2007年夏号

あんどきゅめんてっどでびあん

東京エリアDebian勉強会著

デビアン勉強会

目次

1	パッチ管理ツール quilt の使い方	2
2	dpatch についての小ネタ	19
3	dbs	22
4	darcs 使いかた	27
5	git-buildpackage の使いかた	30
6	git の実装技術的解説	39
7	仮想マシンモニタ KVM	41
8	Bug Squashing Party	43
9	apt-torrent	47
10	プロジェクトトラッカーの勧め	50
11	サーバをエッチにしてみました	53
12	最近 pbuilder ってどうよ?	57
13	Debian on SuperH	61
14	Debian の情報フロー	67
15	Debian 勉強会資料の作成方法	68
16	Debian 勉強会 2006 年結果統計	70
17	Debian 勉強会 2006 年、作業フロー	74
18	Debian 勉強会 2007 年度計画検討結果	75
19	Debian Weekly News trivia quiz	76
20	仮想化友の会との合同クイズ	80
21	Debian Weekly News 問題回答	82

『あんどきゅめんてっど でびあん』について

本書は、東京周辺で毎月行なわれている『東京エリア Debian 勉強会』で使用された資料・小ネタ・必殺技などを一冊にまとめたものです。収録範囲は勉強会第 23 回から第 28 回まで。内容は無保証、つっこみなどがあれば勉強会にて。

1　パッチ管理ツール quilt の使い方

小林 儀匡

本節ではパッチ管理ツール quilt[*1] を取り上げ、その基本操作を中心に説明します。quilt は Debian に特有な (つまり Debian-specific な) ソフトウェアではなく、Debian とは全く関係のない場所で開発されています。しかし、このツールが Debian パッケージの作成に有用であることを理解し、実際に作成に使用してもらえたら嬉しい限りです。

1.1　はじめに

1.1.1　パッチとは

開発者であれば当然のように作成したり読んだり当てたりしたことがある**パッチ**ですが、一般ユーザには見慣れないものだと思います。そこで、最初にパッチについて説明しておきます。

パッチとは、2 つのファイルの差分を含むファイルのことです。次のような 2 つのファイルがあるとしましょう。

```
nori1[3:57]%  cat file1.txt                                    saba:~/tmp/q
あいうえお
かきくけこ
さしすせそ
たちつてと
なにぬねの
はひふへほ
まみむめも
や　ゆ　よ
らりるれろ
わ　を　ん
nori1[3:58]%  cat file2.txt                                    saba:~/tmp/q
あいうえお
ばびぶべぼ
たちつてと
なにぬねの
まみむめも
や　ゆ　よ
らりるれろ
わ　を　ん
```

これらのファイルのどこが違っているかすぐにわかりますか? 一々目で比べる必要はありません。`diff` コマンドでこれらの差分をとることができます。

```
nori1[3:57]%  diff file1.txt file2.txt                         saba:~/tmp/q
2,3c2
< かきくけこ
< さしすせそ
---
> ばびぶべぼ
6d4
< はひふへほ
```

`file1.txt` と `file2.txt` の内容や行番号を考えれば、なんとなく意味がわかると思います。それでかまいません。もちろん、この出力を手で書けるようにならなくてかまいません。

この形式でもよいのですが、通常は `diff` コマンドに`-u` オプションをつけ、次のような **unified diff** という形式で出力させます。

[*1] `http://savannah.nongnu.org/projects/quilt`

```
nori1[3:57]%  diff -u file1.txt file2.txt                        saba:~/tmp/q
--- file1.txt   Thu Jan 18 03:57:14 2007
+++ file2.txt   Thu Jan 18 03:57:22 2007
@@ -1,9 +1,7 @@
 あいうえお
-かきくけこ
-さしすせそ
+ ばびぶべぼ
 たちつてと
 なにぬねの
-はひふへほ
 まみむめも
 や　ゆ　よ
 らりるれろ
```

こちらのほうが、追加・削除された行がどちらのファイルに所属するのか、そしてファイルのどのような部分が異なるのかがわかりやすいでしょう。

パッチとは要は、次のようにしてこういった出力をファイルに収めたものです。

```
nori1[3:58]%  diff -u file1.txt file2.txt > file.diff            saba:~/tmp/q
```

ただし、全く縁のない 2 つのファイルであれば差分はわざわざファイルに収める必要はないでしょう。わざわざパッチを作成するのは、ファイルに変更を加える前後の差分を見たり、後で紹介するように機械的な処理で変更を加えられるようにしたりするためです。そこで、ここまでは file1.txt と file2.txt は無縁のファイルとしてきましたが、ここからは file1.txt を改変したものが file2.txt であると考えてください。つまり file1.txt が改変前のファイルで、file2.txt が改変後のファイル、パッチ file.diff には file1.txt を file2.txt にするための変更が含まれている、と考えてください。

さて、作成したパッチは、patch コマンドを用いて、その中に含む変更を該当ファイルに**適用する**（**当てる**）ことができます。

```
nori1[4:00]%  patch < file.diff                                  saba:~/tmp/q
patching file file1.txt
```

こうして file1.txt の内容は file2.txt の内容と同一になります。

```
nori1[4:01]%  cat file1.txt                                      saba:~/tmp/q
あいうえお
ばびぶべぼ
たちつてと
なにぬねの
まみむめも
や　ゆ　よ
らりるれろ
わ　を　ん
nori1[4:02]%  cmp file1.txt file2.txt                            saba:~/tmp/q
```

また、-R オプションを指定して逆向きに当てることもできます。

```
nori1[4:18]%  patch -R < file.diff                               saba:~/tmp/q
patching file file1.txt
nori1[4:19]%  cat file1.txt                                      saba:~/tmp/q
あいうえお
かきくけこ
さしすせそ
たちつてと
なにぬねの
はひふへほ
まみむめも
や　ゆ　よ
らりるれろ
わ　を　ん
```

file1.txt の内容は元に戻りました。

パッチは必ずしもうまく当たらないことを忘れてはなりません。次のように、file1.txt の 2 行目を手元で改変したとします。

```
nori1[5:29]%  cat file1.txt                                    saba:~/tmp/q
あいうえお
がぎぐげご
さしすせそ
たちつてと
なにぬねの
はひふへほ
まみむめも
や ゆ よ
らりるれろ
わ を ん
```

これに先程のパッチを当てようとすると次のようにエラーになります。

```
nori1[5:30]%  patch < file.diff                                saba:~/tmp/q
patching file file1.txt
Hunk #1 FAILED at 1.
1 out of 1 hunk FAILED -- saving rejects to file file1.txt.rej
```

これは、手元で与えた変更とパッチが加えたい変更が競合しているからです。

もちろん、file1.txt の 2 行目を改変する代わりに、次のように最終行に 1 行加えたとします。

```
nori1[5:32]%  cat file1.txt                                    saba:~/tmp/q
あいうえお
かきくけこ
さしすせそ
たちつてと
なにぬねの
はひふへほ
まみむめも
や ゆ よ
らりるれろ
わ を ん
むふふふふ
```

この場合、手元で加えた変更はパッチが含む変更とは被らないので、パッチはうまく当たります[*2]。

```
nori1[5:36]%  patch < file.diff                                saba:~/tmp/q
patching file file1.txt
nori1[5:52]%  cat file1.txt                                    saba:~/tmp/q
あいうえお
ばびぶべぼ
たちつてと
なにぬねの
まみむめも
や ゆ よ
らりるれろ
わ を ん
むふふふふ
```

これまでは 1 つのファイルの変更を 1 つのパッチに収めてきましたが、1 つのパッチに複数のファイルの変更を収めることも可能です。以下は、diff に-r オプションを渡して、内容の一部が異なる 2 つのファイルを含む 2 つのディレクトリの差分をパッチとして出力させた例です。

[*2] ただしあくまで形式的にであって、意味的に適切かどうかは確認する必要があります。

```
nori1[5:14]%  diff -ur skkdic.orig skkdic > skkdic.diff              saba:~/tmp
nori1[5:14]%  cat skkdic.diff                                       saba:~/tmp
diff -ur skkdic.orig/debian/changelog skkdic/debian/changelog
--- skkdic.orig/debian/changelog        Fri Dec 15 03:36:49 2006
+++ skkdic/debian/changelog     Mon Dec 25 17:39:10 2006
@@ -1,5 +1,7 @@
 skkdic (20061130-2~pre1) unstable; urgency=low

+  * debian/rules: Pass the '-k' option to dh_installchangelogs so that
+    all of ChangeLog* are installed equally.
   * debian/skkdic{,-cdb,-extra}.docs: Deleted since their settings are
     now moved into debian/rules.

diff -ur skkdic.orig/debian/rules skkdic/debian/rules
--- skkdic.orig/debian/rules    Fri Dec 15 03:36:49 2006
+++ skkdic/debian/rules Mon Dec 25 17:39:10 2006
@@ -2,6 +2,7 @@
 include /usr/share/cdbs/1/rules/debhelper.mk
 include /usr/share/cdbs/1/class/makefile.mk

+DEB_DH_INSTALLCHANGELOGS_ARGS := -k
 DEB_INSTALL_CHANGELOGS_ALL := ChangeLog
 DEB_INSTALL_DOCS_ALL := ChangeLog.* READMEs/committers.txt
```

以上で簡単なパッチの説明は終わりです。こういったパッチは、修正した部分を確認したり他人と修正内容をやりとりしたりするのに非常に便利で、開発作業はこれなしではやっていけないと言っても過言ではないでしょう。実際に使うに当たっては patch(1) や diff(1) のマニュアルを参照することをお勧めします。

1.1.2 パッチ管理ツールとは

今回説明する quilt はパッチ管理ツールです。これは、複数のパッチを管理するためのものです。どのような場合に必要になるのでしょう?

先程パッチとは「修正した部分を確認したり他人と修正内容をやりとりしたりするのに非常に便利」だと書きました。その言葉に基づけば、一般にはパッチとは使い捨ての一時ファイルで短寿命です。管理する必要はありません。

しかし、以下のようなケースを考えてみてください。

- ある団体がソフトウェア foo を公開し、定期的にリリースしている。
- A さんは foo を改変して使ったり再配布したりする必要がある。
- A さんの改変内容には複数の論理的に異なる変更が混ざっており、foo に取り込まれるものもあれば取り込まれないものもある。
  - foo のバグの修正 1 (小さな変更なので次のマイナーリリースで適用される。)
  - foo のバグの修正 2 (大きめの変更なので次のメジャーリリースまで適用されない。)
  - foo のバグの修正 3 (バグを回避するための A さん独自の次善策で、開発元からのリリースでは適用されない。)
  - :
  - foo のバグの修正 N
  - A さんの環境に合わせるための Makefile の修正

A さんが全体の変更を 1 つのパッチとして管理するのは不可能に近いでしょう。foo の新しいリリースが出たときに、元のパッチを当てても (うまく当たる部分もあるでしょうが) 間違いなくうまく当たりません。しかも複数の変更が混ざっているのでそれを解決するのは非常に大変でしょう。

変更を複数の論理的なパッチに分割すれば、少しは管理が楽になるでしょう。1 つずつパッチを当て、うまく当たるものと当たらないものを区別できます。当たらないもののうち、既に開発元で加えられた修正のパッチは削除し、残すべきなのに他の変更の影響で部分的にうまく当たらないパッチはうまく当たるよう調整します。それらの作業は面倒かもしれませんが、1 つのパッチとして管理するのに比べれば楽でしょう。ただし、作業中に、現在どのパッチが当たっていてどれが当たっていないかを一々覚えておかなければならないのが大変です。

そこで登場するのがパッチ管理ツールです。パッチ管理ツールは、複数のパッチの取り扱いを楽にするためのツールです。具体的には、どのパッチが当たっておりどれが当たっていないかを管理できます。また、一度に複数のパッ

チを当てたり外したりできるので、全ての (あるいはある一群の) パッチがうまく当たるかどうかを概観するのも楽にできます。A さんがやっているような作業にはうってつけのツールです。

さて、実は Debian パッケージのメンテナ (保守担当者) はこの A さんのような作業をする必要があります。すなわち、開発元のリリースに複数のパッチを独自に当て、それらを管理する必要があるのです。というのも、一般に開発元と Debian とではリリースサイクルや品質の基準が異なるからです。管理すべきパッチは、ソフトウェアの規模が大きくなればなるほど増えるでしょう。そこで Debian パッケージメンテナはパッチ管理ツールを使用することが推奨されています。

ここでは quilt を使用したパッチ管理を紹介します。Debian でよく使われるパッチ管理ツールには、dpatch というものもあります。こちらは Debian パッケージメンテナンスに特化したツールですが、機能的には quilt とそんなに違いはありません。興味がある方は、2005 年 7 月勉強会[*3]の資料を参照してください。

1.2 インストール

quilt は、Debian では `quilt` パッケージに含まれており、`apt-get` や `aptitude` で普通にインストールできます。

1.3 quilt の基本操作

1.3.1 最初に知っておくべきこと

quilt では一般に、`quilt <コマンド>`という書式で様々なコマンドが利用できます。利用可能なコマンドの一覧は次のようにして参照できます。

```
nori1[13:19]%  quilt --help                                          whale:~
使い方: quilt [--trace[=verbose]] [--quiltrc=XX] command [-h] ...
       quilt --version
コマンド一覧:
        add       files   import   previous  setup
        annotate  fold    mail     push      snapshot
        applied   fork    new      refresh   top
        delete    graph   next     remove    unapplied
        diff      grep    patches  rename    upgrade
        edit      header  pop      series

全コマンド共通オプション:

--trace
        コマンドを bash のトレースモード (-x) で実行。内部デバッグ用。

--quiltrc file
        ~/.quiltrc (存在しない場合は代わりに /etc/quiltrc) 以外のコン
        フィギュレーションファイルを指定。内容の詳細については PDF のド
        キュメントを参照。

--version
        バージョン情報を出力して終了。
```

また各コマンドのヘルプメッセージは各コマンドに`-h` オプションを指定すれば表示できます。以下の例では `quilt add` のヘルプメッセージを表示させています。

```
nori1[13:45]%  quilt add -h                                          whale:~
Usage: quilt add [-P patch] {file} ...

Add one or more files to the topmost or named patch.  Files must be
added to the patch before being modified.  Files that are modified by
patches already applied on top of the specified patch cannot be added.

-P patch
        Patch to add files to.
```

1.3.2 パッチスタック

実際に quilt を使う前に、quilt のデータ構造を知っておきましょう。quilt は、あるツリーに対する一連のパッチを一列に並べてスタックとして管理します。

[*3] `http://www.netfort.gr.jp/~dancer/column/2005-debianmeeting.html.ja#6th`

あるソースツリーに、以下のような 5 つのパッチを順に加えたいとします。

1. r1091-remove-trailing-garbage.diff
2. r1092-implement-distclean.diff
3. r1094-add-readme.diff
4. fix-ftbfs-on-m68k.diff
5. work-around-an-error-of-libtool.diff

このとき、これらのパッチは **patches ディレクトリ**に含まれていなければなりません。そして、パッチの整列順序を収めた次のような内容の **series ファイル**[*4]がパッチディレクトリに含まれている必要があります。

```
r1091-remove-trailing-garbage.diff
r1092-implement-distclean.diff
r1094-add-readme.diff
fix-ftbfs-on-m68k.diff
work-around-an-error-of-libtool.diff
```

つまりまとめると、上に挙げた 5 つのパッチを順に加えるようなパッチセットのデータは次のようなものです。

```
nori1[14:22]%  ls patches/*             whale:~/svnwc/deb/serf/trunk/serf-0.1.0
patches/fix-ftbfs-on-m68k.diff
patches/r1091-remove-trailing-garbage.diff
patches/r1092-implement-distclean.diff
patches/r1094-add-readme.diff
patches/series
patches/work-around-an-error-of-libtool.diff
nori1[14:22]%  cat patches/series       whale:~/svnwc/deb/serf/trunk/serf-0.1.0
r1091-remove-trailing-garbage.diff
r1092-implement-distclean.diff
r1094-add-readme.diff
fix-ftbfs-on-m68k.diff
work-around-an-error-of-libtool.diff
```

quilt のパッチデータはこれが全てです。

なおこのデータ構造は、1.3.4 で後述するパッチ作成作業時に quilt が自動的に作成してくれるので、通常は手でいじる必要はありません。次のセクションでは、このような patches ディレクトリを既にもったツリーでパッチを当てたり外したりしてみましょう。

1.3.3 パッチスタックの操作

さて、では quilt を使ってみます。まずはスタック管理のための操作を眺めてみましょう。スタック管理に必要な操作は、もちろん push と pop です。

最初は何もパッチが当たっていないとします。この状態で `quilt push` を実行すると、1 番目のパッチ、つまり `r1091-remove-trailing-garbage.diff` が当たります。このパッチは `Makefile.in` に修正を加えるものです。

```
nori1[14:30]%  quilt push               whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Applying patch r1091-remove-trailing-garbage.diff
patching file Makefile.in

Now at patch r1091-remove-trailing-garbage.diff
```

きちんと当たったようですね。`quilt push` は、引数を与えなければ「次のパッチ」を当てるコマンドです。これで現在ツリーは、`r1091-remove-trailing-garbage.diff` だけが当たった状態になりました。

もう一度 `quilt push` を実行します。

[*4] dpatch における `00list` と同じ役割を果たします。

```
nori1[15:08]%  quilt push              whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Applying patch r1092-implement-distclean.diff
patching file Makefile.in

Now at patch r1092-implement-distclean.diff
```

次のパッチ r1092-implement-distclean.diff も当たりました。これを「r1092-implement-distclean.diff が**一番上に来た**状態」と言うことにしましょう。スタック (stack) とは英語で「積み重ねたもの」の意味です。push 操作ではその上にもの (quilt ではパッチ) を次々に載せていくので、「パッチ○○まで当てられた状態」は、言い換えれば「パッチ○○が一番上に来た状態」です。

push の逆は pop です。すなわち現在一番上にあるパッチを外すのは quilt pop です。

```
nori1[15:15]%  quilt pop               whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Removing patch r1092-implement-distclean.diff
Restoring Makefile.in

Now at patch r1091-remove-trailing-garbage.diff
nori1[15:15]%  quilt pop               whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Removing patch r1091-remove-trailing-garbage.diff
Restoring Makefile.in

No patches applied
```

quilt push や quilt pop に引数としてパッチ名を与えると、そのパッチが一番上に来た状態になるように一連のパッチを当てたり外したりします。また引数として数値を与えると、その数だけパッチを当てたり外したりできます。-a をオプションとして指定すると全てのパッチを当てたり外したりできます。

```
nori1[15:17]%  quilt push 3            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Applying patch r1091-remove-trailing-garbage.diff
patching file Makefile.in

Applying patch r1092-implement-distclean.diff
patching file Makefile.in

Applying patch r1094-add-readme.diff
patching file README

Now at patch r1094-add-readme.diff
nori1[15:17]%  quilt pop r1091-remove-trailing-garbage.diff
Removing patch r1094-add-readme.diff
Removing README

Removing patch r1092-implement-distclean.diff
Restoring Makefile.in

Now at patch r1091-remove-trailing-garbage.diff
nori1[15:18]%  quilt push -a           whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Applying patch r1092-implement-distclean.diff
patching file Makefile.in

Applying patch r1094-add-readme.diff
patching file README

Applying patch fix-ftbfs-on-m68k.diff
patching file Makefile.in

Applying patch work-around-an-error-of-libtool.diff
patching file Makefile.in

Now at patch work-around-an-error-of-libtool.diff
nori1[15:18]%  quilt pop -a            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Removing patch work-around-an-error-of-libtool.diff
Restoring Makefile.in

Removing patch fix-ftbfs-on-m68k.diff
Restoring Makefile.in

Removing patch r1094-add-readme.diff
Removing README

Removing patch r1092-implement-distclean.diff
Restoring Makefile.in

Removing patch r1091-remove-trailing-garbage.diff
Restoring Makefile.in

No patches applied
```

現在一番上にあるパッチは quilt top で確認できます。現在一番上にあるパッチの次のパッチと前のパッチはそれぞれ、quilt next と quilt previous で確認できます。現在当てられているパッチと当てられていないパッチはそ

れぞれ、quilt applied と quilt unapplied で確認できます。そして、全てのパッチ、つまり series ファイルの内容は quilt series で確認できます。

```
nori1[15:31]%  quilt top                whale:~/svnwc/deb/serf/trunk/serf-0.1.0
No patches applied
nori1[15:31]%  quilt next               whale:~/svnwc/deb/serf/trunk/serf-0.1.0
r1091-remove-trailing-garbage.diff
nori1[15:31]%  quilt previous           whale:~/svnwc/deb/serf/trunk/serf-0.1.0
No patches applied
nori1[15:31]%  quilt applied            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
No patches applied
nori1[15:31]%  quilt unapplied          whale:~/svnwc/deb/serf/trunk/serf-0.1.0
r1091-remove-trailing-garbage.diff
r1092-implement-distclean.diff
r1094-add-readme.diff
fix-ftbfs-on-m68k.diff
work-around-an-error-of-libtool.diff
nori1[15:32]%  quilt push 2             whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Applying patch r1091-remove-trailing-garbage.diff
patching file Makefile.in

Applying patch r1092-implement-distclean.diff
patching file Makefile.in

Now at patch r1092-implement-distclean.diff
nori1[15:32]%  quilt top                whale:~/svnwc/deb/serf/trunk/serf-0.1.0
r1092-implement-distclean.diff
nori1[15:32]%  quilt next               whale:~/svnwc/deb/serf/trunk/serf-0.1.0
r1094-add-readme.diff
nori1[15:32]%  quilt previous           whale:~/svnwc/deb/serf/trunk/serf-0.1.0
r1091-remove-trailing-garbage.diff
nori1[15:32]%  quilt applied            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
r1091-remove-trailing-garbage.diff
r1092-implement-distclean.diff
nori1[15:33]%  quilt unapplied          whale:~/svnwc/deb/serf/trunk/serf-0.1.0
r1094-add-readme.diff
fix-ftbfs-on-m68k.diff
work-around-an-error-of-libtool.diff
nori1[15:33]%  quilt series             whale:~/svnwc/deb/serf/trunk/serf-0.1.0
r1091-remove-trailing-garbage.diff
r1092-implement-distclean.diff
r1094-add-readme.diff
fix-ftbfs-on-m68k.diff
work-around-an-error-of-libtool.diff
nori1[15:33]%  quilt pop -a             whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Removing patch r1092-implement-distclean.diff
Restoring Makefile.in

Removing patch r1091-remove-trailing-garbage.diff
Restoring Makefile.in

No patches applied
```

ある特定のファイルを変更するパッチの一覧は quilt files で参照可能です。逆に、ある特定のパッチが変更するファイルの一覧は quilt patches で参照可能です。

```
nori1[15:35]%  quilt files r1091-remove-trailing-garbage.diff
Patch r1091-remove-trailing-garbage.diff is not applied
nori1[15:35]%  quilt push r1091-remove-trailing-garbage.diff
Applying patch r1091-remove-trailing-garbage.diff
patching file Makefile.in

Now at patch r1091-remove-trailing-garbage.diff
nori1[15:35]%  quilt files              whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Makefile.in
nori1[15:36]%  quilt files r1091-remove-trailing-garbage.diff
Makefile.in
nori1[15:36]%  quilt pop -a             whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Removing patch r1091-remove-trailing-garbage.diff
Restoring Makefile.in

No patches applied
nori1[15:36]%  quilt patches Makefile.in
r1091-remove-trailing-garbage.diff
r1092-implement-distclean.diff
fix-ftbfs-on-m68k.diff
work-around-an-error-of-libtool.diff
```

これでパッチスタックの操作の説明は終わりです。パッチを含んだ patches ディレクトリが存在していれば、その中のパッチを自由自在に当てたり外したり、現在当たっているパッチを調べたりできるようになりました。しかしこれだけでは何の役にも立ちません。次のセクションではパッチを新たに追加する方法について説明します。

1.3.4 パッチの追加

いよいよ patches ディレクトリに新たなパッチを追加します。quilt では `quilt new` で新たなパッチに名前をつけて作成を開始し、適当な変更を加えた後、`quilt refresh` でパッチとして保存します。

新たなパッチを追加する場合、まずはパッチをスタックのどこに入れるか決めます[*5]。今回は (特に意味はありませんが) 1 つ目のパッチである `r1091-remove-trailing-garbage.diff` の後に入れましょう。パッチの名前は `love-debian.diff` とし、その内容は、次のような love-debian ターゲットを `Makefile.in` に加えるものにしましょう。

```
love-debian:
        echo "I love debian!!"
```

では作成を開始します。

```
nori1[17:29]%  quilt push r1091-remove-trailing-garbage.diff
Applying patch r1091-remove-trailing-garbage.diff
patching file Makefile.in

Now at patch r1091-remove-trailing-garbage.diff
nori1[17:29]%  quilt new love-debian.diff
Patch love-debian.diff is now on top
```

早速 `Makefile.in` を編集したいところですが、少し待ってください。実は quilt では、パッチに含めるファイルは予め `quilt add` で追加しておかなければなりません。

```
nori1[17:29]%  quilt add Makefile.in    whale:~/svnwc/deb/serf/trunk/serf-0.1.0
File Makefile.in added to patch love-debian.diff
```

その上でファイルの編集をし、その内容を `quilt diff` で確認します。

```
nori1[17:35]%  vim Makefile.in          whale:~/svnwc/deb/serf/trunk/serf-0.1.0
[snip]
nori1[17:38]%  quilt diff               whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Index: serf-0.1.0/Makefile.in
===================================================================
--- serf-0.1.0.orig/Makefile.in        2007-01-18 17:35:11.000000000 +0900
+++ serf-0.1.0/Makefile.in        2007-01-18 17:38:38.000000000 +0900
@@ -99,6 +99,9 @@
                $(INSTALL) -m 644 $$i $(DESTDIR)$(includedir); \
         done

+love-debian:
+        echo "I love debian!!"
+
 clean:
         rm -f $(TARGET_LIB) $(OBJECTS) $(OBJECTS:.lo=.o) $(PROGRAMS) $(TEST_OBJECTS) $(TEST_OBJECTS:.lo=.o)
```

最後に `quilt refresh` で保存します。

```
nori1[17:39]%  quilt refresh            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Refreshed patch love-debian.diff
```

さて、実は `love-debian.diff` は 1 番目のパッチの後に挿入しました。ということは、残りのパッチがうまく当たるか確認しておかなければなりません。

[*5] 1.4 にパッチの並べ方に関する簡単なアドバイスがあります。

```
nori1[17:40]%  quilt push -a            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Applying patch r1092-implement-distclean.diff
patching file Makefile.in
Hunk \#1 succeeded at 104 (offset 3 lines).

Applying patch r1094-add-readme.diff
patching file README

Applying patch fix-ftbfs-on-m68k.diff
patching file Makefile.in

Applying patch work-around-an-error-of-libtool.diff
patching file Makefile.in

Now at patch work-around-an-error-of-libtool.diff
```

新たな行を挿入したので「`Hunk #1 succeeded at 104 (offset 3 lines).`」というメッセージが出ましたが、問題なく当たりました。

`quilt series` を実行すると、きちんと `love-debian.diff` が適切な位置に挿入されていることがわかります。

```
nori1[17:41]%  quilt series             whale:~/svnwc/deb/serf/trunk/serf-0.1.0
r1091-remove-trailing-garbage.diff
love-debian.diff
r1092-implement-distclean.diff
r1094-add-readme.diff
fix-ftbfs-on-m68k.diff
work-around-an-error-of-libtool.diff
nori1[17:41]%  quilt pop -a             whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Removing patch work-around-an-error-of-libtool.diff
Restoring Makefile.in

Removing patch fix-ftbfs-on-m68k.diff
Restoring Makefile.in

Removing patch r1094-add-readme.diff
Removing README

Removing patch r1092-implement-distclean.diff
Restoring Makefile.in

Removing patch love-debian.diff
Restoring Makefile.in

Removing patch r1091-remove-trailing-garbage.diff
Restoring Makefile.in

No patches applied
```

さて、パッチを作成した後になって気付いたのですが、love-debian ターゲットが表示する文字列の中の「debian」は「Debian」と大文字にした方がよさそうです。このようなときはパッチを改変しましょう。`quilt push` ないし `quilt pop` で目的のパッチ (`love-debian.diff`) を一番上に持ってきて、エディタで `Makefile.in` に変更を施した上で `quilt refresh` で保存すれば OK です。`Makefile.in` は既にパッチ `love-debian.diff` の変更対象に含まれているので、エディタでの編集前に `quilt add` で追加する必要はありません。

```
nori1[17:54]%  quilt push love-debian.diff
Applying patch r1091-remove-trailing-garbage.diff
patching file Makefile.in

Applying patch love-debian.diff
patching file Makefile.in

Now at patch love-debian.diff
nori1[17:59]%  vim Makefile.in          whale:~/svnwc/deb/serf/trunk/serf-0.1.0
[snip]
nori1[18:01]%  quilt diff               whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Index: serf-0.1.0/Makefile.in
===================================================================
--- serf-0.1.0.orig/Makefile.in        2007-01-18 17:38:38.000000000 +0900
+++ serf-0.1.0/Makefile.in        2007-01-18 18:01:35.000000000 +0900
@@ -99,6 +99,9 @@
                $(INSTALL) -m 644 $$i $(DESTDIR)$(includedir); \
         done

+love-debian:
+        echo "I love Debian!!"
+
 clean:
         rm -f $(TARGET_LIB) $(OBJECTS) $(OBJECTS:.lo=.o) $(PROGRAMS) $(TEST_OBJECTS) $(TEST_OBJECTS:.lo=.o)

nori1[18:01]%  quilt refresh            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Refreshed patch love-debian.diff
```

さて、このパッチを眺めているうちにまた気が変わって、design-guide.txt という別のファイルにも変更を加え、このパッチに含めようと思い立ちました。design-guide.txt は今のところ love-debian.diff の変更対象外なので、今度は、編集する前に quilt add で変更対象に追加しておかなければなりません。quilt add design-guide.txt を実行した上でエディタで design-guide.txt を開いてもよいのですが、面倒なのでここでは quilt edit コマンドを実行しましょう。このコマンドは、「引数のファイルを quilt add で追加した上で、環境変数 EDITOR で指定されたエディタでそのファイルを開く」という、2 つの操作をまとめて実行するためのものです。

```
nori1[18:19]%  quilt edit design-guide.txt
[snip]
nori1[18:21]%  quilt diff               whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Index: serf-0.1.0/Makefile.in
===================================================================
--- serf-0.1.0.orig/Makefile.in        2007-01-18 17:55:20.000000000 +0900
+++ serf-0.1.0/Makefile.in        2007-01-18 18:01:35.000000000 +0900
@@ -99,6 +99,9 @@
                $(INSTALL) -m 644 $$i $(DESTDIR)$(includedir); \
         done

+love-debian:
+        echo "I love Debian!!"
+
 clean:
         rm -f $(TARGET_LIB) $(OBJECTS) $(OBJECTS:.lo=.o) $(PROGRAMS) $(TEST_OBJECTS) $(TEST_OBJECTS:.lo=.o)

Index: serf-0.1.0/design-guide.txt
===================================================================
--- serf-0.1.0.orig/design-guide.txt        2007-01-18 18:19:30.000000000 +0900
+++ serf-0.1.0/design-guide.txt        2007-01-18 18:20:40.000000000 +0900
@@ -9,6 +9,7 @@
   4. Bucket Read Functions
   5. Versioning
   6. Bucket lifetimes
+  7. Love Debian


 ------------------------------------------------------------------------------
@@ -150,3 +151,10 @@


 ------------------------------------------------------------------------------
+
+7. LOVE DEBIAN
+
+Hey, please love Debian.
+
+
+------------------------------------------------------------------------------
nori1[18:22]%  quilt refresh            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Refreshed patch love-debian.diff
```

これで新たなパッチ love-debian.diff の内容は完成です。しかしこれだけだと後で何をしたいパッチだかわからなくなりそうです。最後に、パッチの説明を加えておきましょう。quilt では、パッチの内容を説明するヘッダを操作するコマンドは quilt header です。-e オプションを指定してコマンドを実行すると、環境変数 EDITOR のエディタでパッチのヘッダを編集できます。また、何もオプションを与えずに実行するとヘッダを表示できます。

```
nori1[18:37]%  quilt header             whale:~/svnwc/deb/serf/trunk/serf-0.1.0
nori1[18:37]%  quilt header -e          whale:~/svnwc/deb/serf/trunk/serf-0.1.0
[snip]
nori1[18:41]%  quilt header             whale:~/svnwc/deb/serf/trunk/serf-0.1.0
A patch to show my love for Debian.
```

本セクションでは、新たにパッチを作成する手順を学びました。これで自由にパッチを追加・編集できます。次のセクションでは、開発元が加えた変更への対応方法を学びます。

1.3.5 開発元が加えた変更への対応

パッチをいじれるようになったところで、開発元が新しいリリースなどで加えた変更に対応してみましょう。この作業では、quilt push の-f オプションが役立ちます。

開発元が加えた変更をマージする前に、まずやっておかなければならないことがあります。それは、ツリーにパッチが適用されていない状態にすることです。既に説明したように、適用されているパッチを全て外すには quilt pop -a を実行します。

```
nori1[20:00]%  quilt pop -a            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Removing patch love-debian.diff
Restoring Makefile.in
Restoring design-guide.txt

Removing patch r1091-remove-trailing-garbage.diff
Restoring Makefile.in

No patches applied
```

その上で、ツリーのファイルを、開発元が新しくリリースしたものに差し替えます (つまり開発元が加えた変更をマージします)。ここでは以下のような変更が入ったとします。

- Makefile.in への love ターゲットと clean-love ターゲットの追加 (love-debian.diff が変更する部分と同じ部分を改変するので、競合)
- README ファイルの追加 (r1094-add-readme.diff の修正内容そのもの)

さあ、新しいツリーにパッチを当てようとするとどうなるでしょうか。とりあえず quilt push -a で一度に全て当てようとしてみましょう。

```
nori1[20:32]%  quilt push -a           whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Applying patch r1091-remove-trailing-garbage.diff
patching file Makefile.in
Hunk #1 succeeded at 103 with fuzz 2 (offset 3 lines).

Applying patch love-debian.diff
patching file Makefile.in
Hunk #1 FAILED at 99.
1 out of 1 hunk FAILED -- rejects in file Makefile.in
patching file design-guide.txt
Patch love-debian.diff does not apply (enforce with -f)
nori1[20:32]%  quilt top               whale:~/svnwc/deb/serf/trunk/serf-0.1.0
r1091-remove-trailing-garbage.diff
```

love-debian.diff を当てるのに失敗して、そこでパッチの適用が止まってしまいました。そこで、-f オプションを quilt push につけて、love-debian.diff を強制的に適用してみます。

```
nori1[20:33]%  quilt push -f           whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Applying patch love-debian.diff
patching file Makefile.in
Hunk #1 FAILED at 99.
1 out of 1 hunk FAILED -- saving rejects to file Makefile.in.rej
patching file design-guide.txt
Applied patch love-debian.diff (forced; needs refresh)
```

Makefile.in がどうなったか眺めてみると、先程 love-debian ターゲットを加えた箇所に、次のように love ターゲットと clean-love ターゲットが追加されています。

```
nori1[20:37]%  cat Makefile.in         whale:~/svnwc/deb/serf/trunk/serf-0.1.0
[snip]
                $(INSTALL) -m 644 $$i $(DESTDIR)$(includedir); \
        done
love:
        echo "Making love..."
clean-love:
        echo "Clearing my old love..."
clean:
        rm -f $(TARGET_LIB) $(OBJECTS) $(OBJECTS:.lo=.o) $(PROGRAMS) $(TEST_OBJECTS) $(TEST_OBJECTS:.lo=.o)
[snip]
```

quilt push -f からのメッセージにあるように、パッチ適用の拒否の理由は Makefile.in.rej に記されています。念のため覗いてみましょう。

```
***************
*** 99,104 ****
                $(INSTALL) -m 644 $$i $(DESTDIR)$(includedir); \
          done

  clean:
          rm -f $(TARGET_LIB) $(OBJECTS) $(OBJECTS:.lo=.o) $(PROGRAMS) $(TEST_OBJECTS) $(TEST_OBJECTS:.lo=.o)

--- 99,107 ----
                $(INSTALL) -m 644 $$i $(DESTDIR)$(includedir); \
          done

+ love-debian:
+         echo "I love Debian!!"
+
  clean:
          rm -f $(TARGET_LIB) $(OBJECTS) $(OBJECTS:.lo=.o) $(PROGRAMS) $(TEST_OBJECTS) $(TEST_OBJECTS:.lo=.o)
```

これは **context diff** と呼ばれる形式ですが、意味が掴めればかまいません。上で説明したように、love-debian ターゲットを加えるための、clean ターゲットの前の 3 行がうまく当たらなかったことがわかるでしょう。

強制的にパッチを適用した結果がどうなったかわかったところで、削除されてしまった大切な love-debian ターゲットを `Makefile.in` に再度追加した上で、`quilt push -f` からのメッセージにあるようにパッチを更新 (refresh) しましょう。

```
nori1[20:45]%  vim Makefile.in          whale:~/svnwc/deb/serf/trunk/serf-0.1.0
[snip]
nori1[20:54]%  quilt diff               whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Index: serf-0.1.0/Makefile.in
===================================================================
--- serf-0.1.0.orig/Makefile.in        2007-01-18 20:32:13.000000000 +0900
+++ serf-0.1.0/Makefile.in        2007-01-18 20:52:22.000000000 +0900
@@ -102,6 +102,10 @@
          echo "Making love..."
 clean-love:
          echo "Clearing my old love..."
+
+love-debian:
+         echo "I love Debian!!"
+
 clean:
          rm -f $(TARGET_LIB) $(OBJECTS) $(OBJECTS:.lo=.o) $(PROGRAMS) $(TEST_OBJECTS) $(TEST_OBJECTS:.lo=.o)

Index: serf-0.1.0/design-guide.txt
===================================================================
--- serf-0.1.0.orig/design-guide.txt        2007-01-18 20:31:56.000000000 +0900
+++ serf-0.1.0/design-guide.txt        2007-01-18 20:36:10.000000000 +0900
@@ -9,6 +9,7 @@
   4. Bucket Read Functions
   5. Versioning
   6. Bucket lifetimes
+  7. Love Debian


 ----------------------------------------------------------------------------
@@ -150,3 +151,10 @@


 ----------------------------------------------------------------------------
+
+7. LOVE DEBIAN
+
+Hey, please love Debian.
+
+
+----------------------------------------------------------------------------
nori1[20:54]%  quilt refresh            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Refreshed patch love-debian.diff
```

さあ、これで `love-debian.diff` の問題は解決しました。再び `quilt push -a` で残りのパッチを全て当てようとしてみましょう。

```
nori1[20:55]%  quilt push -a            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Applying patch r1092-implement-distclean.diff
patching file Makefile.in
Hunk #1 succeeded at 108 (offset 7 lines).

Applying patch r1094-add-readme.diff
The next patch would create the file README,
which already exists!  Applying it anyway.
patching file README
Patch attempted to create file README, which already exists.
Hunk #1 FAILED at 1.
1 out of 1 hunk FAILED -- rejects in file README
Patch r1094-add-readme.diff can be reverse-applied
```

今度は r1094-add-readme.diff を当てるのに失敗して、そこでパッチの適用が止まってしまいました。メッセージによれば、このパッチは逆に当てることができます。それはすなわち、このパッチが既に当たった状態であることを意味します。開発元で README を追加してくれたので、README を追加するための r1094-add-readme.diff は不要になったのでした。そこで、このパッチは削除することにします。

```
nori1[21:05]%  quilt delete r1094-add-readme.diff
Removed patch r1094-add-readme.diff
```

これで r1094-add-readme.diff の問題も解決です。再び quilt push -a で残りのパッチを全て当てようとしてみましょう。

```
nori1[21:05]%  quilt push -a            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Applying patch fix-ftbfs-on-m68k.diff
patching file Makefile.in

Applying patch work-around-an-error-of-libtool.diff
patching file Makefile.in

Now at patch work-around-an-error-of-libtool.diff
```

全てうまく当たりました。これで一連のパッチはうまく当たるようになりました。

```
nori1[21:06]%  quilt pop -a             whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Removing patch work-around-an-error-of-libtool.diff
Restoring Makefile.in

Removing patch fix-ftbfs-on-m68k.diff
Restoring Makefile.in

Removing patch r1092-implement-distclean.diff
Restoring Makefile.in

Removing patch love-debian.diff
Restoring Makefile.in
Restoring design-guide.txt

Removing patch r1091-remove-trailing-garbage.diff
Restoring Makefile.in

No patches applied
```

1.3.6 パッチの依存関係の可視化

quilt の quilt graph コマンドと Graphviz を使うと、適用されたパッチの依存関係を可視化できます。

最初に、全てのパッチを当てておきます。

```
nori1[22:18]%  quilt push -a            whale:~/svnwc/deb/serf/trunk/serf-0.1.0
Applying patch r1091-remove-trailing-garbage.diff
patching file Makefile.in
Hunk #1 succeeded at 103 with fuzz 2 (offset 3 lines).

Applying patch love-debian.diff
patching file Makefile.in
patching file design-guide.txt

Applying patch r1092-implement-distclean.diff
patching file Makefile.in
Hunk #1 succeeded at 108 (offset 7 lines).

Applying patch fix-ftbfs-on-m68k.diff
patching file Makefile.in

Applying patch work-around-an-error-of-libtool.diff
patching file Makefile.in

Now at patch work-around-an-error-of-libtool.diff
```

この状態で何も与えずに quilt graph を実行すると、Graphviz 用のソースファイルが出力されます。これは、同じファイルを変更する複数のパッチの間に依存関係がある、として計算した結果です。

```
nori1[22:18]%  quilt graph              whale:~/svnwc/deb/serf/trunk/serf-0.1.0
digraph dependencies {
        n0 [label="r1091-remove-trailing-garbage.diff"];
        n1 [label="love-debian.diff"];
        n2 [label="r1092-implement-distclean.diff"];
        n3 [label="fix-ftbfs-on-m68k.diff"];
        n4 [style=bold,label="work-around-an-error-of-libtool.diff"];
        n0 -> n1 [len="1.39"];
        n1 -> n2 [len="1.39"];
        n2 -> n3 [len="1.39"];
        n3 -> n4 [len="1.39"];
}
```

このソースを例えば graph.dot というファイルに保存すると、Graphviz を使って (例えば dot -Tpng graph.dot -o graph.png などと実行して) 様々な形式のファイルが生成できます。しかし、ps を出力するのであれば次のように直接出力するのが楽でしょう。

```
nori1[22:35]%  quilt graph -T ps > patchdep-1.eps
```

なお、依存関係の指定はありませんが、内部的に (/usr/share/quilt/graph で) Graphviz の dot コマンドを呼び出しているので、実行には graphviz パッケージがインストールされている必要があります (Bug#407469[*6])。

生成された図は図 1 のようになります。

他方で、同じファイルを変更するだけでなくその変更領域が被っている場合に依存関係がある、として計算すると次のようになります。変更領域の被りが 1 行の場合 (図 2) と 2 行 (図 3) の場合です。

```
nori1[22:44]%  quilt graph --lines=1 -T ps > patchdep-2.eps
nori1[22:44]%  quilt graph --lines=2 -T ps > patchdep-3.eps
```

1.3.7 環境変数

quilt では以下のような環境変数を指定できます。ここでは省略していますが、他にも diff 関連の環境変数がいくつかあります。

QUILT_PATCHES　patches ディレクトリを指定します。指定されていない場合は patches がパッチファイル (および series ファイル) の格納に使用されます。

QUILT_PC　.pc ディレクトリを指定します。.pc ディレクトリは、どのパッチが当てられたかを管理するためのもので、指定されていない場合は.pc となります。

[*6] http://bugs.debian.org/407469

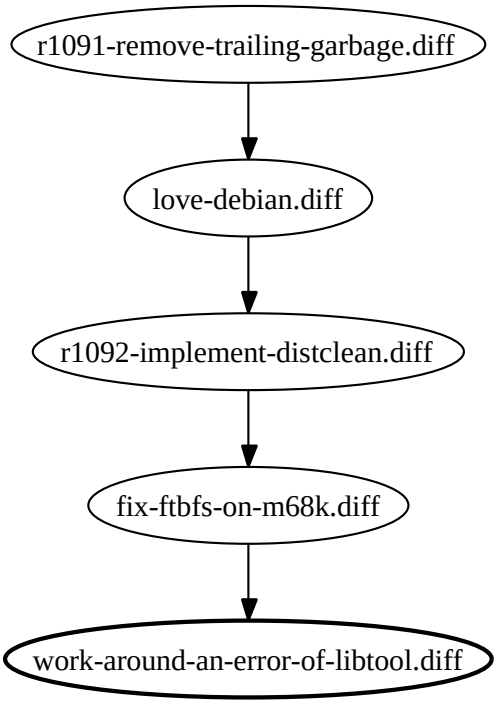


図 1　同じファイルを変更する複数のパッチの間に依存関係がある、として計算した依存関係

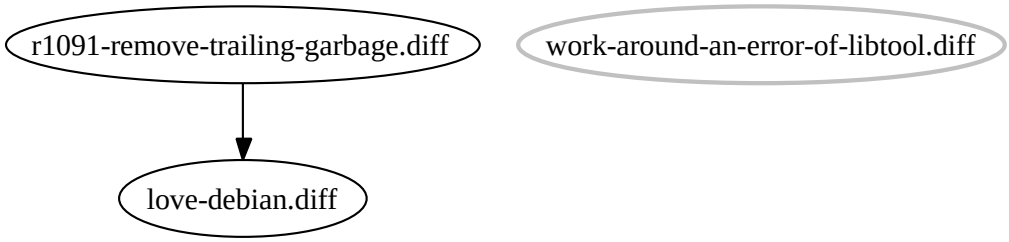


図 2　変更領域が 1 行被っている場合に依存関係がある、として計算した依存関係

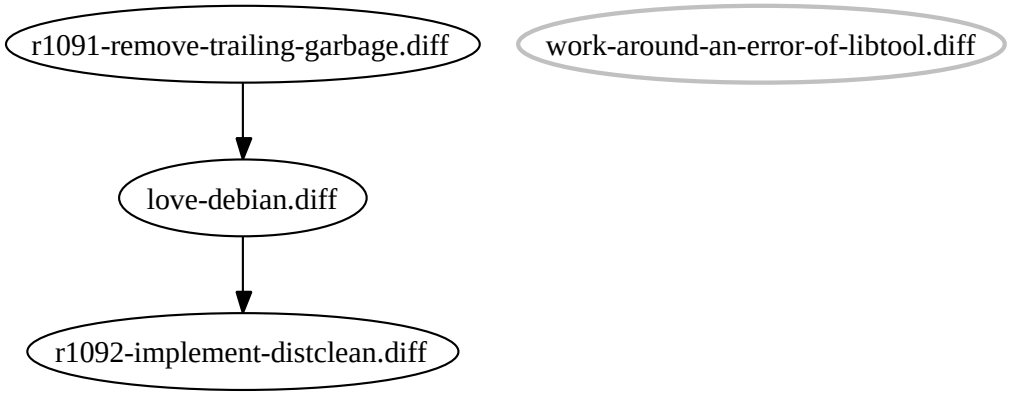


図 3　変更領域が 2 行被っている場合に依存関係がある、として計算した依存関係

1.4 Debian パッケージ作成時の quilt の使用

Debian パッケージ作成時のパッチの管理に quilt を使用するときは、以下を参考にしてください。

- quilt はデフォルトではパッチファイルを patches ディレクトリに入れます。しかし、Debian パッケージ作成用のパッチは、パッケージ作成に用いる他のファイルと同様 `debian` ディレクトリの下にまとめて入れておくことが推奨されています。1.3.7 で説明した環境変数 `QUILT_PATCHES` に `debian/patches` を設定しておくとよいでしょう。
- cdbs では quilt をパッチ管理に使用するのに便利なルールが用意されています。使用する場合は `debian/rules` 内で`/usr/share/cdbs/1/rules/patchsys-quilt.mk` を include してください。なお、cdbs のこのクラスは `patches` に `debian/patches` への symlink を作成します。
- やや一般的な話ですが、パッチを並べる順序は開発元に近い順にするとよいでしょう。つまり、開発元に既に取り込まれている変更が最初に適用され、開発元には取り込まれることのない Debian 独自の変更は最後に適用されるようにすることをお勧めします。

2 dpatch についての小ネタ

上川 純一

dpatch を subversion などといっしょに使うときにもしかすると便利かもしれない小ネタについて御紹介します。

2.1 インストールのしかた

まず最初にインストールのしかたですが、Debian システムであれば、

```
apt-get install dpatch
```

でインストールできます。そうでない場合のインストールについては想定外です。

2.2 作業のしかた

debian/rules を適切に変更したあと、dpatch-edit-patch で編集します。[*7]

パッチは debian/patches 以下で管理されます。debian/patches/00list ファイルに適用するパッチの一覧があり、それを参照して パッチを適用します。

2.3 dpatch に含まれる各種ツールの紹介

各種ツールがあるので何をするものなのか、見てみましょう。

dpatch　メインのツールです。表立って直接利用することはありません。debian/rules などから呼び出されます。

dpatch-list-patch　パッチの一覧を出力します。

dpatch-edit-patch　編集のツールです。パッチを作成、もしくは編集する際に利用します。dpatch-edit-patch **パッチ名 適用するパッチ** という形で指定すると、指定した適用するパッチまでが適用された状態のソースツリーが一時ディレクトリに展開された状態でシェルが起動します。そこで編集し、シェルを終了 (ctrl-D もしくは exit) するとパッチが作成され、debian/patches 以下にファイルが生成されます。

dpatch-convert-diffgz　.diff.gz から dpatch ファイルを生成するためのツールです。

dpatch-get-origtargz　dpatch-edit-patch が内部的に利用するツール。

2.4 debian/ディレクトリのみを展開したパッケージのメンテナンスをする

dpatch で管理すると、Debian のソースパッケージのうち、debian/以下のディレクトリに対しての変更しか発生しなくなります。.orig.tar.gz を展開した状態でのパッケージのメンテナンスは実は必要なく、debian/ 以下だけをバージョン管理ツールで管理して、orig.tar.gz は必要に応じてアップストリームからダウンロードしてくるというスタイルで開発をすることができます。

[*7] /usr/share/doc/dpatch/examples/rules/rules.dh.gz 参照

dpatch-get-origtargz はそのためのツールで、現在の debian/ ディレクトリに対応した .orig.tar.gz を環境変数 DPGO_ORIGTARDIR で指定したパス上に存在する .orig.tar.gz から探してきます。ローカルで見付からない場合には uscan や apt-get source などを活用して、ネットワークからダウンロードしてきてくれます。

```
[12:13:55]dancer64:ecatest> ls -l
合計 0
drwxr-xr-x 5 dancer dancer 700 2007-02-17 12:09 debian
[12:12:59]dancer64:ecatest> dpatch-edit-patch -b 12_users_guide.dpatch
dpatch-edit-patch: * /tmp/ecatest/debian/patches/12_users_guide.dpatch exists, this patch will be updated.
dpatch-edit-patch: * debian/-only layout selected
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています... 完了
1129kB のソースアーカイブを取得する必要があります。
取得:1 http://glantank sid/main ecasound2.2 2.4.4-6 (tar) [1129kB]
1129kB を 0s で取得しました (13.2MB/s)
dpatch-edit-patch: * unpacking ecasound2.2_2.4.4.orig.tar.gz
dpatch-edit-patch: * Copying /tmp/ecatest to reference directory.
dpatch-edit-patch: * Cleaning /tmp/dpep-ref.Nn8155/ecatest
dpatch  deapply-all
12_users_guide not applied to ./ .
11_configure_in_alsa_fix not applied to ./ .
07_configure_in_maintainer_mode not applied to ./ .
rm -rf patch-stamp patch-stampT debian/patched
dh_testdir
dh_testroot
rm -f build-stamp configure-stamp
# Add here commands to clean up after the build process.

[中略]
12_users_guide not applied to ./ .
11_configure_in_alsa_fix not applied to ./ .
07_configure_in_maintainer_mode not applied to ./ .
dpatch-edit-patch: * Applying patches
applying patch 07_configure_in_maintainer_mode to ./ ... ok.
applying patch 11_configure_in_alsa_fix to ./ ... ok.
dpatch-edit-patch: * Copying reference directory /tmp/dpep-ref.Nn8155/ecatest to work directory.
dpatch-edit-patch: * Applying current 12_users_guide.dpatch for editing.
applying patch 12_users_guide to ./ ... ok.

dpatch-edit-patch:

Now launching an interactive shell in your work directory. Edit your files.
When you are done, exit the shell. When you exit the shell, your patch will be
automatically updated based on the changes in your work directory.

If you wish to abort the process, exit the shell such that it returns an exit
code of "230". This is typically done by exiting the shell with the command
'exit 230'.
[12:13:09]dancer64:ecatest>
```

2.5 svn-buildpackage との連携

svn-buildpackage と dpatch の連携については特に考慮されているわけではありません。ただ、連携して利用できないわけではありません。利用するには下記の点を設定すればよいでしょう。

svn-buildpackage も、 debian/ ディレクトリのみを別管理にする方法をサポートしています。それを利用する際に注意する点として、 orig.tar.gz の場所を指示するために、`~/.svn-buildpackage.conf` に次のような行をいれます。

```
svn-override=origDir=$HOME/DEBIAN/svn/tarballs
```

この場所から svn-buildpackage は .orig.tar.gz を探してくれます。

この設定にあわせて、dpatch も同じ場所から orig.tar.gz を探すように環境変数を設定します。

```
DPGO_ORIGTARDIR=$HOME/DEBIAN/svn/tarballs
```

すると、svn-buildpackage と dpatch が連動して動作するようになります。

`-b` オプションを付けて `dpatch-edit-patch` を実行すると、orig.tar.gz を `DPEP_GETORIGTARGZ` を参照して探してくれるようになります。

2.6 参考文献

- 「dpatch をつかってみよう」 2005 年 7 月 Debian 勉強会資料 (あんどきゅめんてっどでびあん 2005 年夏号収録)
- svn-buildpackage HOWTO `/usr/share/doc/svn-buildpackage/HOWTO.pdf`
- svn-buildpackage のメモ `http://tach.arege.net/trac/wiki/Debian/svn-buildpackage`
- svn-buildpackage で .deb パッケージのバージョン管理 `http://www.j96.org/~yuya/d/20041128.html#p02`

3 dbs

岩松 信洋

3.1 dbs とはなにか

dbs は Debian Build System の略です。dpatch や quilt はパッチを管理する方法に重点を置いているのですが、dbs は名前のとおり、ビルドまでの面倒を見るためのツールです。使いかたとしては、dbs も dpatch もあまり変わりません。debian/rules で専用のライブラリを include して使うだけです。ただ、作法がありところどころ違うところもあります。今回は dpatch と比べてどこが違うのかを比べてみようと思います。

3.2 インストールのしかた

```
# apt-get install dbs
```

3.3 使いかた

dbs の使うためのサンプルとして hello-dbs[*8] というパッケージが存在します。これを使ってどのように dbs を使うのか、説明したいと思います。

3.3.1 hello-dbs を取得

hello-dbs のソースパッケージを取得します。

```
% apt-get source hello-dbs
```

3.3.2 展開されたソースパッケージ

展開されたソースパッケージ内をみると、以下のような構成になっています。

```
% ls
debian  hello-1.3.tar.gz
```

ここでわかる dbs を使ったパッケージの特徴として、ソースパッケージは tar.gz 形式で提供されている点です。dh_make を使って生成されたパッケージではこのような構成にはなっていません。upstream のソースパッケージは Debian 標準の形ではないということです。

しかし、debian ディレクトリは、他のパッケージの構成とあまり変わりません。

```
% ls debian/
README.Debian  README.build  changelog  compat  control  copyright  dirs  hello.1  patches  rules
```

[*8] http://packages.debian.org/unstable/devel/hello-dbs

3.3.3 debian/rules ファイル

dbs では debian/rules に設定項目を書くことによって、細かい設定を行うことができます。hello-dbs では以下の設定を行っています。

```
# DBS options

package         := hello-dbs
PWD             := $(shell pwd)
CFLAGS          := -O2 -Wall
INSTALL = install
INSTALL_DATA    := $(INSTALL) -m644
INSTALL_DIR     := $(INSTALL) -p -d -o root -g root  -m  755
INSTALL_FILE    := $(INSTALL) -p    -o root -g root  -m  644
INSTALL_PROGRAM := $(INSTALL) -m755
INSTALL_SCRIPT  := $(INSTALL) -p    -o root -g root  -m  755
SCRIPT_DIR     = /usr/share/dbs

# the dbs rules
TAR_DIR := hello-1.3.orig
include $(SCRIPT_DIR)/dbs-build.mk
```

以下に各設定項目の意味を示します。

- package
  パッケージ名
- PWD
  パッケージカレントディレクトリ
- CFLAGS
  C コンパイラに設定するオプション
- INSTALL_DATA
  インストールするデータのパーミッション
- INSTALL_DIR
  インストール先ディレクトリのパーミッション
- INSTALL_FILE
  インストールするファイルのパーミッション
- INSTALL_PROGRAM
  インストールするプログラムのパーミッション
- INSTALL_SCRIPT
  インストールするスクリプトのパーミッション
- SCRIPT_DIR
  dbs スクリプトディレクトリパス
- TAR_DIR
  tar 解凍後ディレクトリ名

include $(SCRIPT_DIR)/dbs-build.mk で include することにより、dbs を使うことができるようになります。

3.3.4 パッチ

dbs では dpatch と同様、**debian/patches** ディレクトリにパッチを格納する必要があります。パッチはユニファイド diff 形式 で書かれている必要があり、ファイル名の先頭には数字を付けます。数字とファイル名の間はアンダースコアで区切ります。特に拡張子等は必要ありません。dbs はこの数字が割り当てられている順にパッチをソースコードに適用していきます。

dpatch では 00list というファイルにパッチ名を書き、書かれた順にパッチを適用していきます。パッチの順番が

変わったときは、00list ファイルを修正すればいいだけですが、dbs の場合はパッチの順が変わったときはファイル名を変更しないといけません。

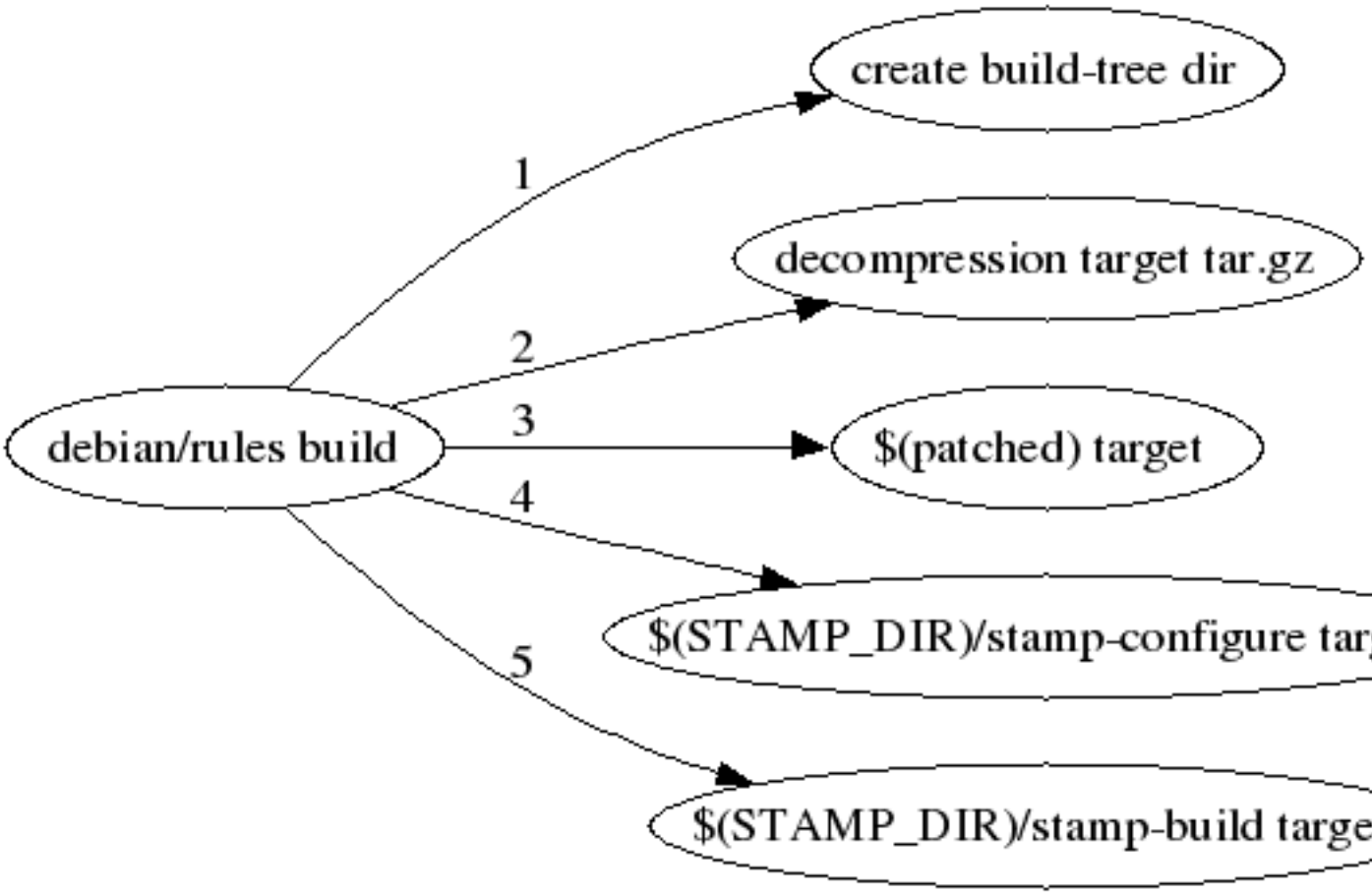
例えば、hello-dbs パッケージでは **00_maillocation** パッチがあります。

```
% ls -l debian/patches
00_maillocation
```

このパッチの前に新しいパッチ **00_patchtest** を当てる必要がある場合は、**00_maillocation** を **01_maillocation** に変更するという作業が必要になります。

3.3.5 パッケージのビルド

パッケージのビルドが行われるとき、以下の図の順でターゲットが呼ばれます。



1. create build-tree ディレクトリ
   $BUILD_TREE で指定されたディレクトリを作成します。デフォルトでは **build-tree** になっています。
2. decompression tar.gz
   ソースコードが圧縮されている tar.gz を build-tree ディレクトリに解凍します。
3. $(patched) target
   debian/patches ディレクトリにあるパッチを適用します。
4. $(STAMP_DIR)stamp-configure target
   stamp-configure ターゲットを実行します。hello-dbs では $BUILD_TREE ディレクトリに移動し、configure を実行します。
5. $(STAMP_DIR)/stamp-build target
   stamp-build ターゲットを実行します。hello-dbs では $BUILD_TREE ディレクトリに移動し、make を実行します。

build 時の dpatch との違いは、

- パッチを当てるためのターゲット名が異なる
  dpatch は patch-stamp になりますが、dbs は $(patched) です。
  $(patched) は **/usr/share/dbs/dbs-build.mk** 内で $(STAMP_DIR)/patchapply と宣言されています。
- マーク用のファイル名が異なる
  stamp-configure や stamp-build というマーク用のファイル名が逆だったりします。(dpatch は configure-stamp / build-stamp)

3.3.6 パッケージの clean

dbs の ソースの clean ターゲットはいたってシンプルです。dpatch は 既に展開されているソースにパッチを適用するため、clean ターゲット時には適用されたパッチを外す処理が必要になりますが、dbs では、build 時に生成されたマークファイル用ディレクトリやソース格納先ディレクトリ ($BUILD_TREE) をばっさり削除します。これには、適用されたパッチの管理等を行わずに済むというメリットがあります。

しかし、dbs は パッケージビルド毎に

1. ディレクトリを削除
2. tar.gz を解凍
3. パッチ適用

とするので、サイズの大きい tar.gz をパッケージ化するときは時間がかかるというデメリットもあります。

3.4 パッチの変更

dbs でパッチの変更を行うために以下のコマンドが提供されています。

3.4.1 dbs-edit-patch

このコマンドは現在のソースからパッチを作成する環境を構築してくれます。hello-dbs で新しいパッチを作りたいという状況になったとします。以下のコマンドを実行します。

```
% dbs-edit-patch  01mogeri_patch
```

実行すると /tmp ディレクトリに 01mogeri_patch というディレクトリが作成されます。ディレクトリの中は以下のようになっています。

```
% ls -l /tmp/01mogeri_patch/
-rwxr-xr-x 1 iwamatsu iwamatsu  546 2007-02-15 06:00 dbs-update-patch
drwxr-xr-x 2 iwamatsu iwamatsu 1024 1996-08-04 23:48 hello-1.3.orig
drwxr-xr-x 2 iwamatsu iwamatsu 1024 1996-08-04 23:48 hello-1.3.orig-old
```

hello-1.3.orig は修正対象のディレクトリで、hello-1.3.orig-old は修正前のディレクトリです。hello-1.3.orig を修正した内容と hello-1.3.orig-old で差分を取り、パッチを生成し、パッチをコピーしてくれるスクリプトが dbs-update-patch になっています。

```
#!/bin/sh -e
cd "/tmp/01mogeri_patch"
HOOK_DIR=/home/iwamatsu/dev/debian/dbs/hello-dbs-1.3/debian/dbs-hooks
test -d "$HOOK_DIR" && run-parts "$HOOK_DIR" --arg update-patch-prediff
find -name "*.bak" -print0 | xargs -0 --no-run-if-empty rm
find -name "*~" -print0 | xargs -0 --no-run-if-empty rm
: > new_patch
diff -ruN  hello-1.3.orig-old hello-1.3.orig >> new_patch || test $? -eq 1
mv new_patch "/home/iwamatsu/dev/debian/dbs/hello-dbs-1.3/debian/patches/01mogeri_patch"
test -d "$HOOK_DIR" && run-parts "$HOOK_DIR" --arg update-patch-postdiff
```

例えば、

```
cat  /tmp/01mogeri_patch/hello-1.3.orig/MOGERI
mogemogeri
```

という修正を行い、**/tmp/dbs-update-patch** を実行した場合、

```
% /tmp/01mogeri_patch/dbs-update-patch
% ls -l debian/patches/01mogeri_patch
-rw-r--r-- 1 iwamatsu iwamatsu 212 2007-02-15 06:08 debian/patches/01mogeri_patch
% cat debian/patches/01mogeri_patch
diff -ruN hello-1.3.orig-old/MOGERI hello-1.3.orig/MOGERI
--- hello-1.3.orig-old/MOGERI   1970-01-01 09:00:00.000000000 +0900
+++ hello-1.3.orig/MOGERI       2007-02-15 06:06:54.000000000 +0900
@@ -0,0 +1 @@
+mogemogeri
```

となります。

3.5 まとめ

今回、dbs を触ってみたのですが、

- ソースが見えない
  ソースが tar.gz で固まっているため、見ることができない。見るには コマンドを使って解凍する必要がある。
- dpatch と比べてソースの編集がしずらい
  コマンドは用意されているんだけど、一回 /tmp 等に持っていって修正して.... という作業が発生するので、めんどうくさいところがある。

という感想です。また、dbs を使っていたパッケージも だんだん dpatch に移行しつつあるのでもう役目は追えたのではないかという気もします。

4　darcs 使いかた

David Smith

darcs はソースコード管理ツールの一つです。Haskell で開発されている注目プロジェクトとして Emacs、Common Lisp、そして Haskell のコミュニティでは大好評です。[*9]

git、Mercurial、monotone、他の数多くの分散ソース管理ツールの仲間であります。しかし、git などの分散ソース管理ツールと違い、バージョンは管理しなく、パッチを管理します。その二つの理念の違い、つまりバージョン管理対パッチ管理、について darcs を活用しながら説明します。

4.1　基本な darcs

4.1.1　とりあえず使ってみよう

先ず、新規リポジトリを作ってみます。

```
exponent,1102:~/workspace> mkdir myproject
exponent,1103:~/workspace> cd myproject
exponent,1104:~/workspace/myproject> darcs ini
exponent,1105:~/workspace/myproject> ls
_darcs
exponent,1106:~/workspace/myproject> echo 'Hello Darcs!' > README
exponent,1107:~/workspace/myproject> darcs whatsnew
No changes!
exponent,1108:~/workspace/myproject> darcs add README
exponent,1109:~/workspace/myproject> darcs whatsnew
\{
addfile ./README
hunk ./README 1
+Hello Darcs!
\}
exponent,1110:~/workspace/myproject> darcs record
Darcs needs to know what name (conventionally an email address) to use as the
patch author, e.g. 'Fred Bloggs <fred@bloggs.invalid>'.  If you provide one
now it will be stored in the file '_darcs/prefs/author' and used as a default
in the future.  To change your preferred author address, simply delete or edit
this file.\footnote{残念ながら、国際化はまだまだ出来ていません。}

What is your email address? David Smith <davidsmith@acm.org>
addfile ./README
Shall I record this change? (1/?)  [ynWsfqadjkc], or ? for help: y

hunk ./README 1
+Hello Darcs!
Shall I record this change? (2/?)  [ynWsfqadjkc], or ? for help: y

What is the patch name? Say hello to darcs
Do you want to add a long comment? [yn]n

Finished recording patch 'Say hello to darcs'
exponent,1111:~/workspace/myproject> darcs changes
Fri Apr 20 15:43:45 JST 2007  David Smith <davidsmith@acm.org>
  * Say hello to darcs
```

darcs ではリポジトリの状態はパッチ集合だけで成り立ちます。パッチを作るのはファイルを darcs に登録し、内容を記録し、最後に名前付けで完成します。ちなみにパッチは名前だけで識別され、バージョン番号は一切ありません。[*10]

[*9] 大好評は多分言い過ぎです。せいぜい部分的に気に入られています。

[*10] 自分の名前とメール先を$HOME/.darcs/author に書いたら聞かれません。

コマンド名	例	意味
init	darcs init	リポジトリを初期化する
whatsnew	darcs whatsnew -v	現在のリポジトリと記録済みの情報との差分を明示する
record	darcs record -m ’バグ#103 を修正’	パッチをリポジトリに記録する
changes	darcs changes –summary	パッチによるチェンジログを生成する
add	darcs add realcsh.c	新しいファイルを darcs に教える
mv	darcs mv realcsh.c dangershell.c	ファイル名前変更

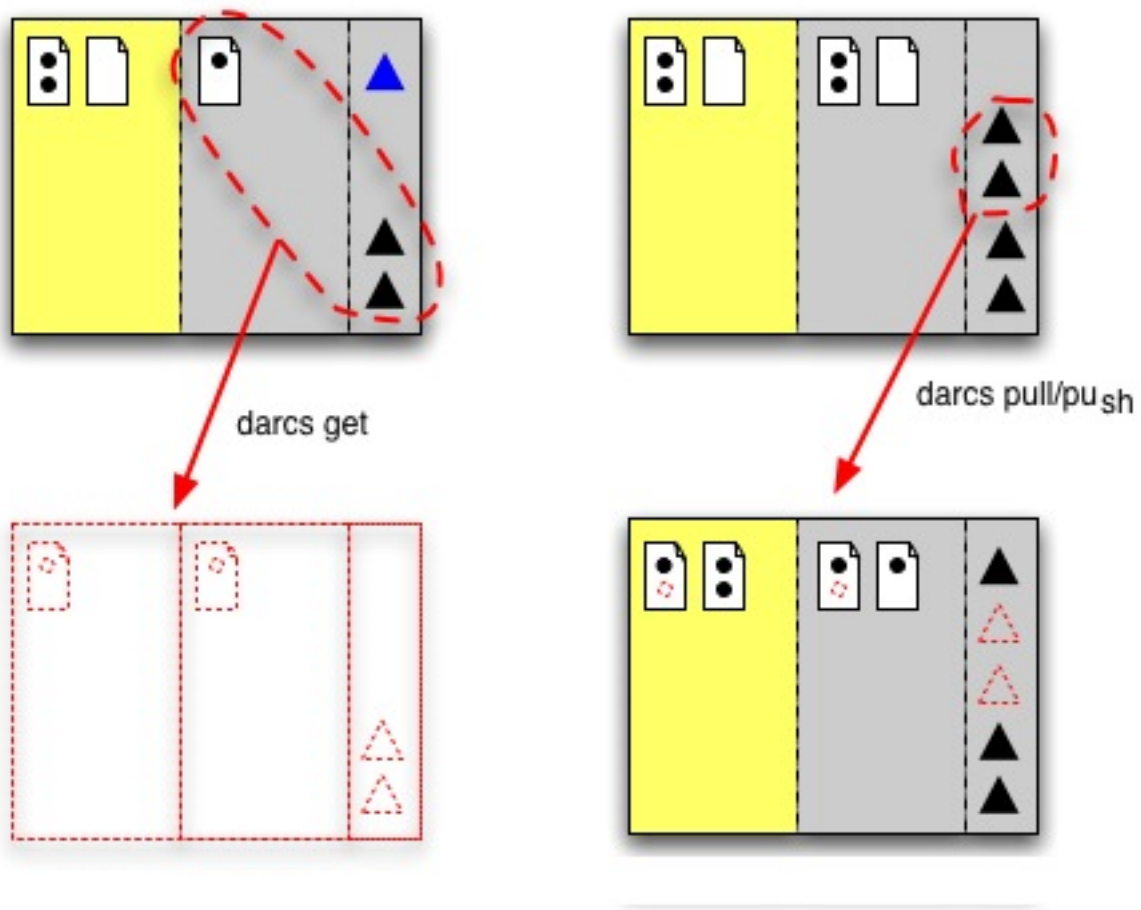
4.1.2 他のリポジトリとの同期

darcs はリポジトリ内のブランチをサポートしていません。複数リポジトリに同じパッチを一個も共有すれば、ブランチと呼んでもいいとのことです。つまり、リポジトリのコピーしかブランチが作れません。darcs get で既に存在するロカールリポジトリ又はネットワークを通して取得出来るリポジトリでコピーできます。[*11]もちろん、ローカルの場合に get を使わず cp でも大丈夫です。

```
exponent,1049:~/workspace> darcs get myproject myproject2
Copying patch 1 of 1... done!
Finished getting.
exponent,1050:~/workspace> cd myproject2
exponent,1051:~/workspace/myproject2> echo 'Darcs rules!' >> README
exponent,1052:~/workspace/myproject2> darcs record -m 'Add more darcs love'
hunk ./README 2
+Darcs rules!
Shall I record this change? (1/?)  [ynWsfqadjkc], or ? for help: y

Finished recording patch 'Add more darcs love'
exponent,1054:~/workspace/myproject2> darcs push ../myproject

Fri Apr 20 18:04:37 JST 2007  David Smith <davidsmith@acm.org>
  * Add more darcs love
Shall I push this patch? (1/1)  [ynWvpxqadjk], or ? for help: y

Finished applying...

exponent,1055:~/workspace/myproject2>
```

この時、先方のリポジトリに書き込み権利がありましたが、公開されている一般的なプロジェクトでは開発者以外はコミット出来ないでしょう。そのため、push/pull の代わりに send/apply を使います。send は指定するパッチをメールで送り、apply はファイルにある darcs 形式パッチをリポジトリに記録します。

[*11] 現在、HTTP 及び SSH のみの通信になっています。



ここまで darcs の機能がほとんど普通だと言えるでしょう。しかしながらパッチとパッチの依存関係を計算するための「パッチ代数」により Cherry Picking（桜取り）というワークフローが他より余程手軽にできます。残念ですが、例を造るのに darcs が無限ループに落ちているばかりのようなので、とにかく素晴らしいと信用してください。

4.1.3 darcs-buildpackage

Debian パッケージ支持と Haskell を習うために John Goerzen さんが darcs-buildpackage を開発しました。現在、ほとんどソース管理ツールでは Debian パッケージ作業のための'-buildpackage' パッケージはあるよう、その中で darcs-buildpackage の使い方も同様ですので共通できます。

残念ながら、darcs は実際にリポジトリサイズに対して動作がとても遅くなることがあったり、小さいリポジトリでもパッチ代数問題の解決のため、ものすごく負荷かかりることもあります。darcs の上流開発は git と Mercurial と比べると活発とは言えませんが、理論的にリポジトリの内容に一切障害を与えなく最適化が着々と行われています。私としては機能が改善するはずだと思いますが Mercurial に大抵負けています。[*12]

[*12] John Goerzen さんも既に darcs より hg に変更しました

5 git-buildpackage の使いかた

上川 純一

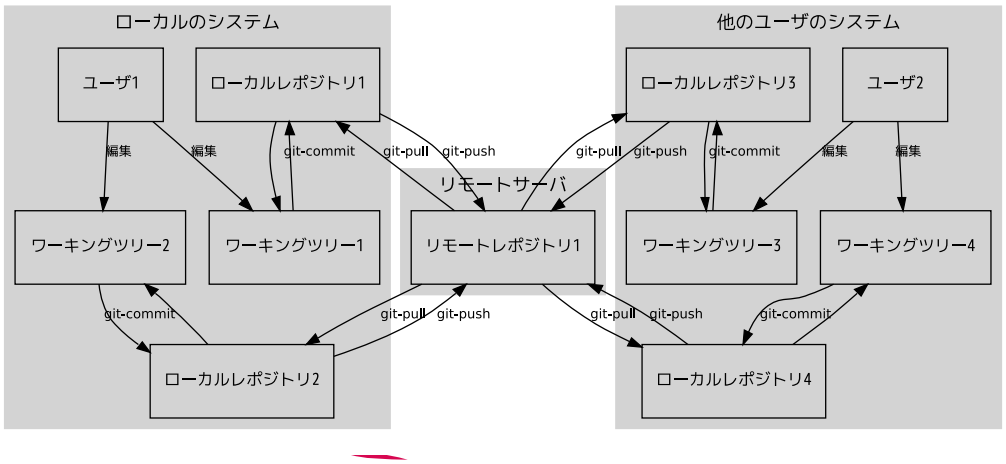
git はソースコードを管理するためのツールです。ソースコード管理のツール、もしくはバージョン管理ツールと呼ばれ、VCS や SCM などと略称されるツールのひとつです。また、旧来の CVS、SVN などの集中モデルの SCM と違い分散モデルを採用しているため、分散 SCM(DSCM) と呼ばれます。 その git を利用して Debian のパッケージを管理するための仕組が git-buildpackage です。

git-buildpackage 利用の詳細に入る前に、基本的な git の利用方法を紹介します。

5.1 git 超入門

5.1.1 git でのデータの流れ

まず最初に git で管理している場合のデータの流れをみてみましょう。各利用者が直接編集しているデータが git-commit や git-push / git-pull コマンドでやりとりされます。[*13]



5.1.2 git 関連用語

ここで、この文書で利用する関連用語を整理しておきます。

[*13] 本当はローカルレポジトリとリモートレポジトリの区別はないため、技術的には直接ユーザ間での git-pull/git-push が可能です。

用語	定義
ワーキングツリー	SCM で管理されている作業用のディレクトリで、ユーザが直接作業できるようにファイルがある場所
ローカルレポジトリ	SCM で管理されているデータワーキングツリーと同じ場所の.git ディレクトリに実体がある。直接ファイルを編集することはできない
リモートレポジトリ	SCM 管理されているデータで、ネットワーク上のどこかに存在しているもの。しばしば他人と共有している。直接ファイルを編集することはできない[*14]
コミット	ワーキングツリーからローカルレポジトリに情報を反映すること
プッシュ	ローカルレポジトリからリモートレポジトリに情報を反映すること
プル	リモートレポジトリからローカルレポジトリとワーキングツリーに情報を反映すること

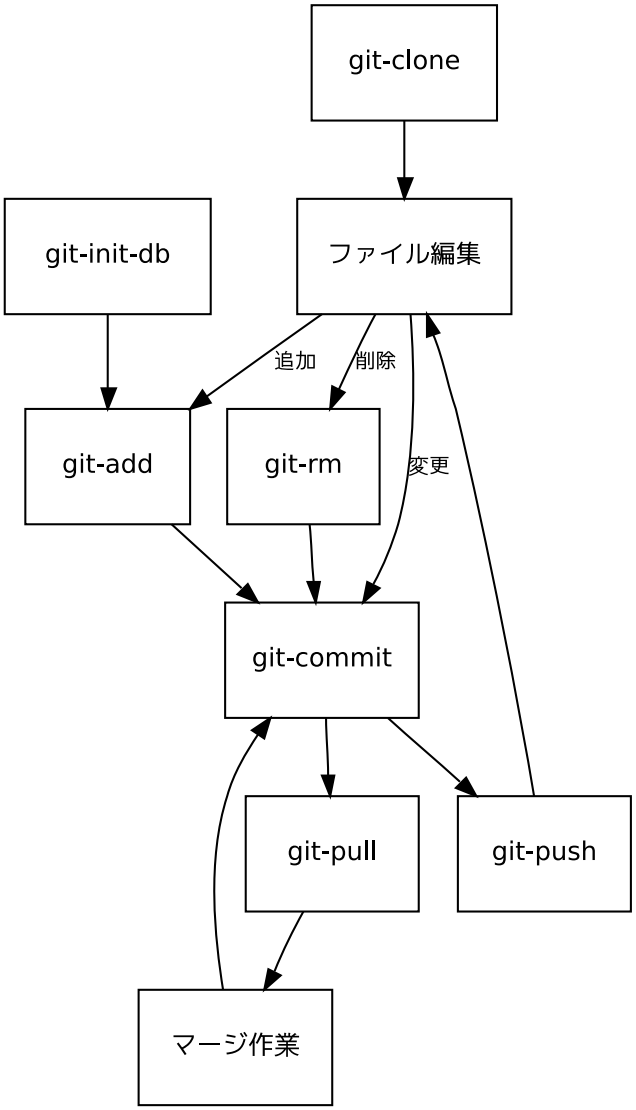
5.1.3 git でよく使うコマンド

git[*15] の基本的な操作はコマンドラインプログラムで実行できるようになっています。git パッケージは多数のコマンドラインのプログラムで構成されています。多数のコマンドはありますが、実は毎日のオペレーションに必要なもの、特に既存の別の SCM から移行してきた場合にいままで行ってきたことと同じことをするために必要なものというのはそれほど多くありません。git でよく使うコマンドを解説します。

コマンド名	例	意味	cvs 相当
git-clone	git-clone *git://XXX/YYY*	リモートレポジトリをローカルにクローンし、ワーキングツリーをチェックアウトする	cvs login, cvs co
git-init-db	git-init-db	ローカルレポジトリ (.git ディレクトリ) を作成する	cvs init
git-pull	git-pull *git://XXX/YYY*	リモートレポジトリの変更をローカルにマージし、ワーキングツリーをアップデートする	cvs up
git-commit	git-commit -a -m 'xxx'	ローカルレポジトリに変更をコミットする	cvs ci の前半
git-push	git-push *git://XXX/YYY*	ローカルレポジトリをリモートレポジトリに送信する	cvs ci の後半
git-add	git-add *filename*	ファイルを次回コミットの際にローカルレポジトリに追加されるように登録する	cvs add
git-rm	git-rm *filename*	ファイルを次回コミットの際にローカルレポジトリから削除されるように登録する	cvs remove
git-status	git-status	ローカルレポジトリに対してワーキングツリーの状況を確認する	cvs status
git-diff	git-diff	ローカルレポジトリとワーキングツリーの差分を表示する	cvs diff

[*15] Debian では `apt-get install git-core` でインストールできる

5.1.4 よくある作業の流れ

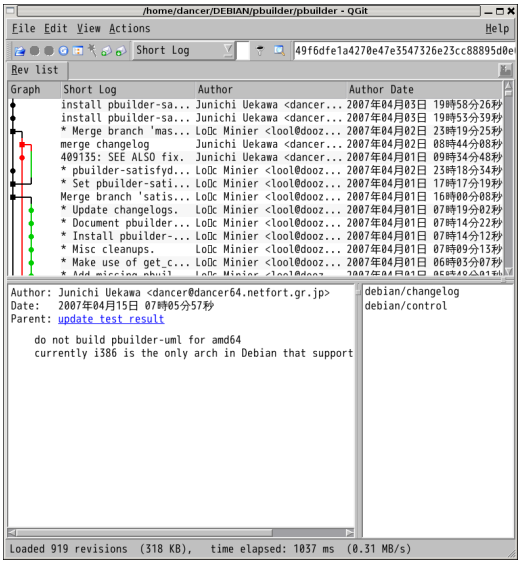


よくある作業の例を紹介します。
まず、新しくローカルレポジトリをつくるのであれば、git-init-db でローカルレポジトリを作成し、ファイルを git-add, git-commit で追加します。そうでなければ、既存のリモートレポジトリを git-clone で複製します。これで、作業可能なワーキングツリーができ、 .git ディレクトリにはローカルレポジトリが作成されました。
ワーキングツリーでファイルを編集して、git-commit でローカルレポジトリに反映します。削除・追加がある場合には、git-add/git-rm コマンドを利用します。通常は、何がコミットされるのか、を git-status コマンドで確認し、ファイルを指定してコミットすることになるでしょう。git-diff コマンドでこれからコミットする予定の差分を確認することもできます。
リモートレポジトリと同期するため、まず git-pull で最新の情報を取得します。ここでコンフリクトがあればワーキングツリーを修正し、git-commit します。問題ないようであれば、git-push でリモートレポジトリに送信します。

5.1.5 GUI ツール

git の履歴情報などを見るのには、視覚的にわかりやすい Qt の GUI である qgit[a] を利用すると履歴や差分が見れて便利です。GUI が利用できない環境では、git-whatchanged コマンドなどを利用すればよいでしょう。他の GUI として git-gui [b]や、gitweb [c]があります。

コミットに特化したツールとして、gct[d] などもあります。



[a] apt-get install qgit でインストール可能

[b] 旧 gitk が git-gui にリネームされた, apt-get install git-gui でインストール可能

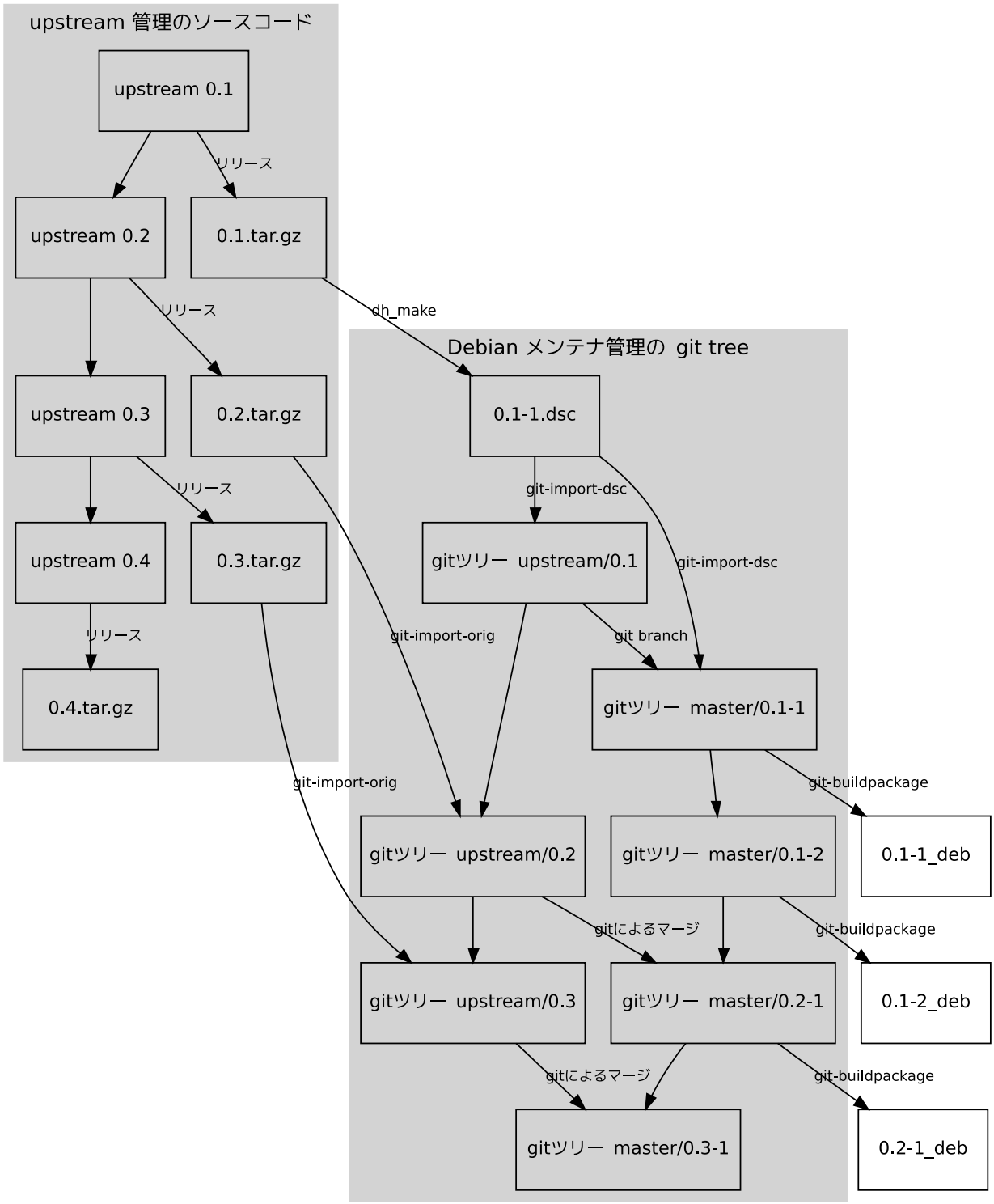
[c] ウェブフロントエンド apt-get install gitweb でインストール。

[d] apt-get install gct

5.2 git-buildpackage の流れ

git-buildpackage*16 を利用した場合のパッケージングの流れを紹介します。前提として、upstream では謎の利用不可能な SCM を利用して開発をすすめており、Debian Developer はリリース毎に出てくる tar-ball しか利用できないものとします。

*16 apt-get install git-buildpackage でインストール可能



5.2.1 git-import-dsc

まず、Debian Developer はアップストリームの tarball を展開し、そこで dh_make を実行し、一連のパッケージング作業を行います。

```
[14:10:52]dancer64:work> cp ../upstream/vvv-0.1.tar.gz .
[14:10:57]dancer64:work>
[14:11:03]dancer64:work> tar xfz vvv-0.1.tar.gz
[14:11:08]dancer64:work> cd vvv-0.1
[14:11:12]dancer64:vvv-0.1> dh_make -f ../vvv-0.1.tar.gz

Type of package: single binary, multiple binary, library, kernel module or cdbs?
 [s/m/l/k/b] s

Maintainer name : Junichi Uekawa
Email-Address   : dancer@debian.org
Date            : Sat, 14 Apr 2007 14:11:28 +0900
Package Name    : vvv
Version         : 0.1
License         : blank
Type of Package : Single
Hit <enter> to confirm:
Done. Please edit the files in the debian/ subdirectory now. You should also
check that the vvv Makefiles install into $DESTDIR and not in / .
[14:14:44]dancer64:vvv-0.1> debuild -us -uc
 fakeroot debian/rules clean
dh_testdir
dpkg-buildpackage (debuild emulation): full upload (original source is included)
Now running lintian...
W: vvv: binary-without-manpage usr/bin/hello-world.sh
W: vvv: script-with-language-extension usr/bin/hello-world.sh
E: vvv: helper-templates-in-copyright
E: vvv: description-is-dh_make-template
W: vvv: wrong-bug-number-in-closes l3:#nnnn
E: vvv: section-is-dh_make-template
Finished running lintian.
[14:16:51]dancer64:vvv-0.1> cd ..

[14:17:42]dancer64:work> ls
vvv-0.1                vvv_0.1-1.diff.gz  vvv_0.1-1_amd64.build    vvv_0.1-1_amd64.deb
vvv-0.1.tar.gz        vvv_0.1-1.dsc           vvv_0.1-1_amd64.changes  vvv_0.1.orig.tar.gz
```

すると、Debian のソースパッケージファイルなどができます。

git-import-dsc は dsc ファイルをオプションにとると、[パッケージ名] ディレクトリを作成し、その中に git のローカルレポジトリを作成し、upstream と master ブランチ (通常利用するブランチ) を作成し、upstream にアップストリームのツリーを入れ、master に debian への改変を加えた後のツリーを入れます。この状態で、git-buildpackage などでパッケージがビルドできる状態になっています。

```
[14:17:42]dancer64:work> git-import-dsc vvv_0.1-1.dsc
Upstream version: 0.1
Debian version: 1
Initialized empty Git repository in .git/
Created initial commit b7e3ddd5bb1033c2cd9ae90d86ea8b6f55fb34be
 3 files changed, 16 insertions(+), 0 deletions(-)
 create mode 100644 Makefile
 create mode 100644 hello-world.sh
 create mode 100755 orig.sh
dpkg-source: warning: extracting unsigned source package (/home/dancer/tmp/git-test/work/vvv_0.1-1.dsc)
dpkg-source: extracting vvv in /home/dancer/tmp/git-test/work/tmpwJLVOR/unpack/vvv-0.1-1
dpkg-source: unpacking vvv_0.1.orig.tar.gz
dpkg-source: applying /home/dancer/tmp/git-test/work/vvv_0.1-1.diff.gz
 VCSCMD:  git
LOGTEXT Imported vvv-0.1-1
into Git repository

Created commit 041935d50363f69bcb7ff1b5e0df43b79784b6b1
 9 files changed, 149 insertions(+), 2 deletions(-)
 create mode 100644 debian/README.Debian
[中略]
 create mode 100755 debian/rules
[14:17:58]dancer64:work> cd vvv
[14:18:09]dancer64:vvv> git-status
# On branch master
nothing to commit (working directory clean)
[14:18:12]dancer64:vvv> git-branch
* master
  upstream
```

5.2.2 git-import-orig

新しいアップストリームのバージョンがリリースされた場合には、git-import-dsc で作成されたディレクトリの中で git-import-orig コマンドを実行して、ローカルレポジトリにインポートします。

```
[14:18:50]dancer64:vvv> git-import-orig ../../upstream/vvv-0.2.tar.gz -u 0.2
Upstream version is 0.2
Repository has uncommitted changes, commit them first:
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#        build-stamp
#        configure-stamp
#        debian/files
#        debian/vvv/
nothing added to commit but untracked files present (use "git add" to track)

[14:19:04]dancer64:vvv> debclean
Cleaning in directory ./.git/refs/tags
Directory ./.git/refs/tags: contains no debian/changelog, skipping
Cleaning in directory .
dh_testdir
dh_testroot
rm -f build-stamp configure-stamp
# Add here commands to clean up after the build process.
/usr/bin/make clean
make[1]: ディレクトリ ‘/home/dancer/tmp/git-test/work/vvv’ に入ります
make[1]: ‘clean’ に対して行うべき事はありません.
make[1]: ディレクトリ ‘/home/dancer/tmp/git-test/work/vvv’ から出ます
dh_clean
[14:19:28]dancer64:vvv> git-import-orig ../../upstream/vvv-0.2.tar.gz -u 0.2
Upstream version is 0.2
Importing ../../upstream/vvv-0.2.tar.gz to upstream branch...
Switched to branch "upstream"
  master
* upstream
 VCSCMD:  git
LOGTEXT Imported vvv-0.2
into Git repository



Created commit 8e61554f99a28de9b10c23ae0319b8cc4aa07b40
 2 files changed, 4 insertions(+), 1 deletions(-)
 create mode 100644 NEWS
Merging to master
Switched to branch "master"
* master
  upstream
 100% (14/14) done
Auto-merged Makefile
CONFLICT (content): Merge conflict in Makefile
Automatic merge failed; fix conflicts and then commit the result.
git-pull returned 1
Couldn't pull upstream to .
Import of ../../upstream/vvv-0.2.tar.gz failed
[14:19:50]dancer64:vvv> cat Makefile
all:

install:
<<<<<<< HEAD:Makefile
        install -o root -g root -m 755 hello-world.sh $(DESTDIR)/usr/bin/hello-world.sh
=======
        install -o root -g root -m 755 hello-world.sh /usr/local/bin/hello-world.sh
>>>>>>> 8e61554f99a28de9b10c23ae0319b8cc4aa07b40:Makefile

clean:

.PHONY: all install clean

[14:19:58]dancer64:vvv> vi Makefile
[14:20:15]dancer64:vvv> git-commit -a -m '0.2 debian changes'
Created commit cffcdabb818fdacc38f1a21bab744c56480d1a44
```

この操作により、upstream ブランチに新しいバージョンがインポートされ、master ブランチに変更がマージされ、適切なタグが作成されます。

5.2.3 git-buildpackage

各種編集を行い、コミットを完了したら、git-buildpackage を実行し、レポジトリの内容から Debian パッケージを作成します。–git-tag オプションを付けておくとビルドしなおしたら適切なタグを自動でつけてくれるので便利です。また、コミットしわすれていると警告を出してくれるのもよいです。

```
[14:21:26]dancer64:vvv> git-buildpackage
[中略]
You have uncommitted changes in your source tree:
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#
#        modified:   debian/changelog
#
no changes added to commit (use "git add" and/or "git commit -a")

Use --git-ignore-new to ignore.
[14:21:34]dancer64:vvv> git-commit -a -m 'update changelog'
Created commit 98ae75017264f86192d30b894191d862b98821f5
 1 files changed, 6 insertions(+), 0 deletions(-)
[14:21:43]dancer64:vvv> git-buildpackage
dh_testdir
dh_testroot
rm -f build-stamp configure-stamp
[中略]
$ git-buildpackage -us -uc --git-tag
```

5.3 アップストリームが git で管理している場合

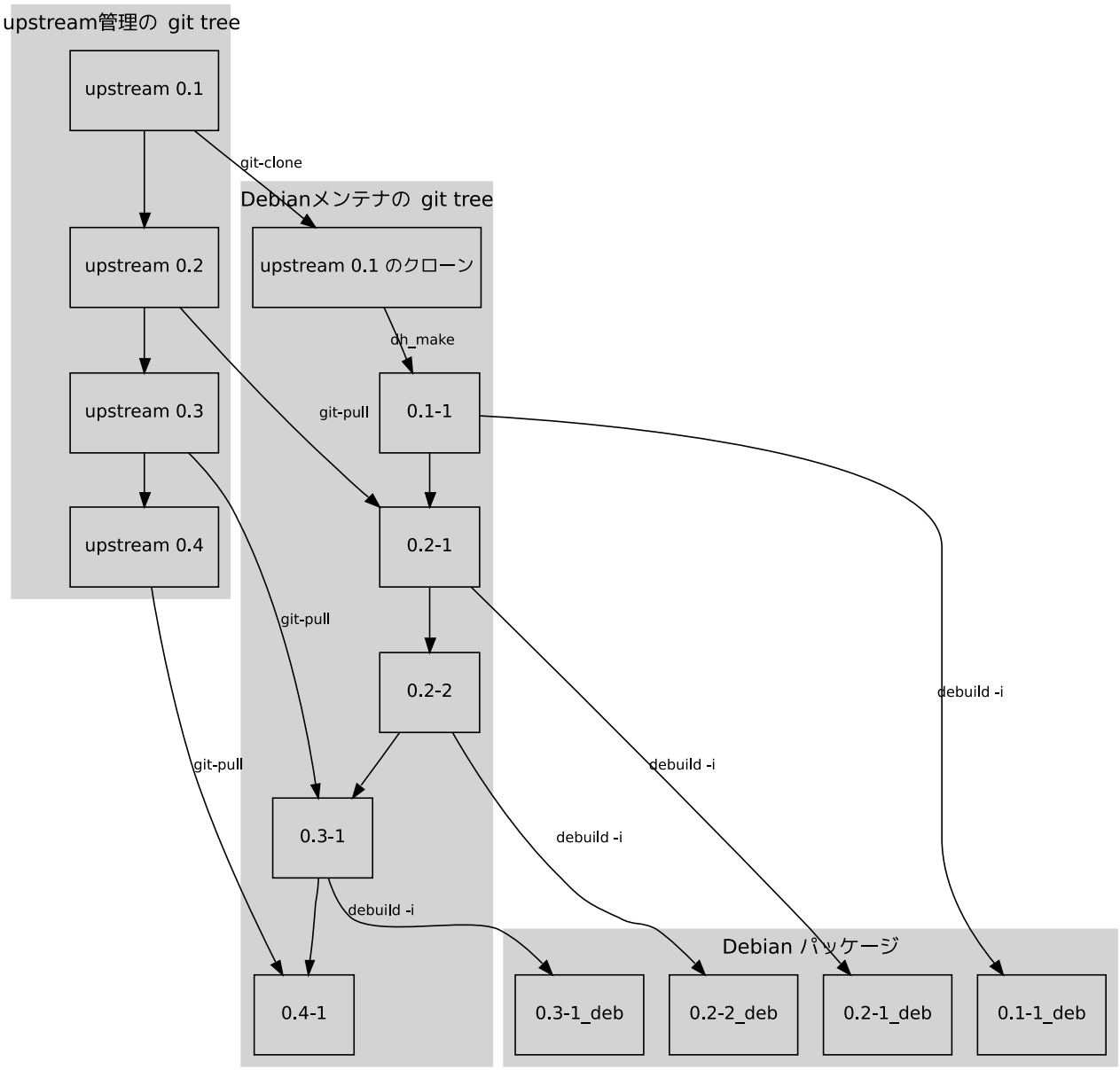
別のシナリオを考えてみます。アップストリームが git で管理している場合に直接 git を利用したワークフローは幾分か簡単にできます。どちらがよいかはわかりませんが、参考までに掲載します。

git-clone したらローカルレポジトリはリモートレポジトリから見たらただのブランチのため、ローカルレポジトリで変更を加えていれば、git-pull するたびにアップストリームの変更をマージすることになります。ビルドする際には、debuild などを直接使えばよいでしょう。ただ、この場合は自動でタグをつけたりはしてくれません。

```
$ debuild -us -uc -i -I
```

そのため、git-import-dsc, git-import-orig を使わない場合においても git-buildpackage を利用するのがよいでしょう。upstream ブランチが必要になるのは git-import-orig をする際だけなので、master ブランチのみしかなくても動きます。

```
$ git-buildpackage -us -uc -i -I
```

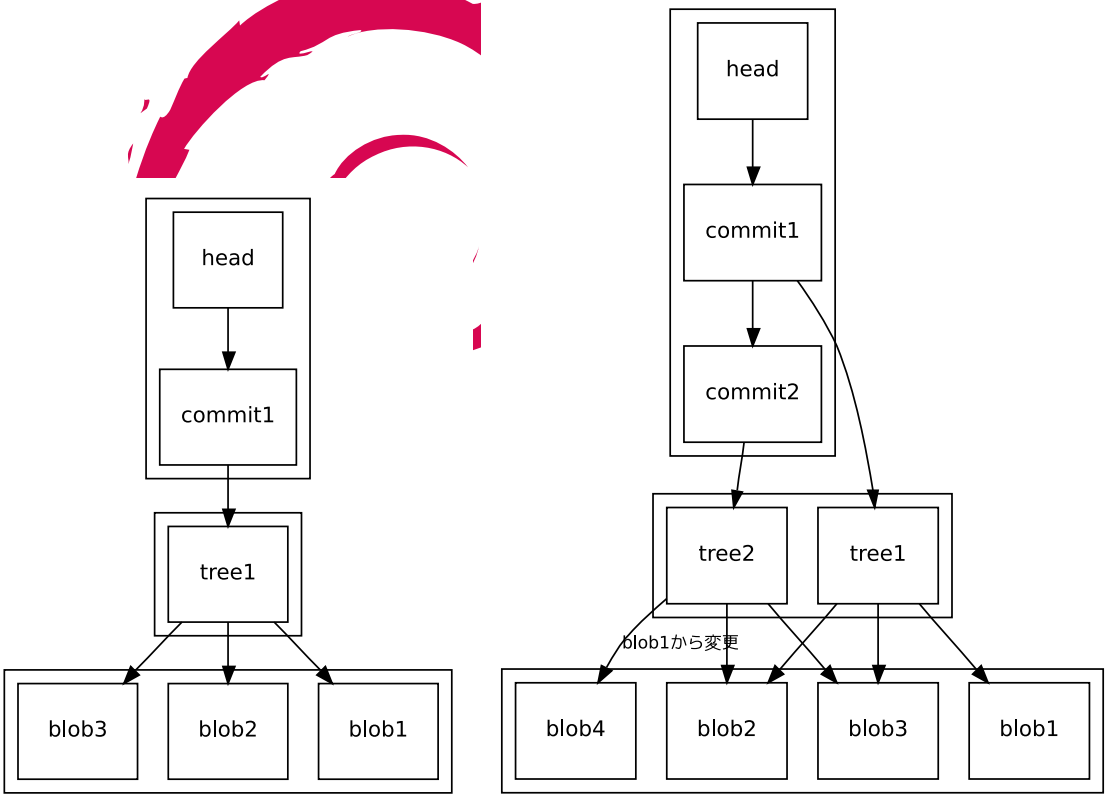

upstream管理の git tree
upstream 0.1
git-clone
Debianメンテナの git tree
upstream 0.2
upstream 0.1 のクローン
dh_make
upstream 0.3
git-pull
0.1-1
upstream 0.4
0.2-1
git-pull
debuild -i
0.2-2
git-pull
debuild -i
0.3-1
debuild -i
debuild -i
Debian パッケージ
0.4-1
0.3-1_deb
0.2-2_deb
0.2-1_deb
0.1-1_deb

6 git の実装技術的解説

上川 純一

簡単に git のレポジトリ[*17]のファイル構成のコンセプトを紹介します。レポジトリのファイルは .git ディレクトリ以下[*18]にあります。基本的なオブジェクトは commit, tree, blob の三種類があります。. git/objects/ 以下にファイルは gzip 圧縮された状態で保存されています。. git/HEAD (最近は.git/refs/heads/master) に最新の commit のハッシュ（ハッシュはデータの sha1sum をとることで取得している）が記録されています。commit の中身を見ると、tree のハッシュと 親 commit とコミットメッセージ情報が含まれています。tree を見ると、そのコミットのときのディレクトリ情報が含まれており、それぞれのファイル実体（blob）の hash 値が含まれています。

なお、objects はばらばらで管理されるとディスクの利用効率が悪いので git-repack コマンドを利用して pack 形式で保存されることが多いです。そうなると、.git/objects/pack に保存されます。



この情報を具体的にコマンドラインで確認してみた例を例示します。特に重要な点は、'head' ファイルへの書き込みという OS 観点ではアトミックに実装できるオペレーションでコミットが実現されているということでしょう。[*19]

[*17] ローカルレポジトリもリモートレポジトリも基本的には同じファイル構成です。
[*18] もしくは環境変数 GIT_DIR で指定された場所
[*19] ファイルの SHA-1 ハッシュ値の衝突が発生しないということが前提です。ただ、現実的に問題になることは無いでしょう。

```
[11:39:25]dancer64:pbuilder> cat .git/refs/heads/master
49f6dfe1a4270e47e3547326e23cc88895d0e05d
[11:39:38]dancer64:pbuilder> git-cat-file -t 49f6dfe1a4270e47e3547326e23cc88895d0e05d
commit
[11:39:44]dancer64:pbuilder> git-cat-file commit 49f6dfe1a4270e47e3547326e23cc88895d0e05d
tree 98dc4f51c0ae895607db43f8b617a7cacfbdc34b
parent e40c851b3017b09a609e2e36af6ae8a20b8ffff3
author Junichi Uekawa <dancer@dancer64.netfort.gr.jp> 1176588357 +0900
committer Junichi Uekawa <dancer@dancer64.netfort.gr.jp> 1176588357 +0900

do not build pbuilder-uml for amd64
currently i386 is the only arch in Debian that supports user-mode-linux
[11:39:47]dancer64:pbuilder> git-cat-file -t 98dc4f51c0ae895607db43f8b617a7cacfbdc34b
tree
[11:39:52]dancer64:pbuilder> git-cat-file tree 98dc4f51c0ae895607db43f8b617a7cacfbdc34b|strings|head -5
100644 .cvsignore
100644 .gitignore
100644 AUTHORS
o[v'a
R100644 COPYING
[11:40:13]dancer64:pbuilder> git-ls-tree 98dc4f51c0ae895607db43f8b617a7cacfbdc34b|head -5
100644 blob 99ee691a57a1d79999965e8f9362da79d18e8ce4    .cvsignore
100644 blob e4e5f6c8b2deb54bf38312dd9e2f53489b60d6a6    .gitignore
100644 blob 30ded29ac4176f5b7627618b27a8cb15c7ea0a52    AUTHORS
100644 blob b7b5f53df1412df1e117607f18385b39004cdaa2    COPYING
100644 blob 97cae37dd0355d4f29837bc02bbad4bcdba98914    ChangeLog
[11:40:18]dancer64:pbuilder> git-cat-file -t 97cae37dd0355d4f29837bc02bbad4bcdba98914
blob
[11:40:36]dancer64:pbuilder> git-cat-file blob 97cae37dd0355d4f29837bc02bbad4bcdba98914|head
2007-04-11  Junichi Uekawa  <dancer@debian.org>

        * AUTHORS, etc: remove $Id$, which is CVS specific

2007-04-10  Junichi Uekawa  <dancer@debian.org>

        * Documentation/pbuilder-doc.xml: update documentation

        * pbuilder-modules: say lenny instead of sarge

[11:40:43]dancer64:pbuilder> ls .git/objects/97/cae37dd0355d4f29837bc02bbad4bcdba98914
.git/objects/97/cae37dd0355d4f29837bc02bbad4bcdba98914
[11:40:58]dancer64:pbuilder> ls -l .git/objects/97/cae37dd0355d4f29837bc02bbad4bcdba98914
-r--r--r-- 1 dancer dancer 27255 2007-04-11 09:01 .git/objects/97/cae37dd0355d4f29837bc02bbad4bcdba98914
```

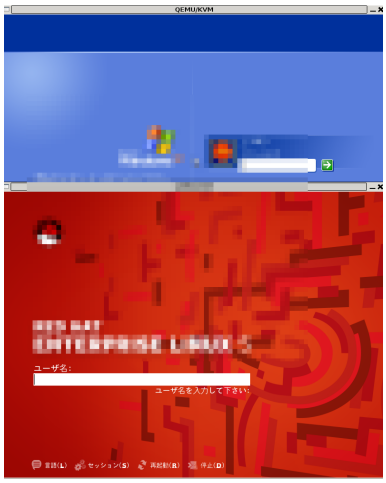
7　仮想マシンモニタ KVM

上川 純一

KVM という仮想マシンモニタがあります。これは、 Intel VT, もしくは、AMD-V 対応のプロセッサ[*20] の仮想化対応機能を活用するための仕組です。2006 年末の時点では、kvm はデバイスドライバとして実装されており、`/dev/kvm` として実装されています。

7.1　使いかた

Linus Kernel 2.6.20 以降ではカーネル側の機構は標準で入っているようです。[*21] Debian で KVM を利用する際の手順を説明します。



`apt-get install kvm` でパッケージをインストールします。kvm コマンドがインストールされます。これが kvm 向けに変更された qemu です。

udev が作成してくれる `/dev/kvm` にアクセスできるようにします。デフォルトは `root:kvm 660`[*22] で、root のみがアクセスできる設定になっています。ユーザを kvm グループに追加すればよいでしょう。

```
# adduser dancer kvm
```

CPU の VT 機能はデフォルトで on になっていない場合があるので、 on にします。これはマシンによって違うようです。 通常 BIOS で設定します。 旧モデルの MacBook の場合は EFI 上で必要なコマンドを発行することになります。[*23]

以上で、kvm コマンドを利用して VT の恩恵にあずかることができるようになります。コマンドラインなどは qemu 互換、むしろ qemu そのものを利用しているので、qemu と同じように動かすことができます。

7.2　実行例

kvm を活用していろいろと試してみましょう。qemu 用のディスクイメージを作成し、適当なディストリビューションのディスクイメージを利用して、インストールします。おそらく、-monitor コマンドで stdio をモニタにするほうが便利でしょう。

[*20] Intel Core Duo や、Opteron Rev. F など
[*21] 2.6.20-rc1 で導入され、まだ 2.6.20 はリリースされていないためこの記事の内容は 1 月 19 日時点での予測です
[*22] バージョン 11-1 以降の場合
[*23] `vmx-var-set.efi` という名前で流通しているようです。

```
$ qemu-img create -f qcow hda 10G
$ kvm -hda hda -cdrom /home/iso/RHEL4-U4-i386-AS-disc1.iso
   -boot d -m 512 -monitor stdio
(qemu) change cdrom /home/iso/RHEL4-U4-i386-AS-disc2.iso
(qemu) change cdrom /home/iso/RHEL4-U4-i386-AS-disc3.iso
(qemu) change cdrom /home/iso/RHEL4-U4-i386-AS-disc4.iso
(qemu) change cdrom /home/iso/RHEL4-U4-i386-AS-disc5.iso
(qemu) change cdrom /home/iso/RHEL4-U4-i386-AS-disc1.iso
(qemu) eject cdrom
```

ここでは某 CDROM が 5 枚あるディストリビューションを例にとっていますが、CDROM が 5 枚あっても、qemu モニターにて CDROM を入れ換えながら GUI 操作を行うことで無事にインストール完了します。リブートするところでゲスト OS がカーネルパニックを起こしますが、そこは適当に kvm 自体を起動しなおせば問題ありません。

```
$ kvm -hda hda  -boot c -m 512 -monitor stdio -localtime  \
 -redir tcp:2222::22
```

7.3 ベンチマークしてみた

qemu を使った場合と kvm を使った場合の速度比較をしてみました。まず、ユーザ空間で完結する例として単純に while でループを回すだけのプログラムです。qemu の場合は 6s かかったものが、 kvm の場合は 0.6s で完了しました。

7.4 パフォーマンスの見えかた

kvm を実行しているホスト OS からどのように見えるのか確認してみましょう。kvm の仕組だと、システムコールを実行して処理を行うことになるのでしょう。

`mpstat -P ALL 1` の出力を一部みてみると、一つの CPU の システム時間がとられているように見えます。

```
00 時 28 分 51 秒  CPU   %user   %nice    %sys %iowait    %irq   %soft  %steal   %idle    intr/s
00 時 28 分 52 秒  all    0.00    0.00   50.00    0.50    0.00    0.50    0.00   49.00   1421.78
00 時 28 分 52 秒    0    0.00    0.00   92.08    0.00    0.00    0.99    0.00    6.93    430.69
00 時 28 分 52 秒    1    0.00    0.00    6.93    0.99    0.00    0.00    0.00   91.09    991.09
```

`top` コマンドで見た場合には、kvm は通常のプロセスと同様に CPU 時間を消費し、メモリを消費しているように見えます。

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3746 dancer    25   0  308m 252m 246m R   82 25.7 100:36.60 kvm
 2512 root      10  -5     0    0    0 S    0  0.0   0:11.32 kjournald
 2806 root      15   0  2556  932  804 S    0  0.1   1:09.36 syslogd
```

`iostat 1 /dev/dm-1` の出力を確認してみます。ゲスト OS のディスク IO によってホスト OS のディスク IO が発生しているのが見れます。

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00   42.50   42.00    0.00   15.50

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
dm-1            805.94       102.97      6400.00        104       6464
```

7.5 他との比較

Xen, qemu, qemu+kqemu, openvz, vserver, user-mode-linux などと機能を比較してみましょう。と思ったらすでにやられていたので、この URL を参照してください。`http://virt.kernelnewbies.org/TechComparison`

8 Bug Squashing Party

岩松 信洋

8.1 はじめに

今回 Debian JP Project で Bug Squashing Party を行いました。その内容と報告をまとめました。

8.2 Bug Squashing Party とは？

Bug Squashing Party[*24] とはバグつぶし大会のことです。Debian の場合だと、stable のリリース前や定期的に行われています。目的としては Release Critical Bug を減らしたり問題のあるパッケージを集中的に直したりします。

8.2.1 Bug Squashing Party の重要な点

Bug Squashing Party は以下の点を決める事が重要です。

- 日時
  Bug Squashing Party をやる日時と開催期間を決める必要があります。
- 場所
  Bug Squashing Party を行う場所を決める必要があります。IRC だけでなく、場所を借りて Face to Face で話合いながら行う必要もあるからです。
- コーディネーターおよび指揮官
  Bug Squashing Party を円滑に進めるため、指揮官が必要です。バグの情報を把握し、参加者に指示を行います。Debian の場合だと、Debian Release Manager や Release チームと話を進めることがあります。このような場合には Release チームに名前が知られている人にコーディネーターになってもらったほうが話が進めやすくなるかもしれません。

8.2.2 コーディネイター、指揮官が行うこと

コーディネイター、指揮官は、以下の事について知っている必要があります。

- Bug Squashing Party の目的を把握する
  ただ単にバグを潰すだけではなく、今回は input method に関してのバグを潰す、といった目的があると全体の流れをコントロールしやすいと思います。
- 他で行われている Bug Squashing Party の主催者やコーディネイターと話をする。
  他で Bug Squashing Party をやっている可能性があるので、話をして無駄な作業を減らすよう動く必要があります。
- バグ対策の指示

[*24] http://wiki.debian.org/BSP

参加者にバグの対策を指示します。

8.2.3 参加者が事前に知っておくこと

参加者はただ単にバグを潰すという作業をするだけではなく Bug Squashing Party はどのようなものなのか理解しておく必要があります。

また、Debian の場合は NMU[*25] を行う可能性が高いので、NMU 時のパッケージバージョンの付け方や Changelog の書き方を知っておく必要があります。もちろん Debian Policy なども事前に読んで勉強しておく必要があります。:-)

参加者の中で Debian Developer の方がおられる場合は、Debian Developer ではない人が修正したパッケージのアップロードしたり、いろいろ助言等をしていただけると助かるかもしれません。

8.3 今回行われた Bug Squashing Party

8.3.1 今回の Bug Squashing Party の目的

今回の Bug Squashing Party の目的は以下のものがあります。

- etch リリース前なので、RC バグを潰したい
- Bug Squashing Party ってどのようなものなのか、ためしにやってみる
- Debian JP Project はちゃんと活動しています、というアピール

8.3.2 今回の Bug Squashing Party の流れ

どのような流れ、内容でおこなったのか以下にまとめました。

1. 事の発端
   OSC 2006 の帰りに gotom さん、上川さん、えとーさん、小林さん、私でサイゼリアで食事していたところ、上川さんが「Bug Squashing Party やりたいねぇ。」から始まったが、この時点でだれが話を進めることにするか決まらない。
2. Debian JP IRC ミーティングで議論
   Debian JP で行われたミーティングで、私が話を進めることになる。
3. debian-devel@debian.or.jp で提案
   Debian JP Project が提供している開発用メーリングリストで提案。[*26] この投稿によるスレッドで武藤さんから助言を頂く。
4. コーディネイターを決める
   Bug Squashing Party を進めるコーディネイターをして、gotom さんに依頼。快諾していだく。
5. 開催日時を決める
   ちょうど 11/23 (木) が休みだったので、この日にする。朝弱い人のために 10 時から行うことにした。このあたりは IRC で決めた気がするが、ログがない。開催内容は以下のとおり。

[*25] Non Maintainer Upload
[*26] [debian-devel:16525] Bug Squashing Party について

日時
2006 年 11 月 23 日 (木) 10:00 - 15:00

会場
IRC
IRC サーバー : osdn.debian.or.jp
IRC チャンネル : #debianjp-bsp
漢字コード : iso-2022-jp

コーディネイター
後藤 正徳さん

6. 開催告知

   debian-users@debian.or.jp および debian-devel@debian.or.jp に開催告知を流した。しかし、反応なし。

7. 開催

   無事開催。しかし人の集まりが悪い。人が集まり始めると gotom さんがいろいろ指示をしてくれて、順調に Bug Squashing Party は進んだ。

   進めるにあたって、参考にしたサイトは以下のとおり。

   - Debian RC バグ曲線

     `http://bugs.debian.org/release-critical/`
   - 現在の RC バグ情報

     `http://bugs.debian.org/release-critical/debian/all.html`
   - リリースチームが使っている RC バグ管理サイト

     `http://bts.turmzimmer.net/details.php`

8. 終了

   15 時に終了。かなり不完全燃焼。NMU するまでが Bug Squashing Party です。

8.3.3 結果

今回行われた Bug Squashing Party の結果は以下の通りです。

- 参加者

  gotom, iwamatsu, ay_, Henrich, mino, dancerj, nori1, kmuto, omote_bot, sato_at_deb-newb, tyuyu
- 活動内容
  - xxdiff 修正 Bug#399764[*27]
  - qemu は問題なし Bug#399382[*28]は close しても問題ない
  - murasaki Bug#378318[*29] hppa only
  - libflash-mozplugin Bug#399508[*30] arm/ia64 only
  - rarpd Bug#395739[*31]
  - cowdancer Bug#384132[*32]
  - zoph Bug#398637[*33]
  - リリースノート追従
  - po-debconf 更新
  - rsjog
  - bookview 最新版ビルドしていま作業中、arm fix 失敗

[*27] `http://bugs.debian.org/399764`
[*28] `http://bugs.debian.org/399382`
[*29] `http://bugs.debian.org/378318`
[*30] `http://bugs.debian.org/399508`
[*31] `http://bugs.debian.org/395739`
[*32] `http://bugs.debian.org/384132`
[*33] `http://bugs.debian.org/398637`

8.3.4 感想

Bug Squashing Party が終わったあと、IRC で意見交換を行いました。

- 5 時間は短い。重いバグが残っているので、簡単なバグフィックスではないので、終わらない。
- 普段手をかけていないアーキテクチャのマシンのアップデートだけで数時間かかる。　（普段から手をかけてメンテナンスしてあげましょう … ）
- IRC ですることに関しては問題ない。
- Bug Squashing Party とは、という説明がないので今後準備したいね。
- IRC なら参加していなくても、ちょくちょく様子がみられてうれしい。
- changelog に closed by Debian JP's BSP とか入れたほうがいいかも。

これらの内容は `http://wiki.debian.org/BSP/DebianJP` としてまとめてあります。

8.3.5 その他の Bug Squashing Party

これは特に Debian だけの特別なイベント（行事？）ではなく、各ディストリビューションやプロジェクトで行っています。

- NetBSD
  The NetBSD Bugathon `http://www.netbsd.org/hackathon/`
- OpenBSD
  hackathon `http://en.wikipedia.org/wiki/Hackathon`
- Gentoo
  Bug day `http://bugday.gentoo.org/`
  毎月第 1 土曜日は Bug day となっている。
- Gnome
  Bug day `http://live.gnome.org/Bugsquad/BugDays`

俺一人 Bug Squashing Party を行ったりしている人もおられるようです。

8.4 まとめ

今回、Bug Squashing Party 開催に関係したのですが、開催までは意外と簡単だったように思います。（協力者がたくさんおられたから、というのが一番大きいですが。）今回の開催時間は 5 時間と短かく、かなり中途半端でした。他の Bug Squashing Party では 24 時間とか平気でやっているようなので、今後は長い時間確保して行いたいと考えています。また、試験前の学生のようにならず、定期的に行っていけたらいいと考えています。

9 apt-torrent

岩松 信洋

Debian パッケージのレポジトリを bittorrent ネットワーク上に置き、それを apt で取得するというものが apt-torrent です。

まだ Debian としては頒布されておらず、フランス人の方が一人でシコシコやっているようです。http://sianka.free.fr/ で開発が行われています。

9.1 bittorrent とは

bittorrent とは、P2P を用いたファイル転送用プロトコルとその通信を行うソフトウェアの事を指します。特徴としては、P2P でファイルの一部をお互いに送受信しあうというプロトコルになっているところです。

通常の P2P ソフトウェアではファイルを提供元に集まるようになっていますが、このプロトコルを使うことにより、ネットワーク帯域のないピアもファイルの配布に協力できるようになっています。

このプロトコル上で Debian パッケージを頒布し、apt で取得するようにしたものが apt-torrent です。

9.2 使い方

9.2.1 ダウンロード

先にも書いたように Debian としては頒布されていません。以下のサイトからダウンロードし、使用します。

http://sianka.free.fr/download/apt-torrent_0.5.0-1_all.deb

apt のソースとして利用できる apt-line も提供されており、以下の apt-line を/etc/apt/sources.list に追記して使用可能です。

- debhttp://sianka.free.fr/apt-torrenttestingmain
- deb-srchttp://sianka.free.fr/apt-torrenttestingmain
- debhttp://sianka.free.fr/apt-torrentunstablemain
- deb-srchttp://sianka.free.fr/apt-torrentunstablemain

9.2.2 設定

パッケージをインストールすると、自動的に /etc/apt/sources.list に追記されます。

以下が追加された apt-torrent 用の apt-line です。

```
### BEGIN APT-TORRENT SOURCE LIST
# Do not edit or remove the markers, they are used by the apt-torrent package

# Uncomment one of the following:
# deb http://127.0.0.1:6968/debian/ unstable main
# deb http://127.0.0.1:6968/debian/ testing main

### END APT-TORRENT SOURCE LIST
```

自分の環境に合わせて apt-line のコメントを外します。

9.2.3 apt-get update してみる

apt-torrent 用の apt-line だけ有効にし、apt-get update を実行してみます。

```
iwamatsu@chimagu:~$ LANG=C sudo apt-get update
Get:1 http://127.0.0.1 unstable Release.gpg [189B]
Get:2 http://127.0.0.1 unstable Release [776B]
Ign http://127.0.0.1 unstable/main Packages/DiffIndex
Get:3 http://127.0.0.1 unstable/main Packages [27.9kB]
Fetched 28.9kB in 3s (8118B/s)
Reading package lists... Done
```

9.2.4 何かパッケージをインストールしてみる

いつも使っている apt-get / aptutide / dselect で torrent ネットワークから Debian Package を取得することが可能です。

しかし、実はどのようなパッケージが torrent 上にあるのか分りません。ドキュメントを読む限り先の URI のサイトで公開されている apt 用のレポジトリ `debhttp://sianka.free.fr/debianunstablemain` にあるものが公開されているようです。

apt のフロントエンドである aptitude を使い、インストールしたときの例を以下に示します。

```
# aptitude install frozen-bubble
Reading Package Lists... Done
Building Dependency Tree
Reading extended state information
Initializing package states... Done
Reading task descriptions... Done
The following NEW packages will be automatically installed:
 fb-music-high frozen-bubble-data
The following packages have been kept back:
 galeon galeon-common
The following NEW packages will be installed:
 fb-music-high frozen-bubble frozen-bubble-data
0 packages upgraded, 3 newly installed, 0 to remove and 2 not upgraded.
Need to get 12.1MB/12.2MB of archives. After unpacking 17.3MB will be used.
Do you want to continue? [Y/n/?]
Writing extended state information... Done
Get:1 http://127.0.0.1 unstable/main fb-music-high 0.1.1 [6950kB]
Get:2 http://127.0.0.1 unstable/main frozen-bubble-data 1.0.0-6 [5155kB]
Fetched 12.1MB in 24s (488kB/s)
Selecting previously deselected package fb-music-high.
(Reading database ... 208931 files and directories currently installed.)
Unpacking fb-music-high (from .../fb-music-high_0.1.1_all.deb) ...
Selecting previously deselected package frozen-bubble-data.
Unpacking frozen-bubble-data (from .../frozen-bubble-data_1.0.0-6_all.deb) ...
Selecting previously deselected package frozen-bubble.
Unpacking frozen-bubble (from .../frozen-bubble_1.0.0-6_i386.deb) ...
Setting up fb-music-high (0.1.1) ...
Setting up frozen-bubble-data (1.0.0-6) ...
Setting up frozen-bubble (1.0.0-6) ...

Reading Package Lists... Done
Building Dependency Tree
Reading extended state information
Initializing package states... Done
Reading task descriptions... Done
```

2007/01/18 現在では正常にパッケージが取得できないため、作者に連絡取り中です。

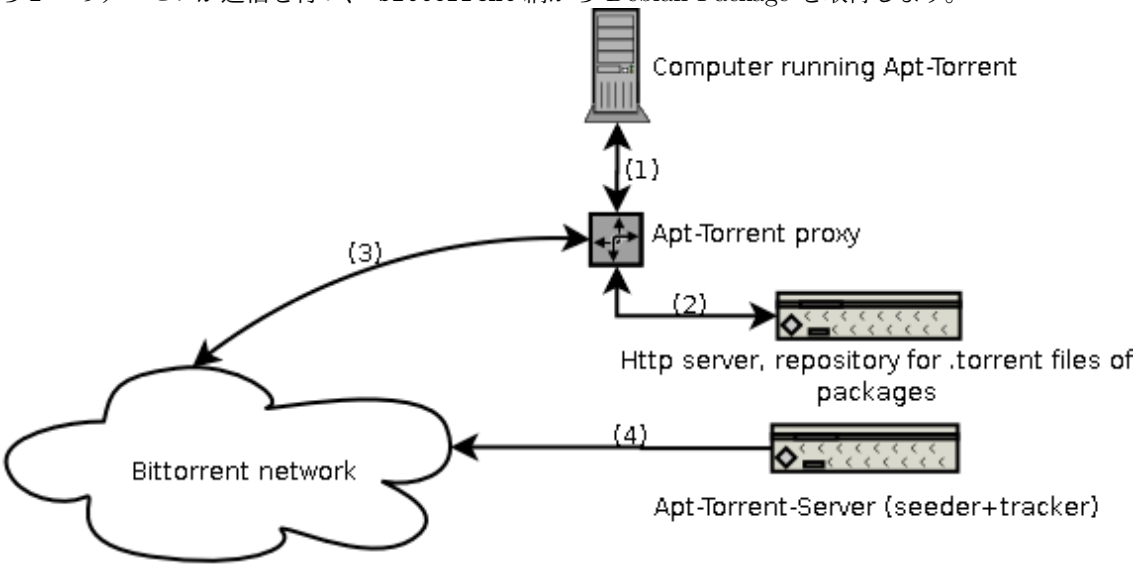
9.3 仕組み

`apt-torrent` は torrent ネットワーク用のデーモンと、パッケージを取得するデーモンの二種類があります。プロセスを見ると、`apt-torrent` のプロセスがあることがわかります[*34]。

```
23404 ?        Ssl    0:00 /usr/bin/pike7.6 /usr/bin/apt-torrent
23411 ?        S      0:00 /usr/bin/pike7.6 /usr/bin/apt-torrent-httpd
23412 ?        S      0:00 /usr/bin/python /usr/bin/btlaunchmany /var/cache/apt-torrent --max_upload_rate -1 --parse_dir_interval 7
23644 pts/0    R+     0:00 ps ax
```

[*34] btlaunchmany は複数の seed 管理用デーモン

apt-torrnet は ファイルを取得するデーモン、apt-torrent-httpd はデータを管理する httpd server です。これら 2 つのデーモンが通信を行い、 bittorrent 網から Debian Package を取得します。



簡単な流れは以下のようになります。

1. クライアントで apt-get update を行います。
   行うと、apt-torrent-proxy を介して、通信を行います。
2. apt-torrent-proxy を介して apt-torrent 用の http server と通信します。
   通信を行い、/etc/apt/sources.list に指定してある apt-line のサーバーから情報を取得を試みます。
3. bittorrent 網 にアクセスします。
4. apt-torrent server から一覧を取得します。
   apt-torrent は seeder[*35] と tracker[*36]を行っている apt-torrent にアクセスします。
5. apt-get install xxxxxxxx を行います apt-torrent-proxy を介して、seeder からデータの取得を試みます。取得したデータは Debian Package なので apt が後は勝手に処理をします。

apt-torrent 用 http server を介して bittorrent 網にアクセスしているので、http で公開されている apt-line からパッケージを取得しているのと変わらない動きをします。

9.4 apt との比較

今では、apt-get 時に http / ftp サーバーに負荷が集中していますが、apt-torrent を使用することによってひとつのパッケージを ピア同士で共有することが可能になります。apt-torrnet は http / ftp サーバーに負荷がかからなくなるひとつの方法になるのではないかと個人的に考えています。変化が激しい unstable / testing は無理としても、stable のパッケージを apt-torrent で頒布するのはいい方法だと思います。今後、開発者と連絡を取り合い、自宅でも apt-torrent server を立てて、実験してみようと模索しているところです。

9.5 その他

同じような機能を持った Winny のプロトコルを使い、apt-winny を実装してみると面白いと思いました。2ch で winny の Linux 版である Linny を開発しているようなので（コードはまだない。）期待したいと思います。

[*35] 対象ファイルのデータを全て持っているピア
[*36] bittorrent ファイルを管理するサーバー

10　プロジェクトトラッカーの勧め

矢吹 幸治

10.1　プロジェクトトラッカーとは何か

プロジェクトトラッカーは、ある作業にかかった時間を記録するためのプログラムです。

10.2　なぜトラッカーを使うのか

このツールは、作業時間を計測するツールです。このトラッカーを使うことにより、

1. 自分の能力を高める。
2. ある作業を開始して完了するまでの時間を予想しやすくする。
3. 自分の時間の使い方を見直すことができる。

という利点があります。欠点としては、自分の作業を記録するのがめんどくさい という点があります。しかし記録する利点に比べれば、かける手間は引き合います。

また、他人に私を観察してもらうよりは、自分で私を観察するほうが、己の自尊心のために安全です。— 「他人に指摘してもらう」ことは多くの人にとって苦痛で、受け入れ難い事です。

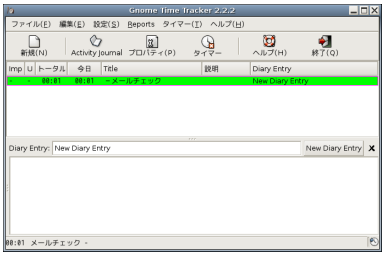
10.3　Debian 4.0(“ Etch ”) で利用できるトラッカー (Tracker) は?

以下のパッケージを利用することができます。

- gnotime – GTK ベース
- karm – Qt ベース
- wmwork – X ベース
- worklog – CUI ベース

10.3.1　gnotime

GTK で書かれた、GNOME と親和性の高いプロジェクトトラッカーです。日本語も使えて、活動履歴 (Activity Journal) も出力できます。Desktop で GNOME を使っているなら、お勧めです。



10.3.2　karm

Qt で書かれたプロジェクトトラッカーです。たぶん便利だと思います。今回は時間切れで試すことができませんでした。私は KDE 使いじゃないので、誰か小ネタでいいので発表で使い勝手レポートをしてくれるとうれしいです。

10.3.3 wmwork

X があれば動く、プロジェクトトラッカーです。

インストール方法は、

```
aptitude install wmwork
```

Small is beautiful。シンプルで Windowmaker や blackbox などのシンプルな window manager と相性が良さそうです。

設定は、wmwork が起動していないときに、 /.wmwork/wmworklog を編集します。詳細は、/usr/share/doc/wmwork/配下のドキュメントを参照してください。以下は、/usr/share/doc/wmwork/exsample/worklog を引用しました。

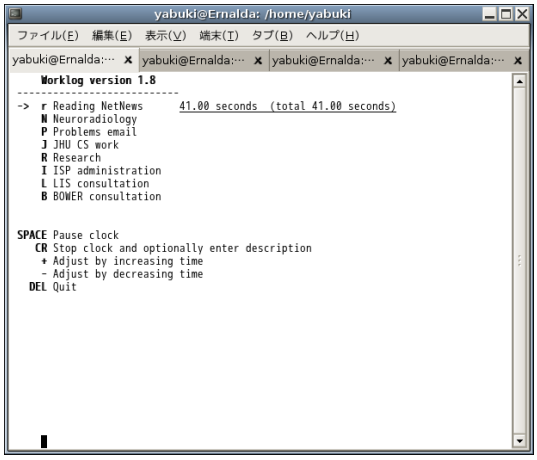
```
1# sample wmwork configuration file
2# do not edit while wmwork is running
3#
4# you may save this file as an initial ~/.wmwork/worklog
5
6A02:0:comment here
7PSI:0
8TEST:0:only 'TES' will be shown
```

上記の設定ファイルをみるとわかりますが、最初の 3 letter が wmwork の表示に使われます。識別子なのかもしれません。また、この最初の部分に指定したファイル名でログが取られます。次に費した時間 (秒単位)、最後にコメントという順です。

~/.wmwork/以下にロックファイルが作成され、2 重起動ができないようになっています。

10.3.4 worklog

CUI で動かすことができます。ssh などでサーバに入って作業をする場合に便利かもしれません。




```
aptitude install worklog
```

このプログラムには、起動するまでに設定ファイルを作成しておく必要があります。詳しくは、/usr/share/doc/worklog/のファイル群を読んでください。設定ファイルの書式は、

```
キー:割り当てる名前
```

です。例えば

```
B:BOWER consultation
L:LIS consultation
R:Research
r:Read NetNews
```

であれば、B,L,R,r のキーで対象となる時間を切替えながら計測できます。残念ながら表示には日本語 (UTF-8) は通りません。(ログには日本語 (UTF-8) がそのまま出ているので lv などでは読むことができました。

/usr/share/doc/worklog/examples/projects に雛型ファイルがあるので、これをコピーして改変して使うのがよいでしょう。

```
alias wl='worklog $HOME/logs/worklog.projects $HOME/logs/worklog.time.log'
```

がお薦めの alias だそうです。確かに、設定ファイルのキーバインド毎にログが生成されるので、ディレクトリを

作っておくのが良いでしょう。

11 サーバをエッチにしてみました

小室 文

11.1 intro

エッチがリリースされてから早一ヵ月たちました。debian-users ml では助けを求める亡者の投稿で ML はかつて無い程の盛り上がりを見せていました (います？現在進行形？)。woody から sarge へのアップグレード時の盛り上がりを知らないのであれですが、初めてのリリースにちょっとした祭を感じました。と言う訳でちょうど GW だったので、サーバ 2 台を sarge からエッチにアップグレードしてみました。

11.2 やること〜リリースノートを読んでみる〜

GW 前後に日本語のリリースノートが出たので、読んでみました。
Debian GNU/Linux 4.0 – リリースノート
`http://www.debian.org/releases/stable/releasenotes`

11.2.1 リリースノートのポイント

1. データや設定情報のバックアップを取り、パッケージ状態のチェック (hold 状態の物は解決をしておく)
2. sarge で最新状態にする
3. エッチで update と upgrade
4. いろいろパッケージをいれる
5. dist-upgrade
6. 最終チェック・再起動・動作確認

11.2.2 エッチの事 (個人的に)

- 全体のパッケージ数 18200〜 / エッチでの新しいパッケージ 6500〜 / sarge から約 65 %のパッケージが更新された
- default encoding が UTF-8 に
- installer が結構変わったらしい
- debian-volatile 公式リリース
- Exim4.5 から Exim4.63 に/ Apache2 から Apache2.2 に / PHP5 / Bind9 から Bind9.3 に

11.3 やってみた

1. バックアップを取る
   /etc/以下、/var/lib/dpkg の中身、dpkg –get-selection '*' の出力、/var/backups など。後あまり影響はないが/home/以下の隠れファイルとかも X など使用している場合はバックアップ対象としていたほうがよい

2. パッケージ状態のチェック & セッションの記録
3. (sarge での) パッケージの更新
   /etc/apt/source.list は sarge のままで aptitude update
   更新候補:libapache-mod-php4 libapache2-mod-php4 libc6 libkrb53 libmagic1 locales man-db php4 php4-comm
4. (エッチでの) パッケージ情報の更新/etc/apt/source.list をエッチ (stable) にして
   aptitude update
   aptitude upgrade
   削除されたパッケージ：libruby1.6 ruby1.6
5. パッケージの確認・インストール
   (a) aptitude install initrd-tools (すでに入っている場合もある)
   新規に入ったパッケージ：libdevmapper1.02 libselinux1 libsepol1 tzdata
   削除されたパッケージ：base-config
   (b) デスクトップ環境があれば
   aptitude install libfam0 xlibmesa-glu x11-common(入っていたら)
   新規に入るパッケージ：libfontenc1 libfs6 libx11-data libxau6 libxdmcp6 libxfont1 xbitmaps xcursor-themes xfonts-encodings xfonts-utils xutils-dev
   削除されたパッケージ：xfree86-common
   (c) カーネルのアップグレード
   aptitude install linux-image-2.6-◎◎ (すでに 2.6 系であれば今すぐに更新をする必要は無い)
6. aptitude dist-upgrade
   アップグレード中に (主に Exim4 と Apache2 設定ファイル) 各種変更部分の選択を迫られる
   使われてないから削除されたパッケージ： libfam0c102 libreadline4 ntp ntp-simple
   新規インストールされたパッケージ：apache2.2-common courier-authlib courier-authlib-userdb cpp-4.1 debian-archive-keyring dmidecode gnupg gpgv laptop-detect libapr1 libaprutil1 libbind9-0 libdb4.3 libdb4.4 libdns22 libedit2 libfribidi0 libgnutls13 libisc11 libisccfg1 libltdl3 liblwres9 libncursesw5 libnewt0.52 libpcap0.8 libpci2 libpq4 libreadline5 libsigc++-2.0-0c2a libslang2 libsp1c2 libsqlite3-0 libssl0.9.8 libtasn1-3 mktemp modconf openbsd-inetd openssh-client openssh-server readline-common sysvinit-utils tasksel-data update-inetd
   削除されたパッケージ：libnewt0.51 libsp1 netkit-inetd ntp-server apache2-common **exim4-daemon-heavy**

7. 確認・再起動
   - /proc/mounts に 'devfs' があれば devfs スタイルのデバイス名の変更をする。
   - lilo をブートローダーとして使用している場合、もう一度実行させておく
     /etc/kernel-img.conf の内容を調べ、do_ bootloader = Yes と書かれていることを確認
   - grub の場合、/etc/kernel-img.conf を編集、update-grub=/sbin/update-grub を/usr/sbin/update-grub に変更する

11.4 怒られた！

以下のパッケージの処理中にエラーが発生しました:
exim4-config postgresql-7.4 postgresql-contrib-7.4 tcsh-kanji postgresql exim4-base postgresql-contrib exim4-daemon-heavy mailagent at exim4 qpopper amavisd-new mutt spamassassin
怒られた原因と解決方法

- debconf: unable to initialize frontend: Gnome
  dpkg-reconfigure debconf にして設定を Dialog に変更。なぜ gnome を選んだのかは自分でも不明です。
- tcsh conflicts with tcsh-kanji
  tcsh-kanji: Depends: tcsh (≥ 6.14.00-6) but it is not installable
  一番分からずはまった。tcsh-kanji depends on tcsh だと思い込んだのとちゃんとエラーを読んでいなかったのが原因。
  aptitude show tcsh
  パッケージ: tcsh
  バージョン: 6.14.00-7
  依存: libc6 (≥ 2.3.6-6), libncurses5 (≥ 5.4-5)
  競合: tcsh-kanji (< 6.14.00-6)
  置換: tcsh-kanji (< 6.14.00-6)
  提供: c-shell, tcsh-kanji
  上記を見る限り tcsh と tcsh-kanji が conflicts しているようなので、tcsh-kanji を remove/purge する
- /etc/exim4/update-exim4.conf.conf: line 32: acl __ check __ helo:: command not found
  昔追加した文を削除して update-config.conf をして/etc/init.d/exim4 reload をする (今は使ってないはず)
- Errors were encountered while processing: exim4-config
  dpkg -l exim4-config では Failed-config になっているので dpkg –configure exim4-config をする
- exim4-daemon-heavy が exim4-daemon-light に replace
  exim4-daemon-heavy が入っているサーバをアップグレードしたら heavy から light に置き換わりました。なぜ。
  以下の新しいパッケージがインストールされます:
  apache2.2-common courier-authlib courier-authlib-userdb cpp-4.1 debian-archive-keyring dmidecode **exim4-daemon-light** gnupg gpgv laptop-detect libapr1 libaprutil1 libbind9-0 libdb4.3 libdb4.4 libdns22 libedit2 libfribidi0 libgnutls13 libisc11 libisccfg1 libltdl3 liblwres9 libncursesw5 libnewt0.52 libpcap0.8 libpci2 libpq4 libreadline5 libsigc++-2.0-0c2a libslang2 libsp1c2 libsqlite3-0 libssl0.9.8 libtasn1-3 mktemp modconf openbsd-inetd openssh-client openssh-server readline-common sysvinit-utils tasksel-data update-inetd
  以下のパッケージが削除されます:
  apache2-common **exim4-daemon-heavy** libnewt0.51 libsp1 netkit-inetd ntp-server

11.5 ついで：エッチを stable にした

もともと testing の時から etch にしていた PC の/etc/apt/source.list を testing から stable に直して aptitude update したらこんなエラーがでました。

```
パッケージリストを読み込んでいます... 完了
W: GPG error: http://security.debian.org stable/updates Release:
公開鍵を利用できないため、以下 の署名は検証できませんでした:
NO PUBKEY A70DAF536070D3A1
W: GPG error: http://cdn.debian.or.jp stable Release:
公開鍵を利用できないため、以下の署名は検 証できませんでした:
NO PUBKEY A70DAF536070D3A1 NO PUBKEY B5D0C804ADB11277
W: これらの問題を解決するためには apt-get update を実行する必要があるかもしれません
```

なので

```
gpg  --keyserver wwwkeys.eu.pgp.net --recv-keys A70DAF536070D3A1
gpg --keyserver wwwkeys.eu.pgp.net --recv-keys B5D0C804ADB11277
gpg --armor --export A70DAF536070D3A1 | apt-key add -
gpg --armor --export B5D0C804ADB11277 | apt-key add -
```

とするとエラーが出なくなりました。/etc/apt/source.list で指定している名前が違う＝パッケージも違う？のでしょうか。

11.6 まとめ

エッチにアップグレードはそんなに難しくない、という事で。リリースノートと aptitude show パッケージ名があれば大方のトラブルには対応出来るかと思います。

12 最近 pbuilder ってどうよ?

上川 純一

この文書は pbuilder とは何か、そして、最近は何がおきたのか、そしてこれから近い将来になにがおきることが予測されるのかということを紹介する記事です。

Debconf7 で紹介する予定の内容です。

12.1 pbuilder の利用コンセプト

pbuilder は chroot 内部で利用するベースファイルシステムイメージを管理し、ビルドのたびに新しいベースファイルシステムイメージを展開することを通して、クリーンルーム環境で Debian パッケージの試験をするのを簡便にします。

基本操作のためのコマンドがいくつかあります。`pbuilder create`、`pbuilder update`、`pbuilder build`[*37]命令がよく利用される例です。詳細な情報が必要であれば、pbuilder のマニュアルを参照してください。`/usr/share/doc/pbuilder/pbuilder-doc.html` にあります。

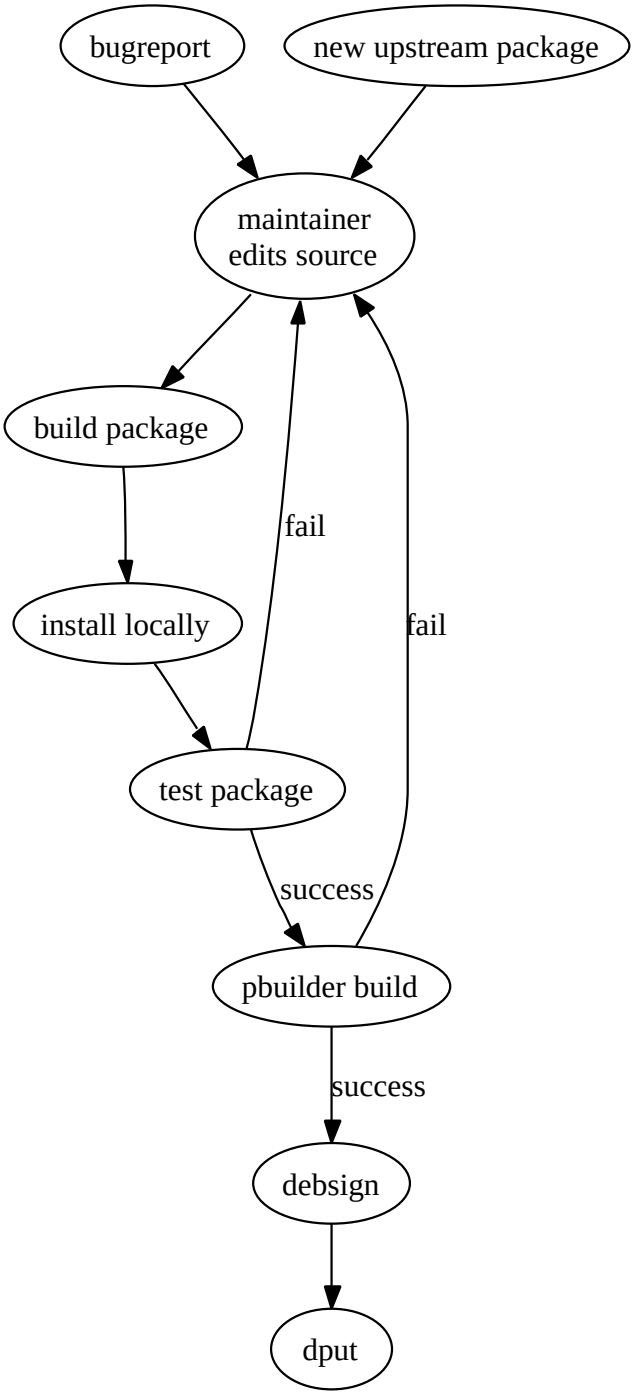
設定が適切に行われ初期化が終了していれば、`pbuilder build` コマンドは`.dsc` ファイル (Debian のソースパッケージ) をあたえられると chroot 内部でパッケージをビルドします。

表 1 コマンドの意味

操作	操作頻度	意味
create	最初に base.tgz を作成するときに一度	ベースファイルシステムの作成
update	一日二回 (unstable のアップデートに伴う)	ベースファイルシステムの更新
build	パッケージビルドのたび	Debian パッケージを chroot 内部でビルドする

Debian Developer の通常のある一日に発生するイベントを検討してみましょう。

[*37] pdebuild 命令のほうが便利な場合があります



pbuilder はパッケージのビルドのプロセスの一部、「確認プロセス」に組み込まれています。[*38] これは、Build-Depends を正しく設定できているのか試験するのに便利です。また簡単な回帰テストのフレームワークとして活用できます。

12.2 pbuilder 自身の開発の仕組み

pbuilder 自身がどう開発されているのか解説します。現在、 alioth で提供されているリソースを活用して co-maintain(共同メンテナンス) されています。最近の主要な開発メンバーは Loïc Minier と上川です。たまに Matt

[*38] これは作業フローの一例で、全員がこういう作業フローになっているというわけではありません。例えば、一部の開発者はローカルでビルドするということをせず、chroot 内部で全部の作業を完結している場合もあり、その場合はローカル環境でのビルドとテストのステップが省略されます。利点の一つとしてはローカル環境に sid の環境をもたなくてすむことがあげられますが、その点については sid を自分自身でも利用しないことになるため、賛否両論です。

Kraai と Mattia Dongili がコミットします。

プロジェクトページは `http://alioth.debian.org/projects/pbuilder` にあり、ホームページは `http://pbuilder.alioth.debian.org/` にあります。ホームページは pbuilder マニュアルになっています。

ソースコードの管理には git を利用しています。レポジトリは下記のコマンドのいずれかを利用してチェックアウトできます。[*39]

```
git-clone git://git.debian.org/git/pbuilder/pbuilder.git
git-clone http://git.debian.org/git/pbuilder/pbuilder.git
git-clone ssh://git.debian.org/git/pbuilder/pbuilder.git
```

12.3 派生物とその状況

pbuilder にはいくつかの派生物があり、異なるバックエンドをサポートしています。それらは別の方法をクリーンルーム試験環境を提供するのに利用しています。それらを簡単に紹介します。

12.3.1 LVM スナップショット版

誰かが LVM スナップショットを base.tgz の管理用に利用する仕組みを提案しました。どっかにメールで投稿されています。ただ、誰も採用して開発を継続しようとはしていないようです。環境の分離方法は chroot を利用しています。LVM スナップショットの利点としては、 tar アーカイブの展開より格段に高速だという点があげられます。

12.3.2 user-mode-linux 版

pbuilder-uml が存在します。どうやらほとんどの人が利用できているようです。Mattia Dongili たちがこの移植版の開発に携わっています。

base.tgz の展開のかわりに UML cow デバイスでクリーンルーム環境の維持が実現されています。また、 chroot のかわりに user-mode-linux を活用しており、結果として各種システムコールが遅くなり全体としては実行オーバヘッドがあります。

12.3.3 cowdancer 版

上川が　cowdancer 版の作業を 2005 年くらいから行っています。安定しているようです。base.tgz の展開が `cp -la` におきかわっており、高速です。

ただし、 cowdancer が libc のコールをフックして実現しているため、一部のパッケージのビルドに影響が出る可能性があります。[*40]

12.3.4 qemu 版

上川が qemu/kqemu/kvm 版の開発を 2007 年初頭から始めました。QEMU の COW ブロックデバイス機能を活用しているため、base.tgz の展開が不要になります。

qemu 版は別アーキテクチャ向けのビルド (クロスビルド) 機能を提供するという特徴があります。たとえば i386 マシン上で ARM 用のパッケージをビルドすることだってできるはずです。

12.4 さらなる開発のアイデア

12.4.1 インストールテスト

インストールテストについては、いくつかの piuparts のようなプロジェクトがあり、pbuilder でも応用できそうです。コンセプトを実装する簡単な例としてのスクリプトは pbuilder で提供しています。`/usr/share/doc/`

[*39] ssh アクセスには alioth のアカウントが必要です

[*40] Etch のリリースの際には、残念ながら Bug 413912 のような問題が発生しました。

pbuilder/examples/execute_installtest.sh です。

```
pbuilder execute \
  /usr/share/doc/pbuilder/examples/execute_installtest.sh \
  pbuilder
```

このコマンドは 指定したパッケージを chroot 内部で　apt-get でインストールしようとし、成功するかしないかを確認してくれます。

12.4.2 パッケージのテスト

パッケージのテストの機能は重要です。とくに、開発者の時間は限られており、手動でテストを繰り返すというのは楽しいことではないからです。pbuilder は例としてフックスクリプトを提供しています。/usr/share/doc/pbuilder/examples/B92test-pkg はパッケージのビルドが成功した場合に、テストを実行するようになっています。

テストファイルは debian/pbuilder-test/NN_name (NN は数字です) におきます。ファイル名は run-parts の標準に従います。[*41].

12.4.3 aptitude

pbuilder は現在 apt-get コマンドを活用しています。しかしながら、aptitude の普及に伴い、aptitude を活用する方法を考える時期にきているかもしれません。

12.4.4 apt-key support

pbuilder はあいかわらず apt-key をサポートしていません。現在の stable リリースで apt-key が提供されているため、そろそろ apt-key をサポートしようかな。

12.4.5 build-dependency parser

Build-Depends を解析するパーサは古く、最適ではありません。それをうけて、Loïc Minier はいくつかの再実装を試行しています。

12.4.6 buildd.net のような仕組みのサポート

pbuilder はたくさんのログを提供するのですが、pbuilder 自体は歴史の概念をもちあわせていません。pbuilder 単体ではログを集めて活用するというようにはなっていません。[*42] 過去のビルドログをローカルに集めて、各ビルド間での差分を確認することを通して、問題を検出できるかもしれません。debdiff などのツールと組み合わせて利用するとよいかもしれませんね。

12.5 References

- http://pbuilder.alioth.debian.org/ か /usr/share/doc/pbuilder/pbuilder-doc.html: pbuilder マニュアル
- cowdancer パッケージ
- piuparts パッケージ
- autodebtest: Ubuntu の自動テストシステム
- schroot / dchroot
- buildd

[*41] ファイル名に'.' が入っていたら無視されますよ!

[*42] pbuildd を作成するというプロジェクトは存在しました。最近どうなってるのかについては把握していません。

13 Debian on SuperH

岩松 信洋

今年に入って、ちまちまと Debian の SH へのポーティング を再始動したのですが、Debconf7 の BOF で Debian porting for SH が通ってしまいました。Debconf7 で発表する内容を以下にまとめたいと思います。

13.1 SuperH とは

SuperH(以下、SH) は ルネサステクノロジ `http://www.renesas.com/` が販売している 組込み向けの CPU です。特徴としては以下のものがあります。

- 日本国産
- 低電圧
- 種類が多い
  SH1/SH2/SH2A/SH3/SH3-DSP/SH4/SH4A/SH4AL/SH4AL-DSP .....

携帯電話、HDD コンポ、 液晶テレビ、カーナビゲーションシステム等で採用され、Linux が動作しています。

13.2 歴史

2000 年前後から Debian に SuperH を移植しようとする活動が行われてきました。それらを紹介します。

13.2.1 第 0 次 SH ブーム

約 7 年前、情報処理推進機機構 (旧 情報処理振興事業協会)、[*43] 略称「IPA」の未踏ソフトウェア創造事業に採択され、SH が Linux に移植されました。[*44] [*45]

そして、この時の移植チームのメンバの一人で、Debian Developer である八重樫 剛史 氏[*46]を中心に、X Hacker である石川 睦氏[*47]が Debian に移植を試みました。このときの状況は、

- サポートアーキテクチャは sh(little endian) と sheb(big endian)
- base はできており、コンパイラも当時最新のもの
- ネイティブで作成するのはリソースが足りないため、DODES プロジェクトを立ち上げ、コンパイル

とすばらしいものでした。

過去に行われた Debian Conference 2001 Tokyo, Japan [*48] で石川氏 が Debian GNU Linux on SuperH [*49] として発表されておられます。

[*43] 英語名 Information-technology Promotion Agency
[*44] http://www.ipa.go.jp/NBP/12nendo/12mito/mdata/5-9gh/5-9gh.pdf
[*45] http://lc.linux.or.jp/lc2001/papers/linux-superh-paper.pdf
[*46] yaegashi@debian.org
[*47] ishikawa@debian.org
[*48] http://www.debian.or.jp/community/events/2001/1025-dcj2001/
[*49] http://www.debian.or.jp/community/events/2001/1025-dcj2001/HANZUBON-debian-sh.html

しかし、八重樫氏が Debian-superh ML に移植を行う旨
http://lists.debian.org/debian-superh/2001/12/msg00013.html
を伝えたところ、

- big endian 必要ないのでは？
  http://lists.debian.org/debian-superh/2001/12/msg00014.html
- SH3/SH3eb , SH4/SH4eb の 4 つのアーキテクチャを入れると、サーバーの容量の問題が発生するため、入れること難しいと思う。
  http://lists.debian.org/debian-superh/2001/12/msg00020.html

という問題提議があり、sh3/sh3eb/sh4/sh4eb にアーキテクチャを分けたまま、移植は止まってしまったのでした。
http://lists.debian.or.jp/debian-devel/200011/msg00044.html
当時、ビルドに使われていたマシン以下の通りです。

図 4　Solution Engine

	Solution Engine
CPU	SH7709 (133Mhz)
memory	32MB
Flash	4MB
IDE	PCMCIA slot
Ethernet	stnic

図 5　CAT 709

	Solution Engine
CPU	SH7709 (133Mhz)
memory	32MB
Flash	8MB
IDE slot	CF Slot
Ethernet	なし

その他に Hewlett-Packard 社から販売されていた、Jornada6xx シリーズも使われていたとのこと。

13.2.2　第 1 次 SH ブーム

約 2 年前、SH を採用した NAS 、LANDISK / LANTANK が I/O データさんから販売されました。これらは I/O データさんに在籍しておられる、kinneko さん [*50] および iohack project http://iohack.sourceforge.net の元、開発が行われ Debian パッケージでシステムが構築されていました。安値であり、自由に触ることができるということで、人気があり、各 Linux 雑誌でも取り扱われ、ブームを築きました。これが 第 1 次 SH ブームです。
以下に LANDISK / LANTANK のスペックを示します。
しかし、SH の販売元であるルネサステクノロジはこのブームをうまく活用することができず、そのまま消えていこうとしていたのでした。

13.2.3　歴史は繰り返さないために

終了してしまったように見えた、 SH の Debian への移植ですが、私がパッケージを再ビルドし、再移植を行うことにしました。理由としては、

[*50] http://d.hatena.ne.jp/kinneko/

図 6　LANDISK

	LANDISK
CPU	SH7751R (266Mhz)
SDRAM	64MB
Flash	ROM
IDE	UDMA133 PATA (ACARD ATP865)
Ethernet	10/100Base-T (RTL8139CL + EEPROM 93C46)
USB	USB2.0 TypeA Conn x2 (NEC D720101GJ)
値段	約 35000 円

図 7　LANTANK

	LANTANK
CPU	SH7751R (266Mhz)
SDRAM	64MB
Flash	ROM
IDE	UDMA133 PATA (ACARD ATP865) 2disk support
Ethernet	10/100Base-T (RTL8139CL + EEPROM 93C46)
USB	USB2.0 TypeA Conn x2 (NEC D720101GJ)
値段	約 19800 円

- SH で 容易に使用できるディストリビューションがない。
  gentoo でサポートされているが、ビルドが面倒。
- 仕事でも使えるようにしたい :)
  個人的理由。
- Debian User だから。

私は今回の移植では、以下のポリシーで作成することにしました。

- Debian でサポートするアーキテクチャ
  SH4 のみをサポートします。SH4A 等もすべて SH4 として扱います。
- SH3 サポート
  SH3 と SH4 の大きな違いは FPU があるか、ないか です。SH3 は SH の Linux カーネルでサポートされた math-emu を使って、math をエミュレーションする方法を取ります。もちろん、math-emu を使うと、遅くなりますが、SH3 を採用した CPU がほとんど存在しないため、コストにはならないと考えています。
  他には、cache の違いもあるので、これをうまくトラップできる仕組みを入れる必要があります。
- big endian サポート
  big endian もサポートするすることも考えています。

13.3　現状

13.3.1　開発メンバー

現在、基本的に開発を一人でやっていますが、以下の方々にサポートをしてもらっています。

- 小島先生 binutils SH メンテナ
  SH の分からないところは質問させていただいています。
- 武藤さん
  buildd の構築で相談に乗ってもらっています。
- kinnneko さん
  いろいろ手伝ってもらっています。
- gotom さん
  Debian glibc メンテナ. 開発用のボードは渡したんだけど.....。
- 上川さん
  LANTANK 買わせたんですが.....。

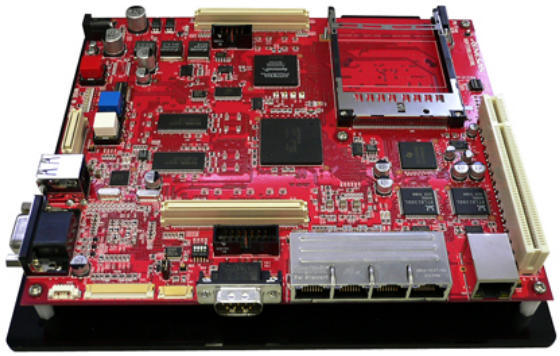
13.3.2 現在使用しているビルドマシン

現在、ビルドに使用しているマシンは以下の通りです。

1. LANTANK x 3

図 8　LANTANK x 3

2. RTS7751R2D



	RTS7751R2D
CPU	SH7751R (266Mhz)
SDRAM	64MB
Flash	ROM
IDE	CF slot and PCMCIA
Ethernet	10/100Base-T (RTL8139CL + EEPROM 93C46)
USB	SM501 USB1.1 (Silicom Motion)

図 9　RTS7751R2D

ルネサスソリューション様から提供していただきました。

13.3.3 パッケージ状況

現在のパッケージ状況は以下の通りです。

- SH4 のみをサポート
  ビッグエンディアンのマシンを持っていないため。
- build-essential ビルド完了

gcc-4.1.2 / binutils / glibc-2.5

- sid debootstrap サポート
  debootstrap できるパッケージができています。
- SH4 buildd 稼働中
  公開はされていませんが、SH4 向けの buildd が稼動中です。今後、buildd.net `www.buildd.net` に登録する予定です。

これらの成果物を `http://www.nigauri.org/~iwamatsu/debian/debian-sh4/` で公開中です。

13.4 今後の課題

- Debian の正式なサポートアーキテクチャにする
- buildd.net への登録
- buildd のメンテナンスが行える体制を作る
  共同メンテナを募集しています。
- 回線の保持
  buildd 用マシンネットワークの確保

13.5 リンク

- SuperH `http://www.renesas.com`
- IRC #debian-superh @ oftc.net
- ML debian-superh@lists.debian.org
- Debian Packages repository `http://www.nigauri.org/~iwamatsu/debian/debian-sh4/`

- iohack project `http://iohack.sourceforge.jp`

14 Debian の情報フロー

上川 純一

Debian etch を活用するためには、何をしたらよいですか? 今日のようにみんなあつまって議論すると、いろいろな新しい発見や情報が出てくるけど、それって普段みんなどうしているの？その疑問を追求するためにワークショップをしてみました。
まず建前から確認してみましょう。

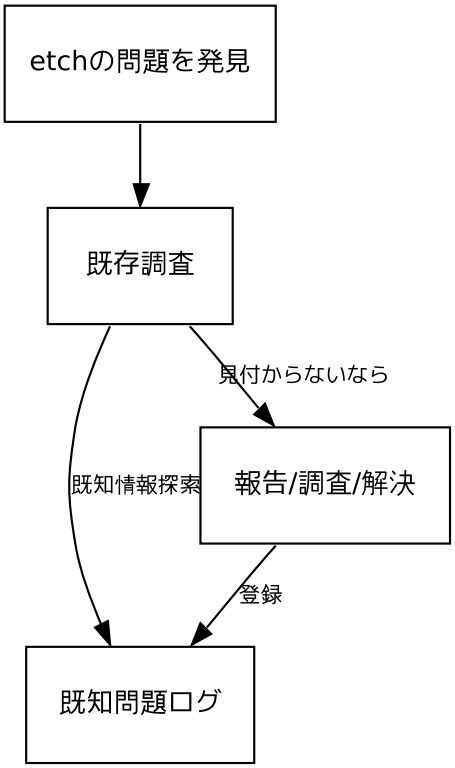
- 基本はリリースノート
- BTS で生きた情報を得る
- debian-users メーリングリスト

しかし実際の運用は少し違うこともあります。理由としては例えば次のようなものがそれぞれあります。

- リリースノート： 一度リリースされてしまうとそう簡単に項目が追加される類のドキュメントではない。
- BTS： 英語なので日本では活用しているメンバーが限られる。また英語が問題とならなかったとしても、BTS の仕組みとして、情報がパッケージ単位で管理されるため複合的に発生する問題は登録しにくい

現状の運用はどうかというと、ワークショップで出てきた話を総合すると、実際は次のような流れになっているようですね。

- IRC でまずきく
- google で検索 (blog とか スレッドテンプレで発見)
- 報告は blog に記述
- 2ch で質問が出たりしたら blog を参照してスレッドテンプレに登録



では、現状を把握したところで、この仕組みのままで改善するか、違う仕組みを考えるか、検討していけばよいはずです。続きはまた。

15 Debian 勉強会資料の作成方法

上川 純一

15.1 レポジトリを取得し、パッケージの準備を行う

レポジトリをチェックアウトします。

```
$ cvs -d :ext:cvs.alioth.debian.org:/cvsroot/tokyodebian .
```

ディレクトリ構成は次のようになっています。

- monthly-report: TeX のソースが全部フラットにおいてあります。ファイル名は、 debianmeetingresumeYYYYMM.tex という名前になっています。プレゼンテーションファイルは debianmeetingresumeYYYYMM-presentation.tex という名前になっています。
  - imageYYYYMM:各月用の画像ファイル
  - debian: デビアンパッケージ用ディレクトリ
- muse: ウェブ（wiki)
- meetinglog: 議事録置場

ビルドに必要なパッケージをインストールします。

```
apt-get install ptex-bin dvipdfmx latex-beamer \
 okumura-clsfiles gs-esp xpdf xpdf-japanese
```

編集に便利なツールもついでにインストールしてみてもよいでしょう。

```
 apt-get install whizzytex advi emacs21 yatex
```

15.2 pLaTeX で文書作成

make コマンド一発で PDF ファイルまで、コンパイルすることができます。Makefile には、あらゆる debianmeeting*.tex ファイルに関して .pdf ファイルを作成するようにルールが作成されています。

注意する点として、印刷を考え、ページ数が 4 の倍数になるようにしてください。

```
SOURCE:=$(wildcard debianmeeting*.tex)
DVIFILES:=$(SOURCE:%.tex=%.dvi)
PDFFILES:=$(SOURCE:%.tex=%.pdf)
all: $(PDFFILES)

%.pdf: %.dvi
        dvipdfmx $<

%.dvi: %.tex
        # check kanji-code of the tex file.
        iconv -f iso-2022-jp -t iso-2022-jp < $< > /dev/null
        platex $<
        platex $<
        platex $<

clean:
        -rm *.dvi *.aux *.toc *~ *.log *.waux *.out _whizzy_* *.snm *.nav *.jqz
```

15.2.1 画像ファイルの処理

画面写真の画像を追加するときは、できるだけサイズの小さい png などを利用してください。グラフなどの線画であれば、eps でかまいません。png であれば、ebb コマンドを利用して bounding box を作成してください。

```
ebb XXX.png
```

ps であれば、 eps2eps でバウンディングボックスを追加してあげるとうまくいきます。sodipodi の出力する ps を eps2eps で処理すれば sodipodi で画像を作成することができます。

15.3 pLaTeX+latex-beamer で文書作成

残念ながら whizzytex でプリビューはうまく動かないです。

がりがりと作成し、xpdf でプリビューしながらがんばって作成してください。

参考：Debian 勉強会のウェブインタフェース

Debian 勉強会のウェブインタフェースについて解説します。

初期は `http://www.netfort.gr.jp/~dancer/column/2005-debianmeeting.html.ja` にあるページを手動で生成していました。

CMS を採用したいところでしたが、CMS を探索している間中ずっと手動で生成しているのも困難なので現状のページにかわりました。`http://tokyodebian.alioth.debian.org/2006-11.html` のようなページになっています。

該当するファイルは CVS レポジトリの muse/ ディレクトリにあります。emacs を wiki 処理系として利用しており、 Makefile から emacs を呼出し、HTML を静的に生成するようになっています。

今後実現していきたい内容としては

- RSS をはくアナウンスページ
- ユーザ参加登録と同時に事前課題登録
- 事後のアンケート
- 登録ユーザへの次回通知
- 事後の資料公開・感想文公開

があります。

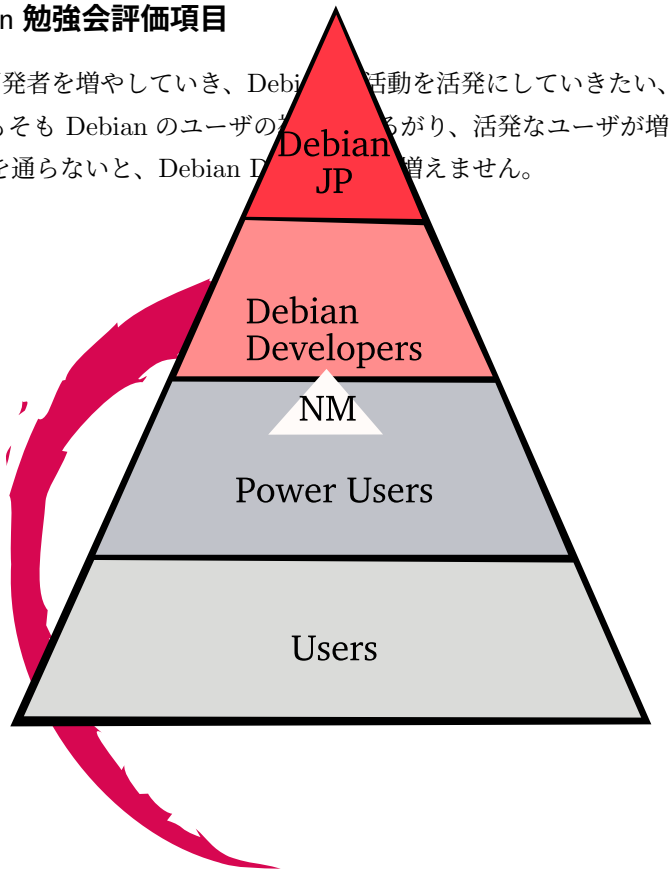
muse-el は一般的な構成ではなく開発もあまり活発でないので、今後利用ツールを変更したいと考えています。

16 Debian 勉強会 2006 年結果統計

上川 純一

16.1 Debian 勉強会評価項目

Debian の開発者を増やしていき、Debi 活動を活発にしていきたい、そういう思いで Debian 勉強会は開催しています。そもそも Debian のユーザの ろがり、活発なユーザが増え、ユーザが開発者になろう、と思って、NM プロセスを通らないと、Debian D 増えません。



Debian 勉強会が成功した、という場合、何がおきた場合でしょうか。

- 直接貢献: バグがどんどんクローズされていき、新機能が追加
- 各種アプリケーションの日本語対応が進捗
- 日本から Debian Developer を生み出す
- 日本の Debian ユーザが増える
- すでに経験の豊富な Debian Developer の知識を展開
- ドキュメントが増える

直接評価できる指標としては下記があるでしょう。

- 日本語で増えたドキュメント数
- Debian Developer の参加者数

- Debian Developer でない参加者数
- 新規の参加者
- 新規に参加して二度以上参加してくれた参加者の数

それでは、値がどういうものか見てみましょう。
概算の値なので、正確ではありません。過去の記録を発掘して、今後の検討のためにひねり出しているものです。

16.2 新規の参加者

- 2005 年 1 月: 20　人
- 2005 年 2 月: 6　人
- 2005 年のこり： 12　人
- 2006 年 -6 月： 9 人
- 2006 年 -10 月: 14 人

16.3 新規に参加して二度以上参加してくれた参加者の数

参加者の統計です。

- 2005 年： 39 人中 21 人
- 2006 年上半期 （-6 月）： 9 人中　5 人
- 2006 年下半期 （-10 月）： 14 人中　2 人

16.4 Debian Developer 比率

のべ参加者の中からのおおよその分析ですが、どれくらいの参加者が Debian Developer で、どれくらいの参加者が New Maintainer Queue に入っているか、の統計です。

- 2005 年：39 人中 DD 4　人?　NM 3　人
- 2006 年 -10 月：36 人中　DD 6 人 NM 6 人

16.5 参加人数

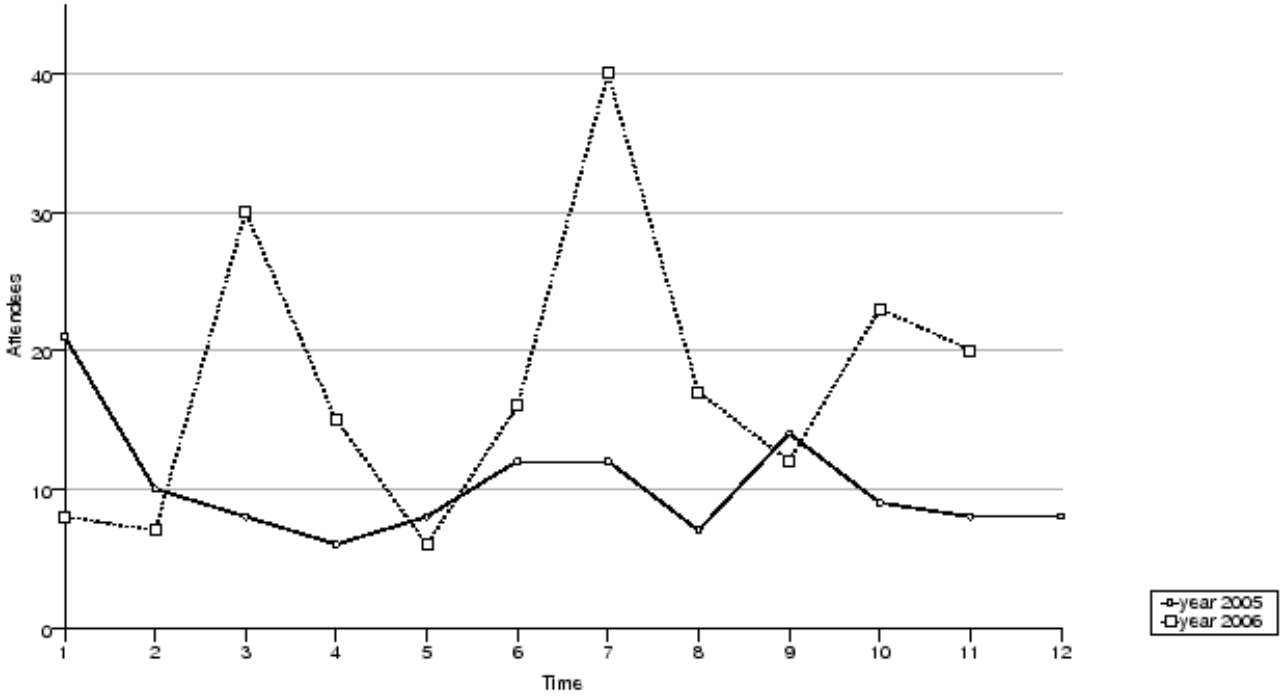


表 2　参加人数 (2006 年)

	参加人数	
2006 年 1 月	8	policy,Debian 勉強会でやりたいこと
2006 年 2 月	7	policy, multimedia
2006 年 3 月	30	OSC: debian 勉強会,sid
2006 年 4 月	15	policy, latex
2006 年 5 月	6	mexico
2006 年 6 月	16	debconf, cowdancer
2006 年 7 月	40	OSC-Do: MacBook Debian
2006 年 8 月	17	13 執念
2006 年 9 月	12	翻訳、Debian-specific、oprofile
2006 年 10 月	23	network, i18n 会議、Flash、apt
2006 年 11 月	20	関西： bug, sid, packaging
2006 年 12 月	14	忘年会

表 3　参加人数 (2005 年、概算)

	参加人数
2005 年 1 月	21
2005 年 2 月	10
2005 年 3 月 (早朝)	8
2005 年 4 月	6
2005 年 5 月	8
2005 年 6 月	12
2005 年 7 月	12
2005 年 8 月	7
2005 年 9 月	14
2005 年 10 月	9
2005 年 11 月	8
2005 年 12 月	8

16.6 実施テーマ

今年は下記のテーマを実施しました。

- Debian weekly news クイズを隔月で
- グループワーク:Debian 勉強会でやりたいこと
- Debian Policy 入門
- Debian Multimedia Project
- Debian 勉強会紹介
- sid のすすめ

- LaTeX
- DebConf2006 報告
- cowdancer
- MacBook Debian
- module-assistant
- oprofile
- 翻訳のすすめ
- Debian-specific
- i18n
- Flash
- Bug tracking system
- Debian packaging.

16.7 会議の構成

今年の Debian 勉強会にはおおきくわけて三種類の会議形態がありました。

- OSC（春、秋）
- 出張（OSC 北海道、メキシコ、KOF）
- 通常

東京の OSC での開催は、通常の勉強会に参加するよりハードルを低く設定しています。できるだけ多くの人たちに参加してもらうことを目的としています。これで Debian 勉強会の雰囲気をしってもらい、興味を持ってもらい、通常の勉強会に参加しやすくなるように配慮しています。

Debian 勉強会は、ふたつの目的をもっています。そのふたつの目的にあわせた会議設定をしています。一つめは Debian 開発者の開拓です。二つ目は Debian ユーザの集まる場所を提供することです。

OSC, KOF など、年三回程度のイベントにおいては、他の主催者の企画したイベントのうえでイベントを実施しています。コストも、集客方法も主催者側に一任しています。また、実務上、事前課題の設定などもできていません。

毎月実施している勉強会については、より目的意識の高い会として位置づけています。Debian 開発の側に参画し、よりドキュメントを生成する側にまわり、みんなで Debian の現在の課題についてブレーンストーミングできるようになることが目標です。そのため、事前課題を準備して、目的を共有できるような仕組を準備しています。

また、現状の勉強会の運営については、毎年一回、12 月開催の勉強会に確認し、来年度の運営方針を確認することにしています。

17 Debian 勉強会 2006 年、作業フロー

上川 純一

どういう作業をしたでしょうか。書き出してみましょう。

17.1 年に一回の作業

- 年次計画を仮決め、毎月どの日に開催するのかを決定して、それをベースにしてその後の議論をする。
- tokyodebian-2006 メーリングリストの作成

17.2 事前準備

- 開催二ヵ月前： 開催場所の予約
- 資料の作成
- 開催二週間前： `http://utage.org/enkai` 宴会君への登録
- 開催二週間前： `http://tokyodebian.alioth.debian.org/` ウェブサイトの更新
- 開催二週間前： mixi と debian-users/debian-devel にアナウンス
- 開催二日前：印刷、資料は 4 の倍数のページ数にして、kinko's にコピー依頼。[*51]
- 開催一日前：宴会場所の予約

17.3 事後処理

- 議事録の作成
- blog トラックバックの収集

[*51] 2006 年 10 月、ウェブベースで依頼してみたところ無事印刷されたので、今後はウェブで依頼する予定。

18　Debian 勉強会 2007 年度計画検討結果

上川 純一

2007　年度の計画を検討した結果次のような内容が出てきました。
Debian が現在市場に提供している付加価値としては下記がある。

- デプロイしやすいしくみであること
- ライセンスが明確であること
- たくさんソフトウェアがあること
- アップグレードが平穏であること
- くだらないパッケージもはいっていること
- ソースコードが簡単にとってこれること
- ユーザが多いこと
- 学ぶことがおおいこと

また外部的な要因として

Windows　VISTA の公開 β テスト開始
Mac OSX　iPhone, Leopard
他のディストリビューション　FC, SUSE, RHEL の新リリースが出る、Ubuntu も出るが、宇宙に再度いってしまうのではないか？
ハードウェア　Quad core, Wii/PS3/.. などが出る
ユーザの期待として　次はさすがに GUI で D-i できるだろう

というのがある。
ユーザの拡大方向としては、下記の案が出てきた。

主婦
学生　教育に Debian を利用するべきだ

また、今年の活動は、来年を考えて動くべきだろう。
以上をふまえての今年度の計画は下記で実施する。

2 小林さん幹事
3 岩松さん担当で OSC
4 えとーさんでエッチインストール大会
5 ごとむさん主催で google 開催（!?）
6 やぶきさんでえじんばら開催
7 Debconf 参加報告会
8 Debian １４周年
9 以降 後で考える。

19 Debian Weekly News trivia quiz

上川 純一

ところで、Debian Weekly News (DWN) は読んでいますか？Debian 界隈でおきていることについて書いている Debian Weekly News。毎回読んでいるといろいろと分かって来ますが、一人で読んでいても、解説が少ないので、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 DWN を読んでみましょう。

漫然と読むだけではおもしろくないので、DWN の記事から出題した以下の質問にこたえてみてください。後で内容は解説します。

19.1 2006 年 41 号

`http://www.debian.org/News/weekly/2006/41/` にある 11 月 28 日版です。
問題 1. Debian Conference 7 日程が決まりました。いつからでしょうか。

☐ A 4/1
☐ B 5/22
☐ C 6/17

問題 2. Debian Project に新しい開発用マシンが導入されました。どのマシンでしょうか。

☐ A Sun Fire T2000
☐ B Sony Playstation 3
☐ C TiVo Series2 DVR

19.2 2006 年 42 号

`http://www.debian.org/News/weekly/2006/42/` にある 2006 年 12 月 26 日版です。
問題 3. Linux Conference Australia で開催される Debian 関係のイベントは今回で何回目か

☐ A 1
☐ B 3
☐ C 6

問題 4. 最近急上昇して Debian 内で 3 位人気のアーキテクチャになったアーキテクチャは？

☐ A ARM
☐ B PPC
☐ C AMD64

問題 5. etch がフリーズされたのはいつ？

□ A 11/11

□ B 12/11

□ C 12/24

問題 6. Debian のインストール CD イメージはどれくらいの頻度で更新されているか？

□ A 毎日

□ B 毎メジャーリリース

□ C 毎マイナーリリース

19.3 2007 年 01 号

http://www.debian.org/News/weekly/2007/01/ にある 1 月 23 日版です。
問題 7. creative commons 2.5 は DFSG 互換か？

□ A ちがう

□ B DFSG 互換です

□ C むしろ GPL とまったく同じ

問題 8. Kenshi Muto のアナウンスによると、Takeshi Yaegashi は何に Debian をインストールするのに成功した?

□ A PLAYSTATION 3

□ B Wii

□ C XBox 360

問題 9. Florian Lohoff はどんな変化に気付いたか?

□ A woody がミラーから取り除かれた

□ B sarge のアーカイブに侵入された形跡がある

□ C etch への開発者の興味が薄れている

問題 10. Joseph Smidt はリリースに関して何を提案したか?

□ A unstable と testing だけをサポートするようにして保守作業を楽にしよう

□ B etch はリリースされないまま古くなりつつあるので適当にリリースして lenny に全力を注ぎ込もう

□ C そろそろ Debian も XP や Vista といったよくわからない名前をリリースにつけるようにしよう

19.4 2007 年 02 号

http://www.debian.org/News/weekly/2007/02/ にある 1 月 30 日版です。
問題 11. Roberto C. Sanchez が提案したのは何か

□ A 全員が自分の誕生日を Debian として祝うべきだ

□ B Debian のスクリーンショットのレポジトリを準備して、それぞれのパッケージに対応させる

□ C Debian を全員使うべきだ

問題 12. Robert Millan が作成した win32 用プログラムは?

□ A Windows Vista を自動的にダウンロードし上書きインストールしてくれるインストーラ

□ B ユーザの手間を省くために、Outlook Express のアドレス帳に載っているアドレスを lists.debian.org のメーリングリストすべてに自動登録してくれるプログラム

□ C Debian Installer を自動的にダウンロード・起動してくれるランチャー

問題 13. 1 月 31 日に締め切られたのは?

□ A 第 24 回東京エリア Debian 勉強会の blog での報告

□ B Debian etch ベータ版のテスターの募集

□ C Debian Conference 7 のスポンサー希望者の参加登録

問題 14. Luis Matos が etch のカーネルに関して提案したのは?

□ A kvm が利用できるように 2.6.20 を使おう

□ B Linux ではなく kFreeBSD を使おう

□ C ハードウェアサポートのためにポイントリリースでカーネルの更新を行おう

19.5 2007 年 03 号

http://www.debian.org/News/weekly/2007/03/ にある 2 月 13 日版です。
問題 15. Debian のウェブサイトについて Manoj Srivastava が気付いたのは?

□ A デザインがやや古びており見た目がイマイチ

□ B 投票ページのナビゲーションバーが長すぎる

□ C 朝起きたらぐるぐるマークが逆回転になっていた

問題 16. 今年の Debian Project Leader 選挙のアナウンスから 4 時間。最初にノミネートしてきたのは?

□ A Junichi Uekawa

□ B Gustavo Franco

□ C Bill Gates

問題 17. Debian Conference 2008 はどこの国で開催されることに決定したか?

□ A イラク

□ B アルゼンチン

□ C パプアニューギニア

19.6 2007 年 04 号

http://www.debian.org/News/weekly/2007/04/ にある 3 月 13 日版です。
問題 18. ウェブアプリケーション関連のパッケージの静的コンテンツはどこにおくべきか？

□ A /var/www に置く

□ B /usr/share/PACKAGE に置く

□ C /srv/XXX に置く

問題 19. Debian Project の MIA アカウントに対して　実施する WaT とは何をするものか

□ A 今年の DPL 選挙に投票しなかった人に対して確認メールを送り反応がない人を引退プロセスに移行する

□ B 気に入らない人を強制退会させる

□ C あれ？Debian Developer だらけの水泳大会

問題 20. etch リリースはどういう暗号鍵で署名されるか？

□ A オンライン鍵とオフライン鍵

□ B オフライン鍵のみ

□ C オンライン鍵のみ

問題 21. Frans Pop がアナウンスした Babelbox は何をするものか？

□ A いろいろな言葉を喋ってくれる

□ B フォントを複数表示

□ C 自動でくりかえし Debian Installer が稼働し、Gnome にしばらくログインしてくれるしくみ

問題 22. DPL 選挙の勝者は？

□ A Iwamatsu

□ B Sam Hocevar

□ C Anthony Towns

19.7 2007 年 5 号

http://www.debian.org/News/weekly/2007/05/ にある 4 月 24 日版です。
問題 23. 3 月 12 日 Alioth で新規に使えるようになったバージョンコントロールシステムはどれか

□ A Mercurial

□ B RCS

□ C git

問題 24. Robert Milan が goodbye-microsoft 0.4.0 の機能として発表したのは何か

□ A Ubuntu 対応

□ B etch 対応

□ C Windows Vista 対応

問題 25. Aurelien Jarno が kFreeBSD の新しいインストール CD を発表したが、対象アーキテクチャは何か

□ A i386

□ B i386 amd64

□ C ppc hppa arm

問題 26. teTeX と TeXLive で何がおきたか？

□ A TeXLive はもう古いので teTeX でおきかえる

□ B teTeX はもう古いので TeXLive でおきかえる

□ C TeX のコンセプトが古いのでもう両方ともやめる

問題 27. Debian etch の CD/DVD イメージは何枚分あるか

□ A 666 枚の CD と 13 枚の DVD

□ B 292 枚の CD と 39 枚の DVD

□ C 1 枚の DVD に全部おさまる

20 仮想化友の会との合同クイズ

仮想化友の会 X Debian 勉強会

番外編として、OSC の仮想化友の会のセッションで利用したクイズです。

20.1 仮想化の常識

問題 28. 仮想化での paravirtualization とはなにか

☐ A 仮想用に OS が変更されている

☐ B パラパラで仮装する

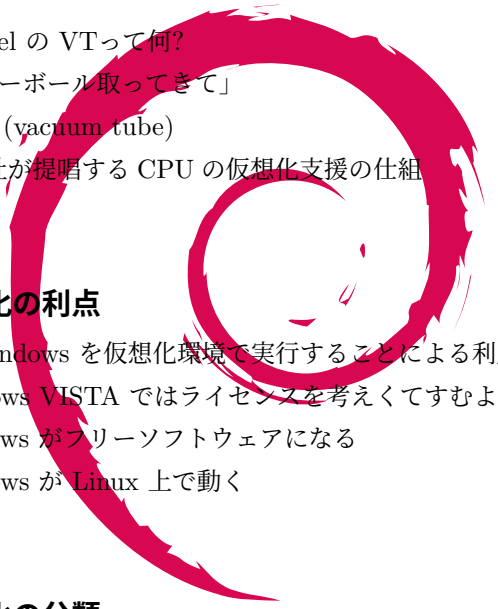
☐ C 並列で仮想化する

問題 29. Intel の VTって何?

☐ A 「バレーボール取ってきて」

☐ B 真空管 (vacuum tube)

☐ C Intel 社が提唱する CPU の仮想化支援の仕組



20.2 仮想化の利点

問題 30. Windows を仮想化環境で実行することによる利点は何か

☐ A Windows VISTA ではライセンスを考えくてすむようになる

☐ B Windows がフリーソフトウェアになる

☐ C Windows が Linux 上で動く

20.3 仮想化の分類

問題 31. 別途カーネルが独立して必要では無い仮想化実装はどれか

☐ A user-mode-linux

☐ B xen

☐ C openvz

問題 32. kvm はなぜカーネルのメインラインにマージされたか?

☐ A 作者がイケメンだった

☐ B 政治力

☐ C 影響範囲のコードが小さい

問題 33. kvm は paravirtualization をどういう方式で実現しているか

□ A 仮想環境で実行される Linux カーネルが paravirt_ops 機構を利用し、VMCALL 命令を発行することでホスト OS に連絡する

□ B 根性・気合い

□ C 愛情

20.4 仮想化の仕組の常識

問題 34. x86 CPU において VMEXIT を発行する命令として、代表例である CPUID 命令の OPCODE は下記のうちどれか。

□ A 0x55

□ B 0x0f 0xa2

□ C 0x5d

問題 35. AMD-V と Intel-VT の一番大きな違は次のどれか

□ A 会社が違う

□ B 命令が違う

□ C 思い入れが違う

問題 36. Xen の Domain-U の U は何か

□ A Unprivileged

□ B User

□ C Unix

問題 37. Xen という名前は何から由来したか

□ A Xeno

□ B 作者の娘の名前

□ C 禅寺

問題 38. KVM はなんの略か

□ A Keyboard Video Mouse

□ B Kernel-based Virtual Machine

□ C 「これ持ってる?」「こんなビデオ持ってるぜ」

問題 39. i386 の場合の Domain-U の ring level は何?

□ A 1

□ B 2

□ C 3

20.5 Debian での常識

問題 40. Debian Project で推奨する仮想化の技術は？

□ A xen

□ B kvm

□ C DFSG に合致するものならなんでもよい

21 Debian Weekly News 問題回答

上川 純一

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

1. C
2. A
3. C
4. A
5. B
6. A
7. A
8. A
9. A
10. A
11. B
12. C
13. C
14. C
15. B
16. B
17. B
18. B
19. A
20. A
21. C
22. B
23. A
24. C
25. B
26. B
27. B
28. A
29. C
30. C
31. C
32. C
33. A
34. B
35. A
36. A
37. A
38. B
39. A
40. C

あんどきゅめんてっど でびあん 2007 年夏号

2007 年 8 月 17 日　　初版第 1 刷発行
東京エリア Debian 勉強会 （編集・印刷・発行）
