東京エリア デビアン 勉強会

Debian勉強会幹事 上川純一

2008 年 9 月 20 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)
   OSC 2008 Hokkaido
7. パッケージ作成 (kernel patch、kernel module)(岩松 信洋) Debconf 発表練習（上川さん）
8. Debconf アルゼンチン、Debian 温泉、コミックマーケット 74
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

目次

2 事前課題

岩松 信洋

今回の事前課題は

1. Debian で気になった翻訳されていないもの
2. あなたの考えている Debian パッケージメンテナの想像図

というものでした。その課題に対して下記の内容を提出いただきました。

2.1 前田さん

2.1.1 Debian 気になった翻訳されていないもの

とりあえずよく使いそうな man マニュアルでどのくらい翻訳されていないのかをざっと調べてみました。

- 無線 LAN 関係の man マニュアル
  - wireless-tools パッケージの man マニュアル。
  - wpa_supplicant(8), wpa_supplicant.conf(5)
  - wpa_gui(8)
- セキュリティ関連ツールの man マニュアル。
  - gpg(1)
  - openssl(1ssl) (ところでこれ、何故 OPENSSL(1SSL) 何でしょうかね？)
  - tripwire(8), twadmin(8)
  - ssh(1), sshd(8), sshd_config(5), ssh_config(5)
  - bastille(1m)
  - chkrootkit(1)
- システム管理系の man マニュアル。
  - syslog-ng(8)
  - inetd(8) (個人的には inetd はもう使っていないですが、デフォルトでは入ってますよね)
  - interfaces(5)
  - logrotate(8)
  - gshadow(5)
  - pam(7)
  - udev(7)
  - vim(1)

面倒になったので途中で調べるの止めました。有線 LAN に比べると設定が面倒なので日本語になっていないと一般の人には余計に敷居が上がるだろうと思います。ただ、無線 LAN とかセキュリティ関連 (特に OpenSSH とか) は、

Upstream の更新頻度＆一度の変更での差異が大きいのも原因でしょうか。やることいっぱいありますね…。

2.2 平澤さん

2.2.1 私の考える Debian パッケージメンテナの想像図

ほんとに想像でしかかけないのですが..... お題を 3 つの単語にわけてみて、想像をふくらましてみます。

まず、“Debian” について。安定性を重視したディストリビューションと聞いている。

きっと、Debian の裏方（メンテナー？）さんがライブラリなどの依存関係なんかを細かく確認をしながらリリースするんだろうなぁ、とか想像する。

次、“パッケージ”. 個別のプログラムを xxx.deb ファイルにまとめられている状態を指す？それとも “current”、“stable”,“testing” と呼ばれている、カテゴリー分けされたプログラムの塊のことを指すのかしら？ よくわからない。

次、 “メンテナ”. 訳するとメンテナンスをする人想像が想像をよび、すでに妄想の状態かも。

いい仕事をする “仕事人” のことを指すのでしょうか？仕事の一部は定型化されていると想像するので、ルーチンワークな部分をやっつけるツールってのがたくさんあるんだろうな？とか想像。

2.3 やまねさん

2.3.1 Debian で気になった翻訳されていないものを教えてください

gksu, update-manager (すいません、まだやってませんでした…)

2.4 山本さん

2.4.1 Debian で気になった翻訳されていないものを教えてください

`http://www.debian.org` の文書は大体翻訳されてきたと思うけど、最近は `http://wiki.debian.org` で盛んに文書が出されたり、編集されたりしてるよね。

特に lenny でリリース予定（だっけ？）の armel のページなんかは `http://www.debian.org/ports` から wiki へのリンクぐらいしかない。当然、翻訳なんかもないが、そろそろなんとかねにゃいかん気もする。`http://wiki.debian.org` で気になるのは、まだまだあるんじゃないかな？

2.4.2 あなたの考えている Debian パッケージメンテナの想像図

きっと、スーパーなハカーに違いない！コーラとピザを片手に、コードを書いているんだ！

2.5 つじかたさん

2.5.1 Debian で気になった翻訳されていないものを教えてください

CDBS のドキュメント。以前、CDBS を勉強しようと思ったのですが、日本語ドキュメントが見つからず挫折しました。今、「入門 Debian パッケージ」読んで勉強してます。

2.6 吉田@板橋さん

2.6.1 Debian パッケージメンテナの想像図

編集者の都合により削除させていただきました。

2.6.2 Debian で気になった翻訳されていないものを教えてください

lintian のメッセージ、apt の man,gpg の man。lintian のメッセージの対処方法 (黙らせ方) がぐぐっても、翻訳サイトに掛けても良くわからない。日本語にすればわかるのか？といえばそれも微妙だが。aptitude の man は日

本語化されているのに apt の man は日本語化されていないぐ (以下略) しても太古の ML の断片が出てくるだけ... gpg(GnuPG) の日本語 man... ぐ (略、現状地球上になさげ。訳してみようと思ったが、量が多すぎて挫折。

2.7 たかはし あきゆき さん

2.7.1 Debian パッケージメンテナの想像図

俺はプログラムなんて書けないけど」が口癖の黒魔術士

2.7.2 Debian で気になった翻訳されていないものを教えてください

lenny のアップデート・マネージャが気になります。

2.8 あけどさん

2.8.1 Debian で気になった翻訳されていないものを教えてください

Debian というか Upstream になるかと思うのですが、ruby や perl のマニュアルページは未だ日本語版が無いようです。マニュアルページでは他にも日本語版の無い物が色々とありますね。それと、翻訳とはちょっと違うのですが `http://manpages.debian.net/` で提供されている日本語版のページで文字化けが気になる所です。Debian 関係では、Debian Wiki の日本語化の残っている所がまだまだありますし、Web ドキュメントの Debian Developer's Reference の日本語ページが去年から手を着けたまま未完了なのが気になっています。(査読をお手伝い途中のままですみません。)

2.8.2 あなたの考えている Debian パッケージメンテナの想像図

具体的な人物が思いつくので想像というよりそのものって感じでしょうか。なので、詳細については控えさせて頂きたく存じます。

2.9 hidewon さん

2.9.1 あなたの考えている Debian パッケージメンテナの想像図

いつもパソコンの画面の前に座り、ジャンクフードを食しながら、あらゆる UNIX コマンドをあやつり、それらの結果に一喜一憂をし、液晶画面をみながら明日のソフトウェアを創造しつづけるスーパーハッカー。とてもすごいひとたち。

パッケージの依存を解決したり、バグが発見されればそれらを瞬時に解決しなければならないのでしょうか。英語のパッケージである場合、それらは英語で作者に問い合わせるなどの折衝役もしなければならないのでしょうか。パッケージが書かれている言語がじぶんのしらないもので書かれている場合、それらを自分でなんとかしなければならないのでしょうか。あるいは修正できるスキルがなければメンテナンスをしてはいけないのでしょうか。プログラミングの経験は必須なのでしょうか。いったいどんな作業がおこなわれているのか知りたいです。ときどきソフトウェアの不具合などがあるとき、パッケージを展開すると知らないひとのなまえとメールアドレスがありました。あれがメンテナでしょうか。そうだとすればあれがぼくと彼らをつなぐ唯一の接点でした。ここへ連絡をとってしまっていいものだろうか？となやむことがよくあります。メンテナというひとがどんなひとでどんなことをしているひとなのかということがわからないので、連絡をとるのが怖かったのです。

メンテナはどんなひとたちでしょうか。

2.10 森田 尚さん

2.10.1 Debian で気になった翻訳されていないもの

自分が知らないだけかもしれませんが、

- jargon - the definitive compendium of hacker slang
- fortunes - Data files containing fortune cookies

とか。

2.11 キタハラ さん

2.11.1 Debian で気になった翻訳されていないものを教えてください

Etch のＧＵＩインストールをした時に説明文の中で日本語になってい無い所があったのを思い出したが、どこだったが思い出せない。　当時のメモを見ると **uswsusp** の説明が英語で迷った」とあるのでこれだと思うが、これを見てもまだ思い出せない。　最近、いろんな事がありすぎたせいか、歳のせいか？

2.12 日比野 さん

2.12.1 Debian で気になった翻訳されていないものを教えてください

私の場合のお約束ということで関数型言語系を挙げておきます。Developing Applications With Objective Caml という原著がフランス語の OCaml の本の英訳が ocaml-book-en というパッケージにあります。その英訳がさらに日本語になると嬉しいなあとか。途中までは `http://d.hatena.ne.jp/sumii/20060408/1144492539` に有ったりしますが。

2.13 岩松

2.13.1 Debian で気になった翻訳されていないものを教えてください

Git, Xfce 全般、awesome

2.13.2 あなたの考えている Debian パッケージメンテナの想像図

ツンデレ。

3 Debian Trivia Quiz

岩松 信洋

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

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

3.1 debian-devel-announce

`debian-devel-announce@lists.deban.org` への投稿内容からです。

問題 1. 今年も Debconf が開催されました。どこで開催されたでしょうか。

☐ A 中国
☐ B アルゼンチン
☐ C スペイン

問題 2. ギブアップ宣言をした Debian サブプロジェクト/チームは何でしょうか

☐ A The Debian Live project
☐ B The Debian EEEPC team
☐ C The Debian Jr. project

問題 3. Lenny frozen が宣言されたのはいつでしょうか？

☐ A 2008/07/26
☐ B 2008/07/27
☐ C 2008/07/28

問題 4. Andreas Schuldei が立ち上げた新しいチームは何でしょうか。

☐ A Debian マーケティング チーム
☐ B Debian Dream チーム
☐ C Debian Chrome チーム

問題 5. Debian GNU/Linux 4.0 のアップデート版が出ましたが、何と呼ばれているでしょうか。

☐ A etch and lenny
☐ B etch with you
☐ C etch and a half

問題 6. リリースに向けての作業が佳境に入っています。このような作業の中、lenny の次のバージョンとなるリリースのコードネームが決まりました。何でしょうか。

☐ A ３つ目エイリアン squeeze
☐ B 言葉遊びのオモチャ spell
☐ C 重量挙げ選手のアクションフィギュア rocky

4 最近の Debian 関連のミーティング報告

岩松 信洋

4.1 東京エリア Debian 勉強会 43 回目報告

8 月の第 43 回東京エリア Debian 勉強会を実施しました。勉強会として、今回の Debconf 開催地である アルゼンチンから IRC を使って行われました。今回の参加者は Debconf からは 上川さん、日本からは 武藤さん、やまねさん、山本さん、小室さん、本庄さん、藤沢さん、前田さん、あけどさん、岩松が参加しました。上川さんを Hub にして、Debian 関係に関しての質問などが行われました。詳細は、第 43 回 Debian 勉強会資料としてまとめましたので、そちらを参照してください。

4.2 Debian 温泉報告

2008 年 8 月 16 日は Debian 15 周年でした。おめでとう！ Debian 勉強会では、15 周年を祝うために有志で Debian 温泉と題した 合宿を行いました。今回は、Debian 勉強会に参加された方から希望者を募って実験的に合宿を行う方式を取りました。参加者はやまねさん、あけどさん、前田さん、小林儀匡さん、山本 浩之さん、日比野 啓さん、鈴木さん、岩松の 8 人でした当初は伊豆の伊東温泉で行う予定でしたが、予定していた場所を予約できず、急遽、草津温泉に変更なりました。伊東温泉を楽しみにしていた方、申し訳ございませんでした。詳細は、第 43 回 Debian 勉強会資料としてまとめましたので、そちらを参照してください。

4.3 コミックマーケット 74

今年の夏もコミックマーケット 74 で Debian 勉強会資料を販売しました。今回からページ数が大幅に増え、120 ページ。今まで行ってきた製本方法では冊子にできないため、くるみ製本にして販売しました。販売数は 50 冊で完売することができました。委託販売していただいた USAGI 補完計画さん、編集に協力していただいた方、どうもありがとうございました。ちなみに 166 円の赤字でした。

4.4 第 90 回 カーネル読書会：Debian パッケージ入門の入門

急遽 YLUG が主催する カーネル読書会 で やまねさんが **Debian パッケージ入門の入門**ということで発表しました。Debian 関係の方から Debian 勉強会に参加された事のある方、パッケージに興味があって参加された方など多方面から参加者として集まり、やまねさんにつっこみを入れていました。いろいろ課題の残る発表でしたが、無事終えることができました。この発表で興味を持ってくれた方が増えて、Debian/Ubuntu 界隈が盛り上がればいいなと思っています。やまねさんの次回発表（反省会）に期待したいと思います。

5 Po4a でドキュメント翻訳の保守を楽にしよう

小林 儀匡

翻訳の保守は新規翻訳よりも手間のかかる作業になりがちです。しかし、これを怠ると原文との乖離は大きくなり、折角の翻訳も価値が下がっていきます。Debian で開発されている Po4a は、プレーンテキスト、XML、HTML、LaTeX、nroff (man) など様々な形式のドキュメント翻訳を PO という形式で管理し、保守性を上げるツールです。今回はこの Po4a の使い方と、新たなドキュメント形式を PO で管理するために知っておくべき内部構造について説明します。

5.1 はじめに ～地味で地道なドキュメント翻訳保守作業～

長期的に見れば、ドキュメント翻訳者の作業で最も大変で最も大切なのは、ドキュメントの翻訳作業ではなく翻訳の**保守**作業でしょう。何週間、何ヶ月もの作業の末、1 万行を超える長いドキュメントの翻訳作業を終えたとしても、原文が変わり続ける限り、保守をしなければ翻訳の価値は下がっていきます。

保守は地味で地道な作業です。原文に対しては、次のような変更が発生します。

- 訳文には影響のない typo の修正
- 記述の追加・変更・削除 (修正を含む)
- パラグラフの位置の変更
- マークアップの変更 (マークアップ言語で書かれたドキュメントの場合)

同時に、次のような非本質的な変更も頻繁に発生します。

- パラグラフのインデントの変更
- パラグラフの改行位置の変更 (主に語句の挿入・変更・削除に起因する)

こういった様々な種類の変更は大抵は混ざっています。ドキュメントの翻訳者は通常、`diff` で差分を眺めて変更を把握し、それを翻訳に反映させます。差分を読んで変更を把握する能力によって、翻訳に反映させる作業が楽にも大変にもなります。

このようなドキュメント翻訳の保守作業に必要な労力を和らげてくれるのが Po4a です。Po4a は、様々な形式のドキュメント翻訳を PO という形式で一括して管理し、保守性を上げるツールです。以下では、まず PO というファイル形式が、翻訳にとってどのように便利なのかを説明し、その上で Po4a がどのようなツールなのか説明します。

5.2 PO

PO は、プログラムのメッセージの翻訳のために作られたファイル形式です。ここではその書式や編集用のツールについて見ていきましょう。

5.2.1 PO の基本的な書式

まずは、PO の基本的な書式を説明します。

PO には、原文と訳文の対からなるエントリが空行区切りで多数収められています。最初に、簡単なエントリの例を紹介します。

```
#: src/apt_config_treeitems.cc:99
msgid ""
"%BOption:%b  %s\n"
"%BDefault:%b %s\n"
"%BValue:%b   %s\n"
msgstr ""
"%B オプション:%b %s\n"
"%B デフォルト:%b %s\n"
"%B 値:%b         %s\n"
```

一目瞭然ですが、msgid から始まる一連の行、msgstr から始まる一連の行の、"で囲まれた部分は、それぞれ原文と訳文です。

「#」で始まる行は基本的にすべてコメントです。特に「#:」で始まる行は、原文を抽出した位置を示す参照用の行です。これは xgettext がメッセージを抽出して POT を生成するときに設定します。翻訳者は、msgid を見ても理解できない場合、この参照コメントをもとにしてソースコードを眺めることができます。

「#:」以外にも、様々な種類のコメントがあります。そのようなコメントを含んだエントリの例を紹介します。

```
# 最後の %s は「(core dumped)」。
#. ForTranslators: "%s update %s" gets replaced by a command line, do not translate it!
#: src/generic/apt/download_update_manager.cc:383
#, c-format
msgid "The debtags update process (%s update %s) was killed by signal %d%s."
msgstr ""
"debtags 更新プロセス (%s update %s) がシグナル %d に kill されました %s。"
```

「#.」で始まる行は、原文のメッセージと一緒にソースコードから抽出されたコメントです。主に、メッセージの翻訳に注意が必要な場合やメッセージが翻訳しにくい場合に、開発者から翻訳者への説明に使われます。これも xgettext がメッセージ抽出時に設定します。

「#,」で始まる行はフラグです。最もよく使われるフラグは fuzzy というフラグでしょう。これについては後述します。これは様々なツールが設定します。

それ以外の、「#」の後に何の記号もない行は、翻訳者が翻訳作業時に勝手に追加したコメントです。各エントリにはこのような翻訳者コメントを自由自在につけることができます。

このようなコメント情報は、プログラムのソースコードに書くコメントと同様、人間の作業にのみ必要となるものなので、MO に変換するときにすべて削除されます[*1]。

5.2.2 fuzzy エントリ

msgmerge が古い PO に新しい POT をマージする際、既に翻訳されているエントリの msgid に似た msgid を持つエントリが追加されることがあります。例えば、エントリ A の msgid に似た msgid を持つエントリ A' が追加されるとします。このとき、msgmerge は A をベースとして A' を翻訳できると判断し、A' の msgstr に A の msgstr の内容を設定した上で、A' に fuzzy フラグをつけます。このようにして作られるエントリが、次のような **fuzzy エントリ**です。

```
#: src/main.cc:181
#, fuzzy, c-format
msgid ""
" -q            In command-line mode, suppress the incremental progress\n"
"               indicators.\n"
msgstr " -q                コマンドラインモードで進行状況を逐一表示しません。"
```

しかしこれだけだと、翻訳者には、msgid がどう変化したのか分かりません。そこで GNU gettext のバージョン 0.16 から導入されたのが、次のような、「#|」で始まるコメントです。

[*1] したがって、msgunfmt で MO を PO に変換すると、コメントをまったく含まない PO が得られます。

```
#: src/main.cc:181
#, fuzzy, c-format
#| msgid ""
#| " -q             In command-line mode, suppress the incremental progress "
#| "indicators."
msgid ""
" -q             In command-line mode, suppress the incremental progress\n"
"                indicators.\n"
msgstr " -q                コマンドラインモードで進行状況を逐一表示しません。"
```

「#|」で始まるコメントは、以前の msgid です。msgmerge に--previous を指定した場合に設定されます。このコメントを使えば、以前の msgid と現在の msgid を比較できるので、どのような変更を msgstr に加えれば翻訳を現在の msgid に対応できるのかが簡単に分かります。PO の保守性を上げる仕組みと言えます。msgmerge に--previous が指定されていない場合は指定しておくとよいでしょう。

5.2.3 ヘッダ

翻訳を扱う際、翻訳者やその連絡先、原文の問題の連絡先、翻訳の更新日時、文字セットなどは重要なメタ情報です。PO では、最初のエントリを**ヘッダ**として使用し、そのエントリの msgstr にこれらのメタ情報を含めることになっています。ヘッダの msgid は空文字列にすると決まっています。

また、著作権情報など、書式が曖昧な情報や長い情報は、ヘッダのコメント (つまりファイルの冒頭) に書くことになっています。

以下は、aptitude の ja.po の例です。

```
# Japanese translations for aptitude
# aptitude の日本語訳
# Copyright (C) 2006-2008 Noritada Kobayashi <nori1@dolphin.c.u-tokyo.ac.jp>.
# This file is distributed under the same license as the aptitude package.
#
msgid ""
msgstr ""
"Project-Id-Version: aptitude 0.4.1\n"
"Report-Msgid-Bugs-To: aptitude@packages.debian.org\n"
"POT-Creation-Date: 2008-09-05 16:05+0200\n"
"PO-Revision-Date: 2008-05-16 03:35+0900\n"
"Last-Translator: Noritada Kobayashi <nori1@dolphin.c.u-tokyo.ac.jp>\n"
"Language-Team: Japanese\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=1; plural=0;\n"
```

なお、ここでは省略しましたが、筆者は、メッセージカタログ内での用語統一を容易にするため、aptitude や Subversion など大規模な PO では、ヘッダのコメントに対訳表を含めています。

5.2.4 その他の書式

ここまでで、PO のエントリの書式についてざっと見ました。

PO のエントリについては、msgid と msgstr から成るもの以外にも、複数形への対応として msgid、msgid_plural、msgstr[n] から成るものがあります。詳しくは GNU gettext のドキュメント[*2]の『3 The Format of PO Files』を参照してください。

5.2.5 PO 編集ツール

PO で翻訳作業を行うには PO 編集ツールが便利です。PO はテキストなのでどんなテキストエディタでも編集できますが、翻訳に含まれる「"」や「\」を手でエスケープするのは厄介です。そのような機械的な処理は、PO に特化した PO 編集ツールに任せるのがよいでしょう。また、PO 編集ツールには翻訳支援機能を持つものもあるので、作業効率も上がるはずです。

PO 編集ツールとしてよく知られているのは、Emacs の PO 専用メジャーモードである po-mode、KDE の PO 編集スイートである KBabel、GNOME の PO エディタである Gtranslator、そして poedit です。

[*2] gettext-doc パッケージ。

ここでは、筆者が慣れている po-mode について、筆者がよく使う操作だけざっと説明します。説明中で何度も登場する「選択エントリ」とは「現在カーソルがあるエントリ」の意味です。

まず、エントリ間の移動には以下のようなキー操作が使えます。

`n` 次のエントリに移動します。
`p` 前のエントリに移動します。
`t` 次の既訳エントリに移動します。
`T` 前の既訳エントリに移動します。
`f` 次の fuzzy エントリに移動します。
`F` 前の fuzzy エントリに移動します。
`u` 次の未訳エントリに移動します。
`U` 前の未訳エントリに移動します。
`o` 次の obsolete エントリに移動します。
`O` 前の obsolete エントリに移動します。
`m` 選択エントリの位置をスタックに push します。
`r` スタックから pop して得られたエントリに移動します。

また選択エントリの操作には以下のようなキー操作が使えます。

`RET` 編集用バッファを開いて、選択エントリの `msgstr` を編集します。
`#` 編集用バッファを開いて、選択エントリの翻訳者コメントを編集します。
`k` 選択エントリの `msgstr` をカット (kill) します。
`K` 選択エントリの翻訳者コメントをカット (kill) します。
`w` 選択エントリの `msgstr` をコピーします。
`W` 選択エントリの翻訳者コメントをコピーします。
`y` 選択エントリの `msgstr` にペースト (yank) します。
`Y` 選択エントリの翻訳者コメントにペースト (yank) します。
`TAB` 選択エントリの fuzzy フラグを取り除きます。
`DEL` 選択エントリが既訳の場合は fuzzy フラグをつけます。未訳または fuzzy の場合は obsolete にします。obsolete の場合は削除します。
`C-j` 選択エントリの `msgid` の内容を `msgstr` にコピーします。
`s` 選択エントリの参照コメントをもとにソースコードの該当行を表示します。参照コメントに複数の位置が記述されている場合は、順番に表示します。

編集用バッファでは通常の Emacs の編集操作ができます。特殊な操作は以下のとおりです。

`C-c C-c` 編集用バッファの内容を `msgstr` または翻訳者コメントの内容として設定し、バッファを閉じます[*3]。
`C-c C-k` 編集用バッファの内容を破棄してバッファを閉じます。

最後に、メインバッファでの PO 全体に関わる操作は以下のとおりです。

`_` 行った変更を取り消します。
`?`, `h` ヘルプを表示します。
`V` `msgfmt` に`--check` (`-c`) をつけて実行し、PO に問題点がないか確認した上で保存します。例えば、`msgid` が改行記号で終わっているのに `msgstr` が改行記号で終わっていない場合、それが報告されます。

以上の簡単な説明から分かると思いますが、po-mode は移動や編集がしやすく、非常に便利です。

[*3] もちろん、この変更はメインバッファで保存操作を行うまで保存されません。

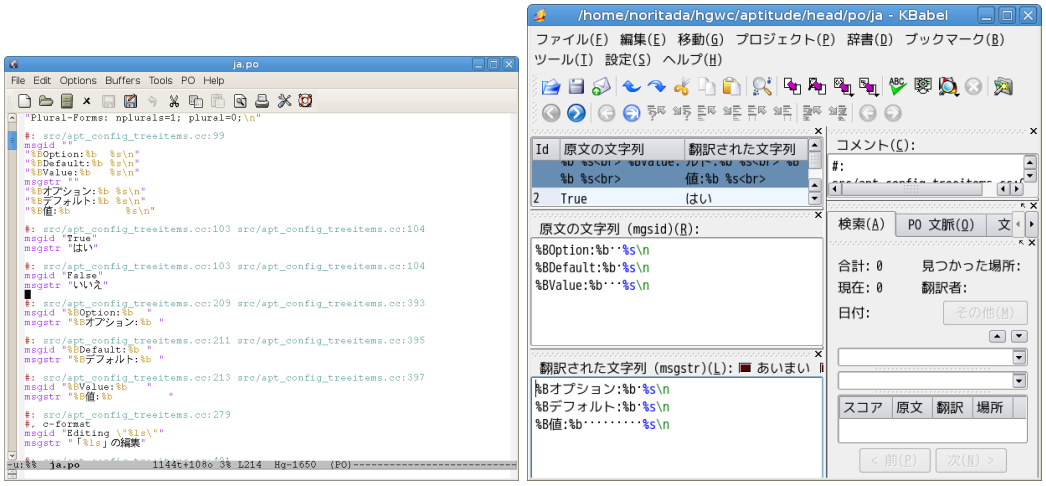

図 1　po-mode と KBabel

ただ、po-mode には今のところ翻訳メモリのような機能はありません。KBabel や Gtranslator は翻訳メモリを備えているので、翻訳メモリに興味のあるかたはこれらを試してみるのもよいでしょう。

5.2.6　GNU gettext ツール群

翻訳作業には PO 編集ツールを使うことになるでしょうが、一方で GNU gettext が提供する、PO や POT に対してコマンドラインから自動処理できるツール群も、知っておくと便利です。以下でざっと説明します。

`msgattrib`　エントリの状態に応じた一括処理をします。例えば、「fuzzy エントリのみにする」、「fuzzy エントリから fuzzy フラグを取り除く」といったことが可能です。
`msgcat`　複数の PO を繋げます。
`msgcmp`　PO と POT が含む `msgid` セットを比較します。
`msgcomm`　複数の PO に含まれる共通の `msgid` を抽出します。
`msgconv`　PO の文字セットを変換します。
`msgen`　英語の PO を作成します。入力ファイルの未訳のエントリの `msgstr` に、`msgid` と同じ文字列を設定します。
`msgexec`　PO に含まれるすべての翻訳に対してコマンドを実行します。
`msgfilter`　PO に含まれるすべての翻訳に対して共通の処理をします。
`msgfmt`　PO をコンパイルして MO に変換します。
`msggrep`　パターンにマッチしたエントリを PO から抽出します。
`msginit`　PO を初期化します。
`msgmerge`　PO に POT をマージします。
`msgunfmt`　MO から PO に逆コンパイルします。
`msguniq`　PO に含まれる重複した `msgid` を 1 つにまとめます。
`xgettext`　ソースコードから `msgid` を抽出して POT を生成します。

開発者が主に使うのは `xgettext` と `msgmerge`、`msgfmt` ですが、他のツールも使いこなせるようになっておくと便利です。使い方は各ツールのマニュアルページを参照してください。

5.2.7　まとめ

PO はプログラム翻訳のために作られたファイル形式で、参照コメントのような翻訳に有用な情報や、fuzzy エントリのような翻訳作業・保守作業の効率を上げるための仕組みを含んでいることを説明しました。また、PO という書式に特化した編集ツールについても説明しました。

5.3 ドキュメント翻訳で PO を使うという考え方

プログラム翻訳のために生み出された PO は、翻訳者にとって翻訳と保守の両方の作業をしやすい環境を提供する存在となりました。その理由としては以下のようなものが考えられます。

- PO 自体に翻訳と保守を支援するための仕組みがある上、翻訳支援機能を持つ PO 編集ツールも存在する。
- メッセージの管理はツールがやってくれるので、人間は翻訳の管理に集中していればよい。
  - メッセージがソースコードのどこにあるかが翻訳者にとって重要でないので、メッセージがソースコード内で移動しても翻訳者は気にする必要がない。
  - `msgmerge` で PO を更新すると、どのエントリの翻訳を更新すべきか瞬時に分かり、PO 編集ツールでそのエントリへ簡単に移動できる。

一方でドキュメント翻訳については、通常は原文をそのままコピーして翻訳作業を開始するため、翻訳以外の部分 (インデント、ドキュメント内での位置など) に変更があった場合、その影響を翻訳者が受けます。プログラム翻訳と比べて分量が多いのに保守性が低いので、結果として翻訳者への負担は非常に大きくなります。この状況は、翻訳者にとって大きな悩みの種でした。

そこで生まれたのが、ドキュメント翻訳に対しても PO を使おうという考え方です。具体的には、ドキュメントの各ブロックをエントリとした PO を作り、ドキュメントと PO とを相互変換できるようにすることで、翻訳者が PO での翻訳管理に集中できるようにします。以下のツールがそのような思想で作られています。

Po4a　Debian で開発されている[*4]。対象とするドキュメント形式は、プレーンテキスト、XML、HTML、LaTeX、nroff (man)、Pod など多数で、Perl で書かれたモジュールおよびプログラムの集合として実装されている。Debian パッケージは `po4a`。

poxml　KDE で、KDE SDK モジュールの 1 つのコンポーネントとして開発されている。対象とするドキュメント形式は DocBook XML で、C++ で実装された、`po2xml`、`xml2pot` などの実行可能プログラムから成る。Debian パッケージは `poxml`。

xml2po　GNOME で gnome-doc-utils の 1 つのユーティリティとして開発されている。対象とするドキュメント形式は DocBook XML である。Python で実装されたシンプルな 1 つのファイルから成るので、手軽にドキュメントのビルドに使用できる。Debian では `gnome-doc-utils` パッケージに含まれる。

poxml や xml2po が DocBook XML のみを扱うツールであるのに対し、Po4a が様々な形式をサポートしているのは、おそらく GNOME や KDE と Debian の立場の違いを反映しているのでしょう。GNOME や KDE はプロジェクト内でドキュメント形式を DocBook XML に統一できますが、ディストリビュータである Debian では、様々なソフトウェアの様々なドキュメントに対応する必要があるのです。

以降のセクションでは、ドキュメント翻訳に PO を使用する 3 つのツールのうち、唯一複数のドキュメント形式をサポートしている Po4a について見ていきます。

5.4 Po4a の基本

Po4a というソフトウェア名は、「po for anything」から来ています。名前からも分かるように、最初から様々なドキュメント形式の翻訳を PO で管理することを目的としており、そのために入力形式として複数の形式を扱うことを想定した作りとなっています。

例えば、以下のような形式が現在サポートされています。

KernelHelp　カーネル設定ヘルプのドキュメント形式です。

[*4] 厳密に書くと、Debian Project のメンバーが、同プロジェクトが提供しているフリーソフトウェアプロジェクトホスティングサービスである Alioth 上で、1 つのプロジェクトとして開発しています。

nroff　Unix の伝統あるマニュアルページの形式です。BSD マニュアルページで使用されている mdoc 形式もサポートされています。初心者が読み書きしにくい形式ですが、Po4a でサポートされたので、インラインのマークアップだけ分かれば (原文の真似をすれば) 翻訳できます。

POD　Perl 関連のドキュメントに使われる形式です。ソースコードの横にコメントとして書き込まれたドキュメントは翻訳しにくいですが、それを PO として抽出し、翻訳しやすくします。

SGML　最近では XML のほうが主流ですが、一昔前のドキュメント形式の主流です。現在サポートされている DTD は DebianDoc-SGML と DocBook SGML のものです。

TeX/LaTeX　著名な組版ソフトウェア LaTeX の形式です。

GNU Texinfo　GNU のドキュメントに使用される形式です。

XML　最近よく使われるドキュメント形式です。現在サポートされている DTD は Dia、DocBook XML、Guide XML、XHTML のものです。

サポートされている形式の一覧は、`po4a-gettextize`、`po4a-translate`、`po4a-updatepo` などのコマンドに `--help-format` オプションを与えると表示できます[*5]。コマンドに`--format` (`-f`) などを与えてドキュメント形式を指定する場合、この一覧に載っている名前を使用してください。

```
noritada[3:39]%  po4a-gettextize --help-format   terra:~/svnwc/build-common/doc
有効フォーマット一覧:
  - dia: 非圧縮 Dia ダイアグラム。
  - docbook: Docbook XML。
  - guide: Gentoo Linux の xml ドキュメントフォーマット。
  - ini: .INI フォーマット。
  - kernelhelp: 各カーネルのコンパイルオプションのヘルプメッセージ。
  - latex: LaTeX フォーマット。
  - man: 古き良きマニュアルページフォーマット。
  - pod: Perl オンラインドキュメントフォーマット。
  - sgml: debiandoc と docbook DTD の双方。
  - texinfo: info ページフォーマット。
  - tex: 汎用 TeX ドキュメント (latex を参照)。
  - text: シンプルなテキストフォーマット。
  - wml: WML ドキュメント。
  - xhtml: XHTML ドキュメント。
  - xml: 汎用 XML ドキュメント (docbook を参照)。
```

5.5　Po4a の使い方

5.5.1　導入 (1): 原文ドキュメントファイルから POT を生成する

Po4a を使ってドキュメントの翻訳をする場合、最初にすべきことは、原文のドキュメントファイルから POT を生成することです。POT の生成には `po4a-gettextize` コマンドを使用します。`-f` オプションの引数にドキュメントファイルの形式を、`-m` オプションの引数に原文ドキュメントファイル (**マスタードキュメント**) の名前を、`-p` オプションの引数に出力する POT のファイル名を与えて、コマンドを実行します。`-M` オプションの引数で、原文ドキュメントファイルの文字セットを指定することも可能です。

特に問題がない場合はすんなりとコマンドの実行が終了し、POT が生成されます。ドキュメントの構造に問題がある場合 (例えば XML においてタグがきちんと閉じられていない場合) はエラーになります。

POT が生成されたら、翻訳作業はプログラム翻訳の場合と同じです。POT をコピーして PO を作成し、ヘッダを適切に設定した上で翻訳作業を始めましょう。

■例　次の例では、`hoge.en.html` という XHTML ドキュメントから `hoge.pot` を生成しています。

`hoge.en.html` (入力):

[*5] 残念ながら今のところ `po4a` コマンドに与えることはできないようです。

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
<title>Test file</title>
</head>
<body>
<h1>Test file</h1>
<p>This is an <a href="apple">apple</a>.</p>
<p>This is an <a href="orange">orange</a>.</p>
</body>
</html>
```

コマンドライン:

```
noritada[14:14]%  po4a-gettextize -v -f xhtml -m hoge.en.html -p hoge.pot
```

`hoge.pot` (出力):

```
# SOME DESCRIPTIVE TITLE
# Copyright (C) YEAR Free Software Foundation, Inc.
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"POT-Creation-Date: 2008-09-20 14:21+0900\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=utf-8\n"
"Content-Transfer-Encoding: ENCODING"

# type: Attribute 'xml:lang' of: <html>
#: hoge.en.html:4 hoge.en.html:4
msgid "en"
msgstr ""

# type: Content of: <html><body><h1>
#: hoge.en.html:6 hoge.en.html:9
msgid "Test file"
msgstr ""

# type: Content of: <html><body><p>
#: hoge.en.html:10
msgid "This is an <a href=\"apple\">apple</a>."
msgstr ""

# type: Content of: <html><body><p>
#: hoge.en.html:11
msgid "This is an <a href=\"orange\">orange</a>."
msgstr ""
```

5.5.2 導入 (2): 原文・翻訳ドキュメントファイルから PO を生成する

原文と同じ形式で翻訳したドキュメントが既にあり、それを Po4a 用の PO に移行したい場合にも、`po4a-gettextize` が使えます。「POT をコピーして PO を作った上で、翻訳ドキュメントの各ブロックをコピー・アンド・ペーストで PO の各エントリに埋め込む」という、間違いなくうんざりする作業は不要です。

ただしこの場合、どの原文がどの訳文に対応するかを Po4a が把握する必要があるので、**原文ドキュメントファイルと翻訳ドキュメントファイルが同じ構造である**ことが前提になります。「同じ構造」とは、Po4a が切り分ける単位、つまりブロックのレベルで見たときに、対応する要素が同じ順序で並んでいる、という意味です。もし翻訳側でブロックの追加や削除が行われていたら、変換はエラーで終わるでしょう。一部のブロックの順序が入れ替わっている場合、変換は見た目は無事に終わるかもしれませんが、入れ替わったブロックの `msgid` と `msgstr` の対応関係はおかしくなっているはずです。

訳注や翻訳者情報を別個のブロックとして翻訳ドキュメントに追加している場合、それは一旦取り除いてください。Po4a では、PO から翻訳ドキュメントを生成する際に、原文にはない要素を追加する方法が提供されています。方法については後述します。

原文ドキュメントファイルと翻訳ドキュメントファイルが同じ構造であれば、`po4a-gettextize` を用いた変換は成功するはずです。`-f` オプションの引数にドキュメントファイルの形式を、POT を生成する場合の一連のオプショ

ンに加えて、-l オプションの引数に翻訳ドキュメントファイルの名前を与えてコマンドを実行します。-L オプションの引数で、翻訳ドキュメントファイルの文字セットを指定することも可能です。

変換に成功すると、すべてのエントリに fuzzy フラグが設定された PO が生成されます。すべてのエントリに fuzzy フラグが設定されるのは、「ざっとでもいいので、ユーザには変換後にすべてのエントリの原文と訳文を確認してほしい」という開発者の意図を反映したものですユーザが自由に編集できる普通のドキュメントに少し制限を与えて Po4a の管理下に置く[*6]際には、何かしらの問題が発生している可能性があるのです。

■例　以下では、先程の hoge.en.html に対応する日本語訳 hoge.ja.html を PO に変換することを試みます。

まずは、とりあえず po4a-gettextize を実行してみました。

hoge.ja.html (入力):

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="ja" xml:lang="ja">
<head>
<title>テストファイル</title>
</head>
<body>
<h1>テスト用のファイル</h1>
<p>これは<a href="apple">リンゴ</a>です。</p>
<p>これは<a href="orange">オレンジ</a>です。</p>
<p>(訳注: オレンジはミカンとは違います。)</p>
</body>
</html>
```

コマンドライン:

```
noritada[14:54]%  po4a-gettextize -v -f xhtml -m hoge.en.html -l hoge.ja.html -p ja.po
po4a gettextize: Original has less strings than the translation (6<7). Please
               fix it by removing the extra entry from the translated file. You
               may need an addendum (cf po4a(7)) to reput the chunk in place
               after gettextization. A possible cause is that a text duplicated
               in the original is not translated the same way each time. Remove
               one of the translations, and you're fine.
po4a gettextization: Structure disparity between original and translated files:
msgid (at hoge.en.html:6 hoge.en.html:9) is of type 'Content of:
<html><body><h1>' while
msgstr (at hoge.ja.html:6) is of type 'Content of: <html><head><title>'.
Original text: Test file
Translated text: テストファイル
(result so far dumped to gettextization.failed.po)
The gettextization failed (once again). Don't give up, gettextizing is a subtle
art, but this is only needed once to convert a project to the gorgeous luxus
offered by po4a to translators.
Please refer to the po4a(7) documentation, the section "HOWTO convert a
pre-existing translation to po4a?" contains several hints to help you in your
task
```

訳注を独立したパラグラフにしたために PO への変換が失敗したと考え、訳注の行を削ってもう一度 po4a-gettextize を実行してみます。入力ファイルの内容は省略します。

コマンドライン:

```
noritada[14:55]%  po4a-gettextize -v -f xhtml -m hoge.en.html -l hoge.ja.html -p ja.po
po4a gettextization: Structure disparity between original and translated files:
msgid (at hoge.en.html:6 hoge.en.html:9) is of type 'Content of:
<html><body><h1>' while
msgstr (at hoge.ja.html:6) is of type 'Content of: <html><head><title>'.
Original text: Test file
Translated text: テストファイル
(result so far dumped to gettextization.failed.po)
The gettextization failed (once again). Don't give up, gettextizing is a subtle
art, but this is only needed once to convert a project to the gorgeous luxus
offered by po4a to translators.
Please refer to the po4a(7) documentation, the section "HOWTO convert a
pre-existing translation to po4a?" contains several hints to help you in your
task
```

なんだか変なエラーが出てしまっています。どうも、title と h1 の原文がともに「Test file」なのに、一方の訳は「テストファイル」で、他方の訳が「テスト用のファイル」となっていることに原因があるようです。そこで、両方の訳を統一して po4a-gettextize にかけると、変換に成功します。

[*6] 例えば、「ブロックの順序など全体的な構造を原文に合わせる必要がある」、「原文の同じ内容のブロックには、同じ訳文を当てる必要がある」など。

hoge.ja.html (入力):

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="ja" xml:lang="ja">
<head>
<title>テストファイル</title>
</head>
<body>
<h1>テストファイル</h1>
<p>これは<a href="apple">リンゴ</a>です。</p>
<p>これは<a href="orange">オレンジ</a>です。</p>
</body>
</html>
```

コマンドライン:

```
noritada[15:07]%  po4a-gettextize -v -f xhtml -m hoge.en.html -l hoge.ja.html -p ja.po
```

ja.po (出力):

```
# SOME DESCRIPTIVE TITLE
# Copyright (C) YEAR Free Software Foundation, Inc.
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"POT-Creation-Date: 2008-09-20 15:07+0900\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=utf-8\n"
"Content-Transfer-Encoding: ENCODING"

# type: Attribute 'xml:lang' of: <html>
#: hoge.en.html:4 hoge.en.html:4
#, fuzzy
msgid "en"
msgstr "ja"

# type: Content of: <html><body><h1>
#: hoge.en.html:6 hoge.en.html:9
#, fuzzy
msgid "Test file"
msgstr "テストファイル"

# type: Content of: <html><body><p>
#: hoge.en.html:10
#, fuzzy
msgid "This is an <a href=\"apple\">apple</a>."
msgstr "これは<a href=\"apple\">リンゴ</a>です。"

# type: Content of: <html><body><p>
#: hoge.en.html:11
#, fuzzy
msgid "This is an <a href=\"orange\">orange</a>."
msgstr "これは<a href=\"orange\">オレンジ</a>です。"
```

5.5.3 PO と原文ドキュメントファイルから翻訳ドキュメントファイルを生成する

po4a-translate

5.5.4 原文ドキュメントファイルの更新を PO に反映させる

po4a-updatepo

5.5.5 PO と翻訳ドキュメントファイルを一括更新する

po4a

5.5.6 原文にない要素を翻訳ドキュメントファイルに挿入する

5.6 Po4a の内部構造

5.6.1 TransTractor::parse()

6 【でびあん】Debian パッケージメンテナというお仕事【現在募集中】

やまね ひでき

6.1 Debian パッケージメンテナのお仕事

実は、Debian パッケージのメンテナになるにはそんなに敷居は高くありません。よっぽど入学試験／就職／転職活動の方が難しいと思うほどです。恐れることはありません。多少なりとも興味がある方はこれからの説明をご覧ください。

6.2 メンテナになる前の下準備

1. GPG 鍵を作っておく (aptitude install gpg && gpg –gen-key)
2. 公的な身分証明書を用意しておく
3. 先ほど作った GPG 鍵と身分証明書を利用して、既存の Debian Developer と GPG キーサインの交換を行っておく
4. メンテナや Developer として問題ないと見なされる作業をしておく (contribute!)

この 4 つだけ準備したら、さぁ、メンテナになる作業の始まりです。

6.3 The way to Debian package maintainer

メンテナになる方法としては 2 点あります。

1. 自分から「このソフトを Debian パッケージにしたい」と申請する
2. 既存のパッケージのメンテナからメンテナンスを引き継ぐ

1. は ITP (Intent To Package) という形で行います。特に RFP (Request For Package) という形でユーザから「このソフトを Debian パッケージにして欲しいなぁ」という声に答えるのが望ましいでしょう。

2. は RFH (Request For Help、協力者募集中)、RFA (Request for Adoption、新メンテナ募集中) や O (Orphaned、みなしご化) というステータスになったパッケージに対して「自分がメンテナになります」と宣言します。

ITP, RFP, RFA, Orphaned は全て Debian Bug Tracking System (BTS) で管理登録されます (BTS の使い方についてはそのうち別途)。これらを総称して「作業が望まれるパッケージ (Work-Needing and Prospective Packages; WNPP)」と呼びます。この WNPP、BTS で追うのは結構辛いので wnpp.debian.net [*7] を使いましょ

[*7] `http://wnpp.debian.net` ではパッケージ化希望されているものや、メンテナが引き取り手を募集しているパッケージを検索できます。
- ページの見た目は良くない ;-)

う。各略語の意味などについては http://www.jp.debian.org/devel/wnpp/ をご覧ください。

6.3.1 ITP、RFP パッケージ化

大抵 1 からパッケージ化作業ですので、ウェブサイトからソースを取得してライセンスを確認の上、パッケージ作りのお作法に従ってパッケージを作成します。パッケージを作成する際は、unstable 環境の lintian で warning などが出なくなるよう (Debian Policy Manual に準拠するよう) に頑張りましょう。パッケージが作成できたら pbuilder でクリーンルーム環境でビルド可能かどうか (Build-Depends に間違いが無いか) のチェックを行います。問題なくビルドできたら piuparts を使ってインストール／アンインストールで問題が無いか (Depends や preinst,postinst,prerm,postrm スクリプトに問題が無いか) を確認しましょう。ここまで来て問題なければ「Eat your own dog food」、つまり自分の環境にパッケージを入れて問題が起きないかどうかチェックするとベストでしょう。

■ライセンスについて　ちなみに一番大変なのはここでの「ライセンス確認」作業だと個人的には思っています。そのソフトのウェブページににちょっと書いてあることを鵜呑みにしたりすると、実は中身は全然違うライセンスだ、とか、実はリンクするライブラリとは矛盾するライセンスだ、とかがありますし、英訳ライセンスがないと Debian のパッケージ管理者 (ftpmaster) がライセンス的にそのパッケージを受け入れるかどうかの判断がそもそも出来ませんので必ず適当な英訳ライセンスへの翻訳作業が必要です。

ライセンスは、独自ライセンスよりも一般的に FLOSS の代表的ライセンス (GPL, BSD, MIT, Artistic など) を使う方が全ての利害関係者にとって利益があるでしょう。何故ならば、ライセンスは「プロトコル」のようなものであり、既存プロトコルを利用する方が新たにプロトコルを実装するよりもはるかに容易に取り扱えるからです。代表的ライセンスでは目的が達成できないときにのみ、独自ライセンスを利用するのが賢いやり方です。

6.3.2 RFA, Orphan への対処

すでに現在のメンテナが引き継ぎ手を募集している状態 (RFA) や、完全にギブアップ or 興味を失ってしまった (Orphan) になったパッケージについては、BTS を利用してパッケージを引き取る旨の宣言 (ITA, Intend To Adapt) を行いましょう。加えて関連のメーリングリストで「自分がやろうと思うけど、どう？」と聞いてみるとさらに良いでしょう。合意がとれたら、debian/changelog にて該当の RFA/Orphan バグを閉じるエントリを追記しておきます。大抵の場合は色々とバグレポートが溜まっているので片付けて同様に changelog に記載しておきましょう。

6.4 アップロード

ここまでが終わったらアップロードを行います。Debian Developer であればパッケージを dput や dupload と呼ばれるコマンドを使って、作成されたソースパッケージとバイナリパッケージを直接 Debian へアップロードします。そうでない場合は適当な場所にアップロードして、Debian Developer に代理アップロードをお願いしましょう (アップロード先は、http:://mentors.debian.net、代理アップロード依頼先は debian-devel@debian.or.jp がおすすめです)。

```
dupload -t anonymous-ftp-master ccspatch_1.6.4-20080903-1_i386.changes
```

この時点でツールは dsc ファイルや changes ファイルに gpg 署名されているかどうかをチェックして、されていない場合はアップロードしないなどしてくれます。

サーバ側では、まず「既にあるパッケージの更新なのか、それとも新規パッケージなのか」の判断を行います。新規パッケージの場合は NEW Queue (http://ftp-master.debian.org/new.html) と呼ばれる所に自動的に移されて、ftpmaster が逐次入れてよいものかどうかのチェックを行っています。おおよそこの工程は 1 、2 週間程度か

- RSS で状況が取得できるので、RSS reader で適宜ピックアップにてチェックすることをお勧めします。

かります。

チェックは特にライセンスの問題が無いかについてなどが争点となります。この作業は高度な判断が要求されるため、人的リソース的にボトルネックになりやすいところと、拒否された場合に「何故このパッケージが拒否されたのか」の記録が容易にトラッキングできないのが問題点です。人的リソースについては、アクティブなメンバーを入れることで以前と比較しても改善がされており、トラッキングについては最近になってシステマチックにチェック可能なように提案が出てきているところです。

更新パッケージについては、サーバ側でソースパッケージの gpg 署名をチェックなどをして「本当にこれは入れていいものかどうか」を判断してダメな場合は reject されます。このチェックについては、アップロード後にサーバから受け入れ／拒否のメールが届きます。

まずアップロードされたパッケージの処理を始めたよ、というメール

```
From: Archive Administrator <dak@ftp-master.debian.org>
Subject: Processing of ccspatch_1.6.4-20080903-1_i386.changes

ccspatch_1.6.4-20080903-1_i386.changes uploaded successfully to localhost
along with the files:
  ccspatch_1.6.4-20080903-1.dsc
  ccspatch_1.6.4-20080903.orig.tar.gz
  ccspatch_1.6.4-20080903-1.diff.gz
  linux-patch-tomoyo_1.6.4-20080903-1_all.deb

Greetings,

        Your Debian queue daemon
```

チェックして incoming repository に入れたよ、というメール

```
From: Debian Installer <installer@ftp-master.debian.org>
Subject: ccspatch_1.6.4-20080903-1_i386.changes ACCEPTED

Accepted:
ccspatch_1.6.4-20080903-1.diff.gz
  to pool/main/c/ccspatch/ccspatch_1.6.4-20080903-1.diff.gz
ccspatch_1.6.4-20080903-1.dsc
  to pool/main/c/ccspatch/ccspatch_1.6.4-20080903-1.dsc
ccspatch_1.6.4-20080903.orig.tar.gz
  to pool/main/c/ccspatch/ccspatch_1.6.4-20080903.orig.tar.gz
linux-patch-tomoyo_1.6.4-20080903-1_all.deb
  to pool/main/c/ccspatch/linux-patch-tomoyo_1.6.4-20080903-1_all.deb


Override entries for your package:
ccspatch_1.6.4-20080903-1.dsc - source devel
linux-patch-tomoyo_1.6.4-20080903-1_all.deb - extra admin

Announcing to debian-devel-changes@lists.debian.org


Thank you for your contribution to Debian.
```

これでパッケージのアップロードが完了しました。BTS に記載されているバグを修正した旨を debian/changelog に適切なスタイルで記述しておくと、BTS から bug closed のメールがやってきます。

6.4.1 その他アップロードに関する注意

ちなみに、Debian バージョンのみの更新の場合 (foo 1.0-1 から foo 1.0-2 への更新など) はオリジナルのソースファイル (orig.tar.gz) をアップロードしないことになっています。もし、orig.tar.gz をアップロードした場合は後から拒否のメールが届きます。

あと注意として、ライブラリファイルや他の多くのパッケージと関連しているようなパッケージで更新をかけると unstable 環境が壊れるような場合は、experimental へアップロードして関係者と調整することが推奨されています。

6.5 メンテナンス作業

BTS を通じてユーザからバグ報告メールが来るので、それに対応してパッケージの debian ディレクトリ以下を更新して、修正出来たら changelog に完了した旨を記載しておきます (dch コマンドが便利です)。特に RC (Release

Critical) バグは早めに修正しましょう。バグが他のパッケージと関連する場合は適当なメーリングリストで関係者と対応を協議して進めていきます。場合によっては、自分のパッケージのバグではなく、他のパッケージのバグである場合もありますので、その場合はバグを BTS で reassign しておきます。パッケージとしてのバグではなく、元々のソース由来のバグの場合は、バグを upstream (開発元) に転送するかパッチを作って upstream と協議しましょう。

そして upstream での更新に合わせてパッケージも更新します。upstream の更新は debian/watch ファイルを適切に記載していれば、DEHS (Debian External Health Status、`http://dehs.alioth.debian.org`) によって定期的にチェックされ、Debian Quality Assuarance の developer ページ (`http://qa.debian.org/developer.php`) で状況を確認できます (あとは upstream のリリースアナウンスが流れるメーリングリストに入っておくと良いでしょう)。きちんと設定された debian/watch ファイルがあれば、パッケージによっては更新がものの 1 分も経たずに出来ます。

チーム制でメンテナンスをするような場合は共有 repository に更新したソースを忘れずに commit しておきましょう。commit 出来ない場合は、チームのメーリングリストにパッチを送ります。幾度か適切なパッチを送りつづけていると、いつの間にか commit 権が与えられていることが多い様です。

なお、バグ修正や Debian Policy に適合するために Debian パッケージ側で色々とパッチを当てている (dpatch や quilt を利用する) と新バージョン側のソースとコンフリクトを起こしてその修正に時間がかかることがありますので、なるべく upstream の開発者とメーリングリストなどを通じて普段からコンタクトを取り合っておいて、パッチ自体が取り込まれて不要になるように働きかけましょう。

この様にしてパッケージ化、メンテナンス等をメンテナは日々行っていきます。特に問題なく時間が経過したパッケージは unstable から testing に移行され、次のリリースを待つことになります。そして、リリースの日を迎えたパッケージは stable へと移されます。

6.6 特殊な場合

以下に頻繁には起きないはずの、ちょっと変わった場合の対応を挙げておきます。

6.6.1 NMU

Non-Maintainer Upload の略で、既存のメンテナ以外の人間がバグ修正のためにパッケージをアップロードすることを指します。あまり頻繁に NMU があるようなパッケージについては、メンテナが本当に活動しているのかどうかをチェックした方がいいでしょう。

6.6.2 FREEEEEEZE!!

特にリリース前には「システム全体の安定化」が必要なため、freeze、つまりパッケージの unstable から testing への自動移行が停止されます。しかし、freeze の間に発覚した、そのままリリースされてしまっては困るようなバグの修正が必要な場合もあります。その場合は リリースマネージャ (RM) に対して「私のこのパッケージを testing に入れるようにしてください！」とお願いします。これを「unblock (exception) request」と一般に呼びます。やり方としてはメーリングリストで RM が理解しやすいフォーマットでメールを投げるだけです。

```
To: debian-release@lists.debian.org
subject: Please unblock <package> <version>

\begin{itemize}
 \item changelog の抜粋
 \item 理由
 \item debian-devel-announce で流れている freeze exception のどれに当てはまるのか
 \item このアップデートによって何が改善されるのか
 \item このアップデートでは regression が起きない説明
 \item その他、RM が納得してくれるような内容
\end{itemize}
```

うまくすれば RM から ”unblocked” という素っ気ないメールが届きます。これが来たら大成功です。今 unstable にあるパッケージは testing へ移行することが可能になります。

6.7 メンテナ権の移譲

パッケージメンテナも人の子、残念ながら諸事情によりメンテナンスが出来なくなるような事態が起きるかもしれません。なるべく一人でメンテナンスせずチームあるいは自分以外の詳しい／信用できるメンテナもアップロードできるようにしておきます。具体的には debian/control の Uploaders フィールドにアップロードしてもいい人の名前とメールアドレスを書いておきます (GPG 署名に使うものと同一である必要があります)。

```
Source: ttf-vlgothic
Section: x11
Priority: optional
Maintainer: Debian Fonts Task Force <pkg-fonts-devel@lists.alioth.debian.org>
Uploaders: Hideki Yamane (Debian-JP) <henrich@debian.or.jp>
DM-Upload-Allowed: yes
Build-Depends: debhelper (>= 5)
Build-Depends-Indep: defoma, bzip2
Standards-Version: 3.8.0
Homepage: http://dicey.org/vlgothic/
Vcs-Svn: svn://svn.debian.org/pkg-fonts/packages/ttf-vlgothic/
Vcs-Browser: http://svn.debian.org/wsvn/pkg-fonts/packages/ttf-vlgothic/
```

そして、メンテナンスが出来なくなりそうな場合は、あまり粘らずにさっさと宣言 (RFA, Orphan) しておきましょう。宣言が無いと「このパッケージ古いな」という悪評が立つだけ…という結果になりかねません。なお、日本の首相と違ってメンテナは頻繁に交代してもきちんとした理由があれば文句は言われることはありません :-)　（できれば最初の方から複数人でのチームメンテナンスなどしておくと困ったことになる確率は減ります）

6.8 終わりに

如何でしたか？パッケージメンテナというお仕事がどのようなものか、多少理解が深まっていただければ幸いです。

Debian 勉強会資料

2008 年 9 月 20 日　　初版第 1 刷発行
東京エリア Debian 勉強会 （編集・印刷・発行）