あんどきゅめんてっどでびあん

東京エリア Debian 勉強会

2005 年 8 月 14 日

目次

1 debhelper 論 その 1

Debhelper とは何か．存在意義は何か．

1.1 Debian package の構成要件

Debian Package のソースパッケージには下記ファイルが必要です．全てのファイルについては規定のフォーマットがあります．

- debian/rules: パッケージをビルドする手順を記述した Makefile.
- debian/control: パッケージの情報を記述した構成ファイル
- debian/copyright: パッケージの著作権情報を記述したファイル．利用許諾を記述.
- debian/changelog: 変更履歴を記述する．

emacs で編集するのであれば，devscripts-el パッケージをインストールすると編集しやすくなっています．vi を利用しているのであれば，devscripts パッケージを利用すると，各ファイルを編集しやすいです．

パッケージの作成の本質は，ディレクトリ以下にアプリケーションをインストールし，それ以下のファイルを tar でかためて，deb ファイルのなかにいれることです．その他に制御情報もありますが，それについては後日．

例えば，debian/tmp/以下に / 以下にインストールされるはずのファイルを配置することでパッケージを作成できます．debian/tmp/usr/bin/binary-test というファイルを作成して，debian/tmp ディレクトリを指定して dpkg-deb コマンドを実行してできたパッケージをインストールしたら/usr/bin/binary-test にファイルがインストールされます．

1.2 dh-make

アップストリームからのソースパッケージから，Debian 用のパッケージのテンプレートを作成します．

さて，dh-make ではどんなファイルが作成されるのでしょうか．

なんでもないサンプルファイルを作成して試しに実行してみます．

```
[02:11:56]ibookg4:/tmp/dh> find -ls
 94712    0 drwxr-xr-x   3 dancer   dancer         80  1月 27 02:08 .
 94686    4 -rw-r--r--   1 dancer   dancer        156  1月 27 02:07 ./test-source_0.1.orig.tar.gz
 94672    0 drwxr-xr-x   2 dancer   dancer         60  1月 27 02:07 ./test-source-0.1
 94675    0 -rw-r--r--   1 dancer   dancer          0  1月 27 02:07 ./test-source-0.1/Makefile
```

dh-make コマンドをうつと，どんなバイナリを作成するのか，という点を聞かれます．

```
[02:13:42]ibookg4:/tmp/dh/test-source-0.1> dh_make

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

Maintainer name : Junichi Uekawa
Email-Address   : dancer@debian.org
Date            : Thu, 27 Jan 2005 02:13:46 +0900
Package Name    : test-source
Version         : 0.1
Type of Package : Single
Hit <enter> to confirm:
Done. Please edit the files in the debian/ subdirectory now. You should also
check that the test-source Makefiles install into $DESTDIR and not in / .
[02:13:51]ibookg4:/tmp/dh/test-source-0.1>
```

```
[02:27:24]ibookg4:/tmp/dh/test-source-0.1> ls debian/
README.Debian  dirs                   manpage.sgml.ex  rules
changelog      docs                   manpage.xml.ex   test-source-default.ex
compat               emacsen-install.ex  menu.ex            test-source.doc-base.EX
conffiles.ex   emacsen-remove.ex   postinst.ex           watch.ex
control        emacsen-startup.ex  postrm.ex
copyright      init.d.ex           preinst.ex
cron.d.ex      manpage.1.ex           prerm.ex
```

大量にファイルができます．これらのファイルで，.ex で終了しているのはサンプルファイルで，特に必要でないのなら削除します．

debian/rules として，次のようなファイルができます．

```
#!/usr/bin/make -f
# -*- makefile -*-
# Sample debian/rules that uses debhelper.
# This file was originally written by Joey Hess and Craig Small.
# As a special exception, when this file is copied by dh-make into a
# dh-make output file, you may use that output file without restriction.
# This special exception was added by Craig Small in version 0.37 of dh-make.

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1



CFLAGS = -Wall -g

ifneq (,$(findstring noopt,$(DEB_BUILD_OPTIONS)))
        CFLAGS += -O0
else
        CFLAGS += -O2
endif

configure: configure-stamp
configure-stamp:
        dh_testdir
        # Add here commands to configure the package.

        touch configure-stamp


build: build-stamp

build-stamp: configure-stamp
        dh_testdir

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

        touch build-stamp

clean:
        dh_testdir
        dh_testroot
        rm -f build-stamp configure-stamp

        # Add here commands to clean up after the build process.
        -$(MAKE) clean

        dh_clean

install: build
        dh_testdir
        dh_testroot
        dh_clean -k
        dh_installdirs

        # Add here commands to install the package into debian/test-source.
        $(MAKE) install DESTDIR=$(CURDIR)/debian/test-source


# Build architecture-independent files here.
binary-indep: build install
# We have nothing to do by default.

# Build architecture-dependent files here.
binary-arch: build install
        dh_testdir
        dh_testroot
        dh_installchangelogs
        dh_installdocs
        dh_installexamples
#        dh_install
#        dh_installmenu
#        dh_installdebconf
#        dh_installlogrotate
#        dh_installemacsen
#        dh_installpam
#        dh_installmime
#        dh_installinit
#        dh_installcron
#        dh_installinfo
        dh_installman
        dh_link
        dh_strip
        dh_compress
        dh_fixperms
#        dh_perl
#        dh_python
#        dh_makeshlibs
        dh_installdeb
        dh_shlibdeps
        dh_gencontrol
        dh_md5sums
        dh_builddeb

binary: binary-indep binary-arch
.PHONY: build clean binary-indep binary-arch binary install configure
```

また，debian/control として下記のようなファイルができます.

```
Source: test-source
Section: unknown
Priority: optional
Maintainer: Junichi Uekawa <dancer@debian.org>
Build-Depends: debhelper (>= 4.0.0)
Standards-Version: 3.6.1

Package: test-source
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: <insert up to 60 chars description>
 <insert long description, indented with spaces>
```

他に必要なファイルとして，debian/changelog などがあります.

1.3 dh-xxxx のオーバビュー

各論に入るまでに dh-xxxx が一般的にどういう動作をするものなのかを説明します．おおざっぱにいうと二種類の動作形態があります.

Debian パッケージを作成する一連の動作の中で，`debian/`**パッケージ名** というディレクトリにインストール用のイメージを作成します．そのディレクトリに対して，`debian/`**機能**`.`**パッケージ名** というファイルで指定した内容を実施してくれるのが `dh-`**機能** スクリプトです.

また，別の機能として，コマンドラインオプションに指定されたものに対しても操作します.

1.4 簡単なところから, dh-installman

man ページをインストールする dh-installman を例にとって，説明します.

man ページは各パッケージのセクションに応じたディレクトリにインストールします．セクション情報は，man ページの roff ソースの最初のところに記述してあります.

例

```
.TH "pbuilder" 8 "2004 Apr 4" "Debian" "pbuilder"
```

この man ページはセクション 8 のマニュアルなので，FHS に従い，`/usr/share/man/man8/`というディレクトリにインストールする必要があります.

下記の実行例は，インストール対象のパッケージ名を指定して，インストールするファイルを指定した場合です.

```
dh_installman -ppbuilder-uml pbuilder-user-mode-linux.1 \
pbuilder-uml.conf.5 pdebuild-user-mode-linux.1
```

上記のコマンドを入力すると

`debian/pbuilder-uml/usr/share/man/man1/pbuilder-user-mode-linux.1` などにファイルがコピーされます.

debhelper の概念で重要なのは，このインストール先のディレクトリがどこであるかということについては debhelper 側で管理しており，たとえポリシーに変更が発生しても，debhelper のみを変更すればよく，パッケージスクリプト側への変更は最小にできる，という点です.

例えば，マニュアルページの例でいくと，以前の標準では，/usr/man/man1/などにインストールすることがポリシーだったのですが，それが変更になり，/usr/share/man/man1 になりました.

2 Debian Social Contract について

松山節で攻めます

2.1 Debian Social Contractって何だ?

Debian プロジェクト憲章 (v1.2) より[*1]

- 「基本文書 (Foundation Document)」とは,Debian プロジェクトの使命や目的にとって決定的に重要とみなされる文書または声明である.
- 「基本文書」は,「Debian 社会契約 (Debian Social Contract)」および「Debian フリーソフトウェアガイドライン (Debian Free Software Guidelines)」というタイトルの文書である.
- 「基本文書」を更新するためには,3:1 の多数決を必要とする. 新たな「基本文書」の発行や既存の「基本文書」の撤回は, この憲章のリストを修正することによってなされる.

国なら憲法,Debian には Social Contract といったところでしょうか.

2.1.1 「Debian は 100% フリーソフトウェアであり続けます.」

Debian プロジェクトは社会契約において Debian GNU/Linux ディストリビューションを完全にフリーなソフトウェアとして維持することを約束しています. フリーソフトウェアとはどういうソフトウェアなのか, というのは Debian Free Software Guidline(DFSG) で記述されています.

2.1.2 「私たちはフリーソフトウェアコミュニティーにお返しをします.」

DSC には

> 私たちのシステムに含まれているソフトウェアを私たちの「上流」で開発している作者に, バグ修正, 改良, ユーザの要求などをフィードバックします.

とありあます. 最近 ML に Debian デベロッパー同士で「もっと上流の人と密にコミュニケーションとろうよ」的な投稿がありました. 実際のところはどうなのでしょうか.

2.1.3 「私たちは問題を隠しません.」

プロジェクトに報告されているバグは全て参照できます. Debian では開発者間, ユーザ間のやりとりのほとんどがメーリングリストを通して行われるそうです. Debian はバグの情報に関することのみならず, プロジェクトに関するいろんなことを公開しているようです.

[*1] http://www.debian.org/devel/constitution

2.1.4 「私たちはユーザとフリーソフトウェアを大切にします.」

DSC には

> 私たちはユーザーとフリーソフトウェアコミュニティからの要求に従います. 彼らの関心と利益を最優先に考えます. 私たちはさまざまな状況におけるコンピュータ利用環境の運用に関してユーザーの必要を満たすように行動します.

とあります.BTS に「要望」という重要度が用意されているところに, こういった姿勢が現われているでしょうか.

2.1.5 「私たちのフリーソフトウェア基準に合致しないプログラムについて.」

Debian プロジェクトは

> 私たちは,Debian フリーソフトウェアガイドラインに適合していないプログラムを使わなければならないユーザーがいることを認めています.

最初の社会契約との折り合いからか,Debian パッケージアーカイブには main,contrib,non-free というカテゴリを作成し, パッケージを整理しています.

2.2 Debian ニュースに見る Debian の思想

拒否: Debian プロジェクトは Sender ID を採用できません[*2]

> ... 省略... また私たちは, インターネットの核となるインフラストラクチャに関する知的財産権 (IPR) は企業に対して認められるべきではないと考えています.IETF が IPR に関するポリシーを見直し, インターネットの核となるインフラストラクチャが妨げられないよう保証する必要があると信じています. ... 省略...

Debian セキュリティ勧告は CVE と互換性があります[*3]

> ... 省略... 公に知られた脆弱性およびセキュリティ上の問題点のすべてについて名前を標準化することを目的とする Common Vulnerabilities and Exposures (CVE) プロジェクトとの協調的な取り組みの中で, 2002 年 6 月以来, 新たなセキュリティ勧告は CVE 識別番号を含めた形で出されています. Debian は 2003 年 5 月に 正式に CVE との互換性を申請しました. ... 省略...

Debian はデスクトップ Linux コンソーシアムに参加します[*4]

> Debian プロジェクトは, 最近設立され, 非営利法人になる予定の デスクトップ Linux コンソーシアム (DLC) の設立メンバーです. ... 省略... Debian デスクトップ サブプロジェクトは, デスクトップ用途に GNU/Linux を利用しているユーザのために Debian の改善に精力を傾け続けている一例です. ... 省略...

[*2] http://www.debian.org/News/2004/20040904
[*3] http://www.debian.org/News/2004/20040330
[*4] http://www.debian.org/News/2003/20030207

3　debhelper 論 その 2

Debhelper の各コマンドは何者か.

3.1　debhelper コマンド編

全体を見渡すために，debhelper 4.2.31 に含まれているコマンドの一覧を表にしてみます.

ここでインストールといっているのは，ファイルをパッケージビルド用のディレクトリにコピーする，という意味です.

debian/packagename というディレクトリ以下にコピーすることにより，builddeb の際に deb ファイルにコピーされ，後にインストールできるようになります.

DEBIAN ディレクトリというのは，debian/packagename/DEBIAN のことで，後に dpkg ファイルの control.tar.gz になる部分です.

名称	入力ファイル	効能
builddeb		.deb ファイルを作成
clean		不要なファイルを消す debian/rules clean 用
compress	package.compress スクリプト	ドキュメントの圧縮
desktop		.desktop ファイルの登録
fixperms		ファイル権限の修正
gconf		gconf schema の登録
gencontrol		dpkg-gencontrol のラッパ. control ファイルを DEBIAN ディレクトリにインストール
install	package.install	ファイルをインストールする
installcatalogs	package.sgmlcatalogs	SGML カタログを登録
installchangelogs		ChangeLog をインストールする
installcron	package.cron.monthly, package.cron.weekly, package.cron.daily package.cron.hourly package.cron.d	cron スクリプトをインストール
installdeb		DEBIAN ディレクトリにファイルをインストール
installdebconf	package.config package.templates	debconf 用のファイルをインストール

installdirs	package.dirs	ディレクトリを作成
installdocs	package.docs	ドキュメントをインストール
installemacsen	package.emacsen-install package.emacsen-remove package.emacsen-startup	emacs 用スクリプトをインストール
installexamples	package.examples	example ファイルをインストールする
installinfo	package.info	info をインストールする
installinit	package.init package.default	init スクリプトをインストールする．default ファイルをインストールする．
installlogcheck	debian/package.logcheck.cracking debian/package.logcheck.violations debian/package.logcheck.violations.ignore debian/package.logcheck.ignore.workstation debian/package.logcheck.ignore.server debian/package.logcheck.ignore.paranoid	logcheck 用のスクリプトを登録
installlogrotate	package.logrotate	logrotate 用のスクリプトを登録
installman	package.manpages	man をインストール
installmanpages		installman を使いましょう
installmenu	package.menu	メニューを追加
installmime	debian/package.mime debian/package.sharedmimeinfo	mime 情報を追加
installmodules	debian/package.modules debian/package.modprobe	modutil で module を追加
installpam	debian/package.pam	PAM 設定ファイルのインストール
installppp	debian/package.ppp.ip-up debian/package.ppp.ip-down	ppp 設定ファイルのインストール
installwm		Window manager を update-alternatives に登録
installxfonts		X フォントの登録
link	package.links	シンボリックリンクの作成
listpackages		処理するパッケージの一覧を出力するだけ
makeshlibs		shlibs ファイルを生成する
md5sums		DEBIAN/md5sums ファイルを生成する
movefiles		install を使ってください．

perl		perl スクリプトの依存関係を自動生成する
python		python モジュールの依存関係の自動生成と，プリコンパイルのための pre/postinst スクリプトの自動生成
scrollkeeper		OMF ファイルの登録
shlibdeps		共有ライブラリ依存関係情報の自動取得
strip		デバッグ情報の strip
suidregister		もう使わないでください
testdir		正しいディレクトリにいることを確認してくれる
testroot		root 権限で実行されていることを確認
testversion		debhelper のバージョンを確認する．今後は Build-Depends で指定してください.
undocumented		もう使わないでください
usrlocal		/usr/local 以下のディレクトリに関してポリシに基づいた作成/削除を実施.

3.1.1 dh-testroot

debian/rules スクリプトの中で，Policy で定義されている「実行には root 権限が必要」となっている部分があります．本当に root 権限で実行されているのか確認してあげる部分です.

`id -u` コマンドの出力が 0 であるか，とか `whoami` コマンドの出力が root であるか，とかで確認します. fakeroot で実行されることが多いため，fakeroot でだまし切れる内容にする必要があります.

今後 selinux などの広がりにより，root である，ということのチェックだけでは不十分になったばあいには debhelper 側で機能が拡張されることでしょう.

3.1.2 dh-testdir

現在のディレクトリが，debian パッケージのソースディレクトリであるか，ということを確認します．ここでは，カレントディレクトリからみて，./debian/rules というファイルが存在するか，などということで確認できます.

3.1.3 dh-clean

作業用に利用したファイルを消します．インストールイメージを作成するために一時的に作成する `debian/`**パッケージ名** ディレクトリの削除はここで実行します．

4　debhelper 論 その 3

Debhelper の処理の流れ.

4.1　debhelper の実行フロー

debhelper の一般的な実行フローにおいてどのファイルがどういう順番でコピーされていくのか，という部分について追跡してみます.

4.1.1　debian/rules clean

debhelper としては，Debian パッケージのビルドに利用した一時ファイルを削除します.
debian/以下のディレクトリの削除を実施します.

- *packagename* ディレクトリ
- *packagename**.debhelper
- files
- tmp

また，一般的な一時ファイルの削除も実施します.

- #*#
- *~
- DEADJOE
- *.orig
- *.rej
- *.bak
- *.SUMS
- TAGS
- core
- */.deps/*.P
- auotm4te.cache

4.1.2　debian/rules build

ここでは，debhelper はほとんどなにもしません．上流のソースコードに対して Make し，コンパイルする作業が主です.
カレントディレクトリが正しい場所か，ということを確認します.

4.1.3 debian/rules install

ここでは，debhelper を利用して，インストール先のディレクトリを作成し，そのディレクトリにソフトウェアをインストールします.

dh_installdirs で，必要なディレクトリを `debian/`*package* 以下に作成します.

`debian/`*package* ディレクトリににソフトウェアをインストールします．autoconf/automake を利用しているソフトウェアであれば，`make install DESTDIR=$(PWD)/debian/package/` のようなコマンドでインストールできます.

4.1.4 debian/rules binary

ここでは，インストールしたソフトウェアを Debian パッケージにするための最終的な微調整を実施します.

dh_testroot で root であることを確認します.

dh_installdocs でドキュメントファイルを `debian/`*package*`/usr/share/doc/XXX` にコピーします.

dh_installexamples でドキュメントファイルを `debian/`*package*`/usr/share/doc/examples/XXX` にコピーします.

dh_installman で `debian/`*package*`/usr/share/man/manX/XXX.X` にコピーします.

dh_link で必要なシンボリックリンクを作成します.

dh_strip で `debian/`*package*`/usr/bin/XXX` や `debian/`*package*`/usr/lib/XXX` に存在している実行ファイルのデバッグ情報を strip します.

dh_compress で `debian/`*package*`/usr/share/doc/XXX` や `debian/`*package*`/usr/share/man/manX/XXX.X` などにある，ポリシーで gzip 圧縮しておくべきとされているファイルを gzip 圧縮します.

dh_fixperms で `debian/`*package*`/usr/share/doc` 以下の実行権限をはずしたりして，アクセス権を修正します.

dh_installdeb で制御ファイルを DEBIAN ディレクトリーにコピーします．`debian/`*package*`.XXX` を `debian/`*package*`/DEBIAN/XXX` にコピーします．ここで対象となるのが，下記です.

- postinst
- preinst
- postrm
- prerm
- shlibs
- conffiles

`debian/`*package*`/etc/` 以下にファイルがある場合は，`debian/`*package*`/DEBIAN/conffiles` に追記されます[*5]．postinst/preinst/postrm/prerm に関しては，`#DEBHELPER#` と記述されている部分は `debian/`*package*`.XXX.debhelper` ファイルの中身で置換されます[*6].

dh_shlibdeps で，共有ライブラリの依存関係を解析します．`debian/`*package* 以下にある実行ファイルと共有ライブラリの一覧を dpkg-shlibdeps に渡します．`debian/`*package*`.substvars` に出力させます．内容とし

[*5] debconf V3 以上

[*6] Debian/Debhelper/Dh_Lib.pm

ては，依存するパッケージの一覧を shlibs:Depends 変数の定義として出力します．

dh_gencontrol は，dpkg-gencontrol コマンドを利用します．debian/changelog ファイルを解析し，バージョンを調べ，debian/control のパッケージ部分に対して，debian/*package*.substvars にある変数置換を実施し，debian/*package*/DEBIAN/control を生成します．また，Installed-Size 情報を du を実行して作成します．

```
--- /tmp/control       2005-04-08 07:49:13.236346992 +0900
+++ debian/whizzytex/DEBIAN/control     2005-04-08 06:17:20.000000000 +0900
@@ -1,6 +1,11 @@
 Package: whizzytex
+Version: 1.2.2-2
+Section: tex
+Priority: optional
 Architecture: all
 Depends: emacsen, tetex-bin (>= 2.0.2-17), advi | xdvi | gv
+Installed-Size: 584
+Maintainer: Junichi Uekawa <dancer@debian.org>
 Description: a WYSIWYG emacs environment for LaTeX
  WhizzyTeX is an emacs minor mode for incrementally
  (TeXing and) previewing a LaTeX file while editing at real-time.
```

dh_md5sums で debian/*package*/DEBIAN/md5sum に debian/*package* 以下にあるファイルの md5sum の結果を記録します．

dh_builddeb で dpkg-deb --build コマンドを呼び出し，パッケージをビルドします．debian/*package*/DEBIAN を制御情報として利用し，debian/*package*/以下をパッケージのデータとして利用し，../*package*_XXXX_XXX.deb などを作成します．

4.1.5 参考資料

debhelper で作成される debian/rules ファイルを参考のため載せておきます．

```
#!/usr/bin/make -f
# -*- makefile -*-
# Sample debian/rules that uses debhelper.
# This file was originally written by Joey Hess and Craig Small.
# As a special exception, when this file is copied by dh-make into a
# dh-make output file, you may use that output file without restriction.
# This special exception was added by Craig Small in version 0.37 of dh-make.

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1



CFLAGS = -Wall -g

ifneq (,$(findstring noopt,$(DEB_BUILD_OPTIONS)))
        CFLAGS += -O0
else
        CFLAGS += -O2
endif

configure: configure-stamp
configure-stamp:
        dh_testdir
        # Add here commands to configure the package.

        touch configure-stamp


build: build-stamp

build-stamp: configure-stamp
        dh_testdir

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

        touch build-stamp

clean:
        dh_testdir
        dh_testroot
        rm -f build-stamp configure-stamp

        # Add here commands to clean up after the build process.
        -$(MAKE) clean

        dh_clean

install: build
        dh_testdir
        dh_testroot
        dh_clean -k
        dh_installdirs

        # Add here commands to install the package into debian/test-source.
        $(MAKE) install DESTDIR=$(CURDIR)/debian/test-source


# Build architecture-independent files here.
binary-indep: build install
# We have nothing to do by default.

# Build architecture-dependent files here.
binary-arch: build install
        dh_testdir
        dh_testroot
        dh_installchangelogs
        dh_installdocs
        dh_installexamples
#        dh_install
#        dh_installmenu
#        dh_installdebconf
#        dh_installlogrotate
#        dh_installemacsen
#        dh_installpam
#        dh_installmime
#        dh_installinit
#        dh_installcron
#        dh_installinfo
        dh_installman
        dh_link
        dh_strip
        dh_compress
        dh_fixperms
#        dh_perl
#        dh_python
#        dh_makeshlibs
        dh_installdeb
        dh_shlibdeps
        dh_gencontrol
        dh_md5sums
        dh_builddeb

binary: binary-indep binary-arch
.PHONY: build clean binary-indep binary-arch binary install configure
```

5 DFSG 教

松山

5.1 DFSGって何だ?

Debian Free Software Guidline の略. Debian Social Contract の一部を成し, またその第一条「Debian は 100% フリーソフトウェアであり続けます」のよりどころとなっています. したがって,DFSG にそったライセンスのもとにないソフトウェアは,(正式の)Debian には含まれません (non-free に入る). また,DFSG は OSI の The Open Source Definition の元となっています.

5.2 DFSGってどんなんだ?

- 自由な再配布
- ソースコードの配布
- 派生ソフトウェアの作成と, オリジナルと同条件下での配布の許可
- 原作者によるソースコードの整合性維持
- すべての個人, 団体の平等
- 目標分野の平等
- ライセンスの配布
- ライセンスは Debian に限定されない
- ライセンスは他のソフトウエアを侵害しない

5.3 フリーなライセンスの例

DFSG-free とみなさえれているライセンスには以下などがあるようです.

- GPL
- BSD
- Artistic

ただ, これらは一般的な話であって, 厳密には個々のパッケージ毎に DFSG-free か否かが判断されるそうです.

5.4 FAQ

DFSG の FAQ からいくつかピックアップしてみました[*7].

[*7] http://people.debian.org/ bap/dfsg-faq.html

どうやって DFSG-free かどうかを判定するのか　人が判定し,debian-legal という ML で議論されるそうです. ただ,DFSG は文字どおりガイドラインであって法ではないので,「DFSG-free だから自由ソフトウェアだ」と断言はできません.

DFSG フリーだと法的リスクがないのか　否. 各自弁護士を雇ってください.

ドキュメントには?　ドキュメントも DFSG-free である必要があります. DFSG は Debian のあらゆる部分に適用されます. また, プログラムのドキュメントに関してはプログラムとの間の往来を考えると, プログラムと同じ方が好ましいが...?

5.5 おわりに

かつて RMS は自由なソフトウェアとは以下のような自由のあるソフトウェアだと説きました.

- 目的を問わず実行する自由
- ソースを調べ修正する自由
- 仲間のために配布する自由
- コミュニティ全体のために修正したものを配布する自由

今の Debian と FSF の関係がどうであれ, 基本的な「思い」というのは同じだと思います. DFSG はそういう「思い」のもとで, より具体的にどうしたらよいのかを示すガイドラインとして有意義だと思います.

6 dpkg-cross

岩松

6.1 はじめに

今回、dpkg-cross について調べてみました。

6.1.1 クロス開発ってなんですか

dpkg-cross についてお話する前にクロス開発というものを理解する必要があります。i386 などの高速な CPU を扱っていると、ソフトウェアをコンパイルするマシンと実行するマシンは一緒だったりしますが、非力な CPU（ ARM とか SupserH など）でコンパイルしようとすると非常に時間がかかります。また、プログラムを動かしたいが、対象のマシンには開発環境がないというときもあります。このような時にコンパイルは高速な CPU で行うと非常に速くコンパイルできたり、別のマシンでコンパイルしたソフトウェアを対象のマシンに転送し、動かすことができます。

このように、クロス開発とはプログラムを開発する環境と実行する環境が異なる開発方法をクロス開発と言います。

6.1.2 dpkg-cross で何をするの？

dpkg-cross はクロス環境用パッケージを生成するときに使用します。クロスで開発するときには対象アーキテクチャのバイナリパッケージが必要になります。

例えば、i386 上で PowerPC で動く libncurses を使ったアプリケーションをクロスコンパイルしたいときには libncurses5_*.*.*_powerpc.deb と libncurses5-dev_*.*.*_powerpc.deb が必要になります。これらのパッケージは PowerPC 用なので 通常 i386 上ではインストールできません。 PowerPC 用のパッケージを持ってたとしても、クロスコンパイル用にディレクトリを作成してそこにヘッダファイルをコピーして..... と面倒くさいです。そこで dpkg-cross の登場になります。

6.2 やってみる

6.2.1 インストール

apt-get install dpkg-cross

以上。

6.2.2 設定

使うアーキテクチャによって設定を行う必要があります。設定ファイルは /etc/dpkg-cross/cross-compile です。以下が内容です。

```
#
# /etc/dpkg-cross/cross-compile: configuration for dpkg-cross & Co.
#

# default architecture for dpkg-cross (to avoid always typing the -a option
# if you do cross installations only for one architecture)
#default_arch = m68k

#
# general section: paths of cross compiling environment
#
# you can set the following variables here:
#  crossprefix: prefix for cross compiling binaries; default: $(ARCH)-linux-
#  crossbase  : base prefix for the following; default: /usr
#  crossdir   : base directory for architecture; default:
#               $(CROSSBASE)/$(ARCH)-linux
#  crossbin   : dir for binaries; default: $(CROSSDIR)/bin
#  crosslib   : dir for libraries; default: $(CROSSDIR)/lib
#  crossinc   : dir for headers; default: $(CROSSDIR)/include
#  crossinfo  : dir dpkg-cross' package info files; default:
#               \$(CROSSLIB)/dpkg-cross-info
#  maintainer : maintainer name to pass to original dpkg-buildpackage
#               in -m option. If not set at all, don't pass a -m, thus
#               dpkg-buildpackage will use the name from the changelog
#               file. If set to the special string CURRENTUSER,
#               dpkg-buildpackage will use the name from the
#               changelog, too, but signing the .changes will be done
#               as the current user (default key).
#  removedeps : comma-separated list of package names that should be removed
#               from depends/conflicts/etc fields
#  keepdeps   : comma-separated list of package names thet should be kept
#               in depends/conflicts/etc fields as is, without adding
#               -arch-cross.
#
# Usually, you need only set crossbase, or maybe also crossdir
#
crossbase = /usr

# A crossroot definition is for the complete-Debian-system-mounted-somewhere
# approach, mainly used for Hurd.
#crossroot-hurd-i386 = /gnu

#
# This setting for maintainer is usually right:
#
maintainer = CURRENTUSER
#
# This list is far from being complete ...
# Please send additions to Nikita Youshchenko <youshcs.msu.su>
```

```
#
removedeps = gcc, binutils, gpm, cpp, debianutils, xfree86-common, libpam-runtime,
xlibs-data, debconf

#:w

# per-package sections: additional environment variables to set
#
# Please send additions to Nikita Youshchenko <youshc@cs.msu.su>

package e2fsprogs:
    unset LD

# package gs-aladdin:
# # must be a native gcc
#   CCAUX = gcc
#
# -------------------------------
# This should fit EmDebian needs:
#
# mode emdebian:
# package all:
#   scope environment:
#       emdebian = true
#   scope makeflags:
#       CROSSPREFIX = $(crossprefix)
#       EXTRA_CFLAGS = ...
#       LIBC = ...
#       CONFIG = ...
```

- default_arch
  対象のアーキテクチャを指定します。PowerPC なら **default_arch = powerpc** と指定します。ひとつのアーキテクチャだけクロスコンパイルを行うときはここに書いておくと、アーキテクチャを指定せずに済みます。
- crossprefix クロスコンパイル用のツールを呼び出すために使用します。クロスコンパイル用の gcc などは **powerpc-linux-gcc** などになっているのでこの形式を指定します。デフォルトでは $(ARCH)-linux- が指定されています。
- crossbase
  クロス開発環境をインストールするベースとなるディレクトリを指定します。**crossbase = /home** と指定すると CROSSBASE に /home が指定されます。デフォルトでは /usr 設定されています。
- crossdir
  クロス開発環境インストール先を指定します。**crossdir = /usr/cross** と指定しますと /usr/cross 以下にインストールされます。デフォルトでは $(CROSSBASE)/$(ARCH)-linux 設定されています。
- crossbin

実行バイナリをインストールするディレクトリを指定します。デフォルトでは $(CROSSDIR)/bin が設定されています。

- crosslib
  ライブラリをインストールするディレクトリを指定します。デフォルトでは $(CROSSDIR)/lib が設定されています。
- crossinc
  ヘッダファイルをインストールするディレクトリを指定します。デフォルトでは $(CROSSDIR)/inc が設定されています。
- crossinfo
  info ファイルをインストールするディレクトリを指定します。デフォルトでは $(CROSSDIR)/info が設定されています。
- removedeps 設定されているパッケージを depends/conflicts/etc から削除します。クロスコンパイル用のパッケージをインストールしやすくするためのものだと思います。
- keepdeps 設定されているパッケージを depends/conflicts/etc から削除しないようにします。

これらの中で重要なのは

- defautl_arch
- crossbase

です。あとは特に気にしなくてよいと思います。

6.2.3 パッケージを作ってみる

クロス開発用のパッケージを作成するには dpkg-cross コマンドを使います。

```
$ ls
libncurses5_5.4-4_powerpc.deb
libncurses5-dev_5.4-4_powerpc.deb
$ dpkg-cross -b -apowerpc libncurses5_5.4-4_powerpc.deb
$ dpkg-cross -b -apowerpc libncurses5-dev_5.4-4_powerpc.deb
```

-a でアーキテクチャを指定します。-b は build を行うという意味です。行うと以下のような名前のパッケージが作成されます。

```
$ ls
libncurses5_5.4-4_powerpc.deb
libncurses5-dev_5.4-4_powerpc.deb
libncurses5-powerpc-cross_5.4-4_all.deb
libncurses5-dev-powerpc-cross_5.4-4_all.deb
```

6.2.4 インストールしてソフトウェアを作ってみる

最初はクロスコンパイル用のパッケージを入れないままコンパイルしてみます。

```
iwamatsu@soki:~/dev/study/src$ ls
sample.c
iwamatsu@soki:~/dev/study/src$ powerpc-linux-gcc -o sample sample.c -lncurses
sample.c:10:20: curses.h: そのようなファイルやディレクトリはありません
sample.c: In function ‘main’:
sample.c:24: error: ‘WINDOW’ undeclared (first use in this function)
sample.c:24: error: (Each undeclared identifier is reported only once
sample.c:24: error: for each function it appears in.)
sample.c:24: error: ‘w_pRootwin’ undeclared (first use in this function)
sample.c:24: error: ‘w_pWin’ undeclared (first use in this function)
```

/usr/power-pc/include に curses.h がないのでエラーになっています。クロスコンパイル用のパッケージをインストールして再度コンパイルしてみます。

```
iwamatsu@soki:~/dev/study/src$ sudo dpkg -i libncurses5-powerpc-cross_5.4-4_all.deb
iwamatsu@soki:~/dev/study/src$ sudo dpkg -i libncurses5-dev-powerpc-cross_5.4-4_all.deb
iwamatsu@soki:~/dev/study/src$ powerpc-linux-gcc -o sample sample.c -lncurses
iwamatsu@soki:~/dev/study/src$ ls
sample.c
sample
iwamatsu@soki:~/dev/study/src$ file sample
sample: ELF 32-bit MSB executable, PowerPC or cisco 4500, version 1 (SYSV),
for GNU/Linux 2.2.0, dynamically linked (uses shared libs), not stripped
```

PowerPC 用のプログラムとしてコンパイルができました。PowerPC なマシンに持っていって動作確認してみて、正常に動作すれば OK です。

6.3 おわりに

今回 dpkg-cross について説明してみました。debian は Redhat 系とくらべてクロス開発環境の構築方法が容易だなと感じました。Redhat だとクロス開発環境用に再コンパイルしなおすという方法しかないようです(私が調べた限りでは)。やっぱ Debian はすばらしいなぁと思いました。

また、今回の事前課題が「さわってみた/さわってみたい Debian のこんなアーキテクチャ移植版」でもありますし、x86 以外のアーキテクチャ(最近なら PowerPC?) を触ってみていろいろコンパイルしてみるのもいいかもしれません。

7 lintian と linda を使っての Debian パッケージの作成精度向上

上川

7.1 Debian パッケージに関する問題のパターンマッチング

Debian パッケージを作成する際に，このファイルにこういう内容を書いているのであれば一般的に問題だろう，とか，このファイルがこの内容になっているのであればテンプレートファイルのままになっているのではないか，ということを指摘するのは実はそんなに難しいことではありません．Debian パッケージのファイルの中を正規表現で検索してしまえば見付かります．そういう一般的な問題の解決方法として，lintian や linda は利用されています．

例えば，間違ったタイプのファイルが間違ったディレクトリに配置されている，というような問題を検出するのに優れています．

7.2 実行してみる

では，実行してみましょう．

lintian を実行してみます．deb パッケージを指定すると，バリナリパッケージを確認します．dsc ファイルを指定すると，ソースパッケージを確認します．changes ファイルを指定すると，そのアップロードに含まれる予定のファイルを全て確認してくれます．何かメッセージが出ると，-i オプションをつけると，詳細なメッセージを表示してくれます．

```
# lintian wysihtml-el_0.11.cvs.1-1_powerpc.deb
# lintian wysihtml*dsc
# lintian wysihtml*changes
E: wysihtml_0.11.cvs.1-1_powerpc.changes: bad-distribution-in-changes-file UNRELEASED
# lintian -i wysihtml*changes
E: wysihtml_0.11.cvs.1-1_powerpc.changes: bad-distribution-in-changes-file UNRELEASED
N:
N:   You've specified an unknown 'target distribution' for your upload in
N:   the debian/changelog file.
N:
N:   Note, that the distributions non-free and contrib are no longer valid.
N:   You'll have to use distribution 'unstable' and 'Section: non-free/xxx'
N:   or 'Section: contrib/xxx' instead.
N:
```

出力が E:ではじまる場合は，ポリシーに反するエラーです．W:は特にポリシーに反するというわけではないですが，直したほうが良いのではないか，と推測される「よくある問題」に対しての警告です．

同様に linda を実行してみます．

```
# linda wysihtml-el_0.11.cvs.1-1_powerpc.deb
# linda wysihtml*dsc
# linda wysihtml*changes
```

この二つのツールの出力は全く同じではないので，とりあえず両方実行しておきましょう．

7.2.1 警告メッセージの対応方法

Debian Policy に基づいてどの部分が問題なのか，を指摘してくれます．修正の方法も書いてある場合もあるので，適切に対応しましょう．

/usr/share/lintian/overrides/, /usr/share/linda/overrides/ にファイルを追加すると，メッセージを無視するように設定することができます．これらのファイルのことを lintian overrides, linda overrides とよびます．

適切でない警告[*8]を出す場合もあるので，そういう場合にはここにファイルを追加するようにパッケージを作成します．個人的には，ユーザに必要でないファイルをシステムに追加することになるため，好きではありません．できることであれば，lintian や linda に適切な修正パッチを投げましょう．

7.3 どうなっているのか

lintian と linda が内部構成としてどうなっているのか解説します．

7.3.1 lintian の構成

lintian は perl script です．Debian Package を一時ディレクトリに展開して，それに対して grep などを活用し，構成を分析します．正規表現でそれっぽい間違いを発見したら，報告します．

/usr/share/lintian/checks/以下に説明文と perl のスクリプトが大量に入っています．これらが順番に実行されます．

追加するためには，そのディレクトリにある.desc ファイルを編集して，perl のスクリプトを作成すればよいようです．

lintian は実行時にまずパッケージを lintian lab と呼ぶ場所に展開してくれるので，その展開されたファイルを解析すれば良いです．

7.3.2 linda の構成

linda は lintian を python をつかって再実装したものです．

/usr/share/linda/checks/ 以下に python のスクリプトが大量に入っています．/usr/share/linda/data/以下にチェックのメッセージの対応表があります．各メッセージは短いのですが，それぞれの文章は gettext 経由で取得するように実装されているようです．

```
msgid "versioned-provides_1"
msgstr ""
"The package shown above is trying to do versioned provides, but they aren't "
"supported by dpkg, and therefore, should not be used."

msgid "versioned-provides_s"
msgstr "Package contains a versioned Provides with package %s."
```

参考として，lintian, linda 各パッケージのアップロードの履歴の状況を調べてみました．lintian が活発でなかった 2002,2003 年の時期に linda が活発に開発されていたような雰囲気がうかがえなくもないです．

```
 zgrep '^ -- ' /usr/share/doc/lintian/changelog.Debian.gz | cut -d, -f2 | awk '{print $3}' | uniq -c
      2 2005
     10 2004
      6 2003
     14 2002
     18 2001
     13 2000
     12 1999
     44 1998


 zgrep '^ -- ' /usr/share/doc/linda/changelog.gz | cut -d, -f2 | awk '{print $3}' | uniq -c
      2 2005
      9 2004
     20 2003
     36 2002
```

[*8] false positive という

7.4 どう応用するか

devscripts に入っている debuild コマンドでパッケージをビルドすると，lintian を自動的に実行してくれます[*9]．debuild コマンドを活用しましょう．

lintian linda で確認できる問題もありますが，他にも問題は発生します．パッケージの問題は他に何があるか，確認してみます．段階がいくつかありますが，簡単に考えてみても，それぞれ別の段階で下記のような不具合が発生するでしょう．

- ソースコードがコンパイルできない．
- deb パッケージがインストールできない．
- アプリケーションが実行できない．
- 操作に対する動作がおかしい．

この問題については lintian と linda の確認はほぼ無力です．

実際は pbuilder などの他のツールを併用します．debdiff で前のバージョンとの差を確認したり，debit コマンド[*10]を利用して，自動テストツールで動作を確認したりします．

あとは，Debian のユーザがパッケージをインストールして，利用してくれて，問題を発見して BTS で報告してくれるのを待つばかりです．

[*9] /etc/devscripts.conf 参照
[*10] 現在は利用できません

8 alternatives -選択せよ-

えとー

8.1 alternatives とは？

パッケージ管理というと依存関係の管理についてのみがクローズアップされるが、パッケージ管理としては、それだけでは不足する部分がある、それを補うものの一つが alternatives です。

日本語訳すると「選択肢」、私の定義ですが、「複数のパッケージを特定機能ごとに一つにまとめて扱い機能ベースのパッケージの管理を提供するもの。」「パッケージを機能の視点で管理する。」「プログラムに OS の統一的な API を提供する。」の 3 点です。

機能としては、「類似の機能を持つプログラムの別名を提供する。」で、symlink を使って実現しています。alternatives とは、その symlink を管理するための機構と言えるでしょう。

なぜわざわざ別名が必要？

1. プログラムからの使い勝手の向上万単位のパッケージの存在する Debian では同じような機能を持つ別々のパッケージが複数存在することは珍しくありません。伝統的に Unix 系 OS ではパイプやリダイレクトなどやスクリプト、プログラムなどから他のプログラムの機能を利用することによりコード量の削減と質の向上を図っています。しかし、プログラムから呼び出す際に同じような機能をもつもの全てを把握し条件分岐などで呼んで行くのはコストから言って現実的ではありません。その解決策としてはプログラムへの統一的な API を提供することができるようすることです。awk スクリプトやコンパイラなどを思い浮べるといいと思います。
2. ユーザへの使い勝手の向上ユーザからしても同じ機能なのに別の別の名前であると不便な場合もあります。MUA や IRC クライアント からの Web Browser を呼び出す際を想像してもらえば想像し易いと思います。
3. menu との連携 Debian 独自なものとして menu がありますが、これと alternatives を連携させることによりメニューをいじらずに、違うパッケージを提供することができます。

8.2 update-alternatives の使い方

see man(重要)

alternatives の制御には /usr/sbin/update-alternatives というコマンドを使います。

```
書式一覧
 update-alternatives --install リンク 一般名 パス 優先度  [--slave スレーブリンク スレーブ一般名 スレーブパス]
 update-alternatives --remove 一般名 パス
 update-alternatives --remove-all ディレクトリ
 update-alternatives --all
 update-alternatives --auto 一般名
 update-alternatives --display 一般名
 update-alternatives --list 一般名
 update-alternatives --config 一般名
 update-alternatives --set 一般名 パス
```

8.3 用語の説明

8.3.1 マスター

- マスター alternatives の対象とするもの
- リンク 一般的なリンクです
  例 /usr/bin/awk, /usr/bin/editor, /usr/bin/pager, /usr/bin/x-www-browser
- 一般名 一般的な名前です、
  例 awk, editor, pager, x-www-browser
- パス それぞれの alternatives のリンク先になります
  例 /usr/bin/awk: /usr/bin/nawk, /usr/bin/mawk, /usr/bin/gawk /usr/bin/editor : /bin/ed, /bin/nano, /usr/bin/vim /usr/bin/pager : /bin/more, /usr/bin/less, /usr/bin/w3m, /usr/bin/lv /usr/bin/x-www-browser : /usr/bin/mozilla, /usr/bin/kazehakase, /usr/bin/mozilla-firefox
- 優先度 auto モードで設定される際の選択基準になる整数値です

8.3.2 スレーブ

- スレーブ alternatives のマスターに付随するファイル、man など 複数指定可能

- スレーブリンク 一般的なリンクに付随するリンク
  例 /usr/share/man/man1/awk.1.gz, /usr/share/man/man1/nawk.1.gz, /usr/share/man/man1/editor.1.gz, /usr/share/man/man1/pager.1.gz, /usr/share/man/man1/x-www-browser.1.gz
- スレーブ一般名 一般的な名前に付属する一般名
  例 awk.1.gz, editor.1.gz, pager.1.gz, x-www-browser.1.gz
- スレーブパス それぞれの alternatives に付随するリンク先
  例
  /usr/share/man/man1/awk.1.gz, /usr/share/man/man1/nawk.1.gz : /usr/share/man/man1/mawk.1.gz, /usr/share/man/man1/gawk.1.gz
  /usr/share/man/man1/editor.1.gz : /usr/share/man/man1/ed.1.gz, /usr/share/man/man1/nano.1.gz, /usr/share/man/man1/vim.1.gz
  /usr/share/man/man1/pager.1.gz : /usr/share/man/man1/more.1.gz, /usr/share/man/man1/less.1.gz, /usr/share/man/man1/w3m.1.gz, /usr/share/man/man1/lv.1.gz
  /usr/share/man/man1/x-www-browser.1.gz : /usr/share/man/man1/mozilla.1.gz, /usr/share/man/man1/kazehak
  /usr/share/man/man1/mozilla-firefox.1.gz

8.3.3 automatic と manual

alternatives は優先度によって自動的に選択する automatic モードと優先度を無視しユーザが選択する manual モードがあります。デフォルトは automatic モードで –config や –set 、 –all を使い変更した場合に manual モードに以降します。automatic モードに戻したければ –auto を使います。

8.4 ユーザとして使う場合

ユーザとして主に使用するコマンド

8.4.1 状態表示 (一般ユーザで可)

```
update-alternatives --display 一般名
update-alternatives --list 一般名
```

8.4.2 設定変更 (root のみ)

```
update-alternatives --auto 一般名
update-alternatives --config 一般名
update-alternatives --set 一般名 パス
```

の 4 つです。

8.4.3 使用例と説明

```
# update-alternatives --display editor             <-- editor という一般名の alternatives のステータスを表示
editor - status is auto.                           <-- alternatives の モード
 link currently points to /usr/bin/vim             <-- 現在のリンク先
/bin/ed - priority -100                            <-- マスターリンク - 優先度 (/bin/ed)
 slave editor.1.gz: /usr/share/man/man1/ed.1.gz    <-- スレーブ一般名 : スレーブリンク先 (/bin/ed)
/bin/nano - priority 40                            <-- マスターリンク - 優先度 (/bin/nano)
 slave editor.1.gz: /usr/share/man/man1/nano.1.gz  <-- スレーブ一般名 : スレーブリンク先 (/bin/nano)
/usr/bin/vim - priority 120                        <-- マスターリンク - 優先度 (/usr/bin/vim)
 slave editor.1.gz: /usr/share/man/man1/vim.1.gz   <-- スレーブ一般名 : スレーブリンク先 (/usr/bin/vim)
Current 'best' version is /usr/bin/vim.            <-- 優先度の一番高いもの

# update-alternatives --list editor                <-- editor という一般名の alternatives のマスターリンク先を表示
/bin/ed                                            <-- マスターリンク先
/bin/nano                                          <-- マスターリンク先
/usr/bin/vim                                       <-- マスターリンク先

# update-alternatives --auto editor                <-- 一般名 editor の alternatives を優先度によって変更
# update-alternatives --config editor              <-- 一般名 editor の alternatives を選択肢から手動で変更

There are 3 alternatives which provide 'editor'.   <-- editor という一般名の選択肢が 3 つある

   Selection    Alternative
-----------------------------------------------
     1        /bin/ed                              <-- alternatives
     2        /bin/nano                            <-- alternatives
*+   3        /usr/bin/vim                         <-- alternatives 現在選択されている

Press enter to keep the default[*], or type selection number:  <-- ここで番号を選択する
Using '/usr/bin/vim' to provide 'editor'.          <-- editor を /usr/bin/vim で提供する

# update-alternatives --set editor /usr/bin/vim    <-- 一般名 editor の alternatives を指定したリンクに変更する
Using '/usr/bin/vim' to provide 'editor'.          <-- editor を /usr/bin/vim で提供する
```

8.5 パッケージメンテナとして使う場合

パッケージメンテナな人が主に使うコマンドパッケージインストール時に使用 (主に postinst)

```
update-alternatives --install リンク 一般名 パス 優先度  [--slave スレーブリンク スレーブ一般名 スレーブパス]
```

パッケージ削除時に使用 (主に prerm)

```
update-alternatives --remove 一般名 パス
```

8.5.1 使用例

```
update-alternatives --install コマンド

update-alternatives --install リンク 一般名 パス 優先度 [--slave スレーブリンク スレーブ一般名 スレーブパス]
```

この書式ですが、slave は省略可能で、使用する場合には複数のスレーブを指定することもできます。

vim の postinst

リンク元のファイルがインストールされてからリンクを貼るので postinst に書く

```
case "$1" in
  abort-upgrade)
    for i in vi view ex editor ; do
      update-alternatives \
        --install /usr/bin/$i $i /usr/bin/vim 120 \
        --slave /usr/share/man/man1/$i.1.gz $i.1.gz /usr/share/man/man1/vim.1.gz

    done
    ;;
  configure)
    for i in vi view ex editor ; do
      update-alternatives \
        --install /usr/bin/$i $i /usr/bin/vim 120 \
        --slave /usr/share/man/man1/$i.1.gz $i.1.gz /usr/share/man/man1/vim.1.gz

    done
    if [ -L /usr/doc/vim ] ; then
      rm /usr/doc/vim
    fi
    ;;
esac
```

vim の prerm

リンク元のファイルがなくなる前に削除するので prerm に書く

```
case "$1" in
  remove)
    for i in vi view ex editor ; do
      update-alternatives --remove $i /usr/bin/vim
    done
    ;;
esac
```

8.6 マニアックなコマンド

update-alternatives –all

全ての選択肢に –config を使い設定を行なう

update-alternatives –remove-all

ディレクトリにある全ての選択肢を削除する（危険です)

8.7 一般的なオプション

```
--verbose 冗長メッセージ
--quiet メッセージを抑制
--test テスト（未実装）
--help ヘルプ
```

8.8 マニアックなオプション

–altdir

リンクを置くディレクトリ

デフォルトは /etc/alternatives/

–admindir

設定ファイルを置くディレクトリ

デフォルトは /var/lib/dpkg/alternatives/

8.8.1 マニアックなオプションの使用例

/usr/local/以下にソースからインストールしたものや alternatives を提供していないパッケージについて alternatives で管理したい場合などに使う 特権ユーザでなくても使えるのも特徴

使用例: /home/foo/eclipse 以下にある /home/foo/eclipse/eclipse を alternatives で管理したい。

```
$ mkdir /home/foo/altdir/
$ mkdir /home/foo/admindir/
$ mkdir /home/foo/bin/
$  /usr/sbin/update-alternatives --altdir /home/foo/altdir/ --admindir /home/foo/admindir/ \
      --install /home/foo/bin/eclipse eclipse /home/foo/eclipse/eclipse 100
```

で、.bashrc などに /home/foo/bin/ を追加 IM とか、MUA の管理にも向いているのではないだろうか。

8.9 update-alternatives ってどうなってるの?

8.9.1 構造

/usr/sbin/update-alternatives

alternatives の制御コマンド

perl で実装されており dpkg パッケージに含まれています。

/var/lib/dpkg/alternatives

alternatives 設定ファイルディレクトリ

設定ファイル例

```
$ cat /var/lib/dpkg/alternatives/editor
alternatives 設定ファイルディレクトリ
/usr/bin/editor                         <-- マスターリンク先
editor.1.gz                             <-- スレーブ一般名
/usr/share/man/man1/editor.1.gz         <-- スレーブリンク先

/bin/ed                                 <-- マスターリンク元 (/bin/ed)
-100                                    <-- 優先度 (/bin/ed)
/usr/share/man/man1/ed.1.gz             <-- スレーブリンク元 (/bin/ed)
/bin/nano                               <-- マスターリンク元 (/bin/nano)
40                                      <-- 優先度 (/bin/nano)
/usr/share/man/man1/nano.1.gz           <-- スレーブリンク元 (/bin/nano)
/usr/bin/vim                            <-- マスターリンク元 (/usr/bin/vim)
120                                     <-- 優先度 (/usr/bin/vim)
/usr/share/man/man1/vim.1.gz            <-- スレーブリンク元 (/usr/bin/vim)
```

/etc/alternatives/

alternatives のリンクのあるディレクトリです。

リンク例

```
$ ls -l /etc/alternatives/editor
lrwxrwxrwx  1 root root 12 2005-06-01 02:43 /etc/alternatives/editor -> /usr/bin/vi
```

```
/usr/share/man/man8/update-alternatives.8.gz
/usr/share/man/de/man8/update-alternatives.8.gz
/usr/share/man/es/man8/update-alternatives.8.gz
/usr/share/man/fr/man8/update-alternatives.8.gz
/usr/share/man/ja/man8/update-alternatives.8.gz
/usr/share/man/pt_BR/man8/update-alternatives.8.gz
```

alternatives マニュアルです。

8.9.2 動作

update-alternatives –auto

1. 一般名を取得
2. /var/lib/dpkg/alternatives/一般名 のステータス情報が manual の場合は auto に変更

3. /etc/alternatives/一般名 のリンク先を /var/lib/dpkg/alternatives/一般名 の優先度の情報を比較し最大のものへのリンクへと変更

update-alternatives –config 及び update-alternatives –set コマンド

1. 一般名とリンク先のパスを取得
2. /var/lib/dpkg/alternatives/一般名 のステータス情報が auto の場合は manual に変更
3. /etc/alternatives/一般名 のリンク先を指定されたリンク先のパスへと変更

update-alternatives –install コマンド

1. 存在しない場合は指定されたパスのリンクを /etc/alternatives/一般名 へと貼る
2. 存在しない場合は /var/lib/dpkg/alternatives/一般名 ファイルを指定された情報に基ずいて生成
3. /etc/alternatives/一般名 のリンク先を /var/lib/dpkg/alternatives/一般名 の優先度の情報を比較し最大のものへのリンクへと変更

update-alternatives –remove コマンド

1. /var/lib/dpkg/alternatives/一般名 から指定された パスの情報を削除
2. /var/lib/dpkg/alternatives/一般名 で alternatives が提供されない場合は/var/lib/dpkg/alternatives/一般名 及び /etc/alternatives/一般名リンク先を削除
3. /var/lib/dpkg/alternatives/一般名 で まだ alternatives が提供されている場合は/etc/alternatives/一般名 のリンク先を /var/lib/dpkg/alternatives/一般名 の優先度の情報を比較し最大のものへのリンクへと変更

8.10 update-alternatives の改善案

update-alternatives 関連の改善案を自分なりに考えてみた。

1. パッケージのファイル情報への反映
   現在の alternatives はパッケージをインストールされるまではそのパッケージがどんな alternatives を提案するのか という情報が解らない。これだと、インストールしてから、普段使用していた alternatives を使おうとしたら期待してたのと別のものが起動してしまい戸惑うことになるし、無駄なハマりの原因となるし、もし、そのような情報を集積できれば、機能からパッケージをより簡単に検索することができるようになる。
2. より柔軟に
   現在の alternatives では、man を slave にした場合などに英語のみしか表示できなくなってしまう、多言語化が進みつつある昨今これだけでは不足ということになってしまっているので、より柔軟な設定を行なえるのが望ましい。ファイルだけではなくコマンドを登録できるようにするなど。
3. menu 編集インターフェースとの連携
   独自に作成した alternatives などを menu に簡単に登録できるようにするなどの、応用的な使い方を簡便にするインターフェースの拡充

8.11 dsys について

8.11.1 目的

dpkg に含まれる パッケージ依存関係管理を行なう /usr/bin/dpkg 以外の/usr/sbin/update-alternatives, /usr/sbin/dpkg-divert, /usr/sbin/dpkg-statoverride の 3 つのコマンドに対応した GUI フロントエンドを提供することを目指している。

8.11.2 歴史

一番初期は update-alternatives のみに対応したもので、ruby 1.6 と ruby-gtk で書いていたが、やっぱ gtk2 だろう！ ということで、ruby-gtk2 に手を出してのんびりしてるうちに galternatives など競合で出てきてしまった。思い出したら実装したりしているがコーディング能力の低さのために遅々として改善されていない。

8.11.3 機能

/usr/sbin/update-alternatives
一覧の表示、個々のステータスの表示、変更、追加、削除
/usr/sbin/dpkg-divert
一覧の表示
/usr/sbin/dpkg-statoverride
一覧の表示、変更、追加、削除

8.11.4 TODO

まずはツールチップを適切に実装すること綺麗なコードの書き方や、デザインがわからないのでどうにかするせめて man くらい、、alternatives と statoverride のインターフェースを意識的に替えていたがそろそろ統一 treeview で右クリックできるようするショートカットの実装

8.11.5 将来

設定ファイルを作るかもアイコン欲しいよね
協力者の方募集中、、

9 Inside Debian-Installer

武藤 健志

9.1 Debian-Installer とは

Debian-Installer（以下 **d-i**）は、Debian GNU/Linux リリース 3.1、コードネーム Sarge から採用された新しいインストーラシステムです。

これまでのインストーラシステム（Woody 以前）として採用されていた Boot-Floppies には、次のような問題がありました。

- サイズの制限。その名のとおりフロッピーディスクを基盤にしたものなので、システム全体をフロッピー容量の 1.2MB〜1.44MB に圧縮して収まるように悪戦苦闘しなければならなかった。
- 動的ロードの不備。動的にインストーラを拡張する方法がなく、機能向上を行うすべがなかった。
- ハードウェア認識の不備。ハードウェアを自動認識する機構が入っていなかった。ユーザーはすべて手動で設定しなければならなかった。
- インストーラのリリースとテスト。インストーラ一式が一体のため、リリースするためには全体をビルドしなければならず、頻繁にリリースすることができなかった。バグへの対処が遅れがちになった。

2000 夏ごろ	Joey Hess がデザインを始める
2000 秋ごろ	サンプル実装が公開される
2002	Woody が Boot-Floppies に基づいてリリース。Sarge で d-i になるよう本格的に目標が立てられる
2003	開発がかなり進展。開発者も増える
2004	d-i rc2 リリース
2005 春	d-i rc3 リリース
2005 夏	Sarge リリース

表 2　d-i の簡単な歴史

9.2 開発体制

d-i の開発は、ほかのどの Debian のサブプロジェクトよりも大規模です。

- リポジトリのコミッタ権限がある人：153 人
- ローカライズコーディネータのコミッタに代行してもらっている人もいるので、実際はもっと多い？
- オフィシャルな Debian Developer は 60 人

- アカウント管理:Alioth (`http://alioth.debian.org/projects/d-i/`)、メーリングリスト:debian-boot@lists.debian.org、バグ追跡:installation-reports 仮想パッケージ名、IRC:#debian-boot

プロジェクトリーダ	Joey Hess
国際化	Christian Perrier、Dennis Stampfer
フロントエンド	Collin Watson
パーティションマネージャ	Anton Zinobiev
ネットワーク設定	Joshua Kwan
ハードウェア認識	Petter Reinholdtsen
移植	Martin Michlmayr、Steve Langaek、…
ドキュメント	Frans Pop

表 3　主な開発者

当初は CVS、現在は Subversion にて協業を行っています。

- `http://svn.d-i.alioth.debian.org/svn/d-i/trunk`（トランク、unstable 向け）
- `http://svn.d-i.alioth.debian.org/svn/d-i/branches/d-i/sarge`（フリーズされた Sarge のブランチ）

```
d-i/
    installer/build/ ——d-i のビルドディレクトリ（make ... を実行すると d-i イメージができる）
    installer/doc/   ——ドキュメント。manual/に XML 形式のインストールマニュアルが収録されている
    packages/        ——各 udeb のソースコード
    scripts/         ——自動化などに使うスクリプト集
```

図 1　d-i リポジトリの構造

9.3　d-i の構造

d-i は、これまでに蓄積されてきたさまざまな Linux/Debian のフレームワークを応用しています。

udeb　通常の deb から実行に不必要なものを削除してスリムにしたもの
cdebconf　C で記述された debconf フレームワーク (いくらか拡張)
devfs　ハードディスクなどのデバイスファイルの取り扱いを容易にする
フレームバッファと国際化端末　国際化端末を実行する (bogl-term および jfbterm)
discover と hotplug　ハードウェアの自動認識

9.3.1 udeb

udeb は、通常の deb とほぼ同じですが、ドキュメントなどの実行に直接関係ないものを削ぎ落としたバイナリパッケージです。

- インストーラだけで利用する特別なメタ情報として、XB-Installer-Menu-Item が含まれているものがある。これは、インストーラのメニューに表示する際の順番を示す
- Depends や Provides で依存関係を処理したり、debconf 用の国際化済み templates によってローカライズされたメニューを表示できる (メニューに表示されるのはパッケージの short description だが、特別に debconf templates の中でこのローカライズも設定している)
- インストーラのメニューを選択することは、postinst を実行するということになる
- 環境によってロードする順序を変えることができる 。たとえば、ネットワークインストールなら最初にネットワークの設定を済ませてから IDE コントローラを検出、CD インストールなら最初に IDE コントローラの設定を済ませてからネットワークを検出

anna	擬似 APT
udpkg	マイクロ dpkg
languagechooser、countrychooser	言語・国情報の設定
hw-detect、ethdetect、hw-detect-full	ハードウェアの検出
preseed	事前構成のロード
netcfg	ネットワーク設定
network-console	SSH サーバーを起動してリモートインストールを実行する
cdrom-retriever、floppy-retriever、net-retriever	udeb パッケージをダウンロードする
partman	パーティションマネージャ
base-installer	ベースシステムを debootsrap で展開し、CPU に合ったカーネルをインストールする
os-prober	インストールされているほかの OS の検出
prebaseconfig	CD の取り出し、現在の状態の保存など

表 4　主な udeb

リポジトリの packages/だけでなく、libc や discover、console-tools のように、外部の deb パッケージが udeb を提供することもあります。

d-i は udeb の集合体で、最初にロードされるインストーラも udeb を基に構成されています。d-i の make 時にどの udeb から構成しておくかによって、CD 向け、ネットワーク向け、USB メモリ向けといったイメージを作成できます。

9.3.2 cdebconf

cdebconf は、Perl で記述されていた debconf を、C で実装し直したものです。

- Perl の膨大なアーカイブの必要なく動作
- 進捗バーなどの機能拡張

今後は普通の環境も、debconf は捨てて (orphan 済み)、cdebconf に移行しようという計画になっています。d-i では cdebconf が大きな役割を果たしています。

- debconf インターフェイスにのっとったフロントエンドを切り替えることができる (デフォルトは dialog)
- templates からインストールの質問を表示し、答えをデータベースに格納する
- 質問に優先度 (critical、high、medium、low) を設定して、質問の詳細と数を変えられる (デフォルトは high)
- データベースに最初から値を入れるようにして、全自動インストールができる (preseed)

9.3.3 devfs

Linux カーネル 2.4 で導入された devfs では、「動的に」「利用可能な」デバイスファイルが作成されるので、ハードディスクのパーティションやフレームバッファの存在を検出するのに便利です。

ただ、devfs はもう obsolete なので、今後は udev に書き換えていくことになるでしょう。

9.3.4 フレームバッファと国際化端末

Linux のデフォルトの端末画面はラテン文字しか表示できないので、国際化されたインストーラを実行するためにはフレームバッファドライバをロードしたあと、その上で国際化端末を実行する、という手順が必要になります。x86 の場合フレームバッファドライバは、vesafb → vga16fb と試行しています。d-i の国際化端末としては、次の 2 つを採用しています。

- bogl-bterm：UTF-8 対応の端末。
- jfbterm：再起動後、CJK 圏でのみ使われる端末 (表示は EUC-JP)。なぜ bogl-bterm で続けなかったかというと…

9.3.5 discover と hotplug

最初のハードウェア認識には discover が使われます。再起動後、標準で使われるのは hotplug です。

- discover：能動的にハードウェアを認識する。データベースには PCI ID 値などを列挙した discover1-data パッケージの情報を使う
- hotplug：ほぼ受動的にハードウェアを認識する。カーネルから送られた情報を使う

9.4 今後の d-i

Etch に向けての TODO はいろいろあります。

- より良いハードウェア認識：discover2 への移行。既存の discover のデータベースの更新 (volatile 経由？)。

- Woody のときに使えたような「レスキューモード」: rescue.udeb が用意されました
- Gtk+ による GUI フロントエンド：実装が進んでいます。ただ、udeb の持っている情報をもっと増やすか、別の GUI 専用のファイルを用意しないと、GUI のメリットを活かした画面にはならないと思います
- 複数言語の選択：だいぶ実装されてきてはいるようです
- フロッピーの取り扱い：今回は技術上、カーネル 2.4 のみの対応としています (カーネル 2.6 はフロッピーに入らない)。カーネルを分割してロードできないか？

10 Debian ツールチェインと glibc, linux-kernel-headers パッケージ

後藤正徳 [*11]

10.1 はじめに

先ほどリリースされた Debian sarge では 11 ものアーキテクチャをサポートしている。これら異なるアーキテクチャが同時に使い物になるためには、当然各アーキテクチャ用バイナリを生成するツールチェインも十分整備されていなければならない。

本文書では、ツールチェインとは何かを簡単に紹介した後、筆者がメンテナンスしている glibc, linux-kernel-headers パッケージに関して説明を行う。

10.1.1 ツールチェインとは

ツールチェイン (toolchain) という用語に正確な定義があるわけではないが、一般にマシンネイティブなバイナリを生成・加工するための一連のツール群を指すことが多い。本節では、まず Debian での代表的なツールチェインパッケージを紹介する。

10.1.2 gcc

GNU Compiler Collection の略称。ツールチェインのコアパッケージであり、ソースコードからアーキテクチャ毎のバイナリへ変換する役割を持つ。なお、gcc 自体はソースコードからアセンブラコードを出力するまでの機能しか持っておらず、そこから先のバイナリ生成部分は後述する binutils のツールを内部で呼び出している。現在サポートするプログラム言語としては C (gcc), C++ (g++), Java (gcj), Fortran (g77, g90) などがある。

Debian パッケージメンテナは Debian-gcc チーム (Matthias Klose, Gerhard Tonn) であるが、事実上 Matthias (doko) 一人がメンテナンスを担っている。

10.1.3 binutils

binutils には、バイナリを生成・操作するツール群が入っている。例えば、gcc によって出力されたアセンブラコードをアセンブルしてオブジェクトコード化するアセンブラ as、複数のオブジェクトコードをリンクしてバイナリを生成するリンカ ld、オブジェクトコードを逆アセンブルしたり解析したりするツール objdump などがある。

Debian パッケージメンテナは James Troup である。最近では C++ transition for etch に絡んで Matthias Klose が主な実作業にあたっている。また、binutils の上流開発者の中で Debian 開発者も兼任している人物に Daniel Jacobowitz がいる。

[*11] GOTO Masanori, gotom@debian.org, Debian Project, 2005-07-02

10.1.4 glibc, linux-kernel-headers

glibc は GNU C Library の略称。gcc, binutils がバイナリを生成するものであるのに対し、glibc は生成したバイナリ実行時に使用される C 言語ライブラリである。C 言語の関数やバイナリ実行時に最初に必要となる初期実行コードなどを含んでいる。なお、C ライブラリは実行環境に依存するため、システムによっては glibc 以外の libc が使用されることもある[*12]。

linux-kernel-headers は glibc のうち /usr/include/linux など Linux カーネルソース起源のヘッダを扱うパッケージである。元々 libc6-dev パッケージにマージされていたが、ソースが別々であることから 2 つに分離された。

10.1.5 gdb

gdb はバイナリ実行時に、ユーザからデバッグを可能とするための機能を提供する。

Debian パッケージメンテナは上流開発者でもある Daniel Jacobowitz である。

10.1.6 **その他**

その他のものとして C++ 向けライブラリである libstdc++ などがある。また、bfd といったバックエンドライブラリや dejagnu といったツール群が存在しているが、本文書では割愛させて頂く。

10.2 glibc, linux-kernel-headers **パッケージの概要**

10.2.1 **歴史**

過去をひもとくと、元々 Linux の C ライブラリとしては H. J. Lu を中心に開発されていた libc4, libc5 が使用されていた。これは GNU C Library バージョン 1 をベースに Linux 向け改良が追加されたものであった。しかし、GNU C Library 自体も Roland McGrath, Ulrich Drepper, Andreas Jaeger を中心に別途開発が続けられており、古い libc5 に代わってより新しい現在の glibc バージョン 2 が libc6 として置き換わる移行が行われた (libc6 transition)。

Debian における glibc パッケージは、当初 Joel Klecker がメンテナンスしていた (1999〜2000)。しかし Joel が病気で逝去したため、代って Ben Collins がメンテナンスを開始した (2000〜2002)。だが、やがて Ben 自身の興味が薄れて放置されるようになってきたため、現在のメンバから構成されるメンテナンスチームが組まれた (2002〜)。その後 Jeff Bailey, Branden Robinson, Daniel Jacobowitz を中心に古い Debian パッケージから debhelper ベースへ完全に書き直され、同時に linux-kernel-headers パッケージが libc6-dev パッケージから分離した (2003)。現在は筆者を中心にメンテナンスが続けられている。

10.2.2 **パッケージメンテナ**

パッケージメンテナは Debian-glibc チーム。構成メンバは以下の通り。

- Ben Collins (benc) … 元 Debian Project Leader。Debian/SPARC ポートメンバ。svn 開発者、Linux ieee1394 カーネルサブシステムメンテナ。
- GOTO Masanori (gotom) … 筆者。glibc 開発者、Linux SCSI カーネルドライバメンテナ。

[*12] 例えば組み込み環境では GNU newlib、Debian netbsd-i386 では、NetBSD 用 libc など。

- Philip Blundell (pb) … Debian/ARM ポートメンバ。glibc を含む上流 ARM ツールチェインや Linux/ARM カーネルのメンテナ。
- Jeff Bailey (jbailey) … Debian/GNU hurd-i386 ポートメンバ。現在 Ubuntu にて ppc64 など先進的なツールチェインメンテナンスを行っている。
- Daniel Jacobowitz (drow) … Debian/PowerPC ポートメンバ。glibc, gdb, gcc, binutils などをフルタイムでハック。最近の glibc まわりでは MIPS TLS や ARM new EABI などをアクティブに作業している。

この他にも以下のメンバが活躍している。移植周り: Bastian Blank、debian-installer 関連: Colin Watson、locale: Petter Reinholdtsen, Denis Barbier、MIPS: Thiemo Seufer, Guido Guenther、hppa: Carlos O'Donell, amd64: Andreas Jochens, Goswin von Brederlow, ia64: David Mosberger, Randolph Chung, s390: Gerhard Tonn。

10.2.3 開発体制

開発は主に debian-glibc@lists.debian.org メーリングリストを中心に行われている。また、時々 IRC にて議論を行って今後の方針などを決定している。パッケージ管理は元々 cvs で行っていたが、現在は alioth の svn ベースに切り替わっている。レポジトリは以下の通り。

```
svn://svn.debian.org/svn/pkg-glibc/glibc-package/trunk
svn://svn.debian.org/svn/pkg-glibc/linux-kernel-headers/trunk
```

10.3 glibc バイナリパッケージとその中身

本節では、glibc の各種バイナリパッケージとそれに含まれるファイルを紹介していく。各ファイルの解説によって、glibc がどのような機能を提供しているかの一端が明らかになるだろう。

ところで、パッケージの中には同じ機能を提供しているにも関わらずアーキテクチャによってついている名前が異なることがある。具体的に i386 での名前を挙げると libc6, libc6-dev, libc6-dbg, libc6-pic, libc6-prof, libc6-udeb がそうである。libc6 を例にアーキテクチャとパッケージ名の対応を表 5 に挙げる。以後では、一番親しんでいるであろう i386 での名前を使用して説明を進める。

パッケージ名	アーキテクチャ
libc6	amd64, arm, i386, m68k, mips, mipsel, powerpc, sparc, s390, hppa, sh3, sh4, sh3eb, sh4eb
libc6.1	alpha, ia64
libc0.3	hurd-i386
libc1	freebsd-i386

表 5　呼称が変化するパッケージ名とアーキテクチャとの対応関係

10.3.1 libc6

C ライブラリとダイナミックローダ等を提供。Priority: required, Section: base。

sid/i386 では約 9100 バイナリパッケージが依存している Debian のコアパッケージ。本パッケージは以下のファイルを含む。

/lib/*.so libc のコアとなる動的ライブラリ[*13]。動的ローダ (i386 では /lib/ld-linux.so.2) と libc (i386 では libc.so.6)、そして周辺ライブラリ (libm, libnss_*, ...) を含む。スレッドシステムにはカーネル 2.4 でも動作する linuxthreads を採用。

/lib/tls/*.so /lib/*.so と同様だが、TLS (Thread Local Storage) が有効になり、デフォルトのスレッドシステムが規格により正しく準拠した NPTL (Native Posix Thread Library) になっている。カーネル 2.6 使用時には、デフォルトの /lib/*.so 動的ライブラリに代わってロードされる。

/lib/*.so.*, /lib/tls/*.so.* 各ライブラリのバージョン番号を指し示すためのシンボリックリンクファイル。

/usr/lib/gconv/* gconv (各文字コード・エンコーディング毎に用意されているコード変換 iconv モジュール) 動的ライブラリと gconv-modules (エイリアスファイル)。

/usr/lib/* pt_chown をインストール。

/usr/share/zoneinfo/* コンパイル済タイムゾーンデータ。環境変数 TZ に指定されたタイムゾーンに応じた時刻を返すために使用される。

/usr/bin/ 各種設定・変換・表示ツール (iconv, locale, localedef, getent, getconf, catchsegv, tzselect, ldd, lddlibc4, zdump, rpcinfo) をインストール。

/usr/sbin/* 設定ツール (zic, iconvconfig, tzconfig) をインストール。

/sbin/* 動的ライブラリキャッシュファイル作成ツール ldconfig をインストール。

/sys sysfs のためのマウントポイント。歴史的事情により存在。

10.3.2 libc6-sparc64, libc6-s390x

64 ビット biarch[*14]向けに作成されている libc6 パッケージ。Priority: standard, Section: base。

Debian sarge では sparc64, s390x にのみ提供されている。基本的にパッケージの中身は libc6 パッケージと同等であるが、64 ビット化した動的ライブラリを /lib の代わりに /lib64, /usr/lib64 配下へインストールする。

10.3.3 libc6-i686, libc6-sparcv9, libc6-sparcv9b

アーキテクチャ依存で提供される最適化パッケージ。Priority: extra, Section: base。

opt パッケージ、hwcap パッケージとも呼ばれる。基本的にパッケージの中身は libc6 パッケージと同等

[*13] 動的ライブラリ (dynamic linked library) とは .so で終わる名前を持ち、バイナリ実行時に動的ローダ (dynamic loader) によって必要に応じてロードされるライブラリファイル。これに対し、静的ライブラリ (static linked library) とは .a で終わる名前を持ち、コンパイル時にバイナリへライブラリコードを直接埋め込んでしまうために用いられるファイル。

[*14] bi: 2 つの、arch: アーキテクチャ。biarch とは、2 つのアーキテクチャを同時に使用可能なこと。

であるが、例えば libc6-i686 では /lib のかわりに /lib/tls/i686/cmov というディレクトリの下に最適化された動的ライブラリがインストールされる。この最適化動的ライブラリは、アーキテクチャが i686 以上かつカーネルが 2.6 以上でプロセッサに cmov 拡張がある[*15]場合、バイナリ実行時に自動的に使用される。なお libc6-i686 は 686 用に最適化しているだけにとどまらず、スレッドサポートが大きく改善されているという重要な特徴がある。

10.3.4 libc6-dev

開発向けパッケージ。Priority: standard, Section: libdevel, Build-Essential。

C, C++ での開発を行うためには必ず必要になる各種ファイルがインストールされる。本パッケージは以下のファイルを含む。

/usr/include/*	ヘッダファイル一式を格納。
/usr/bin/*	gencat, mtrace, rpcgen といった開発用ツールを提供。
/usr/lib/*.a, /usr/lib/*.o	libc6 で提供される動的ライブラリのうちのいくつかが静的ライブラリとしてインストールされる。また、コンパイル時に必ず静的にリンクされるオブジェクトコードが含まれる。
/usr/lib/*.so	libc6 に含まれる /lib/*.so 動的ライブラリへのシンボリックリンクファイル。コンパイル用。

10.3.5 libc6-dev-sparc64, libc6-dev-s390x

64 ビット biarch 向けに作成されている libc6-dev パッケージ。Priority: standard, Section: libdevel。

Debian sarge では sparc64, s390x にのみ提供されている。基本的にパッケージの中身は libc6-dev パッケージと同等であるが、64 ビット化した動的ライブラリを通常とは別パスへインストールする。

10.3.6 libc6-dbg

デバッグライブラリパッケージ。Priority: extra, Section: libdevel。

libc6 や libc6-i686 パッケージで提供される動的ライブラリは strip されてシンボル情報が落とされているが、本パッケージで提供される動的ライブラリはシンボル情報も含んだものが/usr/lib/debug 配下にインストールされる。

本パッケージは、開発時など glibc も含めた gdb デバッグを行う際に使用する。LD_LIBRARY_PATH 環境変数に /usr/lib/debug パスを含めてアプリを起動すると利用できる。

10.3.7 libc6-prof

プロファイルライブラリパッケージ。Priority: extra, Section: libdevel。

libc6-dev で提供される静的ライブラリとほぼ同じだが、プロファイルオプション (-pg) つきでコンパイルされている。gprof を使ってアプリの性能プロファイリングを行いたいときに利用する。

[*15] cmov 拡張は、VIA C3 以外の全ての 686 クラスプロセッサが持っている。

10.3.8 locales

国際化機能 “ロケール” 用データを提供するパッケージ。Priority: standard, Section: base。

Debian で日本語を使いたい場合には必ずインストールされている必要がある。本パッケージは以下のファイルを含む。

/usr/share/locale/* 国際化 (gettext 化) した glibc の出力メッセージデータ (.po)。

/usr/share/i18n/charmaps/* ロケールデータのうち文字コードに関するデータ。

/usr/share/i18n/locales/* ロケールデータのうちロケール情報に関するデータ。

/usr/sbin/locale-gen ロケールデータ生成スクリプト。/etc/locale.gen に基づいてロケールデータを生成し/usr/lib/locale へインストールする。

元々locales パッケージには、コンパイル済のロケールバイナリデータ全てが含まれていたが、使いもしないロケール情報にサイズを食われることに強い反対意見があった。そこで現在は、各ユーザが charmaps (ja_JP.eucJP の eucJP) と locales (ja_JP.eucJP のうち ja_JP) データを使用してユーザが必要とする分だけ手元で生成出来るようになっている。“dpkg-reconfigure locales” によって生成するロケールデータを随時設定変更可能。

10.3.9 nscd

NSCD (Name Server Cache Daemon) パッケージ。Priority: optional, Section: admin。

デフォルトで提供される glibc ネームサービススイッチライブラリ (libnss_*) は、毎回関数を呼び出す度にネットワークへ問い合わせに行く仕様になっている。例えば libnss_dns は、DNS を引くための機能を提供するが、DNS を引く関数を呼ぶ度にネームサーバへ通信が発生してしまう。nscd を使うと、ローカルへ問い合わせをキャッシュするので、その分速度を速めることが可能になる。

10.3.10 glibc-doc

ドキュメントパッケージ。Priority: optional, Section: doc。

glibc に関するドキュメント (/usr/share/doc/glibc-doc/*) と、公式マニュアルである info (/usr/share/info/libc.info*)、そしていくつかのマニュアルページ (/usr/share/man/man3/*) が提供される。

なお、manpages-dev で提供されるマニュアルページと glibc-doc の info ドキュメントは、互いに提供元も内容も異なる全くの別物である。

10.3.11 libc6-pic

PIC アーカイブパッケージ。Priority: extra, Section: libdevel。

元々 boot-floppies のために用意されていたパッケージ。フロッピに入れるには大きすぎる glibc ライブラリから必要な関数だけ抽出し、小さくカスタマイズするために使用する。現在も debian-installer の mklibs で使用されている。

10.3.12 libc6-udeb, libnss-dns-udeb, libnss-files-udeb

debian-installer 用 udeb パッケージ。Priority: extra, Section: debian-installer。

```
/* All asm/ files are generated and point to the corresponding
 * file in asm-sparc or asm-sparc64.
 */

#ifdef __arch64__
# include <asm-sparc64/delay.h>
#else
# include <asm-sparc/delay.h>
#endif
```

図 2　sparc における /usr/include/asm/delay.h の例。#ifdef によって読込むファイルを切り替えている。

libc6-udeb は debian-installer が動作するために必要な最低限の動的ライブラリのみ含む。libnss-*-udeb は ssh を動作させるために s390 用に導入されたもので、フロッピ容量の関係で libc6-udeb から外された動的ライブラリが入る。

10.4　linux-kernel-headers バイナリパッケージとその中身

linux-kernel-headers は、libc6-dev とともに C, C++ 開発時に必要となるパッケージである。ソースは glibc とは異なり Linux カーネルを元にしている。Priority: standard, Section: devel, 事実上の Build-Essential。

中身のほとんどは Linux カーネルヘッダファイルで構成される。本パッケージは以下のファイルを含む。

/usr/include/linux/* Linux カーネルソースでいう include/linux ディレクトリにあるヘッダファイルをコピーしたもの。アーキテクチャ非依存。

/usr/include/asm/* include/asm-* ディレクトリにあるヘッダファイルをコピーしたもの。アーキテクチャ依存。

/usr/include/asm-generic/* include/asm-generic ディレクトリにあるヘッダファイルをコピーしたもの。アーキテクチャ非依存。

ちなみに、アーキテクチャによっては 32 ビットと 64 ビットのカーネルヘッダが別々になっているものがある (sparc, ppc など)。その場合、asm に 32 ビットアーキテクチャ用ヘッダだけが入っていると、64 ビットバイナリをコンパイル出来ないことになってしまう。そこで、asm 以下には振り分けのためのダミーファイルを入れておき、コンパイルオプションの #ifdef によって適宜読むファイルを切り替える仕組みになっている (例を図 2 に示す)。

10.5　glibc, linux-kernel-headers ソースパッケージとその中身

本節では、ハックの一助になるよう glibc, linux-kernel-headers ソースパッケージの中身を簡単に紹介する。

10.6 glibc

glibc ソースパッケージは以下のような構成になっている。ここでは、特に重要な役割を持つファイルを取り上げる。

***.tar.bz2** ソースパッケージ。

prep.sh .dsc ファイルがなくても .orig.tar.gz からソースを解凍可能にするスクリプト。

debian/control, debian/control.in/* control ファイルとその元ファイル control.in/*。debian/rules control として control.in/* から control ファイルを生成する。

debian/debhelper.in/* 各バイナリパッケージ用 debhelper ファイルの雛形を格納。

debian/po/* locales パッケージで使用する debconf フロントエンド用国際化メッセージファイル。

debian/local/* Debian ローカルでインストールするファイル (例えばマニュアルや設定ファイル等) を格納。

debian/patches/* Debian ローカルパッチ。dpatch をベースにした独自のパッチシステムで管理。最新版では 66 個、総計 452KB (!) ある。

debian/rules, debian/rules.d/* ルールファイル rules と、ビルドの各段階に分けてより細かく記述したルールファイル rules.d/* から成る。

debian/shlibver shlib バージョン定義ファイル。

debian/wrapper/* -dbg 生成時用ラッパスクリプト。

debian/sysdeps/* ビルド時に各アーキテクチャ依存の情報を切り替えるための Makefile 集。

10.7 linux-kernel-headers

linux-kernel-headers ソースパッケージは以下のような中身となっている。

autoconfs アーキテクチャ毎に用意されているデフォルトカーネルコンフィグファイル /usr/include/linux/autoconf.h を格納[*16]。

debian debian ディレクトリ。

debian/patches Debian ローカルパッチが入っている。dpatch にて管理。

others ia64, parisc 用の特別ファイル。

testsuite テストプログラム。gcc-2.95, gcc-3.3, g++-3.3 に対して -ansi などのフラグをチェックしてコンパイルエラーが出ないか調べる。

version.h 出自のバージョン番号。/usr/include/linux/version.h としてインストールされる。

なお、debian/patches 以下には現在 36 個、総計 61KB 分のローカルパッチが入っている。これをパッチ種別毎に分類したものが表 6 である。

[*16] 余談だが、本資料を書いている途中で autoconf.h は biarch に対応していないというバグに気付いた。

分類	パッチ数
GCC -ansi 問題関連	9 個
#ifdef __KERNEL__ を忘れたヘッダへの追加/削除	8 個
削除されたシステムコールを元に戻す	2 個
include すべきでないヘッダを取り込んだ時の対策	5 個
クリーンナップ (適切なヘッダを取り込むなど)	5 個
glibc にあわせるため	7 個

表 6　linux-kernel-headers パッケージに含まれるパッチの分類

10.8　苦労話

libc6 パッケージは、通常のパッケージと比較して以下の点が大きな違いである。

- ほぼ全ての Debian ユーザが使用。
- バイナリパッケージの大半 (約 9100 以上) が依存。
- カーネルなどと同様に、アーキテクチャ毎に使用するソースが大きく異なるにも関わらず、全アーキテクチャで同じバージョンを提供しなければならない。

そこで、ここでは実際に起きた問題の一つを取り上げ、メンテナンスにあたって発生する様々な苦労の一端を紹介したい。

10.8.1　glibc 2.3.4-1 (experimental) をインストールすると起動しない問題

発生した問題

これは glibc 2.3.4-1 を experimental に dupload した時に発生した問題である。

i386 アーキテクチャでは、libc6 の他に最適化ライブラリ libc6-i686 が用意されている。カーネル 2.6 + i686 アーキテクチャを使っている場合、libc6-i686 に入っているとバイナリ実行時に/lib/tls/i686/cmov 以下の最適化動的ライブラリが優先して使用されることは既に述べた。この「どの動的ライブラリをロードするのか」をバイナリ起動時に実際に判断するのは、libc6 パッケージに入っている動的ローダ ld.so である。

ところが glibc 2.3.4 になってから、ld.so と libc.so 共通で使用している内部構造体に変更が発生し、新旧の libc.so と ld.so を混ぜて使用できなくなってしまった。これが原因で、以下のようなケースに陥ると、全バイナリがクラッシュして動作しなくなってしまうようになった[*17]。

- sarge の旧 libc6 (2.3.2.ds1) と旧 libc6-i686 (2.3.2.ds1) パッケージを先にインストールした状態で、新 libc6 (2.3.4) にアップグレードする場合。この時、新 libc6 で提供される新しいバージョンの ld.so が、旧 libc6-i686 で提供される古いバージョンの libc.so を読み込んでしまい、クラッシュする。
- 新 libc6 + 新 libc6-i686 をインストールした環境に旧 libc6 を入れ直すとやはりクラッシュする。

[*17] この問題は experimental を常用する強者ユーザによって発覚した。もし、unstable に直接 glibc をアップロードしていたら、それこそ阿鼻叫喚の騒ぎになったことだろう…。

- sparc には libc6 の他に libc6-sparcv9, libc6-sparcv9b という 2 種類の最適化パッケージがあり、それぞれを片方ずつもしくは両方をインストールしたり削除したりしても、クラッシュしてしまう。

状態遷移図の登場

上記のように、パッケージをインストールしたり削除したりといった状況に応じて、問題が起きたり起きなかったりする。そこで、結局最後は libc6 や libc6-i686 パッケージのインストール状況毎に応じて状態遷移図を書いて問題を整理した。それが、図 3 である。

例えば、Lo (libc6 2.3.2.ds1) + Po (libc6-i686 2.3.2.ds1) がインストールされている状態 Lo,Po から Ln (libc6 2.3.4) をインストールしようとすると、色付きの不整合状態 Ln,Po に陥ることが図から読み取ることができる。

解決策

この図 3 に基づき、インストールや削除、アップグレードやダウングレードでも問題が発生しないように工夫した glibc を作成した。具体的には、Debian ローカルパッチで使用可能になっている /etc/ld.so.nohwcap というファイルを場合に応じて制御するというものである。このファイルが存在すると、ld.so は最適化ライブラリを使用せず、libc6 に入っているデフォルトライブラリを使うようになる。つまり、ld.so と libc.so のバージョンはいつでも一致するようになってクラッシュしなくなるわけだ。そこで .preinst/.prerm に手を加え、インストールまたは削除される状態に応じてこのファイルを生成したり削除したりするような変更を行った。

i386 では、上記手法によって問題解決することを確認した。ただし、sparc 版では同様のテストを行える環境がないため、新しい glibc を sid にアップロードすると sparc ユーザによってはシステムクラッシュが経験できるかもしれない。

10.9 etch の TODO

glibc, linux-kernel-headers には、まだまだ様々な作業が積み残されている。本節では今後 etch にて行われる予定の作業のうち、Debian 全体に与える影響が比較的大きいものをピックアップして紹介したい。

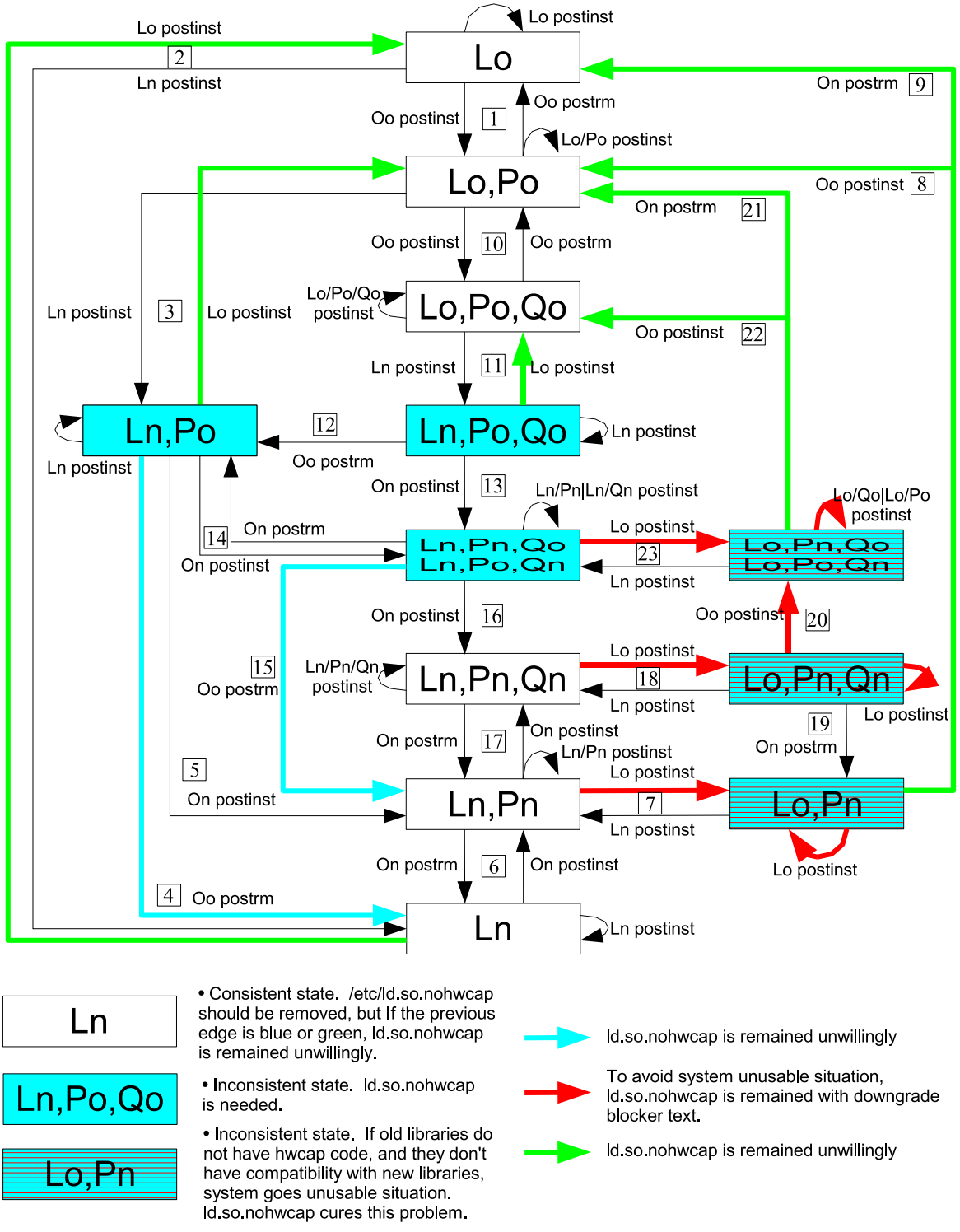
10.9.1 etch におけるツールチェイン移行計画

sarge がリリースするまでには非常に長い年月がかかったため、主要なツールチェインは全て昔のバージョンに塩漬けされた状態になってしまった。これを取り戻すべく、etch では最新版を 7 月に次々と投入予定である。最近の doko + jbailey + gotom の議論によって、次の順番でツールチェインを移行していこうという話になっている。

1. linux-kernel-headers: 2.6.0-test7 → 2.6.12 (sid では順次カーネルバージョンに追従予定)
2. binutils: 2.15 → 2.16
3. gcc: 3.3 → 4.0
4. glibc: 2.3.2.ds1 → 2.3.5 (experimental はさらに最新版へ追従予定)

Debian Glibc package 2.3.2.ds1-20 (sarge) <=> 2.3.4-3 (etch) Diagram for breakable package upgrade/downgrade with sparcv9/sparcv9b optimized packages.

- gotom 2005-04-03 ver.5



Ln

• Consistent state. /etc/ld.so.nohwcap should be removed, but If the previous edge is blue or green, ld.so.nohwcap is remained unwillingly.

Ln,Po,Qo

• Inconsistent state. ld.so.nohwcap is needed.

Lo,Pn

• Inconsistent state. If old libraries do not have hwcap code, and they don't have compatibility with new libraries, system goes unusable situation. ld.so.nohwcap cures this problem.

ld.so.nohwcap is remained unwillingly

To avoid system unusable situation, ld.so.nohwcap is remained with downgrade blocker text.

ld.so.nohwcap is remained unwillingly

図 3　libc6, libc6-i686, libc6-sparcv9, libc6-sparcv9b 全てを考慮したインストール・削除パス

この移行の中で最も影響が大きいのが gcc 4.0 である。gcc 4.0 では C++ ABI (Application Binary Interface) や、いくつかのアーキテクチャ(sparc, mips など) で関数の呼び出し規約に変更が加わっている。特に、C++ ABI 移行では、C++ を利用する全てのパッケージが一時的に名前を変えるなどの対処が必要になってくる。より詳細は以下の URL を参照のこと。

```
"C++ ABI transition for etch"
http://lists.debian.org/debian-release/2005/04/msg00153.html
```

10.9.2 multiarch サポート

現在の Debian は、いわゆる 64 ビットの biarch を部分的にサポートしている。とは言え、64 ビットをサポートするパッケージは libc6 や libncurses など一部のライブラリパッケージのみで、それも debian/rules でインストール先を /lib から /lib64 に振り分けるといった仕掛けを作り込んで実現している。

しかし近年、amd64 や ppc64 など、32 ビットと 64 ビットを両方使えるアーキテクチャが急速に普及してきており、より簡単に両者のバイナリを扱えるパッケージフレームワークの整備が求められている。また、3 種類以上のアーキテクチャを同時に使える multiarch [*18] も可能にしたいという意見がある。

現時点でどう実装していくのかについて議論はまとまっていないが、動的ローダに変更を加える可能性は高い。より詳細は以下の URL を参照のこと。

```
"Multiple architecture problem and proposed solution"
http://www.linuxbase.org/futures/ideas/multiarch/
```

10.9.3 新しいアーキテクチャのサポート

Debian sarge は 11 + 1 アーキテクチャにポーティングされているが、さらにもっと多くのアーキテクチャへポーティングしたいという声がある。

ppc64

現在最もサポートが望まれているのは ppc64 である。Debian で最初にネイティブ ppc64 の開発作業を開始したのは amd64 サポートでも中心的な役割を果たした Andreas Jochens であった[*19]。その後 debian-powerpc メーリングリストにおける議論で、ppc64 は amd64 と比較してネイティブサポートのメリットがそれほど大きくない[*20]ため、現在は biarch サポートでいく方針に固まりつつある。

sh3, m32r, mips64, parisc64

既にいくつかポーティングされているものもあるが、積極的にサポートする方向にはなっていない。sh3 に関しては、マシンの整備が進めば状況はもう少し良くなると考えている。

[*18] 例えば mips にはエンディアン 2 種類とは別に o32, n32, n64 といった異なる ABI が存在する。

[*19] http://debian-ppc64.alioth.debian.org/ を参照のこと。

[*20] ppc32 から ppc64 へはアドレス空間の拡大が中心でありレジスタ数も変わっていないため、ビット幅が増えた分、性能的には遅くなってしまう。

10.9.4 locales-all パッケージの提供

locales パッケージの解説で述べたように、現在コンパイル済ロケールデータはパッケージとして提供されておらず、ユーザがインストール時に必要なものだけコンパイルするようになっている。しかし、このロケールデータのコンパイルにはかなりのメモリと時間を食うため、特に組込み向けシステムでは生成が大変厳しいというバグレポートが報告されている。

そこで、locales パッケージの他に locales-all パッケージを用意し、コンパイル済ロケールデータをあらかじめ用意しておくことで、ロケール生成にかかる時間を節約できるようになる予定である。

10.9.5 timezone パッケージの作成

timezone データは libc6 パッケージに統合されているが、必ずしも必要なデータではないため、組込み機器向けでは libc6 から切離してスリムにしたいという要望がある。また、timezone データそのものは glibc 開発者がメンテナンスしているわけではなく、glibc とは無関係に頻繁に更新されている。

そのため、etch では timezone データを libc6 パッケージから切り離して独立させていく予定である。これまで安定版では timezone データを更新したくても一緒に glibc 一式を再コンパイルしなければならないリスクがあったが、別パッケージにすることでそれを回避できるというメリットもある。

10.9.6 カーネル 2.2 サポートの廃止

現在の Debian glibc は、Linux カーネル 2.2 から 2.6 までをサポートしている。しかし、既にカーネル 2.2 シリーズはメンテナンスされなくなりつつあり、Debian から消えつつある。

glibc はカーネルによってコンパイルするソースを切り替えているが、最適化パッケージをインストールしていない限り、いつまでもカーネル 2.2 の頃にあった古いシステムコールを利用してしまう。そこで、etch ではカーネル 2.2 サポートを廃止し、カーネル 2.4 以上でなければ動作できなくする予定である。

10.9.7 kernel version detection

woody → sarge の移行では、mips, sun4m, hppa, hppa64, real-i386 などがカーネルを最新版へアップグレードしない限り glibc も入れ替えられない状態が発生した。これは、カーネルと glibc の仕様が一緒に変更されたためである。

しかし、一旦カーネルと glibc をアップグレードした後、カーネルを昔のバージョンに戻して再起動されるとシステムがうまく動作しなくなる可能性もある。

そこで、カーネル 2.2 サポート廃止にあわせて、古いカーネルを検出した場合は/etc/init.d/glibc-kernel-check といったスクリプトによってユーザに警告を発せられるような仕組みを整えていく予定である。

10.9.8 NPTL のデフォルト化

RedHat の RHEL4 や Ubuntu ia64, ppc 版では linuxthreads に代わって NPTL が標準 PThread ライブラリとして採用されている。既に上流開発者も linuxthreads は今後メンテナンスをほとんど行わない予定と宣言している。近い将来 Debian でも 2.6 カーネルが主流になってくれば、linuxthreads と 2.4 カーネルはサポートを止めていく方向になるだろう。

10.9.9 バージョンアップの高速化

Debian では sarge の大幅なリリース遅れにより、glibc や linux-kernel-headers がかなり古いバージョンになってしまった。これは、リリースマネージャによるベースパッケージフリーズの影響が大きい[*21]。etch では、SCC の導入によってマイナーアーキテクチャの FTBFS に足をひっぱられなくなることもあるので、出来るだけ最新版を unstable に積極投入していきたい。

また、upstream とのより緊密な開発体制を敷くためにも、experimental を積極活用し、cvs 最新版を出来るだけ experimental へ投入していきたいと考えている。

10.10 おわりに

本文書では主に glibc, linux-kernel-headers を中心に Debian ツールチェインの解説を行なった。良く使われてはいるものの、普段あまり中身を知る機会のないと思われるパッケージであるが、これを機会に関心を持っていただければ幸いである。

[*21] リリースマネージャも sarge リリース間近のときに、2004 年 8 月に実施したベースパッケージフリーズは、ライブラリを無意味に古くさせてしまったと回想している。

11 dpatch をつかってみよう

上川純一

11.1 dpatch とは

Debian のソースパッチを管理するツールです． Debian パッケージでは，ソースパッケージは以下の構成になっています．

- .orig.tar.gz: オリジナルの tarball
- .diff.gz: Debian で作成した差分
- .dsc: dpkg 用制御ファイル

この中で，.diff.gz は一つの大きな差分ファイルとして管理されるため， どの部分がどういうパッチであるか，ということを管理してはくれません． その部分を実装するのが dpatch です．

通常の.diff.gz であると， debian/ディレクトリ以下の Debian パッケージング用の情報と それ以外のソフトウェア自体への修正が混合しています． それを整理するというものです．

11.2 ファイル構成

dpatch では，それぞれの小さな変更をそれぞれ独立したパッチとして扱います．それぞれのパッチを debian/patches/xx_patchname.dpatch という名前 で管理します．例えば，debian/patches/01_configure.dpatch という名前になります． そして，パッチの一覧を debian/patches/00list に適用する順番に記述します．そこでは，01_configure というような形式で記述できます．

この形式を採用しているため，Debian においての変更点が debian/ディレクトリ以下に集まり， また変更点を debian/patches ディレクトリで分類して管理できる，という利点があります．

dpatch でのファイル名が数字ではじまるのは昔はその番号で適用順序を決定していたなごりです．今はあまり数字を利用する必要性はありません．debian/patches/00list ファイルに指定した順番でパッチは適用されます．

11.3 道具

dpatch を利用するための道具を紹介します．

11.3.1 dpatch

dpatch コマンドは，パッチの適用とパッチをはずすという処理を実施してくれるコマンドです．従来は，makefile から include する Make スクリプトとして実装されていましたが，dpatch 2.0 からは実体が /usr/bin/dpatch シェルスクリプトになっています．

古い debian/rules では下記の内容を記述しています.

```
include /usr/share/dpatch/dpatch.make
```

最近の dpatch を利用するソースでは，dpatch コマンドを呼ぶように実装すればよいことになっています.
(/usr/share/doc/dpatch/examples/rules/rules.new.dh.gz から抜粋)

```
#!/usr/bin/make -f
#
# Sample dpatch rules file. Only example. Nothing else. :)
# This one uses the new way with dpatch from dpatch 2.x

export DH_COMPAT = 4

CFLAGS = -Wall -g
ifneq (,$(findstring noopt,$(DEB_BUILD_OPTIONS)))
CFLAGS += -O0
else
CFLAGS += -O2
endif

build: build-stamp
build-stamp: patch
        @echo "--- Compiling"
        dh_testdir
# Do something to build your package here
        touch build-stamp

clean: clean1 unpatch
clean1:
        @echo "--- Cleaning"
        dh_testdir
        dh_testroot
        dh_clean -k
# Clean your build tree

install: build-stamp
        dh_testdir
        dh_testroot
        dh_clean -k
        dh_installdirs
# Install it here

# Build architecture-independent files here.
binary-indep: build install

# Build architecture-dependent files here.
binary-arch: build install
        dh_testdir
        dh_testroot
        dh_installdocs
# And all the other dh_* stuff you need for your package.

# And now the simple things for dpatch. Here we only apply/unapply the patches.
# You can do more things with dpatch, like having patches only applied on
# a special architecture - see the non-dh version of the sample for this!
patch: patch-stamp
patch-stamp:
        dpatch apply-all
        #dpatch call-all -a=pkg-info >patch-stamp

unpatch:
        dpatch deapply-all
        rm -rf patch-stamp debian/patched

binary: binary-indep binary-arch
.PHONY: binary clean binary-indep binary-arch build install patch unpatch \
        clean1
```

11.3.2 dpatch-edit-patch

dpatch で利用するためのパッチを作成するコマンドです.

基本的な使い方は dpatch 管理下にあるソースコードのディレクトリで,

```
dpatch-edit-patch -d '説明文' 03_patchname 02_patchname
```

のように入力すると，二つ目のパラメータに指定したパッチまでのパッチを適用した状態で，一時ディレクトリにソースを展開してくれ，シェルが起動します．パッチ名には.dpatch 拡張子をつける必要はありません.

編集したのち，シェルから出ると，一つ目のオプションに指定した名前のパッチを作成してくれます.

debian/patches/00list ファイルは編集してくれるわけではないので，自分で編集する事になります.
debian/patches/00list ファイルを編集してパッチの名前 (.dpatch 拡張子をぬいたもの) を追加したらそのパッチが適用されるようになります.

11.3.3 dpatch.el

emacs 上で dpatch を使うために必要な，00list ファイルや，.dpatch ファイルの編集用のモードです.

まだまだ未完成です.

目標は dpatch-edit-patch の実装ですが，そこに至る前のコアの dpatch の部分の変更をしているだけで現在は時間が過ぎていってます.

11.4 作業フロー

作業のフローについては，これよりよい方法がある，などありましたら教えてください.

11.4.1 あたらしい upstream が出た時

Debian パッケージとして管理しているソフトウェアで，もととなっているパッケージ (upstream package) の新しいバージョンが出た場合の対応です.

- 前のバージョンから debian/ディレクトリをコピーしてくる
- `debian/patches/xx_patchname.dpatch` をそれぞれ `patch --dry-run -p1` で適用できるかどうか確認する
- `debian/patches/00list` を編集
- `debian/rules patch`
- **適用できないパッチの再作成**

11.4.2 あたらしく package をつくる時

新規にパッケージを作成する手順です.

- `debian/rules` を変更し，`/usr/share/dpatch/dpatch.make` を include し， clean と configure のルールで patch と unpatch ルールが呼ばれるようにする (patch-stamp 等を利用)
- touch コマンドなどで `debian/patches/00list` を作成する
- `dpatch-convert-diffgz` を実行して，とりあえず dpatch ファイルに変換する
- debian/patches/*dpatch ファイルを適当に編集して複数ファイルに分割
- `debian/rules patch` と `debian/rules unpatch` が成功することを確認
- `dpkg-buildpackage` で生成される diff.gz の diffstat をとり，debian 以下以外の場所が diff に含まれていないことを確認
- `dpatch-edit-patch` を利用して追加のパッチを作成

11.4.3 あたらしく patch を作成

- dpatch-edit-patch パッチ名とする.
  例： `dpatch-edit-patch automake`
  もし他のパッチに依存する変更をするなら，そのパッチ名を入力します.
  例 `dpatch-edit-patch automake autoconf`.

これを実施すると，/tmp 以下に適当なディレクトリでソースが編集できるようにシェルが起動します.

- ソースを適当に編集します.
- exit すると，パッチファイルが debian/patches ディレクトリ以下に生成されます.
- 生成されたパッチファイルに適当なコメントを書いておきます
- debian/patches/00list を適当に修正します
- debian/rules unpatch && debian/rules patch && debian/rules unpatch として 一応パッチが動作することを確認します

11.4.4 すでにあるパッチを編集

- dpatch-edit-patch パッチ番号 _ パッチ名とする．例：dpatch-edit-patch 03_automake
- 編集する
- exit

11.5 今後の開発

がんばれ．といいたいところですが，とりあえず現状の仕様をあらいだして，テストを作成して，全部の機能が動作することを確認するところからやろうと考えています．今は謎の機能が多すぎておいそれと触れません.

12　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．

12.1　2005 年 2 号

問題 1. KDE 3.3 が testing にはいるために必要だったのはなにか

□ A 地の理

□ B Britney の手動操作

□ C 天候

問題 2. Gcc に含まれている Java VM はどれか

□ A Kaffe

□ B Gij

□ C JamVM

問題 3. バイナリファームウェアを別途ディスクなどからロードするように実装しなおしたカーネルのデバイスドライバについて問題となっているのは何か

□ A フリーではないものを必要とするものは contrib にはいるべきだが，カーネルは contrib にはいるべきなのか

□ B カーネルにディスクアクセスをさせたくない

□ C ディスクからファームウェアを読み込むようにする機構を作成するのは不可能

問題 4. debhelper を使わないでパッケージを作る方法についてのメールが流れたのはどのメーリングリストか．

□ A debian-mentors

□ B debian-devel

□ C debian-women

問題 5. Joey Hess がこの Changelog のエントリはひどいだろう，と指摘したのは

□ A 下らないジョーク
□ B 女性蔑視な文面
□ C 学生が課題で提出した内容について「あそこの学生は最近はダメだな」と言及したもの

12.2 2005 年 3 号

問題 6. 中国での Debian miniconf はいったいどこで行われるか
□ A Beijing
□ B Pyongyang
□ C Seoul

問題 7. 2004 年に Porto Alegre で実施した debconf の参加者は
□ A 53
□ B 163
□ C 1000

問題 8. LCA Mini-Debconf はどこで実施される？
□ A Australia
□ B Akasaka
□ C Austria

問題 9. 起動時にデーモンを起動しなくする設定をアップグレードした後も変更されないようにする方法として間違っているのは？
□ A update-rc.d -f service remove でシンボリックリンクをすべて削除する
□ B /etc/init.d/service スクリプトの最初に exit 0 を挿入する
□ C invoke-rc.d の policy-rc.d をいじる

問題 10. graphviz は関連図などを作成するのにしばしば使われているソフトウェアですが，最近何が起きた?
□ A やっと DFSG-free になった
□ B 革新的にユーザにとって使いやすくなった
□ C vcg にのっとられた

問題 11. debian/copyright ファイルについて，ライセンスを正確に記述しよう，全ての著作権者の一覧を作成しようというメールを書いたのは
□ A Branden Robinson
□ B Jochen Voss
□ C Henning Makholm

12.3 2005 年 4 号

問題 12. volatile woody に追加された最初のパッケージは？

□ A amavis

□ B spamassassin

□ C whois

問題 13. DevFS に大きく依存して実装している debian-installer の開発チームに大きな衝撃を与えたのが, linux カーネルからの DevFS の削除. それが予定されている時期は

□ A 2005 年 7 月

□ B 2005 年 2 月

□ C 2099 年 9 月 9 日

問題 14. 1 月 16 日の Debian Women IRC ミーティングで決定した事項は

□ A 存在感をあたえつつ debian 全体をおどしているようには見えないように頑張る

□ B Debian Project へのねずみ講形式の女性勧誘

□ C Debian Project からの男性の排除

問題 15. root 以外のユーザが直接/sbin/halt を実行するために必要な手順は

□ A ln -sf /bin/true /sbin/halt

□ B dpkg-statoverride –add root root 02755 /sbin/halt

□ C chmod 02755 /sbin/halt

問題 16. Mozilla Foundation とトレードマークについて同意した内容はどの観点で不足なのか

□ A 金銭的に問題が発生している

□ B DFSG の License Must Not Be Specific To Debian に該当する

□ C 宗教的に受け付けられない条件になっている

12.4 2005 年 5 号

問題 17. MySQL パッケージの変更によって，大きな問題が予想されるものは何か

□ A mysql のデータベース自体の互換性が無い

□ B mysql のライブラリの ABI と soname が変更したために発生する大がかりな再コンパイル作業とそこで現れる問題

□ C コンパイラーのバグ

問題 18. sarge のリリースノートは woody からのアップグレードをどのようにすることを推奨しているか.

□ A aptitude を使って実施すること
□ B アップグレードを開始するまえに祈りをささげること
□ C アップグレードできなくてもめげないこと

問題 19. Debian の Archive が正当であることをある程度証明できる経路を提供するために作られている，Archive の gpg キー．最近何がおきたか
□ A Key が expire したので作り直した
□ B みんなが活用しはじめた
□ C apt-get がサポートはじめた

問題 20. 問題がなかなか解決しないため，リリースの障害となっているので，一時的に sarge のリリース対象から除外されたアーキテクチャはなにか
□ A m68k
□ B sh
□ C mipsel

12.5 2005 年 6 号

問題 21. 前 Debian Project Leader の娘で，今度 Tuxracer について発表する講演者の名前は Elizabeth Garbee だが，何歳から debian をつかっていたか.
□ A 9
□ B 10
□ C 23

問題 22. FTP を利用しないで Debian パッケージをアップロードする方法は何があるか
□ A FTP サーバのセキュリティーホールをさがし，そこからシェルアカウントを取得して cat
□ B gluck の DELAYED キューに ssh でアップロード
□ C alioth 経由でアップロード

問題 23. カーネル 2.6.8 をデフォルトにし，2.6.10 をデフォルトのカーネルに設定できないと判断した理由
□ A 2.6.10 が sparc で動作しない
□ B 2.6.10 になると IA64 のシリアルポートの扱いが変更になる
□ C d-i は現在 2.6.8 で動作しているから

問題 24. パッケージがたくさんあって問題だと言われている NEW キューとは
□ A OVERRIDE の編集が必要な変更がされたパッケージに対して，FTP-master が手動で操作する必要があるが，その未処理のバックログのこと
□ B 新しいパッケージの一覧のこと
□ C リリースに関係ないパッケージの一覧のこと

問題 25. chown で指定するのに利用するユーザ名とグループ名の区切り文字は
□ A ：
□ B .
□ C /

12.6 2005 年 7 号

問題 26. Joerg Jaspert が DAM として発表したのは
□ A Emeritus 状態のデベロッパーに対しては NM 処理に近いものを DAM が直接実施します
□ B Emeritus 状態のデベロッパーはいつでももどりたいときに何の処理も必要なく Debian Developer にもどれます
□ C GPG キーは Advocate さえ署名していればよいです

問題 27. udev を利用している場合 /.dev ディレクトリを削除してもよいのか
□ A 本物の/dev がそこにマウントされているため，システムが起動しなくなる
□ B /.dev なんか必要ないので，消してしまってよいです
□ C むしろ/proc をアンマウントしてください

問題 28. cvs ではなく svk を利用して/etc をメンテナンスする利点でないものはなにか
□ A シンボリックリンクを扱えるバージョンシステムであること
□ B CVS/等の特殊な管理用のディレクトリが生成されない
□ C 将来おきるだろう設定変更が予測できる

12.7 2005 年 8 号

問題 29. debconf を使っているパッケージのうち，po-debconf を使っていない数は
□ A 102
□ B 1020
□ C 10200

問題 30. FTP-master に対して不要になった debian パッケージを削除してほしいと依頼する方法は
□ A IRC の ftpmaster チャンネルで叫ぶ
□ B ftp.debian.org 仮想パッケージに対して BTS でバグを登録する
□ C debian-devel にメールする

問題 31. 実行ファイルとデータファイルを別パッケージにしている場合などで依存関係を厳密に=の関係で指定すると発生する問題は

□ A APT を使用した場合に CPU 負荷が高くなる
□ B FTP サーバに負荷が大きくなる
□ C buildd でバイナリがビルドできるまでバイナリの無いアーキテクチャでインストールできない

問題 32. apt 0.6 の機構で実現するのは
□ A ABI の変更は実施しないため，APT ライブラリに依存するパッケージのスムーズな移行
□ B 暗号学的な認証でインストールするパッケージの出自を確認できるようになる
□ C 妄想したパッケージが自動で生成できるようになる

問題 33. dh-devincludes は何を実現するのか
□ A dev パッケージの依存関係情報をビルド時の情報から自動生成する
□ B パッケージをビルドする際にデビンクーズという人のロゴが現れる
□ C デバイスファイルをパッケージに追加する際の処理をする

問題 34. アプリケーションの移植の際に，arm アーキテクチャの場合のみに問題となるものは何か
□ A int の バイトオーダー
□ B char が signed であるか unsigned であるか
□ C float の形式

問題 35. PHP4 のライセンスで問題であると指摘されている部分は何か
□ A ソースコードの改変が許可されていない
□ B 派生物は PHP と呼ぶことができない
□ C コーヒーをつくれない

12.8 2005 年 9 号

問題 36. クロアチアの Rudjer Boskovic Institute が発表したものは
□ A Debian Cluster Components という HPC クラスタ管理用のツール群
□ B クロアチア名産がすぐにわかる土産集パッケージ
□ C Debian 使ってません

問題 37. Hurd/L4 で最近できたと Marcus Brinkmann が発表したのは
□ A 初めて Banner アプリケーションが実行できた
□ B 初めてカーネルが起動した
□ C 初めて bash が動いた

問題 38. Goswin 曰く Debian の AMD64 移植版は

☐ A sarge で gnome と KDE が入るようになった
☐ B debian-installer はまだ動作していない
☐ C 初めて bash が動いた

12.9　2005 年 10 号

問題 39. 2005 年の DPL 選挙に立候補していないのは
☐ A Anthony Towns
☐ B Bdale Garbee
☐ C Branden Robinson

問題 40. Release Team のミーティングが開催されるのは
☐ A Chicago
☐ B Vancouver
☐ C Beijing

問題 41. Project Scud の目的は
☐ A Debian に敵対する組織の殲滅
☐ B DPL に敵対する陰謀の阻止
☐ C DPL の補佐

問題 42. buildd で発生した難しい問題を解決する方法について Steve Langasek の提案は
☐ A 玄関のとびらを黄色に変えてから，再度パッケージを投入する
☐ B 該当アーキテクチャの buildd の管理者に相談してみる
☐ C そのアーキテクチャを捨てよう，と debien-devel メーリングリストで提案する

問題 43. マニュアルページの中での横棒の表記は
☐ A バックスラッシュの後に - を記述する
☐ B -だけを記述する
☐ C .hf と記述する

12.10　2005 年 11 号

問題 44. Debconf5 に投稿されたプロポーザルの数は
☐ A 22
☐ B 33
☐ C 44

問題 45. Debian logo のライセンスの変更するためになにがおきたか
□ A 原作者から SPI に著作権の移管
□ B 著作権法の改正
□ C ロゴの作り直し

問題 46. ドキュメントや辞書などのデータに GPL ライセンスを適用することにより問題となりうるのはなにか
□ A ライセンス文が大きすぎることによるデータ量の増大
□ B GPL でないアプリケーションから利用する場合の検討
□ C 利用するまえに Free software song を歌う必要がでてくる

問題 47. USB ストレージデバイスで GPG キーを利用する際に，VFAT 上でループバックデバイスを利用しているのはなぜか.
□ A 暗号化したいから
□ B ループバックでしか VFAT はマウントできない
□ C GPG がループバックデバイスしか対応していない

問題 48. Debian の wanna-build サーバの SSH サーバに関して変更が発生したのはなぜか
□ A コネクションキャッシングしないと負荷が高くて死んでしまうから
□ B SSH をビルドすることがかっこよいから
□ C make コマンドをたたく練習がしたかったから

問題 49. Joey Hess が NEW キューに関して提案した内容はなにか
□ A NEW キューを無くしてしまおう
□ B NEW のパッケージに関して人気投票をしてみよう
□ C 新しいパッケージは作成しないことにしよう

問題 50. DPL debate はどこで開催されたか
□ A Vancouver
□ B メッセンジャー
□ C IRC の Freenode ネットワーク上

問題 51. Steve Langasek の出した SCC の提案とはなにか
□ A 多くのアーキテクチャをリリースプロセスから除外する
□ B 世界に残存する m68k を廃棄する
□ C etch のリリースアーキテクチャは２０種類を目指す

問題 52. Asia Debian Miniconf で達成したのはなにか

□ A debian-zh IRC チャンネルの復活
□ B 全員がお腹をこわした
□ C 中国から台湾へのネットワークでの接続

12.11 2005 年 12 号

問題 53. GPLv3 が出て来ることでもっともおそれられているのは
□ A GPLv2 のみに対応するというソースコードが増えて，プロジェクトのフォークが増えること
□ B ライセンス文が長文になりすぎてオープンソースプロジェクトのファイルサイズが大きくなりすぎる事
□ C 誰も読めない言語で書かれていること

問題 54. Creative Commons 2.0 のラインセンスについて debian-legal が出した判定結果は
□ A DFSG Free です
□ B Derived Work について制約が多いので，DFSG Free ではない
□ C GPL と同じです

問題 55. Enrico Zini はどんなアンケートを実施したか
□ A 一日の生活パターンについて
□ B 人生の目標について
□ C Debian の利用目的と方法について

問題 56. 300000 番目のバグレポートはどういう内容か
□ A SPAM
□ B セキュリティーホール
□ C Description の文法的な修正

問題 57. DPL Vote システムに暗号化したメールで投票したらどう処理されるか
□ A gpg で復号化し，devotee システムが処理する
□ B /dev/random から投票結果を生成する
□ C 暗号には現在対応していない

問題 58. autoconf を build 時に呼ぶ理由でないものはなにか
□ A そこに autoconf があるから
□ B ソースパッケージのサイズを削減できる
□ C autoconf が変更したあとで実行してもパッケージがビルドできないという状況をさけられる

12.12　2005 年 13 号

問題 59. 3 月 18 日にあたらしく増えた FTP-master メンバーは誰か

□ A Jeroen van Wolffelaar と Joerg Jaspert

□ B　James Troup と Ryan Murray

□ C　Anthony Towns と Branden Robinson

問題 60. 新しくできたメーリングリスト debian-dak はなににについて語る場所か

□ A Katie さんについて語る場所

□ B Debian のパッケージインフラソフトウェアについて語る場所

□ C Anthony Towns の好きな歌手について語る場所

問題 61. libtool1.4 に build-depend しているパッケージにたいしてなにがおきるか

□ A あたらしい libtool を使うように推奨

□ B 即刻パッケージを削除

□ C libtool を使わないように修正

問題 62. ITP されているものの放置されているパッケージに関して推奨された処理は

□ A メールアドレスを調査して SPAM 送付

□ B バグレポートにメールを追記して，作業を開始する

□ C BTS サーバに新入して，ITP バグレポートをこっそり削除

問題 63. Description の文書でよく間違っていると Florian Zumbiehl が指摘したのは

□ A 英語になっていない

□ B 略語の前の'a' が'an' であるべきの場所がある

□ C ギリシャ文字で書いてある

問題 64. PHP 4 のライセンスで問題となっているのは，なにか

□ A GPL が嫌いだ，と書いてある

□ B 問題はない

□ C　変更したものは，PHP4 という名前を利用できない

問題 65. Marcus Brinkmann が Hurd/Mach をより Hurd/L4 がよいという理由は

□ A l4 のほうが Linux のエミュレーションが優れている

□ B Mach の VM 管理がださい

□ C l4 のほうがデバイスドライバが書きやすい

12.13　2005 年 14 号

問題 66. GNU Hurd のライブ CD を qemu で起動するコマンドラインはどれか

□ A qemu -cdrom hurd-live-cd.iso -boot d

□ B ./gnu-hurd

□ C dd if=hurd-live-cd.iso of=/dev/hdc && reboot

問題 67. SCC 提案に対して，John Goerzen の提案した内容はなにか

□ A 一部のアーキテクチャはバイナリを配付せずにソースのみを配付する

□ B gentoo に全員移動

□ C 全員 i386 以外のアーキテクチャは使わない事にする

問題 68. 遅いアーキテクチャの対応として提案されたツールの案で，エミュレータなのはどれか

□ A qemu

□ B scratchbox

□ C distcc

問題 69. chroot 内でデーモンの起動をしてほしくないときにはなにを使えばよいか

□ A invoke-rc.d のための policy-rc.d スクリプトを作成する

□ B init.d 以下のスクリプトを全部即 exit0 で終了するものに変更する

□ C 起動しないように祈りの踊りをささげる

問題 70. 起動スクリプトの出力をログにとるためにはどうするか

□ A 画面をみながら出力のメモをとる

□ B dmesg コマンドの出力にでるのでそれを使う

□ C echo 'BOOTLOGD_ENABLE=Yes' > /etc/default/bootlogd

問題 71. testing からパッケージが削除された場合の情報はどこに流れるか

□ A apt-cache show パッケージ名

□ B 秘密なので教えられない

□ C 今後は debian-testing-changes に流れる予定

問題 72. gluck.debian.org になにがおきたか

□ A ネットワーク障害による通信の断絶

□ B ディスク障害によるファイルシステムの崩壊

□ C 人的災害によるシステムの破壊

12.14　2005 年 15 号

4 月 12 日時点の情報です.

問題 73. Debian Project Leader 選挙の結果 DPL になったのは

□ A Anthony Towns

□ B Branden Robinson

□ C Michael Jackson

問題 74. Evan Prodomou が Creative Commons のライセンスに関して任じられたのは

□ A Debian 側の Creative Commons の Committee として動くこと

□ B 文書の翻訳

□ C 文書の精査

問題 75. tigon II チップのファームウェアが問題なのであれば，仕様を見て，フリーで実装しなおせばよいのではないか，といったのは

□ A Branden Robinson

□ B Peter De Schrijver

□ C 通りがかりの人

問題 76. パッケージの自動テストについて Petter Reinholdtsen がプロトタイプを作成したというのは

□ A アップグレードの自動テストのスクリプト

□ B リリースクリティカルバグの原因となった人に自動で罰ゲームを選択してくれるシステム

□ C パッケージの使いやすさに関して定量的に計測して，使いにくいパッケージは自動で警告を出してリジェクトできるようなシステム.

問題 77. selinux について Manoj Srivastava が開始したのは

□ A etch での selinux の導入に向けてのプロジェクト

□ B selinux はなかったことにするための隠蔽プロジェクト

□ C selinux に対抗する実装を開始

12.15　2005 年 16 号

問題 78. ミュンヘン市がデスクトップ PC の OS として選択した OS は

□ A Windows

□ B Debian

□ C Solaris

問題 79. Debian 3.0 のアップデートが出たが，それはどのバージョンか

□ A 3.0r5
□ B 3.0rX
□ C 3.0TNG

問題 80. Adrian Bunk が GPL がフリーであるか，ということについて議論した内容は何か
□ A ライセンス文章については変更できないので，フリーではないのではないか
□ B GPL のイデオロギーが気に入らない
□ C RMS が挙動不審だ

問題 81. カーネルチームの IRC ミーティングで決定した内容はなにか
□ A etch 以降は Hurd カーネルを採用する
□ B etch は FreeBSD カーネルと Linux カーネルのソースツリーをマージする方向で検討する
□ C testing に現在はいっているカーネルで基本はフリーズする．

問題 82. ライセンスの文書で GPL に追加で制限を加える場合にどうなるか，という議論でどういう問題点が指摘されたか
□ A Debian は作者の意向を尊重するため，GPL ライセンスで配布されているプログラムに追加で制限を加える場合には，Debian として配布できない可能性がある
□ B ライセンスはいくらでも変更してよいので問題無い．
□ C 作者は神様です

12.16 2005 年 17 号

問題 83. GNOME2.10 はどうなったか
□ A unstable にアップロードされた
□ B experimental にアップロードされた．
□ C sarge に含まれる予定

問題 84. 遅々として Debian に入って来ない mplayer だが，MJRay は mplayer について FAQ を作成した．それによると
□ A ftpmaster を人質にとったので，もうすぐアーカイブにインストールされるだろう．
□ B ftpmaster に賄賂をおくったので，もうすぐアーカイブにインストールされるだろう．
□ C 問題となっているコードは削除しており，アップロードは ftpmaster の承認待ち．

問題 85. snapshot.debian.net に関して提案されなかったことは
□ A debian.org に昇格しよう
□ B バックアップをとりましょう
□ C サーバに DDoS をかけましょう

12.17　2005 年 18 号

問題 86. Leadership meeting にてなされた金銭的な議論は
□ A aKademy への出席に関する参加補助
□ B Leader への給与について
□ C Debian Developer への報酬について

問題 87. PHP アプリケーションにおいてよくある問題で，Martin Schulze がセキュリティーの観点からもっとも問題なので注意してほしいと指摘したのは
□ A 設定ファイルが http で取得できる場所に存在する設計になっている場合
□ B PHP という言語処理系をつかっていること自体
□ C PHP の文法でプログラムを作成しようとする発想

問題 88. Andreas Barth によるとリリースの状況は
□ A ARM の buildd も追加されたので testing-security の準備がほぼ完了した.
□ B 地下組織の暗躍により進展が阻害されている
□ C リリースはそろそろあきらめようかと考えている.

問題 89. Debian Conference で今年は新しいイベントとして何をする予定か
□ A 全員参加のジャンケンゲーム
□ B 一般参加者をつのり，その参加者のためのイベントを最初の日に開催する
□ C Branden Robinson は誰だゲーム

問題 90. Jorgen Schäfer の提案したのは
□ A scheme パッケージのポリシを作成し update-alternatives で管理する
□ B scheme は廃棄して common lisp に移行する
□ C ruby で書かれているスクリプトを全て scheme で実装しなおす

12.18　2005 年 19 号

問題 91. sarge のバージョン番号はどれか
□ A 3.2
□ B 3.1
□ C 4

問題 92. sarge のフリーズが開始して，新しいパッケージのバージョンはリリースマネージャの承認がないと追加できないようになりました．さて，フリーズの開始したのは？

□ A 2005 年 5 月 3 日
□ B 2005 年 5 月 30 日
□ C 2005 年 4 月 1 日

問題 93. amd64 ポートは alioth から移行した．その移行先のサーバは？
□ A ftp.debian.org
□ B amd64.debian.net
□ C amd64.org

問題 94. apt の新しいバージョンを experimental にアップロードする際の問題は何か
□ A libapt-pkg に依存するパッケージを NMU で experimental 向けにアップロードしても新しいバージョンが unstable にアップロードされるたびに experimental にアップロードしたバージョンが削除されてしまう．
□ B apt の最新版は不安定すぎる
□ C ubuntu との利権の衝突

12.19　2005 年 20 号

問題 95. adduser のオプションの話しに端を発して disabled login と disabled password に関して，ssh のパスワードの認証の話しで何が問題となっていたか
□ A UsePam を使った場合のパスワードの扱い
□ B パスワードをつかってログインができない
□ C パスワードを入力しなくてもログインができる．

問題 96. GPL と FDL のドキュメントを一つとして混ぜることはできるか
□ A 両方とも RMS が作成したライセンスなので問題ない
□ B そういうことは気にしなくてよい
□ C 互換性がないので無理だろう，と Anthony DeRobertis は説明した

問題 97. alioth のサーバはどうなるか
□ A amd64 が別のサーバに移動してディスクスペースの問題が解消したので，いままで複数のサーバで構成していたのを一つのサーバに移行する．
□ B 使い勝手が悪いので停止
□ C アカウントが増えすぎなので募集を停止する

問題 98. Lars Wirzenius はパッケージのテストが必要だ，と説明した．その主張した内容で，現状足りないものは何だといっているか
□ A インストールと削除が無事に動作することを確認するには，新しいツールが必要だ．
□ B 暇な人材が必要
□ C 開発者がもっと必要

12.20　2005 年 21 号

問題 99. 12W の電力消費で動くコンパクトな Debian サーバを準備するために，Silas Benett はどうしたか

□ A となりの家の犬を無線 LAN ステーションにした
□ B Mac Mini をかってきて，CDROM ドライブをとりはずしてバッテリーを装着した
□ C Dual Xeon のサーバを購入して Xeon を一つにへらしてみた

問題 100. Michael Banck は Hurd で何をしたか

□ A GNOME と QT を動かした
□ B もうあきらめようと宣言した
□ C tuxracer を動かした

問題 101. Orphan されたパッケージなどの一覧が出て来る WNPP メールは今後どうなるか

□ A 毎週 debian-devel-announce にメールを投げていたのを停止して，debian-wnpp に移動する
□ B もう orphan なんてものは存在しない
□ C orphan したパッケージは今後は無視する

問題 102. Nico Goldeh は unrar パッケージのバージョンがずいぶんさがっていることに気づきました．その理由は

□ A もともとの unrar は non-free で，free な実装ができたのでそちらに移行した
□ B unrar の開発は後向きにすすんでいる
□ C unrar なんて知らないので関係ない

問題 103. Waste パッケージにはどういうライセンス上の問題があるか

□ A ソフトウェアとしてつかいものにならない
□ B 開発元が一旦 GPL でリリースしたが，revoke した
□ C ライセンスが入っていない

問題 104. Debian woody のアップデートが出たが，そのバージョンは

□ A 3.0r6
□ B 3.0.6
□ C 3.0-RC6

12.21　2005 年 22 号

問題 105. Andreas Barth が BTS の LDAP ゲートウェイの高速化のために試したのは
□ A Archived バグが多すぎるので，Archived バグを分離してしまいメモリ負荷を減らす
□ B LDAP をやめて SQL にする
□ C 不要っぽいバグはなかったことにする

問題 106. Philipp Kern は Debian Archive に video セクションを追加しようと提案しました．video セクションが無いため現状は各ビデオ関連のパッケージは
□ A どこのセクションにも属していない
□ B libs セクションに投げ込まれている
□ C 該当するアプリケーションは graphics セクションとデスクトップ環境のセクションに混在している

問題 107. debian-legal のサマリーページについて Frank Lichtenheld はもう削除しようと提案しました．その理由は
□ A 使えないから
□ B 面白くないから
□ C 合意がとれないのであたらしくサマリーが作成できずメンテナンスが不可能

12.22　2005 年 23 号

問題 108. Debian のリリースをおくらせる必要がある，と主張していた KDE のバグは何？
□ A webcollage スクリーンセーバがデフォルトで動作するようになっているので，オンラインのポルノ画像が表示される可能性がある
□ B KDE が動かない
□ C KDE の壁紙が Debian 向けではない

問題 109. Debian GNU/Linux 3.1 がリリースされたのは？
□ A 2005 年 6 月 6 日 (日本時間 6 月 7 日)
□ B　2005 年 7 月 1 日 (日本時間 7 月 2 日
□ C 2005 年 1 月 1 日 (日本時間 1 月 2 日)

問題 110. 3.1r0 の CD セットの問題は
□ A だれも焼けないような枚数になった
□ B デフォルトインストールの sources.list の security.debian.org 向けの行が有効でなかった
□ C 大きすぎて世界中のネットワーク帯域をくいつぶしてしまう

問題 111. Wesley Landaker は GPG キーサインのための確認に実際に会わなくてもできる基準を提案しました．その反論は
□ A GPGって出会い系サイトでしょ？
□ B 写真のみで確認するのであれば，写真は偽造しやすいので，その情報を信頼するのは難しい
□ C 会わなくてよいのだったら GPG は面白くない

問題 112. ライブラリの ABI が変更になった場合に気を付けることは何か
□ A ライブラリパッケージ名を変更しておけば管理者が明示的にアンインストールするまでは古いライブラリバージョンもシステムに残る
□ B 古いライブラリにリンクしてあるプログラムを確実に全部動かなくする
□ C 変更履歴に怒りのコメントを記述する

12.23　2005 年 24 号

6 月 14 日版です.

問題 113. Alioth が新しいサーバに移動するということが各ユーザ宛のダイレクトメールで出た．Branden Robinson が Alioth のアナウンスメールについてコメントを出した．その主旨は.
□ A 匿名で各ユーザにメールを出すのではなく，debian-devel-announce に出そう，さらに誰が出したのかを明確にして出そう
□ B メールは信頼できないプロトコルなので WEB ページに掲示したらよいだろう
□ C SMTP より，SNMP トラップのほうがみんな気づく

問題 114. Andreas Barth は etch に向けてリリースポリシーについてどういう変更が必要だと説明したか
□ A ダメな人禁止
□ B ダメなコード禁止
□ C 共有ライブラリはダイナミックにリンクすべきであり，ソースコードをコピーするのは禁止しよう.

問題 115. etch にむけて c++ の ABI が変更になるため，Matthias Klose は何を宣言したか
□ A C++ でかいたプログラムの粛正
□ B とりあえず今 C++ のライブラリであたらしい ABI バージョンのものについてはアップロードしないようにフリーズする
□ C java の利用禁止

問題 116. Debconf での講演の予定の組み方について今回はどういうこころみがなされたか

□ A 意志決定方法としてさいころを導入してみた
□ B 参加者の都合が悪い日をできるだけ選んだ
□ C 参加したい講演の投票をおこない，参加者ができるだけ参加できるようにスケジューリングをする

問題 117. Scott James Remnant 曰く，dpkg 1.13 で新しく追加されたアーキテクチャ変数名は
□ A DPKG_HOST_ARCH_OS
□ B DEBIAN-ARCHITECTURE
□ C uname -a

問題 118. Roberto Sanchez の作成した FAQ には何がかいてあるか
□ A Debian パッケージをインストールする方法がかいてある
□ B Debian パッケージをカスタマイズしてビルドする手順がかいてある
□ C Debian メンテナを招集する方法がかいてある

問題 119. selinux の進捗についてはどうなっているか
□ A coreutils などのパッケージが libselinux1 に依存するようになってきた
□ B NSA の陰謀であることが判明したので排除
□ C もうほとんどのユーザが SELINUX を使うようになった

12.24 2005 年 25 号

6 月 21 日版です.

問題 120. Oskuro が GNOME について宣言したのは何か
□ A 今後は KDE に移行する
□ B Gnome 2.10 が unstable に入った
□ C GNOME3 がもうすぐリリース

問題 121. woody から sarge にアップグレードするのにもっとも大きい問題は何か
□ A 相互に依存するような依存関係があるパッケージのアップグレード
□ B woody では存在したけど sarge で消えたパッケージの処理
□ C etch に無いパッケージ

問題 122. Aurelien Jarno が BSD 移植版について報告した問題は
□ A GNU Hurd についてはもうあきらめたので考えなくて良い
□ B FreeBSD についてはもうあきらめた.
□ C libselinux1 に依存するパッケージは，selinux が linux のみなので，Linux のみでリンクするようにしてほしい

問題 123. Bill Allombert が menu について報告したのは
□ A menu のセクションの部分については国際化できるようになった
□ B menu はもう使われていないので廃止する
□ C 対応する Window Manager が増えた

問題 124. 矢吹さん曰く大阪でなにかあるらしい，何？
□ A 10 月に mini Debian Conference を開催する
□ B 9 月にたこやきはやぐい競争
□ C 阪神優勝祈念会

12.25　2005 年 26 号

6 月 28 日版です.
問題 125. Debian Policy Committee を Branden Robinson が発表しました．チェアマンは誰？
□ A Manoj Srivastava
□ B Andreas Barth
□ C Matt Zimmerman

問題 126. Andreas Barth は etch 向けのリリースポリシーを発表しました．そのなかにあった記述は
□ A debian/changelog や debian/control ファイルを通常のビルド処理で変更することを禁止する
□ B 二年以内にはリリースしない
□ C パッケージの数は 10 万を越えることを目標にする

問題 127. XML 形式のインタプリタ言語 CYBOP を Debian に含める場合，プログラムはどこにおいたらよい？
□ A ユーザが実行できるなら/usr/bin 以下，ライブラリモジュールなどなら/usr/share/cybop のような場所を指定する
□ B /srv
□ C XML ベースの言語なんてとんでもない

問題 128. cogito というプログラムと git というプログラムが，/usr/bin/git を両方保持していて，conflict: していたことに関しての反論は
□ A /usr/bin/git の機能が全く違う
□ B /usr/bin/git は GNU の商標です
□ C cogito の/usr/bin/git が一番よく使われているから GIT の git なんか不要

問題 129. openafs パッケージは全アーキテクチャではビルドできない．しかし，Architecture: all のパッケージがあるために，各アーキテクチャでビルドが試行されてしまう．そのような状況を表現する最良の方法

は何？
□ A Build-Depends: type-handling でアーキテクチャを指定
□ B Packages-arch-specific を使う
□ C buildd のメンテナに個人的にメールを送る

問題 130. tetex がすでにあるのに TeXlive をパッケージする理由としては
□ A たくさんの TeX パッケージがあり，年に一回リリースしている
□ B tetex は使えない
□ C そこに TeXlive があるから