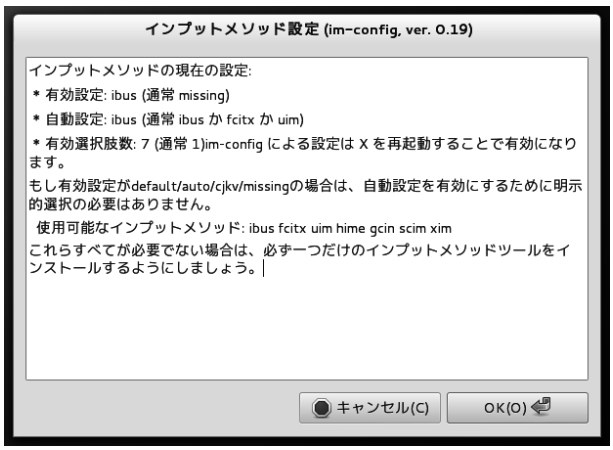
数ある導入された IM 環境の中の一つを選び、それを起動します[*9]。
実際に IM 環境や IM エンジンの設定を有効にするのは、以下のアクションが必要ですので注意して下さい。

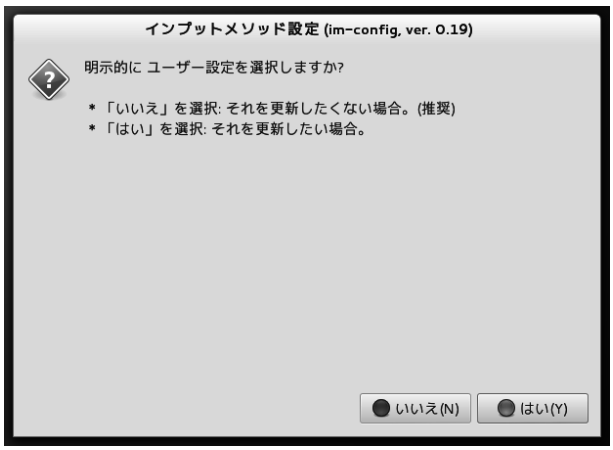
- IM 環境設定には X の再起動が必要です。通常、アカウントからログアウト後の再ログインで実現します。
- IM エンジン設定には IM 環境の再起動が必要です。通常、トレイ等にある IM エンジンの設定プログラムアイコンを右クリックして実現します。

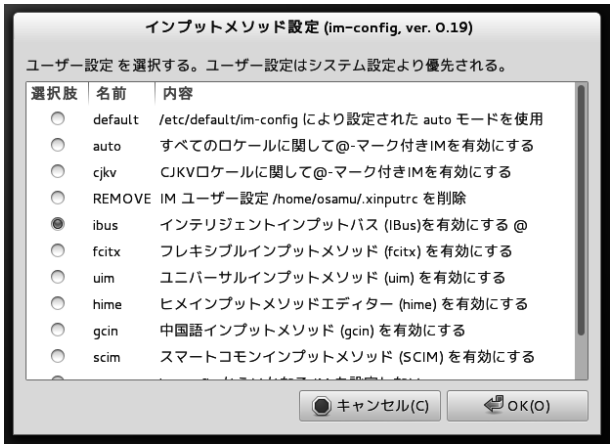
[*8] Chinese-Japanese-Korean-Vietnamese の略
[*9] 例えば、ibus の場合には「/usr/bin/ibus-daemon –daemonize -xim」が起動されます。

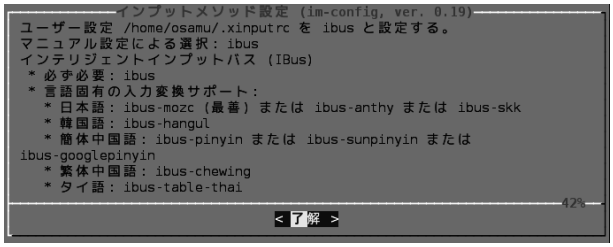
7.4 im-config の日本語 GUI

im-config プログラムは日本語化対応されたシェルスクリプトで実装されたメニュープログラムです。通常は GTK の GUI で表示されます。im-config に-c オプションをつけて起動することで、最後のスクリーン例のようにターミナルの下でもメニュー表示ができます。









7.5 /etc/X11/Xsession.d/

X の起動時には/etc/X11/Xsession.d/の中にある多くのフックスクリプトが順番に実行されます。この/etc/X11/Xsession.d/の中は大体次のようになっています。

```
20x11-common_process-args
30x11-common_xresources
35x11-common_xhost-local
40x11-common_xsessionrc
50x11-common_determine-startup
55gnome-session_gnomerc
60xdg-user-dirs-update
70im-config_launch
75dbus_dbus-launch
90apparmor-notify
90consolekit
90gpg-agent
90qt-a11y
90x11-common_ssh-agent
99x11-common_start
```

im-config のフックスクリプトは、これらのフックスクリプトの 1 つの 75dbus_dbus-launch ということは、すぐ分かりますね。でも、Wheezy の im-config は dbus から起動されるプログラム等にも対応すべく、ちょっと複雑なこと

をしています。
このフックスクリプト 75dbus_dbus-launch が実行される時（PHASE 1）に、IM 関連の環境変数の設定と、$STARTUP という環境変数の更新設定をしています。実際この$STARTUP という環境変数は、他のフックスクリプトででも順次更新設定されます。
そして最後の 99x11-common_start により、「exec $STARTUP」が実行される事で$STARTUP のコマンドが実行されます。この$STARTUP が実行される際 (PHASE 2) に、/usr/bin/im-launch が呼ばれデーモンのプログラムの起動がされます。

7.6 im-config のカスタマイズ

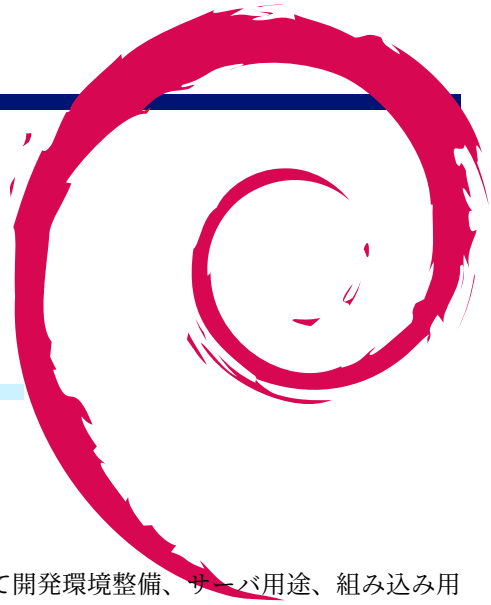
標準設定で動かなければ、X 起動時に読み込み実行される /.xinputrc を使ってカスタマイズしてください。このファイルは、通常 im-config をユーザーアカウントから実行することで作成されます。その内容は、基本的にシェルプログラムです。
典型的な /.xinputrc の例を以下に示します。この内容ですが、im-config パッケージが提供する run_im という専用シェル関数によりその引数の ibus で指定される/usr/share/im-config/data/ibus.rc を呼びだして、ibus を IM 環境として有効とするように設定しています。

```
# im-config(8) generated on Fri, 14 Dec 2012 00:47:26 +0900
run_im ibus
# im-config signiture: b86f7277b9fc27366da40da0b0553a4a  -
```

この /.xinputrc の内容を/usr/share/im-config/data/*.rc ファイルのコピーで置き換えそれを改変することで、IM に関して更なるカスタマイズができます。(上級者テクニックです。) その際には以下の 2 つのルールに注意してシェルプログラムを書いて下さい。

- "$IM_CONFIG_PHASE" = 1 の条件下では、環境変数の設定をします。
- "$IM_CONFIG_PHASE" = 2 の条件下では、デーモンのプログラムの起動をします。

もしバグを見つけた際には、バグ報告を reportbug プログラムを使って im-config パッケージにしてください。



8 debootstrap を有効活用してみよう

杉本典充

debian のインストール処理に使う debootstrap コマンドの応用例として開発環境整備、サーバ用途、組み込み用途、異種 OS 利用とまとめてみました。

8.1 仮想化技術について

最近 KVM を事例とする「仮想化」という言葉をよく聞くようになりました。仮想化にもいくつかの種類があると思い調べてみたところ、以下の例が見つかりました。今回は、コンテナ型仮想化に着目していきます。

仮想化技術	実装例	仮想化環境の特徴
完全仮想化 (エミュレーション型)	QEMU、VirtualBox	既存の OS を無修正のままゲスト環境として動作させる。ホスト OS で動作する仮想化アプリケーションがエミューションする。
完全仮想化 (ハイパーバイザ型)	KVM	既存の OS を無修正のままゲスト環境として動作させる。CPU 等の仮想化機能を使うことでホスト環境におけるオーバーヘッドを極力減らしている。
準仮想化	Xen	ホスト環境とやりとりする API を利用できるように修正が入った OS をゲスト OS として動作させる方式（＝既存の OS そのままでは動かない）
コンテナ型仮想化	OpenVZ、LXC、FreeBSD jail	ホスト環境とゲスト環境は同一カーネルで動作しつつ、ホスト環境から分離したゲスト環境を提供する。

8.2 debian におけるコンテナ環境の作り方

8.2.1 debootstrap

debian においてコンテナ型仮想化の環境を構築する機能として debootstrap があります。debootstrap を実行すると、debian をインストールしたときのルートディレクトリ構造を任意のパスに作成することができます。

debootstrap のインストールは以下の apt-get コマンドで実行できます。deboostrap には debootstrap（sh スクリプトで実装）と cdebootstrap（C 言語で実装）の 2 種類のコマンドがあります。

```
$ sudo apt-get install debootstrap cdebootstrap
$ sudo mkdir -p /srv/chroot
$ cd /srv/chroot
$ sudo debootstrap --arch=amd64 sid ./mysid http://ftp.jp.debian.org/debian
$ ls mysid
bin   dev  home  lib64  mnt  proc  run   selinux  sys  usr
boot  etc  lib   media  opt  root  sbin  srv      tmp  var
```

8.2.2 chroot コマンド

debootstrap で作成したコンテナ環境でプログラムを実行したいのですが、環境変数とディレクトリの配置が合わないためうまくコマンドを実行することができません。そこで chroot コマンドを使います。[*10] [*11]

chroot コマンドを実行するとルートディレクトリでない場所をあたかもルートディレクトリであるように見せることができます。これにより普通に debian をインストールした状態のプログラムの配置と PATH 環境変数がうまく合い、コンテナ環境のプログラムを実行できるようになります。

```
$ sudo chroot /srv/chroot/mysid
# pwd
/
# ls
bin   dev  home  lib64  mnt  proc  run   selinux  sys  usr
boot  etc  lib   media  opt  root  sbin  srv      tmp  var
```

chroot コマンドにも現在ではいくつかの派生版があります。

- chroot
  - 元祖 chroot。実行するには root 権限が必要。
- dchroot
  - chroot を一般ユーザで実行できるように改良した版。(とはいえ debootstrap と設定ファイル作成は root 権限が必要)
- schroot
  - dchroot のセキュリティレベルを向上させた改良版。

今度は schroot コマンドでコンテナ環境に入ってみます。

```
$ sudo apt-get install schroot
$ cd /etc/schroot
$ sudo vi schroot.conf

[mysid]
description=my sid for devel
type=directory
directory=/srv/chroot/mysid
users=norimitu
root-groups=root
personality=linux
preserve-environment=true

$ schroot -c mysid
W: Failed to change to directory ‘ /etc/schroot ’: No such file or directory
I: The directory does not exist inside the chroot.  Use the --directory option to run the command in a different directory.
W: Falling back to directory ‘ /home/norimitu
$ ls -la /srv
total 8
drwxr-xr-x  2 root root 4096 Apr 14 02:59 .
drwxr-xr-x 22 root root 4096 Apr 14 03:04 ..

(何もないです。chroot 環境に入っています)
$ cd /home/norimitu
$ ls
(chroot 元のホスト環境のファイルが出てくる。)
```

直感的には chroot コマンドの方がわかりやすいです。

schroot コマンドの場合は、schroot した環境内のホームディレクトリはホスト環境のホームディレクトリと bind されます。そのため、ホスト環境にあるファイルを schroot した環境からそのまま読み書きできます。(chroot の場

[*10] chroot システムコールは 1982 年にビル・ジョイが作成したものが起源とされています。ビル・ジョイはプログラムをクリーンビルドできる環境がほしかったため通常利用の環境と分離する手段として開発したと言われています。

[*11] クリーンビルド目的で chroot コマンドを使う手法では debian パッケージのビルドにも使われています。

合はそういう仕組みがないため、コピーする必要がある。）

8.3 debootstrap の使いどころ

debootstrap を使うと何が便利なのか使いどころの例を上げてみます。

8.3.1 生活環境と開発 (テスト) 環境を分離する

普段使用する環境は stable を利用するが、バージョン等の都合で部分的に sid で提供されているパッケージを利用したい場合があります。その場合、前述した方法でコンテナ環境を作成し、chroot すれば利用できます。

8.3.2 コンテナ内で異なる CPU アーキテクチャの環境を動かす

debootstrap したコンテナ環境は通常ホスト環境で動作する CPU アーキテクチャで構築します。chroot コマンドでコンテナ環境に入っている場合でもカーネルはホスト OS と同じため、異なる CPU アーキテクチャのバイナリをネイティブに実行できないためです。

amd64 カーネルを実行しているマシンの場合、i386 バイナリは実行できますので i386 環境のコンテナ環境を構築すれば実行できます。

私は諸般の都合で amd64 環境の VPS 上で i386 の postgresql-9.1 を動かしてレプリケーションしています。amd64 と i386 間といった CPU アーキテクチャが異なる場合はレプリケーションができない postgresql の仕様になっているため、VPS のホスト OS は amd64、コンテナ内で i386 環境を動かしてレプリケーションしています。

```
$ sudo mkdir -p /srv/chroot
$ cd /srv/chroot
$ sudo debootstrap --arch=i386 sid ./mysid-i386 http://ftp.jp.debian.org/debian
$ sudo chroot ./mysid-i386
# uname -a
Linux www6368uj 3.2.0-4-amd64 #1 SMP Debian 3.2.41-2 x86_64 GNU/Linux
# apt-get install file
# file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.26,
BuildID[sha1]=0x498625fc854816515ec12819544ebedff51d5c32, stripped
```

また、chroot と QEMU を組み合わせることにより、ホスト OS のカーネルでは動作しない CPU アーキテクチャのコンテナ環境を QEMU を使ったエミュレートで動作させることができます。最近流行りの安価な ARM ボードで debian を動作させるためのディレクトリツリーを作成するのに便利です。[*12]

ホスト環境と異なる CPU アーキテクチャのコンテナ環境を chroot する場合、以下の手順で実行する必要があります。

- debootstrap は「–foreign」引数を渡して実行すること
- コンテナ環境内に「qemu-*-static」ファイルをコピーしておくこと（例：qemu-arm-static）
- chroot で入った後で debootstrap の second stage を実行すること

[*12] 実際に ARM ボード上で debian を起動させるためには SD カード等へのファイルシステム作成、カーネルやドライバの作成、ブートローダーの書き込み等、色々やらないと動きません。

```
$ sudo apt-get install binfmt-support qemu qemu-user-static debootstrap
$ sudo mkdir -p /srv/chroot
$ cd /srv/chroot
$ sudo debootstrap --foreign --arch=armel wheezy ./armdev1 http://ftp.jp.debian.org/debian
$ sudo chroot ./armdev1
chroot: コマンド ‘/bin/bash’ の実行に失敗しました: No such file or directory

$ sudo cp /usr/bin/qemu-arm-static /srv/chroot/armdev1/usr/bin/
$ sudo chroot armdev1
I have no name!@hostname:/#
Linux hostname 3.2.0-4-amd64 #1 SMP Debian 3.2.41-2 armv7l GNU/Linux

I have no name!@hostname:/# apt-get update
bash: apt-get: command not found
I have no name!@hostname:/# ls /debootstrap
arch  debootstrap      debpaths        functions  suite
base  debootstrap.log  devices.tar.gz  required   suite-script
I have no name!@hostname:/# /debootstrap/debootstrap --second-stage
I: Installing core packages...
I have no name!@hostname:/# apt-get update
Reading package lists...

I have no name!@hostname:/# vi /etc/apt/sources.list

deb http://ftp.jp.debian.org/debian wheezy main
deb-src http://ftp.jp.debian.org/debian wheezy main

deb http://security.debian.org/ wheezy/updates main
deb-src http://security.debian.org/ wheezy/updates main

I have no name!@hostname:/# apt-get update
I have no name!@hostname:/# apt-get install file
I have no name!@hostname:/# file /bin/ls
/bin/ls: ELF 32-bit LSB executable, ARM, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.26,
BuildID[sha1]=0x5bc97dbca9ac168932d898a5e2eaf68e8fde5e16, stripped
```

8.3.3 サーバでたくさんのコンテナを常駐させて動かしたい

今までのコンテナ環境は手でコマンドを実行することでコンテナ環境に入っていました。コンテナ環境をサーバのように動かしたいというニーズもあります。今回は Debian GNU/Linux wheezy amd64 上で LXC（Linux Containers）を用いてコンテナ環境を常駐させてみます。

まずホスト環境の設定と環境を確認します。

```
$ sudo apt-get install lxc
$ sudo vi /etc/fstab

(追記します)
cgroup  /sys/fs/cgroup  cgroup  defaults  0   0

$ sudo mount -a
$ lxc-checkconfig
Kernel config /proc/config.gz not found, looking in other places...
Found kernel config file /boot/config-3.2.0-4-amd64
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled
Network namespace: enabled
Multiple /dev/pts instances: enabled

--- Control groups ---
Cgroup: enabled
Cgroup clone_children flag: enabled
Cgroup device: enabled
Cgroup sched: enabled
Cgroup cpu account: enabled
Cgroup memory controller: enabled
Cgroup cpuset: enabled

--- Misc ---
Veth pair device: enabled
Macvlan: enabled
Vlan: enabled
File capabilities: enabled

Note : Before booting a new kernel, you can check its configuration
usage : CONFIG=/path/to/config /usr/bin/lxc-checkconfig
```

LXC で動作するコンテナ環境のネットワークはホスト環境とブリッジ接続する必要があります。

ここでは eth0 はそのままでブリッジデバイスを 1 つ作成し、LXC のコンテナ環境はブリッジデバイスの NAT 環

境下で動作することとします。[*13]

```
$ sudo vi /etc/sysctl.conf

(変更)
net.ipv4.ip_forward=1

$ sudo sysctl -p
net.ipv4.ip_forward = 1

$ vi br-lxc.sh
# script to setup a natted network for lxc guests
CMD_BRCTL=/sbin/brctl
CMD_IFCONFIG=/sbin/ifconfig
CMD_IPTABLES=/sbin/iptables
CMD_ROUTE=/sbin/route
NETWORK_BRIDGE_DEVICE_NAT=lxc-bridge-nat
HOST_NETDEVICE=eth0
PRIVATE_GW_NAT=192.168.20.1
PRIVATE_NETMASK=255.255.255.0
PRIVATE_NETWORK=192.168.20.0/24

${CMD_BRCTL} addbr ${NETWORK_BRIDGE_DEVICE_NAT}
${CMD_BRCTL} setfd ${NETWORK_BRIDGE_DEVICE_NAT} 0

${CMD_IFCONFIG} ${NETWORK_BRIDGE_DEVICE_NAT} ${PRIVATE_GW_NAT} netmask ${PRIVATE_NETMASK} promisc up

${CMD_IPTABLES} -t nat -A POSTROUTING -o ${HOST_NETDEVICE}  -j MASQUERADE
${CMD_IPTABLES} -A FORWARD -i ${NETWORK_BRIDGE_DEVICE_NAT} -o ${HOST_NETDEVICE} -s ${PRIVATE_NETWORK} -j ACCEPT

$ sudo ./br-lxc.sh
$ sudo ifconfig lxc-bridge-nat
lxc-bridge-nat Link encap:イーサネット  ハードウェアアドレス fe:24:c3:eb:6d:c3
          inet アドレス:192.168.20.1 ブロードキャスト:192.168.20.255  マスク:255.255.255.0
          inet6 アドレス: fe80::409a:3cff:fee6:33b0/64 範囲:リンク
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  メトリック:1
          RX パケット:6556 エラー:0 損失:0 オーバラン:0 フレーム:0
          TX パケット:1121 エラー:0 損失:0 オーバラン:0 キャリア:0
      衝突 (Collisions):0 TX キュー長:0
          RX バイト:543131 (530.4 KiB)  TX バイト:95778 (93.5 KiB)
```

コンテナを作成します。LXC におけるコンテナは “/var/lib/lxc” ディレクトリの下にできます。

```
$ sudo lxc-create -n lxc-deb1 -t debian
  (ダイアログ形式でインストールするバージョンやダウンロード先ミラーの URL、
    root パスワードが聞かれるので答える)

$ cd /var/lib/lxc/lxc-deb1
$ ls
config    rootfs
$ sudo vi config

(追記します)
## Network
lxc.utsname = lxc-deb1
lxc.network.type = veth
lxc.network.flags = up

# that's the interface defined above in host's interfaces file
lxc.network.link = lxc-bridge-nat

# name of network device inside the container,
# defaults to eth0, you could choose a name freely
# lxc.network.name = lxcnet0

lxc.network.hwaddr = 00:FF:AA:00:00:01

# the ip may be set to 0.0.0.0/24 or skip this line
# if you like to use a dhcp client inside the container
lxc.network.ipv4 = 192.168.20.101/24
```

LXC コンテナ内で sshd が起動するよう設定しておきます。

```
$ sudo cp /etc/resolv.conf /var/lib/lxc/lxc-deb1/rootfs/etc/
$ sudo vi /var/lib/lxc/lxc-deb1/rootfs/etc/ssh/sshd_config

(変更前)   #ListenAddress 0.0.0.0
(変更後)   ListenAddress 192.168.20.101
```

LXC コンテナを起動します。“-d” オプションはバックグラウンドで起動させるオプションです。これでコンテナ環境が常駐して動作するようになります。

[*13] http://wiki.debian.org/LXC/SimpleBridge

```
$ sudo lxc-start -n lxc-deb1
Using makefile-style concurrent boot in runlevel 2.
Starting OpenBSD Secure Shell server: sshdCould not load host key: /etc/ssh/ssh_host_rsa_key
Could not load host key: /etc/ssh/ssh_host_dsa_key

なぜか ssh 鍵を作る処理が自動実行されず止まってしまうため、手動で作ってみる。
$ sudo chroot /var/lib/lxc/lxc-deb1/rootfs ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key
$ sudo chroot /var/lib/lxc/lxc-deb1/rootfs ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key

$ sudo lxc-start -n lxc-deb1 -d
$ ssh root@192.168.20.101
root@lxc-deb1:~#

  ログインできました。

$ sudo lxc-stop -n lxc-deb1
```

8.3.4 異なる OS 上で debian を構築する

debootstrap は異なる OS 上で debian を構築することができます。ここでは、FreeBSD-8.3-RELEASE amd64 の jail 機能を用いて kfreebsd-amd64 を prisoner として動かしてみます。[*14]

まずは jail 環境を作成するためのツールをインストールします。

OS が異なると debootstrap コマンドがないため、tarball（中身は sh スクリプト版）をダウンロードして実行します。[*15]

```
> cd
> wget http://ftp.jp.debian.org/debian/pool/main/d/debootstrap/debootstrap_1.0.48.tar.gz
> tar xf debootstrap_1.0.48.tar.gz
> cd debootstrap-1.0.48
# su
# setenv DEBOOTSTRAP_DIR ‘pwd‘
# ./debootstrap --arch=kfreebsd-amd64 wheezy /usr/jails/jailkfdeb http://ftp.jp.debian.org/debian
# kldload fdescfs linprocfs linsysfs tmpfs
# umount /usr/jails/jailkfdeb/dev/fd
# umount /usr/jails/jailkfdeb/dev
# mount -t linprocfs linprocfs /usr/jails/jailkfdeb/proc
# mount -t linsysfs linsysfs /usr/jails/jailkfdeb/sys
# mkdir -p /usr/jails/jailkfdeb/lib/init/rw
# mount -t tmpfs tmpfs /usr/jails/jailkfdeb/lib/init/rw
# cp /etc/resolv.conf /usr/jails/jailkfdeb/etc/resolv.conf
```

[*14] http://blog.vx.sk/archives/22-Updated-Tutorial-Debian-GNUkFreeBSD-in-a-FreeBSD-jail.html
[*15] ports で/usr/ports/sysutils/debootstrap もありますのでそちらを使ってもいいです

```
> sudo portsnap fetch
> sudo portsnap update
> cd /usr/ports/sysutils/ezjail
> sudo make
> sudo make install

> sudo touch /etc/fstab.jailkfdeb
   （このファイルがないとエラーになるためとりあえず作成します）
> vi /etc/rc.conf
（追記します）
ifconfig_em0_alias0="inet 192.168.1.63/32"

> ifconfig em0 192.168.1.63 netmask 255.255.255.255 alias
> vi /usr/local/etc/ezjail/jailkfdeb

# To specify the start up order of your ezjails, use these lines to
# create a Jail dependency tree. See rcorder(8) for more details.
#
# PROVIDE: standard_ezjail
# REQUIRE:
# BEFORE:
#

export jail_jailkfdeb_hostname="jailkfdeb"
export jail_jailkfdeb_ip="192.168.1.63"
export jail_jailkfdeb_rootdir="/usr/jails/jailkfdeb"
export jail_jailkfdeb_exec_start="/etc/init.d/rc 3"
export jail_jailkfdeb_exec_stop=""
export jail_jailkfdeb_flags="-l -u root"
export jail_jailkfdeb_mount_enable="YES"
export jail_jailkfdeb_devfs_enable="YES"
export jail_jailkfdeb_devfs_ruleset="devfsrules_jail"
export jail_jailkfdeb_procfs_enable="YES"
export jail_jailkfdeb_fdescfs_enable="YES"


> sudo /usr/local/etc/rc.d/ezjail start jailkfdeb
Configuring jails:.
Starting jails: jailkfdeb.
> jls
   JID  IP Address      Hostname                      Path
    11  192.168.1.63    jailkfdeb                     /usr/jails/jailkfdeb
> su
# jexec 11 /bin/sh

（ここからコンテナ環境の中です）

# uname -irps
GNU/kFreeBSD 8.3-RELEASE-p6 amd64 Intel(R) Core(TM) i5-2500S CPU @ 2.70GHz
# vi /etc/apt/sources.list

deb http://ftp.jp.debian.org/debian wheezy main contrib non-free
deb-src http://ftp.jp.debian.org/debian wheezy main contrib non-free

# apt-get update
# apt-get install openssh-server
# vi /etc/ssh/sshd_config

修正前）  #ListenAddress 0.0.0.0
修正後）  ListenAddress 192.168.1.63

# /etc/init.d/ssh restart
# passwd
# exit

（コンテナ環境からホスト環境に戻ります）

> ssh root@192.168.1.63
```

ssh でコンテナ環境にログインできました。

8.4 参考情報

- Debian Wiki Schroot http://wiki.debian.org/Schroot
- Debian Wiki LXC http://wiki.debian.org/LXC
- Debian Wiki EmDebianCrossDebootstrap http://wiki.debian.org/EmDebian/CrossDebootstrap
- 関西エリア Debian 勉強会 2009 年 04 月号「Debian を新規に install してから常用環境にするまで」 佐々木洋平
- Updated Tutorial: Debian GNU/kFreeBSD in a FreeBSD jail http://blog.vx.sk/archives/22-Updated-Tutorial-Debian-GNUkFreeBSD-in-a-FreeBSD-jail.html

9 Using Drupal on Debian ー CMS から入った人の Debian 体験

紀野 恵

9.1 はじめに

今回は Drupal の紹介と共に、Linux 初心者の Web 製作者がどのように Debian を触ってきたかをお話しするつもりです。

サーバーホスティングがどんどん低価格化し、CMS・Web アプリを VPS や専用サーバー、あるいは AMAZON EC2 のようなクラウド環境で動かすことへの敷居も下がってきています。

特に VPS に関して日本は現状世界で一番競争が激しくコストパフォーマンスの高い環境が手に入ります。これまで共用サーバーでを動かしていたときには手が届かなかったライブラリの導入やパフォーマンス・チューニングが出来るようになり、自由度も性能も一度に手に入れることができるのですが、一方で OS と PHP,MySQL などのミドルウェアをまるごと自分でメンテナンスしなければならなくなりました。

今後もますます増えていくであろう Drupal や Wordpress などの CMS ユーザーに Debian を使ってもらい、コミュニティがさらに活性化する参考にしてもらえばうれしく思います。

9.1.1 自己紹介

名前 紀野 恵 (きの さとし)
Twitter: @qchan_kino
Facebook: satoshi.kino
所属 ANNAI LLC `http://an-nai.jp`
ジオどす `http://geodosu.com`
などあれこれ。

`http://groups.drupal.org/japan/` で活動しています。
OSC 京都、KOF 実行委員や OSS 勉強会運営のお手伝いなども。KOF2012 は Drupal で動いてます！

9.2 Drupalって？

CMS でもあり、Web アプリケーションフレームワークでもあります。

9.2.1 要件

- License: GPL 2
- Web Server: Apache, Nginx, or Microsoft IIS
- PHP: 5.3 以上推奨 PDO 必須
- DB: MySQL, PostgreSQL, SQLite (MS SQL Server, Oracle はモジュールで対応)
  [1]

開発環境では Sqlite がオススメ (さっと立ち上がってポータビリティに優れています)。

9.2.2 人気

- Open Source Awards 優勝多数でアワード殿堂入り
- インストールベースのシェア 3 位（1 位は Wordpress,2 位は Joomla!）

9.2.3 採用実績

世界中の有名サイトでの採用実績多数

- Whitehouse
- Harvard University
- Econmist
- 毎日 jp
- Computer World
- Ubuntu
- Linux Foundation
- SONY MUSIC ENTERTAINMENT

など多すぎて書ききれません。
ブログ的な情報発信型だけでなく、会員コミュニティサイトのように内部で動きのあるサイトを得意としています。
大学、政府系では海外で非常に強い信頼を得ています。

9.2.4 特徴

一企業発のプロダクトではなく、完全にコミュニティーベースの OSS。

- 拡張性の高いアーキテクチャ
- 豊富な API と Hock システム
- オーバーライドの仕組み
- 専門のセキュリティチームがおり、脆弱性が報告されれば迅速に対応

今時の CMS はどれもカスタマイズの機構を備えていますが、Drupal はその拡張性が非常に高く、オーバーライドできることを重要視して作られていますので、ダーティーなハックを極力減らせます。

9.2.5 開発状況

Web の流れを常にキャッチアップしていこうとしています。次期バージョン Drupal8 は真にフレームワーク化しそう。

Restful、Symfony2 採用など大きく変わります。

9.2.6 得意分野

大規模サイトの構築を得意としています。

- リバースプロキシ
- DB レプリケーション標準対応
- Memcache や静的ファイルキャッシュも可能
- Varnish, nginx ,APC などでのキャッシュ対応が容易

9.2.7 多言語対応・共同翻訳システム

標準で多言語対応。また、`http://localize.drupal.org` でコア + 全モジュールの共同翻訳システムが稼働。

9.2.8 マルチドメイン、マルチサイト

標準で対応しています。

9.2.9 コミュニティ

世界中の活発なコミュニティで開発されています。

毎年、春はアメリカ、秋はヨーロッパで大規模なカンファレンスも開催

Drupalcon Munich

- 日時：2012 年 8 月 20 日 〜 24 日
- 参加人数：1800 人
- 収入 Total Revenue €892,221(約 9100 万円）／支出 Total Expenses €858,366(約 8700 万円）

9.2.10 豊富なモジュール

- 提供されているモジュールの数は 5,000 以上
- 全モジュールをコミュニティ内で Git 管理。
- drupal.org 内に Issue や Patch 等、全てが集約されている。

たくさんのモジュールがあることで有名な Drupal だが、、、モジュールの組み合わせ、実装方法が何通りも存在します。

柔軟性が高い反面、学習カーブが厳しいと言われることがあります。

そうした声を反映してか、最近はインストールしてそのまま使えるディストリビューションがとても増えてきました。

9.2.11 ディストリビューション

- アメリカ政府のホワイトハウスが配布しているディストリビューション。市民からの請願・情報を集約するサイト。
- ハーバード大学が開発・配布している教育機関向けディストリビューション
- E コマース向けディストリビューション

- Open Data 特化型ディストリビューション
- CRM のディストリビューション
- PostGIS、GeoServer、OpenLayers のセットになっている GeoMedia に特化したディストリビューション
- カンファレンス参加登録、セッション登録、スケジューリング、チケット決済まで可能な大規模なカンファレンス用サイトを作成できるディストリビューション

など様々公開されています。[2]

9.3 Linux ディストリビューションで Debian を選んだ理由

- 超安定
- Drupal を使うための情報が圧倒的に多い。(Ubuntu 含む)
- 検索のために単語登録してます − ”で” = (debian | ubuntu) drupal
- パッケージが多い
- メジャーバージョンアップが可能
- 個々のパッケージのバージョン固定ができる

一番重要なポイント

コミュニティが活発。勉強会で直接話を聞ける。

9.4 Debian で Drupal などの CMS・Web フレームワークを使うには …

9.4.1 Debian の PHP バージョンを変えたいときの対処法

- 下げるのは以前のバージョンなどから比較的容易に持ってこれるが、上げるのは依存問題を解決できないことが多いしかし極力、ソースからのコンパイルは避けたい。
- Dotdeb にお世話になってます。[3]
  リリースの端境期など Debian パッケージがセキュリテイパッチを当ててくれないときにも便利
  OS アップグレード時には外す事

9.4.2 マニュアルインストールの場合

1. ソース
   PHP の Web アプリはだいたいファイル一括配置。
   Drupal、Wordpress、Typo3、OpenPNE などほとんど同じ。
   (a) `/var/www/`以下に tar ボール解凍
   (b) DB を作成
   (c) Apache の Vhost 設定
   (d) ブラウザからインストーラーにアクセス
2. Drush
   Drupal で共用サーバーを卒業したならまず何を置いても使えるようにしたい。
   - Drush - drupal shell[4]
   - ブラウザからだと煩雑な Drupal の操作をシェルから操作できるようになる。
   - PEAR からのインストールがおすすめ。Drush は開発スピードが早いので頻繁なアップデート追従が楽。
   - 事前課題 VM はファイル直置きなので drush コマンドそのものを使う。

```
drush dl drush --destination=/usr/local/share/
```

- Drush に出来る事
  - Drupal コア、モジュールのインストール・アンインストール
  - 簡易 WebServer 立ちあげ（PHP5.4 以降はライブラリ不要）
  - Drupal デプロイ
  - Drupal コア、モジュールのアップデート
  - Drupal のサイトファイルとデータベースのスナップショット作成
  - スナップショットからのリストア
  - キャッシュクリア
  - 複数サイトの一括コマンド操作
  - DB 操作全般
  - Drupal 設定変数操作
  - Drupal 状況の表示
  - モジュールの Patch を URL から当てる
  - ユーザー追加
  - パスワード変更
  - ロール変更・追加
  - make file からの Drupal ディストリビューションインストール

  モジュールとの連携でまだまだ出来る事が増え続けている。

9.5 パッケージインストールの場合

- `apt-get install drupal7` だけ。
  非常に簡単。素晴らしい。
- ただし、マニュアルインストールとは互換性がないので混ぜないように。[5]
- ここで問題。どこがどう変化したのかわからない。
- まず最初にすべきは`/usr/share/doc/`{ **パッケージ名** } の中を読むこと。
- 設定ファイルはどこか？
  - `dpkg --status` { **パッケージ名** }

    `/etc/cron.d/drupal7`

    `/etc/drupal/7/htaccess`

    `/etc/drupal/7/sites/default/settings.php`

    `/etc/drupal/7/apache2.conf`
- パッケージのインストール後の配置を確認するには。
  - Debian パッケージページの list of files[6]
  - `dpkg -L` { **パッケージ名** }
  - `apt-file list` { **パッケージ名** }[*16]
  - `locate` { **パッケージ名** } [*17]

[*16] インストールされてなくても OK
[*17] 本来の使い方ではない

– ざっくりファイル配置の変化 (*) はパッケージメンテナが追加したファイル
  * /usr/share/drupal7
    ほとんどの Drupal 関係ファイルはここ。残りはこのディレクトリへのシンボリックリンクとなっている。実質の Drupal ルートディレクトリ
  * /var/lib/drupal7/files
    /var/lib/drupal7/backups (*)
    アップロードされたファイル・ディレクトリ
  * /etc/drupal/7
    .htaccess、 /sites 以下 /profile 以下などユーザーが変更する設定やファイル類を配置
  * /var/www
    Drupal ルートディレクトリ (/usr/share/drupal7) へのシンボリックリンク
  * /etc/cron.d/drupal7 (*)
    パッケージが用意している cron ファイル
  * /etc/drupal7/apache2.conf (*)
    Apache2 の設定ファイル /etc/apach2/conf.d へシンボリックリンクを貼って使う
  * /var/lib/drupal7/backups (*)
    パッケージが用意しているバックアップスクリプト用と思われる
  * /usr/share/doc/drupal7/scripts/ 以下 (*)
    パッケージが用意しているさまざまなシェルスクリプト群
  * /etc/dbconfig-common/drupal7.conf (*)
    パッケージが生成するコンフィグファイル
  * /usr/share/doc/drupal7/dbconfig.template (*)
    パッケージが用意しているデータベースコンフィグサンプル
– 最初、どのような考えでこうなっているのかに戸惑った。
  * FHS(Filesystem Hierarchy Standard) に従っている。[7][8]
  * 設定ファイルは /etc 以下に置かなければならない。[9]
  * SELinux の問題も。/var/lib/drupal7/files などは、SELinux の絡みでこうなったらしい。Redhat 系も同じ。[10]
– 番外
  * Redhat 系の drupal リポジトリも似たようなパッケージファイル配置でした。DB や Apache の設定ファイルを作成するスクリプトなどは独自
  * Wordpress はどうなってる？
    · Wordpress の最新版は unstable にしかないが、パッケージ配置構成は大幅に変わっている。今からなら unstable を入れておいたほうが良いだろう。
    · 独自の設定スクリプトを作って、Web からアクセス。
    · wp-content 以下を /var/lib と /usr/share/wordpress に分割している。SELinux 対策かも。
    · Flash を使ったツールが同梱されているので、パッケージには含めていないので別途インストールしなさいと書いてある。
    · /var/www/以下にドキュメントルートのシンボリックリンクを置かず、Apache2 から /usr/share/wordpress に直接振り分けろと書いてある。
    · 独自の apache.conf サンプル
    · MySQL のセットアップはスクリプトを作ってくれている。
    · それぞれパッケージメンテナーの考え方が違うのがわかって面白い。

9.6 Drupal に限った場合の Debian package のデメリット

- Upstream の Drupal と何が変わったのかがわかりにくい。
- バージョンが古い。5000 以上もあるモジュールのパッケージ化は難しい。
- Drush が使えない。
- ローカルや他のディストリなど別環境へ移行がしにくい。[*18]
- Drupal コミュニティで共有されている情報との食い違いが多く戸惑う。初心者ならなおさら難しい。

9.7 Debian 初心者が悩んだポイントと素朴な疑問

1. `/usr/share/doc/{` **パッケージ名** `}` にドキュメントがあると知らず右往左往。
   いつもの解凍した tar ボール内に README.txt などが入っている感覚に引っ張られる。
2. conf ファイルの独特な部分に関して日本語情報がもっと欲しい。日本語情報を探すとどうしても CentOS が多い。
   - Apache2
   - MySQL
   - iptables

   など
3. パッケージの構成として、Upstream のディレクトリ構成をそのまま残して、FHS の要請はシンボリックリンクで済ますことって出来ないのでしょうか。
   せっかくの Debian パッケージをできるなら使いたい。
4. Stable, Old Stable の 2 世代までセキュリティアップデートしてもらえたらいいのにと切に希望。
   - サーバーを立てるタイミングによっては 1 年後にメジャーアップグレードが必須のことも。
   - Ubuntu LTS を選ぶことになるが、Debian との微妙な違いにまた戸惑ったり。 dist-upgradeってやっちゃっていいの？とか個々の設定も微妙に違う。
   - ほんとに実用的かどうかは別にして RedHat クローンのサポート 10 年はちょっとうらやましかったりする。
5. 動いているサーバーのメジャーバージョンアップはやっぱりドキドキする。できるならばしたくない。
   - 皆さん、どうしてますか？
   - リリース時にハンズオンを企画して欲しい！
   - アップグレードをテストする同一環境を作るにはどういう方法がありますか？
   - アップグレードスクリプトの conf ファイルを置き換えますか？ にどう対応してよいかわからない。ベストプラクティスってありますか？
   - Grub で躓いて再起動できなくなった事あり。コンソールがない VPS などだとジエンド。
6. Stable のローリングリリースってできないもんでしょうか。
7. パッケージ製作時の意図を Debian パッケージの Web ページで読めたらいいのに。
   なぜ、このような組み換えをしたのかを説明があると理解しやすい。conf ファイルの位置。ドキュメントの場所など。
8. `/etc/apt/preferences` に書くパッケージの PIN ナンバーの理解が難しい。ここもハンズオン希望。
9. Drupal も同じですが、初心者にしか意味がなくても導入時に書籍があると安心感ありますね。

[*18] 本来サーバー移行時に必要なファイルは`/sites` 以下と DBdump だけだが、ファイルが分割されてしまうので Drush での自動デプロイ、Drupal ホスティング管理ツール Aegir などからの操作ができない。[11]

9.8 まとめ

- Web サイト立ち上げるなら Drupal いいですよ。
- Debian に限らず OSS の開発者、翻訳者の方には普段から大変お世話になっています。
  勉強会・Meetup の定期的な主催がどれほど大変かということも多少ながら理解しているつもりですので、68 回開催は心底尊敬に値します。
  Debian 勉強会とコミュニティの方々にお礼を申し上げると共に、今後も初心者ユーザーを導いていただけるようお願いいたします。

参考文献

[1] System requirements | drupal, http://drupal.org/requirements

[2] Distributions | drupal.org, http://drupal.org/documentation/build/distributions

[3] Dotdeb - The repository for Debian-based LAMP servers, http://www.dotdeb.org/

[4] Drush | drupal.org, http://drupal.org/project/drush

[5] Drupal - Community Ubuntu Documentation, https://help.ubuntu.com/community/Drupal

[6] Debian - Filelist of package drupal7/wheezy/all, http://packages.debian.org/wheezy/all/drupal7/filelist

[7] Debian JP Project - Debian ポリシーマニュアル - オペレーティングシステム, http://www.debian.or.jp/community/devel/debian-policy-ja/policy.ja.html/ch-opersys.html#s9.1

[8] Filesystem Hierarchy Standard - Wikipedia, http://ja.wikipedia.org/wiki/Filesystem_Hierarchy_Standard

[9] Debian JP Project - Debian ポリシーマニュアル - ファイル, http://www.debian.or.jp/community/devel/debian-policy-ja/policy.ja.html/ch-files.html#10.7.2

[10] Bug 472642 - SELinux denies access to /etc/drupal/default/files/, https://bugzilla.redhat.com/show_bug.cgi?id=472642

[11] Aegir, http://www.aegirproject.org/

10 ldapvi & python-ldap で stress-free life

まえだこうへい

10.1 ストレス溜まってませんか？

今回の記事は OpenLDAP の管理を行っている人がターゲットです。LDAP サーバを管理している人は多くはないと思いますが、この記事が LDAP の運用で苦労している人の一助になれば良いと思います。LDAP なにそれ?な人も LDAP アカウントやアドレス帳などで知らぬ間に恩恵を受けているということは大いにあるので、明日は我が身と思ってお読み下さい。また、自分で作ったツールのアカウント管理を LDAP で行いたい、という場合にも参考になると思います。

なお、LDAP の基本的なお話や、Debian システムでの OpenLDAP の入門については、第 61 回 関西 Debian 勉強会 (2012 年 7 月) に佐々木さんが発表&資料を掲載されているのでそちらを参照下さい。[*19]

10.2 stress = LDIF

LDAP の運用で何が面倒かというと、LDIF(LDAP Data Interchange Format) の作成やパースです。LDAP のデータを作成・更新・削除するには通常、LDIF を作成し、ldapadd/ldapmodify/ldapdelete コマンドで実行します。一つのオブジェクトは dn(distiguised name, 識別名) から始まり、それに attribute が続きます。複数オブジェクトを記述するには空行で分割します。下記は RFC2849 の “Example 1: An simple LDAP file with two entries” からの引用です。[*20]

```
version: 1
dn: cn=Barbara Jensen, ou=Product Development, dc=airius, dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
cn: Barbara Jensen
cn: Barbara J Jensen
cn: Babs Jensen
sn: Jensen
uid: bjensen
telephonenumber: +1 408 555 1212

dn: cn=Bjorn Jensen, ou=Accounting, dc=airius, dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
cn: Bjorn Jensen
sn: Jensen
telephonenumber: +1 408 555 1212
```

データフォーマット自体はさほど難しくありませんが、このデータを一から自分でスクラッチで作成するのはとて

[*19] あんどきゅめんてっど Debian2012 年冬号 8.「Debian で作る LDAP サーバ」 http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume2012-fuyu.pdf

[*20] http://www.ietf.org/rfc/rfc2849.txt

も面倒だと思いませんか？例えば上記のようなユーザアカウントの場合、人間にとっては TSV などの形式を表計算ソフトで読んだり編集したりすることでしょう。そこから LDIF に変換したり、あるいはその逆はどうしますか？簡単に思いつくのは次の二つでしょう。

1. 手でちまちま変換する
2. なんらかのプログラムで一括変換する

前者は転記ミスなどの可能性から論外ですが、管理対象のアカウントが少数ならいっそ割り切って LDIF を手で作成するのもありでしょう。しかしユーザ数が大量になるとプログラムで変換しないと時間の無駄です。

10.2.1 そして slapd-config

OpenLDAP 2.3 からは動的に設定を変更できる slapd-config(5) が採用されました。これ自体も LDAP データベースとなっています。current release の 2.4 系では 一部の場合のぞき、従来の slapd.conf(5) は推奨されていません。将来のリリースでは slapd.conf(5) は取り除かれることになっています。[*21]。廃止されるものを使いつづけても仕方ありませんね。しかし、ユーザデータの管理ですら LDIF で悩んでいるのに、さらに slapd 自体の設定までも LDIF を作らないといけないと思うと、もう気が狂いそうです。

10.3 ldapvi で LDIF 地獄からの脱却

そこで、ldapvi です[*22]。このツールは、vipw(8) コマンドや、visudo(8) と同じように、vi のインタフェースで LDAP のデータを変更できます。これで少数のデータを管理するときはもちろん、slapd-config の設定管理の悩みも解決します。

使うには、ldapvi パッケージをインストールします。

```
$ sudo apt-get install ldapvi
```

使い方は、ldap-utils パッケージのコマンドのオプションとほぼ同じです。slapd-config での操作は下記のコマンドを実行します。

```
$ sudo ldapvi -Y EXTERNAL -h ldapi:/// -b cn=config
```

これで slapd-config 管理下のすべてのオブジェクトツリーが ldapvi のフォーマットとして表示されます。ldapvi のフォーマットはほぼ LDIF に似ています。

```
(snip)

0 cn=config
objectClass: olcGlobal
cn: config
olcArgsFile: /var/run/slapd/slapd.args
olcLogLevel: 512
olcPidFile: /var/run/slapd/slapd.pid
olcToolThreads: 1

1 cn=module{0},cn=config
objectClass: olcModuleList
cn: module{0}
olcModulePath: /usr/lib/ldap
olcModuleLoad: {0}back_hdb

2 cn=schema,cn=config
objectClass: olcSchemaConfig
cn: schema
(snip)
```

LDIF との違いは、”dn: cn=config”となっている部分が”0 cn=config”と表示されていることです。この数字は表示されているオブジェクトを 0 を基数とする序数となっています。例えば、新しいオブジェクトを追加する場合は、数字の代わりに”add”を使います。auditlog, ppolicy モジュールを新しくロードする場合は下記のように挿入します。

[*21] http://www.openldap.org/doc/admin24/slapdconf2.html
[*22] http://www.lichteblau.com/ldapvi/

```
(snip)

1 cn=module{0},cn=config
objectClass: olcModuleList
cn: module{0}
olcModulePath: /usr/lib/ldap
olcModuleLoad: {0}back_hdb

add cn=modile,cn=config
objectClass: olcModuleList
cn: module
olcModulePath: /usr/lib/ldap
olcModuleLoad: auditlog.la

add cn=modile,cn=config
objectClass: olcModuleList
cn: module
olcModulePath: /usr/lib/ldap
olcModuleLoad: ppollicy.la

2 cn=schema,cn=config
objectClass: olcSchemaConfig
cn: schema
(snip)
```

追記したら、vi と同様に保存&終了してみます (:wq コマンドを実行)。すると次のメッセージが出力されます。

```
SASL/EXTERNAL authentication started
SASL username: gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth
SASL SSF: 0
     12 entries read
add: 2, rename: 0, modify: 0, delete: 0
Action? [yYqQvVebB*rsf+?]
```

ここで、”y”を入力すると、実際に slapd-config に反映されます。前述の通り、slapd.conf とは違い、slapd の再起動は不要です。”q”を入力するとキャンセルして終了します。

上記での挿入時に、 cn=module0 の”0”を残したまま保存してみてください。

```
add cn=modile{0},cn=config
objectClass: olcModuleList
cn: module{0}
olcModulePath: /usr/lib/ldap
olcModuleLoad: ppollicy.la
```

すると次のようにエラーに修正を要求されます。

```
ldap_add: Naming violation (64)
add: 1, rename: 0, modify: 0, delete: 0
Action? [yYqQvVebB*rsf+?]
```

これはすでに”cn=module0, cn=config”が存在するためです。新規追加時に dn では親や先祖のノードに序数が指定されていれば必要ですが、自身のノードには自動的に付与されるので指定してはいけません。今回は”e”を押し編集モードに戻り、dn および attribute の cn の序数を取り除きましょう。

編集が終わったら、”y”を押して保存する前に、”v”を押します。すると、LDIF 形式で表示されます。これは後々役立ちます。

```
version: 1

dn: cn=module,cn=config
changetype: add
objectClass: olcModuleList
cn: module
olcModulePath: /usr/lib/ldap
olcModuleLoad: auditlog.la

dn: cn=module,cn=config
changetype: add
objectClass: olcModuleList
cn: module
olcModulePath: /usr/lib/ldap
olcModuleLoad: ppolicy.la
```

既にあるオブジェクトの attribute を変更する場合は、値を変更するだけです。オブジェクトを削除する場合は、対象のオブジェクトのすべての行を、特定の attribute だけを削除する場合はその行を削除するだけです。自分で LDIF を作るのと違い、非常に簡単であることがわかるでしょう。フォーマットが間違っている場合は前述のようにエラー

になる上、変更時に slapd の再起動が不要なので、実は slapd.conf での管理よりもとても便利です。

ユーザデータを変更する場合も基本的に同じです。 ldapvi コマンドで指定するオプションが異なるだけです。

```
$ ldapvi -h ldap://localhost -D cn=admin,dc=example,dc=org -b dc=example,dc=org
```

ldapvi の詳細は、ユーザマニュアルが充実しているのでそちらを参照ください。[*23]

10.4 debconf-utils と slapd-config で導入の自動化を進める

OpenLDAP を使う場合、slapd パッケージをインストールしますが、debconf による対話形式の設定が必要になります。はじめてインストールする場合などには便利なのですが、ある程度なれてきて、slapd の設定自体もあらかじめ決めてあると、これはかえって面倒なものになります。予め設定するパラメータは決めてあるのですから自動化したいですよね。そういう場合 debconf-utils パッケージと、 そして slapd-config を上手に使うことで自動化できるます。

10.4.1 事前準備

まず、テスト環境などを用意します。このテスト環境の目的は次の二つです。

1. debconf の設定を抽出する
2. slapd-config 用の LDIF を用意する

前者は slapd パッケージをインストールの際に今後設定したいパラメータを入力しておく必要があります。設定された状態で debconf-utils パッケージの debconf-get-selections コマンドで slapd に関する debconf のパラメータを抽出するためです。後者は slapd パッケージをインストールした直後の状態にしておき、この状態で ldapvi で予め設定したい attribute の値を変更し、前述の LDIF 形式の出力します。

10.4.2 debconf-get-selections でパラメータを抽出する

slapd をインストール状態で次のコマンドを実行します。

```
$ LANG=C sudo debconf-get-selections | grep slapd
slapd   slapd/password2 password
slapd   slapd/internal/generated_adminpw        password
slapd   slapd/internal/adminpw  password
slapd   slapd/password1 password
slapd   slapd/password_mismatch note
slapd   slapd/invalid_config    boolean true
slapd   shared/organization     string  example.org
slapd   slapd/upgrade_slapcat_failure   error
slapd   slapd/backend   select  HDB
slapd   slapd/dump_database     select  when needed
slapd   slapd/allow_ldap_v2     boolean false
slapd   slapd/no_configuration  boolean false
slapd   slapd/move_old_database boolean true
slapd   slapd/dump_database_destdir     string  /var/backups/slapd-VERSION
# Do you want the database to be removed when slapd is purged?
slapd   slapd/purge_database    boolean false
slapd   slapd/domain    string  example.org
```

このうち、”slapd/upgrade_slapcat_failure”と”#”から始まるコメント行は不要です。削除して slapd-debconf.txt などの任意のファイル名で保存します。これを slapd パッケージのインストール時に反映させるには、パッケージインストール前に debconf-set-selections コマンドを使います。

```
$ sudo debconf-set-selections slapd-debconf.txt
$ sudo DEBCONF_FRONTEND=noninteractive apt-get -y --force-yes install slapd
```

slapd の設定用 LDIF は、前述のテスト環境で ldapvi を使って生成します。次の 3 つの設定を行います。

- rootDN のパスワード

[*23] ldapvi User Manual http://www.lichteblau.com/ldapvi/manual/

- loglevel
  - 128
- access control
  - パスワード以外のデータの参照は anonymous で可能
  - データの更新は rootDN のみ行える
  - パスワードのみ自身で変更可能

まず、 ハッシュ化された rootDN のパスワードを slappasswd で生成します。

```
$ sudo slappasswd
New password:
Re-enter new password:
{SSHA}MEj4D59lrtA+6+0J4vIz2RTByA7KT6Zq
```

次に、ldapvi で上記の設定を行います。

```
$ sudo ldapvi -h ldapi:/// -Y EXTERNAL -b cn=config
```

変更箇所は下記の ”¡- (*)” の部分です。

```
0 cn=config
(snip)
olcLogLevel: 128 <- (*)
(snip)

10 olcDatabase={1}hdb,cn=config
(snip)
olcAccess: {0}to attrs=userPassword,shadowLastChange by self write by anonymous auth
 by dn="cn=admin,dc=example,dc=org" write by * none <- (*)
olcAccess: {1}to dn.base="" by * read <- (*)
olcAccess: {2}to * by dn="cn=admin,dc=example,dc=org" write by * read <- (*)
olcAccess: {3}to dn.subtree="dc=example,dc=org" by self read by * read <- (*)
olcAccess: {4}to * by * none <- (*)
(snip)
olcRootPW: {SSHA}MEj4D59lrtA+6+0J4vIz2RTByA7KT6Zq <- (*)
(snip)
```

変更後、LDIF を生成すると下記のように表示されます。

```
version: 1

dn: cn=config
changetype: modify
replace: olcLogLevel
olcLogLevel: 128
-

dn: olcDatabase={1}hdb,cn=config
changetype: modify
replace: olcAccess
olcAccess: {0}to attrs=userPassword,shadowLastChange by self write by anonymous auth
 by dn="cn=admin,dc=example,dc=org" write by * none
olcAccess: {1}to dn.base="" by * read
olcAccess: {2}to attrs=sshPublicKey by self write
olcAccess: {3}to * by dn="cn=admin,dc=example,dc=org" write by * read
olcAccess: {4}to dn.subtree="dc=example,dc=org" by self read by * read
olcAccess: {5}to * by * none
-
replace: olcRootPW
olcRootPW: {SSHA}MEj4D59lrtA+6+0J4vIz2RTByA7KT6Zq
```

残念ながら ldapvi にはこれを任意のファイルとして保存する機能ありません。コンソールをコピーして、適当なファイル名で保存します (今回は setup.ldif)。保存した LDIF を用い、ldapmodify コマンドを使えば slapd サーバに反映させることができます。

```
$ sudo ldapmodify -Y EXTERNAL -H ldapi:/// -f setup.ldif
```

パッケージのインストールから設定まで自動化するなら次のコマンドで行えます。

```
sudo apt-get install debconf ldap-utils
sudo debconf-set-selections slapd-debconf.txt
sudo DEBCONF_FRONTEND=noninteractive apt-get -y --force-yes install slapd
ldapmodify -Y EXTERNAL -H ldapi:// -f setup.ldif
```

例えば、LDAP 関連のツールのテストを、クリーン環境に CI などを使って行う場合は上記のコマンドで、リポジトリに push する度にテスト用の slapd を自動インストールしてテストを実行させることもできます。

10.5 python-ldap で stress-free に

さらにプログラムで LDAP を扱いましょう。今回は python の LDAP binding である、Python-LDAP というモジュールを使います[*24]。Debian では python-ldap パッケージです。

```
$ sudo apt-get install python-ldap
```

python-ldap で扱う基本的なデータ構造は下記のように、LDAP オブジェクトがリストになっており、LDAP オブジェクト自体はタプルになっています。タプルの第一要素が entryDN で、第二要素が辞書になっている LDAP オブジェクトの attributes です。各 attribute の値もリストになっているので、例えば objectClass のように複数の値を持つ場合にも対応できます。そして entryDN と attributes の各値はすべて文字列です。

```
[
    ('DN', {'attribute': ['value'], 'attribute': ['value', 'value', ...], ...}),
    ('DN', {'attribute': ['value'], 'attribute': ['value', 'value', ...], ...}),
    ...
]
```

10.5.1 検索 (ログイン) の例

検索の場合、指定した username が uid として存在するか、存在するなら simple bind で認証できるかの処理を行っています。

```
import ldap

UID = 'user0'
PASSWORD = 'passWord'

# LDAP サーバに接続
conn = ldap.initialize('ldap://localhost')
# anonymous で uid が "username" のユーザを検索
res = conn.search_s('ou=People,dc=example,dc=org', ldap.SCOPE_SUBTREE,
                    '(uid=%s)' % UID, ['uid'])
if res:
    # "username" が見つかった場合
    dn, attr_d = res[0]
    # simple bind で認証
    try:
        conn.simple_bind_s(dn, PASSWORD)
        login_result = True
    except ldap.INVALID_CREDENTIALS:
        # 認証失敗
        login_result = False
else:
    # "username" が存在しない
    login_result = False
```

10.5.2 データ追加

データの追加や更新の場合、ldap.modlist モジュールを使う必要があります。追加の場合には、ldap.modlist.addModList() を使いますが、前述の辞書型の attributes を引数にすると、下記のフォーマットのリストに変換されます。

```
[('attribute', ['value']),
 ('attribute', ['value', 'value', ...]), ...]
```

これを `ldap.add_s(dn, attrs_l)` の引数とすれば新規オブジェクトの追加ができます。サンプルは次の通りです。

[*24] http://www.python-ldap.org/

```
import ldap.modlist

rootdn = 'cn=admin,dc=example,dc=org'
rootpw = 'password'

dn = 'uid=mkouhei,ou=People,dc=example,dc=org'
attrs_d = {'objectClass': ['inetOrgPerson', 'posixAccount', 'shadowAccount'],
           'uidNumber': ['1000'],
           'gidNumber': ['1000'],
           'homeDirectory': ['/home/mkouhei'],
           'loginShell': ['/bin/bash'],
           'uid': ['mkouhei'],
           'cn': ['Kouhei'],
           'sn': ['Maeda']}

# attributes を辞書からリストに変換
attrs_l = ldap.modlist.addModlist(attrs_d)
# rootdn で bind
conn.simple_bind_s(rootdn, rootpw)
# dn を指定して新規オブジェクトを追加
conn.add_s(dn, attrs_l)
```

10.5.3 データの削除

データを削除する場合は、dn を指定するだけです。

```
conn.delete_s(dn)
```

10.5.4 python-ldap の筆者の使用例

サーバ/ネットワークまわりの自動化ツールや、ユーザ向けの Web ベースのツールの開発を python で行っています。認証には LDAP を使うことが多いため、これらのツール用に python-ldap を使ったモジュールを作っています。また、複数の異なる LDAP アカウントのツリーをマージするのにも使っています。ユーザアカウントのデータを変換してマージする必要があるため、LDIF を自分でパースして変換するのではなく、python のリストや辞書で処理できるのはとても楽です。変換元の LDAP サーバには標準スキーマではなく独自拡張のスキーマを使っているものあり、名前すら違う attribute もあるので python で処理できるのが大変便利です。

10.6 まとめ

ldapvi、python-ldap を使うと LDIF を自分で作成したり、パースする必要がなくなることを説明しました。 また、slapd-config であっても ldapvi を使えば、slapd.conf での場合と同様に簡単に管理できることも説明しました。注意点としては、slapd.conf での設定パラメータの名前が slapd-config では多少異なることですが、cn=config のツリーの root を指定すると、 ロードされているスキーマも見ることができるのでそこから検索、類推することができます。ストレスフルな LDAP の管理を少しでも stress-free にできるようになると良いですね。

参考文献

[1] RFC2849 "Example 1: An simple LDAP file with two entries" http://www.ietf.org/rfc/rfc2849.txt

[2] OpenLDAP Software 2.4 Administrator's Guide http://www.openldap.org/doc/admin24/index.html

[3] ldapvi User Manual, http://www.lichteblau.com/ldapvi/manual/

[4] python-ldap Documentation, http://www.python-ldap.org/doc/html/index.html

11 Samba で Linux の認証を Windows に統合してみたり

たかはしもとのぶ

11.1 無慈悲な一言、言われませんか w

Windows の大群の中で Linux サーバを運用しているとよく言われるのが、「その (Linux) サーバの認証、Active Directory でできないの?」という大変無慈悲な (笑) 一言ではないでしょうか。ということで、今回は Samba を使って Linux サーバの認証を Windows に統合する方法について、いろいろご紹介していきたいと思います。

11.2 Sambaって · · · · · ·

一応説明しておきますと、Debian を含む UNIX 系 OS 上でファイル共有をはじめとする Windows サーバの各種機能を実現するオープンソースのソフトウェアです。元々はファイル共有、プリンタ共有の機能から出発していますが、最近では Windows の中核機能である Active Directory のドメインコントローラ機能や、Windows ドメインのクライアント機能など、多くの機能を実装しています。

ただ、そのあたりの機能の紹介をしていると、Samba というより Windows の機能の紹介になってしまうので、今回は Linux メインな方でも需要がありそうな機能ということで、認証統合を取り上げてみました。

11.3 Samba でファイルサーバを構築してみる

まずは、Samba の機能紹介も兼ねて Samba でファイルサーバを構築してみましょう。これはパッケージを適切にインストールすれば簡単です。

```
# apt-get install samba
```

インストールの途中で聞かれる質問は、OK を押して進めておきましょう (後で修正します)。

インストールが終わったら、/etc/samba/smb.conf を編集します。デフォルトでは各ユーザのホームディレクトリを読み取り専用で共有する設定になっていますので、ここでは読み書き可能にしてみます。

```
[homes]
   comment = Home Directories
   browseable = no

# By default, the home directories are exported read-only. Change the
# next parameter to 'no' if you want to be able to write to them.
   read only = no ← yes から変更
```

もう一つ、Samba でファイル共有を行う例として/tmp を読み書き可能で共有してみましょう。次のような記述を smb.conf の末尾に追加します。

```
[tmp-share]
  path = /tmp
  read only = no
```

1 行目が共有名の指定で、ここでは tmp-share という名前を指定しています。2 行目は共有するパスの指定で、ここでは/tmp を指定しました。3 行目は読み書き可能とする設定です。デフォルトでは読み取り専用で共有されます。

次に、このサーバへのアクセスを行うユーザを作成します。/etc/passwd のユーザに加えて、Samba では Windows 独自の形式でハッシュ化された認証情報を扱う必要があるため、smbpasswd コマンドなどを用い、 Samba ユーザという独自のユーザを作成してパスワードを設定する必要があります。

```
# useradd -m local1
# smbpasswd -a local1
New SMB password:
Retype new SMB password:
Added user local1.
```

ここで次のようにして Samba サーバを再起動して smb.conf の設定変更を反映させます。

```
# /etc/init.d/samba restart
Stopping Samba daemons: nmbd smbd.
Starting Samba daemons: nmbd smbd.
```

再起動後、¥¥Samba **サーバの** IP **アドレス** として Windows からアクセスしてみましょう。ファイアウォールなどの設定を行っていなければ、図 3 のようにユーザ名とパスワードを聞かれるダイアログが表示されます。

図 3　ユーザー名とパスワードの入力ダイアログ

先ほど設定した user1 というユーザ名とパスワードを入力すれば、user1 というホームディレクトリの共有と tmp-share という先ほど作成した共有が見えているはずです (図 4)。

11.4　Samba を Active Directory(Windows ドメイン) に参加させてみる

次に、Samba を Active Directory に参加させて、先ほど構築したファイルサーバへのアクセスの際の認証を Windows ドメインに統合してみましょう。これもパッケージを適切にインストールしていれば簡単です。

まず次のようにして/etc/resolv.conf を編集して、DNS ドメイン、DNS サーバとして Active Directory の DNS サーバを指定します (まだ指定していない場合) 。ここでは W2K8R2AD3.LOCAL というドメイン名で DNS サーバの IP アドレスが 192.168.135.100 である場合の例を示します (DHCP から IP アドレスを取得する設定の場合、このファイルは上書きされてしまうので、固定 IP の設定にしておく必要があります):

図 4　エクスプローラーから Samba サーバの共有を参照

```
domain w2k8r2ad3.local ← Active Directory のドメイン名を指定
search w2k8r2ad3.local
nameserver 192.168.1.100 ← Active Directory の DNS サーバ（通常はドメインコントローラ）の IP アドレスを指定
```

ついで、smb.conf ファイルを修正します。ここでは W2K8R2AD3.LOCAL という Active Directory ドメインに参加させる場合を例に取って説明します。

```
[global]
  ...

  workgroup = W2K8R2AD3

  ...

  security = ads
  realm = W2K8R2AD3.LOCAL
```

最後に、net ads join コマンドを使って Samba をドメインに参加させます

```
# net ads join -U administrator
Enter administrator's password:
Using short domain name -- W2K8R2AD3
Joined 'SQUEEZE32-5' to realm 'W2K8R2AD3.local'
No DNS domain configured for squeeze32-5. Unable to perform DNS Update.
DNS update failed!
```

-U オプションに続き、ドメイン参加に使用するユーザを指定します。これは必ずしも Administrator である必要はありませんが、Active Directory 側の設定に依存しますので、事前に確認していてください。参加の際、DNS update failed! というエラーが出ますが、これは無視して構いません。

次に、このサーバへのアクセスを行うユーザを作成します。認証は Active Directory で行うので、Samba ユーザは不要です。/etc/passwd にユーザを追加します。このユーザのユーザ名は Active Directory 側と合わせる必要があります。ここでは Active Directory 側に aduser01 というユーザが既に作成済である前提で、aduser01 を追加して見ます:

```
# useradd -m aduser01
```

ここで次のようにして Samba サーバを再起動して smb.conf の設定変更を反映させます。

```
# /etc/init.d/samba restart
Stopping Samba daemons: nmbd smbd.
Starting Samba daemons: nmbd smbd.
```

再起動後、Windows 側に aduser01 としてログオンの上、~~¥¥~~Samba サーバの IP アドレスとして Windows からアクセスしてみましょう。認証を聞かれることなく、Samba サーバにアクセスできるはずです。何かファイルを作成の上 Debian 上で確認すると、次のようにファイルの所有者が aduser01 になっていることが確認できると思います。

```
# ls -l
total 4
drwx------ 2 root     root     4096 Apr 14 11:24 ssh-XHMOEq1038
-rwxr--r-- 1 aduser01 aduser01    0 Apr 14 11:22 test01.txt
```

Active Directory 参加前との簡単な比較を図 5 に示します:

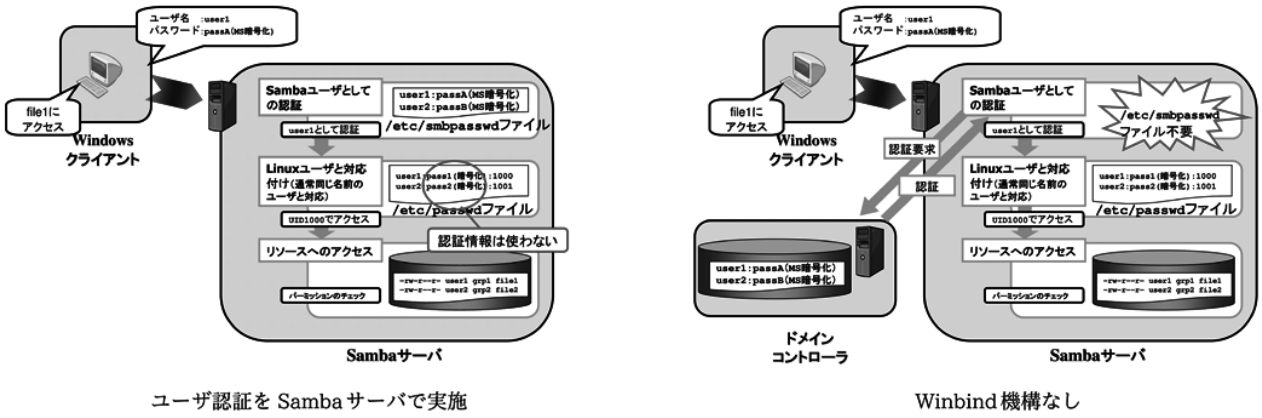


図 5　Active Directory 参加前後の認証フローの比較

11.5　ユーザの作成を自動化する

ここまでの設定で、認証の統合には成功しました。しかしこの状態だと認証を行いたいユーザ毎に Debian 上の `/etc/passwd` にユーザを作成する必要があります。多数のユーザがアクセスする必要がある環境では、ユーザのメンテナンスがかなりの負荷になってしまうと思います。

Samba では **Winbind** という機能を使うことで、ユーザの自動生成が可能になります。パッケージを使っていれば、この設定も簡単に行えます。

まず、次のようにして winbind パッケージをインストールします:

```
# apt-get install winbind
```

`wbinfo -t` コマンドを用いて、次のように RPC が成功することを確認しておきましょう:

```
# wbinfo -t
checking the trust secret for domain W2K8R2AD1 via RPC calls succeeded
```

引き続き、`/etc/nsswitch.conf` の `passwd:`および `group:`行に、次のように `winbind` というキーワードを追加します:

```
passwd:         compat winbind
group:          compat winbind
```

最後に、`smb.conf` 内の次の行のコメントを外し、設定を追加して、winbind を再起動します:

```
# Some defaults for winbind (make sure you're not using the ranges
# for something else.)
   idmap uid = 10000-20000
   idmap gid = 10000-20000
   template shell = /bin/bash
   template homedir = /home/%U
```

ここで Active Directory に例えば aduser02 というユーザを追加して、その情報を取得すると ……

Samba サーバの`/etc/passwd` では特にユーザの追加を行っていないにも関わらず、次のように Winbind 機構によって自動生成されたユーザ情報が返却されるはずです:

```
$ id 'W2K8R2AD3\aduser02'
uid=10001(W2K8R2AD3\aduser02) gid=10000(W2K8R2AD3\domain users) groups=10000(W2K8R2AD3\domain users)
$ getent passwd  'W2K8R2AD3\aduser02'
W2K8R2AD3\aduser02:*:10001:10000:aduser 02:/home/aduser02:/bin/bash
```

Windows にユーザでログオンして Samba サーバの共有に何かファイルを作成すると、次のように、ユーザ名、グ

ループ名に自動生成されたものが表示されていることが確認できます:

```
$ ls -l /tmp
total 4
drwx------ 2 root                  root                       4096 Apr 14 12:40 ssh-wQPxWv1345
-rwxr--r-- 1                  1001 aduser01                      0 Apr 14 11:22 test01.txt
-rwxr--r-- 1 W2K8R2AD3\aduser02 W2K8R2AD3\domain users           0 Apr 14 12:39 test02 - コピー.txt
```

Winbind 機構のない状態と Winbind 機構を有効にした状態での比較を次の図 6 に示します:

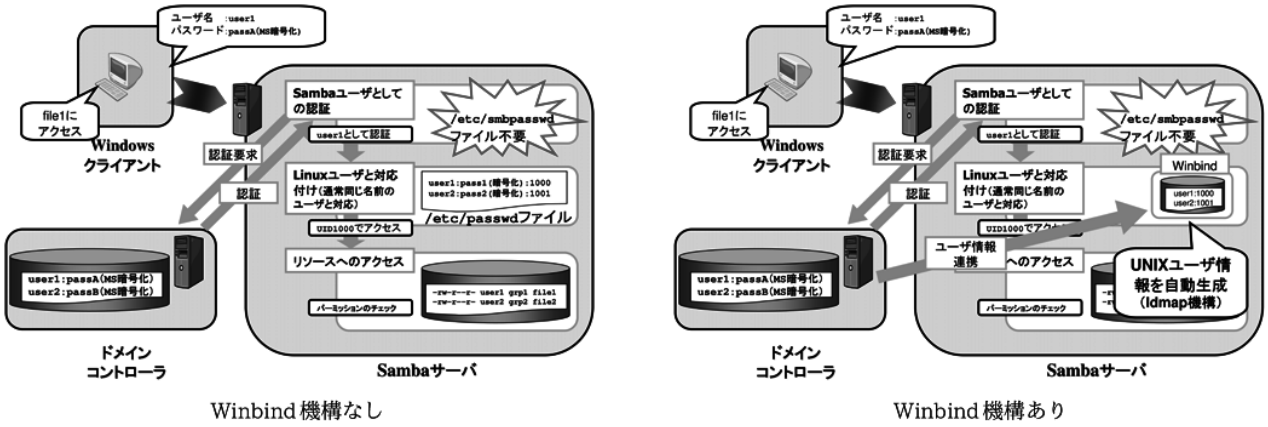


図 6　Winbind 機構有無による認証フローの比較

ここで自動生成されるユーザ、グループの UID,GID やシェルなどの情報は、さまざまな設定でカスタマイズすることが可能です。今回は時間がないので省略しますが、1 点だけ、生成されるユーザ名、グループ名に付加されるドメイン名部分が煩雑だと感じる場合は、`smb.conf` に次のパラメータを設定してください。

```
  winbind use default domain = yes
```

次のようにドメイン名がない単純な名前で表示されるようになります:

```
$ id aduser02
uid=10001(aduser02) gid=10000(domain users) groups=10000(domain users)
```

ただし、当然ですが Samba が所属するドメインが他のドメインと信頼関係を結んでいるような複数ドメイン環境では名前が重複することがあります。

11.6　ssh の認証を Windows に統合してみる

ここまでの設定で、Samba を用いる際のユーザ認証と、ユーザの自動作成について説明しました。最後に Samba 以外のプロダクトからこのユーザ認証を活用する方法について説明します。ここでは ssh を例に取って説明しますが、PAM で認証を行うプロダクトであれば、同様の適用が可能です。

実は、ここまでの設定を行っていれば、Active Directory のユーザを用いて Linux にログインすることが (他の方法でログインを抑止していない限り) 可能です。次に例を示します:

```
$ ssh 192.168.135.35 -l W2K8R2AD3\\aduser02
W2K8R2AD3\aduser02@192.168.135.35's password:
Linux squeeze32-5 2.6.32-5-686 #1 SMP Mon Jan 16 16:04:25 UTC 2012 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Could not chdir to home directory /home/aduser02: No such file or directory
W2K8R2AD1\aduser02@squeeze32-5:/$
```

ホームディレクトリを作成していないため、エラーが発生していますが、ログイン自体は無事に成功していることが分かります。もちろんこの aduser02 というユーザは Winbind 機構によって自動作成されたものですので、/etc/passwd 上には存在していません。

ホームディレクトリがないと何かと不便ですが、手作業で作成するのもゲイがないので、最後にホームディレクトリを自動で作成する方法を紹介します。まず次のようにして libpam-modules をインストールしてください:

```
# apt-get install libpam-modules
```

その後、/etc/pam.d/common-session ファイルに次のように 1 行追加します:

```
session optional                        pam_winbind.so
# end of pam-auth-update config
session    required    pam_mkhomedir.so skel=/etc/skel/ umask=0022 ←この行を追加
```

オプションの詳細は pam_mkhomedir(8) を参照してください。

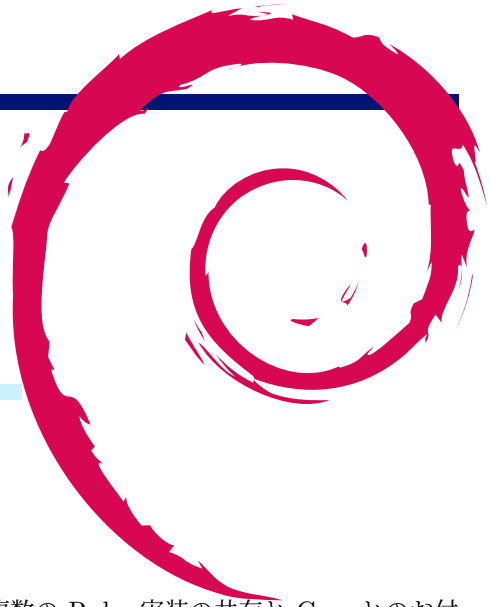
Debian では pam-auth-update というコマンドで自動的に PAM の設定を行う方法もありますが、残念ながら上記の設定はサポートしていないため、手作業でファイルを修正する必要があります。

設定後、先ほどと同様に別のマシンから aduser02 でログインすると、次のように Creating directory ... というメッセージが表示されて、ホームディレクトリが自動作成されていることが確認できます:

```
$ ssh 192.168.135.35 -l W2K8R2AD3\\aduser02
W2K8R2AD3\aduser02@192.168.135.35's password:
Creating directory '/var/home/aduser02'.
Linux squeeze32-5 2.6.32-5-686 #1 SMP Mon Jan 16 16:04:25 UTC 2012 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Apr 14 11:40:23 2013 from 192.168.135.16
W2K8R2AD3\aduser02@squeeze32-5:~$
```



12 Ruby In Wheezy

佐々木洋平

次期安定版 Debian 7.0 (Wheezy) における Ruby 環境について、特に複数の Ruby 実装の共存と Gem とのお付き合いの仕方について、簡単にお話しします。

12.1 “Ruby” in Wheezy

Ruby には複数の実装が存在しています。Debian Wheezy では以下の Ruby インタープリタが使用できます:

インタープリタ	パッケージ名	注意書き
MRI 1.8.7	`ruby1.8`	
MRI 1.9.3	`ruby1.9.3`	`ruby1.9.1` パッケージの実態は `ruby1.9.3` だったりします。
JRuby	`jruby`	パッケージは Debian Java Maintainers によって管理されています
Rubinius	`rubinius`	現在作業中です. ITP#591817
mruby	`mruby`	現在作業中です. ITP#697835

「`ruby1.9.1` パッケージの実態は `ruby1.9.3` だったりします。」について、Ruby の soname が変わっていないので `libruby1.9.1` 等のパッケージが残っており、`ruby1.9.3` パッケージは `/usr/bin/ruby1.9.1` への symbolic link を提供するだけだったりします。

```
% dpkg -S ruby1.9.3
ruby1.9.3: /usr/bin/ruby1.9.3
ruby1.9.3: /usr/share/man/man1/ruby1.9.3.1.gz
ruby1.9.3: /usr/share/doc/ruby1.9.3
ruby1.9.3: /usr/share/doc/ruby1.9.3/NEWS.Debian.gz
ruby1.9.3: /usr/share/doc/ruby1.9.3/NEWS.gz
ruby1.9.3: /usr/share/doc/ruby1.9.3/changelog.Debian.gz
ruby1.9.3: /usr/share/doc/ruby1.9.3/changelog.gz
ruby1.9.3: /usr/share/doc/ruby1.9.3/copyright
ruby1.9.3: /usr/share/doc/ruby1.9.3/README.gz
% ls -la /usr/bin/ruby1.9.3
lrwxrwxrwx 1 root root 9 Feb 23 23:44 /usr/bin/ruby1.9.3 -> ruby1.9.1*
```

他にも、`Topaz` や `HPC Ruby Compiler` など、(個人的に) 気になる実装がありますが、まだ Debian でどうこうというお話にはなっていません。

12.2 複数の Ruby 実装の切り替え

さて、これだけ複数の Ruby 実装があると、それを切り替えて使いたくなるのが人情だと思います。Debian には、同じ機能を提供するソフトウェアをきりかえる `update-alternatives` という仕組みが既にあります。ですが、`/usr/bin/ruby` を切り替えたら `gem` や `irb` なんかも切り替えたくなるでしょう。そのため、これらを一度に変換するためのコマンドが提供されています。

12.2.1 システム全体でデフォルトの Ruby を切り替えるには?: ruby-switch

システム全体でデフォルトの Ruby インタープリタを選択するために、ruby-switch パッケージが使用可能です。これは root として (もしくは sudo を使って) 実行する必要があります。

```
# ruby -v
ruby 1.8.7 (2011-06-30 patchlevel 352) [x86_64-linux]
# ruby-switch --list
ruby1.8
ruby1.9.1
# ruby-switch --set ruby1.9.1
update-alternatives: using /usr/bin/gem1.9.1 to provide /usr/bin/gem (gem) in manual mode.
update-alternatives: using /usr/bin/ruby1.9.1 to provide /usr/bin/ruby (ruby) in manual mode.
# ruby -v
ruby 1.9.3p0 (2011-10-30 revision 33570) [x86_64-linux]
# ruby-switch --auto
update-alternatives: using /usr/bin/ruby1.8 to provide /usr/bin/ruby (ruby) in auto mode.
update-alternatives: using /usr/bin/gem1.8 to provide /usr/bin/gem (gem) in auto mode.
# ruby -v
ruby 1.8.7 (2011-06-30 patchlevel 352) [x86_64-linux]
```

12.2.2 ユーザ毎にデフォルトの Ruby インタープリタを選択するには: rbenv

ユーザアカウント毎にデフォルトの Ruby インタープリタを切り替えるには、rbenv パッケージを使うのが良いでしょう。

```
% ruby -v
ruby 1.8.7 (2011-06-30 patchlevel 352) [x86_64-linux]
% rbenv init
# Load rbenv automatically by adding
# the following to ~/.bash_profile:

eval ‘‘$(rbenv init -)"
% echo ’eval ‘‘$(rbenv init -)"’ >> ~/.bash_profile # or ~/.bashrc, depends on your setup
% rbenv versions
% rbenv alternatives
% rbenv versions
  1.8.7-debian
  1.9.3-debian
% rbenv global 1.9.3-debian
% ruby -v
ruby 1.8.7 (2011-06-30 patchlevel 352) [x86_64-linux]
```

一見ちゃんと動作していないように見えますが、これは現在実行中のシェルが ”ruby” の位置として /usr/bin/ruby をキャッシュしているからです。新しいシェルを開始した後には、デフォルトの Ruby を行ったり来たり切り替えることができます。

```
% ruby -v
ruby 1.8.7 (2011-06-30 patchlevel 352) [x86_64-linux]
% rbenv global 1.9.3-debian
% ruby -v
ruby 1.9.3p0 (2011-10-30 revision 33570) [x86_64-linux]
% rbenv global 1.8.7-debian
% ruby -v
ruby 1.8.7 (2011-06-30 patchlevel 352) [x86_64-linux]
```

12.2.3 Debian パッケージになっていない Ruby をインストールするには: ruby-build

ruby-build を使用することで、Debian でまだ使用可能になっていない Ruby インタープリタをインストールすることができます。しかしながら、このパッケージの README.Debian ファイルに書かれている内容に注意して下さい:

```
  While ruby-build is a great tool to build Ruby versions that are not
  available via APT, you should still use the Debian-packaged versions
  of Ruby whenever possible since they are tested and supported by the
  Debian community.

  Please do not report bugs you encounter while using your homebuilt
  Rubies to the Debian team; Rubies built by yourself are not supported.
```

というわけで、詳細は man ruby-build ということで。

12.3 Debian における Ruby パッケージング

Debian では Ruby 本体のパッケージングは pkg-ruby チーム、Gem 等で配布されている (拡張) ライブラリやアプリケーションは pkg-ruby-extras チームによってメンテされています。次の話題は pkg-ruby-extras チームが良くやっていること、すなわち Gem で配布されている Ruby (拡張) ライブラリの Debian パッケージ化について、です。

12.4 Gem パッケージを Debian パッケージに: gem2deb

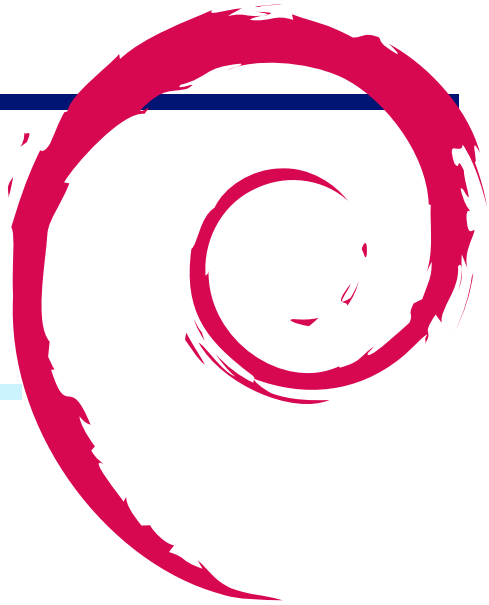
Gem で配布されている Ruby (拡張) ライブラリは gem2deb を用いることで Debian パッケージに変換可能です。**たいていの場合は**。gem2deb によるお手軽パッケージングの成功率は perl の dh_make_perl [*25] ぐらい、だと思って下さい (もっと低いかも)。

たとえば hogehoge という gem を Debian パッケージにしたい場合には

```
% sudo apt-get install gem2deb
...
% gem fetch hogehoge
...
% gem2deb hogehoge[version].gem
...
% sudo dpkg -i ruby-hogehoge_[version].deb
```

で良い筈です。ちなみにこれらは MRI 1.8.7, MRI 1.9.3 向けのパッケージとなります。

[*25] CPAN にある perl ライブラリ/モジュールを Debian パッケージにするコマンド



13 gdb python 拡張その 1

野島 貴英

13.1 はじめに

Debian パッケージのバイナリのバグ潰しを行う時、gdb などのデバッガを使ってデバッグをする事も多いかと思います。しかしながら、プログラムの構造が大規模/複雑になってくると、デバッグ作業も大変になっていきます。また、プログラムについても、抽象化が激しく行われていると、デバッガ無しには解析が非常に困難かと思います。

gdb は v7.0 から python 拡張が搭載されており[*26]、gdb を python スクリプトで操る事ができます。こちらを利用する事により、人手では、工数かかりすぎでなかなか進まなかったデバッグも可能になっていくと思います。

今回は、gdb の python 拡張の一部を説明し、最後は C 言語で作られたプログラムの実行トレースを実際に python 拡張機能を用いて、取得してみるところまでやってみます。

13.2 Debian sid の gdb python 拡張の基本情報

debian sid に含まれている gdb は最初から python 拡張が有効にしてあるようです。表 1 に基本情報を載せます。

項番	項目	値
1	gdb バージョン	7.4.1
2	python バージョン	2.7.3
3	python サーチパス	/usr/share/gdb/python,/usr/lib/python2.7 以下

表 1 Debian sid 環境に含まれる gdb と python の基本情報

今回は debian sid amd64 環境でテストした結果を元に説明を行います。

13.3 python 拡張を簡易的に動かしてみる

gdb より python 言語を簡易的に呼び出す場合、gdb のコマンドラインから接頭辞として python をつけて起動します。また、gdb コマンドラインから python のみを指定すると、Ctrl+d を押下するまで、複数の python 構文を指定できます。(なお、Ctrl+d を押下すると、gdb のコマンドラインに復帰すると同時に入力した複数の python 構文が一度に評価されます)

なお、python で一度定義した変数などは gdb を終了するまで何度でも参照できます。

[*26] http://ftp.gnu.org/gnu/gdb/ 調べ

```
$ gdb
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
... 中略...
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb) python print "hello world"
hello world
(gdb) python a=[1,2,3,4]
(gdb) python print a
[1, 2, 3, 4]
(gdb) python a.append(5)
(gdb) python print a
[1, 2, 3, 4, 5]
(gdb) python
>import sys
>print sys.path
>print sys.version_info
>ここで Ctrl+d
['/usr/share/gdb/python', '/usr/lib/python2.7',
  '/usr/lib/python2.7/plat-linux2', '/usr/lib/python2.7/lib-tk',
  '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload',
  '/usr/local/lib/python2.7/dist-packages',
  '/usr/lib/python2.7/dist-packages',
  '/usr/lib/python2.7/dist-packages/PIL',
  '/usr/lib/python2.7/dist-packages/gst-0.10',
  '/usr/lib/python2.7/dist-packages/gtk-2.0',
  '/usr/lib/pymodules/python2.7']
sys.version_info(major=2, minor=7, micro=3, releaselevel='final', serial=0)
```

13.4 gdb の python 拡張の構造

gdb の python 拡張は図 7 のような構造となっています。

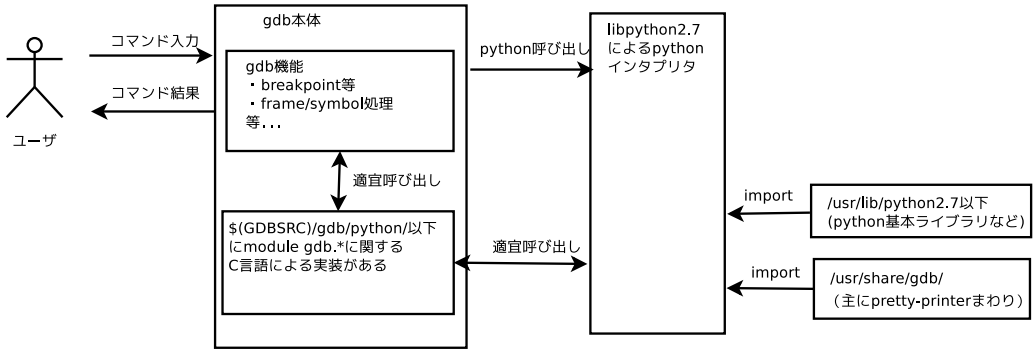


図 7　gdb と python 拡張の構造

13.5 module gdb のマニュアル

gdb 本体に module gdb が実装されているため、module gdb の python ドキュメントについては、gdb 上で python から help(gdb) を呼び出す必要があります。

以下に読み方を示します。なお、以降、(gdb) は gdb のプロンプトを示します。

```
(gdb) python help(gdb)
Help on package gdb:

NAME
    gdb

FILE
    (built-in)

PACKAGE CONTENTS
    command (package)
    printing
    prompt
    types

... 中略...
```

また、gdb の python 拡張についてのさらに詳しい説明は、info gdb にて Extending GDB → python の項目から

参照できます。

13.6 module gdb に定義されているオブジェクト群

図 8〜図 9 に、module gdb に定義されているオブジェクト群の class 図を載せます。

gdb 拡張用の python スクリプトを記載する場合、これらオブジェクトを併用して gdb とのデータのやりとり、あるいは、操作を行います。

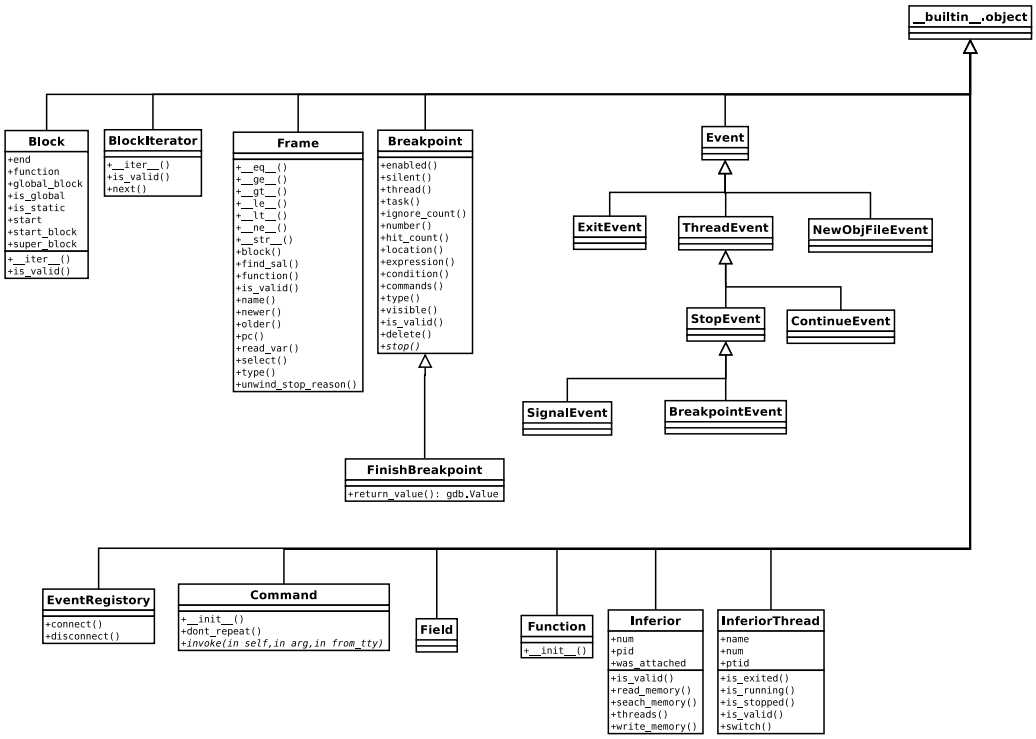


図 8　module gdb の class 図 (その 1)

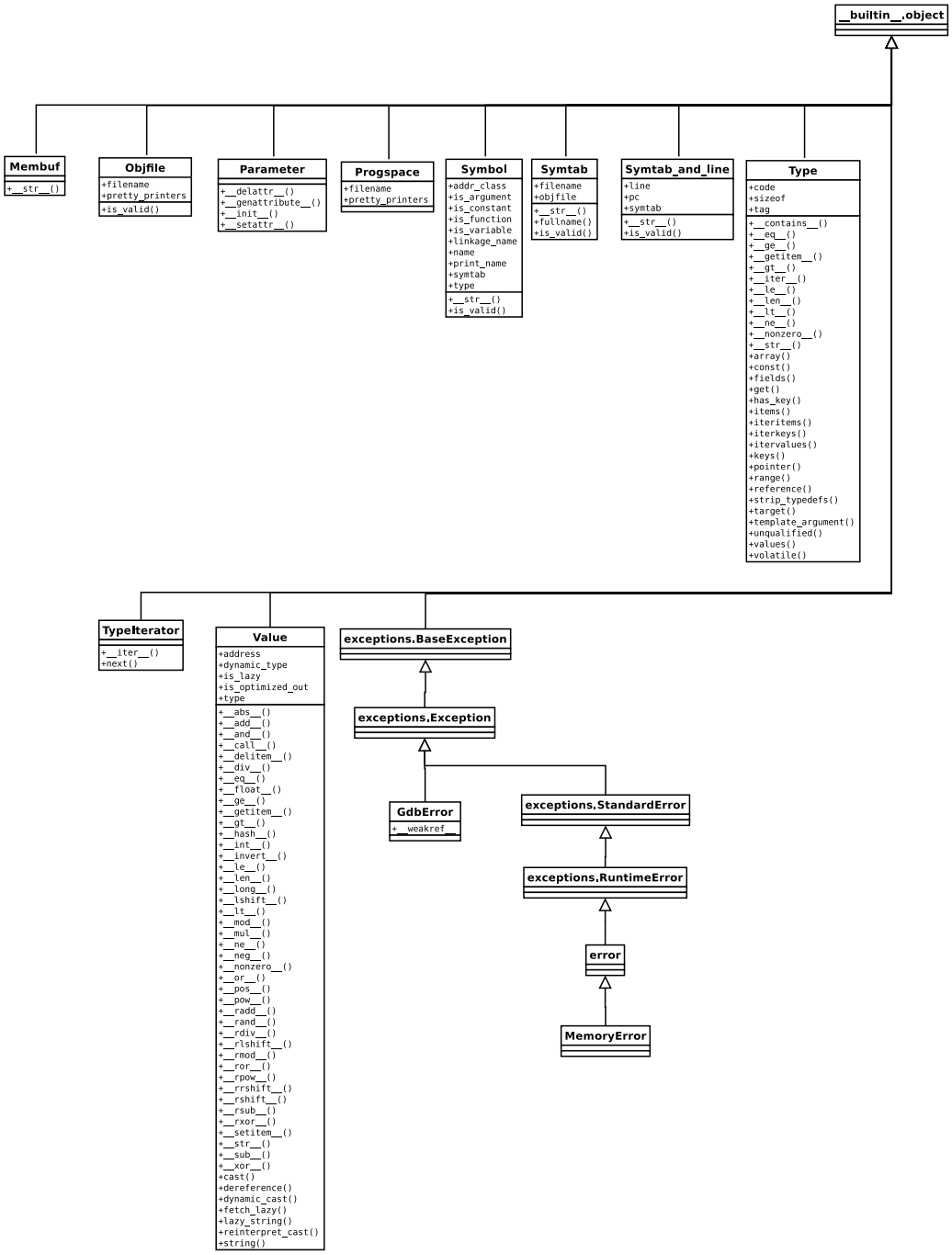


図 9　module gdb の class 図 (その 2)

13.7　gdb のコマンドを増やしてみる

gdb の python 拡張は柔軟な機能を持つため、いろいろな使い方ができます。ここでは、試しに gdb のコマンドを増やしてみます。

gdb のコマンドを python から増やすには、gdb.command クラスを継承したクラスを用意し、gdb.command.__init__() にてコマンド名と共に登録する事により行います。

info gdb の Extending GDB → Python → Python API → Commands In Python に記載されている方法を試して、hello-world コマンドを登録してみます。

```
-----hello.py の中身ここから-----
# -*- coding: utf-8 -*-
# coding:utf-8
import gdb
class HelloWorld (gdb.Command):
  """ Greet the whole world """
  def __init__ (self):
     super(HelloWorld, self).__init__ ("hello-world",gdb.COMMAND_OBSCURE)

  def invoke (self,arg, from_tty):
     print "Hello, World! arg=["+str(arg)+"]"

HelloWorld()
-----hello.py の中身ここまで-----
```

追加したコマンドについていろいろ実行してみます。

```
(gdb) source hello.py
(gdb) hello-[ここで TAB を押すと補完される]
(gdb) hello-world foo,bar,com
Hello, World! arg=[foo,bar,com]
(gdb) help obscure
Obscure features.

List of commands:

... 中略...
hello-world --  Greet the whole world
... 中略...
```

コマンド hello-world が追加されています。また、引数は invoke() の arg に文字列としてまとめて入ります。また、コマンドカテゴリの OBSCURE に登録されている事が help obscure にて判ります。

13.8 作った python スクリプトを自動で読み込ませるには

ところで、gdb の python 拡張を理解するにつれ、高度なデバッグ用スクリプトを用意するようになってくると思います。すると、作った python スクリプトをいつも gdb に自動で読み込ませておきたくなるかと思います。方法としては、以下の 3 つの方法があります。

13.8.1 ${HOME}/.gdbinit を使う方法

以下のようなファイルを`${HOME}/.gdbinit` に記載しておきます。

```
----${HOME}/.gdbinit ここから-----
source /home/foo/bar/my-gdb-func.py
----${HOME}/.gdbinit ここまで-----
```

こうすると、`${HOME}`にホームディレクトリがあるようなユーザが gdb を起動した時に、自動的に/home/foo/bar/my-gdb-func.py がロードされて評価されるようになります。

13.8.2 セクション名：.debug_gdb_scripts を使う方法

バイナリ形式によりますが、任意のセクション名を持つ事が可能なバイナリ形式（例：ELF,DWARF）にて、.gdb_gdb_scripts セクションを作成し、ここにスクリプト名を打ち込んでおく事ができる場合があります。この場合、カレントディレクトリにある、同名のスクリプトを gdb が自動的にロードしてくれます。なお、本機能は、gdb 変数の auto-load-script が on の時に有効です（デフォルトは on。）

以下に例を示します。ここでは先ほどの hello.py を自動でロードするように asm{} 命令で.debug_gdb_scripts セクションを直接指定しています。

```
------hello.c の中身ここから------
#include <stdio.h>

asm(
".pushsection \".debug_gdb_scripts\",\"MS\",@progbits,1\n"
".byte 1\n"
".asciz \"hello.py\"\n"
".popsection \n"
);

int main(int argc,char **argv)
{
        printf("hi there!");
        return 0;
}
------hello.c の中身ここまで------

実行結果:
$ gcc -o hello hello.c
$ ls
hello hello.py hello.c
$ gdb hello
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
... 中略...
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/xxxx/hello...done.
(gdb) info auto-load-scripts
Loaded  Script
Yes     hello.py
        full name: /home/xxxx/hello.py
(gdb) hello-world
Hello, World! arg=[]
```

13.8.3 “バイナリ名”-gdb.py をスクリプト名に使う方法

ファイル名として、“バイナリ名”-gdb.py をファイル名に持つ python スクリプトをカレントディレクトリに置いておくと、gdb がバイナリ名のファイルをロードした時、自動でロードしてくれます。なお、本機能は、gdb 変数の auto-load-script が on の時に有効です（デフォルトは on）

以下の例では、gdb hello とすると、無事先ほどの hello.py が読み込まれ、hello-world コマンドが登録されている事がわかります。

```
------hello.c の中身ここから------
#include <stdio.h>

int main(int argc,char **argv)
{
        printf("hi there!");
        return 0;
}
------hello.c の中身ここまで------

コンパイル：
$ gcc -o hello hello.c
$ mv hello.py hello-gdb.py (←先ほどの hello.py の名前を"バイナリ名"-gdb.py へ変更)
$ ls
hello hello-gdb.py hello.c
$ gdb hello
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
... 中略...
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/xxxx/hello...done.
(gdb) info auto-load-scripts
Loaded  Script
Yes     /home/xxxx/hello-gdb.py
(gdb) hello-world
Hello, World! arg=[]
```

13.9 break/finish と応用例について

デバッガの基本機能に breakpoint があります。こちらの機能を python から利用するには gdb.Breakpoint class 及び、gdb.FinishBreakpoint class を継承する事により行います。これら class を用いれば、gdb の break/watch/finish コマンドを独自拡張できます。

ここでは、応用としてバイナリ内部の関数呼び出しの記録を取るような python スクリプトを書いてみます。

```
-------calltracer.py ここから---------
# -*- coding: utf-8 -*-
# coding:utf-8
import gdb
class _CallTracerFinishBreakpoint(gdb.FinishBreakpoint):
        def __init__(self, name, stack):
                super(_CallTracerFinishBreakpoint, self).__init__(internal=True)
                self._stack_ptr=stack
                self._name=name
                self.silent=True
        def stop(self):
                print (" " * (len(self._stack_ptr)))+"<="+self._name
                self._stack_ptr.pop()
                return False
        def out_of_scope(self):
                print "Abnormal jump out frame"
                print (" " * (len(self._stack_ptr)))+"<="+self._name
                self._stack_ptr.pop()
                return False

class _CallTracerBreakpoint(gdb.Breakpoint):
        def __init__(self, spec, name, stack):
                super(_CallTracerBreakpoint, self).__init__(spec,
                                                            gdb.BP_BREAKPOINT,
                                                            internal = False)
                self._stack_ptr=stack
                self._name=name
                self.silent=True
        def stop(self):
                self._stack_ptr.append(self._name)
                print (" " * (len(self._stack_ptr)))+"=>"+self._name
                try:
                        _CallTracerFinishBreakpoint(self._name, self._stack_ptr)
                except:
                        print "uh? cant put finish break on "+self._name
                return False

class _ReAnalyzeCallTracer(gdb.Command):
        """ reanalyze symbol for calltracer """
        def __init__(self):
                super(_ReAnalyzeCallTracer, self).__init__('reanalyzecalltracer',
                                                    gdb.COMMAND_OBSCURE)
                self._stack=[]
        def _retrive_ptrs(self):
                info=gdb.execute("info break",False, True)
                info_lines=info.splitlines()
                ptrs={}
                for idx in range(0,len(info_lines[1:])):
                        tokens=info_lines[idx+1].split()
                        if len(tokens) > 5:
                                if ptrs.has_key(tokens[4]) == False:
                                        ptrs[tokens[4]]=" ".join(tokens[5:])
                return ptrs
        def invoke(self, arg, from_tty):
                break_info=self._retrive_ptrs()
                gdb.execute("delete",False, True)
                gdb.execute("set pagination off")
                for addr,name in break_info.iteritems():
                        _CallTracerBreakpoint(r'*'+addr,
                                              name,self._stack)
_ReAnalyzeCallTracer()

class _PrepareCallTracer(gdb.Command):
        """ prepare call tracer for c """
        def __init__(self):
                super(_PrepareCallTracer, self).__init__('prepcalltracer',
                                                    gdb.COMMAND_OBSCURE)
        def invoke(self, arg, from_tty):
                gdb.execute("rbreak",False, True)
                gdb.execute("reanalyzecalltracer",False, True)
                print "prepare done!"

_PrepareCallTracer()
-------calltracer.py ここまで---------
-------デバッグ対象：chkfunc.c ここから-------
#include<stdio.h>

void foo_a(const char *str)
{
        printf("%s\n",str);

}
void caller_bar(void)
{
        foo_a("caller is bar!");
}
int main(int argc,char **argv)
{
        foo_a("caller is main!");
        caller_bar();
        return(0);
}
-------デバッグ対象：chkfunc.c ここまで-------
```

実行結果：

```
$ gcc -O0 -g -o chkfunc chkfunc
$ ls
chkfunc.c chfunc calltracer.py
$ gdb ./chkfunc
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
... 中略...
Reading symbols from /home/foo/bar/chkfunc...done.
(gdb) source calltracer.py
(gdb) prepcalltracer
prepare done!
(gdb) run
Starting program: /home/foo/bar/chkfunc
 =><_start>
uh? cant put finish break on <_start>
  =><__libc_start_main@plt>
   =><__libc_csu_init>
    =><_init>
     =><call_gmon_start>
     <=<call_gmon_start>
    <=<_init>
    =><frame_dummy>
     =><register_tm_clones>
     <=<frame_dummy>
    <=<register_tm_clones>
   <=<__libc_csu_init>
   =>in main at chkfunc.c:21
uh? cant put finish break on in main at chkfunc.c:21
    =>in foo_a at chkfunc.c:12
     =><puts@plt>
caller is main!
     <=<puts@plt>
    <=in foo_a at chkfunc.c:12
    =>in caller_bar at chkfunc.c:17
     =>in foo_a at chkfunc.c:12
      =><puts@plt>
caller is bar!
      <=<puts@plt>
     <=in foo_a at chkfunc.c:12
    <=in caller_bar at chkfunc.c:17
    =><__do_global_dtors_aux>
     =><deregister_tm_clones>
     <=<deregister_tm_clones>
    <=<__do_global_dtors_aux>
    =><_fini>
    <=<_fini>
[Inferior 1 (process 6413) exited normally]
Abnormal jump out frame
   <=<__libc_start_main@plt>
warning: Error removing breakpoint -5
(gdb)
```

無事、バイナリ内部の関数呼び出しの記録が取れているかと思います。

13.10 終わりに

今回 gdb の python 拡張のいくつかを紹介してみました。python を用いる事で、いろいろなデバッグ手法が取れるかと思います。

次回は、Frame class/Value class 等の応用について紹介したいと思います。

参考文献

[1] Free Software Foundation, “info gdb”

[2] “PythonGdbTutorial”,`http://sourceware.org/gdb/wiki/PythonGdbTutorial`

[3] Free Software Foundation, `$(GDBSRC)/gdb/testsuite/gdb.python` 以下のテスト用ファイル群

14 日本における DFSG の求める自由と２０１２年改正著作権法

上川 純一

14.1 はじめに

Debian の基本的な思想ともいえる Social Contract、およびその中に含まれる DFSG は自由にソフトウェアの開発ができる環境を理想と考えるものです。ユーザを尊重するよりまえに１００％フリーソフトウェアであることをうたっている点などもあり、ソフトウェアの複製や改変をする自由を尊ぶユーザが Debian を選択しています。

しかし自由なソフトウェアの開発は自明に支持されるものではありません。自由なライセンスのソフトウェアを確保することは難しく、改変不自由な秘密のソフトウェアを買ってくるのは簡単です。各国においての法律、企業による技術的設計・制約などに影響されます。

ソフトウェアを自由に勉強して改変して開発できる環境がないとソフトウェア利用の幅が失われるだけでなく、ソフトウェア開発できる人の層が薄くなり、結果として日本国におけるソフトウェア開発能力の低下に至るでしょう。

自分で書いたソフトウェアを動かせないハードウェアしか市場にでまわってないのであれば、ソフトウェアを作成するという経験を享受する機会が極端に減ります。

一年の棚卸の意味も兼ねて、最近の日本の事情を調べてみました。

14.2 著作権法改正

最近大きな法律の変化としては、２０１２年の改正著作権法があります。議論の経緯や結果どういうことになったのか、などいろいろな情報が提供されているようなので我々のソフトウェア開発にどういう影響がありそうか眺めてみました。

リッピングソフトやマジコンが規制対象になったと報道されていますが、具体的にはどう変わったのでしょうか。文化庁のウェブサイトにある解説 [1] によると「(4) 著作権等の技術的保護手段に係る規定の整備」にあたるようです。暗号の解読をともなう複製行為は民事上違法になり、暗号の解読をできる装置やプログラムの譲渡などを行ったものに刑事罰（非親告罪）が規定されました。

アメリカで成立して一時期大騒ぎになった DMCA 法というのがありました。 DMCA でソフトウェア開発に影響の有りそうな部分も順次日本の著作権法に取り入れられているようです。 今回の著作権法改正では DRM を実装している暗号の解読するためのソフトウェアの作成が問題となっているようです。フリーソフトウェアでは DVD の再生に必要な鍵のライセンスが手に入らないため暗号を解読してしまう（DeCSS）を利用するなどの回避策がとられてきました。海賊行為に流用できるということで業界には睨まれていたのですが、DRM を回避し複製できるソフトウェアの配布について刑事罰が設定されたようです。

具体的には私的使用の複製に「知って DRM 回避して複製した場合」 を例外第３０条の２の例外事項に追加したようです。

著作権法の特色として、民事だけでなく親告罪でありながら刑事罰が規定されている点です。親告罪とは、被害者が告訴しないと公訴を提起できないもので、親告罪でなければ警察が適当に被疑者を捕まえることができるようです。

アメリカでは DMCA 法の影響で DeCSS が配布できないとなった時には Usenet 上で DeCSS のソースコードが SPAM で投稿されたり、DeCSS のソースコードが印刷された T シャツを着ることが流行したりしました。日本も同様の祭りは起きるんでしょうか。

libdvdread パッケージは [3]libdvdcss を利用して CSS の復号化と DVD ビデオの再生ができるのですが知って DRM の回避をしていることになると思われます。ただ、複製するために利用できますが、私的に再生するだけであれば私的使用の「複製」をしているわけではないので著作権法の範囲ではありません。オープンソースソフトウェアだけで DVD の再生をするためにはハードウェアが DVD 復号化を行うか、ソフトウェアが DeCSS をする仕組みが必要になります。オープンソースソフトウェアが DVD の複製に利用できるかもしれないということで配布したものに刑事責任が発生するかもしれない、そういう国になったみたいですね。

テレビのデジタル放送をオープンソースソフトウェアだけで視聴しようとすると friio などのハードウェア機器を活用することになると思いますが、それもまた検討が必要なのかもしれません。

14.3 ハードウェア・ソフトウェアの傾向

別に日本に限った話ではないのですが最近の気になる傾向なのでついでにまとめておきました。

14.3.1 セキュアブート

最近は公開暗号方式で保護されているプラットフォームが増えてきました。昔はゲームハードウェア、Nintendo DS などが鍵を解読しないとソフトウェアを動かせないというので有名でしたが、最近は Android 携帯が署名されたファームウェアしか起動しない、「セキュアブートに」対応している OS が EFI 経由適切に署名された OS しか起動しないという状況になっています。

iPhone などでは Jail break という方法が編み出され、セキュリティーホールをついて自由にソフトウェアを改変して利用しているようです。いつまでも簡単にセキュリティーホールが見つかるという保証はないので、難しい問題です。

14.3.2 デバイスドライバとバイナリブロブ

ハードウェアは一般にはオープンソースソフトウェア専用として開発しているものではなく、また激しい競争のある業界ではお互いにできるだけ多くの部分を秘密にしておきたい。オープンソースの OS でもデバイスドライバのパラメータやロジックなどはできるだけファームウェアとして秘密にしておきたい。下手に改変されて法令準拠できない危険な改変ができないようにしたい[*27]。そういう要求を満たすためにデバイスドライバの多くの部分をユーザ（競合他社）の理解・改変できないバイナリ形式で提供するという慣習があります。

デバイスドライバの秘密の部分を binary blob と呼び Debian では長年問題を検討してきました。Kernel から Binary Blob を分離し、問題を露見し、かつ non-free で Binary Blob を配布することでユーザに直接の不利益にならないようにしています。

個人的にはリバースエンジニアリングについて論じてた委員会の論点がおもしろいと思いました。[2] 相互運用のためのリバースエンジニアリングは著作権法の文面では著作権の範囲で保護されていないので、リバースエンジニアリングを利用者規定で禁止しているものが多いと思います。

14.3.3 マーケット

セキュアブートとの兼ね合いで重要になってくると思われる傾向ですが、Debian Project 自身もデフォルトでは Debian Archive Key で署名されたパッケージしかインストールできないように設定されています。これは Debian Project として出自がわかっている安全だと思われるファイルのみがインストールできるようにすることで誰でも間

[*27] 例えば無線通信関連

に介在できるインターネット経由でダウンロードしているという環境で安全を担保する仕組みになっています。

現在最大規模だと思われる Apple Inc のアプリケーションマーケットではアップルが規約に合致しているか審査してから配布許可を出すというシステムになっています。iOS のハードウェアでは基本的には Apple の許可したソフトウェアでないとインストールできなくなっています。規約でプログラム可能なソフトウェアというのを禁止しているので、iOS のハードウェアではプログラミングに親しむことが難しくなっています。

14.4 おわりに

自由に技術研究できる文化と環境というのは既得権益を保護したい業界、もしくは激しい競争の行われている業界で競合他社から秘密を守りたい業界の求める変化の方向とは一致しません。安全・安心・信頼できるなソフトウェアが欲しいという方向とも必ずしも一致しません。

一方に極端に有利な変更というのはそのまま通らないでしょうが、一 Debian ユーザーとして、自由なソフトウェア開発を支持する層として、今後も状況を注視していきたいと思います。

参考文献

[1] 文化庁 「平成 24 年通常国会　著作権法改正について」http://www.bunka.go.jp/chosakuken/24_houkaisei.html

[2] 文化審議会著作権分科会法制問題小委員会（第 7 回）議事録「リバース・エンジニアリングに係る法的課題についての論点」http://www.bunka.go.jp/chosakuken/singikai/housei/h20_07/shiryo_1.html

[3] libdvdread Debian package documentation “Content Scramble System (CSS)” /usr/share/doc/libdvdread4/README.css

15 月刊 Debian Policy「パッケージ管理スクリプトとインストールの手順」

かわだ てつたろう

月刊 Debian Policy、今回読むのは第 6 章の「パッケージ管理スクリプトとインストールの手順」です。この章ではパッケージをインストール、アップグレード、削除する際にパッケージ管理システムが走らせるスクリプトとその手順について説明されています。

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。

15.1 パッケージ管理スクリプト

パッケージ管理スクリプトは次の 4 つの制御情報ファイルで、パッケージの一部として供給されます。

- preinst
- postinst
- prerm
- postrm

15.1.1 ファイル構成

パッケージ管理スクリプトは実行可能ファイルでなければならず、ファイルのパーミッションは、誰でも読むことと実行可能でなければならないが誰でも書き込み可能ではいけない、755 でなければいけません。また、スクリプトであることが推奨されており、その場合は `#!` (shebang) で始まってなければなりません。

15.1.2 終了ステータス

パッケージ管理システムは終了ステータスコードを参照するので正しく終了ステータスコードを返すことが重要です。処理が成功すれば 0 を、失敗すれば 0 以外を返さなければなりません。0 以外のステータスコードが返された場合、パッケージ管理システムはエラーと判断し以降の処理を停止します。

パッケージ管理スクリプトがシェルスクリプトなら常に *set -e* を使用する必要があります。

15.1.3 PATH 環境変数

PATH 環境変数内から見つかることが期待できるプログラムは絶対パスで呼びだすべきではありません。また、PATH 環境変数はリセットすべきではありませんが、前後に付け加えることはかまいません。

15.1.4 再入結果の同一性

パッケージ管理スクリプトは、一度成功した処理を何度呼び出しても無害でなければならず、失敗や中断した処理を再度呼び出しても前回の残された状態から実行しなければなりません。いずれにせよ全ての処理が成功した場合にはパッケージ管理スクリプトは成功ステータスで終了しなければなりません。

これは、ひどく壊れたパッケージが残らないようにするためにとても重要です。

15.1.5 パッケージ管理スクリプトからのターミナルの制御

パッケージ管理スクリプトは、制御端末がある状態での実行が保証されていません。ユーザとの対話ができない場合がありますので、非対話型で処理できるようにしなければなりません。

どうしても対話が必要な場合 (優先順位が高く、妥当な標準回答がないなど) に制御端末が無ければパッケージ管理スクリプトは異常終了してもかまいません。ただ、できるだけこのような状況は避けるべきです。たいていの場合、パッケージのバグと判断されます。

`debconf` を用いる場合の詳細は「3.9.1 メンテナスクリプト中のプロンプト使用について」を参照してください。

15.2 パッケージ管理スクリプトの呼ばれ方のまとめ

パッケージ管理スクリプトがそれぞれ呼ばれるタイミングをおおまかにいうと次のようになります。

- preinst … パッケージ展開前
- postinst … パッケージ展開後
- prerm … パッケージ削除前
- postrm … パッケージ削除後

ここからは、パッケージ管理スクリプトのすべての呼ばれ方と、呼ばれる際に依存して良い機能についてのまとめです。*new-* はインストール/更新/ダウングレード対象の新バージョンのパッケージから呼ばれるもの、*old-* は更新/ダウングレードされる対象の旧バージョンのパッケージから呼ばれるものです。

15.2.1 preinst

- *new-preinst* install
- *new-preinst* install *old-version*
- *new-preinst* upgrade *old-version*
  - パッケージは展開前
  - essential パッケージと Pre-Depends パッケージのファイルのみがある
  - Pre-Depends パッケージは展開された直後か Half-Configured の可能性がある
- *old-preinst* abort-upgrade *new-version*
  - アップグレードに失敗した場合に呼ばれる
  - 展開されたファイルに依存してはいけない
  - パッケージの依存関係は満たされていないかもしれない
  - Pre-Depends は上記と同様

15.2.2 postinst

- *postinst* configuire *most-recently-configured-version*
  - パッケージと依存パッケージのファイルは展開されている
  - 巡回依存関係になければ依存パッケージは設定されている

- *old-postinst* abort-upgrade *new-version*
- *conflictor's-postinst* abort-remove in-favour *package new-version*
- *postinst* abort-remove
- *deconfigured's-postinst* abort-deconfigure in-favour *failed-install-package version* [removing *conflicting-package version*]
  - パッケージのファイルは展開されている
  - 依存する全てのパッケージは少くとも Half-Installed 状態、完全に展開されていないこともある
  - 依存関係が必要となる処理は実行するべきである、依存関係が満たされていないたいていの場合 `postinst` を異常終了させるのがエラー処理を考慮すると適切な処理である

15.2.3 prerm

- *prerm* remove
- *old-prerm* upgrade *new-version*
- *conflictor's-prerm* remove in-favour *package new-version*
- *deconfigured's-prerm* deconfigure in-favour *package-being-installed version* [removing *conflicting-package version*]
  - パッケージは Half-Installed 状態
  - 依存する全てのパッケージは少くとも Half-Installed 状態、完全に展開されていないエラー状態なこともある

- *new-prerm* failed-upgrade *old-version*
  - *prerm* upgrade が失敗した際に呼び出される
  - 新しいパッケージは展開されておらず *preinst* upgrade と同じ制約下にある

15.2.4 postrm

- *postrm* remove
- *postrm* purge
- *old-postrm* upgrade *new-version*
- *disappearer's-postrm* disappear *overwriter overwriter-version*
  - パッケージのファイルが削除、置き換えられた後に呼び出される
  - 依存関係は考慮されていない状態
  - essential パッケージのみに依存した処理とすること
  - 依存関係が必要な処理で依存関係が満たされていない場合はすべて丁寧に処理を飛ばすようにしなければいけない

- *new-postrm* failed-upgrade *old-version*
  - 古いパッケージの *postrm* upgrade が失敗した場合に呼ばれる
  - 新しいパッケージのファイルは展開されているが、essential パッケージと Pre-Depends パッケージのみへ依存できる

- *new-postrm* abort-install
- *new-postrm* abort-install *old-version*
- *new-postrm* abort-upgrade *old-version*
  - preinst が失敗したエラー処理の一環で新しいパッケージを展開する前に呼ばれる
  - preinst と同じ制約下

15.3 各段階の詳細

ここでは流れを掴みやすくするためエラー処理を省いたパッケージのインストール、設定、削除の詳細をみていきます。エラー処理について元のポリシーに記述されていますので参照してください。

15.3.1 インストール時とアップグレード時のパッケージの展開段階の詳細

1. インストールされたパッケージがある場合
   - *old-prerm* upgrade *new-version*
2. 衝突するパッケージが削除されようとしているか、壊れたパッケージ (Breaks のため) の場合
   (a) –auto-deconfigure が指定されていた場合
      i. Breaks で設定破棄されるパッケージに対して
         - *deconfigured's-prerm* deconfigure in-favour *package-being-installed version*
      ii. 削除されようとしている衝突するパッケージが依存するパッケージに対して
         - *deconfigured's-prerm* deconfigure in-favour *package-being-installed version* removing *conflicting-package version*
   (b) 衝突して削除されるパッケージに対して
      - *conflictor's-prerm* remove in-favour *package new-version*
3. (a) アップグレードの場合
      - *new-preinst* upgrade *old-version*
   (b) 設定ファイルが残っていた場合
      - *new-preinst* install *old-version*
   (c) それ以外 (purge された状態) の場合
      - *new-preinst* install
4. 新しいパッケージのファイルを展開し上書き (古いファイルはバックアップされる)
   - Replace 指定なしに既にシステムにある他のパッケージのファイルを含んでいるとエラー
   - 同じく Replace 指定なしに他のパッケージのディレクトリと同名のファイルやディレクトリ以外のものを含んでいるとエラー
     – –force-overwrite-dir でエラーを無効にできるが、勧められない
     – ディレクトリがシンボリックリンクで置き換えられることはないし、逆もまたない
5. アップグレードされた場合
   - *old-postrm* upgrade *new-version*

   ここが戻れなくなるポイント
6. 新しいパッケージで使われなくなった古いパッケージのファイルを削除
7. 新しいファイルリストに更新
8. 新しいパッケージ管理スクリプトに更新
9. 全てのファイルが上書きされ、要求も依存もされなくなったパッケージがあった場合 (削除されたとみなされ次の処理が行われる)
   (a) *disappearer's-postrm* disappear *overwriter overwriter-version*
   (b) パッケージ管理スクリプトの削除

(c) パッケージがインストールされていない状態となる。(conffile は無視され、prerm も呼び出されない)

10. 新しいパッケージのファイルが他のパッケージのファイルリストに記されていれば、それらをリストから削除
11. バックアップファイルを削除
12. 新しいパッケージを unpacked 状態にする

    もう一つの戻れなくなるポイント
13. 衝突するパッケージがあればそれらの削除処理へ

15.3.2 設定の詳細

1. conffile を更新して
   - *postinst* configure *most-recently-configured-version*

   エラーが起こっても回復処理は行なわれず Failed Config 状態になる

15.3.3 削除と設定の完全削除の詳細

1. *prerm* remove
2. パッケージのファイルを削除 (conffile を除く)
3. *postrm* remove
4. postrm を除く全パッケージ管理スクリプトの削除

   purge でなければここで終了。purge でなくても postrm と conffile がなければ purge と同じ状態になる。
5. conffile とバックアップファイルを削除
6. *postrm* purge
7. パッケージのファイルリストを削除 (postrm もここで削除)

16 月刊 Debian Policy 「オペレーティングシステム」その 1

担当：のがた

月刊 Debian Policy の出番がとうとうやってきてしまった。ということで第 9 章「オペレーティングシステム」についての解説をします。第 9 章は最新版 (3.9.4.0) と日本語訳版 (3.9.1.0) の間で大きな改変がないので、変更された点を中心に解説します。

16.1 第 9 章の内容について

第 9 章では、Debian のオペレーティングシステムとしての構成についてのポリシーが述べられています。
解説されている範囲は以下のように、幅広い範囲になります。

- ファイルシステムの階層構造
- ユーザーとグループ
- システムランレベルと init.d スクリプト
- init.d スクリプトからのコンソールメッセージ
- Cron ジョブ
- メニュー
- マルチメディアハンドラ
- キーボードの設定
- 環境変数
- doc-base を用いた文書の登録
- 代替 init システム

16.2 最新の原文 (3.9.4.0) と日本語訳版 (3.9.1.0) との違い

大きな変更点は 2 つです。

一つは新設された/run ディレクトリの扱いについて (9.1.1「ファイルシステム構造」の例外 7 と 9.1.4「/run と/run/lock」)。もう一つは、SysVInit の代替 Init システム (upstart) の扱いについて (9.11「代替 init システム」) この 2 つの記述が追加されました。

細かな変更では、GNU Hurd のディレクトリ配置についての例外 (9.1.1「ファイルシステム構造」の例外 9) と、Cron ジョブのファイル名について (9.5.1「Cron ジョブのファイル名」) が追加されました。

その他には細かな修正が入っていますが、これらの追加修正以外は日本語訳版とほぼ変わりはないので、9 章を読む場合は日本語訳を参考にしつつ原文をあたると読みやすいと思います。(筆者も Diff を参考にしつつ読みました。)

16.3 9.1 ファイルシステムの階層構造

ファイルシステムの階層構造の解説です。ここでは、インストールされるファイルやディレクトリについての扱いについて解説しています。

16.3.1 9.1.1 ファイルシステム構造

Debian のファイル [*28]やディレクトリ配置は、9 章以外で決められているポリシーと、この節で述べられる例外を除き、Filesystem Hierarchy Standard(FHS) バージョン 2.3 に従います。

1. 「ユーザー固有のアプリケーション設定ファイルをユーザーディレクトリに置く」というオプショナルルールは緩和されました。設定ファイル名は「.」(ドット) から始めることを推奨 (ドットファイル) し、複数の設定ファイルを作成する場合は、一つの「.」(ドット) から始まる名前のディレクトリを作成しその下に設定ファイルを作成します。この場合の設定ファイルは「.」から始めないことを推奨します。
2. 「amd64 の 64 ビットバイナリは/lib64 を使わなければいけない」という制限は廃止されました。
3. 「オブジェクトファイル、内部バイナリ、ライブラリ (libc.so.*を含む) は、/lib{,32} または/usr/lib{,32} 以下に置く」という制限は改正され、/lib/triplet や/usr/lib/triplet に置くことも許可されました。triplet は、インストールするパッケージのアーキテクチャで dpkg-architecture -qDEB_HOST_MULTIARCH [*29]が返す値です。パッケージは、パッケージアーキテクチャに適合しない triplet パスにファイルをインストールできません。例えば Architecture: amd64 パッケージに 32 ビット x86 ライブラリが含まれている場合、それらのライブラリを/usr/lib/i386-linux-gnu [*30] にインストールできません。アプリケーションは/usr/lib/triplet 以下に一つサブディレクトリを作成して使えます。実行時、リンカ/ローダー,ld*は、引き続き/lib または/lib64 以下の既存の場所に置き、利用できることが必須になっています。これは ELF ABI の一部であるためです。
4. /usr/local/share/man と/usr/local/man を同じとみなす」という制限は推奨に緩和されました。
5. 「ウィンドウマネージャは system.*wmrc という一つの設定ファイルを持つこと」「ウィンドウマネージャのサブディレクトリ名はウィンドウマネージャと同じにしなければいけない」という制限は撤廃されました。
6. 「ブートマネージャの設定は/etc に置く、もしくはシムリンクを張る」という制限は推奨に緩和されました。
7. (追加) ルートファイルシステムに/run ディレクトリの追加が許可されました。/var/run が/run に、/var/lock は/run/lock に置き換えられ、後方互換のため/var 以下のディレクトリはシムリンクに置き換えられました。/run と/run/lock は、FHS の/var/run、/var/lock の必要な要件のほかに、ファイル命名規則、ファイル形式の要件、ブート時にファイルが消去されるといった要件、全てに従わなければいけません。/run にあるファイルおよびディレクトリは、テンポラリファイルシステムに保存されなければいけません。
8. ルートファイルシステムに/sys と/selinux ディレクトリを置くことが許可されました。
9. (追加) GNU Hurd システムにおいて、ルートファイルシステムに/hurd と/servers ディレクトリを追加することが許可されました。

FHS は debian-policy パッケージに同梱されているほか、FHS の Web サイト[*31] で確認できます。

16.3.2 9.1.2 サイトごとのプログラム

通常、パッケージは FHS に従うため/usr/local にファイルを置いてはいけません。しかし、システム管理者にサイト固有のファイルを置く場所を示すため空ディレクトリを作成することだけは許可されています。

作成場所は/usr/local 直下ではなく一段下 (/usr/local/*/dir) に作成すること。/usr/local 直下に作成するディレ

[*28] 日本語訳版では、すべてのインストールされたファイル (all installed files) となっていますが、最新版では、すべてのファイル (all files) に改められてます。
[*29] 日本語訳版では「dpkg-architecture -qDEB_HOST_GNU_TYPE」でしたが変更されています。
[*30] 日本語訳版では「/usr/lib/i486-linux-gnu」でしたが変更されています。
[*31] http://www.pathname.com/fhs/

クトリは、FHS セクション 4.5 [*32] に書かれたもの以外作成しないこと、また、FHS セクション 4.5 で列挙されているディレクトリは削除してはいけません。パッケージを削除する際は、空であれば作成したディレクトリは削除すること。

例では emacsen-common パッケージが/usr/local/share/emacs ディレクトリを利用している例が挙げられています。

16.3.3 9.1.3 システムの使うメールディレクトリ

システムが使うメールディレクトリは/var/mail です。特定のメールエージェントだけが使ってはいけませんし、以前使われていた/var/spool/mail は、物理的にあっても使用するべきではありません。

16.3.4 9.1.4 /run と/run/lock

(追加)/run ディレクトリは、通常テンポラリファイルシステムにマウントされ、起動時には消去されます。

Packages therefore must not assume that any files or directories under /run other than /run/lock exist unless the package has arranged to create those files or directories since the last reboot. Normally, this is done by the package via an init script. See Writing the scripts, Section 9.3.2 for more information.

(すみません。/run、/run/lock は init スクリプトがテンポラリ上に作成するので、パッケージはその下にファイルが存在すると仮定してはいけないという意味だと思うのですが、うまく訳せませんでした…。)

パッケージには、/run または/var/run、/var/lock のパスにあるファイルやディレクトリを含めることはできません。/var/run、/var/lock のパスは通常シムリンクか、/run の後方互換性のためリダイレクトされます。

16.4 9.2 ユーザーとグループ

16.4.1 9.2.1 はじめに

Debian では平文パスワードもしくはシャドウパスワードの設定ができます。

一部のユーザー ID(UID) とグループ ID(GID) は、特定のパッケージのためグローバルに予約されています。いくつかのパッケージでは、ユーザーやグループ所有のファイルを含めたり、バイナリコンパイル時に使う必要があるので、Debian システムではこれらの目的のための使用をします。これは重大な制限でローカルの管理ポリシーとぶつからないようにしてください。多くのサイトではローカルのユーザー・グループを割り当てている事が多いので注意してください。

これとは別に動的に割り当てられる ID があります。デフォルトでは適切な順序で割り当てられますが、設定で変更できます。

base-passwd 以外のパッケージは/etc/passwd, /etc/shadow, /etc/group, /etc/gshadow を変更してはいけません。

16.4.2 9.2.2 UID と GID の割り当て

UID と GID の割り当てです。

0-99　Debian プロジェクトが使い Debian システム共通に割り当てられます。
100-999　システム用に動的に割り当てられます。
1000-59999　ユーザーアカウントが動的に割り当てられます。
60000-64999　Debian プロジェクトが共通に割り当てますが、必要に応じて作成されます。
65000-65533　予約済み
65534　nobody ユーザー。対応する gid として nogroup グループを割り当てます。
65535　(uid_t)(-1) == (gid_t)(-1) は利用しないでください。エラーの戻り値として利用します。

[*32] FHS セクション 4.5 を調べるとなかったのだけど、FHS の章立てが狂ってるような気がする。http://www.debian.org/doc/packaging-manuals/fhs/fhs-2.3.html#USRLOCALLOCALHIERARCHY

17 月刊 Debhelper dh_auto_install dh_install

吉田 俊輔

17.1 今月のコマンド

- dh_auto_install
- dh_install

17.2 debian パッケージ構築、全体の流れ

2011 年 10 月勉強会資料より

1. パッケージビルド環境を構築する
2. 不要なファイルを削除する
3. バイナリパッケージに格納するファイルをビルドする
4. ビルドしたファイルをバイナリパッケージにまとめる
5. .changes ファイルを作成する
6. パッケージに署名する

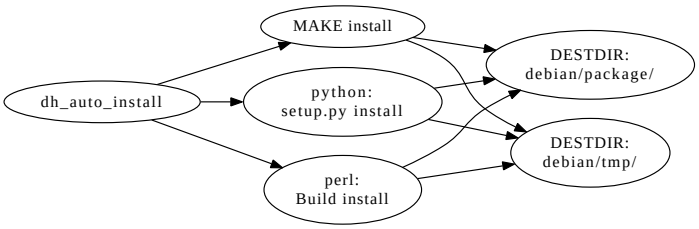
dh_auto_install,dh_install は [4. ビルドしたファイルをバイナリパッケージにまとめる] の中で実行されます。この部分では以下の順で debhelper コマンドが実行されます。

```
dh_testdir -> dh_auto_configure -> dh_auto_build -> dh_auto_test
-> dh_testroot -> dh_prep -> dh_installdirs -> dh_auto_install
-> dh_install -> dh_installdocs -> dh_installchangelogs
-> dh_installexamples -> dh_installman -> dh_installcatalogs
-> dh_installcron -> dh_installdebconf -> dh_installemacsen
-> dh_installifupdown -> dh_installinfo -> dh_pysupport
-> dh_installinit -> dh_installmenu -> dh_installmime
-> dh_installmodules -> dh_installlogcheck -> dh_installlogrotate
-> dh_installpam -> dh_installppp -> dh_installudev -> dh_installwm
-> dh_installxfonts -> dh_installgsettings -> dh_bugfiles -> dh_ucf
-> 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
```

17.3 dh_auto_install 動作解説

dh_auto_install は debhelper のプログラムです。自動的にファイルをパッケージ作成用のディレクトリにインストールします。dh_auto_install は upstearm 等の Makefile の install ターゲット,setup.py, Build.PL を使用します。インストール先はシングルバイナリ (only one binary package) であれば、debian/package/以下です。

multiple binary package の場合は debian/tmp/以下になり、その後 dh_install で適切なディレクトリに移動されます。



- 条件 1:Makefile(または setup.py や Build.PL) が GNU の慣例に準拠し、$(DESTDIR) 変数をサポートしていること。

`http://www.gnu.org/prep/standards/html_node/DESTDIR.html#DESTDIR`
Makefile ファイルを変更する必要があるなら、これら $(DESTDIR) 変数をサポートするように注意しましょう。

- 条件 2:インストール先の指定内容が Filesystem Hierarchy Standard (FHS) に準拠していること。

`http://www.debian.or.jp/community/devel/debian-policy-ja/policy.ja.html/ch-opersys.html`
通常プログラムのビルドに使われている make 等を使って実際のインストール先のかわりに、一時ディレクトリーの下に作成されたファイルツリーのイメージへプログラムをインストール (コピー) する。普通のプログラムインストールと Debian パッケージ作成というこれら二つの違いには、debhelper パッケージの dh_auto_configure と dh_auto_install のコマンドを使うことで (前述の条件を守っていれば、特に意識をせずに) 対応できるはずです。GNU autoconf を使っているプログラムは、自動的に GNU 規約に準拠するので、そのパッケージ作成は簡単にできる (はず)。`http://www.debian.org/doc/manuals/maint-guide/modify.ja.html`

17.4 override 例

マルチパッケージで、DESTDIR を使っていないので override で対応する例 (wide-dhcpv6)

```
override_dh_auto_install:
	$(MAKE) prefix=$(CURDIR)/debian/tmp/usr install
```

この後、dh_install で各パッケージに振り分け (後述)

17.5 dh_install 動作概要

dh_install はパッケージ構造ディレクトリーへインストールするファイルを扱う debhelper プログラムです。これ以外に多くの dh_install*コマンドが存在します。説明書 (documentation), サンプル (examples), マニュアルページ (man pages) のような特定のタイプのファイルのインストールにはするにはそれら専用のプログラムの方が向いています。
dh_install には二つの使用方法があります。

- upstearm の Makefile がインストールを行ってくれないとき、適所へそれらをコピーさせるために使用する。
- 複数のバイナリパッケージを構築するラージ・パッケージを構築するとき

17.6 dh_install*コマンド (debhelper 内)

17.7 ラージ・パッケージの構築

(dh_auto_install または override_dh_auto_install 等を使用して)debian/tmp へすべてインストール。そこから適切なパッケージディレクトリーを構築するためにディレクトリーやファイルを dh_install を使用してコピーすることができます。

コマンド	行数
dh_installcatalogs - install and register SGML Catalogs	126
dh_installchangelogs - install changelogs into package build directories	181
dh_installcron - install cron scripts into etc/cron.*	87
dh_installdeb - install files into the DEBIAN directory	118
dh_installdebconf - install files used by debconf in package build directories	136
dh_installdirs - create subdirectories in package build directories	96
dh_installdocs - install documentation into package build directories	311
dh_installemacsen - register an Emacs add on package	134
dh_installexamples - install example files into package build directories	116
dh_installifupdown - install if-up and if-down hooks	79
dh_installinfo - install info files	87
dh_installinit - install init scripts and/or upstart jobs into package build directories	287
dh_installlogcheck - install logcheck rulefiles into etc/logcheck/	76
dh_installlogrotate - install logrotate config files	60
dh_installman - install man pages into package build directories	268
dh_installmanpages - old-style man page installer (deprecated)	207
dh_installmenu - install Debian menu files into package build directories	99
dh_installmime - install mime files into package build directories	105
dh_installmodules - register modules with modutils	134
dh_installpam - install pam support files	69
dh_installppp - install ppp ip-up and ip-down files	75
dh_installtex - register Type 1 fonts, hyphenation patterns, or formats with TeX	664
dh_installudev - install udev rules files	125
dh_installwm - register a window manager	118
dh_installxfonts - register X fonts	97

debhelper 互換性レベル 7 から、カレント・ディレクトリ (あるいは–sourcedir オプションで指定したディレクトリ) に対象が無ければ、dh_install は debian/tmp をコピー元に使用します。debian/package.install に各パッケージへインストールするべきファイル、およびそれらがインストールされるべきディレクトリーを記載します。フォーマットは、行単位でインストールするべきファイル (複数可) をリストし、行の末尾にそれがインストールされるべきディレクトリーを記載します。
要するに cp コマンドの引数です。
実際に dh_install 内部では cp コマンドが使用されています。

17.8 debian/package.install の例

```
$ cat wide-dhcpv6-client.install
usr/sbin/dhcp6c
usr/sbin/dhcp6ctl
debian/dhcp6c.conf etc/wide-dhcpv6
debian/scripts/dhcp6c-script etc/wide-dhcpv6
debian/scripts/dhcp6c-ifupdown etc/wide-dhcpv6
```

明示的なコピー先なしで、一行に 1 つのファイル名あるいはワイルドカード・パターンを単独で記載すると、dh_install が自動的に使用する目的地を推測します。これは–autodest オプションの動作と同様。

17.9 –autodest オプション

コピー先ディレクトリを推測する。これを指定する場合、debian/package.install ファイルのコピー先ディレクトリは指定しないこと。debian/tmp のディレクトリ配下にあるファイルを debian/tmp を除いて指定ディレクトリの下に対応するようにコピーする。
例

```
$ cat debian/package.install
debian/tmp/usr/bin
debian/tmp/etc/passwd
```

- debian/tmp/usr/bin を debian/package/usr/へコピー
- debian/tmp/etc/passwd を debian/package/etc/へコピー

17.10 –list-misqqsing オプション

ファイル (およびシンボリックリンク) がどのディレクトリにもコピーされなかったときに標準エラー出力に警告を表示する。
ラージ・パッケージに、新しく追加されたファイルを見逃さないためなどに使える。

17.11 -Xitem, –exclude=item オプション

指定したファイル名を含むファイルをコピー対象外とする。

17.12 DEBHELPER のプログラム共通のオプション

アーキティクチャ独立

-i, –indep	Act on all architecture independent packages.

例

```
$ dh_make -m --rulesformat old
```

```
install-indep:
        (中略)
        dh_prep -i
        dh_installdirs -i
        (中略)
        dh_install -i
        (後略)
```

アーキティクチャ依存

-s, –same-arch	This used to be a smarter version of the -a flag, but the -a flag is now equally smart.
-a, –arch	Act on architecture dependent packages that should be built for the build architecture.

例 (同上)

```
install-arch:
        (中略)
        dh_prep -s
        dh_installdirs -s

        # Add here commands to install the arch part
        # of the package into debian/tmp.
        $(MAKE) DESTDIR=$(CURDIR)/debian/hello install

        dh_install -s
        (後略)
```

DEBHELPER のプログラム共有のオプションその他

オプション	動作
-v, –verbose	詳しく動作を表示 (Verbose mode: show all commands that modify the package build directory.)
–no-act	実際の動作をしない (Do not really do anything. If used with -v, the result is that the command will output what it would have done.)
-ppackage, –package=package	Act on the package named package. This option may be specified multiple times to make debhelper operate on a given set of packages.
-Npackage, –no-package=package	Do not act on the specified package even if an -a, -i, or -p option lists the package as one that should be acted on.
–remaining-packages	Do not act on the packages which have already been acted on by this debhelper command earlier (i.e. if the command is present in the package debhelper log). For example, if you need to call the command with special options only for a couple of binary packages, pass this option to the last call of the command to process the rest of packages with default settings.
–ignore=file	Ignore the specified file. This can be used if debian/ contains a debhelper config file that a debhelper command should not act on. Note that debian/compat, debian/control, and debian/changelog can't be ignored, but then, there should never be a reason to ignore those files. For example, if upstream ships a debian/init that you don't want dh_installinit to install, use –ignore=debian/init
-Ptmpdir, –tmpdir=tmpdir	Use tmpdir for package build directory. The default is debian/package
–mainpackage=package	This little-used option changes the package which debhelper considers the "main package", that is, the first one listed in debian/control, and the one for which debian/foo files can be used instead of the usual debian/package.foo files.
-O=option—bundle	This is used by dh(1) when passing user-specified options to all the commands it runs. If the command supports the specified option or option bundle, it will take effect. If the command does not support the option (or any part of an option bundle), it will be ignored.

18 月刊 Debhelper dh_gencontrol dh_listpackages

野島 貴英

18.1 今回のコマンド

以下のコマンドを今回は取り上げます。

- dh_gencontrol
- dh_listpackages

18.2 dh_gencontrol

dh_gencontrol コマンドは、dh コマンドなどから提供される情報を引き継ぎ、dpkg-gencontrol コマンドを呼び出して、DEBIAN/control ファイルと、debian/files ファイルを生成します[*33]。

次に、dh_gencontrol コマンドが取り扱うファイルの説明を記載します。

18.2.1 control ファイル

control ファイルは、debian パッケージシステムでは極めて重要な役割を持ちます。

control ファイルは２つの用途があり、１つはソースパッケージ用の debian/control ファイルと、もう１つは、バイナリパッケージ用の DEBIAN/control ファイルとがあります。このどちらのファイルについても、Debian Policy Manual[1] や、man deb-control に詳しい説明があります。

ちなみに、ソースパッケージ用の debian/control ファイルは、バイナリパッケージを構築するときにどんなバイナリパッケージが必要か、あるいは、どんな名前のバイナリパッケージを生成するかが記載されています。

さらに、ソースパッケージから生成されたバイナリパッケージには、DEBIAN/control ファイルが含まれています。このファイルは、該当のパッケージの動作にあたって必要な他のバイナリパッケージがバージョン情報と共に列挙されています。また、インストールしようとすると、他のバイナリパッケージの内容物に影響をあたえてしまう場合にも、こちらを防ぐための情報などが記載されています。

実際のソースパッケージ用の debian/control ファイルは、”apt-get source パッケージ名” で取得したソースパッケージの展開された構築用ディレクトリ以下で見る事ができます。

[*33] 本質的な動作ではないので、本文中には記載しませんが、dh コマンドが処理を再開できるように debian/パッケージ名.debhelper.log に完了記録も残します

```
$ apt-get source xgalaga
パッケージリストを読み込んでいます... 完了
(... 中略...)
dpkg-source: info: applying 0003-obsolete-xf86dga.patch
$ lv xgalaga-2.1.1.0/debian/control
Source: xgalaga
Section: games
Priority: optional
Build-Depends: autotools-dev,
               debhelper (>= 5),
               dpkg-dev (>= 1.9.0),
...debian/control の各行が続く...
```

一方バイナリパッケージ用の DEBIAN/control ファイルは、バイナリパッケージを入手し、”dpkg-deb -e バイナリパッケージファイル” とすることでカレントディレクトリ以下に取り出すことが出来ます。

```
$ apt-get download xgalaga
取得:1 xgalaga 2.1.1.0-4 をダウンロードしています [285 kB]
285 kB を 3 秒 で取得しました (84.9 kB/s)
$ dpkg-deb -e ./xgalaga_2.1.1.0-4_amd64.deb
$ lv DEBIAN/control
Package: xgalaga
Version: 2.1.1.0-4
Architecture: amd64
Maintainer: Debian Games Team <pkg-games-devel@lists.alioth.debian.org>
Installed-Size: 690
Depends: libc6 (>= 2.7), libx11-6, libxext6, libxmu6, libxpm4, libxt6, libxxf86vm1
Section: games
...DEBIAN/control の各行が続く...
```

また、Debian システムへバイナリパッケージをインストールすると、DEBIAN/control ファイルの中身は /var/lib/dpkg/以下の status ファイルなどに追記されていきます。こちらは apt-get/aptitude/dpkg 等のパッケージ管理コマンドにとって、導入済みパッケージに関するデータベース等として後々利用されていきます。

18.2.2 debian/files

debian/files ファイルは、ソースパッケージから生成したバイナリパッケージの名前、セクション名、重要度が記録されているファイルとなります。

このファイルは後に、バイナリパッケージのアップロードのための情報である.changes ファイルを生成する際に利用されるファイルとなります。

Debian Policy Manual[1] に詳しい説明があります。

18.2.3 debian/substvars

debian/substvars ファイルは、debian/control ファイルのバイナリパッケージ用定義に含まれる`${shlibs:Depends}`マクロ、`${misc:Depends}`マクロ等を、実際の内容に置換するための情報が格納されているファイルとなります。

こちらのファイルは、バイナリパッケージ構築中にて、他の debhelper コマンド（例:dh_shlibdeps コマンド等) により、順次 debian/substvars ファイルへ置換すべき情報が追記されていきます。

詳しい内容については、Debian Policy Manual[1] や、man deb-substvars に説明があります。

```
debian/substvars の中身の例：
shlibs:Depends=libc6-amd64 (>= 2.3.2)
misc:Depends=
```

18.2.4 debian/changelog

debian/changelog ファイルは、Debian パッケージのバージョンに対する変更点の説明、変更に伴い close したバグの番号の情報、変更者の名前とメールアドレス、日時が記録されたファイルとなります。

詳しい説明については、Debian Policy Manual[1] にあります。

18.2.5 dh_gencontrol の動作詳細

以下に dh_gencontrol が呼び出す dpkg-gencontrol も含んだ動作詳細を記載します。

Step 1. dh コマンドから引き継がれた情報（環境変数等）を元に、パッケージの構築用ディレクトリ名、de-

bian/substvars の正確なファイル名、debian/changelog の正確なファイル名を得ます。

Step 2. debian/changelog から、ソースパッケージ及びバイナリパッケージの、バージョン情報を得ます。

Step 3. debian/substvars から置換する予定の値（主に他パッケージに対する依存情報) を取り出します。

Step 4. debian/control から構築予定のバイナリパッケージに関する情報を集めます。

Step 5. debian/substvars から得た他パッケージに関する依存情報をまとめ、

- 依存情報が完全に重複しているもの、
- 同じ名前のパッケージに依存しているのに、バージョン違いで複数指定されているようなもの

を適宜まとめて、必要最小限の依存情報に修正します。

Step 6. バイナリパッケージ構築ディレクトリから、インストールするデータの総容量を計測します。

Step 7. debian/files を作成します。

Step 8. DEBIAN/control を Step.2〜Step.6 で得た情報を用いて生成します。

18.3 dh_listpackages

dh_listpackages コマンドは、debian/control ファイルを参照して、構築予定となるバイナリパッケージの名前の一覧を得ます。

以下に実際に実行してみた結果を載せます。

```
$ apt-get source dpkg
$ cd dpkg-1.16.9
$ dh_listpackages
libdpkg-dev
dpkg
dpkg-dev
libdpkg-perl
dselect
$
```

参考までに、上の例について、debian/control ファイル中で”Package: “というフィールドを行頭に持つような行をすべて検索してみます。

```
$ egrep '^Package' debian/control
Package: libdpkg-dev
Package: dpkg
Package: dpkg-dev
Package: libdpkg-perl
Package: dselect
$
```

となり、先の結果と一致します。

なお、dh_listpackages は、dh コマンドから呼び出された場合には、dh コマンドから引きついた情報を元に、構築予定のバイナリパッケージの一覧を得る事ができます。つまり、dh コマンドから呼び出される他の debhelper コマンドが、どんなパッケージに対して作用する予定かを、dh_listpackages を使って知ることができます。

18.4 おわりに

今回は control ファイルに関わる debhelper コマンドについて説明をしてみました。

参考文献

[1] The Debian Policy Mailing List,“Debian Policy Manual”,version 3.9.4.0, 2012-09-19

19 東京エリア Debian 勉強会資料の準備の方法

上川 純一

19.1 文章ルール

文章は敬体に統一しましょう。

固有名詞は基本としては敬称略、フルネーム、で記述しましょう。日本名称の場合、苗字と名前の間には半角の空白を一文字入れます。

19.2 レポジトリの取得

まず最初に git のレポジトリを取得します[*34] 。読み込み専用であれば、git プロトコル、もしくは、http プロトコルでよいでしょう。書き込み権限を持っているのであれば、ssh プロトコルを利用すれば直接 git push でアクセスすることができます。

```
git clone git://anonscm.debian.org/tokyodebian/monthly-report.git
git clone ssh://git.debian.org/git/tokyodebian/monthly-report.git
```

この結果、カレントディレクトリに monthly-report というディレクトリができます。monthly-report/.git 以下がレポジトリです。

```
$ ls -la monthly-report/ |head
total 179440
drwxr-xr-x 123 dancer dancer   159744 11 月 29 17:09 .
drwxr-xr-x   6 dancer dancer     4096 12 月 14  2009 ..
drwxr-xr-x   8 dancer dancer     4096 11 月 29 17:09 .git
-rw-r--r--   1 dancer dancer      273  6 月 11 07:18 .gitignore
-rw-r--r--   1 dancer dancer      109  7 月 21  2007 .whizzytexrc
-rw-r--r--   1 dancer dancer      302  1 月 14  2012 .yatexrc
-rw-r--r--   1 dancer dancer    17989  7 月  6  2007 COPYING
-rw-r--r--   1 dancer dancer    25168  7 月  6  2007 ChangeLog
-rw-r--r--   1 dancer dancer   101740 11 月 27  2008 EUC-UCS2
```

19.3 コミットの方法

まず、PDF ファイルが生成できることを確認します。Makefile があるので、make コマンドを入力するとビルドしてくれるはずです。文字コードが正しいか、正常にビルドできるか、などのチェックが組み込まれているので、チェックに活用しましょう。

```
make
```

その後、git diff でコミットされる内容を確認します。意図している内容が表示され、問題ないようであれば、git

[*34] git の使いかた詳細については、2007 年 4 月の勉強会資料を参照してください。 apt-get install git-core でインストールできます。

commit コマンドでコミットします。手元のレポジトリに反映されます。

```
git diff
git commit -a -m 'revised XXX'
```

問題がないようであれば、git pull / git push でマージします。git-pull した後にコンフリクトが発生したら、修正し、git commit でコミットしてから git push します。

```
git pull
git push
```

新規のファイルを追加する場合、ファイルを削除する場合には、 git add / git rm コマンドを利用します。

19.4 ファイルの編集

ドキュメントは pLaTeX で作成しています。ファイル名として下記になっています。(YYYY)(MM) は、年と月で、例えば 2012 年 12 月であれば 201212 です。

debianmeetingresume(YYYY)(MM).tex　事前配布資料
debianmeetingresume(YYYY)(MM)-presentation.tex　プレゼンテーション用 (prosper を利用)
image(YYYY)(MM)/　画像ファイルなどの置き場

作業する前にビルドに必要なパッケージをインストールします。

```
# tex から PDF の生成関連
apt-get install texlive-lang-cjk dvipdfmx latex-beamer \
 ghostscript xpdf texlive-latex-extra
```

*35

編集に便利なツールもついでにインストールしてみてもよいでしょう。

```
# apt-get install whizzytex advi emacs21 yatex gs-cjk-resource gv \
  evince poppler-data fonts-japanese-mincho fonts-japanese-gothic
```

tex4ht を利用して HTML 出力をさせる場合は下記もインストールしたらよいでしょう。ただし、2007 年 8 月現在、dvi2ps-fontdata-a2n の影響で dvi 出力ができなくなる副作用があります。

```
# tex4ht での HTML 生成関連
apt-get install dvi2ps-fontdata-a2n dvi2dvi dvipng tex4ht
```

文字コードは iso-2022-jp で統一しています[*36]。たとえば、emacs + yatex を使用している場合で iso-2022-jp をデフォルトにするには、下記のような設定を.emacs にかけばよいでしょう。

```
(add-hook 'yatex-mode-hook
          '(lambda ()
             (progn
               (if (string-match "^/home/user/tokyodebian/" default-directory)
                   (progn (set-buffer-file-coding-system 'iso-2022-jp)
                          (set-buffer-modified-p nil))))))
```

emacs での編集で、outline-mode を利用すると、アウトラインをベースに編集することができ、便利です。tex ファイルの最後に以下のようなエントリーを追加しています。M-x outline-minor-mode で有効にできます。

```
;;; Local Variables: ***
;;; outline-regexp: "\\([ <タブ記号>]*\\(\\(documentstyle\\|documentclass\\|<改行しない>
dancersection\\)\\*?[ <タブ記号>]*[[{]\\|[%<^L>]+\\)" ***
;;; End: ***
```

*35 wheezy だと ptex-bin は存在しない
*36 Windows 版と Linux 版の ptex で共通して扱える文字コードにしたという経緯があります。ただし現状 Windows で全部できる状況ではありません。

- **<タブ記号>**: タブを入力、
- <^L>: ctrl-L を入力、
- **<改行しない>**: ここの改行はみやすいように改行をいれているだけで、実際には改行は入力しない。

また、自動で適切な設定で outline-minor-mode に入るように .emacs に設定してもよいでしょう。

```
(add-hook
 'yatex-mode-hook
 '(lambda ()
    (make-variable-buffer-local 'outline-regexp)
    (setq outline-regexp
          "\\([ \t]*\\\\\\(documentstyle\\|documentclass\\|chapter\\|dancersection\\|
section\\|subsection\\|subsubsection\\|paragraph\\)\\*?[ \t]*[[{]\\|[%\f]+\\)")
    (setq
     outline-level
     (function
      (lambda ()
        (save-excursion
                  (looking-at outline-regexp)
                  (cond
                   ((equal (char-after (match-beginning 0)) 37) (- (match-end 0) (match-beginning 0)))
                   (t (let ((bs (buffer-substring (match-beginning 2) (match-end 2))))
                        (cond ((equal (substring bs 0 2) "do") 15)
                              ((equal (substring bs 0 1) "c") 0)
                              ((equal (substring bs 0 1) "p") 4)
                              ((equal (substring bs 0 2) "da") 1) ; dancersection
                              ((equal (substring bs 0 2) "se") 1) ;section
                              ((equal (substring bs 0 5) "subse") 2) ;subsection
                              ((equal (substring bs 0 8) "subsubse") 3) ;subsubsection
                              (t (length bs)))))))))
    (outline-minor-mode t)))
```

19.4.1 ドキュメントのスタイル

スタイルファイルは monthlyreport.sty パッケージを利用します。過去の資料を参考にしてください。

```
\usepackage{monthlyreport}
```

各担当部分は section として扱います。特別なコマンド dancersection で指定します。形式は `dancersecion{` **タイトル** `}{` **作者名** `}` です。その中で subsection や subsubsection を利用して文書を構成してください。

```
\dancersection{Debian 勉強会資料の準備の方法}{上川 純一}
\label{sec:debmtg2007howtoprepare}
```

19.4.2 目次の処理

目次のエントリは下記の形式で作成します。

```
index { alphabet もしくは、 ひらがなの読み @ 項目名称 }
```

19.4.3 画像ファイルの処理

画面写真の画像を追加するときは、できるだけサイズの小さい png などを利用してください。グラフなどの線画であれば、eps でかまいません。png であれば、ebb コマンドを利用して bounding box を作成してください。

```
$ ebb XXX.png
```

ps であれば、 eps2eps でバウンディングボックスを追加してあげるとうまくいきます。inkscape の出力する ps を eps2eps で処理すれば inkscape で画像を作成することができます。

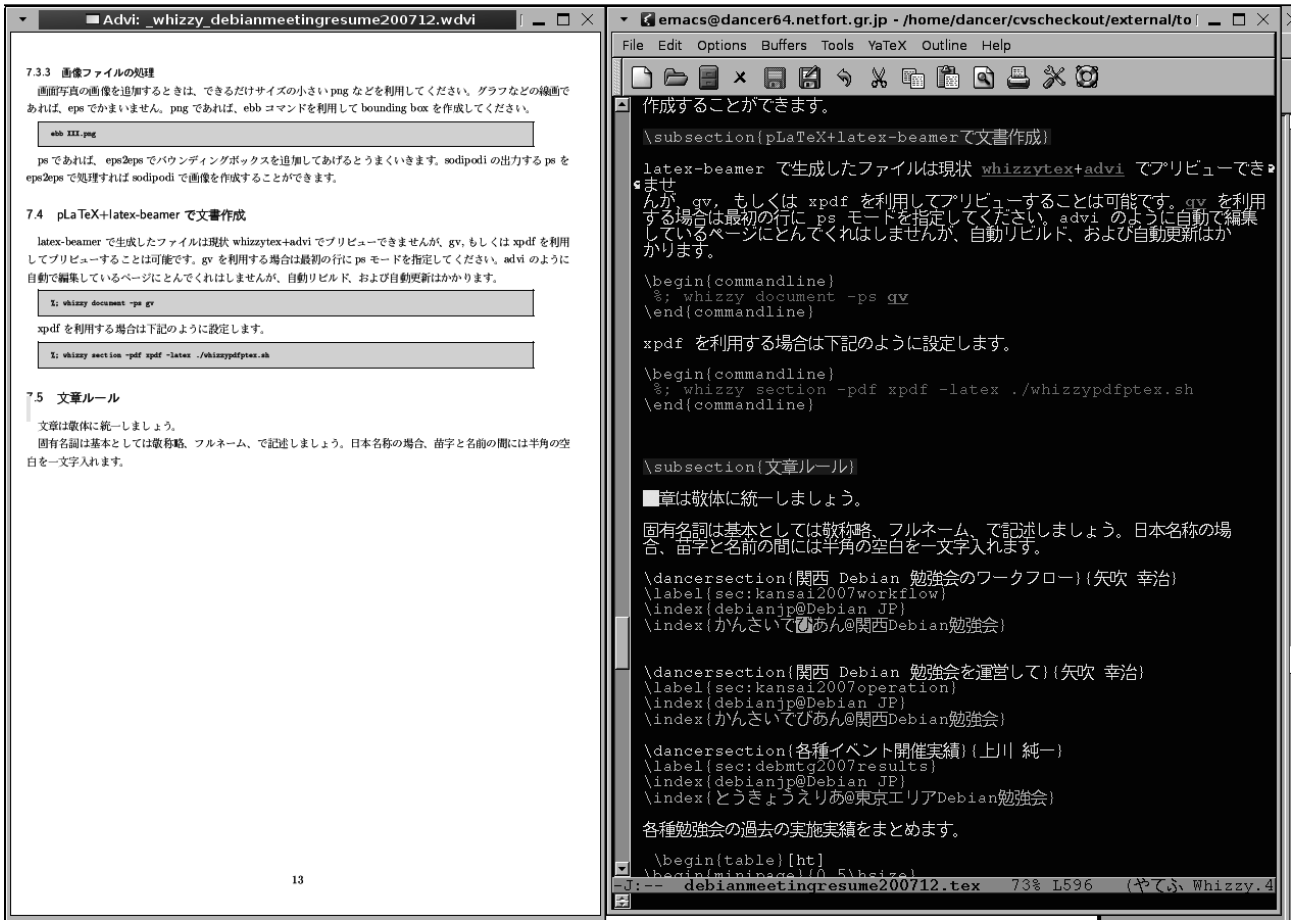
19.5 pLaTeX+latex-beamer で文書作成

latex-beamer で生成したファイルは現状 whizzytex+advi でプリビューできませんが、gv, もしくは xpdf を利用してプリビューすることは可能です。gv を利用する場合は最初の行に ps モードを指定してください。advi のように自動で編集しているページにとんでくれはしませんが、自動リビルド、および自動更新はかかります。

```
%; whizzy document -ps gv
```

xpdf を利用する場合は下記のように設定します。

```
%; whizzy section -pdf xpdf -latex ./whizzypdfptex.sh
```



20 Debian 勉強会予約システム変更履歴

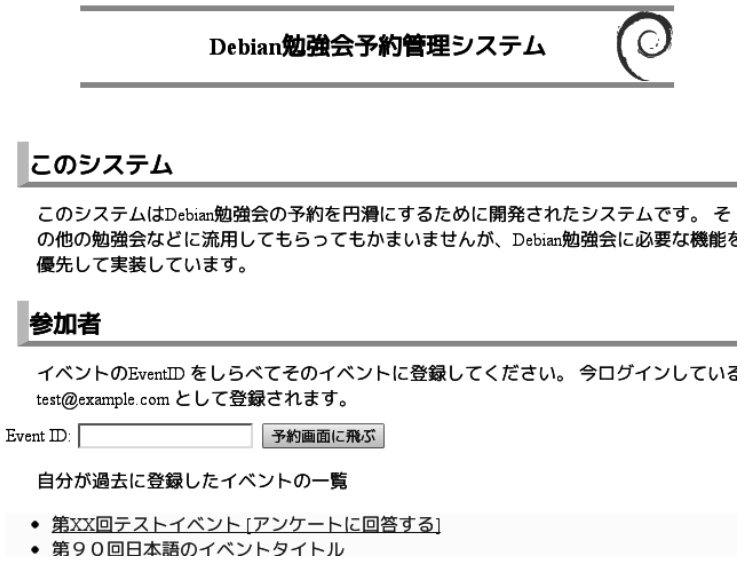
上川純一

Debian 勉強会の予約システムの機能をいくつか変更したので報告します。

20.1 アンケートのリマインダ

アンケートを送付していますが、現状メールで通知しているのみで、メールからしかアンケート回答できません。勉強会に登録している場合にアンケートを依頼してもメールがとどいていないとかメールに気づいていないというフィードバックをもらいました。

しかしながらメール通知以外で勉強会のあとにウェブページに来てもらうというのは難しいと思うのですが、ものは試しという事でトップページにリンクを表示するようにして見ました。参加しているイベントでアンケートの質問が作成されていてアンケートの回答がまだなされていない場合にアンケートのリンクを表示します。



20.2 勉強会に登録した時刻

SVG でなにができるのか試してみる勉強のついでに勉強会幹事用のページにグラフを出すようにして見ました。時刻を１０に分割してそれぞれの時刻においての登録の頻度の推移をみれるようにしています。このグラフを見ることでどのイベントでいつごろ何人登録したのかがわかります。試しに最近の二回を見てみたのですがけっこう違いますね。

HTML5 において SVG の画像をうめこむのは結構簡単で、そのまま SVG タグを埋め込めばいいだけです。inkscape の吐く SVG をみて手書きは無理かなと思っていたのですが基本的な記法だけであれば手書きするのも悪くはないと思える書式でした。

```
<svg width="800" height="120">
         <text x="0" y="120">2012-11-03</text>
         <text x="400" y="120">2012-11-17</text>
         <text x="0" y="100">0</text>
         <text x="0" y="16">6</text>
         <polyline points="
0,83.3333333333
50,66.6666666667
100,83.3333333333
150,100.0
200,100.0
250,100.0
300,100.0
350,100.0
400,0.0
450,33.3333333333 "
         fill="none" stroke="#333">
         </polyline>
   </svg>
```

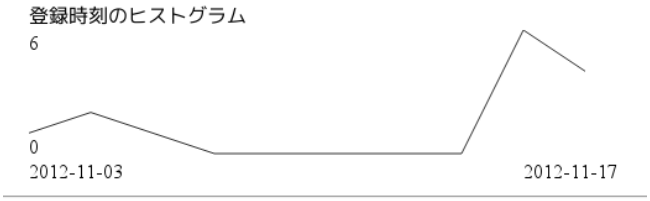


図 10　2012 年 11 月の勉強会の登録時刻

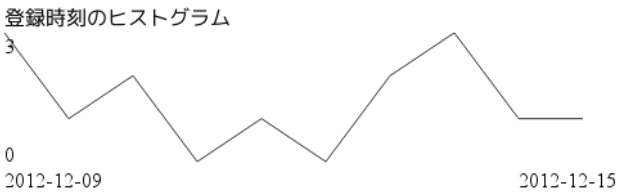


図 11　2012 年 12 月の勉強会の登録時刻

20.3　コミット数

気になったのでコミット数のグラフを作成してみました。四半期ごとにプロットしています。年末に集中してコミットしているのが見て取れます。しばらくいじってない気がしていましたが年に一回くらいは頑張っていじってる時期があるみたいですね。一年を振り返るついでにいろいろ弄りたくなるのかもしれません。

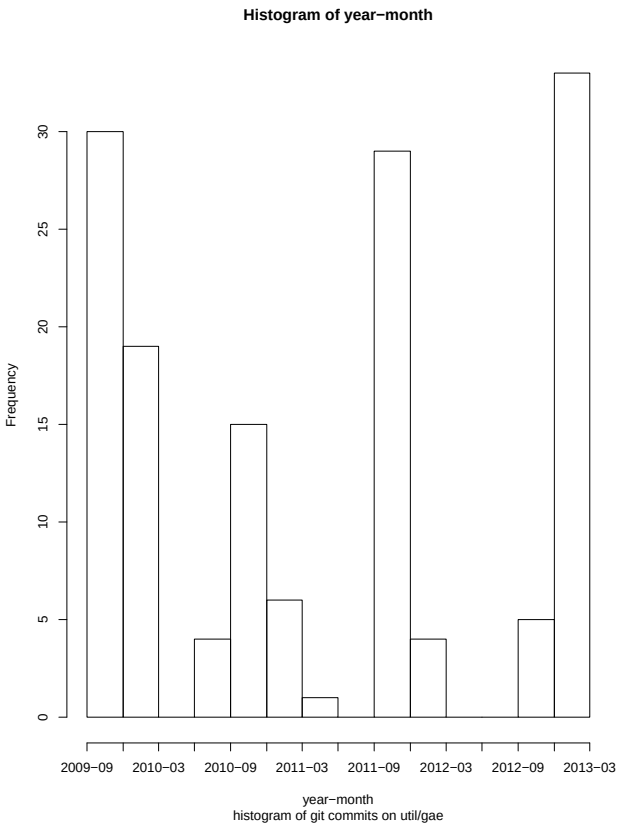


図 12　各四半期毎における util/gae ディレクトリの git コミット数

21 Debian 勉強会予約システムアンケート集計

上川 純一

21.1 アンケート集計結果の処理

東京エリア Debian 勉強会ではアンケートをとっています。集計結果を眺めてみましょう。データは Debian 勉強会予約システムのアンケート出力インタフェース `http://debianmeeting.appspot.com/enquete/showallresults` から取得します[*37]出力形式は各行が Debian 勉強会予約システムに登録している個人で、各列がそれぞれのセッションです。各コラム名は一意なキーとして扱えるようにするためイベント名とセッション名を足してかつイベントのハッシュ値の一部を追加したものを採用しています。

R で処理するためにデータを読み込み前処理を行うのには便利なスクリプトを用意しているのでそれを利用します。

```
> source('getenquete.R')
```

まず全体的なスコアのつき方から紹介しましょう。図 13 にすべてのお題のスコアの分布を並べています。時系列にならんではいますがそれぞれのテーマごとに並んでいるので時系列とは限りません。

図 14 に平均点がどういう分布をしているのかを図示しています。八割くらいは 4 点になり、一割づつ 5 点と 3 点があるようなスコア分布であることが見て取れます。特にひどいときには平均点が 2 点になってるような気もします。ここから読み取れる全体的な傾向としては、ほとんどの場合は平均点で、特によいときには良い点数、特に悪い時には悪い点数をつけている人が多いということでしょうか。

データの癖を確認するために代表的なユーザの例を見てみます。図 15 図 16 にどのスコアを何度つけたのかをグラフにしています。ほとんどの場合は 4 をつけて、一部に 5、 3 をつけ、 2 と 1 はほとんどつけてないという結果になっています。これに数回しか評価に参加していないユーザが 4 をつけるなどのバイアスが加わって全体としては 4 がさらに強調されているような気もします。

気になるのでセッションの具体例をみてみましょう。2010 年 12 月の勉強会の Debian Miniconf 企画会議はぐだぐだだったので仕方がないでしょう。一方、2012 年 1 月の事前課題紹介に辛口の評価が多かったのが気になります。n=4 なのでそんなに少数派でもないようです。今から振り返ってもいまいちぱっとしない内容なのですが運営がまずかったのでしょうか。

[*37] 取得したファイルを `image201301/enquete.csv` においておきます。

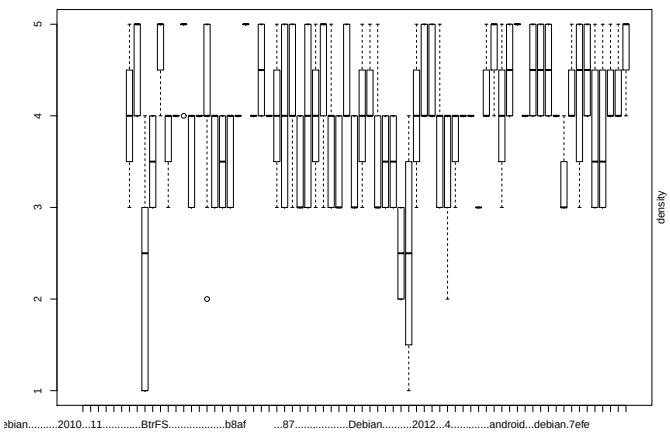


図 13　毎回のスコアの分布

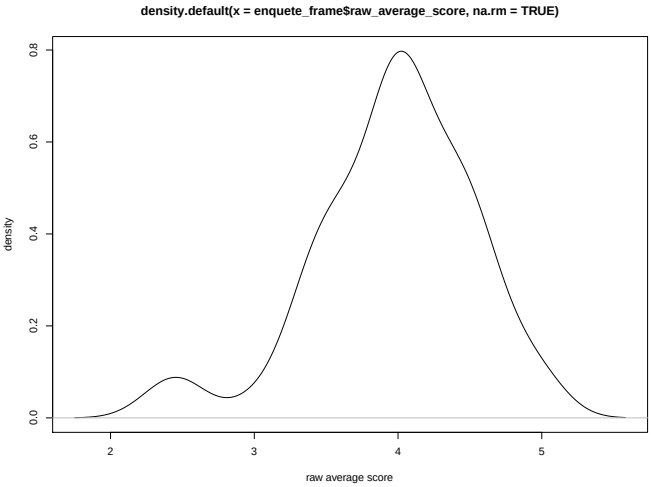


図 14　すべての回を通しての平均点の分布

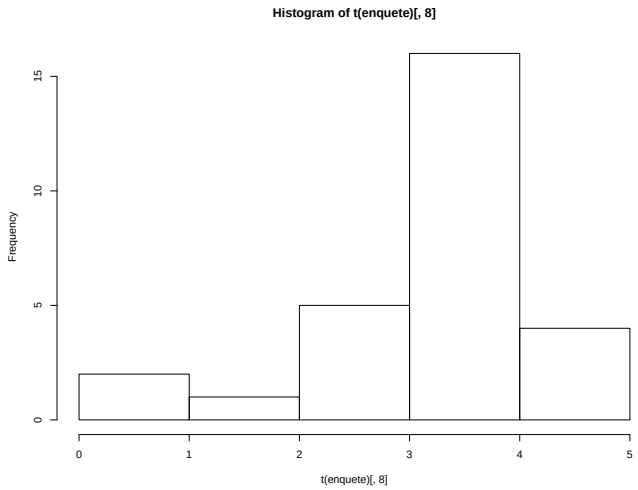


図 15　あるユーザのつけたスコア分布

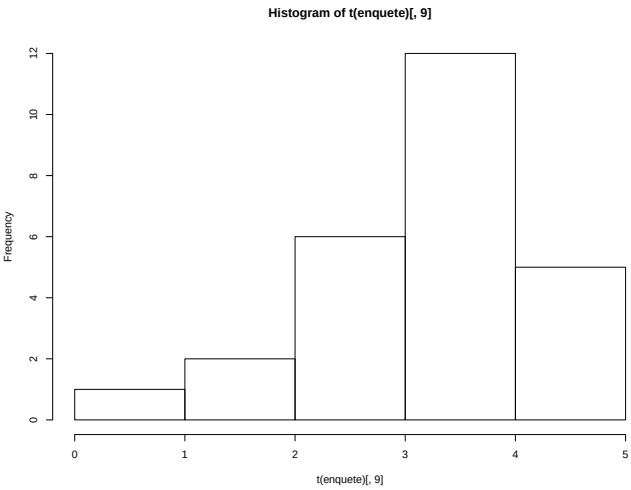


図 16　また別のユーザのつけたスコア分布

```
> raw_average_score[!is.na(raw_average_score) & raw_average_score < 3]
                第 71 回東京エリア Debian 勉強会.2010 年 12 月勉強会.Debian.Miniconf. 企画.2eca
                                                                    2.333333
              第 84 回東京エリア Debian 勉強会.2012 年 1 月勉強会. 事前課題紹介.2012 年企画.f447
                                                                    2.500000
第 84 回東京エリア Debian 勉強会.2012 年 1 月勉強会. 第 3 回月刊 Debhelper.dh_auto_..dh_build.f447
                                                                    2.500000
> enquete_response[!is.na(raw_average_score) & raw_average_score < 3]
                第 71 回東京エリア Debian 勉強会.2010 年 12 月勉強会.Debian.Miniconf. 企画.2eca
                                                                            6
              第 84 回東京エリア Debian 勉強会.2012 年 1 月勉強会. 事前課題紹介.2012 年企画.f447
                                                                            4
第 84 回東京エリア Debian 勉強会.2012 年 1 月勉強会. 第 3 回月刊 Debhelper.dh_auto_..dh_build.f447

> scaled_average_score[' 第 84 回東京エリア Debian 勉強会.2012 年 1 月勉強会. 事前課題紹介.2012 年企画.f447']
第 84 回東京エリア Debian 勉強会.2012 年 1 月勉強会. 事前課題紹介.2012 年企画.f447
                                                                   -1.289405
                                                                            4
```

一方でハイスコア側を眺めてみましょう。最高のスコアは「第 79 回東京エリア Debian 勉強会.2011 年 8 月勉強会.Debian パッケージのビルド方法」と「第 91 回東京エリア Debian 勉強会.2012 年 8 月勉強会. 月刊.Debhelper. 共有ライブラリ編」です。それぞれ n=2 と n=3 で全員最高点を指定した結果です。アンケートに回答してくれた人数が少ないのが気になりますが、力作で聞いててておもしろいものでした。「第 72 回東京エリア Debian 勉強会.2011 年

1 月勉強会.Kinect」4.875($sd = 0.35, n = 8$）と「第 95 回東京エリア Debian 勉強会.2012 年 12 月勉強会. 著作権法改正」4.75($sd = 0.5, n = 4$) が比較的投票数も高くてスコアの高いものです。Kinect はデモ満載でやっていることもぶっ飛んでました。著作権法改正については問題意識を刺激する内容だったのではないでしょうか。

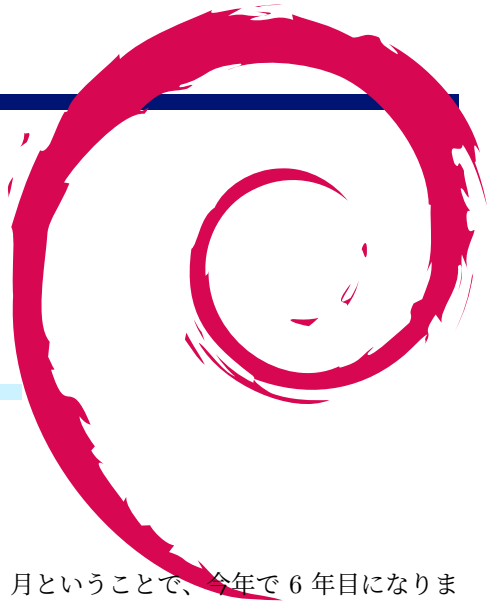
```
>  raw_average_score[!is.na(raw_average_score) & raw_average_score > 4.5]
      第 71 回東京エリア Debian 勉強会.2010 年 12 月勉強会.CACert の準備に何が必要か.2eca
                                                                4.600000
       第 71 回東京エリア Debian 勉強会.2010 年 12 月勉強会. 俺の libsane が火をふくぜ.2eca
                                                                4.666667
                     第 72 回東京エリア Debian 勉強会.2011 年 1 月勉強会.Kinect.f456
                                                                4.875000
   第 79 回東京エリア Debian 勉強会.2011 年 8 月勉強会.Debian パッケージのビルド方法.5dff
                                                                5.000000
            第 91 回東京エリア Debian 勉強会.2012 年 8 月勉強会.Debian で C..11 を使う.9796
                                                                4.666667
第 91 回東京エリア Debian 勉強会.2012 年 8 月勉強会. 月刊.Debhelper. 共有ライブラリ編.9796
                                                                5.000000
                  第 95 回東京エリア Debian 勉強会.2012 年 12 月勉強会. 著作権法改正.3f15
                                                                4.750000
> enquete_response[!is.na(raw_average_score) & raw_average_score > 4.5]
      第 71 回東京エリア Debian 勉強会.2010 年 12 月勉強会.CACert の準備に何が必要か.2eca
                                                                5
       第 71 回東京エリア Debian 勉強会.2010 年 12 月勉強会. 俺の libsane が火をふくぜ.2eca
                                                                3
                     第 72 回東京エリア Debian 勉強会.2011 年 1 月勉強会.Kinect.f456
                                                                8
   第 79 回東京エリア Debian 勉強会.2011 年 8 月勉強会.Debian パッケージのビルド方法.5dff
                                                                2
            第 91 回東京エリア Debian 勉強会.2012 年 8 月勉強会.Debian で C..11 を使う.9796
                                                                3
第 91 回東京エリア Debian 勉強会.2012 年 8 月勉強会. 月刊.Debhelper. 共有ライブラリ編.9796
                                                                3
                  第 95 回東京エリア Debian 勉強会.2012 年 12 月勉強会. 著作権法改正.3f15
                                                                4
```

21.2 アンケートの設計について

今回みられた現象としては、５段階評価の４に評価が集まってしまいました。素朴にとらえると「よい」といってもらえていることになります。しかしこれだと分解能が低いと捉えることもできます。

これは「中心化傾向」もしくは「寛大化傾向」とよばれる現象にあてはまると思います。

アンケート設計の世界ではそれなりにいろいろな対策方法が存在しているようですがまだ僕がよく理解できてません。



22　2012 年の振り返りと 2013 年の企画

佐々木洋平/Debian JP Project

今月が 2012 年最後の関西 Debian 勉強会になります。初回が 2007 年 3 月ということで、今年で 6 年目になりますね。今年は定例となっている OpenSource 関連のイベントへの参加だけではなく「大統一 Debian 勉強会」を開催することができたのが大きな収穫だと思っています。

22.1　勉強会全体について

22.1.1　月刊 Debian Policy

今年は「月刊物をなんかやってみないか?」ということで「月刊 Debian Policy」を始めてみました。Debian Policy Manual, version 3.9.1 の日本語訳は Debian JP Project によって公開されています[*38]。「月刊 Debian Policy」では、最新の Debian Policy Manual [*39] を読み進めて勉強しながら、現在公開されている日本語訳と突き合わせつつ、(あわよくば) これを更新したいと考えています。「Debian Policy Manual を勉強する」という目的は毎回の勉強会で非常に上手く進んでいると思います。今後も進めて行きたいと思いますが、如何でしょうか。

22.1.2　翻訳ネタ

今年後半の関西 Debian 勉強会では「翻訳」に関する発表が幾つか行なわれました。「翻訳」されて母国語で文書が読めることは、ソフトウェアの普及やコミュニティの参加への敷居を下げるという点で、非常に重要です。Debian JP では (不幸な事に)、作業プロセスが不透明であったり、幾つかの (非常に重要な) 翻訳作業が属人化している、という問題もあります。今年の勉強会でせっかく盛り上がりつつある「翻訳熱」を冷ますことなく、今後はなんらかの改善が行なわれると良いかな、と思っています。

22.1.3　その他, 定例ネタ

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」に関する題材として「t-code」「Konoha」に関するお話を複数回にわたって発表して頂きました。具体的に弄りたいパッケージがあると、(発表者のモチベーションが違うのか) 非常に良い勉強会になった気がします。来年は Wheezy がリリースされることですし、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) などの題材に加えて、「Wheezy から始める Debian 環境」みたいなお話もあったら良いかな、と考えています。昨年は案が出たけで結局実施できなかった「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。

[*38] http://www.debian.or.jp/community/devel/debian-policy-ja/policy.ja.html/

[*39] http://www.debian.org/doc/debian-policy/
現時点での最新版は 3.9.4.0, 2012-09-19

22.2 運営

運営に関しては、今年の (特に後半は) 河田さんの獅子奮迅の活躍により勉強会が開催できていました。運営側が多忙となってしまうのはある程度しかたない (かもしれない) ですが、今後はもう少し個人の負担を減らしつつ開催していきたいと考えています。

22.3 イベント/NM 申請

イベント参加については、OSC Kansai@Kyoto, KOF 2012 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。また、1 月には関西初の「Debian 温泉合宿」が、6 月には東京エリア Debian 勉強会と合同で「大統一 Debian 勉強会」を開催しました。これらのイベントは今後も継続して実施する予定です。

Debian 開発者への道、として佐々木と倉敷さんの二人が NM プロセスにて DD になるための申請中です。今後も NM プロセスに挑む参加者や Debian Maintainer となって精力的にパッケージを開発/更新する人が増えると良いですね。

22.4 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 17 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 5 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。

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。

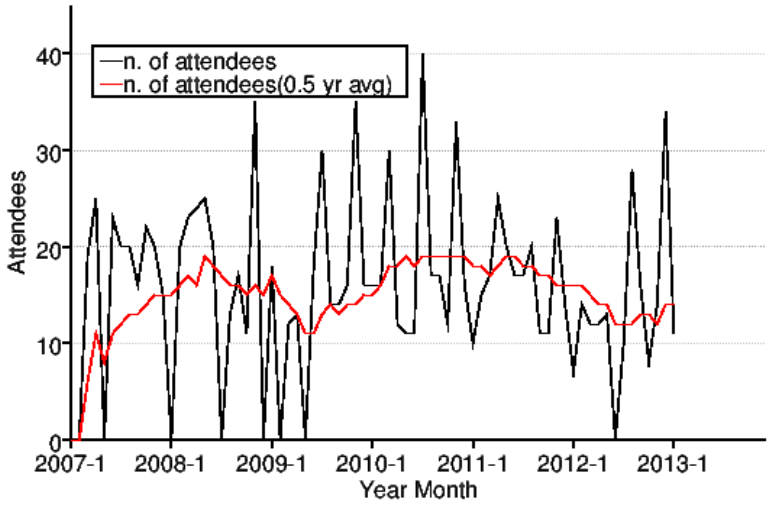


図 17　関西の参加人数推移 (参加人数と 6ヶ月移動平均)

表 2　関西 Debian 勉強会の参加人数とトピック (2007-2008 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

	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 3　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

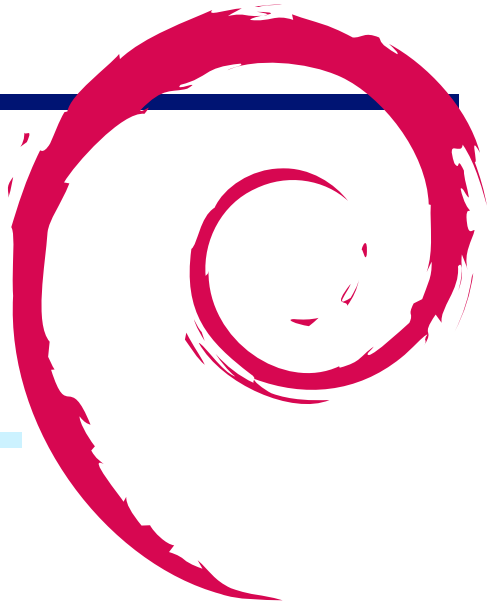
	参加人数	内容
2010 年 1 月	16	Xen, 2010 年企画
2010 年 2 月	16	レンタルサーバでの利用, GAE
2010 年 3 月	30?	OSC2010Kobe
2010 年 4 月	12	デスクトップ環境, 正規表現
2010 年 5 月	11	ubuntu, squeeze
2010 年 6 月	11	debhelper7, cdbs, puppet
2010 年 7 月	40?	OSC2010Kyoto
2010 年 8 月	17	emdebian, kFreeBSD
2010 年 9 月	17	タイル WM
2010 年 10 月	12	initramfs, debian live
2010 年 11 月	33	KOF2010
2010 年 12 月	14	Proxmox, annual review

表 4　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

表 5　関西 Debian 勉強会の参加人数とトピック (2012)

開催年月	参加人数	内容
2012 年 1 月	7	Debian 温泉合宿
2012 年 2 月	14	autofs+pam_chroot, t-code その 1, 月刊 Debian Policy その 1
2012 年 3 月	12	新年度スケジューリング, Konoha その 1, t-code その 2, 月刊 Debian Policy その 2
2012 年 4 月	12	フリーソフトウェアと著作権, Konoha その 2, 月刊 Debian Policy その 3
2012 年 5 月	13	Debian と LDAP(頓挫), ITP 入門, 月刊 Debian Policy その 4
2012 年 6 月	-	大統一 Debian 勉強会
2012 年 7 月	10	Debian と LDAP その 1, 大統一 Debian 勉強会報告, 月刊 Debian Policy その 5
2012 年 8 月	28	OSC 2012 Kansai @ Kyoto, GPG キーサインパーティ
2012 年 8 月	16	Debian と Kerberos, News from EDOS
2012 年 9 月	8	clang によるパッケージビルド, 月刊 Debian Policy その 6
2012 年 10 月	14	翻訳環境構築, DSA の舞台裏
2012 年 11 月	34	KOF 2012
2012 年 12 月	12	Debian on Android, 月刊 Debian Policy その 7



23 2012 年度東京エリア Debian 勉強会の振り返り

上川 純一

今月で 8 年目の東京エリア Debian 勉強会が終了しました。

23.1 基本的な数値

出席数の推移をみましょう。最近は出席が把握できていない回が多いのですが、だいたい参加者数１２人くらいで推移しているようです (図 18)。事前課題の提出数 (図 19) は最近は安定していて提出率が向上しているようです。

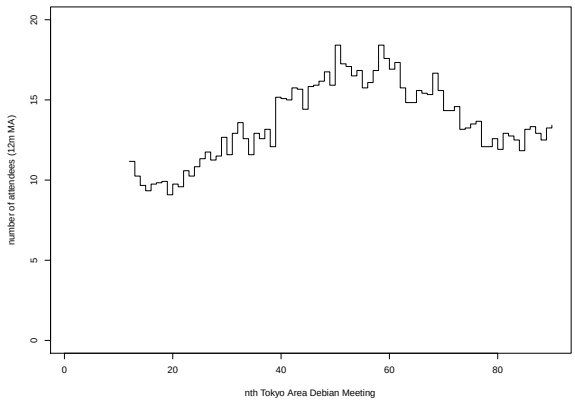


図 18 東京エリア Debian 勉強会出席実績 (12ヶ月移動平均)

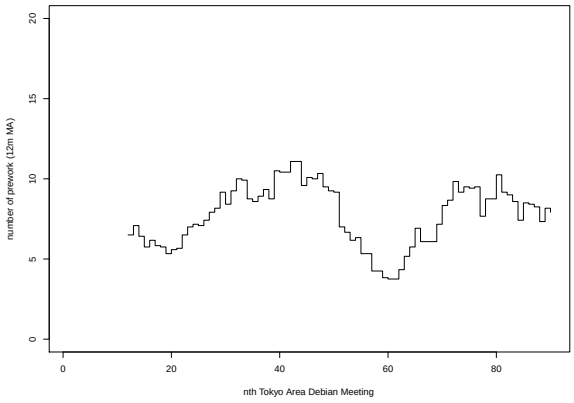


図 19 東京エリア Debian 勉強会事前課題提出実績 (12ヶ月移動平均)

23.2 過去のテーマ

過去のテーマを眺めてみましょう。2012 年は月刊 Debhelper を中心に Debian パッケージ開発の基本的な部分をおさえつつ、Debian を使って開発する開発者のための応用的なテーマについて多くとりあげたと思います。

表 6　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 7　東京エリア Debian 勉強会参加人数 (2007-2008 年)

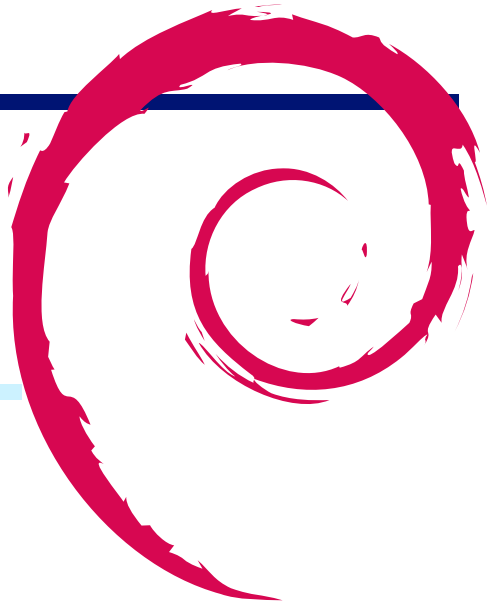
	参加人数	内容
2007 年 1 月	15	一年を企画する
2007 年 2 月	13	dbs, dpatch
2007 年 3 月	80	OSC 仮想化
2007 年 4 月	19	quilt, darcs, git
2007 年 5 月	23	etch, pbuilder, superh
2007 年 6 月	4	エジンバラ開催：Debconf7 実況中継
2007 年 7 月	18	Debconf7 参加報告
2007 年 8 月	25	cdn.debian.or.jp
2007 年 9 月	14	exim
2007 年 10 月	30	OSC Tokyo/Fall(CUPS)
2007 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 8　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 9　東京エリア Debian 勉強会参加人数 (2011-2012 年)

	人数	内容
2011 年 1 月	12	Kinect, アンケートシステム,CACert サイン会
2011 年 2 月	13	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	Apache2 モジュール,Debian on ニフクラ,Debian/m68k, 月刊 PPC64
2011 年 6 月	17	ドキュメント処理系,2011 再計画
2011 年 7 月	3	DebConf 11
2011 年 8 月	12	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, 月刊デブヘルパー開始
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, 月刊デブヘルパー, 振り返り
2012 年 1 月	8	Debian 勉強会予約システム,VPS, twitter, 月刊デブヘルパー, 2012 年計画
2012 年 2 月	4	KDE 開発, 月刊デブヘルパー, cmake, 第 0 回福岡勉強会
2012 年 3 月	?	OSC
2012 年 4 月	13	node.js, android で Debian, 月刊デブヘルパー
2012 年 5 月	14	coffeescript,python
2012 年 6 月	?	大統一 Debian 勉強会
2012 年 7 月	8?	MacBook Air 2011
2012 年 8 月	6?	Debconf 2012, 月刊デブヘルパー,C++11
2012 年 9 月	12	OSC Tokyo Fall
2012 年 10 月	10	Haskell, レゴ, xf86-input-mtrack
2012 年 11 月	14	bluetooth tethering, linux perf, systemd
2012 年 12 月	?	忘年会



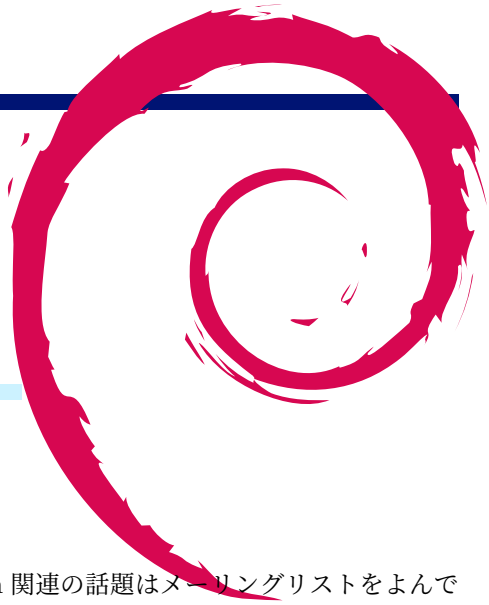
24 Debian 勉強会 2013 年度計画

上川 純一

24.1 2015 年までを妄想する

2011	2012	2013	2014	2015
デスクトップパソコン終了の潮流。 cpu コア単体では高速化しないように。 webos 終了のお知らせ。 adobe flash 復活のお知らせ (キタ), silverlight 終了のお知らせ (台湾を除く)(続いてる?) squeeze リリース (おめでとう) ipv4 割り当ての終了のお知らせ (キタ) 地上波デジタル移行延長。 btrfs まだ頑張る (fedora 乙) java 終了 (sun java 終了) open office が oracle office に (ナイ)	ノートパソコンよりタブレットのほうが売れている。ノートパソコンでは macbookair が常識に。 ノートパソコンで intel じゃないもの (mips/arm) が主流にはまだならず。タブレットのほうが主流。 デスクトップ:ゲーム以外の用途では終了している。 サーバ:個人レベルでは VPS 常識。企業ユースでも cloud か、vps かを自前と比較検討する時代。データセンターを置く国を選べる時代。 携帯電話: ガラパゴスの終焉。日本での携帯電話販売でもスマートフォンが５０%を超えるように。ガラケー向けのネットバンクの提供が終了など、ガラケーからサービスが撤退し始める。LTE 登場、普及しはじめたが、主流になっていない。softbank の二年契約はまだ続いている。sim free への道は耕されたがあたり前にならなかった。 btrfs はまだ生き残っているがまだ使われてない? openstack で ceph 使う人もいる? mysql から mariadb が派生。	コンシューマーはノートパソコンを買わなくなった。ノートパソコンのかわりにスマートフォンを使っている。 スマートフォンが 7 インチくらいまで拡大、タブレットとは何だったのか。 自宅用のデスクトップパソコンのかわりに 10 インチくらいのタブレットを使うように。 サーバ:クラウドで処理するのが主流。python / ruby でコード書いていると CPU が何かわからない。裏で動いている CPU は一般人は知らない。 ARM ホストの仮想化技術が発達。 Oracle がメンテナンスする気がないのが明確になり、java リスクが顕著になる。 固定ゲーム機の終焉。 ゲームは ARM。	Intel がまた ARM に参入、もしくは省電力 CPU を主力に切り替える。 気づいたら自作パソコン業界が終焉している。セキュアブートが普及している。 AMD が ARM コアの CPU を出す。 Java が Oracle 管理からはずれる。 スマートフォンの電池がガラケーなみに持つようになる。 電池消費が重要なアプリ選択の要素となる。スポイトで充電できる、燃料電池が流行る。 AR メガネのプロトタイプが出てくる。	自作スマホの時代。 OpenHardware がモバイルに移行する。技適のパーツ認定基準というのができるようにがんばる。 自宅で回路が印刷できる機器が普及して CPU とかが印刷できるようになるといいな。 タブレットが丸められるようになって巻物になっている。 AMD が x86 撤退。 ハードディスクを見たことがない人がいる。 データセンターを自前でもっているのは発電所を持っているところだけになる。 クラウドの法制度、免責事項、個人情報保護関連の問題が提起され、解決にむけてすすむ。一種データセンタークラウド業者の要求規格が制定される。ユーザ数何人以上は二種免許が必要とか。 データセンターヘイブンとよばれる国が存在する。

2015 年にどうなっているのかを妄想したところで、2013 年度の計画を立てましょう。



25 Debian Trivia Quiz

上川 純一、岩松 信洋

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. wiki.debian.org で使われていた wiki システムのソフトウェアの名前は？
☐ A pukiwiki
☐ B mine
☐ C moin

問題 2. 去年の年末の popcon 調査で No.1 に輝いたプラットフォームは？
☐ A 当然 armel だろ？
☐ B i386
☐ C amd64

問題 3. 1 月頭の DPN で報告のあった、wheezy に残っている RC バグの数は 1 月 5 日時点であと何個？
☐ A あと 321 個
☐ B あと 171 個
☐ C あと 17 個

問題 4. 今年 amazon から Debian Project へいくらかの AWS の利用券をスポンサーしてもらったそうなのですが、金額にするとおいくら？
☐ A 8000USD
☐ B 800USD
☐ C 80000USD

問題 5. DebConf13 の開催地と開催日は？
☐ A 日本 東京都 6 月 29 日
☐ B ニカラグア マナグア 7 月 8-14 日
☐ C スイス ヴォーマルキュ 8 月 11-18 日

問題 6. 世界の Web サーバで最も人気のある Linux ディストリビューション (W3Techs 調べ) は？
☐ A CentOS
☐ B Debian
☐ C Ubuntu

問題 7. 現在 Debian プロジェクトリーダ選挙 2013(Debian Project Leader Elections 2013) が開催されています。現在 (3/16) のステータスは？
☐ A 指名期間
Nomination period
☐ B 選挙運動期間
Campaigning period
☐ C 投票期間
Voting period

問題 8. Ben Hutchings さんが次期 Debian 安定版と一緒に出荷される Linux カーネルに (3.2 系列の mainline には無い) 追加機能が搭載される予定であると述べています。多くの追加点の中に含まれないものは何？
☐ A リアルタイム性を強化する PREEMPT_RT
☐ B Hyper-V guest drivers
強化
☐ C TCP 接続高速化技術
TCP Fast Open

問題 9. Wookey さんがアナウンスした alpha 版の Debian port ARM64 image は？
☐ A Debian/Ubuntu
port image
☐ B Debian/KFreeBSD
port image
☐ C Debian/GnuHurd
port image

問題 10. 700,000 番目のバグが報告された日を当てる 700000thBugContest の結果が出ました。その予想日と対象バグの報告日は？
☐ A 予想日:2013/02/04、
報告日:2013/02/14
☐ B 予想日:2013/02/07、
報告日:2013/02/14
☐ C 予想日:2013/02/14、
報告日:2013/02/07

問題 11. master.debian.org が新しい機械に移行されました。これは何のサーバでしょうか ？
☐ A @debian.org のメールサーバ
☐ B パッケージのマスターサーバ
☐ C パッケージのスポンサー (mentor) を探すサーバ

問題 12. pbuilder でビルドする際に gcc の代替として Clang を使いやすくするパッチが追加されました。誰が書いたパッチでしょうか？
☐ A Sylvestre Ledru
☐ B Junichi Uekawa
☐ C Hideki Yamane

問題 13. DPN - 2013 年 3 月 4 日号に取り上げられた日本のイベントは
☐ A Open Source Conference 2013 Tokyo/Spring
☐ B Open Source Conference 2013 Hamamatsu
☐ C Open Source Conference 2013 Tokushima

問題 14. DSA のサーバを新しくホスティングしてくれるのはどれか
☐ A bytemark
☐ B grnet
☐ C manda

問題 15. wheezy の backports の apt-line で適切なものはどれか
☐ A deb sstp://backports.debian.org/debian/ wheezy-backports backports
☐ B deb http://backports.debian.org/debian/ wheezy-backports main
☐ C deb http://ftp.debian.org/debian/ wheezy-backports main

問題 16. tech ctte 699808 の結果アップロードされたのは
☐ A syslinux 4
☐ B syslinux 5
☐ C grub

問題 17. あたらしく DPL になったのはだれか
☐ A Stephano Zacchiroli
☐ B Lucas Nussbaum
☐ C Gergely Nagy

26 Debian Trivia Quiz 問題回答

上川 純一、岩松 信洋

Debian Trivia Quiz の問題回答です。 あなたは何問わかりましたか？

1. C 誠に遺憾ながら、wiki.debian.org がこの前クラックされたようです。他で同じ登録メールアドレスとパスワードの組を使っている人は、すぐにパスワードを変えた方がよいでしょう。

2. C `http://popcon.debian.org/` 今まで No.1 を決めていた i386 を年末で amd64 が抜き去ったようです。

3. B 簡単に直せる/他治れば勝手に治りそうものを含んで 171 個。手がかかりそうなのはだいたい 116 個。

4. A 日本円にすると約 72 万円分の利用券。QA には十分とのことでした。

5. C ニカラグアは DebConf12 の開催地。DebConf13 はスイスのキャンプ地で開催。6/29 は大統一 Debian 勉強会。

6. B `http://w3techs.com/technologies/history_details/os-linux` に結果のグラフがあります。現在 Linux を使用している web サーバの 32.9% が Debian を利用しており、その割合は現在も増加を続けているそうです。

7. B 指名期間:2013 年 3 月 3 日 (日)UTC ～ 3 月 9 日 (土) UTC 選挙運動期間:2013 年 3 月 10 日 (日) UTC ～ 3 月 30 日 (土) UTC 投票期間:2013 年 3 月 31 日 (日)UTC ～ 4 月 13 日 (土)UTC `http://www.debian.org/vote/2013/vote_001`

8. C Ben Hutchings さんは Debian カーネルチームのメンバーであり、kernel.org の 3.2.y 安定版系列のメンテナ。PREEMPT_RT は I/O の遅延と揺らぎを低減する audio/video アプリケーション利用等向、linux-image-rt-amd64 , linux-image-rt-686-pae で使用可能。Hyper-V guest drivers は 3.2 にも含まれていますが、より改善された 3.4 からの修正を導入。TCP Fast Open(TFO) は TCP の 3way-handshake を改善し、2 回目以降の通信開始時に SYN と data を含んだパケットを送ることで高速化する技術、3.6 時点ではクライアント側のみ実装、サーバ側は 3.7 から。

9. A x86 環境を使用しない self-bootstrap image です。`http://wiki.debian.org/Arm64Port` でステータスが確認できます。mainline kernel の ARM64/AArch64(ARMv8 アーキテクチャ:Cortex A57,Cortex A53 予定の CPU 向け) 対応は 3.7 から。

10. C 最も近い 2013/02/14 を予想した Christian Perrier さんが当てました。結果は `http://wiki.debian.org/700000thBugContest` で公開されています。また、800,000/1,000,000 番目のバグが報告される日を当てるコンテスト `http://wiki.debian.org/800000thBugContest` も開催されています。

11. A 古いサーバはディスク障害等で、寿命と判断され、新しいサーバに移行されました。ftp-master.debian.org は Debian の official package リポジトリです。パッケージのスポンサー (mentor) を探すのは mentors.debian.net。

12. C Sylvestre Ledru さんは Clang のメンテナーの一人です。彼は Clang 3.2 を使用してアーカイブをリビルドした結果について報告しました (12.1% が失敗、つまり 18264packages 中、2204 が失敗)。失敗したパッケージの情報 `http://clang.debian.net/pts.php`。Debian の Clang サポートは着々と進んでいます。

13. A `http://henrich-on-debian.blogspot.jp/2013/02/open-source-conference-2013-tokyospring.html`

14. A Debian プロジェクトのサーバ管理を担うチームが DSA、その管理する Debian サーバのホスティングは、grnet, manda, ubcece が既存のホスティングプロバイダーで、bytemark が今回加わったそうです。

15. C backports がメインのアーカイブと統合され、wheezy-backports から利用できるようになったようです。

16. A 新しいバージョンの syslinux で debian-installer が動かないという問題が有りダウングレードで解決したようです。

17. B Lucas が選挙の結果 DPL になりました

『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』（第第 95 回から第 99 回）および『関西 Debian 勉強会』（第 67 回から第 72 回）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。第 70 回関西 Debian 勉強会資料は別ライセンスため無し、第 97 回東京エリア Debian 勉強会資料はオープンソースカンファレンス 2013 Tokyo/Spring のため無し、第 100 回東京エリア Debian 勉強会資料はアンカンファレンス & Wheezy リリースパーティのため無し、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 2013 年夏号

2013 年 8 月 12 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）