Debian パッケージの作り方

～最初から最後まで～

佐々木洋平
uwabami@debian.or.jp

北大・理/地球流体電脳倶楽部/Debian JP Projcet

2010 年 3 月 5 日

どーも

佐々木です

今日の目標

- Debian パッケージが作れるようになること

今日の目標

- Debian パッケージが作れるようになること
- 月曜からの実習で, TA になってもらうこと

Agenda

- Package
  - deb package inside
  - 余談: dpkg-deb でパッケージを再構築
- パッケージ作成
  - 前準備
    - 環境変数の設定
    - 最低限必要なパッケージの導入
    - ソースの取得, 確認
  - パッケージ作成
    - CDBS
    - rules の調整
    - 制御情報の編集
    - debuild, lintian
- パッケージのビルド・インストールテスト
- まとめ

佐々木のパッケージ作成遍歴

佐々木のパッケージ作成遍歴

1. 売り物ソフトウェアを dpkg で管理するために弄り始める

佐々木のパッケージ作成遍歴

1. 売り物ソフトウェアを dpkg で管理するために弄り始める
2. 自分達の作っているソフトウェアの deb パッケージを作成し始める.

佐々木のパッケージ作成遍歴

1. 売り物ソフトウェアを dpkg で管理するために弄り始める
2. 自分達の作っているソフトウェアの deb パッケージを作成し始める.
3. どうせなら本家に...

佐々木のパッケージ作成遍歴

1. 売り物ソフトウェアを dpkg で管理するために弄り始める
2. 自分達の作っているソフトウェアの deb パッケージを作成し始める.
3. どうせなら本家に... ← イマココ

お願い

Howto の詳細にはあまり触れません

- どんどん質問して下さい.
- そこから派生して, どんどん話が深くなる, 筈

Package

Package?

- バイナリパッケージ
- ソースパッケージ

バイナリパッケージ

コンパイル後のソフトウェアなどをすぐ利用できる形にまとめたもの

- Debian では `.deb` ファイル
- 制御情報とデータを `tar.gz` に圧縮し, バージョン情報とともに `ar(1)` でまとめたもの

ソースパッケージ

バイナリパッケージの素材をまとめたもの

- オリジナルのソース一式
  `.orig.tar.gz`
- パッケージの情報
  `.dsc`
- バイナリパッケージを作成するための変更
  `.diff.gz`
  - Debian 固有のパッケージ等の場合は存在しない

展開してみる

- `dpkg-deb` コマンドを使用
- 例えば **rabbit** を展開してみると...

展開してみる (続き)

```
% dpkg-deb -x rabbit_0.6.1-1_all.deb rabbit
% dpkg-deb -e rabbit_0.6.1-1_all.deb rabbit/DEBIAN
% cd rabbit ls
DEBIAN/  usr/
% tree
.
|-- DEBIAN
|   |-- control
|   |-- md5sums
|   `-- preinst
`-- usr
    |-- bin
    |   |-- rabbit
    ...
    |-- lib
    |   `-- ruby
    |       `-- 1.8
    ...
    `-- share
        |-- doc
        |   `-- rabbit
    ...
179 directories, 575 files
```

制御情報

```
% ls -R DEBIAN
DEBIAN:
control  md5sums  preinst*
```

control メンテナの名前, 対応するソースパッケージ名, 依存関係などを記述

md5sums 提供される各ファイルの md5 checksum

preinst インストール作業の前に実行される hook シェルスクリプト. 他に `postinst`, `prerm`, `postrm` なども使用

バイナリパッケージ再構築

売り物のソフトウェアを dpkg で管理したい, とか?

- alien で rpm を deb に変換
- 上記 dpkg-deb -e|-x でファイルを展開
- 適切に配置, 修正
- dpkg-deb -b で再アーカイブ

例: Intel Compiler Ver.8

最近は Intel Compiler に愛がないのでやってませんが.

```
% tar xvzf l_cc_pc_8.1.028.tar.gz
% cd l_cc_pc_8.1.028
% rm -rf *64*
% sudo alien *.rpm
% rm *.rpm
% sudo chown $USER *.deb
% mkdir tmp
% dpkg-deb -e intel-icc8_8.1-29_i386.deb tmp/DEBIAN
% dpkg-deb -x intel-icc8_8.1-29_i386.deb tmp/
% echo DESTINATION=/opt/‘ls tmp/opt‘ >> tmp/DEBIAN/postinst
% cat <<EOF >> tmp/DEBIAN/postinst
 for FILE in $(find $DESTINATION/bin/ -regex \
 ...                    ...
 ... postinst の修正 ...
 ...                    ...
% dpkg-deb -b tmp intel-icc8_8.1-29_i386.deb
% dpkg -i intel-icc8_8.1-29_i386.deb
% dpkg -i intel-iidb8_8.1-46_i386.deb
% dpkg -i --force-overwrite intel-isubh8_8.1-29_i386.deb
```

それはそれとして

公式パッケージにしよう, とか考えると真面目にパッケージ作ることになります.

パッケージ作成

パッケージ作成

- GNU hello を例に

前準備

環境変数の設定

- パッケージメンテナの名前, メールアドレスを設定
- GPG 鍵の記述に合わせておくと吉

```
DEBFULLNAME="Youhei SASAKI"; export DEBFULLNAME
DEBEMAIL=uwabami@gfd-dennou.org ; export DEBEMAIL
```

最低限パッケージの導入

build-essential メタパッケージを導入

```
% sudo aptitude install build-essential
```

1. libc6-dev
2. g++
3. make
4. dpkg-dev

が導入される

build-essential の情報

```
% apt-cache show build-essential
Package: build-essential
Priority: optional
Section: devel
Installed-Size: 48
Maintainer: Matthias Klose <doko@debian.org>
Architecture: amd64
Version: 11.4
Depends: libc6-dev | libc-dev, g++ (>= 4:4.3.1), make, dpkg-dev (>= 1.13.5)
Filename: pool/main/b/build-essential/build-essential_11.4_amd64.deb
Size: 7126
MD5sum: 86a942017ad93721c91212398a828a0c
SHA1: 5ac2ba90444e1eaed96b2163389a8812eb107b01
SHA256: 3dbd2e6b4e998412a6ad4d32b242523559b536168bcce7f227c7ce30256808a5
Description: Informational list of build-essential packages
 If you do not plan to build Debian packages, you don't need this
 package.  Starting with dpkg (>= 1.14.18) this package is required
 for building Debian packages.
 .
 This package contains an informational list of packages which are
 considered essential for building Debian packages.  This package also
 depends on the packages on that list, to make it easy to have the
 build-essential packages installed.
 .
----- snip -----
```

ソースの取得, 確認

動作確認

- ソフトウェアがちゃんと動作することを確認
- patch 書いたなら取っておくと吉

GNU helloの取得

http://www.gnu.org/software/hello/

GNU hello は configure ; make ; make install で導入する非常に標準的なソフトウェア.

./configure

```
% cd hello-2.4
% ./configure
...
```

- エラーがでなければこれで終了
- 足りないならば...?

apt-file

足りないファイルを提供しているパッケージを探して導入

```
% sudo aptitude install apt-file
% sudo apt-file update
% apt-file search [file 名]
```

- configure が通るまでくりかえす.
- 導入したパッケージはメモしておくと吉.

make

```
% make
make all-recursive
make[1]: Entering directory ‘/home/uwabami/Desktop/hello-2.4’
Making all in contrib
make[2]: Entering directory ‘/home/uwabami/Desktop/hello-2.4/contrib’
make[2]: Nothing to be done for ‘all’.
make[2]: Leaving directory ‘/home/uwabami/Desktop/hello-2.4/contrib’
...
make[2]: Entering directory ‘/home/uwabami/Desktop/hello-2.4’
make[2]: Leaving directory ‘/home/uwabami/Desktop/hello-2.4’
make[1]: Leaving directory ‘/home/uwabami/Desktop/hello-2.4’
```

動作確認

```
% ./src/hello
世界よ、こんにちは！
% ./src/hello -n
 ┌──────────────────┐
 │世界よ、こんにちは！│
 └──────────────────┘
% ./src/hello -g "Good Night. .."
Good Night...
```

事前準備

- 環境変数の設定
- 必要なパッケージの導入
- 動作確認

ライセンスとコピーライトも確認しておくこと!!

パッケージ作成開始

雛形の作成

dh_make コマンドで作成

A. dh-make

```
% apt-file search dh-make
cli-common-dev: /usr/bin/dh_makeclilibs
cli-common-dev: /usr/share/man/man1/dh_makeclilibs.1.gz
...
dh-make: /usr/bin/dh_make
...
% sudo aptitude install dh-make
```

dh_make --help

```
% dh_make --help
...
Copyright (C) 1998-2009 Craig Small <csmall@debian.org>
...
  Usage: dh_make [options]
  -c, --copyright <type> use <type> of license in copyright file
                         (apache|artistic|bsd|gpl1|gpl2|gpl3|lgpl1|lgpl2|lgpl3)
      --dpatch           using dpatch to maintain patches
      --quilt            using quilt to maintain patches
  -e, --email <address>  use <address> as the maintainer e-mail address
  -n, --native           the program is Debian native, don't generate .orig
  -f, --file <file>      specify file to use as the original source archive
  -r, --createorig       make a copy for the original source archive
  -s, --single           set package class to single
  -i, --indep            set package class to arch-independent
  -m, --multi            set package class to multiple binary
  -l, --library          set package class to library
  -k, --kmod             set package class to kernel module
      --kpatch           set package class to kernel patch
  -b, --cdbs             set package class to cdbs
  -a, --addmissing       reprocess package and add missing files
  ...
```

dh_make の代表的なオプション

–createorig, -r オリジナルのソースファイル (.orig.tar.gz) を作成

–copyright gpl, -c gpl ライセンスが代表的なモノの場合, 指定しておくと雛形の時点で結構できあがっていて非常に楽.

–single, -s シングルバイナリのパッケージを作成. ライブラリの場合には -l とする.

–cdbs, -b CDBS(後述) を使用する.

–quilt, –dpatch パッチを当てることが決まっているのであれば --dpatch もしくは--quilt を指定しておくと良い.

dh_make 実行

```
% dh_make --createorig --copyright gpl --single --cdbs
  or
% dh_make -r -c gpl -s -b
Maintainer name : Youhei SASAKI
Email-Address   : uwabami@gfd-dennou.org
Date            : Sun, 25 Oct 2009 00:08:43 +0900
Package Name    : hello
Version         : 2.4
License         : gpl3
Using dpatch    : no
Using quilt     : no
Type of Package : cdbs
Hit <enter> to confirm:
```

debian ディレクトリ

```
.
|-- README.Debian          (パッケージの README)
|-- changelog              (パッケージのチェンジログ)
|-- compat                 (パッケージのバージョン)
|-- control                (パッケージ情報)
|-- copyright              (パッケージのコピーライト情報)
|-- cron.d.ex              (パッケージで cron を使う場合の設定ファイル)
|-- dirs                   (パッケージでデータを配置するディレクトリ名の設定)
|-- docs                   (パッケージに含めるドキュメントファイルを指定する)
|-- emacsen-install.ex     (emacs 用設定ファイル)
|-- emacsen-remove.ex      (emacs 用設定ファイル)
|-- emacsen-startup.ex     (emacs 用設定ファイル)
|-- hello.default.ex       (パッケージで debfonf を使う場合の設定ファイル)
|-- hello.doc-base.EX      (パッケージで doc-base を使う場合の設定ファイル)
|-- init.d.ex              (パッケージで init.d を使う場合の設定ファイル)
|-- init.d.lsb.ex          (パッケージで init.d を使う場合の設定ファイル)
|-- manpage.1.ex           (manpage の雛形)
|-- manpage.sgml.ex        (manpage の雛形)
|-- manpage.xml.ex         (manpage の雛形)
|-- menu.ex                (メニューの雛形)
|-- postinst.ex            (postinst メンテナファイルの雛形)
|-- postrm.ex              (postrm メンテナファイルの雛形)
|-- preinst.ex             (preinst メンテナファイルの雛形)
|-- prerm.ex               (prerm メンテナファイルの雛形)
|-- rules                  (パッケージビルドスクリプト)
`-- watch.ex               (アップストリームチェック用ファイル)
```

debian ディレクトリ (続き)

- パッケージメンテナが弄るのはこのディレクトリ内のファイルのみ
- オリジナルのソースに変更を加える場合には `quilt` や `dpatch` など, パッチシステムを利用すると吉

debian ディレクトリの整理

- .ex, .EX を削除.

```
% rm -f debian/*.ex debian/*.EX
```

CDBS

CDBS

- debian/rules に従いパッケージが作成される
  - いわゆる Makefile.
  - make の文法で必要なターゲットを記述
- そのままだと大変なので
  - `debhelper`
  - `CDBS`

  なんかを使うと吉
  - 特に「./configure ; make ; make install」なソフトウェアは, CDBS が非常に便利

debian/rules

```
% cat debian/rules
#!/usr/bin/make -f

include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/autotools.mk


# Add here any variable or target overrides you need.
```

debhelper.mk debhelper コマンド群を必要に応じて呼ぶ

autotools.mk autotools でのパッケージ作成を自動化

一旦バイナリパッケージを作成

- rules の binary ターゲットを実行

```
% sudo aptitude install fakeroot
% fakeroot debian/rules binary
...
```

- 一つ上にバイナリパッケージができる
- debian/[パッケージ名] に注目

debian/hello

```
|-- DEBIAN
|   |-- control
|   `-- md5sums
`-- usr
    |-- local
    |   |-- bin
    |   |   `-- hello
    |   `-- share
    |       |-- info
    |       |   `-- hello.info
    |       |-- locale
    |       |   |-- bg
    |       |   |   `-- LC_MESSAGES
    |       |   |       `-- hello.mo
--------- snip ------------------
...
103 directories, 61 files
```

debian/hello(続き)

- `hello` が `/usr/local/bin` にある
  - 正しくは...?
- `configure` 時に`--prerfix` が無いから

deian/rules の修正

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

include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/autotools.mk

DEB_CONFIGURE_EXTRA_FLAGS:= --prefix=/usr
```

debian/以下の修正

- 望む配置になるまで修正の繰り返し

debian/* 修正後

```
% ls debian/
changelog  control  copyright  rules*
% cat debian/rules
#!/usr/bin/make -f

include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/autotools.mk

DEB_CONFIGURE_EXTRA_FLAGS:= --prefix=/usr
DEB_INSTALL_DOCS_ALL      := NEWS README TODO
DEB_INSTALL_CANGELOGS_ALL:= ChangeLog.0
DEB_INSTALL_INFO_ALL      := doc/hello.info
```

制御情報の編集

制御情報の編集

具体的には

1. debian/control
2. debian/changelog
3. debian/copyright

の三つです.

control の例

```
Source: hello
Section: devel
Priority: optional
Maintainer: Youhei SASAKI <uwabami@gfd-dennou.org>
Build-Depends: cdbs, debhelper (>= 7), autotools-dev
Standards-Version: 3.8.3
Homepage: http://www.gnu.org/software/hello

Package: hello
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: The classic greeting, and a good example
 The GNU hello program produces a familiar, friendly greeting.  It
 allows non-programmers to use a classic computer science tool which
 would otherwise be unavailable to them.
 .
 Seriously, though: this is an example of how to do a Debian package.
 It is the Debian version of the GNU Project's 'hello world' program
 (which is itself an example for the GNU Project).
```

changelog の例

```
% cat debian/changelog
hello (2.4-1) unstable; urgency=low

  * Initial release

 -- Youhei SASAKI <uwabami@gfd-dennou.org>  Sun, 25 Oct 2009 00:08:43 +0900
```

- ITP のバグ番号を記述する雛形がある
  → ITP している場合には埋めておくと吉

copyright の例

ライセンス表記が明確だと楽ですね.

```
...
It was downloaded from:

    http://www.gnu.org/software/hello/

Upstream Author:

Authors of GNU Hello.

  Copyright (C) 1999, 2005, 2006 Free Software Foundation, Inc.

  Copying and distribution of this file, with or without modification,
  are permitted in any medium without royalty provided the copyright
  notice and this notice are preserved.

The following contributions warranted legal paper exchanges with the
Free Software Foundation.  See also the ChangeLog and THANKS files.

Mike Haertel
David MacKenzie
Jan Brittenson
Roland McGrath
Charles Hannum
Bruce Korb              hello.c, configure.ac.
Karl Eichwalder         all files.
Karl Berry              all files.
```

debuild, lintian

debuild & lintian

- バイナリパッケージとソースパッケージの作成
- パッケージのポリシー違反の check

devscripts と linitian を使用

```
% sudo aptitude install debuild lintian
```

debuild

```
% debuild -rfakeroot -uc -us
...
dpkg-deb: ‘../hello_2.4-1_amd64.deb’ にパッケージ ‘hello’ を構築しています.
 dpkg-genchanges  >../hello_2.4-1_amd64.changes
dpkg-genchanges: including full source code in upload
dpkg-buildpackage: full upload (original source is included)
Now running lintian...
W: hello: new-package-should-close-itp-bug
Finished running lintian.
```

- lintian の Warning を減らす.
- Warning はともかく Error は駄目

パッケージのビルド・インストールテスト

ビルドテスト

大雑把に言えば, ソースパッケージを元に

- `debian/control` の内容を元に構築に必要な他のパッケージを導入し
- `debian/rules` を実行してバイナリパッケージを作成し
- `lintian` に怒られないパッケージができあがる

ことのテスト.

pbuilder

- ビルドテストには pbuilder を使用
  - 必要最小限の Debian パッケージが導入された環境の tar.gz を用意
  - パッケージのビルド時にその tar.gz を展開し chroot してパッケージの作成

pbuilder 環境を導入します

```
% sudo aptitude install pbuilder
% sudo pbuilder  --create --distribution sid
```

/var/cache/pbuilder/base.tgz ができる

ビルドテスト

```
% sudo pbuilder --build \
                --distribution sid \
                --basetgz /var/cache/pbuilder/base.tgz \
                hello_2.4-1.dsc
```

ビルドテストに通ると/var/cache/pbuilder/result 以下に結果が置かれる

インストール/アンインストールテスト

piuparts を使用.

```
% sudo aptitude install piuparts
```

pbuilder で作成した base.tgz を使用してテスト

```
% sudo piuparts -d sid -b /var/cache/pbuilder/base.tgz hello_2.4-1_amd64.deb
...
0m50.1s DEBUG: No broken symlinks as far as we can find.
0m51.3s INFO: PASS: Installation, upgrade and purging tests.
0m51.3s DEBUG: Starting command: ['chroot', '/tmp/tmpZ2-nup', 'umount', '/proc']
0m51.3s DEBUG: Command ok: ['chroot', '/tmp/tmpZ2-nup', 'umount', '/proc']
0m51.7s DEBUG: Removed directory tree at /tmp/tmpZ2-nup
0m51.7s INFO: PASS: All tests.
0m51.7s INFO: piuparts run ends.
```

ここまできたらパッケージは一通り作成完了です.

まとめ

- Deb パッケージ作成, 最初から最後まで
  - `GNU hello` を題材に

触れていないこと

- ライブラリパッケージの作成
  - 共有ライブラリ, 静的ライブラリ+ヘッダ, デバッグシンボル
- 単一のソースから複数のバイナリパッケージを作成
  - カーネルパッケージを作成