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 2012 年冬号 2012 年 12 月 31 日 初版発行

デビアン勉強会

目次

1 Introduction

DebianJP

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。

- ML 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 MacBook Air 4,1 (Mid 2011) にDebian インストールしてみた

上川 純一

2.1 はじめに

Apple の MacBook Air などの製品は通常の PC とは違い起動が EFI 経由でなされるなどの癖があります。久しぶりにどうやってインストールして動かすのかについてまとめてみました紹介します。今回は rEFIt[3] 経由で grub-pc を利用する Mac OS X との dual boot 構成にしています。

2.2 インストール開始から Dual Boot で起動するまで

まず、 Mac OS X がディスク全体を利用しているため、インストールできるスペースがありません。 Mac OS X のコマンドラインから diskutil resizevolume を使ってディスクを小さくします。

```
# sudo diskutil resizevolume disk0s2 30G
```

Mac の起動は BIOS ではなく EFI 経由なのですが、 rEFIt は EFI で起動するブートローダーです。メニュー形式やコマンドラインで Grub をチェーンロードしたりできます。 Mac OS で rEFIt をインストールします。最近の rEFIt はインストーラーに従うと自動でインストールしてくれるようになっているようで自分で Bless コマンドをうったりする必要はありません。

何度か再起動すると起動時に rEFIt の画面に切り替わるようになるようです。

Debian Installer の名刺サイズの ISO イメージを USB メモリ[*1]に dd で書き込んでおくと、再起動したときに rEFIt から debian installer を起動する選択肢が出てきます。

インストールは通常のように行います。引っかかる場所は二箇所。 Grub のインストールの部分とネットワークの設定でしょう。

無線ネットワークの brcmsmac ドライバが対応しているのですが、ファームウェアが必要で Non-free のためインストーラに標準で入っていません。インストーラ用にファームウェアを事前に用意する必要がありますが、僕は面倒だったので USB Ethernet をつなげてインストールしてしまいました。

Grub のインストールの部分はうまく行かず試行錯誤してしまったのですが、 refit パッケージにある gptsync コマンドを実行して GPT と FAT のパーティションテーブルを同期させ、 grub-pc を Debian installer がつくった小さなパーティションにインストールすることで解決しました。多分コマンドラインはこんな感じだったと思います。[*2]Debian installer のシェルの中で target に chroot して作業しました。

[*1] 僕は今回は USB SD リーダーに余った micro SD をさして使いました。

[*2] grub-efi で BIOS エミュレーションではなく EFI モードを使い起動し、 GPT から直接ブートできるはずだと思われるのですが未確認。一部のより新しいハードウェアでは EFI モードで直接起動するほうがうまくうごくデバイスなどがあるらしい。

```
# gptsync
# grub-install (hd0,4)
```

2.2.1 ディスクパーティション構成

GPT / FAT のパーティション構成は結果として次のようになりました。

```
$ cat /proc/partitions
major minor  #blocks  name

   8        0  245117376 sda
   8        1     204800 sda1  # EFI
   8        2   39062500 sda2  # Mac OS X
   8        3     634768 sda3  # Mac OS X recovery?
   8        4        977 sda4  # grub
   8        5  197461914 sda5  # /
   8        6    7752380 sda6  # swap

$ sudo fdisk -l /dev/sda

WARNING: GPT (GUID Partition Table) detected on '/dev/sda'! The util fdisk doesn't support GPT. Use GNU Parted.


Disk /dev/sda: 251.0 GB, 251000193024 bytes
255 heads, 63 sectors/track, 30515 cylinders, total 490234752 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0000221d

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1               1      409639      204819+  ee  GPT
/dev/sda2          409640    78534639    39062500   af  HFS / HFS+
/dev/sda3        78534640    79804175      634768   ab  Darwin boot
/dev/sda4   *    79804176    79806129         977   c0  Unknown
```

Mac OS X のパーティションは Linux からは hfsplus ファイルシステムとしてマウントすることができます。

```
sudo mount /dev/sda2 /mnt/ -o ro
$ mount -v
/dev/sda2 on /mnt type hfsplus (ro,relatime,umask=22,uid=0,gid=0,nls=utf8)
```

2.3 無線 LAN

brcmsmac ドライバで無線 LAN は動きます。 non-free firmware が動作に必要になります。自由じゃない Binary Blob などけしからんですが、 non-free にパッケージがあるのでインストールしておけばうごきます。

```
ii  firmware-brcm80211         0.35                       Binary firmware for Broadcom 802.11 wireless cards
```

2.4 X

2.4.1 画面

Wheezy に入っている Linux カーネル 3.2 だとそのままいい解像度で動いてくれます。 xorg としては `Integrated Graphics Chipset: Intel(R) Sandybridge Mobile (GT2)` を認識してくれているようです。

```
$ xrandr
Screen 0: minimum 320 x 200, current 1366 x 768, maximum 8192 x 8192
eDP1 connected 1366x768+0+0 (normal left inverted right x axis y axis) 256mm x 144mm
   1366x768       60.0*+
   1360x768       59.8     60.0
   1024x768       60.0
   800x600        60.3     56.2
   640x480        59.9
VGA1 disconnected (normal left inverted right x axis y axis)
HDMI1 disconnected (normal left inverted right x axis y axis)
DP1 disconnected (normal left inverted right x axis y axis)
HDMI2 disconnected (normal left inverted right x axis y axis)
HDMI3 disconnected (normal left inverted right x axis y axis)
DP2 disconnected (normal left inverted right x axis y axis)
DP3 disconnected (normal left inverted right x axis y axis)
```

2.4.2 キーボード

英語キーボード版を買ったのですが、何も特に設定した記憶がありません。

```
[    11.567] (II) XINPUT: Adding extended input device "Apple Inc. Apple Internal Keyboard / Trackpad" (type: KEYBOARD, id 10)
[    11.567] (**) Option "xkb_rules" "evdev"
[    11.567] (**) Option "xkb_model" "pc105"
[    11.567] (**) Option "xkb_layout" "us"

[    11.685] (II) XINPUT: Adding extended input device "ACPI Virtual Keyboard Device" (type: KEYBOARD, id 13)
[    11.685] (**) Option "xkb_rules" "evdev"
[    11.685] (**) Option "xkb_model" "pc105"
[    11.685] (**) Option "xkb_layout" "us"
```

2.4.3 キーボードによるマウスクリックエミュレーション

マルチタッチトラックパッドになったので右クリックなどがタッチでできるようになっているため、最近はこの設定は必要ありません。

mouseemu パッケージをインストールするとデフォルトで F10、 F11 がマウスの真ん中ボタンと右ボタンクリックにアサインされます。僕の場合は最初から mouseemu パッケージがインストールされてて気づかずはまりました。

カーネルの USB ドライバ mac hid でも設定できるみたいです。カーネルで設定する場合はここらへんを参照してください:

```
/proc/sys/dev/mac_hid/mouse_button2_keycode
/proc/sys/dev/mac_hid/mouse_button3_keycode
/proc/sys/dev/mac_hid/mouse_button_emulation
```

2.4.4 マルチタッチトラックパッド

デフォルトの状態でも synaptics でシングルタッチトラックパッドとして動きますが、 xserver-xorg-input-mtrack でマルチタッチトラックパッドとして動きます。

```
# apt-get install xserver-xorg-input-mtrack xserver-xorg-input-synaptics-
```

個人的にはタッチ判定が敏感すぎてキーボード入力しているとマウスクリック判定されて悩んでます。ドキュメント [1, 2] によると設定は可能なようです。

ものは試しと一つ指タップを判定しないように `/etc/X11/xorg.conf.d/50-mtrack.conf` に以下の設定を追加してみています:

```
Section "InputClass"
        MatchIsTouchpad "true"
        Identifier "Multitouch Touchpad"
        Driver "mtrack"
        Option "TapButton1" "0"
EndSection
```

2.5 スリープ

なんか電源ボタンをおしたり蓋を閉じたりするとスリープしてくれるようです。

ほんのたまに無線 LAN が認識しなくなりますが brcmsmac モジュールを rmmod / modprobe するとなおるっぽい。さらにほんのたまに電池の残量を認識しなくなる。

参考文献

[1] How to configure input `/usr/share/doc/xorg/howto/configure-input.txt.gz`

[2] xf86-input-mtrack documentation. `/usr/share/doc/xserver-xorg-input-mtrack/README.md.gz`

[3] rEFIt `http://refit.sourceforge.net/`

3 xserver-xorg-input-mtrack

岩松 信洋

Apple 社の Macbook Pro の タッチパッドは 2011 年 (2010 年?) 以降からマルチタッチに対応したトラックパッドを使っています。以前のトラッパッドでもマルチタッチができましたが、一部の機能しか使うことができないようです(pre-multitouch と呼ばれるようです)。 pre-multitouch では xserver-xorg-input-synaptics ドライバで動作していたのですが、最新の Macbook pro / air / Magic Trackpad では動作しません。これらの環境でマルチタッチを使うためには xserver-xorg-input-mtrack / multitouch といった他のドライバを利用する必要があります。今回、これらの情報をまとめました。

3.1 synaptics と mtrack

今まで使われてきた synaptics では、プロトコル A(図 2) が主体です。これはタッチ ID を持っていないため、トラッキングコンタクトを管理できません。

mtrack では、タッチ ID をサポートしたプロトコル「 プロトコル B (図 2) 」を使います。これによりより細かいタッチパッドの制御ができるようになっています。また、「 プロトコル B」を Linux カーネルから受信し、それを X ドライバにわかりやすい(人間にとってわかりやすい) 形にデータを形成するライブラリ mtdev を使います。

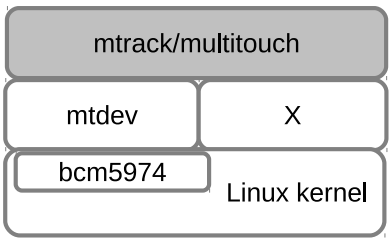


図 1 関係図

```
ABS_MT_POSITION_X x[0]
ABS_MT_POSITION_Y y[0]
SYN_MT_REPORT
ABS_MT_POSITION_X x[1]
ABS_MT_POSITION_Y y[1]
SYN_MT_REPORT
SYN_REPORT
```

図 2 プロトコル A

```
ABS_MT_SLOT 0
ABS_MT_TRACKING_ID 45
ABS_MT_POSITION_X x[0]
ABS_MT_POSITION_Y y[0]
ABS_MT_SLOT 1
ABS_MT_TRACKING_ID 46
ABS_MT_POSITION_X x[1]
ABS_MT_POSITION_Y y[1]
SYN_REPORT
```

図 3　プロトコル B

3.2　multitouch と mtrack

現在、 Macbook Pro / Air 用のタッチパッドをサポートしているドライバは multitouch と mtrack の 2 つがあります。 multitouch が先に作られましたが、基本的な設定しかできないシンプルな設定だったため、 mtrack という multitouch からフォークしたドライバが作成されました。こちらはタッチの圧力やジェスチャーのサポートが行われているため、使っているユーザが多いようです。

3.3　Debian で使う

Debian では既にパッケージ化されており、 APT でインストールできます。

```
$ sudo apt-get install xserver-xorg-input-mtrack
```

インストールされると、図 4 の内容の xorg 設定ファイルが /usr/share/X11/xorg.conf.d/以下にインストールされ、xorg.conf に記述しなくても動作するようになります。

```
Section "InputClass"
    MatchIsTouchpad "true"
    Identifier "Multitouch Touchpad"
    Driver "mtrack"
EndSection
```

図 4　mtrack のデフォルト設定

インストールした段階では、デフォルトの設定で動作します。以下によく使われる設定項目を表 1 示します。

項目	内容	デフォルト値
TrackpadDisable	トラックパッド機能の動作内容と無効化	0
Sensitivity	トラックパッドのスピード	1
FingerHigh	指がタッチとして検知される圧力	5
FingerLow	指がリリースとして検知される圧力	5
IgnoreThumb	親指であるとわかるタッチを無視するか	False
IgnorePalm	手の平であるとわかるタッチを無視するか	False
DisableOnThumb	親指がさわっているとき全てのトラックパッドを無効にするか	False
DisableOnPalm	手の平ががさわっているとき全てのトラックパッドを無効にするか	False
ThumbRatio	親指の幅/長比率	70
ThumbSize	親指の最小限のサイズ	25
PalmSize	手の平の最小限のサイズ	10
BottomEdge	トラックパッドの無視する領域をパーセンテージで指定。	10
ButtonEnable	トラックパッドの物理ボタンを無効にするか	True
ClickFinger1	1 本指でのクリック動作	3
ClickFinger2	2 本指でのクリック動作	2
ClickFinger3	3 本指でのクリック動作	0
TapButton1	1 本指でのダブルタップ動作	1（ クリック&コピー）
TapButton2	2 本指でのダブルタップ動作	3（ ペースト）
TapButton3	3 本指のダブルタップ動作	2（ 右クリック）
TapButton4	4 本指でのダブルタップ動作	0（ 無効）
MaxTapTime	タップを認識する最大時間（ 値を小さくすると早くタップする必要がある。）	120
ScrollDistance	スクロールを有効にするために動かす距離	150
ScrollUpButton	2 本指でのスクロールアップ	4（ スクロールアップ）
ScrollDownButton	2 本指でのスクロールダウン	5（ スクロールダウン）
ScrollLeftButton	2 本指でのスクロールレフト	6（ スクロールレフト）
ScrollRightButton	2 本指でのスクロールライト	7（ スクロールライト）
SwipeDistance	スワイプを有効にするために動かす距離	700
SwipeUpButton	スワイプ（ 3 本指）アップ	8（ ALT + →/進む）
SwipeDownButton	スワイプ（ 3 本指）アップ	9（ ALT + ←/戻る）
SwipeLeftButton	スワイプ（ 3 本指）レフト	10（ 不明）
SwipeRightButton	スワイプ（ 3 本指）ライト	11（ 不明）
Swipe4Distance	スワイプを有効にするために動かす距離	700
Swipe4UpButton	スワイプ（ 4 本指）アップ	8
Swipe4DownButton	スワイプ（ 4 本指）ダウン	9
Swipe4LeftButton	スワイプ（ 4 本指）レフト	10
Swipe4RightButton	スワイプ（ 4 本指）ライト	11
AxisXInvert	X 軸を逆にするか	false
AxisXInvert	Y 軸を逆にするか	false

表 1　よく使う mtrack の設定項目

以下に有効な設定を載せておきます。

- トラックパッドのシングルタップを無効にする
  トラックパッドに触っても(シングルタップしても) 何も起きなくなります。

```
Option "TapButton1" "0"
Option "TapButton2" "0"
Option "TapButton3" "0"
```

- 2本指スクロールの動きをを OS X と同じにする

```
Option "ScrollUpButton" "5"
Option "ScrollDownButton" "4"
```

3.3.1 その他の情報

現在の mtrack ドライバは synaptics のように設定値を動的に変更できません。 xorg の設定を変更したい場合には、ファイルを変更し X サーバを再起動させる必要があります。これでは細かい設定等を行う時に大変なので常に設定を変更できるようにするためのパッチを作成し、アップストリームに送りました。 `https://github.com/BlueDragonX/xf86-input-mtrack/pull/41`

パッチを適用し、以下の設定を行なって X サーバを立ち上げると動的に設定を変更できるようになります。

```
Option "SHMConfig" "true"
```

Debian パッケージへの対応ですが、アップストリームで適用されれば更新したパッケージをアップロードする予定です。

肝心の設定用のツールですが、適当に作ったので後日公開します。

3.3.2 その他の情報(追記)

後日、上記のパッチを適用する必要はなくなり、 xinput コマンドで操作できることを教えてもらいました。

xinput は X の InputClass デバイスの設定を操作するためのツールです。 Debian では xinput パッケージで提供されています。実行すると、認識している InputClas 一覧を表示します。 Macbook Pro 等に搭載されているデバイスは **bcm5974** となりますので、これに対して設定を変更します。

```
$ xinput
Virtual core pointer                      id=2    [master pointer  (3)]
  Virtual core XTEST pointer                id=4    [slave  pointer  (2)]
  Logitech Unifying Device. Wireless PID:1028   id=10   [slave  pointer  (2)]
  bcm5974                                   id=13   [slave  pointer  (2)]
  Mouseemu virtual mouse                    id=15   [slave  pointer  (2)]
..... ( 省略)
```

設定可能項目を出力するには `xinput list-props id` を実行します。 id はデバイス一覧の横にある **id=**の数字を指定します。

```
$ input list-props 13
Device 'bcm5974':
Device Enabled (131):   1
Coordinate Transformation Matrix (133): 1.000000, 0.000000, 0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 0.000000, 1.000000
Device Accel Profile (261): 0
Device Accel Constant Deceleration (262):   1.000000
Device Accel Adaptive Deceleration (263):   1.000000
Device Accel Velocity Scaling (264):    10.000000
....(省略)
```

設定を変更する場合、 xinput コマンドのオプションである **set-int-prop** を使います。例えば、 **Trackpad Scroll Distance (303): 175** の値を 100 に変更する場合は値は数値 (int) なので、以下のように実行します。

```
$ xinput --set-prop --type=int --format=32 13 "Trackpad Scroll Distance" 100
$ xinput list-props 13
....
Trackpad Scroll Distance (303): 100
....
```

というわけで、今後は xinput を使って操作しましょう。

3.4 Debian でマルチタッチを使う場合にはどうしたらいいのか

大統一 Debian 勉強会での赤部さんの発表[*3] にもあったように、 Debian ではまだマルチタッチを提供するツール等が十分ではありません。 X や GTK+ などでは既にマルチタッチは対応していますが、それを使うアプリケーションがなく、 Ubuntu で採用されている utouch [*4] 関連のライブラリもまだパッケージになっていない状態です(ITP はされています）。よって Debian ではまだ iPad や Android タブレット相当の操作はできないと思われます。

3.5 まとめ

mtrack ドライバでマルチタッチの制御はできるようになっていますが、アプリケーションが追いついていないのが現状です。ユニバーサルオペレーティングシステムを目指す以上、マルチタッチは避けて通れない機能なので早くサポートされてほしいものです。

[*3] http://gum.debian.or.jp/download/debian-gum-presentation.akabe.pdf
[*4] https://wiki.ubuntu.com/Multitouch

4　レゴでなめこ収穫機を作ってみた

本庄 弘典

4.1　はじめに

PC 上で動作させた python スクリプトからレゴ マインドストームを操作し、 iPhone や Android で人気のアプリ『 なめこ栽培キット』の収穫機を作りました。

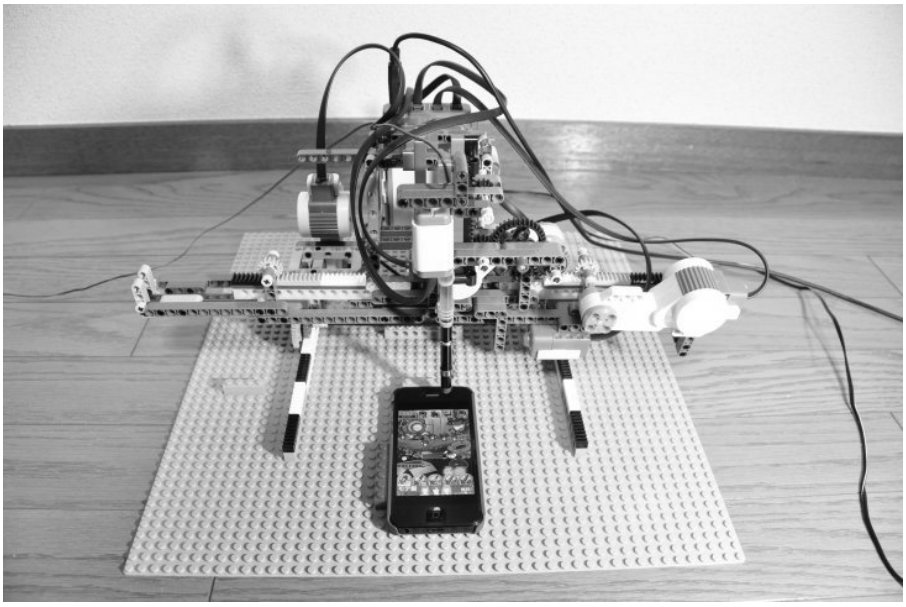


図 5　なめこロボ

4.2　『 なめこ栽培キット』とは

原木になめこの餌を注入し、生えてきたなめこを指でなぞって収穫する iPhone および Android のゲームです。株式会社ビーワークスからリリースされています。

- 正式名称は『 おさわり探偵なめこ栽培キット』。
- iPhone 版での最新作は『 おさわり探偵なめこ栽培キット Deluxe』。
- たまにレアなめこが生える。

4.3 レゴ マインドストームの紹介

株式会社レゴが販売している教育用ロボットで、次のような特徴を持っています。

- レゴブロックを使って組み立てられる
- レゴ テクニックシリーズのパーツを使用する
- モーターや各種センサーを制御できる
- ARM7 のコンピュータユニットから制御する
- ET ロボコンで使われている

4.3.1 レゴ マインドストームの購入方法

ET ロボコンの委員をやっている先生に購入方法を伺ったところ、「(株) アフレルから買ってください」と言われました。

- 教育用レゴ マインドストーム 正規代理店(株) アフレル
- `http://www.afrel.co.jp/`
- 現在価格 39,900 円

こちらでは ET ロボコン用のセット販売などを行っているようですが、特に ET ロボコンにこだわらない場合、Amazon.co.jp から現在価格 34,000 円で購入できます。

また一部のパーツは個別に購入することが可能です。筆者は次のお店を利用しました。

- ホビーショップ デジラ `http://www.dgla.jp/shop/`
- LEGO パーツ専門店 ハック フィン `http://huckfinn-lego.com/`
- レゴ パーツ販売ショップ「ブリッカーズ」 `http://www.brickers.jp/`

マインドストームにはいくつか種類がありますが、最新のものは『レゴ マインドストーム NXT 2.0』で、筆者もこちらを購入しました。

4.4 nxt-python

今回の目的はなめこ栽培キットの自動収穫機であるため、マインドストームを自立動作させる必要がありません。nxt-python は PC とマインドストーム NXT を USB や Bluetooth で接続し、PC 側で実行したスクリプトに従ってマインドストームをコントロールするための python モジュールです。

nxt-python は次の場所で公開されています。

- `http://code.google.com/p/nxt-python/`

Debian sid には python-nxt というパッケージがあり、こちらを導入することで nxt-python が利用できます。

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

4.4.1 サンプルコード

次のコードはポート B に接続されたモーターをパワー 100、360 度回転させるサンプルコードです。

```
#!/usr/bin/python

import nxt.locator
from nxt.motor import *

b = nxt.locator.find_one_brick()
m_left = Motor(b, PORT_B)
m_left.turn(100, 360)
```

なおドキュメントは見つからなかったため、ソースコードを読みながら使い方を調べました。

4.4.2 BaseMotor::turn(power, tacho_units)

turn() は power にパワーを、 tacho_units に角度を指定してモーターを回すメソッドです。

- 回りきるまで制御は帰ってこない。
- デフォルトでは回りきったところでブレーキをかける。
- 途中でモーターが回転しなくなると例外を飛ばす。
- power は-100 ～ 100 を指定可能。
- power に負の値を指定すると回転が逆になる。
- power に-10 ～ 10 くらいの値を指定しても力が弱くてモーターは回らない。

4.4.3 Motor::weak_turn(power, tacho_units)

turn() は無理矢理モーターを手で止めると怪我をするくらい強く回しますが、 weak_turn() はモーターを弱々しく回します。

- 手で止めると簡単にモーターは止まる。
- このとき例外は飛ばさない。
- 手で軽く回すと再び回り始める。
- tacho_units 分回ると制御が帰る。
- 使いどころが分からない。

4.4.4 Motor::run(power=100)

run() は指定されたパワーでモーターを回しつづけます。 run() を実行したのち、制御はすぐに戻ってきます。制御が戻ってきてもモーターは回りつづけます。

4.4.5 Motor::brake(), Motor::idle()

brake() および idle() はモーターを止めるメソッドです。 brake() はモーターを止めて動かないようブレーキをかけますが、 idle() は惰性で回りつづけ、緩やかに止まります。

4.4.6 Motor::get_tacho()

モーターは何度回ったかをカウントしており、 get_tacho() メソッドでこのカウンタを取得できます。 get_tacho() は次のの三つの値を返します。

- .tacho_count
- .block_tacho_count
- .rotation_count

4.4.7 Motor::reset_position(relative)

relative に True を渡すと.block_tacho_count を、 False を渡すと.rotation_count をリセットします。

4.4.8 Touch::is_pressed()

タッチセンサーが押されていれば True を、それ以外は False を返します。

4.4.9 その他

turn() メソッドはデフォルトの動作でブレーキをかけてくれるのですが、指定した数値どおりに動いてはくれません。しかし tacho は正確な値に見えるので、これを頼りに位置指定を行う関数を定義して利用しました。

```
def absturn(m, p):
    d = 1
    c = m.get_tacho().rotation_count
    q = p - c
    if q &lt; 0:
        d = -1
        q = abs(q)
    m.turn(30*d, q)
```

4.5 作成されたもの

実際に動かしているコードを gist で公開しました。

- https://gist.github.com/3897500

動画は YouTube で見られます。

- http://www.youtube.com/watch?v=uIEyWBs68c8
- 「 レゴ なめこ」で検索すると上の方に出てきます。

4.6 最後に

機械制御はなかなか思ったとおりに動いてくれないことが多く、モーターが正確な位置に動いてくれないなどの苦労に見舞われましたが、なんとか軽減することができました。

これを機会に皆さんもレゴ マインドストーム NXT の世界を覗いてみてはいかがでしょうか。

5 Android 携帯で Bluetooth Tethering

上川 純一

5.1 はじめに

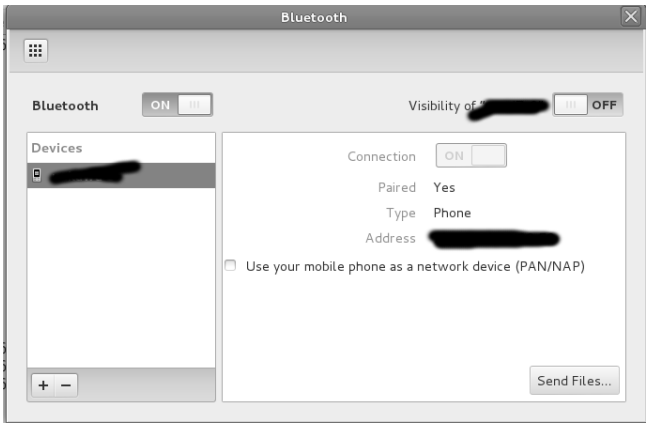
最近の Android 携帯もパソコンも Bluetooth に対応しています。そして Bluetooth profile である PAN や DUN に対応しているものも多いようです。 Android 4.0 あたりで対応するようになったと思われる Bluetooth Tethering を利用することで Android 携帯に Bluetooth PAN 経由で接続し、 Android 携帯の回線を利用して外部のネットワークに接続できるようになります。

WiFi Tethering のように電力消費が高すぎるので毎回設定でオフにする必要があるるということもなく、や USB Tethering のように毎回物理的に接続する必要もないので便利です。カバンの中に携帯をいれたまま接続できるので気楽ですよ。

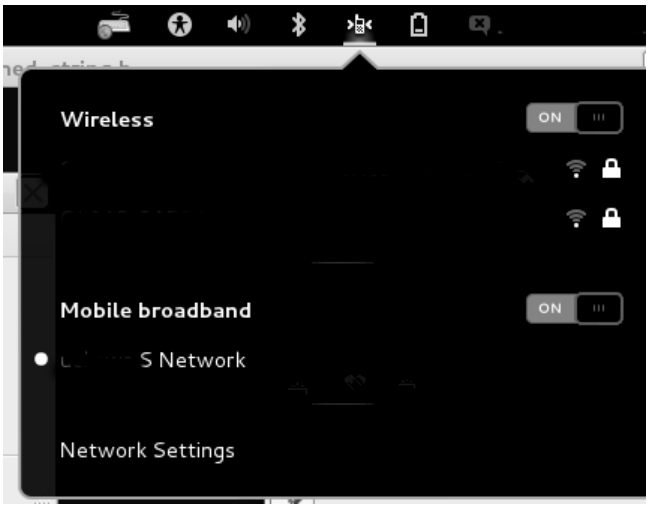
5.2 設定方法

Android 携帯側では Bluetooth Tethering をオンにします。

あと、 Bluetooth のペアリングを行います。携帯電話側の設定で可視状態にしておいて GNOME の設定で追加すればよいでしょう。 (図 5.2)



一旦設定しておくとネットワークの選択候補に Mobile Broadband というのが現れてそこで携帯電話の Bluetooth PAN 接続が選択できるようになります。



5.3 ネットワーク構成

Linux 側からは bnep0 デバイスとして見えます。複数マシンから接続するとそれぞれが別のサブネットに接続されるっぽいのでお互いに通信はできないようです。 WiFi Tethering だと同じサブネットにつながるので個人的にはウェブサーバーとクライアントを接続するためのハブとして便利に利用していたのですが、そういう使い方はできないようです。

```
bnep0     Link encap:Ethernet  HWaddr
          inet addr:192.168.46.43  Bcast:192.168.46.255  Mask:255.255.255.0
```

5.4 まとめ

Bluetooth PAN は便利、ということでした。ただ、なぜだか日本の携帯電話は Bluetooth PAN に対応してないものが多く、また海外携帯の日本モデルはその機能を削ってたりするものがあります[*5]。ここはぜひ Bluetooth PAN 対応じゃない携帯電話は購入しないことで消費者の意見を表明してください。

[*5] 例: 筆者の所有する Galaxy Nexus DoCoMo 版

6 perf でパフォーマンスチューニング

上川 純一

6.1 はじめに

最近のたいていの CPU には Performance Counters という仕組みがあり、特定のイベントが一定回数発生したらある処理をするという事ができるようになっています。それを利用してプロファイラーが実装できて、プログラムのパフォーマンスボトルネックの発見に役立てることができます。昔は oprofile を使っていたのですが、最近の Linux カーネルでは perf という仕組みを使うのが主流のようです。

カーネル側のサポートは標準で入っています。コマンドラインの perf コマンドは linux-base パッケージにはいっていてたいていの環境では標準でインストールされているようにみえるのですが実体はカーネルにあったバージョンを追加でインストールする必要があります。例えば、 linux 3.2 だと linux-tools-3.2 をインストールすることになります。

```
 $ uname -r
3.2.0-3-amd64
 $ sudo apt-get install linux-tools-3.2
```

デバッグシンボルがあると関数名とかがきれいに出る気がするので利用しているライブラリのデバッグシンボルもいれておくとよいでしょう。標準ライブラリはいれておきましょう。

```
 $ sudo apt-get install libc6-dbg libstdc++6-4.7-dbg
```

6.2 perf stat

プログラム単体の実行時間を計測してレポートしてくれるコマンドとして time コマンドがありますが、その代わりにつかえそうなツールとして、 perf stat があります。最近の CPU は負荷によって CPU 周波数が変わり、そういうシステムにおいてパフォーマンスの測定のために実行時間だけを計測するというのは適切ではないのですが、 perf stat はそれ以外に必要そうな値を計測してくれるので便利です。

```
$ perf stat ./apt-index-cmd debian_dists_sid_main_binary-amd64_Packages debian > /dev/null
 Performance counter stats for './apt-index-cmd debian_dists_sid_main_binary-amd64_Packages debian':

       1741.828818 task-clock                #    0.997 CPUs utilized
               165 context-switches          #    0.000 M/sec
                 6 CPU-migrations            #    0.000 M/sec
            27,392 page-faults               #    0.016 M/sec
     4,990,934,326 cycles                    #    2.865 GHz                      [83.29%]
     1,681,297,382 stalled-cycles-frontend   #   33.69% frontend cycles idle     [83.27%]
     1,096,373,883 stalled-cycles-backend    #   21.97% backend  cycles idle     [66.62%]
     7,738,965,303 instructions              #    1.55  insns per cycle
                                             #    0.22  stalled cycles per insn [83.51%]
     1,784,494,907 branches                  # 1024.495 M/sec                    [83.49%]
        32,701,183 branch-misses             #    1.83% of all branches         [83.32%]

       1.746581711 seconds time elapsed
```

6.3 perf record

perf record コマンドはプロファイル情報を記録する命令です。パラメータとして指定したコマンドをそのまま実行して、カレントディレクトリに perf.data ファイルを作成します。後に perf report などでそのプロファイルデータを確認することができます。

```
$ perf record ./apt-index-cmd debian_dists_sid_main_binary-amd64_Packages  debian > /dev/null
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.087 MB perf.data (~3800 samples) ]
$ ls -l perf.data
-rw------- 1 dancer dancer 93560 10 月 10 07:03 perf.data
```

perf record -g オプションをつけるとコールグラフ情報も記録するようです。

デフォルトは一秒 1000 サンプルとるようなので、プログラムの実行時間に応じてサンプル数を適当に調整しましょう。例えば全体で実行が一秒で終わってしまうプログラムの場合は-F 10000 をつけてみるとよりサンプル数がとれていいかもしれません。

6.3.1 gcc のコンパイルオプションの検討

gcc でソースコードコンパイルするときに、-g オプションをつけるとデバッグ情報がつきます。関数シンボルの情報とか行数とかソースコードとかが得られるのでこれは多分重要。

コールグラフがあまりないなぁと思ったら最適化のしすぎとスタックトレースのとりにくさを疑ってみましょう。

通常最適化オプション -O2 をつけてコンパイルしますが、 amd64 の場合、 gcc で-O2 をつけてコンパイルするとフレームポインタがなくなってスタックトレースを取得しにくくなっています。 libunwind 使えば取得できるはずですが、多分 perf はカーネル空間でスタックトレースをとっているのでそうなってないっぽいです。対策として若干オーバーヘッドがありますが、-fno-omit-frame-pointer を指定すると、フレームポインタを操作するコードを生成してくれます

C++ でファンクションオブジェクトとかテンプレートプログラミングしまくっていると、関数がほとんどインライン展開されてしまい、コールグラフに現れる部分が少なくなります。理解不能になってきたらたまに -O あたりでコンパイルしてスタックトレースを見てみましょう。

6.4 perf report

perf report コマンドは取得したプロファイルデータを可視化するコマンドです。テキストコンソールメニュー形式になっていて、気になるシンボルを選択してソースコードのアノテーションを見ることができます。どのソースコードの行に対応するアセンブラのどの命令で CPU 処理時間を費やしたのかを表示してくれます。

```
$ perf report
Events: 1K cycles
 18.65%  apt-index-cmd  apt-index-cmd        [.] available_parser::AptIndexSpiri
  7.38%  apt-index-cmd  libc-2.13.so         [.] _int_malloc
  6.64%  apt-index-cmd  libc-2.13.so         [.] malloc
  5.17%  apt-index-cmd  libstdc++.so.6.0.17  [.] std::string::_M_replace_aux(uns
  4.03%  apt-index-cmd  libc-2.13.so         [.] _int_free
  4.02%  apt-index-cmd  libstdc++.so.6.0.17  [.] __cxxabiv1::__vmi_class_type_in
  3.81%  apt-index-cmd  libstdc++.so.6.0.17  [.] std::string::_M_mutate(unsigned
  3.59%  apt-index-cmd  libstdc++.so.6.0.17  [.] std::ctype<char> const& std::us
  3.00%  apt-index-cmd  libc-2.13.so         [.] __strcmp_sse42
  2.74%  apt-index-cmd  apt-index-cmd        [.] boost::detail::function::functi
  2.72%  apt-index-cmd  libstdc++.so.6.0.17  [.] __dynamic_cast
  2.67%  apt-index-cmd  libc-2.13.so         [.] free
  2.46%  apt-index-cmd  libc-2.13.so         [.] __memcmp_sse4_1
```

ここでエンターをおすと次が表示され

```
available_parser::AptIndexSpirited::MakeIndex()::{lambda(std::vector<char, std::
    0.00 :          40368a:       je     4036ac <available_parser::AptIndexSpir
         :
         :          const std::string& get() const { return my_string_; }
         :
         :          // for 'map' comparison.
         :          bool operator<(const OrderedHashedString& b) const {
         :            if (ordered_hash_ == b.ordered_hash_) {
    2.87 :          40368c:       mov    0x28(%rbx),%rdx
         :              return my_string_ < b.my_string_;
         :            } else {
         :              return ordered_hash_ < b.ordered_hash_;
   34.96 :          403690:       cmp    %rbp,%rdx
    1.43 :          403693:       setb   %cl
         :
         :          const std::string& get() const { return my_string_; }
         :
         :          // for 'map' comparison.
         :          bool operator<(const OrderedHashedString& b) const {
         :            if (ordered_hash_ == b.ordered_hash_) {
```

6.4.1 C++ のコードの場合

C++ でテンプレートを活用しているコードを眺める場合、 perf report の TUI インタフェースだと関数名が十分表示されないなと悩むことになります。とりあえず TUI じゃないインタフェースにするともっと表示されますが、いまいち全体は表示できません。

たとえば関数名がこのように途中で切れてしまいます。まだ僕は回避方法を発見していません。

```
void
MeasureRaw<std::unordered_map<boost::iterator_range<__gnu_cxx::__normal_iterator<char
const*, std::string> >, int,
RangeHash<boost::iterator_range<__gnu_cxx::__normal_iterator<char
const*, std::string> > >,
RangeEqualTo<boost::iterator_range<__gnu_cxx::__normal_iterator<char
const*, std::string> > >,
std::allocator<std::pair<boost::iterator_range<__gnu_cxx::__normal_iterator<char
const*, std::string> > const, int> > >, __gnu_cxx::__normal_iterator
```

6.4.2 おわりに

一番基本的な Perf の使い方を紹介しました。 sudo perf list の出力を確認するとハードウェアのいろいろなカウンタだけでなくカーネルが提供しているさまざまなトレースポイントでのカウンタがあるのがわかります。どう使うか、夢は広がりますね。

参考文献

[1] perf(1) "perf – perfomance analysis tools for Linux" manual page. perf_3.2-report(1), etc.

[2] "Tutorial – Linux kernel profiling with perf", `https://perf.wiki.kernel.org/index.php/Tutorial`

7 systemd

岩松 信洋

7.1 はじめに

世の中の主要な Linux ディストリビューションは SysVinit の init scripts から他の init システムに移行しつつあります。 Fedora や Arch Linux が systemd に移行を始めたということもあり、一部で盛り上がっている（ 阿鼻叫喚ともいう）ようです。まさか いまだに SysVinit を使っている Debian 勉強会参加者がいるとは思えませんが、盛り上がっているようなので Debian と Systemd についてまとめてみました。

7.2 systemd とは?

RedHat に勤めている Lennart Poettering 氏によって開発されている init の代替プログラムです。実際には init の代替だけではなく、 Linux のサービス（ デーモン）管理フレームワークとなっています。今までの init システムの違いはサービスのプロセス管理を pid ではなく、 cgroups を使う点とサービスの起動をソケットとバスを使う点があります。これらは D-Bus を使って行います。これによってシステム立ち上げ処理をより並列的に行えるようになっています。また、System V スタイルと BSD スタイルの両方をサポートしています。

開発は freedesktop.org `http://cgit.freedesktop.org/systemd/systemd/` で行われており、開発は活発で週に一度はバージョンアップしています。最新バージョンは v195 となっています。

7.2.1 systemd の利点

systemd は SysVinit と比べて次のような利点があります。

- 設定が容易。
  SysVinit はシェルスクリプトで記述していたため、開発者によって書き方が異なります。よって設定や内容の理解が難しいことがあります。 systemd は設定方法や項目等が決まっているため、設定しやすくなっています。
- 起動が早い。
  シェルに依存していないのとデーモンが並列起動するため起動が早いです。
- カーネルモジュールの操作、セッション管理、ログ管理、ディスクの暗号化などを統合。

その他、作者による説明を `http://0pointer.de/blog/projects/why.html` から参照できます。

7.3 Debian で使う

systemd はもちろん Debian でも提供されており、 testing / unstable で v44 が利用できます。最新版とバージョンに差がありますが、アップストリームで頻繁にバージョンアップするのでバージョンはあまり問題ではありません。 v44 でも systemd を十分に使うことができます。

いまのところ Debian に関する情報は `http://wiki.debian.org/systemd` にまとまっていますが情報が少なく、内容も古いです。

7.3.1 インストール

先にも書いたように Debian では v44 が最新版です。 apt-get / aptitude でインストールできます。

また、 Linux カーネルは 2.6.39 以上、 devtmpfs, fanotify, autofs4, cgroups が有効になっている必要があります。

インストールは以下のように実行します。

```
$ sudo apt-get install systemd
```

以下のパッケージが依存関係でインストールされます。

```
libsystemd-daemon0
libsystemd-id128-0
libsystemd-journal0
libpam-systemd
```

次に ブートローダに init 指定を追加します。 grub を使っている場合、`/etc/default/grub` の `GRUB_CMDLINE_LINUX_DEFAULT` に **init=/lib/systemd/systemd** を追記します。

```
変更前:
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
変更後:
GRUB_CMDLINE_LINUX_DEFAULT="quiet init=/lib/systemd/systemd"
```

変更後、 **update-grub** を実行し、 grub に設定を反映します。そしてリブートします。設定が間違っていないければ systemd で立ち上がるはずです。

```
$ sudo update-grub
.....
$ sudo reboot
```

7.3.2 起動速度

systemd はアナライザをデフォルトでサポートしています。起動にかかった時間を確認するには `systemd-analyze` を実行します。また、画像で確認したい場合には `prop` オプションを指定して実行します。 SVG フォーマットで出力されるので、リダイレクトしてファイルに保存します。

```
$ systemd-analyze
Startup finished in 1831ms (kernel) + 5669ms (userspace) = 7500ms
$ systemd-analyze plot > systemd-boot.svg
```

試しに自分が常用している環境で起動時間を測定したところ、 SysVinit は約 15 秒、 systemd は約 10 秒でした。

7.4 用語

systemd を扱っていると専門用語が出てきますので、説明します。

- ユニット
  systemd ではデーモンなどの制御対象のことをユニットと呼びます。ユニットにはサービス、デバイス、マウントポイントなど、いくつかの種類があります。このユニットはテキストファイルで記述され、`/lib/systemd/system/` 以下に格納されています。各ユニットは拡張子を持ち、サービスの場合は`.service` となっています。
  mount, swap, automout は起動時に `/etc/fstab` から自動的にユニットを生成してくれます。
- ターゲット
  ターゲットとは SysVinit の runlevel 相当のものです。これはディストリビューションによって異なります。Debian の場合は以下のようになっています。
  この他に graphical.target と emergency.target があります。前者は X による起動を行うときに呼ばれるター

ユニットの種類	説明
service	デーモン
socket	ソケットによるデーモン
target	multi-user.target
device	udev で管理するデバイス
snapshot	ある時点の init の状態
timer	イベントから時間経過
path	監視するパス
mount	マウントポイント
swap	スワップ
automaount	自動マウントポイント

表 2　systemd で提供するユニット

run level	systemd のターゲット
0	poweroff.target
1	rescue.target
2 - 5	multi-user.target
6	reboot.target

表 3　run level とターゲットの対応

ゲット、後者は障害が起こった時に起動できるようにするためのターゲットです。ターゲットはカーネルのブートオプションに `systemd.unit=`で指定できます。何も指定しない場合は default.target が呼ばれるようになっています。

7.5　ユニットの操作方法

systemd に移行した後、デーモン等の制御は /etc/init.d/ 以下を実行するのではなく、**systemctl** コマンドを使って操作します。以下にユニットの操作方法について説明します。

7.5.1　起動しているユニットを表示する

起動しているユニットを表示するには sytemctl を実行します。

```
$ systemct
...
console-setup.service     loaded active exited        LSB: Set console font and
cron.service              loaded active running       LSB: Regular background pr
dbus.service              loaded active running       D-Bus System Message Bus
debian-fixup.service      loaded active exited        Various fixups to make sys
exim4.service             loaded active running       LSB: exim Mail Transport A
getty@tty1.service        loaded active running       Getty on tty1
ifup@eth0.service         loaded active exited        ifup for eth0
...
```

7.5.2　全てのユニットを表示する

操作できるユニットを表示するには `--all` を指定します。

```
$ systemctl --all
UNIT                       LOAD   ACTIVE   SUB       JOB DESCRIPTION
proc-sys...misc.automount loaded active   waiting       Arbitrary Executable Fil
dev-cdrom.device          loaded active   plugged       QEMU_DVD-ROM
dev-disk...QM00003.device loaded active   plugged       QEMU_DVD-ROM
dev-disk...QM00001.device loaded active   plugged       QEMU_HARDDISK
dev-disk...2dpart1.device loaded active   plugged       QEMU_HARDDISK
dev-disk...2dpart2.device loaded active   plugged       QEMU_HARDDISK
...
```

7.5.3 ユニットの状態を確認する

ユニットの状態を確認するには、 status オプションに確認したいユニット名を指定して実行します。

以下に rsyslog.service ユニットの状態を確認する例を示します。

```
$ systemctl status rsyslog.service
    Loaded: loaded (/lib/systemd/system/rsyslog.service; enabled)
    Active: active (running) since Wed, 14 Nov 2012 00:37:18 -0800; 22h ago
   Process: 474 ExecStartPre=/bin/systemctl stop systemd-kmsg-syslogd.service (code=exited, status=0/SUCCESS)
  Main PID: 483 (rsyslogd)
    CGroup: name=systemd:/system/rsyslog.service
            └ 483 /usr/sbin/rsyslogd -n -c5
```

これにより、このユニットは /lib/systemd/system/rsyslog.service によって Wed, 14 Nov 2012 00:37:18 -0800 に起動していることが分かります。

7.5.4 ユニットを起動する

起動していないユニットを起動するには、 start オプションにユニット名を指定して実行します。これは /etc/init.d/**サービス** start と同様の動きとなります。

```
$ sudo systemctl start ユニット名
```

7.5.5 ユニットを停止する

起動しているユニットを停止するには、 stop オプションにユニット名を指定して実行します。これは /etc/init.d/**サービス** stop と同様の動きとなります。

```
$ sudo systemctl stop ユニット名
```

7.5.6 ユニットの設定を再読み込みする

ユニットの設定を再読み込みするには、 daemon-reload オプションにユニット名を指定して実行します。

```
$ sudo systemctl daemon-reload ユニット名
```

実際に動いているデーモンの設定、例えば httpd の設定を再読み込みし、再起動するには reload オプションを使います。

7.5.7 ユニットの自動起動を有効にする

ユニットの自動起動を有効にするには enable オプションにユニット名を指定して実行します。

有効にすると /etc/systemd/system/**ターゲット**.wants/に/lib/systemd/system/にあるユニットへのシンボリックリンクが作成されます。どのターゲットで自動起動が有効になるかは、ユニットファイルの Install セクションで指定します。

```
$ sudo systemctl enable ユニット名
```

7.5.8 ユニットの自動起動を無効にする

ユニットの自動起動を無効にするには disable オプションにユニット名を指定して実行します。無効にすると、/etc/systemd/system/ターゲット.wants/にあるシンボリックリンクが削除されます。

```
$ sudo systemctl disabe ユニット名
```

7.5.9 ユニットの詳細を確認する

ユニットの詳細を確認するには show オプションにユニット名を指定して実行します。これにより指定したユニットと他のユニット、ターゲットの関係などが分かります。

```
$ sudo systemctl show rsyslog.service
Id=rsyslog.service
Names=syslog.service rsyslog.service
Requires=basic.target
Wants=syslog.socket
WantedBy=multi-user.target
Conflicts=shutdown.target
...
```

7.6 ユニットについて

ユニットには各ユニット間の依存関係を記述することができます。依存関係の指定として以下があります。

定義	説明
Before	そのユニットの後に起動されるべきユニット。
After	そのユニットの前に起動されるべきユニット。
Conflicts	同時に起動できないユニット。
Service	ソケットによる起動を行うユニット。
Sockets	ソケットによるユニットの起動を行う場合のソケット情報
Wants	同時に起動してほしいユニット。成功、失敗は関係ない。
Requires	同時に起動されなければならないユニット。ユニットの起動が失敗した場合は要求元も失敗する
BindTo	ユニットをグループとしてまとめる。

表 4　ユニットの依存定義

例えば、 default.target の内容は以下のようになっています。

```
[Unit]
Description=Graphical Interface
Requires=multi-user.target
After=multi-user.target
Conflicts=rescue.target
AllowIsolate=yes
```

このターゲットは multi-user.target と同時に起動され、 multi-user.target の後に起動します。また、 rescue.target と同時に起動できません。

7.7 まとめ

Debian でも問題なく systemd が利用できる環境が整っています。レガシーな SysVinit は捨て、新しい init の世界へ足を踏み入れてみてはいかがでしょうか。

7.8 参考文献

http://www.slideshare.net/moriwaka/systemd

8 Debian で作る LDAP サーバ

佐々木 洋平

8.1 はじめに

京都に来てから最初にやった業務は Solaris8 と NIS の撲滅でした。特に複雑な要件は無かったので OpenLDAP に移行して、ここ数年は順調に動作して (いると思って) います。

· · · で、気がついたら手元の仮想マシン群 (各種 Linux ディストリビューション、 FreeBSD、 Windows7 など) の認証も、ホスト OS(Debian) の OpenLDAP で行なうようになっていました。ラップトップで常に slapd を上げているのもなんかアレですが、気にしたら負けです。

とういわけで、ここでは「 Debian で OpenLDAP サーバ (slapd) を動かして、色んなユーザ認証を一本化するまで」のお話をしてみたいと思います。ちなみに、この企画を思いついた理由は「 ネット上にある多くのドキュメントが古い slapd について書かれていて、実際に運用しようとしたら色々とハマったから」だったんですが、先日倉敷さんに「 Ubuntu Server Guide 読んでねーのかよ (ry」とバッサリ言われてしまいました。 · · · で読んでみたら、記事を書く気力が · · ·。

気をとりなおして、先に進みます。

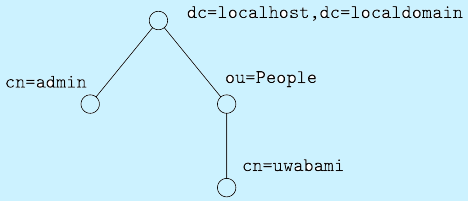
8.2 LDAP:Lightweight Directory Access Protocol

LDAP はディレクトリサービス (正確には X.500 モデルをサポートするディレクトリサービス) に接続するために使用するためのプロトコルの一つです。ここでの「 ディレクトリ」はファイルを管理する階層構造ではなくて、「 住所氏名録、人名簿」の意味でのディレクトリです。 X.500 データモデルはデータ構造 (X.500)、認証 (X.509)、分散処理 (X.518)、複製 (X.525) といった標準規格と、これらにアクセスするためのプロトコルである DAP: Directory Access Protocol があります。この DAP を軽量化し、 TCP/IP の上で実装したのが LDAP です。

8.3 DIT: Directory Information Tree

LDAP がアクセスするディレクトリデータは「 住所録・ 名簿」であり、「 人」や「 物」及びこれらに付随する情報 (パスワードやメールアドレスなど) を管理することに特化しています。これらの情報は頻繁に参照されることはあっても、更新はさほど行なわれません。この点は MySQL などのリレーショナルデータベースと異なる点です。

LDAP のアクセスするデータベースは「 木構造」になっています。この木構造を DIT と呼びます。 DIT の各ノードは「 エントリ」と呼ばれ、これらエントリには rdn: Relative Distinguished Name(相対識別名) が付いています。 rdn をつなげることで、エントリの dn:Distinguished Name(識別名) が一意に定まり、 DIT 中で識別されます。ひとつのエントリには「 属性名」と「 属性値」がペアとなって幾つか格納されています。属性名と属性値のテンプレートが objectClass です。図 (6) に、 DIT の例を示します。この例では、 localhost.localdomain 上で objectClass として person を用いて、アカウント名とパスワードによる認証情報が格納されています。




```
cn=uwabami,ou=People,dc=localhost,dc=localdomain
 objectClass:    person
 cn:             Youhei SASAKI
 sn:             SASAKI
 userPassword:   {SSHA}jP68h6yzJOlnRMda7IFIOLrTSe/lrFgO
 ...
```

図 6　DIT の例。この場合の rootdn は `dc=localhost,dc=localdomain` となり、一番右下のエントリの dn は `cn=uwabami,ou=People,dc=localhost,dc=localdomain` となる。

DIT 中のエントリにどの様な情報を格納するのか、というスキーマは objectClass の集りであり、代表的な用途に用いられるスキーマは既に提供されています。また、必要な情報を独自に定義して、新たに格納することも可能です。

8.4　OpenLDAP の導入と初期設定

LDAP サーバを導入して試してみます。 Debian では OpenLDAP サーバは `slapd` としてパッケージ化されていますので、これを導入します。とりあえずラップトップの母艦と仮想マシン上の複数のクライアントを想定しますので

```
rootdn:  dc=vmhost,dc=localdomain
```

で設定を始めてみます。

先ずは名前解決ができるようにしておきます。自前で DNS を上げているのであれば問題無いのですが、そうではない場合に備えて /etc/hosts あたりに

```
127.0.1.1     vmhost.localdomain vmhost
```

などと追記して `dig` や `nslookup` で名前解決ができることを確認しておきます. 続いてインストールです。

```
$ sudo -s
# apt-get install slapd ldap-utils
```

debconf の dialog が出てくるので適宜答えると良いでしょう。

1. Omit OpenLDAP server configuration? → No
2. DNS domain name: → vmhost.localdomain
3. Organization name: → localdomain
4. Administrator password: → 適宜
5. Database backend to use: BDB でも HDB でもお好きな方を. HDB は subtree の rename をサポートしている。
6. Do you want to the database to be removed when slapd is purged? → Yes
7. Allow LDAPv2 protocol? → No

この状態で、 LDAP のデータベースには二つの DIT が存在します。一つ目は管理用の DIT、二つ目はサービス用のデータを入れていく DIT です。それぞれ以下で確認できます。

```
$ sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b cn=config dn
dn: cn=config

dn: cn=module{0},cn=config

dn: cn=schema,cn=config

dn: cn={0}core,cn=schema,cn=config

dn: cn={1}cosine,cn=schema,cn=config

dn: cn={2}nis,cn=schema,cn=config

dn: cn={3}inetorgperson,cn=schema,cn=config

dn: olcBackend={0}hdb,cn=config

dn: olcDatabase={-1}frontend,cn=config

dn: olcDatabase={0}config,cn=config

dn: olcDatabase={1}hdb,cn=config

$ sudo ldapsearch -x -LLL -H ldap:/// -b dc=vmhost,dc=localdomain dn
dn: dc=vmhost,dc=localdomain

dn: cn=admin,dc=vmhost,dc=localdomain
```

では次に、ユーザ認証/管理ができるようにしてみます。/etc/passwd,shadow,groups 相当のスキーマとして、

- ou=People,dc=vmhost,dc=localdomain というエントリ以下にユーザを格納
- ou=Groups,dc=vmhost,dc=localdomain というエントリ以下にグループを格納
- cn=uwabami,ou=Groups,dc=vmhost,dc=localdomain というグループを追加
- cn=uwabami,ou=People,dc=vmhost,dc=localdomain というユーザを追加

する LDIF(LDAP Data Interchange Format) ファイルを書き, ldapadd で DIT に追加します。

```
$ cat add_People.ldif
#
# ユーザを格納するエントリの追加
#
dn: ou=People,dc=vmhost,dc=localdomain
objectClass: organizationalUnit
ou: People

$ cat add_Groups.ldif
#
# Groups を格納するエントリの追加
#
dn: ou=Groups,dc=vmhost,dc=localdomain
objectClass: organizationalUnit
ou: Groups

$ cat add_account_uwabami.ldif
dn: cn=uwabami,ou=Groups,dc=vmhost,dc=localdomain
objectClass: posixGroup
cn: uwabami
gidNumber: 3000

dn: uid=uwabami,ou=People,dc=vmhost,dc=localdomain
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: uwabami
sn: SASAKI
givenName: Youhei
cn: Youhei SASAKI
displayName: Youhei SASAKI
uidNumber: 3000
gidNumber: 3000
userPassword: {SSHA}jP68h6yzJOlnRMda7IFIOLrTSe/lrFgO
gecos: Youhei SASAKI, admin of this computer.
loginShell: /bin/zsh
homeDirectory: /home/uwabami
```

LDIF ファイルの形式は

- 属性名: 属性値
- 継続行の先頭は空白
- 行頭の # 以降はコメント扱い。空白行は無視

です。また、パスワードは slappasswd コマンドでハッシュにしておきます。

```
$ slappasswd -h {SSHA} -s OpenSeSaMi
{SSHA}jP68h6yzJOlnRMda7IFIOLrTSe/lrFgO
```

注意したいのは uid/gid の数値です。 Debian では一般ユーザは 1000 番台から始まります。ファイルで管理されている/追加されるユーザやグループと被らないように、ここでは 3000 番にしました。

LDIF ファイルを書いたら、この内容を実際に追加します。

```
$ ldapadd -x -D cn=admin,dc=vmhost,dc=localdomain -W -f add_People.ldif
Enter LDAP Password:
adding new entry "ou=People,dc=daphne,dc=localdomain"
...
```

と出れば成功です。 実際にユーザやグループ作成されたか検索してみます

```
$ ldapsearch -x -LLL -b dc=vmhost,dc=localdomain 'uid=uwabami' cn gidNumber
dn: uid=uwabami,ou=People,dc=vmguest,dc=localdomain
cn: Youhei SASAKI
gidNumber: 3000
```

どうやらうまく追加できたようです。

8.5 Linux クライアントのユーザ認証

単なるユーザ認証系として使用するのであれば、 Ubuntu (>=12.04) や Debian (squeeze 以降) で

- `libnss-ldapd`
- `libpam-ldapd`
- `nscld`

を導入し、 debconf の質問に適宜答えるだけでおしまいです.

```
$ sudo -s
# apt-get install libnss-ldapd libpam-ldapd nslcd
```

- LDAP サーバの URI →適宜
- LDAP サーバの検索を開始する DN → dc=vmhost,dc=localdomain
- nsswitch の更新: 設定する名前サービスとして passwd, group, shadow を選択

前半二つは nslcd の設定です。通常の debconf のダイアログでは、ユーザ認証に関する事柄を聞かれないので

```
$ sudo dpkg-reconfigure -plow nslcd
```

として、 rootbinddn や rootpw を設定しておきます。

`finger` や `id` などで、先程追加したユーザが見えることを確認した後、ログインを試してみる良いでしょう。

8.6 まとめ

というわけで、とりあえず OpenLDAP を用いてユーザ認証を統合するまで、について走り書きですがまとめてみました。実際には

- slapd のアクセス制限、 DIT の ACL とか STARTTLS
- インデックス生成によるデータベースの高速化や複製/多重化

なんて話題もありますので、これについては発表の時に補足します。次回はこれらに付け加えて Samba と連携した NT ドメインの管理なんかについてまとめてみるつもりです。

9 Debian ではじめる Kerberos 認証

倉敷 悟

9.1 事前課題の確認

今回の事前課題は、「 MIT 版 Kerberos のサービスを構成するプロセスと、そのプロセスが使用するポートを 2 つ以上挙げてください」でした。

正解は、 MIT 版 Kerberos の管理ガイド[*6]を参照してください。一応、 Kerberos 化されたサービスなどを省いた模範回答例は下記の通りです。

krb5kdc 88/tcp,udp
kadmind 749/tcp
kpropd 754/tcp

9.2 kerberos とは

1980 年代に MIT のアテナプロジェクト[*7]で (X Window System などとともに) 開発された、オープンなネットワークにおける認証システムです。もともと、 MIT のオープンキャンパスにおける課題を解決するために開発されているため、セキュアではない分散環境での動作が前提になっている他、シングル・サインオンもサポートされています。

開発初期に公開するうえで米国の暗号政策 (輸出禁止) を回避する必要があったため、 kerberos には複数の実装があります。ただ、現在では特に問題なく MIT の実装を利用することができますので、別実装のことはひとまずおいておきましょう。興味のある方は kth とか heimdal とかでググってみてください。

バージョン 5 以降では、 kerberos プロトコルとして RFC が発行されています[*8]。また、 Windows2000 以降の AD 認証でも利用されているので、気付かないうちに使っていた、という方もいるかもしれません。

9.2.1 kerberos の用語

レルム ある KDC が管理対象とするネットワークの範囲です。 DNS ドメインと対応させることが多いようです。 ActiveDirectory 的にはフォレストになります。

プリンシパル kerberos の管理対象となるあらゆる個別の要素です。ユーザ、ホスト、サービスなどが含まれます。次の書式の文字列で表現されます。「 名前/識別子@レルム」

鍵発行局 (KDC: Key Distribution Center) kerberos のキモとなるサービスです。管理レルムに所属している全プリンシパルの秘密鍵を保持しており、ユーザの認証 (AS) と、チケットの発行 (TGS) を行います。

[*6] http://web.mit.edu/kerberos/krb5-1.10/krb5-1.10.3/doc/krb5-admin.html#Configuring-Your-Firewall-to-Work-With-Kerberos-V5

[*7] 興味のある方は、「 MIT アテナプロジェクトのすべて」という書籍を読むと楽しめる思います

[*8] http://www.ipa.go.jp/security/rfc/RFC.html#10

チケット　ユーザからの要求に応じて鍵発行局で生成される、期限つきの認証データで、共有鍵で暗号化されています。

TGT (Ticket Granting Ticket)　ユーザが認証に成功した場合に発行されるチケットで、内部に一時的なセッション鍵を保持しています。これを持っているユーザのみ、サービスチケットの発行が許可されます。

鍵テーブル (Key Table)　Kerberos 認証を受けるサービスで必要となる、プリンシパルと鍵の一覧ファイルです。KDC 上でホストプリンシパルとサービスプリンシパルを作成し、それをファイルに出力してから該当のホストにコピーする必要があります。

Kerberos の仕組みや動作の詳細については、「3 分間 NetWorking」というサイト[*9]が語り口も軽く、わかりやすいと思いますので一度読んでみてください。20 分ほどで読めます。

9.3　構成の事前準備

kerberos では、ホストの名前解決ができること、時刻が同期していること、が前提条件として必要になります。ちょっと面倒ですが、確認しておいてください。

適当な仮想化機能を使った実験であれば、名前解決は dnsmasq パッケージを導入して /etc/hosts にさっくり書くのがお手軽で便利かと思います。

9.4　認証サーバの構成

9.4.1　openldap

ユーザ認証の観点では、kerberos はアカウント名とパスワードの組だけを扱いますので、ホームディレクトリやシェルが設定できません。そこで今回は、openldap をアカウントのメタデータを保管するデータベースとして使用します。openldap の構築と設定については、佐々木さんによる前回の関西 Debian 勉強会資料か、同じく佐々木さんによる次回の Debian 勉強会資料を参考にしてください。ここでは、すでに設定が完了しているものとして進めます。

/etc/passwd と /etc/shadow 相当のデータを保持するものとして、オブジェクトクラスは posixAccount と shadowAccount を利用します。ただし、実際には shadowAccount の属性は利用しません。/etc/shadow 相当の実データは kerberos で取り扱うのですが、それは OS から shadow 情報として見えるわけではないので、フェイクのアクセス先という位置付けです。

9.4.2　kerberos

まずは kerberos サービス用のパッケージをインストールしてみましょう。インストールするホストの名前を krbtest.my.domain とすると、次のような構成となります (openldap と kerberos は 1 台のサーバに同居させる想定)。

クライアント	LDAP	Kerberos	TLS	備考
レルム	MY.DOMAIN			
Kerberos (鍵発行局) サーバ	testnode.my.domain			
Kerberos 管理サーバ	testnode.my.domain			
openldap サーバ	testnode.my.domain			

```
unset DEBIAN_FRONTEND ※ pbuilder で実験している場合
apt-get install lv vim
apt-get install krb5-kdc krb5-admin-server
```

krb5-kdc が鍵発行局で、krb5-admin-server がアカウント情報の管理プロトコルを処理します。debconf がいくつか質問をしてくると思いますが、「Yes/No の質問は全てデフォルト通り」、「Default Kerberos version 5 realm」には名前解決で使っているドメイン名、「サーバアドレス」には名前解決できるホスト名、をそれぞれ入力してください。

[*9] http://www5e.biglobe.ne.jp/%257Eaji/3min/ex/sup04.html

kerberos 全般の設定ファイルは、/etc/krb5.conf になります。インストール時に debconf が自動で生成してくれるのですが、変更が不足しているので手編集が必要です。

```
vi /etc/krb5.conf
```

- (必須) [domain_realm] にドメインの設定を追加
- (オプション) [realms] から不要なものを消す
- (オプション) [domain_realm] から不要なものを消す

設定ファイルの編集が終わったら、 kerberos のデータベースを初期化しましょう。 debian には、このためのちょっとしたスクリプトが付属しています。

```
krb5_newrealm
```

root/admin のパスワードを求められますので、入力してください。これを忘れるとどうにもならなくなるので、忘れないように別途しっかり管理しておきましょう。この段階で、空のデータベースが作成され、 krb5-kdc (と kadmind) が起動され、各種プリンシパルの登録や変更ができるようになっています。

9.5 データの登録

9.5.1 kadmin.local

kerberos の管理作業に使うコマンドには、 kadmin、 kadmin.local、 kdb5_util などがあります。このうち、 kadmin.local は管理サーバ上でのみ利用できるものです。 kadmin を使うには kerberos でのユーザ認証をパスする必要がある、つまりプリンシパルを登録してからでないと使えないので、最初は kadmin.local で管理作業を行います。

kadmin.local を実行すると、プロンプトが出力されて管理コマンドの入力待ちになります。'?' でコマンドのリストが表示されますので眺めてみましょう。

```
kadmin.local:  ?
Available kadmin.local requests:

add_principal, addprinc, ank
                         Add principal
delete_principal, delprinc
                         Delete principal
modify_principal, modprinc
                         Modify principal
rename_principal, renprinc
                         Rename principal
change_password, cpw     Change password
get_principal, getprinc  Get principal
list_principals, listprincs, get_principals, getprincs
                         List principals
add_policy, addpol       Add policy
modify_policy, modpol    Modify policy
delete_policy, delpol    Delete policy
get_policy, getpol       Get policy
list_policies, listpols, get_policies, getpols
                         List policies
get_privs, getprivs      Get privileges
ktadd, xst               Add entry(s) to a keytab
ktremove, ktrem          Remove entry(s) from a keytab
lock                     Lock database exclusively (use with extreme caution!)
unlock                   Release exclusive database lock
purgekeys                Purge previously retained old keys from a principal
get_strings, getstrs     Show string attributes on a principal
set_string, setstr       Set a string attribute on a principal
del_string, delstr       Delete a string attribute on a principal
list_requests, lr, ?     List available requests.
quit, exit, q            Exit program.
```

とりあえず、ここではプリンシパルの登録と確認だけざっくり見てみることにします。

9.5.2 ユーザプリンシパル

ユーザの追加は簡単で、プリンシパルの名前としてユーザ名を指定するだけです。パスワードを求められますので、入力してください。

```
kadmin.local:  addprinc kdmtest
WARNING: no policy specified for kdmtest@MY.DOMAIN; defaulting to no policy
Enter password for principal "kdmtest@MY.DOMAIN":
Re-enter password for principal "kdmtest@MY.DOMAIN":
Principal "kdmtest@MY.DOMAIN" created.
```

9.5.3 ホストプリンシパル

ホストを追加する場合は、プリンシパルの名前として host、識別子としてそのホストの FQDN を指定します。また、ホストの場合はパスワードを人間が入力することはないので、手入力でなくランダムなものを自動生成させます。

```
kadmin.local:  addprinc -randkey host/testnode.my.domain
WARNING: no policy specified for host/testnode.my.domain@MY.DOMAIN; defaulting to no policy
Principal "host/testnode.my.domain@MY.DOMAIN" created.
```

9.5.4 サービスプリンシパル

サービスを追加する場合は、プリンシパルの名前としてサービス名、識別子としてサービスを提供するホストの FQDN を指定します。

```
kadmin.local:  addprinc -randkey ldap/testnode.my.domain
WARNING: no policy specified for ldap/testnode.my.domain@MY.DOMAIN; defaulting to no policy
Principal "ldap/testnode.my.domain@MY.DOMAIN" created.
```

9.5.5 登録データの確認

listprincs コマンドで、登録されているプリンシパルの一覧を表示することができます。初期状態では、次のように表示されるはずです。

```
kadmin.local:  listprincs
K/M@MY.DOMAIN
kadmin/admin@MY.DOMAIN
kadmin/changepw@MY.DOMAIN
kadmin/testnode.my.domain@MY.DOMAIN
krbtgt/MY.DOMAIN@MY.DOMAIN
```

9.5.6 鍵テーブルファイル (ktab) の生成

プリンシパル名を指定して ktadd コマンドを実行することで、鍵テーブルを作成/更新することができます。特定サービスホスト用の鍵テーブルファイルは、面倒ですが別途転送しておく必要がありますので、忘れないように注意しましょう。

```
kadmin.local:  ktadd ldap/testnode.my.domain
Entry for principal ldap/testnode.my.domain with kvno 2, encryption type aes256-cts-hmac-sha1-96 added to keytab
FILE:/etc/krb5.keytab.
Entry for principal ldap/testnode.my.domain with kvno 2, encryption type arcfour-hmac added to keytab FILE:/etc/krb5.keytab.
Entry for principal ldap/testnode.my.domain with kvno 2, encryption type des3-cbc-sha1 added to keytab FILE:/etc/krb5.keytab.
Entry for principal ldap/testnode.my.domain with kvno 2, encryption type des-cbc-crc added to keytab FILE:/etc/krb5.keytab.
```

9.6 クライアントの構成

NSS に参照させるライブラリには、いくつかの選択肢が存在します。

クライアント	LDAP	Kerberos	TLS	備考
lib(nss/pam)-ldap	○	×	未対応	古いが多機能
lib(nss/pam)-ldapd	○	×	対応	squeeze 以降の推奨。まだ発展途上
libpam-krb5	×	○	—	
sssd	○	○	必須	RedHat が FreeIPA のために開発している。まだ発展途上

Debian 的には、今回の構成であれば libnss-ldapd + libpam-krb5 が本命ということになるのかも知れませんが、ここでは試しに sssd を使ってみることにします。

sssd は、 RedHat がよくやる、あれこれ一緒くたにまとめる系の再実装の一つです。上記の ldap/krb 向けのクライアント機能の他、 NSCD の機能も持っています。

9.6.1 sssd の構成

sssd は RHEL5 から登場し、 RHEL6 でさらに統合が進んでいる他、日本語のマニュアルもきっちり用意されていますので、 RedHat を使用している人には今後の標準となっていくであろうサービスです。

Debian にもパッケージはあるのですが、当然ながら RHEL に収録されているものとはバージョンが異なりますので、RedHat のマニュアルをそのまま使えない場合があります。是非これをきっかけに Debian 上でアレコレ試してみて、ブログやこの勉強会のセッションでネタにしてみてください。

何はともあれ、 sssd をインストールしてサンプルの設定ファイルをコピーします。

```
apt-get install sssd
cp /usr/share/doc/sssd/examples/sssd-example.conf /etc/sssd/sssd.conf
```

設定ファイルは、のぞいてみれば一目でわかりますが、複数のセクションに分かれています。

sssd **セクション**　デーモンプロセス全体の動作設定です。

```
config_file_version = 2

# Number of times services should attempt to reconnect in the
# event of a crash or restart before they give up
reconnection_retries = 3

# If a back end is particularly slow you can raise this timeout here
sbus_timeout = 30
services = nss, pam

# SSSD will not start if you do not configure any domains.
# Add new domain configurations as [domain/<NAME>] sections, and
# then add the list of domains (in the order you want them to be
# queried) to the "domains" attribute below and uncomment it.
domains = KERBEROS
```

nss **セクション**　nss の参照設定を行います。

```
# The following prevents SSSD from searching for the root user/group in
# all domains (you can add here a comma-separated list of system accounts that
# are always going to be /etc/passwd users, or that you want to filter out).
filter_groups = root
filter_users = root
reconnection_retries = 3

# The entry_cache_timeout indicates the number of seconds to retain an
# entry in cache before it is considered stale and must block to refresh.
# The entry_cache_nowait_timeout indicates the number of seconds to
# wait before updating the cache out-of-band. (NSS requests will still
# be returned from cache until the full entry_cache_timeout). Setting this
# value to 0 turns this feature off (default).
; entry_cache_timeout = 600
; entry_cache_nowait_timeout = 300
```

pam **セクション**　pam の参照設定を行います。

```
reconnection_retries = 3
```

domain **セクション**　参照先サービス個別の設定を行います。

```
# Example LDAP domain
; [domain/LDAP]
; id_provider = ldap
; auth_provider = ldap
# ldap_schema can be set to "rfc2307", which stores group member names in the
# "memberuid" attribute, or to "rfc2307bis", which stores group member DNs in
# the "member" attribute. If you do not know this value, ask your LDAP
# administrator.
; ldap_schema = rfc2307
; ldap_uri = ldap://ldap.mydomain.org
; ldap_search_base = dc=mydomain,dc=org
# Note that enabling enumeration will have a moderate performance impact.
# Consequently, the default value for enumeration is FALSE.
# Refer to the sssd.conf man page for full details.
; enumerate = false
# Allow offline logins by locally storing password hashes (default: false).
; cache_credentials = true

# An example Active Directory domain. Please note that this configuration
# works for AD 2003R2 and AD 2008, because they use pretty much RFC2307bis
# compliant attribute names. To support UNIX clients with AD 2003 or older,
# you must install Microsoft Services For Unix and map LDAP attributes onto
# msSFU30* attribute names.
; [domain/AD]
; id_provider = ldap
; auth_provider = krb5
; chpass_provider = krb5
;
; ldap_uri = ldap://your.ad.example.com
; ldap_search_base = dc=example,dc=com
; ldap_schema = rfc2307bis
; ldap_sasl_mech = GSSAPI
; ldap_user_object_class = user
; ldap_group_object_class = group
; ldap_user_home_directory = unixHomeDirectory
; ldap_user_principal = userPrincipalName
; ldap_account_expire_policy = ad
; ldap_force_upper_case_realm = true
;
; krb5_server = your.ad.example.com
; krb5_realm = EXAMPLE.COM
```

domain セクションには、サンプルの設定雛形が用意されていますが、今回はひとまず無視して、次のように追記してみてください。

```
[domain/KERBEROS]
enumerate = true
id_provider = ldap
access_provider = permit
ldap_uri = ldaps://YOURSERVER/
ldap_search_base = dc=YOUR,dc=DOMAIN
ldap_tls_reqcert = never
ldap_tls_cacert = /etc/ssl/certs/YOUR-CACERT.pem

auth_provider = krb5
chpass_provider = krb5
krb5_kdcip = <kerberos サーバの IP アドレス>
krb5_server = <kerberos サーバの IP アドレス>
krb5_realm = <kerberos レルム名>
krb5_changepw_principal = kadmin/changepw
krb5_auth_timeout = 15
```

domain セクションで名前として指定したものが、 sssd セクションの domains で参照されていることを確認してください。設定ファイルの編集が終わったら、 sssd プロセスを再起動します。

```
service sssd restart
```

9.6.2 nss の切り替え

sssd の設定が終わったら、参照する NS (Name Service) を OS に教えてあげる必要があります。[*10]

```
vi /etc/nsswitch.conf
```

passwd と group と shadow の compat の後ろに sss を追加してください。

[*10] デフォルトでは、ローカルの /etc/passwd のみを探すようになっています

9.7 動作確認とその先

ssh を利用して、認証の動作確認をしてみます。実は ssh はもとから Kerberos に対応しているのですが、今回の構成例ではその機能は利用せず、認証部分は OS (sssd) に任せる形となります。 sshd_config で、 GSSAPI* が no になっていることと、 PasswordAuthentication が yes になっていること、 UsePam が Yes になっていること、を確認しておいてください。先ほど作成したユーザプリンシパルを使ってログインできれば成功です。

Kerberos サーバ上で試してもいまいち感動がないので、できれば複数台の構成で実験してみてみると面白いと思います。シングルサインオンで複数のサーバをわたり歩けるのもなかなか新鮮です。[*11]

今回は本当にさわりの部分だけ紹介してみましたが、自由研究のテーマとしては、「 Active Directory と連携させる」「 kerberos のデータストアを openldap にもたせてみる」「 wallet を使って keytab の運用をラクにしてみる」などが考えられますので、是非挑戦して、未来の勉強会で発表してみてください。

[*11] 最近は ssh の agent forwarding で済んでしまうかもしれませんが……

10 Debian でのC++11 開発環境

上川 純一

10.1 はじめに

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

10.2 サンプルコード

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

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

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

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

10.3 コンパイラー

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

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

10.3.1 GCC

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

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

10.3.2 clang

Clang も同様で、 C++11 モードを指定してあげる必要があります。

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

10.4 ドキュメント

10.4.1 C++11 標準ライブラリ

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

10.4.2 STL

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

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

10.4.3 GCC 独自拡張についてのドキュメント

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

10.4.4 その他のドキュメント

その他これがあればいいだろうというドキュメントを紹介します。

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

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

10.5 最後に

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

参考文献

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

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

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

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

11 clang によるパッケージビルド

かわだ てつたろう

11.1 はじめに

「 Build of the Debian archive with clang」 [1] という Debian のアーカイブを clang で再ビルドするプロジェクトが行なわれています。 clang でパッケージをビルドすることで何がうれしいのかなどを紹介します。

11.2 clang とは

clang[*12]は C/C++、 Objective C/C++ を対象としたコンパイラです。 LLVM(Low Level Virtual Machine)[*13]をバックエンドとして使用し LLVM の一部としてリリースされています。

clang のサイトに挙げられている機能、目的をいくつかみてみると

- コンパイルの高速化とメモリ使用低減
- 親切なメッセージ
- GCC 互換
- 高い規格準拠度
- BSD ライクなライセンス
- モジュール化されたライブラリ構成

といったところがあります。

現在、絶賛開発中なコンパイラです。

11.3 「 Build of the Debian archive with clang」の目標と状況

プロジェクトの目標は

- clang が現実的な選択肢であるか (そうでないか) を証明すること
- 異なるコンパイラを使ってビルドすることによって提供される異なったチェック、警告でソフトウェアのコードの品質を向上すること

が挙げられています。

[*12] http://clang.llvm.org/
[*13] http://llvm.org/

プロジェクトの成果として、 2012 年 6 月時点の状況は 17710 のパッケージが再ビルドされその内 2137 のパッケージ、全体の 12.1% がビルドに失敗しています。この結果には clang を用いたビルドに失敗したがノーマルな sid 環境でのビルドに成功したもの、つまり clang のバグと思われるものは含まれていません。

また、過去には clang2.9、 clang3.0 を用いてビルドされています。

clang Ver	日付	対象パッケージ数	失敗したパッケージ数	失敗したパッケージの割合
2.9	2011/09	16398	2372	14.5%
3.0	2012/01	15658	1381	8.8%
3.1	2012/06	17710	2137	12.1%

11.4 追試

プロジェクトのサイトにはビルド環境のセットアップコードが載せられています。これを用いることで簡単に試してみることができますので試してみましょう。

ただし、セットアップコードを見ていただければ分かりますが/usr/bin{g++,gcc,cpp}-VERSION を一旦削除して clang のシンボリックリンクに置き換えることになります。常用環境では実行しないほうがよいでしょう。

ここでは pbuilder を用いて追試してみました。

pbuilder には環境を変更するための手段として hook が用意されています。そこで、先のセットアップコードを hook として仕込み既存の pbuilder 環境を壊すことなく容易に clang ビルド環境のセットアップができるようにします。

```
$ mkdir ~/clanghook
$ cd ~/clanghook
$ cat << EOS > A10replaceclang
#!/bin/sh

echo "Install of clang"
#apt-get update
apt-get install --yes --no-install-recommends clang -t unstable

echo "Replace gcc, g++ & cpp by clang"
VERSION=4.7
cd /usr/bin
rm g++-$VERSION gcc-$VERSION cpp-$VERSION
ln -s clang++ g++-$VERSION
ln -s clang gcc-$VERSION
ln -s clang cpp-$VERSION
cd -

echo "Block the installation of new gcc version"
echo "gcc-4.6 hold"|dpkg --set-selections
echo "cpp-4.6 hold"|dpkg --set-selections
echo "g++-4.6 hold"|dpkg --set-selections
echo "gcc-4.7 hold"|dpkg --set-selections
echo "cpp-4.7 hold"|dpkg --set-selections
echo "g++-4.7 hold"|dpkg --set-selections

echo "Check if gcc, g++ & cpp are actually clang"
gcc --version|grep clang > /dev/null || exit 1
EOS
$ chmod +x A10replaceclang
$ ln -s A10replaceclang F10replaceclang
```

シンボリックリンクを作成することで login または execute 時にもセットアップされるようにしておきます。

これで準備が整いましたので後は pbuilder でパッケージをビルドする時に hook を指定すれば clang を使ってビルドされるようになります。

```
$ sudo pbuilder --build --hookdir ~/clanghook hello_2.8-2.dsc
```

いくつかのパッケージを試してみましたが確かに clang を用いたビルドの方がエラーチェックが厳しいようです。しかし gcc で出力されていた警告が clang では出力されなくなるケース[*14]もありました。この辺りはコンパイラへのオプショ

[*14] 例えば sl_3.03-17

ンによって変わってくるのかもしれません。

11.5 libc++

libc++ は clang と同じ LLVM のプロジェクトで C++11 をターゲットにした C++ 標準ライブラリです。新しい仕様である C++11 への対応は clang のみではなく標準ライブラリの対応も必要で、 clang による C++11 への対応といった場合は libc++ を用いた場合か libstdc++ へパッチを当てたものになるようです。

この libc++ を使うことはプロジェクト展望に「 libc++ を libstdc++ の代替として提供する」 [2] として含まれています。この試みは順調に進んでおり、 2012/07/31 には libc++ が experimental に入りました。まだ libc++ 自体に問題が多いようですがそれなりに使えています。

パッケージは i386 と amd64 版しか用意されていませんが、試してみようと思われる方は apt-line に experimental を追加し libc++-dev と libc++abi-dev をインストールしてください。

```
$ sudo apt-get -t experimental libc++-dev libc++abi-dev
```

そして、適当に C++ のコードを書いて、コンパイラへオプションで標準ライブラリに libc++ を指定してビルドします。出来上がった実行ファイルを ldd で表示してみると libstdc++ ではなく libc++ がリンクされていることが確認できます。 [3]

```
$ clang++ -stdlib=libc++ foo.cpp -o foo
$ ldd foo|grep c\\+\\+
ldd foo|grep c\\+\\+
  libc++.so.1 => /usr/lib/x86_64-linux-gnu/libc++.so.1 (0x00007f8b5ec60000)
```

```
$ g++ -nostdlib -lc++ -lc++abi -std=c++11 \
  /usr/lib/x86_64-linux-gnu/crt1.o \
  /usr/lib/x86_64-linux-gnu/crti.o \
  /usr/lib/x86_64-linux-gnu/crtn.o \
  -isystem /usr/include/c++/v1 -lc -lgcc_s \
  foo.cpp -o foo
$ ldd foo|grep c\\+\\+
ldd foo|grep c\\+\\+
  libc++.so.1 => /usr/lib/x86_64-linux-gnu/libc++.so.1 (0x00007f5098ce8000)
  libc++abi.so.1 => /usr/lib/x86_64-linux-gnu/libc++abi.so.1 (0x00007f5098a9a000)
```

さらに C++11 の機能を使ってやってみようと思う方は 8 月の東京エリア Debian 勉強会の資料を参考にして試してみてください。

参考文献

[1] Build of the Debian archive with clang, http://clang.debian.net/

[2] Provide an alternative to libstdc++ with libc++, http://wiki.debian.org/SummerOfCode2012/Projects#Provide_an_alternative_to_libstdc.2B-.2B-_with_libc.2B-.2B-

[3] Article complet: libc++: New C++ standard library in Debian, http://sylvestre.ledru.info/blog/sylvestre/2012/08/15/libc_new_c_standard_library_in_debian

12 Haskell の Debian packaging

日比野

この記事では、プログラミング言語 Haskell のパッケージと、そのパッケージがどのように依存関係解決されて Debian 化されているのかについて紹介します。

12.1 HackageDB

http://hackage.haskell.org/ は Haskell で書かれたライブラリやツールのパッケージ配布サイトで、 Perl での CPAN[*15]のようなものです。ここで配布されている Haskell のパッケージ群は HackageDB と呼ばれ、それぞれのパッケージはしばしば Hackage と呼ばれます。次のような URL でそれぞれの Hackage の情報を参照できます。

```
http://hackage.haskell.org/package/<hackage の名前>
```

各 Hackage のソースコードの tarball には、バージョン情報やライブラリ、ビルドツールの依存関係情報が含まれています。

language-objc パッケージの例:

```
Name:          language-objc
Version:       0.4.2.5
...
Library
...
    Build-Depends: base       >= 3 && < 5,
                   filepath   >= 1.1 && < 1.4,
                   process    == 1.1.*,
                   directory  >= 1.1 && < 1.3,
                   array      == 0.4.*,
                   containers >= 0.4     && < 0.6,
                   newtype    == 0.2.*,
                   pretty     == 1.1.*
...
    Build-Tools:    happy, alex
```

12.2 Cabal ライブラリと cabal-install

Cabal [*16] は依存関係にしたがってパッケージを Hackage のサイトから取ってきたり、パッケージのソースツリーと手元の環境からパッケージを build したりするライブラリです。

cabal-install[*17]は Cabal ライブラリをコマンドラインから呼び出すためのツールです。 Hackage の名前は cabal-install ですが、コマンドの名前は cabal です。簡単な利用方法は例えば以下のような感じになります。

[*15] http://www.cpan.org/
[*16] http://hackage.haskell.org/package/Cabal
[*17] http://hackage.haskell.org/package/cabal-install

```
$ cabal unpack language-objc ## hackage.haskell.org からの取得と展開
$ cd language-objc-0.4.2.5
$ cabal configure            ## 依存関係の検査
$ cabal build                ## コンパイル
$ cabal copy                 ## インストール
$ cabal register             ## インストール管理情報を処理系に登録
```

あるいは

```
$ cabal install language-objc  ## 再帰的にインストールを全て実行
```

Cabal ライブラリ, cabal-install のどちらも Haskell で書かれていて HackageDB に置かれています。現在の事実上標準の Haskell 処理系は Glasgow Haskell Compiler(GHC) ですが、 Cabal ライブラリは GHC のソースツリーに含まれる形で配布されています。通常の開発環境で利用されるのはこの含まれている Cabal ライブラリです。

12.3 debian ライブラリと cabal-debian

やはり Haskell で書かれているもので debian[*18] ライブラリと cabal-debian[*19] というツールがあります。

debian ライブラリは Debian ソースパッケージの情報をハンドリングするためのライブラリです。このライブラリを使って作られているのが cabal-debian で、次のように Hackage のソースを Debian のソースパッケージに変換することができます。

```
$ cabal unpack language-objc
$ cd language-objc-0.4.2.5
$ cabal-debian --debianize
```

次のようにうまく依存関係の情報が変換されます。

```
$ cat debian/control
Source: haskell-language-objc
Priority: extra
Section: haskell
...
Build-Depends: debhelper (>= 7.0),
               haskell-devscripts (>= 0.8),
               cdbs,
               ghc,
               ghc-prof,
               libghc-newtype-dev (>= 0.2) | libghc-newtype-dev (<< 0.3),
               libghc-newtype-prof (>= 0.2) | libghc-newtype-prof (<< 0.3),
               libghc-syb-dev (>= 0.3) | libghc-syb-dev (<< 0.4),
               libghc-syb-prof (>= 0.3) | libghc-syb-prof (<< 0.4),
               happy,
               alex
Build-Depends-Indep: ghc-doc,
                     libghc-newtype-doc (>= 0.2) | libghc-newtype-doc (<< 0.3),
                     libghc-syb-doc (>= 0.3) | libghc-syb-doc (<< 0.4)
...
```

ライブラリのパッケージだと、そのまま利用できる程度のものが自動的に生成されます。実行ファイルがあるものについては rules にインストール先を書く必要があります。

現在の Debian では libghc-¡hackage の名前¿-dev,prof,doc という名前で Debian パッケージが作られます。 libghc-*-dev は開発用ライブラリ、 libghc-*-prof はプロファイル情報取得用のライブラリ、 libghc-*-doc はライブラリドキュメントです。

Haskell の場合は haddock というツールでソースコード内の特定の形式のコメントがドキュメントに変換されます。libghc-*-doc はここから作られます。

12.4 haskell-devscripts

cabal-debian で作られた Debian のソースパッケージは cdbs 用の Makefile (hlibrary.mk) を利用するように構成されています。 hlibrary.mk は haskell-devscripts という Debian パッケージに含まれています。

[*18] http://hackage.haskell.org/package/debian
[*19] http://hackage.haskell.org/package/cabal-debian

```
$ cat debian/rules
#!/usr/bin/make -f
include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/hlibrary.mk

# How to install an extra file into the documentation package
#binary-fixup/libghc-language-objc-doc::
#        echo "Some informative text" > debian/libghc-language-objc-doc/usr/share/doc/libghc-language-objc-doc/AnExtraDocFile
```

hlibrary.mk から利用されてるコマンド群 (dh_haskell_*) も haskell-devscripts に含まれています。各コマンドは基本的には GHC のインストール情報管理用のファイル (debian/*/var/lib/ghc/package.conf.d/*.conf) を利用しつつ、*.substvars を出力します。

dh_haskell_depends　Haskell のライブラリの依存関係を Debian の依存関係に変換します。

dh_haskell_extra_depends　データのパッケージや実行プログラムのパッケージといった、ライブラリの依存関係では解決できない依存関係を Debian の依存関係に変換します。

dh_haskell_provides　Haskell のライブラリの Debian 仮想パッケージも含めた提供情報を計算します。

dh_haskell_shlibdeps　Haskell のライブラリが依存しているライブラリアーカイブ (*.a) を提供しているパッケージを検索し Debian の依存関係として出力します。shlibdeps なのにライブラリアーカイブ (*.a) のみ検索するのは、現状の Debian の GHC 環境だと、ライブラリパッケージを共有ライブラリにはコンパイルしていないからです。

12.5 おわりに

HackageDB の依存関係管理と Debian パッケージの関係について書いてみました。関連のツールもよくできているので、Debian 化も簡単にできそうに見えたことと思います。この機会に気になる Hackage を Debian 化してみてはどうでしょうか。

13 基本的なパッケージの作成方法手引書

岩松 信洋

13.1 本日の目的

Debian パッケージ化されていないソフトウェアをパッケージ化して、ビルドテストとパッケージの変更までを体験してみましょう。

13.2 本日の流れ

1. 作業を始める前の準備をする
2. ソフトウェアの動作確認をする
3. パッケージの雛形を作成する
4. debian ディレクトリ以下ファイルを編集する
5. パッケージをビルドする
6. 作成されたファイルを見る
7. パッケージをインストールする
8. パッケージをビルドテストする
9. パッケージのインストール/アンインストールテストをする
10. ソフトウェアの変更し、パッケージ化する
11. 質疑応答

13.2.1 記号の説明

$** が付いている場合は、コンソールからの入力を意味します。**$は入力せずにコマンドを入力してください。
コマンドラインやファイルの中身で \ が書かれている場所は行が続いている事を意味します。入力しないでください。
... は省略を意味します。実際には長い出力がある場合に省略している場合に利用しています。

13.3 ルート権限について

本ハンズオンでは、 root 権限を使った作業を行う場合があります。その場合には sudo コマンドを使って作業をします。 **sudo** コマンドが必要な場合にはコマンドラインの説明のところに **sudo** を指定しています。

13.4 Debian とは

省略

13.5 Debian パッケージについて

省略

13.6 作業を始める前の準備をする

13.6.1 パッケージメンテナ名の設定

パッケージメンテナの名前とメールアドレスを環境変数に設定します。適当なエディタを使って、 ~/**.bashrc** に以下の例のように追記して保存してください。各項目には自分の名前とメールアドレスをいれてください。

```
export DEBFULLNAME="Nobuhiro Iwamatsu"
export DEBEMAIL=iwamatsu@debian.org
```

保存できたら、ターミナルを起動し、

```
$ source ~/.bashrc
```

を実行してください。

13.6.2 パッケージビルドに必要なパッケージのインストール

パッケージビルドに必要なパッケージのインストールをします。 **packaging-dev** パッケージをインストールしてください。

```
$ sudo apt-get install packaging-dev
```

packaging-dev はメタパッケージ[*20]で、インストールすることによって Debian パッケージに必要なパッケージがインストールされます。

- build-essential
  パッケージ作成環境にインストールされていることが前提となっているパッケージを提供するメタパッケージです。gcc、 g++、 make 等を提供します。
- debhelper
  パッケージ作成補助ツールです。
- devscripts
  パッケージをメンテナンスするときに有用なスクリプトを提供します。
- dput または dupload
  パッケージのアップロードをする際に使用します。
- lintian
  パッケージを分析し、バグやポリシー違反を検出するツールです。
- pbuilder または cowbuilder または sbuild
  クリーンルームからパッケージを作成するための機能を提供します。
- quilt
  パッチ管理ツールです。

13.6.3 パッケージ化するソフトウェア

今回は、 **cwidget** を使ったサンプルプログラム **http://people.debian.org/ iwamatsu/dpd/hello-cwidget-20120922.tar.gz** を用意しました。このソフトウェアを Debian パッケージ化します。ダウンロードして、適当なディ

[*20] 他のパッケージに依存するだけのパッケージ

レクトリに展開します。

```
$ mkdir -p ~/debian/dojo/work
$ cd ~/debian/dojo/work
$ wget http://people.debian.org/~iwamatsu/dpd/hello-cwidget-20120922.tar.gz
$ tar -xzf hello-cwidget-20120922.tar.gz
$ ls
hello-cwidget-20120922.tar.gz
hello-cwidget-20120922
```

このソフトウェアは C++ で記述されており、コンパイルに必要なソフトウェアやライブラリがインストールされている場合には、**./configure ; make ; sudo make install** を実行することでコンパイルおよびインストールまでができるようになっています。

13.7 ソフトウェアの動作確認をする

13.7.1 ソフトウェアをコンパイルしてみる

コンパイルできない/動作しないプログラムをパッケージ化してもしょうがないので、動作確認をします。まずは最低限コンパイルに必要なパッケージをインストールする必要があります。それが **build-essential** パッケージです。これはパッケージ化の場合にも必要です。インストールしていない場合には以下のように実行してインストールしてください。

```
$ sudo apt-get install build-essential
```

先ほど解凍したディレクトリに移動します。移動した後、 **configure** を実行します。

```
$ cd hello-cwidget-20120922
$ ./configure
...
Alternatively, you may set the environment variables \
SIGC_CFLAGS
and SIGC_LIBS to avoid the need to call pkg-config.
See the pkg-config man page for more details.
...
```

実行するとエラーになります。ログを見てみると、 **pkg-config** コマンドがなくてエラーになっている事がわかります。さて、 **pkg-config** コマンドはどのパッケージで提供されているのでしょうか。

13.8 提供されているパッケージを探す

Debian で、特定のコマンドやファイルが提供されているパッケージを探すには、 **apt-file** コマンドを利用します。apt-file コマンドは **apt-file** パッケージで提供されているのでインストールしておきましょう。インストールしたら検索用のデータベースを構築します。

```
$ sudo apt-get update
$ sudo apt-get install apt-file
$ sudo apt-file update
```

今回は **pkg-config** コマンドがないので以下のように実行し、パッケージを検索します。

```
$ apt-file search pkg-config
pkg-config: /usr/bin/pkg-config
pkg-config: /usr/share/doc/pkg-config/AUTHORS
pkg-config: /usr/share/doc/pkg-config/NEWS.gz
.....
ruby-pkg-config: /usr/share/doc/ruby-pkg-config/copyright
zsh: /usr/share/zsh/functions/Completion/Unix/_pkg-config
zsh-beta: /usr/share/zsh-beta/functions/Completion/Unix/_pkg-config
```

また、検索探したいファイルのパスがわかっている場合には、パスを指定して検索できます(例: apt-file search /usr/bin/pkg-config) 。

実行すると、指定したファイルを提供しているパッケージ名が出力されます。結果を確認すると、 **pkg-config** パッケージで提供されていることが分かります。 **pkg-config** コマンドが提供されているパッケージが **pkg-config** パッケージとわかったので、インストールします。

```
$ sudo apt-get install pkg-config
```

再度 **configure** を実行してみましょう。

```
$ ./configure
...
No package 'sigc++-2.0' found

Consider adjusting the PKG_CONFIG_PATH environment variable if you
installed software in a non-standard prefix.
...
```

次は **sigc++-2.0** が見つからないようです。エラーメッセージを見ると pkg-config を使って **sigc++-2.0** の情報を取得しようとしているようですが、見つからないためエラーになっていることが分かります。[*21]

先ほどと同じように **apt-file** を利用して検索し、インストールします。

```
$ apt-file search sigc++-2.0.pc
libsigc++-2.0-dev: /usr/lib/x86_64-linux-gnu/pkgconfig/sigc++-2.0.pc
$ sudo apt-get install libsigc++-2.0-dev
```

再度 configure を実行します。

```
$ ./configure
...
checking for CWIDGET... configure: error: Package \
  requirements(cwidget) were not met:

No package 'cwidget' found

Consider adjusting the PKG_CONFIG_PATH environment variable if you
installed software in a non-standard prefix.
...
```

また、エラーになります。今度は **cwidget** 足りないようなので、再度検索してインストールします。

```
$ apt-file search cwidget.pc
libcwidget-dev: /usr/lib/pkgconfig/cwidget.pc
$ sudo apt-get install libcwidget-dev
```

```
$ ./configure
config.status: creating Makefile
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands
```

configure が正常に終了しました。終了すると、 **Makefile** が作成されています。 **make** を実行し、コンパイルします。

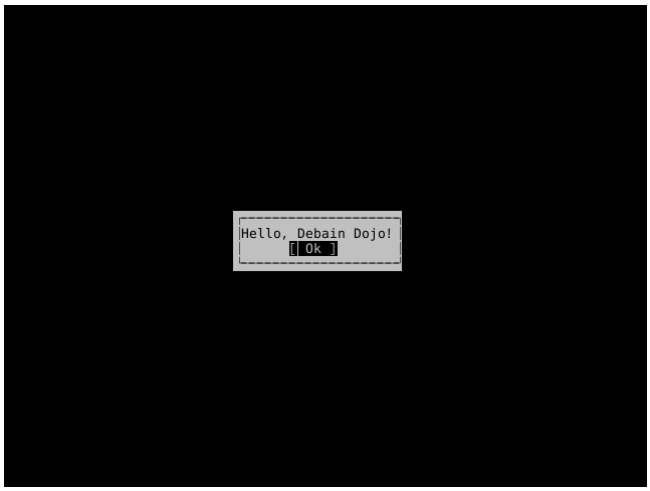
```
$ make
make  all-am
make[1]: ディレクトリ `/home/iwamatsu/debian/dojo/work/hello-cwidget-20120922' に入ります
g++ -DHAVE_CONFIG_H -I.     -I/usr/include/sigc++-2.0 -I/usr/lib/x86_64-linux-gnu/sigc++-2.0/include   \
-I/usr/include/sigc++-2.0 -I/usr/lib/x86_64-linux-gnu/sigc++-2.0/include -I/usr/lib/cwidget   -g -O2 -MT \
hello_cwidget-hello-cwidget.o -MD -MP -MF .deps/hello_cwidget-hello-cwidget.Tpo -c -o \
hello_cwidget-hello-cwidget.o `test -f 'hello-cwidget.cc' || echo './'`hello-cwidget.cc
mv -f .deps/hello_cwidget-hello-cwidget.Tpo .deps/hello_cwidget-hello-cwidget.Po
g++ -I/usr/include/sigc++-2.0 -I/usr/lib/x86_64-linux-gnu/sigc++-2.0/include   \
-I/usr/include/sigc++-2.0 -I/usr/lib/x86_64-linux-gnu/sigc++-2.0/include -I/usr/lib/cwidget   \
-g -O2   -o hello-cwidget hello_cwidget-hello-cwidget.o -lsigc-2.0   -lcwidget -lncursesw -lsigc-2.0
make[1]: ディレクトリ `/home/iwamatsu/debian/dojo/work/hello-cwidget-20120922' から出ます
```

コンパイルも正常に終了したので、試しに実行します。

```
$ ./hello-cwidget
```

図のような画面が表示されたでしょうか。

[*21] わからない人もいると思いますが。



ここまではサンプルプログラムの動作確認です。先にどのようなソフトウェアなのか理解するためにもパッケージング化する前にソースコード等を読んでおくことをお勧めします。

13.9 Debian パッケージの雛形を作成する

Debian パッケージはソースが格納されているディレクトリの中に **debian** ディレクトリを作成し、その中にパッケージビルド用のスクリプトを置き、実行することによってビルドされます。これらを構成するファイルやスクリプトはある程度決まっているため、雛形が用意されています。この雛形を作成するのが **dh_make** コマンドで、 **dh_make** は **dh-make** パッケージで提供されています。

今まで行った ./configure コマンド等で不要なファイルが作成されているため、作業の前に一度 **hello-cwidget-20120922** ディレクトリを削除します。そして再度展開し、作成されたディレクトリに移動します。

```
$ cd ..
$ rm -rf hello-cwidget-20120922
$ tar -xzf hello-cwidget-20120922.tar.gz
$ cd hello-cwidget-20120922
```

以下のコマンドを実行し、 **dh-make** パッケージをインストールします。

```
$ sudo apt-get install dh-make
```

雛形の作成は以下のコマンドを実行します。

```
$ dh_make --createorig -s
```

--createorig オプションはオリジナルソースコードの tar.gz イメージを構築します。 今回はシングルバイナリパッケージ(一つのソースコードから一つのバイナリパッケージがビルドされる）なので**-s** を指定します。実行すると以下のようなメッセージが表示されるので、 Enter キーを押します。

```
Maintainer name  : Nobuhiro Iwamatsu
Email-Address    : iwamatsu@debian.org
Date             : Wed, 12 Sep 2012 12:46:24 +0900
Package Name     : hello-cwidget
Version          : 20120922
License          : blank
Type of Package  : Single
Hit <enter> to confirm:
```

13.9.1 debian ディレクトリ

コマンドを実行すると、 **debian ディレクトリ**が作成され、この中にパッケージ作成に必要な雛形が作成されます。以下に作成されるファイル一覧を示します。

- README.Debian (Debian パッケージの README)

- README.source (ソースの情報を記述する)
- changelog (Debian パッケージのチェンジログ)
- compat (debhelper の API バージョンを指定する)
- control (Debian パッケージ情報)
- copyright (著作権情報)
- dirs (作成するディレクトリ名を指定する)
- docs (インストールするドキュメントファイルを指定する)
- emacsen-install.ex (emacs 用設定ファイル)
- emacsen-remove.ex (emacs 用設定ファイル)
- emacsen-startup.ex (emacs 用設定ファイル)
- hello-cwidget.cron.d.ex (cron 用)
- hello-cwidget.default.ex (debfonf 用)
- hello-cwidget.doc-base.EX (doc-base 用)
- init.d.ex (init.d を使うパッケージ用設定ファイル)
- manpage.1.ex (manpage の雛形)
- manpage.sgml.ex(manpage の雛形)
- manpage.xml.ex (manpage の雛形)
- menu.ex (メニューの雛形)
- postinst.ex (postinst メンテナファイルの雛形)
- postrm.ex (postrm メンテナファイルの雛形)
- preinst.ex (preinst メンテナファイルの雛形)
- prerm.ex (prerm メンテナファイルの雛形)
- rules (パッケージビルドスクリプト)
- source (Debian ソースパッケージ情報を格納するディレクトリ)
- format (Debian ソースフォーマットを指定する)
- watch.ex (アップストリームチェック用ファイル)

13.9.2 不要なファイルの削除

今回のパッケージ化に必要ではないファイルを **debian** ディレクトリ以下から削除します。hello-cwidget は emacs や cron を使わないプログラムなので、基本ファイルのみ(changelog、compat、control、copyright、rules、source) のみでよいでしょう。またサフィックスに .ex と.EX がついてるファイルを使用する場合、サフィックスを取り除いて内容を編集する必要があります(例: watch.ex → watch) 。[*22]

```
$ rm -rf debian/*.ex debian/*.EX debian/README.Debian debian/README.source debian/dirs debian/docs
```

13.10 debian ディレクトリ以下ファイルの編集

13.10.1 debian/changelog ファイルの編集する

Debian パッケージの変更は全て簡潔に Debian changelog ファイル(debian/changelog) に記載する必要があります。フォーマットに関しては Debian policy 4.4 Debian changelog: debian/changelog を参照してください。**debian/changelog** ファイルには既に **ITP**(Intent To Package) [*23] のテンプレート書かれているので削除します。以下のように変更します。

[*22] 必須ファイルは changelog、control、copyright、rules です。
[*23] http://www.debian.or.jp/community/devel/abbreviation.html

```
hello-cwidget (20120922-1) unstable; urgency=low

  * Initial release.

 -- Nobuhiro Iwamatsu <iwamatsu@debian.org>  Wed, 12 Sep 2012 12:46:24 +0900
```

13.10.2 ライセンスとコピーライトをチェックする

ソフトウェアを Debian パッケージにして Debian にインストールする際、重要な点としてソフトウェアのライセンスがあります。そのソフトウェアのライセンスが DFSG（ Debian Free Software Guideline）に適合するかチェックする必要があり、同梱されているファイルを確認する必要があります。ほとんどの場合、ソースには LICENCE ファイルや COPYING ファイルが提供されていますが、一部のファイルは違うライセンスが適用されている場合もあるためです。もちろんファイル毎にライセンスが書かれておらず、 LICENCE ファイル等で包容的にライセンスを決めている場合もあります。

Debian では簡易的にチェックするためのツールとして **licensecheck** があります。実行するとファイルのライセンスを出力します。**--copyright** オプションをつけた場合、コピーライトホルダも出力します。**-r** オプションは再帰チェックです。

```
$ licensecheck -r .
hello-cwidget.cc: BSD (2 clause)
$ licensecheck -r --copyright .
hello-cwidget.cc: BSD (2 clause)
  [Copyright: HOLDERS AND CONTRIBUTORS / 2012 Nobuhiro Iwamatsu <iwamatsu@debian.org>]
```

簡易的ではありますが hello-cwidget で提供されている hello-cwidget.cc のライセンスは **2 clause BSD License** で、コピーライトは **2012 Nobuhiro Iwamatsu <iwamatsu@debian.org>** が持っていることが分かりました。ツールに頼らず、実際にチェックもしておきましょう。

13.10.3 debian/copyright ファイルの編集する

各 Debian パッケージには、著作権と配布条件のライセンス文書が元のままの形式で /usr/share/doc/package/copyright に収録されていなければいけません（ Debian policy 4.5 Copyright: debian/copyright）。パッケージの著作権やライセンス情報を提供するファイルが debian/copyright ファイルとなります。以前はこのファイルのフォーマットは決まっていませんでしたが、今年の 2 月に DEP5（ Debian Enhancement Proposals 5 ）の Machine-readable debian/copyright が受理され、フォーマットが決まりました。フォーマットはヘッダ段落とファイル段落に分かれており、その中に項記述するが決まっています。

DEP5 に基づいて、前で確認したソフトウェアのライセンス用に debian/changelog を変更します。以下のような内容になります。

```
Format: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: hello-cwidget
Source: http://people.debian.org/~iwamatsu/dpd/

Files: *
Copyright: 2012 Nobuhiro Iwamatsu <iwamatsu@debian.org>
License: BSD-2-Clause license

Files: debian/*
Copyright: 2012 Nobuhiro Iwamatsu <iwamatsu@debian.org>
License: BSD-2-Clause license

License: BSD-2-Clause license
 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions are
 met:
 .
 * Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.
 .
 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
 BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
 GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

13.10.4 debian/rules ファイルの編集する

debian/rules にはパッケージのビルド手順を書きます。これはパッケージ作成補助ツールである debhelper を使って書くことが多いです。また、 **debhelper 7** になってからビルド手順が簡潔にかけるようになりました。**./configure ; make ; sudo make install** だけでコンパイルとインストールができるソフトウェアは以下の内容だけでパッケージのビルドができます。

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

%:
    dh $@
```

13.10.5 debian/control ファイルの編集する

debian/control ファイルにはパッケージ全体の情報とビルドされるパッケージの情報を書きます。（ Chapter 5 - Control files and their fields）まずパッケージ全体の情報を書いてみます。

- Source
  パッケージの元になるソースの名前を書きます。
- Section
  パッケージを分類したアプリケーション分野を指定します。
- Priority
  パッケージの優先度を指定します。
- Maintainer
  パッケージメンテナの名前とメールアドレスを書きます。
- Build-Depends パッケージを生成するときに利用するパッケージを指定します。 debhelper は一番利用されているパッケージ作成補助ツールです。
- Standards-Version
  パッケージが準拠している Debian ポリシーマニュアルのバージョンを指定します。現在の最新バージョンは 3.9.4 です。
- Homepage
  ソースが入手できる Web サイトを書きます。

```
Source: hello-cwidget
Section: devel
Priority: extra
Maintainer: Nobuhiro Iwamatsu <iwamatsu@debian.org>
Build-Depends: debhelper (>= 9.0.0)
Standards-Version: 3.9.4
Homepage: http://people.debian.org/~iwamatsu/dpd/
```

次にビルドされるパッケージの情報を書きます。 hello-cwidget では hello-cwidget というプログラムを提供するので、 hello-cwidget という一つのパッケージを提供するようにします。

各項目は以下のような意味です。

- Package
  パッケージ名を書きます。
- Architecture
  ビルド可能なマシンアーキテクチャを指定します。スクリプト言語や画像ファイルなど、アーキテクチャに依存しない場合は **all** を指定します。どのアーキテクチャでも動作するとは **any** を、特定のアーキテクチャのみで動作する場合はその Debian アーキテクチャを指定します。
- Depends
  依存しているパッケージを指定します。**$shlibs:Depends, $misc:Depends** はビルドされたファイルから自動的に依存ファイルを検出し、依存パッケージ名に置換されます。
- Description
  パッケージの説明を書きます。 1 行目は短い説明を書き、 2 行目以降により詳細な説明を書きます。 2 行目以降は先頭に 1 文字空白を入れる必要があります。

```
Package: hello-cwidget
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: Debian Packaging Hands-on sample program
 This is sample program of Debian Hands-on with OSC2009
 Tokyo/Spring and Debian Packaging Dojo.
 This is very easy program that uses cwidget.
```

13.10.6 パッケージをビルドする

パッケージのビルドには **debuild** コマンドを使います。 debuild コマンドは **devscripts** パッケージで提供されています。 **debuild -us -uc** を実行し、パッケージビルドをしてみましょう。ちなみに**-us** はソースパッケージに PGP 署名しない、**-uc** は .changes ファイルに PGP 署名しないというオプションです。

```
$ debuild -us -uc
...
dpkg-buildpackage: full upload (original source is included)
Now running lintian...
W: hello-cwidget source: newer-standards-version 3.9.4 (current is 3.9.3)
W: hello-cwidget: new-package-should-close-itp-bug
W: hello-cwidget: binary-without-manpage usr/bin/hello-cwidget
Finished running lintian.
```

パッケージのビルドが成功すると最後に lintian というパッケージチェックツールが実行されます。いくつか警告が出ていますので説明しておきます。

- newer-standards-version 3.9.4 (current is 3.9.3)
  Standart-Version フィールドの値が新しすぎるという警告です。 lintian が まだ 3.9.4 に対応していないためこの警告が出ます。
- new-package-should-close-itp-bug
  ITP のバグ番号が changelog ファイルに書かれていないという警告です。新しいパッケージを作成し、 Debian にインストールする場合には ITP（ Intent To Package）というバグを登録し、 Debian パッケージの changelog ファイルにこのバグ番号を書くことによってパッケージのアップロード時に対象のバグが閉じられます。他にも方法がありますが、この方法がよく使われます。

- binary-without-manpage usr/bin/hello
  usr/bin/hello の man ファイルがないという警告です (Debian-policy 12.1)。

13.11 作成されたファイルを確認する

debuild を実行した後には Debian パッケージだけでなく、いくつかファイルが作成されています。

```
$ ls ..
hello-cwidget-20120922
hello-cwidget_20120922-1.debian.tar.gz
hello-cwidget_20120922-1.dsc
hello-cwidget_20120922-1_amd64.build
hello-cwidget_20120922-1_amd64.changes
hello-cwidget_20120922-1_amd64.deb
hello-cwidget_20120922.orig.tar.gz
```

- *.debian.tar.gz
  Debian パッケージ用に修正したファイルをまとめたもの。 debian ディレクトリ以下のファイル。
- *.dsc
  Debian のソースパッケージを構成するための情報が書かれたファイル。
- *.orig.tar.gz
  開発元のソースコード。
- *.build
  ビルドログ。
- *.changes
  パッケージ作成後の情報が書かれたファイル。
- *.deb
  Debian パッケージ。

13.12 パッケージをインストールする

パッケージが無事ビルドできたら、実際にインストールしてみます。インストールには **debi** コマンドを使ってインストールします。インストールしたら、動作確認をしてみましょう。

```
$ sudo debi
$ which hello-cwidget
/usr/bin/hello-cwidget
$ hello-cwidget
```

13.13 パッケージのビルドテストをする

パッケージができた後はパッケージのテストを行います。パッケージのビルドテストには **pbuilder** を使います。pbuilder は chroot を使って Debian OS として必要な最低限の環境からパッケージビルドを行うツールです。これによって、パッケージビルドに必要なパッケージが漏れていないかチェックできます。また cowbuilder は chroot 環境を構築する時に copy-on-write を利用できるようにするツールを提供します。

13.13.1 pbuilder パッケージのインストール

```
$ sudo apt-get install pbuilder cowbuilder
```

インストールが完了したら、 pbuilder で cowbuilder を利用できるように、 pbuilder の設定ファイル（ ~/**.pbuilderrc**）に以下の内容を追記します。

```
PDEBUILD_PBUILDER=cowbuilder
```

13.13.2 pbuilder 環境の構築

ビルドテストを行う前に base システムイメージを構築する必要があります。以下のように実行します。

```
$ sudo cowbuilder --create
```

13.13.3 パッケージのビルドテスト

パッケージのビルドテストを行うには、対象とする Debian パッケージのソースが展開されたディレクトリで **pdebuild** を実行します。

```
$ pdebuild
...
```

13.13.4 pdebuild によるパッケージビルドエラー

実行するとビルドエラーになります。なぜエラーになるのでしょうか。考えてみましょう。

13.13.5 再ビルドテスト

エラーになる理由は先にインストールしたパッケージ **libcwidget-dev** をパッケージビルド時の依存関係を記述するフィールド **Build-Depends** に追加していないためです。追加してみましょう。

```
Source: hello-cwidget
Section: devel
Priority: extra
Maintainer: Nobuhiro Iwamatsu <iwamatsu@debian.org>
Build-Depends: debhelper (>= 9.0.0), libcwidget-dev
Standards-Version: 3.9.4
Homepage: http://people.debian.org/~iwamatsu/dpd/
```

追加したら **pdebuild** を実行し、再ビルドします。今度はビルドができるはずです。

```
$ pdebuild
...
```

13.14 パッケージのインストール/アンインストールテストをする

パッケージがビルドできただけでは喜んではいけません。インストール/アンインストールのテストも行いましょう。パッケージのインストール/アンインストールのテストには **piuparts** パッケージを使います。

13.14.1 piuparts のインストール

以下のように実行し、インストールします。

```
$ sudo apt-get install piuparts
```

13.14.2 パッケージのインストール/アンインストールテスト

piuparts も pbuilder と同様に最低限の環境からのインストールをチェックします。

```
$ sudo piuparts -d unstable ../hello-cwidget_20120922-1_amd64.deb
...
0m41.9s DEBUG: Removed directory tree at /tmp/tmpHliOKO
0m41.9s INFO: PASS: All tests.
0m41.9s INFO: piuparts run ends.
```

その他詳しい使い方は マニュアルを参照してください。

13.15 プログラムの編集をする

hello-cwidget を実行して、違和感のある方がおられたと思います。そう、 **Debian** が **Debian** になっていました。これはよくないので修正しましょう。 Debian source-format バージョン 3.0 からは **quilt** によるパッチシステムが標準で利用できるようになっており、これを使うためのツールが整備されています。

13.15.1 ファイルを修正する

早速ファイルを修正します。変更したい箇所は Typo なので grep 等で検索するとよいでしょう。

```
$ grep -r Debain *
hello-cwidget.cc:dialogs::ok(L"Hello, Debain Dojo!",
```

13.15.2 修正した箇所をパッチにする

修正した箇所をパッチするには **dpkg-source** コマンドに **--commit** オプションをつけて実行します。実行すると保存するファイル名を聞かれるので、適当な名前をつけて、エンターキーを押します。パッチにファイル変更内容等が書かれたファイルが表示されますので、この内容を適当に変更して保存します。パッチファイルのヘッダに書かれた情報は DEP3 （ Debian Enhancement Proposals 3 ）の Patch Tagging Guidelines に基づいています。保存すると debian/patches ディレクトリにパッチが保存され、 debian/patches/series ファイルに適用するパッチとして登録されます。

```
$ dpkg-source --commit
dpkg-source: info: local changes detected, the modified files are:
 hello-cwidget-20120922/hello-cwidget.cc
Enter the desired patch name: fix-typo
```

```
Description: <short summary of the patch>
 TODO: Put a short summary on the line above and replace this paragraph
 with a longer explanation of this change. Complete the meta-information
 with other relevant fields (see below for details). To make it easier, the
 information below has been extracted from the changelog. Adjust it or drop
 it.
 .
 hello-cwidget (20120922-1) unstable; urgency=low
 .
   * Initial release.
Author: Nobuhiro Iwamatsu <iwamatsu@debian.org>

---
The information above should follow the Patch Tagging Guidelines, please
checkout http://dep.debian.net/deps/dep3/ to learn about the format. Here
are templates for supplementary fields that you might want to add:

Origin: <vendor|upstream|other>, <url of original patch>
Bug: <url in upstream bugtracker>
Bug-Debian: http://bugs.debian.org/<bugnumber>
Bug-Ubuntu: https://launchpad.net/bugs/<bugnumber>
Forwarded: <no|not-needed|url proving that it has been forwarded>
Reviewed-By: <name and email of someone who approved the patch>
Last-Update: <YYYY-MM-DD>

--- hello-cwidget-20120922.orig/hello-cwidget.cc
+++ hello-cwidget-20120922/hello-cwidget.cc
@@ -8,7 +8,7 @@ int main(int argc, char **argv)
    toplevel::init();

    widgets::widget_ref dialog =
-       dialogs::ok(L"Hello, Debain Dojo!",
+       dialogs::ok(L"Hello, Debian Dojo!",
            util::arg(sigc::ptr_fun(toplevel::exitmain)));

    toplevel::settoplevel(dialog);
...

dpkg-source: info: local changes have been recorded in a new patch: hello-cwidget-20120922/debian/patches/fix-typo
```

各項目は以下のような意味を持ちます。

- Description
  パッチの説明を書きます。
- Origin
  パッチの提供者を書きます。また開発元や他の」サイトからパッチを持ってきているとき、その URL を書きます。
- Bug
  開発元の BTS 登録されているバグ番号がある場合に書きます。
- Bug-Debian
  Debian のバグ番号の URL を書きます。
- Bug-Ubuntu
  Ubuntu のバグ番号の URL を書きます。
- Forwarded
  バグの転送先、その必要の可否を書きます。
- Reviewed-By
  パッチのレビュアを書きます。
- Last-Update
  パッチの更新日を書きます。
- Applied-Upstream
  開発元で適用された/されている場合、そのソースを示す URL を書きます。

全て各必要はなく、状況に合わせて項目を埋めていきます。今回の場合は以下のようになります。

```
Description: Fixed message in dialog
 This patch is fixed typo form Debain to Debian.
Forwarded: not-needed
Origin: other
Author: Nobuhiro Iwamatsu <iwamatsu@debian.org>
Last-Update: 2012/09/22

--- hello-cwidget-20120922.orig/hello-cwidget.cc
+++ hello-cwidget-20120922/hello-cwidget.cc
@@ -8,7 +8,7 @@ int main(int argc, char **argv)
    toplevel::init();

    widgets::widget_ref dialog =
-       dialogs::ok(L''Hello, Debain Dojo!'',
+       dialogs::ok(L''Hello, Debian Dojo!'',
            util::arg(sigc::ptr_fun(toplevel::exitmain)));

    toplevel::settoplevel(dialog);
```

debian/patches/series ファイルを確認してみます。

```
$ cat debian/patches/series
fix-typo
```

13.15.3 差分を適用したパッケージをビルドする

差分を適用したパッケージをビルドするには通常のパッケージビルドと変わりません。**debuild** コマンドを使ってビルドします。debian/patches/series に書かれているパッチが順に適用され、パッケージがビルドされます。

```
$ debuild -us -uc
....
```

作成されたパッケージをインストールして動作確認してみましょう。Typo は治っているでしょうか。

```
$ debi
...
$ hello-cwidget
```

この後には **pbuilder** と **piuparts** を使ってパッケージのテストを行う事も忘れずに。

13.16 質疑応答

以上で、「基本的なパッケージの作成方法」は終了です。何か質問等はありますか?

14 最新のパッケージング事情の説明と使い方

岩松 信洋

14.1 debhelper 7 / 9

よく利用されるパッケージ作成補助ツールに debhelpler があります。 バージョン 7 から大幅な改良が行われました。これらについて簡単に紹介します。

14.1.1 各処理の隠蔽化

バージョン 7 から Debian パッケージの作成に必要な処理が隠蔽化され、シンプルに debian/rules ファイルが書けるようになりました。以前は図 7 のような書き方しかできなかったのですが、バージョン 7 以降は図 8 のような書き方ができるようになっています。

```
...
build: build-stamp
build-stamp:
    dh_testdir
    # Add here commands to compile the package.
    $(MAKE)

    touch build-stamp

clean:
    dh_testdir
    dh_testroot
    rm -f build-stamp install-stamp
....
```

図 7 バージョン 7 前の debian/rules

```
%:
    dh $@
```

図 8 バージョン 7 以降の debian/rules

パッケージをビルドすると分かりますが、バージョン 7 以降では各ターゲットで必要な処理に対応する debhelper スクリプトが呼ばれるようになっています(図 9）。

各ターゲット内や各 debhelper スクリプトが呼ばれる前に処理を行いたい場合には、オーバライド機能を使って各処理をオーバライドします。例えば、 **dh_auto_test** を行う前に **Foo** と出力したい場合には図 10 のように書きます。

```
$ debuild -us -uc
...
 dh_auto_test
   fakeroot debian/rules binary
 dh binary
    dh_testroot
    dh_prep
    dh_installdirs
    dh_auto_install
...
```

図 9　パッケージビルドログの例

```
%:
    dh $@

override_dh_auto_test:
    echo "Foo"
    dh_auto_test
```

図 10　オーバライドの例

14.1.2　サードパーティツール指定方法

debhelper ではパッケージを容易に作成できるツール（ dh_*）が提供されていますが、自分で debhelper のツールを作って、それを Debian パッケージの作成に利用することもできるようになっています。例えば、各プログラミング言語用向けに対応し debhepler ツールは各プログラミング言語のメンテナンスチームによって開発・提供されています。debhelper 7 以降でサードパーティツールを使うには、 debhelper を使う時にツール名を指定します。例えば、 ruby のパッケージを作成するには補助ツールである dh_ruby を使う（ 実際にはちょっと違うのだけど）には図 11 以下のようにします。

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

%:
    dh $@ --buildsystem=ruby --with ruby
```

図 11　ruby の場合

14.2　source 3.0

Debian 6.0（ Squeeze）から採用されている Debian ソースパッケージのフォーマット “3.0 (quilt)” について説明します。

まずは “3.0 (quilt)” の前に、いままで一般に使われてきたフォーマット (“1.0”) を簡単にまとめます。 “1.0” では、ソースパッケージは以下の 3 ファイルで構成されます。

- *packagename*`-`*upstreamversion*`.orig.tar.gz`
- *packagename*`-`*debianversion*`.diff.gz`
- *packagename*`-`*debianversion*`.dsc`

なお、正確には `1.0` は 2 種類あり、上の通常のパッケージのほかに “Debian native な” パッケージがあります。Debian native パッケージは次の 2 ファイルで構成されます。

- *packagename*`-`*version*`.tar.gz`
- *packagename*`-`*version*`.dsc`

ここで、`*.orig.tar.gz` には、通常上流の元のソースツリーが含まれます。`*.diff.gz` には、ソースパッケージから

パッケージなどをビルドするのに必要なスクリプトなどが入った debian/ ディレクトリや、上流のソースに対するパッケージメンテナの変更が含まれます。しかしこのファイル構成には

1. アーカイブの圧縮形式に gzip しか使えない
2. 複数のアーカイブで構成される上流のソースがそのまま扱えない
3. メンテナが当てたソースへのパッチが全部つながってしまっている
4. debian/ 以下にバイナリファイルが直接置けない

などの問題点があります。

そこで、さまざまな方法が検討されました。問題 1 は、これにより上流が bz2 で配布していても gz に圧縮し直さなければならないという問題がありました。*.orig.tar.gz の中身が上流のアーカイブの実体である、といった方法なども（ ちょっと無駄ですが...）使われてきました。この方法はビルド時にその tarball を展開して作業します。パッケージビルドサポートツールである CDBS（ Common Debian Build System）にもこの方法へのサポートがあります。

問題 2 は問題 1 と同様の方法で複数の tarball が入った*.orig.tar.gz を用意して対応していました。問題 3 は、当たっているパッチのそれぞれがどんな意図で行われたのかがわからない、ということ、また、 debian/以下のファイルも上流ソースへのパッチも一緒くたになってしまっていること、が問題でした。そこでまとまった意味のある単位に分割されたパッチをまず用意しておき、それらを debian/patches/ 下に配置し、その細かいパッチをビルド時に当てる/外すフレームワーク (patch system) が利用されています。これには dpatch や quilt などがあります。なお、この細かいパッチのそれぞれには、先頭にパッチの意図を説明する文章を記述することが推奨されています (http://dep.debian.net/deps/dep3/)。

問題 4 は、バイナリファイルの diff を取ろうとしても普通の patch ではできないことが原因なので、 uuencode などでテキストに落として patch を取る、といった手法が用いられてきました。

このような問題を解決するために新たなソースパッケージのフォーマットも検討されました。それが “3.0 (quilt)” フォーマット（ と “3.0 (native)” フォーマット）です。 3.0 (quilt) は次の 3 つ以上のファイルで構成されます。

- *packagename*-*upstreamversion*.orig.tar.*ext*
- *packagename*-*upstreamversion*.orig-*component*.tar.*ext*（ 任意）
- *packagename*-*debianversion*.debian.tar.*ext*
- *packagename*-*debianversion*.dsc

なお、 1.0 にあった Debian native パッケージに相当する 3.0 (native) は次の 2 つのファイルで構成されます。

- *packagename*-*version*.tar.*ext*
- *packagename*-*version*.dsc

ここで、まず tar の拡張子部分 *ext* に gz のほか、 bz2, lzma, xz が利用できるようになりました。これにより問題 1 が解決されました（ 3.0 (native) における主な変更点はこれです）。また、 *component* の部分を適当に変えることにより複数の tarball をきちんと扱えるようになりました。これが問題 2 を解決します。次に、 debian/下のファイルはすべて *.debian.tar.gz に入れることになりました。これですべてが混ざった状態はなくなりました。さらに、 debian/patches/ 下のパッチが、パッチシステム quilt と基本的に同じ方法で dpkg-source(1) によって “ソースパッケージの展開時に” 自動的に当たるようになりました。これにより、ビルド時にパッチを当てるように debian/rules ファイルを記述する必要はなくなりましたし、 debian/control ファイルに Build-Depends: quilt などと書く必要もなくなりました。これらによって問題 3 は解決されました。問題 4 については、 debian/下のファイルを diff として保持することはもはやなくなったので解決し、*.debian.tar.*ext* に直にバイナリファイルを配置できます。

このように Debian のソースファイルフォーマットは新しいバージョンに移行しています。今後は “3.0 (quilt)”、 “3.0 (native)” を使うようにしましょう。

14.3 hardening

次期リリース Debian 7.0 から **Security hardening build flags** を有効にしたパッケージが提供されるようになります。これはパッケージ構築時にセキュリティを強化するコンパイルフラグを (デフォルトで) 有効にするというものです。現在、以下の 4 点を有効にする必要があります。

- Format string checks（ -Wformat -Werror=format-security ）
  format 使う関数（ 例えば printf）の使用が問題を引き起こす可能性がある場合に警告する。
- FORTIFY_SOURCE
  文字列やメモリの操作を行う関数を使用する際にバッファオーバーフローを検出する。
- -fstack-protector --param=ssp-buffer-size=4
  スタック破壊攻撃等によるバッファオーバーフローをチェックするための追加コードを生成する。 4 バイトを超える配列を持つ関数を対象にする。
- -z,now,-z,relro
  リロケーション領域 (GOT など) をリードオンリーにする。

これらのコンパイルオプションを自動的に CFLAGS 変数等に設定する機構は今の所なく、 debian/rules に記述する必要があります。いくつか方法がありますが、よく使われるのは **dpkg-buildflags** で Debian で推奨されるコンパイルオプションを取得し、各変数に設定するというものです。図 12 に例を示します。

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

CPPFLAGS:=$(shell dpkg-buildflags --get CPPFLAGS)
CFLAGS:=$(shell dpkg-buildflags --get CFLAGS)
CXXFLAGS:=$(shell dpkg-buildflags --get CXXFLAGS)
LDFLAGS:=$(shell dpkg-buildflags --get LDFLAGS)

%:
    dh $@
```

図 12　hardening 設定方法

その他の方法は `http://wiki.debian.org/Hardening` を参照してください。

また hardening が有効なバイナリになっているか簡易的にチェックするためのツール **hardening-check**（ hardening-includes パッケージで提供されています）、 **blhc** があります。前者はバイナリからチェック、後者はビルドログからチェックするという違いがあります。 blhc によるチェック結果は buildlogcheck[*24] から参照できます。

14.4 パッケージのスポンサーアップロード

Debian Deloper 以外のパッケージアップロード権限をもってない人はアップロード権限を持っている Debian Developer にパッケージをアップロードしてもらう必要があります。（ Debian Maintainer はアップロードできるが、制限がある。）このような場合パッケージをどこかに置いて、パッケージをチェックしてもらった後で Debian にアップロードされます。チェックしてもらうパッケージを置いたり、このチェックの課程や機械的にできるチェックを行えるサービス **mentors.debian.net** ができました。アップロード権限がない人はこのサービスを使うようにしましょう。

また、代わりにパッケージをアップロードしてくれる人をスポンサーと言うのですが、スポンサーがいない場合、自分で探してアップロードしてもらう必要がありました。しかし状況がトラッキングされていないので、どのパッケージがまだアップロードされていないとか、誰がアップロード担当になっていないのかわからない状態になっていました。現在は **sponsorship-requests** として、 BTS で管理することが推奨[*25] されるようになりました。今後は **sponsorship-requests** を行うようにしましょう。

[*24] `https://buildd.debian.org/~brlink/bytag/W-compiler-flags-hidden.html`
[*25] http://lists.debian.org/debian-mentors/2012/01/msg00578.html

15 月刊 Debhelper dh_makeshlibs,dh_shlibdeps

野島 貴英

15.1 今回のコマンド

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

- dh_makeshlibs
- dh_shlibdeps

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

15.2 予備知識

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

15.3 呼び出し方と動作

以下のように順に呼び出して利用します。

```
$ dh_makeshlibs
$ dh_shlibdeps
```

15.3.1 dh_makeshlibs

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

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

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

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

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

15.3.2 dh_shlibdeps

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

このコマンドは呼び出されると以下の動作を行います。

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

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

15.4 ${shlibs:Depends}、 ${shlibs:Recommends} マクロ

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

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

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

15.5 shlibs ファイルと、 symbol ファイル

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

15.5.1 shlibs ファイル

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

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

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

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

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

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

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

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

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

15.5.2 symbol ファイル

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

```
ファイル形式:
<soname> <main dependency template>
[| <alternative dependency template>]
[ as many alternative dependency templates as needed ]
    <symbol> <first-version>[ <id of dependency template>]
    [ as many symbols as needed ]

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

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

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

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

という事を検討します。もし、こちらが出来ると、例えば sid から testing へバイナリパッケージを移動させるときに、依存関係にある数多くのライブラリ全部も同時に sid から testing へ完全に移動できるようになるまで、バイナリ本体を testing のものとして移動出来ないという問題を極力避ける事ができます。 [6] (図 13 参照)

15.6 substvars ファイル

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

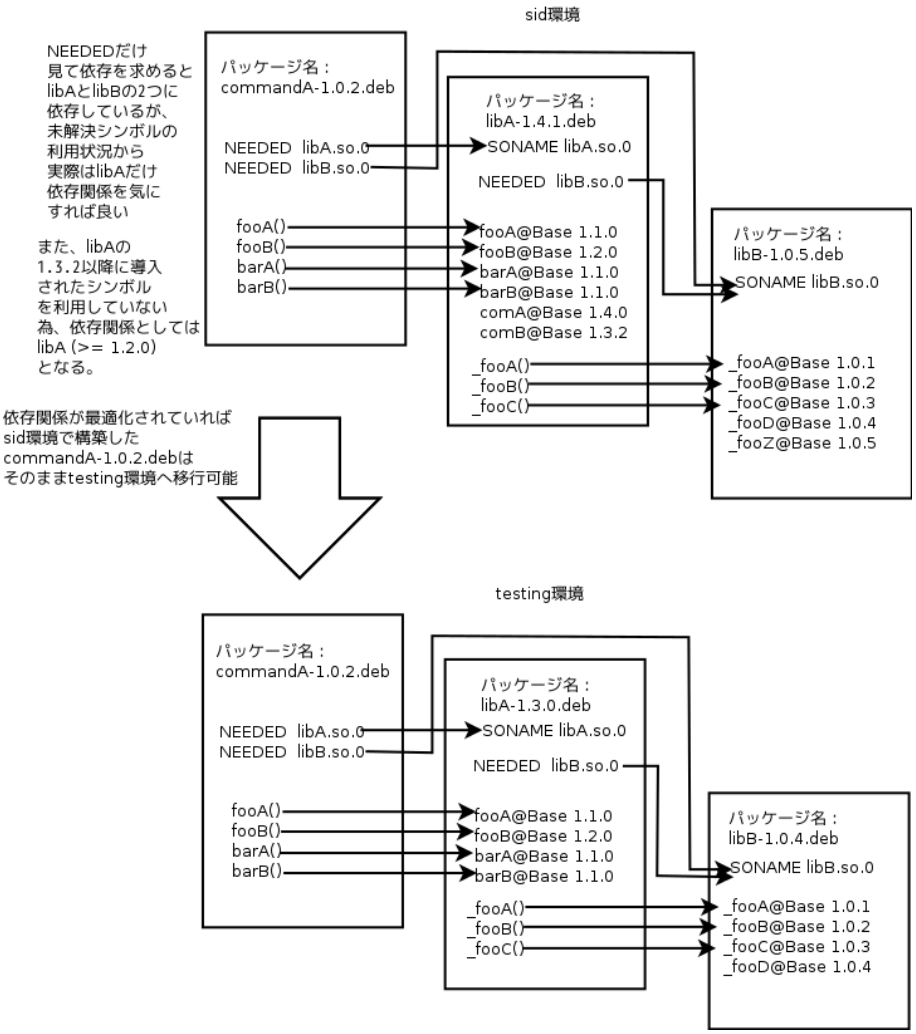


図 13　symbol ファイルを利用した依存関係算出

15.7　内部で主に利用されている dpkg パッケージに含まれるコマンド

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

15.7.1　dpkg-gensymbols

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

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

15.7.2 dpkg-shlibdeps

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

dpkg-shlibdeps の動作は以下のとおりです。

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

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

コマンドを使って探ります。

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

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

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

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

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

参考: Step 5. の動作をどのように行っているかを観察するには、

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

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

15.7.3 おわりに

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

参考文献

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

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

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

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

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

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

16 月刊 Debian Policy 第5回「 ソースパッケージ」

甲斐 正三

16.1 Debian ポリシーとは

1. Debian ディストリビューションに要求されるいくつかの必要条件。
2. Debian アーカイブの構成と内容、オペレーティングシステムとしての Debian の設計に関するいくつかの事項
3. それぞれのパッケージがディストリビューションに受け入れられるために満たさなければならない技術的な必要条件。

16.2 ソースパッケージとは

1. Debian バイナリパッケージの元になるパッケージである。
   (例) シェルスクリプトの bash は bash というソースパッケージからビルドされる。
2. 一つのソースパッケージから複数のソースパッケージがビルドされることがある。
   (例) bash ソースパッケージは bash バイナリパッケージ、 bash-builtins、及び bash-doc パッケージもビルドされる。

16.3 ソースパッケージにおけるデビアンポリシーとは

上記 1 と 2 から、
「 Debian GNU/Linux のソースパッケージの内部構成や、 Debian GNU/Linux として必要なソースパッケージの設計指針についてまとめられたものである」
といえるのではないでしょうか。

16.4 ソースパッケージに対するデビアンポリシーの実際

それでは、「 Debian ポリシーマニュアル Chapter4 - ソースパッケージ」を見ながら、各項目を外観していくことにします。以下の項目番号はマニュアルのそれに対応させています。なお、バージョン 3.9.1.0 以降の更新された版については言及しません。

- 4.01 基準への準拠
  1. 最新のパッケージング基準のバージョンを指定する。
  2. Standards-Version コントロールフィールドで指定。
     (ソースパッケージに含まれる'.dsc' ファイル中)

3. 最新版を次の URL でチェックすること。
   `http://www.debian.org/doc/debian-policy/ch-scope.html#s1.2`
4. 2012 年 7 月 21 日現在の最新バージョン:
   'version 3.9.3.1, 2012-03-04'

- 4.02 パッケージ同士の依存関係
  1. ビルド時に必要なバイナリパッケージを明記すること。
     (ソースパッケージに含まれる'.dsc' ファイル中)
  2. ビルド時にインストールされていてはいけないパッケージも明記のこと。
  3. ただし build-essential パッケージは記載する必要は無い。
  4. 以下のパッケージのみで構成された環境でビルドできなければならない。
     essential パッケージ、
     build-essential パッケージ、
     依存関係を満たすパッケージ。

'build-essential パッケージ' とは何か見てみよう。

`http://packages.debian.org/ja/sid/amd64/build-essential/download`

```
# apt-get install build-essential && view /usr/share/doc/build-essential/essential-packages-list
  base-files base-passwd bash coreutils dash debianutils diffutils
  dpkg e2fsprogs findutils grep gzip hostname ncurses-base
  ncurses-bin perl-base sed login sysvinit-utils sysvinit
  tar bsdutils mount util-linux
```

- 4.03 アップストリームのソースへの変更
  1. パッケージを Debian または Linux 向けにする際、ソースコード変更が生じた場合は、それが Debian 固有の理由でないならアップストリームの作者へ変更依頼すべきである。
  2. 'Makefile' を修正したい場合は'Makefile.in' ファイルを修正する。'Makefile' を修正しても'configure' 実行で消えてしまうので再現できない。

ビルド手順の一例

```
$ autoconf     // 'configure' スクリプトを生成。
$ ./configure  -> Makefile.in を読み込んで Makefile を生成。
$ make         // Malefile を読み込んで make する。
```

* 各 unix 系 OS の違いを自動的に吸収する手段

- 4.04 Debian changelog: debian/changelog
  1. changelog への記載事項を規定
     – 当該パッケージに対する修正や更新
     – 上流の版に加えた Debian 向けの修正や更新
     – chanegelog への記載フォーマットは厳密に決まっている。

実際にビルドしてみよう。

参照先 (下記) の通りにやってみる。
参照先: `http://www.debian.org/doc/manuals/maint-guide/build.ja.html`

- 4.05 著作権表記: debian/copyright
  1. 各パッケージには著作権と配布条件のライセンス文書を元のままの形式で'/usr/share/doc/package/copyright' に収録すること。
- 4.06 makefile でのエラーを捕捉する
  1. make する場合は上流で作られた makefile にせよ debian/rules にせよ、 sh を使うが sh のエラー処理は不完

全なため、 debian/rules はエラーを確実に捕捉できるようスクリプトを記述すること。
(例) 2 つ以上のコマンドをつなぐときは';' ではなく'&&' で。

- 4.07 タイムスタンプ
  1. 上流のソースファイルのタイムスタンプは可能な限り変更しないこと。
- 4.08 ソースパッケージに含まれるものに対する制限
  1. ソースパッケージに次のものは含まないこと。
     - ハードリンク
     - デバイスファイル
     - ソケットファイル
     - setuid や setgid されたファイル
- 4.09 debian/rules - メイン構築スクリプト
  1. debian/rules とはソースパッケージからバイナリパッケージを構築するスクリプトである。
  2. このファイルの先頭は'#! /usr/bin/make -f' と記述すること。
  3. スクリプトは非対話形式であること。
  4. 非対話的である必要最小限のターゲットは以下の 5 つ。
     (dpkg-buildpackage で呼び出されるターゲット)
     - clean: ソースツリーのクリーン
     - binary: ソースパッケージのビルド
     - binary-arch: アーキテクチャ依存のバイナリパッケージをビルド
     - binary-indep: オートビルダ[*26]システム使用時は不要。
     - build: ビルド (コンパイル) を行いバイナリパッケージを構築する。

     (詳細は割愛)
- 4.10 変数置換 debian/substvars
  1. dpkg-gencontrol が DEBIAN/control ファイルを生成するが、このファイルに書き込む前に変数置換を行う。
  2. 事前に変数置換定義を debian/subtvars ファイルに記述しておく。
  3. 変数置換は $変数名 の書式を持つ。
  4. 変数置換の詳細の参照先: deb-substvars(5)
- 4.11 上流のソースの場所の設定 (オプション) - debian/watch
  1. 目的は新しく提供されたパッケージのアップデートを http または ftp で走査できるようにすることである。
  2. このファイルは uscan ユーティリティで使う。
  3. パッケージ品質管理とディストリビューション全体の保守のために使われている。
  4. 使用例 `http://dehs.alioth.debian.org/` および他の Debian QA ツール。
- 4.12 debian/files
  1. このファイルはパッケージビルド中に生成されたファイルを記録するための一時ファイルである。
  2. dpkg-genchanges は、".changes" ファイルを生成するときにこれを用いる。
  3. このファイルをアップロードされるソースパッケージに含めてはいけない。
  4. これらのファイルは、 clean ターゲットによって削除すること。
  5. binary ターゲットを開始する際には、これらのファイルを削除するか空にすること。
  6. dpkg-gencontrol を実行した際に、 debian/files に追加されるエントリも clean ターゲットで削除すること。
  7. ソースパッケージと、 dpkg-gencontrol によってコントロールファイルが生成されたバイナリパッケージ以外のファイルはパッケージのトップ階層ディレクトリの親ディレクトリに置くこと。
  8. debian/files のリストにファイルを追加する場合は dpkg-distaddfile を呼ぶこと。

[*26] 種々の移植版をサポート

- 4.13 コードの便宜的写し
  1. Debian パッケージに、他のパッケージのコードの写しを直接組み込むことはしないこと。
  2. もし、組み込みたいコードが Debian アーカイブライブラリであるなら、バイナリパッケージからライブラリ参照を行なうとよい。
  3. もし組み込みたいコードが Debian に収録されていない場合は、可能ならばそのコードを、前提の依存関係を付けて別にパッケージングすべきである。
- 4.14 ソースパッケージの処理: debian/README.source
  1. dpkg-source: Debian ソースアーカイブの作成、展開を行う。
  2. dpkg-source -x filename.dsc [output-directory]:
     ソースパッケージを Debian ソースコントロールファイル (.dsc) 名として展開先ディレクトリ output-directory に展開する。
  3. dpkg-buildpackage : バイナリパッケージおよびソースパッケージをビルドする。
  4. dpkg-source -x をソースパッケージに対して実行しても、直ぐ編集可能で、変更を加えた後修正されたパッケージを追加作業なしに dpkg-buildpackage で作成できるようなパッケージのソースが得られない場合には、debian/README.source 解説ファイルの追加をする。
  5. debian/README.source 解説ファイルには、以下のすべての手順が説明されていなければならない。
     – 全部のパッチが当たったソースを作成する方法。 (debian/rules の patch ターゲットで行なえるようにする。)
     – ソースを変更し、その変更をセーブしてパッケージ作成の際に適用できるようにする方法。
     – パッケージをビルドする際に、現在当てられているソース修正をすべて元に戻す方法。
     – 新しい上流の版が出た場合、 Debian ソースパッケージをアップグレードする手順。 (可能なら)
     – この説明は具体的なコマンド名やパッケージ名にまで言及すべきである。
     – この説明は Debian パッケージシステムや管理ツールを熟知していなくても理解できるように記述すべきである。

以上

16.5 参考文献

1.「 Debian ポリシーマニュアル バージョン 3.9.1.0, 2011-07-05」
`http://www.debian.or.jp/community/devel/debian-policy-ja/policy.ja.html/ch-source.html`
2. 2006 年 4 月 15 日「 第 15 回東京エリア Debian 勉強会事前配布資料」
`http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume200604.pdf`
3.「 Debian 新メンテナーガイド 第 6 章 パッケージのビルド」
`http://www.debian.org/doc/manuals/maint-guide/build.ja.html`

17 月刊 Debian Policy 第6回「 文書」

岡野 孝悌

Debian ユーザーでもないのに今回の担当となりました おかの です。

今回読むのは第 12 章の「 文書」についてです。多くのソフトウェアパッケージにはマニュアルなどの文書が附属していますし、文書のみからなるパッケージもあります (debian-policy 自体がまさにそうです)。また、マニュアル類のほか、設定ファイルのサンプルや、変更履歴、著作権情報といったものもあります。

17.1 マニュアル (man ページ)

ベル研で Unix が誕生したころ、文書処理システムという口実で予算を取っていたりして、 Unix 関係の文書といえば roff でした[*27]。本章でも man ページの説明に多くを割いています。

17.1.1 インストール場所

roff ファイルを gzip -9 で圧縮して、`/usr/share/man` 以下の適切な場所にインストールします。英語マニュアルは `/usr/share/man` 直下、それ以外は `/usr/share/man/`*locale* 以下の、セクション別のディレクトリ (`man1`, `man2`, ..., `man9`) に置きます。詳細は FHS 参照といいつつ、 FHS とは以下のような差異があります。

- 英語マニュアルは `/usr/share/man/en` 以下ではなく `/usr/share/man` 直下に置く。 (FHS でこれが許されるのは、英語マニュアルしかない場合)
- セクション 9 が存在する (FHS では 1 から 8 までのみ)

man ページの整形には時間がかかるので、整形済みのマニュアルをあらかじめインストールしておけば[*28]、いきなりページャーで表示することができて快適になりますが、パッケージが整形済みマニュアルをインストールしてはいけません。

17.1.2 パッケージへの同梱

man ページ以外の文書は、 *foo*-doc のようにソフトウェア本体と別パッケージに分離することがありますが、 man ページはソフトウェア本体と同じパッケージに含めるのが原則です。

プログラムのマニュアルが存在しない場合、それはバグとして扱われます。上流にマニュアルを追加してもらうか、Debian パッケージで独自にマニュアルを追加する必要があります[*29]。といっても、「 info 見れ」「 この URL 見れ」だけのマニュアル[*30]でも可です。これは、とにかく man を見れば情報に辿り着けることを保証する、という考えにもとづいています。

[*27] History of UNIX Manpages http://manpages.bsd.lv/history.html が参考になります。
[*28] FreeBSD のベースシステムなどでは整形済みのマニュアルが標準で配布されています。
[*29] 現在はバグのあるパッケージはそもそもアップロードできないので、 man がないパッケージはありえないことになります
[*30] そのようなマニュアルの実例としては、 (いずれも上流由来ですが) GNU cpio や netpbm があります。

17.1.3 複数名をもつマニュアル

grep, fgrep, egrep のように、同じマニュアルが複数の名前で参照されることがあります。

Linux Man Page Howto[*31] では、(シンボリックリンクを持たないシステムに配慮し) roff の .so マクロを使う方法がもっとも望ましいとされていますが、Debian では必ずシンボリックリンクがあるためか、原則として .so マクロを使わずにシンボリックリンクを使うよう求めています。ただし、上流で .so マクロを使っている場合は、無理にシンボリックリンクに直す必要はありません。

たとえば grep パッケージ (GNU grep) では、egrep.1.gz, fgrep.1.gz はいずれも grep.1.gz へのシンボリックリンクとなっています。いっぽう、manpages-ja パッケージにはこれらの (古い) 日本語訳がありますが、こちらでは egrep.1 や fgrep.1 は .so マクロを使って grep.1 を参照しています。

なお、これとは逆に、複数の異なるマニュアルが同じ名前で参照されることがあります。たとえば vim と nvi は、通常どちらも vi という名前で参照されます。本章では言及されていませんが、これは Appendix F で説明されている update-alternatives で扱うことができます。また、editor(1) と pager(1) について 11 章で言及されています。

17.1.4 翻訳マニュアル

マニュアル以外の文書にも翻訳はありますが、debian-policy ではマニュアルについてのみ翻訳に触れています。マニュアルはガンガン更新されるので、せっかく翻訳してもすぐ古くなってしまいます。利用者が古い翻訳マニュアルを鵜呑みにしないように、翻訳が古い可能性について言及したりするよう求めています。

apt など、Debian 独自ツールのマニュアル翻訳では、po4a を使って翻訳し (訳が古くなった場合、その部分は原文が表示されるようになる)、かつ注意書きをすることで、この指針が守られているようです。一方、manpages-ja をはじめ、上流で翻訳されたマニュアルでは、守られていないように見えます。

エンコーディングは UTF-8 のほか、一部のレガシーエンコーディング (日本語の場合は EUC-JP) も使えます[*32]。

17.2 info 文書

info 文書について述べています。info 文書は必須ではありません。

- info 文書は `/usr/share/info` 以下へインストールする
- gzip -9 で圧縮する
- install-info 用のセクションとディレクトリエントリ情報を含める

17.3 追加文書

man, info 以外の形式の文書がパッケージに含まれている場合の扱いを述べています。

- info は `/usr/share/doc/`*package* 以下へインストールする
- 大容量で必要性の低い文書は別パッケージにするべき
- ソフトウェアの動作が `/usr/share/doc/`*package* 以下のファイルに依存してはいけない
- `/usr/share/doc/`*package* 自体を他へのシンボリックリンクとできる条件 (著作権関連情報の節にも記載がありますが、適切な copyright ファイルを機械的に抽出できる必要があります)
- `/usr/doc` から `/usr/share/doc` への移行に関する説明[*33]

[*31] `http://www.schweikhardt.net/man_page_howto.html` (JF に古い日本語訳あり)

[*32] man のエンコーディングに関する議論は Bug#440420 参照

[*33] 脚注にも説明がありますが、FSSTND 1.2 までは、`man` や `doc` 等は`/usr/` 直下に置くこととなっていました。FHS 2.0 (1997 年) で、BSD の流儀を取り入れ、`/usr/share/` 以下に置くよう改められています。Debian では potato で移行しています。当時の議論は 1999 年からの debian-ctte メーリングリストのログで見ることができます。

17.4 好ましい文書形式

DocBook などをもとに同じ文書を複数形式で提供するケースがありますが、そのような場合は可能な限り HTML をインストールします。 HTML 以外の形式をどうするかはメンテナーの裁量にまかされています。

17.5 著作権関連情報

パッケージに含まれるものの著作権、ライセンスに関する文書 (`copyright` ファイル) の扱いについて述べています。このファイルは必須です。

Debian パッケージでは `debian/copyright` に含めておき、`/usr/share/doc/`*package*`/copyright` にインストールします。

このファイルには、パッケージの著作権、ライセンス情報を、元のままで含める必要があります (ただし、 GPL などのよく使われるライセンスは `/usr/share/common-licenses` にあらかじめ置かれているものを参照する形にします)。内容の改変はもとより、文書形式の変換もしてはいけません。また、このファイルを圧縮したりシンボリックリンクにしたりしてはいけません。

`copyright` ファイルには、著作権、ライセンス情報のほか、以下の情報を含める必要があります。

- 上流のソースの入手元
- パッケージの原作者名

また、以下の情報を含めるべきです。

- Debian パッケージ作成に関与した Debian メンテナ名 (3.9.3.0 で削除されました; 後述)
- (contrib, non-free の場合) 当該パッケージが Debian の一部ではないことと、その簡潔な理由

17.6 設定例など

設定ファイルやソースファイルなどの例を提供する場合の扱いを述べています。

- `/usr/share/doc/`*package*`/examples` 以下にインストールする。
- ただし、アーキテクチャー依存のファイルは`/usr/lib/`*package*`/examples` 以下にインストールし、シンボリックリンクを張る。
- 前述のとおり、動作が `/usr/share/doc` 以下のファイルに依存してはいけないので、ここに置かれたファイルをプログラムから参照してはいけない。
- *foo*-examples のような、例そのものを提供するパッケージは、パッケージそのものを `/usr/share/doc/`*package* にインストールしてもよい。

17.7 Changelog ファイル

変更履歴の扱いについて述べています。

Debian 由来でないパッケージは、上流の変更履歴と Debian パッケージの変更履歴を区別することを求めています。

- Debian ソースツリー内の `debian/changelog` の圧縮されたコピーを`/usr/share/doc/`*package*`/changelog.Debian.gz` としてアクセスできるようにしなければならない。
- 上流の changelog がある場合は、プレインテキスト形式で、`/usr/share/doc/`*package*`/changelog.gz` としてアクセスできるようにすべき。
- changelog ファイルが HTML 形式の場合は、`/usr/share/doc/`*package*`/changelog.html.gz` で参照できる

ようにすべき。

- 上流のファイル形式の違いによって、別のファイル名を参照しないといけないなどというのはウンコなので、 Lynx を使って HTML をプレインテキストに変換したものを `/changelog.gz` で提供すべき。
- gzip -9 で圧縮すべき

17.8 最近の変更点

日本語訳版がある Version 3.9.1.0 から Version 3.9.4.0 までに加えられた変更点を押さえておきましょう。

17.8.1 Version 3.9.2.0

- 「 Debian GNU/Linux ディストリビューション」が「 Debian ディストリビューション」と改められました。 (Bug#594656)
- debmake で生成される man のサンプルについて言及されていましたが、 lenny 以降では debmake は削除されているため、この記述が削除されました。

17.8.2 Version 3.9.3.0

- `copyright` ファイルに Debian パッケージのメンテナーを記述することとされていましたが、 `changelog` ファイルに書かれているものなので、 `copyright` ファイルへの記述がなくなりました。 (Bug#593533)

Version 3.9.2.0 policy.sgml:9771

In addition, the copyright file must say where the upstream sources (if any) were obtained. It should name the original authors of the package and the Debian maintainer(s) who were involved with its creation.	また、著作権情報ファイル中には元となった上流のソースをどこから手に入れたかを記載しなければなりません。またパッケージの原作者の名前とパッケージ作成に関与した Debian メンテナの名前を載せるべきです。

Version 3.9.3.0 policy.sgml:9863

In addition, the copyright file must say where the upstream sources (if any) were obtained, and should name the original authors.	また、著作権情報ファイル中には元となった上流のソースをどこから手に入れたかを記載しなければなりませんし、パッケージの原作者の名前を載せるべきです。

- 「 機械的に抽出」できるようにするもの (`copyright` ファイル) が何を指しているかが、明確に記述されるようになりました。 (Bug#617516)

Version 3.9.2.0 policy.sgml:9704

`/usr/share/doc/`*package* may be a symbolic link to another directory in `/usr/share/doc` only if the two packages both come from the same source and the first package Depends on the second. These rules are important because copyrights must be extractable by mechanical means.	`/usr/share/doc/`*package* は、`/usr/share/doc` 以下の他のディレクトリ中のシンボリックリンクの相手先と同じソースファイルから作成されたものであること、および相手先に対して “Depends” で依存していることが宣言されていること、の二つの条件を満たすときのみ、シンボリックリンクとすることができます。この規則は、著作権関連ファイルが機械的に抽出できるようにするために大切になるものです。

Version 3.9.3.0 policy.sgml:9796

/usr/share/doc/*package* may be a symbolic link to another directory in /usr/share/doc only if the two packages both come from the same source and the first package Depends on the second. These rules are important because copyright files must be extractable by mechanical means.	/usr/share/doc/*package* は、/usr/share/doc 以下の他のディレクトリ中のシンボリックリンクの相手先と同じソースファイルから作成されたものであること、および相手先に対して “Depends” で依存していることが宣言されていること、の二つの条件を満たすときのみ、シンボリックリンクとすることができます。この規則は、copyright ファイルが機械的に抽出できるようにするために大切になるものです。

- 機械可読な著作権情報ファイルに関する記述が追加されました。ライセンス衝突などの問題を自動的にチェックしたりできることが期待できますが、今のところはオプション扱いです。

Version 3.9.3.0 policy.sgml:9928

Machine-readable copyright information A specification for a standard, machine-readable format for debian/copyright files is maintained as part of the *debian-policy* package. This document may be found in the copyright-format files in the *debian-policy* package. It is also available from the Debian web mirrors at http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/. Use of this format is optional.	**機械可読な著作権情報** 標準のための仕様として、機械可読な形式の debian/copyright ファイルが、この *debian-policy* パッケージの一部として保守されています。この文書は *debian-policy* パッケージに copyright-format ファイルとして含まれています。また、Debian web ミラーの http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/ にもあります。 この形式を使うかどうかは任意です。

17.8.3 Version 3.9.3.1

(変更なし)

17.8.4 Version 3.9.4.0

- copyright ファイルのエンコーディングに関する一文が追加されました。(Bug#661933)

Version 3.9.1.0 policy.sgml:10673

All copyright files must be encoded in UTF-8.	copyright ファイルはすべて UTF-8 でエンコードしなければなりません。

17.9 最後に

次回の月刊 Debian Policy は。。。

18 ソフト開発以外の簡単 Debian contribution

野島 貴英

18.1 はじめに

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

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

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

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

18.3 screenshots.debian.net

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

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

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

以下にスクリーンショットの投稿の仕方を記載します。

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

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

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

Step 4. 導入したプログラムを起動します。

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

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

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

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

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

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

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

18.4 debianart.net

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

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

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

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

18.5 おわりに

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

19 本気で“使える”翻訳環境を構築してみた

八津尾 雄介

19.1 はじめに - 今日の主旨

アプリケーションのローカライズに man ページ、 wiki 等々、普段何気にお世話になっている翻訳ですが、 OSS の世界では (当然ですが) 有志によって行われています。しかし、日本の OSS コミュニティにおいて翻訳者の不足はどのプロジェクトでも共通の課題らしいということをよく耳にします。翻訳は高い技術力を必要とせず、誰でも簡単に参加できます。翻訳支援ツールを使い、翻訳のプロセスを簡略かつ明確にすることで、さらに参加する為の敷居が下がるのではないでしょうか? ということで、翻訳ツールを利用した翻訳環境の構築事例を紹介していきたいと思います。

今回は翻訳ネタ 2 本立てという事で、トランスレータとして各プロジェクトに参加中の皆さん、またこれから翻訳に参加してみようかとお考えの皆さんと一緒に、翻訳環境について議論していきたいと思いますので積極的な発言をお願いします。

19.2 DPN の翻訳効率化

Debian Project News (以下、 DPN) の翻訳に参加しはじめてから半年が過ぎました。 DPN を翻訳していくなかで、いくつか気づいた事があります。

- ほぼ同じ内容の文章を何度も訳している
- 担当者が過去記事を参照せず、同一の事象での訳語が統一できていない
- グロッサリが存在せず、担当者間で訳語が統一できていない
- 明確なスタイルガイドラインが存在せず、暗黙のルールで運営されている
- 過去に訳した膨大な量の訳文 (=資産) を活かしきれていない

このような無駄を無くし、翻訳を効率化するために**翻訳メモリ**が活用できるのではないかと考え、現在快適な翻訳環境作りに取り組んでいるころです。私が考える翻訳環境の柱は次のようになります。

- 訳語を意識せずに揃えること
- 誰が翻訳しても同一の品質を保てること
- 過去の翻訳結果を再利用できること
- 翻訳プロジェクトのメンバ内でこれらの結果を共有できること

19.2.1 翻訳メモリとは?

翻訳メモリとは、“ソースの言語とターゲットの言語を 1 対 1 で保存したデータベースのようなもの”/です。**翻訳メモリツール**は、翻訳メモリから類似の文章を探し出しヒット率が高い文章の訳文を提示したり自動挿入や置換をする事で翻訳の支援を行うツールです。

よく勘違いされるのですが、**翻訳メモリツール**は勝手に翻訳してくれる機械翻訳のようなものではありません。**翻訳メモリ**は自分で育てる必要があり初期状態では全く使いものになりません。また、**翻訳メモリツール**は訳文の正確性の判断はできないので、翻訳メモリに誤訳が混入しそれが繰り返し利用されてしまう可能性がある事を常に念頭に置いておきましょう。

翻訳メモリには TMX（Translation Memory eXchange）という XML ベースの規格があり、広く利用されています。[*34]

産業翻訳でデファクトスタンダードとなっている SDL TRADOS ®[*35] や、本セッションで紹介予定の OmegaT [*36] もこの TMX を採用しています。**翻訳メモリ**の更に詳しい概要と OmegaT の操作方法については、2011 年 10 月の関西 Debian 勉強会資料に掲載されている拙著「翻訳で Debian に貢献しよう」も参考にしてください。

19.2.2 定型記事と翻訳メモリ

翻訳メモリがその真価を発揮できるのが、ある程度定型化された記事を翻訳する場合です。以下の文例を見て下さい。

DPN2012 issue 15 より

```
<toc-add-entry name="newcontributors">New Debian Contributors</toc-add-entry>
<p>
    Seven applicants have been
    <a href="https://nm.debian.org/public/nmlist#done">accepted</a>
    as Debian Developers, nine applicants have been
    <a href="http://lists.debian.org/E1Sta5X-0005lb-Ap@franck.debian.org">accepted</a>
    as Debian Maintainers, and six people have
    <a href="http://udd.debian.org/cgi-bin/new-maintainers.cgi">started to
    maintain packages</a> since the previous issue of the Debian Project News.
    Please welcome（著者注：中略 ここで人名が列挙される）into our project!</p>
```

DPN2012 issue 17 より

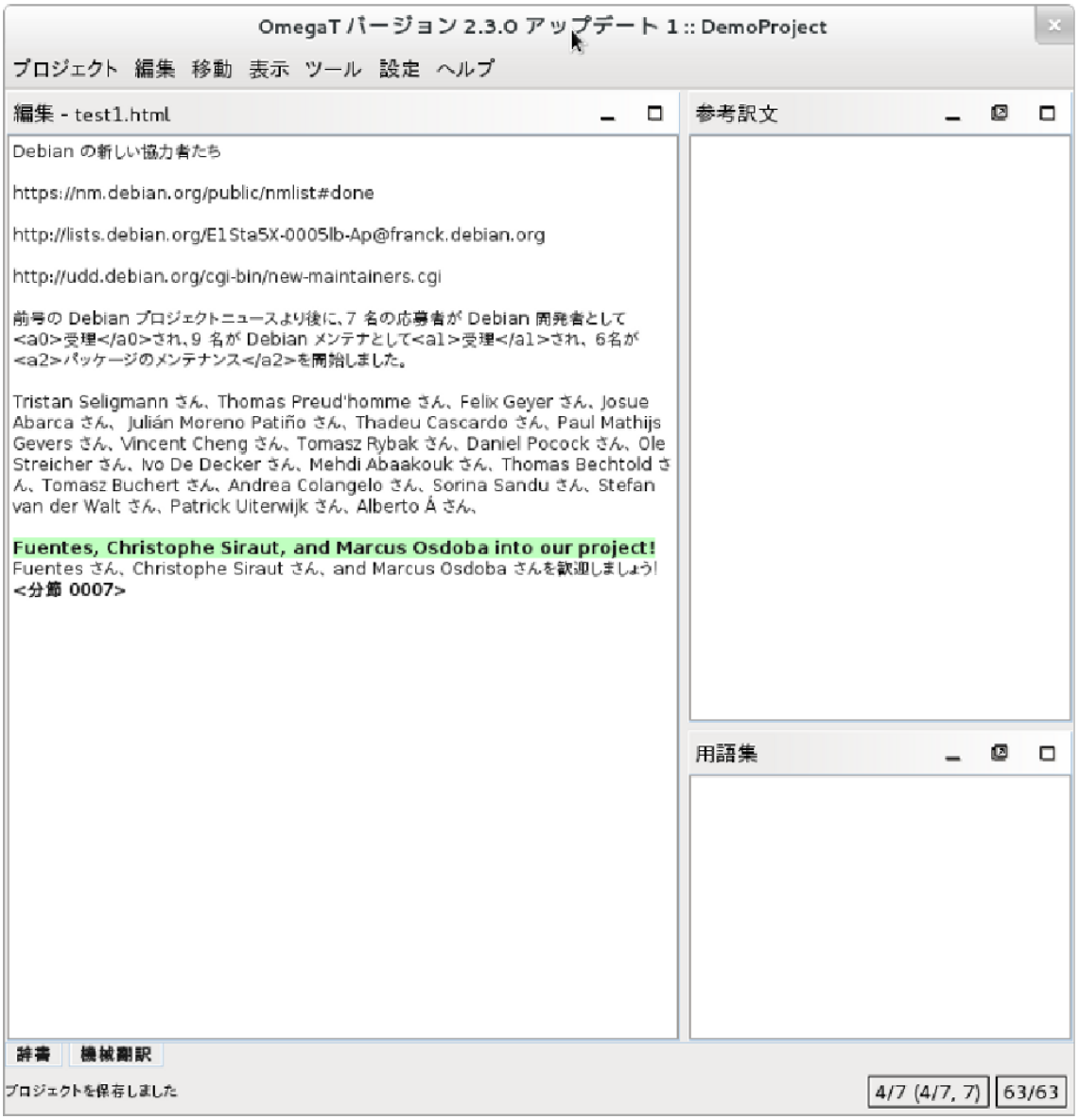
```
<toc-add-entry name="newcontributors">New Debian Contributors</toc-add-entry>
<p>
    Three applicants have been
    <a href="https://nm.debian.org/nmlist.php#newmaint">accepted</a>
as Debian Developers,and two people have
    <a href="http://udd.debian.org/cgi-bin/new-maintainers.cgi">started to
    maintain packages</a> since the previous issue of the Debian Project News.
    Please welcome（著者注：中略 ここで人名が列挙される）into our project!</p>
```

これらの記事はほぼ同じ内容であり、毎月必ず掲載される記事です。にも関わらず、Debian-www メーリングリスト

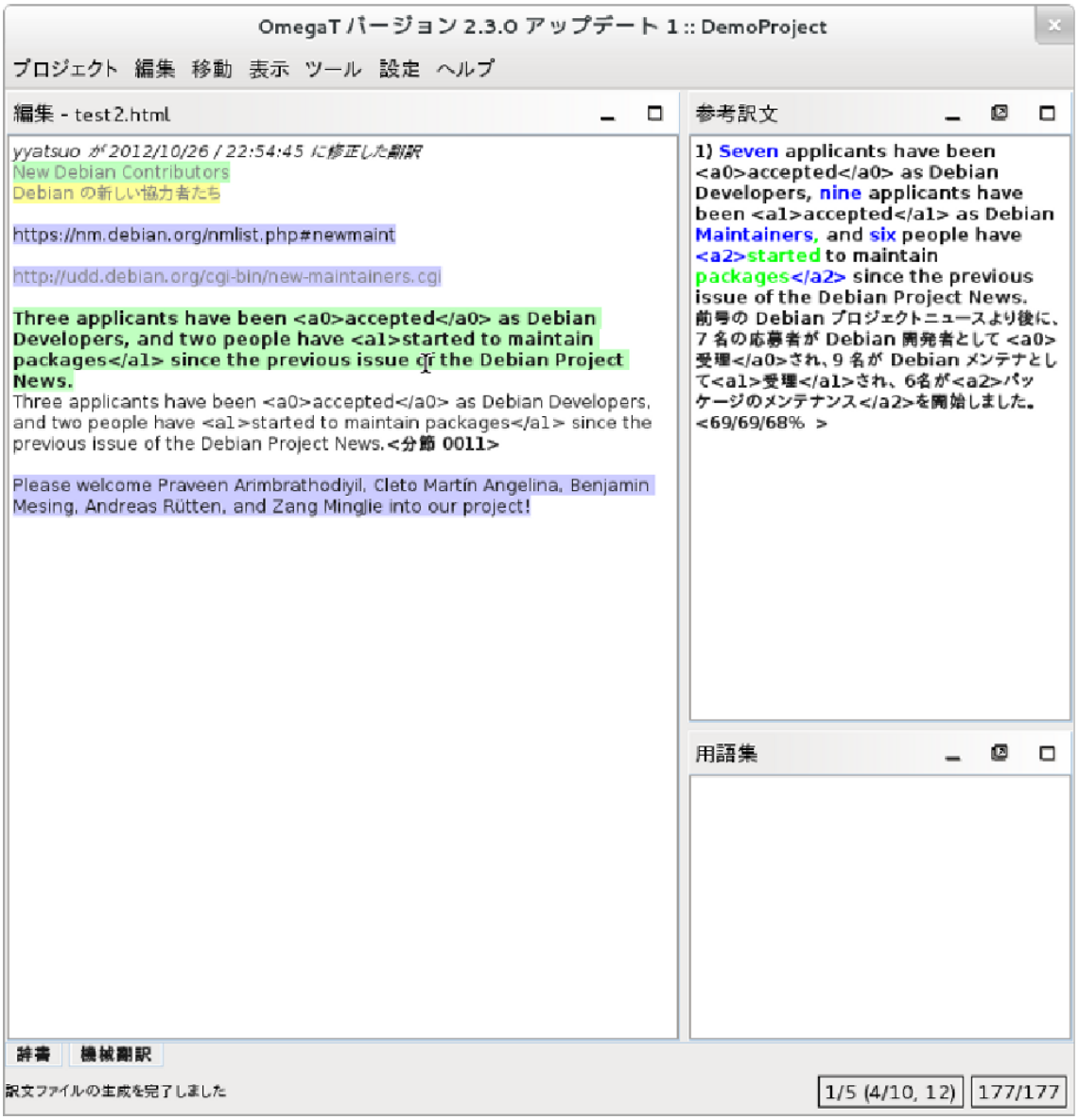
[*34] もともと LISA (Localization Industry Standards Associa tion) によって策定された規格。2011 年、LISA は破産宣告によって解散し、LISA で策定された他の規格と共にパブリックドメインへと帰した。
[*35] http://www.trados.com
[*36] http://www.omegat.org

で配布される原文に添付されている、おそらく何かのスクリプト処理によって生成されたと思われる試訳では全く訳されません。翻訳メモリで一度でも訳しておけば、以降は訳文候補として過去の訳が参照できるのでこのような記事をシームレスに翻訳することができます。では、実際に OmegaT でどの程度効率化できるのか試してみましょう。なお、ここでは OmegaT の詳しい使い方は説明しませんが、起動時に表示されるお手軽スタートガイドに目を通すだけで一通りの作業はこなせるようになるでしょう。

DPN2012 issue 15 の日本語訳



DPN2012 issue 17 を訳しているところ



このように、翻訳メモリを利用することで訳語の揺れを減らし訳文の品質を高めることができます。

19.2.3 訳語の統一

担当者が変わったり単に翻訳者が以前当てた訳語を忘れてしまい、訳語から統一感が失われてしまうという問題はしばしば発生します。この問題に対処する為にはグロッサリのメンテが必要不可欠です。

訳語を決めたらグロッサリに登録しておきましょう。 OmegaT の場合は UTF-8 のタブ区切り形式でグロッサリを作成しておくことができます。[*37]残念ながら、現在メンテされていませんが OmegaT で利用できる Debian 翻訳用のグロッサリも入手可能です。[*38] 私は現在個人でグロッサリの作成とメンテをしていますが、本来であればプロジェクト単位で準備すべきものです。グロッサリを整備しておくことで、新規参入者が迷わずに翻訳を始められるのではないでしょうか。

19.2.4 マークアップと翻訳メモリ

DPN は wml という html によく似た xml の拡張フォーマットで記述されたものがメーリングリストで配布され、それを各担当者が訳しています。 OmegaT を利用する時に注意が必要な事は、 wml がサポートされていないという事です。

[*37] 複合語に難があったりとあまり洗練されていない
[*38] Debian JP Doc/WWW 対訳表: http://www.debian.or.jp/community/translate/

OmegaT でサポートされていないファイルフォーマットは翻訳対象ファイルから見えません。他にも、古いドキュメント類に散見される sgml などはサポートされていないので、ちょっとしたトリックでこれを翻訳できるようにしましょう。

もっとも簡単な方法はテキスト形式にしてしまうことです。翻訳対象ファイルの拡張子を *.txt にしても良いですし、*.txt のファイルフィルタに翻訳したいファイルを登録する方法もあります。しかしこの方法を使うと、タグの中まで翻訳対象となってしまい翻訳メモリの能力を最大限引きだす事ができません。テキスト形式にしてしまう前に、近いフォーマットのフィルタを試してみると良いでしょう。私は wml に html のフィルタを使用しています。いまのところ問題なく使えています。[*39]

19.2.5 翻訳ツール

翻訳メモリを導入することで、私が必要としていることの大半は実現できましたが、理想の環境とはほど遠いのが現状です。まだまだ試行錯誤の段階ではありますが、私が現在 DPN の翻訳に使用しているツールを紹介します。

- DPNToolKit.py - 自作の python スクリプトです。 wml ファイルを分解し OmegaT のプロジェクトフォルダに配置したり、 OmegaT が生成した訳文から査読メール用の文章を生成したりします。
- Debian JP Doc/WWW 対訳表一応導入し、個人的にメンテしています。 OmegaT から参照可能です。
- 英辞郎 - 言わずと知れたオンライン辞書です。第 3 版とオンライン版を適宜使い分けています。 OmegaT から参照可能です。
- Google 翻訳 - 私が知っている機械翻訳の中で、最も精度の高い翻訳結果を出すのが Google 翻訳です。機械翻訳はあくまで参考に使用する程度にしましょう。
- vim - ちょっとした文章は vim で訳してしまうことが多いかも。 W3m プラグインで英辞郎や WikiPedia を参照でき、重宝しています。 OmegaT の外部エディタとして vim や emacs が使えれば最高なのですが…
- translate-toolkit - 今後使用してみたいツール群です。特に po2tmx が気になっています。 po ファイルから tmx 形式の翻訳メモリを生成できます。これを活用すれば、ソフトウェアのローカライゼーションが一気に楽になるのではないでしょうか?

19.3 今後の課題

- 翻訳メモリをプロジェクト内で共有する方法
- OmegaT の残念なインタフェースをどうするか
- OmegaT 以外翻訳メモリを調査する
- GlobalSight について調査する
- 翻訳スタイルガイドラインみたいなもの作りたい

[*39] OmegaT の Okapi プラグインで対応ファイルを増やす方法もある

20 Debian セキュリティ勧告翻訳の舞台裏

久保 博

20.1 はじめに

Debian セキュリティ勧告の翻訳に携わるようになったので、 Debian セキュリティ勧告の簡単な紹介を交えながら、その翻訳の過程がどのようになっているか、紹介します。

20.1.1 Debian セキュリティ勧告

Debian セキュリティ勧告（ Debian Security Advisory 以下略称 DSA を使う）は、 Debian を構成するソフトウェアのセキュリティに関する問題を公開し、対策を勧める文書です。 Debian セキュリティチームの構成員が執筆しており、 DSA- で始まる通し番号が振られ、執筆者のデジタル署名付のメールで `debian-security-announce@lists.debian.org` メーリングリストに発表されています。

また、「 Debian 安全化マニュアル」の 2.3 節 [1] によれば、そもそも DSA を発行することは、 Debian 社会契約 [2] の第三項「 私たちは問題を隠しません」の具体的な取り組みの一つです。

日本では、その日本語訳を `debian-users@debian.or.jp` メーリングリスト (以下、略して `d-u`) に流す取り組みが以前から継続してなされています。

20.1.2 日本語訳作業への参加の経緯

今年（ 2012 年） 7 月まで翻訳を `d-u` に流されていたかねこさんが、しばらくの間インターネット接続が途切れるので翻訳を流すことが滞る旨の予告を `debian-www` メーリングリストで流されました。それを受けて、中断期間中に筆者が翻訳を始めました。その後、かねこさんがインターネット接続が復帰とともに、翻訳活動を引退される旨のメールが流れたので、引続き筆者が DSA の翻訳活動を努めるようになりました。現在、筆者一人で活動しています。

20.2 翻訳作業

20.2.1 環境作り

Debian パッケージ中心で翻訳作業環境を整えています。

■翻訳文編集環境用のパッケージ

emacs23　GNU Emacs
sed　定型文の置換処理のスクリプト用.
omegat　Computer Assisted Translation (CAT) tool. OmegaT

また、辞書もパッケージにあるものはインストールしてあります。

■辞書パッケージ

dict-gcide　A Comprehensive English Dictionary
dict-jargon　dict package for The Jargon Lexicon
dict-moby-thesaurus　最大かつ最も理解しやすい類語辞典
edict　English / Japanese dictionary

更にこれらの辞書を引くために、次のパッケージをインストールして使っています。

■辞書利用環境のパッケージ

lookup-el　emacsen での電子辞書インターフェイス
ndtpd　ネットワークディレクトリ転送プロトコルサーバ
ebnetd　the server of EBNET protocol

20.2.2　事前準備

電子メールで DSA を受信するために、 debian-security-announce メーリングリストを購読します[*40]。

また、 DSA のメールは、セキュリティチームの構成員の PGP 署名が付与されています。これを検証することで、セキュリティチームからの正式な発表であることが確認できますので、電子メールの PGP 署名を検証できるようにしておきます。 gnupg パッケージをインストールしておくと良いでしょう。

20.2.3　DSA 受信から翻訳文送信まで

DSA を受信してから、 d-u に流すまでが、 DSA 翻訳として受け持っている作業です。大まかに分けると、次のような段階を踏んで作業しています。

1. DSA の受信と署名の検証
2. DSA 本文の翻訳
3. debian-users@debian.or.jp メーリングリストへの投稿
4. 後処理 難訳語の用語集への登録など

20.2.4　DSA の受信と署名の検証

debian-security-announce から DSA のメールが届いたら、翻訳に取り掛かりますが、その前に 一応メールのデジタル署名を検証します。

筆者はメーラに Mew[*41] を使っていますが、その場合は C-c C-z で署名の検証ができます。

署名を検証する為の公開鍵が見つからない場合は、公開鍵を用意します。 Debian の環境でしたら、 debian-keyring パッケージで提供される開発者の公開鍵を利用できます。利用方法は簡単で、 GnuPG の設定ファイル ~/.gnupg/gpg.conf に次の行を追加します。

```
keyring /usr/share/keyrings/debian-keyring.gpg
```

作業環境のコンピュータの OS が Debian でない場合や、 debian-keyring パッケージに署名を検証する為の公開鍵が見つからない場合は、別の方法で公開鍵を取得します。

まず、 Debian の開発者データベースである「 debian.org Developers LDAP Search」[*42] で DSA の執筆者を検索

[*40] メーリングリストの購読方法は http://lists.debian.org/debian-security-announce/ を参照。
[*41] mew あるいは mew-beta パッケージでインストールできます。本家のサイトは http://www.mew.org/ です。
[*42] http://db.debian.org/

します。検索結果で PGP の指紋 (fingerprint) を確認したら、コマンドラインで

```
# gpg --keyserver keyring.debian.org --recv-keys "fingerprint"
```

のように操作することで公開鍵を取得できます。うまく公開鍵が取得できたら、もう一度改めてメールの署名を検証します。また、

```
# gpg --list-sigs <メールアドレス | 鍵 ID >
```

で信頼関係を確認しておくのもいいでしょう。

20.2.5　文章の部分毎の翻訳手順

署名の検証ができたら、翻訳に取り掛かります。セキュリティ勧告の文書の構成には決まった型があり、いくつかの部分に分解して扱うことができます。そこで、以下では例として DSA-2541-1, DSA-2546-1, DSA-2548-1, DSA-2549-1 を取り上げ、各部分毎に分けて翻訳する手順を順に説明することにします。

二段組の左側は英語の原文、右側はその翻訳文です。

本文 1 (DSA-2541-1)

原文	日本語訳文
-	- -
	くぼです。
Hash: SHA1	URL 等は Debian-security-announce メーリングリストの元記事を確認ください。

原文には PGP メールのヘッダ部が最初にありますが、翻訳文では最初に翻訳者の挨拶と断り文を入れているので、それに差し替えています。

本文 2 (DSA-2541-1)

```
- -------------------------------------------------------------------------
Debian Security Advisory DSA-2541-1                   security@debian.org
http://www.debian.org/security/                          Raphael Geissert
September 07, 2012                     http://www.debian.org/security/faq
- -------------------------------------------------------------------------
```

翻訳文にそのまま転記しています。

本文 3 (DSA-2541-1)

原文	日本語訳文
Package : beaker Vulnerability : information disclosure Problem type : remote Debian-specific: no CVE ID : CVE-2012-3458 Debian Bug : 684890	Package : beaker Vulnerability : 情報漏洩 Problem type : リモート Debian-specific: いいえ CVE ID : CVE-2012-3458 Debian Bug : 684890

Vulnerability は過去の翻訳を参考に随時日本語へ置き換えます。 Problem type は、 “remote”, “local”, “local (remote)”[*43] のいずれかの値を取りますが、それぞれ「 リモート」、「 ローカル」、「 ローカル (リモート)」へそれぞれ翻訳します。 Debian-specific は、「 はい」か「 いいえ」かのいずれか翻訳します。

あとは、原文をそのまま転記します。

本文 4 (DSA-2546-1)

原文	日本語訳文
Timo Warns discovered that the EAP-TLS handling of freeradius, a high-performance and highly configurable RADIUS server, is not properly performing length checks on user-supplied input before copying to a local stack buffer. As a result, an unauthenticated attacker can exploit this flaw to crash the daemon or execute arbitrary code via crafted certificates.	Timo Warns さんは、高性能で幅広い設定が可能な RADIUS サーバーである freeradius の EAP-TLS の取扱いで、利用者からの入力を局所的なスタックバッファへコピーする前に長さの検査を適切に実施していないことを見つけました。この結果、認証されていない攻撃者が細工を施した証明書でこの欠陥を衝いて、常駐処理を異常終了させたり任意のコードを実行することができます。

ここからは雛型が特にはない、自由な文章になりますが、最初の一文には、必ずパッケージのソフトウェア名と、その簡単な説明があります。上の例ですと、 “a high-performance and highly configurable RADIUS server” の箇所です。簡単な説明は、パッケージの制御ファイルの Description にある説明になっているので、既にパッケージの Description の翻訳が既にあれば、それを優先的に使用するよう心がけています。なければ、 DDTSS[*44] に新しく翻訳が上がっていないか確認するのが良いでしょう。また、過去の DSA[*45] で既に同じ表現があれば、その日本語訳を優先的に使用しています。

複数の CVE 番号の脆弱性が一つの DSA に含まれると、パッケージ全体の脆弱性の説明に続いて CVE 毎の説明が付されることが多いです。 DSA-2548-1 を例に確認してみます。

[*43] Debian セキュリティ FAQ 5 によると、「 脆弱性のあるサービスが継続的にネットワークに接続していなくても、 ネットワークから配置できる特定のファイルにより突くことができる場合」に使われます。 http://www.debian.org/security/faq#localremote

[*44] Debian パッケージ説明文翻訳プロジェクト(DDTP) のウェブインターフェイス

[*45] URL :http://www.debian.org/security/

本文 5 (DSA-2548-1)

原文	日本語訳文
Several vulnerabilities have been discovered in Tor, an online privacy tool.	オンライン個人情報保護ツールの Tor に複数の脆弱性が見つかりました。
CVE-2012-3518 Avoid an uninitialised memory read when reading a vote or consensus document that has an unrecognized flavour name. This could lead to a remote crash, resulting in denial of service.	CVE-2012-3518 解釈できないフレーバー名を持つ投票あるいは総意の文書を読む際に初期化されていないメモリを読むことを回避しました。これは、サービス拒否を引き起こすことになる遠隔での異常終了に繋がる可能性がありました。

このような感じで、 CVE 毎に一ないし二文の説明が CVE 毎に繰り返されます。つまり、 CVE の数が増えると、翻訳する文章も増えることになります。

本文の最後は、決まり文句的な文が続きます。まずは、リリース毎の修正版の案内文が来ます。

本文 6 (DSA-2549-1)

原文	日本語訳文
For the stable distribution (squeeze), these problems have been fixed in version 2.10.69+squeeze4. For the testing distribution (wheezy), these problems will be fixed soon. For the unstable distribution (sid), these problems will be fixed in version 2.12.3.	安定版 (stable) ディストリビューション (squeeze) では、これらの問題はバージョン 2.10.69+squeeze4 で修正されています。 テスト版 (testing) ディストリビューション (wheezy) では、これらの問題は近く修正予定です。 不安定版 (unstable) ディストリビューション (sid) では、これらの問題はバージョン 2.12.3 で修正されています。

リリース毎の修正版の案内文は、半定型なので、かねこさんから引き継いだ特製の sed スクリプトを使って英文から日本語文へ変換してます。この変換に問題がなければそのまま採用し、問題あれば随時翻訳します。

本文 7 (DSA-2549-1)

原文	日本語訳文
We recommend that you upgrade your devscripts packages. Further information about Debian Security Advisories, how to apply these updates to your system and frequently asked questions can be found at: http://www.debian.org/security/	直ぐに devscripts パッケージをアップグレードすることを勧めます。 Debian Security Advisories に関する説明、これらの更新をシステムに適用する方法、 FAQ などは http://www.debian.org/security/ を参照ください。

これも、特製の sed スクリプトを使って日本語に置換します。この部分に関しては、置換結果をほぼそのまま使ってい

ます。

20.2.6 査読

DSA の翻訳では速報性を重視していて、周囲から査読を受けることを求められることがありません。 debian-www メーリングリストに査読依頼を投げると、有志の方が査読してくれるという助言はあるのですが、投げたことがありません。

20.2.7 投稿

翻訳文が整ったら、バージョン番号、パッケージ名などに間違いがないか確認して、 d-u へ投稿します。

20.2.8 翻訳の副産物

用語集 OmegaT 上で作業する場合は、辞書になくて悩んでつけた訳語などは、適宜用語集に登録しています。ですので、用語集が少しずつ貯っています。

翻訳メモリ OmegaT 上で作業する場合、今のところ一つのプロジェクトで作業しており、一つの DSA が発行されたらそのメール本文をファイルに保存して、原文ファイル追加の操作で OmegaT のプロジェクトに取り込んでいます。したがって、今まで翻訳した内容が翻訳メモリに貯っています。

20.3 これからの課題

とりあえず、翻訳作業を引き受けてはみたものの、いろいろ課題があります。

- 共同作業のインフラと手順の整備。
  ゆくゆくは複数人で進めるために、担当の割り振りや進捗の確認、重複して作業することを防ぐ手順とそれを支える仕掛けなどがあると良いのでは、と考えています。また、翻訳作業の副産物として翻訳メモリや用語集が少しずつ出来上がるかもしれないので、それを共有する仕組みがあれば、用語の統一もしやすくなると考えています。
- DSA の翻訳で、パッケージの説明文を新規に翻訳することがあるので、 DDTSS へそれを登録すること。
  せっかく翻訳したものを使い回してパッケージの説明分を翻訳する手間を減らす貢献もしたいと考えていますが、DDTSS のサイト[*46]が落ちているので進められていません。 DDTP のメールインターフェイスを使うなど、別の方法を考えた方が良いかもしれませんね。
- 公式サイトの更新との連係
  公式サイトのセキュリティ情報のページ[*47] にも DSA の日本語訳が掲載されますが、あまり連係は取れていません。 d-u に流してかなり経ってから転載されたり、公式サイトの方に先に別の日本語訳が載ったりすることがあります。

20.4 おわりに

やってみて分かりましたが、 DSA の翻訳に取り組むことで、セキュリティ情報をいち早く掴める上に情報セキュリティ分野の英語力が鍛えられます。

また、 DSA の原文はセキュリティチームのメンバーの PGP 署名付の確実な一次情報ですので、この発表がきっかけで、皆さんが原文に遡ることになれば幸いです。

[*46] http://ddtp.debian.net/ddtss/index.cgi/ja
[*47] http://www.debian.org/security/

参考文献

[1] Javier Fernández-Sanguino Peña, Alexander Reelsen, et al. *Securing Debian Manual "2.3 How does Debian handle security?"* Debian ドキュメンテーションプロジェクト, 2001–2007, http://www.debian.org/doc/manuals/securing-debian-howto/ch2.en.html#s2.3

[2] The Debian Project, *Debian* **社会契約** , 2004, http://www.debian.org/social_contract

[3] Raphael Geissert, *Debian Security Advisory DSA-2541-1* , 2012 , http://www.debian.org/security/2012/dsa-2541.en.html

[4] Nico Golde , *Debian Security Advisory DSA-2546-1* , 2012 , http://www.debian.org/security/2012/dsa-2546.en.html

[5] Moritz Muehlenhoff , *Debian Security Advisory DSA-2548-1* , 2012 , http://www.debian.org/security/2012/dsa-2548.en.html

[6] Raphael Geissert *Debian Security Advisory DSA-2549-1* , 2012 , http://www.debian.org/security/2012/dsa-2549.en.html

21　大統一 Debian 勉強会 2012 参加報告

DebianJP

21.1　大統一 Debian 勉強会とは

Debian JP では、東京と関西の 2 ヶ所を中心に、毎月 Debian 勉強会を開催しています。今回、それらをまとめて、大統一 Debian 勉強会としてイベントの規模を拡大して実施しました。

通常顔をあわせることのないメンバーたちが日本各地から一同に介し友好を深め、技術的な議論を戦わせます。

21.1.1　大統一 Debian 勉強会 2012

第一回目の大統一 Debian 勉強会は、 2012 年 6 月 23 日に、京都の京都大学で開催されました。毎月の勉強会でお馴染みの人たちはもちろん、日本各地から参加者があり、最終的には 100 名弱のイベントとなりました。

21.1.2　会場

京都大学の数学教室をお借りして、カンファレンス用に 2 部屋、ハックラボとして 1 部屋が用意されました。休日の大学ということもあって、構内は比較的落ちついており、勉強会の開催も違和感なくとけこんでいたように思います。

今回は、参加者向けのネットワークとして、無線 AP を用意して京都大学の Proxy 経由で外部に出ていくようにしていました。無線の調子は悪くなかったようですが、 Proxy の設定でつまづく人はそれなりにいたようです。また、 WiMAX ルータを使って Ustream 中継をする予定にしていましたが、やはりというかセッション中は自前の Wifi ルータを使う参

加者が多かったようで、接続が全く安定しませんでした。

21.2 スケジュールとイベント

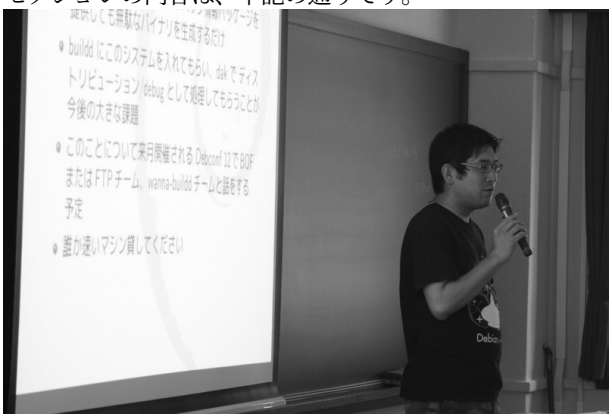
開催期間は 1 日間のみで、 Debconf のような催事はおりこまれていません。 Debian 勉強会としては初の試みになりますが、 2 トラックでのセッション運営を行い、合計 13 (加えて LT 5 件) のセッションが行われました。

セッションの一環として、おなじみのキーサインパーティも開催され、 28 名の参加者がありました。また、パーティには参加せず、個人的にキーサインを行う人もいたようです。

21.3 セッション

トラックを 2 つ設けたとはいっても、特にカテゴリの設定などはしませんでした。セッション内容も Debian 固有の開発話から Linux 全般に通じるネタ、組み込みやデバイス関連の Tips から便利なアプリケーションの紹介まで、千差万別でした。

セッションの内容は、下記の通りです。[*48]



21.3.1 TeXLive 2011(2012/dev) in Debian / 発表者 佐々木洋平

pTeX がドロップされるなど、大きな変更が予定されている Wheezy 向けの TEX 環境の現状紹介と、日本語処理面での導入、設定方法の説明がされました。

21.3.2 Linux-PAM の設定について / 発表者 西山和広

pam-auth-update など、独自のユーティリティに触れたりしつつも、 PAM の構成や設定について、基本的なところから解説されていました。

21.3.3 数学ソフトウェア使ってますか? / 発表者 濱田龍義

最近 MathLibre と改名した、数学ソフトウェアに特化したライブディストリビューションと、収録されている数学ソフトウェアを紹介するセッションでした。

21.3.4 ipython notebook とその周辺 / 発表者 本庄 弘典

最近 Debian にインストールされた ipython notebook とその周辺ツールである ipython qtconsole および ipython nodebook の導入方法、使い方について説明されました。

21.3.5 Debian と LibreOffice / 発表者 あわしろ いくや

OpenOffice から LibreOffice への変遷を軸に、 Debian でのパッケージング状況や翻訳活動の紹介などが説明されていました。

[*48] http://gum.debian.or.jp/ のタイムテーブルに、当日の発表資料と titanpad によるディスカッションの記録が保存されています

21.3.6 debug.debian.net / **発表者 岩松 信洋**

現在、 Debian のバイナリパッケージは、コンパイル時にデバッグ情報が削除されています。デバッグなどの際に必要になるこのデータを提供しようというプロジェクトについての検討報告でした。休眠していた同様のプロジェクトを復活させる方向で動いているそうです。

21.3.7 Rabbit: **時間内に終われるプレゼンツール / 発表者 須藤 功平**

どのようにプレゼンを行うのか、という観点も含めて Rabbit というプレゼンツールの背景にある設計思想を作者さん自らが解説する、というセッションでした。

21.3.8 U-Boot **についてあれこれ / 発表者 野島 貴英**

組み込み分野でよく利用されているブートローダである U-Boot の紹介と、 Barnes & Noble 社 Nook Color 向けに移植した例を紹介しました。

21.3.9 Debian Multiarch Support / **なかお けいすけ**

次期 Debian のリリースゴールの一つに挙げられている Multiarch Support について仕組みと現状について説明されました。

21.3.10 **家庭内** LAN **を高速に**! InfiniBand on Debian

eBay で安価に機材を入手する方法から、 Debian 上での利用方法、また高速な通信についての悦びまで、幅広く説明されていました。

21.3.11 Gentoo/Prefix on Debian

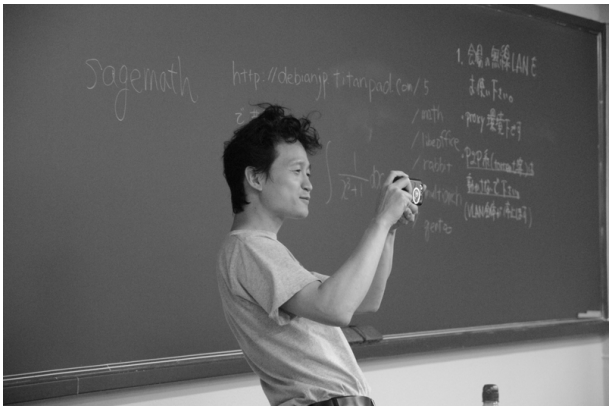
Gentoo Linux には、どんな OS にも Gentoo の皮をかぶせてしまう Prefix という仕掛け (平たく言うと chrooted gentoo) があります。それを一歩進めて、 Debian パッケージとの連携を深める試みについてのセッションでした。

21.3.12 Debian **でもマルチタッチ!**

Ubuntu が独自実装している関連パッケージを Debian にバックポートする方法の説明や、タッチパッドに指を何本も認識させるデモが行われていました。

21.4 Lightning talk

閉会直前に用意していた LT の枠も、事前の発表募集に反応が薄かったわりには当日の発表希望が複数あったため、なんとか無事に終えることができました。 LT のほとんどが Debian Developer によるものでしたが、これはさすがというべきか何というべきか....。



21.5 懇親会

閉会後、会場近くの居酒屋で懇親会を行いました。総勢 60 名弱の参加者があり、普段の勉強会とは一味違うもりあがりを見せていました。

21.6 次回の大統一 Debian 勉強会

今のところ、次回の予定はたっていません。今回を盛況のうちに終えることができたので、何らかの企画はなされるはずです。乞うご期待。

22 Debian Conference 2012 参加報告

岩松 信洋

22.1 Debconf とは

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

22.1.1 Debconf の歴史・経緯

Debian Conference `http://debconf12.debconf.org/` は Debian の開発者たちが一同に介するイベントです。通常顔をあわせることのないメンバーたちが一同に介し友好を深め、技術的な議論を戦わせます。過去の開催履歴を見てみると表 5 のようになります。

表 5 歴代の Debconf 参加者推移

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

22.2 ニカラグア/マナグア

22.2.1 行き方

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

22.2.2 会場

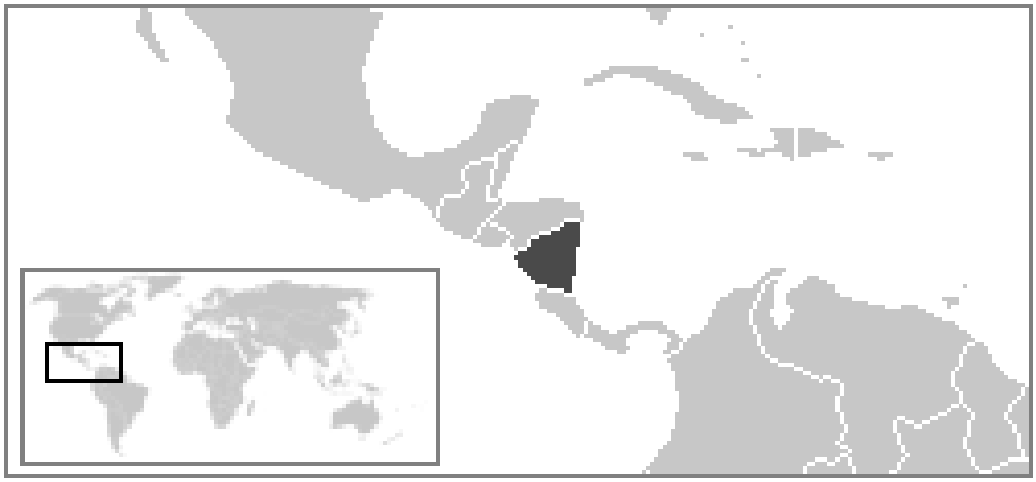


図 14　ニカラグア/マナグアの位置

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

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

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

- Hacklab: Hacklab はハッ ク専用の部屋です。


- 食堂: 食堂は外にテントを張って、そこで食事しました。


- 宿泊したホテル:

22.3 スケジュール

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

22.4 主となった討論

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

22.4.1 Bits from the Release Team

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

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

22.4.2 EFI BOF

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

22.4.3 Multiarch crossbuilding

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

22.4.4 Build Debian with another compiler

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

22.4.5 ARM port(s) update

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

22.4.6 AArch64 planning

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

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

22.4.7 FreedomBox Update

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

22.5 Debconf14

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

22.6 Daytrip

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

22.7 次回の Debconf

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

23 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. 9/29 に行われた Debian 6.0 のアップデートは何回目でしょうか。

☐ A 5
☐ B 6
☐ C 7

問題 2. DMUA フィールドがなくなり、 Debian Maintainer のアップロードが変更されます。今後、アップロードの際にどのように作業する必要があるか?

☐ A FTP master に電話
☐ B スポンサーに dak の処理を依頼する
☐ C 専用アップローダにアップロード

問題 3. IRC 経由で VCS リポジトリを監視するサービスで終了したものは?

☐ A ICPO
☐ B NPA
☐ C CIA

問題 4. Debian Policy メンテナに新しく入ったのは誰か?

☐ A Kei Hibino
☐ B Colin Watson
☐ C Charles Plessy

問題 5. Checksums-SHA1,SHA256 の取り扱いが変更になったが、どう変更されたか?

☐ A 今まで無視されていました。ごめんね。
☐ B オプションだったので、必須としました。
☐ C これらは廃止し、 SHA-512 のみにします。

問題 6. FTP master にあたらしく参加したのは

☐ A iwamatsu
☐ B ansgar
☐ C bdale

問題 7. pdiff で何が改善されたか

☐ A 最大 2 つの Diff をダウンロードすれば良いように変更になった
☐ B 一日 10 個づつ Diff を生成するようになった
☐ C Diff ってなにそれおいしいの?

問題 8. CTTE 573745 で何が決定されたか

☐ A Mattias Klose クビ
☐ B python 終了のお知らせ
☐ C みんな仲良くしようね

問題 9. 新しく Front Desk のメンバーになったのは

☐ A Kouhei Maeda
☐ B Iwamatsu
☐ C Jonathan Wiltshire

問題 10. debian installer 7.0 beta3 の新機能ではないのはどれか

☐ A ipv6

☐ B UEFI

☐ C grub2

問題 11. Debconf13 はどこで開催されるか

☐ A スイス

☐ B 日本

☐ C 中国

問題 12. debian-cloud は何をするメーリングリストか

☐ A 人をけむにまくため

☐ B クラウドサービスで Debian を利用する

☐ C エアリスー

問題 13. Official Logo が変更されたのはなぜか

☐ A 古い Official Logo が DFSG Free じゃなかったから

☐ B 時代に合わなくなってきたから

☐ C DPL の趣味

問題 14. codesearch.debian.net は何をするサービスか

☐ A 正規表現でソースを検索できる

☐ B ソースコードクレクレ

☐ C ブログサービス

24 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. B 6.0.6 です。
2. B
3. C KGB に移行。 ICPA: International Criminal Police Organization, NAP:National Police Agency, CIA: Central Intelligence Agency
4. C Andrew McMillan, Colin Watson, Manoj Srivastava が抜けた
5. B Bug#690293
6. B Ansgar が新しく参加しました。 mhy, joerg, ansgar の三人体制に
7. A apt-get update の遅さがマシになりますね。
8. C python のメンテナのコミュニケーション不足についての議論は結局みんな仲良くしましょうという結論になりましたね。
9. C 4 人になりました: Bernd Zeimetz (bzed) Enrico Zini (enrico) Jan Hauke Rahm (jhr) Jonathan Wiltshire (jmw)
10. C
11. A
12. B
13. A
14. A

『 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』および『 関西 Debian 勉強会』で使用された資料・ 小ネタ・ 必殺技などを一冊にまとめたものです。収録範囲は 2012/07〜2012/11 まで東京エリアは第 90 回から第 94 回まで (第 92 回は OSC 2012 Tokyo/Fall のため収録無し) および Debian パッ ケージング道場、関西エリアは第 61 回から第 65 回まで (第 66 回は KOF 2012 のため収録無し)。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 2012 年冬号

2012 年 12 月 31 日　　初版第 1 刷発行

東京エリア Debian 勉強会/関西 Debian 勉強会（ 編集・ 印刷・ 発行）