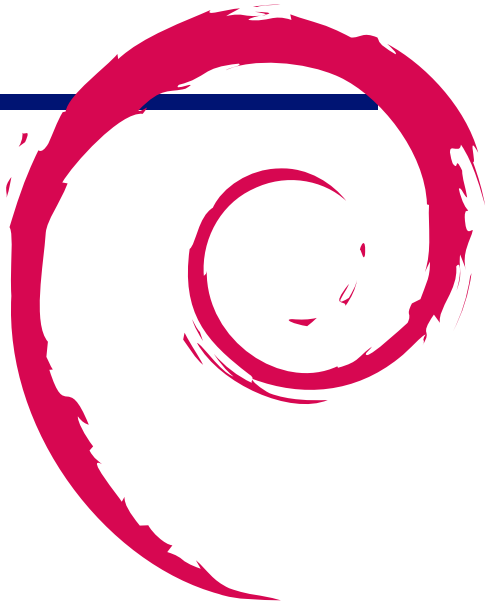
.Deb

銀河系唯一のDebian専門誌

2013 年 3 月 16 日

特集 1： ldapvi & python-ldap **で** stress-free life

特集 2： gdb python **拡張**



1 Introduction

上川 純一

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。

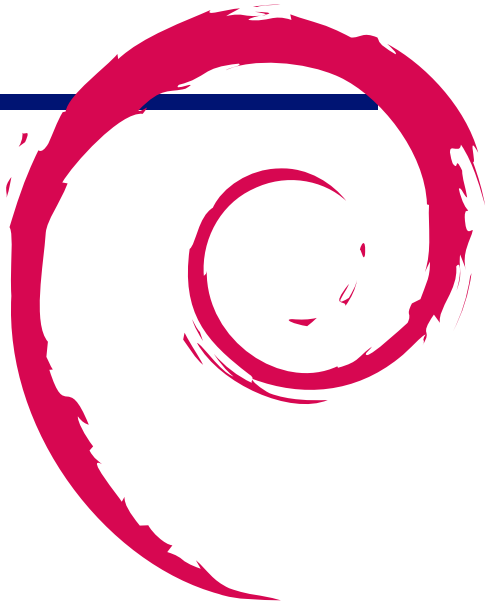
Debian の勉強会ということで究極的には参加者全員が Debian Package をがりがりと作るスーパーハッカーになった姿を妄想しています。情報の共有・活用を通して Debian の今後の能動的な展開への土台として、「 場」としての空間を提供するのが目的です。

２０１３年の計画は下記です:

1. 2013 年の計画立案
2. OSC 東京 nojima
3. UEFI・セキュアブートを Debian でどうやるか
4. セキュア OS 再入門
5. Debian で自動化の夢は見れるか – データセンターで OS のインストールからアプリのデプロイ、サービス公開まで
6. amazon AWS での Debian 入門
7. スイスでキャンプ
8. Debian でスクリプト言語のパッケージ: perl, ruby, python
9. Debian で apache 2.4: サーバ構築、 Apache module プログラミング
10. Raspberry Pi 、 SheevaBox、 Freedombox は今
11. 自由な FPGA
12. 一年間の反省

デビアン勉強会

目次



2 事前課題

上川 純一

今回の事前課題は以下です:

1. Debian でここがバグってるかも? という事について列挙ください。
2. 使ったことのある/使ってみたいデバッガについて語って下さい。
3. LDAP のシステムを管理していますか? している場合は、 slapd.conf と slapd-config どちらを使っていますか? その理由は?

この課題に対して提出いただいた内容は以下です。

2.1 dictoss(杉本 典充)

2.1.1 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。

2.1.2 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。

2.1.3 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。

2.2 キタハラ

2.2.1 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。

2.2.2 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。

2.2.3 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・・・)

2.3 まえだこうへい

2.3.1 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情報を削除できず、 shell モードから消してましたが、今ってどうなんでしょう。

2.3.2 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。

2.3.3 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

2.4 たかはしもとのぶ

2.4.1 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。

2.4.2 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。

2.4.3 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。

2.5 yamamoto

2.5.1 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。

2.5.2 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。

2.5.3 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?

2.6 野島 貴英

2.6.1 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。

2.6.2 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。

2.6.3 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。

2.7 koedoyoshida

2.7.1 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。

2.7.2 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

2.7.3 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...

2.8 鈴木崇文

2.8.1 Debian でここがバグってるかもという点は?

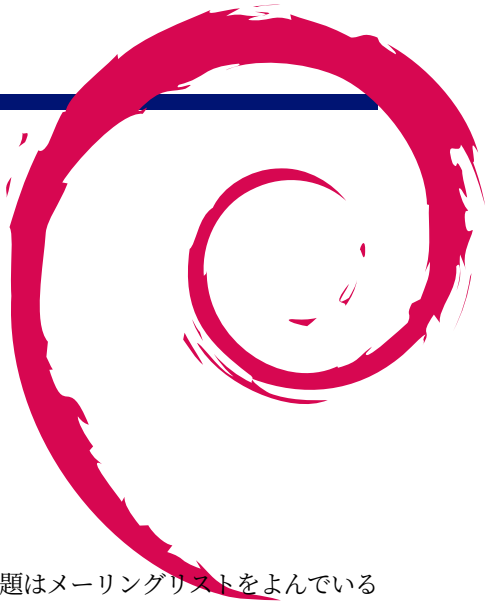
あまりバグってると感じたことはないです。

2.8.2 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。

2.8.3 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。



3 Debian Trivia Quiz

上川純一

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. 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 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 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 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 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

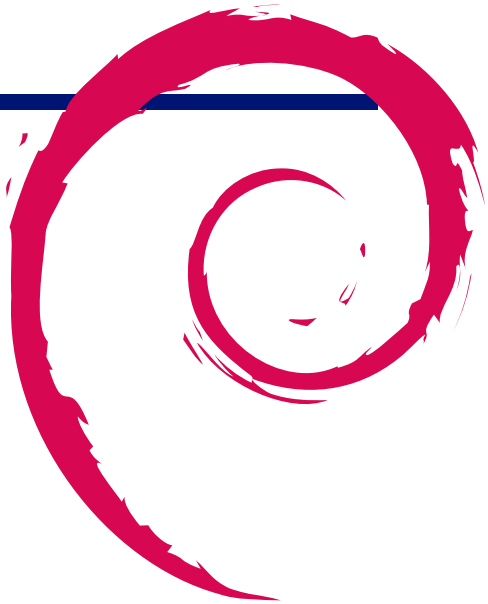
☐ C Debian/GnuHurd
port image

問題 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 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 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 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



4 最近の Debian 関連のミーティング報告

上川純一

4.1 東京エリア Debian 勉強会 96 回目報告

２０１３年最初の東京エリア Debian 勉強会はスクエアエニックスさんのオフィスを間借りして開催しました。参加者は yy_y_ja_jp さん、キタハラさん、 dictoss さん、 koedoyoshida さん、 dai さん、 yamamoto さん、まえださん、上川さん、野島さん (＋他会場お手伝い２名）でした。

野島さん主催の DWN Quiz は今回は数が少ないが難問でいい感じした。

事前課題の紹介は２０１５年の予想と２０１３年にやりたいことの提案でした。いろいろと妄想しているのがわかりました。

全員でグループワークとして、２０１５年の予想、２０１３年の計画を行いました。過去に予想した未来の確認と、あらためて今後３年間の妄想をぶつけあいました。

2011	2012	2013	2014	2015
デスクトップパソコン終了の潮流。 cpu コア単体では高速化しないように。 webos 終了のお知らせ。 adobe flash 復活のお知らせ (キタ), silverlight 終了のお知らせ (台湾を除く)(続いてる?) squeeze リリース (おめでとう) ipv4 割り当ての終了のお知らせ (キタ) 地上波デジタル移行延長。 btrfs まだ頑張る (fedora 乙) java 終了 (sun java 終了) open office が oracle office に (ナイ)	ノートパソコンよりタブレットのほうが売れている。ノートパソコンでは macbookair が常識に。 ノートパソコンで intel じゃないもの (mips/arm) が主流にはまだならず。タブレットのほうが主流。 デスクトップ:ゲーム以外の用途では終了している。 サーバ:個人レベルでは VPS 常識。企業ユースでも cloud か、 vps かを自前と比較検討する時代。データセンターを置く国を選べる時代。 携帯電話: ガラパゴスの終焉。日本での携帯電話販売でもスマートフォンが５０%を超えるように。ガラケー向けのネットバンクの提供が終了など、ガラケーからサービスが撤退し始める。 LTE 登場、普及しはじめたが、主流になっていない。 softbank の二年契約はまだ続いている。 sim free への道は耕されたがあたり前にならなかった。 btrfs はまだ生き残っているがまだ使われてない? openstack で ceph 使う人もいる? mysql から mariadb が派生。	コンシューマーはノートパソコンを買わなくなった。 ノートパソコンのかわりにスマートフォンを使っている。 スマートフォンが 7 インチくらいまで拡大、タブレットとは何だったのか。 自宅用のデスクトップパソコンのかわりに 10 インチくらいのタブレットを使うように。 サーバ: クラウドで処理するのが主流。 python / ruby でコード書いていると CPU が何かわからない。裏で動いている CPU は一般人は知らない。 ARM ホストの仮想化技術が発達。 Oracle がメンテナンスする気がないのが明確になり、 java リスクが顕著になる。 固定ゲーム機の終焉。 ゲームは ARM。	Intel がまた ARM に参入、もしくは省電力 CPU を主力に切り替える。 気づいたら自作パソコン業界が終焉している。セキュアブートが普及している。 AMD が ARM コアの CPU を出す。 Java が Oracle 管理からはずれる。 スマートフォンの電池がガラケーなみに持つようになる。 電池消費が重要なアプリ選択の要素となる。スポイトで充電できる、燃料電池が流行る。 AR メガネのプロトタイプが出てくる。	自作スマホの時代。 OpenHardware がモバイルに移行する。技適のパーツ認定基準というのができるようにがんばる。 自宅で回路が印刷できる機器が普及して CPU とかが印刷できるようになるといいな。 タブレットが丸められるようになって巻物になっている。 AMD が x86 撤退。 ハードディスクを見たことがない人がいる。 データセンターを自前でもっているのは発電所を持っているところだけになる。 クラウドの法制度、免責事項、個人情報保護関連の問題が提起され、解決にむけてすすむ。一種データセンタークラウド業者の要求規格が制定される。ユーザ数何人以上は二種免許が必要とか。 データセンターヘイブンとよばれる国が存在する。

そしてそのあと上川司会で２０１３年の各月のテーマ候補について語り合いました。

最後に野島さんが Debhelper の話をして終了。 GDB の話は時間オーバーしてしまったので次回にもちこすことになりました。

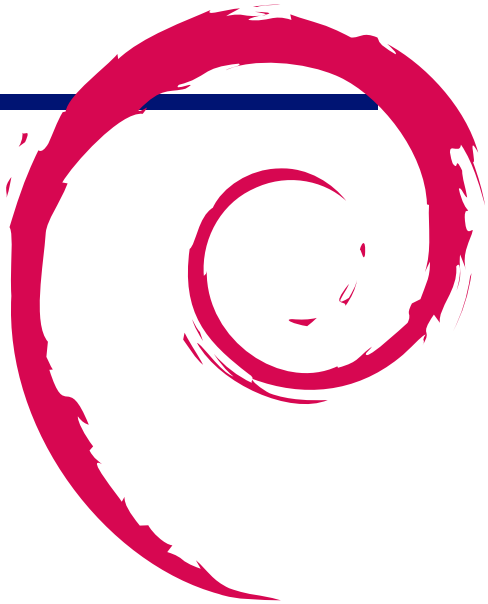
4.2 Open Source Conference 2013 Hamamatsu

- 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。
- 来場者数:約 200 名
- ブース来場者数:約 30 名前後

4.3 OSC 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。
- 土曜来場者数:約 1000 名
- ブース来場者数:約 50 名前後

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



5 ldapvi & python-ldap で stress-free life

まえだこうへい

5.1 ストレス溜まってませんか?

今回の記事は OpenLDAP の管理を行っている人がターゲットです。 LDAP サーバを管理している人は多くはないと思いますが、この記事が LDAP の運用で苦労している人の一助になれば良いと思います。 LDAP なにそれ?な人も LDAP アカウントやアドレス帳などで知らぬ間に恩恵を受けているということは大いにあるので、明日は我が身と思ってお読み下さい。また、自分で作ったツールのアカウント管理を LDAP で行いたい、という場合にも参考になると思います。

なお、 LDAP の基本的なお話や、 Debian システムでの OpenLDAP の入門については、第 61 回 関西 Debian 勉強会 (2012 年 7 月) に佐々木さんが発表&資料を掲載されているのでそちらを参照下さい。[*1]

5.2 stress = LDIF

LDAP の運用で何が面倒かというと、 LDIF(LDAP Data Interchange Format) の作成やパースです。 LDAP のデータを作成・更新・削除するには通常、 LDIF を作成し、 ldapadd/ldapmodify/ldapdelete コマンドで実行します。一つのオブジェクトは dn(distiguised name, 識別名) から始まり、それに attribute が続きます。複数オブジェクトを記述するには空行で分割します。下記は RFC2849 の "Example 1: An simple LDAP file with two entries" からの引用です。[*2]

```
version: 1
dn: cn=Barbara Jensen, ou=Product Development, dc=airius, dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
cn: Barbara Jensen
cn: Barbara J Jensen
cn: Babs Jensen
sn: Jensen
uid: bjensen
telephonenumber: +1 408 555 1212

dn: cn=Bjorn Jensen, ou=Accounting, dc=airius, dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
cn: Bjorn Jensen
sn: Jensen
telephonenumber: +1 408 555 1212
```

データフォーマット自体はさほど難しくありませんが、このデータを一から自分でスクラッチで作成するのはとても面倒だと思いませんか? 例えば上記のようなユーザアカウントの場合、人間にとっては TSV などの形式を表計算ソフトで読んだり編集したりすることでしょう。そこから LDIF に変換したり、あるいはその逆はどうしますか? 簡単に思いつくのは

[*1] あんどきゅめんてっど Debian2012 年冬号 8.「 Debian で作る LDAP サーバ」 http://tokyodebian.alioth.debian.org/pdf/debianmeetingresume2012-fuyu.pdf

[*2] http://www.ietf.org/rfc/rfc2849.txt

次の二つでしょう。

1. 手でちまちま変換する
2. なんらかのプログラムで一括変換する

前者は転記ミスなどの可能性から論外ですが、管理対象のアカウントが少数ならいっそ割り切って LDIF を手で作成するのもありでしょう。しかしユーザ数が大量になるとプログラムで変換しないと時間の無駄です。

5.2.1 そして slapd-config

OpenLDAP 2.3 からは動的に設定を変更できる slapd-config(5) が採用されました。これ自体も LDAP データベースとなっています。 current release の 2.4 系では 一部の場合のぞき、従来の slapd.conf(5) は推奨されていません。将来のリリースでは slapd.conf(5) は取り除かれることになっています。[*3]。廃止されるものを使いつづけても仕方ありませんね。しかし、ユーザデータの管理ですら LDIF で悩んでいるのに、さらに slapd 自体の設定までも LDIF を作らないといけないと思うと、もう気が狂いそうです。

5.3 ldapvi で LDIF 地獄からの脱却

そこで、 ldapvi です[*4]。このツールは、 vipw(8) コマンドや、 visudo(8) と同じように、 vi のインタフェースで LDAP のデータを変更できます。これで少数のデータを管理するときはもちろん、 slapd-config の設定管理の悩みも解決します。

使うには、 ldapvi パッケージをインストールします。

```
$ sudo apt-get install ldapvi
```

使い方は、 ldap-utils パッケージのコマンドのオプションとほぼ同じです。 slapd-config での操作は下記のコマンドを実行します。

```
$ sudo ldapvi -Y EXTERNAL -h ldapi:/// -b cn=config
```

これで slapd-config 管理下のすべてのオブジェクトツリーが ldapvi のフォーマットとして表示されます。 ldapvi のフォーマットはほぼ LDIF に似ています。

```
(snip)

0 cn=config
objectClass: olcGlobal
cn: config
olcArgsFile: /var/run/slapd/slapd.args
olcLogLevel: 512
olcPidFile: /var/run/slapd/slapd.pid
olcToolThreads: 1

1 cn=module{0},cn=config
objectClass: olcModuleList
cn: module{0}
olcModulePath: /usr/lib/ldap
olcModuleLoad: {0}back_hdb

2 cn=schema,cn=config
objectClass: olcSchemaConfig
cn: schema
(snip)
```

LDIF との違いは、”dn: cn=config”となっている部分が”0 cn=config”と表示されていることです。この数字は表示されているオブジェクトを 0 を基数とする序数となっています。例えば、新しいオブジェクトを追加する場合は、数字の代わりに”add”を使います。 auditlog, ppolicy モジュールを新しくロードする場合は下記のように挿入します。

[*3] http://www.openldap.org/doc/admin24/slapdconf2.html
[*4] http://www.lichteblau.com/ldapvi/

```
(snip)

1 cn=module{0},cn=config
objectClass: olcModuleList
cn: module{0}
olcModulePath: /usr/lib/ldap
olcModuleLoad: {0}back_hdb

add cn=modile,cn=config
objectClass: olcModuleList
cn: module
olcModulePath: /usr/lib/ldap
olcModuleLoad: auditlog.la

add cn=modile,cn=config
objectClass: olcModuleList
cn: module
olcModulePath: /usr/lib/ldap
olcModuleLoad: ppollicy.la

2 cn=schema,cn=config
objectClass: olcSchemaConfig
cn: schema
(snip)
```

追記したら、 vi と同様に保存&終了してみます (:wq コマンドを実行)。すると次のメッセージが出力されます。

```
SASL/EXTERNAL authentication started
SASL username: gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth
SASL SSF: 0
     12 entries read
add: 2, rename: 0, modify: 0, delete: 0
Action? [yYqQvVebB*rsf+?]
```

ここで、”y”を入力すると、実際に slapd-config に反映されます。前述の通り、 slapd.conf とは違い、 slapd の再起動は不要です。”q”を入力するとキャンセルして終了します。

上記での挿入時に、 cn=module0 の”0”を残したまま保存してみてください。

```
add cn=modile{0},cn=config
objectClass: olcModuleList
cn: module{0}
olcModulePath: /usr/lib/ldap
olcModuleLoad: ppollicy.la
```

すると次のようにエラーに修正を要求されます。

```
ldap_add: Naming violation (64)
add: 1, rename: 0, modify: 0, delete: 0
Action? [yYqQvVebB*rsf+?]
```

これはすでに”cn=module0, cn=config”が存在するためです。新規追加時に dn では親や先祖のノードに序数が指定されていれば必要ですが、自身のノードには自動的に付与されるので指定してはいけません。今回は”e”を押し編集モードに戻り、 dn および attribute の cn の序数を取り除きましょう。

編集が終わったら、”y”を押して保存する前に、”v”を押します。すると、 LDIF 形式で表示されます。これは後々役立ちます。

```
version: 1

dn: cn=module,cn=config
changetype: add
objectClass: olcModuleList
cn: module
olcModulePath: /usr/lib/ldap
olcModuleLoad: auditlog.la

dn: cn=module,cn=config
changetype: add
objectClass: olcModuleList
cn: module
olcModulePath: /usr/lib/ldap
olcModuleLoad: ppolicy.la
```

既にあるオブジェクトの attribute を変更する場合は、値を変更するだけです。オブジェクトを削除する場合は、対象のオブジェクトのすべての行を、特定の attribute だけを削除する場合はその行を削除するだけです。自分で LDIF を作るのと違い、非常に簡単であることがわかるでしょう。フォーマットが間違っている場合は前述のようにエラーになる上、変

更時に slapd の再起動が不要なので、実は slapd.conf での管理よりもとても便利です。

ユーザデータを変更する場合も基本的に同じです。 ldapvi コマンドで指定するオプションが異なるだけです。

```
$ ldapvi -h ldap://localhost -D cn=admin,dc=example,dc=org -b dc=example,dc=org
```

ldapvi の詳細は、ユーザマニュアルが充実しているのでそちらを参照ください。[*5]

5.4 debconf-utils と slapd-config で導入の自動化を進める

OpenLDAP を使う場合、 slapd パッケージをインストールしますが、 debconf による対話形式の設定が必要になります。はじめてインストールする場合などには便利なのですが、ある程度なれてきて、 slapd の設定自体もあらかじめ決めてあると、これはかえって面倒なものになります。予め設定するパラメータは決めてあるのですから自動化したいですよね。そういう場合 debconf-utils パッケージと、 そして slapd-config を上手に使うことで自動化できるます。

5.4.1 事前準備

まず、テスト環境などを用意します。このテスト環境の目的は次の二つです。

1. debconf の設定を抽出する
2. slapd-config 用の LDIF を用意する

前者は slapd パッケージをインストールの際に今後設定したいパラメータを入力しておく必要があります。設定された状態で debconf-utils パッケージの debconf-get-selections コマンドで slapd に関する debconf のパラメータを抽出するためです。後者は slapd パッケージをインストールした直後の状態にしておき、この状態で ldapvi で予め設定したい attribute の値を変更し、前述の LDIF 形式の出力します。

5.4.2 debconf-get-selections でパラメータを抽出する

slapd をインストール状態で次のコマンドを実行します。

```
$ LANG=C sudo debconf-get-selections | grep slapd
slapd   slapd/password2 password
slapd   slapd/internal/generated_adminpw        password
slapd   slapd/internal/adminpw  password
slapd   slapd/password1 password
slapd   slapd/password_mismatch note
slapd   slapd/invalid_config    boolean true
slapd   shared/organization     string  example.org
slapd   slapd/upgrade_slapcat_failure   error
slapd   slapd/backend   select  HDB
slapd   slapd/dump_database     select  when needed
slapd   slapd/allow_ldap_v2     boolean false
slapd   slapd/no_configuration  boolean false
slapd   slapd/move_old_database boolean true
slapd   slapd/dump_database_destdir     string  /var/backups/slapd-VERSION
# Do you want the database to be removed when slapd is purged?
slapd   slapd/purge_database    boolean false
slapd   slapd/domain    string  example.org
```

このうち、"slapd/upgrade_slapcat_failure"と"#"から始まるコメント行は不要です。削除して slapd-debconf.txt などの任意のファイル名で保存します。これを slapd パッケージのインストール時に反映させるには、パッケージインストール前に debconf-set-selections コマンドを使います。

```
$ sudo bash -c "cat slapd-debconf.txt | debconf-set-selections"
$ sudo DEBCONF_FRONTEND=noninteractive apt-get -y --force-yes install slapd
```

slapd の設定用 LDIF は、前述のテスト環境で ldapvi を使って生成します。次の 3 つの設定を行います。

- rootDN のパスワード
- loglevel

[*5] ldapvi User Manual http://www.lichteblau.com/ldapvi/manual/

  - 128
- access control
  - パスワード以外のデータの参照は anonymous で可能
  - データの更新は rootDN のみ行える
  - パスワードのみ自身で変更可能

まず、 ハッシュ化された rootDN のパスワードを slappasswd で生成します。

```
$ sudo slappasswd
New password:
Re-enter new password:
{SSHA}MEj4D59lrtA+6+0J4vIz2RTByA7KT6Zq
```

次に、 ldapvi で上記の設定を行います。

```
$ sudo ldapvi -h ldapi:/// -Y EXTERNAL -b cn=config
```

変更箇所は下記の ”¡- (*)” の部分です。

```
0 cn=config
(snip)
olcLogLevel: 128 <- (*)
(snip)

10 olcDatabase={1}hdb,cn=config
(snip)
olcAccess: {0}to attrs=userPassword,shadowLastChange by self write by anonymous auth
 by dn="cn=admin,dc=example,dc=org" write by * none <- (*)
olcAccess: {1}to dn.base="" by * read <- (*)
olcAccess: {2}to * by dn="cn=admin,dc=example,dc=org" write by * read <- (*)
olcAccess: {3}to dn.subtree="dc=example,dc=org" by self read by * read <- (*)
olcAccess: {4}to * by * none <- (*)
(snip)
olcRootPW: {SSHA}MEj4D59lrtA+6+0J4vIz2RTByA7KT6Zq <- (*)
(snip)
```

変更後、 LDIF を生成すると下記のように表示されます。

```
version: 1

dn: cn=config
changetype: modify
replace: olcLogLevel
olcLogLevel: 128
-

dn: olcDatabase={1}hdb,cn=config
changetype: modify
replace: olcAccess
olcAccess: {0}to attrs=userPassword,shadowLastChange by self write by anonymous auth
 by dn="cn=admin,dc=example,dc=org" write by * none
olcAccess: {1}to dn.base="" by * read
olcAccess: {2}to attrs=sshPublicKey by self write
olcAccess: {3}to * by dn="cn=admin,dc=example,dc=org" write by * read
olcAccess: {4}to dn.subtree="dc=example,dc=org" by self read by * read
olcAccess: {5}to * by * none
-
replace: olcRootPW
olcRootPW: {SSHA}MEj4D59lrtA+6+0J4vIz2RTByA7KT6Zq
```

残念ながら ldapvi にはこれを任意のファイルとして保存する機能ありません。コンソールをコピーして、適当なファイル名で保存します (今回は setup.ldif)。保存した LDIF を用い、 ldapmodify コマンドを使えば slapd サーバに反映させることができます。

```
$ sudo ldapmodify -Y EXTERNAL -H ldapi:/// -f setup.ldif
```

パッケージのインストールから設定まで自動化するなら次のコマンドで行えます。

```
sudo apt-get install debconf ldap-utils
sudo bash -c 'cat slapd-debconf.txt | debconf-set-selections'
sudo DEBCONF_FRONTEND=noninteractive apt-get -y --force-yes install slapd
ldapmodify -Y EXTERNAL -H ldapi:// -f setup.ldif
```

例えば、 LDAP 関連のツールのテストを、クリーン環境に CI などを使って行う場合は上記のコマンドで、リポジトリ

に push する度にテスト用の slapd を自動インストールしてテストを実行させることもできます。

5.5 python-ldap で stress-free に

さらにプログラムで LDAP を扱いましょう。今回は python の LDAP binding である、 Python-LDAP というモジュールを使います[*6]。 Debian では python-ldap パッケージです。

```
$ sudo apt-get install python-ldap
```

python-ldap で扱う基本的なデータ構造は下記のように、 LDAP オブジェクトがリストになっており、 LDAP オブジェクト自体はタプルになっています。タプルの第一要素が entryDN で、第二要素が辞書になっている LDAP オブジェクトの attributes です。各 attribute の値もリストになっているので、例えば objectClass のように複数の値を持つ場合にも対応できます。そして entryDN と attributes の各値はすべて文字列です。

```
[
    ('DN', {'attribute': ['value'], 'attribute': ['value', 'value', ...], ...}),
    ('DN', {'attribute': ['value'], 'attribute': ['value', 'value', ...], ...}),
    ...
]
```

5.5.1 検索 (ログイン) の例

検索の場合、指定した username が uid として存在するか、存在するなら simple bind で認証できるかの処理を行っています。

```
import ldap

UID = 'user0'
PASSWORD = 'passWord'

# LDAP サーバに接続
conn = ldap.initialize('ldap://localhost')
# anonymous で uid が "username" のユーザを検索
res = conn.search_s('ou=People,dc=example,dc=org', ldap.SCOPE_SUBTREE,
                    '(uid=%s)' % UID, ['uid'])
if res:
    # "username" が見つかった場合
    dn, attr_d = res[0]
    # simple bind で認証
    try:
        conn.simple_bind_s(dn, PASSWORD)
        login_result = True
    except ldap.INVALID_CREDENTIALS:
        # 認証失敗
        login_result = False
else:
    # "username" が存在しない
    login_result = False
```

5.5.2 データ追加

データの追加や更新の場合、 ldap.modlist モジュールを使う必要があります。追加の場合には、 ldap.modlist.addModList() を使いますが、前述の辞書型の attributes を引数にすると、下記のフォーマットのリストに変換されます。

```
[('attribute', ['value']),
 ('attribute', ['value', 'value', ...]), ...]
```

これを `ldap.add_s(dn, attrs_l)` の引数とすれば新規オブジェクトの追加ができます。サンプルは次の通りです。

[*6] http://www.python-ldap.org/

```
import ldap.modlist

rootdn = 'cn=admin,dc=example,dc=org'
rootpw = 'password'

dn = 'uid=mkouhei,ou=People,dc=example,dc=org'
attrs_d = {'objectClass': ['inetOrgPerson', 'posixAccount', 'shadowAccount'],
           'uidNumber': ['1000'],
           'gidNumber': ['1000'],
           'homeDirectory': ['/home/mkouhei'],
           'loginShell': ['/bin/bash'],
           'uid': ['mkouhei'],
           'cn': ['Kouhei'],
           'sn': ['Maeda']}

# attributes を辞書からリストに変換
attrs_l = ldap.modlist.addModlist(attrs_d)
# rootdn で bind
conn.simple_bind_s(rootdn, rootpw)
# dn を指定して新規オブジェクトを追加
conn.add_s(dn, attrs_l)
```

5.5.3 データの削除

データを削除する場合は、 dn を指定するだけです。

```
conn.delete_s(dn)
```

5.5.4 python-ldap の筆者の使用例

サーバ/ネットワークまわりの自動化ツールや、ユーザ向けの Web ベースのツールの開発を python で行っています。認証には LDAP を使うことが多いため、これらのツール用に python-ldap を使ったモジュールを作っています。また、複数の異なる LDAP アカウントのツリーをマージするのにも使っています。ユーザアカウントのデータを変換してマージする必要があるため、 LDIF を自分でパースして変換するのではなく、 python のリストや辞書で処理できるのはとても楽です。変換元の LDAP サーバには標準スキーマではなく独自拡張のスキーマを使っているものあり、名前すら違う attribute もあるので python で処理できるのが大変便利です。

5.6 まとめ

ldapvi、 python-ldap を使うと LDIF を自分で作成したり、パースする必要がなくなることを説明しました。 また、 slapd-config であっても ldapvi を使えば、 slapd.conf での場合と同様に簡単に管理できることも説明しました。注意点としては、 slapd.conf での設定パラメータの名前が slapd-config では多少異なることですが、 cn=config のツリーの root を指定すると、 ロードされているスキーマも見ることができるのでそこから検索、類推することができます。ストレスフルな LDAP の管理を少しでも stress-free にできるようになると良いですね。

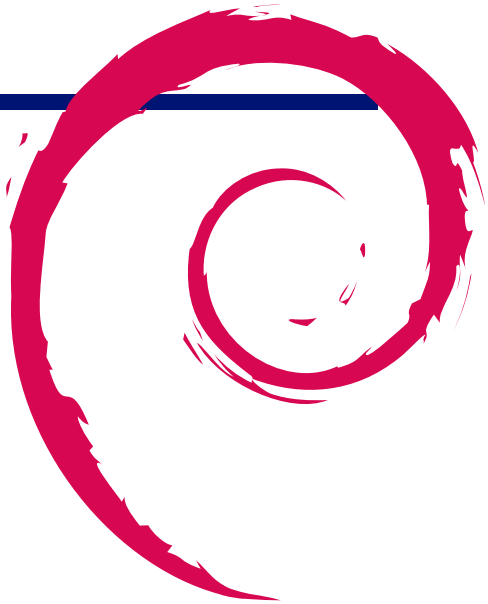
参考文献

[1] RFC2849 "Example 1: An simple LDAP file with two entries" http://www.ietf.org/rfc/rfc2849.txt

[2] OpenLDAP Software 2.4 Administrator's Guide http://www.openldap.org/doc/admin24/index.html

[3] ldapvi User Manual, http://www.lichteblau.com/ldapvi/manual/

[4] python-ldap Documentation, http://www.python-ldap.org/doc/html/index.html



6 gdb python 拡張その 1

野島 貴英

6.1 はじめに

Debian パッケージのバイナリのバグ潰しを行う時、 gdb などのデバッガを使ってデバッグをする事も多いかと思います。しかしながら、プログラムの構造が大規模/複雑になってくると、デバッグ作業も大変になっていきます。また、プログラムについても、抽象化が激しく行われていると、デバッガ無しには解析が非常に困難かと思います。

gdb は v7.0 から python 拡張が搭載されており[*7]、 gdb を python スクリプトで操る事ができます。こちらを利用する事により、人手では、工数かかりすぎでなかなか進まなかったデバッグも可能になっていくと思います。

今回は、 gdb の python 拡張の一部を説明し、最後は C 言語で作られたプログラムの実行トレースを実際に python 拡張機能を用いて、取得してみるところまでやってみます。

6.2 Debian sid の gdb python 拡張の基本情報

debian sid に含まれている gdb は最初から python 拡張が有効にしてあるようです。表 1 に基本情報を載せます。

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

表 1　Debian sid 環境に含まれる gdb と python の基本情報

今回は debian sid amd64 環境でテストした結果を元に説明を行います。

6.3 python 拡張を簡易的に動かしてみる

gdb より python 言語を簡易的に呼び出す場合、 gdb のコマンドラインから接頭辞として python をつけて起動します。また、 gdb コマンドラインから python のみを指定すると、 Ctrl+d を押下するまで、複数の python 構文を指定できます。(なお、 Ctrl+d を押下すると、 gdb のコマンドラインに復帰すると同時に入力した複数の python 構文が一度に評価されます)

なお、 python で一度定義した変数などは gdb を終了するまで何度でも参照できます。

[*7] http://ftp.gnu.org/gnu/gdb/ 調べ

```
$ gdb
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
... 中略...
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(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', '/usr/lib/python2.7/lib-tk',
  '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload',
  '/usr/local/lib/python2.7/dist-packages',
  '/usr/lib/python2.7/dist-packages',
  '/usr/lib/python2.7/dist-packages/PIL',
  '/usr/lib/python2.7/dist-packages/gst-0.10',
  '/usr/lib/python2.7/dist-packages/gtk-2.0',
  '/usr/lib/pymodules/python2.7']
sys.version_info(major=2, minor=7, micro=3, releaselevel='final', serial=0)
```

6.4 gdb の python 拡張の構造

gdb の python 拡張は図 1 のような構造となっています。

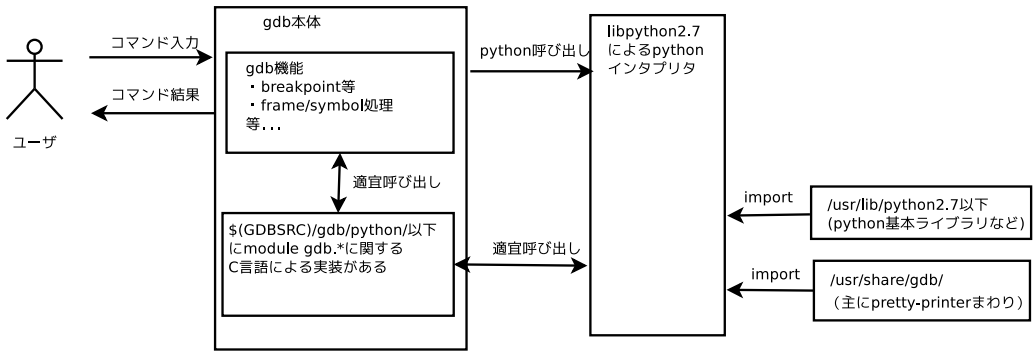


図 1　gdb と python 拡張の構造

6.5 module gdb のマニュアル

gdb 本体に module gdb が実装されているため、 module gdb の python ドキュメントについては、 gdb 上で python から help(gdb) を呼び出す必要があります。

以下に読み方を示します。なお、以降、 (gdb) は gdb のプロンプトを示します。

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

NAME
    gdb

FILE
    (built-in)

PACKAGE CONTENTS
    command (package)
    printing
    prompt
    types

... 中略...
```

また、 gdb の python 拡張についてのさらに詳しい説明は、 info gdb にて Extending GDB → python の項目から参

照できます。

6.6 module gdb に定義されているオブジェクト群

図 2〜図 3 に、 module gdb に定義されているオブジェクト群の class 図を載せます。

gdb 拡張用の python スクリプトを記載する場合、これらオブジェクトを併用して gdb とのデータのやりとり、あるいは、操作を行います。

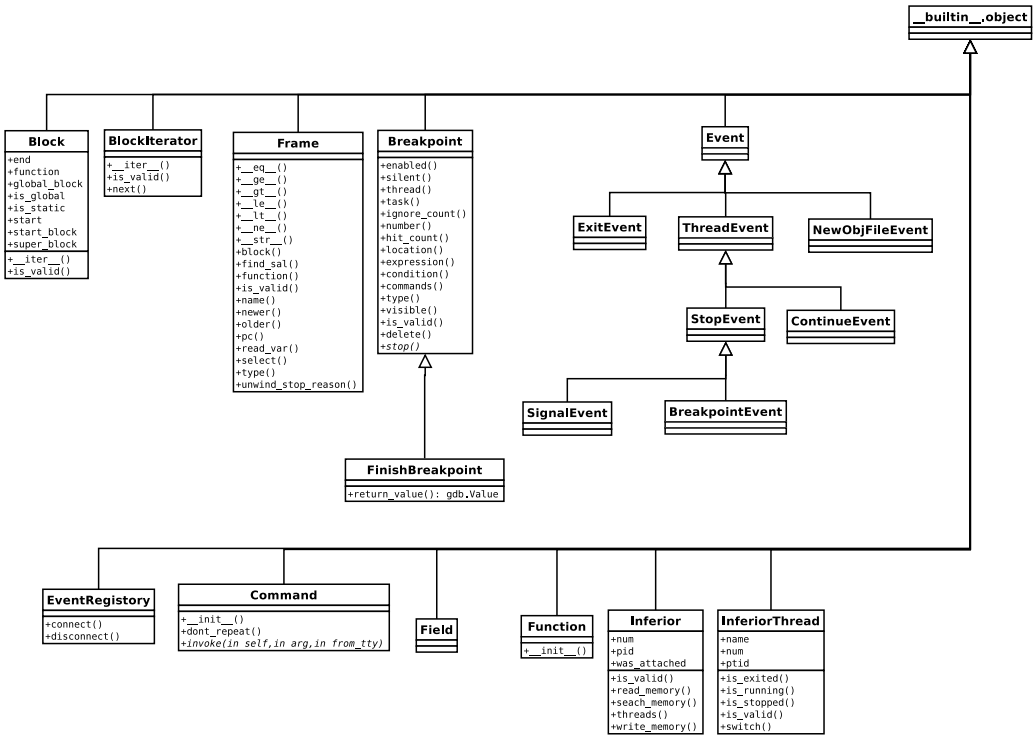


図 2　module gdb の class 図 (その 1)

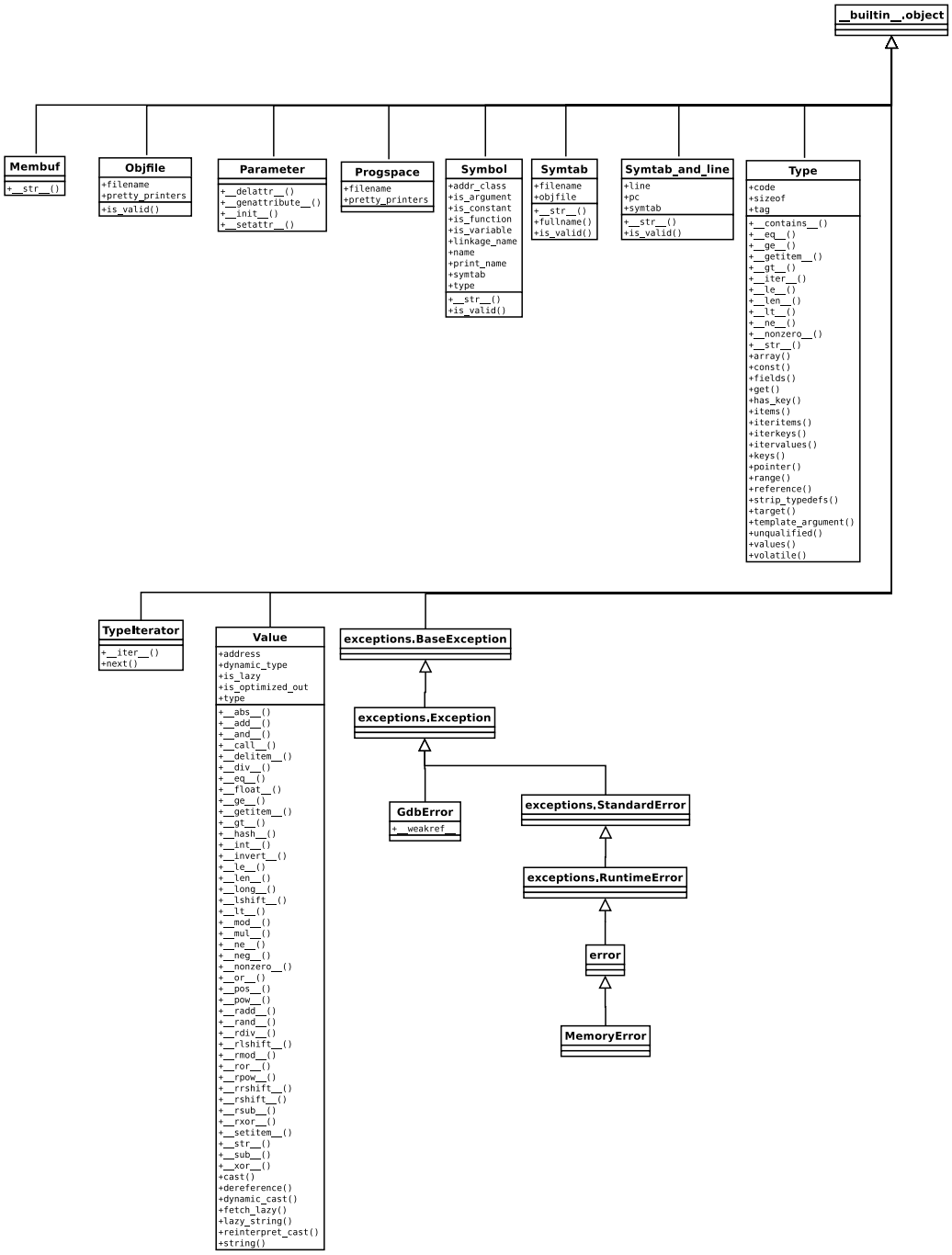


図 3　module gdb の class 図 (その 2)

6.7　gdb のコマンドを増やしてみる

gdb の python 拡張は柔軟な機能を持つため、いろいろな使い方ができます。ここでは、試しに gdb のコマンドを増やしてみます。

gdb のコマンドを python から増やすには、gdb.command クラスを継承したクラスを用意し、gdb.command.__init__() にてコマンド名と共に登録する事により行います。

info gdb の Extending GDB → Python → Python API → Commands In Python に記載されている方法を試して、hello-world コマンドを登録してみます。

```
-----hello.py の中身ここから-----
# -*- coding: utf-8 -*-
# coding:utf-8
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()
-----hello.py の中身ここまで-----
```

追加したコマンドについていろいろ実行してみます。

```
(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...
```

コマンド hello-world が追加されています。また、引数は invoke() の arg に文字列としてまとめて入ります。また、コマンドカテゴリの OBSCURE に登録されている事が help obscure にて判ります。

6.8 作った python スクリプトを自動で読み込ませるには

ところで、 gdb の python 拡張を理解するにつれ、高度なデバッグ用スクリプトを用意するようになってくると思います。すると、作った python スクリプトをいつも gdb に自動で読み込ませておきたくなるかと思います。方法としては、以下の 3 つの方法があります。

6.8.1 ${HOME}/.gdbinit を使う方法

以下のようなファイルを`${HOME}/.gdbinit` に記載しておきます。

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

こうすると、`${HOME}`にホームディレクトリがあるようなユーザが gdb を起動した時に、自動的に/home/foo/bar/my-gdb-func.py がロードされて評価されるようになります。

6.8.2 セクション名: .debug_gdb_scripts を使う方法

バイナリ形式によりますが、任意のセクション名を持つ事が可能なバイナリ形式(例: ELF,DWARF) にて、.gdb_gdb_scripts セクションを作成し、ここにスクリプト名を打ち込んでおく事ができる場合があります。この場合、カレントディレクトリにある、同名のスクリプトを gdb が自動的にロードしてくれます。なお、本機能は、 gdb 変数の auto-load-script が on の時に有効です(デフォルトは on。)

以下に例を示します。ここでは先ほどの hello.py を自動でロードするように asm{} 命令で.debug_gdb_scripts セクションを直接指定しています。

```
------hello.c の中身ここから------
#include <stdio.h>

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

int main(int argc,char **argv)
{
        printf("hi there!");
        return 0;
}
------hello.c の中身ここまで------

実行結果:
$ gcc -o hello hello.c
$ ls
hello hello.py hello.c
$ gdb hello
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
... 中略...
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/xxxx/hello...done.
(gdb) info auto-load-scripts
Loaded  Script
Yes     hello.py
        full name: /home/xxxx/hello.py
(gdb) hello-world
Hello, World! arg=[]
```

6.8.3 “バイナリ名”-gdb.py をスクリプト名に使う方法

ファイル名として、“バイナリ名”-gdb.py をファイル名に持つ python スクリプトをカレントディレクトリに置いておくと、gdb がバイナリ名のファイルをロードした時、自動でロードしてくれます。なお、本機能は、gdb 変数の auto-load-script が on の時に有効です(デフォルトは on)

以下の例では、gdb hello とすると、無事先ほどの hello.py が読み込まれ、hello-world コマンドが登録されている事がわかります。

```
------hello.c の中身ここから------
#include <stdio.h>

int main(int argc,char **argv)
{
        printf("hi there!");
        return 0;
}
------hello.c の中身ここまで------

コンパイル:
$ gcc -o hello hello.c
$ mv hello.py hello-gdb.py (←先ほどの hello.py の名前を"バイナリ名"-gdb.py へ変更)
$ ls
hello hello-gdb.py hello.c
$ gdb hello
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
... 中略...
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/xxxx/hello...done.
(gdb) info auto-load-scripts
Loaded  Script
Yes     /home/xxxx/hello-gdb.py
(gdb) hello-world
Hello, World! arg=[]
```

6.9 break/finish と応用例について

デバッガの基本機能に breakpoint があります。こちらの機能を python から利用するには gdb.Breakpoint class 及び、gdb.FinishBreakpoint class を継承する事により行います。これら class を用いれば、gdb の break/watch/finish コマンドを独自拡張できます。

ここでは、応用としてバイナリ内部の関数呼び出しの記録を取るような python スクリプトを書いてみます。

```
-------calltracer.py ここから---------
# -*- coding: utf-8 -*-
# coding:utf-8
import gdb
class _CallTracerFinishBreakpoint(gdb.FinishBreakpoint):
        def __init__(self, name, stack):
                super(_CallTracerFinishBreakpoint, self).__init__(internal=True)
                self._stack_ptr=stack
                self._name=name
                self.silent=True
        def stop(self):
                print (" " * (len(self._stack_ptr)))+"<="+self._name
                self._stack_ptr.pop()
                return False
        def out_of_scope(self):
                print "Abnormal jump out frame"
                print (" " * (len(self._stack_ptr)))+"<="+self._name
                self._stack_ptr.pop()
                return False

class _CallTracerBreakpoint(gdb.Breakpoint):
        def __init__(self, spec, name, stack):
                super(_CallTracerBreakpoint, self).__init__(spec,
                                                            gdb.BP_BREAKPOINT,
                                                            internal = False)
                self._stack_ptr=stack
                self._name=name
                self.silent=True
        def stop(self):
                self._stack_ptr.append(self._name)
                print (" " * (len(self._stack_ptr)))+"=>"+self._name
                try:
                        _CallTracerFinishBreakpoint(self._name, self._stack_ptr)
                except:
                        print "uh? cant put finish break on "+self._name
                return False

class _ReAnalyzeCallTracer(gdb.Command):
        """ reanalyze symbol for calltracer """
        def __init__(self):
                super(_ReAnalyzeCallTracer, self).__init__('reanalyzecalltracer',
                                                    gdb.COMMAND_OBSCURE)
                self._stack=[]
        def _retrive_ptrs(self):
                info=gdb.execute("info break",False, True)
                info_lines=info.splitlines()
                ptrs={}
                for idx in range(0,len(info_lines[1:])):
                        tokens=info_lines[idx+1].split()
                        if len(tokens) > 5:
                                if ptrs.has_key(tokens[4]) == False:
                                        ptrs[tokens[4]]=" ".join(tokens[5:])
                return ptrs
        def invoke(self, arg, from_tty):
                break_info=self._retrive_ptrs()
                gdb.execute("delete",False, True)
                gdb.execute("set pagination off")
                for addr,name in break_info.iteritems():
                        _CallTracerBreakpoint('r'*'+addr,
                                              name,self._stack)
_ReAnalyzeCallTracer()

class _PrepareCallTracer(gdb.Command):
        """ prepare call tracer for c """
        def __init__(self):
                super(_PrepareCallTracer, self).__init__('prepcalltracer',
                                                    gdb.COMMAND_OBSCURE)
        def invoke(self, arg, from_tty):
                gdb.execute("rbreak",False, True)
                gdb.execute("reanalyzecalltracer",False, True)
                print "prepare done!"

_PrepareCallTracer()
-------calltracer.py ここまで---------
-------デバッグ対象: chkfunc.c ここから-------
#include<stdio.h>

void foo_a(const char *str)
{
        printf("%s\n",str);

}
void caller_bar(void)
{
        foo_a("caller is bar!");
}
int main(int argc,char **argv)
{
        foo_a("caller is main!");
        caller_bar();
        return(0);
}
-------デバッグ対象: chkfunc.c ここまで-------
```

実行結果:

```
$ gcc -O0 -g -o chkfunc chkfunc
$ ls
chkfunc.c chfunc calltracer.py
$ gdb ./chkfunc
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
... 中略...
Reading symbols from /home/foo/bar/chkfunc...done.
(gdb) source calltracer.py
(gdb) prepcalltracer
prepare done!
(gdb) run
Starting program: /home/foo/bar/chkfunc
 =><_start>
uh? cant put finish break on <_start>
  =><__libc_start_main@plt>
   =><__libc_csu_init>
    =><_init>
     =><call_gmon_start>
     <=<call_gmon_start>
    <=<_init>
    =><frame_dummy>
     =><register_tm_clones>
     <=<frame_dummy>
    <=<register_tm_clones>
   <=<__libc_csu_init>
   =>in main at chkfunc.c:21
uh? cant put finish break on in main at chkfunc.c:21
    =>in foo_a at chkfunc.c:12
     =><puts@plt>
caller is main!
     <=<puts@plt>
    <=in foo_a at chkfunc.c:12
    =>in caller_bar at chkfunc.c:17
     =>in foo_a at chkfunc.c:12
      =><puts@plt>
caller is bar!
      <=<puts@plt>
     <=in foo_a at chkfunc.c:12
    <=in caller_bar at chkfunc.c:17
    =><__do_global_dtors_aux>
     =><deregister_tm_clones>
     <=<deregister_tm_clones>
    <=<__do_global_dtors_aux>
    =><_fini>
    <=<_fini>
[Inferior 1 (process 6413) exited normally]
Abnormal jump out frame
   <=<__libc_start_main@plt>
warning: Error removing breakpoint -5
(gdb)
```

無事、バイナリ内部の関数呼び出しの記録が取れているかと思います。

6.10 終わりに

今回 gdb の python 拡張のいくつかを紹介してみました。 python を用いる事で、いろいろなデバッグ手法が取れるかと思います。

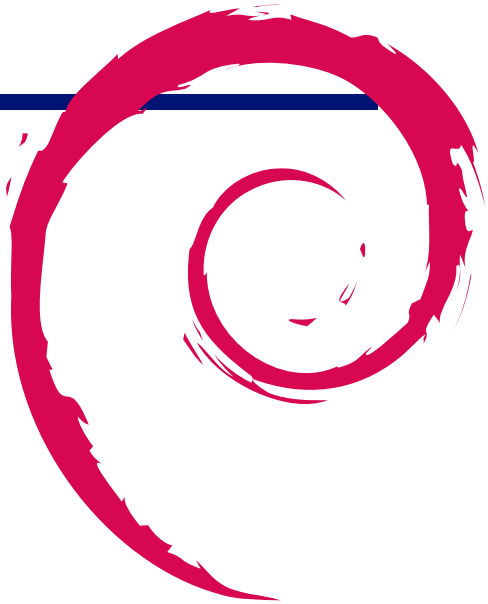
次回は、 Frame class/Value class 等の応用について紹介したいと思います。

参考文献

[1] Free Software Foundation, “info gdb”

[2] “PythonGdbTutorial”,`http://sourceware.org/gdb/wiki/PythonGdbTutorial`

[3] Free Software Foundation, `$(GDBSRC)/gdb/testsuite/gdb.python` 以下のテスト用ファイル群



7 月刊 Debhelper dh_auto_install dh_install

吉田 俊輔

7.1 今月のコマンド

- dh_auto_install
- dh_install

7.2 debian パッケージ構築、全体の流れ

2011 年 10 月勉強会資料より

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

7.3 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

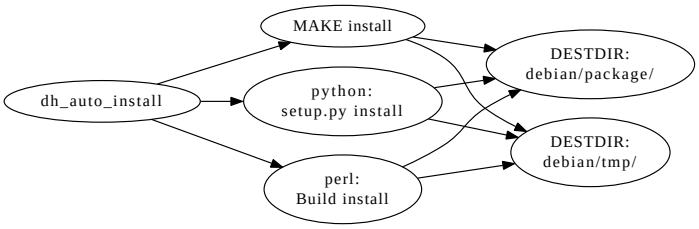
- 以下の debhelper コマンドが実行されます。

```
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
```

7.4 dh_auto_install 動作解説

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。



- 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`
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`

7.5 override 例

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)

7.6 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。

7.7 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

7.8 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

7.9 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。 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。

7.10 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。

7.11 –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

7.12 –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。

7.13 -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。

7.14 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)
```

7.15 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)
```

7.16 DEBHELPER のプログラム共有のオプションその他

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.
–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.

Debian 勉強会資料

2013 年 3 月 16 日　　初版第 1 刷発行

東京エリア Debian 勉強会（ 編集・ 印刷・ 発行）