



関西 Debian 勉強会担当者 佐々木・倉敷・のがた

2011 年 06 月 26 日

1 Introduction

Debian JP



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
- 定期的に来まれる場所
- 資料の作成

それでは, 楽しい一時をお楽しみ下さい.

勉強会

Debian

関西

目次

1	Introduction	1
2	最近の Debian 関係のイベント報告	3
3	事前課題	5
4	IPv6 のトンネル接続を試してみた話	9
5	vcs-buildpackage ~git,svn 編~	23
6	今後の予定	30
7	メモ	31

2 最近の Debian 関係のイベント報告

Debian JP



2.1 OSC 2011 Sendai 出張 Debian 勉強会

2011 年 5 月 21 日 (土) に宮城県仙台市でオープンソースカンファレンス 2011 仙台が開催されました。Debian 勉強会では吉田@板橋が展示で出展しました。

1. 「東京エリア Debian 勉強会/関西 Debian 勉強会」の紹介
2. 「あんどきゅめんでっどでびあん (Debian 勉強会資料のサマリ)」の展示、販売
3. 「Debian Squeeze の展示」具体的には「Debian GNU/kFreeBSD」の展示

主に上記をおこないました。メインホール入り口正面という配置にも恵まれ、多くの方に立ち寄って頂きました。



また、関西からはのがたさんが参加されています。今日時間があれば、お話を伺いたい所です。

2.2 OSC 2011 Hokkaido 出張 Debian 勉強会

2011 年 6 月 11 日 (土) に北海道札幌市でオープンソースカンファレンス 2011 北海道が開催されました。

Debian 勉強会ではタイトル「Debian 6.0(squeeze) & Debian Update」として佐々木がセッションを行いました。セッションには 14 名が参加し、質問ではバグ報告の手順を確認する質問が出されました。

GPG キーサインパーティの参加者はコーディネーターを含めて 3 名でした。「ある程度知識のある人々は既に必要な人とキーサインを終えているし、そうでない人はそもそも GPG に興味すらない現状をなんとかしないとイケないですね」というご意見も頂きました。GPG の啓蒙活動を引き続き行っていく必要があります*¹。

*¹ 来月の OSC KansaiKyoto でも GPG キーサインパーティを開催する予定です。よろしくお祈りします!



2.3 第 77 回東京エリア Debian 勉強会

2011 年 6 月 18 日 (土) に第 77 回東京エリア Debian 勉強会が開催されました。上川さんによる XSLT による Debian JP 定例会議議事録処理系の話や前田さんによる Sphnix および Doxygen の話がありました。詳細は PDF 公開資料をご覧ください*²

2.4 Debian 6.0.2 released

Squeeze の二つ目の point release となる Debian 6.0.2 が 2011 年 6 月 25 日 (この原稿を書いている最中) にリリースされました。

<http://www.debian.org/News/2011/20110625>

多くの bugfix、security fix がなされています。詳細は上記 URL をご覧ください。

2.5 Debian JP なう

ご存知の方もいらっしゃるかもしれませんが、twitter アカウント [debianjp](#) が最近活発に活動中です。Debian JP の中の人々が呟いております。まだフォローしていない方は是非どうぞ!

*² Doxygen についてはちょっと深いところまで掘り進めている所なので、そのうち機会があったらお話できると良いな、と思っています (佐々木)

3 事前課題

Debian JP



今回は以下の事前課題を設定しました。

1. <http://test-ipv6.jp/> で IPv6 接続性をテストしてください。(出来れば複数の環境で)
2. World IPv6 Day (以降) で何か影響がありましたか。(例: 特定のサイトへのアクセスが遅くなった、何度かリロードしないと表示されなくなった)
3. Debian パッケージを作成した事の無い人は Debian のソースパッケージの構成を復習しておいて下さい。

参加者の皆さんによる回答は以下の通りです (一部こちらで整形してます。ご容赦下さい)。

3.1 佐々木洋平

直前に回答する駄目な人です。

1. 北大、京都大、神戸大、九大の学内にある計算機からテストしましたが、v6 は軒並駄目です。とはいえ、大学毎の情報基盤センター (に類する所) でトンネルなりなんなり噛ませていると思いますので、まあそういうモンかな、とか思ったり。
2. 特にこれといって不具合もなく。
3. source format 3.0 (git) とか soruce format 3.0 (bzt) というモノの存在を知ってモニョッとしました。

3.2 古川竜雄 (frkwtt@gmail.com)

1. <http://test-ipv6.jp/> で IPv6 接続性をテストしてください

```
* 一般のインターネット上で見えるあなたの IPv4 アドレスは ***
* IPv6 アドレスが検出されませんでした
* World IPv6 Day は 2011 年 6 月 8 日です。このブラウザ、この場所からは問題ないことが予想されます。
* あなたは IPv4 インターネットのみを閲覧できるようです。あなたは IPv6 のみのサイトに到達できません。
* あなたの DNS サーバ (おそらくお使いのプロバイダが運用) は、IPv6 インターネットアクセスがない、もしくは利用しないように設定されているようです。このあなたの事前対応点
10/10 コンテンツ側が IPv4 および IPv6 を提供した際の、あなたの IPv4 の安定性と準備状況
0/10 コンテンツ側が IPv6 のみになった際の、あなたの IPv6 の安定性と準備状況
```

Yahoo BB とぶららで全く同じ結果を示しました。

```
# cat /proc/version
Linux version 2.6.26-2-686 (Debian 2.6.26-26lenny1) (dannf@debian.org) \
(gcc version 4.1.3 20080704 (prerelease)
(Debian 4.1.2-25)) #1 SMP Thu Nov 25 01:53:57 UTC 2010
```

2. World IPv6 Day (以降) で何か影響がありましたか。
何も変わりません。IPv6 を使ってないから??

3.3 水野源

1. 我が家は v4 しか使えません!
2. 特になし

3.4 やつお

1. 3ヶ所から試してみましたが全て「× IPv6 アドレス が検出されませんでした」となりました。
2. あまり意識していないのですが、特に問題無いと思います。
3. 勉強しておきます...

3.5 kaimasami0001

課題 1 次の (a),(b)2 環境を用意しました。

- (a). ネイティブ debian squeeze(64bit)
- (b). 上の環境上の kvm-qemu 上の debian squeeze(32bit)

両環境とも OS 上では IPv6 の設定をしていない、私のプロバイダは今年 5~7 月に IPv6 に対応中、ということと、<http://test-ipv6.jp/>への接続結果もそれなりの結果でした。

10/10	コンテンツ側が IPv4 および IPv6 を提供した際の、あなたの IPv4 の安定性と準備状況
0/10	コンテンツ側が IPv6 のみになった際の、あなたの IPv6 の安定性と準備状況

課題 2 World IPv6 Day (以降) での影響

10 秒~20 秒まっても表示しないサイトがありました。それまで見たことのないサイトだったので原因が IPv6 のせいかどうかは不明。

課題 3 Debian ソースパッケージの構成の復習

すみません。復習はまだです。

3.6 山田 洋平

1. IPv6 に対応していないけど、みんな同じだから大丈夫。みたいに言われました。(自宅からと携帯電話からと)
2. 話には聞いていましたが何も気付きませんでした。
3. 土曜日見ておきます。

3.7 小林 克希

1. 自宅も会社も IPv4 10/10, IPv6 0/10 でした。
2. あまり気づいた点はないです。(そもそも、昔に比べて web を見なくなってしまったのもあるのですが.....。)
3. 一応、Ubuntu に勝手に自前パッケージを取り込まれたことくらいはあるのですが(笑)、もう数年前の話でほとんど覚えてないので復習しておきます。

初参加の者です。参加表明がギリギリとなってしまう申し訳ございません。昨年の 10 月に大阪に引っ越してきました。Debian は学生時代はよくいじっていたのですが、最近はやさぱりで squeeze のリリースにも全然気づいてませんでした.....。せっかくの機会ですので、また少しいじってみたいと思っています。よろしくお願ひします。

3.8 かわだてつたろう

1. WiMAX から行ないましたが事前対応点の IPv6 は 0/10 でした。
2. 影響を感じるようなことはありませんでした。
3. hello のパッケージを再作成してみました。DEP-5 フォーマットで書くのはかなり大変ですね。

3.9 佐藤誠

1. 一応アクセスはしましたが、V6 を優先にできませんでした。
2. 基本的に非対応の環境におりました。何も変わったことはないです。
3. 確認します。

3.10 松澤二郎

1.

```
IPv4 DNS レコードのテスト
ok (0.456s) 利用 ipv4
IPv6 DNS レコードのテスト
bad (0.365s)
デュアルスタック DNS レコードのテスト
ok (0.506s) 利用 ipv4
デュアルスタック DNS と大きなパケットのテスト
ok (0.364s) 利用 ipv4
DNS を利用しない IPv4 のテスト
ok (0.332s) 利用 ipv4
DNS を利用しない IPv6 のテスト
bad (0.054s)
IPv6 ラージパケットのテスト
bad (0.043s)
お使いのプロバイダの DNS サーバの IPv6 の利用状況のテスト
bad (0.885s)
```

2. 特にありません。
3. 実用途ではパッケージ作成は未経験です。「Debian パッケージ 60 分クッキング」を見ました。

3.11 木下

1. IPv6 接続性テスト

```
IPv4 DNS レコードのテスト ... ok (0.486s) 利用 ipv4
IPv6 DNS レコードのテスト ... bad (0.017s)
デュアルスタック DNS レコードのテスト ... ok (3.497s) 利用 ipv4
デュアルスタック DNS と大きなパケットのテスト ... ok (0.078s) 利用 ipv4
DNS を利用しない IPv4 のテスト ... ok (0.482s) 利用 ipv4
DNS を利用しない IPv6 のテスト ... bad (0.004s)
IPv6 ラージパケットのテスト ... bad (0.004s)
お使いのプロバイダの DNS サーバの IPv6 の利用状況のテスト ... ok (3.453s) 利用 ipv4
```

結論：私の自宅のネット環境では IPV6 は無理かも ...

2. World IPv6 Day (以降) で何か影響がありましたか。
今のところ、特になし。

3.12 lurdan

1. IPv6 でつながる環境は所持していないようです
2. 特にネガティブな影響もなく、完全に他人事でした。
3. いちおう。してる。つもり。

3.13 よしだともひろ

1. <http://test-ipv6.jp/> での接続性テスト (自宅で実施した際のサマリー)

```
一般のインターネット上で見えるあなたの IPv4 アドレスは XXX.XXX.XXX.115
IPv6 アドレスが検出されませんでした
World IPv6 Day は 2011 年 6 月 8 日です。このブラウザ、この場所からは問題ないことが予想されます。
あなたは IPv4 インターネットのみを閲覧できるようです。あなたは IPv6 のみのサイトに到達できません。
あなたの DNS サーバ (おそらくお使いのプロバイダが運用) は、IPv6 インターネットアクセスがあるように思われます。
あなたの事前対応点
10/10 コンテンツ側が IPv4 および IPv6 を提供した際の、あなたの IPv4 の安定性と準備状況
0/10 コンテンツ側が IPv6 のみになった際の、あなたの IPv6 の安定性と準備状況
```

2. World IPv6 Day (以降) で何か影響がありましたか。
現在のところ、特に問題はないように思います。
3. Debian のソースパッケージの構成を復習
4. つい最近『入門 Debian パッケージ』を入手して、読み始めたところです。「古いので今から読むのはやめといった方がよい」というようなことはないでしょうか？

3.14 kozo2

1. WiMAX でこのようになりました

```
IPv4 DNS レコードのテスト
ok (0.208s) 利用 ipv4
IPv6 DNS レコードのテスト
bad (0.106s)
デュアルスタック DNS レコードのテスト
ok (0.140s) 利用 ipv4
デュアルスタック DNS と大きなパケットのテスト
ok (0.207s) 利用 ipv4
DNS を利用しない IPv4 のテスト
ok (0.112s) 利用 ipv4
DNS を利用しない IPv6 のテスト
bad (0.013s)
IPv6 ラージパケットのテスト
bad (0.012s)
お使いのプロバイダの DNS サーバの IPv6 の利用状況のテスト
bad (0.011s)
```

2. 特にありませんでした
3. 前回 (第 47 回 関西 Debian 勉強会) の lurdan さんの dpkg のおさらいを見ました

3.15 川江

1. 私の事前対応状況は以下です。

```
10/10 コンテンツ側が IPv4 および IPv6 を提供した際の、あなたの IPv4 の安定性と準備状況
0/10 コンテンツ側が IPv6 のみになった際の、あなたの IPv6 の安定性と準備状況
2. 特になし。
```

3.16 山下康成

1. (藁

```
10/10 コンテンツ側が IPv4 および IPv6 を提供した際の、あなたの IPv4 の安定性と準備状況
0/10 コンテンツ側が IPv6 のみになった際の、あなたの IPv6 の安定性と準備状況
```

2. 何も気がつきませんでした
3. Debian パッケージ 60 分クッキングをざっと見ました。必要な技能は備わっているようです。sh script は読み書きできます。Makefile もまあ読み書きできます。好奇心は、、、

4 IPv6 のトンネル接続を試してみた話

西山和広



4.1 概要

今回の話は、プロバイダが IPv6 ネイティブ接続にまだ対応していない状況、つまり直接外に繋がるのは IPv4 のみの接続の環境で IPv6 を使う話です。

4.2 IPv6 概要

4.2.1 IPv6 とは何か

最近情報は増えてきているので省略します。古い情報だと RFC が更新されていたり廃止されていたりして、現状とは合わなくなっているものもあるので注意が必要です。

4.2.2 IPv6 の利点と欠点

IPv6 の利点としては以下のようなものが思いつきます。

- アドレス空間が広い
- 実装が普及している
- 機能的な利点はそんなにない
 - IPsec とか Mobile IP とか IPv4 でも可能

IPv6 の欠点としては以下のようなものが思いつきます。

- IPv4 と互換性がない
- まだ広く利用されていない
 - トラブルの対処方法とかあまりない

4.2.3 なぜ IPv6 を試そうと思ったか

試し始めてから World IPv6 Day が実施されるなど、状況がどんどん変わっていますが、試そうと思った最初の理由は以下のようなものです。

- JPNIC の IPv4 アドレスも枯渇したから
- おもしろそうだったから
- 余裕のあるうちにのんびりとやりたかったから

- 必要になってからあわててやりたくない

4.2.4 IPv6 対応とは

IPv6 対応にはいろいろな状態があると思いますが、今回は以下の種類を考えてみました。

- クライアント側
 - IPv6 のみのサーバに接続できる (<http://ipv6.google.com/> など)
 - IPv4 と IPv6 両対応のサーバに IPv6 で接続できる (<http://www.kame.net/> など)
 - DNS を IPv6 経由で解決できる (/etc/resolv.conf で nameserver に IPv6 アドレスを設定)
 - * これが出来ないと IPv6 のみに移行できない
- サーバ側
 - IPv6 アドレス指定で接続できる (グローバル IPv6 アドレス設定)
 - ホスト名で指定した場合でも IPv6 で接続できる (DNS に AAAA レコード設定)

4.2.5 IPv6 アドレスの表記

IPv6 アドレスは 128 ビットを 16 ビットごとに「:」で区切って 16 進数で表記します。省略表記が RFC 4291 で決まっていますが、省略の仕方で複数の省略表記が可能なので、RFC 5952 で推奨表記が決められました。

- 例: 2001:0db8:0000:0000:0000:0000:0000:0001
- RFC 4291, RFC 5952 のルールで省略表記
 - 先頭の 0 を省略 2001:db8:0:0:0:0:1
 - 0 の連続は 1 回だけ :: で省略 2001:db8::1
- 他にはアルファベットは小文字推奨など
- IPv4 のネットワークアドレスやサブネットマスクに相当するものは 2001:db8::/32 の /32 のようにネットワークプレフィックスとそのビット長を付けて表記します。

2001:db8::/32 は例示用 IPv6 アドレス (RFC 3849) になっているなど、用途により IP アドレスの範囲が決まっています。(RFC 5156)

4.3 トンネル

4.3.1 トンネルの種類

今回試したものは

- teredo (RFC 4380)
- 6to4 (RFC 3056)
- 6rd (RFC 5969)

の 3 種類です。ISATAP (RFC 5214) など他の方式もありますが、今回の話の対象外です。

4.4 teredo

4.4.1 teredo の特徴

teredo は以下の特徴があります。

- NAT の中でも使えるトンネル
- UDP/IPv4 にカプセル化して通信
- クライアントで使うには手軽で簡単

- サーバには向かない (仕組みを考えるとたぶん無理)

この特徴により NAT の中から IPv6 接続したい人には便利なのですが、外部との接続を制限すべき環境では、firewall などで UDP の 3544 番ポートへの接続を制限する必要があります。

4.4.2 IPv6 アドレス

- 接続先でも 2001:0000::/16 のアドレス (省略しない場合 2001:0000: で始まる文字列の IPv6 アドレス) は teredo とわかるので WWW:WWW の部分を逆引きするなどの対処が可能です。
- 2001:0000:XXXX:XXXX:YYYY:ZZZZ:WWW:WWW の形式
 - XXXX:XXXX は Teredo サーバの IPv4 アドレスを 16 進数にしたもの
 - YYYY は NAT の種類などのフラグ
 - ZZZZ はクライアントの外部 UDP ポート番号を変換したもの
 - WWW:WWW は Teredo クライアントの外部 IPv4 アドレスを変換したもの

4.4.3 使い方

- Windows は XP 以降で対応しています。
- Debian では miredo パッケージをインストールするだけで teredo で接続できます。
 - 自動で起動
 - /etc/miredo.conf で設定
 - デフォルトの ServerAddress の teredo-debian.remlab.net はフランスなので、アメリカにある teredo.ipv6.microsoft.com に変更した方が良いかもしれません

4.4.4 接続確認

/sbin/ifconfig で teredo の存在を確認します。

```
$ /sbin/ifconfig teredo
teredo    Link encap:不明なネット  ハードウェアアドレス 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
inet6 アドレス: fe80::ffff:ffff:ffff/64 範囲:リンク
inet6 アドレス: 2001:0:4137:9e76:34c1:f58:XXXX:XXXX/32 範囲:グローバル
UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1280  メトリック:1
RX パケット:0 エラー:0 損失:0 オーバラン:0 フレーム:0
TX パケット:3 エラー:0 損失:0 オーバラン:0 キャリア:0
衝突 (Collisions):0 TX キュー長:500
RX バイト:0 (0.0 B)  TX バイト:144 (144.0 B)

$
```

接続に成功しているとき、ログ (/var/log/syslog) に以下のように出ます。

```
miredo[4105]: Starting...
miredo[4107]: New Teredo address/MTU
miredo[4107]: Teredo pseudo-tunnel started
miredo[4107]: (address: 2001:0:4137:9e76:34c1:f58:XXXX:XXXX, MTU: 1280)
```

4.5 6to4

4.5.1 6to4 の特徴

6to4 はグローバル IPv4 アドレスが必要ですが、申し込みなどをしなくても誰でも自由に使えます。

トンネルの方法として、プロトコル 41 (TCP や UDP ではない) でカプセル化して IPv4 で通信します。(他の主なプロトコルの番号は ICMP: 1, TCP: 6, UDP: 17 でその層のプロトコルを使用しています。)

anycast で自動的に近いリレールータを経由 (192.88.99.1) します。

4.5.2 6to4 の問題点

6to4 には以下のような問題点があるため、廃止が検討されています。

- 往復の経路は基本的に異なる
- 通信経路が把握できないし制御もできない
- トラブルがおきると対処が困難

4.5.3 IPv6 アドレス

接続先でも 2002::/16 のアドレス (2002: で始まる文字列の IPv6 アドレス) は 6to4 とわかるので、アクセスログ解析などでは xxx.yyy.zzz.www を逆引きするなどの対処が可能です。

2002:XXYY:ZZWW::/48 が使用可能

- XXYYZZWW は IPv4 アドレスの xxx.yyy.zzz.www を 16 進数にしたもの

2002:XXYY:ZZWW:VVVV::/64 を LAN に割り当てるなどの使い方が可能 (VVVV は任意の値)

4.5.4 使い方

基本的には「/etc/network/interfaces」で設定するだけで使えます。

6to4 による IPv6 接続 (Linux 編) << さくらインターネット研究所 <http://research.sakura.ad.jp/2010/12/27/tunnel-6to4-linux/>などを参考にして設定します。

以下は会社のマシンで設定したときの例です。

まず 6to4 のアドレスを計算します。

```
$ printf "2002:%02x%02x:%02x%02x::1\n" 220 218 54 201
2002:dcda:36c9::1
```

次に tun6to4 という iface の設定を追加します。

```
$ sudoedit /etc/network/interfaces
auto tun6to4
iface tun6to4 inet6 v4tunnel
address 2002:dcda:36c9::1
netmask 16
gateway ::192.88.99.1
local 220.218.54.201
endpoint any
ttl 64
```

最後に「sudo ifup tun6to4」で有効にします。「auto tun6to4」も設定しているので、再起動でも有効になるはず

4.5.5 接続確認

/sbin/ifconfig で tun6to4 を確認します。

```
$ /sbin/ifconfig tun6to4
tun6to4  Link encap:IPv6-in-IPv4
          inet6 アドレス: 2002:dcda:36c9::1/16 範囲:グローバル
          inet6 アドレス: ::220.218.54.201/128 範囲:Compat
          UP RUNNING NOARP MTU:1480 メトリック:1
          RX パケット:55939508 エラー:0 損失:0 オーバラン:0 フレーム:0
          TX パケット:84141144 エラー:1175 損失:0 オーバラン:0 キャリア:886
          衝突 (Collisions):0 TX キュー長:0
          RX バイト:5470501110 (5.0 GiB) TX バイト:88417369465 (82.3 GiB)

$
```

4.6 6rd

4.6.1 6rd の特徴

- 6to4 と同様にリレールータを経由
- リレールータはプロバイダが用意
- プレフィックスもプロバイダのものになる

- teredo や 6to4 と違って IPv4 の方が優先されるということがない

4.6.2 使い方

squeeze のカーネル 2.6.32 は 6rd に対応していないバージョンなので、バックポートカーネルをインストールします。backports の apt-line を適切に設定した後、「sudo aptitude install -t squeeze-backports linux-image-2.6.38-bpo.2-amd64」でインストールします。依存関係で linux-base も更新されるようです。

6rd による IPv6 接続 (概要編) << さくらインターネット研究所 <http://research.sakura.ad.jp/2011/01/05/tunnel-6rd-intro/> を参考にして「/etc/network/interfaces」に設定します。

以下はさくらの VPS で squeeze にしているマシンで設定した例です。

まず 6rd のアドレスを計算します。

```
$ printf "2001:55c:%02x%02x:%02x%02x::1\n" 49 212 40 201
2001:55c:31d4:28c9::1
$
```

次に tun6rd という iface の設定を追加します。

```
$ sudoedit /etc/network/interfaces
auto tun6rd
iface tun6rd inet6 v4tunnel
    address 2001:e41:31d4:28c9::1
    netmask 32
    local 49.212.40.201
    endpoint any
    gateway ::61.211.224.125
    ttl 64
    up ip tunnel 6rd dev tun6rd 6rd-prefix 2001:e41::/32
    up ip link set mtu 1280 dev tun6rd
```

最後に「sudo ifup tun6rd」で有効にします。「auto tun6rd」も設定しているので、再起動でも有効になるはず

4.6.3 接続確認

/sbin/ifconfig で tun6rd を確認します。

```
$ /sbin/ifconfig tun6rd
tun6rd    Link encap:IPv6-in-IPv4
          inet6 addr: 2001:e41:31d4:28c9::1/32 Scope:Global
          inet6 addr: ::49.212.40.201/128 Scope:Compat
          UP RUNNING NOARP MTU:1280 Metric:1
          RX packets:16336 errors:0 dropped:0 overruns:0 frame:0
          TX packets:21361 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1854845 (1.7 MiB) TX bytes:6845907 (6.5 MiB)

$
```

4.7 6to4 ルータ

4.7.1 IPv6 ルータとは

- ネットワークプレフィックスや有効期限を広告 (RA (Router Advertisement): ルータ広告)
- RA を受け取った端末が IPv6 アドレスを生成して自動設定 (RFC 4862)
- RA によるステートレスアドレス自動設定 (SLAAC)
 - 端末がネットワークに繋がると RS (Router Solicitation: ルータ要請) を「ff02::2」(全ルータアドレスあてのマルチキャスト) に送信
 - ルータが RA を「ff02::1」(全ノードアドレスあてのマルチキャスト) に送信
 - RA のプレフィックスを使って IPv6 アドレス生成
 - 重複アドレス検出 (DAD) して問題なければ利用開始
- IPv4 の DHCP と違って RA の送信元をデフォルトルートとして自動設定
- DNS 設定配布は別途考える必要あり

- DNS の設定は今のところ別途 DHCPv6 を使うのが無難?
- RA の RDNSS というオプション (RFC 6106) は使われていない?

DNS 設定についてはすぐにはわからなかったもので、今回は IPv4 の DNS をそのまま使うことにして、IPv6 の DNS サーバ設定はしませんでした。

セキュリティ問題も関係しているのので、このあたりの仕様はまだ更新される可能性が高いです。

4.7.2 LAN 側 IPv6 アドレス設定

まず LAN 側に設定するネットワークプレフィックスを決めます。

今回は 6to4 により 2002:dcda:36c9::/48 が自由に使えます。その中から JPNIC の枯渇の日の 4 月 15 日を埋め込んで 2002:dcda:36c9:415::/64 を使うことにしました。

ルータ広告を送信する iface には固定の IPv6 アドレスが必須のようなので、今回はわかりやすいように先頭の 2002:dcda:36c9:415::1 を設定しました。

ネットワーク設定全体としては以下のようにしました。

```
$ cat /etc/network/interfaces
auto lo
iface lo inet loopback

allow-hotplug eth0
iface eth0 inet static
address 192.168.0.2
netmask 255.255.255.0
iface eth0 inet6 static
address 2002:dcda:36c9:415::1
netmask 64

allow-hotplug eth1
iface eth1 inet static
address 220.218.54.201
netmask 255.255.255.240
gateway 220.218.54.193

auto tun6to4
iface tun6to4 inet6 v4tunnel
address 2002:dcda:36c9::1
netmask 16
gateway ::192.88.99.1
local 220.218.54.201
endpoint any
ttl 64
$
```

4.7.3 radvd

radvd は RA (ルータ広告) を送信するデーモンです。

インストールしただけでは自動起動しないので、`/usr/share/doc/radvd/README.Debian` を参考にして設定して起動します。

```
$ sudo /etc/init.d/radvd start
Starting radvd:
* /etc/radvd.conf does not exist or is empty.
* See /usr/share/doc/radvd/README.Debian
* radvd will *not* be started.
$
```

まず README.Debian などを参考にして `/etc/sysctl.d/ipv6.conf` を作成しました。再起動前に反映させるには「`sudo sysctl -p /etc/sysctl.d/ipv6.conf`」です。

```
$ cat /etc/sysctl.d/ipv6.conf
net.ipv6.conf.all.accept_ra=0
net.ipv6.conf.all.forwarding=1
```

`/etc/radvd.conf` は `/usr/share/doc/radvd/examples/radvd.conf.example` を参考にして以下のように設定しました。

グローバル IP アドレスから一意に決まるものをわざわざ書くのは嫌だったので、prefix の先頭は「0:0:0」にして「`Base6to4Interface eth1;`」で自動設定するようにしています。

```

$ cat /etc/radvd.conf
interface eth0
{
    AdvSendAdvert on;
    MinRtrAdvInterval 3;
    MaxRtrAdvInterval 10;
    AdvDefaultPreference low;
    AdvHomeAgentFlag off;
    prefix 0:0:0:0415::/64
    {
        AdvOnLink on;
        AdvAutonomous on;
        AdvRouterAddr off;
        Base6to4Interface eth1;
        AdvPreferredLifetime 120;
        AdvValidLifetime 300;
    }
};

```

後は「`sudo /etc/init.d/radvd start`」して LAN 内のマシンにグローバル IPv6 アドレスが割り振られることを確認したり `http://ipv6.google.com/` が表示できることを確認したりします。

4.7.4 libvirt に IPv6 設定

この libvirt の設定だけ Debian squeeze ではなく Ubuntu 11.04 (natty) で確認しています。

「`virsh net-edit default`」で network 要素直下に「`<ip family='ipv6' address='2002:dcda:36c9:415::1' prefix='64' />`」のように設定を追加します。普通は「`</network>`」の行の直前に追加すれば良いと思います。

試した環境では一度保存すると以下のように 2 行になってしまいましたが、XML 的にはほぼ同じなので気にしなくても良いと思います。

```

<ip family='ipv6' address='2002:dcda:36c9:415::1' prefix='64'>
</ip>

```

全体的な例は <http://libvirt.org/formatnetwork.html> を参考にしてください。

これだけで自動的に `virbr0` に「`2002:dcda:36c9:415::1/64`」のアドレスが振られたり `radvd` が起動したりします。

4.8 プライバシ拡張アドレス

RA によるステートレスアドレス自動設定 (SLAAC) で現状の Linux の実装などでは MAC アドレスを元にしたアドレスになります。(MAC アドレスを 1 ビット変更して FF FE を真ん中に入れる)

そのため、接続する IPv6 ネットワークが変わって prefix が違ってても接続元マシンが特定できる可能性があります。

その対策として、プライバシ拡張アドレス (RFC 3041) の機能を有効にすればランダムな値から IPv6 アドレスが生成されるようになります。

実際の設定については試していないので紹介しておくだけにします。

- Ubuntu Linux で匿名アドレス (RFC3041) を有効にする <http://dr.slump.jp/IPv6/rfc3041/>
 - 例: `sysctl -w net.ipv6.conf.eth0.use_tempaddr=2`
- 高木浩光@自宅の日記 - Mac ユーザは IPv6 を切るか `net.inet6.ip6.use_tempaddr=1` の設定を <http://takagi-hiromitsu.jp/diary/20080730.html>
- iPhone、RFC3041 (IPv6 プライバシ拡張) に対応
 Kenichi Maehashi's Blog <http://blog.kenichimaehashi.com/?article=13044202150>
 - iOS 4.3 のセキュリティコンテンツについて
http://support.apple.com/kb/HT4564?viewlocale=ja_JP&locale=ja_JP

4.9 firewall

4.9.1 注意点

IPv6 ではルータではない一般のノードでもユニキャストアドレスやループバックアドレス以外に、リンクローカルアドレスやマルチキャストアドレスなど複数の IPv6 アドレスを持つので、firewall の設定に IPv4 より注意が必要です。

IPv4 で OP25B などの設定をしている場合は teredo などが抜け穴にならないように注意が必要です。

4.10 iptables-apply

- iptables パッケージに入っている ip6tables-restore をリモートからでも安全に実行できるようにするものです。
- タイムアウトするまでに y を入力しなかったら元のルールに戻してくれるので、設定をミスして ssh の接続が遮断されるルールにしようとしてしまっても安全です。
- /usr/sbin/iptables-apply で /etc/network/iptables を適用できるのと同様に /usr/sbin/ip6tables-apply で /etc/network/ip6tables を適用できます。
- 起動時にも適用するには「/sbin/ip6tables-restore < /etc/network/ip6tables」をどこかで実行する必要があります。
- たとえば以下のように lo の pre-up で実行します。

```
iface lo inet loopback
pre-up /sbin/iptables-restore < /etc/network/iptables
pre-up /sbin/ip6tables-restore < /etc/network/ip6tables
```

4.11 ufw

- Uncomplicated FireWall の略
 - Ubuntu FireWall ではない
- sudo ufw allow OpenSSH や sudo ufw allow 80/tcp などのように簡単に使える iptables のラッパー

4.11.1 ufw の設定ファイル

/etc/default/ufw で基本的な設定をして、ufw コマンドでその他の設定をして、before{,6}.rules で特殊な設定をするということになります。

- /etc/default/ufw
 - デフォルトのポリシーなどを設定
- /etc/ufw/ufw.conf
 - 「ufw enable」や「ufw disable」などの設定が保存されている
- /lib/ufw/user{,6}.rules
 - ufw allow などでの設定を保存
 - なぜか /lib 以下にある
- /etc/ufw/before{,6}.rules
 - 必要なら編集
 - ポートフォワーディングなどの nat テーブルの設定は ufw コマンドでは出来ないので、このファイルで設定

4.11.2 /etc/default/ufw

- IPV6=yes で IPv6 も有効にする
- IPV6=yes にする前に設定した ufw allow は IPv4 のまま
- IPv6 でも許可するには sudo ufw allow 22/tcp などを実行し直す
- from や to で IPv4 アドレスや IPv6 アドレスを指定すれば個別の設定も可能 (例: ufw allow in on virbr0 proto udp from 0.0.0.0/0 port 68 to 0.0.0.0/0 port 67)

こまめに「sudo ufw status」で確認するとわかりやすいです。IPv6 でも許可できていれば以下のように (v6) の行があります。

たとえば IPV6=no のときに「sudo ufw allow OpenSSH」で許可していて、IPV6=yes にしてから IPv6 でも許可した場合は以下ようになります。

```
$ sudo ufw status
Status: active

To Action From
--
OpenSSH ALLOW Anywhere

$ sudo ufw allow OpenSSH
Skipping adding existing rule
Rule added (v6)
$ sudo ufw status
Status: active

To Action From
--
OpenSSH ALLOW Anywhere
OpenSSH (v6) ALLOW Anywhere (v6)

$
```

4.11.3 /etc/ufw/before{,6}.rules

/etc/default/ufw で DEFAULTOUTPUTPOLICY を REJECT にした場合は ufw{,6}-before-input と同様の icmp などの許可を ufw{,6}-before-output にする必要がありました。

before.rules の ICMP の設定として

```
# ok icmp codes
-A ufw-before-input -p icmp --icmp-type destination-unreachable -j ACCEPT
-A ufw-before-input -p icmp --icmp-type source-quench -j ACCEPT
-A ufw-before-input -p icmp --icmp-type time-exceeded -j ACCEPT
-A ufw-before-input -p icmp --icmp-type parameter-problem -j ACCEPT
-A ufw-before-input -p icmp --icmp-type echo-request -j ACCEPT
```

の下に

```
-A ufw-before-output -p icmp --icmp-type destination-unreachable -j ACCEPT
-A ufw-before-output -p icmp --icmp-type source-quench -j ACCEPT
-A ufw-before-output -p icmp --icmp-type time-exceeded -j ACCEPT
-A ufw-before-output -p icmp --icmp-type parameter-problem -j ACCEPT
-A ufw-before-output -p icmp --icmp-type echo-request -j ACCEPT
```

を追加しました。

before6.rules の ICMPv6 の設定として

```
# for stateless autoconfiguration (restrict NDP messages to hop limit of 255)
-A ufw6-before-input -p icmpv6 --icmpv6-type neighbor-solicitation -m hl --hl-eq 255 -j ACCEPT
-A ufw6-before-input -p icmpv6 --icmpv6-type neighbor-advertisement -m hl --hl-eq 255 -j ACCEPT
-A ufw6-before-input -p icmpv6 --icmpv6-type router-solicitation -m hl --hl-eq 255 -j ACCEPT
-A ufw6-before-input -p icmpv6 --icmpv6-type router-advertisement -m hl --hl-eq 255 -j ACCEPT
```

```
# ok icmp codes
-A ufw6-before-input -p icmpv6 --icmpv6-type destination-unreachable -j ACCEPT
-A ufw6-before-input -p icmpv6 --icmpv6-type packet-too-big -j ACCEPT
-A ufw6-before-input -p icmpv6 --icmpv6-type time-exceeded -j ACCEPT
-A ufw6-before-input -p icmpv6 --icmpv6-type parameter-problem -j ACCEPT
-A ufw6-before-input -p icmpv6 --icmpv6-type echo-request -j ACCEPT
```

の下にそれぞれ

```
-A ufw6-before-output -p icmpv6 --icmpv6-type neighbor-solicitation -m hl --hl-eq 255 -j ACCEPT
-A ufw6-before-output -p icmpv6 --icmpv6-type neighbor-advertisement -m hl --hl-eq 255 -j ACCEPT
-A ufw6-before-output -p icmpv6 --icmpv6-type router-solicitation -m hl --hl-eq 255 -j ACCEPT
-A ufw6-before-output -p icmpv6 --icmpv6-type router-advertisement -m hl --hl-eq 255 -j ACCEPT
```

```
-A ufw6-before-output -p icmpv6 --icmpv6-type destination-unreachable -j ACCEPT
-A ufw6-before-output -p icmpv6 --icmpv6-type packet-too-big -j ACCEPT
-A ufw6-before-output -p icmpv6 --icmpv6-type time-exceeded -j ACCEPT
-A ufw6-before-output -p icmpv6 --icmpv6-type parameter-problem -j ACCEPT
-A ufw6-before-output -p icmpv6 --icmpv6-type echo-request -j ACCEPT
```

を追加しました。

MULTICAST は 5353/udp のみ許可の設定になっていますが、何か問題があれば変更します。

before.rules の MULTICAST 設定変更例としては

```
# allow MULTICAST mDNS for service discovery (be sure the MULTICAST line above
# is uncommented)
-A ufw-before-input -p udp -d 224.0.0.251 --dport 5353 -j ACCEPT
```

となっているのを以下に変更します。

```
# allow MULTICAST, be sure the MULTICAST line above is uncommented
-A ufw-before-input -s 224.0.0.0/4 -j ACCEPT
-A ufw-before-output -d 224.0.0.0/4 -j ACCEPT
```

上記のように ICMPv6 は許可しているので、before6.rules の変更は通常は必要ないと思います。

4.12 tcp wrapper の設定例

localhost と libvirt のデフォルトの LAN と今回設定した 6to4 経由の接続のみ許可する場合は以下のようになります。

- /etc/hosts.deny で「ALL: ALL」と設定
- /etc/hosts.allow で以下のように設定

```
sshd: 127.0.0.1 [::1]
sshd: 192.168.122.0/24
sshd: [2002:dcda:36c9:415::]/64
```

4.13 サーバ

4.13.1 サーバ一般

netstat -lnt で確認して tcp6 でも LISTEN していれば IPv6 対応しています。

```
$ netstat -lnt | grep ':22 '
tcp        0      0 0.0.0.0:*           LISTEN
tcp6       0      0 :::22             LISTEN
$
```

4.13.2 OpenSSH

- 少なくとも以下の設定が IPv6 対応に影響します。
 - /etc/ssh/sshd_config
 - * ListenAddress
 - * AllowUsers user@ipaddr の ipaddr 部分
 - tcp wrapper
 - iptables

4.13.3 Apache2

一部の VirtualHost を IPv6 対応環境に移動しましたが、IP アドレスに依存する設定をしていなかったため、何も問題はありませんでした。

4.13.4 munin

munin が使っているライブラリが IPv6 に対応していないという理由で munin も IPv6 に対応していませんでした。

4.14 トラブルシューティング

4.14.1 ping6 で最初のパケットだけ時間がかかる

- トンネル接続の場合はそういうもの

4.14.2 ping6 でパケットロスが多い

libvirt で libvirt で自動起動される radvd とは別に自分で radvd を起動してしまうと ping6 での確認のときに packet loss が 70% 以上になるなどわかりにくいトラブルになりました。

そのときに「radvd[...]: our AdvValidLifetime on eth0 for ... doesn't agree with ...」のようなログが出ていて、調べると <http://www.wiggy.net/texts/ipv6-howto/> に radvd のようなルータ広告 (RA) するデーモンが複数動いている場合におきる問題だということがわかったので、libvirt で自動起動されている radvd が存在するのを確認し、手動で起動していた方の radvd を止めることで解決しました。

4.14.3 ifconfig で teredo がない

- miredo のログ (/var/log/syslog) で原因を調べます。
- firewall で塞がれていないか確認します。
- NAT の種類によっては繋らないかもしれません
 - 以前のバージョンだと NAT の種類によっては「Unsupported symmetric NAT detected.」で繋らないことがありましたが、今の squeeze に入っている miredo 1.2.3-1 では対応しているように見えます。
 - VMware など NAT の段数を増やすと繋がったことがあります。

4.14.4 亀が踊らない

- <http://www.kame.net/> で亀が踊らないときの原因の調べ方
- IPv6 で接続可能か?
 - <http://ipv6.google.com/> が表示できるか?
 - * ipv6 は AAAA のみ設定されている
 - * 表示できなければ、そもそも IPv6 で外と繋がっていない可能性が高い
 - 詳細は <http://test-ipv6.com/> や <http://test-ipv6.jp/> でテストする
- getaddrinfo(3) を調べる
 - 繋がっていれば、次に getaddrinfo(3) で IPv6 が優先されているかを調べます。
- getaddrinfo(3) の例: IPv6 優先
こういう結果が返ってくれば IPv6 で繋るはずです。

```
$ getent ahosts www.kame.net
2001:200:dff:fff1:216:3eff:feb1:44d7 STREAM orange.kame.net
2001:200:dff:fff1:216:3eff:feb1:44d7 DGRAM
2001:200:dff:fff1:216:3eff:feb1:44d7 RAW
203.178.141.194 STREAM
203.178.141.194 DGRAM
203.178.141.194 RAW
```

- getaddrinfo(3) の例: IPv4 のみ

以下のように A と AAAA が設定されているはずなのに A レコードの情報しか返ってこない場合は、プロバイダが “filter-aaaa-on-v4” の設定をしていてフィルタされているのかもしれないので、常用はお勧めませんが Google Public DNS (8.8.8.8, 8.8.4.4) を使えば回避できるかもしれません。

```
$ getent ahosts www.kame.net
203.178.141.194 STREAM orange.kame.net
203.178.141.194 DGRAM
203.178.141.194 RAW
$
```

- getaddrinfo(3) の例: IPv4 優先

以下のような出力の場合は teredo や 6to4 よりも IPv4 が優先されている (RFC 3484) のが原因です。ipv6.google.com のように IPv6 のみのサイトに接続するときだけ teredo や 6to4 の IPv6 接続が使われるようになっていきます。

```
$ getent ahosts www.kame.net
203.178.141.194 STREAM orange.kame.net
203.178.141.194 DGRAM
203.178.141.194 RAW
2001:200:dff:fff1:216:3eff:feb1:44d7 STREAM
2001:200:dff:fff1:216:3eff:feb1:44d7 DGRAM
2001:200:dff:fff1:216:3eff:feb1:44d7 RAW
$
```

RFC になっているようにこの挙動の方が望ましいので、/etc/gai.conf で設定可能ですが、一時的に変更するだけにして試した後は戻すべきです。

Windows は netsh でポリシーテーブルを変更すれば良いそうです*3

<http://www.tokyo6to4.net/index.php/6to4%E3%81%AE%E5%88%A9%E7%94%A8%E6%96%B9%E6%B3%95#.E3.83.9D.E3.83.AA.E3.82.B7.E3.83.BC.E3.83.86.E3.83.BC.E3.83.96.E3.83.AB.E3.81.AE.E7.B7.A8.E9.9B.86.E6.96.B9.E6.B3.95>

- gai.conf の設定

<http://slashdot.jp/~ohhara/journal/519278> を参考にして teredo で接続している場合は/etc/gai.conf に「label 2001:0::/32 1」と設定すると亀が踊りました。試した後は忘れずに設定を戻しておきましょう。

RFC 3484 に関連する設定として ip addrlabel もありますが、今回は変更しなくても大丈夫でした。

デフォルトの設定は以下のようにすべてコメントで書かれていました。

```
#label ::1/128 0
#label ::/0 1
#label 2002::/16 2
#label ::/96 3
#label ::ffff:0:0/96 4
#label fec0::/10 5
#label fc00::/7 6
#label 2001:0::/32 7
```

teredo の場合は以下のように変更しました。

```
label ::1/128 0
label ::/0 1
label 2002::/16 2
label ::/96 3
label ::ffff:0:0/96 4
label fec0::/10 5
label fc00::/7 6
label 2001:0::/32 1
```

*3 以下、リンクはページの都合上折り返していますが一行で

6to4 で接続している場合は同様に 2002::

```
label ::1/128      0
label ::/0         1
label 2002::
```

glibc のリゾルバの設定ファイルになるため、プロセスの起動時のみ読み込まれているようで、設定の反映にはブラウザの再起動が必要でした。

glibc の設定ファイルなので静的リンクされていて glibc を使わないバイナリなどは影響を受けないと思います。

4.14.5 端末は IPv6 対応なのに IPv6 のサイトに繋がらない

WPAD (Web Proxy Auto-Discovery Protocol) で設定していた proxy が IPv6 対応していないサーバで動いていたために繋がらないということがありました。

4.14.6 一部のサイトへのアクセスが遅い・繋がらない

- IPv6 から IPv4 へのフォールバックに時間がかかっている可能性がある
 - IPv6 経由で繋がらない原因を調べる
 - サーバ側で AAAA を登録しているのにサーバが止まっている (A の方では動いている) というのもあるらしい
 - * 根本的な対処はサーバ側でもらうしかない
- Opera で何度かリロードしないと bit.ly など短縮 URL の一部を展開しなくなったということがあったらしい
 - <http://togetter.com/li/146832>
 - IPv6 を無効にしたら直ったという話

4.14.7 何も変えていないのに繋がらなくなった

- 上流の問題かもしれないので確認
 - メンテナンス中ではないか
 - * 「昨日の「さくらの 6rd」接続不良ですが、弊社の IPv6 バックボーン側でメンテナンスが実施されていたようです。」 https://twitter.com/jq6xze_1/status/83335591873351680
 - radvd が止まっていないか
 - * 止まっていれば起動
 - 不正な RA が送信されていないか
 - * 不正な RA を送信しているルータを探して対処
 - * Windows の ICS (インターネット接続の共有) が原因になることがあるらしい
 - * 意図的に不正な RA が送信されていると対処は困難 (IPv4 の ARP spoofing に似た問題になる)
 - * RFC 6105 (IPv6 Router Advertisement Guard) や RFC 3971 (SEcure Neighbor Discovery (SEND)) などで対処
 - 6to4 が廃止されてリレールータ消滅?
 - * すぐにはなさそうですが、将来的には可能性がありそうです
 - teredo サーバ停止?
 - * teredo で最初に接続する teredo サーバが停止していないか
 - * teredo サーバへの接続が firewall などで止められていないか

4.14.8 短いと通信できるのに長いと通信出来ない

- MTU 問題ではないか

4.15 まとめ

- IPv6 接続はクライアントとして試すだけなら簡単でした。
- サーバも openssh や apache2 なら問題は起きにくいようです。
- munin や proxy など問題がなさそうと思っていたところで問題が起きることもあります。
- IPv6 の仕様は更新され続けていて、セキュリティ問題もまだまだこれからのようなので、引き続き最新の情報を追いかけていく必要があります。

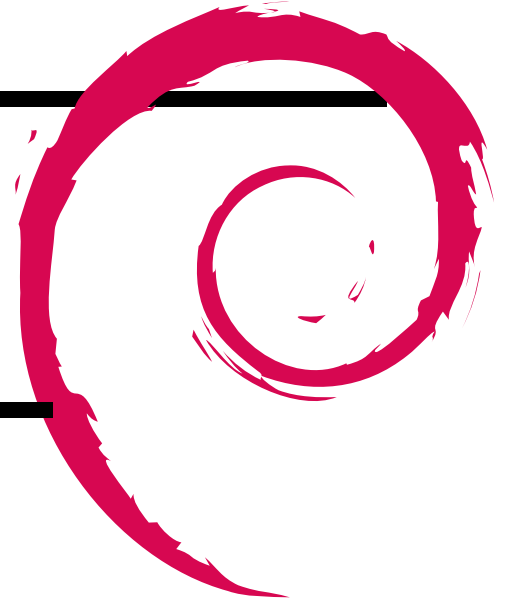
4.16 参考

参考文献

- [1] 今からはじめる IPv6 ~ プロトコル基礎編 ~ <http://www.nic.ad.jp/ja/materials/iw/2010/proceedings/s2/iw2010-s2-01.pdf>
- [2] 絶対わかる！IPv4 枯渇対策 & IPv6 移行超入門 ISBN 978-4-8222-6769-8

5 vcs-buildpackage ～git,svn 編～

佐々木 洋平



5.1 始めに

今日のお題は、パッケージ作成に Git や Subversion を使用するソフトウェアである、git-buildpackage と svn-buildpackage です。自分の抱えている野良パッケージの多くが Ruby 関連だったこともあって、PkgRubyExtras に参加したところ、パッケージ管理を Subversion から Git へ移行するタイミングだった様で、(幸か不幸か) 両 VCS を使用してのパッケージ作成を体験しました*4。

そこで覚えたツールの使い方とか、実際に作業する際のハマリ所とかについて簡単に紹介できたら良いな、とか思っています。とはいえ、実際にはコラボする人々(=チーム)毎に work flow があるので、あまり一般論は言えない訳ですけど。

5.2 バージョン管理?

バージョン管理システム (VCS) についてはほとんど説明しませんが、簡単に。

VCS を使った事のない人に VCS の利点を説明する時、佐々木は良く「良い感じのバックアップ」という言い方をしています。例えば

- 過去の変更履歴を残しておく
- 過去の任意の状態に簡単に戻せる
- 過去にどんな変更を行なったか、を把握しやすくする (ログをきちんと書いておく必要はありますが)

また、複数人で開発を進めている時には、同じファイルに同じ様な変更を加えている場合、最後に保存した人の変更だけが残ってしまい (上書きされてしまい)、それ以外の変更が失われてしまいます。VCS を使っていると、同じファイルへの変更は「衝突」として検出されるので、こういった事態を防ぐ事ができます。

さらに、特定のバージョンに名前をつけておき (tag と言います) そのバージョンへの変更を行ったり、新機能の開発を本番の開発と分離して進めて (本番が main line と呼ばれるのに対して、branch を分ける、branch を切る、と言います)、完成した後で本番の開発に統合したりできます。

Subversion と Git は、それぞれ「集中型 VCS」と「分散型 VCS」の代表です。集中型 VCS では単一のリポジトリ (データ置き場) を開発者全員が参照し作業するのに対して、分散型 VCS では、各人が個別にリポジトリを持ち、各々のデータを適宜同期を取って作業を進めます。最近は分散型 VCS(DVCS と言ったりします) がトレンドですね*5。

*4 自分一人だったら絶対 Git で作業するんですけどね

*5 集中型 VCS としては CVS なんかもありますが、今更 CVS するのは止めた方が良いでしょう。また DVCS としたは Git 以外に darcs, bzt, Mercurial なんかがあります。bzt についてはそのうち山下 尊也さんが語ってくれると思います

5.3 パッケージのバージョン管理?

さて、「パッケージのバージョン管理」って何をしているのでしょうか? Debian パッケージを「バージョン管理」する目的は幾つかありますが、例えば

- 履歴の管理。パッケージの変更点は `debian/changelog` に書かれます (書きます) が、実際にファイル自体を参照できると大変便利です。
- `stable` に含まれたパッケージにタグを付けておく。
 - セキュリティ対応などを、そのタグから派生するブランチで対応する
- 複数人でのパッケージメンテ/チームでのパッケージメンテ作業の環境を容易に構築できる
- `upstream` が VCS を使用している場合の連携が簡単になる (かも)

... でしょうか。

5.4 何を管理するの?

Debian のソースパッケージは、`source format 3.0` では以下のファイル群で構成されます:

`.orig.tar.ext`

`upstream` のソース。複数のソースからなる場合には基本となるソースにこの名前をつける。`ext` は圧縮の拡張子であり、`gz`, `bz2`, `xz` が使用可能。

`.orig-component.tar.ext`

`upstream` が複数のソースから構成される場合のファイル名。`dpkg-source -x` パッケージ名.`dsc` などで展開するとファイルの中身は `component/` 以下の展開される。

`.debian.tar.ext`

`debian/` ディレクトリの中身

`.dsc`

パッケージの情報。上記ファイル群のハッシュなどを記載している。

パッケージを VCS で管理する場合、

1. `.orig.tar.ext`
2. `.debian.tar.ext`

のバージョンを管理することになります。

5.5 どうするの?: `svn-buildpackage` 編

では実際にどうやっているのでしょうか? ここでは GNU Hello を例に、`svn-buildpackage` を使った場合について簡単に述べてみます。なお、佐々木は既に `svn-buildpackage` をあまり使っていないので、Git はまだ良くわからないけれど Subversion ならわかる、という人向けのお話です。

5.5.1 インストール

```
% sudo aptitude install svn-buildpackage
```

5.5.2 パッケージをリポジトリに追加する

一度適当なパッケージを作成してみましょう。ここでは GNU hello のパッケージが作成できたとします。横着したい人は `apt-get source hello` でも良いでしょう。

テストのために簡単なリポジトリを作成します。ここでは `/Work/svn/` 以下にリポジトリを作成します。

```
% svnadmin create ~/Work/svn/
% svn mkdir file:///home/uwabami/Work/svn/trunk -m "create trunk"
% svn mkdir file:///home/uwabami/Work/svn/tags -m "create tags"
% svn mkdir file:///home/uwabami/Work/svn/branches -m "create branches"
% svn ls file:///home/uwabami/Work/svn
branches/
tags/
trunk/
```

次に、作成したパッケージをリポジトリに追加します。追加するコマンドは `svn-inject` です。

```
% svn-inject -l 2 -o -c 0 hello_2.7-1.dsc file:///home/uwabami/Work/svn
```

`svn-inject` のオプションの詳細は `man` を見てもらうとして、ここでは

```
-l
  Layout type.
  1 (default) means package/{trunk,tags,branches,...} scheme,
  2 means the {trunk,tags,branches,...}/package scheme.
-o
  Only keep modified files under SVN control (including the debian/ directory),
  track only parts of upstream branch
-c number
  Checkout nothing (0), trunk directory (1) or everything (2) when
  the work is done.
```

を用いています。単一のパッケージを単一のリポジトリで管理する場合には `-l 1` が良いでしょう。

5.5.3 パッケージのビルド

`svn-inject` が終わったので、リポジトリからファイルを取得してみます。

```
% svn checkout file:///home/uwabami/Work/svn/trunk svn-work
A trunk/hello
A trunk/hello/debian
A trunk/hello/debian/control
A trunk/hello/debian/source
A trunk/hello/debian/source/format
...
```

ここでは `svn-work` にファイルをチェックアウトしました。実際に `svn-work/trunk/hello` に移動して、パッケージを作成してみます。

```
% cd svn-work/trunk/hello
% sudo apt-get build-dep hello
% svn-buildpackage
Complete layout information:
  trunkDir=/home/uwabami/svn-work/trunk/hello
  trunkUrl=file:///home/uwabami/Work/svn/trunk/hello
dpkg-checkbuilddeps
Orig tarball not found (expected ../tarballs/hello_2.7.orig.tar.gz)
mergeWithUpstream mode detected, looking for ../tarballs/hello_2.7.orig.tar.gz
I: mergeWithUpstream property set, looking for upstream source tarball...
E: Could not find the upstream source file! (should be ../tarballs/hello_2.7.orig.tar.gz)
```

... 転びました。 `svn-buildpackage` では、パッケージ作成時の一時ディレクトリ (`build-area`) と upstream のソースの保管場所 (`tarballs`) の存在を仮定しています。これを準備しましょう (ちなみに `svn-buildpackage` は失敗しました時点で、これらのディレクトリが既に作成されています)。

```
% cd ..
% ls
build-area/ hello/ tarballs/
% cd hello
% uscan --download-current-version --destdir=../tarballs/
```

`watch` ファイルがきちんと書かれている/書けていると、`uscan` 一発で良いので楽です。ではもう一度パッケージ

をビルドしてみます。

```
% svn-buildpackage
Complete layout information:
  buildArea=/home/uwabami/svn-work/trunk/build-area
  origDir=/home/uwabami/svn-work/trunk/tarballs
  trunkDir=/home/uwabami/svn-work/trunk/hello
  trunkUrl=file:///home/uwabami/Work/svn/trunk/hello
...
dpkg-deb: './hello_2.7-1_amd64.deb' にパッケージ 'hello' を構築しています。
signfile hello_2.7-1.dsc
gpg: " Santiago Vila <sanvila@debian.org> "をとばします: 秘密鍵が得られません
gpg: [stdin]: clearsign failed: 秘密鍵が得られません

dpkg-genchanges >./hello_2.7-1_amd64.changes
dpkg-genchanges: including full source code in upload
dpkg-source --after-build hello-2.7
dpkg-buildpackage: full upload (original source is included)
dpkg-buildpackage: warning: Failed to sign .dsc and .changes file
Command 'dpkg-buildpackage' failed in '/home/uwabami/svn-work/trunk/build-area/hello-2.7',
how to continue now? [Qri?]: ignore
```

最後の gpg sign で止まっていますので i(ignore) で終わらせましょう。どうやら上手くできたいですね。あとは、通常通り debian/ 以下を更新していき、リリース時 (dput/dupload 時) にタグを付けたり、branch を切つてメンテしたりしてきます。

5.5.4 new upstream release

upstream で新しいパッケージがリリースされた時には svn-upgrade コマンドを使って、新しいソースを登録します。とはいえ、今回やった様に debian ディレクトリのみを管理している場合には、uscan の download 先を見て、適宜更新していただくだけで良いでしょう。

5.5.5 tips?

佐々木は以下のコマンドを alias として登録しています。svn-pbuilder は cowbuilder 呼び出しのための wrapper スクリプトです。

```
if [ -f /usr/bin/svn-buildpackage ]; then
  alias svn-b="svn-buildpackage -rfakeroot -us -uc --svn-ignore --svn-lintian --svn-dont-clean"
  alias svn-bc="svn-buildpackage --svn-builder='svn-pbuilder' --svn-lintian --svn-dont-clean"
  alias svn-bct="svn-buildpackage --svn-builder='svn-pbuilder' --svn-lintian --svn-tag --svn-retag --svn-dont-clean"
  alias svn-bcl="svn-buildpackage --svn-builder='svn-pbuilder-local' --svn-lintian --svn-dont-clean"
fi
```

5.6 どうするの?: git-buildpackage 編

以下では既存のパッケージとして rabbit^{*6} の更新作業を例に、git-buildpackage を使った作業を述べてみます。

5.6.1 インストール

```
% sudo aptitude install git-buildpackage
```

Recommends に pristine-tar と cowbuilder があります。これらもインストールしておくといいでしょう。cowbuilder に関しては、先月の水野さんの資料を参照して下さい。pristine-tar については後述。

5.6.2 パッケージのリポジトリへの追加

Git なので、リポジトリの作成とか面倒な事はありません。既存のパッケージの Git リポジトリ作成には git-import-dsc を使用します。

^{*6} <http://rabbit-shockers.org/>

```
% apt-get source rabbit
% git-import-dsc --pristine-tar rabbit_0.9.2-3.dsc
gbp:info: No git repository found, creating one.
Initialized empty Git repository in /home/uwabami/Downloads/rabbit/.git/
gbp:info: Tag upstream/0.9.2 not found, importing Upstream tarball
/usr/bin/pristine-tar: committed rabbit_0.9.2.orig.tar.gz.delta to branch pristine-tar
gbp:info: Version '0.9.2-3' imported under 'rabbit'
```

この際に `--pristine-tar` オプションをつけることを推奨します。また、これまでのバージョンの `.dsc` ファイルと `.orig.tar.gz` ファイルがある場合には `git-import-dscs` コマンドを使うと良いでしょう。tag を良きにはからしてくれます。

さて、これで `rabbit` という Git リポジトリが作成されました。実際に中を見てみましょう。

```
% cd rabbit
% git branch
* master                <-- debian/ 入りのフルソース
  pristine-tar          <-- orig.tar.{gz,bz2} のバイナリデルタ
  upstream              <-- debian/ 無し (upstream) のソース
% git tag
debian/0.9.2-3
upstream/0.9.2
```

ブランチの意味は上記の通りです。 `git-import-dscs` を使うと、バージョンに応じて tag が沢山並んでいると思います。 `master`,

`pristine-tar` はちょっと特殊です。このブランチには、import される前の `.tar.gz`(もしくは `.bz2`) のバイナリデルタのみがコミットされています。 `git-buildpackage` を実行すると `upstream` ブランチ、もしくは `upstream/バージョン番号タグ` から、元々の `orig.tar.{gz,bz2}` を生成します。 `pristine-tar` ブランチのバイナリデルタが無いと (圧縮率が違ったりして) 元々の `upstream` のソースを正しく生成できなかつたりします。

5.6.3 パッケージのビルド

では、パッケージをビルドしてみましょう。

```
% sudo apt-get build-dep rabbit
% cd rabbit
% git-buildpackage
...
Successfully signed dsc and changes files
```

ちなみに `notify-send` コマンドがあると、 `git-buildpackage` の結果を通知してくれます。

あとは通常通り `debian/` 以下を更新していき、リリース毎にタグをつけたり、適当に `branch` を切って作業していきます。また (Subversion のところでは紹介しませんでした) `git-dch` コマンドによって、Git のコミットログから `debian/changelog` を生成することができます (コミットログの最初の行しか生かされませんので、粒度の小さいコミットが求められます)。

5.6.4 New upstream release

`upstream` で新しいリリースが出た場合には `git-import-orig` で新しい `.orig.tar.{gz,bz2}` を取り込みます。

```
% uscan --download
rabbit: Newer version (0.9.3) available on remote site:
  http://rabbit-shockers.org/download/rabbit-0.9.3.tar.gz
  (local version is 0.9.2)
rabbit: Successfully downloaded updated package rabbit-0.9.3.tar.gz
  and symlinked rabbit_0.9.3.orig.tar.gz to it
% git-import-orig --pristine-tar ../rabbit_0.9.3.orig.tar.gz
What is the upstream version? [0.9.3]
gbp:info: Importing '../rabbit_0.9.3.orig.tar.gz' to branch 'upstream'...
gbp:info: Source package is rabbit
gbp:info: Upstream version is 0.9.3
/usr/bin/pristine-tar: committed rabbit_0.9.3.orig.tar.gz.delta to branch pristine-tar
gbp:info: Merging to 'master'
...
gbp:info: Successfully imported version 0.9.3 of ../rabbit_0.9.3.orig.tar.gz
```

既存の `master` と自動的に `merge` が行なわれますので、適宜修正してきます。

5.6.5 patch-queue ブランチ?

source format 3.0 (quilt) では, upstream への変更点を quilt を用いてパッチで管理します。通常通り quilt を用いてパッチを作成 (もしくは debuild が走ったさいにパッチとして抽出) するのも良いのですが、折角なので git format-patch でパッチを生成する方法について述べてみます。

git-buildpackage には gbp-pq というコマンドが提供されています。pq は patch-queue の略です。先ず、現時点で debian/patches 以下にあるパッチを patch-queue ブランチへ登録します。

```
% quilt pop -a
% gbp-pq import
```

この時点で、master ブランチから patch-queue/master ブランチへ切り代わります。debian/patches 以下にあったパッチがファイル一つ毎に一つのコミットとして登録されます。適宜 rebase するなどして、パッチをマージしたり削除したりしていきます。パッチの修正が終わったら

```
% git checkout master
% gbp-pq export
```

で、patch-queue/master のコミットがそれぞれパッチとして debian/patches 以下に置かれます。あとは

```
% quilt push -a
% git-buildpackage
```

で ok です。この方式の利点は、個々のパッチが追跡しやすくなること、git format-patch の出力結果なので、upstream が Git を用いている場合には upstream に投げ易くなること、でしょうか*7?

ちなみに、git-buildpackage のオプションには --git-debian-branch= がありますので、

```
% git-buildpackage --git-debian-branch=patch-queue/master
```

とすると、パッチが当たった (quilt push -a) 状態の tree を用いてパッケージ作成ができます。

5.6.6 リモトリポジトリとのやりとり

適宜 git clone/push/fetch すれば良いと思いますが

1. pristine-tar, upstream ブランチ、upstream/バージョン番号タグは必ず push する

に気をつけましょう。また、リモトリポジトリから git-buildpackage 用に clone するための gbp-clone コマンドも用意されています。

他にも upstream が Git を使用していると、結構幸せになれます。git remote で、upstream の Git リポジトリの master や、特定の tag とこちらの upstream を紐付けておくと、単一のリポジトリで全て作業を行なえたりします。

5.6.7 Tips?

佐々木は以下を alias に登録しています。

```
if [ -f /usr/bin/git-buildpackage ]; then
  alias git-b="git-buildpackage --git-ignore-new --git-builder='debuild -rfakeroot -i.git -I.git -sa -k891D7E07'"
  alias git-bc="git-buildpackage --git-ignore-new --git-builder='git-pbuilder'"
  alias git-bct="git-buildpackage --git-ignore-new --git-tag --git-builder='git-pbuilder'"
  alias git-bcl="git-buildpackage --git-ignore-new --git-builder='git-pbuilder-local'"
fi
```

... 以下、当日& 後日補足予定...

*7 とはいえ、毎度 quilt pop/push -a するの面倒かしらん

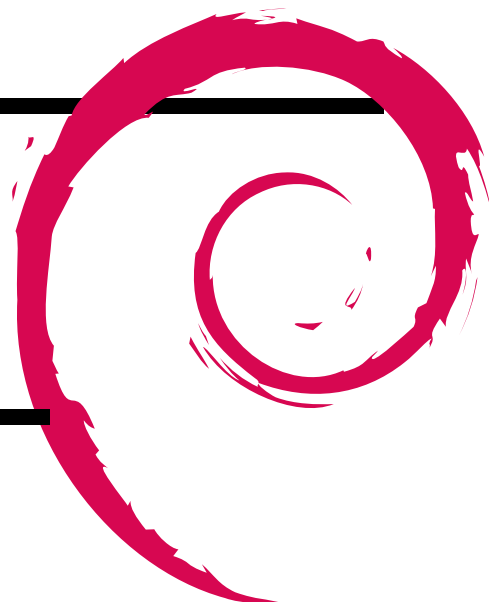
5.7 最後に

以上、簡単に svn-buildpackage, git-buildpackage についてお話ししました。実際にパッケージを作成する際には、共同作業との合意や Team Policy によって、それなりにルールがありますのでそれを参考にしてください。

また、結局 git,svn-buildpackage で multiple upstream を使用する場合の方法論とかはちゃんと調べられませんでした。結構需要ありそうなんですけれどね。bzd-buildpackage はその辺上手く動作するらしいので、山下 尊也さんの発表を期待して待つ事にします。

6 今後の予定

Debian JP



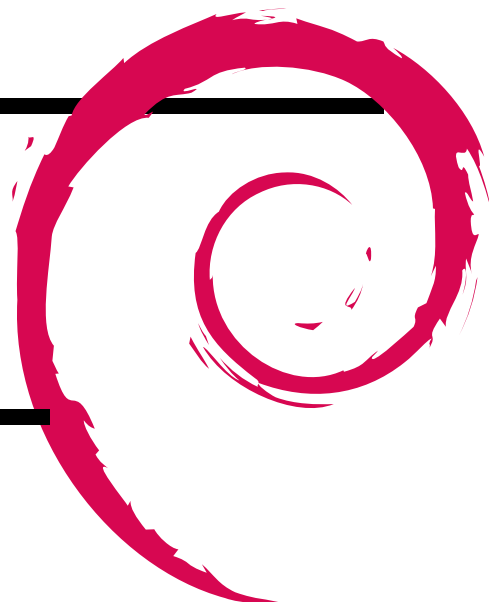
6.1 第 49 回関西 Debian 勉強会 in OSC 2011 KansaiKyoto

7 月の関西 Debian 勉強会は、7 月 16 日に、オープンソースカンファレンス KansaiKyoto においてセッションとして開催します。セッションの時間帯が 10:00 - 10:50 なので、みなさま頑張って早起きしましょう。

今の所、「Debian: Squeeze, Wheezy, and Sid」という玉虫色のお題にしています (つまり... ゲフゲフ)。話者は佐々木の予定です。

ところで、最近、関西 Debian 勉強会の開催日程を日曜ではなく土曜にしたい、という声を聞くことが多いのですが、皆さんはいかがでしょうか。といった内容で一度勉強会予約システムのアンケート機能を使ってみようと考えています。

7 メモ



勉強会

Debian

関西



Debian 勉強会資料

2011年06月26日 初版第1刷発行
関西 Debian 勉強会 (編集・印刷・発行)
