東京エリア Debian 勉強会

第 98 回 2013 年 3 月度

吉田俊輔
koedoyoshida@gmail.com

2013 年 3 月 16 日

設営準備にご協力ください。

会場設営よろしくおねがいします。

Agenda

- 注意事項
  - トイレはエレベーターホール反対側にあります。
  - 飲料は手動販売機があります。ゴミ箱はトイレの手前給湯室にあります。
  - 遅れてきた人は次に遅れてきた人を迎えに行ってください。
- 最近あった Debian 関連のイベント報告
  - 第 96 回 東京エリア Debian 勉強会
  - OSC 浜松
  - OSC 東京
- Debian Trivia Quiz
- 事前課題紹介
- ldapvi & python-ldap で stress-free life
- 月刊 Debhelper
- gdb python 拡張その 1

イベント報告

第96回 東京エリアDebian勉強会

- 開催場所は新宿のスクエアエニックスさんでした。
- Debian 勉強会アンケート
- 2013-2015 年を妄想する
- 月刊 Debhelper

がありました。

Open Source Conference 2013 Hamamatsu

- 開催場所は浜松市市民協働センターさんでした。
- 展示のみの出展で吉田、赤部さんで行いました。

Open Source Conference 2013 Tokyo/Spring

- 開催場所は明星大学さんでした。
- セミナーは土曜日に「Debian Update」のタイトルで野島さんが行いました。
- 展示は土曜のみ出展で吉田、やまねさん、杉本さんなどで行いました。

セミナー資料は下記で公開されています。
http://www.ospn.jp/osc2013-spring/modules/article/article.php?articleid=5
http://tokyodebian.alioth.debian.org/2013-02.html
やまねさんのリポートが下記にあります。
http://henrich-on-debian.blogspot.jp/2013/02/open-source-conference-2013-tokyospring.html

DWN quiz

Debian 常識クイズ

Debian の常識、もちろん知ってますよね? 知らないなんて恥ずかしくて、知らないとは言えないあんなことやこんなこと、みんなで確認してみましょう。
今回の出題範囲は
debian-devel-announce@lists.debian.org,
debian-devel@lists.debian.org に投稿された内容と
Debian Project News などからです。

問題1. DebConf13 の開催地と開催日は?

-  A 日本 東京都 6 月 29 日
-  B ニカラグア マナグア 7 月 8-14 日
-  C スイス ヴォーマルキュ 8 月 11-18 日

問題1. DebConf13 の開催地と開催日は?

-  A 日本 東京都 6 月 29 日
-  B ニカラ
  14 日
-  C スイス
  11-18 日

答えは:

C

問題1. DebConf13 の開催地と開催日は?

解説: ニカラグアは DebConf12 の開催地です。 DebConf13 はスイスのキャンプ地で開催です。 6/29 は皆さん予定を空けておきましょう。

問題2. 世界のWebサーバで最も人気のあるLinuxディストリビューション(W3Techs調べ)は?

-  A CentOS
-  B Debian
-  C Ubuntu

問題2. 世界のWebサーバで最も人気のあるLinuxディストリビューション(W3Techs調べ)は?

-  A CentOS
-  B Debian
-  C Ubuntu

答えは:

B

問題2. 世界のWebサーバで最も人気のあるLinuxディストリビューション(W3Techs調べ)は?

解説: http://w3techs.com/technologies/history_details/os-linux に結果のグラフがあります。現在 Linux を使用している web サーバの 32.9% が Debian を利用しており、その割合は現在も増加を続けているそうです。

問題3. 現在Debian プロジェクトリーダ選挙2013(Debian Project Leader Elections 2013)が開催されています。現在(3/16)のステータスは?

- 
  A 指名期間
  Nomination period
- 
  B 選挙運動期間
  Campaigning period
- 
  C 投票期間
  Voting period

問題3. 現在Debian プロジェクトリーダ選挙2013(Debian Project Leader Elections 2013)が開催されています。現在(3/16)のステータスは?

-  A 指名期間
  Nomination period
-  B 選挙運動
  Campaigning
-  C 投票期間
  Voting period

答えは:

B

問題3. 現在Debian プロジェクトリーダ選挙2013(Debian Project Leader Elections 2013)が開催されています。現在(3/16)のステータスは?

解説: 指名期間:2013 年 3 月 3 日 (日) 00:00:00 UTC ～
3 月 9 日 (土) 23:59:59 UTC
選挙運動期間:2013 年 3 月 10 日 (日) 00:00:00 UTC ～
3 月 30 日 (土) 23:59:59 UTC
投票期間:2013 年 3 月 31 日 (日) 00:00:00 UTC ～
4 月 13 日 (土) 23:59:59 UTC
http://www.debian.org/vote/2013/vote_001

問題4. Ben Hutchings さんが次期 Debian 安定版と一緒に出荷される Linux カーネルに (3.2 系列の mainline には無い) 追加機能が搭載される予定であると述べています。多くの追加点の中に含まれないものは何?

-  A リアルタイム性を強化する PREEMPT_RT
-  B Hyper-V guest drivers 強化
-  C TCP 接続高速化技術 TCP Fast Open

問題4. Ben Hutchings さんが次期 Debian 安定版と一緒に出荷される Linux カーネルに (3.2 系列の mainline には無い) 追加機能が搭載される予定であると述べています。多くの追加点の中に含まれないものは何?

答えは:

- A リアルタイム性を強化する PREEMPT RT

- B Hyper-V 強化

- C TCP 接続 TCP Fast

C

問題4. Ben Hutchings さんが次期 Debian 安定版と一緒に出荷される Linux カーネルに (3.2 系列の mainline には無い) 追加機能が搭載される予定であると述べています。多くの追加点の中に含まれないものは何?

解説: Ben Hutchings さんは Debian カーネルチームのメンバーであり、 kernel.org の 3.2.y 安定版系列のメンテナ。 PREEMPT_RT は I/O の遅延と揺らぎを低減する audio/video アプリケーション利用等向、 linux-image-rt-amd64 , linux-image-rt-686-pae で使用可能。 Hyper-V guest drivers は 3.2 にも含まれていますが、より改善された 3.4 からの修正を導入。 TCP Fast Open(TFO) は TCP の 3way-handshake を改善し、 2 回目以降の通信開始時に SYN と data を含んだパケットを送ることで高速化する技術、 3.6 時点ではクライアント側のみ実装、サーバ側は 3.7 から。

問題5. Wookey さんがアナウンスした alpha 版の Debian port ARM64 image は?

-  A Debian/Ubuntu port image
-  B Debian/KFreeBSD port image
-  C Debian/GnuHurd port image

問題5. Wookey さんがアナウンスした alpha 版の Debian port ARM64 image は?

-  A Debian/Ubuntu port image
-  B Debian/ port image
-  C Debian/ port image

答えは:

A

問題5. Wookey さんがアナウンスした alpha 版の Debian port ARM64 image は?

解説: x86 環境を使用しない self-bootstrap image です。`http://wiki.debian.org/Arm64Port` でステータスが確認できます。 mainline kernel の ARM64/AArch64(ARMv8 アーキテクチャ:Cortex A57 や Cortex A53 と呼ばれる予定の CPU 向け) 対応は 3.7 から。

問題6. 700,000番目のバグが報告された日を当てる700000thBugContestの結果が出ました。その予想日と対象バグの報告日は?

- 
  A 予想日:2013/02/04、
  報告日:2013/02/14
- 
  B 予想日:2013/02/07、
  報告日:2013/02/14
- 
  C 予想日:2013/02/14、
  報告日:2013/02/07

問題6. 700,000番目のバグが報告された日を当てる700000thBugContestの結果が出ました。その予想日と対象バグの報告日は?

- 
  A 予想日:2013/02/04、
  報告日:2013/02/14
- 
  B 予想日:
  報告日:
- 
  C 予想日:
  報告日:

答えは:

C

問題6. 700,000番目のバグが報告された日を当てる700000thBugContestの結果が出ました。その予想日と対象バグの報告日は?

解説: 最も近い2013/02/14を予想したChristian Perrierさんが当てました。結果は`http://wiki.debian.org/700000thBugContest`で公開されています。また、800,000/1,000,000番目のバグが報告される日を当てるコンテスト`http://wiki.debian.org/800000thBugContest`も開催されています。

問題7. master.debian.orgが新しい機械に移行されました。これは何のサーバでしょうか？

-  A @debian.orgのメールサーバ
-  B パッケージのマスターサーバ
-  C パッケージのスポンサー(mentor)を探すサーバ

問題7. master.debian.orgが新しい機械に移行されました。これは何のサーバでしょうか？

答えは:

-  A @debian.org のメールサーバ
-  B パッケー
-  C パッケ (mentor) そ

A

問題7. master.debian.orgが新しい機械に移行されました。これは何のサーバでしょうか？

解説: 古いサーバはディスク障害等があったので、寿命と判断され、データが損失する前に新しいサーバに移行されました。ftp-master.debian.orgはDebianのofficial packageリポジトリです。パッケージのスポンサー(mentor)を探すのはmentors.debian.net。

問題8. pbuilder でビルドする際に gcc の代替として Clang を使いやすくするパッチが追加されました。誰が書いたパッチでしょうか?

-  A Sylvestre Ledru
-  B Junichi Uekawa
-  C Hideki Yamane

問題8. pbuilder でビルドする際に gcc の代替として Clang を使いやすくするパッチが追加されました。誰が書いたパッチでしょうか?

-  A Sylvestre Ledru
-  B Junichi
-  C Hideki Y

答えは:

C

問題8. pbuilder でビルドする際に gcc の代替として Clang を使いやすくするパッチが追加されました。誰が書いたパッチでしょうか?

解説: Sylvestre Ledru さんは Clang のメンテナーの一人です。彼は Clang 3.2 を使用してアーカイブをリビルドした結果について報告しました (12.1% が失敗、つまり 18264packages 中、 2204 が失敗)。失敗したパッケージの情報 `http://clang.debian.net/pts.php`。 Debian の Clang サポートは着々と進んでいます。

問題9. DPN - 2013年3月4日号に取り上げられた日本のイベントは

-  A Open Source Conference 2013 Tokyo/Spring
-  B Open Source Conference 2013 Hamamatsu
-  C Open Source Conference 2013 Tokushima

問題9. DPN - 2013年3月4日号に取り上げられた日本のイベントは

-  A Open Source Conference 2013 Tokyo/Spring
-  B Open 2013 Ham
-  C Open 2013 Toku

答えは:

A

問題9. DPN - 2013年3月4日号に取り上げられた日本のイベントは

解説: http://henrich-on-debian.blogspot.jp/2013/02/open-source-conference-2013-tokyospring.html 詳細は後ほど。

事前課題

dictoss(杉本　典充)

DDebian でここがバグってるかもという点は?

squeeze の virt-manager。 KVM 上で動作ている仮想マシンのコンソールを操作しているとき、キーボード操作でアンダースコアが入力できない。

使ったことのある/使ってみたいデバッガは?

gdb と pdb(python) は使ったことがあります。コマンド単独で実行するより、 Emacs の GUD モード内で動作させたほうがソースコードを見つつデバッグできるので効率がいいです。(eclipse を使ったときのような感じ)。 ssh しつつデバッグするときは GUI がないので必要に迫られて覚えた記憶があります。

LDAP の管理していますか? 何を使ってますか?

LDAP システムの管理はしたことはありません。

キタハラ

Debian でここがバグってるかもという点は?
今、ぱっと思いつかない。
使ったことのある/使ってみたいデバッガは?
最近、 prinf とか syslog 程度で済むプログラムしか作っていないなぁ。
LDAP の管理していますか? 何を使ってますか?
していない。(実は大昔に、 Netscape の Directory Server を・・・)

まえだこうへい

Debian でここがバグってるかもという点は?

Debian installer では LVM を使っているとパーティションを削除できず、 shell モードから消してましたが、今ってどうなんでしょう。

使ったことのある/使ってみたいデバッガは?

ちょっと gdb, pdb など使ったことありますが、ちゃんと使ったがないのでそろそろ学習せんとあかんかなぁと思ってます。

LDAP の管理していますか? 何を使ってますか?

今回の発表内容なので割愛

たかはしもとのぶ

Debian でここがバグってるかもという点は?
うーん、あまり思いつかないです。パッケージの設定が適切でないと思うことはありますが。
使ったことのある/使ってみたいデバッガは?
gdb もしくは Emacs + gdb を使ったことがあります。
LDAP の管理していますか? 何を使ってますか?
管理しています。ファイルベースの方が扱いやすいので、どうしてもという場合以外は slapd.conf にしています。

yamamoto

Debian でここがバグってるかもという点は?

aptitude (armehf と ppc64 では再現しているが、他のは大丈夫みたい?) #659341 なのだろうが、パッチを書けるほどは理解していない。

ghc (ppc64 だけかも知れないが、確認不可。多分 7.6.1-2 の頃からだと思う) なんか特定のパッケージの特定位置で、ビルド中に固まる。固まった時は buildd ユーザで、 ssh と gcc のゾンビプロセスがいつも残っている。

libccid 日本では結構有名な NTTCom のカードリーダに関する typo。 1.4.6 でサポートされなくなり、 1.4.8 で復活したが、その時 enbug した。最近 wheezy に上げて気づいたが、 BTS しないとと思いながらも、多忙によりまだ投げられてない。

使ったことのある/使ってみたいデバッガは?

バグの状況説明で gdb で bt したことがあるぐらい。

LDAP の管理していますか? 何を使ってますか?

LDAP って、それおいしいの?

野島　貴英

Debian でここがバグってるかもという点は?

sid/experimental 使ってるとそりゃもう遭遇します(←そのためのものですから無問題ですが）。

今も見る例: aptitude が LANG=ja_JP.utf8 でこける事がある(再現条件不明）、 gstreamer-plugins で字幕が使えない(upstream の問題? ）、 gnome-shell がたまにハング(upstream の問題? ）、 gnome-keyring がまれにハング(upstream の問題? ）などなど。ただそれでも、結局 Debian の問題じゃなくて、 upstream の問題のようなものばかり...

※ Bug 報告しろよという話もある。

使ったことのある/使ってみたいデバッガは?

printf(笑)/gdb/perldb/pydb/adb/apd/xdebug などなど。

LDAP の管理していますか? 何を使ってますか?

秘密。 slapd.conf 直接編集かな。

理由: そんなに変更しないので、他の手段を使ってないだけだったりという消極的理由。

koedoyoshida

Debian でここがバグってるかもという点は?
debug イメージが標準で提供されていないこと。
使ったことのある/使ってみたいデバッガは?

→使ったことがある crash,lkst,gdb

→使ってみたい VisualStudio 並みにグラフィカルなデバッグ環境

LDAP の管理していますか? 何を使ってますか?
過去の遺産の slapd.conf のシステムが...

鈴木崇文

Debian でここがバグってるかもという点は?
あまりバグってると感じたことはないです。
使ったことのある/使ってみたいデバッガは?
gdb を使用したことがあります。 emacs+gdb で使いやすい環境を作れたら使ってみたいです。統合開発環境としての emacs+gdb 連携ベストプラクティス的なものに興味があります。 vim+gdb も試したことがありますが、結局 gdb を直接使うようになってしまいました。
LDAP の管理していますか? 何を使ってますか?
LDAP 使ったことないですね。

ldapvi & python-ldap で stress-free life

月刊 Deb-helper

今月のコマンド

- dh_auto_install
- dh_install

debian パッケージ構築、全体の流れ

1. パッケージビルド環境を構築する
2. 不要なファイルを削除する
3. バイナリパッケージに格納するファイルをビルドする
4. ビルドしたファイルをバイナリパッケージにまとめる
5. .changes ファイルを作成する
6. パッケージに署名する

2011 年 10 月勉強会資料より

4. ビルドしたファイルをバイナリパッケージにまとめる

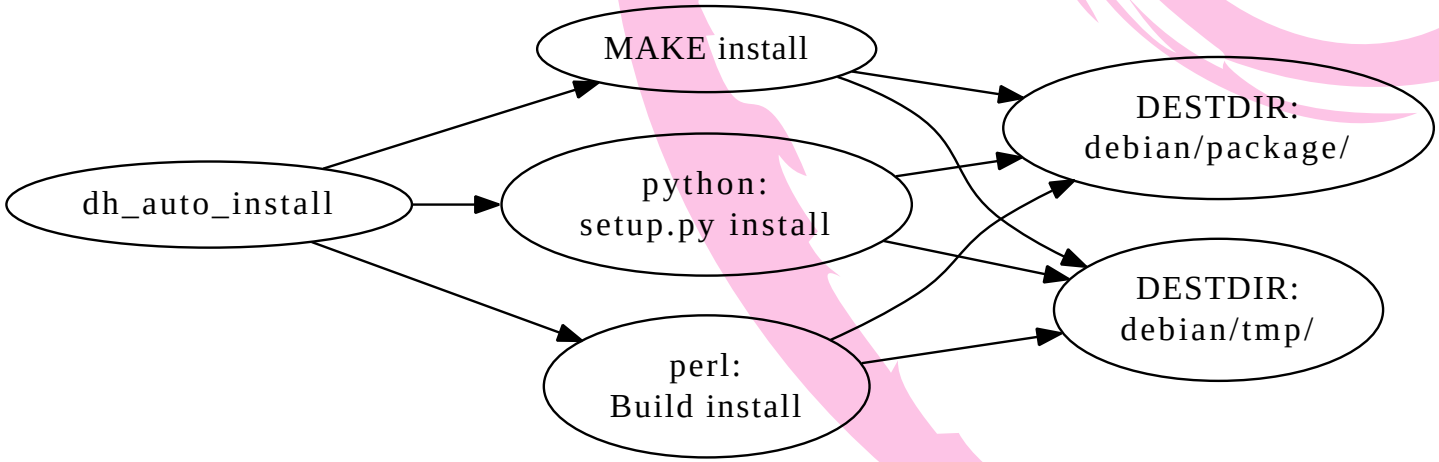
- 以下の debhelper コマンドが実行されます。

```
dh_testdir -> dh_auto_configure -> dh_auto_build -> dh_auto_test
-> dh_testroot -> dh_prep -> dh_installdirs -> dh_auto_install
-> dh_install -> dh_installdocs -> dh_installchangelogs
-> dh_installexamples -> dh_installman -> dh_installcatalogs
-> dh_installcron -> dh_installdebconf -> dh_installemacsen
-> dh_installifupdown -> dh_installinfo -> dh_pysupport
-> dh_installinit -> dh_installmenu -> dh_installmime
-> dh_installmodules -> dh_installlogcheck -> dh_installlogrotate
-> dh_installpam -> dh_installppp -> dh_installudev -> dh_installwm
-> dh_installxfonts -> dh_installgsettings -> dh_bugfiles -> dh_ucf
-> dh_lintian -> dh_gconf -> dh_icons -> dh_perl -> dh_usrlocal
-> dh_link -> dh_compress -> dh_fixperms -> dh_strip -> dh_makeshlibs
-> dh_shlibdeps -> dh_installdeb -> dh_gencontrol -> dh_md5sums
-> dh_builddeb
```

2011 年 10 月勉強会資料より

dh_auto_install 動作解説 1

dh_auto_install は debhelper のプログラムです。自動的にファイルをパッケージ作成用のディレクトリにインストールします。 dh_auto_install は upstearm 等の Makefile の install ターゲット,setup.py, Build.PL のを使用して、インストール先はシングルバイナリ (only one binary package) であれば、debian/package/以下です。 multiple binary package の場合は debian/tmp/以下になり、その後 dh_install で適切なディレクトリに移動されます。



dh_auto_install 動作解説2

- 条件 1:Makefile(または setup.py や Build.PL) が GNU の慣例に準拠し、$(DESTDIR) 変数をサポートしていること。

`http://www.gnu.org/prep/standards/html_node/DESTDIR.html#DESTDIR`

Makefile ファイルを変更する必要があるなら、これら $(DESTDIR) 変数をサポートするように注意しましょう。

- 条件 2:インストール先の指定内容が Filesystem Hierarchy Standard (FHS) に準拠していること。

`http://www.debian.or.jp/community/devel/debian-policy-ja/policy.ja.html/ch-opersys.html`

dh_auto_install 動作解説 3

通常プログラムのビルドに使われている make 等を使って実際のインストール先のかわりに、一時ディレクトリーの下に作成されたファイルツリーのイメージへプログラムをインストール (コピー) する。普通のプログラムインストールと Debian パッケージ作成というこれら二つの違いには、 debhelper パッケージの dh_auto_configure と dh_auto_install のコマンドを使うことで (前述の条件を守っていれば、特に意識をせずに) 対応できるはずです。 GNU autoconf を使っているプログラムは、自動的に GNU 規約に準拠するので、そのパッケージ作成は簡単にできる (はず)。 `http://www.debian.org/doc/manuals/maint-guide/modify.ja.html`

dh_auto_install 動作解説 3

マルチパッケージで、 DESTDIR を使っていないので override で対応する例 (wide-dhcpv6

```
override_dh_auto_install:
        $(MAKE) prefix=$(CURDIR)/debian/tmp/usr install
```

この後、 dh_install で各パッケージに振り分け (後述)

dh_install 動作概要

dh_install はパッケージ構造ディレクトリーへインストールするファイルを扱う debhelper プログラムです。これ以外に多くの dh_install*コマンドが存在します。説明書 (documentation), サンプル (examples), マニュアルページ (man pages) のような特定のタイプのファイルのインストールにはするにはそれら専用のプログラムの方が向いています。

dh_install*コマンド(debhelper内)

コマンド	行数
dh_installcatalogs - install and register SGML Catalogs	126
dh_installchangelogs - install changelogs into package build directories	181
dh_installcron - install cron scripts into etc/cron.*	87
dh_installdeb - install files into the DEBIAN directory	118
dh_installdebconf - install files used by debconf in package build directories	136
dh_installdirs - create subdirectories in package build directories	96
dh_installdocs - install documentation into package build directories	311
dh_installemacsen - register an Emacs add on package	134
dh_installexamples - install example files into package build directories	116
dh_installifupdown - install if-up and if-down hooks	79
dh_installinfo - install info files	87
dh_installinit - install init scripts and/or upstart jobs into package build directories	287
dh_installlogcheck - install logcheck rulefiles into etc/logcheck/	76
dh_installlogrotate - install logrotate config files	60
dh_installman - install man pages into package build directories	268
dh_installmanpages - old-style man page installer (deprecated)	207
dh_installmenu - install Debian menu files into package build directories	99
dh_installmime - install mime files into package build directories	105
dh_installmodules - register modules with modutils	134
dh_installpam - install pam support files	69
dh_installppp - install ppp ip-up and ip-down files	75
dh_installtex - register Type 1 fonts, hyphenation patterns, or formats with TeX	664
dh_installudev - install udev rules files	125
dh_installwm - register a window manager	118
dh_installxfonts - register X fonts	97

dh_install 動作概要 2

二つの使用方法があります。

- upstearm の Makefile がインストールを行ってくれないとき、適所へそれらをコピーさせるために使用する。
- 複数のバイナリパッケージを構築するラージ・パッケージを構築するとき

ラージ・ パッケージの構築

(dh_auto_install または override_dh_auto_install 等を使用して)debian/tmp へすべてインストール。そこから適切なパッケージディレクトリーを構築するためにディレクトリーやファイルを dh_install を使用してコピーすることができます。debhelper 互換性レベル 7 から、カレント・ ディレクトリ (あるいは–sourcedir オプションで指定したディレクトリ) に対象が無ければ、 dh_install は debian/tmp をコピー元に使用します。

dh_install 動作概要 3

debian/package.install に各パッケージへインストールするべきファイル、およびそれらがインストールされるべきディレクトリーを記載します。フォーマットは、行単位でインストールするべきファイル (複数可) をリストし、行の末尾にそれがインストールされるべきディレクトリーを記載します。
要するに cp コマンドの引数です。
実際に dh_install 内部では cp コマンドが使用されています。

debian/package.install の例

```
$ cat wide-dhcpv6-client.install
usr/sbin/dhcp6c
usr/sbin/dhcp6ctl
debian/dhcp6c.conf etc/wide-dhcpv6
debian/scripts/dhcp6c-script etc/wide-dhcpv6
debian/scripts/dhcp6c-ifupdown etc/wide-dhcpv6
```

明示的なコピー先なしで、一行に１つのファイル名あるいはワイルドカード・パターンを単独で記載すると、dh_install が自動的に使用する目的地を推測する。これは–autodest オプションの動作と同様。

–autodest オプション

コピー先ディレクトリを推測する。これを指定する場合、debian/package.install ファイルのコピー先ディレクトリは指定しないこと。 debian/tmp のディレクトリ配下にあるファイルを debian/tmp を除いて指定ディレクトリの下に対応するようにコピーする。例

```
$ cat debian/package.install
debian/tmp/usr/bin
debian/tmp/etc/passwd
```

debian/tmp/usr/bin を debian/package/usr/へコピー
debian/tmp/etc/passwd を debian/package/etc/へコピー

–list-misqqsing オプション

ファイル(およびシンボリックリンク)がどのディレクトリにもコピーされなかったときに標準エラー出力に警告を表示する。ラージ・パッケージに、新しく追加されたファイルを見逃さないためなどに使える。

-Xitem, –exclude=item オプション

指定したファイル名を含むファイルをコピー対象外とする。

DEBHELPERのプログラム共通のオプション1

アーキティクチャ独立

-i, –indep	Act on all architecture independent packages.

例

```
$ dh_make -m --rulesformat old
```

```
install-indep:
        (中略)
        dh_prep -i
        dh_installdirs -i
        (中略)
        dh_install -i
        (後略)
```

DEBHELPERのプログラム共通のオプション2

アーキティクチャ依存

-s, –same-arch	This used to be a smarter version of the -a flag, but the -a flag is now equally smart.
-a, –arch	Act on architecture dependent packages that should be built for the build architecture.

例(同)

```
install-arch:
        (中略)
        dh_prep -s
        dh_installdirs -s

        # Add here commands to install the arch part
        # of the package into debian/tmp.
        $(MAKE) DESTDIR=$(CURDIR)/debian/hello install

        dh_install -s
        (後略)
```

DEBHELPERのプログラム共有のオプションその他1

オプション	動作
-v, –verbose	詳しく動作を表示 (Verbose mode: show all commands that modify the package build directory.)
–no-act	実際の動作をしない (Do not really do anything. If used with -v, the result is that the command will output what it would have done.)
-ppackage, –package=package	Act on the package named package. This option may be specified multiple times to make debhelper operate on a given set of packages.
-Npackage, –no-package=package	Do not act on the specified package even if an -a, -i, or -p option lists the package as one that should be acted on.
–remaining-packages	Do not act on the packages which have already been acted on by this debhelper command earlier (i.e. if the command is present in the package debhelper log). For example, if you need to call the command with special options only for a couple of binary packages, pass this option to the last call of the command to process the rest of packages with default settings.

DEBHELPERのプログラム共有のオプションその他2

オプション	動作
–ignore=file	Ignore the specified file. This can be used if debian/ contains a debhelper config file that a debhelper command should not act on. Note that debian/compat, debian/control, and debian/changelog can't be ignored, but then, there should never be a reason to ignore those files. For example, if upstream ships a debian/init that you don't want dh_installinit to install, use –ignore=debian/init
-Ptmpdir, –tmpdir=tmpdir	Use tmpdir for package build directory. The default is debian/package
–mainpackage=package	This little-used option changes the package which debhelper considers the "main package", that is, the first one listed in debian/control, and the one for which debian/foo files can be used instead of the usual debian/package.foo files.
-O=option—bundle	This is used by dh(1) when passing user-specified options to all the commands it runs. If the command supports the specified option or option bundle, it will take effect. If the command does not support the option (or any part of an option bundle), it will be ignored.

次回発表者は?

〜さんですー

gdb python 拡張その1

デバッガの使い道

- デバッグのお供に
- リバースエンジニアリングのお供に

参考: OSS のリバースエンジニアリング

OSS をリバースエンジニアリングしたくなる状況:

- とにかく抽象化を使いこなしてるコード
- データドリブンなコード
- そもそも俺の理解を越えたコード

と戦う時。

... ソースコード読んだだけでは動作がわからん...

で、デバッガで実行して動作を追いかけると、いろいろな発見があったり、動作が良くわかったり、アーキテクチャが見えたりする。

gdb

- gdb とは
  幅広いプラットフォームで利用でき、多数のバイナリ形式に対応でき、組み込み用途にリモートデバッグまで出来る、非常に強力なデバッガ。

gdb+python

- gdb v7.0 から gdb に python が合体!
- Debian sid 梱包の gdb も python が有効!

強力なデバッガがさらに強力に!
gdb もこれで batteries included!

Debian sid の gdb の python 具合

項番	項目	値
1	gdb バージョン	7.4.1
2	python バージョン	2.7.3
3	python サーチパス	/usr/share/gdb/python, /usr/lib/python2.7 以下

ちと困る事

とにかく、文献が少ない!

- info gdb → Extending GDB → python の項目か、
- `http://sourceware.org/gdb/wiki/PythonGdbTutorial`

ぐらい。

他にあったら教えてーっ

補足: さらに困った事

「しまった、俺はpythonの書き方も知らなかったな。そういえば。」
(即席で、文法も覚える必要がががが...)

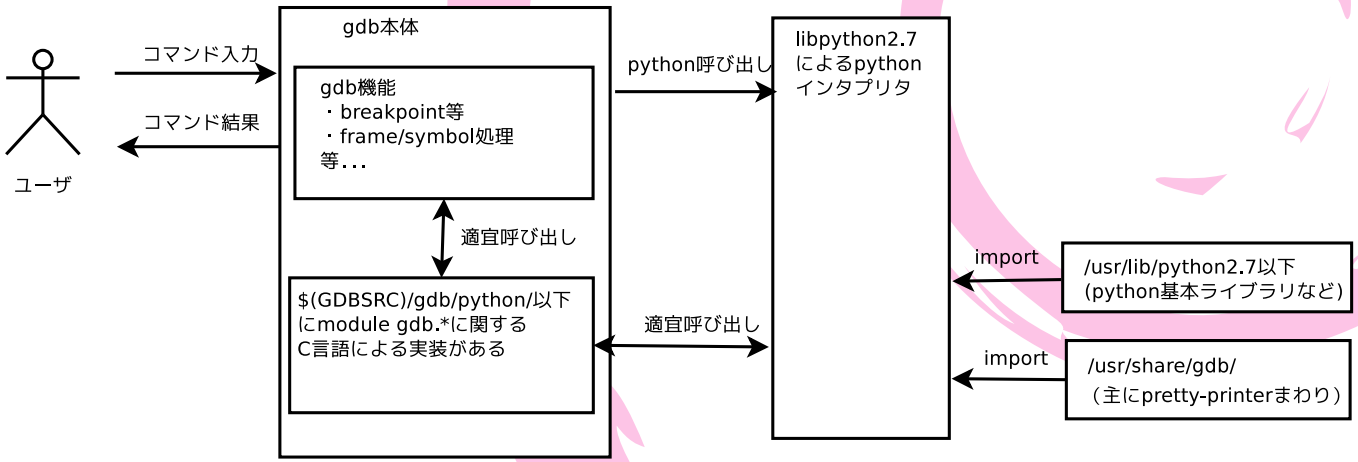
とりあえず動かしてみる

とりあえず、動かしてみる。 (gdb) は gdb のプロンプトです。

```
$gdb
(gdb) python print "hello world"
hello world
(gdb) python a=[1,2,3,4]
(gdb) python print a
[1, 2, 3, 4]
(gdb) python a.append(5)
(gdb) python print a
[1, 2, 3, 4, 5]
(gdb) python
>import sys
>print sys.path
>print sys.version_info
>ここで Ctrl+d
['/usr/share/gdb/python', '/usr/lib/python2.7',
  '/usr/lib/python2.7/plat-linux2', ... 中略...
```

gdb+python 構造

バックグラウンドで python 動かしてるんだろうと思ったら大間違い。実際には gdb に python 処理系を結合してた。



module gdb の定義はどこ

python で module の定義を知るには、 pydoc がある。でも、gdb の中に実装された module gdb の定義を読むには...

python の help() 関数を呼べばいい

```
(gdb) python help(gdb)
Help on package gdb:

NAME
    gdb

FILE
    (built-in)

PACKAGE CONTENTS
    command (package)
    printing
... 中略...
```

module gdb の class 図が見たい

先程の module gdb の定義から、地道に class 図を起こしてみた。
参照: 第 98 回東京エリア Debian 勉強会資料 6.6 章の図参照。

基本的使い方: gdb コマンドを増やす (その 1)

gdb に新規のコマンドを増やし、 python と結合するには gdb.Command クラスを継承したオブジェクトを生成して増やす。 gdb 側に増やすコマンド名は gdb.Command オブジェクトのコンストラクタ経由で登録する。

hello.py の中身:

```
import gdb
class HelloWorld (gdb.Command):
  """ Greet the whole world """
  def __init__ (self):
     super(HelloWorld, self).__init__ ("hello-world",
                                       gdb.COMMAND_OBSCURE)

  def invoke (self,arg, from_tty):
     print "Hello, World! arg=["+str(arg)+"]"

HelloWorld()
```

基本的使い方: gdb コマンドを増やす(その2)

実行してみた。

```
(gdb) source hello.py
(gdb) hello-[ここで TAB を押すと補完される]
(gdb) hello-world foo,bar,com
Hello, World! arg=[foo,bar,com]
(gdb) help obscure
Obscure features.

List of commands:

... 中略...
hello-world --  Greet the whole world
... 中略...
```

作ったスクリプトの自動読み込み

いつもいつも"source スクリプト名" は面倒なので、自動で読み込ませたい時は:

- `$(HOME)/.gdbinit` で読ませる
- バイナリ側`.gdb_gdb_scripts` セクションに埋め込む
- “バイナリ名”-gdb.py という名前でスクリプトを用意する

の方法があります。

.gdbinit で読ませる

以下のファイルを HOME ディレクトリ配下に用意:

```
----${HOME}/.gdbinit ここから-----
source /home/foo/bar/my-gdb-func.py
----${HOME}/.gdbinit ここまで-----
```

.gdb_gdb_scripts で読ませる

以下の asm 命令を打ち込んでおく。

```
asm(
".pushsection \".debug_gdb_scripts\",\"MS\",@progbits,1\n"
".byte 1\n"
".asciz \"hello.py\"\n"
".popsection \n"
);
```

ちと脆弱性の香りが...

“バイナリ名”-gdb.py で読ませる

```
$ ls
hello hello-gdb.py hello.c
```

gdb hello とすると、 hello-gdb.py が自動で読み込まれる。

基本的使い方： breakpoint を操る

gdb.Breakpoint クラスを継承したオブジェトを用意すると、
指定した breakpoint で特定の処理をさせる事が可能。
main で break して hi! と表示する例:

```
class _ExampleBreakpoint(gdb.Breakpoint):
    def __init__(self, spec):
        super(_ExampleBreakpoint,
              self).__init__(spec,gdb.BP_BREAKPOINT,
                             internal = False)
    def stop(self):
         print "hi!"

_ExampleBreakpoint("main")
```

gdb.BP_BREAKPOINT を指定すると、 breakpoint になり、
gdb.BP_WATCHPOINT を指定すると、 watchpoint となる。

基本的使い方： finish を操る

gdb.FinishBreakpoint クラスを継承したオブジェトを用意すると、現在のスタックフレームから飛び出ようとするとbreakする。
スタックフレームから出るとhi!と表示する例:

```
class _ExFinishBreakpoint(gdb.FinishBreakpoint):
    def __init__(self):
        super(_ ExFinishBreakpoint,
              self).__init__(internal=False)
    def stop(self):
        print "hi!"

    def out_of_scope(self):
        print "hi!"
```

stop() は、 return によりスタックフレームから出る場合、out_of_scope() は longjump とかで一機に飛び出す場合にbreakする。

基本的使い方: コマンドの結果を得る

gdb のコマンドの実行結果を得る事ができると便利な場合があります。

```
result=gdb.execute('info break',False,True)
```

result に gdb のコマンド"info break" の結果が文字列クラスで格納されます。

以上を活用してみた例

解析用途で、プログラムのコールツリーを取ってみたかった。
具体的なスクリプトのソースと使い方は、
第 98 回東京エリア Debian 勉強会資料 6.9 章参照。

calltracer.py 動作説明

Step 1. rbreak を実行して、デバッグ情報にある全関数に一旦 breakpoint を仕掛ける

Step 2. info break の結果をパースして、関数のアドレスと、名前の両方を得る。

Step 3. カスタマイズした breakpoint と入れ替える。

Step 4. break したら、スタック追跡変数 (self.stack) を+1して関数名を表示。同時にカスタマイズした finish を仕掛ける。

Step 5. finish で break したら、関数名を表示してスタック追跡変数を-1 する。

Step 6. Step 4.〜Step 5. を繰り返す。

終わりに

python のおかげで、
なんかもう夢ひろがりんぐ
まだまだ機能満載なので、
次回も発表続けるよどこまでも!

今後のイベント

今後のイベント

- 2013 年 4 月 Debian 勉強会

今日の宴会場所

今日の宴会場所

未定