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 2019 年夏号 2019 年 8 月 12 日 初版発行

デビアン勉強会

目次

1 Introduction 2

2 Debian Update 3

3 Buster デスクトップの日本語入力 10

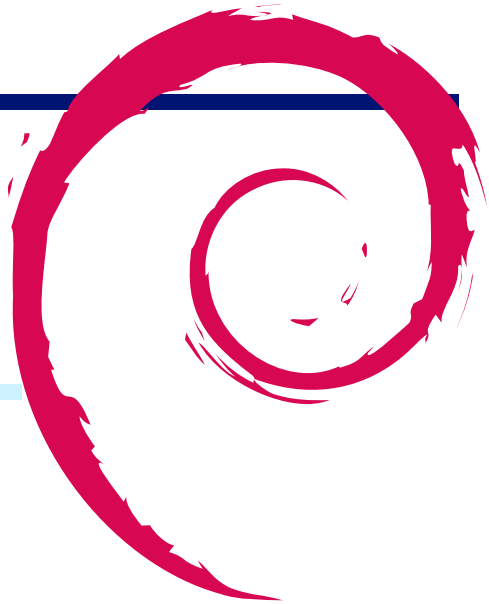
4 /usr Merge について 19

5 Systemd に入門してみた 24

6 systemd と sysvinit のデーモンで動作する Debian パッケージの作り方を調べてみる 32

7 grml-debootstrap を用いた USB 起動メモリの作成 37

8 Rust で書いたツールの debian パッケージングに挑戦してみた話 (仮) 42



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。

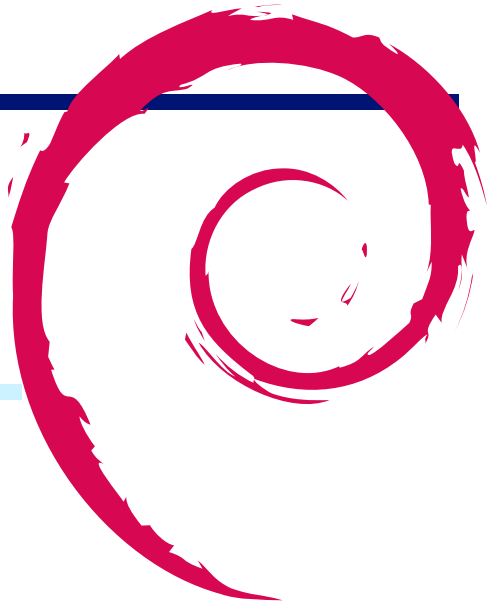
1.2 関西 Debian 勉強会

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 Update

杉本

2.1 目次

- Debian とは？
- Debian JP Project と Debian 勉強会
- Debian 10 (Buster)
- Debian Updates
- 今後のイベント

2.2 Debian とは？

フリー/オープンなユニバーサルオペレーティングシステム を作成しようとするボランティアベースのプロジェクト

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

世界規模で開発が行われており、63ヶ国、約 1000 名の Debian 公式開発者が開発を行っている。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開発者が参加していることになる。

- 様々な用途に使える汎用的な作り
  - デスクトップ PC、ノート 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

  - Ubuntu や Raspbian といったディストリビューションのベースは Debian
  - 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

- 2019 年 2 月の時点で、最新版は Debian 9.8 (コードネーム: Stretch)
  - パッケージ数は約 51000 を提供
  - 公式にサポートする CPU アーキテクチャは 10
- 約 2 年毎にリリース
- 次のリリース Debian 10 (コードネーム: Buster) は 2019 年中頃にリリースすると思われる
- コードネームはトイ・ストーリーのキャラクターを採用

- 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ーの在り方
- 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、内容やルール、ファイル配置の方針を定義

まとめると

- Debian はフリー/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 1000 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 とは

- 日本で 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、パッケージメンテナの育成など

2.4 Debian 勉強会

- 2005 年 1 月開始
- Debian Developer 上川さん発起人
- 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、ユーザによる勉強会

2.5 Debian 勉強会:解決したい内容

- 問題
  - ML と IRC で情報交換していた

  - face-to-face であう場所がない
  - 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
  - 資料を作成する。(GPL で！)

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

2.6 最新安定版 Debian 10 (Buster)

Debian 10 には「Buster」というコードネームをつけています。

- Debian は time-based freeze を採用しており、およそ 2 年毎のリリースを目指す

2.6.1 テーマ

テーマは Alex Makas さん提案の「futurePrototype」に決定。

`https://bits.debian.org/tag/artwork.html`

`https://wiki.debian.org/DebianArt/Themes/futurePrototype`

2.6.2 提供ソフトウェアのバージョン

ソフトウェアは以下のバージョンでテスト中

- Linux カーネル 4.19
- ツールチェイン (GCC 8.2.0, binutils 2.31.1, glibc 2.28), LLVM 7.0.1, 6.0.1
- Perl 5.28.1, Python 2.7.15/3.7.2, Ruby 2.5.1, PHP 7.3.2, Go 1.11.5, OpenJDK 11.0.2, 8u171
- GNOME 3.30, KDE 5.14.5, Cinnamon 3.8.8, MATE 1.20, Xfce 4.12.5, lxde 0.99.0, lxqt 0.14.0
- MariaDB 10.3.12, PostgreSQL 11.1, sqlite 3.26.0, 2.8.17
- OpenSSH 7.9p1, OpenSSL 1.1.1a, GnuPG 2.2.12/1.4.23
- etc..

2.6.3 インストーラ

Debian Installer Buster Alpha 5 Release

- https://www.debian.org/devel/debian-installer/News/2019/20190202
- いち早く Buster を試してみたい方は以下からダウンロードしてインストールできます。
  - https://www.debian.org/devel/debian-installer/index.ja.html

2.6.4 リリースまでの流れ

Debian 10 のリリースまでの流れ
https://wiki.debian.org/DebianBuster

- 2019-01-12: Transition freeze
  - これ以降、大規模な変更や他のパッケージに大きな影響を与える変更を行ってはいけない
- 2019-02-12: Soft freeze　←今はこの段階
  - 変更は小規模で的を絞った内容に限って行う
  - 例）バグ修正、セキュリティ修正、他のパッケージとの調整
  - この期日までに testing に移行していないパッケージは次のリリースに含まれません
- 2019-03-12: Full freeze
  - unstable にアップロードしたパッケージを testing に移行するにはリリースチームのレビューが必要
  - debdiff を添えて unblock というバグ報告を行いレビューしてもらう

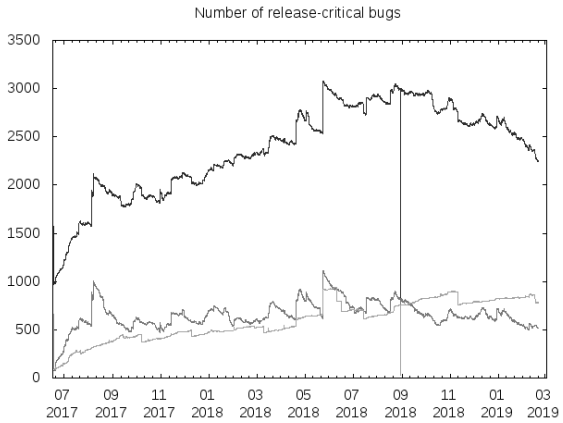
- Debian では、リリースクリティカルバグ（RC bug）が 0 個になったときにリリースする
- 現在のリリースクリティカルバグの一覧
  - https://bugs.debian.org/release-critical/

Release-critical bugs status

Wed Feb 20 12:00:00 UTC 2019

Total number of release-critical bugs: 2253
Number that have a patch: 297
Number that have a fix prepared and waiting to upload: 32
Number that are being ignored: 36
Number concerning the current stable release: 778
Number concerning the next release: 520

Number concerning the previous stable release: 34



2.6.5 Debian 9 から 10 の変更点

【ご注意ください】この資料は 2 月現在のものです。リリースノート[*1]を参照してください。
usr-merge の完全適用は Buster では一部見送り

- https://wiki.debian.org/UsrMerge
- /bin,/sbin,/lib を、/usr/bin,/usr/sbin,/usr/lib へのシンボリックにすること
- 一部のパッケージで問題が出たため、技術委員会も含めて Buster ではどうするか 議論になった

[*1] https://www.debian.org/releases/buster/amd64/release-notes/index.ja.html

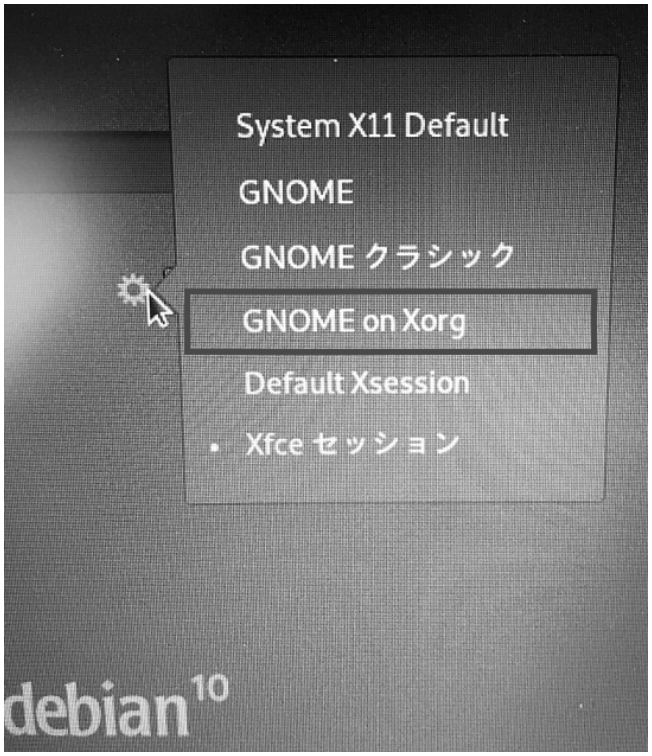
  - 議論の結果、usr-merge の取り組みは Buster のリリース後に改めて進めことになった
  -「tech-ctte: Should debootstrap disable merged /usr by default?」 https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=914897
  -「usrmerge – plan B?」のスレッドを参照 https://lists.debian.org/debian-devel/2018/11/msg00354.html

linux kernel と glibc のバージョンアップに伴う変更

- apparmor はデフォルトで有効になる
- Buster をコンテナ環境で動かす場合の制限
  - glibc-2.26 以降を採用しており、linux-3.2 以降が必要
  - ホスト側がとても古いサーバの場合に Buster がコンテナ環境で動かない場合あり
- Buster をホスト環境で動かす場合の制限
  - amd64 において、古い'virtual syscall' がデフォルトで無効
  - glibc-2.13 以下のシステムはコンテナ環境で動かない
    - Debian-7 以下、RHEL/CentOS-6 以下が該当
  - 有効にする場合はカーネルパラメータに vsyscall=emulate を追加

GNOME は Wayland での動作がデフォルト

- 「GNOME on Xorg」も選べます
- 他のデスクトップ環境は Xorg で動作します
- GNOME で Wayland を使う場合に日本語を入力できないことがある
  - 日本語入力処理（Input Method）周りに問題が残っており、修正対応中



- openssl-1.1.1 系の採用により TLSv1.3 が利用可能
- openssh において SSH 1 プロトコルが完全削除。SSH 2 プロトコルを利用すること。

- Debian 10 では、Debian 9 と同じアーキテクチャをサポートする[*2]
  - amd64, i386（i686 以降）, armel, armhf, arm64, mips, mipsel, mips64el, ppc64el, s390x

2.6.6 バグレポート

- 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

2.7 Debian Updates

2.7.1 released timeline

- 2018/11/10: Updated Debian 9.6 released
- 2019/01/23: Updated Debian 9.7 released
- 2019/02/16: Updated Debian 9.8 released

2.7.2 Bits from the DPL

Bits from the DPL

- 月に 1 回”debian-devel-announce@lists.debian.org”の ML に Chris さんがプロジェクトの進捗を報告している
- 時間のない方でもこれを読んでおけば Debian Project の大まかな動きがわかる

- 2018/10/31 `https://lists.debian.org/debian-devel-announce/2018/10/msg00005.html`
- 2018/11/30 `https://lists.debian.org/debian-devel-announce/2018/11/msg00007.html`
- 2018/12/31 `https://lists.debian.org/debian-devel-announce/2018/12/msg00006.html`
- 2019/01/31 `https://lists.debian.org/debian-devel-announce/2019/01/msg00010.html`

2.7.3 Rust

- 2018/11/02: Rust now available on 14 Debian architectures

  Rust コンパイラのブートストラップが mips、mips64el、mipsel、powerpcspe でもできるようになった。`https://lists.debian.org/debian-devel-announce/2018/11/msg00000.html`

2.7.4 SSH restriction

- 2018/11/10: incoming SSH restriction for *.debian.org

  debian.org のドメインを持つ一部のホストにおいて、セキュリティの観点から debian.org のサーバからのみ ssh 接続を受け付けるように設定を変更した。`https://lists.debian.org/debian-devel-announce/2018/11/msg00003.html`

2.7.5 CTTE vendor-specific patch

- 2018/11/13: CTTE decision on vendor-specific patch series (bug #904302)

[*2] 2019 年 2 月の本記事作成時にはサポートアーキテクチャはまだ確定していなかった。

DPKG にあるベンダー固有パッチ機能（主に派生ディストリビューションの Ubuntu などが利用している）を継続利用するか利用を止めるか議論があった。議論の結果、Debian 10 Buster のリリース後は利用を止めるべきとの提案がされた。
https://lists.debian.org/debian-devel-announce/2018/11/msg00004.html

2.7.6 GSoC2019

- 2019/01/16: Google Summer of Code 2019

Google Summer of Code 2019 で取り組むプロジェクトの wiki ページが立ち上がった。応募の例は、プロジェクトの例は「Android SDK Tools in Debian」、「Continuous Integration for biological applications inside Debian」、「Debian PHP Packaging」、「New Contributor Wizard」「Reproducible Builds」など。https://lists.debian.org/debian-devel-announce/2019/01/msg00007.html

2.8 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」、GPG 関連の記事など（隔月連載）

3　Buster デスクトップの日本語入力

yy_y_ja_jp

Debian の日本語入力は uim-mozc がデフォルト[*3]で、お好みで ibus や fcitx など、anthy や skk なども使えます。Stretch (9) までと同様に Buster (10) でもふつうに使えるように、im-config というパッケージで対応を行いました。

Debian は長らく GNOME がデフォルトデスクトップです。Stretch では図 1 のような感じでした。Buster では図 2 のようになり、あまり変わっていないように見えます。しかし、中身はかなり変わりました。

図 1　Stretch のデフォルトデスクトップ

図 2　Buster のデフォルトデスクトップ

特に変わったのは GNOME で、Stretch までは GNOME on Xorg がデフォルトでしたが、Buster では GNOME on Wayland

[*3] 一部のアーキテクチャは uim-anthy

がデフォルトになりました。Wayland では Xorg で行われていた設定が何もされないなど様々な問題がありましたが、基本的には使えるようにしました。

3.1 デフォルトデスクトップ

Debian インストーラのソフトウェア選択画面では複数のデスクトップ環境を選べますが（図 3）、そのうち一番上にあるものがデフォルトデスクトップと呼ばれています。なおこの画面で選べるデスクトップ環境は、Buster では LXQt が加わった以外は特に並び順も変わっていません。

図 3　Debian インストーラのソフトウェア選択画面

ここで GNOME にもチェックを入れて進むと、GNOME デスクトップとともにログイン画面のディスプレイマネージャgdm3がインストールされます。

インストール完了後に起動すると gdm3 のログイン画面になります。ユーザーを選択するとパスワード入力画面になりますが、ここで Stretch の GNOME から変わったところがあります。サインインボタンの左隣にある歯車をクリックすると、Stretch のログイン画面では図 4 のように “System X11 Default” が選択されており、また一番下のメニューは “GNOME on Wayland” となっていました。しかし Buster では図 5 のように一見 Stretch のときにもあったようなメニューの “GNOME” が選択されており、また一番下のメニューが “GNOME on Xorg” になっています。同じメニュー名 “GNOME” でも Stretch までは実体が GNOME on Xorg でしたが、Buster のものは GNOME on Wayland になったのです。

つまりデフォルトデスクトップが GNOME であることは変わっていませんが、正確には

- Stretch までは GNOME on Xorg
- Buster では GNOME on Wayland

となっています。

3.2 Stretch までの im-config

日本語などを入力するには入力メソッドフレームワークと入力メソッドエンジンの設定が必要です。この設定は GTK+ や Qt といった GUI ライブラリ、もしくは XIM (X Input Method) に対して行う必要があり、例えば GTK+ では `GTK_IM_MODULE` といった環境変数が使われています。また、入力を変換するための補助ツールのデーモンや、入力メソッドの状態を表示するためのツールバーなどを起動する必要があります。Debian には入力メソッドフレームワークが uim、ibus、fcitx など複数あり、日本語環境のデフォルトは uim ですがお好みで選ぶことができます。また Debian にはデスクトップも GNOME に限らずたくさんあり自由に選べます。どのデスクトップを選んでも、そしてどの入力メソッドフレームワークを選んでも、インストールしたら何も触らなくてもデフォルトでこのような設定や起動がされるように、シェルスクリプトで書かれた im-config パッケージが自動設定してくれます。自動設定される値はインストールされているパッケージやログインユーザーの使う言語をもとに動的に変わります。

図 4　Stretch のログイン画面

図 5　Buster のログイン画面

なお、お好みで im-config を手動でユーザー毎やシステム全体に設定変更したり、im-config を使わずすべて手動で設定することもできるようになっています。

Stretch までの im-config について詳しくは 2013 年夏号の記事[*4]をご覧ください。当時は Wheezy (7.0) でしたが大きくは変わっていません。

[*4] https://tokyodebian-team.pages.debian.net/pdf2013/debianmeetingresume2013-natsu.pdf

3.3 XIM と GUI ライブラリ

im-config は XIM 関連で大きく 2 つの設定をしてきました。

1 つ目は XIM への設定、`XMODIFIERS` 環境変数です。これは主に XIM で入力を直接扱う（GUI ライブラリを使わない）アプリケーション、例えば xterm など向けの設定ですが、2 つ目の設定とも関連があります。

2 つ目は GUI ライブラリに対する設定です。uim などの入力メソッドフレームワークは GUI ライブラリそれぞれにプラグインを提供する必要があり、それがインストールされていれば GUI ライブラリは直接 uim などと通信できます。ただ、それがインストールされていなくても GUI ライブラリは XIM と通信する機能があります。GTK+ の場合は `GTK_IM_MODULE=xim` とすれば XIM と通信します。1 つ目の設定と組み合わせることで、GTK+ は XIM 経由で uim などと通信することが可能です。

しかし、XIM による入力は特定の条件でフリーズしたりとうまく動かないことがあることが知られていました。

3.4 Stretch までの uim

2017 年冬号の記事 “Debian Stretch のインプットメソッドの現状” [*5]でも取り上げられていましたが、図 1 も見るとわかりますが Stretch の GNOME では uim の状態が画面上で見えません。KDE などほかのデスクトップでも見えない、ないしわかりにくいものがありました。GNOME や KDE などデスクトップなどが変化していったことが原因かとみられます。

3.5 GNOME

Debian は Wheezy で GNOME 3 を取り込みましたが、それより前にあった GNOME 2 とは大きく異なります。GNOME シェルというものを導入し、デスクトップでは標準的なシステムトレイを廃止してトップバー（画面最上部の黒いバー）に置き換えました。そのため今までシステムトレイに表示していた uim の状態は見えなくなってしまいました。そしてトップバーに表示するには GNOME 3 独自の JavaScript API で描画しなければならず対応をむずかしくしています。

さらに、Wheezy には GNOME 3.4、Jessie (8) には GNOME 3.14 を含んでいましたが、バージョン 3.5 頃から GNOME は入力メソッドフレームワークとして ibus を組み込みはじめたようで、GNOME は独自 D-Bus API を ibus 専用に提供して通信しているようで、`GTK_IM_MODULE` 環境変数が設定されていなくても GTK+ で ibus が動作するようです。ibus 以外を使うこともできると記されてはいるものの、ibus-daemon が削除されていれば使えるとされています[*6]。つまり GNOME で uim や fcitx など ibus 以外を使うには、im-config などの設定を切り替えるだけでなく、ibus-daemon というバイナリ（つまり `/usr/bin/ibus-daemon`）を含んでいる ibus パッケージを削除しないと動作保証されない、というモジュール化とは真逆の方向性で残念な状況です。実際、uim や fcitx の動作中に ibus パッケージがインストールされたままだと ibus を使っていないのにトップバーの表示がおかしくなったり、キー入力に ibus が同時に反応したりということも起こったりするようです。

そして先ほど見たとおり、Stretch リリースより後に今までの Xorg に代わり Wayland をデフォルトにしたようです[*7]。GNOME シェルは明らかに Wayland の場合に機能制限しています。GTK+ にはウィンドウを Alt+Tab やタスクバーといったウィンドウ一覧に表示しないようにする設定[*8]ができるのですが、Wayland 上ではこの設定を無視します[*9]。そのため GNOME on Wayland ではツールバーウィンドウが Alt+Tab 対象になってしまい入力しづらくなります。おそらくそんなツールバーなど廃止してしまってトップバー右側にでも GNOME 3 独自の JavaScript API で描画せよという主張なのでしょう。しかしそれでは他のデスクトップが困ります。

Wayland 自体については後にまとめます。

3.6 KDE

Debian は Stretch で KDE 5 を取り込みました。uim にはシステムトレイのほかに KDE 4 向けの Plasma ウィジェットがありますが、KDE 5 では互換性がなく動作しません。

なお KDE 5 には kimpanel という入力メソッドの状態表示用 Plasma ウィジェットが標準で組み込まれており、タスクバーの中にも最初から表示されています。fcitx 開発者が作ったようで、fcitx-module-kimpanel パッケージをインストールすれば fcitx の状態を表示できるようです。fcitx 以外の入力メソッドフレームワークでも kimpanel のプロトコルで通信しさえすれば kimpanel

[*5] `https://tokyodebian-team.pages.debian.net/pdf2017/debianmeetingresume2017-fuyu.pdf`

[*6] `https://wiki.gnome.org/ThreePointFive/Features/IBus#How_to_use_other_IM_frameworks`

[*7] 正確には Stretch に入っている GNOME 3.22 で GNOME 本家は Wayland デフォルトにしたものの、Debian は Xorg デフォルトに戻していたようです (gnome-session 3.21.90-3)。Stretch がリリースされたので本家に合わせて Wayland デフォルトにしたようです。

[*8] 正確にはデスクトップ環境に対してタスクバーに表示しないよう要請する機能です。
`https://developer.gnome.org/gtk3/stable/GtkWindow.html#gtk-window-set-skip-taskbar-hint`

[*9] `https://bugzilla.gnome.org/show_bug.cgi?id=771329`

に状態表示できるようです。さらにこの kimpanel プロトコルを使って GNOME 3 でも状態表示すべく GNOME シェル拡張 gnome-shell-extension-kimpanel パッケージを作ったようです。これを使えば GNOME 3 独自の描画を入力メソッドフレームワークそれぞれが頑張らずに済むかもしれません。

また KDE も Wayland 対応を進めているようで、plasma-workspace-wayland パッケージをインストールすると試すことができます[*10]。

3.7 Wayland – Xorg との違い

Xorg は昔からありますが、Wayland は後発としてディスプレイサーバを一から作り直したようでさまざまな機能をおそらくあえて削っているようです。

- セッションの初期化
  - 環境変数などの設定
  - デーモンなどの起動
- ウィンドウの配置
  - 位置情報の取得・設定
  - 最前面表示など前後関係の設定

Xorg の復習もしながら、それぞれの対策案もまとめます。

3.7.1 セッションの初期化

Debian の Xorg では、X セッションは `/etc/X11/Xsession` など[*11]から呼ばれる `/etc/X11/Xsession.d/` などにあるシェルスクリプトで初期化されます。シェルスクリプトなので環境変数などの設定もデーモンなどの起動もなんでもできます。これらは gdm3 などのディスプレイマネージャや `startx` などのコマンドが X セッションを立ち上げるときに実行していました。なお、`~/.xsessionrc` などこれらのスクリプトが読み込んでくれるスクリプトを自分のホームディレクトリに置くことで自動設定から変更することもできますし、昔ながらの `~/.xsession` など[*12]を置いてカスタマイズすることもできます[*13]。

入力メソッド周りの設定・起動は im-config パッケージのスクリプトが 2 段階に分けて行ってきました。

- 1 段階目（シェル変数 `IM_CONFIG_PHASE` が 1 のとき）は `/etc/X11/Xsession.d/` に置かれている `/etc/X11/Xsession.d/70im-config_launch`[*14]で、環境変数などの設定を行います。
- 2 段階目（`IM_CONFIG_PHASE` が 2 のとき）が `/usr/bin/im-launch` でデスクトップを起動する直前に実行され[*15]、デーモンなどの起動をしてきました。

なお途中で前述の `~/.xsessionrc` なども読まれているので、ホームディレクトリにあるそれらでカスタマイズされていればそれらが尊重されて設定・起動されます。

しかし、Wayland セッションは `/etc/X11/Xsession.d/` を見ないのは当然とはいえ、何も初期化を扱いません。gdm3 や KDE と同時にインストールされる sddm といったディスプレイマネージャは X セッションと Wayland セッション両方へのログインに対応していますが、どれも X セッションを立ち上げるときだけ上のような共通の初期化を行うようになっており、Wayland セッションでは何もしていません。なお `startx` のようにディスプレイマネージャを使わずコマンドラインから Wayland セッションを起動する汎用の方法も用意しないようです。各デスクトップの Wayland セッションを独自にコマンドライン起動することはできるようですが[*16]、その場合もそのデスクトップ独自の初期化以外はされません。そのため何らかのほかの汎用的な方法で初期化する必要があります。一方で “汎用的な方法” を単に採用すると、汎用的である以上 Wayland セッションだけでなく X セッションでもその初期化が実行されてしまう、つまり X セッションでは二重に初期化されてしまうことになるので、

- 二重に初期化されても（ほぼ）同じ結果になるようにする
- 今までの X セッションの方法を捨てて全部新たな方法で初期化する

など何らかの方法で対処しなければなりませんが、後者はほぼ完全互換でない限り困る人が多く非現実的です。

■環境変数などの設定 “汎用的な方法” はいくつか考えられますが、それぞれ課題があるので見ていきます。

- `/etc/profile.d/` にシェルスクリプト (`*.sh`) を置く
  このディレクトリは `/etc/profile` から読み込まれます。この `/etc/profile` は本来（コマンドライン）ログインシェル起動時に読み込まれるものですが、gdm3 や sddm では独自に読み込んでいます。そのため、これらのディスプレイマ

[*10] 後述しますが現時点ではこの上で起動されるアプリケーションはデフォルト XWayland で動作しているようです。
[*11] Debian の gdm3 では `/etc/X11/Xsession` に似せた `/etc/gdm3/Xsession` が使われています。
[*12] これらも結局は `/etc/X11/Xsession.d/` から起動されています。
[*13] `https://www.debian.org/doc/manuals/debian-reference/ch07.ja.html#_starting_the_x_window_system`
[*14] 70 という大きめの番号にすることで、これより前で `~/.xsessionrc` などのカスタマイズファイルが読まれるようにしてあります。
[*15] このタイミングで実行されるように `/etc/X11/Xsession.d/70im-config_launch` が仕込んであります。
[*16] GNOME on Wayland セッションは `env XDG_SESSION_TYPE=wayland dbus-run-session gnome-session` で起動できるようです。

ネージャではここに置けば設定はされます[*17]が、あらゆるコマンドライン端末などで余計な環境変数設定作業が行われてしまうことになるので避けられるなら避けたいところです。なお、Xfce などほかのデスクトップと同時にインストールされる lightdm などのディスプレイマネージャでは（少なくとも Buster 時点では）読み込まれないので、今までの X セッションの方法を完全に置き換えるまではできません。

- `systemd --user` 独自空間の変数を設定する
  Debian では Jessie から systemd が標準の init システムです。systemd ではユーザー毎にログイン直後に `systemd --user` というプロセスが立ち上がり、その内部で環境変数とは別に独自の変数を持っています[*18]。Stretch リリースよりも後の最近の systemd（バージョン 233 以降）ではこの変数を静的・動的に設定できるようになりました。`/etc/environment.d/*.conf` や `~/.config/environment.d/*.conf` に `KEY=VALUE` と書いておくと、`KEY` という変数名に `VALUE` という値が設定されます。動的に設定するには、`/usr/lib/systemd/user-environment-generators/` 以下に `KEY=VALUE` といった行を生成して出力するプログラムをシェルスクリプトでもなんでも置くことでできます。
  さて、この独自変数を設定したところでこれは環境変数ではありません[*19]。ここで現状 gdm3 だけはこの独自変数の値を環境変数として取り込むようになっており、デフォルトデスクトップとともにインストールされる gdm3[*20]に限っていえば環境変数の動的設定に使えそうです。それ以外では使えないですから当然今までの X セッションの方法を完全に置き換えるまではできません[*21]。
  また、この `systemd --user` プロセスはデスクトップセッションとは独立しており、例えば同じユーザーでコンソールログインしたりするとそのプロセスは共有されます。そのためデスクトップをログアウトしても終了しないことがあります。また、同じユーザーがすべてログアウトしたとしても少なくともしばらくの間は生き続けるようです。つまり設定を変更して再ログインしてもプロセスが変更前の設定を覚えたままになっていることがあるので、他人と同時利用しない PC であればログアウトだけでなく再起動するのが確実です。
  そしてこのプロセスはデスクトップセッションと独立していることから、設定時にはどのデスクトップでログインしようとしたのかすらわかりません。Xorg なのか Wayland なのか、はては SSH ログインなのかも区別できないので、今までの X セッションの方法を維持するなら Xorg 上で二重設定を回避できません。
  なお今までの X セッションの方法にも悪影響があります。もともと X セッションの方法ではほぼまっさらの未設定の状態で設定を行っていました。しかし、この `systemd --user` プロセスはログイン直後の非常に早いタイミングで起動されるので、この初期化よりも後に今までの X セッションの方法による初期化が行われることになり、既に `systemd --user` により設定済みの上に `~/.xsessionrc` などユーザーによるカスタマイズも含む X セッションの方法による初期化を行うことになってしまいます。何らかの対処が必要です。
- `pam_env(7)` で使われる変数を設定する
  `pam_env.so` は PAM (Pluggable Authentication Module) という認証フレームワークのモジュールで、ディスプレイマネージャ、コンソール、SSH サーバなどでのログイン認証時に環境変数を設定してくれるものです。`/etc/environment` などに `KEY=VALUE` と書いておくと、このモジュールから読み込まれれば環境変数 `KEY` が `VALUE` に設定されます。しかしこのモジュールはおそらくセキュリティ上の懸念からか Debian ではデフォルトではあまり使わないようになっており、ホームディレクトリに `~/.pam_environment` を置けば読み込んでくれる機能もあるのですがデフォルトでは sddm でのみ使われています[*22]。また、これらの設定ファイルには静的な決まった値しか書けません。設定を動的に変える機能がほぼないため[*23]im-config には使えません。`pam_env.so` を改造した動的に入力メソッド周りを設定するモジュールを作ろうと思えば作れるでしょうが、そこまで複雑なことを認証フレームワークのモジュールでやるのはやり過ぎでしょう。
- （参考）ディスプレイマネージャやデスクトップ独自の設定ファイルを使う
  例えば gdm3 は `/etc/xprofile` や `~/.xprofile` などを、GNOME は `~/.gnomerc` を独自に読み込むようで、ここに挙げたものはどれもシェルスクリプトなので動的に設定できるようです。ただどれも汎用的でないので可能な限り避けたいところです。

■デーモンなどの起動　こちらの “汎用的な方法” は 1 つくらいしかありません。デスクトップ環境に共通の仕様を提供している freedesktop.org という団体が定める XDG Autostart[*24]という仕組みで、現代的なデスクトップ環境（GNOME、

[*17] 実際 Ubuntu の im-config は Wayland でなんとか動かすため`/etc/profile.d/input-method-config.sh` を仮置きしていたようです。
[*18] `systemctl --user show-environment` で中身が見られます。
[*19] `https://github.com/systemd/systemd/pull/3904` で当初は環境変数を設定する機能にしたかったようですが、セキュリティ上の懸念から独自変数を設定するだけの機能として systemd 233 に追加されたようです。
[*20] なお gdm3 は `systemd --user` を使うために systemd に強く依存しているので、sysvinit など systemd 以外の init では使えません。
[*21] sysvinit など systemd 以外の init 上で systemd-logind 相当の機能を使えるようにする elogind パッケージがありますが、`systemd --user` 相当の機能はなく、提供の予定もありません。
[*22] `/etc/pam.d/` 内の設定ファイルで `pam_env.so` の行に `user_readenv=1` と書く必要があります。Ubuntu の gdm3 では有効になっているようです。
[*23] `pam_env.conf(5)` で他の変数からの展開ができるくらいです。
[*24] `https://standards.freedesktop.org/autostart-spec/autostart-spec-latest.html`

KDE、Xfce など）は対応しており自動起動設定に使われています。`.desktop` という拡張子の “デスクトップファイル”[*25]を `/etc/xdg/autostart/` や `~/.config/autostart/` に置いておくと、XDG Autostart 対応デスクトップのログイン直後に起動されます。なお非対応のデスクトップでは使えないので、やはりこれで今までの X セッションの方法を完全に置き換えるまではできません。なお今までの X セッションの方法を維持するなら Xorg 上で二重起動しないようにする必要があります。

3.7.2 ウィンドウの配置

■位置情報の取得・設定 Xorg ではウィンドウなどの位置情報を取得したり、指定した位置にウィンドウを配置することができました。しかし Wayland はそのような機能を提供しません。今あるカーソルの位置もわからないので、入力メソッドフレームワークが変換候補を表示しようとしてもカーソルの近くに出せません。ツールバーウィンドウを右下の決まった位置に置いたりすることもできません。

■最前面表示など前後関係の設定 Xorg ではウィンドウを最前面に固定したりすることができましたが、Wayland 自体にはそのような機能がありません。そのためツールバーウィンドウを最前面にできないので、例えばエディタのウィンドウを画面全体に最大化するとツールバーは隠れてしまい状態が見えなくなってしまいます。

■対策案 どちらも解決するために次のような方法が考えられます。

- Wayland コンポジター（GNOME シェル[*26]など）の独自 API で実装する
  Wayland ではコンポジターというディスプレイサーバーが表示します。コンポジターはさまざまなアプリケーションの複数のウィンドウを適切に配置したり最前面にしたりしていますし、ウィンドウのメニューから手動で指定さえできるようにしてあるわけなので、Wayland 自体に API がなくてもコンポジターが API を提供していればそれを使えるはずです[*27]。ibus は GNOME に組み込まれているためこの方法を使っていますが、ほかの入力メソッドフレームワーク向けに GNOME がこの API を公開するつもりはなさそうなので、この方法は現状 ibus 以外では使えません。
- XWayland を使う
  Wayland コンポジター内で XWayland という X サーバを動かすことができます。この XWayland の中では普通の X アプリケーションを動かすことができ、普通の Xorg のように位置の取得や設定、最前面設定などができるようです。なお普通の Xorg と同様なので、GNOME のタスクバー非表示も機能します。GTK+ や Qt はこの切り替えに対応しており、環境変数さえ設定すれば簡単に切り替えられるようです。GTK+ アプリケーションの場合は `env GDK_BACKEND=x11` でプログラムを起動すれば XWayland 内で表示されます[*28]。

3.8 Buster の im-config

ここまで見てきた対策案を踏まえて、Buster の im-config (0.43-1) では次のようにすることにしました。

まず、結局今までの X セッションの方法を完全に置き換えられるような互換性高い方法は存在しないので、Xorg では今まで通り設定や起動をすることにします。

そして Wayland（Xorg 以外）向けには `systemd --user` の独自変数を設定し[*29]、XDG Autostart を使って Wayland デスクトップのときだけ[*30]起動することにします[*31]。しかしこのままだと Xorg で二重設定になってしまいます。また、ユーザーがカスタマイズしたスクリプトでデーモンを起動していたりすると二重起動になってしまいます。二重設定は諦めざるを得ませんが、ユーザーがカスタマイズした設定を尊重して可能な限り二重起動されないように試みます。具体的には、`systemd --user` が最初に起動されるのは回避不可能なので、それより後に実行される X セッションでの起動時に

- （`systemd --user` や `~/.xessionrc` などにより）先に設定された設定値
- まっさらな未設定にした状態で、今までの im-config で初期化し、それによる設定値

の 2 つを比較することで、少なくともユーザーがデフォルトとは異なる値にカスタマイズしようとしていたかを判定します[*32]。異なる値が見つかった場合は明らかにカスタマイズされているということなので im-config での起動を行いません。ただ、見つからなかった場合でもたまたま im-config が設定しようとした値と同じ値をユーザーがカスタマイズした設定ファイルで設定していたかもしれませんが、これを知る方法はないのでそのような場合は二重起動になります。自分で入力メソッド周りの設定スクリプトを書いている方は、二重起動などに遭遇したら一度自分の設定を削ってみて試していただきたいです。

[*25] これも freedesktop.org が定めています。`https://specifications.freedesktop.org/desktop-entry-spec/latest/`

[*26] 正確には mutter

[*27] ただし標準にならない限りコンポジター（つまりデスクトップ）毎に独自 API ばかりになってしまうでしょう。

[*28] Qt アプリケーションの場合は `env QT_QPA_PLATFORM=xcb` ですが指定していなくても現状は Xorg (XWayland) が標準のようで、Wayland にしたいときは `env QT_QPA_PLATFORM=wayland` とします。

[*29] `/usr/lib/systemd/user-environment-generators/70-im-config`

[*30] XDG Autostart のデスクトップファイル内ではどんなディスプレイサーバやデスクトップで実行されているかの情報が環境変数で取得できます。Wayland では `XDG_SESSION_TYPE` 環境変数が `wayland` になります。

[*31] `/etc/xdg/autostart/im-launch.desktop`

[*32] `/usr/bin/im-launch` の `IM_CONFIG_CHECK_ENV` の部分でこのようなちょっと複雑なことをしています。

また、XWayland でツールバーなどを起動することにします。今のところ uim のツールバーは XWayland を使うようにしました[*33]。

また、GNOME は ibus-daemon が存在すると起動していなくても衝突することから、残念ながら ibus-daemon が存在するときは Xorg 以外の環境の im-config は何も設定・起動しないようにしました。つまり Wayland で ibus パッケージがインストールされていると im-config は動作しないようになります。なおこれは本当は GNOME の問題ですが、`systemd --user` の設定時はデスクトップの種類がまったくわからないため、そして今のところ Xorg 以外でまともに動作するデスクトップは GNOME だけなので、Xorg 以外では常に何もしないようにしてあります[*34]。

また、XIM では入力に問題が起きることがあることから、GUI ライブラリが XIM 経由で入力メソッドフレームワークと通信するのを止めました。つまり入力メソッドフレームワークの GUI ライブラリプラグインをすべてインストールしていないと、未インストールの GUI ライブラリを使ったアプリでは入力できなくなります[*35]。なおこれは、`systemd --user` はディスプレイマネージャであっても SSH サーバであってもあらゆるログイン時に起動されるため、im-config の設定に時間が掛かりすぎて問題になってしまったための回避策でもあります[*36]。

何か問題を見つけたらぜひ im-config パッケージにバグレポートをお願いします[*37]。

3.9 Buster の uim

Stretch ではデフォルトで状態表示できていなかったので、Buster の uim は図 2 のようにデフォルトでツールバーを表示するよう改修されました。なお、Stretch リリースより後に uim は Wayland 対応されたのですが、GNOME `<del>`による嫌がらせ`</del>` や Wayland 自体のせいで入力するのが難しい状態なので、im-config から起動されるときに `env GDK_BACKEND=x11` を設定することで常に Xorg ないし XWayland で動作するようにしてあります。

3.10 今後

いくつも課題があります。

- 例えば KDE Plasma Wayland では現状日本語入力できません。デフォルトになってしまった GNOME on Wayland に限っては systemd かつ gdm3 で日本語入力できるようにしましたが、KDE ともにインストールされる sddm などが何らかの汎用的な方法で環境変数などを設定できるようになる必要があります。本当は freedesktop.org などの団体が特定のデスクトップや init システムなどに依存しない設定のための標準仕様を提供してくれるのが望ましいです[*38]。
- 最近では flatpak[*39]や snap[*40]といったコンテナでアプリケーションをパッケージ化する技術が増えてきていますが、日本語入力ができません。これはホスト側（手元のマシン）にある入力メソッドのデーモンなどとゲスト側（コンテナ内）にあるアプリケーションが通信できないからです。最近の flatpak では ibus が GNOME との間で使っている D-Bus 通信をできるようにコンテナに穴を開けているらしいですが、ibus 以外の入力メソッドフレームワークも ibus と同じ D-Bus API で通信しなければならないという流れでしょうか。しかしその API を GNOME は公開するつもりがありません。困りました。ただ、今までは GNOME と ibus だけで知っていた API を flatpak プロジェクトにも教えたということであれば、ほかで同じ API を勝手に使ったとしてもそんなに頻繁には API が変更されないことは期待できそうな気がします。
- GTK+4 など新しい GUI ライブラリが出てきたら対応はいつも必要です。GTK+4 については GTK+3 までと同じ環境変数 `GTK_IM_MODULE` が使えるようです。なおこれは GUI ライブラリに限らず Wayland などのディスプレイサーバやコンテナについても同じことが言えます。
- そもそも環境変数を使っているとコンテナなどいろいろな環境への情報伝搬が難しくなっています。ibus と GNOME との独自 D-Bus API が最良なのかはわかりませんが、他の標準的な方法でも通信できるようになると良いのかもしれません[*41]。

[*33] 実は fcitx は調べきれてませんが何らかの方法で問題を回避してツールバーを表示しているようです。

[*34] GNOME の問題なので実は GNOME on Xorg でも発生し得る問題ですが、互換性のため Xorg での挙動は変えていません。

[*35] uim を使うには uim パッケージを推奨パッケージ (Recommends) まですべてインストールするようにしてください。なお Debian インストーラで普通にインストールすればこの状態になります。

[*36] `https://bugs.debian.org/925160`

[*37] もちろん勉強会や Twitter、メーリングリストなどで教えていただいても構いません。

[*38] Wayland は何もないから困っているわけですが、実は Xorg でも Debian は `/etc/X11/Xsession.d/` を使っているものの、設定方法はある程度似通った部分はあるもののディストリビューション毎にまちまちだったりします。

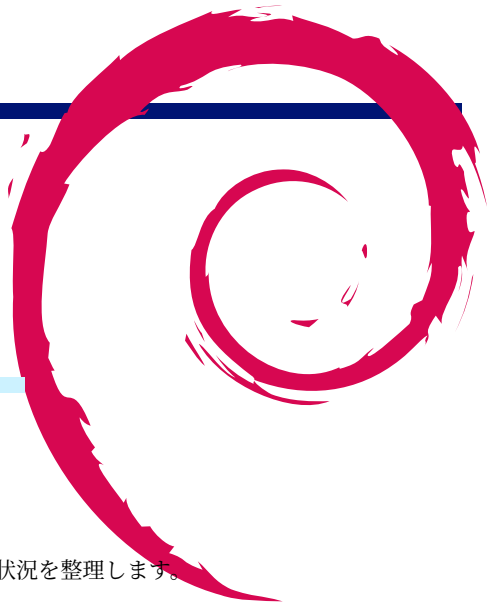
[*39] flatpak という Debian パッケージをインストールすると flatpak パッケージを扱えるようにはなります。

[*40] Ubuntu はデフォルトですが、snapd という Debian パッケージをインストールすると snap パッケージを扱えるようにはなります。

[*41] もともと XIM も共通プロトコルだったような気はしますが...

3.11 まとめ

Buster では GNOME on Wayland がデフォルトデスクトップになりましたが、im-config に手を入れてなんとか日本語入力できるようにしました。また uim の課題もいくらか修正されています。Debian はさまざまなデスクトップや入力メソッドフレームワークを自由に選べます。im-config は、どのデスクトップを選んでも、そしてどの入力メソッドフレームワークを選んでも、インストールしたら何も触らなくてもデフォルトで設定・起動してくれる支援パッケージです。Wayland では標準的な環境設定方法があまりないため、今回デフォルトデスクトップである systemd + gdm3 + GNOME Wayland の組み合わせについては `systemd --user` の独自変数と XDG Autostart を使って動作するようにしました。KDE Plasma Wayland などの Wayland 環境や、flatpak、snap といったコンテナ環境、今後出てくるだろう GUI ライブラリなどでも自由に選んでそのままで使える状態にするのが課題です。



4 /usr Merge について

yy_y_ja_jp

Debian では Buster から “/usr Merge” がデフォルトになりそうです。概要と状況を整理します。

4.1 概要

“/usr Merge” とは、**/bin/**, **/sbin/**, **/lib/** などの中にあるもの（バイナリ等）を **/usr/** 内に移す、つまり **/usr/** 内にバイナリ等をすべて統合することです。例えば **/bin/ls** を **/usr/bin/ls** に移します。また、今まで通り **/bin/ls** でも使えるように互換性のため **/bin** は **/usr/bin** へのシンボリックリンクにします。**/sbin** なども同様です。

なお、このような構成にしたい理由は systemd の wiki[*42]などにまとめられているようです[*43]。

Debian を新規インストールするときには、Debian インストーラが内部で実行する debootstrap がこの構成を行います。既存の Debian 環境には usrmerge パッケージをインストールすればこの構成にしてくれます[*44]。なお、以前の構成へ元に戻すことは困難です。

Stretch インストーラで作った Debian 環境の / 直下は次のようなディレクトリ構成でした。

```
$ ls -l /
合計 76
drwxr-xr-x  2 root root  4096  5 月 14 00:42 bin
drwxr-xr-x  3 root root  4096  5 月 14 00:43 boot
drwxr-xr-x 17 root root  2980  5 月 14 00:56 dev
drwxr-xr-x 76 root root  4096  5 月 14 00:56 etc
drwxr-xr-x  3 root root  4096  5 月 14 00:43 home
lrwxrwxrwx  1 root root    29  5 月 14 00:38 initrd.img -> boot/initrd.img-4.9.0-9-amd64
lrwxrwxrwx  1 root root    29  5 月 14 00:38 initrd.img.old -> boot/initrd.img-4.9.0-9-amd64
drwxr-xr-x 15 root root  4096  5 月 14 00:42 lib
drwxr-xr-x  2 root root  4096  5 月 14 00:36 lib64
drwx------  2 root root 16384  5 月 14 00:35 lost+found
drwxr-xr-x  3 root root  4096  5 月 14 00:35 media
drwxr-xr-x  2 root root  4096  5 月 14 00:35 mnt
drwxr-xr-x  2 root root  4096  5 月 14 00:35 opt
dr-xr-xr-x 80 root root     0  5 月 14  2019 proc
drwx------  2 root root  4096  5 月 14 00:35 root
drwxr-xr-x 14 root root   460  5 月 14 00:57 run
drwxr-xr-x  2 root root  4096  5 月 14 00:56 sbin
drwxr-xr-x  2 root root  4096  5 月 14 00:35 srv
dr-xr-xr-x 13 root root     0  5 月 14 00:57 sys
drwxrwxrwt  8 root root  4096  5 月 14 00:56 tmp
drwxr-xr-x 10 root root  4096  5 月 14 00:35 usr
drwxr-xr-x 11 root root  4096  5 月 14 00:35 var
lrwxrwxrwx  1 root root    26  5 月 14 00:38 vmlinuz -> boot/vmlinuz-4.9.0-9-amd64
lrwxrwxrwx  1 root root    26  5 月 14 00:38 vmlinuz.old -> boot/vmlinuz-4.9.0-9-amd64
$
```

一方、Buster インストーラで作った Debian 環境では次のようになります。なお、現時点ではインストーラに “/usr Merge” しないようにする設定はなさそうです[*45]。

[*42] https://www.freedesktop.org/wiki/Software/systemd/TheCaseForTheUsrMerge/

[*43] Debian wiki https://wiki.debian.org/UsrMerge

[*44] Buster の次のバージョン Bullseye ではインストール推奨となるかもしれません。https://bugs.debian.org/841666

[*45] https://bugs.debian.org/923091

```
$ ls -l /
合計 60
lrwxrwxrwx  1 root root     7  5月 14 00:36 bin -> usr/bin
drwxr-xr-x  3 root root  4096  5月 14 00:43 boot
drwxr-xr-x 17 root root  3160  5月 14 00:56 dev
drwxr-xr-x 67 root root  4096  5月 14 00:56 etc
drwxr-xr-x  3 root root  4096  5月 14 00:43 home
lrwxrwxrwx  1 root root    30  5月 14 00:38 initrd.img -> boot/initrd.img-4.19.0-4-amd64
lrwxrwxrwx  1 root root    30  5月 14 00:38 initrd.img.old -> boot/initrd.img-4.19.0-4-amd64
lrwxrwxrwx  1 root root     7  5月 14 00:36 lib -> usr/lib
lrwxrwxrwx  1 root root     9  5月 14 00:36 lib32 -> usr/lib32
lrwxrwxrwx  1 root root     9  5月 14 00:36 lib64 -> usr/lib64
lrwxrwxrwx  1 root root    10  5月 14 00:36 libx32 -> usr/libx32
drwx------  2 root root 16384  5月 14 00:36 lost+found
drwxr-xr-x  3 root root  4096  5月 14 00:36 media
drwxr-xr-x  2 root root  4096  5月 14 00:36 mnt
drwxr-xr-x  2 root root  4096  5月 14 00:36 opt
dr-xr-xr-x 79 root root     0  5月 14  2019 proc
drwx------  2 root root  4096  5月 14 00:36 root
drwxr-xr-x 15 root root   460  5月 14 00:58 run
lrwxrwxrwx  1 root root     8  5月 14 00:36 sbin -> usr/sbin
drwxr-xr-x  2 root root  4096  5月 14 00:36 srv
dr-xr-xr-x 13 root root     0  5月 14 00:56 sys
drwxrwxrwt  8 root root  4096  5月 14 00:56 tmp
drwxr-xr-x 13 root root  4096  5月 14 00:36 usr
drwxr-xr-x 11 root root  4096  5月 14 00:36 var
lrwxrwxrwx  1 root root    27  5月 14 00:38 vmlinuz -> boot/vmlinuz-4.19.0-4-amd64
lrwxrwxrwx  1 root root    27  5月 14 00:38 vmlinuz.old -> boot/vmlinuz-4.19.0-4-amd64
$
```

4.2 経緯

“/usr Merge” は以前 Fedora で特に行われていたようです[*46]。なお Fedora は必須化したようです。

Debian では当初 Stretch でデフォルトにする予定でしたが、問題の解決が間に合いそうになく延期されました[*47]。

その後 Buster に向けて debootstrap 1.0.102 で再度デフォルトにすることにしました[*48]が、問題があるためデフォルトでなくすべきではという議論があり、Debian 技術委員会 (tech-ctte) にも持ち込まれました[*49]。

Debian 技術委員会は、debootstrap のメンテナーが決めたことは覆さない、と決定しました[*50]。同時に、Debian における “/usr Merge” の現状と今後の望ましい姿を示しました。

4.3 現状

Debian 技術委員会の決定により、debootstrap メンテナー次第で “/usr Merge” がデフォルトになるかどうかは決まります。現時点で testing (Buster) 以降のバージョン向けに debootstrap を実行すると次のようになっており、メンテナーが変更しなければこのまま Buster リリースとなります。

- 通常の環境では “/usr Merge” がデフォルト有効（無効にするには `--no-merged-usr` オプションを付ける）
- パッケージビルド用の環境（`--variant=buildd` オプション）ではデフォルト無効

なお、現状では “/usr Merge” された環境でビルドされたバイナリパッケージを “/usr Merge” されていない環境に持っていくと動作しないものがあり、それらのパッケージに随時バグ報告してくれている方がいるようです[*51] 。よくある理由としては、ビルド中にバイナリの場所を `/bin/` の中よりも先に `/usr/bin/` を探しに行き、“/usr Merge” された環境では `/usr/bin/` のパスをコンパイル結果に埋め込むからです。例えば quilt パッケージの少し古い修正[*52]以前のバージョン 0.65-2 を “/usr Merge” された環境でビルドしてみます。`dh_auto_configure` 経由で実行された `./configure` が `/bin/bash` ではなく `/usr/bin/bash`、`/bin/cp` ではなく `/usr/bin/cp`、などを検出してビルドしています。この場合はビルドされたパッケージ内のシェルスクリプトの先頭が `#! /usr/bin/bash` となったため lintian がたまたま別件のポリシー違反として検出してくれています。

[*46] https://fedoraproject.org/wiki/Features/UsrMove
[*47] https://lists.debian.org/debian-devel-announce/2017/01/msg00004.html
[*48] https://bugs.debian.org/839046
[*49] https://bugs.debian.org/914897
[*50] https://lists.debian.org/debian-devel-announce/2019/03/msg00001.html
[*51] https://bugs.debian.org/cgi-bin/pkgreport.cgi?tag=usrmerge;users=md@linux.it
[*52] https://bugs.debian.org/913226

```
$ dget http://snapshot.debian.org/archive/debian/20180809T030926Z/pool/main/q/quilt/quilt_0.65-2.dsc
dget: retrieving http://snapshot.debian.org/archive/debian/20180809T030926Z/pool/main/q/quilt/quilt_0.65-2.dsc
(snip)
quilt_0.65-2.dsc:
      Good signature found
   validating quilt_0.65.orig.tar.gz
   validating quilt_0.65-2.debian.tar.xz
All files validated successfully.
dpkg-source: info: extracting quilt in quilt-0.65
(snip)
dpkg-source: info: applying fix-mail-threading
$ cd quilt-0.65/
$ debuild -us -uc
 dpkg-buildpackage -us -uc -ui
dpkg-buildpackage: info: source package quilt
dpkg-buildpackage: info: source version 0.65-2
(snip)
dh_auto_configure -- --with-docdir=/usr/share/doc/quilt --with-sendmail=/usr/sbin/sendmail --with-awk=/usr/bin/awk
  ./configure --build=x86_64-linux-gnu --prefix=/usr --includedir=\${prefix}/include --mandir=\${prefix}/share/man
  --infodir=\${prefix}/share/info --sysconfdir=/etc --localstatedir=/var --disable-silent-rules
  --libdir=\${prefix}/lib/x86_64-linux-gnu --libexecdir=\${prefix}/lib/x86_64-linux-gnu --runstatedir=/run
  --disable-maintainer-mode --disable-dependency-tracking --with-docdir=/usr/share/doc/quilt
  --with-sendmail=/usr/sbin/sendmail --with-awk=/usr/bin/awk
configure: WARNING: unrecognized options: --disable-silent-rules, --disable-maintainer-mode, --disable-dependency-tracking,
  --with-docdir
checking for a BSD-compatible install... /usr/bin/install -c
checking whether #! works in shell scripts... yes
checking for bash... /usr/bin/bash
checking whether /usr/bin/bash quoting works... yes
checking for gcp... no
checking for cp... /usr/bin/cp
checking for gdate... no
checking for date... /usr/bin/date
(snip)
dpkg-deb: building package 'quilt' in '../quilt_0.65-2_all.deb'.
dpkg-deb: building package 'quilt-el' in '../quilt-el_0.65-2_all.deb'.
 dpkg-genbuildinfo
 dpkg-genchanges  >../quilt_0.65-2_amd64.changes
dpkg-genchanges: info: not including original source code in upload
 dpkg-source --after-build .
dpkg-source: warning: Testsuite field contains value autopkgtest, but no tests control file debian/tests/control
dpkg-buildpackage: info: binary and diff upload (original source NOT included)
Now running lintian quilt_0.65-2_amd64.changes ...
W: quilt source: orig-tarball-missing-upstream-signature quilt_0.65.orig.tar.gz
E: quilt: missing-depends-on-sensible-utils usr/share/quilt/edit
E: quilt: missing-depends-on-sensible-utils usr/share/quilt/header
E: quilt: missing-depends-on-sensible-utils usr/share/quilt/mail
E: quilt: wrong-path-for-interpreter usr/bin/quilt (#!/usr/bin/bash != /bin/bash)
E: quilt: wrong-path-for-interpreter usr/share/quilt/add (#!/usr/bin/bash != /bin/bash)
(snip)
Finished running lintian.
$
```

問題を回避するため、少なくとも Debian 公式のパッケージビルドには当面 “/usr Merge” されていない環境を使うこととなりました。また、Debian 公式レポジトリにアップロードされたパッケージは、“/usr Merge” されていない環境でビルドした結果と “/usr Merge” された環境でビルドした結果を比較してこのような差分が発生していないか Reproducible Builds プロジェクトの CI[*53]環境で検証も行われており[*54]、これも元にバグ報告・修正が行われています。

手元で動く pbuilder, cowbuilder や sbuild などクリーンルームでのパッケージビルドツールも同様に “/usr Merge” されていない環境が作られてその中でビルドしてくれます。新たに cowbuilder でビルド環境を作ると次のように構成されます。

```
# cowbuilder --create
I: Invoking pbuilder
I: forking: pbuilder create --buildplace /var/cache/pbuilder/base.cow --mirror http://ftp.jp.debian.org/debian
  --distribution sid --no-targz --extrapackages cowdancer
W: /root/.pbuilderrc does not exist
I: Running in no-targz mode
(略)
Processing triggers for libc-bin (2.28-10) ...
I: Copying back the cached apt archive contents
I: unmounting dev/ptmx filesystem
I: unmounting dev/pts filesystem
I: unmounting dev/shm filesystem
I: unmounting proc filesystem
I: unmounting sys filesystem
#
```

[*53] Continuous Integration、継続的インテグレーション

[*54] https://tests.reproducible-builds.org/debian/issues/unstable/paths_vary_due_to_usrmerge_issue.html

```
# ls -l /var/cache/pbuilder/base.cow/
合計 80
drwxr-xr-x  2 root root 4096  5月 14 23:50 bin
drwxr-xr-x  2 root root 4096  5月 14 05:25 boot
drwxr-xr-x  2 root root 4096  5月 14 23:50 build
drwxr-xr-x  4 root root 4096  5月 14 23:50 dev
drwxr-xr-x 30 root root 4096  5月 14 23:50 etc
drwxr-xr-x  2 root root 4096  5月 14 05:25 home
drwxr-xr-x  7 root root 4096  5月 14 23:50 lib
drwxr-xr-x  2 root root 4096  5月 14 23:50 lib64
drwxr-xr-x  2 root root 4096  5月 14 23:50 media
drwxr-xr-x  2 root root 4096  5月 14 23:50 mnt
drwxr-xr-x  2 root root 4096  5月 14 23:50 opt
drwxr-xr-x  2 root root 4096  5月 14 05:25 proc
drwx------  2 root root 4096  5月 14 23:50 root
drwxr-xr-x  4 root root 4096  5月 14 23:50 run
drwxr-xr-x  2 root root 4096  5月 14 23:50 sbin
drwxr-xr-x  2 root root 4096  5月 14 23:50 srv
drwxr-xr-x  2 root root 4096  5月 14 05:25 sys
drwxrwxrwt  2 root root 4096  5月 14 23:50 tmp
drwxr-xr-x 10 root root 4096  5月 14 23:50 usr
drwxr-xr-x 11 root root 4096  5月 14 23:50 var
#
```

パッケージを作るときは安全のためこれらのツールでビルドするようにしましょう。

なお、多くの場合 PATH 環境変数に従ってバイナリの場所を探しているので /bin/ よりも /usr/bin/ が優先されています。なのでもともと “/usr Merge” されていない環境であってもクリーンルームでビルドしていないと次のようなことも起こります。/usr/local/bin/bash が存在する環境で dpkg-buildpackage コマンドを実行すると、PATH 環境変数では /bin（や /usr/bin）よりも /usr/local/bin が優先されるので /bin/bash ではなく /usr/local/bin/bash が使われてしまいます。なおこのように /usr/local/bin/ の中のものを使うのは明らかにポリシー違反なので一部ツールでは対策されており、dpkg-buildpackage コマンドの代わりに devscripts パッケージにある debuild コマンドを使えば回避できます。これは PATH 環境変数に /usr/local/bin を除いたものを設定した上でビルドしてくれます。ですがまずは先ほどのクリーンルームなビルドツールを使うのが良いでしょう。

また、“/usr Merge” された環境（に限らないですが）の既知の注意点として、dpkg --search (dpkg -S) が指定したファイルのパッケージを見つけてくれないことがあります[*55]。

```
$ which ls
/usr/bin/ls
$ dpkg -S /usr/bin/ls
dpkg-query: パターン /usr/bin/ls に一致するパスが見つかりません
$ dpkg -S /bin/ls
coreutils: /bin/ls
$
```

なお、おそらくこの挙動が変更されることはなさそうです。debhelper をはじめ様々なツールも現在の挙動に依存しています。ディレクトリの対応付けなどを認識しながらファイルを検索するオプションは別に用意されるかもしれません。

4.4 今後

Debian 技術委員会は今後 Debian の望ましい姿として、Buster の次のバージョン (Bullseye) では “/usr Merge” されていない環境とされている環境の両方が使えるべきで、またどちらの環境でビルドされたパッケージも両方の環境で動作すべきと示しました。

先ほどの quilt は、その次のバージョン 0.65-3 で “/usr Merge” された環境でビルドされたパッケージが “/usr Merge” されていない環境に持っていっても動作するよう次のように修正されました。debian/rules ファイルの dh_auto_configure コマンド（結果的には ./configure）呼び出しの引数最後に --with-bash=/bin/bash を追加し、PATH 環境変数に関係なく決め打ちで /bin/bash を見に行くようにしました。なお ./configure は --with-bash= を BASH= と書くこともできます。今のところこのように差分が見つかった各パッケージを修正中のようです。新しくパッケージングしたいアプリにも少なくとも差分があればこのような対処が必要になります。

さて、現時点でこれをどうシームレスに実現するのかは不明です。少なくとも Debian 公式レポジトリにアップロードされた後に検出する CI 環境は既にありますが、新しいアプリをパッケージングするときに事前に対応できるようにならないと真に実現したことにならないでしょう。そのためには、debhelper 等のパッケージング時のビルド補助ツールが頑張り、lintian 等のパッケージング後の検査ツールも強化していく感じでしょうか。PATH 環境変数で /bin よりも /usr/bin を優先させることもできるかもしれませんが悪影響が読めません。現時点でこの問題の自衛方法としては Reproducible Builds の取り組みで使われているツールを手元で動かすくらいでしょう。diffoscope で 2 つのファイルをバイナリの中身まで再帰的に差分表示してくれます[*56]。ただ 2

[*55] https://bugs.debian.org/858331

[*56] x11-apps で発生している差分の表示例 https://tests.reproducible-builds.org/debian/dbd/buster/amd64/x11-apps_7.7+7.diffoscope.html。表示に使った diffoscope コマンド引数はこの HTML ページのタイトル (<title>) に書かれています。

つの環境作って両方でビルドするのも、そしてこの “/usr Merge” で発生している差分を見極めるのも正直面倒なので[*57]、まずはクリーンルームでビルドして上げておいて、後で CI 環境などに指摘されたら直すくらいが現実的ではないかと思います。

[*57] 完全な Reproducible Builds は本当は実現すべきですが大変なので、今回発生する差分だけでも対処するのが現実的です。

5 Systemd に入門してみた

あっきー@papi_tokei

(発表資料から編集者が適当に変換したので見にくいところはご容赦ください。)

5.1 なぜ今頃 systemd？

使用している Openblocks EX1 のファームのバージョンが 2 系から 3 系に上がった際に、sysvinit から systemd に変更されたので、systemd の知識が必要になったからです…

5.2 OS が立ち上がるまで

- 電源 ON ⇒
- BIOS が起動⇒
- BIOS がブートローダー (GRUB) を起動⇒
- ブートローダーがカーネルを起動⇒
- カーネルが init プロセス (sysvinit or systemd) を起動⇒
- inti プロセス (PID1) が諸々のサービスを起動

5.3 Debian の init プロセスは？

- Debian squeeze(Debian 6) までは sysvinit
- Debian wheezy(Debian 7) は sysvinit、ただしプレビューとして systemd が利用可能
- Debian jessie(Debian 8) 以降はデフォルト systemd、ただし sysvinit も利用可能

5.4 Init プロセスを見てみる (sysvinit)

```
hiroya@debian:~$ ps aux | head
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.1  15820  1916 ?        Ss   02:48   0:00 init [2]
root          2  0.0  0.0      0     0 ?        S    02:48   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    02:48   0:00 [ksoftirqd/0]
root          4  0.0  0.0      0     0 ?        S    02:48   0:00 [kworker/0:0]
root          5  0.0  0.0      0     0 ?        S<   02:48   0:00 [kworker/0:0H]
root          7  0.0  0.0      0     0 ?        S    02:48   0:00 [rcu_sched]
root          8  0.0  0.0      0     0 ?        S    02:48   0:00 [rcu_bh]
root          9  0.0  0.0      0     0 ?        S    02:48   0:00 [migration/0]
root         10  0.0  0.0      0     0 ?        S<   02:48   0:00 [lru-add-drain]
hiroya@debian:~$
```

```
hiroya@debian:~$ sudo ls -l /proc/1/exe
lrwxrwxrwx 1 root root 0 May 26 02:48 /proc/1/exe -> /sbin/init
hiroya@debian:~$
```

```
hiroya@debian:~$ dpkg -S /sbin/init
sysvinit-core: /sbin/init
hiroya@debian:~$
```

5.5 Init プロセスを見てみる (systemd)

```
hiroya@debian:~$ ps aux | head
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.1  0.5 105196 10636 ?        Ss   00:01   0:03 /sbin/init
root           2  0.0  0.0      0     0 ?        S    00:01   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   00:01   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   00:01   0:00 [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   00:01   0:00 [kworker/0:0H-kblockd]
root           8  0.0  0.0      0     0 ?        I<   00:01   0:00 [mm_percpu_wq]
root           9  0.0  0.0      0     0 ?        S    00:01   0:00 [ksoftirqd/0]
root          10  0.0  0.0      0     0 ?        I    00:01   0:00 [rcu_sched]
root          11  0.0  0.0      0     0 ?        I    00:01   0:00 [rcu_bh]
hiroya@debian:~$
hiroya@debian:~$ ls -l /sbin/init
lrwxrwxrwx 1 root root 20  5月 25 05:58 /sbin/init -> /lib/systemd/systemd
hiroya@debian:~$
```

```
hiroya@debian:~$ dpkg -S /sbin/init
systemd-sysv: /sbin/init
hiroya@debian:~$
```

5.6 systemd のメリット

5.6.1 これまでの init プロセスでは以下の問題が発生していた

- シェルスクリプトを順番に実行していくので、システムの起動に時間がかかる
- サービスごとに cgroups でリソース配分を調整するなどの仕組みがない

5.6.2 systemd では？

- プロセスの起動を出来る限り並列で行おうとするので、起動が早い
- Cgroup でプロセスの管理を行うので、リソース管理、生存確認などが出来る
- リファレンスがめっちゃ丁寧に書かれている

5.7 sysvinit の起動プロセス

```
# /etc/inittab: init(8) configuration.
# $Id: inittab,v 1.91 2002/01/25 13:35:21 miquels Exp $

# The default runlevel.
id:2:initdefault:

# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rcS

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
# Normally not reached, but fallthrough in case of emergency.
z6:6:respawn:/sbin/sulogin

# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
```

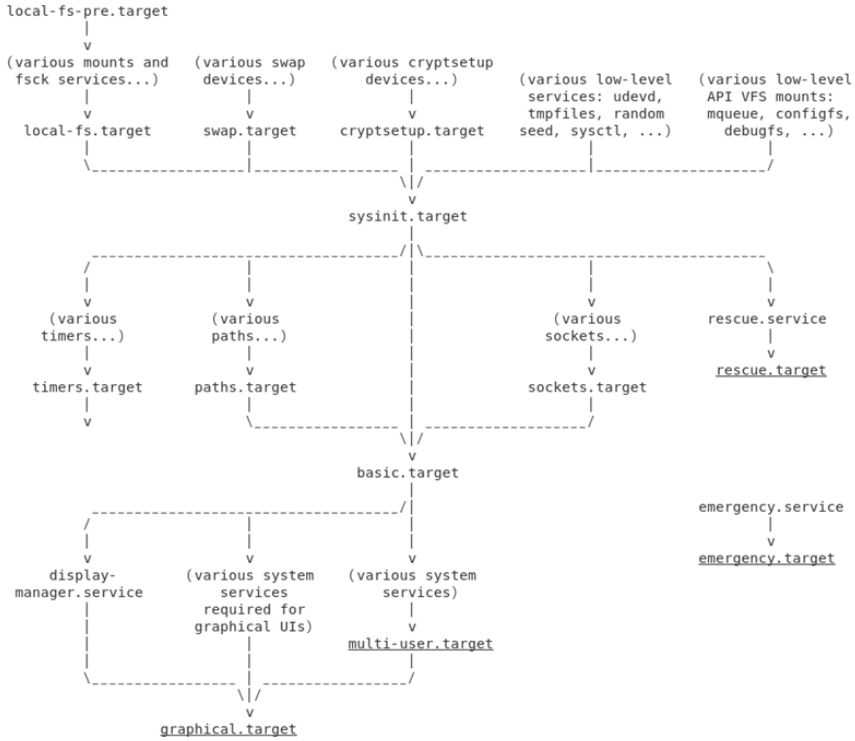
/etc/init.d/rc

```
for i in /etc/rc$runlevel.d/S$level*
do
        [ ! -f $i ] && continue

        suffix=${i#/etc/rc$runlevel.d/S[0-9][0-9]}
        if [ "$previous" != N ]
        then
                #
                # Find start script in previous runlevel and

                        # have to re-stop the service.
                        #
                        [ -f $previous_stop ] && [ ! -f $previous_start ] && continue
                fi

        fi
        SCRIPTS="$SCRIPTS $i"
done
startup $ACTION $SCRIPTS
```

/etc/rc*.d 以下のスクリプトを順番に実行する

```
startup() {
         action=$1
         shift
         scripts="$@"
         for script in $scripts ; do
                 $debug "$script" $action
         done
}
;;
```

5.8 systemd の起動プロセス



5.9 systemd のユニット

systemd はプロセスをユニットという単位で管理する
全部で 11 個のユニットが存在する

- サービスユニット
- ターゲットユニット
- ソケットユニット
- …

5.10 サービスユニット

```
hiroya@debian:~$ ls /lib/systemd/system | grep .service | head -n 20
ModemManager.service
NetworkManager-dispatcher.service
NetworkManager-wait-online.service
NetworkManager.service
accounts-daemon.service
alsa-restore.service
alsa-state.service
alsa-utils.service
anacron.service
apparmor.service
apt-daily-upgrade.service
apt-daily.service
autovt@.service
avahi-daemon.service
bluetooth.service
bolt.service
bootlogd.service
bootlogs.service
bootmisc.service
checkfs.service
hiroya@debian:~$
```

- プロセスを制御・管理するためのユニット
- .service で終わるユニットファイル
- 一般的なユースケースで使用するユニットはほとんどこれ

5.11 ターゲットユニット

```
hiroya@debian:~$ ls /lib/systemd/system | grep .target | head -n 20
basic.target
bluetooth.target
boot-complete.target
cryptsetup-pre.target
cryptsetup.target
ctrl-alt-del.target
default.target
emergency.target
exit.target
final.target
getty-pre.target
getty.target
getty.target.wants
graphical.target
graphical.target.wants
halt.target
halt.target.wants
hibernate.target
hybrid-sleep.target
initrd-fs.target
hiroya@debian:~$
```

- 複数のユニットをグルーピングするためのユニット
- .target で終わるユニットファイル
- ユニット間の起動順序をわかりやすくするために利用される

5.12 systemd の起動順序

ユニットの関係性は「依存」と「順序」の 2 つの考え方が存在

依存

- プロセス同士が同時起動するかどうか
- あくまで、同時に起動するかどうかであり、起動順序は考慮しない

順序

- プロセス起動の順序関係を定義する

例えば、プロセス A がプロセス B を必要とする場合 (依存)、明示的に順序を指定しないと、プロセス A と B は同時に起動する

まず、default.target から依存関係、順序関係を構築

```
hiroya@debian:~$ ls -l /lib/systemd/system/default.target
lrwxrwxrwx 1 root root 16  5月 25 05:58 /lib/systemd/system/default.target -> graphical.target
hiroya@debian:~$
```

```
hiroya@debian:~$ systemctl list-dependencies graphical.target | head
graphical.target
● ├─accounts-daemon.service
● ├─exim4.service
● ├─gdm.service
● ├─rtkit-daemon.service
● ├─switcheroo-control.service
● ├─systemd-update-utmp-runlevel.service
● ├─udisks2.service
● └─multi-user.target
●   ├─anacron.service
hiroya@debian:~$
```

5.13 systemd ではどんなプロセスが起動するの？

sysvinit では、/etc/rc*.d 内のプロセスが起動する

systemd では、以下のディレクトリのプロセスが起動対象

- /etc/systemd/system(ユーザーのコンフィグ)
- /lib/systemd/system(インストールパッケージのコンフィグ)

```
hiroya@debian:~$ ls /lib/systemd/system | head
ModemManager.service
NetworkManager-dispatcher.service
NetworkManager-wait-online.service
NetworkManager.service
accounts-daemon.service
alsa-restore.service
alsa-state.service
alsa-utils.service
anacron.service
anacron.timer
hiroya@debian:~$
```

5.14 systemctlって？

systemd にあれこれと命令を出すことが出来るコマンド

- systemctl status … unit の状態を見ることが出来る
- systemctl edit … drop in という起動で unit ファイルを編集
- systemctl show/cat … unit ファイルを表示
- systemctl enable/disable … プロセスの自動起動を制御
- systemctl daemon-reload … unit ファイルの変更を反映
- systemctl start/stop/restart … プロセスの起動制御
- systemctl list-unit-files –type=service … サービスの一覧を表示

5.15 実際にサービスを作ってみる (自動起動)

sampleA.service

```
[Service]
ExecStart=/bin/bash -c "date > /home/hiroya/sampleA.txt; sleep 10"

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

現在の日時を sampleA.txt の出力するプロセス
multi-user.taget の逆依存を指定

```
hiroya@debian:~$ systemctl show sampleA | head
Type=simple
Restart=no
NotifyAccess=none
RestartUSec=100ms
TimeoutStartUSec=1min 30s
TimeoutStopUSec=1min 30s
RuntimeMaxUSec=infinity
WatchdogUSec=0
WatchdogTimestampMonotonic=0
RootDirectoryStartOnly=no
hiroya@debian:~$
```

sampleA の明示的な設定は 2 行だけだが、後ろでいろいろな設定が勝手に付与される

5.16 自動起動の有効化

```
hiroya@debian:~$ sudo systemctl enable sampleA
Created symlink /etc/systemd/system/multi-user.target.requires/sampleA.service → /etc/systemd/system/sampleA.service.
hiroya@debian:~$
```

```
hiroya@debian:~$ sudo systemctl list-unit-files --type=service | grep sampleA
sampleA.service                             enabled
hiroya@debian:~$
```

5.17 実際にサービスを作ってみる (依存あり)

sampleA.service

- sampleB.service への依存を追加
- sampleA には、sampleB が必要ということ

```
[Unit]
Requires=sampleB.service

[Service]
ExecStart=/bin/bash -c "date > /home/hiroya/sampleA.txt; sleep 10"

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

sampleB.service

```
[Service]
ExecStart=/bin/bash -c "date > /home/hiroya/sampleB.txt"
```

```
hiroya@debian:~$ cat sampleA.txt sampleB.txt
2019年  7月 23日 火曜日 00:26:20 JST
2019年  7月 23日 火曜日 00:26:20 JST
hiroya@debian:~$
```

sampleA から sampleB が呼び出されていることがわかるまた、依存のみなので、sampleA と sampleB は同時に起動している

5.18 実際にサービスを作ってみる (順序あり)

sampleA.service

```
[Unit]
Requires=sampleB.service
Before=sampleB.service

[Service]
Type=oneshot
ExecStart=/bin/bash -c "date > /home/hiroya/sampleA.txt; sleep 10"

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

Before と Type=oneshot を追加することで、sampleA の後に、sampleB を起動することが出来る

```
hiroya@debian:~$ cat sampleA.txt sampleB.txt
2019年  7月 23日 火曜日 00:29:35 JST
2019年  7月 23日 火曜日 00:29:45 JST
hiroya@debian:~$
```

sampleA の 10 秒後に sampleB が起動していることがわかる

5.19 journalctlって？

- systemd 環境では、ログの収集は journald(system-journald) が担当
- journalctl は journald に対して命令を行う事ができるコマンド
- Systemd のサービスのログを表示できるので、エラーの際は便利

```
hiroya@debian:~$ sudo systemctl start sampleC
hiroya@debian:~$ sudo journalctl -u sampleC
-- Logs begin at Tue 2019-07-23 00:44:29 JST, end at Tue 2019-07-23 00:45:28 JST. --
 7月 23 00:45:25 debian systemd[1]: Started sampleC.service.
 7月 23 00:45:25 debian bash[2592]: hello debian
 7月 23 00:45:25 debian systemd[1]: sampleC.service: Succeeded.
hiroya@debian:~$
```

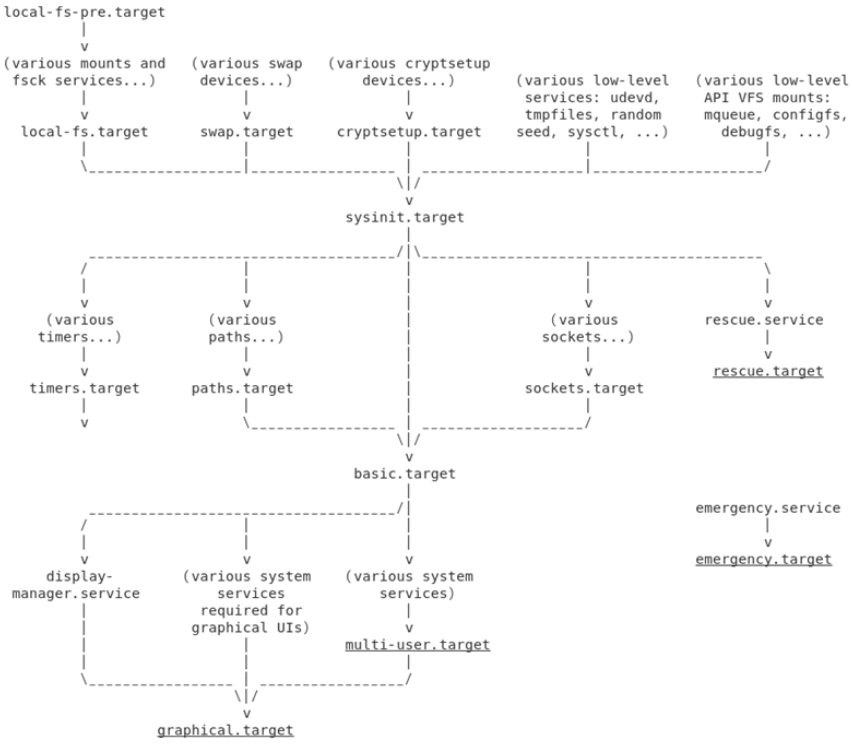
5.20 graphical.target の遷移を邪魔してみる

sample.service

```
[Unit]
Before=sysinit.target
DefaultDependencies=no
OnFailure=emergency.target

[Service]
Type=oneshot
ExecStart=/bin/false

[Install]
RequiredBy=sysinit.target
```

sysinit.target が完了せず、次のターゲットに遷移せずに、強制的に emergency.target に遷移する

systemd の起動プロセス (再掲)



sample.service

```
[Unit]
Before=sysinit.target
DefaultDependencies=no
#OnFailure=emergency.target

[Service]
Type=oneshot
ExecStart=/bin/false

[Install]
RequiredBy=sysinit.target
```

OnFailure をコメントアウトすると、正常に OS が起動しなくなる (プロンプト受付をしなくなる)

もし、sysinit.target よりも前のサービスを作成する場合は、OnFailure を設定しておくと安心

6 systemd と sysvinit のデーモンで動作する Debian パッケージの作り方を調べてみる

杉本　典充

6.1 はじめに

デーモンプログラムを debian パッケージ化するにはどのようにパッケージを作成すればよいのか理解していないため、調べてみました。

6.2 Debian における init の仕組み

一般的に UNIX/Linux 系の OS においてカーネルが最初に呼び出す「init」というユーザランドプログラム[*58]があり、この init が常駐プログラム（＝デーモン）の起動や停止を管理します。

Debian における init システムの解説は、Debian リファレンスの「第 3 章 システムの初期化」に記述があります[*59]。

Debian で採用している init システムは、昔から存在する「sysvinit」、新世代の「systemd」があり[*60]、Debian 8 からは systemd を標準の init システムとして採用しています[*61]。

Debian では systemd と sysvinit を切り替えることができます[*62]。そのためデーモンプログラムの debian パッケージを作るときは、systemd と sysvinit のどちらの環境でもデーモンを起動できるようパッケージを作る必要があります。

6.3 Debian における init が呼び出す起動・停止の設定ファイル

6.3.1 systemd: /lib/systemd/system/[package].service

Debian の systemd が管理するデーモンの起動・終了は、”systemctl” コマンドで行います。

systemctl コマンドで管理するデーモンの設定ファイルは、”/lib/systemd/system/[package].service” に置いています。この ”[package].service” ファイルにプログラムのファイルパスや環境設定ファイル、実行するコマンドを記述します[*63]。

例として、nginx-full パッケージを見てみましょう。

nginx を起動する場合は以下のコマンドを実行します。

```
# systemctl start nginx
```

nginx を停止する場合は以下のコマンドを実行します。

```
# systemctl stop nginx
```

[*58] 一般的にプロセス番号 「1」の場合が多いです。

[*59] https://www.debian.org/doc/manuals/debian-reference/ch03.ja.html

[*60] 他にも OpenRC という init もあります。

[*61] systemd が標準なのは Debian GNU/Linux の場合です。systemd は linux カーネル専用のプログラムのため、kFreeBSD や Hurd では動きません。

[*62] https://wiki.debian.org/Init

[*63] service ファイルの記述方法は、https://www.freedesktop.org/software/systemd/man/systemd.service.html を参照。

上記の systemctl コマンドを実行したとき、以下の”nginx.service”に従って nginx の起動・終了を systemd が行っています。

```
$ cat /lib/systemd/system/nginx.service | grep -v "\#"

[Unit]
Description=A high performance web server and a reverse proxy server
Documentation=man:nginx(8)
After=network.target nss-lookup.target

[Service]
Type=forking
PIDFile=/run/nginx.pid
ExecStartPre=/usr/sbin/nginx -t -q -g 'daemon on; master_process on;'
ExecStart=/usr/sbin/nginx -g 'daemon on; master_process on;'
ExecReload=/usr/sbin/nginx -g 'daemon on; master_process on;' -s reload
ExecStop=-/sbin/start-stop-daemon --quiet --stop --retry QUIT/5 --pidfile /run/nginx.pid
TimeoutStopSec=5
KillMode=mixed

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

6.3.2 sysvinit: /etc/init.d/[package]

Debian の sysvinit が管理するデーモンの起動・終了は、”service”コマンドで行います[*64]。

service コマンドで管理するデーモンの設定ファイルは、”/etc/init.d/[package]”に置いています。この”[package]”ファイルはシェルスクリプトであり、プログラムのファイルパスや環境設定ファイル、実行するコマンドを記述します。

service コマンドから呼び出すスクリプトを作成するにあたり、以下のヘルパースクリプトがあります。

- /lib/init/vars.sh
- /lib/lsb/init-functions
- /sbin/start-stop-daemon

例として、nginx-full パッケージを見てみましょう。

nginx を起動する場合は以下のコマンドを実行します（systemctl の場合とコマンドの順番が違います）。

```
# service nginx start
```

nginx を停止する場合は以下のコマンドを実行します。

```
# service nginx stop
```

上記の service コマンドを実行したとき、nginx の起動・終了を ”/etc/init.d/nginx” が行っています。

※長いため一部省略しています。

[*64] Debian GNU/Linux 8 以降では systemd がデフォルトのため `https://wiki.debian.org/Init` の手順を行うと sysvinit に変更できます。

```
$ cat /etc/init.d/nginx

#!/bin/sh

### BEGIN INIT INFO
# Provides:       nginx
# Required-Start:    $local_fs $remote_fs $network $syslog $named
# Required-Stop:     $local_fs $remote_fs $network $syslog $named
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: starts the nginx web server
# Description:       starts nginx using start-stop-daemon
### END INIT INFO

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
DAEMON=/usr/sbin/nginx
NAME=nginx
DESC=nginx

# Include nginx defaults if available
if [ -r /etc/default/nginx ]; then
        . /etc/default/nginx
fi

STOP_SCHEDULE=''${STOP_SCHEDULE:-QUIT/5/TERM/5/KILL/5}''

test -x $DAEMON || exit 0

. /lib/init/vars.sh
. /lib/lsb/init-functions

# Try to extract nginx pidfile
PID=$(cat /etc/nginx/nginx.conf | grep -Ev '^\s*#' | awk 'BEGIN { RS='';[;{}]'' } { if ($1 == ``pid'') print $2 }' | head -n1)
if [ -z ``$PID'' ]; then
    PID=/run/nginx.pid
fi

if [ -n ``$ULIMIT'' ]; then
    # Set ulimit if it is set in /etc/default/nginx
    ulimit $ULIMIT
fi

start_nginx() {
    # Start the daemon/service
    #
    # Returns:
    #   0 if daemon has been started
    #   1 if daemon was already running
    #   2 if daemon could not be started
    start-stop-daemon --start --quiet --pidfile $PID --exec $DAEMON --test > /dev/null \
        || return 1
    start-stop-daemon --start --quiet --pidfile $PID --exec $DAEMON -- \
        $DAEMON_OPTS 2>/dev/null \
        || return 2
}

  (snip)

stop_nginx() {
    # Stops the daemon/service
    #
    # Return
    #   0 if daemon has been stopped
    #   1 if daemon was already stopped
    #   2 if daemon could not be stopped
    #   other if a failure occurred
    start-stop-daemon --stop --quiet --retry=$STOP_SCHEDULE --pidfile $PID --name $NAME
    RETVAL=''$?''
    sleep 1
    return ``$RETVAL''
}

(snip)

case ``$1'' in
    start)
        log_daemon_msg ``Starting $DESC'' ``$NAME''
        start_nginx
        case ``$?'' in
            0|1) log_end_msg 0 ;;
            2)   log_end_msg 1 ;;
        esac
        ;;
    stop)
        log_daemon_msg ``Stopping $DESC'' ``$NAME''
        stop_nginx
        case ``$?'' in
            0|1) log_end_msg 0 ;;
            2)   log_end_msg 1 ;;
        esac
        ;;

(snip)

    *)
        echo ``Usage: $NAME {start|stop|restart|reload|force-reload|status|configtest|rotate|upgrade}'' >&2
        exit 3
        ;;
esac
```

6.4 init 設定を行う debian パッケージ作成方法

6.4.1 前提

Debian GNU/Linux 9 (stretch) の環境を前提とします。

debian パッケージを作成するツールは、ほとんどのパッケージで debhelper が使われています[*65] 。今回は、debhelper を使ってパッケージ作成を行うことを前提として説明します。

debian パッケージを作成するにあたり、基本的な事項を知るドキュメントは以下になりますので読んでみてください。

- `https://wiki.debian.org/ja/Packaging`
- packaging-tutorial パッケージの PDF ファイル
  /usr/share/doc/packaging-tutorial/packaging-tutorial.ja.pdf[*66]
- Debian 新メンテナーガイド
  `https://www.debian.org/doc/manuals/maint-guide/index.ja.html`
- Debian Policy Manual
  `https://www.debian.org/doc/debian-policy/`

debhelper で debian パッケージの作成を行う場合は以下のパッケージをインストールしておくとよいでしょう[*67]。

```
# apt-get update
# apt-get install build-essential dpkg-dev devscripts debhelper dh-make quilt
```

6.4.2 systemd に対応する設定

以下の web ページに解説ドキュメントがありますので、これを解説していきます。

- `https://wiki.debian.org/Teams/pkg-systemd/Packaging`

systemd の設定を行うには通常のパッケージ作成作業のほか、以下を追加で作業する必要があります。

- debian/rules の修正
- debian/control の修正
- **”debian/package.service” の作成**

debian/rules の修正

debian/rules ファイルは、dh_make コマンドを実行したときにひな形ファイルが作成されます。

systemd で管理するデーモンの場合は、以下のように ”–with systemd” の定義を追加します。この ”–with” オプションは debhelper の addon の処理を実行するように指定するものです[*68]。

```
%:
    dh $@ --with systemd
```

この ”–with systemd” の addon 指定がある場合は debian パッケージのビルド処理で ”dh_systemd_enable”、”dh_systemd_start” を実行するようになります。

”dh_systemd_enable” は、”# systemctl {enable/disable} package” を実行するよう指定するもので、次回の OS 起動時に自動で package のデーモンを起動するか指定できます。

”dh_systemd_start” は、”# systemctl {start/stop/restart} package” を実行するよう指定するコマンドで、パッケージのインストール処理でデーモンを起動するか、再起動するかなどを指定できます。

debian/control の修正

”debian/control”ファイルは、dh_make コマンドを実行したときにひな形ファイルが作成されます。

”debian/control” ファイルを書く上で、debhelper の互換性レベルを記述する”debian/compat” ファイルがあります。この”debian/compat”ファイルの中身が”10”以上であれば ”debian/control” ファイルを次のように書きます。

[*65] `https://www.debian.org/doc/manuals/packaging-tutorial/packaging-tutorial.ja.pdf` の 2019 年 3 月 4 日の資料において debhelper の採用率は 98% との数字があります。

[*66] web 版はこちら。 `https://www.debian.org/doc/manuals/packaging-tutorial/packaging-tutorial.ja.pdf`

[*67] 昔は systemd に対応する場合には dh-systemd パッケージが必要でしたが、stretch では debhelper に統合されているためインストール不要です。

[*68] man dh(1) を参照。

```
Build-Depends: debhelper (>= 9.20160709)
```

もし、”debian/compat” の中身が ”10” 未満であれば systemd 向けの処理は別途 dh-systemd パッケージが必要なため[*69]、次のように書きます。

```
Build-Depends: debhelper (>= 9.20150101), dh-systemd (>= 1.5)
```

debian/package.service の作成

”package.service” ファイルは、dh_make コマンドを実行したときにひな形ファイルが作成されません。
そのため、他のパッケージの ”package.service” ファイルを参考にしつつ、service ファイルのマニュアルをみて作成してください[*70]
”package.service” ファイルを作成しておくと、debian パッケージのビルド処理中の ”dh_installinit” の処理で ”package.service” ファイルを ”/lib/systemd/system” 配下へインストールするよう処理します。

6.4.3 sysvinit に対応する設定

sysvinit の設定を行うには、通常のパッケージ作成作業のほか、以下を追加で作業する必要があります。

- **”debian/package.init” の作成**

debian/package.init の作成

”packages.init” ファイルは、dh_make コマンドを実行したときにひな形ファイルが作成されません[*71]。
他のパッケージの init ファイルを参考にするか、古いバージョンの debhelper に含まれる”init.d.ex” ファイルを取得して作成してください[*72]。
”package.init” ファイルを作成しておくと、debian パッケージのビルド処理中の ”dh_installinit”の処理で ”package.init” ファイルを ”/etc/init.d” 配下へインストールするよう処理します。

6.5 まとめ

Debian における init の説明とデーモンの起動・終了を行う debian パッケージを debhelper でどう作るか説明しました。debian/[package].service ファイル、debian/[package].init ファイルの存在を覚えておくとよいと思います。
debhelper を使った debian パッケージのビルドでは dh_installinit の処理で systemd/sysvinit の設定ファイルをパッケージに配置しています。
また、1 つのソースパッケージから複数のバイナリパッケージを作る場合にファイル名の命名規則が異なるなど違いも出てきます。それらについては他のパッケージを参考にして覚えていくとよいと思います。

6.6 参考文献

- 「Teams pkg-systemd Packaging」 https://wiki.debian.org/Teams/pkg-systemd/Packaging
- mkouhei (2014) 「How to create a Debian package of support to sysvinit, upstart, systemd」 https://d.palmtb.net/2014/01/30/how_to_create_a_debian_package_of_support_to_sysvinit__upstart__systemd.html
- @henrich (2016) 「dh-systemd は debhelper 9.20160709 で統合された」 https://qiita.com/henrich/items/e1651e3284c6b3d0d39e

[*69] Debian 8 jessie 向けのパッケージを作成する場合が該当します。
[*70] https://www.freedesktop.org/software/systemd/man/systemd.service.html
[*71] 過去には dh_make コマンドで”debian/init.d.ex” というひな形ファイルが作成されたのですが、Debian 8 における systemd への移行の過程で作成されなくなりました。
[*72] https://sources.debian.org/data/main/d/dh-make/1.20140617/lib/debian/init.d.ex

7 grml-debootstrap を用いた USB 起動メモリの作成

野首 貴嗣

(発表資料から編集者が適当に変換したので見にくいところはご容赦ください。)

7.1 自己紹介

- NOKUBI Takatsugu (野首 貴嗣)
- knok@debian.org / knok@daionet.gr.jp
- Twitter: @knok
- Debian developer since 2002
- bo 時代からのマシンを運用し続けている

7.2 目次

- USB メモリを使う動機
- grml-debootstrap 概要
- セットアップ準備
- 設定
- ブートローダーの設定

7.3 USB メモリを使う動機

- 内臓ストレージに影響なし
- USB メモリ以外の手法
  - chroot 環境 (debootstrap)
    - daemon 類の扱いが面倒
  - コンテナ (LXC/LXD/Docker 等)
  - 完全仮想環境 (VirtualBox, KVM 等)
    - 物理デバイスを扱いたい時大変
- 物理マシンの環境を汚さない
  - 大抵はライセンスされた proprietary OS が入っている
  - リカバリー領域、診断ツールなど
- 一つのメディアを複数のデバイスで使える
- 複数のデバイスを用意すれば異なる環境を用意できる
- 物理デバイスすべてを扱える
  - 仮想環境と比較して
- NAS の起動デバイスとして (Microserver)

7.4 grml-debootstrap

GRML Live Linux の一部

- http://grml.org/
  - システム管理者向け Live Linux 環境
  - FAI に対応
- パッケージメンテナンス: Grml Team
- 関連ツール: grml-rescueboot, grml2usb
- ソースコード: https://github.com/grml/grml-debootstrap

7.5 セットアップ準備

USB メモリの選定

- 用途と容量
  - デスクトップ用途なら 32GB もあれば十分
  - ストレージを使わないサーバー用途なら 16GB 程度で十分
- 速度
- 耐久性
  - 安物はヘビーなアクセスが蓄積するとすぐ壊れる
- EFI に対応する
  - EFI 領域を分ける必要あり
  - GPT を使う
  - オプション: MBR にも対応するなら 2048 セクタ目からパーティションを開始する
- 方針: スワップパーティションは作らない
- root ファイルシステムの選択
  - ext4
  - f2fs - 最新の grub でしか対応していない点に注意

7.6 設定

```
$ sudo gdisk /dev/sda
GPT fdisk (gdisk) version 1.0.1
Command (? for help): n
Partition number (1-128, default 1):
First sector (34-31358942, default = 2048) or {+-}size{KMGTP}:
Last sector (2048-31358942, default = 31358942) or {+-}size{KMGTP}: +256M
Current type is 'Linux filesystem'
Hex code or GUID (L to show codes, Enter = 8300): ef00
Changed type of partition to 'EFI System'

Command (? for help): n
Partition number (2-128, default 2):
First sector (34-31358942, default = 526336) or {+-}size{KMGTP}:
Last sector (526336-31358942, default = 31358942) or {+-}size{KMGTP}:
Current type is 'Linux filesystem'
Hex code or GUID (L to show codes, Enter = 8300):
Changed type of partition to 'Linux filesystem'

Command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
PARTITIONS!!

Do you want to proceed? (Y/N): y
OK; writing new GUID partition table (GPT) to /dev/sda.
The operation has completed successfully.
```

7.7 インストール

```
$ sudo mkfs.vfat -F32 /dev/sda1
mkfs.fat 4.1 (2017-01-24)
sudo mkfs -t f2fs /dev/sda2

        F2FS-tools: mkfs.f2fs Ver: 1.7.0 (2016-07-28)

Info: Debug level = 0
Info: Label =
Info: Trim is enabled
Info: Segments per section = 1
Info: Sections per zone = 1
Info: sector size = 512
Info: total sectors = 30832607 (15054 MB)
Info: zone aligned segment0 blkaddr: 256
Info: format version with
  "Linux version 4.18.0-0.bpo.1-amd64 (debian-kernel@lists.debian.org) (gcc version 6.3.0 20170516 (Debian 6.3.0-18+deb9u1))
  #1 SMP Debian 4.18.6-1~bpo9+1 (2018-09-13)"
Info: Discarding device
Info: This device doesn't support BLKSECDISCARD
Info: This device doesn't support BLKDISCARD
Info: Overprovision ratio = 1.640%
Info: Overprovision segments = 249 (GC reserved = 129)
Info: format successful
```

–efi オプションは EFI 環境で起動していないと動作しない点に注意

```
$ sudo grml-debootstrap -m http://deb.debian.org/debian -r stretch -t /dev/sda2 --efi /dev/sda1 \
  --grub /dev/sda --arch amd64 --filesystem ext4 --password toor
 * EFI support detected.
 * grml-debootstrap [0.78] - Please recheck configuration before execution:

   Target:          /dev/sda2
   Install grub:    /dev/sda
   Install efi:     /dev/sda1
   Using release:   stretch
   Using hostname:  xps13
   Using mirror:    http://deb.debian.org/debian
   Using arch:      amd64
   Config files:    /etc/debootstrap

   Important! Continuing will delete all data from /dev/sda2!

 * Is this ok for you? [y/N] y
 * EFI partition /dev/sda1 seems to have a FAT filesystem, not modifying.
 * Running mkfs.f2fs  on /dev/sda2

        F2FS-tools: mkfs.f2fs Ver: 1.7.0 (2016-07-28)

Info: Debug level = 0
Info: Label =
Info: Trim is enabled
Info: Segments per section = 1
Info: Sections per zone = 1
Info: sector size = 512
Info: total sectors = 30832607 (15054 MB)
Info: zone aligned segment0 blkaddr: 256
Info: format version with
  "Linux version 4.18.0-0.bpo.1-amd64 (debian-kernel@lists.debian.org) (gcc version 6.3.0 20170516 (Debian 6.3.0-18+deb9u1))
  #1 SMP Debian 4.18.6-1~bpo9+1 (2018-09-13)"
Info: Discarding device
Info: This device doesn't support BLKSECDISCARD
Info: This device doesn't support BLKDISCARD
Info: Overprovision ratio = 1.640%
Info: Overprovision segments = 249 (GC reserved = 129)
Info: format successful
blockdev: ioctl error on BLKRRPART: Device or resource busy
/usr/sbin/grml-debootstrap: line 1119: filesystem: command not found
 * Mounting /dev/sda2 to /mnt/debootstrap.4583
 * Running debootstrap  for release stretch (amd64) using http://deb.debian.org/debian
 * Executing: debootstrap --arch amd64   stretch /mnt/debootstrap.4583 http://deb.debian.org/debian
I: Retrieving InRelease
I: Retrieving Release
I: Retrieving Release.gpg
I: Checking Release signature
I: Valid Release signature (key id 067E3C456BAE240ACEE88F6FEF0F382A1A7B6500)
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
I: Resolving dependencies of base packages...


Finished chroot installation, exiting.
 * Removing chroot-script again
 * Unmount /mnt/debootstrap.4583
 * Removing /var/cache/grml-debootstrap/variables_sda2
 * Removing /var/cache/grml-debootstrap/stages_sda2
 * Finished execution of grml-debootstrap. Enjoy your Debian system.
```

EFI パーティションの/EFI/BOOT 以下に BOOTX64.EFI という名称でファイルを配置

```
# mount /dev/sda1 /mnt
# cd /mnt/EFI
# mkdir BOOT
# cp /boot/efi/EFI/debian/grubx64.efi BOOT/BOOTX64.EFI
# umount /mnt
```

同様に grub-efi-ia32 を用いて BOOT/BOOTx32.EFI を用意すれば 32bit UEFI 対応もできる
MBR に書き込めば EFI/MBR 両対応も可能

7.8 動作確認

- disk グループに追加
  - sudo adduser user disk
  - ログインし直す
- Raw disk に対応する vmdk の作成

```
$ vboxmanage internalcommands createrawvmdk -filename \
~/usb.vmdk -rawdisk /dev/sda
```

- VirtualBox で仮想マシンを作成
  - ストレージに先程作成した usb.vmdk を指定
  - UEFI を有効化
- 起動
- 起動ストレージ
  - fstab, grub.cfg に UUID で記述
- Network
  - if addr でインターフェース名を確認して dhclient
- Timezone
  - dpkg-reconfigure tzdata
- Hardware Clock
  - /etc/adjtime の末尾修正
  - LOCAL で RTC をローカル扱いに
- Keymap
  - dpkg-reconfigure keyboard-configuration
  - `https://wiki.debian.org/Keyboard`
  - お好みで XKBOPTIONS=”ctrl:swapcaps”
- grml-debootstrap を紹介
  - インストーラを使わずストレージに Debian 環境を構築
- USB メモリを使う動機
  - 複数のデバイスで同じ環境を利用できる
  - メディアを分けることで異なる用途に使える
  - 起動マシンの環境を汚さない
- UEFI 環境向けにデフォルトのブートローダーを配置
- VirtualBox を動作テストに使える
  - 細かい調整、パッケージの変更などもできる

7.9 参考文献

参考文献

- BIOS, UEFI 両方で起動可能な debian インストール USB メディアを作る `http://pman0214.github.io/blog/debian-install-bios-efi.html`
- 32bit UEFI 搭載の 2-in-1 PC に ubuntu14.04 をインストールした記録 `https://qiita.com/shirotamago/items/a4b0c8863a492abe50ad`
- Boot your USB Drive in VirtualBox `http://agnipulse.com/2009/07/boot-your-usb-drive-in-virtualbox/`

F2FS はどうか?

- 寿命は伸びるかも?(未検証)

  - USB メモリで TRIM は発行できるか
  - デバイス依存, hdparm -I で確認
- stretch の grub2 は f2fs をサポートしていない
  - /boot を別に作ることで対応可能
- パフォーマンス
  - F2FS Benchmarks From USB Flash Storage (Phoronix) `https://www.phoronix.com/scan.php?page=article&item=linux_f2fs_usb3&num=1`
  - 一長一短?

8 Rust で書いたツールの debian パッケージングに挑戦してみた話 (仮)

小林 克希

8.1 はじめに

今回の内容ですが、最近個人的に触りだした Rust というプログラミング言語について、そもそもどういう言語なんだというご紹介 (といっても、私も勉強中の身ですが) と、Rust で書いたツールの deb を作ってみる、というところまでを目標としています。

8.2 Rust とは

Rust とは、Mozilla がブラウザエンジン Servo のために開発したシステムプログラミング言語です。2018 年のスタックオーバーフローの調査[*73]で愛されている言語ランキング 1 位になるなど、最近では結構メジャーになっていている感じの言語です。特徴としては、安全性・並行性について考えられて設計されている点で、かつ、みんな大好き (?) 静的型付け言語です。

なお、Rust プログラマーの事を rustacean(ラストシアン) というようです。これは、“Crustacean(甲殻類)“ から来ているらしく、そのせいか、オライリー本の表紙はオオヒロバオウギガニというカニですし、非公式マスコットもかわいらしいカニ (名前は Ferris[*74]) になってます。

8.2.1 Rust のツールチェーンをインストール

ではさっそく Rust のツールチェーンをインストールしてみましょう。さて、もちろん Rust のツールチェーンの debian パッケージはありますし、ローカルマシンにインストールせずにブラウザで色々試せるサイト[*75] もあるのですが、Rust は 2018 年の年末に 2018 Edition[*76]という大きめな改版があったので、ひとまず Rust 公式の手段でツールチェーンをインストールしましょう。例によって例のごとく、皆様の大事なホームディレクトリに色々と入れてしまいますが、バージョンアップの頻度はそれなりにあるので、Rust の最新を追っていくなら、無理せずに公式のツールチェーンを入れてしまった方が無難かと思います。

Rust のツールチェーンですが、以前はそうでもなかったみたいですが、最近では rustup[*77]というインストーラーを使うのが公式手順のようです。いかにも最近のツールっぽいですが、以下のように curl コマンドを叩いてインストールします。

```
% curl https://sh.rustup.rs -sSf | sh
```

rustup を使うと、バイナリは `~/.cargo` 以下にインストールされます。パスを通したい場合、 `~/.cargo/env` というファイルがあるので、そちらを `source` してあげるとよろしいかと思います。ひとまず、 `cargo` というコマンドが使えるようになっていたら大丈夫です。なお、「どうしても deb で」という方は `cargo` パッケージをインストールしてください。その際、`rustc` のバージョンが 2018 Edition である 1.31 以降である事が望ましいです。

[*73] https://insights.stackoverflow.com/survey/2018
[*74] http://rustacean.net/
[*75] https://play.rust-lang.org/
[*76] https://blog.rust-lang.org/2018/12/06/Rust-1.31-and-rust-2018.html
[*77] https://rustup.rs/

8.2.2 Hello World してみる

では、cargo が使えるようになったところで、Hello World してみましょう。Rust のプロジェクトは、 `cargo new` で作成することができます。昔は`--bin` をつけないとライブラリ用のプロジェクトになってましたが、最近の cargo であればデフォルトがバイナリ用のプロジェクトになります。それでは実行してみましょう。

```
% cargo new hello
     Created binary (application) `hello` package
% tree
.
├── Cargo.toml
└── src
    └── main.rs

1 directory, 2 files
```

実行すると、こんな感じに Cargo.toml というファイルと、main.rs というソースファイルが作成されます。そう、Rust のソースファイルの拡張子は.rs です。そのため、Rust のプロジェクトのウェブサイトは、.rs ドメインで作られてる事がおおいです。ちなみに、.rs はセルビアのドメインです。

実は、この時点で Hello World の半分が完了しています。src/main.rs の中身を見てみましょう。

```
fn main() {
    println!("Hello, world!");
}
```

なんと、既に Hello World のコードが生成されています。なお、詳細は省きますが、 `println!()` は関数に見えますが、ビックリマークが付いている物は、Rust ではマクロであったりします。まぁ、深く付き会う前は、特に気にしなくて良いかと思います。ビルドして実行するには `cargo run` すれば OK です。

```
% cargo run
   Compiling hello v0.1.0 (/path/to/hello)
    Finished dev [unoptimized + debuginfo] target(s) in 0.31s
     Running `target/debug/hello`
Hello, world!
```

8.3 Rust の構文等の紹介

それでは、ここからは簡単に Rust の構文等の紹介をしていきたいと思います。なお、私も勉強中の身ですので、微妙に嘘を書いていたらすいません。正確な事は、The Book[*78]と呼ばれる気合の入ったドキュメントがありますのでそちらを参考にしていただけたらと思います。

なお、Rust は比較的学習が難しいとか言われてますが、なんで難しいのかについて、個人的にはオライリーの Rust 本で引用されてる Quora に書かれたコメント[*79]がしっくり来たので引用しておきます。

> I've found that Rust has forced me to learn many of the things that I was slowly learning as "good practise" in C/C++ before I could even compile my code.

私は最近会社の仕事の関係で C++ の勉強もするハメになってるのですが、まさに上記と同じ感想です。なにか、C++ がこなれて来て、C++11 あたりでようやく良い感じになった機能だけを取り入れていったのが Rust なんじゃないかなぁと。

8.3.1 変数宣言

変数の宣言は `let` キーワードを使います。型は、最近の言語っぽくコロンの後に書きますが、型が推論できるなら省略可能です。また、シャドーイングすることも可能で、指定がなければ immutable な変数になるので、mutable な変数を作る場合は `mut` キーワードを使います。具体的な例を以下に示します。

[*78] https://doc.rust-lang.org/book/

[*79] https://www.quora.com/What-do-C-C++-systems-programmers-think-of-Rust/answer/Mitchell-Nordine

```
fn main() {
    let x = 1;
    println!("x = {}", x);
    let x = 1.25;
    println!("x = {}", x);
    x = 1;   // エラーになる（そもそも immutable だけど）: expected floating-point number, found integer
    x = 1.5; // エラーになる: cannot assign twice to immutable variable

    let mut y: u32 = 1;
    y -= 1;
    y -= 1;
    println!("y = {}", y);
}
```

例の 2 つ目の `y -= 1;` ですが、デバッグビルドだとオーバーフローを検知して実行時にパニックを起こします。リリースビルド (`--release` オプション付きで `cargo` を実行) だとパニックは発生しませんが、もちろんおかしな値が表示されます。

なお、基本的な型は以下のようなものがあります。

整数　符号無し: `u8`, `u16`, `u32`, `u64`, `usize`
符号付き: `i8`, `i16`, `i32`, `i64`, `isize`
浮動小数点　単精度: `f32`、**倍精度:** `f64`
ブール　`bool`
文字　`char` (Unicode の 1 文字)
文字列　`String`
- ただし、文字列リテラルの `str` があってとてもややこしい

配列　`[T; N]` (`T`: 型, `N`: 要素数)
ベクター　`Vec<T>` (`T`: 型)
スライス　`&[T]` (`T`: 型)

`String` と `str` が少々ややこしいですが、イメージとしては C++ の `std::string` と `char` の関係に近いような感じです。`String` の方は可変長で、メモリ上のイメージを図にすると図 8.3.1 のような感じです。`&`がついてる `str` は、この後述する参照になってます。ひとまず、以下がちょっとしたサンプルコードです。

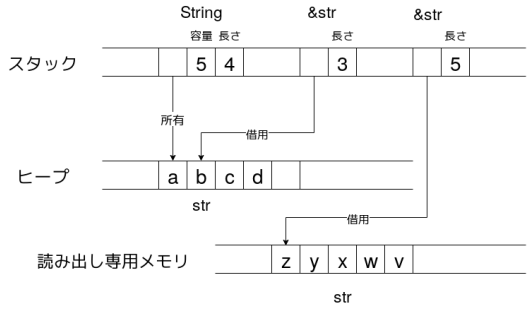


図 6　Rust における String と str の違い

```
fn main() {
    let a = "abcd".to_string(); // String::from("abcd"); でも可
    let b = &a[1..];
    let c = "zyxwv";

    let mut a = "abcd".to_string();
    //  ^^^  a を mutable な変数として宣言
    a.push_str("efg");  // OK

    let mut c = "zyxwv";

    // エラー!: no method named ‘push_str‘ found for type ‘&str‘ in the current scope
    c.push_str("ut");  // str には追加できないので push_str メソッドはない
}
```

8.3.2　関数定義

では、関数を定義してみましょう。Rust での関数定義は、`fn` キーワードと ‘`->`’ トークンを使います。まずは、単純に足し算を行う関数 `add` を定義してみます。

```
fn add(a: i32, b: i32) -> i32 {
    return a + b;
}

fn main() {
    println!("{}", add(1, 2));
}
```

これを cargo で実行すると、(驚くような事は一切ないですが) 以下のような結果になります。

```
% cargo run
Compiling function v0.1.0 (/path/to/examples/function)
Finished dev [unoptimized + debuginfo] target(s) in 0.20s
Running `target/debug/function`
3
```

なお、ここでは return を使いましたが、Rust ではセミコロンを抜かすと値を作るため、セミコロンを抜かして書いた以下のような add 関数も上記で定義した関数と同じ動作をします (セミコロンを書くとエラーになります)。

```
fn add(a: i32, b: i32) -> i32 {
    a + b
}
```

8.3.3 所有権

さて、ここから一気に Rustっぽい話になります。Rust では、メモリの安全性が考えられていると書きましたが、そのために所有権 (英語では ownership) という概念を使っています。C++ でもある概念ですが、たとえば以下のようなプログラムがあったとします。

```
fn main() {
    let i0 = 1;
    let i1 = i0;
    let i2 = i0;

    let s0: String = "hoge".to_string();
    let s1 = s0;
    let s2 = s0;
}
```

さて、どうなると思うでしょうか? 正解は、以下のように s2 に s0 を代入しているところでエラーになります。

```
error[E0382]: use of moved value: `s0`
 --> ownership/src/main.rs:8:15
  |
6 |     let s0: String = "hoge".to_string();
  |         -- move occurs because `s0` has type `std::string::String`, which does not implement the `Copy` trait
7 |     let s1 = s0;
  |              -- value moved here
8 |     let s2 = s0;
  |              ^^ value used here after move
```

Rust では、代入、関数の引数、関数の戻り値などで値の所有権が移動するため、一度所有権を移譲してしまったら、移譲元の変数はその後アクセスできなくなります (コンパイル時に怒られる)。ところが、上記のコードでは整数の方はエラーになりません。これは何故かというと、整数など一部の型 (サイズが固定で小さいものが主?) は移動せずにコピーになります。実はエラーメッセージにある Copy trait というのがミソで、整数型はこの Copy trait を実装しているからそういった動作になっています。trait については後述します。

関数の引数でも所有権は移動するので、以下のようなコードもアウトです。

```
fn consume(_s: String) {}

fn main() {
    let h: String = "Hello World".to_string();
    consume(h);
    let hh = h;
}
```

ビルドすると、関数 consume コール時に h が移動してしまっているため以下のようなエラーになります。

```
12 |     let h: String = "Hello World".to_string();
   |         - move occurs because `h` has type `std::string::String`, which does not implement the `Copy` trait
13 |     consume(h);
   |             - value moved here
14 |     let hh = h;
   |              ^ value used here after move
```

逆に、関数の戻り値でも所有権を移動できるので、以下のようなコードが書けたりします。

```
fn generate(i: i32) -> Vec<i32> {
    let mut v = Vec::new();
    v.push(i);
    return v;
}

fn main() {
    let v1 = generate(10);
    let mut v2 = generate(100);
    v2.push(101);
    println!("v1: {:?}", v1); // v1: [10]
    println!("v2: {:?}", v2); // v2: [100, 101]
}
```

8.3.4 参照と借用

関数呼び出しなどでは、所有権を渡さずに値を使いたい場合が多々あるかと思いますが、その場合には参照を使います。なお、Rust では借用とも言います。記法としては、C++ と同じく`&`を使うのですが、C++ とは、

- 借用する側 (参照される側) に '`&`' を付ける
- 関数の呼び出し元と呼び出し先両方に '`&`' が必要

といった点で異なる感じになります。先程、所有権の移動でエラーになった関数呼び出しのコードを参照をつかってエラーにならないように書き換えたものが以下になります。

```
fn consume(_s: &String) {}

fn main() {
    let s0: String = "hoge".to_string();
    let s1 = &s0;
    let s2 = &s0;

    let h: String = "Hello World".to_string();
    consume(&h);
    let hh = h;
}
```

このコードであれば、関数 `consume` をコールしても、`h` の所有権はうつっていないため、その後 `hh` に所有権を移譲することができます。参照の解決には、以下のように`*`を使います。

```
fn main() {
    let i0 = 5;
    let i1 = &i0;

    println!("i0: {}, i1: {}", i0, *i1);
}
```

が、関数で使う場合だったり、色々な場面で省略できたりするみたいです。

```
fn catlength(a: &String, b: &String) -> usize {
    return a.len() + b.len();
}

fn main() {
    let s0 = "hoge".to_string();
    let s1 = "fuga".to_string();
    println!("{}", catlength(&s0, &s1));
}
```

8.3.5 ライフタイム

Rust には、GC はなくて C 言語のように変数スコープがあり、スコープを外れた変数は随時消えていきます。例としては、以下のようなコードを書くと、ドロップされた後に値を使おうとしているためコンパイル時の怒られます。

```
fn main() {
    {
        let a = 0;
    } // ここでドロップ
    println!("{}", a);
}
```

実行すると以下のように怒られます。

```
error[E0425]: cannot find value ‘a‘ in this scope
--> lifetime/src/main.rs:5:20
  |
5 |     println!("{}", a);
  |                    ^ not found in this scope
```

参照している場合とて怒られます。例として、以下のようなコードを書きます。

```
fn main() {
    let r;
    {
        let b = 0;
        r = &b;
    }
    println!("{}", r);
}
```

コンパイル時に怒られます。

```
error[E0597]: ‘b‘ does not live long enough
--> lifetime/src/main.rs:11:9
   |
11 |         r = &b;
   |         ^^^^^^ borrowed value does not live long enough
12 |     }
   |     - ‘b‘ dropped here while still borrowed
13 |     println!("{}", r);
   |                    - borrow later used here
```

でも、実はこの**「コンパイル時の怒られる」**というのがミソなのです。C/C++ でよくやる解放済みのメモリアクセスとかがコンパイル時にみつかるのです。素敵!! とはいえ、ややこしいのが関数です。以下がエラーになります。

```
fn longest(x: &str, y: &str) -> &str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

こんな感じで怒られます。

```
error[E0106]: missing lifetime specifier
--> lifetime/src/main.rs:1:33
  |
1 | fn longest(x: &str, y: &str) -> &str {
  |                                 ^ expected lifetime parameter
  |
= help: this function’s return type contains a borrowed value, but the signature does not say whether it is \
borrowed from ‘x‘ or ‘y‘
```

このコードの問題はなにかというと、実行するまで `x` と `y`(どちらも参照) のどちらを返すかわからないので、戻り値のライフタイムがどうなるかわからないので、その後の処理がチェックできないという事で怒られています。ではどうするかというと、コンパイラが言っている通り lifetime parameter を追加します。具体的には、生存期間`’a` という表現を用いて以下のように書きます。

```
fn longest<’a>(x: &’a str, y: &’a str) -> &’a str {
    if x.len() > y.len() {
        x
    } else {
        y
    }
}
```

これによって、この関数 `longest` の戻り値は、`x` か `y` の短い方のライフタイム以上のライフタイムを持つという記載になり、コンパイラはこれによって値の借用のチェックを行ないます。

8.3.6 構造体

Rust には class がなく、構造体 `struct` を使います。定義は C 言語の構造体と同じような (型は後置になります) 感じですが、宣言する場合は各要素の名前を逐一書く必要があります。ただし、要素名と同名の変数で初期化するときのみ省略可能です。あとは、他の変数から値をコピーして一部だけ値を入れるといった記法もあります。

```
struct Point {
    x: f64,
    y: f64,
}

fn main() {
    let _p0 = Point { x: 1.0, y: 1.0 };

    let x = 0.0;
    let y = 0.0;
    let _p1 = Point { x, y }; // 変数名とメンバー名が一緒
}
```

なお、構造体ですが、メソッドを生やすことができます。その際、`impl` キーワードを用いて、構造体の定義から離れた場所に書きます。各関数の最初の引数は特別な引数である `self` への参照である必要があります。メンバーの値を書き換えたい場合は `mut` を付けます。

```
struct Point {
    x: f64,
    y: f64,
}

impl Point {
    fn abs(&self) -> f64 {
        (self.x * self.x + self.y * self.y).sqrt()
    }
}

fn main() {
    let p0 = Point { x: 1.0, y: 1.0 };
    println!("{}", p0.abs()); // => 1.4142135623730951
}
```

8.3.7 ジェネリック

C++ のテンプレートや Java のジェネリクスのように、Rust でも型だけ異なるような関数や構造体を一般的な形で記述することができます。

```
struct GenPoint<T> {
    x: T,
    y: T,
}

fn main() {
    let _p1 = GenPoint::<i32> { x: 1, y: 1 };
    let _p2 = GenPoint { x: 1, y: 1 };
    let _p3 = GenPoint { x: 1.5, y: 2.0 };
}
```

8.3.8 trait

Java や Go のインターフェース的なものとして、Rust にはトレイト (trait) というものがあります。さきほど出てきた Copy trait もこれになります。トレイトは `trait` キーワードで定義しますが、たとえばダックタイピングな trait を書くと以下のようになります。

```
// Duck は……
trait Duck {
    fn walk(&self);   // walk して
    fn quack(&self);  // quack するもの!!
}
```

この状態で、この Duck trait を実装する構造体を `impl` キーワードで以下のように作成します。

```
struct RealDuck {}

impl Duck for RealDuck {
    fn walk(&self) {
        println!("duck walking");
    }

    fn quack(&self) {
        println!("quack")
    }
}

struct Dog {}

impl Duck for Dog {
    fn walk(&self) {
        println!("dog walking");
    }

    fn quack(&self) {
        println!("bow")
    }
}
```

そうすると、以下のように Duck 型を引数とする関数 test_duck() に RealDuck と Dog のどちらも渡すことができるようになります。

```
fn test_duck(duck: &Duck) {
    duck.walk();
    duck.quack();
}

fn main() {
    let duck = RealDuck {};
    let dog = Dog {};

    test_duck(&duck);
    test_duck(&dog);
}
```

なお、引数に複数の trait を実装した型を要求することができます。たとえば、

```
fn notify(item: impl Summary + Display) {
```

は 2 つの trait、Summary と Display を同時に実装した型を引数 item に要求します。これだと読みづらい感じもするので、以下の書き型もできます。

```
fn notify<T: Summary + Display>(item: T) {
```

また、引数が一杯になってきたら上記でも辛いので、以下のような where をつかって記法もあります。

```
fn some_function<T, U>(t: T, u: U) -> i32
    where T: Display + Clone,
          U: Clone + Debug
{
```

8.3.9 enum

さて、enum です。個人的にはかなり気にいってますが、Rust の enum は、ちょっとおかしなくらいに色々とできます。もちろん、C 言語のような以下のような enum は作れます。

```
enum Fruits {
    Apple,
    Banana,
    Orange,
    Peach,
}

fn main() {
    let _hoge = Fruits::Apple;
}
```

が、Rust の enum は、各値の中にさらに値を持たせることができます。しかも、型や値の個数はバラバラで構いません。たとえば、以下のような enum を定義できます。なお、ここで出てきている (u8, u8, u8, u8) というのは 8-bit 符号無し整数を 4 つ持つタプル型です。

```
enum IpAddr {
    V4((u8, u8, u8, u8)),
    V6(String),
}
```

このように定義してあげると、実際の値を作る際、以下のように値を入れて enum の値を作ることができます。

```
fn main() {
    let v4_addr = IpAddr::V4((127, 0, 0, 1));
    let v6_addr = IpAddr::V6("::1".into());
}
```

このようにして作った値は、`match` を使って値を取り出せます。C 言語の `switch` 文的な感じで使えます。以下のコードは enum 値全てのケースを記述していますが、`default` 的な処理は "`_ => { /* 処理 */ }`" として書けます。

```
fn extract_addr(addr: &IpAddr) {
    match addr {
        IpAddr::V4(a) => {
            println!("{}.{}.{}.{}", a.0, a.1, a.2, a.3);
        }
        IpAddr::V6(a) => {
            println!("{}", a);
        }
    }
}

fn main() {
    extract_addr(&v4_addr); // => 127.0.0.1
    extract_addr(&v6_addr); // => ::1
}
```

8.3.10 Option<T>と Result<T,E>

Rust には例外も nullptr もありませんが、その代わりにジェネリックな enum である `Option<T>`と `Result<T,E>`を使ってエラー等を処理します。それぞれの機能をざっくり説明すると、

- `Option<T>`
  - 値がない可能性がある事を示す (nullptr の代わり)
- `Result<T,E>`
  - 失敗する可能性がある事を示す (例外の代わり)

といった感じでしょうか。それぞれの定義は以下のようにされています。

```
pub enum Option<T> {
    None,
    Some(T),
}

pub enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

例えば、1 つ目のコマンドライン引数を受けとり、それを i32 型に変更して表示する関数を書くと次のようになります。まず引数が一つも無い場合があるため、引数があれば `Some()` の中身に引数が、なければ `None` が取り出されます。その後、`Some()` の中身を取り出しますが、i32 型としてパースできない場合があるため、成功したか失敗したかを `Ok()` か `Err()` かで判定しています。

```
use std::env;

fn main() {
    let i = match env::args().nth(1) {
        None => {
            panic!("no arguments");
        }
        Some(arg) => match arg.parse::<i32>() {
            Ok(fig) => fig,
            Err(e) => {
                panic!("error: {}", e);
            }
        },
    };

    println!("i: {}", i);
}
```

実際には、上記の処理をもうちょっと綺麗に書けるように?演算子や `expect()` メソッドなどが定義されています。が、ここでは詳細は省きます。

8.3.11 crate

Rust のプログラムは、クレート (crate) と呼ばれる単位の組み合わせで構成されます。今回は、詳細については割愛して、crates.io[*80]にて公開されているクレートの使い方について簡単に紹介します。といっても、実はとても簡単で、cargo の設定ファイルの Cargo.toml に crates.io で使いたいクレートのページに掲載されている式を貼り付けるだけです。

今回はオプション解析のクレートである clap[*81]を使ってみましょう。サイトにある式を、Cargo.toml の [dependencies] の項目に追記してあげれば OK です。

```
[dependencies]
clap = "2.32.0"
```

では、これをつかって、簡単なプログラムを作ってみます。とりあえず、-n オプションだけあるような似非 echo コマンドを以下のように書いてみました。

```
use clap::{App, Arg};

fn main() {
    let m = App::new("echo")  // コマンド名
        .arg(Arg::with_name("STRING").multiple(true))  // 複数の引数
        .arg( // '-n' を定義
            Arg::with_name("n")
                .short("n")
                .help("do not output the trailing newline"),
        )
        .get_matches();  // 実行

    let out = match m.values_of("STRING") {
        //          ↓地味にイテレータを使用
        Some(v) => v.collect::<Vec<&str>>().join(" "),
        None => "".into(),
    };

    print!("{}", out);

    if !m.is_present("n") {
        println!("");  // '-n' オプションがなければ改行
    }
}
```

Rust 2018 Edition と呼ばれる 2018 年の年末にリリースされたバージョン以降では、これまで必要だった `extern crate clap;` という記述が不要になっています。

ということで、ここまでかなり高速に Rust の構文等の紹介をしてきましたが、まったくもって極一部ですので、興味があれば The Book を読んでもらえればと思います。

8.4 Debian パッケージにしてみる

ここでようやく Debian な話題ですが、ひとまずここでは先程作成した clap で作った似非 echo コマンドを debian パッケージにしてみるということをやります。

Rust の単一バイナリの debian パッケージ化について、一番簡単なのは `cargo` のサブコマンド用の `cargo-deb`[*82]を用いる方法があるようです。あるようなんですが……残念ながらこのツール、とても公式に入れられないような deb しか吐かないようです。ドキュメントもないし云々。まぁ、/usr/bin にえいやと入れたいならアリかもですが。

で、さすがにそれだとアレなので、もうちょっとちゃんとした deb を作るべく、Debian Rust packaging team の Wiki[*83]に、それなりにしたがってみようかと思います。どうやら、crates.io にある crate については、debcargo[*84]というツールを使うと良い感じに処理してくれるようです。が、今回は crate.io にはアップされていないもので作りたいという話なんですが、debcargo は対応していなさそうです。そのため、debcargo が使っている dh-cargo をつかって地道に対応してみることにしました。

まずは、dh_make を実行して debian ディレクトリを作り、できあがった debian/rules の build ルールの dh にオプション`--byildsystem cargo` を追加することと、debian/control の Build-Depends に dh-cargo と librust-clap-dev を追加してみました。ひとまずここでビルドしてみます。

[*80] https://crates.io/
[*81] https://clap.rs/
[*82] https://github.com/mmstick/cargo-deb
[*83] https://wiki.debian.org/Teams/RustPackaging
[*84] https://salsa.debian.org/rust-team/debcargo

```
% debuild-pbuilder
-> Attempting to satisfy build-dependencies
-> Creating pbuilder-satisfydepends-dummy package
Package: pbuilder-satisfydepends-dummy
Version: 0.invalid.0

... (snip) ...

dh_autoreconf -O--buildsystem=cargo
dh_auto_configure -O--buildsystem=cargo
cp: 'debian/cargo-checksum.json' を stat できません: そのようなファイルやディレクトリはありません
```

なにかファイルがないと言われてしまいました。Wiki を見ると、どうやら upstream の crate のチェックサムを含めたファイルを用意してやる必要があるようです。これは、/usr/share/cargo/registry 以下に deb でインストールされたライブラリたちを使うためのようなのですが、あまり Wiki を読んでも作り方がわかりませんでした。どうも、cargo-vendor というツールを利用していそうなのですが……。とか思っていたのですが、

```
% apt source ripgrep
% cat rust-ripgrep-0.10.0/debian/cargo-checksum.json
{"package":"Could not get crate checksum","files":{}}
```

と、公式の ripgrep の deb のソースをみると、このファイルが哀しいことになっていました。それでもちゃんとビルドは通るようです。ためしに、debian/cargo-checksum.json を touch で空ファイルとして作ってみたら、普通にビルドが進んでしまいました。いったんわすれます……。

ところが、まだコケます。

```
debian cargo wrapper: running subprocess (['env', 'RUST_BACKTRACE=1', '/usr/bin/cargo', '-Zavoid-dev-deps', \
'build', '--verbose', '--verbose', '-j4', '--target', 'x86_64-unknown-linux-gnu'],) {}
error: failed to select a version for the requirement `textwrap = "= 0.10.0"`
candidate versions found which didn't match: 0.11.0
location searched: directory source `/path/to/clap-test/debian/cargo_registry` (which is replacing registry \
`https://github.com/rust-lang/crates.io-index`)
required by package `clap v2.32.0`
```

clap が依存している textwrap のバージョンが 0.10.0 だけど、手元にあるのは 0.11.0 だと。たしかに、textwrap の deb のバージョンは 0.11.0 でした。が、そうなるとおなじく clap を使っている ripgrep のビルドが通っているのが解せません。と調べていくと、GitHub のリポジトリ[*85]に置いてあるバージョンと ripgrep の deb の orig ソースを見比べると、Cargo.toml が若干異なり、Cargo.lock が編集されていることがわかりました。そして、消された Cargo.lock の中には、textwrap の 0.10.0 のチェックサムが記載されています。そもそも、Cargo.toml はで開発者がざっくりとした依存を書くファイルで、Cargo.lock は、cargo が実情に基づいて自動で生成し、`cargo update` などすることによってバージョンが更新されるものです。

で、確認していくと、どうやら/usr/share/cargo/registry 以下にある textwrap の Cargo.toml の textwrap の依存が crates.io 登録時に以下のように書きかわっていることがわかりました。

```
% grep -A 1 dependencies.textwrap debian/cargo_registry/clap-2.32.0/Cargo.toml
[dependencies.textwrap]
version = ">= 0.10, < 0.12"
```

ちなみに、GitHub の clap のコードには以下としか書いてないです。

```
[dependencies]
bitflags               = "1.0"
unicode-width          = "0.1.4"
textwrap               = "0.10.0"
```

ということで、変更の基準はまったくわかってませんが (今後の課題とさせていただきます!!)、今回の似非 echo コマンドについては、Cargo.lock を消せば最後までビルドされました。lintian のエラーも警告も取れてないですが、ひとまず動きますよ!!

[*85] `https://github.com/BurntSushi/ripgrep`

```
% dpkg -I rust-clap-test_0.1.0-1_amd64.deb
new Debian package, version 2.0.
size 258160 bytes: control archive=648 bytes.
360 バイト、    11 行      control
285 バイト、     4 行      md5sums
Package: rust-clap-test
Version: 0.1.0-1
Architecture: amd64
Maintainer: Katsuki Kobayashi <rare@tirasweel.org>
Installed-Size: 770
Depends: libc6 (>= 2.18), libgcc1 (>= 1:4.2)
Section: unknown
Priority: optional
Homepage: <insert the upstream URL, if relevant>
Description: <insert up to 60 chars description>
<insert long description, indented with spaces>

% dpkg -c rust-clap-test_0.1.0-1_amd64.deb
drwxr-xr-x root/root         0 2019-03-23 18:06 ./
drwxr-xr-x root/root         0 2019-03-23 18:06 ./usr/
drwxr-xr-x root/root         0 2019-03-23 18:06 ./usr/bin/
-rwxr-xr-x root/root    776600 2019-03-23 18:06 ./usr/bin/clap-test
drwxr-xr-x root/root         0 2019-03-23 18:06 ./usr/share/
drwxr-xr-x root/root         0 2019-03-23 18:06 ./usr/share/doc/
drwxr-xr-x root/root         0 2019-03-23 18:06 ./usr/share/doc/rust-clap-test/
-rw-r--r-- root/root       197 2019-03-23 18:06 ./usr/share/doc/rust-clap-test/README.Debian
-rw-r--r-- root/root       190 2019-03-23 18:06 ./usr/share/doc/rust-clap-test/changelog.Debian.gz
-rw-r--r-- root/root      1693 2019-03-23 18:06 ./usr/share/doc/rust-clap-test/copyright
```

8.5 まとめ

ということで、しりすぼみ感は拭えませんが、結局 Rust な deb を作るには、現状では crates.io に登録して debcargo をベースにつくるというのが公式のようでした。が、もちろん dh-cargo があるので crates.io に登録しなくても deb にはできそうです。cargo 自体は、外部クレートに git リポジトリを指定したりできるので、debcargo も git リポジトリに対応できたら良いですが、今回の Cargo.toml/Cargo.lock を crates.io が編集している兼ね合いもあるので、色々と大変かもしれません。

というか、そもそも librust-*パッケージ達の依存関係はあやしいのかもしれません。つい先日も、ripgrep がビルドできないというバグも報告されていましたし[*86]。とは言え、debcargo 自体は粛々と開発が進んでいるようで、これを書いてる直前くらいにも、buster には入らないけど post-install で test ができるようになったり[*87]しているようです。なんにせよ、Rust 自体は良い言語かと思うので、今後流行っていけば良いなぁと思います。

[*86] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=920958

[*87] https://alioth-lists.debian.net/pipermail/pkg-rust-maintainers/2019-March/005296.html

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。

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

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』（2018 年 12 月-2019 年 5 月,2019 年 7 月）および『関西 Debian 勉強会』（2018 年 12 月-2019 年 5 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。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 2019 年夏号

2019 年 8 月 12 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）