東京エリア デビアン 勉強会

Debian勉強会幹事 上川純一

2008 年 4 月 19 日

1 Introduction

上川 純一

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

Debian 勉強会の目的は下記です。

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

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

以上を目的とした、2008 年アジェンダです：

1. 新年会「気合を入れる」
2. Open Source Conference Tokyo (3/1)
3. データだけのパッケージを作成してみる、ライセンスの考え方 (David Smith)
4. バイナリ一つのパッケージを作成してみる (吉田@板橋)
   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)
   アップストリームの扱い (svn/git/cvs)(岩松 信洋さん)
5. バイナリの分けたパッケージの作成。(前田さん)
   バイナリの分け方の考え方、アップグレードなどの運用とか。
6. パッケージ作成 (dpatch/debhelper で作成するパッケージ)(小林儀匡さん)
   man の書き方 (roff or docbook)(でんさん)
7. パッケージ作成 (kernel patch、kernel module) 、Debconf 発表練習
8. Debconf アルゼンチン、共有ライブラリパッケージ作成
9. Open Source Conference Tokyo/Fall、デーモン系のパッケージの作成、latex、 emacs-lisp、フォントパッケージ
10. パッケージの cross-compile の方法、amd64 上で i386 のパッケージとか、OSC-Fall 報告会、Debconf 報告会
11. 国際化 po-debconf / po 化 / DDTP
12. 忘年会

デビアン勉強会

目次

2 事前課題

上川 純一

今回の事前課題は以下です。

1. 「パッケージをつくってはまったこと・作ってみたいパッケージ」
2. 「あなたはソースコードを管理しています。VCS を使うことでメリットを得ているのですが、使っていない友人にどう説明しますか？『○○○を使ったおかげで○○○になりました。なぜなら … 。だからあなたもつかったほうがよいですよ。』」

この課題に対して提出いただいた内容は以下です。

2.1 Hiroyuki Yamamoto

2.1.1 「パッケージをつくってはまったこと・作ってみたいパッケージ」

- パッケージをつくってはまったこと
  パッケージ自体を作ることにはあまりはまったことはないですが、最近、locale は UTF-8 が主流になってきていますが、仮想ターミナルで動く EUC-JP なプログラムを UTF-8 対応しようとした時、文字コードのバイト数が 2 バイト固定から 2-4 バイトに可変となっているため、コードを大幅にいじらないといけなくなっている場合があります。これからのユーザは UTF-8 でインストールするでしょうし、UTF-8 対応を無視もできなくて、困っています。あと、どうでも良いことですが、dh-make がポリシー 3.7.3 に対応したテンプレを吐くようになるのはいつになるのかな？
- 作ってみたいパッケージ
  某巨大掲示板の専用ブラウザ kita2 をオフィシャルに上げようと思っています。アップストリームの作者には必要なファイルを入れてもらうよう交渉したところ。次期リリース待ち。

2.1.2 「あなたはソースコードを管理しています。VCS を使うことでメリットを得ているのですが、使っていない友人にどう説明しますか？『○○○を使ったおかげで○○○になりました。なぜなら … 。だからあなたもつかったほうがよいですよ。』」

VCS は使っていません。是非、私にお薦めして下さい。

2.2 沖中

2.2.1 「パッケージをつくってはまったこと・作ってみたいパッケージ」

今まで本格的にパッケージを作ったことがないので、はまったことはないです。強いて言うなら、どこから手をつけていいか分かっていないところとか、Makefile が分からないってことでしょうか。既存のパッケージを参考にしようとしたのですが、何をやっているのか分からず、とても難解に見えます。Makefile の記法を覚えるところから始め

ようと思っています。

私の作ってみたいパッケージは、探せば誰かが作っていたりするので、これはぜひ私が!というものはないです … 。更新が頻繁にされて、パッケージかが追いついていないものを自前で作成する時がほとんどです。あとは、自作のソフトくらいでしょうか。

2.3 akedo

2.3.1 「パッケージをつくってはまったこと・作ってみたいパッケージ」

パッケージを作った事は無いので、作ってみたいパッケージと言う事で考えてみました。インターネット公開サーバを個人で管理してたりすると、あるとちょっと安心なのが krfilter です。これは iptables でフィルタするための IP アドレスのリスト (設定スクリプト付き) です。韓国 (kr) だけでなく中国 (cn) や台湾 (tw) もあり、個人的にはこれに .br とか .th も要りそうな気がしてます。本格的なサーバには RBL(DNSBL) の方が良いかも知れません。

2.4 山本　琢

2.4.1 「あなたはソースコードを管理しています。VCS を使うことでメリットを得ているのですが、使っていない友人にどう説明しますか？」

思いつくのは cvs ですが、

- 過去のソースコードを呼び出せる（個人での世代バックアップが不要）
- 変更履歴が参照できる (cvs log)
- 書いたソースとバージョンとの紐付けができる

2.5 やまね

2.5.1 「パッケージをつくってはまったこと」

- ライセンス調整が大変

  開発元にお伺いを立てて変更を依頼したり、物によっては日本語ライセンスのみで英訳が必要だったりします。　ライセンス解釈の問題があったりすることもありますし、non-free と扱われて reject を食らうことも。
- pbuilder で build しないでアップロードしたら依存関係が変

  自分のマシンだと experimental が混じっているので、依存関係の問題が出たことが数回。

2.5.2 「作ってみたいパッケージ」

すでに作りかけのパッケージがいくつも…。

2.6 前田 耕平

2.6.1 「パッケージをつくってはまったこと」

技術的な話ではないのですが、ライセンスのとこで悩みます。GPL を採用していると謳っているソフトウェアでも、下記のような感じで書き換えていないのものがあると、本当に GPL 採用しているのかと困ってしまいます。

```
> one line to give the program's name and an idea of what it does.
> Copyright (C) yyyy name of author
```

パッケージをつくっても、自分で使う為だけにしかやらないのはそういうところが、面倒だからなんだと思います。きっと。

2.7 本庄

2.7.1 パッケージをつくってはまったこと

はまったわけではないですが、debian/control などの debian/以下をどのように書くべきか、いつも悩みます。

2.7.2 作ってみたいパッケージ

HTML::DOMbo という CPAN モジュールが便利なのですが、CPAN モジュールということ、名前がちょっと怪しいということで微妙です。

2.8 堀内寛己

勉強会/懇親会には欠席の予定ですが、課題だけ。

2.8.1 「あなたはソースコードを管理しています。VCS を使うことでメリットを得ているのですが、使っていない友人にどう説明しますか？」

もう語りつくされたことだと思いますが、CVS 以上の VCS なら、「いつどこでバグが入りこんだか突き止めやすい」ということでしょうか。たとえテスト駆動開発をしていたとしても、回帰テストをすり抜けるバグというのもありうるわけで、その点でも、いつまでも重要なメリットだと思います。

2.9 濱野

2.9.1 パッケージをつくってはまったこと・作ってみたいパッケージ

`LD_PRELOAD` を使うようなパッケージを作ろうとした時に、共有ライブラリにバージョンが付いているからダメというような lintian の警告ではまりました。これに限らず、lintian には良く解らない理由で何度も何度も怒られました。ポリシーを理解していないからなので自業自得のような気がします。

2.9.2 「あなたはソースコードを管理しています。VCS を使うことでメリットを得ているのですが、使っていない友人にどう説明しますか？」

今頃になって CVS で管理していたいろいろなモノを git に移行しています。git を使ったおかげでディレクトリの移動が簡単に出来るようになりました。遠慮せずにローカルレポジトリにガシガシコミットすることも出来ます。でも時々、更新されない CVS のキーワード `$Id:$` などを見るたび物悲しくなります。

2.10 日比野 啓

2.10.1 パッケージをつくってはまったこと

あるライブラリをパッケージ化したときに、ちょっとパッチを当てる必要があって修正を行ないました。そのライブラリの Makefile ではコンパイル単位のソースファイルがいくつかのディレクトリに分散して置けるようになっていて、Makefile はそのディレクトリを順番にソースファイルを探しにいくように (GNU make の vpath) なっていたのです。修正前のファイルをたまたま vpath の手前にあるディレクトリに置きっぱなしにしていたためにコンパイルしなおしても修正が反映されない原因がわからなくてハマりました。

2.10.2 作ってみたいパッケージ

OCaml 言語のネイティブコンパイラに動的ライブラリをビルドする機能がマージされたのですが、その機能が入ったコンパイラおよび、ライブラリ一式を Debian 化して使えるようにしたいです。リリース版のバージョンに入るにはもうしばらくかかりそうなので。

2.11 藤沢理聡

2.11.1 VCS を使うことでメリットを得ているが、使っていない友人にどう説明するか

研究室で大きなシステムを組むことになったとき、VCS は欠かせません。実際に作成するシステムのプログラム群の管理はもちろんのこと、ドキュメントなどの関係者全員で共有したいデータの管理にも有効です。特に研究室などでは、メンバーが集まる時間がなかなか合わなかったり、人によってはリモートアクセスで開発等を行ったりするため、データが更新されるタイミングがわかりづらいこともあります。そのような場合に、VCS を用いることで最近変更された箇所などを把握しやすいことは、システム開発を進める上で大きな利点です。

2.12 jitsukata

2.12.1 パッケージをつくってはまったこと

はまったのとは少し違いますが、パッケージを作成するのに複数のスタイルがあることに混乱しました。debhelper、CDBS、dpatch、dbs、quilt、yada の使い分け、関連性がよく分かりません。それぞれメリット・デメリットがあると思いますが、複数のスタイルがあることで、何を使えば良いのか分かりにくいと感じました。

2.13 キタハラ

2.13.1 「パッケージをつくってはまったこと・作ってみたいパッケージ」

debian のパッケージに関しては、今年の３月１日に試みたのが初めてで、講義の速度についていけず、未完に終わったという恥ずかしい経験しかないもので、何を書いてよいのやら … 。

無難に「自作のプログラムをパッケージにしてみたい」と書こうと思ったが、最近プログラムを書いていなかった。（某ＯＳ上ではマクロとかスクリプトとかを山ほど書いては捨てているのですが … ）

3 Debian Trivia Quiz

小林 儀匡

ところで、みなさん Debian 関連の話題においついていますか？Debian 関連の話題はメーリングリストをよんでいると追跡できます。ただよんでいるだけでははりあいがないので、理解度のテストをします。特に一人だけでは意味がわからないところもあるかも知れません。みんなで一緒に読んでみましょう。

今回の出題範囲は `debian-devel-announce@lists.deban.org` に投稿された内容からです。

問題 1. パッケージの品質確認に有用な次のツールのうち、lenny および sid から削除されたものは?

☐ A lintian
☐ B linda
☐ C piuparts

問題 2. init スクリプトの実行順序を依存関係から自動的に決定できるようにすることが lenny のリリースゴールの一つになっているが、2008 年 4 月 16 日現在、/etc/init.d に init スクリプトを持つパッケージの数は?

☐ A 873
☐ B 857
☐ C 98

問題 3. 次のうち、Debian Installer Lenny Beta 1 における改良点でないものは?

☐ A NTP を用いた時刻合わせ
☐ B volatile のサポートの追加
☐ C Mac OS X からのインストーラ起動のサポート

問題 4. 新たな Debian プロジェクトリーダー (DPL) に選ばれた Steve McIntyre が目標の一つに掲げているものは?

☐ A Communications within the project
☐ B Make Debian sexy again
☐ C Welcoming people to the project

問題 5. Debian Weekly News を復活させようというメールを debian-devel-announce に流したのは?

☐ A Martin Schulze
☐ B Joey Hess
☐ C Alexander Schmehl

問題 6. lenny のリリースに関する現況として正しくないものは?

☐ A GCC 4.3 がデフォルトコンパイラとなり、それでコンパイルできないことが RC バグとなった
☐ B lenny のリリースに向けて既に必須パッケージがフリーズされている
☐ C デフォルトの syslog デーモンを rsyslog に変更することが決定した

問題 7. dpkg 1.14.18 で新たにいくつかのソースパッケージ形式がサポートされるようになったが、それらのうち次期デフォルトとされる「3.0 (quilt)」が持っていない機能は?

☐ A ソースパッケージの upstream tarball として複数のファイルを使用できる
☐ B ソースパッケージの diff.gz ファイルを、適切に複数のパッチに分けてくれる
☐ C ソースパッケージに Debian 固有のバイナリファイルを挿入できる

問題 8. 今年の DPL 選挙の投票率は?

☐ A 62.248%

☐ B 53.084%

☐ C 37.302%

問題 9. etch の次期ポイントリリースは「etch and a half」になるとされているが、そこに含まれる予定の Linux カーネルのバージョンは?

☐ A 2.6.18

☐ B 2.6.24

☐ C 2.6.25

問題 10. 次の開発者のうち、新たに FTP マスターに任命されたのは?

☐ A Sam Hocevar

☐ B Joerg Jaspert

☐ C James Troup

問題 11. 次の開発者のうち、Debian Marketing Team のメンバーに任命されていないのは?

☐ A Sam Hocevar

☐ B Andreas Schuldei

☐ C Moritz Muehlenhoff

問題 12. Joerg Jaspert が DAM に任命されたのは Sam Hocevar の任期の切れる何時間前か？

☐ A 24 時間

☐ B 2 時間

☐ C -1 時間

問題 13. Software Design 2008 年 4 月号 155 ページの git-import-dsc の事例にのっているパッケージ名は？

☐ A ecasound

☐ B pbuilder

☐ C Windows Media Player

問題 14. 昨日 Debian 勉強会の参加メンバーで Debian Developer になったのはだれか？

☐ A 岩松

☐ B 小林

☐ C Charles Plessy

4 最近の Debian 関連のミーティング報告

上川 純一

4.1 東京エリア Debian 勉強会 38 回目報告

3 月に第 38 回東京エリア Debian 勉強会を実施しました。今回の参加者は小林さん、あけどさん、Henrich さん、前田さん、山本浩之さん、堀内さん、hisashim さん、市川憲人さん、沖中研心さん、David Smith さん、関根卓さん、gotom さん、野村さん、山根俊昭さん、satoken さん、osamu matsumoto さん、osamu kimura さん、キタハラさん、青木修さん、Ian さん、吉田@板橋さん、関根@ Google さん、jitsukata さん、Noriaki Sato さん、すずきくにおさん、藤崎さん、CCG さん、原田さん、日比野啓さん、でんすけ@相模原さん、濱野さん、奥野由紀さん、シさん、高橋さん、Emmet Hikory、鵜飼文敏さん、上川の 37 人でした。

まず、クイズを今回も実施しました。今回も、debian-devel-announce の内容から出題しました。7 問目までで全員不正解になったので、8 問目で敗者復活したところ、やまねさんが最後まで勝ち残りました。おめでとうございます。

3 月 1 日に開催された Open Source Conference 参加について山根さんが報告しました。

2008 年はテーマとして DEB パッケージの開発・管理に関連した内容にしよう、ということにしていたので今回は最初のテーマとして、データパッケージの話をしました。David Smith が発表しました。CDBS を使った場合に簡単にパッケージが作成できるという流れでの説明でした。

Debian パッケージでのライセンスの取扱いについて紹介しました。ライセンスを調べてパッケージで debian/copyright を作成する場合の一般的な手順を紹介しました。

今回は宴会は「庵 GuRi（あぐり） 5566」にて開催しました。

5 バイナリーつだけのパッケージを作成してみる

吉田　俊輔

5.1 前提/対象者

この記事は Debian パッケージを作った経験が無い人むけに一番わかりやすいと思われる dh_make を使ったパッケージの作成について説明しています。

前提知識としてはある程度 make(Makefile) を理解している必要がありますが、多くはありません。若干はこのドキュメント中で説明します。

5.2 deb パッケージを作る/使う理由・メリット

通常のパッケージシステムのメリットとして tar ボール (*.tar.gz などで圧縮されたアーカイブファイル) を使った場合に比較して、複数マシンへのインストール、環境再現が簡単、便利、パッケージのアンインストール、バージョンアップが容易といった点が挙げられます。

deb パッケージを作るメリットとしては、それらに加え、ソースファイルとパッチファイルの管理が簡単、ビルドに必要な環境（のパッケージ）を整備するのが簡単、等のメリットが挙げられるでしょう。

もちろんパッケージを作りたい理由は人それぞれと思いますが、自分の使いたいアプリ/機能がパッケージになっていないという理由が多いでしょう。

Debian Developer(Debian 開発者:DD) になるためと言う人もいるかもしれません。

5.3 dh_make とは

dh_make コマンドは dh-make パッケージに含まれるコマンドで、パッケージを作る際に一般的な内容のひな形を作ってくれるツールです。

ソフトウェアのソースコードを展開したディレクトリの中で、このコマンドを実行すると、対話的なやりとりを行ったあと、ひな形として debhelper スタイルのディレクトリとファイルを準備します[*1]。

5.4 debhelper スタイル

deb パッケージを作る方法 (スタイル) についてはいくつかあります。今回説明するのは dh_make の debhelper スタイルについてです。そのほかに同様に debhelper スタイルを使用した dpatch,CDBS,dbs,quilt,yada といったスタイルがあるそうです。

その他にもスクリプト言語向け等にもいくつかスタイルがあるようです。debhelper スタイルは debian ではもっと

[*1] CDBS スタイルのひな形も作成可能です。

も一般的でシンプルなスタイルです。現在の debian パッケージで採用しているパッケージの割合がもっとも多いと言われています。基本的に dh_コマンドを rules というファイルからから呼び出す形式となります。

5.5 dh_make での debhelper スタイルのパッケージ作成に最低限必要なファイル

dh_make が作成する、debhelper スタイルのパッケージ作成に最低限必要なファイルとして以下の 2 つのディレクトリと、4つのファイルがあります。

まず、作業に使うディレクトリ名の形式が決まっています。その下に debian ディレクトリが作成され、パッケージングに必要なファイルはここにすべて置かれます。必要なファイルについてはひな形が dh_make で準備されます。

```
softname-version/（ディレクトリ名）
  debian/（Debian ディレクトリの存在=deb パッケージが作成可能）
     control（パッケージ情報：パッケージ名、バージョンの記述）
     changelog（更新履歴）
     copyright（著作権情報の記述）
     rules（rules パッケージの作成方法を記述）
```

5.6 その他のファイル

以下は dh_make が準備するサンプルのファイルです。これらも同様に dh_make で debian ディレクトリの下に準備されます。必須ではありませんので、必要に応じて利用します。

```
debian/
  README.Debian(オリジナルとパッケージ化したときの差分情報）
  conffiles.ex(設定ファイル名のリスト）
  cron.d.ex(cron で実行するファイル名リスト）
  dirs
  docs
  init.d.ex(サービスの起動スクリプト）
  manpage.1.ex, manpage.sgml.ex(man ファイル）
  preinst.ex(インストール前のスクリプト）
  postinst.ex(インストール後のスクリプト）
  prerm.ex(アンインストール前のスクリプト）
  postrm.ex(アンインストール後のスクリプト）
  等
```

5.7 新規作成の流れ

db_make を使ったパッケージの新規作成の流れとしては以下のようになります。

- ソースアーカイブ展開
- テンプレート作成

```
$ dh_make
```

- debian ディレクトリ配下の編集
- changelog 記述

```
$ dch
```

- control ファイル (パッケージ名や依存関係を記述)
- copyright ファイル (著作権情報の記述)
- rules ファイル (ビルド手順の修正、確認、不要なコードの削除)
- ビルド

```
$ dpkg-buildpackage(deb パッケージの作成）
```

5.8 ソースアーカイブ展開

通常、tar ボール等で配布されているファイルを展開し、その展開されたディレクトリ名を修正（リネーム）します。

dh_make でひな形を作るためには packagename-version の形式である必要がありますので、上記の形式になっていない場合は適切にリネームを行ってください。

5.9 dh_make(debian ディレクトリ、テンプレートの作成)

dh_make でのひな形の作成です。

上記で適切な形式にリネームされたディレクトリに cd し、dh_make を実行します。その際、可能ならライセンスの指定を行った方が良いでしょう。後々の手間が省けます。

また、環境変数からメンテナ名とそのメールアドレスを取得しますので、あらかじめ下記の環境変数を設定しておくことをおすすめします。これも後々の手間が省けます。

dh_make を実行すると、パッケージの種類を選択します。今回の例では Single binary (s) を選択します。

その他、許諾条件（ライセンス）の指定 (-c) も可能です。指定できる許諾条件（ライセンス）は gpl、lgpl、artistic、bsd です。

指定できるパッケージ種類は Single binary (s)/Multiple binary (m)/Library (l)/Kernel module (k)/cdbs (b) です。

環境変数は、DEBFULLNAME(メンテナ名)、DEBEMAIL(メールアドレス) で指定します。

5.10 dch(debian/changelog の記述)

次に、debian/changelog の記述の記述について説明します。まず、このファイルはテキストファイルですが、形式が厳密に決まっており、直接エディタで修正するのはおすすめしません。

たとえば、

```
$ dch -i
```

を使えば定型の形式部分は自動的に記入できるのでおすすめです。

次に、debian/changelog の先頭にあるバージョン情報がこれから作成されるパッケージのバージョンとして使用されます。必要なら修正を行ってください。

次がパッケージの状態ですが、通常は unstable で問題ありません。

次は緊急度ですがこれも low で問題ないでしょう。

実際の変更履歴としての内容はアスタリスク (*) の後にコメントとして記述します。その後の定型 dch が作成してくれる内容をそのまま使うのが良いでしょう。

debian/changelog:

```
smp-mgzip (1.2c-1) unstable; urgency=low
 * Initial release (Closes: #nnnn)  <nnnn is the bug number of your ITP>
-- yoshida syunsuke <koedoyoshida@gmail.com>  Sun, 30 Mar 2008 22:21:28 +0900
```

5.11 control の記述

control ファイルにパッケージ名や依存関係を記述します。

ここではソースパッケージとバイナリパッケージで分けて記載します。

Section には main (完全にフリーなソフトウェア)、 non-free (実際の所フリーであるとはいえないソフトウェア)、そして contrib (それ自身はフリーなソフトウェアであるけれども、non-free な ソフトウェアが無ければ使えないも

の) があります。 更に、これらの下には各パッケージをおおまかに分類する 論理的なサブセクションが用意されており、 そこに含まれるパッケージの種類を簡単に説明するような 名前がつけられています。 管理者専用のプログラムのために「admin」、 基本的なツールのために「base」、 プログラマーのためのツールが含まれる「devel」、 文書の「doc」、ライブラリの「libs」、 電子メールの読み書きに使うリーダや 電子メールサーバを構築するためのデーモンは「mail」、ネットワーク関係のアプリケーションやデーモンの「net」、 他のどんな分類にもあてはまらないような X11 用の プログラムは「x11」など、そしてさらに多くのものが 用意されています。デフォルトは main ですので、ここに記載すると main のサブセクションを指定するということになります。

Priority はこのパッケージをインストールすることが ユーザにとってどれくらい重要なものかを示しています。

新規パッケージの場合、優先度「optional」（選択可能） としておけば、通常は問題無いでしょう。

バイナリパッケージについては CPU 等のアーキティクチャに依存する物 (バイナリの実行ファイル等を含む場合) は any, スクリプトやドキュメントなどアーキティクチャに依存しない場合は all を指定します。その他の項目は必要なら修正してください。

debian/control:

```
Source: smp-mgzip(ソースパッケージ名)
Section: unknown(パッケージのセクション)
Priority: extra(パッケージの重要度)
Maintainer: yoshida syunsuke <koedoyoshida@gmail.com>(メンテナーの名前とメールアドレス)
Build-Depends: debhelper (>= 5), autotools-dev(ビルドに必要なパッケージ)
Standards-Version: 3.7.2(テンプレート作成に参照した Debian ポリシー標準のバージョン)

Package: smp-mgzip(バイナリパッケージ名)
Architecture: any(対応アーキティクチャ)
Depends: ${shlibs:Depends}, ${misc:Depends}(パッケージの依存関係)
Description: <insert up to 60 chars description>(パッケージの説明)
 <insert long description, indented with spaces>
```

5.12 copyright の記述

ファイルの著作権と許諾条件（ライセンス）についての記述です。

dh_make 実行時にライセンスを指定しておけばひな形が作成されます。ソフトウェアによってはディレクトリやファイルごとに別のライセンスの場合もありますので、注意してください。debian/copyright

```
This package was debianized by yoshida syunsuke <koedoyoshida@gmail.com> on
Sun, 30 Mar 2008 22:21:28 +0900.

It was downloaded from <fill in http/ftp site>

Upstream Author: <put author(s) name and email here>

Copyright: <put the year(s) of the copyright, and the names of the
           copyright holder(s) here>

License:
 (中略)
On Debian systems, the complete text of the GNU General
Public License can be found in `/usr/share/common-licenses/GPL'.

The Debian packaging is (C) 2008, yoshida syunsuke <koedoyoshida@gmail.com> and
is licensed under the GPL, see above.
```

5.13 rules の記述

debian/rules にはパッケージングを行うための手順を記述します。Makefile 形式の記述方法ですが、パッケージングの支援をする dh_xxx コマンド群 (devhelper パッケージ) を並べるのがほとんどです。dpkg-buildpackage が期待している必須ターゲットは：

```
build: (ビルド実行)
binary: (バイナリパッケージの生成、通常　binary-indep と binary-arch の実行)
binary-indep: (アーキティクチャ独立のバイナリパッケージ作成)
binary-arch:(アーキティクチャ依存のバイナリパッケージ作成)
```

5.14 rules の記述 build ターゲット

dh_make のデフォルトの場合、build ターゲットは build-stamp に依存している = build-stamp ターゲット (make、コンパイル等の実作業) を呼び出します。

同様に build-stamp は config.status(configure の実行) に依存しています。

```
debian/rules
build: build-stamp

build-stamp:  config.status
        dh_testdir

        # Add here commands to compile the package.
        $(MAKE)
        #docbook-to-man debian/smp-mgzip.sgml > smp-mgzip.1

        touch $@
```

5.15 rules の記述 binary ターゲット

```
binary: binary-indep binary-arch

binary-indep: build install

binary-arch: build install
        dh_testdir
        dh_testroot
        dh_installchangelogs ChangeLog
       (中略)
        dh_builddeb
```

5.16 rules の記述 install ターゲット

```
install: build
        dh_testdir
        dh_testroot
        dh_clean -k  --exclude ./mgzip.c.orig
        dh_installdirs

        # Add here commands to install the package into debian/smp-mgzip.
        $(MAKE) prefix=$(CURDIR)/debian/smp-mgzip/usr install
```

5.17 オリジナルの make ファイルの修正

Single binary (s) の場合、Debian のパッケージングツールは$(CURDIR)/debian/(**パッケージ名**)/以下にあるファイルをまとめてパッケージにします。

「make install」実行で$(CURDIR)/debian/(**パッケージ名**)/以下にインストールするようにオリジナルの tarball 等から展開された、Makefile を準備します。

元々の Makefile が提供されていればそれを修正すればよいでしょう。無ければ Makefile を作成してください。詳細な Makefile の作り方については書籍「make 改訂版」を参照してください。

5.18 パッケージ作成

簡易なパッケージ作成のテストとして下記のコマンドを実行するとパッケージ作成が行われます。$ dpkg-buildpackage -us, -uc -rfakeroot 上記のファイルが適切に作成されていれば、以下のファイルが作成されます。

```
xxx-y.y.y_zzzz.deb(バイナリパッケージ)
ソースパッケージ
xxx-y.y.y.dsc(ソース概要:control より生成)
xxx-y.y.y_zzzz.diff.gz(パッケージ化用の差分)
xxx-y.y.y.orig.tar.gz(オリジナルソースファイル)
xxx-y.y.y_zzzz.changes(パッケージ変更点: control+changelog)

凡例：
        xxx:パッケージ名
        y-y-y:バージョン,
        zzzz: Debian アーキテクチャ(i386,amd64,all, 等))
```

上記の例で dpkg-buildpackage に渡しているオプションの意味は以下の物です。

- -us, -uc(gpg サインをしない)
- -rfakeroot(一般ユーザでの作成)

5.19 dpkg-buildpackage と debuild の違い

dpkg-buildpackage とよく似たプログラムに debuild があります。違いとしては以下の点があります。

- dpkg-buildpackage:
  メリット　dh_make で作ったばかりの設定ファイルでも deb を作成可能
  デメリット 一般ユーザで使用するときは-rfakeroot の指定が必要
- debuild:
  メリット　-rfakeroot 指定が省略可能、ルールに従ったチェックの実行を行ってくれる。
  デメリット 各種ルールに従って設定ファイルを書かないとエラーや警告となる作成されるパッケージに関しては両方とも同じ

5.20 ファイルへの署名

これまでの例で、dpkg-buildpackage や debuild で指定した -us -us のオプションはファイルへの電子署名を省略するためのオプションです。

- -us, -uc Do not sign the source package or the .changes file, respectively.

パッケージを自分のみで使用する場合はともかく、外部へ公開することも視野にいれるのであれば、電子署名を行うのが適切でしょう。

debian パッケージに署名するには GnuPG で鍵を用意し、パッケージ作成時に debuild、dpkg-buildpackage を使用するとデフォルトでパッケージに署名が行われます。

5.21 補足：ファイルの公開方法

パッケージ公開用のファイル (Packages.gz,Sources.gz) を作ります。

```
$ apt-ftparchive packages . | gzip -c9 > Packages.gz
$ apt-ftparchive sources . | gzip -c9 > Sources.gz
```

apt-ftparchive コマンドは apt-utils パッケージに入っています。

ローカルで確認

sources.list(aptline) に以下のような記述を追加し apt-get update;apt-get install でインストールできるかを確認します

```
deb file:/usr/src/ ./
```

ネットワークで確認

ローカルで確認できたディレクトリ構成のまま、そっくり http で公開します。sources.list(aptline) も同様に記述します。

```
deb-src http://XXX ./
```

5.22 補足：deb ソースファイルの展開とリビルド

.dsc,.orig.tar.gz,*.diff.gz を入手して展開し、

```
$ dpkg-source -x XXX.dsc
$ cd 作成されたディレクトリ
```

必要なら修正を実施してリビルドします

```
$ debuild（または dpkg-buildpackage）
```

5.23 参考資料

- 入門 Debian パッケージ (ISBN:4-7741-2768-X)
- Debian 新メンテナガイド[*2]
- Debian GNU/Linux スレッドテンプレ　自作パッケージを作りたい[*3]
- make 改訂版 (ISBN:4-900900-60-5)

[*2] http://www.jp.debian.org/doc/manuals/maint-guide/

[*3] http://debian.fam.cx/index.php?AptGet#n8109a54\

6 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

岩松 信洋

6.1 はじめに

最近の Debian Package は、VCS[*4]を使って管理することができるようになっており、実際に利用しているメンテナも多くなりました。VCS と言っても様々ですが、Debian では代表的な VCS である CVS, Subversion や 最近流行りの Git、マイナーな VCS である darcs などを使うことができます。今回は、今一番イケていると思われる Git を使い、Debian Package をバージョン管理することによりどのような利点があるのか、どのようなツールを使ってメンテナンスを行えばいいのか、などについてを説明します。

6.2 なぜパッケージも VCS で管理するのか

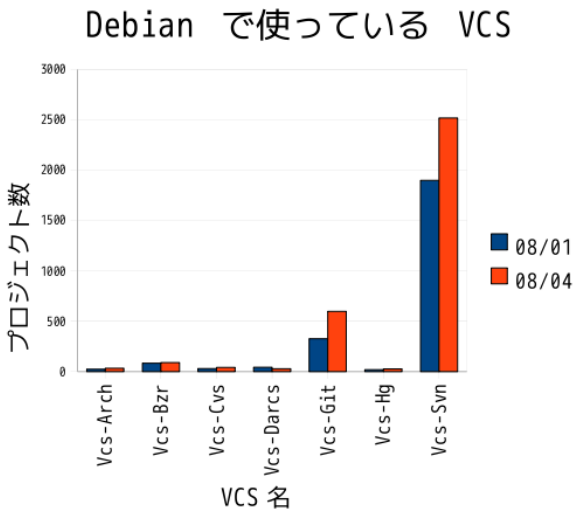
Debian Package を VCS で管理することによる良い点は以下が考えれます。

- 異なるバージョンのパッケージを管理することができる。
  Debian パッケージメンテナは、stable, testing, unstable のソースコードを管理している事が多いです。また、多くの Debian バージョンのソースコードを管理することもあります。VCS を使うことによって、各バージョンの状態を管理しやすくなります。
- チームでの開発体制を容易に構築できる。
  Debian Package はチームでメンテナンスすることが多くなっています。VCS を使うことによって、チームでの開発が容易に行うことができるようになります。
- アップストリームとの連携
  アップストリームのソースコードを追従するために、Debian 側で VCS を使い、連携して開発を行うことができるようになります。また、Debian の独自パッチの管理、トラッキングも容易になります。

6.3 Debian で使用可能な VCS

現在、 Debian で利用可能な VCS は以下の通りです。プロジェクト管理用サーバである Alioth でもこれらの VCS は利用可能になっています。また、最近では Debian Package のタグに 利用している VCS を付けることが可能になっており、各 VCS の利用状況を見ることも可能です。ここ数か月の VCS の利用状況を調べてグラフにしました。Git と Subversion を採用している開発者が増えていることがわかります。

[*4] Version Control System



6.4 ソースパッケージを Git で管理するためのツール git-buildpackage

では、実際に VCS を使って Debian Package を管理するためにはどのようにすればいいのでしょうか。Debian パッケージを Git で管理するためのツールとして、 git-buildpackage があります。これを使うことによって、Git を使ってパッケージのソースコードを管理することができるようになります。インストールはいつものとおり、**apt-get** or **aptitude** を使ってインストールすることが可能です。

```
$sudo apt-get install git-buildpackage
```

6.4.1 git-buildpackage で提供されるコマンド

git-buildpackage で提供されるコマンドは表 1 の 4 つしかありません。これらのコマンドと Git コマンド を使って、パッケージのメンテナンスをすることになります。Git の細かい知識は必要ありませんが、基本的な使い方は知っておく必要があります。

提供されるコマンド	機能
/usr/bin/git-buildpackage	パッケージを作成する
/usr/bin/git-dch	Git のコミットログから Debian Changelog を作成する。
/usr/bin/git-import-dsc	既存の Debian Package を Git にインポートする。
/usr/bin/git-import-orig	アップストリームからリリースされたソースコードを Git にインポートする。

表 1 git-buildpackage で提供されるコマンド

6.5 Git の簡単な使い方

表 2 によくつかう Git コマンドを紹介します。これだけ知っていれば、Git を使った開発を行うことが可能なはずです。

6.6 既にパッケージ化されているものを Git で管理する

Debian パッケージには 2 つの状態があると考えられます。一つは既にパッケージ化されているもの、もうひとつは今から Debian パッケージにしようとしているものです。まずは、既に Debian パッケージになっているものを Git

Git コマンド（一部）	機能
git init	ローカルリポジトリを作成する。
git add	ローカルリポジトリのキャッシュ(index) に管理対象のファイルを追加する。
git commit	ローカルリポジトリに変更を反映する。
git rm	ローカルリポジトリから管理対象ファイルを削除する。
git diff	差分を取得する。
git branch	ブランチを作成する。
git checkout	作成したブランチをチェックアウトする。
git format-patch	パッチを作成する。

表 2　よく使う Git のコマンド

で管理する方法を説明します。

まず、git-import-dsc コマンドを使い、　Git リポジトリに現在のソースコードの状態を取り込みます。コマンドのオプションに、パッケージの dsc ファイル[*5]を指定します。実行すると、パッケージ名でディレクトリが作成され、Git リポジトリが作成されます。また、ブランチとして、master ブランチと、upstream ブランチが作成されます。

Debian 関係のコードは master ブランチ、Upstream のソースコードは upstream ブランチで管理されるようになります[*6]。

```
$ git-import-dsc ../isight-firmware-tools_1.0.2-1.dsc
Upstream version: 1.0.2
Debian version: 1
No git repository found, creating one.
Initialized empty Git repository in .git/
Everything imported under isight-firmware-tools
$ ls
isight-firmware-tools
$ cd isight-firmware-tools
$ git branch
* master
  upstream
```

6.6.1　インポート時のログ

パッケージをインポートしたときに、Git のコミットログに現在のバージョンのコミットログが書き込まれます。また、 Debian Version は Git のタグ機能を使い、タグ名として保存されます。

```
$ git log
commit 9c3669a233afe69d7be2aa8ad1995e6b19c841aa
Author: Nobuhiro Iwamatsu <iwamatsu@nigauri.org>
Date:   Sun Apr 6 21:48:40 2008 +0900

    Imported Debian patch 1.0.2-1
$ git tag
debian/1.0.2-1
upstream/1.0.2
```

6.6.2　ソースコードを変更し、修正を管理する

ソースコードを修正し、Debian Package　で配布する部分を管理するには、今までどおり、dpatch などのパッチ管理システムを使う必要があります。作成したパッチをリポジトリにコミットする時にに、**git add**, **git commit** コマンドを使い、リポジトリに反映させます。

[*5] Debian パッケージの制御ファイル
[*6] ブランチは git branch コマンドで表示可能

```
$ dpatch-edit-patch 05_change_ift-load_install_dir
... いろいろ修正 ...
$ exit
$ vi debian/patches/00list
$ git add debian/patches/05chage_ift-load_install_dir.dpatch
$ git commit -s debian/patches/00list debian/patches/05_chage_ift-load_install_dir.dpatch
/* エディタが起動するので、コミットログを記述 */

Change ift-load install dir.

Signed-off-by: Nobuhiro Iwamatsu <iwamatsu@nigauri.org>

$ git log
commit c9865153ae1949956fdfe3827c0da9b36c2f0ddb
Author: Nobuhiro Iwamatsu <iwamatsu@nigauri.org>
Date:   Sun Apr 6 21:23:20 2008 +0900

    Change ift-load install dir.

    Signed-off-by: Nobuhiro Iwamatsu <iwamatsu@nigauri.org>
```

6.6.3 git-buildpackage を使った Debian パッケージの作成

Debian パッケージを作成するには、 git-buildpackage コマンドを使います。–git-ignore-new　オプションは Git に反映されていない修正を無視するためのオプションです。

```
$ git-buildpackage --git-ignore-new -us -uc
```

6.6.4 パッケージをリリースする

新しい Debian バージョンのパッケージをリリースする場合は、**git-dch** コマンドに–**release** オプションを付けます。実行することにより、エディタが立ち上がり、Git のコミットログから、Debian Changelog が作成されます。Changelog を作成したら、**git-buildpackage** コマンドに –**git-tag** オプションを付けてパッケージを作成します。–**git-tag** を付けると、リポジトリに Debian バージョン用のタグが Debian changelog より作成され、リリース情報が付加されます。

```
$ git-dch --release
$ git-buildpackage --git-ignore-new --git-tag
$ git tag
debian/1.0.2-1
debian/1.0.2-2
upstream/1.0.2
```

6.6.5 新しいバージョンにする

新しいバージョンにするには、git-import-orig コマンドを使い、リリースされた新しいバージョンの Tar ボールを指定します。指定することにより、ファイル名からバージョンを取得し、新たに Upstream 用のタグが作成されます。また、アップストリームのバージョンが上がるため、自動的に Debian Changelog に 次の Debian Package バージョンが追記されます。

```
$ git-import-orig /tmp/isight-firmware-tools-1.2.tar.gz
Upstream version is 1.2.0
Importing '/tmp/isight-firmware-tools-1.2.tar.gz' to branch 'upstream'...
Switched to branch "upstream"
rm 'isight.rules.in'
rm 'po/fr_FR.po'
Created commit f5c85da: Imported Upstream version 1.2.0
 33 files changed, 4434 insertions(+), 1332 deletions(-)

.......<snip>

 src/udev.c                              |  164 +++
 33 files changed, 4434 insertions(+), 1332 deletions(-)
 rename po/{fr_FR.po => fr.po} (66%)
 create mode 100644 src/50-isight-firmware.fdi
 create mode 100644 src/callout.c
 create mode 100644 src/isight-firmware.fdi
 rename isight.rules.in => src/isight.rules.in (100%)
 create mode 100644 src/load.h
 create mode 100644 src/udev.c
Succesfully merged version 1.2 of /home/iwamatsu/Desktop/isight-firmware-tools-1.2.tar.gz into .
$ git branch
debian/1.0.2-1
debian/1.0.2-2
upstream/1.0.2
upstream/1.2
$ cat debian/changelog
isight-firmware-tools (1.2-1) unstable; urgency=low

  * New Upstream Version

 -- Nobuhiro Iwamatsu <iwamatsu@nigauri.org>  Fri, 11 Apr 2008 17:18:23 +0900
```

6.7 新たにパッケージ化する場合

あたらしくにソフトウェアを Debian Package にして、 git-buildpackage で管理する場合は 最初に Git の機能が必要です。まず、ローカル Git リポジトリを作成します。次に作成したリポジトリに移動し、**git-import-orig** コマンドにアップストリームのソースコードを指定し、実行します。ソースコードは、gzip、bzip2 などで圧縮されたものと、展開されたソースコードディレクトリを指定することが可能です。また、実行する際に、**-u** オプションで、アップストリームのバージョンを指定する事が可能です。リポジトリを作成した後は upstream ブランチに移動し、**dh_make** 等を使ってパッケージの雛形作成し、上で説明した流れでメンテナンスを行います。

```
$ mkdir isight-firmware-loader-1.2
$ cd isight-firmware-tools-1.2
$ git init /* ローカル Git リポジトリを作成する */
$ git-import-orig -u 1.2   /tmp/isight-firmware-tools-1.2.tar.gz /* ソースコードをコミット */
Upstream version is 1.2
Initial import of '/tmp/isight-firmware-tools-1.2.tar.gz' ...
Succesfully merged version 1.2 of /tmp/isight-firmware-tools-1.2.tar.gz into .
$ git log
commit 9bf014aee2f834576f8f03d67ab66e8c85726832
Author: Nobuhiro Iwamatsu <iwamatsu@nigauri.org>
Date:   Tue Apr 8 21:42:55 2008 +0900

    Imported Upstream version 1.2
$ git branch
* master
  upstream
$ git tag
upstream/1.2
$ git branch upsteam
$ dh_make
$ git branch master
```

7 アップストリームの VCS と付き合う

岩松 信洋

VCS を使ってアップストリームがソフトウェアの開発を行っている事が多くあります。アップストリームでは、Subversion を使っているが、パッケージメンテナは Git を使ってパッケージを行っている場合があったり、同じ VCS を使っている場合もあります。今回は　Subversion を例にして、お互い VCS を使っている場合、どのように付き合っていくことができるのか説明します。

7.1 アップストリームが Subversion を使っている場合

Subversion で管理されているソースコードを取得したり、Subversion リポジトリへコミットするツールとして、git-svn パッケージがあります。 git-svn を使うことによって、容易に お互いのリポジトリ間を行き来することができるようになります。

7.1.1 Subversion のリポジトリから　Git のリポジトリへ ソースコードを取得する

Subversion のリポジトリから　Git のリポジトリへ ソースコードを取得するには適当なディレクトリを作成し、**git svn** の **clone** オプションを使って行います。　取得した後は、Git の操作で開発を行うことができます。

```
$ mkdir test
$ git svn clone svn://test/trunk test-0.0.1
```

7.2 取得したコードを元に Debian Package を作成する

取得したコードから新しく Debian Package を作成するためには、自分でタグを付ける必要があります。**git-buildpackage** ではタグと Debian changelog から Upstream の情報を取得し、orig.tar.gz 相当のものを作成するため、タグを付ける必要があります。

```
$ git branch
master
$ git branch upstream
$ git checkout upstream
$ git tag upstream/0.0.1
$ dh_make --createorig
$ git branch master
.... Debian Package 用のファイル作成などを行う ....
$ git-buildpackage -us -uc --git-ignore-new
$ debuild clean
$ git add debian
$ git commit -a
$ git-buildpackage -us -uc --git-ignore-new --git-tag
```

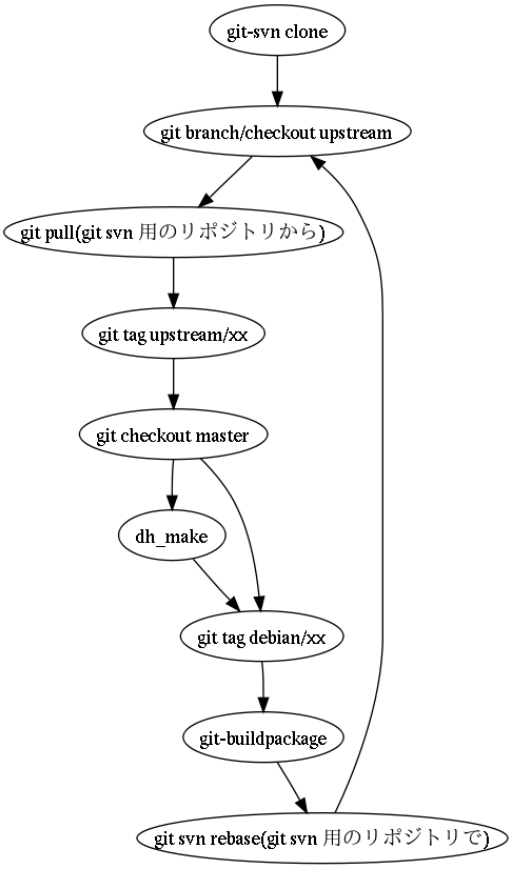
7.2.1 Subversion リポジトリの情報を取得する

Subversion リポジトリの情報を取得するには、 **rebase** オプションを使います。最初から git svn でリポジトリの操作を行っている場合は、rebase を使うことにより、Upstream のコードを パッケージ側に反映させることができ

ます。

```
$ git checkout upstream
$ git svn rebase
```

7.3 すでにあるパッケージと Git リポジトリを元に Debian Package を作成する



git svn で取得した Git リポジトリと 既にある Debian Package を連携させるには操作が少し必要です。まず、**git-import-dsc** で 現在の Debian Package を git-buildpackage で管理できるようにした後、upstream ブランチに git svn で取得したリポジトリから pull をします。pull することにより、コミットログの共有することができます。しかし、Debian Changelog の操作や、**git tag** を使ったタグの操作を手動で行う必要があるのが問題点です。**git-import-org** を使って、tar.gz や ソースコードを指定して、マージすることも可能ですが、Upstream 側のコミットログが取り込まれないため、 Git を使うメリットがあまり無いと私は考えています。このあたりを改善していく事が今後の課題になりそうです。

```
$ git svn clone svn://svn.berlios.de/linux-uvc/linux-uvc/trunk\
 linux-uvc.git
$ git import-dsc  ../../../debian/linux-uvc_0.1.0.svn193-2.dsc
$ cd linux-uvc
$ git branch
* master
  upstream
$ git tag
debian/0.1.0.svn193-2
upstream/0.1.0.svn193
$ git checkout upstream
$ git pull ../linux-uvc.git/
$ git tag upstream/0.1.0.svn201
$ git checkout master
$ dch -v 0.1.0.svn201
$ git-buildpackage -us -uc --git-ignore-new
$ debuild clean
$ git commit -a
$ git-buildpackage -us -uc --git-ignore-new --git-tag
```

8 Nexenta Core Platform を使ってみる

上川 純一

8.1 はじめに

Nexenta[*7]とは Debian GNU/Linux のパッケージングシステムを OpenSolaris[*8]に移植したもののようです。OpenSolaris とは Solaris のオープンソース版のようです。Nexenta Core Platform が 2008 年 2 月にリリースされました。今回はそれを試してみます。

8.2 ダウンロード

http://www.nexenta.org/os/DownloadMirrors からリンクをたどりダウンロードします。今回は http://mirror.stanford.edu/nexenta/isos/nexenta-core-platform_1.0-b82_x86.iso.zip を利用しました。

unzip コマンドで展開すると iso イメージが作成されます。

```
[21:54:42]dancer64:nexenta> unzip nexenta-core-platform_1.0-b82_x86.iso.zip
Archive:  nexenta-core-platform_1.0-b82_x86.iso.zip
  inflating: nexenta-core-platform_1.0-b82_x86.iso
```

8.3 Qemu 環境でのインストール

まず、qemu 用のディスクイメージを作成します。

```
$ qemu-img create -f qcow2 nexenta.cow 3GB
Formatting 'nexenta.cow', fmt=qcow2, size=3145728 kB
```

qemu を起動します。[*9]

```
$ qemu-system-x86_64 -hda nexenta.cow \
 -cdrom  nexenta-core-platform_1.0-b82_x86.iso \
 -boot d \
 -m 512
```

メニューを選択し順番にインストール作業を進めます。

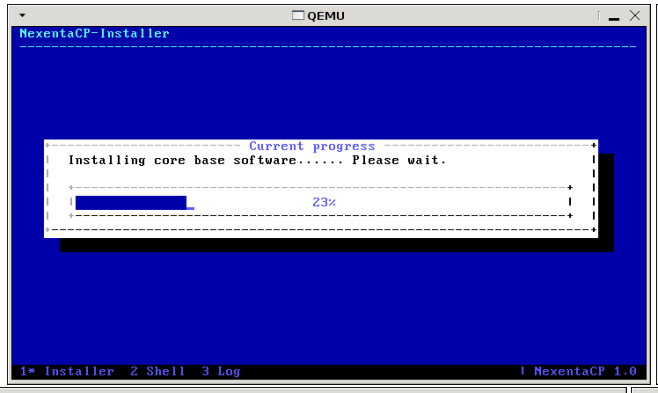
[*7] http://www.nexenta.org/os

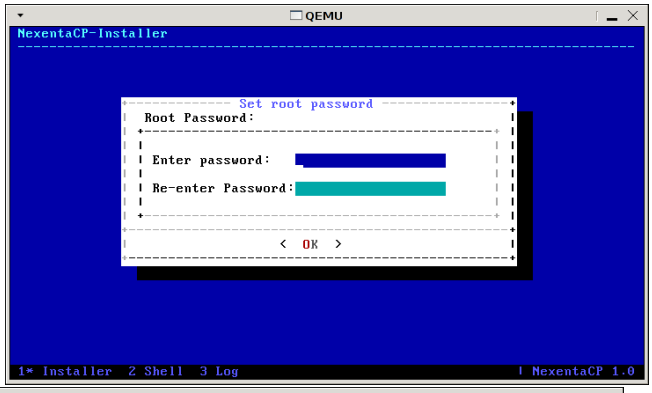
[*8] http://www.opensolaris.org/os/

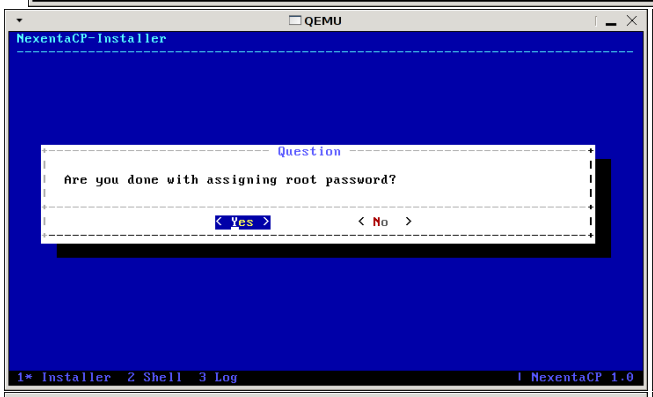
[*9] 時間の設定についてはただしい組み合わせを見つけられませんでした。BIOS 時間をローカルタイムとして扱うわけでもなく、UTC として扱うわけでもないように見えます。正しい設定をご存知でしたらご一報ください。

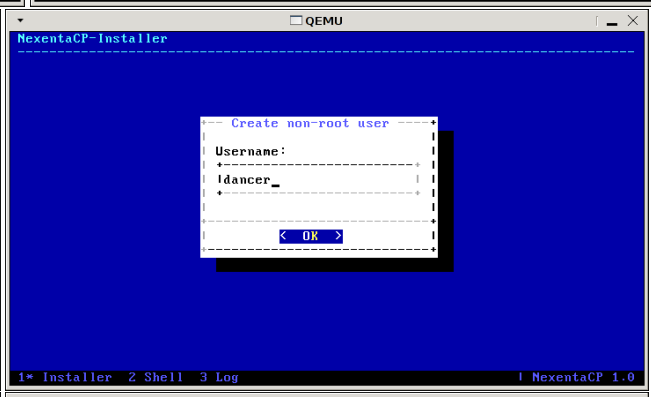
インストールが開始したらしばし待ちます。[*10]

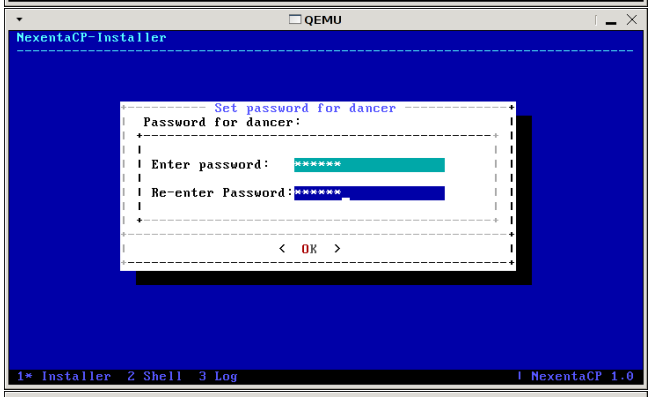
[*10] AMD Athlon 64 3500+ のマシン (64bit) で kqemu を利用して 2 時間かかりました

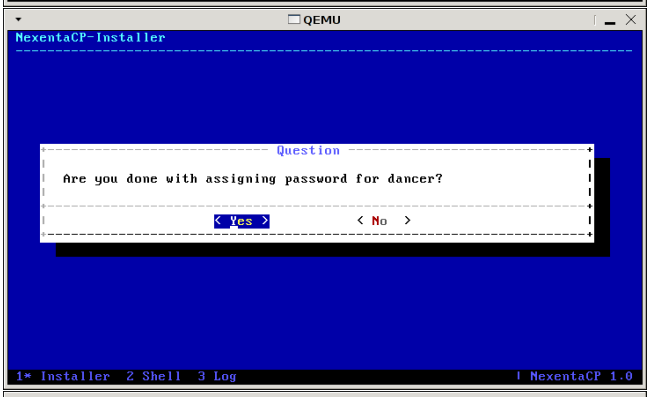

QEMU
NexentaCP-Installer
Current progress
Installing core base software..... Please wait.
23%
1* Installer 2 Shell 3 Log
NexentaCP 1.0


QEMU
NexentaCP-Installer
Set root password
Root Password:
Enter password:
Re-enter Password:
< OK >
1* Installer 2 Shell 3 Log
NexentaCP 1.0

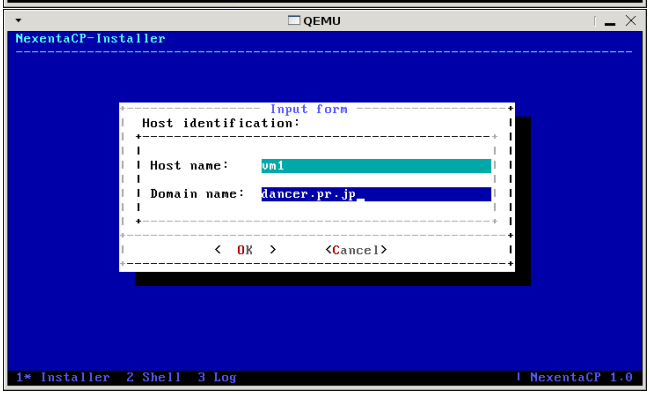

QEMU
NexentaCP-Installer
Question
Are you done with assigning root password?
< Yes >
< No >
1* Installer 2 Shell 3 Log
NexentaCP 1.0

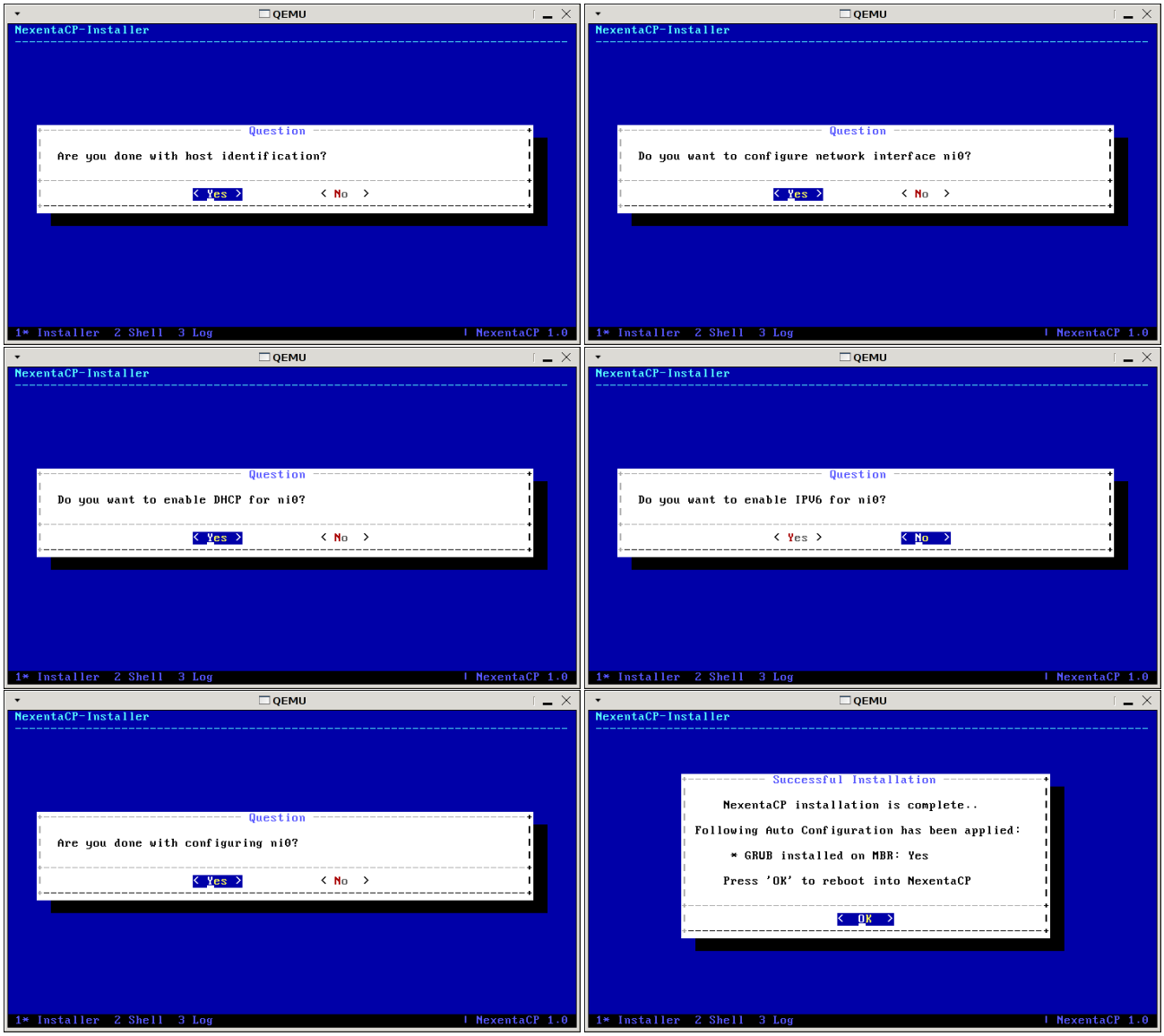

QEMU
NexentaCP-Installer
Create non-root user
Username:
!dancer
< OK >
1* Installer 2 Shell 3 Log
NexentaCP 1.0


QEMU
NexentaCP-Installer
Set password for dancer
Password for dancer:
Enter password: ******
Re-enter Password: ******
< OK >
1* Installer 2 Shell 3 Log
NexentaCP 1.0


QEMU
NexentaCP-Installer
Question
Are you done with assigning password for dancer?
< Yes >
< No >
1* Installer 2 Shell 3 Log
NexentaCP 1.0


QEMU
NexentaCP-Installer
Question
Do you want to commit the changes (username dancer)?
< Yes >
< No >
1* Installer 2 Shell 3 Log
NexentaCP 1.0


QEMU
NexentaCP-Installer
Input form
Host identification:
Host name: vm1
Domain name: dancer.pr.jp
< OK > <Cancel>
1* Installer 2 Shell 3 Log
NexentaCP 1.0

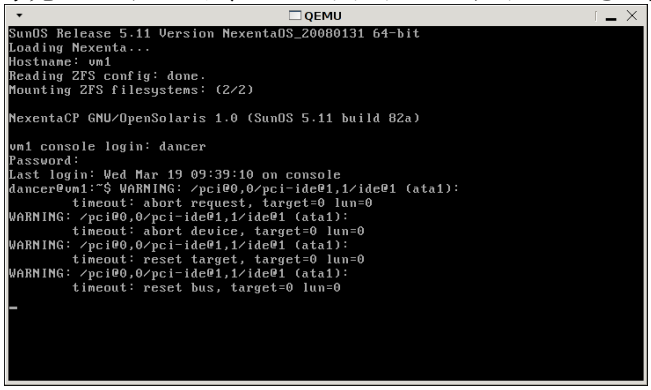


8.4 実行してみる

それでは、インストールした OS を起動してみましょう。メモリは 512MB 程度割り当てないと起動しないようです。

```
$ qemu-system-x86_64 -hda nexenta.cow \
 -m 512 -kernel-kqemu \
 -redir tcp:2222::22
```

手元のシステムでは、コマンドラインでログインができる状態まで 3 分程度かかりました。



通常のコンソールのログイン画面が立ち上がり、ログイン可能になります。

デフォルトで sshd [*11] などもインストールされているようです。netstat で確認すると listen しているようですので、ssh でログインして作業することが可能なようです。ここでは、 qemu で -redir オプションを使い、 SSH の 22 番ポートをホスト OS の 2222 ポートからアクセスできるようにしています。

-nographic -serial stdio オプションも追加してヘッドレスで稼働させることも可能です。ただし、デフォルトではシリアルコンソールには何も出力されないようです。

ネットワークデバイスは ni0 になっているようです。IP アドレスは qemu の機能で DHCP で提供されているアドレス（10.0.2.15）になっています。

8.5 Debian との互換性

初期インストールのパッケージ数が非常に多いです。OpenSolaris のデフォルトインストールに相当するものをそのままパッケージにしているのでしょうか？SUNW ではじまるパッケージ名などがそれっぽさを物語ります。

```
dancer@vm1:~$ dpkg -l | wc -l
464
dancer@vm1:~$ dpkg -l
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase=bad)
||/ Name           Version        Description
+++-==============-==============-============================================
ii  adduser        3.80nexenta3   Add and remove users and groups
ii  alien          8.73.4         install non-native packages with dpkg
ii  apt            0.6.46.4nexent Advanced front-end for dpkg
ii  apt-utils      0.6.46.4nexent APT utility programs
ii  aptitude       0.4.4-1nexenta terminal-based apt frontend
[中略]
ii  sunw1394       5.11.82-1      Sun IEEE1394 Framework
ii  sunwaac        5.11.82-1      Adaptec AdvanceRaid Controller SCSI HBA Driv
ii  sunwad810      5.11.82-1      SUNW W1100z & W2100z Audio Drivers
ii  sunwadixp      5.11.82-1      SUNW Audio Driver for ATI IXP
ii  sunwadpu320    5.11.82-1      Adaptec Ultra320 Driver
ii  sunwafe        5.11.82-1      ADMtek Ethernet Driver
ii  sunwagp        5.11.82-1      AGP GART Driver
ii  sunwahci       5.11.82-1      Advanced Host Controller Interface (AHCI) SA
ii  sunwamd8111s   5.11.82-1      AMD8111 FAST Ethernet Network Adapter Driver
ii  sunwamr        5.11.82-1      LSI MegaRAID SCSI HBA Driver
[中略]
ii  sunwsshcu      5.11.82-1      SSH Common, (Usr)
ii  sunwsshdr      5.11.82-1      SSH Server, (Root)
ii  sunwsshdu      5.11.82-1      SSH Server, (Usr)
ii  sunwsshr       5.11.82-1      SSH Client and utilities, (Root)
ii  sunwsshu       5.11.82-1      SSH Client and utilities, (Usr)
ii  sunwtavor      5.11.82-1      Sun Tavor HCA driver
[略]
```

8.6 ないパッケージをビルドしてみた

/etc/hosts をいじってみようとしたら、シンボリックリンクになってました。しかも root 権限でも書き込み不可になっています。

```
root@vm1:/export/home/dancer# ls -l /etc/hosts
lrwxrwxrwx 1 root root 12 Mar 19 08:15 /etc/hosts -> ./inet/hosts
root@vm1:/export/home/dancer# ls -l /etc/inet/hosts -l
-r--r--r-- 1 root sys 1078 Apr  6 10:28 /etc/inet/hosts
```

気をとりなおして、 /etc/apt/sources.list に適当にソースを追加します。

```
deb-src http://cdn.debian.or.jp/debian unstable main contrib non-free
```

apt-listbugs が存在しないみたいなので、ビルドしてみようと思います。

まず、なぜだか racc, rdtool が足りないのでビルドしてインストールします。無事に apt-listbugs 自体もビルドできたので、意気揚々とインストールしてみます。

[*11] Debian のパッケージとは異なり、SUNWsshdu, SUNWsshdr などのパッケージになっています。見た感じは alien で作成されているパッケージのようです。

```
root@vm1:/tmp/apt-listbugs-0.0.88# debi
Selecting previously deselected package apt-listbugs.
(Reading database ... 27441 files and directories currently installed.)
Unpacking apt-listbugs (from apt-listbugs_0.0.88_all.deb) ...
dpkg: dependency problems prevent configuration of apt-listbugs:
 apt-listbugs depends on ruby (>= 1.8); however:
  Package ruby is not installed.
 apt-listbugs depends on libruby1.8 (>= 1.8.5); however:
  Version of libruby1.8 on system is 1.8.4-1nexenta1.3.
 apt-listbugs depends on libdpkg-ruby1.8 (>= 0.3.2); however:
  Package libdpkg-ruby1.8 is not installed.
 apt-listbugs depends on libgettext-ruby1.8; however:
  Package libgettext-ruby1.8 is not installed.
 apt-listbugs depends on libxml-parser-ruby1.8; however:
  Package libxml-parser-ruby1.8 is not installed.
 apt-listbugs depends on libhttp-access2-ruby1.8 (>= 2.0.6); however:
  Package libhttp-access2-ruby1.8 is not installed.
dpkg: error processing apt-listbugs (--install):
 dependency problems - leaving unconfigured
Errors were encountered while processing:
 apt-listbugs
debi: debpkg -i failed
```

残念、いくつかエラーがあるようです。ruby 自体が古いというエラーや、いくつかのライブラリが足りないというエラーがでています。

解決できそうな問題も、解決できなさそうな問題もあります。先は長いか？

8.7 まとめ

今回は Nexenta Core Platform をとりあえず仮想マシンにインストールしてみました。いろいろと Debian GNU/Linux と違う部分があったので、今後どのように開発がすすむのか、気になります。

Debian 勉強会資料

2008 年 4 月 19 日　　初版第 1 刷発行
東京エリア Debian 勉強会 （編集・印刷・発行）