Grand Unified Debian

銀河系唯一のDebian専門誌

～大統一Debian勉強会2013特集号～

東京エリア Debian **勉強会/関西** Debian **勉強会/福岡** Debian **勉強会**

– 2013/06/29 **初版発行**

デビアン勉強会

目次

1 はじめに ~ 大統一 Debian 勉強会 2013 開催概要

「 大統一 Debian 勉強会 2013」実行委員会

開催趣意書

Debian JP Project では 2005 年 1 月から東京地域を中心とした Debian に関する勉強会「 東京エリア Debian 勉強会」、 2007 年 3 月からは関西地域を中心とした「 関西エリア Debian 勉強会」を毎月、 2013 年 1 月からは九州福岡地域で「 福岡 Debian 勉強会」を行なっています。

Debian 勉強会の大きな目的として、

- Debian Developer の育成
- 普段全国各地にいる人々が face-to-face で出会える場を提供する

があります。

昨年 (2012 年)、各地の Debian 勉強会が京都に集結し、 Debian ユーザ、 Debian 開発者の交流を目的とする「 大統一 Debian 勉強会」を開催し大盛況に終わりました。今年は場所を東京に移し、大統一 Debian 勉強会 2013 を開催します。

2013 年度の本イベントのテーマは

「 Level up Debian」

です。

今回のイベントに参加することによって、 Debian の新しい使い方を知り、一歩進んだ活用ができるようになることを考えています。

2012 年に開催した大統一 Debian 勉強会の成功を受け、よりよいイベントの開催と世界各国で毎年開催されている DebConf（ Debian カンファレンス）[*1] を日本で開催することを目指しています。

[*1] http://debconf.org/

本資料に関して

会場で頒布される本資料は大統一 Debian 勉強会 2013 のスポンサーのご支援により、出版できました。スポンサーの皆様に感謝します。ありがとうございました。

スポンサー一覧 (敬称略)

- プラチナスポンサー
  - 株式会社マイクロソフト
- ゴールドスポンサー
  - ぷらっとホーム株式会社
  - 株式会社トップスタジオ
  - 株式会社サーバーワークス
- シルバースポンサー
  - ミラクル・リナックス株式会社
  - 株式会社サイバーエージェント
  - さくらインターネット株式会社
- メディアスポンサー
  - アイティメディア株式会社
  - 株式会社技術評論社
  - gihyo.jp
  - sourceforge.jp
  - 株式会社アスキー・メディアワークス
- ウェブサイトスポンサー
  - ANNAI LLC
- 協賛
  - 会場提供: 日本大学
  - ペットボトル（ミネラルウォーター） 提供: 株式会社サイバーエージェント

2 debhelper におけるプログラミング言語ビルドツール対応の仕組みについて

岩松 信洋

Debian は多くのプログラミング言語をサポートしており、それらを使ったソフトウェアが多く存在します。 Debian の各プログラミング言語サポートチームはパッケージのメンテナンスを軽減するために パッケージ作成サポートツールの一つである debhelper 向けツールを作成し、提供しています。一部で流行っているプログラミング言語 Erlang ですが、もちろん Debian でもサポートされています。最近の Erlang アプリケーションのほとんどは rebar というビルドサポートツールを使ってビルドされますが、まだ rebar をサポートした debhelper 向けツールがありません。よって rebar を使っているアプリケーションを Debian パッケージにする場合には必要な処理を debian/rules に記述する必要があります。私は rebar と rebar を使った Erlang アプリケーションパッケージをメンテナンスしています。今回 rebar を使った Erlang プログラムを他のプログラミング言語同様に、容易に Debian でサポートできるように、 rebar の debhelper 実装 `dh-rebar` を開発することにしました。本章ではこのツール `dh-rebar` についての解説と debhelper におけるプログラミング言語対応の仕組みについて説明します。

2.1 rebar とは

まず、 rebar の仕組みについて簡単に説明します。 reabr はプログラミング言語 Erlang のビルドツールです。 Riak の開発元として有名な Basho[*2] によって開発が行われています。 rebar は Erlang を使ったビルドやテスト、ドキュメント作成からプロジェクトのひな形作成までサポートしており、今では多くの Erlang アプリケーションで利用されています。 rebar によって提供されている主要なコマンドを以下に示します。

- rebar compile : Erlang プログラムをコンパイルする。
- rebar clean : コンパイル時に作成された一時ファイルの削除する。
- rebar eunit : テストを実行する。
- rebar doc : ドキュメントの作成する。
- rebar xref : モジュールや関数の依存関係のチェックを行う。
- rebar get-deps : ビルドに必要なライブラリなどのソースコードを取得し、コンパイルする。

これらのコマンドは単体で呼ぶこともできますが、多くの場合ソースコードに Makefile が含まれており、 Makefile から rebar の各コマンドが利用する形になっています。またインストールに関する処理は用意されておらず、ビルドされたものをそのまま参照するか、ユーザがインストール処理を行う必要があります。図 1 に Makefile の例を示します。

Makefile の例を見てもわかるように、 rebar は相対パスで指定されていることが多く、 rebar のバイナリをソースコードに含めるている事がほとんどです。この理由として、過去に rebar の仕様が頻繁に変更されたことがあり、そのたびにビルドできないという問題があったようです。その対策のためにソースコードに rebar バイナリを含めるようになったとのことです。そのため、 Debian パッケージにする場合にはソースコードから rebar バイナリを削除する作業が必要とな

[*2] `http://basho.com/`

```
(省略)
deps:
    @./rebar get-deps

app:
    @./rebar compile
(省略)
```

図 1　Makefile 例

ります。

2.1.1　rebar を使った Debian パッケージの作業

上記のようなビルドシステムである rebar を使っているソフトウェアを Debian パッケージにする場合、主に以下のような作業が必要になります。

1. rebar バイナリをソースから削除し、ソースコードをリパックする。パッケージのバージョンに +dfsg をつける。`debian/README.source` にこれについて記載する。
2. dh_make を実行して雛形を作成する。
3. override_dh_auto_build build が呼ばれた時には rebar compile、 override_dh_auto_test が呼ばれた時に rebar test、 override_dh_auto_clean が呼ばれた時に rebar clean 実行するなどの処理を debian/rules に追加する。
4. Debian の Erlang パッケージはプレフィックスに erlang- をつける。
5. 作成されたバイナリを usr/lib/erlang/lib/$(PACKAGE)-$(VERSION)/以下にインストールされるように処理する。
   PACKAGE には元のパッケージ名、 VERSION には元のパッケージのバージョンが指定される。例えば erlang-foo の Debian バージョン 1.0.0+dfsg-1 の場合は PACKGE に foo、 VERSION には 1.0.0 が入る。
6. .beam と .erl は 圧縮されないように dh_compress で無視するように設定する。

上記の作業を行った後の `debian/rules` は図 2 のようになります。

```
#!/usr/bin/make -f

PACKAGE = $(shell dpkg-parsechangelog | sed -rne 's/^Source: erlang-(.*)/\1/p')
VERSION = $(shell dpkg-parsechangelog | sed -rne 's/^Version: ([0-9.]+)(\+dfsg)?.*$$/\1/p')

%:
    dh $@

override_dh_auto_build:
    rebar compile -vv skip_deps=true

override_dh_auto_clean:
    dh_auto_clean
    rm -rf ebin

override_dh_auto_install:
    dh_auto_install

    install -d \
        $(CURDIR)/debian/erlang-$(PACKAGE)/usr/lib/erlang/lib/$(PACKAGE)-$(VERSION)/ebin
    install -m 644 ebin/* \
        $(CURDIR)/debian/erlang-$(PACKAGE)/usr/lib/erlang/lib/$(PACKAGE)-$(VERSION)/ebin/
    install -d \
        $(CURDIR)/debian/erlang-$(PACKAGE)-dev/usr/lib/erlang/lib/$(PACKAGE)-$(VERSION)/include
    install -m 644 include/* \
        $(CURDIR)/debian/erlang-$(PACKAGE)-dev/usr/lib/erlang/lib/$(PACKAGE)-$(VERSION)/include/

override_dh_gencontrol-arch:
    erlang-depends -v -a -perlang-$(PACKAGE)
    erlang-depends -v -a -perlang-$(PACKAGE)-dev
    dh_gencontrol -a -perlang-$(PACKAGE)
    dh_gencontrol -a -perlang-$(PACKAGE)-dev

override_dh_compress:
    dh_compress -X.erl -X.beam
```

図 2　debian/rules 例

Debian パッケージビルドサポートツールの一つである debhelper のバージョン 7（ 以下、 debhelper7）の機能を使っているにも関わらず、 35 行と比較的長い `debian/rules` になっています。ざっと見てみると、共通化できる部分が

いくつかありそうです。共通化できる部分をまとめるともっと短い debian/rules になりますし、コーディングミスも減ることが考えられます。また、仕様変更や問題があった場合にも 共通部分を修正するだけで対応できるようになります。そこで共通部分を debhelper の機能 dh-rebar として提供することにしました。次に debhelper による rebar を使った Erlang プログラムのサポートについて説明します。

2.2 debhelper によるプログラミング言語のサポート

debhelper7 からビルドシステムを追加する機能 Buildsystem とビルドシーケンスをコントロールする機能 Sequence が追加されました。これらの機能には、オーバライドできる機能もあるため、これを使うことによって容易にプログラミング言語やツールのサポートができるようにもなっています。 Buildsystem と Sequence の概要と rebar での実装方法について実際のコードを使って説明します。

2.2.1 Buildsystem / ビルドシステム

Buildsystem では clean、 configure、 build、 install、 test 関数が提供され、デフォルトではシンプルな Makefile をターゲットにした処理が行われるようになっています。これらの関数はオーバーライドすることができ、各々に必要な処理を新しい Buildsystem として記述することによって、プログラミング言語などをサポートできるようになってます。記述した Buildsystem を/usr/share/perl5/Debian/Debhelper/Buildsystem/ にインストールし、 dh コマンド(debhelper) の --buildsystem オプションで指定することによってパッケージビルド時に利用されるようになります。利用できる Buildsystem は dh_auto_build の --list オプションで確認できます(図 3)。

```
$ dh_auto_build --list
perl_makemaker       Perl ExtUtils::MakeMaker (Makefile.PL)
makefile             simple Makefile
rebar                rebar [3rd party]

Auto-selected: makefile
```

図 3 dh_auto_build --list 出力例

次に、 Buildsystem で提供される関数をオーバーライドする例を図 4 に示します。例では build 関数をオーバーライドし、$DH_REBAR_MAKEFILE で指定されている dh-rebar.Makefile の build ターゲットが実行されるようになっています。これにより、 debian/rules から build ターゲットが呼ばれた時に、新しく作成した rebar Buildsystem の build 関数が呼ばれるようになります。

```
package Debian::Debhelper::Buildsystem::rebar;

use strict;
use base 'Debian::Debhelper::Buildsystem';

my $DH_REBAR_MAKEFILE="/usr/share/dh-rebar/make/dh-rebar.Makefile";

sub DESCRIPTION { "rebar" }

( 省略)

sub build {
    my $this=shift;
    $this->doit_in_sourcedir("make", "--no-print-directory", "-f", $DH_REBAR_MAKEFILE, "build", @_);
}

( 省略)
```

図 4 Buildsystem オーバライド例

次に Buildsystem から実行される dh-rebar.Makefile について説明します。内容の一部を図 5 に示します。

dh-rebar.Makefile では build ターゲットが呼ばれた時に$(BUILD_TARGET) が実行されるようになっており、$(EXEC_REBAR_COMMANDS) の内容によって変化します。$(EXEC_REBAR_COMMANDS) に compile がある場合は BUILD_TARGET に rebar_compile が追加され、$(EXEC_REBAR_COMMANDS) に xref がある場合は BUILD_TARGET に rebar_xref が追加されます。そして rebar_compile では rebar の compile、 rebar_xref では xref が実行されるようになっています。$(EXEC_REBAR_COMMANDS) などの変数は debian ディレクトリにある

```
( 省略)
include debian/dh-rebar.conf
( 省略)
# build
ifneq (,$(findstring xref, $(EXEC_REBAR_COMMANDS)))
BUILD_TARGET+=rebar_xref
endif
ifneq (,$(findstring compile, $(EXEC_REBAR_COMMANDS)))
BUILD_TARGET=rebar_compile
endif

build: $(BUILD_TARGET)

rebar_compile:
    $(H)echo $@
    rebar compile skip_deps=true -v

rebar_xref:
    $(H)echo $@
    rebar xref skip_deps=true -v
( 省略)
```

図 5　dh-rebar.Makefile

```
EXEC_REBAR_COMMANDS=compile
PKG_NAME=
PKG_VARSION=
#REBAR_LIB_DIR=
#REBAR_BIN_DIR=
#REBAR_INCLUDE_DIR=
```

図 6　dh-rebar.conf 例

dh-rebar.conf（ 図 6）を参照するようになっています。このファイルによって、実行される Makefile のターゲットやバイナリがインストールされる参照先や、実行する Makefile のターゲットをコントロールできるようになっています。

rebar の場合は、ソフトウェアのビルドやテストがコマンド毎に決まっているのでその動作に合わせて dh-rebar.Makefile を記述しました。 Buildsystem からは 図 7 のような流れでビルドを行うようになっています。

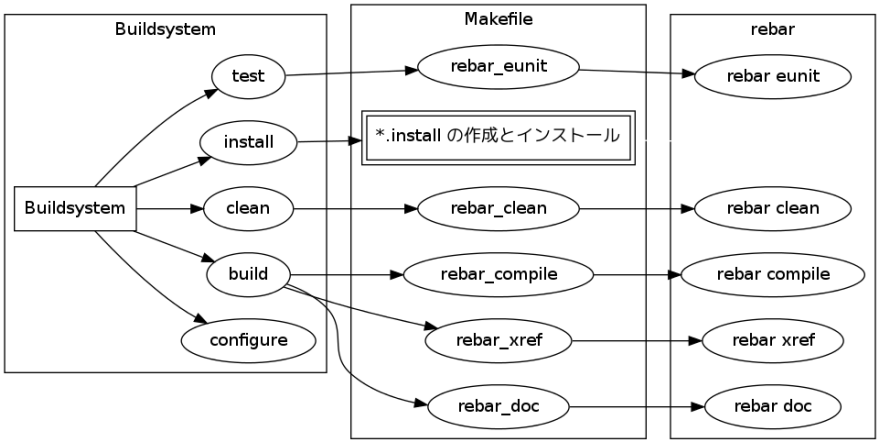


図 7　全体の流れ

また、今回は Buildsystem から Makefile を呼ぶようにしましたが、代わりにシェルや python などのスクリプトで実装してもかまいません。

2.2.2　Sequence / シーケンス

Sequence は dh 内で実行される dh コマンド群のパッケージビルドシーケンスをコントロールします。 dh では実行されるパッケージビルドシーケンスとして、決まったシーケンスを持っており何もしない場合にはこの決まったシーケンスが実行されます。 Sequence 機能を使うと、ある dh コマンドを実行しないよう抑制したり、ある dh コマンドの前後に独自の処理を挿入できるようになります。

Erlang の場合はソースコードやバイナリのサフィックスに.erl や.beam が付与されているのですが、これらのファイ

ルをパッケージにする場合に圧縮させたくありません。圧縮すると Erlang のプログラムとして利用できないためです。よって Erlang パッケージの場合は必ず dh_compress (パッケージ内のファイルを圧縮するコマンド) を実行する時にこれらのファイルを無視するように設定します。これには add_command_options を使って、 dh_compress にオプションを設定します。

```
add_command_options("dh_compress", "-X.erl -X.beam");
```

また、ある debhelper コマンドの前後に別の debhelper のコマンドを実行させたい場合があります。この場合 insert_before または insert_after を使って、実行したいコマンドを指定します。

Erlang パッケージ間の依存関係をチェックし、バイナリパッケージの control ファイルに記述するツール erlang-depends があるのですが、これを dh_gencontrol[*3]の前に呼ぶ必要があります。 erlang-depends は debhelper7 に対応していないので、ラッパーした dh_rebar を作成し、以下のように記述しました。

```
insert_before("dh_gencontrol", "dh_rebar");
```

このような設定を記述したファイル(図8) を rebar.pm として、 usr/share/perl5/Debian/Debhelper/Sequence/ にインストールします。そして dh のオプション --with で指定することによりファイル内で指定したシーケンスがパッケージビルド時に有効になります。

```
#! /usr/bin/perl

use warnings;
use strict;
use Debian::Debhelper::Dh_Lib;

insert_before("dh_gencontrol", "dh_rebar");

add_command_options("dh_compress", "-X.erl -X.beam");

1;
```

図8　Sequence ファイル

2.3　dh-rebar を使った debian/rules

上記で説明した dh-rebar で提供される Buildsytem と Sequence を使って debian/rules を書きなおすと、以下のようになります。 35 行が 3 行になり、シンプルになりました。また、各種設定は debian/dh-rebar.conf にまとめられるため、 debian/rules も特に編集する必要はなくなります。

```
#!/usr/bin/make -f
%:
    dh $@ --buildsystem=rebar --with rebar
```

2.4　まとめ

debhelper7 から提供されている Buildsystem と Sequence を使うと、 Debian パッケージにする時に必要な機能をまとめることができます。また、まとめられた機能を使うことによって容易にパッケージにできるようになり、メンテナンス性もよくなります。今後新しいプログラミング言語やビルドツールをサポートするときの手助けになれば幸いです。

[*3] debian/control を変換しインストールする debhelper ツールの一つ

3 gdb+python 拡張を使ったデバッグ手法

野島 貴英

debian 付属の gdb はすでに python 拡張が有効になっている状態でパッケージ化されています。今回の発表では、debian に用意されている機能をフル活用して、 PHP5 の PHP 言語レベルのソースデバッグを gdb+python をつかってやってみる事について述べます。

なお、ここでは、 debian sid(jessie/sid) を用い、パッケージから導入した gdb 7.6、 PHP 5.5.0rc3 を利用しています。

3.1 gdb+python 拡張今までのおさらい

gdb は、バイナリ形式の実行ファイルをデバッグするのに、極めて強力なシンボリックデバッガです。 gdb は python を内臓することにより、さらに強力なデバッグが可能になっています。 debian では squeeze から有効になっています。

東京エリア debian 勉強会の 2013 年 3 月にては、 gdb+python 拡張に搭載されている gdb.Command/ gdb.Breakpoint/ gdb.FinishBreakpoint を使い、 C 言語による実行ファイルの関数トレースを取るところまで紹介しました [1]。

3.2 debian の*-dbg パッケージについてちょっとだけ

debian には、バイナリパッケージのデバッグ用シンボル情報だけを集めた*-dbg パッケージ群が提供されています。利用の仕方は簡単で、デバッグしたいバイナリパッケージの名前に-dbg と付いたパッケージを導入し、 gdb を動かすだけです。

```
% sudo aptitude install 'php5-dbg'
% apt-get source php5
% gdb --args /usr/bin/php5 -r 'phpinfo();'
... 中略...
(gdb) b main
Breakpoint 1 at 0x4640a0: file /tmp/buildd/php5-5.5.0~rc3+dfsg/sapi/cli/php_cli.c, line 1200.
(gdb) set substitute-path /tmp/buildd/ ./
(gdb) run
Starting program: /usr/bin/php5 -r phpinfo\(\)\;
... 中略...
Breakpoint 1, main (argc=3, argv=0x7fffffffe668)
    at /tmp/buildd/php5-5.5.0~rc3+dfsg/sapi/cli/php_cli.c:1200
1200        {
(gdb) l (←現在実行中のソースコードを閲覧)
1195        #ifdef PHP_CLI_WIN32_NO_CONSOLE
1196        int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
1197        #else
1198        int main(int argc, char *argv[])
1199        #endif
1200        {
1201        #ifdef ZTS
... 中略...
```

(なお、上の例ではソースコード指定に "set substitution-path" を利用しましたが、現状では必ずしもこれでうまく行くものばかりではない事を後に述べます。)

なお、*-dbg パッケージに関しての現状と詳細は 2012 年の大統一 Debian 勉強会の岩松さんの発表"debug.debian.net"[2] に詳しいです。ここでは、そこで触れられていなかった件をいくつか載せておきます。

3.2.1 ソースコードのパスの指定について

*-dbg パッケージを利用して gdb からソースコードを指定するための指定方法は、以下の 2 種類の方法を適宜選んで指定します。

- gdb の “set substitute-path” で指定する方法
  これは、パッケージがビルドされる際、 gcc に対してソースがフルパスで指定されているときに使える方法となります。先ほどの PHP5 の例のように、一つの”set substitute-path” を指定するだけで、ソースのサブディレクトリに配置されているソースコードも正確に自動で gdb が追跡してくれますので、デバッグ時のソース閲覧に大変都合が良いです。
- gdb の “dir” でソースの位置を複数指定する方法
  これは、パッケージがビルドされる際、 gcc に対してファイル名のみ指定されてコンパイルされている場合に使います。こちらの場合、*-dbg パッケージのシンボルにソースのパスを示す情報が欠落している為、ソースの位置は”dir” コマンドでヒントとして指定することになります。 gdb はユーザが”dir” コマンドで指定したソースの存在するパス全部を記憶しており、このパスに同名のファイルがあるかを調べ、最初に合致したものを表示します。つまり、完全に同名のファイルが複数のサブディレクトリに配置されていたものをビルド及びリンクした場合は、gdb は自動では正しいソースファイル上での位置を表示できない為、注意が必要です。

3.2.2 debhelper の COMPATIBILITY LEVEL 9 から搭載されたデバッグシンボルファイルの形式について

パッケージ構築の際、 debhelper の COMPATIBILITY LEVEL 9 から、導入されたデバッグ用シンボルファイルのファイル名は、パッケージの実行ファイルのバイナリに埋め込まれている BuildID [3] というハッシュ値を元につけられています。以前の形式では、実行バイナリの名前がそのままデバッグ用シンボルファイルとなっていました。

```
#↓ COMPATIBILITY LEVEL 9 形式の場合
$ file /usr/lib/x86_64-linux-gnu/libgstreamer-0.10.so.0.30.0
/usr/lib/x86_64-linux-gnu/libgstreamer-0.10.so.0.30.0: ELF 64-bit LSB
shared object, x86-64, version 1 (SYSV), dynamically linked,
BuildID[sha1]=44ff321f11ffd750f8c351ffa3f5d20028d2f6a6, stripped
# 表示された BuildID の、上2桁及び残りの桁をもちいて
# /usr/lib/debug/.build-id/上 2 桁/残りの桁.debug となるファイルが
#デバッグ用シンボルファイルとなる。
$ ls /usr/lib/debug/.build-id/44/ff321f11ffd750f8c351ffa3f5d20028d2f6a6.debug
/usr/lib/debug/.build-id/44/ff321f11ffd750f8c351ffa3f5d20028d2f6a6.debug
# 試しに COMPATIBILITY LEVEL 9 形式で作成されたバイナリを gdb でロードしてみる
# gdb 起動メッセージに含まれる Reading symbols from 以下のファイル名のとおり、
# BuildID で決まるシンボルファイルを読みにいく。
$ gdb /usr/lib/x86_64-linux-gnu/libgstreamer-0.10.so.0.30.0
GNU gdb (GDB) 7.6-debian
... 中略...
Reading symbols from /usr/lib/x86_64-linux-gnu/libgstreamer-0.10.so.0.30.0...
Reading symbols from /usr/lib/debug/.build-id/44/ff321f11ffd750f8c351ffa3f5
d20028d2f6a6.debug...done.
(gdb) quit
#↓ COMPATIBILITY LEVEL 9 形式未満の場合
# シンボルファイルは"/usr/lib/debug/"+"デバッグ対象の実行ファイルの絶対パス名"に格納
$ ls -l /usr/lib/debug/usr/bin/php5
-rw-r--r-- 1 root root 22036903  6 月  2 01:44 /usr/lib/debug/usr/bin/php5
```

gdb のようなデバッグ用シンボルファイルを扱うソフトウェアとしては、バイナリの BuildID さえ分かれば、正しいデバッグ用シンボルファイルをすぐに特定できるので都合がよかったりします。従来のように同じファイル名のデバッグ用シンボルファイルを用意するという方法の場合、正しいデバッグ用シンボルファイルであることを保証するには、わざわざシンボルファイルのチェックサムを取って確認するぐらいしかありませんでした [3]。

現在の debian sid では COMPATIBILITY LEVEL は様々な LEVEL のものが混在している状況の為、他のバイナリパッケージでは従来の/usr/lib/debug/”+”デバッグ対象の実行ファイルの絶対パス名”も未だ多数存在しています。

3.3 PHP5.5 の PHP レベルのソースデバッグしてみる

PHP5 系の PHP レベルでのソースデバッグは、すでにかなり強力なものが存在しており、例えば xdebug モジュールを利用し、 vim-nox+debugger plugin/netbean/eclipse 等の DBGp をサポートできるエディタプラグイン/統合環境を

使うことにより、簡単で、かなり強力なデバッグ環境を使う事ができます。しかしながら、ここでは、それでは困るような状況で、どうしても gdb から PHP5 の PHP のソースコードデバッグをせざるを得ない[*4]時にどうするかを検討してみます。

3.3.1 gdb.Value クラス

gdb を使って PHP5 で実行中の PHP のソース上の情報を知る必要があるため、 PHP5 の内部変数にアクセスする必要があります。 gdb の python 拡張ではこういうときに便利な gdb.Value クラスが用意されています。 gdb.Value クラスは、 gdb がアクセスできる変数をそのまま保持でき、様々な参照ができる能力があります。

```
$gdb --args /usr/bin/php5 ./foo.php
... 中略...
(gdb) set substitute-path /tmp/buildd/ /home/yours/php5-src/
(gdb) b zend_vm_execute.h:356
(gdb) run
(gdb) c
Breakpoint 1, execute_ex (execute_data=0x7ffff7e2b0a0)
    at /tmp/buildd/php5-5.5.0~rc3+dfsg/Zend/zend_vm_execute.h:356
356                     if ((ret = OPLINE->handler(execute_data TSRMLS_CC)) > 0) {
(gdb) p executor_globals.current_execute_data
$1 = (struct _zend_execute_data *) 0x7ffff7e2b0a0
(gdb) p executor_globals.current_execute_data->function_state->function->op_array->filename
$1 = 0x7ffff7e5f8d8 "/home/yours/foo.php" (←現在実行中の php ソースファイル名)
(gdb) pi (← gdb 搭載の python をインタラクティブモードにする)
>>> edata=gdb.parse_and_eval("executor_globals.current_execute_data")
>>> print edata['function_state']['function']['op_array']['filename']
0x7ffff7e5f8d8 "/home/nojima/prog/my-works/php5-dbg/test4.php"
```

こちらの例では、 PHP5.5 の内部変数である、 executor_globals.current_execute_data という構造体から、 edata という gdb.Value オブジェクトを gdb.parse_and_eval() を使って生成し、 gdb.Value オブジェクトの dereference() メソッドの能力を用いて一気にポインタを辿って値を取り出してみました。

他に gdb.Value クラスですが、こちらに関数へのポインタを代入すると、 python 側から PHP5.5 の内部関数を呼び出したりもできます。(次の章にて使っています。)

3.3.2 応用

以上を応用して、 PHP5.5 の gdb+python の組み合わせで PHP ソースファイルの実行トレースを取ってみます。なお、今回のスクリプトの実行結果は、

```
表示フォーマット:
"関数スコープ名" in 実行中のファイル名:実行中の行番号
例;
main 関数中の/home/yours/test.php ファイルの 4 行目を実行中の場合は、
"main" in /home/yours/test.php:4
と表示。
```

となります。

[*4] 例: うっかり PHP5 で Daemon 作ったら不具合出ちゃったので、なんとか gdb にて動いているプロセスに attach して PHP5 ソースデバッグしたいとか、あるいは、 PHP プログラムの特定の場所で PHP5 が SEGV するのを検証したい (笑) とか....

```
------php5.5-gdb.py ここから-----------------
# -*- coding: utf-8 -*-

class _Php5ExecuterHook(gdb.Breakpoint):
    """ peeking execution """
    def __init__(self,spec):
        super(_Php5ExecuterHook, self).__init__(spec,
                                                gdb.BP_BREAKPOINT,
                                                internal=False)
        self._phpfile=(gdb.parse_and_eval("zend_get_executed_filename")).dereference()
        self._phpfunc=(gdb.parse_and_eval("get_active_function_name")).dereference()
        self._phplineno=(gdb.parse_and_eval("zend_get_executed_lineno")).dereference()
    def _get_exec_info(self):
        filename=((str(self._phpfile())).partition(' '))[2]
        funcname=((str(self._phpfunc())).partition(' '))[2]
        lineno=int(self._phplineno())
        return (funcname,filename,lineno)

    def stop(self):
        print "%s in %s:%d" % (self._get_exec_info())
        return False

class _SetupAnalyzePhp5Executer(gdb.Command):
    """ setup php5 executer tracer """
    def __init__(self):
        super(_SetupAnalyzePhp5Executer, self).__init__('setupphp5executer',
                                                        gdb.COMMAND_OBSCURE)

    def invoke(self, arg, from_ttyp):
        gdb.execute("set pagination off")
        _Php5ExecuterHook("zend_vm_execute.h:356")

_SetupAnalyzePhp5Executer()
------php5.5-gdb.py ここまで-----------------
$ cat -n test4.php
     1        <?php
     2        function func_a() // 1
     3        {
     4                echo "func_a!\n";
     5        }
     6        function func_b()
     7        {
     8                echo "func_b!\n";
     9        }
    10        func_a();
    11        func_b();
    12        ?>
$ gdb --args /usr/bin/php5 test4.php
... 中略...
(gdb) source php5.5-gdb.py
(gdb) setupphp5executer
Breakpoint 1 at 0x746d70: file /tmp/buildd/php5-5.5.0~rc3+dfsg/Zend/zend_vm_execute.h, line 356.
(gdb) run
... 中略...
"main" in "/home/nojima/prog/my-works/php5-dbg/test4.php":2
"main" in "/home/nojima/prog/my-works/php5-dbg/test4.php":6
"main" in "/home/nojima/prog/my-works/php5-dbg/test4.php":10
"func_a" in "/home/nojima/prog/my-works/php5-dbg/test4.php":4
func_a!
"func_a" in "/home/nojima/prog/my-works/php5-dbg/test4.php":5
"main" in "/home/nojima/prog/my-works/php5-dbg/test4.php":11
"func_b" in "/home/nojima/prog/my-works/php5-dbg/test4.php":8
func_b!
"func_b" in "/home/nojima/prog/my-works/php5-dbg/test4.php":9
"main" in "/home/nojima/prog/my-works/php5-dbg/test4.php":13
(gdb)
```

簡易なものではありますが、無事、 PHP のソースレベルの実行トレースが取れています。

3.4 おわりに

今回 PHP ソースの実行トレースを取る例を使い、 debian の元で gdb+python 拡張を使ったデバッグ手法について述べました。こちらを応用すれば、 PHP ソースレベルのブレークポイントを作るなど、様々な応用ができます。

gdb+python 拡張のデバッグ能力はまだまだ沢山便利な機能が入ってます。是非、 debian パッケージのデバッグのお供にいかがでしょうか?

参考文献

[1] 第 98 回東京エリア Debian 勉強会資料"gdb python 拡張その 1", http://tokyodebian.alioth.debian.org/2013-03.html

[2] 2012 年大統一 Debian 勉強会"debug.debian.net", http://gum.debian.or.jp/2012/

[3] RolandMcGrath/BuildID, http://fedoraproject.org/wiki/RolandMcGrath/BuildID

4 とある Web 企業での Debian システムの使い方。

前田耕平

4.1 背景

筆者が勤務している株式会社サイバーエージェントのエンドユーザ向けのサービスではほぼ CentOS を利用しています。一方、データセンター (以後 DC) 運用チームが提供するサービスでは Ubuntu の利用比率が高く、またエンドユーザ向けサービスにおいても増えつつあります。その中で筆者が関わっている DC 利用者[*5]向けのサービスのツール開発、パッケージメンテナンス、パッケージの配布の観点でのシステム管理のプロセスについて説明します。職場では RHEL 系が多いが、 Debian システムを使いたい、増やしたい、という人へのヒントになればと思います。

4.1.1 CentOS しかなかった?

実は筆者が転職した当時[*6]、筆者が参画したチームでは社内エンジニア向けの開発環境を OpenStack Cactus on Ubuntu 10.04 LTS で提供していました。とは言え当時 Ubuntu を利用していたのはこのチームくらいで、他はほぼ CentOS だけでした。その後、 2012 年 4 月に OpenStack Essex を本番&開発環境向けに提供し始める時に Ubuntu 12.04 LTS(以後 precise) を使い始めました。

4.1.2 新 DC の運用では Ubuntu が増えてます

今年 2 月にリリースした新データセンター[*7]では、 DC 運用側の提供サービス用の仮想マシンには主に precise を使っています[*8]。また Wheezy の自動インストール&ローカルパッケージアーカイブミラーの提供も開始しました。実のところ、利用比率が高いのは筆者の担当するシステムの種類が単に多いためと言っても過言でもないでしょう[*9]。

4.2 Debian/Ubuntu を使っているシステム

precise を使っているシステムは主に、 LDAP、メール、パッケージアーカイブ、 SCM、監視システム、構成管理、トラフィックビューワーなど[*10]です。新しくシステムを作る時には、 sid を開発環境として次のような流れで行います。

1. precise の公式パッケージ (main, universe が対象) で要件を満たせるものは公式パッケージを使う
2. precise では満たせず、 sid で可能ならそのソースパッケージを precise 用にリビルドする (例 python-requests)
3. sid のパッケージでダメでもパッチがあれば適用しリビルドする (例 openssh-lpk パッチ適用の openssh)
4. あるフリーソフトウェアが、 sid の公式パッケージにも無い場合、 Debian パッケージを作成する

[*5] つまり社内やグループ会社のエンジニアのことです。
[*6] 2011 年 9 月
[*7] 技術評論社 Software Design 2013 年 5,6 月号および Web+DB PRESS Vol.74 のインタビュー記事参照。
[*8] ホスト OS には CentOS ですが。
[*9] 他の担当者には筆者が Ubuntu を薦めました。また当初 OpenStack を使う予定だったことも理由の一つです
[*10] 別のセッションで発表されている linkdraw はこの描画ツールです。

5. あるフリーソフトウェアを Deb パッケージにするのが困難ならば upstream の手順に従って導入 (例 GitLab)
6. 以上に該当しなければ自分でツールを開発し、最終的に Debian パッケージにする (例 django アプリなど)

今は wheezy を使い始めていますが流れは同じです。開発環境及びパッケージメンテナンス用の環境には筆者は sid を使っています。チームでの開発言語は主に Python なので、他の同僚もそれぞれ好きな環境を使っています。

4.3 sid での開発＆パッケージメンテナンス環境

4.3.1 sid での開発

開発に必要な Python モジュールは sid の Debian パッケージを前提にします。無い時は PyPI で公開されているライブラリを使いますが、前述の通り Debian パッケージにします。使用するミドルウェアも sid のパッケージを前提にします。この際、 precise の公式パッケージにあるか (前述の 1)、機能に問題ないか (同 2,3) などを調べます。

また開発するツールは Python パッケージにします。依存するモジュールも含めて pure python の場合には、virtualenv で python パッケージの動作確認の後、 Debian パッケージの作成を行います。

4.3.2 sid での Debian パッケージのメンテナンス

sid でのパッケージメンテナンスは Python パッケージからの Debian パッケージの作成がメインです。 debian ディレクトリは開発しているツールの Git リポジトリで一緒に管理し、これを使います。

1. python ソースパッケージの作成 (`python setup.py sdist`)、 tarball の展開
2. dh_make の実行, debian ディレクトリの削除, リポジトリからの debian ディレクトリのコピー
3. debuild の実行
4. pbuilder でのビルド
5. piuparts でのインストールテスト
6. debsign での署名

debian ディレクトリのコピー以外は、通常の Debian パッケージのメンテナンスとほぼ同じです。

4.3.3 precise 向けのパッケージメンテナンス

precise 用のパッケージを作成する時、次の 3 つのケースに別れます。

A. sid で作成したソースパッケージを使い、リビルドだけを行う場合
B. 依存するモジュールの Debian パッケージがない場合
C. 依存するモジュールの Debian パッケージはあるが、バージョンが古く機能的に足りない場合

実際には B, C の場合は対応としては基本同じで sid の公式パッケージのソースパッケージを使います。

4.3.4 パターン A の場合

precise 用のパッケージの作成には、 precise 上で作成した pbuilder の base.tgz を sid 上で使い行います[*11]。 precise の base.tgz は最初に APT Line に universe も追加います。デフォルトのものは main しか無いためです。

```
(host)$ sudo pbuilder --login --basetgz /path/to/precise-base.tgz --save-after-login
(chroot)# sed -i 's/\(main\)/\1 universe/g' /etc/apt/sources.list
(chroot)# exit
```

これをベースとして、次の 3 つ以上のコピーを用意します (ベースとしたものも含みます)。

a) ソースパッケージを展開し、 precise 用に修正＆リビルドするための作業環境
b) a) で precise 用に作成したソースパッケージをクリーンビルドするための環境(前述のパターン A)

[*11] Debian に Ubuntu の公開鍵は含まれず直接作成できないためこのようにしていますが、 Debian Wiki(`http://wiki.debian.org/PbuilderTricks`) によると、 Debian 上で Ubuntu の base.tgz も作れるようです。

c) 自分でリビルドした or 作成したパッケージに依存する場合のクリーンビルド環境 (前述のパターン B, C)

a) の作成用の環境を初めて使う際にはビルド用のユーザを作成します。またパッケージ作成に必要なパッケージや、環境変数の設定も行なっておきます。

a) を用意するのは、 sid 用に作った Python 関連のソースパッケージはそのままビルドできないためです。主な理由は筆者がテストコードの実行用に使っている py.test[*12]です。 sid では python-pytest パッケージですが[*13]、precise には python-pytest パッケージはなく、 python-py パッケージを使います[*14]。そのため、 debian/control の `Build-Depends` セクションを書き換える必要があります。また、 precise では Python 2 は 2.7 しかサポートしていません。 sid では下記のように debian/rules の `verride_dh_auto_test` ターゲットで Python のバージョン毎に `py.test-$$py` でパスを指定していますが、 precise では `py.test` のみのため、 debian/rules も書き換える必要があります。

```
override_dh_auto_test:
        set -e; \
        for py in $(shell pyversions -vr); do \
                py.test-$$py -v $(CURDIR)/_build/lib.\
                $(shell python -c 'import distutils.util as d; print d.get_platform()')-$$py; done
```

またそのままビルドすると debian/changelog の distribution が unstable になるのでこの修正が必要です。

初期設定が終わったら、 pbuilder の chroot 下[*15]にソースパッケージ一式をコピーし、必要な修正後、 `debuild` を実行します。 `debuild` で生成されたソースパッケージを使って、次はホストの sid で b) の base.tgz でクリーンビルドを行います。この際、"`--debbuildopts '-sa'`"オプションを指定し、.changes ファイルに.orig.tar.{gz,xz} が含まれるようにします。これは後述のローカルアーカイブ作成に必要になるためです。

```
(host)$ sudo pbuilder --build --basetgz /path/to/precise-base.tgz --debbuildopts '-sa' somename_x.x-1ubuntu1.dsc
```

生成された precise 用のソースパッケージおよびバイナリパッケージはローカルアーカイブに登録します。

4.3.5 パターン B, C の場合

パターン A との違いは、依存パッケージに precise の公式パッケージを利用できない点です。依存パッケージを sid からのソースパッケージをリビルドするか、または自分で一から作る必要があります。そのため sid(で作成済み) の依存パッケージのソースパッケージを a) で precise 向けにリビルドし、 c) にそれをインストールし、同様に a) で作った最終目的のソースパッケージを c) でクリーンビルドする、という流れです。

4.4 カスタムビルドパッケージのアーカイブ作成

公式パッケージのアーカイブミラーは `apt-mirror` を使っています。一方、独自作成した、もしくは precise 用にリビルドしたパッケージ用のアーカイブ作成には reprepro[*16]というパッケージを使っています。

まず、 reprepro のインストールし、アーカイブ管理用のユーザ (ここでは repomgr とします) を作成、このユーザの GPG キー作成、 GPG の公開鍵を `apt-key` で登録を行います。次に reprepro の設定です。アーカイブを作成するディレクトリを作成します。

```
$ sudo mkdir -p /path/to/debpkg-archive/{debian,ubuntu}/{conf,incoming,log,tmp}
$ sudo chown repomgr: /path/to/debpkg-archive
```

conf ディレクトリの下に次の 3 つのファイルを作成します。[*17]

[*12] Python 3 用は py.test-3
[*13] Python 3 用は python3-pytest
[*14] python-pytest は python-py に依存します。
[*15] /var/cache/pbuilder/build/プロセス ID/ 以下に展開されます。
[*16] `http://mirrorer.alioth.debian.org/`
[*17] conf/distributions の UDebComponents は openssh-lpk パッチ適用の openssh パッケージ用に必要なため追記しています。

conf/options

```
verbose
basedir /path/to/debpkg-archive/ubuntu
ask-passphrase
```

conf/incoming

```
Name: precise
IncomingDir: incoming
TempDir: tmp
LogDir: log
Allow: precise
Default: precise
```

conf/distributions

```
Origin: ca
Label: ca
Suite: precise
Codename: precise
Architectures: amd64 i386 source
Components: custom
UDebComponents: custom
Description: CA local archive for precise
SignWith: yes
```

incoming ディレクトリ[*18]に、署名済みのソースパッケージ一式[*19]と、バイナリパッケージを配置し、 conf/incoming の `Name` の値を引数に `reprepro prcessincoming` コマンドを実行します[*20]。

```
$ cd /path/to/debpkg-archive
$ reprepro processincoming precise
```

後は、この/path/to/debpkg-archive/{pool,dists} を、 httpd サーバで公開し (例: http://example.org/ubuntu/)、これを使用したいホストで repomgr の公開鍵を apt-key で登録し、 APT line に下記を追加すれば使用できます。

```
deb http://example.org/ubuntu/ precise custom
```

これらの key ID, APT line の登録は、 OS 自動インストール時に行い、ユーザには意識させないようにしています。

4.5 RPM パッケージのリポジトリ

現状、 CentOS および SL 用の公式パッケージリポジトリのミラーと、カスタム RPM パッケージ用のリポジトリも用意しています。これらは前述の Debian パッケージアーカイブと同じ precise 上で提供しています。公式パッケージミラーは `rsync` で同期しているので precise でも問題ありません。後者のカスタムパッケージ用のリポジトリは precise の createrepo パッケージを使って作成しています[*21]。

RPM パッケージを作成する人は多数いますが、新 DC 運用側の提供サービスには運用チーム以外の人はリモートログインできません。そこでパッケージアップロード用の Web UI を提供しています[*22]。アップロードするとファイル及びパッケージのメタ情報をデータベースに登録し、リポジトリサーバからアップロードされたパッケージを取得し、ローカルリポジトリのディレクトリに保存、 `createrepo` コマンドを自動実行する仕組みを提供しています。

4.6 まとめ

弊社の新 DC で提供するツールの開発、 Debian パッケージのメンテナンスプロセス、パッケージアーカイブの作成方法、 Ubuntu 上での RPM パッケージリポジトリの管理方法について説明しました。 RHEL 系をメインで使用している環境では、 Debian システムを使いたくてもできないことは多々あります。 IT エンジニアが自分で選べる＆責任を持って運用する自由がある今の職場環境は大きな要因の一つです。しかし、身も蓋も無い言い方をすれば自分で動かなければ実現しないのは事実でしょう。幸い、筆者と同様に Debian を使える自由があるなら、今回紹介した内容がお役に立てれば幸いです。

参考文献

reprepro manual: `/usr/share/doc/reprepro/manual.html`
Debian Wiki: `http://wiki.debian.org/SettingUpSignedAptRepositoryWithReprepro#Configuring_reprepro`
Debian **パッケージ作成方法** 9. reprepro: `http://iishikawa.s371.xrea.com/note/debian-package.html#id2410195`
Setting up and managing an APT repository with reprepro:
`http://www.jejik.com/articles/2006/09/setting_up_and_managing_an_apt_repository_with_reprepro/`

[*18] conf/options の `basedir` と conf/incoming の `IncomingDir` を結合したパスです。
[*19] .orig.tar.{gz,xz}, .dsc, .changes, .debian.tar.gz
[*20] repomgr ユーザとパッケージメンテナーが異なる場合は失敗するため、 repomgr ユーザでパッケージメンテナーの公開鍵をインポート、サインしておく必要があります。
[*21] 弊社のブログにも書きました。 `http://ameblo.jp/principia-ca/entry-11531008993.html`
[*22] python-rpm と Flask で実装しています。

5 善き Debian 者は堅牢なる Gnuk Token を使う

g 新部 裕

5.1 善き Debian 者について

”善き Debian 者”とは、 Debian の理念に同意し、範を示し、 Debian を社会の役に立てようと志す者でしょう。

Debian では GNU Privacy Guard (GnuPG) が使われますが、この小論は、 GnuPG には Gnuk Token が有用で、その利用は Debian だけでなく、社会にとっても善いでしょう、という主張です。一読願えれば幸いです。

5.2 GNU Privacy Guard (GnuPG)

GnuPG は電子署名、認証、そして暗号を利用するツールで、 OpenPGP の仕様 [1] にそった自由ソフトウェアの実装です。豊富な機能がありますが、広く利用されているのは、そのうち、公開鍵暗号を用いた電子署名でしょう。

5.3 Debian のディストリビューションの開発と配布における電子署名

通常、自動的に行われるので意識されないと思われますが、ディストリビューションのユーザは、配布されているパッケージに改竄がないことを電子署名によって検証し、利用しています。以下では、開発と配布において電子署名が用いられる 3 つの場面と、ユーザが検証する仕組みを説明し、どのように注意が払われているかを解説します。

5.3.1 公開鍵への電子署名

OpenPGP は Web of Trust (WoT) という概念によって、公開鍵とその所有者の結びつきを確実にします。これは集中型の信頼モデルを用いる PKI(公開鍵基盤) とまったく異なります。 OpenPGP では、利用者が公開鍵に相互に電子署名を行うことで、信頼モデルの基盤を作り上げるのです。

それは、いわば”草の根”型の信頼モデルのため、 GnuPG のユーザ、特に自由ソフトウェアの開発者や活動家は、顔を合わせる機会に、互いの公開鍵を確認しあって電子署名を行い、この基盤の強化につとめます。カンファレンスなどに併設して催される”キーサイン・パーティ”は、公開鍵の確認作業を定式化した集まりです。

公開鍵への電子署名は、公開鍵とその所有者の結びつきを、別の公開鍵の所有者が保証するものです。

5.3.2 Debian 開発者/メンテナの鍵とパッケージへの電子署名

Debian 開発者/メンテナとして認定されるためには、その公開鍵が、複数の Debian 開発者によって署名されている必要があります。これは、少なくとも複数の Debian 開発者によって、公開鍵とその所有者の結びつきが保証されていることを意味します。現在の Debian 開発者/メンテナの公開鍵はパッケージ debian-keyring で利用可能です。

Debian アーカイブにパッケージを登録できるのは、 Debian 開発者/メンテナだけですが、これはパッケージの電子署名を、 Debian 開発者/メンテナの公開鍵で検証することにより、確認されています。

dput などで、 Debian アーカイブのキューにパッケージがアップロードされると、そのパッケージの電子署名が Debian 開発者/メンテナのものかどうか検証されます。この検証を通ったものだけが、 Debian アーカイブに登録される対象となります。アップロードされたパッケージは、アーカイブに登録される際、その電子署名は外されます。そして、パッケージの内容のチェックサムが管理用の Packages ファイルに登録されます。

5.3.3 Debian アーカイブの Release ファイルへの電子署名

Debian アーカイブでは二種類の鍵が利用されて電子署名が行われています。ひとつは ftp-master のアーカイブ鍵です。もうひとつは安定版のリリース毎のアーカイブ鍵です。この二種類の Debian アーカイブの公開鍵はパッケージ debian-archive-keyring で利用可能です。上述のように、パッケージの内容のチェックサムは Packages ファイルに登録されますが、この Packages ファイルの内容のチェックサムが Release ファイルに登録されます。

Debian アーカイブの管理ファイルの構成は以下のようになっています。

`/debian/dists/wheezy/Release : Release` **ファイル**
`/debian/dists/wheezy/Release.gpg : 電子署名`
`/debian/dists/wheezy/main/binary-i386/Packages.bz2 : Packages` **ファイル**

この Release ファイルが、アーカイブ鍵によって署名され、 Release.gpg という名前のファイルが用意されています。安定版は二種類の鍵の両方で署名されます。 testing, unstable, experimental そして security アーカイブは、ftp-master のアーカイブ鍵だけで署名されます。

ここで注意が必要なのは、アーカイブ鍵は少数の Debian 開発者によって署名されていますが、それ以外では、 Debian 開発者/メンテナの鍵とアーカイブ鍵との間には特に関係はないことです。[*23]

5.3.4 パッケージの配布を受け取る際の検証

apt ではパッケージの改竄防止の検証を行う機能があり、以下の 3 段階で検証されます。

1. Release ファイルとその電子署名 Release.gpg をアーカイブ鍵で検証し、正当性を確認します。
2. 次に、取得した Packages ファイルのチェックサムを取り、 Release ファイルに記載された Packages ファイルのチェックサムと比較し、 Packages ファイルの正当性を確認します。
3. 最後に、取得した当該パッケージのチェックサムを取り、 Packages ファイルに記載された当該パッケージのチェックサムと比較し、パッケージの正当性を確認します。

ここでの電子署名の検証には、システムにインストールされている (`/etc/apt/trusted.gpg.d`)、アーカイブ鍵の公開鍵が使われることに注意してください。通常の GnuPG の利用では、検証において OpenPGP の WoT が用いられ、署名に用いられている鍵と利用者 (の鍵) との関係が問題とされますが、ここでは利用者の鍵は問題とされません。

5.4 Gnuk Token について

Gnuk Token は GnuPG の秘密鍵を保持するセキュリティ・トークンの名称です。

Gnuk Token は、”秘密鍵をどこにどのように置いといたらいいか”という問題に対する解決策の一つです。持ち運びが容易で、かつ、分解してデータを取り出すことが困難な、独立の小さなデバイスに秘密鍵を保持することで、この問題を解決し、より安全な GnuPG の運用ができることを目指しています。

Debian 開発者/メンテナは、相互の公開鍵への署名、パッケージへの署名、または、投票への署名のためなど、さまざまな用途があり、自分自身のの鍵を管理して直接 GnuPG を利用する必要があります。 Debian 開発者/メンテナは、いわば、 GnuPG のヘビーユーザですから、 Gnuk Token は Debian 開発者/メンテナにとって有用でしょう。

Gnuk Token は Gnuk という自由ソフトウェアを利用し、 STM32F103 という MCU を使用します。以下では、特にソフトウェアについて解説し、続いて、その堅牢性について述べます。

[*23] ArchLinux では事情が異なり、配布のための鍵は多くの開発者によって署名されています [2]。

5.4.1 Gnuk

Gnuk Token は、 Gnuk と名付けられた自由ソフトウェアを利用します。その諸元を説明します。

- 3 つの独立した (署名、暗号、認証) RSA 2048-bit の鍵 (署名に要する時間は約 1.5 秒)
- GPLv3+ でライセンスされる自由ソフトウェア / 開発環境は自由ソフトウェアだけ
- 実装について
  - OpenPGPcard Protocol 2.0、 ISO7816-3、 CCID ドライバ、 USB のドライバを自前で実装
  - PolarSSL の AES と RSA ルーチンを利用
  - カーネルは ChibiOS/RT を利用
- 機能テストとして python-nose を利用
- ファームウェア・アップグレード機能を有する
- ボードは FST-01 の他、 Olimex STM32 H103, STM8S Discovery Kit の STM32 部分など各種をサポート

5.4.2 堅牢性について

Gnuk Token の開発にあたっては、少なくとも下記の事項について、考察を行っています。

- 共有ではないユーザ自身で管理する独立のデバイス
- ハードウェアの盗難を防ぐために肌身離さず保持することが可能
- 秘密鍵はワンチップの MCU のフラッシュ ROM に保持、暗号の計算をデバイスで行う
- パスフレーズ 3 回の失敗でロックされるため、盗難されてもブルートフォース攻撃はできない
- JTAG デバッガからのアクセスは禁止している
- チップを破壊しての情報の取り出しは IC カードに比べれば強くないかも知れない
- 一般のコンピュータのディスクからのデータを取り出しに比べれば十分困難
- 秘密鍵は暗号化してフラッシュ ROM に保存されている
- 暗号の計算についてはサイドチャネル攻撃も考慮 (して作られた PolarSSL を利用)
- ソフトウェアの脆弱性による秘密鍵の漏洩に対処するため、機能は限定し、実装は十分注意
- ただしアドレスのランダマイズはアドレス空間が小さいため有効性が少なく、行わない
- ファームウェア・アップグレードの際には秘密鍵は削除される

5.5 Gnuk Token の利用方法

Gnuk Token の利用には、環境の準備と、 Gnuk Token の個人設定が必要です。一旦使えるようになれば、後は通常の GnuPG の利用と同じです。

GnuPG を使う際のユーザと関連プログラム、および Gnuk Token の関係は、図 9 のようになります。 Debian では、squeeze 以降であれば Gnuk Token の利用ができます。

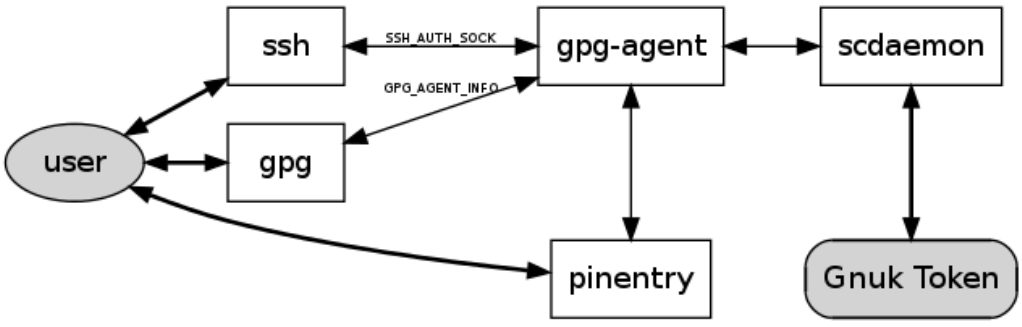


図 9 GnuPG と Gnuk Token

5.5.1 SSH について

あまり知られていませんが、 GnuPG の gpg-agent では ssh-agent のサービスを同時に動作させることができます。これにより、 GnuPG の鍵を使って SSH の認証を行うことができます。 Gnuk Token を使えば、 GnuPG の電子署名や暗号のための秘密鍵だけでなく、ネットワークアクセスのための秘密鍵も統合して、独立のデバイスに分離して管理することができます。

5.5.2 Gnuk Token の個人設定

最初に、 `gpg --card-edit` でパーソナライゼーションを行います。 Gnuk Token にログイン名や公開鍵の置き場所の URL を登録できます。そして、 RSA 2048-bit 鍵を 3 つ (それぞれ署名、暗号、認証用) 用意し、この鍵を書き込みます。具体的な手順の詳細は、 GnuPG の Smart Card HowTO[3] や"FST-01 で Gnuk を使う"[4] を参照してください。

5.5.3 具体的な利用例: 署名

ファイルに対する電子署名は、通常、下記のように`--clearsign` オプションで起動します。最初の起動の場合、pinentry のウィンドウがポップアップし、パスフレーズを聞いてきますので入力します。

```
$ gpg --clearsign <FILE>.txt
```

公開鍵への署名は、`--sign-key` オプションで起動し、サブセッションでパラメータを設定して行います。

```
$ gpg --sign-key <NAME>
```

Debian パッケージへの署名は、.changes ファイルと.dsc ファイルの二つのファイルに行われます。特に指定しなければ、パッケージビルドの際、自動的に GnuPG が起動されて署名が行われます。`-uc` オプションと `us` オプションを指定して自動で署名を行わないようにもできます。署名なしでビルドしたパッケージに、後で署名だけ行うには、 `debsign` コマンドを用います。必要であれば`-k` オプションで鍵の ID を指定します。

Git リポジトリへのコミットに対して署名するには、 `git commit` の際、`-s` オプションを使います。

5.5.4 具体的な利用例: 認証 (SSH)

OpenSSH で利用するには、 GnuPG の認証鍵から、 OpenSSH の形式の公開鍵を得る必要があります。下記のコマンドで、 SSH の公開鍵のフォーマットでデータが取得できます。

```
$ gpgkey2ssh <YOUR-KEY-ID>
```

ここで key ID として 16 進数を指定する場合、全部大文字とする必要があります。この SSH の公開鍵のデータをリモートのホストに登録します。最初の認証に別の認証が使える場合には、直接、 `ssh-copy-id` コマンドを使って登録できます。また、 Debian 開発者が Debian に SSH 鍵を登録する場合は下記のようになるでしょう。

```
$ gpgkey2ssh <YOUR-KEY-ID> | gpg --clearsign | mail changes@db.debian.org
```

この設定ができたら、後は、 SSH でリモートホストにアクセスできます。認証には約 1.5 秒かかります。

参考文献

[1] J. Callas, et al., RFC4880, OpenPGP Message Format, 2007.

[2] Arch Linux Master Keys, `https://www.archlinux.org/master-keys/`

[3] Rebecca Ehlers, et al., The GnuPG Smartcard HOWTO, 2005.

[4] FST-01 で Gnuk を使う, `http://no-passwd.net/fst-01-gnuk-handbook/`

[5] Threads and only Threads, `http://gitorious.org/chopstx`

6 Azure de Debian

Kazumi HIROSE

6.1 はじめに

Windows Azure は 2009 年からマイクロソフトが提供しているパブリッククラウドです。昨年 6 月にそれまで提供していた、 Linux が動作する仮想マシン(IaaS) が提供される事になりました。

本セッションでは、マイクロソフトの最近の OSS への取り組みや、オープンソースに関連した Windows Azure の機能と、実際に Debian で活用する方法をご紹介します。

6.2 Microsoft のオープンソースに対する取り組み

この数年、 Microsoft はオープンソースへの支持・支援を積極的に行なっています。

Linux カーネルへのコミットメント、 Apache Foundation VP である Gianugo Rabellino 氏が入社、オープンソースコミュニティと協力する事を目的とした、 MS Open Technologies 社を設立しています。

HTML5 や HTTP2.0 を含む、 150 以上の標準化機関と 400 を越えるワーキンググループと共同作業を行っており、コミュニティにも積極的な支援をしています。

最近追加されたサービス・アプリケーションや SDK では、自社プラットフォーム以外へも、 iOS や Android の SDK を提供しています。

日本でも、 WordPress/EC-CUBE/MODX/XOOPS Cube/Baser CMS などのコミュニティや、 PHP Matsuri/PHP Conference/Ruby Kaigi/Python Conference/Open Source Conference などのイベントに協賛、支援をしています。

6.3 Windows Azure について

ここでは OSS や Linux に関連する機能に絞り、最近のトピックスをご紹介します。

6.3.1 管理ポータルの刷新

Silverlight ベースだった管理ポータルが HTML5 ベースに作り変えられ、 Linux やスマートフォンからもアクセスできるようになり、日本語を含めた 11 ヶ国語で表示できるようになりました。

6.3.2 新規リージョン提供の発表

日本・オーストラリアにリージョンが展開されることが発表されました。

6.3.3 "仮想マシン"の提供と Linux のサポート

Hyper-V のゲスト OS を提供します。仮想マシンの対象 OS に CentOS/Ubuntu Linux/suse Liunx が提供され、各ディストリビューションのサポートは、 OpenLogic/Canonical/SUSE が対応します。

公式以外のイメージは、前述の MS Open Technologies により VMDEPOT で提供されています。

6.3.4 "Web サイト"の提供

Shared ホスティング型 PaaS の"Web サイト"が提供されました。 ASP.NET/PHP/Node.js/Python で開発された WEB アプリケーションを、 FTP/Git/Mercurial や github/BitBucket/Codeplex/Team Foundation Service/Dropbox といったサービスからもデプロイできます。

また、ギャラリーには WordPress などがコミュニティにより提供されており、容易にサイト構築ができます。

6.3.5 Windows Azure Xplat CLI Tool

Node.js ベースで Windows Azure を CLI 操作できるツールが提供されました。 Windows Azure CLI tool for Windows, Mac and Linux (Apache License) `https://github.com/WindowsAzure/azure-sdk-tools-xplat`

6.3.6 Windows Azure SDK の公開

Java/Node.js/.NET/PHP/Python/Ruby の SDK が Apache License で github に公開されました。

6.4 Azure における Debian のサポート状況

Debian に限った話ではありませんが、 Azure で Linux を動作させるには、以下の要件を満たしている事が前提になっています。

- VHD 形式の仮想 HDD イメージである事
- カーネルモジュールに Hyper-V 用モジュールが必要
- WALinuxAgent が必要

6.4.1 VHD 形式の仮想 HDD イメージである事

Windows 8 Pro / Windows Server 2012/2008R2 では Hyper-V をインストールする事が出来ますので、そこで VHD 形式の HDD イメージが作成できます。 VMWare VMDK や VirtualBox VDI で作成されたイメージの場合、現時点では NHC を用いて仮想 HDD イメージの変換で対応できます。

6.4.2 カーネルモジュールに Hyper-V 用モジュールが必要

Linux integrated service と呼ばれていたソフトウェアが導入されている事が必要ですが、 Linux3.2 に GPL として取り込まれた事で、 Wheezy には既に導入されています。導入されているかの確認は以下の方法で行えます。

```
$ sudo lsmod | grep hv
hv_utils               12986  0
hv_netvsc              18304  0
hv_storvsc             17423  2
hv_vmbus               32029  4 hv_storvsc,hv_netvsc,hid_hyperv,hv_utils
scsi_mod              162269  5 libata,sr_mod,sg,hv_storvsc,sd_mod
```

6.4.3 WALinuxAgent が必要

Windows Azure Linux Agent は、仮想マシンの CPU 利用率やメモリ利用率などのモニタリング機能や管理機能を提供する Python スクリプトです。 WA Linux Agent (Apache License) `https://github.com/Windows-Azure/WALinuxAgent`

Debian では、 Jessie にパッケージがありますので、 APT の PIN 機能でパッケージを借ります。

6.4.4 /etc/apt/sources.list

```
deb http://ftp.jp.debian.org/debian/ wheezy main
deb http://security.debian.org/ wheezy/updates main
deb http://ftp.jp.debian.org/debian/ wheezy-updates main
deb http://ftp.jp.debian.org/debian/ jessie main
```

6.4.5 /etc/apt/preferences

```
Package: *
Pin: release a=stable
Pin-Priority: 700

Package: *
Pin: release a=testing
Pin-Priority: 80

Package: *
Pin: release a=unstable
Pin-Priority: 90
```

6.4.6 waagent のインストール

```
# apt-get install waagent/jessie
Reading package lists... Done
Building dependency tree
Reading state information... Done
Selected version '1.3.2-1' (Debian:testing [amd64]) for 'waagent'
... 中略 ...
Do you want to continue [Y/n]?
```

6.4.7 ディスクイメージの準備

汎用化ディスクイメージを作成します。このコマンドで環境に依存するファイルや設定をクリーンアップします。

```
# waagent -deprovision
WARNING! The waagent service will be stopped.
WARNING! All SSH host key pairs will be deleted.
WARNING! Nameserver configuration in /etc/resolv.conf will be deleted.
WARNING! Cached DHCP leases will be deleted.
WARNING! root password will be disabled. You will not be able to login as root.
Do you want to proceed (y/n)? y
# shutdown -h now
```

6.4.8 Windows Azure へアップロード

Windows Azure Xplat CLI をインストールして、ディスクイメージをアップロードします。

6.4.9 Node.js の導入

```
# apt-get install python g++ make
# wget -N http://nodejs.org/dist/node-latest.tar.gz
# tar xzvf node-latest.tar.gz && cd node-v*
# ./configure
# make && make install
```

6.4.10 Windows Azure Xplat CLI の導入

```
# npm install azure-cli -- g
$ azure
info:            _   _____   _ ___ ___
info:           /_\  |_  / | | | _ \ __|
info:      _ ___/ _ \__/ /| |_| |   / _|___ _ _
info:    (___  /_/ \_\/___|\___/|_|_\___| _____)
info:       (_______ _ _)          _ ______ _)_ _
info:              (_____________ _ )   (___ _ _)
info:
info:    Windows Azure: Microsoft's Cloud Platform
info:
info:    Tool version 0.6.16
```

6.4.11 Windows Azure のアカウント設定のダウンロードおよびインポート

```
$ azure account download // ブラウザが起動しダウンロードします。
$ azure account import [Downloaded File]
```

6.4.12 ディスクイメージのアップロード

```
$ azure vm image create "Debian7.0-Wheezy-Minimal" --location "East Asia" --os Linux WheezyOnWA.vhd
info:    Executing command vm image create
+ Retrieving storage accounts
info:    VHD size : 30 GB
info:    Uploading 31457280.5 KB
Requested: 79.6% Completed: 79.0% Running:  92 Time:36m15s Speed: 11431 KB/s
```

アップロードしたディスクイメージは、インスタンスとして起動する事も VMDEPOT に登録する事もできます。

作成のイメージは VMDEPOT に登録していますので、是非使っていただければ幸いです。 http://vmdepot.msopentech.com/Vhd/Show?vhdId=2427&version=2457

7 Dennou Club Library を Debian 公式パッケージにするには?

佐々木洋平

7.1 はじめに - はなしの「まくら」

Dennou Club Library(以下、 DCL) は地球流体電脳倶楽部 (http://www.gfd-dennou.org/) の関係者が長年にわたり蓄積してきた[*24]、欠損値処理や地図投影などの機能が標準的にサポートされた描画ライブラリです。 DCL は FORTRAN77 で書かれており, これをベースに f2c によって生成されたコードに手を入れた C 版, これを用いる Ruby 拡張ライブラリも提供しています[*25]。日本に限らず世界各地の大気海洋科学の研究者に用いられており、ソースコードは全て公開され「無責任無保証の元で改変と再配布は自由に行なえる状況」にあります。

しかしながら、長年の蓄積と当時の牧歌的な状況などからか、コピーライトやライセンスは (上記文章は含まれているものの) あまり明確ではなく、このままでは「フリーソフトウェア」と名乗って良いものか定かではありません。本稿では、筆者が「DCL を Debian の公式パッケージとしたい! 」と思って行なった/行なっているアレやコレについてまとめてみます。

7.2 「地球流体電脳倶楽部」とは?

「地球流体電脳倶楽部」は, 主に地球惑星科学, 天文学における「流体」を研究する分野の研究者と学生からなる草の根 (もしくはボランティア) 集団です[*26]. 全国各地の有志から構成されており, 普段はネットワーク上で活動していることが多い (active member は 20 名程?) です. 詳細に関しては「地球流体電脳倶楽部について (http://www.gfd-dennou.org/html/about/)」あたりをご覧下さい.

我々は好んで Debian を使用しています。我々の「ポリシー」[*27] において

> 研究教育のための情報は無料でなければならない
> 少なくとも税金で運営されている国立大学、国立研究所が提供する情報は無料であるべきである。また、その情報の出所が明らかでなければならない。さらに、その情報の再配布が可能であることが望ましい。

と掲げている通り、 Debian ファンが多い集団でもあったりします。「無料」と掲げている点が悩ましい所ではありますが...。現在、北大・京大・九大にサーバを設置し Debian のミラーをやっていたりします。関係者が他の大学/研究機関へ異動し (機関の許可があれば) 他にも増えるかもしれません (減るかもしれません)。

[*24] 今回歴史を遡った所、 1987 年頃にプロトタイプが公開されていた模様です。ただ, 本当のプロトタイプは PC-8801 で動いていたという話 (噂?) もありますので、正直いつからあるのか定かではありません (笑)

[*25] 数値計算は FORTRAN, fortran, C で行なっても, その後のデータ解析までこれらの言語でやるのは結構シンドイです。「計算量自体が少ない場合。例えば C で 1 秒以内に済む計算が、 Ruby では 1 分かかるかもしれない。しかし、そのためのプログラム開発に C では 5 時間かかり、Ruby では 30 分程度で済むならどうだろう。もしもそのプログラムを実行するのが高々数 10 回程度としたら、 Ruby を使うことはおかしな選択ではない [2]。」と、筆者も思っています。

[*26] たまに誤解されますが、特定の資金源や専任スタッフが存在するわけではありません。

[*27] ポリシー: http://www.gfd-dennou.org/html/about/policy.html

7.3 事の発端

筆者は、地球流体電脳倶楽部で開発・配布しているソフトウェアを野良 Debian パッケージとして公開しています:

```
# sid, GPG: 0xAEE995F4
deb http://www.gfd-dennou.org/library/cc-env/Linux/debian-dennou squeeze/
deb-src http://www.gfd-dennou.org/library/cc-env/Linux/debian-dennou squeeze/

# wheezy, GPG: 0xAEE995F4
deb http://www.gfd-dennou.org/library/cc-env/Linux/debian-dennou squeeze/
deb-src http://www.gfd-dennou.org/library/cc-env/Linux/debian-dennou squeeze/
```

自分が計算機にソフトウェアをインストールする際に毎回ビルドするのが手間で作成/整理し始めたのですが、最近は自分でリポジトリを用意するのが面倒になってきましたし,「ライセンスに問題が無いならいっそ Official に入れてしまおう」と思いついたのが事の発端です[*28]

7.3.1 Ruby まわりは upload できたけれど

ライセンスがクリアなソフトウェア (特に Ruby まわりのソフトウェア) に関しては、既に Official に upload してあります。代表的なのは

- `ruby-netcdf`: NetCDF を Ruby で扱うライブラリ[*29]
- `ruby-grib`: ECMWF GRIB を Ruby で扱うライブラリ[*30]
- `ruby-lapack`: LAPACK の 1 to 1 wrapper [*31]

でしょうか。これらは BSD 2-Clause or Ruby's なので、Official にアップロードするのには特に困りませんでした。

7.3.2 で「GPL 準拠」なの? それ、良いの?

データの入出力、解析を行なうためのツールはちゃくちゃくと整理・Official への upload が進んでいた訳ですが、肝心の描画ライブラリがなかなか進みません。そもそも当時はライセンスが明確にされておらず、そもそも `debian/copyright` を書くことができませんでした。しかしながら、FAQ [3] には

> **DCL はフリーですか?**
> はい、フリーソフトウェアです。著作権は GNU Public License に準拠しております。因みにパッケージ内の CREDITS には版権を以下のように定めております。
>
> > 本資源の版権は地球流体電脳倶楽部に属する。資源の利用にあたっては地球流体電脳倶楽部の定める規定にしたがっていただきたい。原則として、教育的目的の場合には自由に使用・改変して良いものとしている。

とあるわけです[*32]。

7.4 歴史を紐解く

というわけで、本当に GNU General Public License (以下, GPL) 準拠になっているのか確認した上で、`debian/copyright` 書いてみましょうか、という事になります。これが明確になってないと ITP/RFS しにくいですし。

コードの大部分は地球流体電脳倶楽部関係者によって開発・整理されているため、Copyright holder が誰であるか問い合わせる/明確にする作業は着々と進んでいました[*33]。... 進んでいたのですが。

[*28] 「Ubuntu のリポジトリは無いの?」という問い合わせが増えたのも理由の一つではあります (一応 PPA は有志によって用意されています)。実態は上記 apt リポジトリの (とあるタイミングでの) clone ですが。

[*29] PTS: ruby-netcdf, `http://packages.qa.debian.org/r/ruby-netcdf.html`

[*30] PTS: ruby-grib, `http://packages.qa.debian.org/r/ruby-grib.html`

[*31] PTS: ruby-lapack, `http://packages.qa.debian.org/r/ruby-lapack.html`

[*32] "GNU Public License" は typo. ではありません. 念のため

[*33] 実際には Copyright が明記されていないコードに関して原作者に問い合わせ `(c) GFD-Dennou Club` に変更/明記するだけです。

7.5 問題点

以下, 現状把握されている問題点を解決済の物も含めて解説してみます。

7.5.1 Numerical Recipe 由来: 解決済み

乱数生成に関するコードの一部に Numerical Recipes 由来の SUBROUTINE が使われていました。良く知られている通り Numerical Recipes のライセンスはフリーではありません[*34]。

そもそも FORTRAN77 には乱数ルーチンが無い (!) ため、些細な事ですが自前で実装する必要がありました。デフォルトでは C の `rand()` を FORTRAN77 から呼び出しているだけであり、これが使えない場合の代替手段として提供していた SUBROUTINE でしたので、結局該当 SUBROUTINE を破棄することにしました。

7.5.2 Hershey フォント: 解決 (?)

DCL では軸ラベル, タイトル等の描画等に Hershey フォントを使用しています。

Hershey フォントは Hershey, A.V. が 1967 年に作成したベクタフォントです [4]。原論文は参照していませんが、その後 USENET でフォントデータが配布されており、その際のライセンスは

```
USE RESTRICTION:
      This distribution of the Hershey Fonts may be used by anyone for
      any purpose, commercial or otherwise, providing that:
             1. The following acknowledgements must be distributed with
                    the font data:
                    - The Hershey Fonts were originally created by Dr.
                           A. V. Hershey while working at the U. S.
                           National Bureau of Standards.
                    - The format of the Font data in this distribution
                           was originally created by
                                  James Hurt
                                  Cognition, Inc.
                                  900 Technology Park Drive
                                  Billerica, MA 01821
                                  (mit-eddie!ci-dandelion!hurt)
             2. The font data in this distribution may be converted into
                any other format ((*EXCEPT*)) the format distributed by
                the U.S. NTIS (which organization holds the rights
                to the distribution and use of the font data in that
                particular format). Not that anybody would really
                ,*want* to use their format... each point is described
                in eight bytes as ‘‘xxx yyy:’’, where xxx and yyy are
                the coordinate values as ASCII numbers.
```

となっています[*35]。さて, これはどういう扱いになるのでしょうか?

次期バージョンの DCL は GTK/Cairo を使って描画するように下層の入れ替えを進めていますので、もしかしたらこの疑問は不要になるのかもしれません。ただ、昔の図の再現性が無くなるのはちょっと痛いです。

7.5.3 海岸線データ: 解決 (?)

DCL では地図投影を行なった際に、地図を描画することができます。この際の海岸線データの出所について、あまり明瞭に記載がされていません。日本付近の高精細海岸線データに関しては、とりあえず配布物から抜いて、別途確認中です。

7.5.4 colormap: 解決 (?)

現在 DCL で使用できる colormap の一覧は `http://www.gfd-dennou.org/library/dcl/dcl-5.4.8/src/env1/colormap/colormap_gallery.html` にあります。幾つかの colormap は他のソフトウェアの colormap を **参考にして** 作っています。 (例えば, GNU Octave[5], IDV[6], NCL[7] など)。

コードをそのまま持ってきた訳ではない, 出力結果を元に色調を手で合わせて作成された colormap なのですが、この場合にはライセンスはどうなる/どうすべきなのでしょうね. とりあえず、現状は配布物に同梱されていますが、抜くことも容

[*34] (自称を含めた) 数値計算屋さんにはあまりちゃんと広まっていませんが、 Numerical Recipes は、内容に関する批判もさることながら、「 この本を読んでコードを書くとコードが公開できない」という点において、決して科学・教育に使ってはいけない本だと、筆者は考えます。ちなみに内容に関しては、読んでいないので批判できません。コードの質は一時に比較して改善されているらしいですね。

[*35] その後幾つか glyph が追加されていきましたが、現在 ghostscript に含まれている改良 (悪?) 版は, glyph を追加した人によって “These are “freeware”, not to be sold.” という文言が追加されたため Debian では `non-free/gsfonts-other` で提供されています。

易に可能な状況にしてあります。

7.5.5 欲望との戦い: 引用して欲しいよね?

科学研究では論文を書く/書いた論文が引用される、というプロセスが重視されることが多いです。研究・教育におけるソフトウェア開発は (ソフトウェアそのものを研究する場合を除くと)、あまり評価されていません。これについて「使ったらなるべく引用して欲しいよね」となるのが人情かもしれません。

DFSG-FAQ[8], 10.g (draft 版ですので, 項目はまた変わるかもしれません) には “You have a valid concern.” から続く, 以下の文言があります:

> Computer scientists often receive inadequate credit for their scientific contributions. But putting such a clause in the license would render your software non-free. Instead we suggest a note, not part of the license itself, reminding users of the rules of scientific propriety. Eg:
>
> > SCIENTISTS: please be aware that the fact that this program is released as Free Software does not excuse you from scientific propriety, which obligates you to give appropriate credit! ...(略

ちょっとキツメの書きかたなので, どういう文章にするかは考える必要があるかもしれません.

7.6 まとまらない

これらの問題が解決した段階で、ようやくライセンスを「GPL 準拠」にしよう、という話になるわけですが、現状では BSD 2-Clause にしたい、という意見が (開発者の) 大半を占めており、問題が無いのであればそうしましょう、ということになっています。これは「大学で使っていたソフトウェアはそのままどこに行っても使えて欲しい」という気持からです.

一方で、 DCL とは (あまり) 関係がありませんが「数値モデルを共同で開発しましょう」という際に、ライセンスについて

> 有能な技術者が理解のない上官の下で孤立しているようなとき、外部ソフトウェアの差分をその開発コミュニティにフィードバックすること自体は、まあ承認を得ることが可能であるけれど、その差分のライセンスに注目がいったら最後、職務著作であるからライセンス設定について然るべきものによる正当な意思決定をしなければならないという、反論不能かつ事実上実行不能な指示＝リスク回避が行われることがままあるわけですね。
>
> そのときおおもとのソフトウェアが (L)GPL であってくれれば、差分が少なくとも (L)GPL でなければならないという制約がかかるので、意思決定がバイパスされます。ガバナンスが崩壊している組織においては、意思決定が不可能であることがボトルネックですから、これで仕事ができるようになるわけです。まあ、要するに外圧スキームですが。

という話もあって、個人的には LGPL-2+ あたりにしておいた方が良いかもしれない、とか思っていたりします。まとまってませんが、そんなこんなで。

参考文献

[1] 地球流体電脳倶楽部、 http://www.gfd-dennou.org/

[2] 堀之内 武 (著)、立石 孝彰 (編)「数値計算と可視化」、 Ruby Library Report 第 5 回、 Rubyist Magazine、 No.6、 http://magazine.rubyist.net/?0006-RLR

[3] 地球流体電脳ライブラリ Q and A 集, http://www.gfd-dennou.org/arch/dcl/dcl-QA/dcl-QandA.html

[4] Hershey, A. V., 1967: “Calligraphy for Computers, Dahlgren, VA”, U.S. Naval Weapons Laboratory, *NWL Report*, **2101**. NTIS AD662398.

[5] GNU octave: http://www.gnu.org/software/octave/

[6] IDV: http://www.unidata.ucar.edu/software/idv/

[7] NCL: http://www.ncl.ucar.edu/

[8] DFSG and Software License FAQ (Draft): http://people.debian.org/~bap/dfsg-faq.html

8　自作描画ツール Linkdraw の紹介

村越 俊克

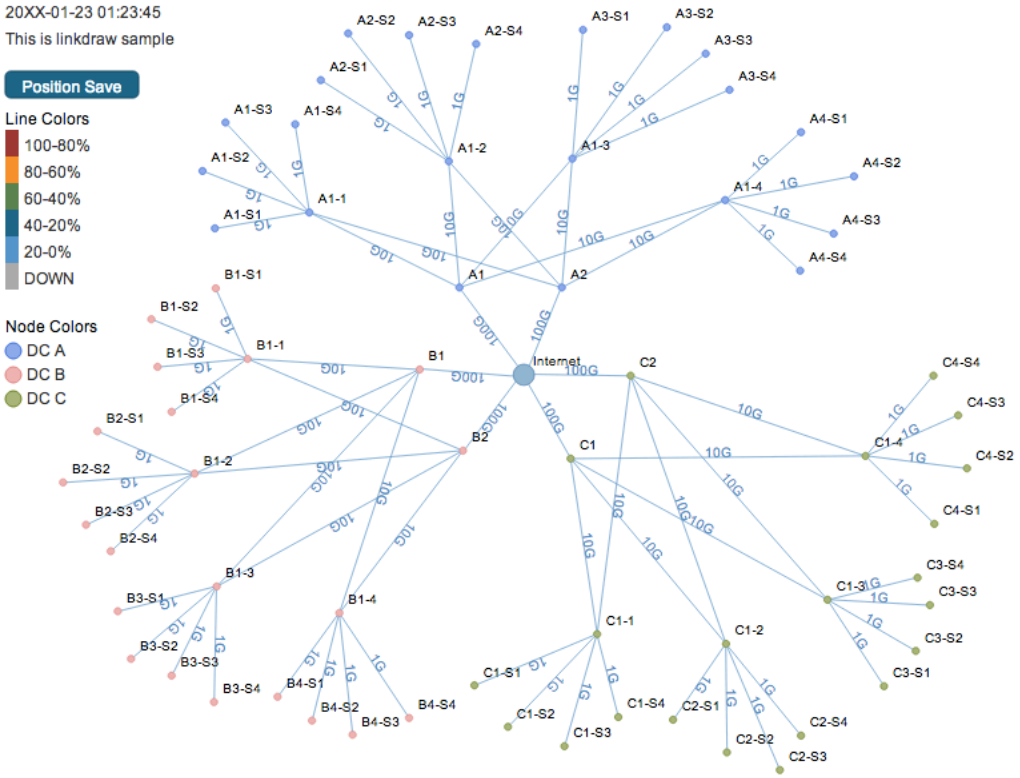
8.1　はじめに

私は、サーバやネットワーク周りの運用をしているエンジニアです。日々、運用をしていると様々な “手間” に遭遇します。そして、その都度いろいろな”ツール”を利用したり自作しては、ちょっとだけ楽をしております。今回は、Linkdraw という自作ツールを紹介します。

8.2　Linkdraw

”繋がりを描く”という名前の通り Linkdraw とは、アイテムとアイテム間の繋がりを描画する為のツールです。描画されたアイテムは、ドラッグアンドドロップで動かしたりハイパーリンクを埋め込む事も可能な為、ドキュメントや wiki 等の運用情報とリンクすることもできます。

以下は、サンプルです。



8.3 インストール

Linkdraw は、現在、野良パッケージとなります。また、 jQuery と d3 に依存しており jQuery は、公式パッケージですが d3 は、公式パッケージが存在しない為、こちらについても野良パッケージを作成しております。今回は、これらを使った説明となります。

まず、 jquery をインストールします。

```
$ sudo apt-get install libjs-jquery
```

次に、 d3.js と linkdraw の野良パッケージをインストールします。

```
$ wget http://linkdraw.org/deb/libjs-d3_3.1.10-1_all.deb
$ wget http://linkdraw.org/deb/libjs-linkdraw_0.2.4-1_all.deb
$ sudo dpkg -i libjs-d3_3.1.10-1_all.deb libjs-linkdraw_0.2.4-1_all.deb
```

コンテンツを設置したい場所 (path/to/linkdraw) へ symlink を設置します。

```
$ sudo ln -s /usr/share/javascript/jquery/jquery.min.js /path/to/linkdraw/js/jquery.min.js
$ sudo ln -s /usr/share/javascript/d3/d3.min.js /path/to/linkdraw/js/d3.min.js
$ sudo ln -s /usr/share/javascript/linkdraw/linkdraw.js /path/to/linkdraw/js/linkdraw.js
```

サンプルがインストールされているはずですのでこれらを setup.sh で設置します。

```
$ cd /usr/share/linkdraw
$ bash utils/setup.sh /path/to/linkdraw
```

サンプルを確認してみましょう。

```
  http://example.org/path/to/linkdraw/sample.html
```

8.4 使い方

まず、 Linkdraw と依存関係にある jQuery と d3 を import します。 (外部から import できなければインストールで設置したローカルの symlink で import)

```
<script src=''http://code.jquery.com/jquery-1.10.1.min.js''></script>
<script>!window.jQuery && document.write('<script src=''/path/to/jquery.min.js''><\/script>')</script>

<script src=''http://d3js.org/d3.v3.min.js''></script>
<script>!window.d3 && document.write('<script src=''/path/to/d3.v3.min.js''><\/script>')</script>
<script src=''http://linkdraw.org/linkdraw.js''></script>
<script>!window.linkdraw && document.write('<script src=''path/to/linkdraw.js''><\/script>')</script>
```

次に、描画する HTML 要素の id を指定して config ファイルのパス、ポジションファイルのパス、描画サイズなどを指定します。

```
<script type=''text/javascript''>
$(function(){
     $(''#sample'').linkDraw({             //  描画する HTML 要素の id を指定。
       ''configPath'': ''config.json'',      // 描画の為の設定ファイル。
       ''positionPath'': ''position.json'',  // ポジションを設定するファイル。
       ''positionWriter'': ''write.cgi'',    // ポジション情報を書き込む為の CGI プログラム .
       //''positionSave'': false,          // ポジション保存ボタンの表示。（デフォルト有効）
       //''zoom'': false,                  // 描画されたアイテムの拡大縮小。（デフォルト有効）
       //''drag'': false,                  // 描画されたアイテムの移動。（デフォルト有効）
       ''width'':  400,                    // svg 横 (px)
       ''height'': 300,                    // svg 縦 (px)
       ''interval'': 10                    // configPath で指定している設定ファイルの
                                         // 読込みインターバル（秒）。
     });
  });
</script>
```

以上が、 HTML の header 部分となります。

そして、 body の描画させたい箇所へ以下を追加します。

```
<div id=''sample''></div>
```

以上で、 HTML の設定は、完了です。

次に configPath で指定した設定ファイル “config.json” を設置します。

```
  vi config.json
```

サンプルとして以下を書いて保存します。 (A と B という 2 つのノード間に線を 1 本引く設定)

```
{
    ‘‘lines’’: [
        { ‘‘source’’: ‘‘A’’, ‘‘target’’: ‘‘B’’,’’color’’: '', ‘‘width’’: ‘‘1’’, ‘‘descr’’: '', ‘‘link’’: '' }
    ]
}
```

次に、 positionPath で指定したポジションファイル “position.json” を設置します。

このファイルは、空で構いません。 また、 CGI プログラム によってポジションが保存される為、書き込み権限が必要となります。

```
touch position.json
chmod 666 position.json
```

最後に、ポジションを保存する為の CGI プログラムを設置します。これは、 linkdraw に付属している CGI (/usr/share/linkdraw/src/write.cgi) で構いません。 (python2.6 以上で動作)

```
chmod 755 write.cgi
```

以上で、必要なものが揃いました。

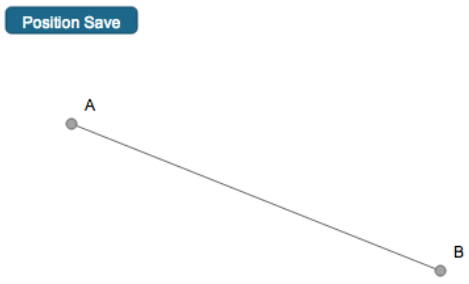
path/to/linkdraw 配下

```
    jquery-1.10.1.min.js # jQuery(symlink)
    d3.v3.min.js         # d3(symlink)
    linkdraw.js          # linkdraw(symlink)
    sample.html          # HTML
    config.json          # 描画する為の設定
    position.json        # ポジションを保存するファイル
    write.cgi            # ポジションを保存する為の CGI
```

ブラウザで確認してみましょう。

```
http://example.org/path/to/linkdraw/sample.html
```

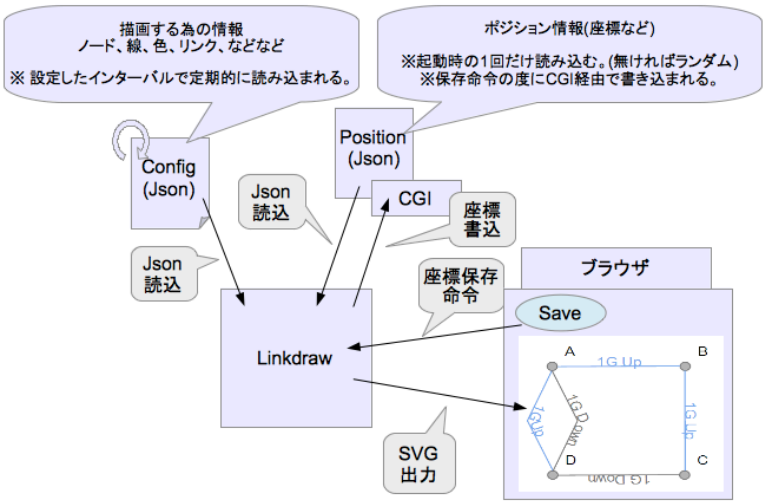
以下の様に表示されたら成功です。



アイテムを動かして “Position Save” を押すとその時のポジションが保存されます。ブラウザで再読み込みを行うと次回からは保存された座標で描画されます。また、 “configPath” で指定した設定ファイルの内容を更新していくことで描画するアイテムを制御することができます。設定ファイルの読み込みは、 “interval” で設定した値 (秒) で行われる為、ブラウザの再読み込み (更新) は不要です。 cron などで設定ファイルを定期的に置き換えることで自動更新が可能です。

8.5 仕組み

Linkdraw は json 形式で描画したいアイテムの情報を設定すると SVG で描画します。アイテムのポジションはランダムな座標が与えられブラウザ上に表示されます。表示されたアイテムはドラッグアンドドロップで動かすことが可能で、動かした後のポジションは保存することができます。保存されたポジション情報は、次回起動時に読み込まれる為、トポロジの形状を維持することができます。設定情報の json は、定期的に読み込まれ描画に反映されます。



8.6 まとめ

Linkdraw を使うとアイテムとアイテム間の繋がりを簡単に描画することができます。現在私は Linkdraw を使ってデータセンタ内部の物理情報とその上に乗るトラフィックを自動描画しています。誰かが設置した機器や追加したケーブルが自動で反映されトラフィックも把握できるのでちょっとだけ楽をしています。 Linkdraw についての情報は、日々更新していますので興味のある方は、 `https://github.com/mtoshi/linkdraw/wiki` をご確認下さい。

9 パッケージテストツール改善への取り組み

やまねひでき

9.1 はじめに

Debian にはパッケージの品質を向上するために各種ツールが開発されています。パッケージビルドの依存関係を正常に保つため、クリーンルームビルドを行う”pbuilder/cowbuilder”、 Debian ポリシーに沿った形でパッケージが作成されているかをチェックできる”lintian”、パッケージのインストール・アンインストールを chroot 環境で実施し問題を発見する”piuparts”などが代表的なものです。

この中で、あまり有効活用されていないのが”piuparts”です。その理由は”実施時間が膨大にかかる” ”デフォルトの出力が冗長であり、必要な情報を見つけ出すのが困難”という 2 点であると推測されます。本稿では簡単なテストプログラムを作成し、 piuparts における上記の問題を改善するアイデアを提案します。

9.2 piuparts の実行時間と内容の分析

現在、 ITP 中の”birdfont”という単体パッケージについて筆者の常用環境であるデスクトップ PC 上（ ストレージは一般的な HDD を利用）で piuparts を実施し、時間を計測しました。 piuparts のオプションは”-p”（ pbuilder で作った base.tgz を利用する）のみを選択しています。

```
$ time sudo piuparts -p birdfont_0.18-1_amd64.deb
real 8m53.714s
user 0m18.525s
sys  0m5.576s

$ time sudo piuparts -p birdfont_0.18-1_amd64.deb
real 12m31.246s
user 0m18.661s
sys  0m5.444s
```

ブレがあるが、おおよそ 9-12 分間の時間がかかるのがわかります (real)。 piuparts の動作としては chroot 環境に pbuilder のように最小限の”クリーンルーム”環境を作り、必要なパッケージを逐一ダウンロードしてインストールを行います。計測結果にあるように、一度のテストに最低でも 10 分近くを要する上、発生するパッケージのインストール作業に伴う disk I/O が強烈で他の作業に支障が出るほどであることから、開発者にとっては piuparts の実施は相当のストレスであると予想できます。一方、実際にプログラムが動作している時間は双方ともに 18 秒 (user) と極小であることから、何らかの形で”待ち時間”を減らしてやることで処理時間の改善が行えるであろうことが予想できます。

また、複数パッケージを生成する twitter クライアント”hotot”についても同様に piuparts を実施しました。

```
$ time sudo piuparts -p hotot_0.9.8.13+git20130311-2_amd64.changes

real 37m24.258s
user 0m52.284s
sys  0m16.888s
```

やはり real と比較すると user time が極端に小さいことが見て取れます。

9.3 SSD/tmpfs による処理時間の削減

ディスクアクセスが大量に発生していることから、HDD ではなく SSD、または”tmpfs” を利用した場合の速度を引き続き hotot を対象に比較することにしました。

```
$ time sudo piuparts -p -t /ssd hotot_0.9.8.13+git20130311-2_amd64.changes

real 2m3.441s
user 0m52.500s
sys  0m17.792s

$ time sudo piuparts -p -t /tmpfs hotot_0.9.8.13+git20130311-2_amd64.changes

real 1m41.406s
user 0m51.740s
sys  0m12.176s
```

双方とも大きな改善が認められます（37 分 → 1.7〜2 分）。しかし、機器の増設と費用の関係上、Debian の開発者すべてがこの手法を手元のマシン上で採用できる訳でないため、ソフトウェア側でも改善を引き続き試みました。

9.4 COW アプローチの採用

おおよそ、piuparts の動作は 5 つのステージに分けられます。

1. create base system with debootstrap（最小限の環境を debootstrap で構築）
2. download related packages (関連パッケージのダウンロード)
3. extract & install packages (パッケージの展開とインストール)
4. remove & purge packages (パッケージの削除)
5. upgrade packages（パッケージのアップグレード）

1 については、一応既に考慮されており、pbuilder で作成した base.tgz を利用するというオプションが用意されています。ここで、筆者は pbuilder からの連想で、pbuilder よりも高速に動作が可能な”cowbuilder”に着目しました。pbuilder に対する cowbuilder と同じアプローチで piuparts も COW(Copy On Write）を使ってディスクへの書き込みを減らす形で処理時間の短縮が図れると推測しました。

上川純一氏 (pbuilder 及び cowbuilder 作者) による Japan Linux Conference 2007 での cowbuilder の論文 (`http://lc.linux.or.jp/paper/lc2007/CP-06.pdf`) を参考にした所、いくつかの COW 実装 (LVM2, user-mode-linux, qemu, fl-cow, unionfs, aufs）について比較を行っていました。2007 年当時、既存実装では一長一短だった部分があったため cowbuilder では独自実装を行ったようですが、筆者はこの中で取り上げられていた Linux カーネルでの union ファイルシステム実装”aufs”が当時より好条件となっていることを発見しました。メインラインには残念ながらマージされていないものの、安定して開発が継続されており、確認した所 squeeze/wheezy/jessie の Debian のデフォルトカーネルで有効になっているため利用は非常に簡易です。よって、本稿では aufs を利用した改善を考察していきます。

9.5 短縮可能な部分の洗い出しとテスト実装結果

1. create base system with debootstrap（最小限の環境を debootstrap で構築）
2. download related packages (関連パッケージのダウンロード)
3. extract & install packages (パッケージの展開とインストール)
4. remove & purge packages (パッケージの削除)
5. upgrade packages（パッケージのアップグレード）

1 については、base.tgz の展開に時間がかかるので、最初に base.cow を用意し、COW によってそれを下敷きにして利用することでこれを短縮します（数秒程度）。

2 については、ホスト側のパッケージキャッシュを union mount することでダウンロード時間を削減し、かつダウンロードしたパッケージも union mount することで使いまわして毎回のダウンロードを行わないようにします。birdfont

を piuparts でテストした際にかかった時間は”Fetched 40.4 MB in 12s (3341 kB/s)”だったので 12 秒短縮（ 世界平均が 500kB/s ぐらいなので、 1 分は短縮可能）となります。この点はテストするパッケージの依存関係が巨大であればあるほど短縮が可能です。

3 については特に変更する点は見出せませんでした。

4 については、 uninstall と purge はそれぞれインストールして実施しているようですが、一旦 install ができた状態を COW で保持してその上に uninstall 用のレイヤーと purge 用のレイヤーを用意すればインストールは一回で済むので、ここを改善しました。 5 については、現状ではパッケージバージョンの比較などを行っていないためダウングレードテストになってしまう場合がありますが、作成ツール側ではアップグレード発生時以外はすべてキャンセルするなどして省略を図りました。

以上を元に、 shell script でテスト実装”yasp”（ Yet Another Simple/Speedy Piuparts）を作成し、テストを実施しました。

```
$ time sudo yasp test hotot_0.9.8.13+git20130311-2_amd64.changes
kernel: Linux

for sid
caching .deb files for hotot-common...
caching .deb files for hotot-gtk...
caching .deb files for hotot-qt...
caching .deb files for hotot...
INSTALL: install package(s)...
debconf: delaying package configuration, since apt-utils is not installed
lsof: WARNING: can't stat() fuse.gvfs-fuse-daemon file system /home/henrich/.gvfs
      Output information may be incomplete.
Done
REMOVE: remove package(s)...
Done
PURGE: purge package(s)...
(snip)
Done
Package version 1:0.9.8.13+git20130311-4 (in sid)
Package vesrion 1:0.9.8.13+git20130311-2 (now you test)
1:0.9.8.13+git20130311-2 =< 1:0.9.8.13+git20130311-4
Skipping UPGRADE test...

PASS: install/remove/purge/upgrade test

real32m28.090s
user0m16.484s
sys0m11.872s
```

上記のように、 37 分 → 32.5 分と 10% 以上の短縮が行えました。メッセージ出力自体も debconf 絡みのエラーメッセージなど以外はシンプルになっているため、状況を把握しやすくなっています。また、これを tmpfs 上で実施した所、以下のように 30 秒弱でテスト自体を実施できました。

（ 作業ディレクトリを tmpfs 上にした場合の結果）

```
real 0m34.477s
user 0m16.820s
sys  0m7.772s
```

9.6 結論

インストールテストの大幅な作業時間の短縮には SSD や tmpfs などの高速な IO を提供できるデバイスを利用することが重要です。しかし、ツール自体にも本稿で示唆したように aufs を利用するなどして改善の余地があることがわかります。可能な限り piuparts へのフィードバックを実施し、より Debian パッケージメンテナらが手軽にテストを実施できるよう改善を行いたいと思います。

10 /etc/network/interfaces について

西山和広

Debian でネットワークの設定を行うには /etc/network/interfaces ファイルに書く方法や NetworkManager を使う方法などがあります。サーバー系では主に /etc/network/interfaces が使われていて、デスクトップ環境や無線 LAN のついたノート PC などでは NetworkManager が使われることが多いようです。

ここでは /etc/network/interfaces について説明します。

10.1 /etc/network/interfaces とは?

Linux ではディストリビューションごとにネットワーク設定の仕組みが異なっています。 Debian 系では /etc/network/interfaces がネットワークの設定ファイルです。これは Red Hat Enterprise Linux や CentOS での /etc/sysconfig/network や /etc/sysconfig/network-scripts/ifcfg-eth0 などの設定ファイルに相当します。

/etc/network/interfaces は主に ifupdown パッケージで提供される ifup (ネットワークインターフェイス設定コマンド) と ifdown (設定解除コマンド) で使われます。他にも NetworkManager や guessnet などのツールやスクリプトから参照されることもあるようです。[*36]

10.2 interfaces ファイルの構造

interfaces ファイルは stanza (スタンザ) と呼ばれる固まりを並べる構造になっています。

例えば以下のような interfaces ファイルなら 6 個の stanza を含んでいます。

```
# lo の auto stanza と iface stanza
auto lo
iface lo inet loopback

# eth0 の allow-hotplug stanza と iface stanza
allow-hotplug eth0
iface eth0 inet dhcp

# eth1 の allow-hotplug stanza と iface stanza
allow-hotplug eth1
iface eth1 inet static
    address 192.168.1.1
    netmask 255.255.255.0
```

最初の 2 個の stanza は lo という 127.0.0.1 や ::1 に対応するループバックインターフェイスの設定で、ここをいじることは普通はないと思います。

次の 2 個の stanza は eth0 という最初のネットワークインターフェイスは DHCP で自動で設定を取得しています。

最後の 2 個の stanza は eth1 というネットワークインターフェイスは 192.168.1.1 という IPv4 アドレスを設定しています。 address や netmask の行も iface stanza に含まれます。

[*36] resolvconf や wireless-tools や wpasupplicant などは後述する /etc/network/if-*.d/ の仕組みを使っているので、 /etc/network/interfaces ファイルを直接参照することはありません。

10.2.1 stanza とは

stanza は ”iface”, ”mapping”, ”auto”, ”source”[*37] や ”allow-”[*38]の行が stanza の始まりです。 stanza は 1 行だけのこともあれば複数行になっていることもあります。

以下の例ではそれぞれ 1 行ずつの stanza が 5 個あります。

```
auto lo
iface lo inet loopback
allow-hotplug eth0
iface eth0 inet dhcp
allow-hotplug eth1
```

次の例では 3 行で 1 個の stanza になっています。 2 行目以降はインデントされていることもありますが、インデントはあってもなくてもかまいません。

```
iface eth1 inet static
    address 192.168.1.1
    netmask 255.255.255.0
```

10.2.2 コメント

”#”から始まる行はコメントです。行の途中や行末に”#”があっても設定値の一部になるだけでコメントにはなりません。

10.3 stanza の種類

10.3.1 auto stanza

auto stanza は `ifup -a` のコマンド実行時に up するインターフェイスの設定です。複数設定したい場合は、 1 行に複数並べてもいいし、複数回 auto stanza を書いても構いません。

```
# 1 行で書く例
auto eth0 eth1
```

```
# 複数書く例
auto eth0
auto eth1
```

10.3.2 allow-hotplug stanza

allow-hotplug stanza は”`ifup --allow=hotplug  eth0 eth1`”のように実行されたときに up する interface の設定です。[*39]

udev で NIC などが認識されたタイミングで up するインターフェイスを設定します。 auto だとデバイスの認識のタイミングの問題で up に失敗することがあるので、最近は allow-hotplug eth0 のようになっていることが多いようです。

etch や lenny では `/etc/udev/rules.d/` の中に、 squeeze や wheezy では `/lib/udev/rules.d/` の中に

```
SUBSYSTEM=="net", RUN+="net.agent"
```

という設定があり、`/lib/udev/net.agent` 経由で”`ifup --allow=hotplug`”が実行されています。

10.3.3 mapping stanza

mapping stanza はプログラムの実行結果によって設定を切り替えたいときに使います。

`/usr/share/doc/ifupdown/examples/` 以下に、ノート PC で差した PC カードの MAC アドレスによって設定を変えたり (get-mac-address.sh)、特定の IP アドレスに ping が通るかどうかによって設定を変えたり (ping-places.sh)

[*37] source stanza が使えるのは wheezy 以降のみです。
[*38] allow-hotplug と allow-auto があるようです。
[*39] interfaces の man page では ”allow-” stanza と説明されていて、 allow-auto は auto と同じ意味だと書かれています。

するサンプルファイルがあります。

10.3.4 source stanza

wheezy 以降の ifupdown パッケージでは source stanza というものが使えます。 interfaces ファイルの中で別のファイルを読み込む (include する) ものです。

例えば

```
source /etc/network/interfaces.d/*
```

という設定を書けば /etc/network/interfaces.d/lo や /etc/network/interfaces.d/eth0 などに設定を分割して取り込むということも出来ます。

ただし NetworkManager などの別のツールやスクリプトで /etc/network/interfaces を直接見ている場合に問題が起きる可能性があるという議論[*40]があるようです。

10.3.5 iface stanza

iface stanza は、 IP アドレスなどのネットワーク設定を記述します。"iface 名前 アドレスファミリ メソッド"という行で始まります。

iface stanza の最初の引数には eth0 などのような物理インターフェイス名を記述します。物理インターフェイス名の代わりに、論理名で home と記述して"ifup ath0=home"のように使うこともできます。[*41]

その他の設定については後述します。

10.4 iface オプション

アドレスファミリなどに関係なく共通で使えるオプションとして、以下の 4 種類があります。

pre-up ネットワークを up する前に必要な無線 LAN 関係の設定などに使われる。

up (post-up でも同じ) ネットワークを up した後に追加で設定するのに使われたりインターフェイスの増減が影響するデーモンの再起動をしたり VPN の接続をしたり manual メソッドで up 処理を書くのに使われたりする。

down (pre-down でも同じ) VPN の切断をしたり切断対象のネットワークの DNS サーバーの設定を外したりする。

post-down 無線 LAN 関係の停止処理などに使われる。

実行の順序は、それぞれ以下のようになります。

ifup pre-up → ifup の内部処理 → up (post-up)

ifdown down (pre-down) → ifdown の内部処理 → post-down

上記の設定と同じタイミングで `/etc/network/if-*.d/` に置かれたスクリプトも実行されます。[*42]

スクリプトには iface stanza の情報として環境変数 IFACE, LOGICAL, ADDRFAM, METHOD, MODE, PHASE, VERBOSITY が渡されます。他のオプションは"IF_"で始まる環境変数で渡されます。

wireless-tools パッケージを入れたら"wireless-"で始まるオプションが使えたり、 wpasupplicant パッケージを入れたら"wpa-"で始まるオプションを使えたり、 resolvconf パッケージを入れたら"dns-"で始まるオプションが使えたり、 ifenslave-2.6 パッケージを入れたら"slaves"オプションが使えたりするのは、この"IF_"で始まる環境変数を"/etc/network/if-*.d/"以下のスクリプトで使っているからです。

以下ではパッケージで導入されるスクリプトと独自スクリプトの例について説明します。

[*40] https://lists.debian.org/debian-devel/2013/01/msg00157.html

[*41] 以前、/etc/network/run/ifstate (wheezy では /run/network/ifstate) に"lo=lo"とか"eth0=eth0"のようにあって、何の意味があるんだろうと思っていたのですが、"ifup ath0=home"とすると ifstate に"ath0=home"と書き込まれていて、こういう場合に"="の左右が違うことがあるということを知りました。

[*42] run-parts で実行されるため /etc/cron*ly 以下などと同じくファイル名に"."などが含まれていると実行されません。

10.4.1 パッケージで導入されるスクリプトの例

パッケージで導入されるスクリプトの例として、 resolvconf パッケージを取り上げます。

resolvconf パッケージの説明には”resolvconf はシステムのネームサーバ情報を最新に保つためのフレームワークです。情報を提供するプログラム (ifup、 ifdown、 DHCP クライアント、 PPP デーモン、ローカルのネームサーバなど) と、情報を使うプログラム (DNS キャッシュやリゾルバライブラリなど) との間に入ります。”と書いています。ここではifup、 ifdown に関係する部分だけ説明します。

resolvconf パッケージをインストールすると

1. /etc/network/if-up.d/000resolvconf
2. /etc/network/if-down.d/resolvconf

というファイルが出来ます。

000resolvconf の内容をコメントや空行を除外して引用します。

```
% egrep '^[^#]' /etc/network/if-up.d/000resolvconf | cat -n
     1  [ -x /sbin/resolvconf ] || exit 0
     2  case "$ADDRFAM" in
     3    inet|inet6) : ;;
     4    *) exit 0 ;;
     5  esac
     6  R=""
     7  if [ "$IF_DNS_DOMAIN" ] ; then
     8          R="${R}domain $IF_DNS_DOMAIN
     9  "
    10  fi
    11  if [ "$IF_DNS_SEARCH" ] ; then
    12          R="${R}search $IF_DNS_SEARCH
    13  "
    14  fi
    15  if [ "$IF_DNS_SORTLIST" ] ; then
    16          R="${R}sortlist $IF_DNS_SORTLIST
    17  "
    18  fi
    19  for NS in $IF_DNS_NAMESERVERS ; do
    20          R="${R}nameserver $NS
    21  "
    22  done
    23  echo -n "$R" | /sbin/resolvconf -a "${IFACE}.${ADDRFAM}"
```

まず 1 行目には /sbin/resolvconf の存在チェックがあります。これはパッケージが削除されて設定ファイルだけ残っている時にエラーにならないようにするための処理です。

次に 2 から 5 行目では環境変数 ADDRFAM をチェックして IPv4 と IPv6 のときだけ続きの処理を実行します。ADDRFAM や IFACE は iface の行の情報が設定されている環境変数です。

続きの設定はほぼ同じような処理の繰り返しなので、一番良く使われると思われる dns-nameservers について説明します。

/etc/network/interfaces ファイルの iface stanza で”dns-nameservers 192.168.0.1”や”dns-nameservers 192.168.0.1 192.168.0.2”のように設定します。

その設定が 19 行目で `IF_DNS_NAMESERVERS` という環境変数で参照されています。他の設定も含めて resolvconf パッケージの独自設定には `dns-` を付けるようになっています。

設定を書く時に `dns-` を付け忘れて”nameservers 192.168.0.1”のように書いてしまうと resolvconf には反映されずに悩むことになるので注意が必要です。

このように環境変数で連携しているため、無効な設定があっても他で使っているかもしれないということで、設定ミスの自動チェックは難しいようです。

10.4.2 環境変数の例

独自スクリプトの例に入る前にスクリプトのなかで使える環境変数の例を紹介しておきます。

独自スクリプトを作成する時には、まず”up env > /var/tmp/env.txt”などで使える環境変数を確認するのがおすすめです。デバッグ中にも意図した通りの環境変数が設定されているかどうかを確認することをおすすめします。”ifup -v eth0”のように”-v”付きで実行された場合は VERBOSITY が 1 になるので、その場合だけデバッグ用の出力を増やすようにす

るのもおすすめです。

まず、単純な”iface eth0 inet dhcp”だけの設定で”up env > /var/tmp/env.txt”で保存した環境変数を例として載せておきます。他にも pre-up や down や post-down でも 調べてみたところ、 pre-up だと PHASE=pre-up になっているという違いがあり、 down や post-down だと PHASE=pre-down や PHASE=post-down になっている他に MODE=stop になっているという違いがありました。

```
METHOD=dhcp
MODE=start
LOGICAL=eth0
PHASE=post-up
ADDRFAM=inet
VERBOSITY=0
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
IFACE=eth0
PWD=/
```

そして、もっと複雑な例としてブリッジ設定や後述する独自スクリプトの設定などが入っている時の例も載せておきます。

/etc/network/interfaces で

```
iface br0 inet static
        # bridge
        bridge_ports eth0
        bridge_stp off
        bridge_fd 0
        bridge_maxwait 0
        # static
        address 192.168.253.29
        netmask 255.255.255.0
        gateway 192.168.253.1
        # iproute
        ip2-table 100
        ip2-net 192.168.253.0/24
        ip2-gateway 192.168.253.1
        post-up /etc/network/ip2-route.sh
        pre-down /etc/network/ip2-route.sh
        pre-up env > /var/tmp/env-$IFACE-$PHASE.txt
        post-down env > /var/tmp/env-$IFACE-$PHASE.txt
        # resolvconf
        dns-nameservers 192.168.253.1
```

のように設定している時に”/var/tmp/env-br0-pre-up.txt”は以下のようになります。元の設定ファイルで”-”で書いていても”_”で書いていても環境変数では”_”になるという点に注意が必要かもしれません。

```
IF_BRIDGE_FD=0
METHOD=static
MODE=start
LOGICAL=br0
IF_IP2_GATEWAY=192.168.253.1
PHASE=pre-up
IF_BRIDGE_MAXWAIT=0
IF_ADDRESS=192.168.253.29
ADDRFAM=inet
VERBOSITY=0
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
IF_IP2_NET=192.168.253.0/24
IF_GATEWAY=192.168.253.1
IF_METRIC=100
IF_NETMASK=255.255.255.0
IFACE=br0
IF_BRIDGE_STP=off
PWD=/
IF_IP2_TABLE=100
IF_BRIDGE_PORTS=eth0
IF_BROADCAST=+
IF_DNS_NAMESERVERS=192.168.253.1
```

IF_BRIDGE_ で始まる環境変数は bridge-utils パッケージに含まれる/etc/network/if-post-down.d/bridge と /etc/network/if-pre-up.d/bridge 使われます。

IF_IP2_ で始まる環境変数は次で説明する独自スクリプトで使っています。

10.4.3 独自スクリプトの例

例として以前作成したインターネットへの経路が複数ある環境での設定スクリプトを載せておきます。

example では 192.168.0.1 側でも 192.168.0.1 側でも自宅サーバーを公開していて、リクエストが来た側に応答を返すような設定になっています。

```
% cat /etc/network/ip2-route.sh
#!/bin/sh
#
# example:
#
#auto eth0
#iface eth0 inet static
#       address 192.168.0.2
#       netmask 255.255.255.0
#       gateway 192.168.0.1
#       ip2-table 100
#       ip2-net 192.168.0.0/24
#       ip2-gateway 192.168.0.1
#       post-up /etc/network/ip2-route.sh
#       pre-down /etc/network/ip2-route.sh
#auto eth1
#iface eth1 inet static
#       address 192.168.1.2
#       netmask 255.255.255.0
#       ip2-table 101
#       ip2-net 192.168.1.0/24
#       ip2-gateway 192.168.1.1
#       post-up /etc/network/ip2-route.sh
#       pre-down /etc/network/ip2-route.sh
#

[ -n "$IF_IP2_NET" ] || exit 0
[ -n "$IF_IP2_TABLE" ] || exit 0
[ -n "$IF_IP2_GATEWAY" ] || exit 0

if [ "$VERBOSITY" -eq 1 ]; then
        set -x
fi
case "$PHASE" in
        *up)
                ip route add $IF_IP2_NET dev $IFACE src $IF_ADDRESS table $IF_IP2_TABLE
                ip route add default via $IF_IP2_GATEWAY table $IF_IP2_TABLE
                ip rule add from $IF_ADDRESS table $IF_IP2_TABLE
        ;;
        *down)
                ip rule del from $IF_ADDRESS table $IF_IP2_TABLE
                ip route del default via $IF_IP2_GATEWAY table $IF_IP2_TABLE
                ip route del $IF_IP2_NET dev $IFACE src $IF_ADDRESS table $IF_IP2_TABLE
        ;;
esac

exit 0
```

まず ip2-net, ip2-table, ip2-gateway のすべてを必須設定として、抜けがある場合は何もしないようにしています。

次に VERBOSITY をチェックしています。ここでは ifup -v eth0 のように実行された時に後続のコマンドが実際にどういうコマンドラインで実行されるのかを表示するようにして、デバッグしやすくしています。

最後に up か down かに応じて ip コマンドを実行しています。

この設定ファイルを /etc/network/if-up.d/ と /etc/network/if-down.d/ の中にシンボリックリンクを作成しても良かったのですが、この例では /etc/network/ip2-route.sh に置いて、 post-up と pre-down で実行するようにしています。スクリプトをどう設置するのかは管理しやすい方法を選べば良いと思います。

スクリプトの説明はここまでです。次からは /etc/network/interfaces の設定の説明に戻ります。

10.5 inet アドレスファミリ

IPv4 のネットワーク設定です。

10.5.1 loopback メソッド

```
iface lo inet loopback
```

のことです。

iptables の設定に iptables-restore を使っているのなら、

```
# The loopback network interface
auto lo
iface lo inet loopback
  pre-up /sbin/iptables-restore < /etc/network/iptables.txt
```

のように lo の pre-up に設定しておけば他の interface が up される前に設定できていいかもしません。

10.5.2 static メソッド

address と netmask は必須です。

broadcast と network は address と netmask から自動設定可能なので、わざわざ書いて間違える可能性を増やすよりも省略しておく方がおすすめだと思います。

gateway も設定することが多いです。

10.5.3 manual メソッド

up や down などで全部自前で設定するときや "/etc/network/if-*.d/" 以下のスクリプトで設定する時などに使います。

bonding の slave 用 NIC に NetworkManager などが余計なことをしないように"iface eth1 inet manual"だけ書いておくという使い方も可能です。

10.5.4 dhcp メソッド

dhclient などで dhcp クライアントとしてネットワークを設定します。

10.6 inet6 アドレスファミリ

IPv6 のネットワーク設定です。

端末として接続するだけなら自動設定されることが多いので、わざわざ書くことは少ないかもしれません。ルーターやサーバーなどで固定 IP アドレスを付けたり、トンネルで接続する場合に書くことになります。接続方法によって設定内容が変わってくるので、詳細は省略します。

10.7 設定書き換え時の注意

ネットワーク接続中に /etc/network/interfaces を書き換えるのはあまりお勧めしません。

ifdown のときにも /etc/network/interfaces を見て何をするかが決まっているので、 dhcp を static に書き換えてから /etc/init.d/networking restart をしてしまうと、 dhcp クライアントのプロセスが残ってしまって、いつの間にか IP アドレスが変わって悩む、という現象が起きることがあるので注意が必要です。たとえば他にも up で route add されていることを前提にして down に route del を書いていると、手動で up に書いた処理を実行しないと down の処理でエラーになるというような問題があります。

安全な方法としては ifdown で停止した後で /etc/network/interfaces を書き換えて、 ifup するという手順になります。

10.8 参考文献

1. man interfaces で参照出来る interfaces(5) の man page
2. Debian リファレンスの 第 5 章 ネットワークの設定 http://www.debian.org/doc/manuals/debian-reference/ch05.ja.html

11　ngraph-gtk で楽々グラフ作成

伊東宏之

5 月にリリースされた Debian 7.0 “Wheezy” から ngraph-gtk が公式パッケージとして提供されるようになりました。 Ngraph は MS-DOS の時代から利用されてきた歴史あるグラフ作成プログラムです。現在は Windows 用がフリーウェア、 unix 用が GPL version 2 のフリーソフトとして配布されています。 ngraph-gtk はこの unix 用のソースを元に GTK+ 用に移植したものです。当初はウィジェット・ツールキットを Motif から GTK+ に変えただけでしたが、その後個人的に使いやすいように機能追加、仕様変更を行なったため、今では外観や機能にだいぶ違いが出てきました。

11.1　概要

ngraph-gtk は x-y グラフを作成するプログラムです。簡単な操作で綺麗なグラフを作成することができます。また、数式変換機能や任意関数でのフィッティングもできますので、簡単なデータ解析にも使えます。もちろんグラフのキャプションを作成する機能もありますし、作成したグラフは PostScript, PDF, SVG, PNG などにエクスポートできますので、Inkscape など他のソフトでさらに編集したり、 TeX の文章に貼り付けたりすることもできます。さらに sh 互換のスクリプト言語を使用して機能を追加したり、グラフの自動作成も可能です。

ngraph-gtk を使ったグラフの作成は 1. データを用意する、 2. ngraph-gtk でデータを開く、 3. 必要に応じて、データの変換やフィッティングの設定などを行う、 4. 軸のラベル、キャプションなどで体裁を整える、という手順になります。 ngraph-gtk-doc パッケージには詳しい日本語マニュアルが含まれており、その中の”2. チュートリアル”を読めば基本的な使い方は習得できるはずです。基本的な使い方はそちらを参照していただくことにして、ここではマニュアルに記述されていない仕様やオリジナルとの相違点などを多く記述したいと思います。ぜひ ngraph-gtk や ngraph-gtk-doc パッケージをインストールして、プログラムを動かしたり、ドキュメントに目を通したりしてみてください。

11.2　データファイル

11.2.1　一般的なデータ

ngraph-gtk は ascii テキストで記述されたデータファイルをグラフに描画するという思想で作られています。例えば $y = x^2$ のようなグラフを描かせる時でもデータファイルを用意する必要があります。データファイルは”データ”メニューの”追加”で描画用のデータとして設定できます。標準の設定では半角スペース、水平タブ、コンマ “,”、カッコ “(”, “)” のいずれかを区切り文字として、第 1 カラムが X、第 2 カラムが Y のデータとして読み込まれます。区切り文字の設定はファイル設定ダイアログで設定を変えることができますが、指定できるのは表示可能な ascii 文字のみです。また区切り文字が連続する場合はまとめて一つの区切りと解釈されます。例えば下記のような行は第 1 カラムが 2、第 2 カラムが 4 となります (区切り文字設定がデフォルトの場合) 。

```
2，，，4
```

データ設定ダイアログで”CSV 形式”にチェックすると上記の行は第 1 カラムが 2、第 2、第 3 カラムが 欠損、第 4 カラムが 4 となります。”CSV 形式”という表記は誤解を招きやすいのですが、半角スペース以外の区切り文字については連続

していても一つの区切りとして扱わないという設定で、よくデータの交換に使われる CSV 形式とは必ずしも互換ではありません。例えば ” (ダブルクォート) で囲まれた数値は正常に読み込めません。

X が等間隔で Y のデータしかないデータファイルの場合 X のカラム指定を 0 にするとテータを読み込んだ順に 1, 2, 3... が X の値になるので、このようなデータも問題なく描画できます。

文字列から浮動小数点への変換は strtod(3) を使用しています。これにより strtod(3) がサポートする 10 進数、 16 進数、無限 (INF, INFINITY)、 NAN を読み込むことができます。ただし ngraph-gtk では無限をサポートしていないので、無限は NAN として扱われます。また strtod(3) で変換できなかった場合 “CONT”, “BREAK”, “UNDEF”, “=”, “|” のいずれかであればデータとして読み込まれます。 “CONT”, “BREAK”, “=”, “|” は欠損データとして扱われますが、直線や曲線のプロットの場合 “BREAK”, “=” の前後で線が分割されます。 “UNDEF” は未定義値として扱われます。これらの文字列の読み込み時には大文字・ 小文字は区別されません。

11.2.2 矩形、対角線、矢印プロット用データ

ngraph-gtk では矩形や矢印のプロットも可能ですが、この場合データファイルの形式が変わります。例えば下記のデータを X カラムを 1、 Y カラムを 4 として、対角線でプロットすると (0, 1) から (3, 4) に線が引かれます。すなわち Y カラムとして指定されたデータは Y 座標ではなく、対角線の終点あるいは矩形の反対側の角の X 座標となります。

```
0, 1, 2, 3, 4
```

11.2.3 誤差棒プロット用データ

誤差棒のプロットも可能ですが、この場合もデータファイルの形式が変わります。例えば下記のデータを X カラムを 1、 Y カラムを 3 として、横誤差棒でプロットすると (1, 3) から (-2, 3) に誤差棒が引かれます ($1 = 0 + 1,\ -2 = 0 + (-2)$) 。一方、縦誤差棒でプロットすると (0, 7) から (0, -2) に誤差棒が引かれます ($7 = 3 + 4,\ -2 = 3 + (-5)$) 。誤差棒の両端につく線の長さはデータ設定ダイアログの”サイズ”で変えることができます。

```
0, 1, -2, 3, 4, -5
```

11.2.4 日付データ

ngraph-gtk では日付データもプロットすることができますが、ちょっとコツが必要です。具体的には日付として扱いたい数値を MJD (修正ユリウス日[*43]) に変換する必要があります。このために数式変換の関数には *MJD*(year, month, day) および *UNIX*2*MJD*(time) という関数が用意されています。例えば次のようなデータがあった場合、区切り文字に “/” を追加して X 軸の数式変換を “MJD(%1, %2, %3)” とします。

```
2013/06/28 0
2013/06/29 1
2013/06/30 2
```

次に X 軸の”スケール-スケール法”を”MJD”にします。これで軸のナンバリングに日付が表示されるようになりますが、表示が混み合って見難くなることがあるので軸設定ダイアログで”目盛数字-位置-方向”を”軸と斜交 1”などに設定しておくのがお勧めです。日付の表記は”目盛数字-フォーマット-日時書式”で strftime(3) とほぼ同様の書式を使って指定することもできます。

日付のデータが “2013-06-29” のような書式で与えられている場合は、区切り文字に “-” を追加することで同様に描画できますが、この時他のカラムに負の値を示す “-” があるとこれも区切り文字として解釈されてしまいます。今のところこの現象を回避することはできないので、あらかじめ awk などを使って書式を変換しておくことになります。

[*43] 1858 年 11 月 17 日 0000UT からの日数。

11.3 数式変換機能

読み込まれたデータは数式変換機能により指定した数式で変換してからプロットすることができます。数式はデータ設定ダイアログで指定できます。

数式変換機能は Ngraph のソースは使用せずに完全に新しく書き直しています。従来は数式設定時にはスキャナが数値リテラルや定数、演算子などのトークンを 1 バイトの識別子や浮動小数点に変換するのみで、構文解析は計算時に毎回行われるようになっていました。新しいコードでは数式設定時に構文木の生成まで行い、グラフ描画前に簡単な最適化を行なってから実際の計算が実行されます。これにより計算速度が大幅に向上し、また数式設定時に文法エラーのチェックも行われるようになりました。また、代入可能な変数や配列が使用可能になったり、各種関数が追加されたりという機能向上も行われています。

11.3.1 演算子

数式変換では四則演算や各種関数を使用できます。一部の演算子は一般的な表記と異なるので、注意が必要です。例えば代入演算子は :=、剰余演算子は \ を使います。また階乗を計算する ! や、べき乗を計算する ^ などの演算子も使うことができます。論理否定演算子は存在しないので *NOT*() 関数を使用します。

11.3.2 変数と配列

変数は e 以外のアルファベット 1 文字が使えます (e はネイピア数として定数で使われているため) 。また変数 x, y に値を代入することは可能ですが、データ読み込み時に指定したカラムのデータで上書きされます。変数は 0 で初期化されているので、初期化のための代入は必ずしも必要ではありません。なぜ変数に使えるのがアルファベット 1 文字かというと、複数文字の変数を使える場合、定数の綴りを間違えた時に変数と解釈されて間違いに気づきにくくなるという点と、現状では数式変換には 1 行の数式しか記述できないため 1 文字の変数でもわかりにくくなることはないだろうと考えたためです。

ngraph-gtk では配列も使うことができます。配列名は添字のための [ ] で定数と区別できるため、 % かアルファベットで始まり、_ かアルファベット・数字で構成される任意の長さの文字列を使うことができます。

11.3.3 関数定義

数式変換で独自の関数を定義することもできます。関数定義は下記の様な書式で行います。

```
def func(a, @b) {c := a * b[0]; sin(c)}; a[0] := PI; func(1/2, a)
```

2 番めの仮引数のように @ をつけると引数が配列であることを表します。関数の戻り値は最後に評価された式の値になります。この例では $\sin(1/2 \times \pi)$ を計算することになるので、式全体の値は 1 になります。変数はすべてローカルスコープで、グローバル変数はないので、関数に値を渡すときは引数か、メモリー関数を使うことになります。

11.3.4 最適化

数式変換では描画する前に簡単な最適化を行います。最適化では定数を数値に変換、数値のみの項は事前に計算、数値による割り算は掛け算に変換といった処理を行なっています。例えば式 1 は式 2 のように最適化されます。

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + x/10 \tag{1}$$
$$= 55 + x \times 0.1 \tag{2}$$

一方、式 3 のように書いた場合、式 4 の様に扱われるため足し算に関しては最適化は行われません。

$$x/10 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 \tag{3}$$
$$= ((((((((((x \times 0.1) + 1) + 2) + 3) + 4) + 5) + 6) + 7) + 8) + 9) + 10) \tag{4}$$

11.3.5 浮動小数点の誤差

ngraph-gtk ではすべての計算は倍精度浮動小数点で行われますが、よく知られているように浮動小数点の計算では誤差に注意が必要です。例えば 0.5 − 0.4 = 0.1 ですが、 ngraph script (詳細は後述) の iexpr コマンドで調べると 0.5 − 0.4 は 0.1 と等しくないことになってしまいます。

```
Ngraph$ iexpr '0.5-0.4::0.1'
0
```

このような場合は *EQ*() 関数で比較する桁数を指定すると期待通りの結果を得られることが多くなります。

```
Ngraph$ iexpr 'EQ(0.5-0.4, 0.1)'
0
Ngraph$ iexpr 'EQ(0.5-0.4, 0.1, 13)'
1
```

2 番めの実行例のように *EQ*() 関数の第 3 引数で比較の桁数を 13 桁に指定すると、 14 桁目が四捨五入されるため 0.5 − 0.4 が 0.1 と等しいという結果が得られます。

11.3.6 矩形、対角線、矢印、誤差棒プロットの数式変換

これらのプロットでは、 1 行のデータに対して、誤差の上限・下限のように 2 回数式変換が実行されます。この 2 回の呼び出しは定数 *FIRST* で区別できます (1 回目の呼び出しは *FIRST* が真) 。例えば次のようなデータを 3 ± 0.2 の誤差棒でプロットするときは *IF*(*FIRST*, %02 + %03, , %02 − %03) という数式が使えます。数式変換で y を使っていないのは、誤差棒プロットのデータフォーマットに従って、 1 回目の呼び出しでは $y = 3 + 0.2$、 2 回目は $y = 3 + 0$ となるので、数式変換使用時はわかりにくくなるためです。

```
1, 3, 0.2
```

なお、この複数回呼び出される数式はすべて独立した数式として扱われています。そのため、下記の式 (5) のように変数を使って複数の呼び出しの間で値を共有することはできません。このような場合は式 (6) のようにメモリー関数を使うことができます。メモリー関数はすべての数式変換で共有されるので、複数のデータファイル間で値をやり取りすることもできます。

$$IF(FIRST, a := 1, a * 2) \tag{5}$$

$$IF(FIRST, M(0,1), RM(0) * 2) \tag{6}$$

11.3.7 その他

式は “;” を区切りとして、複数記述することができます。最後の式の値が数式変換の結果として使用されます。複数の式を記述する別の方法として *PROG*1(), *PROG*2(), *PROGN*() 関数を使用することもできます。例えば *DIF*(y) とほぼ同様の計算は *PROG*1($y - a, a := y$) の様に記述できます。 ngraph-gtk version 6.06.09 までは式の終端を表す記号として “=” が使われていたため代入や比較の演算子には記号 “:” を使用していました (:, ::, >:, +: など) 。しかし、使っていてちょっとわかりにくい印象だったので 6.06.10 以降は代入演算子として “:=” (Pascal と同じ) 、比較や自己代入演算子は C と同じ表記が使えるようにしました。

ngraph-gtk の数式変換には制御構造の構文がありません。条件分岐や繰り返しは *IF*() や *FOR*() などの関数を使用します。関数の引数では式の終端を表す “;” は使用できないので、複数の式を実行するためには *PROG*1() 関数などを使用します。なお *FOR*() 関数では関数呼び出し時に繰り返しの回数が決まってしまいます。これは数式変換の実行を中断する機能がないので、無限ループになった時プログラム自体を強制終了させる以外に停止手段がないためです。

11.4 ngraph script

11.4.1 ngraph shell

ngraph-gtk には ngraph shell というインタープリターがあり、このシェルで ngraph script という言語を実行できます。マニュアルから引用すると”ngraph スクリプトの文法は、 UNIX の sh (シェル) からジョブ制御機能を取り去り、オブジェクト操作命令を追加したもの”で、機能や実行速度の面から複雑な処理は外部のプログラムに任せるという思想で作られています。”ジョブ制御機能を取り去り”とあるとおり、コマンドをバックグラウンドで実行することはできません。またパイプも各プロセスはひとつずつ順に起動され、データは作業ファイル経由で受け渡されます。

ngraph-gtk ではグラフは ngraph オブジェクトの集合で構成されており ngraph script から各オブジェクトの設定を変更できますので GUI を全く使わずにグラフを書くことも原理的には可能です。しかし、グラフを出力するためには多くの設定や操作を行う必要があるため、すべて手動でグラフを作成するのはあまり現実的ではありません。実際には他のプログラムからスクリプトを出力してグラフの自動作成や作成支援を行ったり、グラフ設定の一部を手動で変更するというような使い方が一般的と思います。

例えばすべてのデータで線の太さを変えたいというような場合 GUI で一つ一つ設定するのは面倒な作業ですが ngraph script を使えば下記の様に簡単です。

```
Ngraph$ file:0-!:line_width=80
```

また ngraph-gtk にはアドインと呼ばれる機能があります。これは登録したスクリプトを実行するだけの機能ですが、メニューに登録できるため GUI で使いやすくなっています。

前述のとおり script の文法は sh と互換性があるため、詳細は unix や linux のマニュアルで確認できますが、独自に追加されたコマンドや、オブジェクトの仕様については今のところドキュメントが存在しないので、既存のスクリプトを参考にしたり、ソースを確認したりしないと詳細がわかりません。これらのドキュメントの充実は今後の課題です。

11.4.2 ngraph object

先に述べた通り ngraph-gtk では ngraph オブジェクトでグラフを構成します。 ngraph-gtk ではオブジェクトのメンバー変数・関数のことを”フィールド”と呼んでいます。オブジェクトのフィールドには読込・書込・実行のパーミションがあり、ユーザーからの設定変更などを制限しています。オブジェクトは new コマンドで生成され del コマンドで削除されます。

```
Ngraph$ new file
Ngraph$ del file:0
```

生成されたオブジェクトは id が付与されます (id フィールド) 。また各インスタンスはつぎの id のインスタンスへのポインタを保持しています (next フィールド)。リストの先頭のインスタンスはオブジェクト自身が知っています。なお next フィールドを持たないオブジェクトは 1 つのインスタンスしか生成できません。また init フィールドに実行属性がない場合、そのオブジェクトのインスタンスは生成できません。なおインスタンスの id はインスタンスを削除したり、並び順を変えたりすると振り直されますので、実行中に変わる可能性があります。

上記のような仕組みから、オブジェクトのインスタンスにはそのオブジェクト経由でアクセスすることになります。インスタンスの設定変更や機能の呼び出しは”オブジェクト名:インスタンスリスト:フィールド名=引数”の書式になっています。”インスタンスリスト”ではインスタンスの id をコンマで区切って複数指定したり、”3-6”の様に範囲で指定することもできます。また”!”は最後の id を表します。例えば次のような指定が可能です。

```
Ngraph$ arc:1,3,5-8,!:line_width=80
```

また、オブジェクトの name フィールドにアルファベットまたは “_” で始まりアルファベット・数字および “_” で構成される名前を設定できます。名前を設定されたインスタンスにはその名前で指定することも可能です。

```
Ngraph$ arc:1:name=my_arc
Ngraph$ arc:my_arc:rx=6000
```

インスタンスの設定値を得たい時は、下記のように get コマンドかオブジェクト置換を使用します。

```
Ngraph$ get file:0 file
file:test1.dat
Ngraph$ echo ${file:0:file}
test1.dat
```

オブジェクト置換の中ではシェル変数などの展開は行われないため id やフィールドの指定にシェル変数を使用したい場合は、かならず get コマンドを使用することになります。

11.5 動作の高速化

先に述べた通り ngraph-gtk では数式変換機能を新しく書きなおして高速化しましたが、ほかにもコードを見なおして高速化した箇所がいくつかあります。例えば、従来はオブジェクト、オブジェクトフィールド、シェルコマンドなどの探索に線形探索が使われていましたが ngraph-gtk ではハッシュを利用しています。またデータファイルの読み込みも極力無駄な処理を行わないように見なおしました。これらの最適化により、次に示すようにかなりの高速化が実現できています。

```
1 000 000 行のデータを読み込んで gra2null デバイスに出力
mngraph はオリジナル、 ngraph は ngraph-gtk の実行ファイル。

$ time mngraph -i file_test.ngp
mngraph -i file_test.ngp  4.60s user 0.02s system 99% cpu 4.617 total
$ time ngraph -i file_test.ngp
ngraph -i file_test.ngp  1.86s user 0.04s system 99% cpu 1.903 total
```

本資料のライセンスについて

本資料はフリー・ソフトウェアです。あなたは、 Free Software Foundation が公表した GNU GENERAL PUBLIC LICENSE の ”バージョン２”もしくはそれ以降が定める条項に従って本プログラムを再頒布または変更することができます。

本プログラムは有用とは思いますが、頒布にあたっては、市場性及び特定目的適合性についての暗黙の保証を含めて、いかなる保証も行ないません。詳細については GNU GENERAL PUBLIC LICENSE をお読みください。

excuse you from the conditions of this License. If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all. For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices. Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded. In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time. Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number. If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation. If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission. For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this. Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

    NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

     END OF TERMS AND CONDITIONS

    How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program. It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year> <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License. Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary. Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs. If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library. If this is what you want to do, use the GNU Library General
Public License instead of this License.

ソースコードについて

このプログラムは tex で記述されたものです。ソースコードをご希望の場合は dancer@debian.org まで連絡ください。ちなみにソースコードは

git://anonscm.debian.org/tokyodebian/monthly-report.git

の printed-2013-gum タグから取得できます。

```
$ git clone -b printed-2013-gum git://anonscm.debian.org/tokyodebian/monthly-report.git
```

Debian オープンユーズロゴ ライセンス

Copyright (c) 1999 Software in the Public Interest
Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
files (the "Software"), to deal in the Software without restriction,
including without limitation the rights to use, copy, modify, merge,
publish, distribute, sublicense, and/or sell copies of the Software,
and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN
AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR
IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

『 あんどきゅめんてっどでびあん』について

本書は、 2013 年 6 月 29 日に開催された『 大統一 Debian 勉強会 2013』で発表/使用された資料・ 小ネタ・ 必殺技などを一冊にまとめたものです。内容は無保証、つっ こみなどがあれば、東京エリア Debian 勉強会/関西 Debian 勉強会/福岡 Debian 勉強会にて。

あんどきゅめんてっどでびあん - 大統一 Debian 勉強会 2013 特集号 -

2013 年 06 月 29 日　　初版第 1 刷発行

2013 年 08 月 12 日　　初版第 2 刷発行

東京エリア Debian 勉強会/関西エリア Debian 勉強会/福岡 Debian 勉強会（ 編集・ 印刷・ 発行）