



関西 Debian 勉強会担当者 山下 尊也

2008 年 10 月 19 日

1 Introduction

山下 尊也

関西 Debian 勉強会は Debian GNU/Linux のさまざまなトピック (新しいパッケージ、Debian 特有の機能の仕組、Debian 界限で起こった出来事、などなど) について話し合う会です。

目的として次の三つを考えています。

- ML や掲示板ではなく、直接顔を合わせる事での情報交換の促進
- 定期的に集まれる場所
- 資料の作成

それでは、楽しい一時をお楽しみ下さい。

会強勉コンドビト関係

目次

1	Introduction	1
2	最近の Debian 関係のイベント報告	3
3	はじめての CDBS	4
4	cdn.debian.or.jp, cdn.debian.net における取り組み	11
5	今後の予定	20
6	メモ	21

2 最近の Debian 関係のイベント報告

山下 尊也



2.1 第 17 回 関西 Debian 勉強会

2.1.1 発表内容

- Debian Live に Ubiquity を移植できるか？
- Debian Mentors の紹介
- 10 分でわかる Debian フリーソフトウェアガイドライン (DFSG)

「Debian Live に Ubiquity を移植できるか？」は、Debian では、廃止されたパッケージの利用などの問題があり、簡単には出来ないと言う事になりましたが、Ubiquity のようなものが live-helper にはないため、今後も、live-helper の動きなどに注目したいと思います。

「Debian Mentors の紹介」については、初心者がパッケージなどを作成する際に本家とは違い、初心者がある程度許してもらえる空間として紹介して頂きました。これにより、私をはじめ多くの方が、最初の一步を歩む事が出来たと思います。

「10 分でわかる Debian フリーソフトウェアガイドライン (DFSG)」では、木下さんが携帯電話を用いて、DFSG について紹介して下さいました。DFSG の内容にライセンスが関係するため、ライセンスについて熱い議論がかわされました。

その後、急遽、佐々木さんに講師をして頂きました。

懇親会は、鳥料理が中心の南風で行いました。懇親会の際に、なぜ東京エリア Debian 勉強会と関西 Debian 勉強会では、なぜ \LaTeX を利用しているのか？また、 \LaTeX 以外のツールではダメなのか？などの議論がありました。東京エリア Debian 勉強会は、11 月に \LaTeX 合宿を行うようです。関西では、12 月に資料作成や、今年度のまとめなどを行いたいと思っているので、そこで \LaTeX でのハンズオンをやりたいと考えています。

他にこんなツールでも資料作成出来るよ !! というのがあれば、そのツールについても検討したいので、ぜひ 12 月の際に紹介して下さい。

3 はじめての CDBS

佐々木 洋平



3.1 はじめに

CDBS – Common Debian Build System は、debian/rules の共通部分を抽出し共有することで、debian/rules を簡潔かつ理解しやすくすることを目的としたツールです。[CDBS Documentation Rev.0.1.2] の Introduction によれば、当初の開発動機は GNU autoconf & GNU automake を使用している Debian パッケージの重複した debian/rules をなんとかしたい、だった様です:

The motivating factor for CDBS was originally that more and more programs today are created using GNU Autoconf configure scripts and GNU Automake, and as such they are all very similar to configure and build. It was realized that a lot of duplicated code in everyone's debian/rules could be factored out.
...

今では CDBS はもっと汎用的になっており、autotools に限らず、Gnome, KDE, Python, Haskell など、様々なパッケージの作成に使用できるようになっています。[Online CDBS Gallery] によると、現時点で CDBS を使用しているソースパッケージは 1805 個、ソースパッケージ全体の ~ 16% だそうです*1。

ドキュメントで上げられている CDBS の利点は以下の通りです*2:

1. 簡潔で、可読性が高く、効率的な debin/ruls を作成できる。
2. debhelper と autotools の呼び出しを自動化することによって、繰り返し作業を気にする必要がなくなる。
3. カスタマイズ性に限界が無いので、メンテナはパッケージ作成時のより重要な問題に集中できる。
4. 提供されるクラスは十分にテストされており、よくある問題を回避するための汚い hack をする必要がない。
5. 既存のパッケージを CDBS へ移行するのは容易である。
6. 拡張性が高い。

確かにそうなんですけれども、以下の例では逆に何をしているのかわからなくて不安になったりします:

```
#!/usr/bin/make -f
include /usr/share/cdb/1/rules/debhelper.mk
include /usr/share/cdb/1/class/autotools.mk
```

ここでは [CDBS Documentation Rev. 0.4.0] の冒頭をざっくり解説してみようと思います。

*1 これ、正しいですかね?

*2 [CDBS Documentation Rev. 0.4.0] の CDBS advantages の超訳です - -;)

3.2 CDBS のディレクトリ構成& ファイル群

CDBS の実態は細分化された Makefile 群です。実際に /usr/share/cdb/ 以下を眺めてみましょう:

```
% ls -aR /usr/share/cdb/
/usr/share/cdb/:
./ ../ 1/

/usr/share/cdb/1:
./ ../ class/ rules/

/usr/share/cdb/1/class:
./          autotools-vars.mk  hbuild.mk          perlmodule-vars.mk
../         autotools.mk        kde.mk             perlmodule.mk
ant-vars.mk cmake.mk           langcore.mk        python-distutils.mk
ant.mk      docbookxml.mk      makefile-vars.mk   qmake.mk
autotools-files.mk  gnome.mk          makefile.mk

/usr/share/cdb/1/rules:
./  buildcore.mk  debhelper.mk  patchsys-quilt.mk  tarball.mk
../  buildvars.mk  dpatch.mk    simple-patchsys.mk  utils.mk
```

中間ディレクトリ 1 は将来の API 変更を考慮したディレクトリです。rules, class ディレクトリ以下にあるファイルが細分化された Makefile です。パッケージを作成するにはこれらのファイルを適宜 debian/rules 内で include して使用します。個々の Makefile の依存関係は以下の通りです:

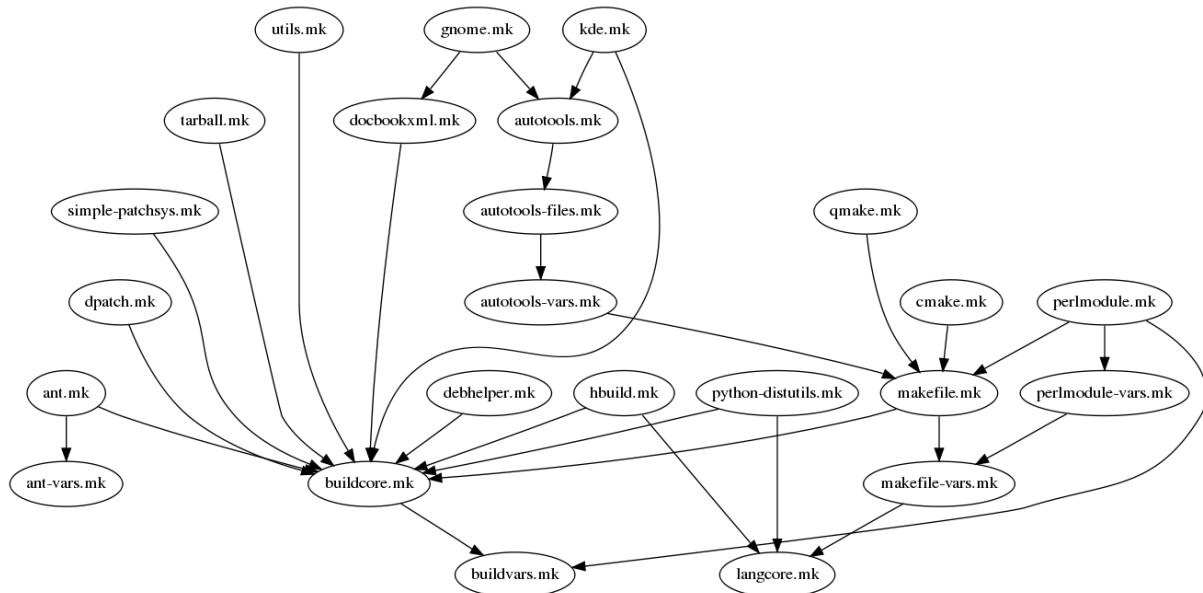


図 3.1 CDBS で提供される Makefile の依存関係 ([CDBS Documentation Rev.0.1.2]).

3.3 基本となる rules

3.3.1 buildvars.mk

buildvars.mk を include するとパッケージ作成のための環境変数が設定されます。設定される変数の一覧を表 3.1 に記載します。良く使われるのは CURDIR と DEB_DESTDIR でしょう。これらの変数を変更したい場合には、buildvars.mk を include した後で debian/rules 内部で以下の様に設定します:

```
# where sources are
DEB_SRCDIR = $(CURDIR)/src
# in which directory to build
DEB_BUILDDIR = $(DEB_SRCDIR)/build
# in which directory to install the software
DEB_DESTDIR = $(CURDIR)/destination
```

表 3.1 /usr/share/cdbs/1/rules/buildvars.mk で設定される変数

CURDIR	パッケージを作成しているディレクトリの名前.
DEB_SOURCE_PACKAGE	ソースパッケージの名前.
DEB_VERSION	完全な Debian Version.
DEB_NOEPOCH_VERSION	Debian version without epoch .
DEB_ISNATIVE	native パッケージの場合は空ではない (条件分岐に使用)..
DEB_ALL_PACKAGES	作成される全てのパッケージのリスト.
DEB_INDEP_PACKAGES	アーキテクチャに依存しないパッケージのリスト.
DEB_ARCH_PACKAGES	アーキテクチャに依存するパッケージのリスト.
DEB_PACKAGES	通常の (udeb ではない) パッケージのリスト.
DEB_UDEB_PACKAGES	udeb の場合, そのリスト.
DEB_ARCH	Debian アーキテクチャ. 後方互換のために残されている..
DEB_HOST_ARCH_CPU	Debian アーキテクチャの CPU 情報.
DEB_HOST_ARCH_OS	Debian アーキテクチャの OS 情報.
DEB_DESTDIR	パッケージ作成の際にソフトを install するディレクトリ. . 単一パッケージの場合, \$(CURDIR)debian/パッケージ名 複数のパッケージを作成する場合には \$(CURDIR)debian/tmp

3.3.2 buildcore.mk によるターゲットの設定

buildcore.mk はパッケージ作成のための基本的なターゲットを提供します. [Debian Policy Manual] によれば, パッケージ作成の際 debian/rules ファイルの中で外部から参照されうるターゲットは, 以下の通りです:

- build
- binary, binary-arch, binary-indep
- build-arch, build-indep (optional)
- get-orig-source (optional)
- clean (optional)
- path (optional)

buildcore.mk を include するとこれらのターゲットが細分化されて提供されます. buildcore.mk が提供するターゲットの一覧を図 3.2 に示します.

実際には, buildcore.mk 自体はパッケージに対してなんの処理もしません. よって適宜 rules を記述することになります. 例として, [CDBS Documentation Rev. 0.4.0] に記載されている foo について解説します. ここで foo は

- ソースパッケージ foo を生成
- バイナリとして foo(arch-dep) と foo-data(arch-indep) を生成

するパッケージだとします.

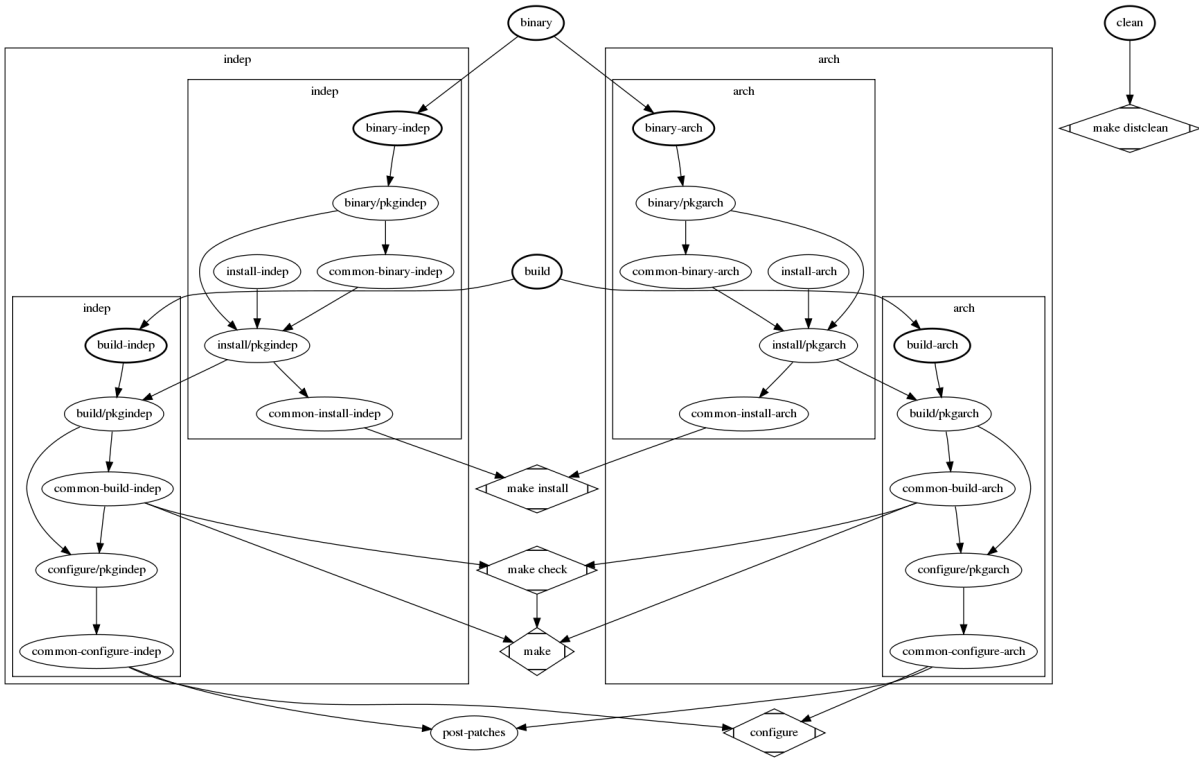


図 3.2 buildcore.mk で提供されるターゲットの流れ ([CDBS Documentation Rev.0.1.2]).

```
#!/usr/bin/make -f
include /usr/share/cdb/1/rules/buildcore.mk

# pre-configure action
makebuilddir/foo::
    ln -s plop plop2
# post-configure action
configure/foo::
    sed -ri 's/PLOP/PLIP/' Makefile
configure/foo-data::
    touch src/z.xml
# post-build action
build/foo::
    /bin/bash debian/scripts/toto.sh
build/foo-data::
    $(MAKE) helpfiles
# post-install action
install/foo:
    cp debian/tmp/myfoocmd debian/foo/foocmd
    find debian/foo/ -name "CVS" -depth -exec rm -rf {} \;
install/foo-data:
    cp data/*.png debian/foo-data/usr/share/foo-data/images/
    dh_stuff -m ipot -f plop.bz3 debian/foo-data/libexec/
# post deb action
binary/foo:
    strip --remove-section=.comment --remove-section=.note --strip-unneeded \
    debian/foo/usr/lib/foo/totoz.so
# pre-clean action
cleanbuilddir/foo::
    rm -f debian/fooman.1
```

コメントとしてターゲットを記述するタイミングを記載しました。ターゲットの記法は

target/package::

です。後ろの :: が重要です。

3.3.3 debhelper.mk による dh_ の自動化

CDBS の一番の御利益が、この debhelper.mk です。CDBS では主な dh_ コマンドの呼び出しを debhelper.mk において行なうため、debian/rules 内の dh_ の殆んどが不要になります。debhelper.mk によって呼び出される dh_ を

表 3.2 に示します。

表 3.2 /usr/share/cdb/1/rules/debhelper.mk で管理される dh_ コマンド

dh_builddeb	dh_installchangelogs	dh_installemacsen	dh_installman
dh_perl	dh_clean	dh_installcron	dh_installexamples
dh_installmenu	dh_shlibdeps	dh_compress	dh_installdeb
dh_installinfo	dh_installpam	dh_strip	dh_fixperms
dh_installdebconf	dh_installinit	dh_link	dh_gencontrol
dh_installdirs	dh_installogcheck	dh_makeshlibs	dh_install
dh_installdocs	dh_installogrotate	dh_md5sums	

debhelper.mk によって呼び出される dh_ コマンドに対しては、パラメータ設定は (大抵の場合) 不要です。debhelper の呼び出しをカスタマイズする変数は debhelper.mk 冒頭のコメント行を参照して下さい。[CDBS Documentation Rev. 0.4.0] には以下の例があります:

依存関係がシビアな共有ライブラリについて

```
DEB_DH_MAKESHLIBS_ARGS_libfoo := -V''libfoo (>= 0.1.2-3)''
DEB_SHLIBDEPS_LIBRARY_arkrpg := libfoo
DEB_SHLIBDEPS_INCLUDE_arkrpg := debian/libfoo/usr/lib/
```

ChangeLog のファイル名が一般的でない場合

```
DEB_INSTALL_CHANGELOGS_ALL := ProjectChanges.txt
```

.py を圧縮せずにパッケージ化する場合

```
DEB_COMPRESS_EXCLUDE := .py
```

ここでの記法

:=

に注意して下さい。:= は上書きです。CDBS の提供する変数に追加する場合には += を使用します。

3.3.4 patch の管理

dpatch, quilt, そして CDBS 用の simple-patchsys の rules が提供されています。quilt, dpatch については include するだけで patch を適用する rule が適用されます。

simple-patchsys の場合は debian/patches に patch を置くだけで、パッケージ作成時に patch を適用し clean の際には元に戻します。patch level は 3 まで ok です。自動的に適用しようと試みます。

3.4 class を使用する

前小説で rules について簡単にまとめました。ここでは幾つかの class について紹介します。

3.4.1 Makefile の場合:makefile.mk

autotools を使わず Makefile のみを使用するソフトウェアには makefile.mk が便利です。[CDBS Documentation Rev. 0.4.0] では、元々の Makefile が

- 名前が MaKeFile で
- make mrproper で clean
- make myprog で build

- make check で check
- make install で install

というソフトウェアの場合について例示しています:

```
#!/usr/bin/make -f
include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/makefile.mk

DEB_MAKE_CLEAN_TARGET := mrproper
DEB_MAKE_BUILD_TARGET := myprog
DEB_MAKE_INSTALL_TARGET := install DESTDIR=$(CURDIR)/debian/tmp/
# no check for this software
DEB_MAKE_CHECK_TARGET := check
# allow changing the makefile filename in case of emergency exotic practices
DEB_MAKE_MAKEFILE := MakeFile
# example when changing environment variables is necessary :
DEB_MAKE_ENVVARS := CFLAGS='-fomit-frame-pointer'
```

3.4.2 Autotools の場合: autotools.mk

いわゆる `configure && make && make install` なソフトウェアの場合は `autotools.mk` が便利です。冒頭にも例示しましたが、標準的な `autotools` を使用するソフトウェアの場合には

```
#!/usr/bin/make -f
include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/autotools.mk
```

となります。

`configure` へのオプションや環境変数の設定を行なう場合には以下の様にします:

```
DEB_CONFIGURE_EXTRA_FLAGS := --with-ipv6 --with-foo
COMMON_CONFIGURE_FLAGS := --program-dir=/usr
DEB_CONFIGURE_SCRIPT_ENV += LDLFLAGS='-Wl,-z,defs -Wl,-01'
```

ここでも `+=`, `:=` の意味は変わりません。例えば

```
#!/usr/bin/make -f
include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/autotools.mk

# normally
DEB_MAKE_INSTALL_TARGET := install DESTDIR=$(DEB_DESTDIR)
# example to work around dirty makefile
# DEB_MAKE_INSTALL_TARGET := install prefix=$(CURDIR)/debian/tmp/usr
DEB_MAKE_CLEAN_TARGET := distclean
# example to activate check rule
DEB_MAKE_CHECK_TARGET := check
# overriding make-only environment variables :
# (should never be necessary in a clean build system)
# (example borrowed from the bioapi package)
DEB_MAKE_ENVVARS := 'SKIPCONFIG=true'
```

など。

他にも Perl, Python, Ruby, GNOME, KDE, Ant, HBuild(Haskell) 用の class があります

3.5 まとまってないまとめ

そんな所で時間が切れてしまいました*3。

CDBS を使いはじめたら、もう `debhelper` には戻れない体になってしまうわけですが、いかんせん CDBS ってドキュメント少ないんですよね。この文書が最初の一步になれば幸いです。

*3 数日前 HDD が逝って四苦八苦しめました。うー

参考文献

- [CDBS Documentation Rev. 0.4.0] Marc (Duck) Dequènes, Arnaud (Rtp) Patard, 2007: CDBS Documentation, <http://perso.duckcorp.org/duck/cdbs-doc/cdbs-doc.xhtml>
CDBS の online ドキュメンテーションです。パッケージに含まれているドキュメントより、こっちの方が情報が多く、参考になります。
- [CDBS Documentation Rev.0.1.2] Marc (Duck) Dequènes, Arnaud (Rtp) Patard, Peter Eisentraut, Colin Walters, 2007: [/usr/share/doc/cdbs/cdbs-doc.html](http://usr/share/doc/cdbs/cdbs-doc.html).
Lenny の CDBS(ver. 0.4.52) 付属のドキュメントです。Web で公開されているドキュメントよりは古いです。
- [CDBS 移行への 1st step] Tatsuki Sugiura, 2006: CDBS 移行への 1st step <http://sugi.nemui.org/doc/cdbs/cdbs-trans-1st.html>
既存のパッケージを CDBS へ移行する場合、非常に参考になると思います。
- [Online CDBS Gallery] Online CDBS Gallery, <http://cdbs.ueberalles.net/index.html>
CDBS を使っている debian/rules を見ることができます。新たに CDBS へ移行する際に参考になるでしょう。
- [Debian Policy Manual] Ian Jackson & Christian Schwarz, 1996: Debian Policy Manual, <http://www.debian.org/doc/debian-policy/>
- [Debian パッケージ作成の手引き] 小林 儀匡, Debian パッケージ作成の手引き, <http://www.debian.or.jp/~nori/debian-packaging-guide/index.html>
Debian パッケージ作成の手引きです。「debhelper を使わない場合 使う場合 CDBS への移行」と順序立てて説明されています。
- [やまだ & 鵜飼 (2006)] やまだあきら (著), 鵜飼文敏 (監修), 2006: 入門 Debian パッケージ, 技術評論社, ISBN4-7741-2768-X

4 cdn.debian.or.jp, cdn.debian.net における取り 組み

荒木 靖宏

いつでも必要なソフトウェアやコンテンツを安価に入手する手段として CDN が広くつかわれている。Debian は deb の安定入手手段の有無がシステムの信頼性を左右するシステムであり、その特殊性を考慮した CDN システムが必要となる。今回は cdn.debian.or.jp, cdn.debian.net における取り組みを紹介する。

4.1 CDN とは

Content Delivery Network (CDN) はウェブコンテンツ配置および配送方法として Akamai 社によりサービスされ広く知られることになった。当初から一部の人気の高いサーバへのトラフィック集中によるサーバ停止の回避、海外のリッチコンテンツ取得の高速化、トラフィック分散によるネットワークおよびサーバの利用平準化などの理由で広く受け入れられた。

CDN という用語自体は WWW に限ることなく、一般にコンテンツを取得するための配送手段や方法全体を指す場合がある。たとえば、Winny や Bittorrent などのコンテンツを取得するために特別に設計されたプロトコルを用いて、P2P ネットワークを構成するような手法も含まれる。

4.2 Debian における CDN の現状

4.2.1 利用法とユーザから見た動作

cdn.debian.or.jp では Debian でインストール時から広く deb ファイルの入手に使われる apt で使える CDN として設計し、運用している。そのため、Debian における CDN の利用法は至極簡単である。/etc/apt/source.list に記述する APT リポジトリとして、

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

以上のように指定するだけでユーザは今までと変わらず apt コマンドを使用できる。

現在、cdn.debian.or.jp, ftp.jp.debian.org, cdn.debian.net の名で運用している。

サービス時の手順と構成は以下ようになる。(図 4.1)

1. ユーザが apt-get コマンドを行うと cdn.debian.or.jp を DNS で問い合わせる
2. cdn.debian.or.jp を管理する DNS はサーバ候補 (surrogate) 選択する
3. 選択結果を DNS のリプライとして返す
4. apt は cdn.debian.or.jp として Surrogate C を使用する。

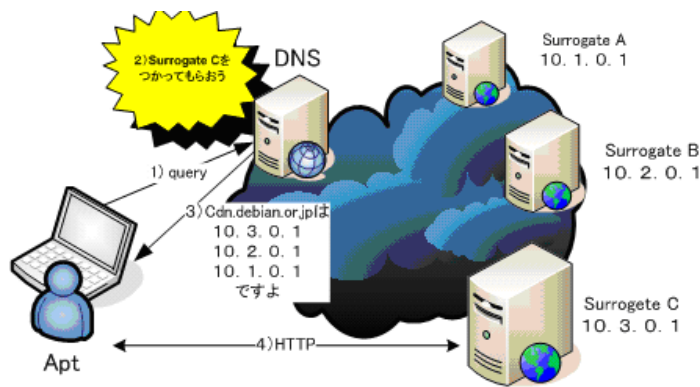


図 4.1 ユーザから見た cdn.debian.or.jp の動作

4.2.2 構築の方法

実際にはクライアント DNS は次のように動作している。

1. クライアント DNS が cdn.debian.or.jp を検索すると、CNAME deb.cdn.araki.net が debian.or.jp の DNS (BIND) から返答される。
2. クライアント DNS は deb.cdn.araki.net を得るために cdn.araki.net の NS レコードを問い合わせ、ns.cdn.araki.net, plat.debian.or.jp, debian.topstudio.co.jp, osdn2.debian.or.jp を得る。
3. クライアント DNS はいずれかのホストに deb.cdn.araki.net の A レコードを問い合わせ、サロゲートの IP アドレスを dns_balance から得る。
4. クライアント DNS は、ユーザクライアントに 3 で得た情報を返す。

2 を実現するための Araki.net の BIND 設定は次のようになる。

```
cdn          IN      NS      ns.cdn
             IN      NS      plat.debian.or.jp.
             IN      NS      debian.topstudio.co.jp.
             IN      NS      osdn2.debian.or.jp.
```

DNS Balance はユーザの IP アドレスと、何らかの方法でサーバをランクづけした表を元にユーザが接続すべきサイトを指示します。この表は一定時間毎に読み込み直され、これにより動的な負荷分散が可能です。

(DNS Balance の配布ページ <http://www.dnsbalance.ring.gr.jp> より)

3 を実現するための dns_balance の設定は次のようなサーバをランク付けした ruby 形式のファイルである。

```
$addr_db = {
  "default" => {
    "ns.cdn.araki.net" => [
      [[210,157,158,38], 0],
    ],
    "localhost" => [
      [[127,0,0,1], 0],
    ],
    "deb.cdn.araki.net" => [
      [[61,115,118,67], 1000],
      [[133,50,218,117], 10],
      [[202,229,186,27], 20],
      [[133,5,166,3], 10],
      [[130,54,59,159], 10],
      [[210,157,158,38], 9900],
    ],
  },
}
```

この設定ファイルに基づき、ホスト IP アドレスの後ろの数字は優先度であり、1 (優先度高) から 9999 (優先度最低) までの整数で指定する。

4.2.3 cdn.debian.or.jp のシステムと動作

CDN システムが完全に動作しユーザから使用されるためには、システムが完全なファイルを提供すること、システムが安定して動作すること、そして CDN を使った場合に高速に動作していることが求められる。

提供ファイルの完全性

このために以下二点を満たさねばならない。

- 個々のファイルがコンテンツ提供者たる deb ファイル配布元と同一であること
- apt-get update の結果取得するファイル群がどの Surrogate でも入手できること

前者については、deb はそのファイルの md5 値、sha1 値とともに配布され、ユーザが使用する apt で確認後に利用されるため CDN を使用した場合でも問題にならない。

後者についてはユーザが apt-get update を行ったときに接続する Surrogate と apt-get dist-upgrade を行ったときに接続する Surrogate は同一であるとは限らないため、DNS が Surrogate として返すサーバが保持するファイルは同一である必要がある。cdn.debian.or.jp では Debian プロジェクトで一般に行われている方法と同様に、rsync プロトコルを用い、push ミラーを行っている (図 4.2)。そのため、cdn.debian.or.jp のサロゲート内で最上流にあるサーバとミラーが同一であることを 2 分毎に rsync ミラー終了時に作成されるスタンプファイルを確認して、同一でないサーバはサロゲート候補から一時的に除外している。

ただし、これはミラーの配送ツリーの管理を行う必要があるため、日本国内のミラーに限定している。

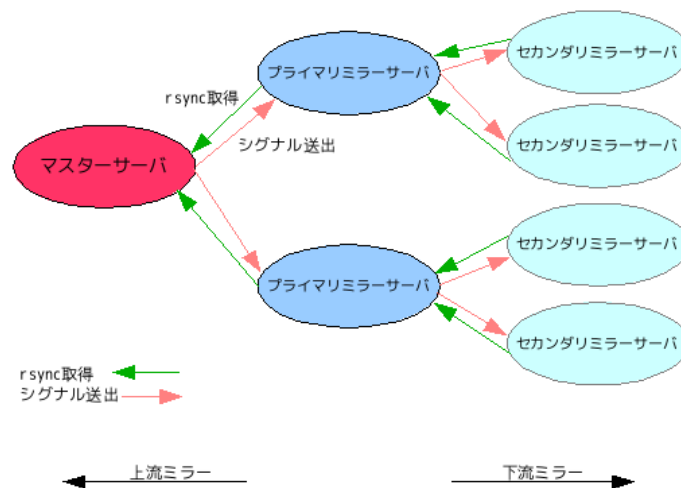


図 4.2 Debian サロゲートの rsync によるミラー

システムの安定動作

先に述べたように、ユーザは CDN を使用する際には DNS を最初を使用するため、DNS の安定運用がカギとなる。そのため、cdn.debian.or.jp を管理する DNS サーバはまったく独立に動作するサーバで行っている。

また、サーバの動作を確認は、5 秒以内に HTTP のレスポンスを返さないサーバはサロゲート候補から一時的に除外している。

後述するように、サロゲートの生死確認は 2 分ごとに dns_balance の動作に反映される。ローカル DNS にキャッシュされる情報とあわせて最悪でも 3 分以内に動作していないサーバは排除される。

高速動作

cdn.debian.or.jp では DNS で問い合わせされるとサロゲートリストとして複数の IP アドレスを返す。この IP アドレスはラウンドロビンで選択しているわけではなく、提供可能なサーバキャパシティやネットワーク速度を考慮し、設定している。

4.2.4 IP アドレスの位置情報を使用したサーバ選択

当初は cdn.debian.or.jp として運用していたものの、その後に global に分散する Debian ミラーに対応させた。

global に CDN を展開する場合には地理的に近いサーバ群からある程度絞込むのが有効である。現在、GeoIP など無料で IP と地理情報のマッピング提供者が現れており、debian パッケージになっていることもあり、本システムでは Maxmind の GeoIP を使用している。

Dns_balance には AS あるいはネットワークアドレスごとに振り分ける IP アドレスの指定が可能であるが、

- Debian のミラーは国別地域別に制御されていること
- 16 ビットで指定される AS と AS 間の経路の測定が難しい
- 個々のネットワーク間の経路を指定するのは現実的でない

以上の理由から、dns_balance に接続する IP アドレスの国名、大陸名を逆引きし、その結果をつかって返す A レコードを変更するように dns_balance を拡張している。

大陸別の設定ファイルは次のように、6 大陸別である。

```
yaar@loon3:~/playground/cdn$ ls continent/  
AF_deb_cdn_araki_net.rb  EU_deb_cdn_araki_net.rb  OC_deb_cdn_araki_net.rb  
AS_deb_cdn_araki_net.rb  NA_deb_cdn_araki_net.rb  SA_deb_cdn_araki_net.rb
```

さらに国別の設定ファイルを置く。

```
yaar@loon3:~/playground/cdn$ ls country/  
FRA_deb_cdn_araki_net.rb  JPN_jp_cdn_araki_net.rb  
JPN_deb_cdn_araki_net.rb  KOR_deb_cdn_araki_net.rb
```

これらの設定ファイルをつかって、dns_balance が実際に読み込む次のような設定ファイルを 2 分ごとに作成している。サーバの生死確認などはこのタイミングで反映される。

```
$addr_db = {"default"=>{"localhost"=>[[[127, 0, 0, 1], 0]], "deb.cdn.araki.net"=>[[[61, 115, 118, 67], 1000], [[61, 206, 119, 174], 20], [[202, 229, 186, 27], 20], [[203, 178, 137, 175], 9000], [[210, 157, 158, 38], 9900]], "ns.cdn.araki.net"=>[[[210, 157, 158, 38], 0]], "jp.cdn.araki.net"=>[[[61, 115, 118, 67], 1000], [[61, 206, 119, 174], 20], [[202, 229, 186, 27], 20], [[203, 178, 137, 175], 9000], [[210, 157, 158, 38], 9900]]}, "KOR"=>{"deb.cdn.araki.net"=>[[[143, 248, 234, 110], 20]]}, "SA"=>{"deb.cdn.araki.net"=>[]}, "EU"=>{"deb.cdn.araki.net"=>[[[141, 76, 2, 4], 9000]]}, "AF"=>{"deb.cdn.araki.net"=>[]}, "AS"=>{"deb.cdn.araki.net"=>[[[61, 115, 118, 67], 1000], [[61, 206, 119, 174], 20], [[202, 229, 186, 27], 20], [[203, 178, 137, 175], 9000], [[210, 157, 158, 38], 9900]]}, "JPN"=>{"deb.cdn.araki.net"=>[[[61, 115, 118, 67], 1000], [[61, 206, 119, 174], 20], [[202, 229, 186, 27], 50], [[203, 178, 137, 175], 9000], [[210, 157, 158, 38], 9900]]}, "jp.cdn.araki.net"=>[[[61, 115, 118, 67], 1000], [[61, 206, 119, 174], 20], [[202, 229, 186, 27], 50], [[203, 178, 137, 175], 9000], [[210, 157, 158, 38], 9900]]}, "NA"=>{"deb.cdn.araki.net"=>[[[204, 152, 191, 39], 9000], [[128, 30, 2, 36], 9000], [[35, 9, 37, 225], 9000]]}, "OC"=>{"deb.cdn.araki.net"=>[[[150, 203, 164, 37], 9000]]}, "FRA"=>{"deb.cdn.araki.net"=>[[[193, 54, 19, 19], 9000], [[194, 2, 0, 36], 9000]]}}
```

4.3 使用実績

以下 3ヶ月分の plat.debian.or.jp で動作している DNS へのアクセス実績を示す。

- 2008 年 7 月 16 日から 10 月 15 日までの三ヶ月間の利用実績を解析した。
- 本システムでは DNS の A または Any に対して返答を戻すことから、それ以外のクエリに関しては無効なアクセスとして処理している。
- 地域はアクセス元の IP アドレスを GeoIP ライブラリから算出している。
- 全 DNS 動作ホストは、plat.debian.or.jp, osdn2.debian.or.jp, debian.topstudio.co.jp であり、この 3 倍のクライアントからのアクセスがあると推測される。

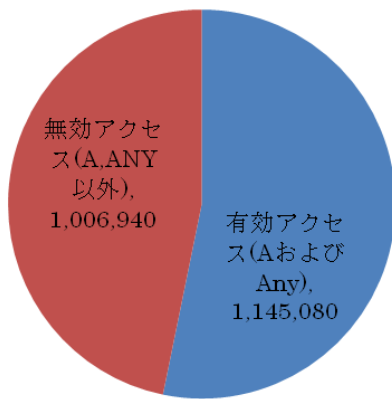


図 4.3 DNS アクセス種別

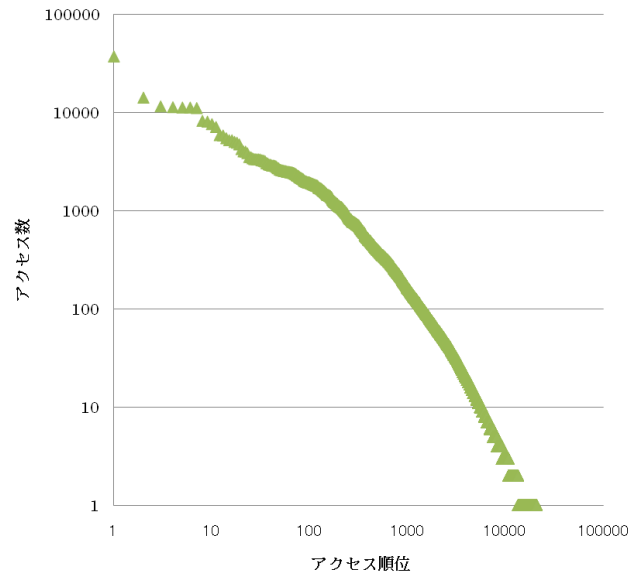


図 4.4 ホスト別アクセス順位とアクセス数

ユニークホスト数は 20554 であるが、その分布は極端に偏っている。

図 4.5 DNS クエリ数上位 20 カ国

JPN	808138
USA	87767
CAN	82566
KOR	38596
CHIN	26670
FIN	14558
TWIN	12527
IND	7941
IDN	6463
HKG	5616
RUS	3975
SGP	3656
DEU	2990
GBR	2792
AUS	2635
ESP	2632
THA	2623
MYS	2475
PHL	2359
ITA	2196

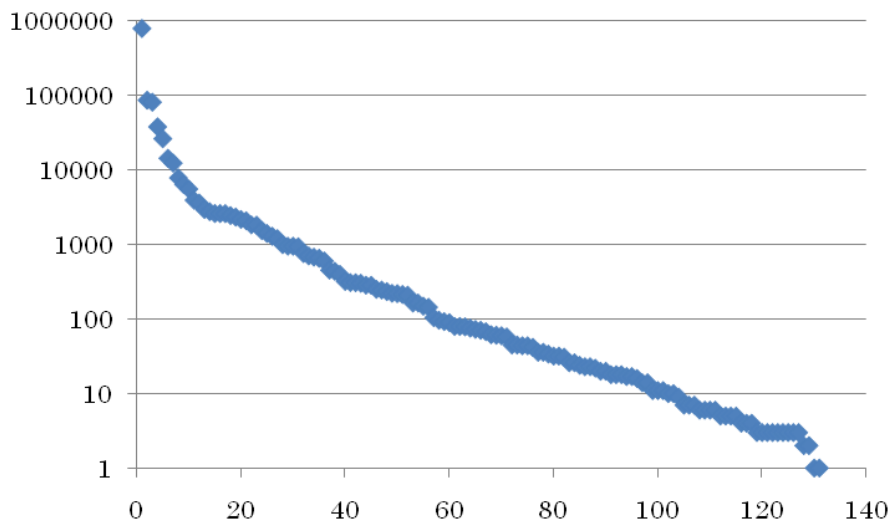


図 4.6 DNS クエリ数上位国 (横軸)-クエリ数 (縦軸)

なお、国が判定できないのは 1552 件、率にして 0.135% であった。

図 4.7 CDN 振り分け先実績

地域	アクセス数
アジア	570137
アジア (日本と韓国除く)	55539
日本	509367
韓国	5231
ヨーロッパ	16804
ヨーロッパ (フランス除く)	15154
フランス	1650
北米	142207
オセアニア	10766
アフリカ	160
その他	404996
総有効アクセス数	1145070

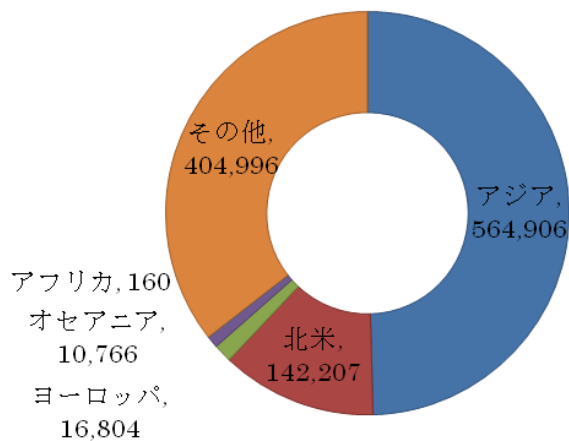


図 4.8 地域別振り分け実績

図 4.10 日本での振り分け実績

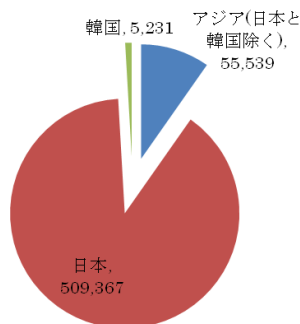


図 4.9 アジア地域の振り分け実績

ホスト名	IP	振り分け数
runner.oyu-net.jp.	61.206.119.174	690596
dwarf.topstudio.co.jp.	202.229.186.27	660091
hanzubin.st.wakwak.ne.jp	61.115.118.67	649512
studenno.kugi.kyoto-u.ac.jp.	130.54.59.159	599550
dennou-q.geo.kyushu-u.ac.jp	133.5.166.3	468730
frost.nemui.org.	218.219.152.77	445589
dennou-h.ep.sci.hokudai.ac.jp.	133.87.45.30	379888
ftp.nara.wide.ad.jp	203.178.137.175	58825
plat.debian.or.jp	210.157.158.38	6095

アジア地域向けのサーバおよび韓国地域向けのサーバは日本向けのサーバ群設定と全く同一のものを設定している。

日本での振り分け実績は、設定した頻度情報をほぼ正確に反映しており、プログラムの動作および、期間中の各サロゲートに大きな障害がなかったことを示している。

4.4 関連研究とディスカッション

<http://prisms.cs.umass.edu/~kevinfu/papers/secureupdates-hotsec06.pdf> でのサーベイ結果のように、パッケージ配布にはいくつかの方法と危険性が指摘されている。このサーベイ結果で指摘されるように、Debian では個々のファイルごとに sha-1 および MD5, SHA-256 のハッシュ値を配布している。

加えて、本システムの日本での運用においては、マスターサーバからサロゲートへのファイル同期時に配送ツリーの確認とファイルの一致を確認しており、障害時の排除機能を有するため、単なるファイルのミラーリング以上の安全性は確保されている。

OpenSuse における配送 (mirrorbrain, http://en.opensuse.org/Build_Service/Redirector) は sourceforge をはじめ多くの CDN で採用している方法である。クライアントへのサロゲート決定の方法は GeoIP の使用しており、本システムと同等のものである。

ただし、Debian の apt システムの制限により、本システムでは HTTP リダイレクトではなく、DNS を使用している。

また、mirrorbrain ではサロゲートへの配送の確認は行っていないものと思われる。

apt の P2P 対応も有力な配布手段である。apt-p2p (<http://www.camrdale.org/apt-p2p/>) により、apt の P2P 対応が進められている。有力な候補である。ただし、Kashimir プロトコルをクライアントで直接使うものであり、ネットワーク利用ポリシーとの競合や install 時に利用できない点、取得希望ファイルが入手できない場合に HTTP にフォールバックするため速度に問題があり、今後検証すべき問題も多い。

サロゲートが正しく動作しているのか、その能力に応じたサロゲート使用ができているかどうかは、分散したファイルサーバからのファイル入手において重要な要素である。この組み合わせを考えると、表 4.1 のように分類される。本システムでは、サーバやサーバとクライアント間のネットワークの実測を行わないこと、このサロゲートの運用は独立して行うことでサロゲートへ特別なプログラムを必要としないことで、コストの劇的な削減を可能としている。

表 4.1

	申告ベースの能力	動的測定・実績ベースの能力
申告ベースのヘルスチェック	Debian ミラーサーバリスト	Debian ミラーの口コミ・トライアンドエラー
動的測定・実績ベースのヘルスチェック	本システム	理想的なシステムだがコスト大商用 CDN など

4.5 課題と将来の展望

ここまで説明してきた、`cdn.debian.or.jp` の動作には改善すべき点が多数存在する。改善の展望としていくつか挙げる。

4.5.1 アクセス実績あるいはローカルポリシーに基いた CDN 配置

アクセス実績によると、日本、米国、カナダ、韓国、中国、フィンランド、台湾、インド、インドネシア、ホンコン、ロシアと続く。現状では、大陸毎に設定されたサロゲートは存在するものの、日本、韓国、アメリカを除けば各国毎に設定されたサロゲートは存在しない。

このアクセス実績にもとづき、これらの国内のミラーを生かした設定を近い将来行いたい。

apt-get コマンドの HTTP REDIRECT

apt-get コマンドは HTTP REDIRECT に対応していない。

そのため、前述したように `mirrorbrain` などの CDN は使用することができない。

HTTP REDIRECT は、いったん HTTP GET などで接続してきたクライアントに対して、新たにそのリソースが存在する URL を通知するものである。この仕組みをうまくつかった CDN として、Coral Content Distribution Network (Coral CDN) がある。Coral CDN はサロゲート間で P2P によるファイル配置し、そのインターフェースとして、HTTP を使用し、しかも使用にはインターネットから取得可能なファイルであれば制限をかけていない。さらに、Apache を使った一時配布サーバでは HTTP REDIRECT をつかって Coral CDN に誘導することも推奨されている。ただし、現状で、Coral CDN を使うために、

```
deb http://cdn.debian.or.jp.nyud.net:8090/debian/ stable main contrib non-free
```

を指定することも可能だが、少なくとも日本においては Coral CDN を担うサロゲートが存在しないこともあって非常に低速である。ただし韓国や中国では広くつかわれており、将来の拡張に使用したい。

マスターサーバからサロゲートへのミラーリング方法

本システムでは既存の `rsync` によるミラーリング手法には手をつけていない。行ったのは `rsync` をつかってタイムスタンプを確認し、サロゲートのヘルスチェックのひとつに使用したのみである。debian における `rsync` の使い方現状のミラーリングに由来する問題は

1. `master.debian.org` からの末端までのあいだは冗長化されていない
2. `rsync` はディレクトリの同期を取る方法であり、ファイルの転送に必ずしも適していない。deb のように依存関係を記述したメタデータを含むファイルであれば、ファイル毎の push であっても配送に問題はない
3. 配送がユニキャストである
4. 利用頻度や重要度にかかわらず同様の配送を行っている

等さまざまである。

Debian が潤沢なネットワーク環境と強力な CPU を要するホストでない場合でもミラーリングないしサロゲートからのファイル入手を可能とする FLUTE などの配送手法が必要になるろう。

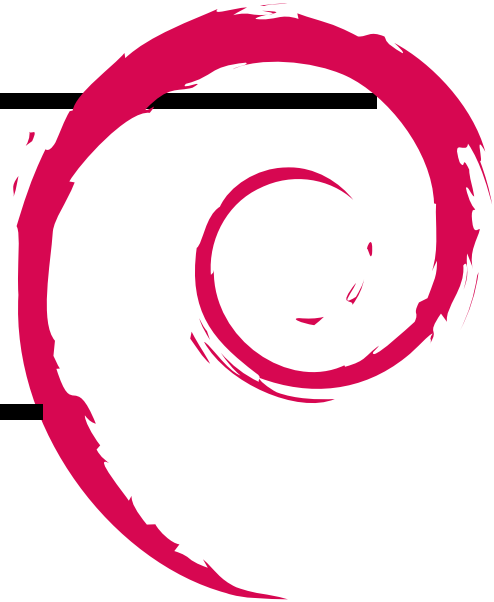
4.6 おわりに

いつでも必要なソフトウェアやコンテンツを安価に入手する手段として CDN はこれからも様々な発展を続けると考える。Debian は deb の安定入手手段の有無がシステムの信頼性を左右するシステムであり、CDN の広範な活用が今後ますます求められるようになると思う。

本システムの導入により、サロゲート個々の安定性がクライアントに直接影響することはかなり抑えることができるが、より確実なパッケージ入手のために、信頼性が高く安定動作しているサロゲートの増加を Debian プロジェクトでは引き続き求めている。

5 今後の予定

山下 尊也



5.1 次回

次回は、2008 年 11 月 7 日, 8 日に行われる関西オープンフォーラム^{*4}にて開催する予定です。

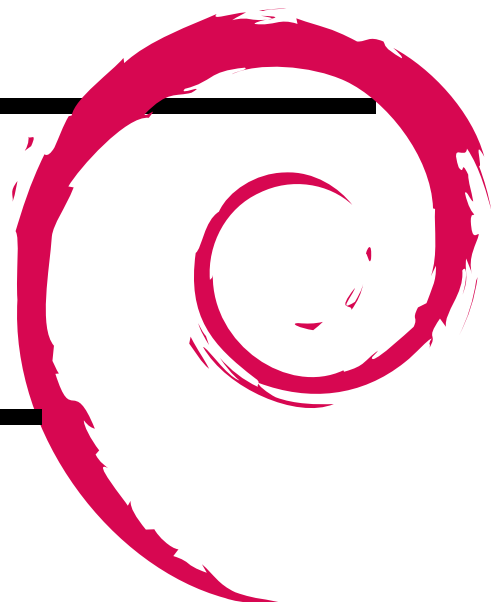
5.2 KDR のお知らせ

関西 Debian 勉強会の有志で関西 Debian 勉強会とは独立した形で、週に一度、読書会 (KDR) を開いています。詳しくは KDR 公開用ページ^{*5} をご覧下さい。

^{*4} <http://k-of.jp/>

^{*5} <http://qwik.jp/kdrweb/>

6 メモ



関西下ビアノ勉強会



Debian 勉強会資料

2008 年 10 月 19 日 初版第 1 刷発行
関西 Debian 勉強会（編集・印刷・発行）
