第 85 回 東京エリア Debian 勉強会資料

月刊 Deb専

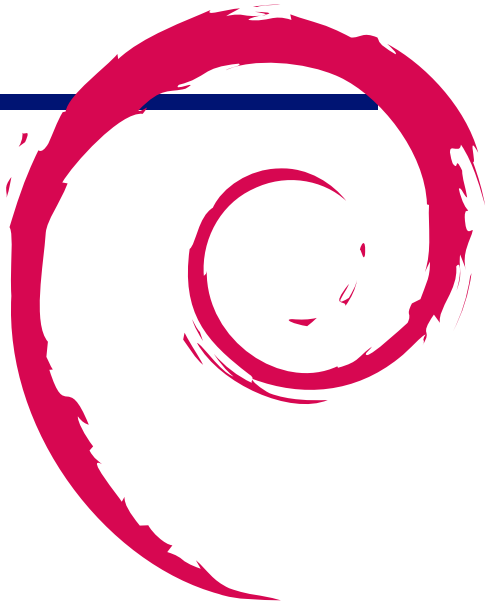
日本唯一のDebian専門月刊誌

2012 年 2 月 18 日

特集1：Ｄｅｂｉａｎ開発者のＫＤＥ環境あれこれ

特集2：月刊Ｄｅｂｈｅｌｐｅｒ

特集3：ｃｍａｋｅつかってみる



1 Introduction

野島　貴英

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

Debian 勉強会の目的は下記です。

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

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

デビアン勉強会

目次

1 Introduction 1

2 事前課題 3
- 2.1 鈴木崇文 3
- 2.2 dictoss(杉本　典充) 3
- 2.3 yamamoto 3
- 2.4 野島　貴英 3
- 2.5 日比野　啓 3

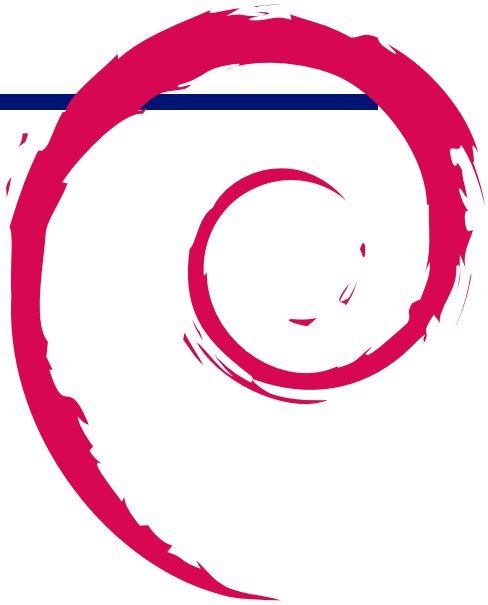
3 最近の Debian 関連のミーティング報告 4
- 3.1 東京エリア Debian 勉強会 84 回目報告 4

4 Debian Trivia Quiz 5

5 Debian 開発者の KDE 環境あれこれ 6
- 5.1 利用者としての KDE 導入方法 6
- 5.2 開発者としての experimental 版 KDE 導入方法 (KVM+spice) 7
- 5.3 Debian と KDE 環境のバージョン 9
- 5.4 KDE 環境の開発の特徴 9
- 5.5 Debian での KDE 環境のパッケージ開発 10
- 5.6 超簡易的に KDE 用プログラムの Debian パッケージを作ってみる 10
- 5.7 おわりに 11
- 5.8 参考文献 11

6 月刊 Debhelper 12
- 6.1 パッケージの make の前に… . 12
- 6.2 一連の dh_ほげほげコマンドへの追加の仕組み 12
- 6.3 dh_dpatch_patch コマンド . . 13
- 6.4 autotools だって使いたい . . . 13
- 6.5 おわりに 14

7 cmake 使ってみる 15
- 7.1 cmake とは 15
- 7.2 使ってみる 15
- 7.3 IDE 用のプロジェクトファイルを生成してみる 17
- 7.4 おわりに 17
- 7.5 参考文献 17



2 事前課題

野島 貴英

今回の事前課題は以下です:

1. 皆さんの Debian desktop 環境と、利用にあたって何か工夫があれば 200 文字以内でアピールください。(contrib 目標/未来指向なアピールはさらに絶賛歓迎)

この課題に対して提出いただいた内容は以下です。

2.1 鈴木崇文

gnome で desktop を利用しています。工夫というほどではないですが、便利なツールとして KDE 系のアプリや EBView を使っています。リモートデスクトップ (RDP) & VNC 用には、KRDC を使い、スクリーンショット用には KSnapshot を使っています。あとは英辞郎を購入して、 EBView でいつでも翻訳できるようにしています。

2.2 dictoss(杉本 典充)

低性能マシンは startx + icewm、中高性能マシンは gdm + xfce4 か gnome と使い分けています。 KDE は最近使ってないです。(KDE は重そうなイメージがある) カスタマイズしているのは、ページャの個数を 6 つに増やしている、複数のターミナルを重ならないように同時起動するシェルスクリプトを作り一発で画面をターミナルで埋め尽くせるようにしています。(昔タイル型ウィンドウマネージャ使えばいいのに、とか突っ込まれました)

2.3 yamamoto

メインに使っているのは、録画サーバも兼ねた Debian squeeze (amd64) です。外出時は気分次第で、 sid の i386 ネットブックと sid の amd64 ノート PC を選んでいます。どれも、特に何の変てつもない、ただの KDE 環境です。

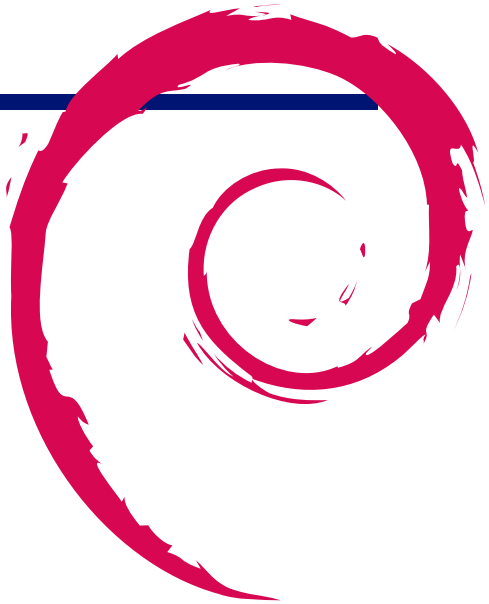
デスクトップとしての見た目は、壁紙すらデフォルトのままで、改造してませんが、機体としては家の LAN にぶら下がったマシン間を「 何か (?)」が行き来する、魔改造スクリプトがいくつか仕込んであり、ものぐさな私にはとっても快適です。

2.4 野島 貴英

GNOME 3.2.2 を Debian で利用しています。 unstable ではもの足りず、 experimental から upgrade して引っ張ってきてます。特にクールなカスタマイズは何もしてないですが、gnome-shell が javascript など解釈できるということから、将来ちょっとしたガジェットぐらい作ってみたいなーと思うこの頃です。あと、 gxconsole(`http://gnomefiles.org/content/show.php/gxconsole?content=132145`) が GNOME3.2.2 になっても、やっぱり欲しかったので、 GNOME 3.2.2 用に移植したい...

2.5 日比野 啓

普段はタイル型ウィンドウマネージャの XMonad を使っています。マルチディスプレイに対するサポートが使いやすくてプレゼンのときにも便利で気にいっています。最近、趣味のプログラミングのメインで使っている言語が Haskell なので、 Haskell でカスタマイズできることも魅力です。 gnome-session との組合せも使ってみましたが、なぜか sid ではうまく動かなくて残念。あと、画面の上下が狭くなるのが嫌なので、なんとかする方法が知りたい。



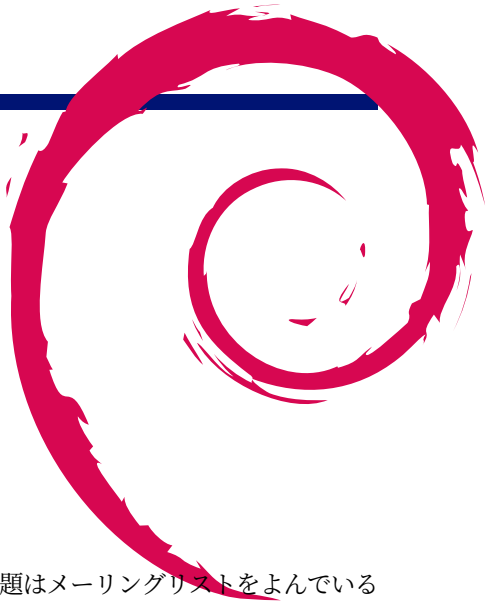
3 最近の Debian 関連のミーティング報告

野島　貴英

3.1 東京エリア Debian 勉強会 84 回目報告

1 月の東京エリア Debian 勉強会は久しぶりにあんさんぶる荻窪にて行われました。上川さんから、 Debian 勉強会予約システムに関する脆弱性と考察、また、 Debian の使える VPS に関する発表、岩松さんから Debian をつかった twitter 連携について発表がありました。また恒例の月刊 Debhelper は山田さんにより発表が行われ、 dh コマンドまわりをさらに深堀りした件と、 dh_builddeb コマンドに関して発表が行われました。

今後ますます WEB システム用途にも Debian は使われていくと思います。このあたりを意識した発表が多かったのが印象的でした。



4 Debian Trivia Quiz

野島 貴英

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

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

問題 1. 2/14 頃に wheezy に関して公募が行われました。何の公募でしょうか?

☐ A I18N 関係のサーバーシステム再構築の公募
☐ B 応援団長/イメージタレントの公募
☐ C Look and Feel のアート分野に関する公募

問題 2. 今年 2 月にセキュリティアップデートが終了した Debian のバージョンは何でしょう?

☐ A lenny
☐ B sarge
☐ C woody

問題 3. 1/25 に alioth になにがおきたか

☐ A vasks.debian.org が起動しなくなった
☐ B wagner.debian.org が起動しなくなった
☐ C churro が起動しなくなった

問題 4. 今年の DebConf12 はいつ開かれる予定?

☐ A 2012/7/1-7/7
☐ B 2012/7/8-7/14
☐ C 2012/のお盆

問題 5. wheezy に入る予定の Linux カーネルバージョンはいくつでしょう?

☐ A 3.0 系列
☐ B 3.1 系列
☐ C 3.2 系列

問題 6. インストール/アップグレード/消去テストを担う QA チームの強力なツールの名前は?

☐ A init
☐ B piuparts
☐ C upstart

問題 7. 2/18 最新の Debian の安定版のリリース番号はいくつでしょうか?

☐ A 6.0.1
☐ B 6.0.2
☐ C 6.0.4

問題 8. Debian Games Team から 2/25,2/26 に行われる作業協力の呼びかけがアナウンスされてます。何でしょうか?

☐ A バグ取りパーティー (BSP) の呼びかけ
☐ B Games Team の IRC 会議参加の呼びかけ
☐ C ゲームのスクリーンショット取り協力の呼びかけ

問題 9. W3Techs の調査によると、世界で利用されている Linux ベースの Web サーバーで 2012 年 1 月で僅差ではあるものの No.1 になったディストリビューションは何でしょう?

☐ A Debian に決まってるじゃないか
☐ B CentOS
☐ C Ubuntu Server

5 Debian 開発者の KDE 環境あれこれ

野島 貴英

最近の Debian をそのままインストールすると、特に指定しない場合 GNOME というデスクトップ環境がインストールされます。しかしながら、 Debian ではいくつもデスクトップ環境が用意されており、ユーザは自由にこれらを選んで使うことができます。デスクトップ環境はユーザにとってはいつも使う環境ですから、いろいろとこだわりもあるかとおもいます。今回はそんな中、 KDE というデスクトップ環境についてあれこれ語ってみます。

5.1 利用者としての KDE 導入方法

Debian の安定版の利用を検討していて、 KDE 環境をいわゆる利用者として使う為にインストールするやり方について簡単に述べます。

1. 安定版の Debian のインストール DVD を用意します。
2. インストーラのメニュー画面が出ましたら、 TAB キーをおすと画面下の方に編集可能な行が現れますので、以下の例ように”desktop=kde” という文言を追加します。なお、日本語 106 キーボードを使っている場合、キートップの刻印の通りに”=” を押しても”=” 文字が入力できない場合がありますが、この場合は”^” の刻印のキーを押すと”=” 文字が入力できます。

```
/install.amd/vmlinuz vga=788 initrd=/install.amd/initrd.gz --- quiet desktop=kde
```

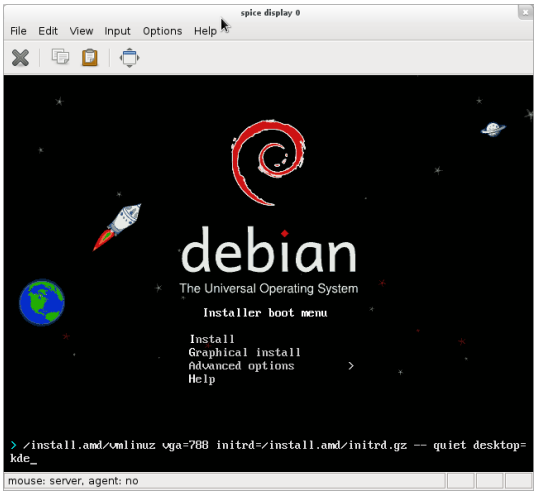


図 1 安定版インストール画面で TAB キーを押したときの様子

3. あとは通常どおりインストールを行います。インストールを進めていくと「 インストールするソフトウェアの選択:」のメニューが現れますので、”Debian desktop environment” を選択しておいてください。

4. インストールが完了しましたら、リブートを行います。
5. KDE 環境が起動します。

以上となります。簡単ですね。

5.2 開発者としての experimental 版 KDE 導入方法 (KVM+spice)

東京エリア Debian 勉強会にいらっしゃるような方々には、前述のインストールと環境ではきっと「ぬるゲー（笑）」な感じのはずです。その場合、是非とも experimental 版の KDE 環境を利用いただき、BTS 書き/パッチ開発/翻訳/デバッグなどの開発活動に勤しんでみましょう。ここでは、開発者向け KDE 環境導入について簡単に述べます。

- 開発者向けに experimental 版導入を前提にします。
- 仮想環境である KVM を利用して仮想環境上に導入します。これなら、ディスクイメージファイルをとっておけば、うっかり experimental 環境で aptitude full-upgrade して全く立ち上がらなくなっても（実話）あっさり復帰できます。
- サウンドももちろん欲しいので仮想デスクトップ環境として spice を使います。
- いつでもどこでも開発できるようにモバイル環境に構築します。

図 2 の KDE 開発環境の用意を想定します。

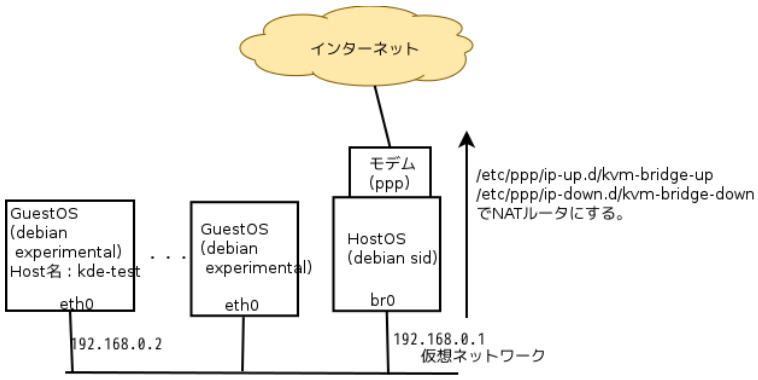


図 2　KDE 開発環境

以下は導入に関しての流れです。（細かい事は割愛します。操作にあたっては適宜 root 権限が必要だったりします）

1. HostOS となる PC の BIOS を操作して、CPU の仮想技術支援機構のスイッチを ON にしてブートしておきます。
2. HostOS に `http://www.debian.org/CD/netinst` から名刺サイズの CD イメージを落として置きます。
3. HostOS の/etc/network/interfaces に以下の追記を行い、br0 を作っておきます。

```
# 追記はここから。 aptitude install bridge-utils はやっておくこと。
auto br0
iface br0 inet static
        address 192.168.0.1
        netmask 255.255.255.0
        bridge_ports none
        bridge_stp off
        bridge_fd 0
        bridge_maxwait 0
```

4. HostOS の/etc/sysctl.d/bridge-filter-workaround.conf を作り、sysctl -p /etc/sysctl.d/bridge-filter-workaround.conf を実行して、br0 のフィルタを無効化しておきます。

```
# /etc/sysctl.d/bridge-filter-workaround.conf の中身
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
```

5. HostOS の/etc/ppp/ip-up.d/kvm-bridge-up,/etc/ppp/ip-down.d/kvm-bridge-down を作っておきます。他にフィルタとか必要であれば適当にどうぞ。

```
#!/bin/sh
# /etc/ppp/ip-up.d/kvm-bridge-up の中身
PATH=/bin:/usr/bin:/sbin:/usr/sbin
CDPATH=
sysctl -w net.ipv4.ip_forward=1
iptables -t nat -A POSTROUTING -o $PPP_IFACE -j MASQUERADE
iptables -A FORWARD -i br0 -o $PPP_IFACE -j ACCEPT
```

```
#!/bin/sh
# /etc/ppp/ip-down.d/kvm-bridge-down の中身
#!/bin/sh
PATH=/bin:/usr/bin:/sbin:/usr/sbin
CDPATH=
sysctl -w net.ipv4.ip_forward=0
iptables -t nat -D POSTROUTING -o $PPP_IFACE -j MASQUERADE
iptables -D FORWARD -i br0 -o $PPP_IFACE -j ACCEPT
```

6. HostOS に kvm/libvirt/spice-client-gtk パッケージを導入しておきます。
7. HostOS にて GuestOS 用の kde-test.xml を以下の雛形で作成して virsh define kde-test.xml しておきます。[*1]

```
<domain type='kvm'>
  <name>kde-test</name>
  <memory>1048576</memory>
  <vcpu>1</vcpu>
  <os>
    <type arch='x86_64' machine='pc-1.0'>hvm</type>
    <boot dev='hd'/>
    <boot dev='cdrom'/>
    <bootmenu enable='yes'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='writeback'/>
      <source file='/var/lib/libvirt/images/kde-test.img'/>
      <target dev='vda' bus='virtio'/>
    </disk>
    <disk type='file' device='cdrom'>
      <driver name='qemu' type='raw'/>
<!-- directory of cdimage は適当に変更ください -->
      <source file='/directory of cdimage/debian-6.0.4-amd64-businesscard.iso'/>
      <target dev='hdc' bus='ide'/>
      <readonly/>
    </disk>
    <controller type='ide' index='0'/>
    <interface type='bridge'>
<!-- mac アドレスは適当に変更ください -->
      <mac address='52:54:00:31:cd:5a'/>
      <source bridge='br0'/>
      <model type='virtio'/>
    </interface>
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <graphics type='spice' port='5900' autoport='no'>
      <clipboard copypaste='yes'/>
    </graphics>
    <sound model='ac97'\>
    <video>
      <model type='qxl' vram='9216' heads='1'/>
    </video>
    <memballoon model='virtio'>
    </memballoon>
  </devices>
</domain>
```

8. HostOS で仮想環境用のディスクを 10GB ぐらいで作っておきます。

[*1] virt-install は何故か自分の experimental な環境では Segmentation Fault で落ちてしまうのでここでは使いません。BTS しときます。

```
qemu-img create -f raw /var/lib/libvirt/images/kde-test.img 10G
```

9. HostOS で KVM を起動して、 spice クライアントを接続します。

```
virsh start kde-test; spicy -h 127.0.0.1 -p 5900 &
```

10. GuestOS の Debian インストーラが起動したら、 TAB キーを押し、画面下に現れた編集可能な行に、”priority=medium” を以下のように入力してインストールを開始します。

```
 /install.amd/vmlinuz vga=788 initrd=/install.amd/initrd.gz --- quiet priority=medium
```

インストール途中「 Debian アーカイブのミラーを選択」のメニューにて”sid” を選択し、「 インストールするコンポーネント」として「 ssh サーバー」のみ (他は選択しない) とします。

11. インストールが完了すると、テキストコンソールから Debian sid な GuestOS へログインできるようになります。
12. GuestOS にログインして以下の行を/etc/apt/source.list へ付け加えます

```
#追加内容
deb http://ftp.jp.debian.org/debian/ experimental main
deb-src http://ftp.jp.debian.org/debian/ experimental main
```

13. GuestOS の/etc/apt/preference.d に Debian KDE チーム製の experimental パッケージ用 preference ファイルをインストールします。

```
cd /etc/apt/preference.d && wget http://pkg-kde.alioth.debian.org/files/kde-experimental
```

14. GuestOS で以下を実行し、 experimental な KDE 環境を一気に入れてしまいます。

```
aptitude update;aptitude aptitude install task-kde-desktop task-japanese-kde-desktop;aptitude clean
```

15. インストールが終わったら、 GuestOS をリブートします。 GuestOS で KDE の experimental 版が起動し、グラフィカルなログイン画面が現れます。

5.3 Debian と KDE 環境のバージョン

Debian のバージョンと KDE のバージョンの対応を表 1 に載せます。

Debian	stable	testing	unstable	experimental	upstream
KDE	4.4	4.6	4.6	4.7.4	4.8.0

表 1　Debian のバージョンと KDE のバージョン

KDE の upstream は 2012 年 1 月 25 日に 4.8.0 をリリースしたばかりなので、まだ experimental も追いついていない状態です。

5.4 KDE 環境の開発の特徴

KDE 環境の開発は以下のような特徴があります。

1. Qt（ キュート）ライブラリを使う。
2. C++ のコードが基本
3. autotools の代わりに cmake が使われる

このため、 Debian ではパッケージ開発の為に pkg-kde-tools パッケージが用意されています。

5.5 Debian での KDE 環境のパッケージ開発

Debian では KDE 環境のパッケージ開発用に pkg-kde-tools というパッケージを別に用意しています。こちらを導入すると KDE 環境のパッケージ構築の際に便利な機能が使えるようになります。

項番	拡張されるもの	拡張	備考
1	dh	–with kde	debhelper に kde 用の拡張を指定
2	dh_auto_*	–buildsystem=kde	dh_auto_*が cmake を使うようになる、 KDE 環境用の設定を行う等
3	CDBS	kde.mk	CDBS で KDE 用の拡張が利用できるようになる
4	その他	variables.mk など	debian/rules の中で$(DEB_CMAKE_KDE4_FLAGS) などが使える等

表 2　pkg-kde-tools をインストールした時の拡張

5.6 超簡易的に KDE 用プログラムの Debian パッケージを作ってみる

ここでは超簡易的に KDE 用プログラムの Debian パッケージを作ってみます。
まず、事前準備として、

- 環境は 5.2 章の experimental 環境を用意ください。
- 必要なパッケージ (cmake パッケージ等) [*2]

次に khello-1.0.0/なるディレクトリに `http://techbase.kde.org/Development/Tutorials/First_program` にある、 main.cpp と CMakeLists.txt を配置します。

```
$ cd khello-1.0.0
$ ls
CMakeLists.txt main.cpp
$
```

次に、オリジナルの tar.gz アーカイブを作成しておきます。

```
$ cd ..
$ tar czf khello_1.0.0.orig.tar.gz khello-1.0.0
$ ls -F
khello-1.0.0/  khello_1.0.0.orig.tar.gz
$
```

dh_make を使って debian/ディレクトリを仕込みます。あとは rules ファイル以外いつも通り、パッケージを作成するようにファイルを作成しておきます。

```
$ cd khello-1.0.0/debian
$ ls -F
README.Debian  changelog  control    docs   source/
README.source  compat     copyright  rules
$
```

pkg-kde-tools パッケージを利用する rules ファイルを記載します。

```
# pkg-kde-tools を使った KDE 開発用 debian/rules ファイルの中身。

%:
	dh $@ --with kde
```

あとは、 dpkg-buildpackage -uc -us -rfakeroot を実行してビルドします。

[*2] KDE 環境向けの開発が全く初めての人は、細かい事が判ってくるまで、 aptitude build-dep kdeutils しておいて KDE パッケージ開発に必要なパッケージをあらかじめまとめて導入しておくという手もあります

```
$ dpkg-buildpackage -us -uc -rfakeroot
dpkg-buildpackage: source package khello
... 中略...
dpkg-source: info: building khello in khello_1.0.0-1.debian.tar.gz
dpkg-source: info: building khello in khello_1.0.0-1.dsc
 debian/rules build
dh build  --with kde
   dh_testdir
   dh_auto_configure --buildsystem=kde
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
... 中略...
```

無事、–buildsystem=kde が利用され、 cmake が実行されています。

しばらく待つと無事に khello_1.0.0-1_amd64.deb などが出来上がります。

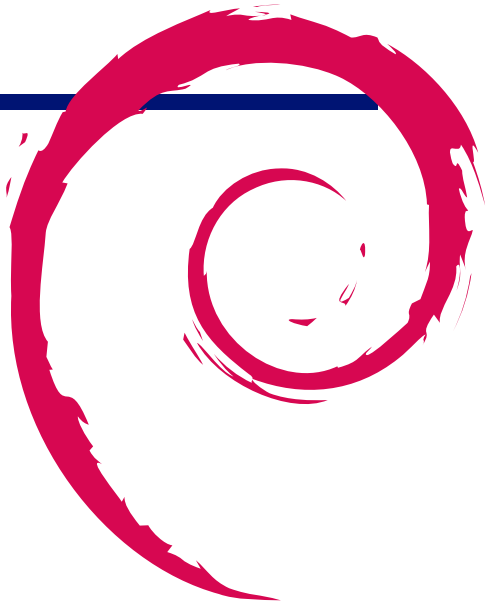
ほら、 pkg-kde-tools のおかげでパッ ケージ開発も簡単でしょ?でしょ?

5.7 おわりに

今回は、 Debian 開発者の為の KDE 環境の構築と、簡単なパッ ケージ作成について、一通り記載してみました。これを機に、 KDE 環境に関する開発をされる方が増えるとうれしいと思っ ています。

5.8 参考文献

- `http://pkg-kde.alioth.debian.org/` Debian KDE Team のホームページ。
- `http://techbase.kde.org` KDE Techbase
- `http://kde.org/` KDE 本家
- `http://www.spice-space.org/` SPICE 仮想デスクトッ プデバイス本家



6 月刊 Debhelper

山本 浩之

6.1 パッケージの make の前に…

先月までに学んできたように、 Debhelper は、基本的にビルドに必要な一連の dh_ほげほげコマンドを自動実行します。

例えば Debhelper は、 dh_auto_configure というコマンドを提供しており、これはご想像の通り、

```
./configure --build='dpkg_architecture_value("DEB_BUILD_GNU_TYPE")' --prefix=/usr --includedir=/usr/include \
--mandir=/usr/share/man --infodir=/usr/share/info --sysconfdir=/etc --localstatedir=/var \
--libdir=/usr/lib/\$multiarch --libexecdir=/usr/lib/\$multiarch --disable-maintainer-mode \
--disable-dependency-tracking --host='dpkg_architecture_value("DEB_HOST_GNU_TYPE")
```

をしているだけです。

しかし、メンテナによって autotools を利用し、 confugure スクリプトをビルドの度に毎回 configure.ac から生成したい人もいるでしょうし、もしかすると Makefile の元となる Makefile.in だって、毎回 Makefile.am から生成したい人もいるでしょう。また、 Debian パッケージオリジナルのパッチをあててパッケージを作るのは、ごく当たり前のように行なわれています。

そこで今月は、一連の dh_ほげほげコマンドへの追加の仕組みと、その例として、 dpatch パッケージで提供される dh_dpatch_patch コマンドと、 autotools-dev パッケージで提供される dh_autotools-dev_updateconfig コマンドの追加について解説しましょう。

6.2 一連の dh_ほげほげコマンドへの追加の仕組み

基本となる一連の dh_ほげほげコマンドは、大元のコマンドである dh スクリプトに記述してあり、すべてについては先月や先々月に話されているので、割愛します。

特に make 直前に実行されるものは、

```
dh_testdir  #カレントディレクトリの確認
dh_auto_configure  #./configure の実行
```

だけです。

勿論、これだけではパッチもあてられないですし、 configure スクリプトの再生成もできません。

そこで dh スクリプトは、一連の dh_ほげほげコマンドを列挙する配列にし、これを$sequences$sequence のスカラ変数として保持しています。 (この辺は perl にあまり詳しくないので、ちょっと間違っているかも…)

また、この$sequences から dh_ほげほげコマンドを除いたり (remove_command)、ある dh_ほげほげコマンドの前に追加する (insert_before) サブルーチンも用意されています。

dh スクリプトは、/usr/share/perl5/Debian/Debhelper/Sequence/ディレクトリのなかにあるファイルを参照しており、ここに

```
insert_before("dh_auto_configure", "dh_追加ほげほげ")
```

という記述のあるファイル (アドオン) が追加されると、 dh_auto_configure の前に dh_追加ほげほげコマンドが実行されるようになります。

ただし、 rules の

```
dh $@ --with 〜
```

でアドオンが指定されない限り評価はされないので、ビルドに不必要な dh_ほげほげコマンドを入れていても大丈夫なはずです。

6.3 dh_dpatch_patch コマンド

dpatch パッケージをインストールすると、/usr/share/perl5/Debian/Debhelper/Sequence/ディレクトリに dpatch.pm ファイルが入り、これには、

```
insert_before("dh_auto_configure", "dh_dpatch_patch")
insert_before("dh_clean", "dh_dpatch_unpatch")
```

という記述があります。これは見て想像できるとおり、 make の直前に実行される./configure のさらに直前に、dh_dpatch_patch を加えています。また、下の記述は、以前にビルドしたことがある場合に、パッチする前の状態にする dh_dpatch_unpatch コマンドも追加されています。

この dh_dpatch_patch は、パッケージソースの debian/patches/00list に記述されたファイル名のパッチを先頭からパッチするコマンド、 dpatch スクリプトを実行します。

すなわち、もし configure スクリプトに dpatch でなんらかのパッチをあてて実行したければ、 dpatch パッケージを Build-dep し、 debian/patches/ディレクトリにパッチファイルとそれに合わせた 00list ファイルを用意し、

```
dh $@ --with dpatch
```

と rules ファイルに記述しておけば良いはずです。

make だけしたいならば、ターミナルで、

```
$ dh_dpatch_patch
$ ./configure 〜
$ make
```

でも大丈夫です。

6.4 autotools だって使いたい

ビルドするマシンの環境に合わせて、 configure スクリプトなどを調整してくれるツールとして、 GNU autotools というものがあります。次にこの GNU autotools を Debhelper で利用する方法について述べましょう。

dh-autoreconf パッケージをインストールすると、依存関係で automake、 autoconf と、 automake に依存して autotools-dev がインストールされます。 autotools-dev パッケージは/usr/share/perl5/Debian/Debhelper/Sequence/ディレクトリに autotools-dev.pm ファイルが入ります。これには、

```
insert_before("dh_auto_configure", "dh_autotool-dev_updateconfig")
insert_before("dh_clean", "dh_autotool-dev_restoreconfig")
```

と記述されています。

dh_autotool-dev_updateconfig コマンドは、カレントディレクトリ以下で実行しているシステムタイプの標準名を推測するための config.guess と config.sub ファイルを探し、 config.guess.dh-orig と config.sub.dh-orig ファイルに名前を換え、それぞれ/usr/share/misc/ディレクトリにある最新 autotool-dev の config.guess と config.sub をコピーしてき

ます。

dh-autoreconf パッケージは/usr/share/perl5/Debian/Debhelper/Sequence/ディレクトリに autoreconf.pm ファイルが入ります。これには、

```
insert_before("dh_auto_configure", "dh_autoreconf")
insert_before("dh_clean", "dh_autoreconf_clean")
```

と記述されています。

dh_autoreconf コマンドは、簡単に言うと、 automake、 autoconf をしてくれる autoreconf スクリプトを呼び出し、configure や Makefile.in を再生成します。

dh_autoreconf を使いたいときは、このパッケージに Build-dep し、

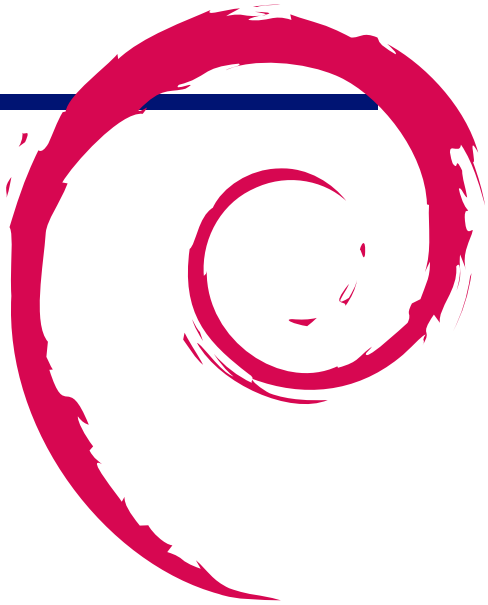
```
dh $@ --with autoreconf
```

と rules ファイルに記述しておけば良いはずです。 debian/autoreconf ファイルにディレクトリのリストがあれば、そこだけ configure や Makefile.in を更新してくれます。

autoreconf.pm ファイルでは「 dh_auto_configure より前だよ」という指定しかありませんから、 dh_autoreconf が実行されてパッチがあてられるのか、パッチがあてられてから dh_autoreconf が実行されるのかまでは記述されていません。 dh スクリプトを見た限りでは「 –with」オプションの順でリストされているようです。つまり、 dh-autoreconf パッケージを利用するソースパッケージの Makefile に対して BTS でパッチを書く場合は、「 –with」オプションの順を確認する必要がありそうです。

6.5 おわりに

今回は make を実行する前に使用される configure スクリプトなどの、 Debhelper を使ったカスタマイズ法について、駆け足で説明してみました。



7 cmake 使ってみる

野島 貴英

7.1 cmake とは

KDE 環境の開発に使われているツールに cmake があります。これは従来の autotools のようなものです。が、autotools に比べて次に述べる代表的な特徴があります。

- バイナリのプログラムである
  autotools は、ご存知の通り、中は sh スクリプトとなっています。これは/bin/sh を基本コマンドとして持つ従来の UNIX 系の OS で使うなら非常に都合がよいのですが、そもそも/bin/sh を持たないシステムの元で利用しようとすると動作できません。これでは、例えば、標準的な C プログラムをコンパイル出来る環境なのに、/bin/sh が無いという本質ではない理由の為に移植性を損なうのはちょっと残念です。
  cmake はバイナリのプログラムなので、コマンド単体で動作することができ、/bin/sh など UNIX のコマンドが無い場所でも問題なく動作できます。
- 様々なプラットフォーム用の構築システムに対応できる
  autotools は make に特化したツールとなります。ここで、そもそも Makefile が一般的ではない開発環境(例: Microsoft Visual Studio 等の様々な IDE) の場合、 Makefile よりも IDE のプロジェクトファイルを生成できた方がより都合がよかったりします。 cmake は一本の CMakeLists.txt を用意するだけで、 Makefile や、 IDE 環境用のプロジェクトファイルを生成できたりする能力があります。この為、 autotools を利用したソースパッケージのように、 Makefile.am と、例えば.vcproj ファイルを別々に修正して UNIX/Windows 間の移植性を保つというような作業から開発者が開放される可能性を意味します。
- その他
  詳しいサマリは、 DDJ ジャーナルの `http://drdobbs.com/cpp/184405251` にサマリされているような機能がある模様です(まだ自分は未評価です。)
  この記事からいくつか抜粋すると、
  - QT ライブラリの moc コマンド/ITK の CABLE/VTK のラッパー生成コマンドに対応したステートメント
  - 静的ライブラリ、動的ライブラリの生成を容易に切り替えれるようにする機能
  - ファイルの依存関係の自動生成、並列ビルドのサポート

  がある模様です。

7.2 使ってみる

百聞は一見にしかずなので、ちょっと使ってみます。
cmake パッケージをシステムに導入します。

```
aptitude install cmake
```

次に以下のソース (hello.c,config.h.in) を用意します。

```
/*hello.c*/
#include <stdio.h>
#include "config.h"
int main(int argc,char **argv)
{
        printf("hello world\n");
#if defined(HAVE_EXIT)
        printf("yes, this system has exit()\n");
#endif
        return(0);
}
```

```
/*config.h.in*/
#cmakedefine HAVE_EXIT
```

次に、 CMakeLists.txt を用意します。

```
# cmake のバージョンは 2.8 以上
cmake_minimum_required(VERSION 2.8)
# project の名前を宣言
project(hello)

# cmake 提供のマクロをロードする。ここでは関数がシステムにあるかを確かめるマクロ
# を使ってみる。
include (${CMAKE_ROOT}/Modules/CheckFunctionExists.cmake)

# exit() 関数をチェックしてみる。あれば HAVE_EXIT を定義せよという意味。
check_function_exists(exit HAVE_EXIT)

configure_file (
  "${PROJECT_SOURCE_DIR}/config.h.in"
  "${PROJECT_BINARY_DIR}/config.h"
)
# cc -I に何指定するか
include_directories ("${PROJECT_BINARY_DIR}")

# hello は hello.c から出来るという事を指定
add_executable(hello hello.c)
```

これら 3 つのファイルを hello-src/以下に配置します。

```
$ ls -lR
.:
合計 4
drwxr-xr-x 2 nojima nojima 4096  2 月 17 03:15 hello-src

./hello-src:
合計 8
-rw-r--r-- 1 nojima nojima 46  2 月 17 03:15 CMakeLists.txt
-rw-r--r-- 1 nojima nojima  34  2 月 17 04:21 config.h.in
-rw-r--r-- 1 nojima nojima 91  2 月 17 03:10 hello.c
$
```

今回はビルド用ディレクトリ (hello-build) を作り、移動します。

```
$ ls
hello-src
$ mkdir hello-build
$ cd hello-build
```

cmake を実行します。

```
$ cmake ../hello-src
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
... 中略...
-- Looking for exit
-- Looking for exit - found
-- Configuring done
-- Generating done
-- Build files have been written to: /.../cmake-test/hello-build
$ ls
CMakeCache.txt  CMakeFiles  Makefile  cmake_install.cmake  config.h
```

自動的に環境チェックが行われ Makefile/config.h が出来上がります。 exit 関数も見つかったとの表示が行われまし

た。ここで make してみます。

```
$ make
Scanning dependencies of target hello
[100%] Building C object CMakeFiles/hello.dir/hello.c.o
Linking C executable hello
[100%] Built target hello
$ ls -F
CMakeCache.txt  CMakeFiles/  Makefile  cmake_install.cmake  config.h  hello*
$ ./hello
hello world
yes, this system has exit()
$
```

CMakeLists.txt から無事に実行バイナリ (hello) が出来上がりました。また、 defined(HAVE_EXIT) も True となり、 exit() 関数がある時のコードもコンパイルされています。

7.3 IDE 用のプロジェクトファイルを生成してみる

cmake を引数無しで実行すると、 help が出てきます。このヘルプの文章の中に、どんな IDE 用のプロジェクトファイルを生成できるかについて説明があります。試しに手元の Debian マシンで実行すると、

```
$ cmake
... 中略..
The following generators are available on this platform:
  Unix Makefiles               = Generates standard UNIX makefiles.
  CodeBlocks - Unix Makefiles = Generates CodeBlocks project files.
  Eclipse CDT4 - Unix Makefiles
                               = Generates Eclipse CDT 4.0 project files.
  KDevelop3                    = Generates KDevelop 3 project files.
  KDevelop3 - Unix Makefiles   = Generates KDevelop 3 project files.
$
```

ここでは試しに先ほどの hello-build ディレクトリ以下で KDevelp3 project ファイルを生成してみます。

```
$ cmake -G KDevelop3 ../hello-src
... 中略...
$ ls
MakeCache.txt  Makefile             config.h        hello.kdevelop.filelist
CMakeFiles     cmake_install.cmake  hello.kdevelop  hello.kdevses
$
```

確かに KDevelp3 用のプロジェクトファイル (hello.kdevelop 等) が生成されています。

7.4 おわりに

cmake は KDE の他にも mysql でも採用されています。また、 wikipedia(`http://ja.wikipedia.org/wiki/CMake`) によれば、利用しているアプリケーションも続々増えている模様です。

使いこなせると強力なツールとなりそうな感じです。皆さんも使ってみてはいかがでしょうか? Debian なら aptitude で簡単に導入できますので、是非試してみてください。

7.5 参考文献

- `http://www.cmake.org/` cmake 本家
- `http://www.cmake.org/cmake/help/cmake_tutorial.html` cmake チュートリアル
- `http://drdobbs.com/cpp/184405251?pgno=1` DDJ ジャーナルの記事

Debian 勉強会資料

2012 年 2 月 18 日　　初版第 1 刷発行

東京エリア Debian 勉強会（ 編集・ 印刷・ 発行）