Grand Unified Debian

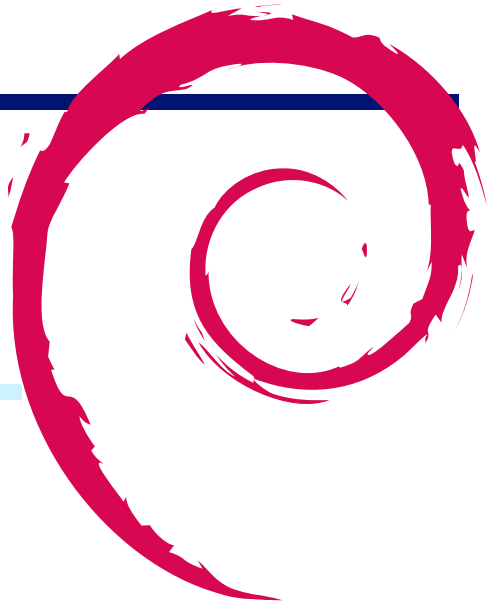
銀河系唯一のDebian専門誌

東京エリア/関西Ｄｅｂｉａｎ勉強会

あんどきゅめんてっど でびあん 2022 年夏号 2022 年 8 月 13 日 初版発行

デビアン勉強会

目次



1 Introduction

1.1 東京エリア Debian 勉強会

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

Debian 勉強会の目的は下記です。

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

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

1.2 関西 Debian 勉強会

関西 Debian 勉強会は Debian GNU/Linux のさまざまなトピック (新しいパッケージ、Debian 特有の機能の仕組、Debian 界隈で起こった出来事、などなど）について話し合う会です。

目的として次の三つを考えています。

- メーリングリストや掲示板ではなく、直接顔を合わせる事での情報交換の促進
- 定期的に集まれる場所
- 資料の作成

それでは、楽しい一時をお楽しみ下さい。

2 Debian Updates (OSC 2021 Online/Fall)

杉本典充

2.1 アジェンダ

- Debian とは
- Debian JP Project と Debian 勉強会
- Debian 11 (bullseye)
- 今後のイベント

2.2 Debian とは

- `https://www.debian.org/`
- ボランティアでフリー/オープンでユニバーサルな「オペレーティングシステム (OS) 」を開発する「コミュニティ」

2.2.1 Debian というオペレーティングシステム

- 様々な用途に使える汎用的な作り
  - ノート PC、デスクトップ PC などの普段利用するコンピュータの OS
  - Linux サーバ
    - 例：web サーバのシェア：`https://w3techs.com/technologies/details/os-linux/all/all`）
  - 組込デバイスのベース OS （多くの CPU で動作する）
- 「Debian」ベースな派生 OS の源流
  - 例：Ubuntu、Kali Linux、Raspberry Pi OS
  - 派生先のディストリビューションと相互に情報交換をして開発している
- 2021 年 10 月の時点で、最新版は Debian 11.1 (コードネーム: bullseye)
  - パッケージ数は 約 59,551 以上を提供
  - 公式にサポートする CPU アーキテクチャは 9
- コードネームはトイ・ストーリーのキャラクター名を採用
- 次のメジャーリリース は Debian 12 (コードネーム: bookworm)
- リリースサイクル
  - メジャーリリース：おおよそ 2 年ごと

    - Debian は time-based freeze を採用
  - 標準サポート期間：3 年
  - LTS（Long Term Support）：標準サポート期間終了後から 2 年
    - サポートするアーキテクチャが amd64、i386、arm 系に絞られる
    - 全パッケージをサポートするわけではなく、主要なパッケージに絞ってサポート

2.2.2 Debian とコミュニティ

- Debian への参加者は世界中にいます
  - 公式の Debian 開発者（Debian Developer）は、60 ヶ国以上に約 1,100 名
  - パッケージメンテナや翻訳などの貢献者も入れるともっと多くの人たちが参加
- Debian 社会契約
  - Debian 開発者 たちが目指すフリーソフトウェアコミュニティの在り方
- Debian フリーソフトウェアガイドライン（DFSG）
  - Debian 社会契約の一部
  - Debian が考えるフリーソフトウェアの定義
  - オープンソースの定義のひな形にもなっている
- Debian Policy
  - `https://www.debian.org/doc/debian-policy/`
  - Debian パッケージの区分、内容、ルール、ファイル配置の方針などの技術的な定義
- Debian はボランティアのみで開発しています

表 1　ディストリビューション開発における人的体制の違い

ディストリ	企業	ボランティア
Fedora	RedHat 支援あり	あり
RHEL	RedHat	なし
CentOS	RedHat 支援あり	あり
Debian	なし	あり
Ubuntu	Canonical	あり
openSUSE	SUSE 支援あり	あり
SLES	SUSE	なし

- DebConf
  - 年に一回、Debian 開発者が集まって開催するカンファレンス
  - 通常はオフラインで集まるが、COVID-19 の影響で 2020 年と 2021 年はオンラインで開催
    - 2019/07/21 - 07/28: Debconf19: CURITIBA - BRAZIL
    - 2020/08/23 - 08/29: Debconf20: Online
    - 2021/08/24 - 08/28: Debconf21: Online
      - `https://debconf21.debconf.org/`
      - 発表のビデオがありますのでぜひご覧ください

2.2.3 小括

- フリー/オープンなオペレーティングシステム (OS) を作成しようとする人たちが集まるボランティアベースのプロジェクト
- 自分たちの考えるフリーという言葉に関する定義、開発目的、パッケージングポリシーを厳格に決めている
- 世界中に 1100 人以上の開発者がおり、他のディストリビューションのベースとして採用されている
- 約 2 年毎にリリースが行われ、多くのパッケージとアーキテクチャをサポートしている
- 上記のような特徴から様々なところで利用されている Linux ディストリビューション

2.3 Debian JP Project と Debian 勉強会

2.3.1 Debian JP Project

- `https://www.debian.or.jp/`
- 日本において Debian を普及させることを目的とした任意団体
- 活動内容
  - Debian の日本語による情報発信
  - ユーザとの情報交換
  - Debian 開発者やパッケージメンテナの育成など

2.3.2 Debian 勉強会

Debian 勉強会の始まり

- 2005 年 1 月開始
- Debian 開発者 上川さんが発起人
- 東京と関西で月に一回コンスタントに開催している Debian 開発者、Debian ユーザによる勉強会
  - 東京エリア Debian 勉強会
    - `https://tokyodebian-team.pages.debian.net/`
  - 関西 Debian 勉強会
    - `https://wiki.debian.org/KansaiDebian`

解決したい問題

- 問題
  - ML と IRC で情報交換していた
  - face-to-face で会う場所がない
  - まとまったドキュメントが出てこない
- Debian 勉強会の提案
  - 定期的に集まる
  - 資料を作成して公開（GPL-2+）

    `https://salsa.debian.org/tokyodebian-team/monthly-report`

最近の勉強会

- Debian 界隈やパッケージング関連の話題など専門の人に話を聞く
- Debian で気になった事柄を調べてレポートする
- 前回の内容（東京 9 月）:
  - 場所: オンライン
  - DebConf 21 のイベント共有会
- 各地のイベントで Debian 普及活動

2.4 Debian 11 bullseye

2.4.1 リリース情報

- Debian 11 (コードネーム：bullseye)
  - 2021 年 8 月 14 日にリリース
  - 最新版は Debian 11.1 （2021 年 10 月 9 日 リリース）
  - リリースノート
    - `https://www.debian.org/releases/bullseye/releasenotes`
  - インストールガイド
    - `https://www.debian.org/releases/bullseye/installmanual`

2.4.2 CPU アーキテクチャ

- 利用できる CPU アーキテクチャは 9 つ
  - amd64、i386
  - arm64、armhf、armel
  - mips64el、mipsel
  - ppc64el
  - s390x
- 前回リリース Debian 10 buster から廃止
  - (big endian の) mips

2.4.3 提供するソフトウェア

表 2　Debian 11 bullseye で提供する主なソフトウェア (1)

Linux kernel	5.10
GNOME	3.38
KDE Plasma	5.20
LXDE	11
LXQt	0.16
MATE	1.24
Xfce	4.16
Cinnamon	4.8.6
Chromium	90.0
OpenSSH	8.4p1
OpenSSL	1.1.1k
GnuPG	2.2.27、1.4.23

表 3　Debian 11 bullseye で提供する主なソフトウェア (2)

Firefox ESR	78
Thunderbird	78
LibreOffice	7.0
GIMP	2.10.22
Inkscape	1.0.2
MariaDB	10.5
PostgreSQL	13
sqlite	3.34.1、2.8.17
Emacs	27.1
Vim	8.2

2.4.4　Debian 11 bullseye の新機能

システム全体の新機能

- コントロールグループ v2 (cgroupv2)
  - bullseye における systemd はデフォルトで cgroupv2 を利用
- systemd の永続的なジャーナル機能はデフォルトで有効
  - ファイルを /var/log/journal/ 配下に保存するよう変更
  - ファイルは adm グループも読み取りできるため注意
- カーネルによる exFAT サポート
  - Debian 10 buster で exFAT を利用するために必要であった exfat-fuse パッケージが不要になった
- 新しい汎用的な open コマンド
  - ファイルの拡張子ごとに適切なコマンドでファイルを開けるようになった

アプリケーションの新機能

- ドライバレスでのスキャンと印刷
  - 印刷では IPP-over-USB プロトコルを扱える新しい ipp-usb パッケージを提供
  - スキャナでは公式の SANE ドライバレスバックエンドである libsane1 を提供
- 新しい Fcitx 5 インプットメソッド
  - Fcitx 5 は中国語、日本語、韓国語やその他の多くの言語のためのインプットメソッド
  - Fcitx 5 では Wayland をサポートし、より優れたアドオンサポートを提供

その他の新機能

- Debian Med チームによる貢献の反映
  - 疫学方面で利用されるツールのソフトウェアをパッケージ化

  - 生命科学と医学の分野における新しいパッケージを追加
  - 既存のパッケージに対する継続的インテグレーションのサポート強化
- man ページの翻訳の改善
  - bullseye リリースが存続する期間中は翻訳の改善を backports アーカイブ経由で提供予定
- 代替 init システムのサポート改善
  - デフォルトは systemd
  - 代替 init システム (System-V 形式の init や OpenRC など) をサポート

アップグレード時の注意

- apt のセキュリティアーカイブの構成 (=URL) が変更されたため、手動で書き換えをお願いします

```
# vi /etc/apt/sources.list

# debian 10 buster
deb http://security.debian.org/debian-security buster/updates main contrib

↓

# debian 11 bullseye
deb https://deb.debian.org/debian-security bullseye-security main contrib
```

アップグレードの最中に 新たに ssh 接続ができない時間が長くなっているため、ssh 接続してアップグレードする場合は以下の手順がおすすめです。

- /etc/apt/sources.list を書き換える
- apt-get update を実行する
- apt-get install openssh-server を実行する
- apt-get upgrade、apt-get dist-upgrade を実行する

動作の変更や制約

- Intel 社製 GPU のデフォルトドライバーが新しい VA-API に変更
- XFS ファイルシステムは barrier/nobarrier オプションをサポートしない
- パスワードのハッシュ化に yescrypt をデフォルトで利用するよう変更
- NSS NIS および NIS+ 利用に新しいパッケージが必要
- unbound での分割された設定ファイルの扱い
- 非推奨になる rsync のパラメータ
- Vim のアドオンの取り扱いの変更
- OpenStack と cgroups v1 について
- OpenStack API ポリシーのファイルについて
- アップグレード中の sendmail にダウンタイムがある
- fuse から fuse3 へアップグレード処理で fuse パッケージを削除する
- GnuPG オプションファイルの変更
- Linux がユーザー名前空間をデフォルトで有効にする
- Linux が bpf() の非特権呼び出しをデフォルトで無効
- bullseye に redmine がない
- bullseye の Exim 4.94 は実質メジャーアップデート
- SCSI デバイスの検出順が決定的ではなくなった

- rdiff-backup は サーバーとクライアントを同時にアップグレードする必要あり
- Intel CPU のマイクロコードでの問題
- libgc1c2 パッケージ関連のアップグレードは 2 回実行が必要
- fail2ban が bsd-mailx の mail コマンドを使ったメール送信ができない

2.4.5 非推奨になった事項

linux カーネル、ブートローダー関連

- Linux カーネルは isdn4linux (i4l) のサポートを提供しない
- aufs-dkms は bullseye から削除
  - overlayfs への移行を推奨
- lilo は bullseye から削除
  - grub2 を利用してください
- armel、armhf 利用者向け
  - 以下ハードウェア向けの kernel をビルドできなくなったため、Debian 11 bullseye ではサポートされない
    - QNAP Turbo Station (TS-xxx)
    - HP Media Vault mv2120

アプリケーション関連

- ネットワーク接続マネージャーである wicd は、アップグレード後には利用できない
- libappindicator ライブラリは提供されない
  - fork である libayatana-appindicator を提供するよう切り替え
- bullseye では chef を提供されない
  - 最もよい移行先は Chef Inc が提供するパッケージへの切り替え
- (mailman2 の) mailman パッケージが廃止され、 mailman3 のみ提供
- python2.7 パッケージは提供するが upstream が EoL のため python3 への移行を推奨
  - bullseye では python2.7 関連のライブラリパッケージの大部分を削除済み
  - Debian 12 で python2.7 の提供をやめるかどうかは議論中

2.4.6 バグレポートのお願い

- 何かおかしい動作や不具合を見つけた場合はバグレポートをお願いします
- バグレポートの例 `https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=903529`
- バグレポートの仕方（レポートは英語で送る必要あり）
  - `https://www.debian.org/Bugs/Reporting.ja.html`
- バグレポートの前にちょっと相談してみたい方は、日本語の Debian JP メーリングリストや、SNS で相談してみてください
  - `https://www.debian.or.jp/community/ml/openml.html`
  - Twitter: @debian_jp

2.5 日本語による Debian の情報

- Debian JP Project `https://www.debian.or.jp`
- 東京エリア Debian 勉強会 `https://tokyodebian-team.pages.debian.net/`
- 関西 Debian 勉強会 `https://wiki.debian.org/KansaiDebianMeeting`
- Twitter `@debian_jp`
- 雑誌 Software Design 技術評論社発行 「Debian Hot Topics」（隔月連載）

3 Web ブラウザでメモの同時共有ができる Etherpad lite の紹介

杉本典充

3.1 はじめに

Debian 勉強会を進行をするにあたり、参加者が同時にメモの編集ができる環境がほしいと思っていました。Web ブラウザを使って複数人が同時にメモの参照や編集ができるツールである「Etherpad lite」を使ってみました。

3.2 Etherpad と Etherpad lite について

Etherpad（`https://etherpad.org/`）とは、Web ブラウザを使って複数人が同時にリアルタイムでメモの参照と編集ができるオープンソースソフトウェア[*1]です。

歴史的には（初代の） Etherpad は 2008 年に etherpad.com（現在は etherpad.org へリダイレクトされる） でプロプラエタリなサービスとして登場し、Java と Scala で実装したものでした。その後 2009 年に etherpad.com は Google に買収され、買収の直後に etherpad のソースコードが Apache License の元で公開されてオープンソースとなりました[*2][*3]。

（初代の）Etherpad がオープンソースで公開された後、2011 年 8 月 22 日に JavaScript と Node.js で Etherpad の機能を再実装した「etherpad-lite」の ver 1.0 が登場しました。etherpad の開発を行う The Etherpad Foundation は今後 etherpad-lite の開発に専念することを決めたようで、（初代の）Etherpad は 2013 年に開発を終了しています。なお、etherpad-lite については The Etherpad Foundation の元で現在でも開発が続けられています。

こうした背景があり（初代の）”Etherpad” の開発は既に終了していることから、現在では単に ”etherpad” というと「etherpad-lite」を指すようになっています[*4]。

3.3 Etherpad lite にたどり着くまでの利用変遷

だいぶ前となりますが Web ブラウザを使って複数人が同時にメモを編集でき無料で使える Web サービスとして titanpad.com[*5] がありました。titanpad.com は便利なサービスのため Debian 勉強会や他の勉強会でも利用されていましたが、2017 年 12 月 31 日をもってサービスを終了しました[*6]。

[*1] `https://github.com/ether/etherpad-lite`

[*2] `https://code.google.com/archive/p/etherpad/`

[*3] リリースアナウンス `https://web.archive.org/web/20091221023828/http://etherpad.com/ep/blog/posts/etherpad-open-source-release`

[*4] `https://etherpad.org/#download` のダウンロード先のリンクは etherpad-lite になっています。

[*5] ソースコードのアーカイブはこちら `https://github.com/titanpad/titanpad`

[*6] `http://blog.titanpad.com/2016/11/shutting-down-titanpad_12.html`

その後、Debian 勉強会ではメモの同時編集ツールとして DebConf などでも使われている gobby [*7] を使っていましたが gobby は使いにくいという声（特に debian を使っていない方）もあり、Web ブラウザで利用できるメモの同時編集ツールを再度探していました。

ツールを探していく中で Web サービスはでなく自分のサーバにインストールするタイプの「etherpad-lite」があることを知り、最近は Debian 勉強会に導入し BoF で利用しています。

3.4 Etherpad lite のインストール

3.4.1 Node.js のインストール

etherpad-lite のインストール手順は以下 URL に記載されています。

`https://github.com/ether/etherpad-lite#installation`

現在の etherpad-lite-1.8.8 では nodejs-10.17.0 以上の環境が必要であり、以下コマンドで nodejs をインストールします。

```
$ curl -sL https://deb.nodesource.com/setup_14.x | sudo -E bash -
# apt install -y nodejs
```

3.4.2 Etherpad lite のダウンロードと配置

etherpad-lite のプログラムを以下 URL からダウンロードして、/opt に配置します。

```
# wget https://github.com/ether/etherpad-lite/archive/1.8.8.zip
# cp 1.8.8.zip /opt/etherpad-lite-1.8.8.zip
# cd /opt
# unzip etherpad-lite-1.8.8.zip
# ln -s etherpad-lite-1.8.8 etherpad-lite
```

etherpad-lite のプログラムを実行するスクリプト ”bin/run.sh” は、実行時にプログラムを置いているディレクトリの配下に ”node_modules” ディレクトリを作成して依存ライブラリを自動でダウンロードします。そのため、プログラムを実行するユーザはこのディレクトリに書き込み権限を持つ必要があります。プログラムを実行するユーザとグループとして ”etherpad” ユーザ及びグループを作成し、etherpad-lite ディレクトリとファイルのオーナーとグループを変更します。

```
# adduser etherpad
# chown -fR etherpad:etherpad /opt/etherpad-lite-1.8.8
# ls -l /opt/etherpad-lite-1.8.8
合計 180
-rw-r--r-- 1 etherpad etherpad    64  2 月 18 22:17 APIKEY.txt
-rw-r--r-- 1 etherpad etherpad 40276  2 月 16 02:47 CHANGELOG.md
-rw-r--r-- 1 etherpad etherpad  8217  2 月 16 02:47 CONTRIBUTING.md
-rw-r--r-- 1 etherpad etherpad  1681  2 月 16 02:47 Dockerfile
-rw-r--r-- 1 etherpad etherpad 11353  2 月 16 02:47 LICENSE
-rw-r--r-- 1 etherpad etherpad   849  2 月 16 02:47 Makefile
-rw-r--r-- 1 etherpad etherpad  8229  2 月 16 02:47 README.md
-rw-r--r-- 1 etherpad etherpad   118  2 月 16 02:47 SECURITY.md
-rw-r--r-- 1 etherpad etherpad    64  2 月 18 22:16 SESSIONKEY.txt
lrwxrwxrwx 1 etherpad etherpad     7  2 月 18 22:10 bin -> src/bin
drwxr-xr-x 6 etherpad etherpad  4096  2 月 16 02:47 doc
drwxr-xr-x 2 etherpad etherpad  4096  2 月 16 02:47 node_modules
-rw-r--r-- 1 etherpad etherpad 21152  2 月 16 02:47 settings.json.docker
-rw-r--r-- 1 etherpad etherpad 19209  2 月 16 02:47 settings.json.template
drwxr-xr-x 9 etherpad etherpad  4096  2 月 18 22:17 src
-rw-r--r-- 1 etherpad etherpad    51  2 月 16 02:47 start.bat
lrwxrwxrwx 1 etherpad etherpad     9  2 月 18 22:10 tests -> src/tests
drwxr-xr-x 2 etherpad etherpad  4096  2 月 18 22:34 var
```

次に設定ファイル ”settings.json” を配置します。設定内容はひとまずデフォルトのコピーとしておきます。

```
# cd /opt/etherpad-lite-1.8.8
# cp -p settings.json.template settings.json
```

etherpad-lite を起動できることを確認してみます。

[*7] `https://github.com/gobby/gobby/wiki`

```
# cd /opt/etherpad-lite-1.8.8
# sudo -u etherpad bin/run.sh
```

以下の URL のアクセスすると etherpad-lite に接続できます。

`http://localhost:9001/p/debianmeeting20210220`

3.4.3 systemctl で起動できるよう service ファイルを配置する

毎回手動で etherpad-lite を起動するのも面倒です。そのため、systemctl から起動できるように設定します。

systemctl で起動できるように以下のような service ファイルを作成します。

```
# vi /etc/systemd/system/etherpad-lite.service

[Unit]
Description=etherpad-lite

[Service]
Type=simple
ExecStart=/opt/etherpad-lite/bin/run.sh
WorkingDirectory=/opt/etherpad-lite
Restart=always
RestartSec=10
User=etherpad
Group=etherpad
Environment=NODE_ENV=production

[Install]
WantedBy=multi-user.target
```

systemctl daemon-reload を実行して追加した service ファイルを systemd に再読み込みさせ、etherpad-lite を起動します。

```
# systemctl daemon-reload
# systemctl start etherpad-lite
# systemctl status etherpad-lite
● etherpad-lite.service - etherpad-lite
   Loaded: loaded (/etc/systemd/system/etherpad-lite.service; disabled; vendor p
   Active: active (running) since Fri 2021-02-19 22:11:05 JST; 4s ago
 Main PID: 26451 (run.sh)
    Tasks: 13 (limit: 1149)
   Memory: 155.0M
   CGroup: /system.slice/etherpad-lite.service
           ├─ 26451 /bin/sh /opt/etherpad-lite/bin/run.sh
           ├─ 26454 /bin/sh src/bin/installDeps.sh
           └─ 26491 npm
```

これで、systemctl で管理しつつ etherpad-lite を起動できるようになりました。

3.4.4 Web サーバへのアクセスを Etherpad lite へリバースプロキシする

これまでの手順で起動できるようになった etherpad-lite の状態では、9001 番ポートで待ち受けします。クライアントとなる Web ブラウザから etherpad-lite サーバの 9001 番ポートへ接続する場合に、利用するネットワークやファイアウォール設定などの環境によって接続できない場合があります。

以下 URL を参考に apache2.4 の mod_proxy を使ってリバースプロキシの設定を行い、クライアントからは 80 番ポート（http://）、または 443 番ポート（https://）で接続できるようにしてみました。

`https://github.com/ether/etherpad-lite/wiki/How-to-put-Etherpad-Lite-behind-a-reverse-Proxy`

まずは apache2.4 をインストールします。

```
# apt update
# apt install apache2
```

apache2.4 の mod_proxy などの利用するモジュールを有効にします。

```
# a2enmod proxy
# a2enmod proxy_http
# a2enmod proxy_balancer
# a2enmod lbmethod_byrequests
# a2enmod rewrite
# a2enmod proxy_wstunnel
```

etherpad-lite の 9001 番ポートへリバースプロキシする設定ファイル ”etherpad-lite.conf” を作成します。

```
# vi /etc/apache2/conf-available/etherpad-lite.conf

    ProxyVia On
    ProxyRequests Off
    ProxyPreserveHost on

    <Location /etherpad/>
        ProxyPass http://localhost:9001/ retry=0 timeout=30
        ProxyPassReverse http://localhost:9001/
    </Location>
    <Location /etherpad/socket.io>
        # This is needed to handle the websocket transport through the proxy, since
        # etherpad does not use a specific sub-folder, such as /ws/ to handle this kind of traffic.
        # Taken from https://github.com/ether/etherpad-lite/issues/2318#issuecomment-63548542
        # Thanks to beaugunderson for the semantics
        RewriteEngine On
        RewriteCond %{QUERY_STRING} transport=websocket    [NC]
        RewriteRule /(.*) ws://localhost:9001/socket.io/$1 [P,L]
        ProxyPass http://localhost:9001/socket.io retry=0 timeout=30
        ProxyPassReverse http://localhost:9001/socket.io
    </Location>

    <Proxy *>
      Options FollowSymLinks MultiViews
      AllowOverride All
      Order allow,deny
      allow from all
    </Proxy>
```

a2enconf コマンドを実行して ”etherpad-lite.conf” の設定を apache2.4 へ組み込み、apache2.4 を reload します。

```
# a2enconf etherpad-lite
Enabling conf etherpad-lite.
To activate the new configuration, you need to run:
  systemctl reload apache2

# systemctl reload apache2
```

これで以下の URL のように クライアントから etherpad-lite サーバへ 80 番ポートに接続して使えるようになりました。

`http://www.pcdennokan.wjg.jp/etherpad/p/debianmeeting20210220`

3.4.5 etherpad-lite の設定

デフォルトの設定ファイル ”settings.json” はクライアントからのアクセスに認証を必要としない設定になっています。最低限の認証を行えるように設定する例を紹介します。以下の設定は 1 つのアカウントを全ユーザで共用する設定になっています。

```
--- settings.json.template      2021-02-16 02:47:33.000000000 +0900
+++ settings.json       2021-02-18 22:15:43.144967658 +0900
@@ -315,7 +315,7 @@
     *
     * Note: "/admin" always requires authentication.
     */
-  "requireAuthentication": false,
+  "requireAuthentication": true,

   /*
    * Require authorization by a module, or a user with is_admin set, see below.
@@ -428,22 +428,26 @@
     *          follow the section "secure your installation" in README.md
     */

-  /*
   "users": {
     "admin": {
       // 1) "password" can be replaced with "hash" if you install ep_hash_auth
       // 2) please note that if password is null, the user will not be created
-      "password": "changeme1",
+      "password": "adminpassword",
       "is_admin": true
     },
-    "user": {
+    //"user": {
       // 1) "password" can be replaced with "hash" if you install ep_hash_auth
       // 2) please note that if password is null, the user will not be created
-      "password": "changeme1",
+    //  "password": "changeme1",
+    //  "is_admin": false
+    //}
+    "myuser": {
+       // 1) "password" can be replaced with "hash" if you install ep_hash_auth
+       // 2) please note that if password is null, the user will not be created
+      "password": "myuserpassword",
       "is_admin": false
     }
   },
-  */
```

そのほか、https://static.etherpad.org/index.html で多数の etherpad-lite 用のプラグインがありますので実用するにあたり自分の利用方法にあるものを探してみるとよいと思います。

3.5 おわりに

etherpad-lite の歴史とインストール手順を説明し、etherpad-lite サーバを作ってみました。最近はクラウドサービスの Office 系アプリケーションでも同時編集が可能になっているサービスもありますが、オープンソースなアプリケーションを使って複数人でメモの同時編集する環境を構築できますのでオープンソースライフを楽しみつつ実用していただけるのでないかと思います。

3.6 参考情報

- etherpad.com
  https://etherpad.org/
- EtherPad Open Source Release
  https://web.archive.org/web/20091221023828/http://etherpad.com/ep/blog/posts/etherpad-open-source
- How to put Etherpad Lite behind a reverse Proxy
  https://github.com/ether/etherpad-lite/wiki/How-to-put-Etherpad-Lite-behind-a-reverse-Proxy

4 Gnuk のエミュレーションモードを試してみよう

野首貴嗣

4.1 Gnuk とは

Gnuk は、OpenPGP card 2.0 に準拠した USB 暗号化トークンの実装 `https://git.gniibe.org/cgit/gnuk/gnuk.git/` の名称です。対応したマイコン (ハードウェア) と組み合わせると、OpenPGP の暗号化・復号化処理をそのハードウェア上で行うことができます。鍵の保持もハードウェア上で行うため、物理的なセキュリティをもたらします。既にファームウェアが入った製品としては、FST-01 `https://seeeddoc.github.io/FST-01/` があります。

一方で、一度ハードウェア上に置かれた鍵を複製したり、PC にダウンロードしたりすることはできません。一度ハードウェア上で生成した鍵は (factory reset を有効化したファームウェアを入れていない限り) 作りなおしができません。

しかし、Gnuk 1.2.8 から USB/IP を用いたデバイスエミュレーションがサポートされたため、それを使った練習が容易になりました。

4.2 前準備

以下のソフトウェア (Debian パッケージ) が必要です。

- scdaemon (GnuPG と USB/Gnuk の通信のために必要)
- usbip (Gnuk プロセスの動作に必要)
- python-cffi, python3-pytest (テストの動作に必要)
- gnupg2 (2.2.27 以降が必要)

4.3 Gnuk の clone, ビルド

```
$ git clone -b STABLE-BRANCH-1-2 git://git.gniibe.org/gnuk/gnuk.git
$ cd gnuk
$ git submodule update --init
$ cd src
$ ./configure --target=GNU_LINUX --enable-factory-reset
$ make
```

*8

*8 セミナーのときはコミット 07be37f45e76f50c84bed588e1933a0b8b149c91 を使いました。

4.4 プロセスの起動

Gnuk は FSIJ が取得した USB Vendor/Product ID を用います。https://www.fsij.org/pages/misc/usb-id.html

```
$ cd build
$ ./gnuk --vidpid=234b:0000
Gnuk (emulation with USBIP), a GnuPG USB Token implementation
Copyright (C) 2021 Free Software Initiative of Japan
This is free software under GPLv3+.
USBIP thread started.
You can use this by attaching following commands:
    # modprobe vhci_hcd
    # usbip attach -r 127.0.0.1 -b 1-1
User interaction thread for AckBtn started.
```

4.5 USB デバイスとして認識

先ほど起動した gnuk プロセスはフォアグラウンドで動作しているので、別のターミナルを用意して以下を実行してください。

```
  $ sudo modprobe vhci_hcd
  $ sudo usbip attach -r 127.0.0.1 -b 1-1
  $ lsusb | grep 234b
  Bus 005 Device 002: ID 234b:0000
  $ lsusb -t
  /:  Bus 03.Port 1: Dev 1, Class=root_hub, Driver=vhci_hcd/8p, 480M
      |__ Port 1: Dev 3, If 0, Class=Chip/SmartCard, Driver=, 12M
  $ sudo lsusb | tail
  [1234859.035877] usb 3-1: New USB device found, idVendor=234b, idProduct=0000, bcdDevice= 0.00
  [1234859.035879] usb 3-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
  [1234859.035880] usb 3-1: Product: Gnuk Token
  [1234859.035880] usb 3-1: Manufacturer: Free Software Initiative of Japan
  [1234859.035881] usb 3-1: SerialNumber: FSIJ-1.2.17-EMULATED
```

4.6 test suite の実行

正しくデバイスが認識できていたら、test suite が動作します。

```
$ cd gnuk/tests
$ pytest-3 -x
========================= test session starts =========================
platform linux -- Python 3.7.3, pytest-3.10.1, py-1.7.0, pluggy-0.8.0
rootdir: /home/knok/gnuk/tests, inifile:
collected 416 items

test_000_empty_card.py ..............................              [  7%]
test_001_personalize_card.py ..................................... [ 15%]
.................                                                 [ 19%]
test_002_personalize_reset.py ............                        [ 22%]
(略)
test_025_kdf_none.py ...                                          [100%]

====================== 416 passed in 11.43 seconds ======================
```

4.7 gpg コマンドの動作

コマンド gpg --card-status を実行することで認識されたデバイス情報が出力されます。

```
$ gpg --card-status
Reader ...........: 234B:0000:FSIJ-1.2.17-EMULATED:0
Application ID ...: D2760001240102000FFFEF1420A7A0000
Version ..........: 2.0
Manufacturer .....: unmanaged S/N range
Serial number ....: F1420A7A
Name of cardholder: [未設定]
Language prefs ...: [未設定]
:
```

gnuk デフォルトの PIN は以下の通りです (https://git.gniibe.org/cgit/gnuk/gnuk.git/tree/README より)。

- 管理者: 12345678
- ユーザー: 123456

4.8 鍵生成手順

まず初期化を行います。カード編集モードにて、管理者になり factory reset を実行します。

```
$ gpg --edit-card
gpg/card> admin
gpg/card> factory-reset
```

次に属性の変更を行います。この際に PIN を要求されます。

```
gpg/card> key-attr
```

鍵の新規作成を行います。

```
gpg/card> generate
```

この際の鍵生成フローは、一般的な GnuPG でのフローと同様です。必要な情報を入力してください。また、失効証明書がローカルファイルに生成されます。保存しておきましょう。

最後にカード操作を終了します。

```
gpg/card> quit
```

これで、本来デバイス上に保存されるべき gnuk の情報が `$HOME/.gnuk-flash-image` ファイルとして保存されます。ミスをしてやり直す場合はこのファイルを消せばよいでしょう。

4.9 暗号化・復号化・署名

あとは基本的に通常の GnuPG の利用方法と同様で、gnuk が扱えます。USB/IP デバイスにアクセスするためには、スペシャルファイル `/dev/bus/usb/$bus/$dev` へのパーミッションが必要です。通常、pam-systemd によってログインしたユーザーの権限が付与されます (ルール: `/lib/udev/rules.d/60-scdaemon.rules`)。

秘密鍵へのアクセスは復号化時 (–decrypt) のみ必要です。暗号化 (–encrypt) や署名 (–sign) には必要ありません。

また、gpg-agent を介して ssh もできます。

4.10 参考リンク

- `https://www.fsij.org/category/gnuk.html`
- `https://git.gniibe.org/cgit/gnuk/gnuk.git/`
- `http://www.gniibe.org/category/fst-01.html`
- `https://en.wikipedia.org/wiki/OpenPGP_card`
- `https://wiki.debian.org/GnuPG`
- `https://wiki.debian.org/Smartcards`

5 Debian で Android アプリ開発と Kotlin

杉本典充

5.1 はじめに

最近 Android アプリを開発しており、Debian で Android アプリを開発するにはどのような準備をすればよいのか、最近の Android アプリの開発で使われるプログラム言語「Kotlin」の Debian における対応状況について調べてみました。

5.2 Debian で Android アプリを開発する

5.2.1 Android 情報のドキュメントの場所

Android アプリの開発者向けのドキュメントは `https://developer.android.com/?hl=ja` にあります。

この Web ページでは Android アプリの開発に利用する Android Studio のダウンロードとインストール方法、最近の Android 情報や開発者ガイド、ガイドライン、チュートリアルなどの情報を公開しています。

5.2.2 Android Studio のインストール

Linux 環境に Android Studio をインストールする手順は、開発者向けドキュメントの `https://developer.android.com/studio/install?hl=ja#linux` に記載があり、Debian Wiki[*9] にも情報があります。インストール手順には tarball をダウンロードして任意のディレクトリに展開する方法と、snap 環境にインストールする方法があります。

ここでは、`https://developer.android.com/studio#downloads` から利用規約を読み承諾してダウンロードし[*10]、展開して Android Studio を起動するコマンドを示します。

```
$ tar xf android-studio-2020.3.1.26-linux.tar.gz
$ cd android-studio/bin
$ bash studio.sh
```

studio.sh を起動すると、Android Studio の画面が表示され、実行に必要な追加のソフトウェアを大量にダウンロードします。ダウンロードとインストールが完了すると Android Studio を起動できます。

なお、Android Studio は IntelliJ IDEA ベースで作られています。そのため Android Studio を実行するには Java 環境が必要であり tarball の中に OpenJDK 11 が含まれています（Debian が提供する openjdk-11-jre パッケージをインストールせずに実行できます）。

[*9] `https://wiki.debian.org/AndroidStudio`

[*10] ダウンロードする tarball は 935 MiB ありますので、定額で高速にインターネット接続できる環境でセットアップすることをおすすめします。

5.2.3 Android エミュレータを実行するために KVM を設定する

Android アプリをデバッグするために Android のエミュレータ (=仮想マシン) が提供されています。Android のエミュレータのイメージファイルは qemu で動作する仮想マシンになっているため、PC 上で KVM を使えるように以下のパッケージをインストールします。

```
# apt-get update
# apt-get install qemu-kvm
```

5.2.4 Android アプリを実機でデバッグ実行できるように設定する

Android Studio ではビルドした Android アプリをデバッグ実行することができます。デバッグ実行すると、設定したエミュレータ上または Android スマートフォンの実機上でアプリを動かすことができます。Android アプリを Android スマートフォンの実機上で実行する手順は、`https://developer.android.com/studio/run/device?hl=ja` に説明があります。

Debian では PC と Android スマートフォンの実機を USB 接続したことを検知するためのパッケージを提供しており、android-sdk-platform-tools-common パッケージをインストールすると udev の設定ファイルである "/lib/udev/rules.d/51-android.rules" をインストールします。パッケージのインストール後は、udev の設定ファイルを再読み込みします。なお、USB 接続する Android スマートフォンが Pixel シリーズの場合は、Debian 11 bullseye が提供する linux kernel にドライバが含まれているため、別途ドライバのインストールは不要です。

```
# apt-get install android-sdk-platform-tools-common
# systemctl reload udev
```

次に Android スマートフォンを設定します。Android スマートフォンの設定画面を開いて開発者向けオプションを有効にし[*11]、開発者向けオプションの設定画面にある「USB デバッグ」を有効にします。この状態の Android スマートフォンを Android Studio を実行している PC に USB 接続すると、Android スマートフォンの画面にフィンガープリントを確認する画面が出てきますので「はい」を選択します。少し待つと Android Studio の上部にある Android アプリの実行対象デバイスを表示するメニューに Android スマートフォン実機の型番が表示されます。ここまで準備ができると Android Studio でアプリをデバッグ実行すれば 実機で Android アプリを動かすことができます。

5.3 Debian での Kotlin

5.3.1 ドキュメントの情報

Kotlin の upstream の Web サイトは `https://kotlinlang.org/` にあり、ソースコードは Apache License 2.0 の下でソースコードを公開しています[*12]。2021 年 12 月 18 日現在では、バージョン 1.6.10 を提供しています。

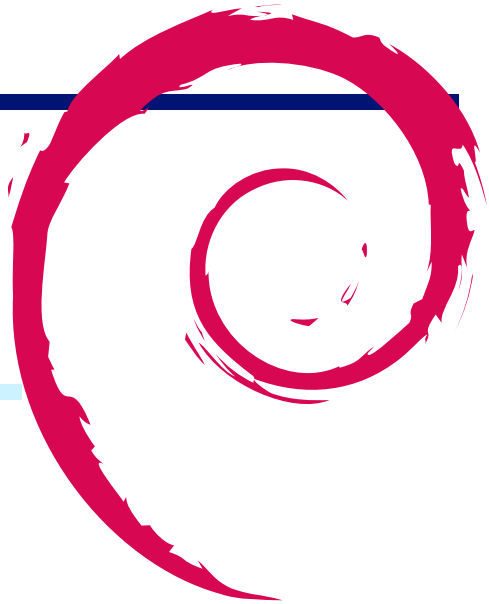
Debian における Kotlin の情報は、`https://wiki.debian.org/Kotlin` に記載があります。Debian パッケージでは unstable のみですが kotlin パッケージを提供しており（kotlin-1.3.31）、`https://salsa.debian.org/java-team/kotlin` で開発作業を進めています。upstream のバージョンと比べると Debian パッケージのものはだいぶ遅れている状況ですが、セルフビルドを目標に作業を進めています。

5.4 おわりに

Debian で Android アプリを開発するために Android Studio を設定してみました。Kotlin を Debian で使うにはまだ発展途上の状況ですが、ぜひ新しいバージョンが利用できるようになってほしいです。

[*11] 「ビルド番号」の項目を 7 回連続でタップすると開発者向けオプションが有効になります。 `https://developer.android.com/studio/debug/dev-options?hl=ja`

[*12] `https://github.com/JetBrains/kotlin`



6 Debian での Node

上川純一

6.1 node.js の最近

Javascript はウェブのプログラミング言語として広く使われています。node.js は v8 という Javascript エンジンで動作する Javascript 環境です [1]。v8 は Chrome の javascript エンジンとして広く使われていおり、同じ環境をブラウザ以外でも使えるというのが便利なポイントです。

Javascript は基本的にはシングルスレッドで動くのでイベントベースでプログラミングすることになり、はじめにそこをうまく扱った標準ライブラリを整備したあたりが普及の決め手だったのでしょう。たとえばほぼすべてのファイル API がコールバック形式になっています。

Javascript の言語は ECMAScript という名称で仕様化されています。その仕様は ES6 で大きく進化しました。ES6（ES2015）以降は仕様の呼称も変わり、年が入るようになりました。そして毎年仕様がリリースされています [5]。大きな変化は Promise と async/await という言語の進化により大きく書き方が変わり、それにともなって順次コアの API も書き換わっていっている過程のように見えます。

ES6 以前の Javascript だと callback を使うのでなにかするたびに function が一段かまされインデントが深くなっていきます。

```
var fs = require('fs');

fs.readFile('./cat.js', {encoding: 'utf-8'}, function(err, data) {
    if (err) throw err;
    console.log(data);
})
```

ES6 以降の Javascript だと async await が使えるようになっていて

```
const fs  = require('fs');

const main = async () => {
    const buffer = await fs.promises.readFile('./cates2015.js',
                                              {encoding: 'utf-8'});
    console.log(buffer);
}

main().catch((e)=>{
    console.error(e);
    process.exit(1);
});
```

あと javascript で必ずはまる変数スコープとか this の扱いとかを比較的意識しなくて良くなるのが良いと思います。

ここ数年は Node のバージョンは偶数バージョンが LTS の安定版となっていて、一年に一回リリースされています [2]。ES2015 から ES2020 の各機能のサポート状況については Node.js ES2015 Support [6] のページがよくまとまっているようです。

nodejs リリース	リリース日	end of life	上川にとって特筆する言語機能
v10	2018-04-24	2021-04-30	v8 6.6, fs/promises の試験導入
v12	2019-04-23	2022-04-30	v8 7.4
v14	2020-04-21	2023-04-30	v8 8.1, ES Modules の標準サポート、
v15	2020-10-20	2021-06-01	
v16	2021-04-??		

二年に一回安定版リリースをしている Debian としては一年に一回大きなアップデートをしている Node.js の最新版をおいかけるのは結構大変でしょう。

6.2 Debian ユーザとしての Node でプログラムを書いてみる

6.2.1 Debian 公式パッケージのインストール

Debian 公式のパッケージとして nodejs パッケージが提供されています。それだけでは実行できる環境が揃うだけです。Nodejs の強みは npm というパッケージリポジトリです。Debian 公式リポジトリでは npm という Debian パッケージに npm パッケージ管理システムのクライアント npm コマンドが入っています。[*13]

```
$ apt-get install -y npm nodejs
```

しかし、いくら Debian のパッケージが多いといってもすべてのパッケージを提供するのは現実的ではないのと他の nodejs を利用している一般開発者は npm 管理のパッケージを利用しているためそことの相互運用のことも考えると悩ましいですが Debian パッケージより node.js 公式パッケージを使うことになりそうです。

npm install して依存関係をインストールすると Debian パッケージを使うわけではなく、`http://npmjs.com` からインストールすることになります。

Debian パッケージとして依存関係をインストールなら該当するパッケージを探すことになります。インストールされたライブラリは /usr/lib/nodejs 以下にインストールされています。

手元の Socket.io テストアプリの例を見てみましょう。

```
{
  "name": "wssrtc",
  "description": "Node.js socket.io app running on Cloud Run for webrtc",
  "version": "v0.0.1",
  "private": true,
  "dependencies": {
    "cli": "",
    "ejs": "",
    "express": "",
    "socket.io": ""
  },
}
```

Debian パッケージは node-express などだと思われるんですが、cli, socket.io にそれぞれ該当するパッケージが見つかりませんでした。昔はあったような気がするのに。

npm install を使うと `npmjs.com` からダウンロードしてきて `./node_modules` 以下にインストールします。しかしなんか Node.js のバージョン 10 がサポートされませんと警告が出たり微妙な雰囲気です。

```
$ npm install
npm WARN npm npm does not support Node.js v10.21.0
npm WARN npm You should probably upgrade to a newer version of node as we
npm WARN npm can't make any promises that npm will work with this version.
npm WARN npm Supported releases of Node.js are the latest release of 4, 6, 7, 8, 9.
npm WARN npm You can find the latest version at https://nodejs.org/
npm WARN deprecated debug@4.1.1: Debug versions >=3.2.0 <3.2.7 || >=4 <4.3.1 have a low-severity ReDos regression when used in a Node.js envi
npm WARN ws@7.4.2 requires a peer of bufferutil@^4.0.1 but none is installed. You must install peer dependencies yourself.
npm WARN ws@7.4.2 requires a peer of utf-8-validate@^5.0.2 but none is installed. You must install peer dependencies yourself.

added 96 packages from 107 contributors in 3.568s
```

[*13] yarn というのもあるよと教えてもらいましたが今後の課題

6.2.2 Node.js 公式パッケージのインストール

Node.js から提供されているパッケージのインストール手段としては公式に配布されているバイナリの最新の LTS 版を利用するのがよいでしょう [3]。ダウンロード方法について書いてあるページに従うとバイナリインストールから始まるのですがそこからパッケージ版へのリンクがあり、Debian 等は別のページにあり、Debian パッケージのインストールに到達するには 3 ページ遷移する必要があります。Debian パッケージのインストール手順に従うと、サイトからダウンロードしたシェルスクリプトを root 権限で実行することになります。

```
# curl -sL https://deb.nodesource.com/setup_14.x | bash -
# apt-get install -y nodejs
```

スクリプトでやっていることは基本的には次のような一連のコマンドでした。

```
$ curl -sSL https://deb.nodesource.com/gpgkey/nodesource.gpg.key | sudo apt-key add -  # 鍵の追加
$ VERSION=node_14.x
$ DISTRO="$(lsb_release -s -c)"
$ echo "deb https://deb.nodesource.com/$VERSION $DISTRO main" | sudo tee /etc/apt/sources.list.d/nodesource.list  # パッケージ情報の追加
$ echo "deb-src https://deb.nodesource.com/$VERSION $DISTRO main" | sudo tee -a /etc/apt/sources.list.d/nodesource.list  # ソースパッケージ情報
$ sudo apt update && sudo apt install nodejs  # インストール
```

ここまでの作業を行うと npm コマンドや nodejs コマンドが利用できるようになっています。Debian 公式パッケージでは npm コマンドがわかれていますが、nodejs 提供のパッケージでは npm コマンドも nodejs パッケージに含まれています。

Debian では以前 node コマンドがすでに存在していたのでポリシーに基づいて node という名前がつかえず、nodejs という名前を利用しています、stretch までは node-legacy パッケージをインストールしてシンボリックリンクを用意していました[*14]。現在は node コマンドが利用できるようになっており、nodejs もシンボリックリンクとして提供されているようです。

node.js の API のドキュメント [4] を参照しながらプログラムを書くことになるでしょう。

6.3 Docker での Debian ユーザとしての Node の利用

Docker を利用する場合は、Node.js がインストールされたインスタンスが Dockerhub にあるのでそれをベースに使うのがよいのではないでしょうか。`https://hub.docker.com/_/node/` にあり、`FROM node` と何も指定しないと `node:current` が使われて、それは debian stretch ベースのイメージのようです。buster が使いたい場合は `FROM node:buster` を指定するのがよいでしょう。

手元の Dockerfile では次のように書いていました。

```
FROM node:14

COPY . .
RUN npm install
```

6.4 Debian Developer としての Node.js パッケージ

Debian パッケージを Nodejs の `package.json` から生成すること自体は簡単です。npm2deb をつかえばよいでしょう。おそらく困難は Debian ポリシーに従うと Debian パッケージとして追加するには Debian パッケージだけで依存関係を解決する必要があります。そのため依存関係を全部追加し、ライセンスなどのチェックを行うことにあると思われます。あとは npm 管理じゃない環境にスナップショットを維持することになるので面倒が増えます。

[*14] `https://lists.debian.org/debian-devel-announce/2012/07/msg00002.html` に名前衝突についての CTTE Resolution があります

6.4.1 Node.js バージョン

Debian 公式のパッケージ「nodejs」のバージョンを２０２０年１２月２５日時点で確認したところ、最新の LTS 版を experimental に入れるという運用のようです。保守的ですが２年間安定版として利用されることを見越すとこれが現実的でしょう。最新の Node.js の機能に依存しているソフトウェアは扱いが難しそうです。

Debian release	nodejs バージョン
stretch	4.8.2
stretch-backports	8.11.1
buster	10.21.0
bullseye	12.19.0
sid	12.19.0
experimental	14.13.0

6.4.2 Node.js パッケージポリシー

Node.js パッケージのポリシーが書いてある Wiki ページがありました [7]。そこにはこんなことが書いてあります：

- バイナリーとソースパッケージは node-foo という名前にする
- MUST: nodejs に Build-Depends を指定すること
- /usr/lib/nodejs/ か /usr/lib/nodejs/foo/にインストールすること。場所は複数のファイルがあるのか単一ファイルなのかに依存する。
- package.json を /usr/lib/nodejs/foo/package.json にいれておくこと
- ウェブブラウザから利用可能であれば libjs-foo バイナリパッケージを作成する。 Javascript/Policy を参照。
- require('foo') が動くように確認する自動テストを含めること。
- c++ アドオンは nodejs-abi-*process.versions.modules* に依存すること。

`node-`ではじまるバイナリパッケージは 1400 くらいあるようです。ただ npm にあがっているパッケージの数は 140 万くらいのようで毎日平均 1000 パッケージが追加されているようです。

6.4.3 Debian Javascript policy

歴史的には Debian にある既存の Javascript のコードの多くはブラウザで利用できるようになっているもので、それについては Debian Javascript Policy で統一しようとしているようです [8]。

javascript-common パッケージを Recommend してパッケージ名は `libjs-` で始めるようにするというものですが Buster では 223 パッケージあるようです。

6.4.4 npm2deb を使ってみる

ためしに npm2deb を使ってみました。公開されている npm パッケージをとってきてテンプレートを作成してくれるというものでした。しかし実際に使おうとするとものによりますが依存関係を全部確認して Debian に入れるのが大変そうです。

```
$ npm2deb create simple-peer

This is not a crystal ball, so please take a look at auto-generated files.

You may want fix first these issues:

simple-peer/node-simple-peer/debian/changelog: -- FIX_ME debian author  Sat, 16 Jan 2021 10:39:59 +0900
simple-peer/node-simple-peer/debian/control:Uploaders: FIX_ME debian author
simple-peer/node-simple-peer/debian/control:Description: FIX_ME write the Debian package description
simple-peer/node-simple-peer/debian/copyright:Copyright: 2021, FIX_ME debian author
simple-peer/node-simple-peer_itp.mail:Subject: ITP: node-simple-peer -- FIX_ME write the Debian package description
simple-peer/node-simple-peer_itp.mail:Owner: FIX_ME debian author
simple-peer/node-simple-peer_itp.mail:  Description     : FIX_ME write the Debian package description
simple-peer/node-simple-peer_itp.mail: FIX_ME: This ITP report is not ready for submission, until you are
simple-peer/node-simple-peer_itp.mail:FIX_ME: Explain why this package is suitable for adding to Debian. Is
simple-peer/node-simple-peer_itp.mail:FIX_ME: Explain how you intend to consistently maintain this package

Use uscan to get orig source files. Fix debian/watch and then run
$ uscan --download-current-version


*** Warning ***
Using fakeupstream to download npm dist tarballs, because upstream
git repo is missing tags. Its better to ask upstream to tag their releases
instead of using npm dist tarballs as dist tarballs may contain pre built files
and may not include tests.

Warnings occurred:
 [error]   get-browser-rtc: dependency node-get-browser-rtc not in debian
 [error]   queue-microtask: dependency node-queue-microtask not in debian
```

参考文献

[1] “Node.js” https://nodejs.org/

[2] “Releases—Node.js”, https://nodejs.org/ja/about/releases/

[3] NodeSource Node.js Binary Distributions https://github.com/nodesource/distributions/blob/master/README.md#debinstall

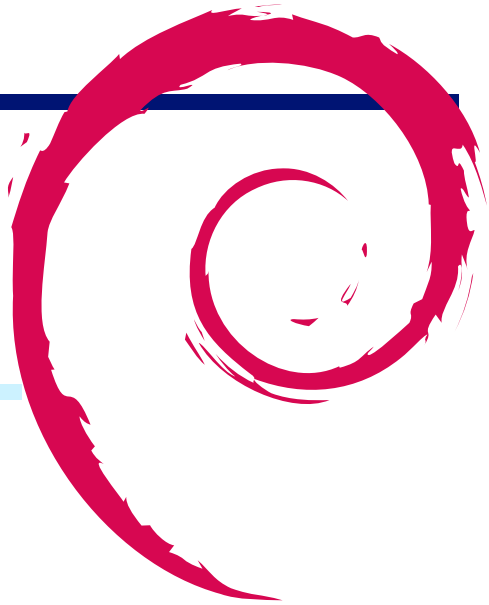
[4] “Node.js v15.4.0 Documentation” https://nodejs.org/api/

[5] “ECMAScript 2015 (ES6) とそれ以降のバージョン” https://nodejs.org/ja/docs/es6/

[6] “Node.js ES2015 Support” https://node.green/

[7] “Node.js modules policy” https://wiki.debian.org/Javascript/Nodejs/Manual

[8] “Javascript Policy” https://wiki.debian.org/Javascript/Policy



7 Debian における Go

上川純一

Go 言語を Debian で使ってみましょう。

まず入門するにはオンラインでのチュートリアルが充実しているので最初はローカルにインストールすることすら必要ないでしょう [4] [5]。しかし次第に本格的に使い始めようとすると手元にインストールしたくなるはずです。

7.1 Debian ユーザとして Go でプログラムを書いてみる

Go のインストール手順 [3] に従って公式パッケージをダウンロードして Tar を展開すると最新のバイナリがインストールできます。しかしあとからアンインストールするときとかに面倒ですしアップデートも自動ではありません。ここは Debian パッケージが使いたいなぁと思います。

Debian の公式パッケージは Buster では 2021 年 4 月現在 1.11 ですが、Buster backports にはもう少し新しい 1.14 があります。まず /etc/apt/sources.list に buster-backports を追加しましょう。[*15]

```
deb https://deb.debian.org/debian buster-backports main
```

そしておもむろにインストール

```
$ sudo apt update
$ sudo apt install -t buster-backports golang
$ sudo apt install -t buster-backports golang-mode  # もし Emacs ユーザなら
```

Go の追加のパッケージは Debian パッケージではなく Go の仕組みでインストールしましょう。Debian パッケージはビルド時の依存関係を満たすためだけにパッケージされているようです。理由は 2 つあり、

- ユーザが go get を使うことで go build などに必要な環境をユーザが設定することになる。
- ユーザ権限では書き込めないのと、.git ディレクトリが存在しないため通常の go get -u でのアップデートなどが不可能なためです。

以上の理由により Debian パッケージでは/usr/share/gocode/src/にソースがインストールされるのですが特に指定しないと Go のコンパイル時に利用してくれません。

たとえば sqlite3 を利用したいなと思ったら go get コマンドを利用してモジュールを ~/go 以下にインストールします。 [*16]

```
$ go get github.com/mattn/go-sqlite3
```

すると以下のようなプログラム (db.go) がコンパイルできるようになります。

[*15] 現在フリーズ中の次期安定版予定の bullseye だと 1.15 になっているようです。さらには sid には 1.16 パッケージがあるようです。

[*16] Debian パッケージだと golang-github-mattn-go-sqlite3-dev があるのだがそれはパッケージのビルド用にしか使えないようだ。

```
import (
        "database/sql"
        _ "github.com/mattn/go-sqlite3"
        "log"
)

func main() {
        db, err := sql.Open("sqlite3", "./ac.db")
        if err != nil {
                log.Fatal(err)
        }
        defer db.Close()
}
```

コンパイルは go build で実行します。ただビルドするだけじゃなくてそのまま実行までしたければ go run で実行できます。

```
$ go build db.go
$ ./db

$ go run db.go
```

7.1.1 Golang の Module システム

Golang の標準のモジュール管理システムは GOPATH が従来利用されていたのですが、1.11 から Go Modules[7] が提供され、1.14 からデフォルトに切り替わりました。

go.mod ファイルにモジュールメタデータが管理されています。[*17]。ホームディレクトリ以下の go/pkg 以下にモジュールがインストールされるようです。[*18]

Go Module を使う Go のコードを新規に開発するには、モジュールのディレクトリで go mod init で go.mod を作成します。

```
$ go mod init example.com/your/packagename
$ cat go.mod
module example.com/your/packagename

go 1.15

$ go test  # カレントディレクトリのモジュールのテストを走らせる
```

サブディレクトリのパッケージのときに import 文で example.com/your/packagename/subdir のように指定するとローカルのサブディレクトリを利用してくれます。

それ以外の名前を指定したら go build 時にダウンロードしてきて、go.mod に情報を追記してくれるようです。

7.1.2 Docker ユーザとしての Go

Docker で Debian で Go を使う場合には Dockerhub の Golang の Docker イメージ [8] を使うのが効率よくお手軽でしょう。

golang という名前でいくつかのバージョンが提供されておりデフォルトは Buster イメージのようです。golang:latest は２０２１年４月現在では golang:1.16.3-buster になっており docker pull や Dockerfile に golang と指定すると Debian の上に Golang をインストールしたイメージが入るようです。

たとえばこんな Dockerfile を書いて CI に利用しています。

```
FROM golang
COPY . .
RUN make
```

7.2 Debian 開発者としての Go パッケージ

Debian に Go でできたソフトウェアをパッケージとしていれたいときには追加で作業が必要になります。いちユーザとして利用していた場合との差異としてはポリシー準拠のために Debian で依存関係すべて含めてビルドできる

[*17] 以前はディレクトリ構成と GOPATH 環境変数で指定されたディレクトリで管理していました
[*18] 以前はホームディレクトリ以下の go/src

ようにする必要があるということです。つまり依存関係にあったものを go get でダウンロードしてくるのではなく Debian パッケージを利用します。

Debian にある Go パッケージ作成は pkg-go チームが中心となって行っているようです [6]、全てではないですが大半のパッケージはこのチームがやっているようです。

dh-golang を使って Debian パッケージのビルドは行うようです。dh-make-golang コマンドを使うと Debian パッケージの雛形を作成してくれるようです。

```
$ dh-make-golang make github.com/ncabatoff/fakescraper
2021/04/10 01:46:49 Starting "dh-make-golang v0.4.0 linux/amd64"
2021/04/10 01:46:49 Downloading "github.com/ncabatoff/fakescraper/..."
2021/04/10 01:46:50 Determining upstream version number
2021/04/10 01:46:50 Package version is "0.0~git20201102.4b37ba6"
2021/04/10 01:46:50 Determining dependencies
2021/04/10 01:46:50 Generating temp tarball as "/tmp/dh-make-golang230078837"
2021/04/10 01:46:50 Moving tempfile to "golang-github-ncabatoff-fakescraper_0.0~git20201102.4b37ba6.orig.tar.xz"
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:         git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:         git branch -m <name>
2021/04/10 01:46:50 Adding remote "origin" with URL "git@salsa.debian.org:go-team/packages/golang-github-ncabatoff-fakescraper.git"
2021/04/10 01:46:50 Adding remote "github" with URL "https://github.com/ncabatoff/fakescraper"
2021/04/10 01:46:50 Running "git fetch github"
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 14 (delta 0), reused 0 (delta 0), pack-reused 13
Unpacking objects: 100% (14/14), 3.26 KiB | 238.00 KiB/s, done.
From https://github.com/ncabatoff/fakescraper
 * [new branch]      master     -> github/master
2021/04/10 01:46:52 Could not determine license for "github.com/ncabatoff/fakescraper": GET https://api.github.com/repos/ncabatoff/fakescrape
2021/04/10 01:46:52 Setting debian/watch to track git HEAD
2021/04/10 01:46:52 Could not determine license for "github.com/ncabatoff/fakescraper": GET https://api.github.com/repos/ncabatoff/fakescrape
2021/04/10 01:46:52 Done!

Packaging successfully created in /tmp/kk/golang-github-ncabatoff-fakescraper
    Source: golang-github-ncabatoff-fakescraper
    Binary: golang-github-ncabatoff-fakescraper-dev

Resolve all TODOs in itp-golang-github-ncabatoff-fakescraper.txt, then email it out:
    /usr/sbin/sendmail -t < itp-golang-github-ncabatoff-fakescraper.txt

Resolve all the TODOs in debian/, find them using:
    grep -r TODO debian

To build the package, commit the packaging and use gbp buildpackage:
    git add debian && git commit -a -m 'Initial packaging'
    gbp buildpackage --git-pbuilder

To create the packaging git repository on salsa, use:
    dh-make-golang create-salsa-project golang-github-ncabatoff-fakescraper

Once you are happy with your packaging, push it to salsa using:
    gbp push

NOTE: Full upstream git history has been included as per pkg-go team's
      new workflow.  This feature is new and somewhat experimental,
      and all feedback are welcome!
      (For old behavior, use --upstream-git-history=false)

The upstream git history is being tracked with the remote named "github".
To upgrade to the latest upstream version, you may use something like:
    git fetch github          # note the latest tag or commit-ish
    uscan --report-status     # check we get the same tag or commit-ish
    gbp import-orig --sign-tags --uscan --upstream-vcs-tag=<commit-ish>
```

参考文献

[1] "The Go Programming Language" https://golang.org/

[2] "Documentation – The Go Programming Language", https://golang.org/doc

[3] https://golang.org/doc/install

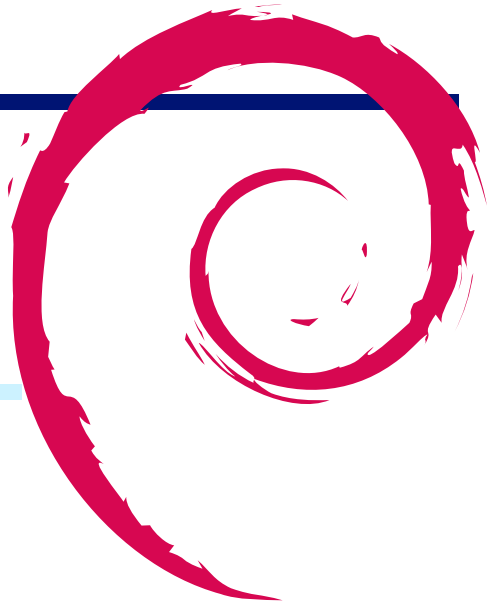
[4] "A Tour of Go" https://tour.golang.org/welcome/1

[5] https://go-tour-jp.appspot.com/welcome/1

[6] "Debian Go Packaging" https://go-team.pages.debian.net/packaging.html

[7] "'Go modules"' https://github.com/golang/go/wiki/Modules

[8] "golang Docker Official Images" https://hub.docker.com/_/golang



8 piuparts について調べてみた

杉本典充

8.1 はじめに

Debian パッケージの作り方を学ぶことは多かったのですが、作ったパッケージをテストする方法の知識がありませんでした。

今回は作成した Debian パッケージのインストールテストを行うツールである piuparts について調べてみました。

8.2 piuparts と piuparts.debian.org

piuparts とは、Debian パッケージがインストール、アップグレード、アンインストールを正しく処理することをテストするツールです[*19]。2021 年 7 月 11 日現在、Debian unstable において piuparts-1.1.3 の Debian パッケージを配布しています[*20]。

piuparts パッケージの説明には「chroot 環境に最小の Debian をインストールした環境を作り、その chroot 環境でパッケージのインストールとアップグレードとアンインストールを行い、インストール前とアンインストール後のディレクトリの状態を比較して、パッケージの処理に問題があるか判定する」と記述があります。

Debian デベロッパー レファレンスの Appendix「1 Debian メンテナツールの概要」に piuparts が紹介されており[*21]、Debian パッケージを扱う場合には piuparts を知っておくとよいと記載されています。

また、Debian Project ではパッケージを自動でテストするシステム `https://piuparts.debian.org/` を運用しています。このサイトでは日々アップロードされる Debian パッケージを piuparts を使ってテストを実行し続けています。テスト結果は `https://piuparts.debian.org/` の web サイトを見ることができます。

8.3 piuparts を使ったパッケージのテスト

8.3.1 テストする hello パッケージのビルド

ここでは、unstable にある hello パッケージ `https://packages.debian.org/sid/hello` を参考にしてみます。

まずは hello の ソースパッケージをダウンロードして、debian パッケージを手元でビルドしてみます。

[*19] `https://packages.debian.org/ja/sid/piuparts`

[*20] Debian 10 buster では python2.7 で動くツールでしたが、Debian 11 bullseye では python3 に対応しました。

[*21] `https://www.debian.org/doc/manuals/developers-reference/tools.ja.html#piuparts`

```
# apt-get update
# apt-get install build-essential
# apt-get build-dep hello
# apt-get source hello
# cd hello-2.10/
# dch
  -> バージョンを 1 つ上げます
# debuild -us -us
# ls -l .. | grep deb
-rw-r--r--  1 norimitu norimitu  36356  7 月 11 04:53 hello-dbgsym_2.10-2.1_amd64.deb
-rw-r--r--  1 norimitu norimitu   6224  7 月 11 04:53 hello_2.10-2.1.debian.tar.xz
-rw-r--r--  1 norimitu norimitu  56488  7 月 11 04:53 hello_2.10-2.1_amd64.deb
```

hello_2.10-2.1_amd64.deb パッケージができました。これ以降の記事では、piuparts コマンドを使ってこの Debian パッケージをテストしてみます。

8.3.2 piuparts の実行とログ

piuparts コマンドを使うには、piuparts パッケージをインストールします。

```
# apt-get install piuparts
# which piuparts
/usr/sbin/piuparts
```

piuparts コマンドを実行して hello パッケージのテストを実行します。”-l” オプションは、piuparts を実行したときに出力する標準出力をログファイルに追記するオプションです。

```
# piuparts hello_2.10-2.1_amd64.deb -l /tmp/piuparts_hello.log
```

最初はテストするパッケージのメタ情報を dpkg –info コマンドで出力します。

```
0m0.0s INFO: --------------------------------------------------------------------------------
0m0.0s INFO: To quickly glance what went wrong, scroll down to the bottom of this logfile.
0m0.0s INFO: FAQ available at https://wiki.debian.org/piuparts/FAQ
0m0.0s INFO: The FAQ also explains how to contact us in case you think piuparts is wrong.
0m0.0s INFO: --------------------------------------------------------------------------------
0m0.0s INFO: piuparts version 1.1.3 starting up.
0m0.0s INFO: Command line arguments: /usr/sbin/piuparts hello_2.10-2.1_amd64.deb -l /tmp/piuparts.log
0m0.0s INFO: Running on: Linux sid1 4.19.0-17-amd64 #1 SMP Debian 4.19.194-1 (2021-06-10) x86_64
0m0.0s DEBUG: Starting command: ['dpkg', '--info', 'hello_2.10-2.1_amd64.deb']
0m0.0s DUMP:
   new Debian package, version 2.0.
   size 56488 bytes: control archive=1872 bytes.
       758 bytes,    20 lines      control
      3601 bytes,    49 lines      md5sums
   Package: hello
   Version: 2.10-2.1
   Architecture: amd64
   Maintainer: Santiago Vila <sanvila@debian.org>
   Installed-Size: 280
   Depends: libc6 (>= 2.14)
   Conflicts: hello-traditional
   Breaks: hello-debhelper (<< 2.9)
   Replaces: hello-debhelper (<< 2.9), hello-traditional
   Section: devel
   Priority: optional
   Homepage: http://www.gnu.org/software/hello/
   Description: example package based on GNU hello
    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).
    0m0.0s DEBUG: Command ok: ['dpkg', '--info', 'hello_2.10-2.1_amd64.deb']
```

次に、debootstrap（’–no-merged-usr’ オプションを指定）を実行してテストに使う chroot 環境を作ります。このとき、/proc や /dev などを bind mount しています。

```
0m0.0s DEBUG: Created temporary directory /tmp/tmphcw125x_
0m0.0s DEBUG: Setting up minimal chroot for sid at /tmp/tmphcw125x_.
0m0.0s DEBUG: Starting command: ['debootstrap', '--variant=minbase',
'--keyring=/usr/share/keyrings/debian-archive-keyring.gpg', '--include=eatmydata',
'--no-merged-usr', '--components=main', 'sid', '/tmp/tmphcw125x_', 'http://deb.debian.org/debian']
  (debootstrap の標準出力は省略)
0m41.2s DEBUG: Command ok: ['debootstrap', '--variant=minbase',
'--keyring=/usr/share/keyrings/debian-archive-keyring.gpg', '--include=eatmydata', '--no-merged-usr',
'--components=main', 'sid', '/tmp/tmphcw125x_', 'http://deb.debian.org/debian']
0m41.2s DEBUG: Starting command: ['mount', '-t', 'proc', 'proc', '/tmp/tmphcw125x_/proc']
0m41.2s DEBUG: Command ok: ['mount', '-t', 'proc', 'proc', '/tmp/tmphcw125x_/proc']
0m41.2s DEBUG: Starting command: ['mount', '-t', 'devpts',
'-o', 'newinstance,noexec,nosuid,gid=5,mode=0620,ptmxmode=0666', 'devpts', '/tmp/tmphcw125x_/dev/pts']
0m41.2s DEBUG: Command ok: ['mount', '-t', 'devpts',
'-o', 'newinstance,noexec,nosuid,gid=5,mode=0620,ptmxmode=0666', 'devpts', '/tmp/tmphcw125x_/dev/pts']
0m41.2s DEBUG: Starting command: ['mount', '-o', 'bind', '/tmp/tmphcw125x_/dev/pts/ptmx',
'/tmp/tmphcw125x_/dev/ptmx']
0m41.2s DEBUG: Command ok: ['mount', '-o', 'bind', '/tmp/tmphcw125x_/dev/pts/ptmx', '/tmp/tmphcw125x_/dev/ptmx']
0m41.2s DEBUG: Starting command: ['mount', '-o', 'bind', '/dev/pts/4', '/tmp/tmphcw125x_/dev/console']
0m41.2s DEBUG: Command ok: ['mount', '-o', 'bind', '/dev/pts/4', '/tmp/tmphcw125x_/dev/console']
0m41.2s DEBUG: Starting command: ['mount', '-t', 'tmpfs', '-o', 'size=65536k', 'tmpfs', '/tmp/tmphcw125x_/dev/shm']
0m41.2s DEBUG: Command ok: ['mount', '-t', 'tmpfs', '-o', 'size=65536k', 'tmpfs', '/tmp/tmphcw125x_/dev/shm']
0m41.2s DEBUG: sources.list:
  deb http://deb.debian.org/debian sid main
0m41.2s DEBUG: Created policy-rc.d and chmodded it.
0m41.2s DEBUG: Created resolv.conf.
```

debootstrap で作成したディレクトリに chroot で入り、apt-get update を実行して試験対象のリポジトリのパッケージ情報を取得します。また、このタイミングで chroot のテスト環境のディレクトリ状態を保存しています。

```
0m41.2s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', 'update']
0m44.4s DUMP:
  Hit:1 http://deb.debian.org/debian sid InRelease
  Get:2 http://deb.debian.org/debian sid/main Translation-en [6509 kB]
  Fetched 6509 kB in 2s (2650 kB/s)
  Reading package lists...
0m44.4s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', 'update']
0m44.4s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'sh', '-c', 'apt-cache dumpavail | md5sum']
0m44.5s DUMP:
  c0117488ce119ec145559966f8a0cef8  -
0m44.5s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'sh', '-c', 'apt-cache dumpavail | md5sum']
0m44.5s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', 'clean']
0m44.5s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', 'clean']
0m44.5s DEBUG: Recording chroot state
```

dpkg-query、dpkg-divert –list を実行してパッケージの情報や中身を調べています。

```
0m44.6s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-query', '-W',
'-f', '${Status}\\t${binary:Package}\\t${Package}\\t${Version}\\n']
  (dpkg-query の結果は省略)
0m44.6s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-query', '-W',
'-f', '${Status}\\t${binary:Package}\\t${Package}\\t${Version}\\n']
0m44.6s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-divert', '--list']
0m44.6s DUMP:
  diversion of /usr/share/man/man1/sh.1.gz to /usr/share/man/man1/sh.distrib.1.gz by dash
  diversion of /bin/sh to /bin/sh.distrib by dash
0m44.6s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-divert', '--list']
0m44.6s INFO: apt-cache does not know about any of the requested packages
0m44.6s DEBUG: Starting command: ['lsof', '-w', '+D', '/tmp/tmphcw125x_']
0m44.8s DEBUG: Command failed (status=1), but ignoring error: ['lsof', '-w', '+D', '/tmp/tmphcw125x_']
0m45.3s DEBUG: No broken symlinks as far as we can find.
0m45.3s DEBUG: Starting command: ['lsof', '-w', '+D', '/tmp/tmphcw125x_']
0m45.4s DEBUG: Command failed (status=1), but ignoring error: ['lsof', '-w', '+D', '/tmp/tmphcw125x_']
0m45.9s DEBUG: No broken symlinks as far as we can find.
0m45.9s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-query', '-f', '${Version}\n', '-W', 'apt']
0m45.9s DUMP:
  2.2.4
0m45.9s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-query', '-f', '${Version}\n', '-W', 'apt']
```

piuparts コマンドの引数に指定した Debian パッケージを chroot のテスト環境内にコピーし、apt-get install コマンドを実行してコピーした Debian パッケージをインストールします。

```
0m45.9s DEBUG: Copying hello_2.10-2.1_amd64.deb to /tmp/tmphcw125x_/tmp
0m45.9s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', '-y',
'--allow-downgrades', 'install', './tmp/hello_2.10-2.1_amd64.deb']
0m47.1s DUMP:
  Reading package lists...
  Building dependency tree...
  The following NEW packages will be installed:
    hello
  0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
  Need to get 0 B/56.5 kB of archives.
  After this operation, 287 kB of additional disk space will be used.
  Get:1 /tmp/hello_2.10-2.1_amd64.deb hello amd64 2.10-2.1 [56.5 kB]
  debconf: delaying package configuration, since apt-utils is not installed
  Selecting previously unselected package hello.

  Preparing to unpack /tmp/hello_2.10-2.1_amd64.deb ...
  Unpacking hello (2.10-2.1) ...
  Setting up hello (2.10-2.1) ...
0m47.1s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', '-y',
'--allow-downgrades', 'install', './tmp/hello_2.10-2.1_amd64.deb']
0m47.1s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-query',
'-f', '${Package} ${Status}\n', '-W', 'hello']
0m47.1s DUMP:
  hello install ok installed
0m47.1s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-query',
'-f', '${Package} ${Status}\n', '-W', 'hello']
0m47.1s INFO: Installation of ['./tmp/hello_2.10-2.1_amd64.deb'] ok
```

chroot のテスト環境内のディレクトリを調べたり、インストールしているパッケージのバージョンを調べています。

```
0m47.1s DEBUG: Removing /tmp/tmphcw125x_/./tmp/hello_2.10-2.1_amd64.deb
0m47.1s DEBUG: Starting command: ['lsof', '-w', '+D', '/tmp/tmphcw125x_']
0m47.3s DEBUG: Command failed (status=1), but ignoring error: ['lsof', '-w', '+D', '/tmp/tmphcw125x_']
0m47.7s DEBUG: No broken symlinks as far as we can find.
0m48.0s DEBUG: Starting command: ['debsums', '--root', '/tmp/tmphcw125x_', '-ac', '--ignore-obsolete']
0m48.5s DEBUG: Command ok: ['debsums', '--root', '/tmp/tmphcw125x_', '-ac', '--ignore-obsolete']
0m48.5s DEBUG: Starting command: ['dpkg-query', '-f', '${Version}\n', '-W', 'adequate']
0m48.5s DUMP:
  0.15.6
0m48.5s DEBUG: Command ok: ['dpkg-query', '-f', '${Version}\n', '-W', 'adequate']
0m48.5s INFO: Running adequate version 0.15.6 now.
0m48.5s DEBUG: Starting command: ['adequate', '--root', '/tmp/tmphcw125x_', 'hello']
0m48.6s DEBUG: Command ok: ['adequate', '--root', '/tmp/tmphcw125x_', 'hello']
0m48.6s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-query', '-W',
'-f', '${Status}\\t${binary:Package}\\t${Package}\\t${Version}\\n'
0m48.6s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-query', '-W',
'-f', '${Status}\\t${binary:Package}\\t${Package}\\t${Version}\\n']
```

chroot のテスト環境内でインストールしたパッケージを apt-get remove を実行してアンインストールします。これで Debian パッケージのインストールからアンインストールするテストは終了です。

```
0m48.6s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get',
'remove', 'hello']
   (省略)
0m49.1s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', 'remove', 'hello']
0m49.1s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg', '--purge', 'hello']
0m49.2s DUMP:
  dpkg: warning: ignoring request to remove hello which isn't installed
0m49.2s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg', '--purge', 'hello']
0m49.2s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg', '--purge', '--pending']
0m49.2s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg', '--purge', '--pending']
0m49.2s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg', '--remove', '--pending']
0m49.2s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg', '--remove', '--pending']
0m49.2s DEBUG: Starting command: ['lsof', '-w', '+D', '/tmp/tmphcw125x_']
0m49.4s DEBUG: Command failed (status=1), but ignoring error: ['lsof', '-w', '+D', '/tmp/tmphcw125x_']
0m49.8s DEBUG: No broken symlinks as far as we can find.
0m49.8s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-divert', '--list']
0m49.8s DUMP:
  diversion of /usr/share/man/man1/sh.1.gz to /usr/share/man/man1/sh.distrib.1.gz by dash
  diversion of /bin/sh to /bin/sh.distrib by dash
0m49.8s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-divert', '--list']
0m49.8s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', 'clean']
0m49.9s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', 'clean']
0m49.9s DEBUG: Recording chroot state
0m50.0s INFO: PASS: Installation and purging test.
```

引き続き、インストール、アップグレード、アンインストールのテストを実行します。まずはリポジトリにある同名の hello パッケージを apt-get install でインストールします。

```
0m50.0s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-cache', 'show',
  '--no-all-versions', 'hello']
0m50.9s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-cache', 'policy', 'hello']
0m50.9s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', '-y', 'install', 'hello']
0m51.5s DUMP:
  Reading package lists...
  Building dependency tree...
  Reading state information...
  The following NEW packages will be installed:
    hello
    0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
0m51.5s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', '-y', 'install', 'hello']
0m51.5s DEBUG: Starting command: ['lsof', '-w', '+D', '/tmp/tmphcw125x_']
0m51.7s DEBUG: Command failed (status=1), but ignoring error: ['lsof', '-w', '+D', '/tmp/tmphcw125x_']
0m52.2s DEBUG: No broken symlinks as far as we can find.
0m52.2s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-query', '-f', '${Version}\n', '-W', 'apt']
0m52.2s DUMP:
  2.2.4
0m52.2s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-query', '-f', '${Version}\n', '-W', 'apt']
```

次に piuparts コマンドの引数に指定した Debian パッケージを chroot のテスト環境内にコピーし、apt-get install コマンドを実行してコピーしたパッケージでアップグレードします。

```
0m52.2s DEBUG: Copying hello_2.10-2.1_amd64.deb to /tmp/tmphcw125x_/tmp
0m52.2s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', '-y',
'--allow-downgrades', 'install', './tmp/hello_2.10-2.1_amd64.deb']
0m52.8s DUMP:
  Reading package lists...
  Building dependency tree...
  Reading state information...
  The following packages will be upgraded:
    hello
  1 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

0m52.8s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', '-y',
'--allow-downgrades', 'install', './tmp/hello_2.10-2.1_amd64.deb']
0m52.8s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-query',
'-f', '${Package} ${Status}\n', '-W', 'hello']
0m52.8s DUMP:
  hello install ok installed
0m52.8s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-query',
'-f', '${Package} ${Status}\n', '-W', 'hello']
0m52.8s INFO: Installation of ['./tmp/hello_2.10-2.1_amd64.deb'] ok
```

次に apt-get remove コマンドを実行してパッケージをアンインストールします。これで Debian パッケージのインストールからアップグレード、アンインストールするテストは終了です。

```
0m54.3s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', 'remove', 'hello']
0m54.9s DUMP:
  Reading package lists...
  Building dependency tree...
  Reading state information...
  The following packages will be REMOVED:
    hello
  0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.
0m54.9s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', 'remove', 'hello']
0m54.9s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg', '--purge', 'hello']
0m54.9s DUMP:
  dpkg: warning: ignoring request to remove hello which isn't installed
0m54.9s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg', '--purge', 'hello']
0m54.9s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg', '--purge', '--pending']
0m54.9s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg', '--purge', '--pending']
0m54.9s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg', '--remove', '--pending']
0m54.9s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg', '--remove', '--pending']
0m54.9s DEBUG: Starting command: ['lsof', '-w', '+D', '/tmp/tmphcw125x_']
0m55.1s DEBUG: Command failed (status=1), but ignoring error: ['lsof', '-w', '+D', '/tmp/tmphcw125x_']
0m55.5s DEBUG: No broken symlinks as far as we can find.
0m55.5s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-divert', '--list']
0m55.5s DUMP:
  diversion of /usr/share/man/man1/sh.1.gz to /usr/share/man/man1/sh.distrib.1.gz by dash
  diversion of /bin/sh to /bin/sh.distrib by dash
0m55.5s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'dpkg-divert', '--list']
0m55.5s DEBUG: Starting command: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', 'clean']
0m55.5s DEBUG: Command ok: ['chroot', '/tmp/tmphcw125x_', 'eatmydata', 'apt-get', 'clean']
0m55.5s DEBUG: Recording chroot state
0m55.7s INFO: PASS: Installation, upgrade and purging tests.
```

最後に debootstrap で構築した chroot 環境をクリーンナップします。

```
0m55.9s DEBUG: Starting command: ['umount', '/tmp/tmphcw125x_/dev/shm']
0m55.9s DEBUG: Command ok: ['umount', '/tmp/tmphcw125x_/dev/shm']
0m55.9s DEBUG: Starting command: ['umount', '/tmp/tmphcw125x_/dev/console']
0m55.9s DEBUG: Command ok: ['umount', '/tmp/tmphcw125x_/dev/console']
0m55.9s DEBUG: Starting command: ['umount', '/tmp/tmphcw125x_/dev/ptmx']
0m55.9s DEBUG: Command ok: ['umount', '/tmp/tmphcw125x_/dev/ptmx']
0m55.9s DEBUG: Starting command: ['umount', '/tmp/tmphcw125x_/dev/pts']
0m56.0s DEBUG: Command ok: ['umount', '/tmp/tmphcw125x_/dev/pts']
0m56.0s DEBUG: Starting command: ['umount', '/tmp/tmphcw125x_/proc']
0m56.0s DEBUG: Command ok: ['umount', '/tmp/tmphcw125x_/proc']
0m56.0s DEBUG: Starting command: ['rm', '-rf', '--one-file-system', '/tmp/tmphcw125x_']
0m56.1s DEBUG: Command ok: ['rm', '-rf', '--one-file-system', '/tmp/tmphcw125x_']
0m56.1s DEBUG: Removed directory tree at /tmp/tmphcw125x_
0m56.1s INFO: PASS: All tests.
0m56.1s INFO: piuparts run ends.
```

8.4 piuparts コマンドのオプションの紹介

8.4.1 -d オプション：バージョンを指定してテスト

piuparts に "-d _distro_" オプションを指定すると、deboostrap する環境のバージョンを指定してテストを実行します。"-d" オプションは省略でき、省略した場合は "-d sid" を指定したときと同じ動きをします。

```
# piuparts hello_2.10-2.1_amd64.deb -d sid -l /tmp/piuparts.log
```

この "-d" オプションは複数を連続して指定することができます。その場合、最初に指定した"-d" オプションのバージョンで debootstrap してからパッケージをインストールし、それ以降に指定した "-d" のバージョンへ順番に apt-get dist-upgrade を行い、最後まで dist-upgrade が完了した状態で piuparts コマンドに指定したパッケージのインストールテストを実行します。

以下の例では、buster の環境を deboostrap で構築して hello パッケージをインストールし、bullseye へ apt-get dist-upgrade し、sid へ apt-get dist-upgrade した後に、hello_2.10-2.1_amd64.deb のアップグレードとアンインストールのテストを実行します。

```
# piuparts hello_2.10-2.1_amd64.deb -d buster -d bullseye -d sid -l /tmp/piuparts.log
```

8.4.2 インストール済みパッケージのテスト

これまでの紹介では Debian パッケージ自体を piuparts の引数に指定してきました。テストするパッケージは .deb ファイルではなく、"–apt"オプションとパッケージ名を指定することで実行している PC にインストールしている Debian パッケージのテストを行うことができます。

```
# piuparts hello --apt -l /tmp/piuparts.log
```

8.4.3 -p オプション：pbuilder のイメージを使う

Debian パッケージをクリーン環境でビルドするツールに pbuiler[*22] があります。

以下のように pbuilder コマンドを実行して debootstrap したクリーン環境で Debian パッケージのビルドを行うことができます。

```
# pbuilder create
# pbuilder update
# pbuilder build hello_2.10-2.1.dsc
```

pbuilder create を実行したとき、または pbuilder update を実行したときに pbuilder が利用する deboostrap のイメージを「/var/cache/pbuilder/base.tgz」に保存します。

piuparts に "-p" オプションを指定すると、debootstrap を実行する代わりに pbuilder が保存している「/var/cache/pbuilder/base.tgz」をテスト環境に利用します。

deboostrap を実行するとインターネットから毎回約 100 MByte 程度のファイルをダウンロードするため"-p" オ

[*22] https://packages.debian.org/ja/sid/pbuilder

プションを使うことでネットワーク帯域やダウンロード時間を節約できます。

似たようなオプションに ”–basetgz=file” があり、base.tgz ファイルを任意のパスで指定することもできます。

8.4.4 –docker-image：docker イメージを使ってテストする

piuparts に ”–docker-image=IMAGE” オプションを指定すると、テストに使う環境に debootstrap で構築したイメージを使わず、docker イメージを使ってテストを実行します。

実際試してみると、chroot コマンドを使って chroot のテスト環境内でコマンドを実行するところが docker run、docker inspect、docker top、docker exec などを使ってテストを実行する動きに変わります。

8.4.5 –merged-usr：/usr マージした環境で debootstrap する

piuparts に ”–merged-usr” オプションを指定すると、debootstrap するときに ”–merged-usr” オプションを指定して実行します（オプションなしの場合は ”–no-merged-usr” を指定しています）。

これで /usr マージした環境で Debian パッケージのインストールテストを実行することができます。

8.4.6 –arch：アーキテクチャを指定して deboostrap する

piuparts に ”–arch=ARCH” オプションを指定すると、debootstrap するときに”–arch=ARCH” を指定して実行します。

つまり、異なるアーキテクチャの chroot 環境を構築して Debian パッケージのインストールテストを実行することができます。ただ、テストする PC が –arch オプションに指定したアーキテクチャで動作する環境であること、Debian パッケージも –arch オプションに指定したアーキテクチャ向けにビルドしてある必要があります。

8.5 終わりに

piuparts を使ってパッケージのテストを試してみました。今回試した範囲では Debian Developer が作成したテストが合格するパッケージのみでした。今後パッケージを自分で作成するときには様々な不具合に遭遇と思いますので、piuparts を使ってうまくテストできるようになれるとよいと思います。

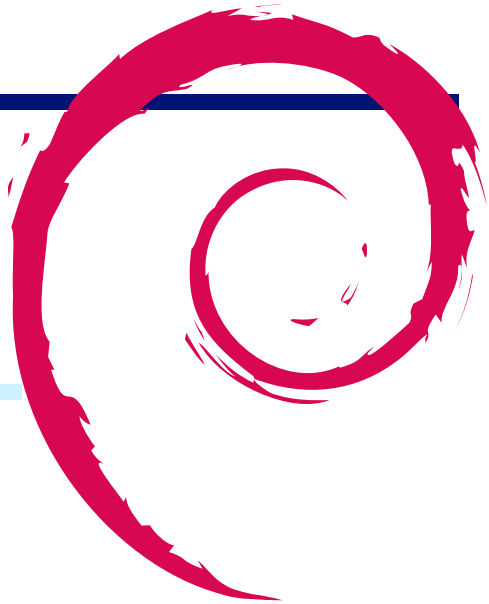
なお、piuparts コマンドはネットワーク帯域を多く使うため、回線の帯域や通信量の残量に注意してご利用ください。[*23]

8.6 参考資料

- Debian Wiki - piuparts `https://wiki.debian.org/piuparts`
- 岩松 信洋、「piuparts の使い方」、あんどきゅめんてっど でびあん 2010 年夏号 `https://tokyodebian-team.pages.debian.net/pdf2010/debianmeetingresume201004.pdf`
- Holger Levsen、DebConf19「i’m (a bit) sick of maintaining piuparts.debian.org (mostly) alone, please help」[*24]

[*23] `https://piuparts.debian.org/doc/README_server.txt` に piuparts のサーバでは 100 Mbps の帯域を簡単に使い切ってしまうから気をつけてください、と注意書きがあります。

[*24] `https://debconf19.debconf.org/talks/103-im-a-bit-sick-of-maintaining-piupartsdebianorg-mostly-alone-please-help/`



9 DebConf 21 のイベント共有

勉強会参加者一同

9.1 参加者（5 名）

- dictoss
- yy_y_ja_jp
- tyamadajp
- uwabami
- yosukesan

9.2 DebConf 21 のビデオを見た感想

9.2.1 DebConf21 Opening

- 昨年とほぼ同じ構成
- 数分で終わった

9.2.2 OpenStack Cluster Installer in Bullseye: what's new, what to expect

- DebConf21 のオンライン会場を提供するクラウドはこの OpenStack で提供していた

9.2.3 Cloud BoF

- 提供しているイメージ
  - 提供している arch は 3 つ (amd64、arm64、ppc64el)
  - AWS 向け、Azure 向け、OpenStack 向け、QEMU-KVM 向けのイメージをビルドしている
- イメージは完全に自動ビルドする仕組みになっている
  - stable image は手動トリガーにより自動でリリースされる
- ifupdown がいけていないとの意見あり
- 質問
  - OpenStack 関連のものが多かった
  - メージの利用者統計は何を使っているか
    - クラウドサービスの market place
    - popcon

9.2.4 Mass Campaigns to increase Debian adoption in India

- まだ見れていないが、興味がある（tyamadajp）

9.2.5 Debian Science BoF

- Debian が提供する FORTRAN パッケージは、LLVM ベースのコンパイラを利用できるものを提供中

9.2.6 Probing into the world of Data Science

- NumPy の紹介などの科学技術計算の一般的な話が多かった

9.2.7 Making use of snapshot.debian.org for fun and profit

- snapshot.debian.org を使い倒しているセッションで、使い倒しすぎていて、次世代 snapshot.debian.org を作るに至ったとのこと
- snapshot.debian.org ベースの debootstrap や bisecting tool など製作した便利ツールを色々紹介していた
- 最大のネックは snapshot.debian.org の不安定さということが明らかにされ、それを改善するために以下を作るに至ったという話の流れ
  - snapshot.debian.org のパッケージ情報を安定して参照できるメタデータサービス
  - snapshot.debian.org のアーカイブ内容をもっと安定的に配信できるサーバ
- 紹介されているツールも有用そうでおすすめ

9.2.8 Summer with Debian

- Google Summer Of Code に Debian Project で支援できることを紹介する話
- 特に何のプロジェクトや作業に取り組みたいと応募が来ている、という話は出てこなかった
- 学生側から何を支援してくれるのか、という質問に終始していた感じがある

9.2.9 Contemporary networking configuration with ifupdown-ng

- ifupdown-ng を作った人が ifupdown-ng を紹介する話
- ifupdown(1) は軽量でシンプルだが、拡張性に乏しい（Debian ではデフォルトで利用）
- ifupdown2 は拡張性があるが、Python 製
- ifupdown-ng は ifupdown2 にインスパイアを受けて開発した
- Alpine では ifupdown-ng をすでに採用している
  - Alpine-3.13 では 0.10.3 を採用していて、デフォルトになっている（苦情は少ない）
  - Alpine-3.14 では 0.11.2 を採用し、Wi-Fi クライアントのサポートが入る予定
- 今後の予定
  - reload のサポート
  - ifupdown-ng をパッケージ化してくれる DM を探している
  - ifmond udev のようなイメージの、ネットワークの変更をイベント監視するデーモンの開発
- 感想
  - nmcli のようなコマンドラインツールがほしいイメージが作者にあるように思った (dictoss)

9.2.10 Create YOUR custom Debian image with recipes

- Debian のバイナリパッケージを活用してカスタム Debian/Linux の rootfs やシステムイメージを簡単高速再現性ありでビルドしてくれる「isar」を紹介する話
- isar は bitbake を裏で使った構成になっている
  - システムやコンテナのイメージ生成では debootstrap から bitbake まで既成のツール・方式が色々あるが、前者だとイメージ化に向かず、後者はソースビルドを主に想定していて「簡単・高速・再現性あり」というのがなかなか満たせない
- isar は Debian package やソースビルド機能を bitbake で駆動することでソースレベルのカスタマイズもできるイメージビルドツールになっている

9.2.11 Python Team BoF

- Python 2 の廃止について
  - python2.7 関連のパッケージは 99% 削除した
  - pypy、virtualenv、他 ... などのビルドでは未だに python2.7 に依存している。独自にビルドすれば依存は外せるが互換性がなくなる
- vendoring 問題
  - PEP 517、PEP 518 に対応したツールから deb パッケージを作るツールがない (誰かやってほしい)
  - PEP 686 の対応をどうするか
    - Python 側で行っているディスカッション[*25]
    - python-pip-whl という debian の python package だけで依存性を解決する機能を upstream で作ってもらっているが、upstream は心良く思っていない
    - システムと pip で同じソースがインストールされるのは debian のポリシーに違反すると思うが、pip のパッケージ数の恩恵をユーザーが受けられるし、debian pakcage を管理する側も楽になるため許容してはどうかという意見あり
    - python パッケージ管理関連に依存があるパッケージだけは、セキュリティに問題がないようにしたい
      - セキュリティチームに連絡

9.2.12 GoVarnam, a new "Intelligent" Input Method for Indian Languages in Desktop & Mobile

- インドの言語 マラヤーラム語などの入力メソッド
- Google の Gmail などにも付いてる模様
  - Google Input Tools のフリー版を目指してるようで、情報がオンラインに流れずオフラインで使えることを強調していた
- 発表中にデモをしていた（ただしマラヤーラム語で何を書いてるのかはよくわからなかった）
- フロントエンドには ibus と Web インターフェースを用意していた
- 日本語のローマ字入力・かな入力のような感じで入力方法は 2 つある模様
- 日本語の形態素解析ほどまではやってなさそうであったが、一緒に取り組める部分もあるのではとも思った (yy_y_ja_jp)

[*25] https://discuss.python.org/t/graceful-cooperation-between-external-and-python-package-managers-pep-668/10302

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。

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not 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。ソースコードは

https://salsa.debian.org/tokyodebian-team/monthly-report.git

から取得できます。

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、東京および関西周辺で毎月行なわれている『東京エリア Debian 勉強会』『関西 Debian 勉強会』およびその合同開催分の（2020 年 12 月-2021 年 11 月）で使用された資料・小ネタ・必殺技などを一冊にまとめたものです。2021 年 4 月、2021 年 6 月、2021 年 8 月については資料がないなどにより収録なし。内容は無保証、つっこみなどがあれば勉強会にて。

あんどきゅめんてっど でびあん 2022 年夏号

2022 年 8 月 13 日　　初版第 1 刷発行

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