Grand Unified Debian

銀河系唯一のDebian専門誌

東京エリア/関西Ｄｅｂｉａｎ勉強会

あんどきゅめんてっど でびあん 2013 年冬号 2013 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 Debian topic update

野島 貴英

2.1 どこからの update?

2013 年 2 月 OSC Tokyo/Spring からのアップデートとなります。

2.2 Wheezy リリース!!

2013/5/4 Debian 7.0(Whezzy) リリース `http://www.debian.org/News/2013/20130504`

2.3 Wheezy 入手の国内 mirror

日本国内で入手するには、

- インストール用イメージについては
  - i386/amd64 Debian GNU/Linux 用イメージ
    `http://cdimage.debian.or.jp/`
  - Debian GNU/kFreeBSD 用イメージ
    `http://cdimage.debian.or.jp/kfreebsd.html`
- パッケージ群の ftp ミラーサイト
  `http://ftp.jp.debian.org/debian/`

2.4 Wheezy インストール方法

Wheezy のインストールについては、
Debian wheezy – インストールガイド
`http://www.debian.org/releases/stable/installmanual`
をご覧くださいませ。

2.5 Wheezy リリースノートについて

- Wheezy の変更点やら、
- Squeeze → Wheezy へのアップグレードのやり方と注意点

については、
「 Debian 7.0 – リリースノート」

http://www.debian.org/releases/stable/releasenotes
をご覧くださいませ。

2.6 さらに Wheezy アップデートの状況

- 2013/6/15 にて Wheezy の 1st アップデート (Debian 7.1)
  http://www.debian.org/News/2013/20130615
  33 個のセキュリティ修正と、 60 個のパッケージを更新。
  (kernel 側は数々のバグ fix 及び drm は 3.4.47 対応、 deboostrap の次期リリースの jessie 対応追加、その他パッケージはバグフィックスが主)
- 2013/10/12 にて Wheezy の 2nd アップデート (Debian 7.2)
  http://www.debian.org/News/2013/20131012 58 個のセキュリティ修正と、 102 個のパッケージを更新、 6 個のパッケージを廃止
  (kernel 側は 3.2.51 対応及び drm は 3.4.61 対応、 iceweasel17 未対応のパッケージを排除、その他パッケージはバグフィックスが主)

2.7 パッケージ数

- Wheezy(stable)
  バイナリパッケージ数: 35985 +7862
  ソースパッケージ数 : 17172 +2567

- Jessie(testing)
  バイナリパッケージ数: 38308 +2141
  ソースパッケージ数 : 18998 +1666

- Sid(unstable)
  バイナリパッケージ数: 40373 +1865
  ソースパッケージ数 : 20236 +1465

(増減数は前回 OSC の時の stable/testing/unstable からの差分)

2.8 開発者数

- 2013 年 02 月
  Debian Developer 約 1743 名
  Debian Maintainer 約 185 名
- 2013 年 10 月
  Debian Developer 約 1776 名 +33
  Debian Maintainer 約 183 名 -2

- 参考 2013 年 7 月時点で: 61 ヶ国 2012 年 6 月と比較して +2 ケ国
  日本人 50 名 （ アクティブメンバ 35 名)

2.9 Jessie

Wheezy も出たことですし、次の Debian のメジャーリリースのコードネームは Jessie です。

2.10 Jessie Freeze 予定

Freeze 予定: 2014 年 11 月 5 日 23:59 UTC

2.11 Jessie Release Goal

リリースゴールの提案が 2013/9/11-9/30 までの間募集されました。
将来リリースゴールとするかを検討中の一覧(すでに release チームにより検討から落とされたものは排除済み) :

- SysV 用の起動スクリプトを持つパッケージは、もれなく systemd にも対応する。
- ELF バイナリの堅牢化(Hardening) が不十分と報告されているパッケージをなおす (Wheezy から引き続き)
- debian/rules ファイルは CC/CXX フラグを外部から指定出来るようにする(要は CC=foo CXX=bar dpkg-buildpackage が出来るようにメンテする)
- clang も gcc の代わりにコンパイラとして利用できるようにする。
- すべてのパッケージを piuparts にて検証済みにする。
- クロスコンパイルが容易に出来るように整備する(Multiarch はこの一環)
- 140 個の基本的なコマンドのパッケージは、クロスコンパイラを用意すれば、そのままターゲットのアーキテクチャ用のバイナリパッケージが作れるようにする。
- ユーザが SELinux をより簡単に使えるようにする。
- より完全な UTF-8 対応

2.12 Jessie の Relese についての情報源

- Debian Relase Management
  `http://release.debian.org/`
- Bits from the Release Team (Jessie freeze info)
  `https://lists.debian.org/debian-devel-announce/2013/10/msg00004.html`

2.13 Debian パッケージ開発のお供にいくつか

ソースコードに関して充実してまいりました。

- ソースコード検索
  `http://codesearch.debian.net`
  (エンジンのソースは: `https://github.com/debiancodesearch/dcs`)

- ソースコード閲覧
  `http://sources.debian.net/`
- 参考:ソースパッケージに含まれるパッチトラッカ
  `http://patch-tracker.debian.org/`

2.14 DPL 選挙 in 2013

今年の DPL(Debian Project Leader) は、
Lucas Nussbaum さんが当選されました。
(ID: lucas) lucas のプロフィールについては↓
`http://en.wikipedia.org/wiki/Lucas_Nussbaum` 選挙時の声明↓
`http://www.debian.org/vote/2013/platforms/lucas`

2.15 最近の Debian のパッケージ開発のトレンド

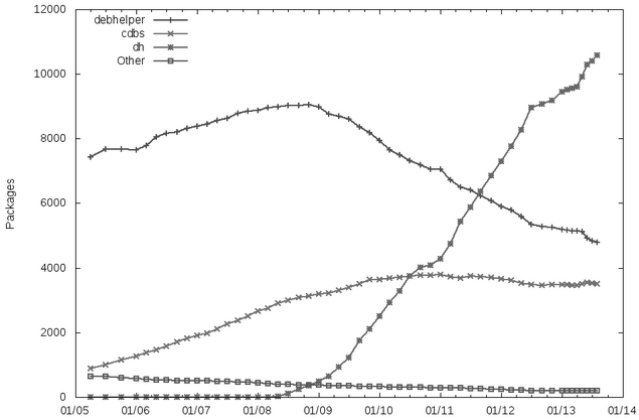


図 1 dh によるパッケージ開発が主流

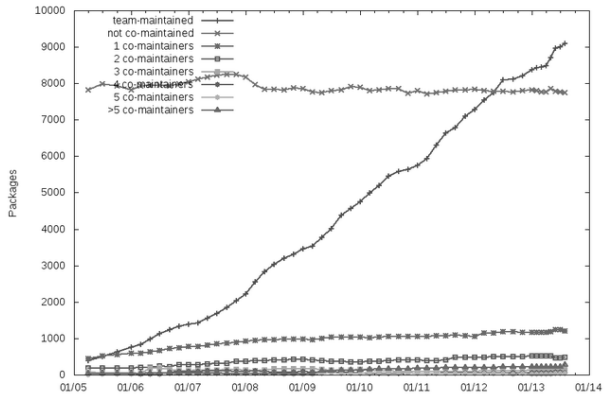


図 2 グループでパッケージ開発が主流に!

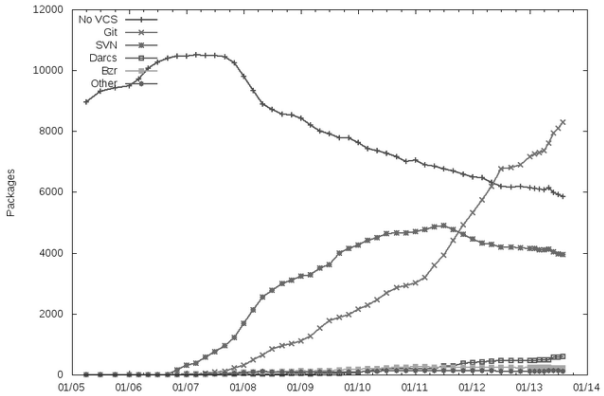


図 3 VCS は git が主流

2.16 Debian 20 歳

2013 年 8 月 16 日。 Debian 20 歳。

2.17 Debian Conference 2013

今回はスイスで 2013/8/11-8/18 まで開催されました。

2.18 Debian Conference 2013 の様子

- 会場でのスケジュール詳細↓
  `http://penta.debconf.org/dc13_schedule/`
- ビデオもあるよ!↓
  `http://www.irill.org/videos/debconf13`
- 一部だけど英語の字幕もあるよ? ↓
  `http://wiki.debconf.org/wiki/Videoteam/Subtitles`

2.19 Debian Conference 2013 ビデオ視聴 Tips

ヒアリング苦手な人は英語の字幕つかってみよう!

以下は totem での例:

Step 1. 字幕の使えるプレーヤー(例: totem) とか用意する。

```
$ sudo apt-get install totem
```

Step 2. 動画をもってくる。(例: Bit from DPL)

```
$ wget http://www.irill.org/media/debconf13/Bits_from_the_DPL.webm
```

Step 3. 字幕をとってくる

```
$ sudo apt-get install git
git clone http://anonscm.debian.org/git/debconfsubs/debconfsubs.git
( あるいは、 http://anonscm.debian.org/git/debconfsubs/debconfsubs.git
をブラウザで開いて snapshot というリンクから tgz ファイルを落としてくる)
```

Step 4. totem に Step 2. の動画を指定して起動して stop ボタンを押す。

Step 5. totem の「 表示」 →「 字幕」 →「 字幕の選択」を選択し、 Step 3. で取ってきた対応する字幕ファイル (.srt ファイル) を指定する。

Step 6. totem の再生ボタンをそのまま押すと字幕(英語) が現れる。

2.20 Debian Conference 2013 トピック紹介

とりあえず、見どころをいくつか

2.20.1 Bit from the Debian Project Leader

まずは、 “Bit from the Debian Project Leader” から、 (プレゼン資料: `http://www.lucas-nussbaum.net/blog/wp-content/uploads/2013/08/bits.pdf`)

1. Debian の紹介と、現在の Debian のコミュニティの様子について説明(割と宣伝な感じです)
2. 現在の Debian の課題と提案
   - Cloud の IaaS/PaaS/SaaS 環境の元でどうやって自由を獲得するのか?
   - テスト版(testing) を利用してローリングリリースとしたらどうか? (もちろん、安定版 (stable) はちゃんと残す)
   - いろいろマンパワーが足らないので、とにかく新しい開発者/貢献してくれる人を捕まえよう
   - Debian の運営について、 DPL の負荷減らしも含めて、もっとうまいこと改善したい。

テスト版 (testing) 使ったローリングリリースの件は、皆さん関心が大変高いようで、質疑応答がすべてローリングリリースの件になってました...

2.20.2 Lightning talk

Debconf13 の Lightning talk がおもしろかったのでいくつか...

- Coqelicot
  `https://coquelicot.potager.org`
  ※ Coqelicot はフランス語でひなげしの花の意味
  いわゆる Web ベースのファイル共有サービスのプログラムを作ったとのこと。特徴として、ファイルは全部暗号化されてストア(しかも暗号化キーはストアされない) 、時間がたてば自動で消去、消去の時には zero で埋めるなどの特徴がある。ライセンスは AGPL。
- notmuch
  `http://notmuchmail.org/` 大量のメールを高速に扱うためのソフトウェアの紹介。 vim/emacs のフロントエンドなどもある。タグづけにも対応。

- messaging bus system for debian infrastracture
  `http://www.fedmsg.com/en/latest/`
  Fedora PJ のインフラを支えるために作られたメッセージバスシステムの紹介。 Debian にもいかが?という内容。debian を支えるいろいろなインフラ上の仕組み(ftp キューとか、 bugreport とか) で発生する様々なイベントをこのバスシステムに流し込むと、いろいろなメディアに柔軟に接続することができる。
  (例: ftp キューでリジェクトされたら、 WEB にも出るとかのシステムが作りやすく なる)
  `http://www.fedmsg.com/en/latest/topology/` を観るとわかりやすい。また、
  繋げることができるようになったシステムの一覧↓
  `http://www.fedmsg.com/en/latest/status/` 発表長すぎて司会(ドラ娘) に止められたのが残念。

- dedup.debian.net の紹介
  `http://dedup.debian.net/`
  バイナリパッケージ内部のファイルの md5sum とか sha1 とかのハッシュをとり、他のバイナリパッケージに重複したものがあるかどうかを調べるサイトの紹介。例として、 freefem++,calibre のパッケージについて、どの程度他パッケージにも含まれているファイルがあるかをデモ。 gzip されたファイルも元ファイルの md5sum などを

取って比較できる。
例: http://dedup.debian.net/binary/freefem++
http://debdup.debian.net/compare/calibre/python-odf
(calibre と python-odf は見てのとおり重複しまくり)

- Skarphed - Webmanagement system
  http://www.skarphed.org/
  Web サイトを非常に簡単に構築できるシステムを開発したとのことで、そのデモの紹介。登録したサーバーに、非常に簡単に Web サイトを構築できる。 AGPL で提供とのこと。
- organaize of your life with org mode.
  http://orgmode.org/ja/
  emacs で動く、独特なテキスト文章作成環境のデモ。今回プレゼンすら、 org mode でやった。見ればわかりますが、 TODO 作成など、とても便利そうな印象を受けます。
- openhatch
  https://openhatch.org
  foss の貢献者らのための foss プロジェクトマッチングサービスの紹介。 Web 上で、 foss に貢献するために必要な基本的なスキルトレーニングをつむ、難易度に応じた foss プロジェクトを紹介する、イベントを紹介するなどのサービスが受けられる。

3 git-buildpackage 入門 again

佐々木 洋平

3.1 はじめに

最近では Debian パッケージの管理になんらかのバージョン管理システム (VCS) を使うことが一般的になってきました。VCS usage for Debian source packages[*1] によれば、現状ではソースパッケージの 70.44% がなんらかの VCS を使用しており、そのうち 65% が Git を使用しています[*2]。そんな訳で、Git によるパッケージ管理の知識はそろそろマストアイテムになりつつあるのかもしれません。

Debian パッケージを Git で管理するために使われるソフトウェアとして代表的なのが `git-buildpackage` です。これまでにも勉強会において `git-buildpackage` に関する発表が幾つか行なわれています。

- 東京エリア 2007 年:「 git-buidpackage の使い方」 by 上川純一
- 東京エリア 2008 年:「 バージョン管理ツールを使い Debian パッケージを管理する Git 編」 by 岩松信洋
- 関西 2011 年:「 vcs-buildpackage ～Git, svn 編～」 by 佐々木洋平
- 関西 2011 年:「 vcs-buildpackage ～Git の場合 (again)～」 by 佐々木洋平

当日は上記資料からの変更点ともう少し進んだ使い方について、参加者の事前課題をベースに、

1. `debootstrap` を例に、 native パッケージの場合の branch について
2. `ruby-bio` を例に、 Ruby 関連のパッケージでテストが転ぶ場合の対処について
3. `OpenStack` を例に、デカいパッケージの管理について
4. `yc-el` を例に、既存の vcs-buildpackage から Git への移行について

お話ししました。

git-buildpackage は精力的に開発が進められており、便利な機能が次々と追加されています。興味のある方は

```
$ man gbp
```

を読んでみて下さい。[*3]。

[*1] (declared) VCS usage for Debian source packages: `http://upsilon.cc/~zack/stuff/vcs-usage/`

[*2] 上記 URL[*1] によれば Subversion の利用が 28%、Git と subversion で全体の 93% を占めています。近年、Subversion から Git へ移行するチームが増えていることもあり、dgit 含め、今後は Git による Debian パッケージの管理が主流となるのかもしれません。

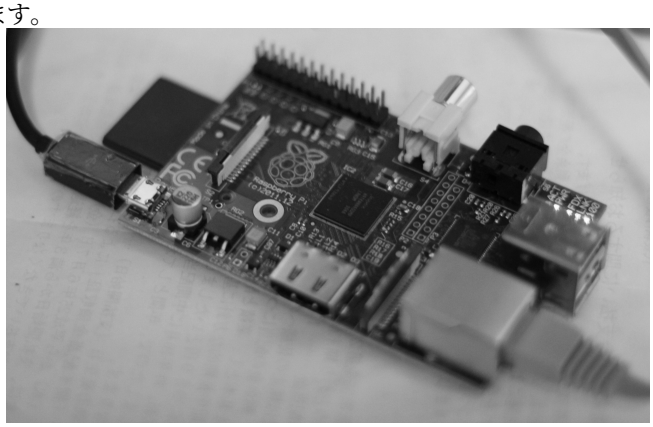
[*3] 現状は「 man には書かれているけれど…」というか。そのうち Debian Wiki の方にも追記/更新しておきたいな、と思っています。

4 raspbian on raspberry pi

上川 純一

4.1 はじめに

raspberry pi は市販されている安価な ARM のボードです。日本国内であれば RS Components から直販されています。



Debian が動くデバイスという観点でみると、 CPU として ARM1176JZF-S が搭載されているデバイスです。ARM11 (ARMv6) で、ハードウェアで浮動少数点演算ができる VFP が搭載されており、 ARMv7 以降の機能である NEON には対応していません。

HDMI 出力、イーサネット、 USB ホストなどがあるため、モニタとキーボードを接続すると普通のパソコンのように利用できるようになっています。電源は micro USB 端子で、 1A 以上が供給できるようになっている最近のスマートホン用の電源であれば流用できます。

4.2 Raspbian

Raspberry pi 用の Linux ディストリビューションの一つが Raspbian です。今回はそれを採用します。

4.2.1 Raspbian でなにがうれしいのか?

Debian そのものではなく Raspbian を利用する理由はなんでしょうか。インストール画面がカスタマイズされていますが、それ以外に何のメリットがあるのでしょうか。 Debian GNU/Linux の wheezy 時点の安定版でサポートされているアーキテクチャは 2 つあり、 armel と armhf です。

raspberry pi は armel で動作します。しかし、 armel は armv4 互換で、 armv6 の機能を生かしてくれません。armv6 では VFPv2 (ARM の浮動小数点ユニット) や SIMD (整数演算がストリーミング計算できる) 命令ができることになっているので、それを利用しない手はないです。一方ハードウェア浮動小数点命令を活用する設定になっている armhf は一方で armv7 以上でのみ動くので、 armv6 では動いてくれません。

そこで登場するのが Raspbian です。これは armv6, VFP 対応で Debian パッケージをコンパイルしなおしたディストリビューションです。

アーキテクチャ名は armhf となっており、 Debian の armhf アーキテクチャのパッケージをそのままインストールすることができますが、実行してもサポートされていない命令を実行した時点でエラーを出力するようです。

表 1　各 Debian arm アーキテクチャの違い

	dpkg アーキテクチャ表示	浮動小数点演算 ABI	命令セットアーキテクチャ
Debian armel	armel	soft	armv4
Debian armhf	armhf	hard	armv7
raspbian	armhf	hard	armv6

4.2.2　インストール

rapbian はインストール済の SD カードを入手するという事も可能ですが、そうでない場合は本体だけでインストールが完了する手順というのはおそらくないので、別途パソコンを用意してください。 SD カードを用意して、配布されている SD カードの起動イメージを dd で書きこみ、本体に挿入して起動するだけでよいです。 [1, 2]

HDMI でモニターに接続して、 USB でキーボードを接続して、イーサネット[*4]も結線しておきます。電源となる USB ケーブルを接続すると LED が点灯して起動します。しばらく起動メッセージが表示され、完了するとメニューが表示されます。

SD カードのイメージを書き込むとなるとパーティションのサイズが気になるかもしれませんが、ファイルシステムを SD カード全体の大きさへ拡張するなどの操作がメニューにあります。

locale の設定とかはあとで適当にやりましょう。

```
$ sudo dpkg-reconfigure locales
```

4.2.3　便利な小技

raspberry pi 自体に HDMI 出力や USB などがついているため、メイン開発機として利用することも可能ではありますが、通常は別のマシンなどがあって作業することになると思います。そのときに便利なのが avahi-daemon と sshfs です。

avahi-daemon をいれておけば mdns 対応のクライアントからなら “raspberrypi.local” というホスト名で接続できるようになります。

ssh の設定はするだろうから、 sshfs でファイルシステムをマウントしておけばファイルの共有が楽にできます。 sshfs は ssh の接続さえできればファイルシステムをマウントできるので便利です。個人的には最近はファイル共有は sshfs か git を使っています。

```
$ sshfs corei7.local:path/to/work ./mnt/
```

4.2.4　Raspbian での呼び出し規約

Raspbian（ armhf）では armel と違い、浮動小数点関連の関数の呼び出し規約自体も変わり、使えるレジスタとして r0-r31 だけでなく d0-d31 も利用できるようになります。

具体的なコンパイル例を見てみましょう。 C++ のコード例があった場合にこれがどうコンパイルされるかを見てみます。

[*4] 本体には物理的に RTC（ 時計）が搭載されていないため、起動時に現在時刻が設定されません。ネットワークにつながっている場合はネットワークから時間を取得することになるので通常は問題にはならないようですが、びっくりします。

表 2　1000000000 回 double の掛け算を行うループを実行するのにかかる時間(秒) 、一回試行

	armel	armhf	amd64(corei7)
-O1	4.5	4.6	0.35
-O0	134	130	2.6

```
double a = 1.1;
double b = 2.3;
cout << a*b;
```

armel: mfloat-abi=soft の場合のアセンブリの出力例。 double が r レジスタで扱われていて、ライブラリコールが行われています。

```
30:        e50b4018        str        r4, [fp, #-24]
34:        e24b1014        sub        r1, fp, #20
38:        e8910003        ldm        r1, {r0, r1}
3c:        e24b301c        sub        r3, fp, #28
40:        e893000c        ldm        r3, {r2, r3}
44:        ebfffffe        bl        0 <__aeabi_dmul>
```

armhf: mfloat-abi=hard のアセンブリ出力例。 ABI が変わり、 d レジスタで関数呼び出しの値が渡されるようになり、 vmul で掛け算が行われています。

```
30:   e50b4018        str     r4, [fp, #-24]
34:   ed1b6b05        vldr    d6, [fp, #-20]  ; 0xffffffec
38:   ed1b7b07        vldr    d7, [fp, #-28]  ; 0xffffffe4
3c:   ee267b07        vmul.f64        d7, d6, d7
40:   e59f0028        ldr     r0, [pc, #40]   ; 70 <main+0x70>
44:   eeb00b47        vmov.f64        d0, d7
48:   ebfffffe        bl      0 <_ZNSolsEd>
4c:   e3a03000        mov     r3, #0
```

4.2.5　hard float はどれくらい速度向上に貢献するのか?

気になったので掛け算の速度をベンチマークしてみたのですが、サブルーチンを呼び出すコードになっている armel と vmul.64 を直接呼ぶようになっている armhf の違いがよくわからんでした。もっと劇的に違うと思ったのに。

```
double mulbench(int iter, double a, double b) {
  double c;
  for (int i = 0; i < iter; ++i) {
    c = a * b;
  }
  return c;
}

int main(int argc, char** argv) {
  cout << mulbench(atoi(argv[1]), 1.2, 3.5) << endl;
  return 0;
}
```

4.3　raspberry pi にデフォルトで搭載されているセンサー

Raspberry pi は外部センサーを付けないと何もできない感じがしますが実は一部計測できるものもあります。電圧と CPU の温度がそれです。

温度を表示してみましょう。ミリ度 C で表示されるようです。今の温度は 55 度のようです。

```
$ cat /sys/class/thermal/thermal_zone0/temp
 55148
```

時系列で温度のグラフを作ってみました。

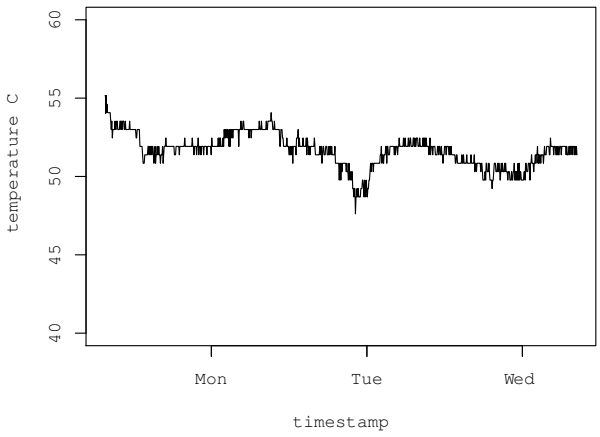


図 4　Raspberry pi CPU 温度 (℃) の時系列での変化

この温度センサーの値は 0.001 ℃単位 (m ℃?) で取得できますが、結果を見た限りでは実際に取得できる値は連続ではなく離散的で、 538 m ℃ 単位のようです。

4.4　おわりに

Raspberry Pi を購入してみて二ヶ月くらい放置していたのですが、いいかげんたちあげて見ました。しかしまだ長期的に何をさせるのかは考え中。

参考文献

[1] raspberry pi のソフトウェアダウンロードサイト (raspbian へのリンクが貼ってある) `http://www.raspberrypi.org/downloads`

[2] raspbian のサイト `http://www.raspbian.org/`

5 Debian linux kernel / armmp フレーバ

岩松 信洋

5.1 はじめに

Debian linux kernel の 3.9 から armhf アーキテクチャに armmp (ARM Multi Platform) フレーバが入りました。本稿では armmp の仕組みと対応方法について説明します。

5.2 arm multi platform とその仕組み

Linux kernel バージョン 3.8 ぐらいから arm multi platform（ 以下、 ARMMP） サポートが入りました。これは一つの linux kernel バイナリで複数の ARM Soc、ターゲットボードをサポートする仕組みです。今のところ、 mvebu（ Marvell SoCs）を筆頭に Versatile Express、 OMAP などがサポートされています。バージョン 3.11 ではほとんどの ARM SoC がサポートされる予定になっています。

複数 SoC やターゲットボードへの対応方法ですが、これはカーネル起動時に Device Tree (以下、 DT）と呼ばれるハードウェア情報を渡すことによって、カーネルを各デバイス向けに初期化し動作するようになります。

5.2.1 ARMMP カーネルの起動方法

通常 Linux カーネルを起動する場合、 zImage だけで起動しますが、 ARMMP や最近の ARM 向けカーネルは DT が必要になっています。 DT を絡めたカーネル起動方法は 3 つほどあります。起動方法別に Linux 上での操作(linux$が linux 上での操作）と U-Boot [*5]上での操作(u-boot$は U-boot 上での操作）を以下に紹介します。

1. zImage に dtb (DT blob。 DT のバイナリ) を付加する。そしてその zImage で起動する。
   まずカーネルの CONFIG_ARM_APPENDED_DTB が必要です。有効になっていない場合は有効にしましょう。
   そしてビルドされた zImage と dtb を結合します。

   ```
   linux$ cat arch/arm/boot/zImage arch/arm/boot/dts/armada-xp-openblocks-ax3-4.dtb  > arch/arm/boot/zImage+dtb
   ```

   上で作成した zImage+dtb をターゲット機器上にコピーします。以下の例は tftp を使って 0x2000000 にコピーしています。 0x2000000 はターゲットによって異なるので環境に合わせて変更してください。そして go コマンドにロードしたアドレスを指定して実行し、カーネルを起動させています。

   ```
   u-boot$ tftpboot 2000000 zImage+dtb
   u-boot$ go 2000000
   ```

   比較的新しい U-Boot では zImage をメモリからブートさせるためのコマンド、 bootz があります。カーネルイメージのチェックや初期化などを行うので bootz が提供されている場合はこちらを使うほうがよいです。

[*5] U-Boot は ARM 組込機器でよく利用されるブートローダ

```
u-boot$ tftpboot 2000000 zImage+dtb
u-boot$ bootz 2000000
```

2. 1. で作成した zImage を uImage に変換して起動する。
   以下は 結合したイメージを uImage 形式に変換しています。最初の `make uImage` では zImage を uImage に変換していますが、その後 zImage と dtb を結合(zImage+dtb) し、 `arch/arm/boot/.uImage.cmd` を使って結合したイメージを uImage (uImage+dtb) に変換しています。[*6]

```
linux$ make uImage
linux$ cat arch/arm/boot/zImage arch/arm/boot/dts/armada-xp-openblocks-ax3-4.dtb  > arch/arm/boot/zImage+dtb
linux$ ‘cut -f 3- -d ’ ’ < arch/arm/boot/.uImage.cmd | sed -e ’s/zImage/zImage+dtb/g’ -e ’s/uImage/uImage+dtb/g’‘
```

   起動に利用するフォーマットが uImage 形式の場合、 `go` コマンドではなく `bootm` コマンドのを使います。

```
u-boot$ tftpboot 2000000 uImage+dtb
u-boot$ bootm
```

   これらの方法は ARMMP ではない SoC 単体で DT を使う場合のみ有効です。カーネルがロードされるアドレスなどは SoC やボード毎に異なるのですが、 SoC 単体ではカーネルのコンフィグ情報としてこれらを持っているので上記の説明で uImage が作成できます。しかし ARMMP の場合は SoC 毎にこれらの値が異なるのでターゲット SoC 毎にこれらの値を変更する必要があります。 ARMMP で実行すると以下のようなエラーになるでしょう。

```
linux$  ‘cut -f 3- -d ’ ’ < arch/arm/boot/.uImage.cmd | sed -e ’s/zImage/zImage+dtb/g’ -e ’s/uImage/uImage+dtb/g’‘
  CHK     include/generated/uapi/linux/version.h
  CHK     include/generated/utsrelease.h
make[1]: ‘include/generated/mach-types.h’ is up to date.
  CALL    scripts/checksyscalls.sh
  CHK     include/generated/compile.h
  Kernel: arch/arm/boot/Image is ready
  Kernel: arch/arm/boot/zImage is ready
multiple (or no) load addresses:
This is incompatible with uImages
Specify LOADADDR on the commandline to build an uImage
make[1]: *** [arch/arm/boot/uImage] Error 1
make: *** [uImage] Error 2
```

   ロードアドレスなどの情報は mkimage [*7]で指定できます。[*8]
   `-a` でロードアドレス、`-e` でエントリポイントアドレスを指定します。

```
linux$ cat zImage foo.dtb > zImage+dtb
linux$ mkimage -A arm -O linux -T kernel -C none -a 0x2000000 -e 0x2000040 -n ’Linux-marvell’ -d arch/arm/boot/zImage+dtb \
arch/arm/boot/uImage+dtb
```

   [*9]

3. uImage と dtb blob 別に読み込んで起動する。
   ARMMP は 一つのバイナリで複数の ARM SoC、ボードをサポートすることが目的なので、上記の方法ではサポートできません。よって、 uImage と dtb blob を分けて起動させるのが理想です。 U-Boot の場合は カーネルを起動するコマンド `bootm` を使います。

```
linux$ mkimage -A arm -O linux -T kernel -C none -a 0x2000000 -e 0x2000040 -n ’Linux-marvell’ \
-d arch/arm/boot/zImage arch/arm/boot/uImage
```

```
u-boot$ tftpboot 2000000 uImage
u-boot$ tftpboot 3000000 dtb
u-boot$ bootm 2000000 - 3000000
```

   bootm コマンドの 第 1 引数は uImage がロードされているアドレス、第 2 引数は uInitrd (initrd イメージの uImage) がロードされているアドレス、第 3 引数には dtb がロードされているアドレスを指定します。-は 指定

[*6] 後述する mkimage を直接呼んでもよい
[*7] U-Boot 用のイメージを作成するツール。
[*8] arch/arm/boot/.uImage.cmd 内では mkimage を呼んでいる
[*9] カーネル開発者は./scripts/mkuboot.sh を使う事が多いかも。

なしを意味します。

ちなみに dtb の格納されてるアドレスはブートローダによってカーネル起動時の r2 レジスタに設定され、カーネルが指定されているアドレスからデータを読み込み起動する用になっています。これは Linux カーネルの Documentation/arm/Booting に書かれています。

5.2.2 カーネルモジュールの対応

カーネルモジュールも DT によって設定できます。必要なカーネルドライバを組み込みにしてカーネルを起動させるのもよいのですが、 ARMMP の場合カーネルが肥大化しますので、 SoC のコア部分は最低限の機能は組み込みに設定し、各デバイス用ドライバはモジュールにして initrd などからロードするようにするのがよいでしょう。 Debian の場合はサポートボード毎に zImage を用意せず、上記の方法で対応しています。

5.3 Debian でのサポート

Debian の armhf アーキテクチャの armmp フレーバ は上記で説明した機能を持ったカーネルをサポートするためのものです。今回 Plat'Home さん[*10] の Openblocks AX3 を Debian でサポートするために実装しました。 AX3 は mvebu という Marvell 製 SoC をまとめている SoC アーキテクチャとしてサポートされているのですが、これは ARMMP サポートのみで実装されているので、 Debian 側でも ARMMP サポートする必要がありました。また、他の今後 ARM カーネルは ARMMP に移行することが決定していたので、誰かやる必要があったというのも理由の一つです。

5.4 Debian でのカーネル提供と利用方法

Debian はカーネルイメージを uImage で提供していません。 uImage で提供した場合、カーネルがロードされるアドレスが固定値になってしまうので、複数のデバイスをサポートできません。 Debian では カーネルを zImage(vmlinuz) として提供しています。

しかしこのままでは U-Boot を使ったデバイスなどで使う場合に手間がかかるので、 Debian では flash-kernel[*11] パッケージでサポートしています。

flash-kernel は設定されたデータをもとにカーネルや initrd を U-Boot などで扱える形式に変換し、フラッシュメモリに書き込む機能等を提供します。

データの形式は以下のようになります。このデータは/usr/share/flash-kernel/db/all.db に記述されます。

```
Machine: Marvell Armada 370/XP (Device Tree)
Kernel-Flavors: armmp
DTB-Id: armada-xp-openblocks-ax3-4.dtb
DTB-Append: yes
U-Boot-Kernel-Address: 0x2000000
U-Boot-Initrd-Address: 0x0
Boot-Device: /dev/sda1
Boot-Kernel-Path: /boot/uImage
Boot-Initrd-Path: /boot/uInitrd
Required-Packages: u-boot-tools
Bootloader-Sets-root: no
```

flash-kernel を実行すると設定されているデータと起動しているカーネル情報(/proc/cpuinfo、または/proc/device-tree/model）を元にカーネルと inittd を変換し、インストールします。

[*10] http://www.plathome.co.jp/

[*11] http://packages.qa.debian.org/f/flash-kernel.html

```
linux$ cat /proc/cpuinfo | tail -3
Hardware        : Marvell Armada 370/XP (Device Tree)
Revision        : 0000
Serial          : 0000000000000000

linux$ flash-kernel
flash-kernel: installing version 3.10-1-armmp
Generating kernel u-boot image... done.
Installing new uImage.
Generating initramfs u-boot image... done.
Installing new uInitrd.
Installing new dtb.

linux$ ls /boot
System.map-3.10-1-armmp  initrd.img-3.10-1-armmp  uInitrd
config-3.10-1-armmp      uImage                   vmlinuz-3.10-1-armmp
```

flash-kernel によって作成されたイメージを使った起動方法はボード毎に異なります。 OpenBlocks AX3 の場合は以下の方法で SSD から起動できるはずです。

```
u-boot$ ide reset
u-boot$ ext2load ide 0 2000000 /boot/uImage
u-boot$ ext2load ide 0 3000000 /boot/uInitrd
u-boot$ bootm 2000000 3000000
```

5.5 終わりに

ARMMP の仕組みと、 Debian の対応方法について説明しました。

~~flash-kernel のデータはぷらっとホームさんと相談して入れようと思っているのでまだコミットされていません。~~
~~数日後にはパッチが BTS に上がるでしょう。 OpenBlocks A6 もサポートします。~~
~~また、エントリーポイントを設定する項目と~~
~~DT の model を指定する項目がないのでこれも対応する必要があります。~~
~~パッチはつくってあるので後日 BTS します。~~
~~あと、残作業としてはカーネル 3.10 では mvebu が動作しない~~
~~（他でも同じかも）のでパッチを当てる必要があります。~~
~~（コミット: faefd550c45d8d314e8f260f21565320355c947f）。~~
~~これも BTS します。ということでまだまだやることはたくさんあります。~~

既にこれらのパッチは取り込まれ、 flash-kernel で ARMMP と OpenBlocks AX3 がサポートされました。

6 puppet による構成管理の実践

倉敷 悟

6.1 概要

このセッションでは、まっさらな Wheezy 環境に puppet をセットアップし、実際にマニフェストを書いて、 Debian アーカイブの部分ミラーを構成させるところまで、実際に手を動かしながら実習していきます。

6.2 完成図の検討

まずは、構成管理としてやりたいことを検討します。今回のお題は、 Debian アーカイブの部分ミラーを構成する、ということになりますので、ざっくりと下記のような構成で検討してみます。

- アーキテクチャは amd64 のみ
- ディストリビューションは Wheezy (更新も含む)
- とりあえず必要最低限 (required) のパッケージだけをミラーする
- リポジトリの管理ツールとして reprepro を使う
- puppet は reprepro の環境設定のみ行う。ミラー実行は (とりあえず) 手動
- ミラーへのアクセスは http

細かいところは、多少は好みで変更してもらっても構いません。ただしミラー対象を増やしたいのであれば、回線がリッチな時をおすすめします。

上記に付随して、

- Web サーバは apache2 を使う
- Secure Apt のために GPG 設定が必要
- 配置場所は /var/www/debian で所有者 www-data にする

といったことも要件として検討しておく必要があります。

他にもあるかも知れませんが、おいおい加えていくこともできますので、ひとまずこれくらいの想定で進めます。

6.3 puppet のインストールと編集環境のセットアップ

まずは、適当なファイルにマニフェストを書き殴りましょう。

今回、 puppet はスタンドアローンでバッチ的に動作させる形で使います。そのため、 puppet のコマンドラインクライアントをインストールしてください。また、構成管理する OS 上でマニフェストの操作も行いますので、好みのテキス

トエディタをインストールします。emacs であれば、puppet-el [*12] というパッケージを使うと、予約語の色分けなどをしてくれるので便利です。作業を記録するためにバージョン管理システムもインストールしておきましょう。

```
$ sudo apt-get install puppet emacs puppet-el git tig

$ mkdir kdm201308
$ cd kdm201308
$ git init
```

6.4 全体像を下書きしてみる

最初に考えたおおよその完成イメージを実現するため、トップレベルのマニフェストを「こう書けたら嬉しい」という感じで下書きしてみます。

先程作成しておいたディレクトリで、site.pp というファイルを編集していきます。

```
$ mkdir manifests
$ emacs manifests/site.pp
```

```
# 必要になりそうなパッケージを class として列挙
class {
  'apache2':;
  'reprepro':;
  'gnupg':;
}

# reprepro のミラーディレクトリをよしなに設定（してほしい）
reprepro::mirror { '/var/www/debian':
  owner => 'www-data', group => 'www-data',
  architecture => 'amd64',
  distribution => 'wheezy',
  partial => 'required',
}

# apache のミラー公開用サイトを追加（してほしい）
apache2::site {
  'mirror':
    content => ' 後で';
}
```

要するに、やりたいことは、

- reprepro でいい感じのミラーを作る
- 作ったディレクトリを適当に Web アクセスできるようにする

ということですね。

とりあえず並べた class の要素はまだ存在しないため、このままでは実行できません。それぞれ作ってあげる必要があります。

ひとまず、作業を git に入れておきましょう。ログは好きなように書いてください。

```
$ git add site.pp
$ git commit
```

このあたりで、必要そうであれば、前回のおさらいも兼ねて puppet のマニフェスト書式について口頭で簡単なおさらいをします。資料としては前回の分 (2010/06) を参照してください。

6.5 必要な class を作る

さて、ここからが本番です。何はともあれ、不足している class 定義の枠と、インストールするパッケージの指定だけしてみましょう。

[*12] vim の場合は vim-puppet

```
class apache2 {
  package { 'apache2':; }
}

class reprepro {
  package { 'reprepro':; }
}

class gnupg {
  package { 'gnupg':; }
}
```

これだけでも、とりあえず必要なパッケージは入ります。凝ってみたいなら、リファレンスを見ながらパラメータを追加してみてもいいでしょう。

これだけでは、パッケージ固有の設定は debconf 任せになってしまいますので、設定を追加するために、都合のよいリソース定義が必要になります。まずは、全体像で書いたリソースを受ける枠をとりあえず作って……

```
define reprepro::mirror (
  $owner,
  $group,
  $architecture,
  $distribution,
  $partial
  ) {

}

define apache2::site (
  $content
  ) {

}
```

それぞれの内部をどのように構成するか考えてみましょう。このセッションでは、かなりやっつけ感の高い構成で流してしまいますが、実際にマニフェストを書くときは、構成管理しようとしている対象のパッケージの動きをしっかり観察するようにしてください。

ここにファイルを置くためには、このパッケージが入っていなくてはならない、このファイルはこういうパーミッションで置かれている、このリソースの後にこういう処理が必要だ、パッケージの削除が失敗しないか、などなど……。[*13]

構成管理をしっかり書こうと思えば、その対象をよく知っておく必要があります。そこの手間が省けるツールではないので誤解のないようにしましょう。ありもののレシピ使い回せばいいんじゃんウェーイ、という考え方もご時世としてはありなのかも知れませんが、その後の運用を誰かに押しつけてトンズラできる環境でないのならばオススメしません。

さて、構成管理ツールが流行するよりも前から、 dpkg はその一部を担っているという経緯もありますので、

- 一部、やってることが puppet と被る (プロセスの起動制御など)
- 設定方針が debian policy で決まっているため、やりたい設定と矛盾する場合がある

といったあたりに注意して、設定を煮詰めていきましょう。

6.5.1 apache2

実は、今回はいろいろな前提の上にのっかっているため、 apache2 に追加の設定は不要です。デフォルトのままで、reprepro によって作成されるミラーにそのままアクセスできるはずです。

ただ、それだとさすがに味気ないですから、簡単にサイト設定の仕組みを追加してみましょう。これは Debian においては、/etc/apache2/sites-{enabled|available}/ ディレクトリと a2ensite/a2dissite コマンドによって制御されている部分ですね。

きっちり書くなら、 available ディレクトリに file リソースの content でファイルを配置し、 ensure パラメータの内容によって enable/disable を制御する exec リソースを書いておく、といったことをすればいいと思います。ただし、今回は面倒なので単純に「 content の内容を直接 sites-enabled につっこむ」だけにしておきます。

すると、おおよそ次のようになるかと思います。

[*13] パッケージシステムとの整合をとる話は、それだけでセッションが成立する程度にはトピックがあるので、また別の機会に。

```
define apache2::site (
  $content
  ) {

  file { "/etc/apache2/sites-enabled/${name}":
    content => $content,
    require => Package['apache2'],
    notify  => Service['apache2'],
  }
}
```

設定ファイルを変更したら、プロセスも再起動して欲しいので、 notify も追加しました。この Service はまだ定義していないので、 class の方に対応する定義を書いておきましょう。

```
class apache2 {
  package { 'apache2':; }
  service { 'apache2':; }
}
```

これで、 apache2 の

- パッケージを入れ
- 適当なサイト設定を行い
- サービスを起動する

簡単な class が出来ました。

後は、実際に流しこむサイト設定を用意する必要があります。今回は、さきほど書いたようにデフォルトのままでも動くので、そのままデフォルト設定をパクっておくことにします。

一度 apache2 パッケージをインストールし、 sites-available/default ファイルをコピーしましょう。

```
$ sudo apt-get install apache2
$ cp /etc/apache2/sites-available/default site-mirror.erb
$ sudo apt-get purge apache2 && sudo apt-get autoremove
```

このファイルをテンプレートとして content に渡してあげたいので、所定の場所に配置した上で、マニフェストから読みこむようにしておきます。

```
$ mkdir templates
$ mv site-mirror.erb  templates

apache2::site {
  'mirror':
    content => template('site-mirror.erb');
}
```

これで、 apache2 については一応完成です。このあたりで、 git commit しておきましょう。

パラメータの渡し方、ファイルの指定方法などは、選択肢がいくらでもあって、正解はありません。かけられる時間や、どれくらい類似度の高い構成を繰り返すのか、などマニフェストを作成する状況にあわせて調整するようにしてください。[*14]

6.5.2 reprepro

今回は reprepro 自体の使い方は主眼ではないので、簡単に紹介だけしておきます。

reprepro は、指定したディレクトリに Debian のミラーを構成するツールです。指定ディレクトリに決まったフォーマットで設定ファイルを用意しておけば、柔軟にミラーを構成できます。そのため、ちょっと手元でとか社内利用などの用途で、気軽にミラーを用意することができます[*15]。

イメージできるほど対象のソフトウェアに馴染んでいない場合は、まず puppet 抜きで動作させてみて、挙動を把握するようにしましょう。ともあれ、手動で reprepro をインストールします。

[*14] 実例としてあげている処理は、考慮の足りない、ダメなやっつけ仕事の例ですので、反面教師として頂ければ……

[*15] パブリックなフルミラーを用意したい場合は、別途専用のツールがあります

```
$ sudo apt-get install reprepro
```

reprepro を動作させるために、 conf/distributions と conf/updates、 2 つの設定ファイルが必要になります。ミラーのファイル置き場にするディレクトリに移動してください。ここでは、 wheezy の部分ミラーを作成しますので、ひとまず下記の内容でファイルを作成しましょう。

distributions には、公開する対象 (sources.list でディストリビューションコードネームとして参照されるもの) を指定します。今回の例では wheezy となります。

```
Codename: wheezy
Architectures: amd64
Description: wheezy-mirror
Components: main contrib
Update: wheezy wheezy-updates wheezy-security
```

updates には、ミラーに含めるパッケージを取得してくる上流のリポジトリを指定します。

```
Name: wheezy
Method: http://ftp.jp.debian.org/debian
Suite: wheezy
Components: main contrib
Architectures: amd64
VerifyRelease: AED4B06F473041FA
FilterFormula: Priority (==required)

Name: wheezy-updates
Method: http://ftp.jp.debian.org/debian
Suite: wheezy-updates
Components: main contrib
Architectures: amd64
VerifyRelease: AED4B06F473041FA
FilterFormula: Priority (==required)

Name: wheezy-security
Method: http://security.debian.org/debian-security
Suite: wheezy/updates
VerifyRelease: AED4B06F473041FA
FilterFormula: Priority (==required)
```

Secure APT 用の公開鍵は、次のようにしてインポートしておきます。

```
$ gpg --recv-key AED4B06F473041FA
```

これによって、このミラーが上流からダウンロードする時に、公開鍵による確認が行なわれます。

設定が終わったら、正しく動作するか確認します。

```
$ reprepro -V update wheezy
```

少し時間がかかりますが、指定したミラーからファイルをダウンロードするはずです。終わったら、ノードの /etc/apt/souces.list を少しいじって、 apt-get からアクセスできるかどうか確認しておくといいでしょう。

さて、ここまでで reprepro が動作する設定ができあがりましたので、これを puppet のマニフェストに移していきましょう。

一旦、 reprepro で作成したファイル類を全て削除します (別にしなくても構いませんが、ここまでのテスト内容を掃除するためです)。

```
$ sudo rm -r /var/www/debian
```

先程作成した site.pp を開き、設定ファイルを流しこむ方法を考えます。パラメータをテンプレートで受けるなど、やり方はいくつかありますが、ここでは単純に、ファイルをそのまま置くだけとしておきます。

ミラーは、配置するディレクトリを変更すれば、ノード内に複数作成することができる (べきな) ので、クラスではなく、リソースとして定義します。

```
class reprepro {
  package { 'reprepro':; }
}

define reprepro::mirror (
  $distributions,
  $updates,
  ) {
  # ミラーの配置ディレクトリを作成する
  ## リソース名としてフルパスが渡されてくる想定
  file { "$name":
    ensure => directory,
  }
  -> file { "${name}/conf":
    ensure => directory,
  }
  -> file { # 設定ファイルの名前はパラメータとして受け取る
    "${name}/conf/distributions":
      content => $distributions;
    "${name}/conf/updates":
      content => $updates;
  }
}
```

パラメータ指定をやめてファイルを直接与えることにしたので、呼び出し元も変更する必要があります。

```
# reprepro をよしなに設定してほしい
reprepro::mirror { '/var/www/mirror':
  owner => 'www-data', group => 'www-data',
  distributions => ' 上で作成したファイルの内容をペースト
最後に改行が必要なので注意
',
  updates       => ' こちらも同様に
'
}
```

ここまでで、 puppet を実行することで、 reprepro の設定ファイルが準備され、後はコマンドを実行するだけ、という状態になります。忘れないように、 git commit しておきましょう。

最終的には reprepro コマンドの実行も puppet にやらせるといいのですが、話が広がってしまうので、今回はここまでとしておきます。

6.6 マニフェストの適用とデバッグ

さて、ここまでで作成したマニフェストを実行すると、

- apache2 がセットアップされて、ミラーのディレクトリが公開される
- ミラーのディレクトリに reprepro の設定が用意される

はずですので、早速試してみましょう。まずは、 dry run から。

```
$ puppet apply -v /etc/puppet/manifests/site.pp --noop
```

スペルミスや、記号抜けなど、書式違いがあれば puppet が教えてくれますので、出力内容を確認してみてください。

問題なさそうなら、 −−noop を削って、そのまま実行しましょう。

6.7 マニフェストの整理

puppet は、モジュールという単位でマニフェストを分割することができます。

今回は、さほど大きなマニフェストを書いているわけではないので、 site.pp だけでも十分見通しがききますが、 1 ファイルにダラダラ書いていくにも限度があります。ある程度大きく育ってきたら、マニフェストを分割しましょう。

もし実例が見たい、ということであれば、後半部分でやりますのでリクエストをください。

6.7.1 パッケージモジュール

モジュールをどういう単位で切り出すかは、個人の好みもあると思いますが、ここではとっかかりとして「パッケージ単位」をおすすめしておきます。

では、パッケージに対応した puppet モジュールの置き場所を決めましょう。例えば、

```
$ sudo  mkdir -p /etc/puppet/modules/
```

などとします。このディレクトリ以下に、モジュール名のディレクトリを作成していくことになりますが、 Debian での利用を考えると、「 ソースパッケージ名」を使うといいでしょう (バイナリパッケージが細かく分割されていたりするので)。

例えば、 apache2 のモジュールを作るのであれば、

```
$ mkdir -p /etc/puppet/modules/apache2/manifests
$ emacs /etc/puppet/modules/apache2/manifests/init.pp
```

のようにします。「 モジュール名/manifests/init.pp」はルールなので、別のファイル名にすることはできません。とりあえず、先程作成した、 apache2 クラスの内容をこちらに移動させてみてください。

パッケージモジュールの内容を決める上では、次のことに注意するといいでしょう。

- パッケージ固有の内容に絞り、サイト固有の設定は可能な限り含めない
- 後で再利用できるようにパラメータを決める

6.7.2 サービスモジュール

パッケージモジュールを切り離した後は、ノード/サイト固有の設定が site.pp に残っている状態になっているはずです。そのうち、今作業しているノードだけでなく別のノードでも再利用しそうなロール (うちでは Web サーバはいつもこの設定で作るようにしてるんだ、とか) も、可能であればモジュールとして切り出しておきましょう。

puppet から見ると、用途に関係なくモジュールはモジュールなのですが、パッケージモジュールと区別するために、便宜的にサービスモジュールと呼んでおきます。 puppet の書式ガイドライン的には、モジュール名として頭に s_ をつけることになっています。

サービスモジュールはサイト固有の内容になりますので、パッケージモジュールほど境界を気にしなくてもいいでしょう。

例えば、部分ミラーのロールを作るのであれば、

```
mkdir -p /etc/puppet/services/s_mirror/manifests
emacs /etc/puppet/services/s_mirror/manifests/init.pp
```

として、今回作成した reprepro の設定内容自体や、 apache2 のサイト設定などを放りこむところからはじめるといいと思います。

6.8 ここからの育て方

ひとまず、今回のセッションで Debian アーカイブのローカルミラーがお手軽に作れるようになっているハズでので、次のステップとして、

- reprepro の設定ファイル指定がダサすぎるのでなんとかする
- ローカルミラーを参照するように sources.list を設定する
- ローカルミラーに独自パッケージを追加できるように reprepro の設定を追加する
- 手動のままだった reprepro のコマンド実行をリソースに含める
- reprepro のコマンド実行を cron に登録してみる
- germinate を使って必要なパッケージを指定してミラーの対象にする

といった拡張からはじめていくと、とっかかりやすいのではないかと思います。公式のリファレンスを眺めながら頑張ってみましょう。

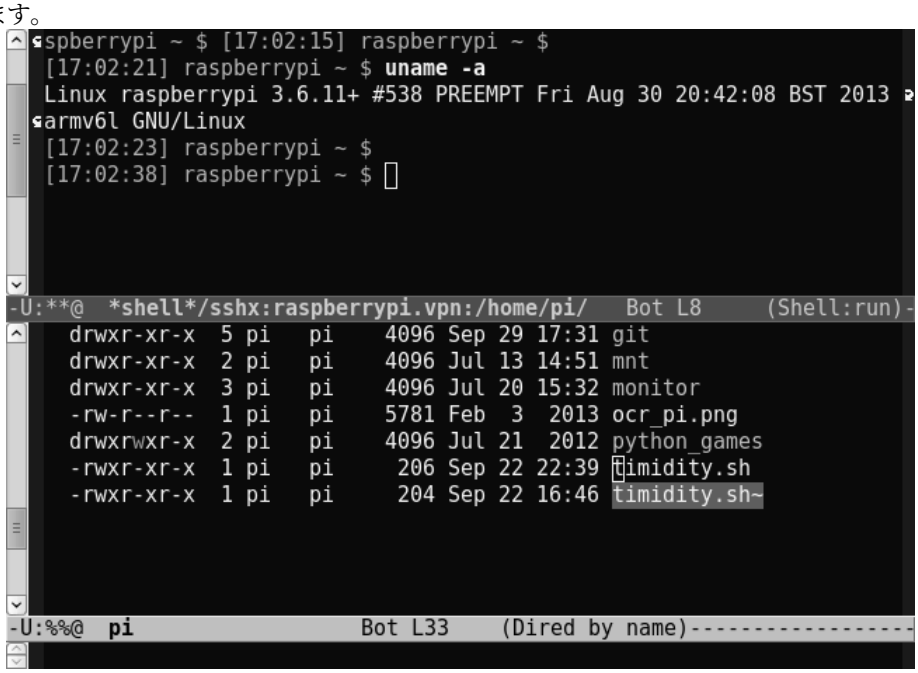
7 tramp 入門

上川純一

7.1 emacs でリモートファイル編集する Tramp のすすめ

リモートサーバでファイルの編集はどうしていますか? mosh を使っている? sshfs を使っている? いろいろあると思いますが mosh を使うとローカルのファイルと扱いが変わってしまい、リモートの vi だったり emacs を使って編集することになり、設定の同期がほぼ不可能になります。一方 sshfs だとローカルファイルのように扱うことができるのですが、root などの別のユーザ(root?) としてファイルの編集ができにくかったりしますし、リモートホストでコマンドを実行しようとすると別途 ssh でログインして作業することになり、 ssh セッションをひらきながら sshfs で実行し、ローカルのパスとリモートのパスの違いを意識しながら作業することになります。

emacs ユーザの方々に朗報です。 tramp とは `/sshx:hostname:path/to/file` という特殊なファイル名を指定すると透過的に ssh を使ってリモートのファイルを取得して編集できるようにしてくれる仕組みです。また、 dired でリモートのディレクトリのファイル管理もローカルと変わらないように行えます。さらに便利なのは `M-x shell`, `M-x compile` などリモートでコマンドを実行するコマンドを利用すると hostname に ssh でログインして path/to/file をカレントディレクトリとした状態でコマンドを実行してくれるところです。個人的には mosh などを利用してシェルのコマンドを実行するより、 emacs のローカルバッファでコマンドラインを編集して確定時にリモートに送信するスタイルが気に入っています。



7.2 リモートホスト側の設定

ssh で接続される側のの設定ですが、 tramp はシェルのセッションの入出力を利用し、機械的にプロンプトを検出するので、プロンプト(PS1 など) をカスタマイズしていると動かない場合があります。 tramp のために `.bashrc` はシンプルにするといいかもしれません。

例えば、 raspbian のデフォルトは色がつきまくってたりしてファンシー過ぎてうまく動きませんでした。

7.3 ssh の設定とチューニング

ローカルの ssh の設定ファイル `~/.ssh/config` に設定したほうがよいものを紹介します。マニュアルは `man ssh_config`[2] で参照できます。

7.3.1 ControlMaster で接続を再利用

手元ではこんな設定にしています。

```
ControlMaster auto
ControlPersist 120
ControlPath ~/tmp/ssh-%r@%h:%p
```

ControlPersist に指定した秒数間、 ssh で接続する際にデーモンプロセスが起動して、コネクションを張りっぱなしにし、コネクションを再利用してくれます。

手元では、 ControlPath を明示的に指定しています。ホームディレクトリ以下の tmp 以下にしているのはファイル名が予想可能になるので衝突を防ぐためです。

ControlMaster を Auto にしておくとすでにコネクションが開いていない場合はデーモンをたちあげて、もしコネクションが開いている場合にはそれを利用するようになります。

リモートで true コマンドを実行する速度を計測してみたところ随分高速になり (図 3)、 ping time の二倍くらいになってくれるようです。

command	time(ms)
ping	271
ssh connection sharing on	544
ssh connection sharing off	4070

表 3　ping test against wagner.debian.org

7.3.2 keep-alive

tramp はネットワーク接続の切断を ssh プロセスの生死で検出します。 ssh はネットワーク接続がきれたりしてもすぐにはそれを検出しないので、キープアライブを送信するようにします。

```
ServerAliveInterval 3
ServerAliveCountMax 5
```

本当はパソコンがサスペンドから復活して新しいネットワーク接続につながったら ssh プロセスに再接続してほしいのですが、その方法をまだ発見できていません。

7.3.3 その他の設定

Mac の Rendezvous とか Linux の Avahi とか設定しているとホスト名.local という名前で名前解決ができるようになりますが、そうすると IP アドレスは DHCP だと一定ではないため、 IP に対しての鍵が違うと怒られます。その場合はホスト IP をチェックしないようにするとよいでしょう。

```
Host *.local
 CheckHostIP no
```

あとデフォルトではいろいろな鍵を試したりする設定になっていますが、ついでに鍵を指定して公開鍵認証にしておくとよいんじゃないでしょうか。

```
Host raspberrypi.vpn
 Hostname そのホストの IP アドレス
 IdentityFile ~/.ssh/ssh-keygen で作ったファイルのパス
 IdentitiesOnly yes
```

7.4 tramp の.emacs での設定とチューニング

.emacs にほとんど設定しなくてもデフォルトの設定でそれなりにうごきます。マニュアルは info 形式で提供されています [1]。

tramp では/sudo:root@hostname:/ のような形式でリモートホスト上で sudo で別ユーザになってからファイルにアクセスするように指定することが可能です。ただ、そのためにはリモートホストへの到達手段を指定する必要があります。たとえば、ローカルネットワークにあるホスト *.local で sudo をサポートしたいという要望であれば、次のような設定を追記しておきます。

```
(add-to-list 'tramp-default-proxies-alist
     '("\\.local\\'" "\\`root\\'" "/ssh:%h:"))
```

あと、デフォルトの設定だとリモートアクセスしているというメッセージがステータスに表示されるのですが、正直邪魔なのでそれを消すのもよいかと思います。

```
(custom-set-variables '(tramp-verbose 1))
```

参考文献

[1] “TRAMP User Manual”, info page, http://www.gnu.org/software/emacs/manual/html_node/tramp/index.html#Top

[2] “ssh_config(5)” manual page,

8 Linux とサウンドシステム

坂本 貴史

8.1 ALSA の概要

- Advanced Linux Sound Architecture
- ALSA は Linux のためのサウンドデバイスを開発するプロジェクト
- 1998 年あたりに始まる
- Linux ディストリビューションおよび Android が ALSA を利用してデバイスとの入出力を行う
- 主要な開発者は Linux ディストリビューター、半導体メーカー、 Android メーカーに在籍

8.2 音声入出力に関係するハードウェアとその制御及びデータ

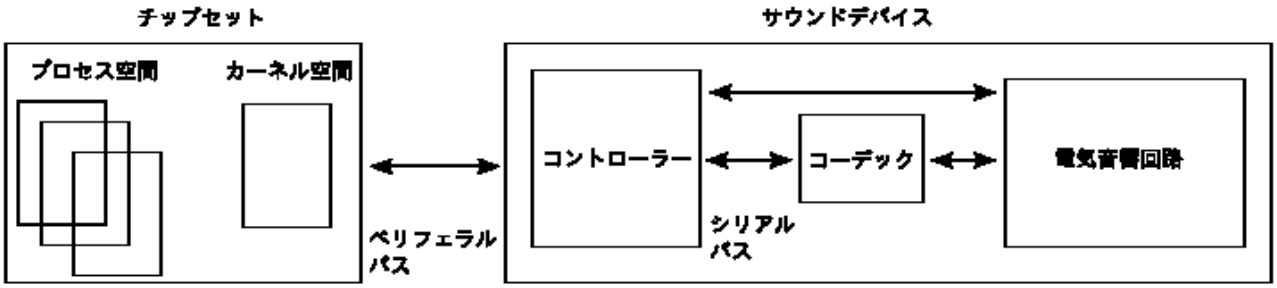


図1. 簡易モデル

8.3 ALSA のユーザー/カーネル空間のインターフェイス

- ALSA は Linux 専用のサブシステム。 Unix ライクにデバイスノードへのファイル操作を行ってデバイスを操作する
- サウンドデバイスはキャラクター型のデバイス。`/dev/snd` 以下に設けられる。
- open(2)/read(2)/write(2)/close(2) だけでデバイスを制御するのは難しい。
- ioctl(2) で柔軟な操作を行う。 libasound2 で専用 API を提供。
- procfs の`/proc/asound` 以下のノードで ALSA の基本情報を提供している

8.4 Linux カーネル/ドライバーと ALSA

8.4.1 デバイスクラスと ALSA

- Linux にはデバイスクラスがある。 soundcore モジュールが sound クラスを提供する。 ALSA は sound クラスのドライバー。
- ALSA Core は各ドライバーに対し、以下の機能を提供
  - ユーザー空間とのインターフェイスを提供する API
  - Linux の driver/base(sysfs 含む) に対する共通 API
  - Linux の procfs に対する共通 API
  - Open Sound System との互換レイヤ
  - その他ヘルパー関数
- ALSA のドライバーは、 Linux の各バスドライバーが提供する API を利用し、 ALSA のインターフェイスとデバイスを中継する役割を果たす

8.5 ALSA のカーネルランドコンフィグレーション

- ALSA のカーネルランドは Linux のローダブルモジュールとして実装されている
- modprobe を使い、ローダブルモジュールにオプションを渡すことができる

8.6 ALSA のユーザーランドコンフィグレーション

- libasound2 は設定ファイルのパース機能を持つ
- libasound2 はシステムレベル、ユーザーレベルでランタイム設定を変更できる
- libasound2 はプラグイン構造を持つ

8.7 ALSA のアプリケーション

- alsa-utils、 PulseAudio、 Jack Audio Connection Kit

8.8 圧縮データと gstreamer/libav

- サウンドシステムは基本、 PCM データを扱うよう作られている
- 圧縮データから PCM 標本を取り出す処理をどこかで入れる必要がある

8.9 Android と tinyalsa、 AudioFlinger

- Android はカーネルランドに Linux カーネルを使っていて、 ALSA を利用している
- ユーザーランドは libasound2 ではない tinyalsa を使う

8.10 その他の話題

- salsa、 COMPRESS/tinycompress、 Audio Visual Bridge (AVB)、 Firewire と ALSA

9 ALSA のユーザーランド解説

坂本 貴史

- ライブラリ、設定ファイル、プラグインなどがある
- ドキュメントはパッケージ「libasound2-doc」で「/usr/share/doc/libaound2-doc/html/」以下へインストールできる
- Android は独自にユーザーランドの実装を持っている

9.1 ライブラリ

- asound
- 設定ファイルのパース機能を持つ
  - システムレベル、ユーザーレベルでランタイム設定
- プラグインを持つ
  - 任意の機能を付与できる

9.2 PCM ノード

- ランタイムに設けられる。
- aplay -L あるいは arecord -L で一覧を取得できる。
- PCM のキャラクターデバイスを隠蔽し特定の機能を付与したもの
- この特定の機能を付与するために用いられるのが PCM プラグイン
- 設定ファイルが PCM ノードを設けている
- ライブラリの PCM インターフェイスでハンドルを開く時に指定する

```
$ aplay -L
default
    Playback/recording through the PulseAudio sound server
sysdefault:CARD=Intel
    HDA Intel, ALC889 Analog
    Default Audio Device
front:CARD=Intel,DEV=0
    HDA Intel, ALC889 Analog
    Front speakers
surround40:CARD=Intel,DEV=0
    HDA Intel, ALC889 Analog
    4.0 Surround output to Front and Rear speakers
surround41:CARD=Intel,DEV=0
    HDA Intel, ALC889 Analog
    4.1 Surround output to Front, Rear and Subwoofer speakers
```

```
surround50:CARD=Intel,DEV=0
    HDA Intel, ALC889 Analog
    5.0 Surround output to Front, Center and Rear speakers
surround51:CARD=Intel,DEV=0
    HDA Intel, ALC889 Analog
    5.1 Surround output to Front, Center, Rear and Subwoofer speakers
surround71:CARD=Intel,DEV=0
    HDA Intel, ALC889 Analog
    7.1 Surround output to Front, Center, Side, Rear and Woofer speakers
```

9.3 設定ファイル

- ALSA 特有の記法で書かれている
- システムレベルは/usr/share/alsa 以下にある
  - /usr/share/alsa/alsa.conf の「 defaults.namehint.extended off」が、 hw/plughw/dmix/dsnoop の表示を抑制
- ユーザーレベルは「 .asoundrc」をホームディ レクトリに配置

9.4 プラグイン

- プラグインには 2 つのタイプがある

PCM タイプ PCM ノードに機能を追加する

MIXER タイプ Control ノードに機能を追加する

- aplay -L でいろんなノードが見えるのは PCM プラグインのおかげ。

front フロント出力のためのもの

surroundXXX サラウンド出力のためのもの

hw 標本化周波数、量子化ビット数、チャンネル数などをちゃんと指定する必要がある

plughw 上記をよしなに設定してくれる

pulse 出力を PulseAudio に流しこんだり、入力を PulseAudio から受けたりする

dmix 複数の出力をひとつのストリームに合成する

dsnoop ひとつの入力を複数のストリームにする

default ALSA の配布するパッ ケージでは、 dmix/dsnoop を入出力スレーブとしている

- その他のプラグイン

bluetooth Bluez を利用して音声機能を持つ Bluetooth デバイスを使う

ladspa LADSPA というフレームワークのプラグインを使う

9.5 PulseAudio

- 複数のアプリケーショ ンの出力を束ねたり、入力を分割したりする
- 最近のデスクトッ プ環境は、 libasound2 の pulse プラグインを有効にし、 default ノードを pulse ノードに設定 こうすることで、 ALSA を直接利用するアプリケーショ ンの入出力を PulseAudio に向けている
- PulseAudio のモジュ ール構造
  - いろんな機能を共有ライブラリの形で提供。自在にロード・ アンロードできる。
  - http://www.freedesktop.org/wiki/Software/PulseAudio/Documentation/User/Modules/
- ALSA 以外のサブシステムを使える
  - Bluez
  - Network

10 OpenVPN を使ってみた

上川純一

10.1 はじめに

VPN（ 仮想プライベートネットワーク）とはインターネット上にある (WAN) に閉域ネットワーク (LAN) を仮想的に構築する方法です。

OpenVPN は認証と接続方式の部分で TLS[5] [*16]を活用しています。 TLS は HTTPS などで利用されている仕組みで、公開鍵認証をつかって最初の鍵交換をおこない、そこで交換した対象鍵をつかって実際の通信をすることで効率良く暗号通信ができるようになっています。

10.2 今回実現したいこと

今回想定する例としてとある個人の開発環境ネットワーク構成図を見てみましょう(図 5）。

自宅にはごく一般的なプロバイダ契約でネットワークをひいていて、ラップトップと携帯電話はどういうネットワークを経由してどういう IP アドレスでつながるのかよくわからないという構成です。

携帯電話とラップトップから自宅にあるサーバ Raspberry Pi にアクセスできるようにしたい、そう思った時にプライベート IP で接続しているプロバイダー接続がネックになります。

開発や実験などに便利な環境を実現したい、それは Raspberry pi とラップトップと携帯電話が同じ LAN にいるような環境です。

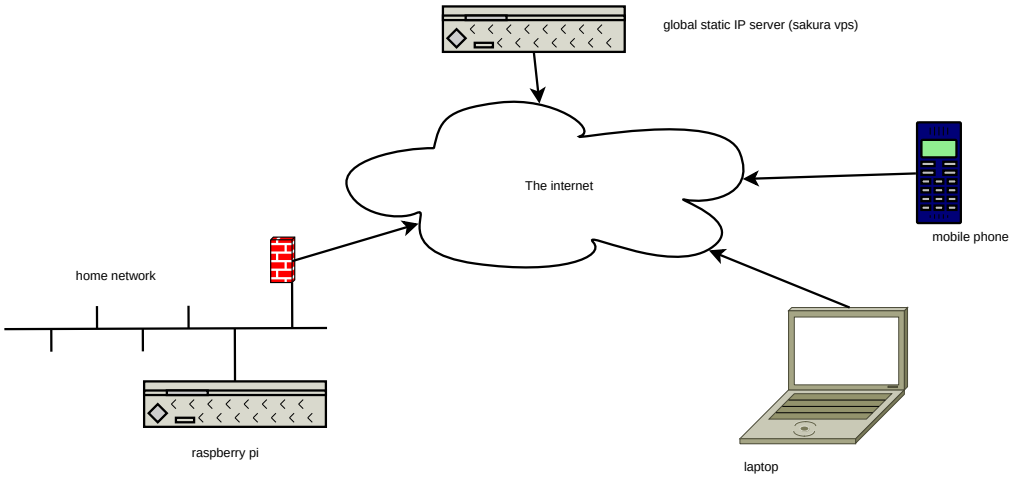


図 5 とある個人の開発環境ネットワーク構成図

[*16] 規格が TLS になるまえは SSL という規格でした

10.3 OpenSSL, CA の仕組み

OpenVPN で利用できる認証のしくみは複数あり、一番簡単な方法は共有鍵方式だと思われます[*17]。ここでは自分の直接の管理下にあるマシンを複数台管理する場合に便利そうな方式、自前で CA を立てて PKI(x509) で構築する方法について紹介します。

今回紹介する方式を実現する副作用として、 openvpn に脆弱性があったり、認証情報がもれると外部のユーザが仮想のプライベートネットワークに接続できるようになります。また、ネットワークにはユーザ認証がないので、 VPN 接続しているホストにログインできればプライベートネットワークにアクセスできるようになります。

10.4 インストール

Debian では openvpn パッケージとして提供されています。また、 CA 関連は openssl に依存しているので openssl パッケージもインストールしましょう。

```
$ sudo apt-get install openvpn openssl
```

`/etc/openvpn/*.conf` に設定ファイルを配置します。 Debian での openvpn の設定は、/etc/openvpn ディレクトリにある conf という拡張子になっているファイルを設定ファイルとして認識して起動時に利用するようです。デフォルトの状態では、最初は何も設定ファイルがないので何も起動していません。

クライアント側も openvpn パッケージをインストールします。

10.5 OpenVPN の接続性試験

まず、サーバとクライアント間で接続できるかどうかを確認する方法について紹介します。サーバとクライアントで openvpn を起動して相互に接続し、 openvpn が接続できる状態であることを確認します。 ping とかうってみるのがよいでしょう。

openVPN は UDP もしくは TCP 接続を利用することができます。どちらかはつながるでしょう。

```
server$ sudo openvpn --dev tun1 --ifconfig 10.1.1.1 10.1.1.2
client$ sudo openvpn --dev tun1 --remote サーバホスト名 --ifconfig 10.1.1.2 10.1.1.1

client$ ifconfig
tun1      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.1.1.2  P-t-P:10.1.1.1  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:168 (168.0 B)  TX bytes:168 (168.0 B)
client$ ping 10.1.1.1
```

10.6 CA の作成

まず最初に認証局を作成します。とりあえずは easy-rsa を使うとよいでしょう。 wheezy の openvpn パッケージでは `/usr/share/doc/openvpn/examples/easy-rsa/2.0/` にファイルがおいてありそれをコピーしてつかえということのようです。

[*17] 認証なし暗号化なしという方法もあってそっちのほうが簡単ですが openvpn の接続性テスト以外ではあまりつかわないと思います

```
 # cd /etc/openvpn
 # sudo cp -R /usr/share/doc/openvpn/examples/easy-rsa/2.0/ easy-rsa/
 # cd easy-rsa/
 # vi vars

export KEY_COUNTRY="JP"
export KEY_PROVINCE="TOKYO"
export KEY_CITY="Suginami-ku"
export KEY_ORG="uekawa"
export KEY_EMAIL="dancerj@gmail.com"
#export KEY_CN=changeme
#export KEY_NAME=changeme
#export KEY_OU=changeme
#export PKCS11_MODULE_PATH=changeme
#export PKCS11_PIN=1234

 # . ./vars
 # ./clean-all
 # ./build-ca
```

いろいろと質問されますが、あまり重要な質問はない気がします。 “Common Name” が名前で、ここで聞かれている名前は CA の名前です。が、この名前が重要な場面はあまりない気がします。

注意事項としては何も設定しないと全員ファイルが読めるようになっているのですが、認証情報関連のファイルは秘密であることが重要なので、 root のみが読めるようにしましょう。

10.7 各種証明書の作成

サーバはクライアントが信頼できる CA に署名された証明書を使っていることを確認することになります。クライアントはサーバの証明書が信頼できる CA にに署名されていることを確認することになります。

正式な手順はクライアント・サーバ各ノードで Certificate Signing Request (CSR) を作って CA 側で署名して CRT ファイルを返してあげるということになります。今回は略式でサーバを CA として運用し発行機関としても併用します。OpenVPN サーバが乗っ取られたら CA としても乗っ取られるのですが OpenVPN サーバが乗っ取られた時点で全ノードの再設定が必要になると思うので、新しく自己署名 CA を作りなおしてしまうという方針でいきます。

まずサーバの証明書を作成します。今回 sakura という名前のサーバの証明書を作ってみましょう。

```
 # ./build-key-server sakura
```

CN(common name) のところにはホスト名をいれる慣習になっているようで、その値は後々ログなどで表示されます。A challenge password と an optional company name は CSR に追加ではいる情報みたいですが、 CSR を読むことはないのでこの場合は何も入力しなくてもよいようです。証明書を作成して証明書の署名と登録を連続しておこなうのでちょっとわかりにくいです。

クライアントの証明書を作成します。

```
 # ./build-key client1
 # ./build-key nexus4
```

10.8 HMAC 鍵の作成

TLS をのハンドシェークは CPU 負荷の高い処理なので DoS される可能性もあります。そこで、 HMAC 鍵を使ってその前段でフィルタリングをかけることができるようです。ついでにつくってしまいましょう。

```
 # openvpn --genkey --secret ta.key
```

10.9 OpenVPN のサーバ設定

TLS 接続用にサーバ側で必要となる Diffie Hellman パラメータ(なにそれ?) を生成します。

```
 # ./build-dh
```

/etc/openvpn/server.conf に設定ファイルを作成します。

```
port 1194
proto udp
dev tun

user nobody
group nogroup

tls-auth      /etc/openvpn/easy-rsa/keys/ta.key 0 # server is 0.
ca      /etc/openvpn/easy-rsa/keys/ca.crt
cert    /etc/openvpn/easy-rsa/keys/sakura.crt
key     /etc/openvpn/easy-rsa/keys/sakura.key  # keep secret
dh      /etc/openvpn/easy-rsa/keys/dh1024.pem

server 10.55.2.0 255.255.255.0  # internal tun0 connection IP
ifconfig-pool-persist ipp.txt

keepalive 10 120

comp-lzo         # Compression - must be turned on at both end
persist-key
persist-tun

status log/openvpn-status.log

verb 3  # verbose mode
client-to-client
```

この設定ファイルは log/ディレクトリにステータスログを出力する設定なので、ディレクトリ /etc/openvpn/log/ をつくっておきます。

/etc/openvpn/ipp.txt に IP アドレスの割り当て設定が記録されます。デフォルトでは 60 秒に一回更新されるようになっています。

keepalive 10 120 の設定では、サーバとクライアントはお互いに 10 秒に一回 ping で生死監視をして、 2 分間接続がなかったら再起動します。

/etc/openvpn/*.conf にファイルがあると起動時にサーバを起動してくれるようになるのでこれで設定は完了です。

10.9.1 トポロジー

デフォルトの TUN デバイスのトポロジーは net30 です。これはクライアント毎にサブネットマスク /30 の ipv4 アドレス(4 個づつ) を割り当てることになります。ブリッジモードで TAP を使う、もしくは p2p・ subnet トポロジを使うとクライアントあたり一つの IP アドレスになるみたいですが、どうせクラス A のプライベートアドレスを大量に使えるので特に必要もないので検証してません。

10.9.2 tun or tap

仮想ネットワークデバイスは TUN と TAP の二択あります。 TAP はイーサネットレベル(L2) の仮想デバイスで、TUN は IP レベル(L3) の仮想デバイスです。今回は iOS と Android の openvpn クライアントが TUN しかサポートしていないので TUN を使うことになります。

	ネットワークレイヤー	機能	つかえる OS
tap	L2	ブリッジ(ブロードキャストパケットが到達する)	linux
tun	L3	ルータ	linux, iOS, Android など

10.10 OpenVPN のクライアント設定

CA からいくつかのファイルをコピーしてくる必要があります。 Debian の場合はサーバと大体ディレクトリ構成をあわせておけばよいでしょう。 root でしか読み込めないように権限を設定しておくのをお忘れなく。

ca.crt, client.crt, client.key, ta.key が必要になります。

Android の OpenVPN アプリケーションの場合 [6] は、 /sdcard 以下に適当な名前でディレクトリを作成してそこに必要なファイルを配置します。設定ファイルは拡張子を conf ではなく ovpn にしておかないと認識してくれないようです。一回 OpenVPN アプリケーションで Import したらその SD カード上のディレクトリの中身は必要なくなるので消しましょう。

```
client
dev tun
port 1194
proto udp

remote xyz.sakura.ne.jp 1194             # vpn server ip : port
nobind

tls-auth      ta.key 1 # client is 1.
ca ca.crt
cert android-nexus4.crt
key android-nexus4.key

remote-cert-tls server
comp-lzo
persist-key
persist-tun

verb 3
```

remote-cert-tls server で証明書がサーバー鍵であることを確認します。./build-key-server で作られた証明書は “key usage” に サーバ証明書であることが記述されています。これを指定しないと有効な CA をに署名された証明書であればどれでもつなぎに行きます。たとえば流出したクライアントの証明書でサーバを偽装することが可能になります。

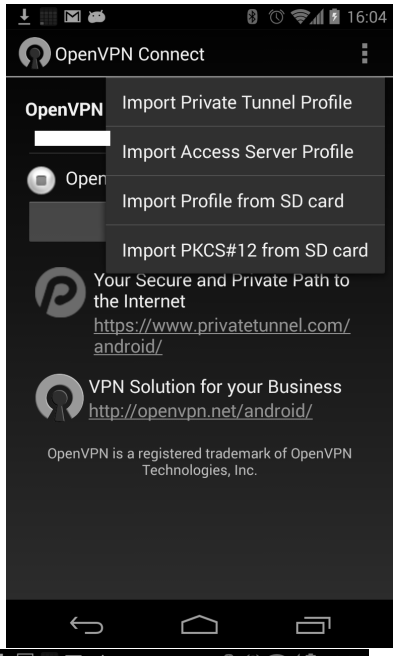
起動方法は、 `/etc/network/interfaces` に記述することもできますが [3] デフォルトの状態でネットワークの設定が変更になったら勝手に OpenVPN 接続を試行してくれる設定になっているようです。[*18]

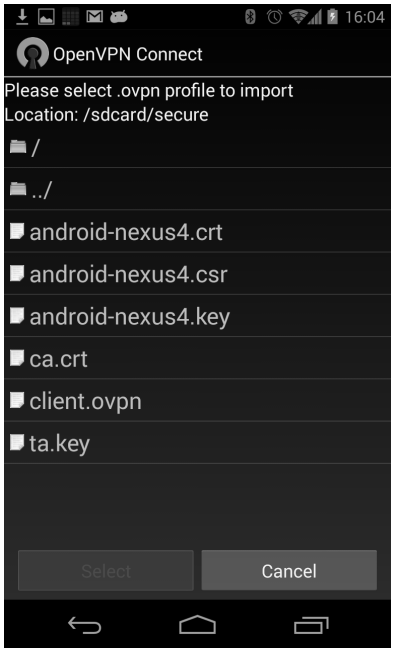
10.10.1 Android UI

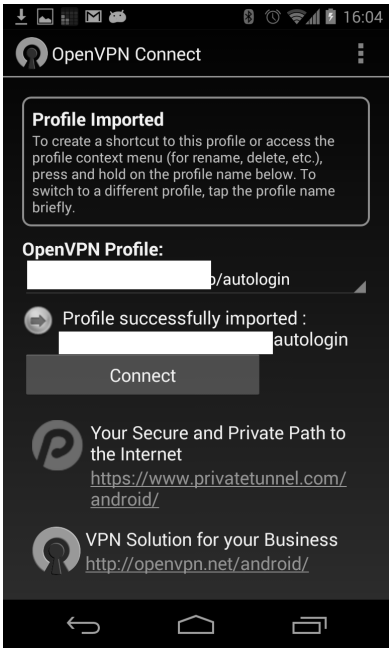
Android デバイスに必要なファイルをコピーします。

```
$ ls -1
android-nexus4.crt
android-nexus4.csr
android-nexus4.key
ca.crt
client.ovpn
ta.key
$ adb push . /sdcard/secure
push: ./ca.crt -> /sdcard/secure/ca.crt
push: ./android-nexus4.key -> /sdcard/secure/android-nexus4.key
push: ./ta.key -> /sdcard/secure/ta.key
push: ./android-nexus4.crt -> /sdcard/secure/android-nexus4.crt
push: ./client.ovpn -> /sdcard/secure/client.ovpn
push: ./android-nexus4.csr -> /sdcard/secure/android-nexus4.csr
```

[*18] 多分 `/etc/network/if-up.d/openvpn` の設定による







インポートしたら sdcard からファイルを削除します。

10.11 応用

特に今回必要でなかったので設定しなかったのですが違う設定も可能なのでそれも紹介します。

ルーティングなどは openvpn サーバ側で設定すればクライアントに設定を配布(push) してくれるようです。

10.11.1 ローカルネットワークへのゲートウェイを設定したい

redirect-private オプションで VPN 経由でのルーティングを設定します。たとえば、 VPN 経由のゲートウェイ(たとえば 10.0.0.1) 経由で 自宅のネットワーク(例えば 192.168.1.0/24) にアクセスできるようにしたいときに使えます。

`/usr/share/doc/openvpn/examples/sample-config-files/firewall.sh` に iptables の設定例が掲載されています。

10.11.2 暗号化経路のためにすべてのパケットを VPN 経由にしたい

redirect-gateway オプションで指定できます。

公共の WiFi スポット経由での通信などが安心できない人向けのオプションです。しかし DHCP の再リースや DNS の名前解決なども VPN 経由になってしまうと困る場合もあるので bypass-dhcp, bypass-dns オプションなどがあるようです。

10.12 おわりに

openvpn 2.3 で git リポジトリの再編を行ったようで、 easy-rsa とかが分離されたっぽいです。

マニュアルを読んでいると Windows 版の専用オプションとかが複雑にからみあっていて実際どれをつかえばよいのか読み解きにくいです。最低限動かすまではそこまで難しくないのですがルーティングとかデフォルトゲートウェイを変更しようかと考え始めると考えることがいくつかあるようです。しかし一般論で考えても、ネットワーク 3 つ以上を接続する場合のファイアウォールとかルーターの設定はそれなりに複雑ですから、それを考えると妥当なのかもしれませんね。

10.12.1 ssh の各機能との比較

開発者・運用者愛用のツール ssh とどう違うのかとおもってみてみたのですが、 ssh にも VPN 機能もあるので比較しにくいですね。ただ、通常の運用では root 同士で ssh することはあまりないかと思いますが、 VPN を利用するには root で ssh できる必要があるようです。

openvpn のメリットとしては openvpn には接続が切れた時を検出して再接続する仕組みがあること、 TCP だけでなく UDP も使えること、 root 権限で ssh ログインを許可しないでよいことなどがあげられると思います。

表 4　ssh の機能の対応

	ssh オプション	機能
ポートフォワーディング	-L, -R	特定のローカルの TCP ポート番号をリモートのポート番号とマッピングする
プロキシ	-D	SOCKS に対応しているアプリケーションのプロキシを提供
VPN	-w	IP レベルで相互に見えるよう仮想的にネットワークを構築する

参考文献

[1] `http://wiki.debian.org/OpenVPN`

[2] openvpn(8) manpage

[3] OpenVPN README.Debian `/usr/share/doc/openvpn/README.Debian.gz`

[4] OpenVPN Community Software `http://openvpn.net/index.php/open-source.html`

[5] The Transport Layer Security (TLS) Protocol Version 1.2 `https://tools.ietf.org/html/rfc5246`

[6] OpenVPN Connect Android app `https://play.google.com/store/apps/details?id=net.openvpn.openvpn`

11 wayland を動かす

野島 貴英

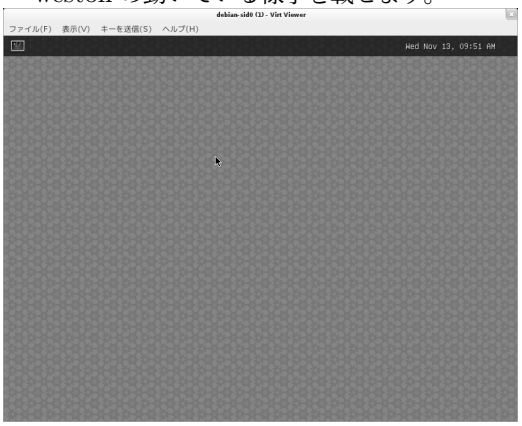
11.1 wayland

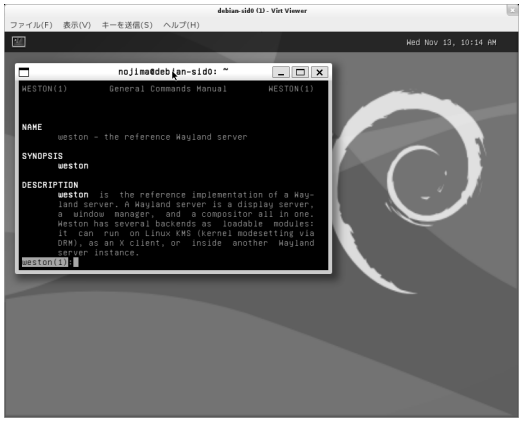
wayland とは、 Kristian Høgsberg さんが中心となって作っているディスプレイサーバーのプロトコルのことです。wayland プロトコルを扱えるディスプレイサーバーとして weston があります。

従来からある Unix で有名なディスプレイサーバーのプロトコルとしては、 X プロトコルがあり、 X プロトコルを扱えるディスプレイサーバーに X があります。しかしながら、 X は 1984 年頃の設計から始まって、ずっとハードの進化にあわせてつぎはぎしてきたため、いろいろ実装と機能に無理が生じています。 wayland は、 X に比べて、圧倒的に洗練された設計で、シンプルに、プロトコルとディスプレイサーバーを実現したものとなります [1]。

11.2 weston の動いている様子

weston の動いている様子を載せます。





11.3 weston の対応出力デバイス

weston の対応できる出力デバイスは表 11.3 となります。

11.4 debian での weston 稼働のための下準備

debian で weston を動作させるには、以下の下準備が必要です。

Step 1. weston の導入をします。

```
# aptitude install weston
```

Step 2. systemd パッケージを導入するなどして、環境変数 XDG_RUNTIME_DIR が設定されるようにします。

項番	出力デバイス	バックエンド名	パッケージに搭載済	備考
1	DRM/KMS	drm-backend.so	○	
2	FrameBuffer	fbdev-backend.so	○	
3	X	x11-backend.so	○	
4	Wayland	wayland-backend.so	○	
5	Headless	headless-backend.so	○	
5	Rassberry Pi	rpi-backend.so	×	
6	RDP	rdp-backend.so	×	

表 5　出力デバイスと weston の対応状況

```
# aptitude install systemd
```

Step 3.　weston-launch にユーザを加え、ログインしなおします。

```
# usermod -a -G weston-launch <your-login-id>
... 重要: この後ログインしなおす...
```

Step 4.　環境変数 XDG_RUNTIME_DIR が設定されているか確かめます。なお、何らかの理由で、 systemd パッケージ付属の systemd-logind が動作しない等の理由により、 XDG_RUNTIME_DIR が設定されていないのであれば、 export XDG_RUNTIME_DIR=/tmp などして書き込みが可能なディレクトリを指定しておきます。

```
[1] systemd-logind が無事動いている場合↓
$ env | fgrep XDG_RUNTIME_DIR
XDG_RUNTIME_DIR=/run/user/1000

[2] systemd-logind が何らかの理由で動作していない場合↓
$ env | fgrep XDG_RUNTIME_DIR
... なにも表示されない...
$ export XDG_RUNTIME_DIR=/tmp
```

11.5　その 1:X 上で動かしてみる

一番簡単に動かせます。 X のターミナルソフト上で、

```
$ weston
```

とするだけで、 x11-backend.so が読み込まれ、 weston の窓が開き、 weston が動作を開始します。

停止する時は、 weston を起動しているターミナルで、 Ctrl-C を押下します。

11.5.1　X 上で動かしてみる時の注意点

何故か、手元の nvidia 製グラフィクスカードにて、 non-free の nvidia ドライバを使っていると、画面真っ黒のウィンドウが開いてしまいました。–use-pixmap を指定して weston を起動して、 weston が EGL 等のハードウェアアクセラレーションを利用しないようにしても同様だったりします。一方で、 intel 製のグラフィクスチップを載せているノート PC 上では問題なく動作しました。

こちらの原因はまだつかめていません。

11.6　その 2:KMS/DRM で動かしてみる

intel のグラフィックスチップが搭載されている PC(例: ノート PC とか) であれば、 linux の KMS/DRM ドライバで動作します。また、 AMD/Nvidia のグラフィクスチップであれば、 linux の KMS/DRM ドライバの radeon/nouveau ドライバで動作すると思われますが、自分は未評価です。どなたか radeon/nouveau で動作した方がいらっしゃれば教えてください。

Step 1. グラフィカルなログイン画面が出ているようであれば、こちらを停止させます。

例: gdm3 が立ち上がっている場合の止め方

```
Ctrl-Alt-F1 等を押下してコンソールに切り替える
$ su
# service gdm3 stop
```

Step 2. KMS/DRM ドライバが有効であることを確かめます。

```
$ lsmod | egrep '(i915|radeon|nouveau)'
...(i915/radeon/nouveau のどれかの文字列が出れば OK)...
```

Step 3 weston を動かします。

```
$ weston-launch
```

11.6.1 KMS/DRM 上で動かしてみる時の注意点

linux の KMS/DRM ドライバは、 i915/radeon/nouveau 以外は動作するかどうかは正直やってみないと分かりません。

例えば、 debian にて、仮想化技術の KVM では cirrus チップセット用の KMS/DRM ドライバが virt-viewer の元で使えるのですが、 weston は egl 側 cirrus 未対応によるエラーもしくは、 Segfault で落ちてしまうためどうにも動作しませんでした。こちらも原因が自分では正確につかめていません。

11.7 その 3.FrameBuffer デバイス上で動かしてみる

linux の FrameBuffer デバイス上で動かしてみます。今のところ仮想化技術の KVM 上で動かすのに便利です。ここでは、実際に KVM を使って動かします。

なお、大変残念なことに、現在の debian sid で導入できる weston パッケージのバージョン 1.3.0 では、仮想端末制御に関する upstream 側のバグのために、 FrameBuffer デバイスでは動作しません。幸い、こちらのバグが修正された、weston-1.3.1 がリリースされましたので、ここでは、このバージョンを簡易的に debian パッケージ化して導入します。

Step 1. weston を動かす対象の debian sid を KVM を使ってインストールします。なお、 KVM ホスト OS 側の debian 機の br0 はインターネットに接続できるように設定されているものとします。 br0 のセットアップについて、詳しくは [2] を参照してください。

```
# aptitude install libvirt-bin virtinst
# qemu-img create -f raw /var/lib/libvirt/images/debian-sid0 10G
# wget http://cdimage.debian.or.jp/7.2.0/multi-arch/iso-cd/debian-7.2.0-amd64-i386-netinst.iso
# virt-install --connect=qemu:///system -n debian-sid0 --ram 512  --cdrom /home/yours/debian-7.2.0-amd64-i386-netinst.iso \
  --disk /var/lib/libvirt/images/debian-sid0,bus=virtio,size=10,format=raw,cache=writeback \
  --bridge=br0,model=virtio --vnc --hvm --accelerate
```

なお、インストールの際の設定は表 6 のとおりを仮定します。

Step 2. インストール完了したら、すぐに debian sid へアップグレードします。

```
debian-sid0 にログインの後、
debian-sid0 $ su
debian-sid0 # cat > /etc/apt/sources.list
deb http://ftp.jp.debian.org/debian/ sid main contrib non-free
deb-src http://ftp.jp.debian.org/debian/ sid main contrib non-free
<ctrl+d>を押下
debian-sid0 # aptitude update;aptitude full-upgrade;aptitude clean
```

Step 3. weston-1.3.1-1 パッケージを作って導入します。

項番	設定項目	指定内容	備考
1	select a language	Japanese	
2	場所の選択	日本	
3	ネットワークの設定	手動設定。IP:192.168.0.2, Netmask:255.255.255.0, Gateway:192.168.0.1, Host 名:debian-sid0	
4	パッケージマネージャの設定	ミラー:日本, ミラーサイト:ftp.jp.debian.org, HTTP プロキシ: 空欄	
5	ソフトウェアの選択	SSH サーバー、標準システムユーティリティのみ選択。あとはすべて解除。	

表 6　インストーラで選択する項目

```
debian-sid0 # aptitude build-dep weston/sid
debian-sid0 # aptitude install libxcb-composite0-dev fakeroot
debian-sid0 # exit
debian-sid0 $ mkdir weston weston-work
debian-sid0 $ cd weston
debian-sid0 $ apt-get source weston/sid
debian-sid0 $ cd ../weston-work
debian-sid0 $ wget -O weston_1.3.1.orig.tar.gz \
  http://cgit.freedesktop.org/wayland/weston/snapshot/weston-1.3.1.tar.gz
debian-sid0 $ tar xzf weston_1.3.1.orig.tar.gz
debian-sid0 $ cd weston-1.3.1
debian-sid0 $ tar xzf ../../weston/weston_1.3.0-1.debian.tar.gz
debian-sid0 $ cd debian;
debian-sid0 $ patch -p1 <<__HERE
--- debian.org/changelog        2013-10-11 20:04:50.000000000 +0900
+++ debian/changelog    2013-11-12 14:48:36.219299000 +0900
@@ -1,3 +1,9 @@
+weston (1.3.1-1) unstable; urgency=low
+
+  * update to upstream
+
+ -- your name <foo@bar.com>  Fri, 11 Nov 2013 12:34:56 +0900
+
 weston (1.3.0-1) unstable; urgency=low

    [ Sven Joachim ]
diff -ru debian.org/control debian/control
--- debian.org/control  2013-10-11 19:58:17.000000000 +0900
+++ debian/control      2013-11-12 14:49:32.347299000 +0900
@@ -35,6 +35,7 @@
   libpam0g-dev,
   libvpx-dev,
   libsystemd-login-dev,
+ libxcb-composite0-dev,
 Standards-Version: 3.9.4
 Homepage: http://wayland.freedesktop.org/
 Vcs-Git: git://anonscm.debian.org/pkg-xorg/wayland/weston
__HERE
debian-sid0 $ cd ..
debian-sid0 $ env DEB_BUILD_OPTIONS='noopt nostrip' \
   dpkg-buildpackage -rfakeroot -us -uc 2>&1 | tee ../build.log
debian-sid0 $ cd ..
debian-sid0 $ su
debian-sid0 # dpkg -i ./weston_1.3.1-1_amd64.deb
```

Step 3.　11.4 章の Step 2.〜Step 4. に記載の下準備をしておきます。(Step 1. は先ほど構築した weston パッケージが導入済みですので不要です)

Step 4.　FrameBuffer のセットアップをします。なお weston を動かすためには、color depth は 24bit でなければなりません。

```
debian-sid0 # aptitude install fbset
debian-sid0 # modprobe cirrusfb
debian-sid0 # fbset -g 800 600 800 600 24
```

Step 5.　一般ユーザになり、weston を起動します。

```
debian-sid0 # exit
debian-sid0 $ weston-launch -- --backend=fbdev-backend.so --log=weston.log
```

11.8 weston のカスタマイズ

weston はデフォルトのままだと少し寂しい画面なので、ちょっとカスタマイズしてみます。カスタマイズは$HOME/.config/weston.ini にいろいろ記載すると、いろいろとカスタマイズできます。ここでは壁紙を入れ、ウィンドウポップアップがアニメーションするようなカスタマイズをしてみます。

```
$ cat >.config/weston.ini  <<__HERE
[shell]
background-image=/usr/share/images/desktop-base/spacefun-grub.png
background-type=tile
locking=true
animation=zoom
binding-modifier=ctrl

[launcher]
icon=/usr/share/weston/terminal.png
path=/usr/bin/weston-terminal
__HERE
```

11.9 weston の構造

図 6 に wayland アプリケーションと weston の構造を載せます。

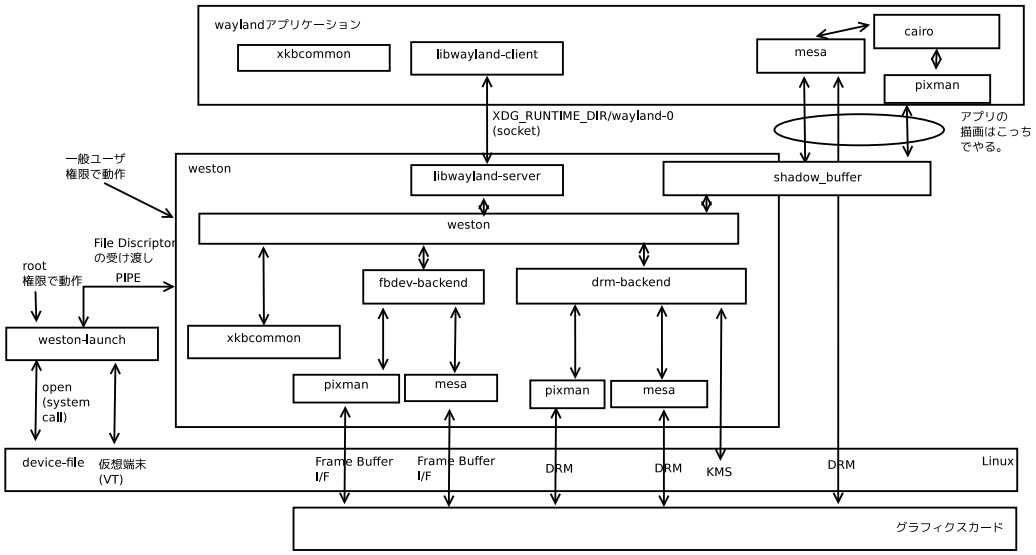


図 6　wayland アプリケーションと weston

特徴として、

- 基本的にアプリケーション側も含めて、ローカルシステムで描画を行うことを前提にした作りになっています。
- cairo/mesa/xkbcommon 等を最大限利用して、 weston 自体は非常にシンプルな設計になっています。
- プロセスのユーザ権限を制限するため、 weston-lanch には root 権限が最低限必要な部分のみの操作を、 weston は一般ユーザでの権限による動作をするように、権限が分割されています。なお、現在の実装では weston-launch は操作・オープン済みの端末の FileDiscriptor と、デバイスファイルの File Descriptor、 weston-launch へのシグナルを、 weston の求めに応じて引き渡す機能が実装されています。

11.10 その他

debian で利用できる weston/wayland にはちょっとしたクセがあります。

- Xwayland が動作しません。これは weston が起動する X に wayland 用の I/F を搭載するパッチが摘要されてい

なければならない（ X 起動時に-wayland オプションが使えなければならない）のですが、こちらは未だ X のパッケージに含められていない為となります。なお、 2013/10 頃にこちらの debian パッケージ用のパッチが debian-x の ML に流れていました。

- gtk 等の有名なツールキットは wayland 未対応の状態です。

11.11 おわりに

次期ディスプレーサーバーになるかもしれない weston と wayland について、いろいろ試してみました。プログラム本体は非常に小さいプログラムですし、発展途上ですので、いじってみようという方は、今ならたやすく改造/解析し放題の旬な時です。 Hack の対象にぜひ。

参考文献

[1] Daniel Stone,“The real story behind Wayland and X”,linux.conf.au 2013, http://people.freedesktop.org/~daniels/lca2013-wayland-x11.pdf, http://www.youtube.com/watch?v=RIctzAQOe44

[2] 野島 貴英,「 Debian 開発者の KDE 環境あれこれ」, 第 85 回東京エリア Debian 勉強会資料, http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume201202.pdf

12 月刊 Debhelper dh_strip

野島 貴英

12.1 今月のコマンド

今月は dh_strip を取り上げます。紙面の都合で dh_strip の 1 コマンドの紹介に今回はとどめておきます。

12.2 dh_strip

dh_strip コマンドは、“debian/tmp/” 等のビルドディレクトリ以下を再帰的に探し回り、ビルドしたばかりの実行ファイルのバイナリ、共有ライブラリのバイナリ、静的ライブラリのバイナリを見付けると、片っ端からバイナリに含まれているデバッグシンボルを取り除いてくれます。

また、別途コマンドラインオプションを指定すると、取り除いたデバッグシンボルを別ファイルに取り置きます。このため、dh_strip コマンドは*-dbg パッケージ作成の時に重要な役割を持ちます。

12.2.1 すでに発表済みの件

実は、dh_strip の*-dbg パッケージを作成する為の動作については、大統一 Debian 2012 のセッションである「debug.debian.net」[1] のプレゼン資料に詳しいので、こちらを参照してください。

また、*-dbg パッケージを使って実際にデバッグをしてみる件については、大統一 Debian 2013 のセッションである「gdb+python 拡張を使ったデバッグ手法」[2] に紹介していますので、こちらも参照ください。

ここでは、これらの発表で扱わなかった件を重点的に記載します。

12.2.2 debug/*.so,lib*_g.a ファイルの扱い

dh_strip コマンドは、元々デバッグ用途として用意されている debug/*.so ファイルや、lib*_g.a ファイルについては、何も処理を行いません。

12.2.3 コマンドラインオプションと動作

dh_strip コマンドは表 7 のコマンドラインオプションを解釈します。

12.2.4 COMPATIBILITY LEVEL での動作の違い

dh_strip はソースパッケージに含まれる debian/compat で指定される COMPATIBILITY LEVEL により、振る舞いが変わります。

–dbg-package オプション指定時の振る舞いの違いは表 8 に、また、-k/–dbg-package オプション指定時に生成されるデバッグシンボルファイルの違いについて表 9 に、記載します。

オプション名	動作
man debhelper 記載のオプション	こちらは全 debhelper に共通のオプションです。動作は man debhelper に記載されていたり、以前の月刊 debhelper で説明されているとおりとなりますので、ここでは割愛します。
-Xitem, –exclude=item	item で指定されるファイル名を持つバイナリを処理の対象から外します。複数のファイルを指定したい場合は、繰り返し指定すると指定できます。例: dh_strip -Xfile1 -Xfile2
–dbg-package=package	*-dbg パッケージを作る際に、デバッグシンボルを格納するパッケージ名を指定する為に利用します。通常、*-dbg パッケージを作る場合は、こちらのオプションを利用する事となります。なお、ソースパッケージにある debian/compat で指定される COMPATIBILITY LEVEL で振る舞いが異なります(後述) 。
-k	分離したデバッグシンボルをバイナリパッケージに含める場合にこちらのオプションを利用します。この場合、出来上がったパッケージにはバイナリの他に、分離したデバッグシンボルも/usr/lib/debug/以下に含まれる形で収録されます。–dbg-package が同時に指定されると、–dbg-package の指定の方が優先されます。

表 7　dh_strip コマンドラインオプション

COMPATIBILITY LEVEL	–dbg-package 指定時の動作
4 以下	“dh_strip –dbg-package=xxxx –dbg-package=yyyy” のように–dbg-package を複数指定する事が出来ます。ここで、こちらで指定した名前 (xxxx や、 yyyy) に合致するパッケージを処理する際、” パッケージ名-dbg” というパッケージ名を*-dbg パッケージのパッケージ名として自動的に利用します。また、ソースパッケージに含まれる debian/control には、これら*-dbg パッケージの為の記述は不要となります。
5 以上	–dbg-package は 1 つ指定するのが原則となります。もし複数指定した場合は、最初に指定された内容のみ利用されます。また、–dbg-package=xxxxx とすると、デバッグシンボルを格納するパッケージ名として、 xxxxx がそのまま使われます。例えば、 libfoo パッケージと、 foo パッケージを構築し、これらパッケージに含まれているバイナリのデバッグシンボルを foo-dbg パッケージに全部入れるには、–dbg-package=foo-dbg と指定します。また、ソースパッケージに含まれる debian/control には、生成予定の*-dbg パッケージの為の記述は必須となり、万一記載されていない場合はエラーとして扱われ、この場合は dh_strip はエラーを表示して終了します。

表 8　COMPATIBILITY LEVEL と–dbg-package オプションの振る舞いの違い

12.2.5　デバッグシンボル

dh_strip で取り分けられるデバッグシンボルは、 elf バイナリベースのシステム[*19]であれば、 DWARF[3] をベースとした形式のファイルとなります。

ここで DWARF は、コンパイラを用いて CPU の命令に直接変換するようなコンパイル言語(C/C++ 等) を用いて作成したバイナリを対象として、ソースレベルのデバッグを行う場合に必要な/便利な情報を格納する為に作られた、よくできたフォーマットです。元々は System V 用のデバッガである sdb の為に Bell 研で開発されたフォーマットがベースとのことです[*20]。 DWARF は、”Debugging With Attributed Record Formats” の略とのことです。

図 7 にバイナリと、デバッグシンボルの中身を簡単に図示します。

12.2.6　デバッグシンボルファイルの情報を覗いてみる

実はバイナリや、デバッグシンボルの中身を見る方法は知っておくと、いろいろデバッグ等で役立つ場面が多いです。以下に簡単に参照する方法を載せます。

[*19] お馴染みの i386/amd64 用 linux の場合等

[*20] Bell 研とか、 System V とか、今ではもう完全にオッサンホイホイな用語となってしまいました...

COMPATIBILITY LEVEL	-k/–dbg-package 指定時のデバッグシンボルファイルの違い
8 以下	“/usr/lib/debug/+ バイナリのインストール先パス/+ バイナリファイル名” に格納されるようにデバッグシンボルファイルが生成されます。また、デバッグに関する情報はコンパイラが生成したままの形で保存されるため、デバッグシンボルファイルのサイズは大きくなりがちです。
9 以上	バイナリが BuildID[2] を含んでいる場合は、 “/usr/lib/debug/.build-id/BuildID の上位 2 桁/+BuildID の残りの桁 +.debug” に格納されるようにデバッグシンボルファイルが生成されます。 BuildID を含んでいなければ COMPATIBILITY LEVEL 8 以下と同様のファイル名となります。また、 BuildID の有無にかかわらず、デバッグに関する情報は zlib により圧縮されて格納されます。そのため、デバッグシンボルファイルのサイズはできるだけ小さくなるようになっています。

表 9　COMPATIBILITY LEVEL と-k/–dbg-package オプションの振る舞いの違い

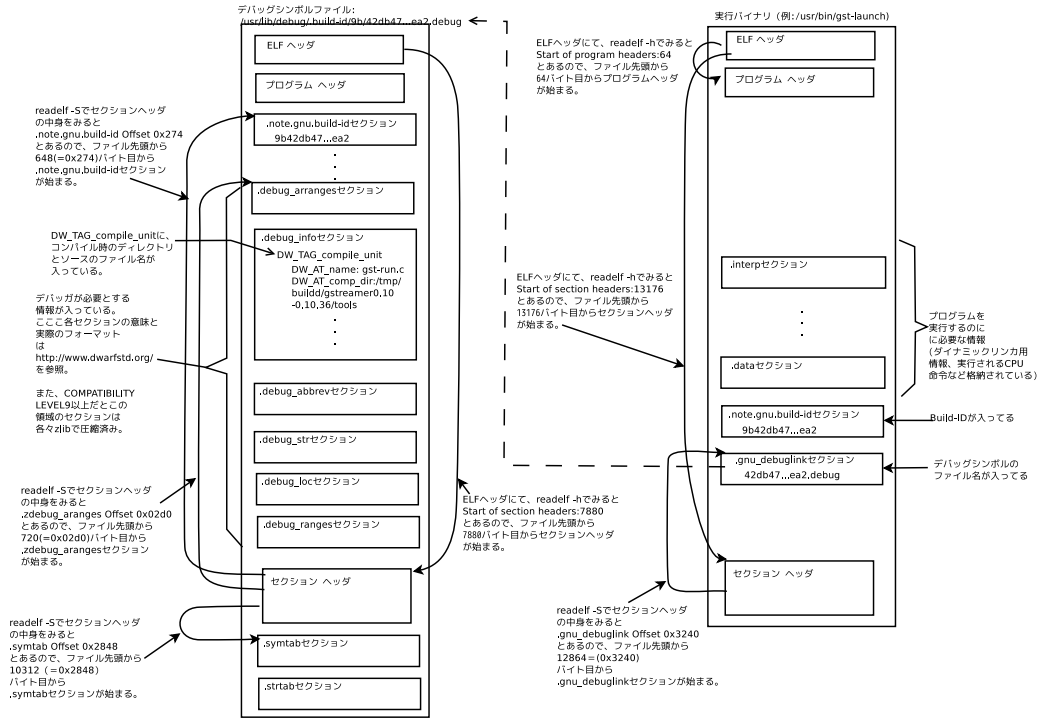


図 7　バイナリと、デバッグシンボルの中身

まず、バイナリにはどんなヘッダ情報とセクションが入っているかのサマリを掌握してみます。

```
$ sudo aptitude install binutils
$ readelf -e /usr/bin/gst-launch
ELF ヘッダ
   マジック:  7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
   ... 中略...
セクションヘッダ:
  [番] 名前              タイプ            アドレス          オフセット
       サイズ            EntSize           フラグ Link  情報  整列
  [ 0]                   NULL              0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS          0000000000400238  00000238
       000000000000001c  0000000000000000   A       0     0     1
... 中略...
  [27] .gnu_debuglink    PROGBITS          0000000000000000  00003240
       0000000000000034  0000000000000000           0     0     1
  [28] .shstrtab         STRTAB            0000000000000000  00003274
       00000000000000fe  0000000000000000           0     0     1
... 中略...
```

次にどんなデバッグシンボルファイルが使われる予定かを見てみます。

```
$ readelf -x .gnu_debuglink /usr/bin/gst-launch
セクション '.gnu_debuglink' の 十六進数ダンプ:
  0x00000000 34326462 34373631 31386162 32623831 42db476118ab2b81
  0x00000010 30343161 63643830 65343565 38396534 041acd80e45e89e4
  0x00000020 32383965 61322e64 65627567 00000000 289ea2.debug....
  0x00000030 dbf8060d                             ....
$
```

デバッグシンボルのファイル名の形式から、 COMPATIBILITY LEVEL 9 のソースパッケージで構築されたデバッグシンボルファイルである事がわかります。このため、 BuildID から、/usr/lib/debug/.build-id/9b/42db476118ab2b81041acd80e45e89e4289ea2.debug がデバッグシンボルのファイルとなります。

次にデバッグシンボルファイルから、ビルド時のディレクトリ位置を求めてみます。

```
$ aptitutde install libgstreamer0.10-0-dbg
$ readelf -wi /usr/lib/debug/.build-id/9b/42db476118ab2b81041acd80e45e89e4289ea2.debug | lv
.debug_info セクションの内容:

  コンパイル単位 @ オフセット 0x0:
   長さ:        0x1b3b (32-bit)
   バージョン:    2
   Abbrev Offset: 0x0
   ポインタサイズ:8
 <0><b>: 省略番号: 1 (DW_TAG_compile_unit)
    <c>   DW_AT_producer    : (間接文字列、オフセット: 0x442): GNU C 4.7.2

    <10>   DW_AT_language    : 1        (ANSI C)
    <11>   DW_AT_name        : (間接文字列、オフセット: 0x57a): gst-run.c

    <15>   DW_AT_comp_dir    : (間接文字列、オフセット: 0x36b): /tmp/buildd/gstr
eamer0.10-0.10.36/tools
  ... 中略...
```

.debug_info セクションに定義されている、 DW_TAG_compile_unit の結果から、
ディレクトリ/tmp/buildd/gstreamer0.10-0.10.36/tools に配置されている gst-run.c がコンパイルされている事がわかります。

では、この結果を元に、 gdb の set substitute-path を使って gdb でソースの閲覧をやってみます[*21]。

```
$ pwd
/home/yours/src/gstreamer/
$ apt-get source gstreamer0.10-tools/sid
$ gdb --args /usr/bin/gst-launch
... 中略...
Reading symbols from /usr/bin/gst-launch...Reading symbols from
/usr/lib/debug/.build-id/9b/42db476118ab2b81041acd80e45e89e
4289ea2.debug...done.
(gdb) set substitute-path /tmp/buildd/ /home/yours/src/gstreamer/
(gdb) b main
(gdb) run
(gdb) l
313         return candidates;
314       }
315
316       int
317       main (int argc, char **argv)
318       {
319         GHashTable *candidates;
320         gchar *dir;
```

無事ソース閲覧ができています。

12.3 おわりに

今回は、 dh_strip コマンドをネタに、デバッグシンボルファイルの中を覗き、その結果を元に最後は gdb でソース閲覧ができるようになるまでをやってみました。

また、 dh_strip の周辺技術を見てみると、ソースコードのデバッグができるようになる為に、とても多くの人の成果物を活用していることが実感できました。

[*21] なお、本方法は、 2013 年大統一 Debian 勉強会 [2] の当時では、自分がまだ見つけていなかった方法でして... これで gdb の dir コマンドを使わなくて済み、今後は*-dbg パッケージを導入するだけで便利にソースの閲覧できるようになります。

参考文献

[1] 2012 年大統一 Debian 勉強会「 debug.debian.net」, http://gum.debian.or.jp/2012/

[2] 2013 年大統一 Debian 勉強会「 gdb+python 拡張を使ったデバッグ手法」, http://gum.debian.or.jp/2013/slide_data_list

[3] The DWARF Debugging Standard, http://www.dwarfstd.org/Home.php

13 Debian 勉強会の資料の ePUB 化を試みた

まえだこうへい

13.1 ePUB 化の動機

最近はかなりスマートフォンやタブレットが普及し、それ以外の eBook reader も東京周辺の電車内でも見かけるようになりました。 Debian 勉強会の参加者もそれらのデバイスを持っている人も今までちらほら見かけています。 Debian 勉強会の事前配布資料や年 2 回の「 あんどきゅめんてっどでびあん[*22]」をそれらのデバイスで読む機会も増えているのではないでしょうか。[*23]

ところで、 Debian 勉強会の資料は LaTeX をソースとして PDF として配布されています。 PDF は印刷物と同じレイアウトになるので、本勉強会のように印刷した資料を頒布する上では最適です。しかし、画面で閲覧するのには向いていないと思いませんか。その一番の理由は多くの PDF リーダーは、画面に最適な大きさで自動的にリサイズされないためだと考えています。[*24]読みたい部分を拡大表示することはできますが、画面サイズに合わせた文字が巻き返しはされず、画面外にでた続きの文章は横方向にスクロールしなくてはなりません。これは面倒です。

そこで ePUB です。 O'Reilly, 達人出版会から出版されている Ebook の多くは ePUB が採用されているので、これらを既に購入、読んでいる人もいるでしょう。ページあたりの構成が画面サイズに最適化され、かつ、フォントサイズの変更でもページが自動的にリサイズされるので、これらのデバイスで読むのにはうってつけでしょう。

今回は Debian 勉強会の資料を、 LaTeX のソースコードには極力手を加えずに[*25]ePUB を生成できないか検証しました。

13.2 LaTeX からの ePUB 生成の方法

LaTeX から ePUB を生成するにはいくかの方法があります。取りうる手段としては 4 パターンです。

1) LaTeX から直接 ePUB を生成する
2) a. TeX もしくは b. DVI から, XML もしくは HTML を生成し、 2)-2. ePUB に変換する
3) PDF から ePUB に変換する

[*22] 現在名前は違いますが、これが一番通りは良いと思うのでこのまま表記します。
[*23] 勉強会当日参加者に質問したらほとんどいないという結果でした。
[*24] ここでは印刷物に比べて読みにくい、という観点では論じません。
[*25] PDF は今まで通り印刷物用に必要なので。

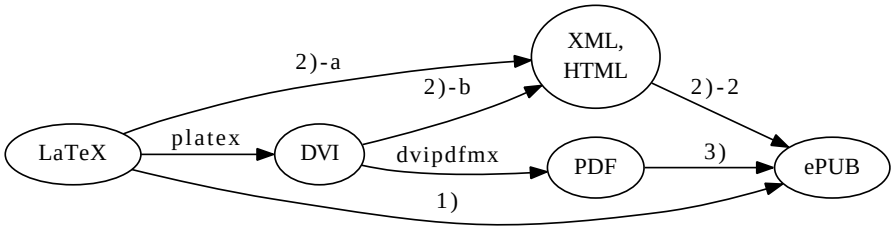


図 8　LaTeX から ePUB への変換

これらを行うためのツールについて検証しました。

13.3　検証結果

基本的に Debian パッケージになっているツールで検証を行いました。対象は次の 2 つのファイルです。

1. Debian 勉強会の資料 (debianmeetingresume2013007.tex)
2. latex2epub のサンプル TEX ファイル (sample.tex)

その結果は下記の通りです。

パターン	ツール名	入力	出力	結果	
				Debian 勉強会の資料	latex2pub のサンプル
1)	Pandoc	LATEX	ePUB	NG	OK
1)	latex2epub	LATEX	ePUB	NG	OK
2)-a	LATEXML	LATEX	XML	NG	OK
2)-b	TEX4ht	DVI	HTML	NG	NG
2)-b	htplatex(TEX4ht)	TEX	HTML	OK	NG
3)	Pandoc	HTML	ePUB	OK	N/A
4)	Calibre	PDF	ePUB	OK	OK

Debian 勉強会の資料と、 latex2epup のサンプル (以後、サンプルファイル) とで結果に大きな差がありますが、エラーメッセージを見る限りでは、その主な原因は使用している documentclass の違いと、マクロの有無によるものではないかと考えられます。が、ここはあまり深堀できていません。

13.4　変換できたケースでの課題

Debian 勉強会の資料で変換できたケースでの課題を挙げます。

13.4.1　htplatex & pandoc

現状では次の問題がありますが、今回検証した中では一番まともに読める形で変換することができました。

- tabular が table に変換されず、表にならない
- 表紙の画像が追加されない
- あんどきゅめんてっどでびあん夏号、冬号をそれが含まれる月よりも先に変換すると、その中で使われている画像が html ディレクトリにコピーされず、ファイルが無いため pandoc 実行時に失敗する
- TEX4ht で HTML 変換時に自動生成される画像のファイル名が異なり、同じく pandoc 実行時に失敗する月もある (2013 年 4 月など)
- PDF を生成する場合 (= make を実行する場合)、 htplatex が依存するパッケージ dvi2ps-fontdata-a2n をアンインストールする必要がある。これをアンインストールしないと、 dvipdfmx での DVI から PDF 変換時に、”`** ERROR ** Virtual fonts nested too deeply`”というエラーが出て失敗する

なお、 TEX4ht は、実は上川さんが既に通った道でした。[*26] Debian 勉強会の資料のリポジトリの中に、 htplatex というシェルスクリプトがあります。このスクリプトは Debian 勉強会の資料の LATEX のファイルを TEX4ht を使って HTML 化するものです。

このスクリプトを使った手順は次のとおりです。

このスクリプトの２行目に使い方、３行目に依存パッケージのインストールについて記載されています。まず、必要なパッケージをインストールします。

```
$ apt-get install dvi2ps-fontdata-a2n dvi2dvi dvipng
```

次にスクリプトを実行します。

```
$ ./htplatex debianmeetingresume200708.tex jp,2,sections+
```

すると、./html/ディレクトリ下に LATEX から変換された HTML が生成されます。これを pandoc を使って ePUB に変更します。

```
$ cd html
$ pandoc -o debianmeetingresume200708.epub debianmeetingresume200708*.html
```

なお、このスクリプトは実行時に、”-e”オプションをつけることで、 ePUB を生成することができるようにしました。

```
$ ./htplatex -e debianmeetingresume200708.tex jp,2,sections+
```

変更内容は下記リンク先をご参照下さい。

http://goo.gl/2KshN0

13.4.2 calibre

calibre では次の課題があります。

- 目次のレイアウトが崩れる
- デフォルトでは行間が広すぎる
- 図が表示されない場合もある
- tabular が表として表示されない

変換時に、”ヒューリスティック処理を有効にする”にチェックを入れ、”外観”の”段落の間の間隔を削除する”にチェックを入れると、一部は多少改善されます。

13.5 結論

現時点では、 LATEX からの ePUB 生成は、不完全ながらも一応可能なようです。どれを選択するかはユーザ次第ではありますが、いずれにしても更に使えるようにするには、各ツールでのパターンマッチングの機能を充実させる必要があるのではないかと思います。

しかし、 iPad mini くらいの大きさのデバイスでは、 Debian 勉強会の資料の PDF 版でもギリギリ拡大させなくても読めてしまいます。ですので、 PDF をそのまま読むのが一番読みやすい、という状況を解消できるようにしていくのが、我々の今後のアクションとなるに違いないでしょう。

13.6 追記

勉強会当日のディスカッションの結論としては、リフロー可能な PDF リーダをフリーソフトウェアとして作るのが正だろう、という話に落ち着きました。ちなみに自由ではないソフトウェアとしては、 Adobe Reader ならリフロー可能なよ

[*26] http://lists.debian.or.jp/debian-users/200708/msg00110.html

うです。 http://code.kzakza.com/2013/05/pdf_reflow/

参考文献

[1] What ’ s the best TEX-to-HTML or TEX-to-ePUB converter? http://boolesrings.org/krautzberger/2013/01/05/whats-the-best-tex-to-html-or-tex-to-epub-converter/

[2] Tools for Converting LATEXto XML http://jblevins.org/log/xml-tools

[3] Pandoc http://johnmacfarlane.net/pandoc/README.html

[4] LXir http://www.lxir-latex.org/

[5] Hermes http://hermes.roua.org/

[6] TEXWiki http://oku.edu.mie-u.ac.jp/~okumura/texwiki/

[7] LATEX をウエッブに載せよう http://osksn2.hep.sci.osaka-u.ac.jp/~naga/miscellaneous/tex4ht/tex4ht-howto.html

[8] LATEX2EPUB http://kmuto.jp/d/index.cgi/computer/latex2epub.htm

14 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. Stefano Zacchiroli さんが新しく作成したサービスは?

☐ A chiebukuro.debian.net
☐ B 2ch.debian.net
☐ C sources.debian.net

問題 2. 新しく FTP master チームに入った人は?

☐ A Gergely Nagy
☐ B Kouhei Maeda
☐ C Joerg Jaspert

問題 3. Debian GNU/Hurd がリリースされましたが、バージョンはいくつでしょう。

☐ A 2013
☐ B 3.141592
☐ C 7.0

問題 4. Sylvestre Ledru が JDK についてアナウンスしたのは

☐ A OpenJDK7 にきりかえ
☐ B JDK6 の削除
☐ C JDK8 への移行

問題 5. OpenJDK 7でサポートされていないアーキテクチャはどれか

☐ A mipsel
☐ B amd64
☐ C i386

問題 6. Summer Of Code のコーディネーションメンバーでないのは誰か

☐ A David Bremner
☐ B Nicolas Dandrimont
☐ C Nobuhiro Iwamatsu

問題 7. Brian Gupta が Debian のトレードマークとして USPTO に追加登録しようと提案したのは何か

☐ A ロゴ
☐ B Debian の文字列
☐ C DD

問題 8. alioth になにがおきたか

☐ A 懸賞があたった
☐ B RAID のハードディスクが 2 個壊れた
☐ C 新アーキテクチャに移行した

問題 9. DSA が DPL の承認なく使える予算はいくらか

☐ A $ 0
☐ B $ 100
☐ C $ 400

問題 10. Jessie のフリーズはいつか

☐ A 2013 年 11 月 5 日
☐ B 2014 年 11 月 5 日
☐ C 2015 年 11 月 5 日

問題 11. policy 3.9.5.0 によるとバイナリパッケージ内の
ファイル名のエンコーディングはなにか

☐ A UTF-8

☐ B Latin-1

☐ C sjis

15 索引

16 Debian Trivia Quiz 問題回答

上川 純一、岩松 信洋

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

1. C Stefano Zacchiroli さんが、 Debian パッケージで提供されているソースコードすべてを閲覧・検索出来る sources.debian.net を作った
2. A Paul Tagliamonte,Scott Kitterman,Luke Faraone, Gergely Nagy が加入
3. A
4. A java-common を OpenJDK 7 に切り替えるそうですよ、しかし一部のアーキテクチャは OpenJDK 6 のまま。
5. A s390, mips, mipsel, kfreebsd, sparc がサポートされていないようです。
6. C 例年 Summer of code スポンサーで開発する内容を調整するボランティアがいます。
7. A 'Debian' は登録されていたがロゴは登録されていなかったので登録することにしたようです。
8. B RAID のハードディスクが 2 つ壊れてファイルシステムが壊れたそうです。
9. C ディスクが壊れて交換するのにも DPL をまたないといけなかったのかな。
10. B あと一年ありますね
11. A とうとう ASCII 以外が認められるようになりましたか。

『 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 2013/06〜2013/11 まで東京エリアは第 102 回から第 106 回まで (第 104 回はキャンセルのため収録無し、第 105 回東京エリア Debian 勉強会/オープンソースカンファレンス 2013 Tokyo/Fall 分を含む) および、関西エリアは第 74 回から第 78 回まで (第 74 回は OSC2013Kansai@Kyoto のため収録無し、第 78 回は関西オープンソース 2013 のため収録無し)。東京エリア第 101 回、関西第 73 回は大統一 Debian 勉強会 2013 のため別冊。内容は無保証、つっこみなどがあれば勉強会にて。

あんどきゅめんてっど でびあん 2013 年冬号

2013 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）