Debian Developer's Reference

リリース *14.18*

Developer's Reference Team

2026-11-02

目 次

第 1 章　この文書が扱う範囲について　3

第 2 章　Applying to Become a Member　5
2.1　さあ、はじめよう . . . 5
2.2　Debian メンター (mentors) とスポンサー (sponsors) について . . . 6
2.3　Registering as a Debian member . . . 6

第 3 章　Debian 開発者の責務　9
3.1　パッケージメンテナの責務 . . . 9
3.1.1　次期安定版 (stable) リリースへの作業 . . . 9
3.1.2　安定版 (stable) にあるパッケージをメンテナンスする . . . 9
3.1.3　リリースクリティカルバグに対処する . . . 9
3.1.4　開発元/上流 (upstream) の開発者との調整 . . . 10
3.2　管理者的な責務 . . . 10
3.2.1　あなたの Debian に関する情報をメンテナンスする . . . 11
3.2.2　公開鍵をメンテナンスする . . . 11
3.2.3　投票をする . . . 11
3.2.4　Decision making . . . 12
3.2.5　優雅に休暇を取る . . . 12
3.2.6　脱退について . . . 12
3.2.7　リタイア後に再加入する . . . 13

第 4 章　Resources for Debian Members　15
4.1　メーリングリスト . . . 15
4.1.1　利用の基本ルール . . . 15
4.1.2　開発の中心となっているメーリングリスト . . . 15
4.1.3　特別なメーリングリスト . . . 16
4.1.4　新規に開発関連のメーリングリストの開設を要求する . . . 16
4.2　IRC チャンネル . . . 16
4.3　ドキュメント化 . . . 17
4.4　Debian のマシン群 . . . 17
4.4.1　バグ報告サーバ . . . 18
4.4.2　ftp-master サーバ . . . 18
4.4.3　www-master サーバ . . . 18
4.4.4　people ウェブサーバ . . . 18
4.4.5　salsa.debian.org: Git repositories and collaborative development platform . . . 18
4.4.6　GitHub.com: Submitting pull requests to upstream repositories . . . 19
4.5　開発者データベース . . . 19
4.6　Debian アーカイブ . . . 19
4.6.1　セクション . . . 22

4.6.2 アーキテクチャ . . . 22
4.6.3 パッケージ . . . 23
4.6.4 ディストリビューション . . . 23
4.6.4.1 安定版 (stable)、テスト版 (testing)、不安定版 (unstable) . . . 23
4.6.4.2 テスト版ディストリビューションについてのさらなる情報 . . . 24
4.6.4.3 試験版 (experimental) . . . 24
4.6.5 リリースのコードネーム . . . 25
4.7 Debian ミラーサーバ . . . 26
4.8 Incoming システム . . . 26
4.9 パッケージ情報 . . . 27
4.9.1 ウェブ上から . . . 27
4.9.2 dak ls ユーティリティ . . . 27
4.10 Debian パッケージトラッカー . . . 28
4.11 Developer's packages overview . . . 28
4.12 Debian での FusionForge の導入例: Alioth . . . 29
4.13 Goodies for Debian Members . . . 29

第 5 章 パッケージの取扱い方 31
5.1 新規パッケージ . . . 31
5.2 パッケージの変更を記録する . . . 32
5.3 パッケージをテストする . . . 32
5.4 ソースパッケージの概要 . . . 33
5.5 ディストリビューションを選ぶ . . . 34
5.5.1 特別な例: 安定版 (stable) と 旧安定版 (oldstable) ディストリビューションへアップロードする . . . 34
5.5.2 Special case: the stable-updates suite . . . 35
5.5.3 特別な例: testing/testing-proposed-updates へアップロードする . . . 36
5.6 パッケージをアップロードする . . . 36
5.6.1 Source and binary uploads . . . 36
5.6.2 ftp-master にアップロードする . . . 37
5.6.3 遅延アップロード . . . 37
5.6.4 セキュリティアップロード . . . 37
5.6.5 他のアップロードキュー . . . 38
5.6.6 Notifications . . . 38
5.7 パッケージのセクション、サブセクション、優先度を指定する . . . 38
5.8 New upstream versions . . . 39
5.9 バグの取扱い . . . 39
5.9.1 バグの監視 . . . 40
5.9.2 バグへの対応 . . . 40
5.9.3 バグを掃除する . . . 40
5.9.4 新規アップロードでバグがクローズされる時 . . . 42
5.9.5 セキュリティ関連バグの取扱い . . . 43
5.9.5.1 セキュリティ追跡システム . . . 43
5.9.5.2 秘匿性 . . . 44
5.9.5.3 セキュリティ勧告 . . . 44
5.9.5.4 セキュリティ問題に対処するパッケージを用意する . . . 45
5.9.5.5 修正したパッケージをアップロードする . . . 46

5.10 Subscribing to package updates . . . 47
5.11 パッケージの移動、削除、リネーム、放棄、引き取り、再導入 . . . 47
5.11.1 パッケージの移動 . . . 47
5.11.2 パッケージの削除 . . . 48
5.11.2.1 Incoming からパッケージを削除する . . . 49
5.11.3 パッケージをリプレースあるいはリネームする . . . 49
5.11.4 パッケージを放棄する . . . 49
5.11.5 パッケージを引き取る . . . 50
5.11.6 パッケージの再導入 . . . 50
5.12 移植作業、そして移植できるようにすること . . . 51
5.12.1 移植作業者に対して協力的になる . . . 51
5.12.2 移植作業者のアップロード (porter upload) に関するガイドライン . . . 52
5.12.2.1 再コンパイル、あるいは binary-only NMU . . . 53
5.12.2.2 あなたが移植作業者の場合、source NMU を行う時は何時か . . . 53
5.12.3 移植用のインフラと自動化 . . . 54
5.12.3.1 メーリングリストとウェブページ . . . 54
5.12.3.2 移植用ツール . . . 54
5.12.3.3 wanna-build . . . 55
5.12.4 あなたのパッケージが移植可能なものでは**ない**場合 . . . 55
5.12.5 non-free のパッケージを auto-build 可能であるとマークする . . . 56
5.13 Non-Maintainer Upload (NMU) . . . 56
5.13.1 いつ、どうやって NMU を行うか . . . 56
5.13.2 NMU と debian/changelog . . . 58
5.13.3 DELAYED/ キューを使う . . . 58
5.13.4 メンテナの視点から見た NMU . . . 59
5.13.5 ソース NMU vs バイナリのみの NMU (binNMU) . . . 59
5.13.6 NMU と QA アップロード . . . 60
5.13.7 NMU とチームアップロード . . . 60
5.14 Package Salvaging . . . 60
5.14.1 When a package is eligible for package salvaging . . . 61
5.14.2 How to salvage a package . . . 61
5.15 共同メンテナンス . . . 62
5.16 テスト版ディストリビューション . . . 63
5.16.1 基本 . . . 63
5.16.2 不安定版からの更新 . . . 63
5.16.2.1 時代遅れ (Out-of-date) . . . 64
5.16.2.2 テスト版からの削除 . . . 65
5.16.2.3 循環依存 . . . 65
5.16.2.4 テスト版 (testing) にあるパッケージへの影響 . . . 65
5.16.2.5 詳細について . . . 65
5.16.3 直接テスト版を更新する . . . 66
5.16.4 よく聞かれる質問とその答え (FAQ) . . . 67
5.16.4.1 リリースクリティカルバグとは何ですか、どうやって数えるのですか? . . . 67
5.16.4.2 どのようにすれば、他のパッケージを壊す可能性があるパッケージをテスト版 (testing) へインストールできますか? . . . 67
5.17 The Stable backports archive . . . 67
5.17.1 基本 . . . 67

5.17.2 Exception to the testing-first rule . . . 68
5.17.3 Who can maintain packages in the stable-backports archive? . . . 68
5.17.4 When can one start uploading to stable-backports? . . . 68
5.17.5 How long must a package be maintained when uploaded to stable-backports? . . . 68
5.17.6 How often shall one upload to stable-backports? . . . 68
5.17.7 How can one learn more about backporting? . . . 69

第 6 章 パッケージ化のベストプラクティス 71

6.1 debian/rules についてのベストプラクティス . . . 71
6.1.1 ヘルパースクリプト . . . 71
6.1.2 複数のバイナリパッケージ . . . 72
6.2 debian/control のベストプラクティス . . . 72
6.2.1 The package name . . . 72
6.2.2 パッケージ説明文の基本的なガイドライン . . . 73
6.2.3 パッケージの概要、あるいは短い説明文 . . . 73
6.2.4 長い説明文 (long description) . . . 74
6.2.5 開発元のホームページ . . . 75
6.2.6 バージョン管理システムの場所 . . . 75
6.2.6.1 Vcs-Browser . . . 75
6.2.6.2 Vcs-* . . . 75
6.3 debian/changelog のベストプラクティス . . . 76
6.3.1 役立つ changelog のエントリを書く . . . 76
6.3.2 Selecting the upload urgency . . . 76
6.3.3 changelog のエントリに関するよくある誤解 . . . 77
6.3.4 changelog のエントリ中のよくある間違い . . . 77
6.3.5 NEWS.Debian ファイルで changelog を補足する . . . 78
6.4 セキュリティに関するベストプラクティス . . . 79
6.5 メンテナスクリプトのベストプラクティス . . . 79
6.5.1 Generic guidelines . . . 79
6.5.2 Expected behavior of complex postinst scripts . . . 80
6.6 debconf による設定管理 . . . 80
6.6.1 debconf を乱用しない . . . 80
6.6.2 作者と翻訳者に対する雑多な推奨 . . . 81
6.6.2.1 正しい英語を書く . . . 81
6.6.2.2 翻訳者へ丁寧に接する . . . 81
6.6.2.3 誤字やミススペルを修正する際に fuzzy を取る . . . 82
6.6.2.4 インターフェイスについて仮定をしない . . . 83
6.6.2.5 一人称を使わない . . . 83
6.6.2.6 性差に対して中立であれ . . . 83
6.6.3 テンプレートのフィールド定義 . . . 83
6.6.3.1 Type . . . 83
6.6.3.2 Description: short および extended 説明文 . . . 84
6.6.3.3 Choices . . . 85
6.6.3.4 Default . . . 85
6.6.4 Template fields specific style guide . . . 85
6.6.4.1 Type フィールド . . . 85
6.6.4.2 Description フィールド . . . 85

6.6.4.3 Choices フィールド . . . 86
6.6.4.4 Default フィールド . . . 86
6.7 国際化 . . . 87
6.7.1 debconf の翻訳を取り扱う . . . 87
6.7.2 ドキュメントの国際化 . . . 88
6.8 Best practices for debian/patches . . . 88
6.9 パッケージ化に於ける一般的なシチュエーション . . . 89
6.9.1 autoconf/automake を使っているパッケージ . . . 89
6.9.2 ライブラリ . . . 89
6.9.3 ドキュメント化 . . . 89
6.9.4 特定の種類のパッケージ . . . 89
6.9.5 アーキテクチャ非依存のデータ . . . 90
6.9.6 ビルド中に特定のロケールが必要 . . . 90
6.9.7 移行パッケージを deboprhan に適合するようにする . . . 91
6.9.8 .orig.tar.{gz,bz2,xz} についてのベストプラクティス . . . 91
6.9.8.1 手が入れられていないソース (Pristine source) . . . 91
6.9.8.2 upstream のソースをパッケージしなおす . . . 92
6.9.8.3 バイナリファイルの変更 . . . 93
6.9.9 デバッグパッケージのベストプラクティス . . . 93
6.9.9.1 Automatically generated debug packages . . . 93
6.9.9.2 Manual -dbg packages . . . 94
6.9.10 メタパッケージのベストプラクティス . . . 94

第 7 章 パッケージ化、そして… 95
7.1 バグ報告 . . . 95
7.1.1 一度に大量のバグを報告するには (mass bug filing) . . . 96
7.1.1.1 Usertag . . . 96
7.2 品質維持の努力 . . . 97
7.2.1 日々の作業 . . . 97
7.2.2 バグ潰しパーティ (BSP) . . . 97
7.3 他のメンテナに連絡を取る . . . 97
7.4 活動的でない、あるいは連絡が取れないメンテナに対応する . . . 98
7.5 Dealing with repeatedly negligent Debian developers . . . 99
7.6 Debian 開発者候補に対応する . . . 99
7.6.1 パッケージのスポンサーを行う . . . 99
7.6.1.1 新しいパッケージのスポンサーを行う . . . 100
7.6.1.2 既存パッケージの更新をスポンサーする . . . 101
7.6.2 Granting upload permissions to DMs . . . 102
7.6.3 新たな開発者を支持する (advocate) . . . 103
7.6.4 新規メンテナ申請 (new maintainer applications) を取り扱う . . . 103

第 8 章 国際化と翻訳 105
8.1 どの様にして Debian では翻訳が取り扱われているか . . . 105
8.2 メンテナへの I18N & L10N FAQ . . . 106
8.2.1 翻訳された文章を得るには . . . 106
8.2.2 どの様にして提供された翻訳をレビューするか . . . 106
8.2.3 どの様にして翻訳してもらった文章を更新するか . . . 106

8.2.4 どの様にして翻訳関連のバグ報告を取り扱うか . . . 106
8.3 翻訳者への I18N & L10N FAQ . . . 107
8.3.1 どの様にして翻訳作業を支援するか . . . 107
8.3.2 どの様にして提供した翻訳をパッケージに含めてもらうか . . . 107
8.4 l10n に関する現状でのベストプラクティス . . . 107

第 9 章 Debian メンテナツールの概要 109
9.1 主要なツール . . . 109
9.1.1 dpkg-dev . . . 109
9.1.2 debconf . . . 109
9.1.3 fakeroot . . . 110
9.2 パッケージチェック (lint) 用ツール . . . 110
9.2.1 lintian . . . 110
9.2.2 lintian-brush . . . 110
9.2.3 piuparts . . . 110
9.2.4 debdiff . . . 111
9.2.5 diffoscope . . . 111
9.2.6 duck . . . 111
9.2.7 adequate . . . 111
9.2.8 i18nspector . . . 112
9.2.9 cme . . . 112
9.2.10 licensecheck . . . 112
9.2.11 blhc . . . 112
9.3 debian/rules の補助ツール . . . 112
9.3.1 debhelper . . . 112
9.3.2 dh-make . . . 113
9.3.3 equivs . . . 113
9.4 パッケージ作成ツール . . . 113
9.4.1 git-buildpackage . . . 113
9.4.2 debootstrap . . . 113
9.4.3 pbuilder . . . 114
9.4.4 sbuild . . . 114
9.5 パッケージのアップロード用ツール . . . 114
9.5.1 dupload . . . 114
9.5.2 dput . . . 114
9.5.3 dcut . . . 114
9.6 メンテナンスの自動化 . . . 114
9.6.1 devscripts . . . 115
9.6.2 reportbug . . . 115
9.6.3 autotools-dev . . . 115
9.6.4 dpkg-repack . . . 115
9.6.5 alien . . . 115
9.6.6 elpa-dpkg-dev-el . . . 116
9.6.7 dpkg-depcheck . . . 116
9.6.8 debputy . . . 116
9.7 移植用ツール . . . 116
9.7.1 dpkg-cross . . . 117

9.8 ドキュメントと情報について 117
9.8.1 debian-policy 117
9.8.2 doc-debian 117
9.8.3 developers-reference 118
9.8.4 maint-guide 118
9.8.5 debmake-doc 118
9.8.6 packaging-tutorial 118
9.8.7 how-can-i-help 118
9.8.8 docbook-xml 118
9.8.9 debiandoc-sgml 119
9.8.10 debian-keyring 119
9.8.11 elpa-debian-el 119

Developer's Reference Team <developers-reference@packages.debian.org>

- Copyright © 2019 - 2026 Holger Levsen
- Copyright © 2015 - 2020 Hideki Yamane
- Copyright © 2008 - 2015 Lucas Nussbaum
- Copyright © 2004 - 2007 Andreas Barth
- Copyright © 2002 - 2009 Raphaël Hertzog
- Copyright © 1998 - 2003 Adam Di Carlo
- Copyright © 1997 - 1998 Christian Schwarz

This manual is free software; you may 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, or (at your option) any later version.

This 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.

A copy of the GNU General Public License is available as /usr/share/common-licenses/GPL-2 in the Debian distribution or on the World Wide Web at the GNU web site. You can also obtain it by writing to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

This is Debian Developer's Reference version `14.18`, released on `2026-11-02`.

If you want to print this reference, you should use the pdf version. This manual is also available in some other languages.

第1章　この文書が扱う範囲について

The purpose of this document is to provide an overview of the recommended procedures and the available resources for Debian developers and maintainers.

The procedures discussed within include how to become a member (*Applying to Become a Member*); how to create new packages (**新規パッケージ**) and how to upload packages (**パッケージをアップロードする**); how to handle bug reports (**バグの取扱い**); how to move, remove, or orphan packages (**パッケージの移動、削除、リネーム、放棄、引き取り、再導入**); how to port packages (**移植作業、そして移植できるようにすること**); and how and when to do interim releases of other maintainers' packages (*Non-Maintainer Upload (NMU)*).

また、このリファレンスで触れるリソースには、メーリングリスト (**メーリングリスト**) およびサーバ (*Debian* **のマシン群**)、Debian アーカイブの構成に関する解説 (*Debian* **アーカイブ**)、パッケージのアップロードを受け付ける様々なサーバの説明 (*ftp-master* **にアップロードする**)、パッケージの品質を高めるために開発者が利用できるリソースについての解説 (*Debian* **メンテナツールの概要**) などがあります。

初めに明らかにしておきたいのですが、このリファレンスは Debian パッケージに関する技術的な詳細や、Debian パッケージの作成方法を説明するものではありません。また、このリファレンスは Debian に含まれるソフトウェアが準拠すべき基準を詳細に解説するようなものでもありません。その様な情報については全て、Debian ポリシーマニュアル に記述されています。

さらに、この文書は**公式なポリシーを明らかにするものではありません**。含まれているのは Debian システムに関する記述と、一般的な合意がなされたベストプラクティスに関する記述です。すなわち「規範」文書と呼ばれるものではない、ということです。

第2章　Applying to Become a Member

2.1 さあ、はじめよう

So, you've read all the documentation, you've gone through the Debian New Maintainers' Guide (or its successor, Guide for Debian Maintainers), understand what everything in the `hello` example package is for, and you're about to Debianize your favorite piece of software. How do you actually become a Debian developer so that your work can be incorporated into the Project?

Firstly, subscribe to `debian-devel@lists.debian.org` if you haven't already. Send the word `subscribe` in the `Subject` of an email to `debian-devel-REQUEST@lists.debian.org`. In case of problems, contact the list administrator at `listmaster@lists.debian.org`. More information on available mailing lists can be found in **メーリングリスト**. `debian-devel-announce@lists.debian.org` is another list, which is mandatory for anyone who wishes to follow Debian's development.

参加後、何かコーディングを始める前に、しばらくの間「待ち」(投稿せずに読むだけ) の状態でいるのが良いでしょう。それから、重複作業を避けるために何の作業をしようとしているのか表明をする必要があります。

もう一つ、購読すると良いのが `debian-mentors@lists.debian.org` です。詳細は *Debian* **メンター** *(mentors)* **とスポンサー** *(sponsors)* **について** を参照してください。IRC チャンネル `#debian` も役に立つでしょう。*IRC* **チャンネル** を見てください。

何らかの方法で Debian に対して貢献したいと思った時、同じような作業に従事している既存の Debian メンテナにコンタクトしてみてください。そうすれば経験豊かな開発者から学ぶことができます。例えば、既にあるソフトウェアを Debian 用にパッケージ化するのに興味を持っている場合、スポンサーを探しましょう。スポンサーはあなたと一緒にパッケージ化作業を手伝い、あなたの作業が満足する出来になったら Debian アーカイブにパッケージをアップロードしてくれます。スポンサーは、`debian-mentors@lists.debian.org` メーリングリストへパッケージとあなた自身の説明とスポンサーを探していることをメールして見つけましょう (詳細については **パッケージのスポンサーを行う** および https://wiki.debian.org/DebianMentorsFaq を参照)。さらに、Debian を他のアーキテクチャやカーネルへ移植するのに興味を持っている場合、移植関連のメーリングリストに参加して、どうやって始めればいいのかを尋ねましょう。最後に、ドキュメントや品質保証 (Quality Assuarance、QA) の作業に興味がある場合は、この様な作業を行っているメンテナ達の集まりに参加して、パッチや改善案を送ってください。

メールアドレスのローカルパートが非常に一般的な場合、落とし穴にハマる可能性があります。mail、admin、root、master のような単語は使わないようにするべきです。詳しくは https://www.debian.org/MailingLists/ を参照してください。

2.2 Debian メンター (mentors) とスポンサー (sponsors) について

メーリングリスト debian-mentors@lists.debian.org が、パッケージ化の第一歩目や他の開発者と調整が必要な問題などで手助けを必要としている新米メンテナに用意されています。新たな開発者は皆、このメーリングリストに参加することをお勧めします (詳細は **メーリングリスト** を参照してください)。

一対一での指導 (つまり、プライベートなメールのやり取りで) の方が良い、という人もこのメーリングリストに投稿しましょう。経験豊かな開発者が助けになってくれるはずです。

In addition, if you have some packages ready for inclusion in Debian, but are waiting for your new member application to go through, you might be able to find a sponsor to upload your package for you. Sponsors are people who are official Debian Developers, and who are willing to criticize and upload your packages for you. Please read the debian-mentors FAQ at https://wiki.debian.org/DebianMentorsFaqfirst.

メンターあるいはスポンサーになりたいという人は、*Debian* **開発者候補に対応する** でより詳細な情報が手に入ります。

2.3 Registering as a Debian member

Before you decide to register with Debian, you will need to read all the information available at the New Members Corner. It describes in detail the preparations you have to do before you can register to become a Debian member. For example, before you apply, you have to read the Debian Social Contract. Registering as a member means that you agree with and pledge to uphold the Debian Social Contract; it is very important that members are in accord with the essential ideas behind Debian. Reading the GNU Manifesto would also be a good idea.

The process of registering as a member is a process of verifying your identity and intentions, and checking your technical skills. As the number of people working on Debian has grown to over 1000 and our systems are used in several very important places, we have to be careful about being compromised. Therefore, we need to verify new members before we can give them accounts on our servers and let them upload packages.

実際に登録する前に、あなたは良い仕事ができる貢献者となり得ることを示さねばなりません。バグ追跡システムを介してパッチを送ったり、既存の Debian 開発者のスポンサーによるパッケージの管理をしばらくの間行うなどして、これをアピールします。付け加えておくと、我々は貢献してくれる人達が単に自分のパッケージをメンテナンスするだけにではなく、プロジェクト全体について興味を持ってくれることを期待しています。バグについての追加情報、できればパッチの提供などによって他のメンテナを手助けできるのであれば、早速実行しましょう!

Registration requires that you are familiar with Debian's philosophy and technical documentation. Furthermore, you need an OpenPGP key which has been signed by an existing Debian maintainer. If your OpenPGP key is not signed yet, you should try to meet a Debian Developer in person to get your key signed. There's a Key Signing Coordination page which should help you find a Debian Developer close to you. (If there is no Debian Developer close to you, alternative ways to pass the ID check may be permitted as an absolute exception on a case-by-case-basis. See the identification page for more information.)

If you do not have an OpenPGP key yet, generate one. Every developer needs an OpenPGP key in order to sign and verify package uploads. You should read the manual for the software you are using, since it has much important information that is critical to its security. Many more security failures are due to human error than to software failure or high-powered spy techniques. See **公開鍵をメンテナンスする** for more information on maintaining your public key.

Debian uses the GNU Privacy Guard (package gnupg version 2 or better) as its baseline standard. You can use some other implementation of OpenPGP as well. Note that OpenPGP is an open standard based on RFC 9580.

Your key length must be greater than 2048 bits (4096 bits is preferred); there is no reason to use a smaller key, and doing so would be much less secure.

あなたの公開鍵が subkeys.pgp.net などの公開鍵サーバにない場合は、新規メンテナ 手順 2: 身分証明 にあるドキュメントを読んでください。このドキュメントにはどうやって公開鍵サーバに鍵を登録するのかが記載されています。新規メンテナグループは、まだ登録されていない場合はあなたの公開鍵をサーバに登録します。

幾つかの国では、一般市民の暗号関連ソフトウェアの使用について制限をかけています。しかし、このことは暗号関連ソフトウェアを暗号化ではなく認証に利用する際には完全に合法であるような場合には、Debian パッケージメンテナとしての活動を妨げることにはなりません。あなたが認証目的にすら暗号技術の利用が制限される国に住んでいる、と言う場合は、我々に連絡をしていただければ特別な措置を講じることができます。

To apply as a new member, you need an existing Debian Developer to support your application (an advocate). After you have contributed to Debian for a while, and you want to apply to become a registered developer, an existing developer with whom you have worked over the past months has to express their belief that you can contribute to Debian successfully.

When you have found an advocate, have your OpenPGP key signed and have already contributed to Debian for a while, you're ready to apply. You can simply register on our application page. After you have signed up, your advocate has to confirm your application. When your advocate has completed this step you will be assigned an Application Manager who will go with you through the necessary steps of the New Member process. You can always check your status on the applications status board.

For more details, please consult New Members Corner at the Debian web site. Make sure that you are familiar with the necessary steps of the New Member process before actually applying. If you are well prepared, you can save a lot of time later on.

第3章　Debian 開発者の責務

3.1 パッケージメンテナの責務

As a package maintainer, you're supposed to provide high-quality packages that are well integrated into the system and that adhere to the Debian Policy.

3.1.1 次期安定版 (stable) リリースへの作業

Providing high-quality packages in unstable is not enough; most users will only benefit from your packages when they are released as part of the next stable release. You are thus expected to collaborate with the release team to ensure your packages get included.

より具体的には、パッケージがテスト版 (testing) に移行しているかどうかを見守る必要があります (**テスト版ディストリビューション** 参照) 。テスト期間後に移行が行われない場合は、その理由を分析してこれを修正する必要があります。(リリースクリティカルバグや、いくつかのアーキテクチャでビルドに失敗する場合) あなたのパッケージを修正するのが必要な場合もありますし、依存関係でパッケージが絡まっている状態からの移行を完了する手助けとして、他のパッケージを更新 (あるいは修正、またはテスト版 (testing) からの削除) が必要な事を意味する場合もあります。障害となる理由 (blocker) を判別できない場合は、リリースチームが先の移行に関する現在の障害に関する情報を与えてくれることでしょう。

3.1.2 安定版 (stable) にあるパッケージをメンテナンスする

パッケージメンテナの作業の大半は、パッケージの更新されたバージョンを不安定版 (unstable) へ放り込むことですが、現状の安定版 (stable) リリースのパッケージの面倒をみることも伴っています。

安定版 (stable) への変更は推奨されてはいませんが、可能です。セキュリティ問題が報告された時はいつでも、セキュリティチームと修正版を提供するように協力する必要があります (**セキュリティ関連バグの取扱い** 参照)。important (あるいはそれ以上) な重要度のバグが安定版 (stable) のバージョンのパッケージに報告されたら、対象となる修正の提供を検討する必要があります。安定版 (stable) リリースマネージャに、そのような更新を受け入れられるかどうかを尋ね、それから安定版 (stable) のアップロードを準備するなどができます (**特別な例: 安定版** *(stable)* **と 旧安定版** *(oldstable)* **ディストリビューションへアップロードする** 参照)。

3.1.3 リリースクリティカルバグに対処する

大抵の場合、パッケージに対するバグ報告については **バグの取扱い**で記述されているように対応する必要があります。しかしながら、注意を必要とする特別なカテゴリのバグがあります― リリースクリティカルバグ (RC bug) と呼ばれるものです。critical、grave、serious の重要度が付けられている全てのバグ報告によって、そのパッケージは次の安定版 (stable) リリースに含めるのには適切ではないとされます。そのため、(テスト版 (testing) にあるパッケージに影響する場合に) Debian のリリースを遅らせたり、(不安

定版 (unstable) にあるパッケージにのみ影響する場合に) テスト版 (testing) への移行をブロックする可能性があります。最悪の場合は、パッケージの削除を招きます。これが RC バグを可能な限り素早く修正する必要がある理由です。

If, for any reason, you aren't able to fix an RC bug in a package of yours within 2 weeks (for example due to time constraints, or because it's difficult to fix), you should mention it clearly in the bug report and you should tag the bug `help` to invite other volunteers to chime in. Be aware that RC bugs are frequently the targets of Non-Maintainer Uploads (see *Non-Maintainer Upload (NMU)*) because they can block the `testing` migration of many packages.

RC バグへの関心の無さは、しばしば QA チームによって、メンテナが正しくパッケージを放棄せずに消えてしまったサインとして判断されます。MIA チームが関わることもあり、その場合はパッケージが放棄されます (**活動的でない、あるいは連絡が取れないメンテナに対応する** 参照)。

3.1.4 開発元/上流 (upstream) の開発者との調整

A big part of your job as Debian maintainer will be to stay in contact with the upstream developers. Debian users will sometimes report bugs that are not specific to Debian to our bug tracking system. These bug reports should be forwarded to the upstream developers so that they can be fixed in a future upstream release. Usually it is best if you can do this, but alternatively, you may ask the bug submitter to do it. Ideally, you also get upstream to *subscribe* for Debian Package Tracker notifications for their software in Debian.

Debian 固有ではないバグの修正はあなたの義務ではないとはいえ、できるなら遠慮なく修正してください。そのような修正を行った際は、上流の開発者にも送ってください。時折 Debian ユーザ／開発者が上流のバグを修正するパッチを送ってくる事があります。その場合は、あなたはパッチを確認して上流へ転送する必要があります。

In cases where a bug report is forwarded upstream, it may be helpful to remember that the bts-link service can help with synchronizing states between the upstream bug tracker and the Debian one.

ポリシーに準拠したパッケージをビルドするために上流のソースに手を入れる必要がある場合、以降の上流でのリリースにおいて手を入れなくても済むために、ここで含まれる修正を上流の開発者にとって良い形で提案する必要があります。必要な変更が何であれ、上流のソースからフォークしないように常に試みてください。

As most upstreams nowadays use git for version control, in most cases git-buildpackage offers the most convenient way to create and maintain patches in Debian that so they are submit upstream. For details, see git-buildpackage man pages about using `pq` to write and test `debian/patches` as git commits, and having git remote `upstreamvcs` to easily cherry-pick patches to and from upstream git branches.

開発元の開発者らが Debian やフリーソフトウェアコミュニティに対して敵対的である、あるいは敵対的になってきているのを見つけた場合は、ソフトウェアを Debian に含める必要があるかを再考しなければならなくなるでしょう。時折 Debian コミュニティに対する社会的なコストは、そのソフトウェアがもたらすであろう利益に見合わない場合があります。

3.2 管理者的な責務

Debian のような大きさのプロジェクトは、あらゆる事を追いかけられる管理者用のインフラに依っています。プロジェクトメンバーとして、あらゆる物事が滞り無く進むように、あなたにはいくつかの義務があります。

3.2.1 あなたの Debian に関する情報をメンテナンスする

Debian 開発者に関する情報が含まれた LDAP データベースが https://db.debian.org/ にあります。ここに情報を入力して、情報に変更があった際に更新する必要があります。特に、あなたの debian.org アドレス宛メールの転送先アドレスが常に最新になっているのを必ず確認してください。debian-private の購読をここで設定した場合、そのメールを受け取るアドレスについても同様です。

データベースについての詳細は **開発者データベース** を参照してください。

3.2.2 公開鍵をメンテナンスする

Be very careful with your private keys. Do not place them on any public servers or multiuser machines, such as the Debian servers (see *Debian* **のマシン群**). Back your keys up; keep a copy offline. Read the documentation that comes with your software; read the PGP FAQ and OpenPGP Best Practices.

鍵が盗難に対してだけではなく、紛失についても安全であることを保証する必要があります。失効証明書 (revocation certificate) を生成してコピーを作って下さい (紙にも出力しておくのがベストです)。これは鍵を紛失した場合に必要になります。

公開鍵に対して、署名したり身元情報を追加したりなどしたら、鍵を `keyring.debian.org` の鍵サーバに送付することで Debian キーリングを更新できます。更新は少なくとも月に 1 度は `debian-keyring` パッケージメンテナによって実施されます。

まったく新しい鍵を追加したりあるいは古い鍵を削除したりする必要がある時は、別の開発者に署名された新しい鍵が必要になります。以前の鍵が侵害されたり利用不可能になった場合には、失効証明書 (revocation certificate) も追加する必要があります。新しい鍵が本当に必要な理由が見当たらない場合は、Keyring メンテナは新しい鍵を拒否することがあります。詳細は https://keyring.debian.org/replacing_keys.html で確認できます。

同様に鍵の取り出し方について *Registering as a Debian member* で説明されています。

Debian での鍵のメンテナンスについて、より突っ込んだ議論を `debian-keyring` パッケージ中のドキュメントおよび https://keyring.debian.org/ サイトで知ることができます。

3.2.3 投票をする

Debian は本来の意味での民主主義ではありませんが、我々はリーダーの選出や一般投票の承認において民主主義的なプロセスを利用しています。これらの手続きについては、Debian 憲章で規程されています。

毎年のリーダー選挙以外には、投票は定期的には実施されず、軽々しく提案されるものではありません。提案はそれぞれ `debian-vote@lists.debian.org` メーリングリストでまず議論され、プロジェクトの書記担当者が投票手順を開始する前に複数のエンドースメントを必要とします。

書記担当者が `debian-devel-announce@lists.debian.org` 上で複数回投票の呼びかけを行うので、投票前の議論を追いかける必要はありません (全開発者がこのメーリングリストを購読することが求められています)。民主主義は、人々が投票に参加しないと正常に機能しません。これが我々が全ての開発者に投票を勧める理由です。投票は GPG によって署名／暗号化されたメールによって行われます。

(過去と現在の) 全ての提案リストが Debian 投票情報ページで閲覧できます。提案について、どの様に起案され、支持され、投票が行われたのかという関連情報の確認が可能になっています。

3.2.4 Decision making

Besides the issues covered in **投票をする**, Debian is closer to a do-ocracy, than a democracy. Generally speaking, this means that decisions are taken by the people who do the work, which in most cases is package maintainers and DPL delegates. Public discussion:

- while encouraged and may be appreciated as input to take a decision,
- is not required to take a decision,
- nor can overrule a decision (the latter is possible via a General Resolution or by escalating to the Debian Technical Committee).

In the spirit of do-ocracy, members of the Debian community are expected to contribute in concrete and constructive ways, rather than demand that others do any kind of work for them.

For more on the social aspects of contributing to Debian, you are strongly encouraged to read the Debian Community Guidelines at https://people.debian.org/~enrico/dcg/

3.2.5 優雅に休暇を取る

予定していた休暇にせよ、それとも単に他の作業で忙しいにせよ、開発者が不在になることがあるのはごく普通のことです。注意すべき重要な点は、他の開発者達があなたが休暇中であるのを知る必要があることと、あなたのパッケージについて問題が起こった場合やプロジェクト内での責務を果たすのに問題が生じたという様な場合は、必要なことを彼らが何であってもできるようにすることです。

通常、これはあなたが休暇中にあなたのパッケージが大きな問題 (リリースクリティカルバグやセキュリティ更新など) となっている場合に、他の開発者に対して NMU (*Non-Maintainer Upload (NMU)* 参照) を許可することを意味しています。大抵の場合はそれほど致命的なことはおきませんが、他の開発者に対してあなたが作業できない状態であることを知らせるのは重要です。

他の開発者に通知するために行わなければならないことが 2 つあります。まず、`debian-private@lists.debian.org` にサブジェクトの先頭に [VAC] と付けたメールを送り[1]、いつまで休暇なのかを示しておきます。何か問題が起きた際への特別な指示を書いておくこともできます。

他に行うべき事は **開発者データベース** 上 であなたを vacation とマークする事です (この情報は Debian 開発者のみがアクセスできます)。休暇から戻った時には vacation フラグを削除するのを忘れないように!

理想的には、休暇にあわせて GPG coordination pages に登録して、誰かサインを希望している人がいるかどうかをチェックします。開発者がまだ誰もいないけれども応募に興味を持っている人がいるようなエキゾチックな場所に行く場合、これは特に重要です。

3.2.6 脱退について

Debian プロジェクトから去るのを決めた場合は、以下の手順に従ってください:

- **パッケージを放棄する** の記述に従って、全てのパッケージを放棄 (orphan) してください。
- 一緒にメンテナンスしているパッケージやチームとしてメンテナンスしているパッケージの Uploaders: フィールドから自身を削除してください。

[1] これは、休暇のメッセージを読みたくない人がメッセージを簡単に振り分け可能にするためです。

- @debian.org メールアドレスの alias (例: press@debian.org) 経由でメールを受け取っていて削除したい場合、Debian システム管理者に対する RT チケットをオープンしてください。チケットをオープンするには、削除したい alias のアドレスから、`admin@rt.debian.org` 宛でサブジェクトのどこかに "Debian RT" と入れて送信します。
- Please remember to also retire from teams, e.g. remove yourself from team wiki pages or salsa groups.
- Login into https://nm.debian.org/ and then open https://nm.debian.org/process/emeritus to request emeritus status and write a goodbye message that will be automatically posted on debian-private.

  Salsa is used as authentication provider to the NM site, in case you run into problems opening the retirement process yourself, contact NM front desk using `nm@debian.org`

上記のプロセスに従うのは重要です。何故なら活動を停止している開発者を探してパッケージを放棄するのは、非常に時間と手間がかかることだからです。

3.2.7 リタイア後に再加入する

リタイアした開発者のアカウントは、**脱退について** の手続きが開始された際に「emeritus」であるとマークされ、それ以外の場合は「removed」となります。「emeritus」アカウントになっているリタイアした開発者は、以下のようにすればアカウントを再度有効にできます:

- Log on onto nm.debian.org using your salsa credentials.
- Follow the Return from Emeritus Wizard.
- 短縮された NM プロセスを通過します (リタイアした開発者が P&P および T&S の肝心な部分を覚えているのを確認するためです)。

In case you run into problems contact `nm@debian.org` for further instructions.

リタイアした開発者で「removed」アカウントの人は、NM をもう一度通り抜ける必要があります。

第4章　Resources for Debian Members

In this chapter you will find a very brief roadmap of the Debian mailing lists, the Debian machines which may be available to you as a member, and all the other resources that are available to help you in your work.

4.1 メーリングリスト

Debian 開発者 (それにユーザ) の間で交わされるやり取りの大半は `lists.debian.org` で提供されている広範囲に渡るメーリングリスト群で行われています。どうやって購読／解除するのか、どうやって投稿するか（あるいはしないのか)、どこで過去の投稿を見つけるのか、どうやって過去の投稿の中から探すのか、どうやってメーリングリスト管理者と連絡をとるのか、その他メーリングリストに関する様々な情報については https://www.debian.org/MailingLists/ を参照してください。

4.1.1 利用の基本ルール

メーリングリストのメッセージに返信する際には、大本の投稿者が特別に要求しない限り、同報メール (CC) を送らないようにしてください。メーリングリストに投稿する人は必ず返信を見ているはずです。

クロスポスト (同じメッセージを複数のメーリングリストに投稿する) のはお止め下さい。いつものネット上と同じ様に、返信文では引用を削って下さい。概して投稿するメッセージについては、通常の慣習をしっかりと守ってください。

詳細については行動規範を参照してください。Debian コミュニティガイドラインも読むと良いでしょう。

4.1.2 開発の中心となっているメーリングリスト

開発者が利用すべき Debian の中核メーリングリスト:

- `debian-devel-announce@lists.debian.org` は開発者に重要な事を伝える際に使われます。全開発者がこのメーリングリストを購読する事が望まれます。
- `debian-devel@lists.debian.org` は様々な技術関連の事柄を話し合うのに使われます。
- `debian-policy@lists.debian.org` は Debian ポリシーについて話し合い、それに対して投票を行うのに使われます。
- `debian-project@lists.debian.org` はプロジェクトに関する様々な非技術関連の事柄を話し合うのに使われます。

他にも様々な事柄に特化したメーリングリストが利用できます。一覧については https://lists.debian.org/ を参照してください。

4.1.3 特別なメーリングリスト

debian-private@lists.debian.org は Debian 開発者間でのプライベートな話し合い用に使う特別なメーリングリストです。つまり、理由がなんであれここに投稿された文章は公開するべきではないものであることを意味しています。このため、これは流量が少ないメーリングリストで、ユーザは本当に必要でない限りは debian-private@lists.debian.org を使わないように勧められています。さらに、このメーリングリストから誰かへメールを転送しては**いけません**。様々な理由からこのメーリングリストのアーカイブはウェブから見ることはできませんが、master.debian.org 上のシェルアカウントを使って ~debian/archive/debian-private/ ディレクトリを参照することで確認できます。

debian-email@lists.debian.org は、特別なメーリングリストです。ライセンス、バグ、その他について upstream の作者にコンタクトを取る、他の人とプロジェクトについて議論した内容をアーカイブしておくのに役立つ Debian に関するメールをまとめた「福袋」として使われています。

4.1.4 新規に開発関連のメーリングリストの開設を要求する

Before requesting a mailing list that relates to the development of a package (or a small group of related packages), please consider if using an alias (via a .forward-aliasname file on master.debian.org, which translates into a reasonably nice *you-aliasname@debian.org* address) is more appropriate.

lists.debian.org 上での通常のメーリングリストが本当に必要であると決意した場合は、HOWTO に従って進めてリクエストを埋めてください。

4.2 IRC チャンネル

いくつもの IRC チャンネルが Debian の開発のために用意されています。チャンネルは主に Open and free technology community (OFTC) のネットワーク上にホストされています。irc.debian.org の DNS エントリは irc.oftc.net へのエイリアスです。

Debian 用のメインのチャンネルは一般的にいって #debian になります。これは巨大な、多目的のチャンネルで、ユーザがトピックやボットによって提供される最近のニュースを見つけることができる場所です。#debian は英語を話す人たち用のもので、他の言語を話す人達のために同様なものには #debian.de、#debian-fr、#debian-br など他にも似通った名前のチャンネルがあります。

Debian 開発での中心のチャンネルは #debian-devel です。これはとてもアクティブなチャンネルで、大抵 150 人以上が常にログインしています。このチャンネルは Debian で作業する人達のためのチャンネルであって、サポート用のチャンネルではありません (そのためには #debian があります)。このチャンネルは、こっそり覗いてみたい (そして学びたい) 人に対してもオープンでもあります。このチャンネルのトピックは、開発者にとって興味深い情報に溢れています。

#debian-devel は公開チャンネルなので、debian-private@lists.debian.org で話されている話題について触れるべきではありません。この目的の為には、#debian-private という鍵で守られた他のチャンネルがあります。この鍵は master.debian.org:~debian/archive/debian-private/ で取得可能です。

There are other additional channels dedicated to specific subjects. #debian-bugs is used for coordinating bug squashing parties. #debian-boot is used to coordinate the work on the debian-installer. #debian-doc is occasionally used to talk about documentation, like the document you are reading. Other channels are dedicated to an architecture or a set of packages: #debian-kde, #debian-dpkg, #debian-perl, #debian-python...

同様に非英語圏の開発者のチャンネルも存在しています。例えば `#debian-devel-fr` は Debian の開発に興味があるフランス語を使う人々のためのチャンネルです。

Channels dedicated to Debian also exist on other IRC networks.

4.3 ドキュメント化

This document contains a lot of information which is useful to Debian developers, but it cannot contain everything. Most of the other interesting documents are linked from The Developers' Corner. Take the time to browse all the links; you will learn many more things.

4.4 Debian のマシン群

Debian ではサーバとして動いている複数のコンピュータがあり、この多くは Debian プロジェクトにおいて重要な役割を果たしています。マシンの大半は移植作業に利用されており、全てインターネットに常時接続されています。

マシンのうち幾つかは、Debian マシン利用ポリシーで定められたルールに従う限り、個々の開発者の利用が可能となっています。

とにかく、これらのマシンをあなたが Debian 関連の目的に合うと思ったことに利用できます。システム管理者には丁寧に接し、システム管理者からの許可を最初に得ることなく、非常に大量のディスク容量／ネットワーク帯域／ CPU を消費しないようにしてください。大抵これらのマシンはボランティアによって運用されています。

Debian で利用しているパスワードと Debian のマシンにインストールしてある SSH 鍵を保護することに注意してください。ログインやアップロードの際にパスワードをインターネット越しに平文で送るような Telnet や FTP や POP などの利用方法は避けてください。

あなたが管理者でも無い限り、Debian サーバ上には Debian に関連しないものを一切置かないようにしてください。

Debian のマシン一覧は https://db.debian.org/machines.cgi で確認可能です。このウェブページはマシン名、管理者の連絡先、誰がログイン可能か、SSH 鍵などの情報を含んでいます。

Debian サーバでの作業について問題があり、システム管理者らに知らせる必要があると考えた場合は、https://rt.debian.org/ にあるリクエストトラッカーの DSA (Debian System Administration) チームのキュー一覧でオープンになっている問題の一覧を確認できます (ユーザー名: "debian" と `master.debian.org:~debian/misc/rt-password` にあるパスワードでログインできます)。新たな問題を報告するには、単に `admin@rt.debian.org` にメールを送ってください。"Debian RT" をサブジェクトのどこかに入れるのを忘れずに。DSA チームに連絡を取るには、プライベートな情報あるいはその他の秘密にしておくべき情報を含む場合には `dsa@debian.org` を、それ以外の場合は `debian-admin@lists.debian.org` へメールしてください。DSA チームは OFTC の IRC チャンネル `#debian-admin` にも居ます。

システム管理に関連しない、特定のサービスについて問題がある場合 (アーカイブからパッケージを削除する、ウェブサイトの改善提案など) は、大抵の場合「擬似パッケージ」に対してバグを報告することになります。どうやってバグ報告をするかについては **バグ報告** を参照してください。

中心となっているサーバのうち幾つかは利用が制限されていますが、そこにある情報は他のサーバへミラーされています。

4.4.1 バグ報告サーバ

bugs.debian.org がバグ報告システム (BTS) の中心となっています。

Debian のバグについて定量的な分析や処理をするような計画がある場合、ここで行ってください。ですが、不要な作業の重複や処理時間の浪費を減らすため、何であれ実装する前に debian-devel@lists.debian.org であなたの計画を説明してください。

4.4.2 ftp-master サーバ

The ftp-master.debian.org server holds the canonical copy of the Debian archive. Generally, packages uploaded to ftp.upload.debian.org end up on this server; see **パッケージをアップロードする**.

このサーバの利用は制限されています。ミラーが mirror.ftp-master.debian.org 上で利用可能です。

Debian FTP アーカイブについて問題がある場合、通常 ftp.debian.org 擬似パッケージに対するバグ報告を行うか、ftpmaster@debian.org へメールをする必要がありますが、**パッケージの移動、削除、リネーム、放棄、引き取り、再導入**にある手順も参照してください。

4.4.3 www-master サーバ

メインの web サーバが www-master.debian.org です。公式 web ページを持ち、新たな参加者に対する Debian の顔となっています。

If you find a problem with the Debian web server, you should generally submit a bug against the pseudo-package www.debian.org. Remember to check whether or not someone else has already reported the problem to the Bug Tracking System.

4.4.4 people ウェブサーバ

people.debian.org は、開発者個人の何か Debian に関連するウェブページのために使われているサーバです。

ウェブに置きたい何か Debian 特有の情報を持っている場合、people.debian.org 上のホームディレクトリの public_html 以下にデータを置くことでこれが可能となっています。これには https://people.debian.org/~*your-user-id*/ という URL でアクセス可能です。

他のホストではバックアップされないのに対して、ここではバックアップされるので、これを使うのは特定の位置づけのものだけにするべきです。

大抵の場合、他のホストを使う唯一の理由はアメリカの輸出制限に抵触する素材を公開する必要がある時です。その様な場合はアメリカ国外に位置する他のサーバのどれかを使えます。

何か質問がある場合は、debian-devel@lists.debian.org にメールして下さい。

4.4.5 salsa.debian.org: Git repositories and collaborative development platform

If you want to use a git repository for any of your Debian work, you can use Debian's GitLab instance called Salsa for that purpose. Gitlab provides also the possibility to have merge requests, wiki pages, bug trackers among many other services as well as a fine-grained tuning of access permission, to help working on projects collaboratively.

For more information, please see the documentation at https://wiki.debian.org/Salsa/Doc.

Any Debian package hosted on Salsa has also access to the Salsa CI . The Salsa CI pipeline mimics the tests that are run after each upload to Debian, but instead of having to wait for results or risk the health of the Debian repositories, Salsa CI provides you with instant feedback about any problems the changes you made may have created or solved.

4.4.6 GitHub.com: Submitting pull requests to upstream repositories

If some upstream repository is hosted on GitHub.com, you can use the Debian organization to create repository forks and submit changed branches with pull requests to upstream maintainers.

The organization is open to all Debian Members. To request membership, open an issue in the Debian/.github meta repository.

4.5 開発者データベース

The Developers Database, at https://db.debian.org/, is an LDAP directory for managing Debian developer attributes. You can use this resource to search the list of Debian developers. Part of this information is also available through the finger service on Debian servers; try `finger yourlogin@db.debian.org` to see what it reports.

開発者らは、以下に挙げるような自身に関する様々な情報を変更するためにデータベースにログインができます。

- forwarding address for your debian.org email as well as spam handling. See https://db.debian.org/forward.html for a description of all the options.
- debian-private の購読
- 休暇中かどうか
- 住所、国名、Debian 開発者世界地図で使われている住んでいる地域の緯度経度、電話・ファックス番号、IRC でのニックネーム、そしてウェブページのアドレスなどの個人情報
- Debian プロジェクトのマシン上でのパスワードと優先的に指定するシェル

当然ですが、ほとんどの情報は外部からはアクセス可能にはなっていません。より詳細な情報については、https://db.debian.org/doc-general.html で参照できるオンラインドキュメントを読んでください。

開発者は公式 Debian マシンへの認証に使われる SSH 鍵を登録することもできますし、新たな *.debian.net の DNS エントリの追加すら可能です。これらの機能は https://db.debian.org/doc-mail.html に記述されています。

4.6 Debian アーカイブ

Debian ディストリビューションは大量のパッケージ (現在約 `40000` 個) と幾つかの追加ファイル (ドキュメントやインストール用ディスクイメージなど) から成り立っています。

以下が完全な Debian アーカイブのディレクトリツリーの例です：

```
dists/stable/main/
dists/stable/main/binary-amd64/
dists/stable/main/binary-armel/
dists/stable/main/binary-i386/
     ...
dists/stable/main/source/
     ...
dists/stable/main/disks-amd64/
dists/stable/main/disks-armel/
dists/stable/main/disks-i386/
     ...

dists/stable/contrib/
dists/stable/contrib/binary-amd64/
dists/stable/contrib/binary-armel/
dists/stable/contrib/binary-i386/
     ...
dists/stable/contrib/source/

dists/stable/non-free/
dists/stable/non-free/binary-amd64/
dists/stable/non-free/binary-armel/
dists/stable/non-free/binary-i386/
     ...
dists/stable/non-free/source/

dists/stable/non-free-firmware/
dists/stable/non-free-firmware/binary-amd64/
dists/stable/non-free-firmware/binary-armel/
dists/stable/non-free-firmware/binary-i386/
     ...
dists/stable/non-free-firmware/source/

dists/testing/
dists/testing/main/
     ...
dists/testing/contrib/
     ...
dists/testing/non-free/
     ...
dists/testing/non-free-firmware/
     ...

dists/unstable
dists/unstable/main/
```

(次のページに続く)

(前のページからの続き)

```
     ...
dists/unstable/contrib/
     ...
dists/unstable/non-free/
     ...
dists/unstable/non-free-firmware/
     ...

pool/
pool/main/a/
pool/main/a/apt/
     ...
pool/main/b/
pool/main/b/bash/
     ...
pool/main/liba/
pool/main/liba/libalias-perl/
     ...
pool/main/m/
pool/main/m/mailx/
     ...
pool/non-free/d/
pool/non-free/d/doc-rfc/
     ...
pool/non-free-firmware/f/
pool/non-free-firmware/f/firmware-nonfree/
     ...
```

見て分かるように、一番上のディレクトリは dists/ と pool/ という 2 つのディレクトリを含んでいます。後者の " pool " はパッケージが実際に置かれており、アーカイブのメンテナンスデータベースと関連するプログラムによって利用されます。前者には stable、testing、そして unstable というディストリビューションが含まれます。ディストリビューションサブディレクトリ中の Packages および Sources ファイルは pool/ ディレクトリ中のファイルを参照しています。以下の各ディストリビューションのディレクトリツリーは全く同じ形式になっています。以下で stable について述べていることは unstable や testing ディストリビューションにも同様に当てはまります。

dists/stable は、main、contrib、non-free、non-free-firmware という名前の 4 つのディレクトリを含んでいます。

それぞれ、source パッケージ (source) のディレクトリとサポートしている各アーキテクチャ (binary-i386、binary-amd64 など) のディレクトリがあります。

main は特定のアーキテクチャ (disks-i386、disks-amd64 など) 上で Debian ディストリビューションをインストールする際に必要となるディスクイメージと主要なドキュメントの基本部分が入っている追加ディレクトリを含んでいます。

4.6.1 セクション

Debian アーカイブの main セクションは**公式な Debian ディストリビューション**を構成するものです。main セクションが公式なのは、我々のガイドライン全てに合致するものであるからです。他の 2 つのセクションはそうではなく、合致は異なる程度となっています…つまり、Debian の公式な一部では**ありません**。

main セクションにある全てのパッケージは、Debian フリーソフトウェアガイドライン (DFSG) 及び Debian ポリシーマニュアルに記載されている他のポリシーの要求事項に完全に適合していなければなりません。DFSG は我々の定義する「フリーソフトウェア」です。詳細は Debian ポリシーマニュアルを確認してください。

contrib セクションにあるパッケージは DFSG に適合している必要がありますが、他の要求事項を満たせてはいないことでしょう。例えば、non-free パッケージに依存している、などです。

DFSG を満たさないパッケージは non-free か non-free-firmware セクションに配置されます。これらのパッケージは Debian ディストリビューションの一部としては考えられてはいませんが、我々はこれらを利用できるようにしており、non-free ソフトウェアのパッケージについて (バグ追跡システムやメーリングリストなどの) インフラストラクチャを提供しています。

Debian ポリシーマニュアル は 4 つのセクションについてより正確な定義を含んでいます。上記の説明はほんの触りに過ぎません。

アーカイブの最上位階層で 4 つのセクションに別れていることは、インターネット上の FTP サーバ経由であれ、CD-ROM であれ、Debian を配布したいと考える人にとって大事なことです…その様な人は main セクションと contrib セクションのみを配布することで、法的なリスクを回避できます。例えば、non-free セクションにあるパッケージのいくつかは商的な配布を許可していません。

その一方で、CD-ROM ベンダは non-free 内のパッケージ群の個々のパッケージライセンスを簡単に確認でき、問題が無ければその多くを CD-ROM に含めることが出来ます。(これはベンダによって大いに異なるので、この作業は Debian 開発者にはできません)。

Note that the term section is also used to refer to categories which simplify the organization and browsing of available packages: admin, net, utils, etc. Once upon a time, these sections (subsections, rather) existed in the form of subdirectories within the Debian archive. Nowadays, these exist only in the Section header fields of packages.

4.6.2 アーキテクチャ

はじめのうちは、Linux カーネルは Intel i386 (またはそれより新しい) プラットフォーム用のみが利用可能で、Debian も同様でした。しかし、Linux は日に日に広まり、カーネルも他のアーキテクチャへと移植され、そして Debian はそれらのサポートを始めました。そして、沢山のハードウェアをサポートするだけでは飽き足らず、Debian は hurd や kfreebsd のような他の Unix カーネルをベースにした移植版を作成することを決めました。

Debian GNU/Linux 1.3 was only available as i386. Debian 2.0 shipped for i386 and m68k architectures. Debian 2.1 shipped for the i386, m68k, alpha, and sparc architectures. Since then Debian has grown hugely. Debian 9 supports a total of ten Linux architectures (amd64, arm64, armel, armhf, i386, mips, mips64el, mipsel, ppc64el, and s390x) and two kFreeBSD architectures (kfreebsd-i386 and kfreebsd-amd64).

特定の移植版についての開発者／ユーザへの情報は Debian 移植版のウェブページで入手可能です。

4.6.3 パッケージ

Debian パッケージには 2 種類あります。ソースパッケージとバイナリパッケージです。

フォーマットに応じて、ソースパッケージは必須の `.dsc` ファイルに加え、一つあるいはそれ以上のファイルから成り立ちます:

- フォーマット "1.0" では、`.tar.gz` ファイルか、`.orig.tar.gz` ファイルと `.diff.gz` ファイルの二つを持っています。
- フォーマット "3.0 (quilt)" では、必須となる開発元の tarball である `.orig.tar.{gz,bz2,xz}`、それからオプションで、開発元の追加 tarball である `.orig-`*component*`.tar.{gz,bz2,xz}` をいくつか、そして必須の debian tarball、`debian.tar.{gz,bz2,xz}` です。
- フォーマット "3.0 (native)" では、単一の `.tar.{gz,bz2,xz}` tarball のみを持っています。

If a package is developed specially for Debian and is not distributed outside of Debian, there is just one `.tar.{gz,bz2,xz}` file, which contains the sources of the program; it's called a "native" source package. If a package is distributed elsewhere too, the `.orig.tar.{gz,bz2,xz}` file stores the so-called `upstream source code`, that is the source code that's distributed by the `upstream maintainer` (often the author of the software). In this case, the `.diff.gz` or the `debian.tar.{gz,bz2,xz}` contains the changes made by the Debian maintainer.

`.dsc` ファイルはソースパッケージ中のすべてのファイルをチェックサム (`md5sums`, `sha1sums` および `sha256sums`) と共にリストしたものと、パッケージ関連の追加情報 (メンテナ、バージョン、etc) を含んでいます。

4.6.4 ディストリビューション

The directory system described in the previous chapter is itself contained within `distribution directories`. Each distribution is actually contained in the `pool` directory in the top level of the Debian archive itself.

簡単にまとめると、Debian アーカイブは FTP サーバのルートディレクトリを持っています。例えば、ミラーサイトでいうと `ftp.us.debian.org` では Debian アーカイブそのものは /debian に含まれており、これは共通した配置となっています (他には `/pub/debian` があります)。

ディストリビューションは Debian ソースパッケージとバイナリパッケージと、これに対応した `Sources` と `Packages` のインデックスファイル (これら全てのパッケージのヘッダ情報を含む) から構成されています。前者は `pool/` ディレクトリに、そして後者はアーカイブの `dists/` ディレクトリに含まれています (後方互換性のため)。

4.6.4.1 安定版 (stable)、テスト版 (testing)、不安定版 (unstable)

常に 安定版 (`stable`) (`dists/stable` に属します)、テスト版 (`testing`) (`dists/testing` に属します)、不安定版 (`unstable`) (`dists/unstable` に属します) と呼ばれるディストリビューションが存在しています。これは Debian プロジェクトでの開発プロセスを反映しています。

活発な開発が不安定版 (`unstable`) ディストリビューションで行われています (これが、何故このディストリビューションが``開発ディストリビューション``と呼ばれることがあるかという理由です)。全ての Debian 開発者は、このディストリビューション内の自分のパッケージを何時でも更新できます。つまり、このディストリビューションの内容は日々変化しているのです。このディストリビューションの全てが正しく動作するかを保証することについては特別な努力は払われていないので、時には文字通り不安定 (unstable) となります。

テスト版ディストリビューション ディストリビューションは、パッケージが特定の判定規準を満たした際に不安定版から自動的に移動されることで生成されています。この判定規準はテスト版に含まれるパッケージとして十分な品質であることを保証する必要があります。テスト版への更新は、新しいパッケージがインストールされた後、毎日 2 回実施されています。**テスト版ディストリビューション** を参照してください。

一定の開発期間後、リリースマネージャが適当であると決定すると、テスト版 (testing) ディストリビューションはフリーズされます。これは、不安定版 (unstable) からテスト版 (testing) へのパッケージ移動をどのように行うかのポリシーがきつくなることを意味しています。バグが多すぎるパッケージは削除されます。バグ修正以外の変更がテスト版 (testing) に入ることは許可されません。いくらかの時間経過後、進行状況に応じてテスト版 (testing) ディストリビューションはより一層フリーズされます。テスト版ディストリビューションの取扱い詳細については debian-devel-announce にてリリースチームが発表します。リリースチームが満足する程度に残っていた問題が修正された後、ディストリビューションがリリースされます。リリースは、テスト版 (testing) が安定版 (stable) へとリネームされる事を意味しており、テスト版 (testing) 用の新しいコピーが作成され、以前の安定版 (stable) は旧安定版 (oldstable) にリネームされ、最終的にアーカイブされるまで存在しています。アーカイブ作業では、コンテンツは archive.debian.org へと移動されます。

この開発サイクルは、不安定版 (unstable) ディストリビューションが、一定期間テスト版 (testing) を過ごした後で安定版 (stable) になる仮定に基づいています。一旦ディストリビューションが安定したと考えられたとしても、必然的にいくつかのバグは残っています… これが安定版ディストリビューションが時折更新されている理由です。しかし、これらの更新はとても注意深くテストされており、新たなバグを招き入れるリスクを避けるためにそれぞれ個々にアーカイブに収録されるようになっています。安定版 (stable) への追加提案は、proposed-updates ディレクトリにて参照可能です。proposed-updates にある合格したこれらのパッケージは、定期的にまとめて安定版ディストリビューションに移動され、安定版ディストリビューションのリビジョンレベルが 1 つ増えることになります (例: ‘ 6.0 ’ が ‘ 6.0.1 ’ に、‘ 5.0 ’ が ‘ 5.0.8 ’ に、以下同様)。詳細に付いては、**特別な例: 安定版** *(stable)* **と 旧安定版** *(oldstable)* **ディストリビューションへアップロードする**を参照してください。

Note that development in unstable during the freeze should not be continued as usual, as packages are still build in unstable, before they migrate to testing, thus unstable should only contain packages meant for testing. Thus only upload to unstable during freezes, if you are planning to request an unblock (or if the package is not in testing).

If you want to develop new stuff for after the freeze, upload to experimental instead.

4.6.4.2 テスト版ディストリビューションについてのさらなる情報

パッケージは通常、不安定版 (unstable) におけるテスト版への移行基準を満たした後でテスト版 (testing) ディストリビューションへとインストールされます。

より詳細については、**テスト版ディストリビューション**を参照してください。

4.6.4.3 試験版 (experimental)

試験版 (experimental) は特殊なディストリビューションです。これは、'安定版' や '不安定版' と同じ意味での完全なディストリビューションではありません。その代わり、ソフトウェアがシステムを破壊する可能性がある、あるいは不安定版ディストリビューションに導入することですら不安定過ぎる (だが、それにもかかわらず、パッケージにする理由はある) ものであるような、とても実験的な要素を含むソフト

ウェアの一時的な置き場であることを意味しています。試験版 (experimental) からパッケージをダウンロードしてインストールするユーザは、十分に注意を受けているのを期待されています。要するに、試験版 (experimental) ディストリビューションを利用すると、どのようになるかは全くわからないということです。

以下が、試験版 (experimental) 用の sources.list 5 です:

```
deb http://deb.debian.org/debian/ experimental main
deb-src http://deb.debian.org/debian/ experimental main
```

ソフトウェアがシステムに多大なダメージを与える可能性がある場合、試験版 (experimental) へ配置する方が良いでしょう。例えば、実験的な圧縮ファイルシステムは恐らく試験版 (experimental) に行くことになるでしょう。

パッケージの新しい上流バージョンが新しい機能を導入するが多くの古い機能を壊してしまう場合であれば、アップロードしないでおくか試験版 (experimental) へアップロードするかしましょう。新しいバージョン、ベータ版などで、利用する設定が完全に変わっているソフトウェアは、メンテナの配慮に従って試験版 (experimental) へ入れることができます。もしも非互換性や複雑なアップグレード対応について作業している場合などは、試験版 (experimental) をステージングエリアとして利用することができるのです。その結果、テストユーザは早期に新しいバージョンの利用が可能になります。

試験版 (experimental) のソフトウェアは不安定版 (unstable) へ説明文に幾つかの警告を加えた上で入れることも可能ではありますが、お勧めはできません。それは、不安定版 (unstable) のパッケージはテスト版 (testing) へ移行し、そして安定版 (stable) になることが期待されているからです。試験版 (experimental) を使うのをためらうべきではありません。何故なら ftpmaster には何の苦痛も与えませんし、試験版 (experimental) のパッケージは一旦不安定版 (unstable) により大きなバージョン番号でアップロードされると定期的に削除されるからです。

システムにダメージを与えないような新しいソフトウェアは直接不安定版 (unstable) へ入れることが可能です。

試験版 (experimental) の代わりとなる方法は、people.debian.org 上の個人的な web ページを使うことです。

4.6.5 リリースのコードネーム

Every released Debian distribution has a code name: Debian 11 is called bullseye; Debian 12, bookworm; Debian 13, trixie; the next release, Debian 14, will be called forky and Debian 15 will be called duke. There is also a *pseudo-distribution*, called sid, which is the current unstable distribution; since packages are moved from unstable to testing as they approach stability, sid itself is never released. As well as the usual contents of a Debian distribution, sid contains packages for architectures which are not yet officially supported or released by Debian. These architectures are planned to be integrated into the mainstream distribution at some future date. The codenames and versions for older releases are listed on the website.

Debian はオープンな開発体制 (つまり、誰もが開発について参加／追いかけが可能) となっており、不安定版 (unstable) および テスト版 (testing) ディストリビューションすら Debian の FTP および HTTP サーバネットワークを通じてインターネットへ提供されています。従って、リリース候補版を含むディレクトリをテスト版 (testing) と呼んだ場合、このバージョンがリリースされる際に安定版 (stable) へとリネームする必要があるということを意味しており、すべての FTP ミラーがディストリビューションすべて (とても巨大です) を再回収することになります。

一方、最初からディストリビューションディレクトリを Debian-x.y と呼んでいた場合、皆 Debian リリース x.y が利用可能になっていると考えるでしょう。(これは過去にあったことで、CD-ROM ベンダが Debian 1.0 の CD-ROM を pre-1.0 開発版を元に作成したことによります。これが。何故最初の公式 Debian のリリース版が 1.0 ではなく 1.1 であったかという理由です)。

従って、アーカイブ内のディストリビューションディレクトリの名前はリリースの状態ではなくコードネームで決定されます (例えば 'trixie' など)。これらの名称は開発期間中とリリース後も同じものであり続けます。そして、簡単に変更可能なシンボリックリンクによって、現在の安定版リリースディストリビューションを示すことになります。これが、stable、testing、unstable へのシンボリックリンクがそれぞれ相応しいリリースディレクトリを指しているのに対して、実際のディストリビューションディレクトリではコードネームを使っている理由です。

4.7 Debian ミラーサーバ

各種ダウンロードアーカイブサイトおよびウェブサイトは、中央サーバを巨大なトラフィックから守るために複数ミラーが利用可能となっています。実際のところ、中央サーバのいくつかは公開アクセスが出来るようにはなっていません - 代わりに一次ミラーが負荷を捌いています。このようにして、ユーザは常にミラーにアクセスして利用可能になっており、Debian を多くのサーバやネットワーク越しに配布するのに必要な帯域が楽になり、ユーザが一次配布元に集中しすぎてサイトがダウンしてしまうのをおおよそ避けられるようになります。一次配布ミラーは内部サイトからのトリガーによって更新されるので、可能な限り最新になっている (我々はこれをプッシュミラーと呼んでいます)。

利用可能な公開 FTP/HTTP サーバのリストを含む、Debian ミラーサーバについての全ての情報が https://www.debian.org/mirror/ から入手可能です。この役立つページには、内部的なものであれ公開されるものであれ、自分のミラーを設定することに興味を持った場合に役立つ情報とツールも含まれています。

Note that mirrors are generally run by third parties who are interested in helping Debian. As such, developers generally do not have accounts on these machines.

4.8 Incoming システム

Incoming システムは、更新されたパッケージを集めて Debian アーカイブにインストールする役割を果たしています。これは ftp-master.debian.org 上にインストールされたディレクトリとスクリプトの集合体です。

全てのメンテナによってアップロードされたパッケージは UploadQueue というディレクトリに置かれます。このディレクトリは、毎分 queued と呼ばれるデーモンによってスキャンされ、*.command ファイルが実行されて、そのまま正しく署名された *.changes ファイルが対応するファイルと共に unchecked ディレクトリに移動されます。このディレクトリは ftp-master の様に制限されており、ほとんどの開発者には見えるようにはなっていません。ディレクトリはアップロードされたパッケージと暗号署名の完全性を照合する dak process-upload スクリプトによって 15 分毎にスキャンされます。パッケージがインストール可能であると判断されると、done ディレクトリに移動されます。これがパッケージの初アップロードの場合 (あるいは新たなバイナリパッケージを含んでいる場合)、ftpmaster による許可を待つ場所である new ディレクトリに移動されます。パッケージが ftpmaster によって手動でインストールされるファイルを含む場合は byhand ディレクトリ に移動します。それ以外の場合は、エラーが検出されるとパッケージは拒否されて reject ディレクトリへと移動されます。

パッケージが受け入れられると、システムは確認のメールをメンテナに送り、アップロードの際に修正済みとされたバグをクローズし、auto-builder がパッケージのリコンパイルを始めます。Debian アーカイブに

実際にインストールされるまで、パッケージはすぐに https://incoming.debian.org/ にてアクセス可能になります。この作業は 1 日に 4 回行われます (様々な経緯から 'dinstall run' とも呼ばれています)。そしてパッケージは incoming から削除され、他のパッケージ全てと共に pool にインストールされます。他のすべての更新 (例えば Packages インデックスファイルや Sources インデックスファイル) が作成されると、一次ミラー全てを更新する特別なスクリプトが呼び出されます。

アーカイブメンテナンスのソフトウェアは、あなたがアップロードした OpenPGP/GnuPG で署名された .changes ファイルも適切なメーリングリストへと送信します。パッケージの Distribution が stable に設定されてリリースされた場合、案内は debian-changes@lists.debian.org に送られます。パッケージの Distribution として unstable や experimental が設定されている場合、案内は代わりに debian-devel-changes@lists.debian.org や debian-experimental-changes@lists.debian.org へと投稿されます。

ftp-master は利用が制限されているサーバなので、インストールされたもののコピーは mirror.ftp-master.debian.org 上で全ての開発者が利用できるようになっています。

4.9 パッケージ情報

4.9.1 ウェブ上から

パッケージはそれぞれ複数の目的別のウェブページを持っています。https://packages.debian.org/*package-name* は各ディストリビューション中でそれぞれ利用可能なパッケージバージョンを表示します。バージョン毎のリンク先のページはパッケージの説明、依存関係、ダウンロードへのリンクを含んだ情報を提供しています。

バグ追跡システムは個々のパッケージのバグを記録していきます。https://bugs.debian.org/*package-name* というような URL で与えたパッケージ名のバグを閲覧できます。

4.9.2 dak ls ユーティリティ

dak ls は dak ツールスイートの一部で、全ディストリビューションおよびアーキテクチャの中から利用可能なパッケージバージョンをリストアップします。dak ツールは ftp-master.debian.org 上と、mirror.ftp-master.debian.org 上のミラーにて利用できます。パッケージ名に対して一つの引数を使います。実際に例を挙げた方が分かりやすいでしょう:

```
$ dak ls evince
evince     | 3.22.1-3+deb11u2 | oldstable           | source, amd64, arm64, armel,␣
↪armhf, i386, mips, mips64el, mipsel, ppc64el, s390x
evince     | 3.22.1-3+deb11u2 | oldstable-debug     | source
evince     | 3.30.2-3+deb12u1 | stable              | source, amd64, arm64, armel,␣
↪armhf, i386, mips, mips64el, mipsel, ppc64el, s390x
evince     | 3.30.2-3+deb12u1 | stable-debug        | source
evince     | 3.38.2-1         | testing             | source, amd64, arm64, armel,␣
↪armhf, i386, mips64el, mipsel, ppc64el, s390x
evince     | 3.38.2-1         | unstable            | source, amd64, arm64, armel,␣
↪armhf, i386, mips64el, mipsel, ppc64el, s390x
evince     | 3.38.2-1         | unstable-debug      | source
```

(次のページに続く)

(前のページからの続き)

```
evince     | 40.4-1           | buildd-experimental | source, amd64, arm64, armel,␣
↪armhf, i386, mips64el, mipsel, ppc64el, s390x
evince     | 40.4-1           | experimental        | source, amd64, arm64, armel,␣
↪armhf, i386, mips64el, mipsel, ppc64el, s390x
evince     | 40.4-1           | experimental-debug  | source
```

この例では、不安定版 (unstable) でのバージョンは テスト版 (testing) のバージョンと違っており、テスト版のパッケージは全アーキテクチャについて、binary-only NMU されたパッケージになっています。それぞれのバージョンのパッケージは、全アーキテクチャ上で再コンパイルされています。

You do not need an account on `ftp-master.debian.org` to get this information. The `rmadison` command, from the `devscripts` package (see *devscripts*), queries the same data over HTTPS; for example, `rmadison --suite=unstable --architecture=amd64 evince` restricts the output to one suite and one architecture. The data is also available directly from the web at `https://api.ftp-master.debian.org/madison?package=evince`.

4.10 Debian パッケージトラッカー

The Debian Package Tracker is an email and web-based tool to track the activity of a source package. You can get the same emails that the package maintainer gets, simply by *subscribing* to the package in the Debian Package Tracker.

PTS は各ソースパッケージについての大量の情報をまとめたウェブインターフェイスを https://tracker.debian.org/ に持っています。その機能はたくさんの有用なリンク (BTS、QA の状態、連絡先情報、DDTS の翻訳状態、buildd のログ) や様々な所からの情報 (最近の changelog エントリ 30 個、testing の状態など…) を集めたものです。特定のソースパッケージについて知りたい場合に非常に有用なツールです。さらに、一旦認証すれば、どのパッケージについてもクリックひとつで購読とキャンセルができます。

特定のソースパッケージに関しては `https://tracker.debian.org/pkg/`*sourcepackage* のような URL で直接ウェブページに飛べます。

For more in-depth information, you should have a look at its documentation. Among other things, it explains you how to interact with it by email, how to filter the mails that it forwards, how to configure your VCS commit notifications, how to leverage its features for maintainer teams, etc.

4.11 Developer's packages overview

QA (quality assurance、品質保証) ウェブポータルが https://qa.debian.org/developer.php から利用できます。これは、一人の開発者のすべてのパッケージの一覧表を表示します (集団で行っている場合は、共同メンテナとしてとして表示されます)。この表は開発者のパッケージについてうまく要約された情報を与えてくれます: 重要度に応じたバグの数やそれぞれのディストリビューションで利用可能なバージョン番号、testing の状態やその他有用な情報源へのリンクなどを含んでいます。

open な状態のバグやどのパッケージに対して責任を持っているのかを忘れないため、定期的に自身のデータを見直すのは良い考えです。

4.12 Debian での FusionForge の導入例: Alioth

Until Alioth was deprecated and eventually turned off in June 2018, it was a Debian service based on a slightly modified version of the FusionForge software (which evolved from SourceForge and GForge). This software offered developers access to easy-to-use tools such as bug trackers, patch managers, project/task managers, file hosting services, mailing lists, VCS repositories, etc.

For many previously offered services replacements exist. This is important to know, as there are still many references to alioth which still need fixing. If you encounter such references please take the time to try fixing them, for example by filing bugs or when possible fixing the reference.

4.13 Goodies for Debian Members

Benefits available to Debian Members are documented on https://wiki.debian.org/MemberBenefits.

第5章　パッケージの取扱い方

この章では、パッケージの作成、アップロード、メンテナンス、移植についての情報を扱います。

5.1 新規パッケージ

もしあなたが Debian ディストリビューションに対して新たなパッケージを作成したいという場合、まず作業が望まれるパッケージ (Work-Needing and Prospective Packages (WNPP)) の一覧をチェックする必要があります。WNPP 一覧をチェックすることで、まだ誰もそのソフトをパッケージ化していないことや、作業が重複していないことを確認します。詳細については WNPP のページ を読んでください。

パッケージ化しようとしているソフトについて、誰もまだ作業していないようであれば、まずは `wnpp` 擬似パッケージ (pseudo-package) に対してバグ報告を投稿する必要があります (**バグ報告**)。このバグ報告には、パッケージの説明 (他の人がレビューできます)、作業しようとしているパッケージのライセンス、ダウンロードが可能な現在の URL を含めた新規パッケージの作成予定 (自分自身が分かるだけではないもの) を記述します。

You should set the subject of the bug to `ITP:` *foo* `--` *short description*, substituting the *name of the new package* for *foo*. The severity of the bug report must be set to `wishlist`. Please send a copy to `debian-devel@lists.debian.org` by using the X-Debbugs-CC header (don't use CC:, because that way the message's subject won't indicate the bug number). If you are packaging so many new packages (>10) that notifying the mailing list in separate messages is too disruptive, send a summary after filing the bugs to the debian-devel list instead. This will inform the other developers about upcoming packages and will allow a review of your description and package name.

新規パッケージがアーカイブへインストールされる際にバグ報告を自動的に閉じるため、`Closes:` *#nnnnn* というエントリを新規パッケージの changelog 内に含めてください (**新規アップロードでバグがクローズされる時** を参照)。

パッケージについて、NEW パッケージキューの管理者への説明が必要だろうと思う場合は、changelog に説明を含めて `ftpmaster@debian.org` へ送ってください。アップロード後であればメンテナとして受け取ったメールに返信してください。もしくは既に再アップロード最中の場合は reject メールに対して返信してください。

セキュリティバグを閉じる場合は、CVE 番号を `Closes:` *#nnnnn* と同じく含めるようにしてください。これは、セキュリティチームが脆弱性を追跡するのに役立ちます。アドバイザリの ID が分かる前にバグ修正のためのアップロードが行われた場合は、以前の changelog エントリを次のアップロード時に修正するのが推奨されています。このような場合でも、元々の changelog での記載に可能な限り背景情報へのポインタを全て含めてください。

我々がメンテナに意図しているところをアナウンスする様に求めるのには、いくつもの理由があります。

- (潜在的な新たな) メンテナが、メーリングリストの人々の経験を活かすのを手助けし、もし他の誰かが既に作業を行っていた場合に知らせる。

- そのパッケージについての作業を検討している他の人へ、既に作業をしているボランティアがいることを知らせ、労力が共有される。
- debian-devel-changes@lists.debian.org に流される一行の説明文 (description) と通常どおりの「Intial release」という changelog エントリよりも、残った他のメンテナがパッケージに関してより深く知ることができる。
- 不安定版 (unstable) で暮らす人 (そして最前線のテスターである人) の助けになる。我々はそのような人々を推奨すべきである。
- メンテナや他に興味を持つ人々へ、プロジェクトで何が行われているのか、何か新しいことがあるかということ関して、告知は良い印象を与える。

5.2 パッケージの変更を記録する

パッケージについて行った変更は debian/changelog に記録されなくてはいけません。これらの変更には、何が変更されたのか、(不確かであれば) 何故なのか、そしてどのバグが閉じられたのかの簡潔な説明文を付加する必要があります。このファイルは /usr/share/doc/*package*/changelog.Debian.gz、あるいはネイティブパッケージの場合は /usr/share/doc/*package*/changelog.gz にインストールされます。

debian/changelog ファイルは、幾つもの異なった項目からなる特定の構造に従っています。一点を取り上げてみると、distribution については**ディストリビューションを選ぶ**に記述されています。このファイルの構造について、より詳細な情報は Debian ポリシーの debian/changelog という章で確認できます。

changelog への記載は、パッケージがアーカイブにインストールされる際、自動的に Debian バグを閉じるのに利用できます。**新規アップロードでバグがクローズされる時** を参照してください。

ソフトウェアの新しい開発元のバージョン (new upstream version) を含むパッケージの changelog エントリは、以下のようにするのが慣習です:

```
* New upstream release.
```

There are tools to help you create entries and finalize the changelog for release– see *devscripts* (command dch), git-buildpackage (command gbp dch) and *elpa-dpkg-dev-el*.

debian/changelog **のベストプラクティス** も参照してください。

5.3 パッケージをテストする

パッケージをアップロードする前に、基本的なテストをする必要があります。最低限、以下の作業が必要です (同じ Debian パッケージの古いバージョンなどが必要になるでしょう):

- パッケージに対して lintian を実行する。以下のようにして lintian を実行できます: lintian -v*package-version*.changes これによって、バイナリパッケージ同様にソースパッケージを確認できます。lintian が生成した出力を理解していない場合は、lintian が問題の説明を非常に冗長に出力するようにする -i オプションを付けて実行してみてください。

  通常、lintian がエラーを出力するようであれば、パッケージをアップロードしては**いけません** (エラーは E で始まります)。

  lintian についての詳細は、*lintian* を参照してください。

- もし古いバージョンがあれば、それからの変更点を分析するために追加で `debdiff` を実行する (*debdiff* を参照) 。
- Install the package and make sure the software works in an up-to-date `unstable` system.
- Upgrade the package from an older version to your new version.
- パッケージを削除して、再インストールする。
- Installing, upgrading and removal of packages can either be tested manually or by using the `piuparts` tool.
- With the package installed, run `adequate <package name>` to test its installed state for policy violations. Ideally, you should automate this step using `autopkgtest` (see *adequate*).
- ソースパッケージを違うディレクトリにコピーして展開し、再構築する。これは、パッケージが外部の既存ファイルに依っているか、`.diff.gz` ファイル内に含まれているファイルで保存されている権限に依るかどうかをテストします。

5.4 ソースパッケージの概要

Debian のソースパッケージには 2 種類あります：

- いわゆる ネイティブ `(native)` パッケージ。元のソースと Debian で当てられたパッチの間に差が無いもの
- オリジナルのソースコードの tarball ファイルに、Debian によって作成された変更点を含む別のファイルが付随している (より一般的な) パッケージ

ネイティブパッケージの場合、ソースパッケージは Debian のソース control ファイル (`.dsc`) とソースコードの tarball (`.tar.{gz,bz2,xz}`) を含んでいます。ネイティブではないパッケージのソースパッケージは Debian のソース control ファイルと、オリジナルのソースコードの tarball (`.orig.tar.{gz,bz2,xz}`)、そして Debian での変更点 (ソース形式" 1.0 "は `.diff.gz`、ソース形式" 3.0 (quilt) "は `.debian.tar.{gz,bz2,xz}`) を含んでいます。

ソース形式 " 1.0 " では、パッケージが native かどうかはビルド時に `dpkg-source` によって決められていました。最近では望むソース形式を `debian/source/format` に " 3.0 (quilt) " または " 3.0 (native) " と記述することによって明示することが推奨されています。この章の残りの部分は native ではないパッケージについてのみ記しています。

The first time a version is uploaded that corresponds to a particular upstream version, the original source tar file must be uploaded and included in the `.changes` file. Subsequently, this very same tar file should be used to build the new diffs and `.dsc` files, and will not need to be re-uploaded.

デフォルトでは、`dpkg-genchanges` および `dpkg-buildpackage` は前述されている changelog エントリと現在のエントリが異なった upstream バージョンを持つ場合にのみ、オリジナルのソース tar ファイルを含めようとします。この挙動は、`-sa` を使って常に含めたり、`-sd` を使うことで常に含めないようにするように変更できます。

アップロード時にオリジナルのソースが含まれていない場合、アップロードされる `.dsc` と diff ファイルを構築する際に `dpkg-source` が使用するオリジナルの tar ファイルは、**必ず**既にアーカイブにあるものと 1 バイトも違わぬものでなくてはなりません。

Please notice that, in non-native packages, permissions on files that are not present in the `*.orig.tar.{gz,bz2,xz}` will not be preserved, as diff does not store file permissions in the patch. However, when using source format " 3.0 (quilt) ", permissions of files inside the `debian` directory are preserved since they are stored in a tar archive.

5.5 ディストリビューションを選ぶ

アップロードでは、パッケージがどのディストリビューション向けになっているかを指定してあることが必要です。パッケージの構築プロセスでは、`debian/changelog` ファイルの最初の行からこの情報を展開し、`.changes` ファイルの `Distribution` 欄に配置します。

パッケージは、通常 `unstable` へアップロードされます。`unstable` あるいは `experimental` へのアップロードはこれらの suite を changelog のエントリに記します。サポートされている他の suite へのアップロードは、曖昧さを避けるために suite のコードネームを使う必要があります。

実際には、他にも指定可能なディストリビューションがあります: `codename-security` ですが、その詳細については **セキュリティ関連バグの取扱い** を読んでください。

同時に複数のディストリビューションへ、パッケージをアップロードすることはできません。

5.5.1 特別な例: 安定版 (stable) と 旧安定版 (oldstable) ディストリビューションへアップロードする

安定版 (`stable`) へのアップロードは、安定版リリースマネージャによるレビューのため、パッケージは `proposed-updates-new` キューに転送され、許可された場合は Debian アーカイブの `stable-proposed-updates` ディレクトリにインストールされます。ここから、ここから、安定版 (`stable`) の次期ポイントリリースに含まれることになります。

Uploads to a supported `stable` release should target their suite name in the changelog, i.e. `trixie` or `bookworm`. You should normally use `reportbug` and the `release.debian.org` pseudo-package to send a *source* `debdiff`, rationale and associated bug numbers to the stable release managers, and await a request to upload or further information.

If you are confident that the upload will be accepted without changes, please feel free to upload at the same time as filing the `release.debian.org` bug. However if you are new to the process, we would recommend getting approval before uploading so you get a chance to see if your expectations align with ours.

Either way, there must be an accompanying bug for tracking, and your upload must comply with these acceptance criteria defined by the stable release managers. These criteria are designed to help the process be as smooth and frustration-free as possible.

- The bug you want to fix in `stable` must be fixed in `unstable` already (and not waiting in NEW or the delayed queue).
- The bug should be of severity "important" or higher.
- Bug meta-data - particularly affected versions - must be up to date.
- Fixes must be minimal and relevant and include a sufficiently detailed changelog entry.
- A source debdiff of the proposed change must be included in your request (not just the raw patches or "a debdiff can be found at $URL").

- The proposed package must have a correct version number (e.g. `...+deb13u1/...~deb13u1` for `trixie` or `+deb12u1/~deb12u1` for `bookworm`) and you should be able to explain what testing it has had. See the Debian Policy for the version number: https://www.debian.org/doc/debian-policy/ch-controlfields.html#special-version-conventions
- The update must be built in an `stable` environment or chroot (or `oldstable` if you target that).
- Fixes for security issues should be co-ordinated with the security team, unless they have explicitly stated that they will not issue an `DSA` for the bug (e.g. via a "no-dsa" marker in the **セキュリティ追跡システム**).
- Do not close `release.debian.org` bugs in debian/changelog. They will be closed by the release team once the package has reached the respective point release.

It is recommended to use `reportbug` as it eases the creation of bugs with correct meta-data. The release team makes extensive use of usertags to sort and manage requests and incorrectly tagged reports may take longer to be noticed and processed.

旧安定版 (`oldstable`) ディストリビューションへのアップロードはアーカイブされてない限り可能です。安定版 (`stable`) と同じルールが適用されます。

In the past, uploads to `stable` were used to address security problems as well. However, this practice is deprecated, as uploads used for Debian security advisories (`DSA`) are automatically copied to the appropriate `proposed-updates` archive when the advisory is released. See **セキュリティ関連バグの取扱い** for detailed information on handling security problems. If the security team deems the problem to be too benign to be fixed through a `DSA`, the stable release managers are usually willing to include your fix nonetheless in a regular upload to `stable`.

5.5.2 Special case: the `stable-updates` suite

Sometimes the stable release managers will decide that an update to stable should be made available to users sooner than the next scheduled point release. In such cases, they can copy the update to the `stable-updates` suite, use of which is enabled by the installer by default.

Initially, the process described in **特別な例: 安定版 *(stable)* と 旧安定版 *(oldstable)* ディストリビューションへアップロードする**. should be followed as usual. If you think that the upload should be released via `stable-updates`, mention this in your request. Examples of circumstances in which the upload may qualify for such treatment are:

- The update is urgent and not of a security nature. Security updates will continue to be pushed through the security archive. Examples include packages broken by the flow of time (c.f. `spamassassin` and the year 2010 problem) and fixes for bugs introduced by point releases.
- The package in question is a data package and the data must be updated in a timely manner (e.g. `tzdata`).
- Fixes to leaf packages that were broken by external changes (e.g. video downloading tools and `tor`).
- Packages that need to be current to be useful (e.g. `clamav`).
- Uploads to `stable-updates` should target their suite name in the changelog as usual, e.g. `trixie`.

Once the upload has been accepted to `proposed-updates` and is ready for release, the stable release managers will then copy it to the `stable-updates` suite and issue a Stable Update Announcement (`SUA`) via the `debian-stable-announce` mailing list.

Any updates released via `stable-updates` will be included in `stable` with the next point release as usual.

5.5.3 特別な例: `testing/testing-proposed-updates` へアップロードする

詳細については、**直接テスト版を更新する** にある情報を参照してください。

5.6 パッケージをアップロードする

5.6.1 Source and binary uploads

Each upload to Debian consists of a signed `.changes` file describing the requested change to the archive, plus the source and binary package files that are referenced by the `.changes` file.

If possible, the version of a package that is uploaded should be a source-only changes file. These are typically named `*_source.changes`, and reference the source package, but no binary `.deb` or `.udeb` packages. All of the corresponding architecture-dependent and architecture-independent binary packages, for all architectures, will be built automatically by the build daemons in a controlled and predictable environment (see *wanna-build* for more details).

For many source-only uploads you can use tag2upload instead, which means that you only need to push a signed Git tag to Salsa, instead of generating and signing `.dsc` and `.changes` files yourself. See https://wiki.debian.org/tag2upload.

There are several situations where a source-only upload is not possible.

The first upload of a new source package (see **新規パッケージ**) must include binary packages, so that they can be reviewed by the archive administrators before they are added to Debian.

If new binary packages are added to an existing source package, then the first upload that lists the new binary packages in `debian/control` must include binary packages, again so that they can be reviewed by the archive administrators before they are added to Debian. It is preferred for these uploads to be done via the `experimental` suite.

Uploads that will be held for review in other queues, such as packages being added to the `*-backports` suites, might also require inclusion of binary packages.

The build daemons will automatically attempt to build any `main` or `contrib` package for which the build-dependencies are available. Packages in `non-free` and `non-free-firmware` will not be built by the build daemons unless the package has been marked as suitable for auto-building (see *non-free* **のパッケージを** *auto-build* **可能であるとマークする**).

The build daemons only install build-dependencies from the `main` archive area. This means that if a source package has build-dependencies that are in the `contrib`, `non-free` or `non-free-firmware` archive areas, then uploads of that package need to include prebuilt binary packages for every architecture that will be supported. By definition this can only be the case for source packages that are themselves in the `contrib`, `non-free` or `non-free-firmware` archive areas.

Bootstrapping a new architecture, or a new version of a package with circular dependencies (such as a self-hosting compiler), will sometimes also require an upload that includes binary packages.

5.6.2 ftp-master にアップロードする

To upload a package, you should upload the files (including the signed changes and dsc file) with anonymous ftp to `ftp.upload.debian.org` in the directory /pub/UploadQueue/. To get the files processed there, they need to be signed with a key in the Debian Developers keyring or the Debian Maintainers keyring (see https://wiki.debian.org/DebianMaintainer).

changes ファイルは最後に転送する必要があることに注意してください。そうしないとアーカイブのメンテナンスを行っているソフトが changes ファイルをパースして全てのファイルがアップロードされていないと判断して、アップロードは reject されるかもしれません。

パッケージのアップロードを行う際には *dupload* や *dput* が便利なことにも気づくことでしょう。これらの便利なプログラムは、パッケージを Debian にアップロードする作業を自動化するのに役立ちます。

パッケージを削除もしくはアップロードを取り消すには ftp://ftp.upload.debian.org/pub/UploadQueue/README と *dcut* Debian パッケージを参照してください。

Finally, you should think about the status of your package with relation to `testing` before uploading to `unstable`. If you have a version in `unstable` waiting to migrate then it is generally a good idea to let it migrate before uploading another new version. You should also check the *Debian* **パッケージトラッカー** for transition warnings to avoid making uploads that disrupt ongoing transitions.

5.6.3 遅延アップロード

It is sometimes useful to upload a package immediately, but to want this package to arrive in the archive only a few days later. For example, when preparing a *Non-Maintainer Upload (NMU)*, you might want to give the maintainer a few days to react, while at the same time having the certainty that your upload will eventually take effect if the maintainer does not. A one-shot approach of (i) file bug with a patch (or Merge Request) and (ii) at once upload to DELAYED, is much more efficient than having to wait for days between the two steps.

delayed ディレクトリにアップロードされると、パッケージは the deferred uploads queue に保存されます。指定した待ち時間が終わると、パッケージは処理のため通常の incoming ディレクトリに移動されます。この作業は `ftp.upload.debian.org` の `DELAYED/`*X*`-day` ディレクトリへのアップロードを通じて自動的に処理されます (*X* は 0 から 15 の間です)。0-day は一日に複数回 `ftp.upload.debian.org` へアップロードするのに使われます。

dput を使うと、パッケージを遅延キューに入れるのに `--delayed`*DELAY* パラメータを使えます。

Sometimes you'll need to remove a previously uploaded package from the DELAYED queue (e.g. because the package maintainer has asked you so). dcut(1) has examples of how to do so.

5.6.4 セキュリティアップロード

Do **NOT** upload a package to the security upload queue (on `*.security.upload.debian.org`) without prior authorization from the security team. If the package does not exactly meet the team's requirements, it will cause many problems and delays in dealing with the unwanted upload. For details, please see **セキュリティ関連バグの取扱い**.

5.6.5 他のアップロードキュー

ヨーロッパにはもう一つのアップロードキューが ftp://ftp.eu.upload.debian.org/pub/UploadQueue/ にあります。操作方法は ftp.upload.debian.org と同じですが、ヨーロッパ圏の開発者に対しては、より速いはずです。

パッケージは ssh を使って ssh.upload.debian.org へアップロードすることも可能です。ファイルは /srv/upload.debian.org/UploadQueue に置く必要があります。このキューは**遅延アップロード**をサポートしていません。

5.6.6 Notifications

Debian アーカイブメンテナはパッケージのアップロードに関して責任を持っています。多くの部分は、アップロードはアーカイブ用のメンテナンスツール dak process-upload によって日々自動的に行われています。特に、不安定版 (unstable) に存在しているパッケージの更新は自動的に処理されます。それ以外の場合、特に新規パッケージの場合は、アップロードされたパッケージをディストリビューションに含めるのは手動で行われます。アップロードが手動で処理される場合は、アーカイブへの変更は実施されるまでに一ヶ月ほどかかります。お待ちください。

どの場合であっても、パッケージがアーカイブに追加されたことや、バグがアップロードで閉じられたことを告げるメールでの通知を受け取ることになります。あなたが閉じようとしたバグが処理されてない場合は、この通知を注意深く確認してください。

インストール通知は、パッケージがどのセクションに入ったかを示す情報を含んでいます。不一致がある場合は、それを示す別のメール通知を受け取ります。以下も参照ください。

キュー経由でアップロードした場合は、キューデーモンソフトウェアもメールで通知を行うことに留意してください。

Also note that new uploads are announced on the *IRC* **チャンネル** channel #debian-devel-changes. If your upload fails silently, it could be that your package is improperly signed, in which case you can find more explanations on ssh.upload.debian.org:/srv/upload.debian.org/queued/run/log.

5.7 パッケージのセクション、サブセクション、優先度を指定する

debian/control ファイルの セクション (Section) フィールドと 優先度 (Priority) フィールドは実際にアーカイブ内でどこに配置されるか、あるいはプライオリティが何かという指定ではありません。debian/control ファイル中の値は、実際のところは単なるヒントです。

アーカイブメンテナは、override ファイル内でパッケージについて定められたセクションと優先度を常に確認しています。override ファイルと debian/control で指定されたパッケージのフィールドに不一致がある場合、パッケージがアーカイブにインストールされる際に相違について記述されたメールを受け取ります。debian/control ファイルを次回のアップロード時に修正することもできますし、override ファイルに変更を加えるように依頼するのもよいでしょう。

パッケージが現状で置かれているセクションを変更するには、まずパッケージの debian/control ファイルが正しいことを確認する必要があります。次に、ftp.debian.org に対し、あなたのパッケージに対するセクションあるいは優先度について古いものから新しいものへ変更する依頼のバグ登録をします。override: PACKAGE1:section/priority, [...], PACKAGEX:section/priority のようなサブジェクトを使い、バグ報告の本文に変更に関する根拠を記述してください。

override ファイル についての詳細は、dpkg-scanpackages 1 と https://www.debian.org/Bugs/Developer#maintincorrect を参照してください。

セクション で書かれているように、セクション (Section) フィールドにはセクション同様にサブセクションも記述するのに注意ください。セクションが main の場合は、それは書かないようにしてください。利用可能なサブセクションは https://www.debian.org/doc/debian-policy/ch-archive.html#s-subsections で検索できます。

5.8 New upstream versions

To update a package for a new upstream release:

1. Read the upstream changelog, NEWS, and whatever other documentation they may have released with the new version.
2. If possible, inspect the full diff between the old and new upstream sources, potentially filtering out uninteresting parts using filterdiff (e.g. `filterdiff -x '*.po'`). If it's too big to review thoroughly, you can use `diffstat` to get a feel for the scope and nature of the changes (and thus where new bugs may appear), and to keep an eye for anything suspicious (e.g. unexpected use of network or the appearance of dubious binary blobs).
3. Port the old Debian packaging to the new version. This basically involves incrementing the `debian/changelog` and merging `debian/patches` from the old package to the new one.
4. Check to see if any bugs have been fixed that are currently open in the BTS. If they have been, close them in the `debian/changelog`.
5. If the patch/merge did not apply cleanly, figure out why. A patch may fail to apply if it's already been applied in the new upstream release, or if the upstream files the patch applies to have been substantially modified (or deleted).
6. If any changes were made to the build system (you'd know from steps 1 and 2), update the `debian/rules` and `debian/control` build dependencies if necessary.
7. Build the new package in an isolated environment, e.g. using `sbuild` or `pbuilder`. This ensures that all required build dependencies are listed in `debian/control` and eliminates the possibility of interference of any third party packages in your system.
8. Verify that the new package builds correctly and if so carry out the checks in **パッケージをテストする**.

5.9 バグの取扱い

すべての開発者は Debian バグ追跡システムを取り扱えるようでなければいけません。これは、どの様にしてバグ報告を正しく登録するか (**バグ報告** 参照)、どの様に更新及び整理するか、そしてどの様にして処理をして完了するかを知っていることを含みます。

バグ追跡システムの機能は、Debian BTS 開発者向け情報に記載されています。これには、バグの完了処理・追加メッセージの送信・重要度とタグを割り当てる・バグを転送済み (Forwarded) にする・その他が含まれています。

バグを他のパッケージに割り当てし直す、同じ問題についての別々のバグ報告をマージする、早まってクローズされたバグの再オープンなどの作業は、いわゆる制御メールサーバと呼ばれるものを使って処理さ

れています。このサーバで利用可能なすべてのコマンドは、BTS 制御サーバドキュメントに記載されています。

5.9.1 バグの監視

良いメンテナになりたい場合は、あなたのパッケージに関する Debian バグ追跡システム (BTS) のページを定期的にチェックする必要があります。BTS には、あなたのパッケージに対して登録されている全てのバグが含まれています。登録されているバグについては、以下のページを参照することで確認できます: `https://bugs.debian.org/`*yourlogin*`@debian.org`

メンテナは、`bugs.debian.org` のメールアドレス経由で BTS に対応します。利用可能なコマンドについてのドキュメントは https://www.debian.org/Bugs/ で参照可能ですし、もし `doc-debian` パッケージをインストールしてあれば、ローカルファイル `/usr/share/doc/debian/bug-*` で見ることも可能です。

定期的にオープンになっているバグについてのレポートを受け取るのも良いでしょう。あなたのパッケージでオープンになっているバグの全一覧を毎週受け取りたい場合、以下のような cron ジョブを追加します:

```
# ask for weekly reports of bugs in my packages
0 17 * * fri    echo "index maint address" | mail request@bugs.debian.org
```

address は、あなたの公式な Debian パッケージメンテナとしてのメールアドレスに置き換えてください。

5.9.2 バグへの対応

When responding to bugs, make sure that any discussion you have about bugs is sent to the original submitter of the bug, the bug itself and (if you are not the maintainer of the package) the maintainer. Sending an email to *123*`@bugs.debian.org` will send the mail to the maintainer of the package and record your email with the bug log. If you don't remember the submitter email address, you can use *123*`-submitter@bugs.debian.org` to also contact the submitter of the bug. The latter address also records the email with the bug log, so if you are the maintainer of the package in question, it is enough to send the reply to *123*`-submitter@bugs.debian.org`. Otherwise you should include *123*`@bugs.debian.org` so that you also reach the package maintainer.

FTBFS である旨のバグを受け取った場合、これはソースからビルドできないこと (Fails to build from source) を意味します。移植作業をしている人たちはこの略語をよく使います。

既にバグに対処していた場合 (例えば修正済みの時)、説明のメッセージを *123*`-done@bugs.debian.org` に送ることで `done` とマークしておいて (閉じて) ください。パッケージを変更してアップロードすることでバグを修正する場合は、**新規アップロードでバグがクローズされる時** に記載されているように自動的にバグを閉じることができます。

`close` コマンドを `control@bugs.debian.org` に送って、バグサーバ経由でバグを閉じるのは**決して**してはいけません。そのようにした場合、元々の報告者は何故バグが閉じられたのかという情報を得られません。

5.9.3 バグを掃除する

パッケージメンテナになると、他のパッケージにバグを見つけたり、自分のパッケージに対して報告されたバグが実際には他のパッケージにあるバグであったりということが頻繁にあるでしょう。バグ追跡システムの機能は Debian 開発者向けの BTS ドキュメント に記載されています。バグ報告の再指定 (reassign)

やマージ (merge)、そしてタグ付けなどの作業は BTS 制御サーバのドキュメント に記述されています。この章では、Debian 開発者から集められた経験を元にしたバグの扱い方のガイドラインを含んでいます。

他のパッケージで見つけた問題についてバグを登録するのは、メンテナとしての責務の一つです。詳細については **バグ報告** を参照してください。しかし、自分のパッケージのバグを管理するのはさらに重要です。

以下がバグ報告を取り扱う手順です:

1. 報告が実際にバグに関連するものか否かを決めてください。ユーザはドキュメントを読んでいないため、誤ったプログラムの使い方をしているだけのことが時々あります。このように判断した場合は、ユーザに問題を修正するのに十分な情報を与えて (良いドキュメントへのポインタを教えるなどして) バグを閉じます。同じ報告が何度も繰り返し来る場合には、ドキュメントが十分なものかどうか、あるいは有益なエラーメッセージを与えるよう、誤った使い方を検知していないのでは、と自身に問い直してください。これは開発元の作者に伝える必要がある問題かもしれません。

   If the bug submitter disagrees with your decision to close the bug, they may reopen it until you find an agreement on how to handle it. If you don't find any, you may want to tag the bug `wontfix` to let people know that the bug exists but that it won't be corrected. Please make sure that the bug submitter understands the reasons for your decision by adding an explanation to the message that adds the `wontfix` tag.

   If this situation is unacceptable, you (or the submitter) may want to require a decision of the technical committee by filing a new bug against the `tech-ctte` pseudo-package with a summary of the situation. Before doing so, please read the recommended procedure.

2. バグが実際にあるが、他のパッケージによって引き起こされている場合は、バグを正しいパッケージに再指定 (reassign) します。どのパッケージに再指定するべきかが分からない場合は、*IRC* **チャンネル** か `debian-devel@lists.debian.org` で聞いてください。再指定するパッケージのメンテナに通知をしてください。例えば *packagename*`@packages.debian.org` 宛にメッセージを Cc: してメール中に理由を説明するなどします。単に再指定しただけでは再指定された先のメンテナにはメールは **送信されません**ので、彼らがパッケージのバグ一覧を見るまでそれを知ることはありません。

   バグがあなたのパッケージの動作に影響する場合は、バグを複製し (clone)、複製したバグをその挙動を実際に起こしているパッケージに再指定することを検討してください。さもなければ、あなたのパッケージのバグ一覧にバグが見つからないので、多分ユーザに同じバグを何度も繰り返し報告される羽目になる可能性があります。あなたは、再指定 (reassign) によって「自分の」バグということを防ぎ、バグの複製 (clone) によって関係があることを記載しておく必要があります。

3. 時々、重要度の定義に合うようにバグの重要度を調整する必要もあります。これは、人々はバグ修正を確実に早くしてもらうために重要度を極端に上げようとするからです。要望された変更点が単に体裁的なものな時には、バグは要望 (wishlist) に格下げされるでしょう。

4. バグが確かにあるが既に他の誰かによって同じ問題が報告されている場合は、2 つの関連したバグを BTS の merge コマンドを使って 1 つにマージします。このようにすると、バグが修正された時に全ての投稿者に通知がいきます (ですが、そのバグ報告の投稿者へのメールは報告の他の投稿者には自動的には通知されないことに注意してください)。merge コマンドや類似の unmerge コマンドの技術詳細については、BTS 制御サーバドキュメントを参照してください。

5. バグ報告者は情報を書き漏らしている場合、必要な情報を尋ねる必要があります。その様なバグに印をつけるには `moreinfo` タグを使います。さらに、そのバグを再現できない場合には、`unreproducible` タグを付けます。誰もそのバグを再現できない場合、どうやって再現するのか、さらに情報を何ヶ月経っても、この情報が誰からも送られてこない場合はバグは閉じても構いません。

6. If the bug is related to the packaging, you just fix it. If you are not able to fix it yourself, then tag the bug as `help`. You can also ask for help on `debian-devel@lists.debian.org` or `debian-qa@lists.debian.org`. See **開発元/上流** *(upstream)* **の開発者との調整** if it's an upstream problem. If you have the required skills you can prepare a patch that fixes the bug and send it to the author at the same time. Make sure to send the patch to the BTS and to tag the bug as `patch`.

7. ローカル環境でバグを修正した、あるいは VCS リポジトリに修正をコミットした場合には、バグに `pending` タグを付けてバグが修正されたことと次のアップロードでバグが閉じられるであろうことを回りに知らせます (`changelog` に `closes:` を追加します)。これは複数の開発者が同一のパッケージで作業している際に特に役立ちます。

8. Once a corrected package is available in the archive, the bug should be closed indicating the version in which it was fixed. This can be done automatically; read **新規アップロードでバグがクローズされる時**.

5.9.4 新規アップロードでバグがクローズされる時

バグや問題があなたのパッケージで修正されたとしたら、そのバグを閉じるのはパッケージメンテナとしての責任になります。しかし、バグを修正したパッケージが Debian アーカイブに受け入れられるまではバグを閉じてはいけません。従って、一旦更新したパッケージがアーカイブにインストールされたという通知を受け取った場合は、BTS でバグを閉じることができますし、そうしなければいけません。もちろん、バグは正しいバージョンで閉じなくてはいけません。

ですが、アップロード後に手動でバグをクローズしなくても済む方法があります— `debian/changelog` に以下の特定の書き方にて修正したバグを列挙すれば、それだけで後はアーカイブのメンテナンスソフトがバグをクローズしてくれます。例:

```
acme-cannon (3.1415) unstable; urgency=low

  * Frobbed with options (closes: Bug#98339)
  * Added safety to prevent operator dismemberment, closes: bug#98765,
    bug#98713, #98714.
  * Added man page. Closes: #98725.
```

技術的に言うと、どの様にしてバグを閉じる changelog が判別されているかを以下の Perl の正規表現にて記述しています:

```
/closes:\s*(?:bug)?\#?\s?\d+(?:,\s*(?:bug)?\#?\s?\d+)*/ig
```

We prefer the `closes: #`*XXX* syntax, as it is the most concise entry and the easiest to integrate with the text of the `changelog`. Unless specified differently by the `-v`-switch to `dpkg-buildpackage`, only the bugs closed in the most recent changelog entry are closed (basically, exactly the bugs mentioned in the changelog-part in the `.changes` file are closed).

歴史的に、*Non-Maintainer Upload (NMU)* と判別されるアップロードは closed ではなく `fixed` とタグがされてきましたが、この習慣はバージョントラッキングの進化によって廃れています。同じことが `fixed-in-experimental` タグにも言えます。

If you happen to mistype a bug number or forget a bug in the changelog entries, don't hesitate to undo any damage the error caused. To reopen wrongly closed bugs, send a `reopen` *XXX* command to the bug tracking system's control address, `control@bugs.debian.org`. To close any remaining bugs that were fixed by your upload, email

the `.changes` file to *XXX*`-done@bugs.debian.org`, where *XXX* is the bug number, and put Version: *YYY* and an empty line as the first two lines of the body of the email, where *YYY* is the first version where the bug has been fixed.

上に書いたような changelog を使ったバグの閉じ方は必須ではない、ということは念頭に置いておいてください。行ったアップロードとは無関係に単にバグを閉じたい、という場合は、説明をメールに書いて *XXX*`-done@bugs.debian.org` に送ってバグを閉じてください。そのバージョンのパッケージでの変更がバグに何も関係ない場合は、そのバージョンの changelog エントリではバグを**閉じないで**ください。

どのように changelog のエントリを書くのか、一般的な情報については *debian/changelog* **のベストプラクティス** を参照してください。

5.9.5 セキュリティ関連バグの取扱い

機密性が高いその性質上、セキュリティ関連のバグは注意深く取り扱わねばなりません。この作業をコーディネイトし、未処理のセキュリティ問題を追い続け、セキュリティ問題についてメンテナを手助けしたり修正自体を行い、セキュリティ勧告を出し、`security.debian.org` を維持するために Debian セキュリティチームが存在します。

Debian パッケージ中のセキュリティ関連のバグに気づいたら、あなたがメンテナであるかどうかに関わらず、問題に関する正確な情報を集め、まずは `team@security.debian.org` 宛にメールを出してセキュリティチームへ連絡を取ってください。お望みであれば、Debian セキュリティ担当窓口の鍵を使ってメールを暗号化できます。詳細は https://www.debian.org/security/faq#contact を参照してください。チームに問い合わせること無く 安定版 (`stable`) 向けのパッケージを**アップロードしないでください**。例として、役に立つ情報は以下のようなものになります:

- バグが既に公開されているか否か
- バグによって、どのバージョンが影響を受けると分かっているか。サポートされている Debian のリリース、ならびに テスト版 (`testing`) 及び 不安定版 (`unstable`) にある各バージョンをチェックしてください。
- 利用可能なものがあれば、修正内容 (パッチが特に望ましい)
- 自身で準備した修正パッケージ (まずは **セキュリティ問題に対処するパッケージを用意する** を読んで、debdiff の結果、あるいは `.diff.gz` と `.dsc` ファイルだけを送ってください)
- テストについて何かしらの手助けになるもの (攻撃コード、リグレッションテストなど)
- 勧告に必要になる情報 (**セキュリティ勧告** 参照)

パッケージメンテナとして、あなたは安定版リリースについてもメンテナンスする責任を持ちます。あなたがパッチの評価と更新パッケージのテストを行うのに最も適任な人です。ですから、以下のセキュリティチームによって取り扱ってもらうため、どのようにしてパッケージを用意するかについての章を参照してください。

5.9.5.1 セキュリティ追跡システム

セキュリティチームは集約的なデータベース、Debian セキュリティ追跡システム (Debian Security Tracker) をメンテナンスしています。これはセキュリティ問題として知られている全ての公開情報を含んでいます: どのパッケージ／バージョンが影響を受ける／修正されているか、つまりは安定版、テスト版、不安定版が脆弱かどうか、という情報です。まだ機密扱いの情報は追跡システムには追加されません。

特定の問題について検索することもできますし、パッケージ名でも検索できます。あなたのパッケージを探して、どの問題がまだ未解決かを確認してください。できれば追加情報を提供するか、パッケージの問題に対処するのを手伝ってください。やり方は追跡システムのウェブページにあります。

5.9.5.2 秘匿性

Debian 内での他の多くの活動とは違い、セキュリティ問題に関する情報については、暫くの間秘密にしておく必要がしばしばあります。これによって、ソフトウェアのディストリビュータがユーザが危険にさらされるのを最小限にするため、公開時期を合わせることができます。今回がそうであるかは、問題と対応する修正の性質や、既に既知のものとなっているかどうかによります。

開発者がセキュリティ問題を知る方法はいくつかあります:

- 公開フォーラム (メーリングリスト、ウェブサイトなど) で知らせる
- 誰かがバグ報告を登録している
- 誰かがプライベートなメールで教えてきた

最初の二つのケースでは、情報は公開されていて可能な限り早く修正することが重要です。しかしながら最後のケースは、公開情報ではないかもしれません。この場合は、問題に対処するのに幾つか取り得る選択肢があります:

- セキュリティの影響度が小さい場合、問題を秘密にしておく必要はなく、修正を行ってリリースするのが良い場合がしばしばあります。
- 問題が深刻な場合、他のベンダと情報を共有してリリースをコーディネイトする方が望ましいでしょう。セキュリティチームは様々な組織／個人と連絡を取りつづけ、この問題に対応することができます。

どのような場合でも、問題を報告した人がこれを公開しないように求めているのであれば、明白な例外として Debian の安定版リリースに対する修正を作成してもらうためにセキュリティチームへ連絡すること以外、この様な要求は尊重されるべきです。機密情報をセキュリティチームに送る場合は、この点を明示しておくのを忘れないでください。

機密を要する場合は、修正を不安定版 (`unstable`) (や公開 VCS リポジトリなどその他どこへも) へ修正をアップロードしないよう、注意してください。コードその物が公開されている場合、変更の詳細を難読化するだけでは十分ではなく、皆によって解析され得る (そしてされる) でしょう。

機密であることを要求されたにも関わらず、情報を公開するのには 2 つの理由があります: 問題が一定期間既知の状態になっている、あるいは問題や攻撃コードが公開された場合です。

セキュリティチームは、機密事項に関して通信を暗号化できる PGP 鍵を持っています。詳細については、セキュリティチーム FAQ を参照してください。

5.9.5.3 セキュリティ勧告

セキュリティ勧告は現在のところ、リリースされた安定版ディストリビューションについてのみ、取り扱われます。テスト版 (`testing`) や 不安定版 (`unstable`) についてでは**ありません**。リリースされると、セキュリティ勧告は email-debian-security-announce; メーリングリストに送られ、セキュリティのウェブページに掲載されます。セキュリティ勧告はセキュリティチームによって記述、掲載されます。しかし、メン

テナが情報を提供できたり、文章の一部を書けるのであれば、彼らは当然そんなことは気にしません。勧告にあるべき情報は以下を含んでいます:

- 以下のようなものを含めた問題の説明と範囲:
  - 問題の種類 (権限の上昇、サービス拒否など)
  - 何の権限が得られるのか、(もし分かれば) 誰が得るのか
  - どのようにして攻撃が可能なのか
  - 攻撃はリモートから可能なのかそれともローカルから可能なのか
  - どのようにして問題が修正されたのか

  この情報によって、ユーザがシステムに対する脅威を評価できるようになります。
- 影響を受けるパッケージのバージョン番号
- 修正されたパッケージのバージョン番号
- どこで更新されたパッケージを得るかという情報 (通常は Debian のセキュリティアーカイブからです)
- References to upstream advisories, CVE identifiers, and any other information useful in cross-referencing the vulnerability

5.9.5.4 セキュリティ問題に対処するパッケージを用意する

あなたがセキュリティチームに対し、彼らの職務に関して手助けできる方法の一つは、安定版 Debian リリース用のセキュリティ勧告に適した修正版パッケージを提供することです。

When an update is made to the stable release, care must be taken to avoid changing system behavior or introducing new bugs. In order to do this, make as few changes as possible to fix the bug. Users and administrators rely on the exact behavior of a release once it is made, so any change that is made might break someone's system. This is especially true of libraries: make sure you never change the API (Application Program Interface) or ABI (Application Binary Interface), no matter how small the change.

これは、開発元の新しいリリースバージョン (new upstream version) への移行が良い解決策ではないことを意味しています。代わりに、関連する変更を現在の Debian 安定版リリースに存在しているバージョンへバックポートするべきです。通常、開発元のメンテナは助けが必要であれば手伝おうとしてくれます。そうでない場合は、Debian セキュリティチームが手助けすることができます。

いくつかのケースでは、例えば大量のソースコードの変更や書き直しが必要など、セキュリティ修正をバックポートできないことがあります。この様な場合、新しいバージョン (new upstream version) へ移行する必要があるかもしれません。しかし、これは極端な状況の場合にのみ行われるものであり、実行する前に必ずセキュリティチームと調整をしなければなりません。

これに関してはもう一つ重要な指針があります: 必ず変更についてテストをしてください。攻撃用コード (exploit) が入手可能な場合には、それを試してみて、パッチを当てていないパッケージで成功するか、修正したパッケージでは失敗することかどうかを確かめてみてください。他の確認として、セキュリティ修正は時折表面上はそれと関係が無いような機能を壊すことがあるので、通常の動作も同様にテストしてください。

脆弱性の修正に直接関係しない変更をパッケージへ**加えない**ようにしてください。この様な変更は元に戻さなくてはならなくなるだけで、時間を無駄にします。他に直したいバグがパッケージにある場合は、セ

キュリティ勧告が発行された後、通常通りに proposed-update にアップロードを行ってください。セキュリティ更新の仕組みは、それ以外の方法では安定版リリースから reject されるであろう変更をあなたのパッケージに加える方法ではありませんので、この様な事はしないでください。

変更点を可能な限り見直してください。以前のバージョンとの変更点を繰り返し確認してください (これには `patchutils` パッケージの `interdiff` や `devscripts` の `debdiff` が役立ちます。*debdiff* を参照してください)。

以下の項目を必ず確認してください

- `debian/changelog` で **正しいディストリビューションを対象にする**: *codename*`-security` (例えば `trixie-security`)。*distribution*`-proposed-updates` や `stable` を対象にしないでください!
- 説明が十分にされている、意味がある changelog エントリを書くこと。他の人は、これらを元に特定のバグが修正されているのかを見つけ出します。登録されている **Debian バグ** に対して `closes:` 行を追加すること。外部のリファレンス、できれば **CVE 識別番号** を常に含めること、そうすれば相互参照が可能になります。しかし、CVE 識別番号がまだ付与されていない場合には、それを待たずに作業を進めてください。識別番号は後ほど付与することができます。
- **バージョン番号**が正しいことを確認する。現在のパッケージより大きく、しかし以降のディストリビューションよりパッケージバージョンが小さい必要があります。分からない場合は `dpkg --compare-versions` でテストしてください。以前のアップロードで既に使っているバージョン番号を再利用しないように注意してください。そうしないと番号が binNMU と衝突します。`+deb`*X*`u1` (*X* はメジャーリリース番号) を追加するのが通例です。例えば `1:2.4.3-4+deb13u1` とします。もちろん 1 はアップロードするごとに増やします。
- これまでに (以前のセキュリティ更新によって) `security.debian.org` へ開発元のソースコードをアップロードしていなければ、**開発元のソースコードを全て含めて**アップロードするパッケージをビルドする (`dpkg-buildpackage -sa`)。以前、同じ開発元のバージョンで `security.debian.org` にアップロードしたことがある場合は、開発元のソースコード無しでアップロードしても構いません (`dpkg-buildpackage -sd`)。
- 通常のアーカイブで使われているのと**全く同じ** ``*.orig.tar.{gz,bz2,xz}`` を必ず使うようにしてください。さもなくば、後ほどセキュリティ修正を main アーカイブに移動することができません。
- ビルドを行っているディストリビューションからインストールしたパッケージだけが含まれている**クリーンなシステム上で**パッケージをビルドしてください。その様なシステムを自分で持っていない場合は、debian.org マシン (*Debian* **のマシン群** を参照してください) を使うこともできますし、chroot を設定することもできます (*pbuilder* と *debootstrap* を参照してください)。

5.9.5.5 修正したパッケージをアップロードする

Do **NOT** upload a package to the security upload queue (on `*.security.upload.debian.org`) without prior authorization from the security team. If the package does not exactly meet the team's requirements, it will cause many problems and delays in dealing with the unwanted upload.

セキュリティチームと調整する事無しに `proposed-updates` へ修正したものをアップロード**しない**ようにしてください。`security.debian.org` からパッケージは `proposed-updates` ディレクトリに自動的にコピーされます。アーカイブに同じ、あるいはより高いバージョン番号のものが既にインストールされている場合は、セキュリティアップデートはアーカイブシステムに reject されます。そうすると、安定版ディストリビューションはこのパッケージに対するセキュリティ更新無しで終了してしまうでしょう。

Once you have created and tested the new package and it has been approved by the security team, it needs to be uploaded so that it can be installed in the archives. For security uploads, the place to upload to is `ftp://ftp.security.upload.debian.org/pub/SecurityUploadQueue/`.

セキュリティキューへアップロードしたものが許可されると、パッケージは自動的にすべてのアーキテクチャに対してビルドされ、セキュリティチームによる確認の為に保存されます。

Uploads that are waiting for acceptance or verification are only accessible by the security team. This is necessary since there might be fixes for security problems that cannot be disclosed yet.

セキュリティチームのメンバーがパッケージを許可した場合は、proposed パッケージに対する `ftp-master.debian.org` 上の適切な *distribution*`-proposed-updates` と同様に `security.debian.org` 上にインストールされます。

5.10 Subscribing to package updates

As a maintainer of a package, you get email notifications for various kinds of events (uploads, BTS messages etc). You can subscribe to receive such notifications also for packages you are not a maintainer of (or packages you maintain collaboratively), by mailing `control@tracker.debian.org` with `subscribe <pkgname>` either in the subject or the body of the email (see https://qa.pages.debian.net/distro-tracker/usage/messages.html#email-messages for details).

5.11 パッケージの移動、削除、リネーム、放棄、引き取り、再導入

アーカイブの変更作業のいくつかは、Debian のアップロードプロセスでは自動的なものにはなっていません。これらの手続きはメンテナによる手動での作業である必要があります。この章では、この様な場合に何をするかのガイドラインを提供します。

5.11.1 パッケージの移動

時折、パッケージは属しているセクションが変わることがあります。例えば「`non-free`」セクションのパッケージが新しいバージョンで GPL になった場合、パッケージは「main」か「contrib」に移動する必要があります。[1]

パッケージのどれかがセクションを変更する必要がある場合、希望するセクションにパッケージを配置するためパッケージの control 情報を変更してから再アップロードします (詳細については Debian ポリシーマニュアルを参照してください)。必ず `.orig.tar.{gz,bz2,xz}` を (開発元のバージョンが新しいものになったのではなくても) アップロードに含める必要があります。新しいセクションが正しい場合は、自動的に移動されます。移動されない場合には、何が起こったのかを理解するために ftpmaster に連絡を取ります。

一方で、もしパッケージの一つのサブセクション (例:「devel」「admin」) を変更する必要がある、という場合には、手順は全く異なります。パッケージの control ファイルにあるサブセクションを修正して、再アップロードします。また、**パッケージのセクション、サブセクション、優先度を指定する** に記述してあるように override ファイルを更新する必要が出てくるでしょう。

[1] パッケージがどのセクションに属するかのガイドラインは Debian ポリシーマニュアルを参照してください。

5.11.2 パッケージの削除

If for some reason you want to completely remove a package (say, if it is an old compatibility library which is no longer required), you need to file a bug against `ftp.debian.org` asking that the package be removed; as with all bugs, this bug should normally have normal severity. The bug title should be in the form `RM:` *package [architecture list]* `--` *reason*, where *package* is the package to be removed and *reason* is a short summary of the reason for the removal request. *[architecture list]* is optional and only needed if the removal request only applies to some architectures, not all. Note that the `reportbug` will create a title conforming to these rules when you use it to report a bug against the `ftp.debian.org` pseudo-package.

If you want to remove a package you maintain, you should note this in the bug title by prepending `ROM` (Request Of Maintainer). There are several other standard acronyms used in the reasoning for a package removal; see https://ftp-master.debian.org/removals.html for a complete list. That page also provides a convenient overview of pending removal requests.

Note that removals can only be done for the `unstable`, `experimental` and `stable` distributions. Packages are not removed from `testing` directly. Rather, they will be removed automatically after the package has been removed from `unstable` and no package in `testing` depends on it. (Removals from `testing` are possible though by filing a removal bug report against the `release.debian.org` pseudo-package. See **テスト版からの削除**.)

例外として、明示的な削除依頼が必要ない場合が一つあります: (ソース、あるいはバイナリ) パッケージがソースからビルドされなくなった場合、半自動的に削除されます。バイナリパッケージの場合、これはこのバイナリパッケージを生成するソースパッケージがもはや存在しないということを意味します。バイナリパッケージがいくつかのアーキテクチャで生成されなくなったという場合には、削除依頼は必要です。ソースパッケージの場合は、関連の全バイナリパッケージが別のソースパッケージによって上書きされるのを意味しています。

削除依頼では、依頼を判断する理由を詳細に書く必要があります。これは不必要な削除を避け、何故パッケージが削除されたのかを追跡できるようにするためです。例えば、削除されるパッケージにとって代わるパッケージの名前を記述します。

通常は自分がメンテナンスしているパッケージの削除のみを依頼します。その他のパッケージを削除したい場合は、メンテナの許可を取る必要があります。パッケージが放棄されたのでメンテナがいない場合は、まず `debian-qa@lists.debian.org` で削除依頼について議論をしてください。パッケージの削除についての合意ができたら、削除依頼の新規バグを登録するのではなく、`wnpp` パッケージに対して登録されているバグを reassign して `O:` に題名を変更するべきです。

Further information relating to these and other package removal related topics may be found at https://wiki.debian.org/ftpmaster_Removalsand https://qa.debian.org/howto-remove.html.

パッケージを破棄しても構わないか迷う場合には、意見を聞きに `debian-devel@lists.debian.org` へメールしてください。`apt` の `apt-cache` プログラムも重要です。`apt-cache showpkg` **パッケージ名** として起動した際、プログラムは**パッケージ名**の被依存関係を含む詳細を表示します。他にも `apt-cache rdepends`、`apt-rdepends`、`build-rdeps` (`devscripts` パッケージに含まれる)、`grep-dctrl` などの有用なプログラムが非依存関係を含む情報を表示します。みなしご化されたパッケージの削除は、`debian-qa@lists.debian.org` で話し合われます。

一旦パッケージが削除されたら、パッケージのバグを処理する必要があります。実際のコードが別のパッケージに含まれるようになったので、別のパッケージへバグを付け替える (例えば、`libfoo13` が上書きするので、`libfoo12` が削除される) か、あるいはソフトウェアがもう Debian の一部では無くなった場合にはバグを閉じるかする必要があります。バグを閉じる場合、過去の Debian のリリースにあるパッケージバージョンで

修正されたとマークするのを避けてください。バージョン <most-recent-version-ever-in-Debian>+rm で修正されたとマークしなければなりません。

5.11.2.1 Incoming からパッケージを削除する

以前は、incoming からパッケージを削除することが可能でした。しかし、新しい incoming システムが導入されたことにより、これはもはや不可能となっています。[4] その代わりに、置き換えたいパッケージよりも高いバージョンのリビジョンの新しいパッケージをアップロードする必要があります。両方のバージョンのパッケージがアーカイブにインストールされますが、一つ前のバージョンはすぐに高いバージョンで置き換えられるため、実際にはバージョンが高い方だけが 不安定版 (unstable) で利用可能になります。しかし、もしあなたがパッケージをきちんとテストしていれば、パッケージを置き換える必要はそんなに頻繁には無いはずです。

5.11.3 パッケージをリプレースあるいはリネームする

あなたのパッケージのどれかの開発元のメンテナらが、ソフトウェアをリネームするのを決めた時 (あるいはパッケージを間違えて名前を付けた時)、以下の二段階のリネーム手続きに従う必要があります。最初の段階では、debian/control ファイルに新しい名前を反映し、利用しなくなるパッケージ名に対して Replace、Provides、Conflicts を設定する変更をします (詳細に関しては Debian ポリシーマニュアル 1 を参照)。注意してほしいのは、利用しなくなるパッケージ名がリネーム後も動作する場合のみ、Provides を付け加えるべきだということです。一旦パッケージをアップロードがアップロードされてアーカイブに移動したら、ftp.debian.org に対してバグを報告してください (**パッケージの削除** 参照)。同時にパッケージのバグを正しく付け替えするのを忘れないでください。

他に、パッケージの作成でミスを犯したので置き換えたいという場合があるかもしれません。これを行う方法は唯一つ、バージョン番号を上げて新しいバージョンをアップロードすることです。通常、古いバージョンは無効になります。これはソースを含めた各パッケージ部分に適用されることに注意してください: パッケージの開発元のソース tarball を入れ替えたい場合には、別のバージョンをつけてアップロードする必要があります。よくある例は foo_1.00.orig.tar.gz を foo_1.00+0.orig.tar.gz、あるいは foo_1.00.orig.tar.bz2 で置き換えるというものです。この制約によって、ftp サイト上で各ファイルが一意の名前を持つことになり、ミラーネットワークをまたがった一貫性を保障するのに役立ちます。

5.11.4 パッケージを放棄する

パッケージをもうメンテナンスできなくなってしまった場合、ほかの人に知らせて、パッケージが放棄 (orphaned) とマークされたのが分かるようにする必要があります。パッケージメンテナを Debian QA Group <packages@qa.debian.org> に設定し、疑似パッケージ wnpp に対してバグ報告を送信しなければなりません。バグ報告は、パッケージが今放棄されていることを示すように O: **パッケージ名**--**短い要約** というタイトルにする必要があります。バグの重要度は 通常 (normal) に設定しなければなりません; パッケージの重要 (priority) が standard より高い場合には重要 (important) に設定する必要があります。必要だと思うのならば、メッセージの X-Debbugs-CC: ヘッダのアドレスに debian-devel@lists.debian.org を入れてコピーを送ってください (そう、CC: を使わないでください。その理由は、CC: を使うと、メッセージの題名がバグ番号を含まないからです)。

[4] Though, if a package still is in the upload queue and hasn't been moved to Incoming yet, it can be removed. (see *ftp-master* **にアップロードする**)

パッケージを手放したいが、しばらくの間はメンテナンスを継続できる場合には、代わりに `wnpp` へ `RFA: パッケージ名--短い要約` という題名でバグ報告を送信する必要があります。`RFA` は `Request For Adoption` (引き取り依頼) を意味しています。

より詳細な情報は WNPP ウェブページにあります。

5.11.5 パッケージを引き取る

新たなメンテナが必要なパッケージの一覧は 作業が望まれるパッケージ (WNPP、Work-Needing and Prospective Packages list) で入手できます。WNPP でリストに挙がっているパッケージのどれかに対するメンテナンスを引き継ぎたい場合には、情報と手続きについては前述のページを確認してください。

It is not OK to simply take over a package without assent of the current maintainer– that would be package hijacking. You can, of course, contact the current maintainer and ask them for permission to take over the package.

However, when a package has been neglected by the maintainer, you might be able to take over package maintainership by following the package salvaging process as described in *Package Salvaging*. If you have reason to believe a maintainer is no longer active at all, see **活動的でない、あるいは連絡が取れないメンテナに対応する**.

Complaints about maintainers should be brought up on the developers' mailing list. If the discussion doesn't end with a positive conclusion, and the issue is of a technical nature, consider bringing it to the attention of the technical committee (see the technical committee web page for more information).

古いパッケージを引き継いだ場合は、おそらくバグ追跡システムでパッケージの公式メンテナとして表示されるようにしたいことでしょう。これは、一旦 `Maintainer` 欄を更新した新しいバージョンをアップロードすれば自動的に行われますが、アップロードが完了してから数時間はかかります。しばらくは新しいバージョンをアップロードする予定が無い場合は、*Debian* **パッケージトラッカー** を使ってバグ報告を受け取ることができます。しかし、以前のメンテナにしばらくの間はバグ報告が届き続けても問題無いことを確認してください。

To change the maintainer of a pseudo-package, e-mail `owner@lists.debian.org` and ask them to update their `pseudo-packages.maintainers` file. Include the package name, the name and e-mail address for the old and new maintainer, and a reason for the change. Remember to CC the old maintainer on the e-mail.

5.11.6 パッケージの再導入

パッケージは、リリースクリティカルのバグやメンテナ不在、不人気あるいは全体的な品質の低さ等により削除されることがよくあります。再導入プロセスはパッケージ化の開始時と似ていますが、あらかじめその歴史的経緯を調べておくことにより、落し穴にはまるのをいくらか避けることができます。

まず初めに、パッケージが削除された理由を確認しましょう。この情報はそのパッケージの PTS ページのニュースから削除の項目か削除ログを探すことにより見つけられます。削除のバグにはそのパッケージが削除された理由や、そのパッケージの再導入にあたって必要なことがいくらか提示されているでしょう。パッケージの再導入ではなくどこか他のソフトウェアの一部への乗り替えが最適であるということが提示されているかもしれません。

以前のメンテナに連絡を取り、パッケージの再導入のために作業していないか、パッケージ共同保守に関心はないか、必要になったときにパッケージのスポンサーをしてくれないか、等を確認しておくと良いでしょう。

新しいパッケージ (**新規パッケージ**) の導入前に必要なことは全てやりましょう。

利用できる中で適切な最新のパッケージをベースに作業しましょう。これは unstable の最新版かもしれません。また、snapshot アーカイブにはまだ存在するでしょう。

前のメンテナにより利用されていたバージョン管理システムに有用な変更が記録されているかもしれないので、確認してみるのは良いことです。以前のパッケージの control ファイルにそのパッケージのバージョン管理システムにリンクしているヘッダが無いか、それがまだ存在するか確認してください。

(testing や stable、oldstable ではなく) unstable からパッケージが削除されると、そのパッケージに関連するバグは全て閉じられます。閉じられたバグを全て (アーカイブされているバグを含めて) 確認し、+rm で終わるバージョンで閉じられていて現在でも有効なものを全て unarchive および reopen してください。有効ではなくなっているものは修正されているバージョンがわかればすべて修正済みとしてください。

Package removals from unstable also trigger marking the package as removed in the **セキュリティ追跡システム**. Debian members should mark removed issues as unfixed in the security tracker repository and all others should contact the security team to report reintroduced packages.

5.12 移植作業、そして移植できるようにすること

Debian がサポートするアーキテクチャの数は増え続けています。あなたが移植作業者ではない、あるいは別のアーキテクチャを使うことが無いという場合であっても、移植性の問題に注意を払うことはメンテナとしてのあなたの義務です。従って、あなたが移植作業者でなくても、この章の大半を読む必要があります。

Porting is the act of building Debian packages for architectures that are different from the original architecture of the package maintainer's binary package. It is a unique and essential activity. In fact, porters do most of the actual compiling of Debian packages. For instance, when a maintainer uploads a (portable) source package with binaries for the i386 architecture, it will be built for each of the other architectures, amounting to 8 more builds.

5.12.1 移植作業者に対して協力的になる

移植作業者は、難解かつ他には無いタスクを抱えています。それは、彼らは膨大な量のパッケージに対処する必要があるからです。理想を言えば、すべてのソースパッケージは変更を加えないできちんとビルドできるべきです。残念なことに、その様な場合はほとんどありません。この章は Debian メンテナによってよくコミットされる「潜在的な問題」のチェックリストを含んでいます– よく移植作業者を困らせ、彼らの作業を不必要に難解にする共通の問題です。

The first and most important thing is to respond quickly to bugs or issues raised by porters. Please treat porters with courtesy, as if they were in fact co-maintainers of your package (which, in a way, they are). Please be tolerant of succinct or even unclear bug reports; do your best to hunt down whatever the problem is.

移植作業者が遭遇する問題のほとんどは、何といっても、ソースパッケージ内での**パッケージ作成のバグ**によって引き起こされます。以下は、確認あるいは注意すべき項目のリストです。

1. Make sure that your Build-Depends and Build-Depends-Indep settings in debian/control are set properly. The best way to validate this is to use the debootstrap package to create an unstable chroot environment (see *debootstrap*). Within that chrooted environment, install the build-essential package and any package dependencies mentioned in Build-Depends and/or Build-Depends-Indep. Finally, try building your package within that chrooted environment. These steps can be automated by the use of the pbuilder program, which is provided by the package of the same name (see *pbuilder*).

chroot を正しく設定できない場合は、dpkg-depcheck が手助けになることでしょう (*dpkg-depcheck* 参照)。

ビルドの依存情報の指定方法については、Debian ポリシーマニュアルを参照してください。

2. 意図がある場合以外は、アーキテクチャの値を all または any 以外に指定しないでください。非常に多くの場合、メンテナが Debian ポリシーマニュアルの指示に従っていません。アーキテクチャを単一のものに指定する (i386 あるいは amd64 など) は大抵誤りです。

3. ソースパッケージが正しいことを確かめてください。ソースパッケージが正しく展開されたのを確認するため、dpkg-source -x*package*.dsc を実行してください。そして、ここでは、一からパッケージを dpkg-buildpackage でビルドするのに挑戦してみてください。

4. debian/files や debian/substvars を含んだソースパッケージを出していないかを確かめてください。これらは、debian/rules の clean ターゲットによって削除されるべきです。

5. Make sure you don't rely on locally installed or hacked configurations or programs. For instance, you should never be calling programs in /usr/local/bin or the like. Try not to rely on programs being set up in a special way. Try building your package on another machine, even if it's the same architecture.

6. 構築中の既にインストールしてあるパッケージに依存しないでください (上記の話の一例です)。もちろん、このルールには例外はありますが、そのような場合には手動で一から環境を構築する必要があり、パッケージ作成マシンで自動的に構築することはできません。

7. 可能であれば、特定のバージョンのコンパイラに依存しないでください。もし無理であれば、その制約をビルドの依存関係に反映されているのを確認してください。だとしても異なったアーキテクチャでは時折異なったバージョンのコンパイラで統一されているので、それでも恐らく問題を引き起こすことになるでしょう。

8. debian/rules で、Debian ポリシーマニュアルが定めるように、binary-arch 及び binary-indep ターゲットに分かれて含まれていることを確かめてください。両方のターゲットが独立して動作するのを確かめてください。つまり、他のターゲットを事前に呼び出さなくても、ターゲットを呼び出せるのを確かめるということです。これをテストするには、dpkg-buildpackage -B を実行してください。

9. When you can't support your package on a particular architecture, you shouldn't use the Architecture field to reflect that (it's also a pain to maintain correctly). If the package fails to build from source, you can just let it be and interested people can take a look at the build logs. If the package would actually build, the trick is to add a Build-Depends on unsupported-architecture [the-not-supported-arch]. The buildds will not build the package as the build dependencies are not fulfilled on that arch. To prevent building on 32-bits architectures, the architecture-is-64-bit build dependency can be used, as architecture-is-little-endian can be used to prevent building on big endian systems.

5.12.2 移植作業者のアップロード (porter upload) に関するガイドライン

パッケージが移植作業を行うアーキテクチャで手を入れずに構築できるのであれば、あなたは幸運で作業は簡単です。この章は、その様な場合に当てはめられます: きちんとアーカイブにインストールされるために、どうやってバイナリパッケージを構築・アップロードするかを記述しています。他のアーキテクチャでコンパイルできるようにするため、パッケージにパッチを当てる必要がある場合は、実際のところ、ソース NMU を行なうので、代わりに **いつ、どうやって *NMU* を行うか** を参照してください。

移植作業者のアップロード (porter upload) は、ソースに何も変更を加えません。ソースパッケージ中のファイルには触る必要はありません。これは debian/changelog を含みます。

dpkg-buildpackage を dpkg-buildpackage -B -m*porter-email* として起動してください。もちろん、*porter-email* にはあなたのメールアドレスを設定します。これは debian/rules の binary-arch を使ってパッケージのバイナリ依存部分のみのビルドを行います。

If you are working on a Debian machine for your porting efforts and you need to sign your upload locally for its acceptance in the archive, you can run debsign on your .changes file to have it signed conveniently.

5.12.2.1 再コンパイル、あるいは binary-only NMU

時折、最初の移植作業者のアップロード作業は困難なものになります。パッケージが構築された環境があまり良くないからです (古すぎる、使われていないライブラリがある、コンパイラの問題、などなど…)。その場合には、更新した環境で再コンパイルする必要があるでしょう。しかし、この場合にはバージョンを上げる必要があり、古いおかしなパッケージは Debian アーカイブ中で入れ替えられることになります (現在利用可能なものよりバージョン番号が大きくない場合、dak は新しいパッケージのインストールを拒否します)。

binary-only NMU がパッケージをインストール不可能にしてしまっていないことを確認する必要があります。ソースパッケージが、dpkg の substitution 変数 $(Source-Version) を使って内部依存関係を生成しているアーキテクチャ依存パッケージとアーキテクチャ非依存パッケージを生成した場合に起こる可能性があります。

changelog の変更が必要かどうかに関わらず、これらは binary-only NMU と呼ばれます– この場合には、他の全アーキテクチャで古すぎるかどうかや再コンパイルが必要かなどを考える必要はありません。

このような再コンパイルは、特別な「magic」バージョン番号を付けるのを必要とするので、アーカイブのメンテナンスツールは、これを理解してくれます。新しい Debian バージョンで、対応するソースアップデートが無くても、です。これを間違えた場合、アーカイブメンテナは (対応するソースコードが欠落しているので) アップロードを拒否します。

再コンパイルのみの NMU への「magic」は、b **番号** という形式に従った、パッケージのバージョン番号に対するサフィックスを追加することで引き起こされます。例えば、再コンパイル対象の最新バージョンが 2.9-3 の場合、バイナリのみの NMU は 2.9-3+b1 というバージョンになる必要があります。最新のバージョンが 3.4+b1 (つまり、ネイティブパッケージで、前回が再コンパイルの NMU) の場合、バイナリのみの NMU は 3.4+b2 というバージョン番号にならねばいけません。[2]

最初の移植作業者のアップロード (porter upload) と同様に、パッケージのアーキテクチャ依存部分をビルドするための dpkg-buildpackage の正しい実行の仕方は dpkg-buildpackage -B です。

5.12.2.2 あなたが移植作業者の場合、source NMU を行う時は何時か

移植作業者は、通常は非移植作業者同様に *Non-Maintainer Upload (NMU)* にあるガイドラインに沿ってソース NMU を行います。しかし、移植作業者のソース NMU に対する待ち時間は非移植作業者より小さくなります。これは、移植作業は大量のパッケージに対応する必要があるからです。さらに、状況はパッケージがアップロードされるディストリビューションに依って変わります。これは、アーキテクチャが次の安

[2] 過去においては、再コンパイルのみの状態を意味するために、このような NMU はリビジョンの Debian 部分の三つ目の番号を使っていました。しかし、この記法はネイティブパッケージの場合に曖昧で、同一パッケージでの再コンパイルのみの NMU と、ソース NMU と、セキュリティ NMU の正しい順序が付けられなかったため、この新しい記法で置き換えられました。

定版リリースに含められるかどうかによっても変わります。リリースマネージャはどのアーキテクチャが候補なのかを決定してアナウンスします。

あなたが不安定版 (unstable) へ NMU を行う移植作業者の場合、移植作業についての上記のガイドライン、そして 2 つの相違点に従う必要があります。まず、適切な待ち時間です— バグが BTS へ投稿されてから NMU を行って OK になるまでの間— 移植作業者が 不安定版 (unstable) ディストリビューションに対して行う場合は 7 日間になります。問題が致命的で移植作業に困難を強いるような場合には、この期間は短くできます (注意してください。この何れもがポリシーではなく、単にガイドラインに沿って相互に了解されているだけです)。安定版 (stable) や テスト版 (testing) へのアップロードについては、まず適切なリリースチームと調整をしてください。

次に、ソース NMU を行う移植作業者は BTS へ登録したバグの重要度が serious かそれ以上であることを確認してください。これは単一のソースパッケージが、すべての Debian でサポートされているアーキテクチャでコンパイルされたことをリリース時に保証します。数多くのライセンスに従うため、すべてのアーキテクチャについて、単一のバージョンのバイナリパッケージとソースパッケージを持つことがとても重要です。

移植作業者は、現在のバージョンのコンパイル環境やカーネル、libc にあるバグのために作られた単なる力業のパッチを極力回避すべきです。この様なでっち上げの代物があるのは、仕方がないことが時折あります。コンパイラのバグやその他の為にでっち上げを行う必要がある場合には、#ifdef で作業したものが動作することを確認してください。また、力業についてドキュメントに載せてください。一旦外部の問題が修正されたら、それを削除するのを皆が知ることができます。

移植作業者は、待ち期間の間、作業結果を置いておける非公式の置き場所を持つこともあります。移植版を動作させている人が、待ち期間の間であっても、これによって移植作業者による作業の恩恵を受けられるようになります。もちろん、この様な場所は、公式な恩恵や状況の確認を受けることはできませんので、利用者は注意してください。

5.12.3 移植用のインフラと自動化

パッケージの自動移植に役立つインフラストラクチャと複数のツールがあります。この章には、この自動化とこれらのツールへの移植の概要が含まれています。全体の情報に付いてはパッケージのドキュメントかリファレンスを参照してください。

5.12.3.1 メーリングリストとウェブページ

各移植版についての状況を含んだウェブページは https://www.debian.org/ports/ から参照できます。

Debian の各移植版はメーリングリストを持っています。移植作業のメーリングリストは https://lists.debian.org/ports.html で見ることができます。これらのリストは移植作業者の作業の調整や移植版のユーザと移植作業者をつなぐために使われています。

5.12.3.2 移植用ツール

移植用のツールの説明をいくつか **移植用ツール** で見ることができます。

5.12.3.3 wanna-build

The wanna-build system is used as a distributed, client-server build distribution system. It is usually used in conjunction with build daemons running the buildd program. Build daemons are slave hosts, which contact the central wanna-build system to receive a list of packages that need to be built.

wanna-build is not yet available as a package; however, all Debian porting efforts are using it for automated package building. The tool used to do the actual package builds, sbuild, is available as a package; see its description in *sbuild*. Please note that the packaged version is not the same as the one used on build daemons, but it is close enough to reproduce problems.

Most of the data produced by wanna-build that is generally useful to porters is available on the web at https://buildd.debian.org/. This data includes nightly updated statistics, queueing information and logs for build attempts.

我々はこのシステムを極めて誇りに思っています。何故ならば、様々な利用方法の可能性があるからです。独立した開発グループは、実際に一般的な用途に合うかどうか分からない異なった別アプローチの Debian にシステムを使うことができます (例えば、gcc の配列境界チェック付きでビルドした Debian など)。そして、Debian がディストリビューション全体を素早く再コンパイルできるようにもなります。

buildd の担当である wanna-build チームには、debian-wb-team@lists.debian.org で連絡が取れます。誰 (wanna-build チーム、リリースチーム) に連絡を取るのか、どうやって (メール、BTS) 連絡するのかを決めるには、https://lists.debian.org/debian-project/2009/03/msg00096.html を参照してください。

binNMU や give-back (ビルド失敗後のやり直し) を依頼する時には、https://release.debian.org/wanna-build.txt で記述されている形式を使ってください。

5.12.4 あなたのパッケージが移植可能なものではない場合

いくつかのパッケージでは、Debian でサポートされているアーキテクチャのうちの幾つかで、構築や動作に問題を抱えており、全く移植できない、あるいは十分な時間内では移植ができないものがあります。例としては、SVGA に特化したパッケージ (i386 と amd64 のみで利用可能) や、すべてのアーキテクチャではサポートされていないようなハードウェア固有の機能があります。

壊れたパッケージがアーカイブにアップロードされたり buildd の時間が無駄に費やされたりするのを防ぐため、幾つかしなければならないことがあります:

- まず、サポートできないアーキテクチャ上ではパッケージがビルドに**失敗する**のを確認しておく必要があります。これを行うには幾つかやり方があります。お勧めの方法は構築時に機能をテストする小さなテストスイートを用意して、動かない場合に失敗するようにすることです。これは、全てのアーキテクチャ上で、壊れたものをアップロードするのを防ぎ、必要な機能が動作するようになればパッケージがビルドできるようになる、良い考えです。

  さらに、サポートしているアーキテクチャ一覧が一定量であると信ずるのであれば、debian/control 内で any からサポートしているアーキテクチャの一覧に変更するべきです。この方法であれば、ビルドが同様に失敗するようになるのに加え、実際に試すことなく人間である読み手にサポートしているアーキテクチャが分かるようにできます。

- autobuilder が必要もなくパッケージをビルドしようとしないように、wanna-build スクリプトが使うリストである Packages-arch-specific に追加しておく必要があります。現在のバージョンは https://wiki.debian.org/PackagesArchSpecific から入手できます: 変更依頼をする相手はファイルの一番上を参照してください。

Please note that it is insufficient to only add your package to `Packages-arch-specific` without making it fail to build on unsupported architectures: A porter or any other person trying to build your package might accidentally upload it without noticing it doesn't work. If in the past some binary packages were uploaded on unsupported architectures, request their removal by filing a bug against `ftp.debian.org`.

5.12.5 non-free のパッケージを auto-build 可能であるとマークする

By default packages from the `non-free` and `non-free-firmware` sections are not built by the autobuilder network (mostly because the license of the packages could disapprove). To enable a package to be built, you need to perform the following steps:

1. 法的に許可されているか、技術的にパッケージが auto-build 可能かどうかを確認する;
2. `debian/control` のヘッダ部分に `XS-Autobuild: yes` を追加する;
3. メールを `non-free@buildd.debian.org` に送り、何故パッケージが合法的、かつ技術的に auto-build できるものなのかを説明する

5.13 Non-Maintainer Upload (NMU)

すべてのパッケージには最低一人のメンテナがいます。通常、この人達がパッケージに対して作業をし、新しいバージョンをアップロードします。時折、他の開発者らが新しいバージョンをアップロードできると便利な場合があります。例えば、彼らがメンテナンスしていないパッケージにあるバグを修正したいが、メンテナが問題に対応するのには助けが必要な場合です。このようなアップロードは *Non-Maintainer Upload (NMU)* と呼ばれます。

5.13.1 いつ、どうやって NMU を行うか

NMU を行う前に、以下の質問について考えてください:

- Have you geared the NMU towards helping the maintainer? As there might be disagreement on the notion of whether the maintainer actually needs help or not, the DELAYED queue exists to give time to the maintainer to react and has the beneficial side-effect of allowing for independent reviews of the NMU diff.
- Does your NMU really fix bugs? ("Bugs" means any kind of bugs, e.g. wishlist bugs for packaging a new upstream version, but care should be taken to minimize the impact to the maintainer.) Using NMUs to make changes that are likely to be non-consensual is discouraged.
- As more specific examples, the following changes are generally considered acceptable, unless there are good reasons for not following those practices in a particular package: using the latest released debhelper compatibility level; using dh; using 3.0 (quilt); using lintian-brush.
- メンテナに十分な時間を与えましたか? BTS にバグが報告されたのは何時ですか? 一、二週間忙しいことはあり得ないことでは無いです。そのバグはすぐに修正しなければならないほど重大ですか? それとも、あと数日待てるものですか?
- その変更にどれくらい自信がありますか? ヒポクラテスの誓いを思い出してください: 「何よりも、害を及ぼすことなかれ」 動かないパッチを当てるよりもパッケージの重大なバグをそのままオープンな状態にしておく方が良いですし、パッチによってバグを解決するのではなく隠してしまうかもしれません。自分が 100% 何をしたのか分かっていないのであれば、他の人からアドバイスをもらうの

も良い考えでしょう。NMU で何かを壊したのであれば、多くの人がとても不幸になるであろうことを覚えておいてください。

- 少なくとも BTS で、NMU するのを明確に表明しましたか? 何も反応が得られなかった場合、他の手段 (プライベートなメール、IRC) でメンテナに連絡をとるのも良い考えです。
- メンテナがいつも活動的で応答してくれる場合、彼に連絡を取ろうとしましたか? 大概の場合、メンテナ自身が問題に対応して、あなたのパッチをレビューする機会が与えられる方が好ましいと思われます。これは、NMU をする人が見落としているだろう潜在的な問題にメンテナは気付くことができるからです。大抵、メンテナが自分でアップロードする機会が与えられる方が、皆の時間を使うよりも良いやり方です。

NMU をする際には、まず NMU をする意図を明確にしておかねばなりません。それから、BTS へ現在のパッケージと提案する NMU との差分をパッチとして送付する必要があります。`devscripts` パッケージにある `nmudiff` スクリプトが役に立つでしょう。

While preparing the patch, you had better be aware of any package-specific practices that the maintainer might be using. Taking them into account reduces the burden of integrating your changes into the normal package workflow and thus increases the chances that integration will happen. A good place to look for possible package-specific practices is `debian/README.source`.

そうするべき十二分な理由が無い限り、メンテナに対応する時間を与えるべきです (例えば `DELAYED` キューにアップロードすることによってこれを行います)。以下が delayed キューを使う際のお勧めの値です:

- 7 日以上経っているリリースクリティカルバグのみを修正するアップロードで、バグに対するメンテナの活動が 7 日間見られなく、修正が行われている形跡が無い: 0 日
- 7 日以上経っているリリースクリティカルバグのみを修正するアップロード: 2 日
- リリースクリティカルバグや重要なバグの修正のみのアップロード: 5 日
- Other NMUs: 15 days

この値は例に過ぎません。セキュリティ問題を修正するアップロードや、移行を阻む些細なバグを修正するなど、いくつかのケースでは修正されたパッケージが 不安定版 (`unstable`) にすぐ入るようになるのは望ましいことです。

時々、リリースマネージャが特定のバグに対して短い delay 日数の NMU を許可を認めることがあります (7 日より古いリリースクリティカルバグなど)。また、一部のメンテナは Low Threshold NMU list に自身を挙げており、遅延なしの NMU アップロードを許可しています。しかしそのような場合であっても、特にパッチが BTS で以前手に入らなかったり、メンテナが大抵アクティブであるのを知っている場合など、アップロードの前にメンテナに対して数日与えるのは良い考えです。

After you upload an NMU, you are responsible for the possible problems that you might have introduced. You must keep an eye on the package (*Subscribing to package updates* is a good way to achieve this).

これは、軽率な NMU を行うための許可証ではありません。明らかにメンテナがアクティブで時期を逃さずパッチについて対応している場合や、このドキュメントに書かれている推奨を無視している場合など、あなたによるアップロードはメンテナと衝突を起こすでしょう。NMU のメリットについて、自分が行ったことの賢明さを常に弁護できるようにしておく必要があります。

5.13.2 NMU と debian/changelog

Just like any other (source) upload, NMUs must add an entry to `debian/changelog`, telling what has changed with this upload. The first line of this entry must explicitly mention that this upload is an NMU, e.g.:

```
* Non-maintainer upload.
```

NMU のバージョンのつけ方は、ネイティブなパッケージとネイティブではないパッケージでは異なります。

パッケージがネイティブパッケージの場合 (バージョン番号に Debian リビジョンが付かない)、バージョンはメンテナの最後のアップロードのバージョン + +nmu*X* となり、*X* は 1 から始まる数字になります。最後のアップロードが同様に NMU の場合は、数字を増やします。例えば、現在のバージョンが `1.5` だとすると、NMU はバージョンが `1.5+nmu1` になります。

パッケージがネイティブパッケージではない場合は、バージョン番号の Debian リビジョン部分 (最後のハイフン以下の部分) にマイナーバージョン番号を追加します。例えば、現在のバージョンが `1.5-2` であれば、NMU は `1.5-2.1` というバージョンになります。開発元のバージョンが新しくなったものが NMU でパッケージになった場合は、Debian リビジョンは `0` に設定されます。例えば `1.6-0.1` です。

どちらの場合でも、最後のアップロードも NMU だった場合には数字が増えます。例えば、現在のバージョンが `1.5+nmu3` (既に NMU されたネイティブパッケージ) の場合、NMU は `1.5+nmu4` というバージョンになります。

特別なバージョン付け方法が必要とされるのは、メンテナの作業を混乱させるのを避けるためです。何故ならば、Debian リビジョンのために整数を使っていると、NMU の際に既に準備されていたメンテナによるアップロードや、さらには ftp NEW queue にあるパッケージともぶつかる可能性があります。これには、アーカイブのパッケージが公式メンテナによるものではない、と視覚的に明らかにする利点もあります。

パッケージをテスト版や安定版にアップロードする場合、バージョン番号を「分岐」する必要が時々あります。これは例えばセキュリティアップロードが該当します。そのため、+deb*X*u*Y* 形式のバージョン番号を使うようにしてください。*X* はメジャーリリース番号で *Y* は 1 から始まるカウンターです。例えば、`trixie` (Debian 13) が安定版の間は安定版バージョン `1.5-3` のパッケージへのセキュリティ NMU ならバージョン `1.5-3+deb13u1` となりますが、`forky` へのセキュリティ NMU ではバージョン `1.5-3+deb14u1` となります。

5.13.3 DELAYED/ キューを使う

NMU の許可を求めた後で待っているのは効率的ではありません。NMU した人が作業にもどるために頭を切り替えるのに手間がかかるからです。`DELAYED` キュー (**遅延アップロード** 参照) は、開発者が NMU をするのに必要な作業を同時にできるようになります。例えば、メンテナに対して更新したパッケージを 7 日後にアップロードするのを伝えるのではなく、パッケージを `DELAYED/7` にアップロードしてメンテナに対して対応するのに 7 日間あることを伝えるべきです。この間、メンテナはもっとアップロードを遅らせるかアップロードをキャンセルするかを尋ねることができます。

You can cancel your upload using *dcut*. In case you uploaded `foo_1.2-1.1_all.changes` to a `DELAYED` queue, you can run `dcut cancel foo_1.2-1.1_all.changes` to cancel your upload. The `.changes` file does not need to be present locally as you instruct *dcut* to upload a command file removing a remote filename. The `.changes` file name is the same that you used when uploading.

`DELAYED` キューは、無用のプレッシャーをメンテナに与えるために使われるべきではありません。特に、メンテナはアップロードを自身ではキャンセルできないので、delay が完了する前にアップロードをキャン

セルあるいは遅らせることができるのはあなただ、という点が重要です。

DELAYED を使った NMU を行って delay が完了する前にメンテナがパッケージを更新した場合には、アーカイブに既により新しいバージョンがあるので、あなたのアップロードは拒否されます。理想的なのは、メンテナがそのアップロードであなたが提案した変更 (あるいは少なくとも対応した問題の解決方法) を含めるのを取り仕切ることです。

5.13.4 メンテナの視点から見た NMU

誰かがあなたのパッケージを NMU した場合、これは彼らがパッケージを良い形に保つのを助けたいと思っているということです。これによって、ユーザへ修正したパッケージをより早く届けます。NMU した人に、パッケージの副メンテナになることを尋ねるのを考えてみるのも良いでしょう。パッケージに対して NMU を受け取るのは悪いことではありません。それは、単にそのパッケージが他の人が作業する価値があるという意味です。

NMU を承認するには、変更と changelog のエントリを次のメンテナアップロードに含めます。バグは BTS で close されたままになりますが、パッケージのメンテナバージョンに影響していると表示されます。

Note that if you ever need to revert a NMU that packages a new upstream version, it is recommended to use a fake upstream version like *CURRENT*+really*FORMER* until one can upload the latest version again. More information can be found in https://www.debian.org/doc/debian-policy/ch-controlfields.html#epochs-should-be-used-sparingly.

Note that the easiest way to both check if your package has been NMUed, and also automatically download and commit the changes into a git-buildpackage maintained git repository is to run `gbp import-dsc --verbose --pristine-tar apt:<package>/sid`. This example command assumes you are working on the `debian/latest` branch preparing the next upload to Debian unstable, and it assumes your `apt` has the `deb-src` line active for Debian unstable.

5.13.5 ソース NMU vs バイナリのみの NMU (binNMU)

NMU のフルネームは**ソース** *NMU* です。もう一つ別の種類があって、**バイナリのみの** *NMU (binary-only NMU)* あるいは *binNMU* と名付けられています。binNMU も、パッケージメンテナ以外の誰かによるパッケージのアップロードです。しかし、これはバイナリのみのアップロードです。

ライブラリ (や他の依存関係) が更新された時、それを使っているパッケージを再ビルドする必要があるかもしれません。ソースへの変更は必要ないので、同じソースパッケージが利用されます。

binNMU は、通常 wanna-build によって buildd 上で引き起こされます。`debian/changelog` にエントリが追加され、なぜアップロードが必要だったのか、という説明と **再コンパイル、あるいは** *binary-only NMU* で記述されているようにバージョン番号を増やします。このエントリは、その次のアップロードに含めるべきではありません。

buildd は、アーカイブするために、バイナリのみのアップロードとして、そのアーキテクチャに対してパッケージをアップロードします。厳密に言えば、これは binNMU です。しかし、これは通常 NMU とは呼ばれず、`debian/changelog` にエントリを追加しません。

5.13.6 NMU と QA アップロード

NMUs are uploads of packages by somebody other than their assigned maintainer. There is another type of upload where the uploaded package is not yours: QA uploads. QA uploads are uploads of orphaned packages.

QA アップロードは、ほとんど通常のメンテナによるアップロードと同じです: 些細な問題であっても、なんでも修正します。バージョン番号の付け方は通常のものですし、delay アップロードをする必要もありません。違いは、パッケージのメンテナあるいはアップローダとして記載されていない点です。また、QA アップロードの changelog のエントリは以下のように最初の一行が特別になっています:

```
* QA upload.
```

あなたが NMU をしたいと思い、かつ、メンテナが活動的ではない場合、パッケージが放棄されてないかどうかを確認するのが賢明です (この情報はパッケージ追跡システム (PTS) のページで表示されています)。放棄されたパッケージに対して最初の QA アップロードを行うときは、メンテナは `Debian QA Group <packages@qa.debian.org>` に設定する必要があります。まだ QA アップロードがされていない放棄されたパッケージには、以前のメンテナが設定されています。この一覧は https://qa.debian.org/orphaned.html で手に入ります。

Instead of doing a QA upload, you can also consider adopting the package by making yourself the maintainer. You don't need permission from anybody to adopt an orphaned package; you can just set yourself as maintainer and upload the new version (see **パッケージを引き取る**).

5.13.7 NMU とチームアップロード

Sometimes you are fixing and/or updating a package because you are member of a packaging team (which uses a mailing list as `Maintainer` or `Uploader`; see **共同メンテナンス**) but you don't want to add yourself to `Uploaders` because you do not plan to contribute regularly to this specific package. If it conforms with your team's policy, you can perform a normal upload without being listed directly as `Maintainer` or `Uploader`. In that case, you should start your changelog entry with the following line:

```
* Team upload.
```

5.14 Package Salvaging

Package salvaging is the process by which one attempts to save a package that, while not officially orphaned, appears poorly maintained or completely unmaintained. This is a weaker and faster procedure than orphaning a package officially through the powers of the MIA team. Salvaging a package is not meant to replace MIA handling, and differs in that it does not imply anything about the overall activity of a maintainer. Instead, it handles a package maintainership transition for a single package only, leaving any other package or Debian membership or upload rights (when applicable) untouched.

Note that the process is only intended for actively taking over maintainership. Do not start a package salvaging process when you do not intend to maintain the package for a prolonged time. If you only want to fix certain things, but not take over the package, you must use the NMU process, even if the package would be eligible for salvaging. The NMU process is explained in *Non-Maintainer Upload (NMU)*.

Another important thing to remember: It is not acceptable to hijack others' packages. If followed, this salvaging process will help you to ensure that your endeavour is not a hijack but a (legal) salvaging procedure, and you can

counter any allegations of hijacking with a reference to this process. Thanks to this process, new contributors should no longer be afraid to take over packages that have been neglected or entirely forgotten.

The process is split into two phases: In the first phase you determine whether the package in question is *eligible* for the salvaging process. Only when the eligibility has been determined you may enter the second phase, the *actual* package salvaging.

For additional information, rationales and FAQs on package salvaging, please visit the Salvaging Packages page on the Debian wiki.

5.14.1 When a package is eligible for package salvaging

A package becomes eligible for salvaging when it has been neglected by the current maintainer. To determine that a package has really been neglected by the maintainer, the following indicators give a rough idea what to look for:

- NMUs, especially if there has been more than one NMU in a row.
- Bugs filed against the package do not have answers from the maintainer.
- Upstream has released several versions, but despite there being a bug entry asking for it, it has not been packaged.
- There are QA issues with the package.

You will have to use your judgement as to whether a given combination of factors constitutes neglect; in case the maintainer disagrees they have only to say so (see below). If you're not sure about your judgement or simply want to be on the safe side, there is a more precise (and conservative) set of conditions in the Package Salvaging wiki page. These conditions represent a current Debian consensus on salvaging criteria. In any case you should explain your reasons for thinking the package is neglected when you file an Intent to Salvage bug later.

5.14.2 How to salvage a package

If and *only* if a package has been determined to be eligible for package salvaging, any prospective maintainer may start the following package salvaging procedure.

1. Open a bug with the severity "important" against the package in question, expressing the intent to take over maintainership of the package. For this, the title of the bug should start with `ITS: package-name`[3]. You may alternatively offer to only take co-maintenance of the package. When you file the bug, you must inform all maintainers, uploaders and if applicable the packaging team explicitly by adding them to `X-Debbugs-CC`. Additionally, if the maintainer(s) seem(s) to be generally inactive, please inform the MIA team by adding `mia@qa.debian.org` to `X-Debbugs-CC` as well. As well as the explicit expression of the intent to salvage, please also take the time to document your assessment of the eligibility in the bug report, for example by listing the criteria you've applied and adding some data to make it easier for others to assess the situation.
2. In this step you need to wait in case any objections to the salvaging are raised; the maintainer, any current uploader or any member of the associated packaging team of the package in question may object publicly in response to the bug you've filed within `21 days`, and this terminates the salvaging process.

   The current maintainers may also agree to your intent to salvage by filing a (signed) public response to the bug. They might propose that you become a co-maintainer instead of the sole maintainer. On team

[3] ITS is shorthand for *"Intend to Salvage"*

maintained packages, a member of the associated team can accept your salvaging proposal by sending out a signed agreement notice to the ITS bug, alternatively inviting you to become a new co-maintainer of the package. The team may require you to keep the package under the team's umbrella, but then may ask or invite you to join the team. In any of these cases where you have received the OK to proceed, you can upload the new package immediately as the new (co-)maintainer, without the need to utilise the `DELAYED` queue as described in the next step.

3. After the 21 days delay, if no answer has been sent to the bug from the maintainer, one of the uploaders or team, you may upload the new release of the package into the `DELAYED` queue with a minimum delay of `seven days`. You should close the salvage bug in the changelog and you must also send an nmudiff to the bug ensuring that copies are sent to the maintainer and any uploaders (including teams) of the package by `CC'ing` them in the mail to the BTS.

   During the waiting time of the `DELAYED` queue, the maintainer can accept the salvaging, do an upload themselves or (ask to) cancel the upload. The latter two of these will also stop the salvaging process, but the maintainer must reply to the salvaging bug with more information about their action.

5.15 共同メンテナンス

共同メンテナンス (collaborative maintenance) は、Debian パッケージのメンテナンス責任を数人でシェアすることを指す用語です。この共同作業は、通常はより上質で短いバグ修正時間をもたらすので、大抵の場合は常に良い考えです。優先度が標準 (`standard`) あるいは基本セット (base) の一部であるパッケージは、共同メンテナ (co-maintainer) を持つことを強くお勧めします。

大抵の場合、主メンテナに加えて一人か二人の共同メンテナが居ます。主メンテナは `debian/control` ファイルの `Maintainer` 欄に名前が記載されている人です。共同メンテナは他のすべてのメンテナで、通常 `debian/control` ファイルの `Uploaders` に記載されています。

もっとも基本的なやり方では、新しい副メンテナの追加は大変簡単です:

- Set up the co-maintainer with access to the sources you build the package from. Generally this implies you are using a network-capable version control system, such as `Git`. Salsa (see *salsa.debian.org: Git repositories and collaborative development platform*) provides Git repositories, amongst other collaborative tools.
- 共同メンテナの正確なメンテナ名とアドレスを `debian/control` ファイルの最初の段落の `Uploaders` 欄に追加します。

```
Uploaders: John Buzz <jbuzz@debian.org>, Adam Rex <arex@debian.org>
```

- The co-maintainers should subscribe themselves to the appropriate source package (see *Subscribing to package updates*).

共同メンテナンスのもう一つの形態はチームメンテナンスです。これは、複数のパッケージを同じ開発者グループでメンテナンスする場合にお勧めです。その場合、各パッケージの `Maintainer` 欄と `Uploaders` 欄は注意して扱わねばいけません。以下の二つの案からいずれかを選ぶのがお勧めです:

1. パッケージの主に担当をするチームメンバーを `Maintainer` 欄に追加します。`Uploaders` 欄には、メーリングリストのアドレスとパッケージの面倒をみるチームメンバーを追加します。
2. Put the mailing list address in the `Maintainer` field. In the `Uploaders` field, put the team members who care for the package. In this case, you must make sure the mailing list accepts bug reports without any human

interaction (like moderation for non-subscribers).

どのような場合でも、すべてのチームメンバーを Uploaders 欄に入れるのは良くない考えです。これは、Developer's Package Overview の一覧 (*Developer's packages overview* 参照) を実際には対応していないパッケージで散らかしてしまい、偽りの良いメンテナンス状態を作り出します。同じ理由から、パッケージを一回アップロードするのであれば、「チームアップロード (Team Upload)」ができるので、チームメンバーは Uploaders 欄へ自分を追加する必要はありません (*NMU* **とチームアップロード** 参照)。逆にいえば、Maintainer 欄にメーリングリストのアドレスのみで Uploaders 欄に誰も追加していないままにしておくのは良くない考えです。

5.16 テスト版ディストリビューション

5.16.1 基本

パッケージは通常、不安定版 (unstable) におけるテスト版への移行基準を満たした後でテスト版 (testing) ディストリビューションへとインストールされます。

これらは、すべてのアーキテクチャ上で同期していなければならず、インストールできなくなるような依存関係を持ってはいけません。また、テスト版 (testing) にインストールされる際に既知のリリースクリティカルバグを持っていない必要があります。このようにして、テスト版 (testing) は常にリリース候補に近いものである必要があります。詳細は以下を参照してください。

5.16.2 不安定版からの更新

テスト版 (testing) ディストリビューションを更新するスクリプトは、日に二回、更新されたパッケージのインストール直後に動作します。これらのスクリプトは britney と呼ばれます。これは、テスト版 (testing) ディストリビューションに対して Packages ファイルを生成しますが、不整合を避けてバグが無いパッケージのみを利用しようとする気の利いたやり方で行います。

不安定版 (unstable) からのパッケージの取り込みは以下の条件です:

- The package must have been available in unstable for a certain number of days, see *Selecting the upload urgency*. Please note that the urgency is sticky, meaning that the highest urgency uploaded since the previous testing transition is taken into account;
- 新たなリリースクリティカルバグを持っていないこと (不安定版 (unstable) で利用可能なバージョンに影響する RC バグであって、テスト版 (testing) にあるバージョンに影響するものではない);
- あらかじめ unstable でビルドされた際に、全アーキテクチャで利用可能になっていなくてはいけません。この情報をチェックするのに *dak ls* **ユーティリティ** に興味を持つかもしれません;
- 既にテスト版 (testing) で利用可能になっているパッケージの依存関係を壊さないこと;
- パッケージが依存するものは、テスト版 (testing) で利用可能なものか、テスト版 (testing) に同時に受け入れられるものでなくてはいけない (そして、それらは必要な条件をすべて満たしていれば、テスト版 (testing)) に入る);
- プロジェクトの状況。つまり、テスト版 (testing) ディストリビューションの**フリーズ**中は、自動的な移行がオフになります。

パッケージがテスト版 (testing) に入る処理がされるかどうかは、テスト版ディストリビューションのウェブページのテスト版 (testing) スクリプトの出力を参照するか、devscripts パッケージ中の grep-excuses プログラムを使ってください。このユーティリティは、パッケージがテスト版 (testing) への進行の通知をし続けるのに、crontab 5 で手軽に使うことができます。

update_excuses は、なぜパッケージが拒否されたのか正確な理由を必ずしも表示しません。自分自身で何がパッケージが含まれるのを妨げているのか、探す必要があるかもしれません。テスト版のウェブページが、そのような問題を引き起こす良くある問題についての情報を与えてくれるでしょう。

時折、相互依存関係の組み合わせが非常に難解なのでスクリプトが解決できないことがあるため、パッケージがテスト版 (testing) に決して入らないことがあります。詳細は下記を参照してください。

Some further dependency analysis is shown on https://release.debian.org/migration/— but be warned: this page also shows build dependencies that are not considered by britney.

5.16.2.1 時代遅れ (Out-of-date)

For the testing migration script, outdated means: There are different versions in unstable for the release architectures (except for the architectures in outofsync_arches; outofsync_arches is a list of architectures that don't keep up (in britney.py), but currently, it's empty). Outdated has nothing whatsoever to do with the architectures this package has in testing.

以下の例を考えてみましょう:

	alpha	arm
テスト版	1	-
不安定版	1	2

The package is out of date on alpha in unstable, and will not go to testing. Removing the package would not help at all; the package is still out of date on alpha, and will not propagate to testing.

ですが、もしも ftp-master が不安定版 (unstable) のパッケージ (ここでは arm の) を削除した場合:

	alpha	arm	hurd-i386
テスト版	1	1	-
不安定版	2	-	1

この場合、パッケージは不安定版 (unstable) ですべてのリリースアーキテクチャで最新になります (それから、もう一つの hurd-i386 は、リリースアーキテクチャではないので問題になりません)。

時折、すべてのアーキテクチャでまだビルドされていていないパッケージを入れられるか、という質問がでてきます: いいえ。 単純にいいえ、です (glibc などをメンテしている場合を除きます)。

5.16.2.2 テスト版からの削除

時折、パッケージは他のパッケージがテスト版へ入るために削除されます: これは、**他の**パッケージが他のすべての面で準備ができている場合にテスト版に入るようにする場合のみ発生します。例えば、a が新しいバージョンの b とはインストールできない場合を考えてみましょう。その場合、a は b が入るために削除されるかもしれません。

Of course, there is another reason to remove a package from `testing`: it's just too buggy (and having a single RC-bug is enough to be in this state).

さらに、パッケージが 不安定版 (`unstable`) から削除され、テスト版 (`testing`) にはこれに依存するパッケージがもうなくなった場合、パッケージは自動的に削除されます。

5.16.2.3 循環依存

britney によってうまく取扱われない状況は、パッケージ a がパッケージ b の新しいバージョンに依存していて、そしてその逆も、というものです。

この場合の例:

	テスト版	不安定版
a	1; depends: b=1	2; depends: b=2
b	1; depends: a=1	2; depends: a=2

パッケージ a あるいはパッケージ b が更新対象と見做されない。

現状、このような場合はリリースチームによる手動でのヒントが必要になります。あなたのパッケージのどれかにこのような状況が起きた場合は、`debian-release@lists.debian.org` にメールを送って連絡を取ってください。

5.16.2.4 テスト版 (testing) にあるパッケージへの影響

Generally, there is nothing that the status of a package in `testing` means for transition of the next version from `unstable` to `testing`, with two exceptions: If the RC-bugginess of the package goes down, it may go in even if it is still RC-buggy. The second exception is if the version of the package in `testing` is out of sync on the different arches: Then any arch might just upgrade to the version of the source package; however, this can happen only if the package was previously forced through, the arch is in outofsync_arches, or there was no binary package of that arch present in `unstable` at all during the `testing` migration.

この要旨: 影響は、テスト版 (`testing`) にあるパッケージが、同じパッケージの新しいバージョンになるのは、新しいバージョンの方が楽にできそうだから、ということです。

5.16.2.5 詳細について

詳細について知りたい場合ですが、britney の動作は以下のようになります:

パッケージが、適切な候補であるかどうかを決めるために確認が行われます。これによって、更新が実行されます。パッケージが候補とみなされない理由でもっともよくあるのは、まだ日数が経過していない (too young)、RC バグがある、いくつかのアーキテクチャで古くなりすぎている、というものです。britney のこ

の部分では、リリースマネージャーが britney がパッケージを検討できるように、hints と呼ばれる様々なサイズのハンマーを使います (下記を参照してください)。

さて、より複雑な部分に差し掛かります: Britney が適正候補を使ってテスト版 (testing) を更新しようとします。このため、britney はテスト版ディストリビューションに個々の適正な候補を追加しようとします。テスト版 (testing) でインストール不可能なパッケージの数が増えないのであれば、パッケージは受け入れられます。その時から、受け入れられたパッケージはテスト版 (testing) の一部として取り扱われ、このパッケージを含めるためのインストールチェックテストが引き続き行われます。リリースチームからの hints は、実際の内容に応じて、このメインの処理の前後に処理されます。

より詳細を見たい場合は、https://release.debian.org/britney/update_output/ で探すことができます。

The hints are available via https://release.debian.org/britney/hints/, where you can find the description as well. With the hints, the Debian Release team can block or unblock packages, ease or force packages into testing, remove packages from testing, approve uploads to **直接テスト版を更新する** or override the urgency.

5.16.3 直接テスト版を更新する

テスト版 (testing) ディストリビューションは、上記で説明したルールに従って 不安定版 (unstable) からのパッケージで作られています。しかし、時折、テスト版 (testing) の為だけに構築したパッケージをアップロードする必要があるという場合があります。そのためには、testing-proposed-updates にアップロードするのが良いでしょう。

Keep in mind that packages uploaded there are not automatically processed; they have to go through the hands of the release manager. So you'd better have a good reason to upload there. In order to know what a good reason is in the release managers' eyes, you should read the instructions that they regularly give on debian-devel-announce@lists.debian.org.

You should not upload to testing-proposed-updates when you can update your packages through unstable. If you can't (for example because you have a newer development version in unstable), you may use this facility. Even if a package is frozen, updates through unstable are possible, if the upload via unstable does not pull in any new dependencies.

バージョン番号は、通常 +deb*X*uY を付加することで指定されます。*X* は Debian のメジャーリリース番号で *Y* は 1 から始まる数です。例: 1:2.4.3-4+deb13u1

アップロードでは、以下の項目のいずれも見落とさないように必ずしてください:

- 本当に testing-proposed-updates を通す必要があり、unstable ではダメなことを確認する;
- 必ず、最小限な量だけの変更を含めるようにする;
- 必ず changelog 中に適切な説明を含める;
- 必ず、対象とするディストリビューションとして testing **リリースのコードネーム** (e.g. forky) を記述している;
- 必ず 不安定版 (unstable) ではなく テスト版 (testing) でパッケージを構築・テストしている;
- バージョン番号が testing および testing-proposed-updates のものよりも大きく、unstable のものよりも小さいのを確認する;
- Ask for authorization for uploading from the release managers.

- アップロードしてすべてのプラットフォームで構築が成功したら、debian-release@lists.debian.org 宛でリリースチームに連絡を取って、アップロードを承認してくれるように依頼しましょう。

5.16.4 よく聞かれる質問とその答え (FAQ)

5.16.4.1 リリースクリティカルバグとは何ですか、どうやって数えるのですか?

ある重要度以上のバグすべてが通常リリースクリティカルであると見なされます。現在のところ、critical (致命的)、grave (重大)、serious (深刻) バグがそれにあたります。

そのようなバグは、Debian の 安定版 (stable) リリース時に、そのパッケージがリリースされる見込みに影響があるものと仮定されます: 一般的に、パッケージでオープンになっているリリースクリティカルバグがある場合、そのパッケージはテスト版 (testing) に入らず、その結果安定版 (stable) ではリリースされません。

The unstable bug count comprises all release-critical bugs that are marked to apply to *package/version* combinations available in unstable for a release architecture. The testing bug count is defined analogously.

5.16.4.2 どのようにすれば、他のパッケージを壊す可能性があるパッケージをテスト版 (testing) へインストールできますか?

ディストリビューションにおけるアーカイブの構造では、一つのバージョンのパッケージだけを持つことができ、パッケージは名前によって定義されています。そのため、ソースパッケージ acmefoo がテスト版 (testing) にインストールされると、付随するバイナリパッケージ acme-foo-bin、acme-bar-bin、libacme-foo1、libacme-foo-dev の古いバージョンが削除されます。

However, the old version may have provided a binary package with an old soname of a library, such as libacme-foo0. Removing the old acmefoo will remove libacme-foo0, which will break any packages that depend on it.

Evidently, this mainly affects packages that provide changing sets of binary packages in different versions (in turn, mainly libraries). However, it will also affect packages upon which versioned dependencies have been declared of the ==, <=, or << varieties.

When the set of binary packages provided by a source package changes in this way, all the packages that depended on the old binaries will have to be updated to depend on the new binaries instead. Because installing such a source package into testing breaks all the packages that depended on it in testing, some care has to be taken now: all the depending packages must be updated and ready to be installed themselves so that they won't be broken, and, once everything is ready, manual intervention by the release manager or an assistant is normally required.

この様に複雑な組み合わせのパッケージで問題がある場合は、debian-devel@lists.debian.org あるいは debian-release@lists.debian.org に連絡を取って手助けを求めてください。

5.17 The Stable backports archive

5.17.1 基本

Once a package reaches the testing distribution, it is possible for anyone with upload rights in Debian (see below about this) to build and upload a backport of that package to stable-backports, to allow easy installation of the version from testing onto a system that is tracking the stable distribution.

One should not upload a version of a package to stable-backports until the matching version has already reached the testing archive.

5.17.2 Exception to the testing-first rule

The only exception to the above rule, is when there's an important security fix that deserves a quick upload: in such a case, there is no need to delay an upload of the security fix to the stable-backports archive. However, it is strongly advised that the package is first fixed in unstable before uploading a fix to the stable-backports archive.

5.17.3 Who can maintain packages in the stable-backports archive?

It is not necessarily up to the original package maintainer to maintain the stable-backports version of the package. Anyone can do it, and one doesn't even need approval from the original maintainer to do so. It is however good practice to first get in touch with the original maintainer of the package before attempting to start the maintenance of a package in stable-backports. The maintainer can, if they wish, decide to maintain the backport themselves, or help you doing so. It is not uncommon, for example, to apply a patch to the unstable version of a package, to facilitate its backporting.

5.17.4 When can one start uploading to stable-backports?

The new stable-backports is created before the freeze of the next stable suite. However, it is not allowed to upload there until the very end of the freeze cycle. The stable-backports archive is usually opened a few weeks before the final release of the next stable suite, but it doesn't make sense to upload until the release has actually happened.

5.17.5 How long must a package be maintained when uploaded to stable-backports?

The stable-backports archive is maintained for bugs and security issues during the whole life-cycle of the Debian stable suite. Therefore, an upload to stable-backports, implies a willingness to maintain the backported package for the duration of the stable suite, which can be expected to be about 3 years from its initial release.

The person uploading to backports is also supposed to maintain the backported packages for security during the lifetime of stable.

It is to be noted that the stable-backports isn't part of the LTS or ELTS effort. The stable-backports FTP masters will close the stable-backports repositories for uploads once stable reaches end-of-life (ie: when stable becomes maintained by the LTS team only). Therefore there won't be any maintenance of packages from stable-backports after the official end of life of the stable suite, as uploads will not be accepted.

5.17.6 How often shall one upload to stable-backports?

The packages in backports are supposed to follow the developments that are happening in Testing. Therefore, it is expected that any significant update in testing should trigger an upload into stable-backports, until the new stable is released. However, please do not backport minor version changes without user visible changes or bugfixes.

5.17.7 How can one learn more about backporting?

You can learn more about how to contribute directly on the backport web site.

It is also recommended to read the Frequently Asked Questions (FAQ).

第6章　パッケージ化のベストプラクティス

Debian's quality is largely due to the Debian Policy, which defines explicit baseline requirements that all Debian packages must fulfill. Yet there is also a shared history of experience which goes beyond the Debian Policy, an accumulation of years of experience in packaging. Many very talented people have created great tools, tools which help you, the Debian maintainer, create and maintain excellent packages.

この章では、Debian 開発者へのベストプラクティスをいくつか提供します。すべての勧めは単に勧めであり、要求事項やポリシーではありません。Debian 開発者らからの主観的なヒント、アドバイス、ポインタです。あなたにとって一番うまくいくやり方を、どうぞご自由に選んでください。

6.1 debian/rules についてのベストプラクティス

The following recommendations apply to the `debian/rules` file. Since `debian/rules` controls the build process and selects the files that go into the package (directly or indirectly), it's usually the file maintainers spend the most time on.

6.1.1 ヘルパースクリプト

`debian/rules` でヘルパースクリプトを使う根拠は、多くのパッケージ間でメンテナらに共通のロジックを利用・共有させるようになるからです。メニューエントリのインストールについての問いを例にとってみましょう: ファイルを `/usr/share/menu` (必要であれば、実行形式のバイナリのメニューファイルの場合 `/usr/lib/menu`) に置き、メンテナスクリプトにメニューエントリを登録・解除するためのコマンドを追加する必要があります。これはパッケージが行う、非常に一般的なことです。なぜ個々のメンテナがこれらのすべてを自分で書き直し、時にはバグを埋め込む必要があるでしょう? また、メニューディレクトリが変更された場合、すべてのパッケージを変更する必要があります。

ヘルパースクリプトがこれらの問題を引き受けてくれます。ヘルパースクリプトの期待するやり方に従っているならば、ヘルパースクリプトはすべての詳細について考慮をします。ポリシーの変更はヘルパースクリプト中で行えます; そして、パッケージを新しいバージョンのヘルパースクリプトでリビルドする必要があるだけです。他に何の変更も必要ありません。

Debian **メンテナツールの概要** には、複数の異なったヘルパーが含まれています。もっとも一般的で (我々の意見では) ベストなヘルパーシステムは `debhelper` です。`debmake` のような、以前のヘルパーシステムはモノリシックでした: 使えそうなヘルパーの一部を取り出して選ぶことはできず、何を行うにもヘルパーを使う必要がありました。ですが、`debhelper` は、いくつもの分割された小さな `dh_*` プログラムです。たとえば、`dh_installman` は man ページをインストールして圧縮し、`dh_installmenu` は menu ファイルをインストールするなどします。つまり、`debian/rules` 内で使える部分では小さなヘルパースクリプトを使い、手製のコマンドを使うといった十分な柔軟性を与えてくれます。

You can get started with `debhelper` by reading debhelper(7) and looking at the examples that come with the package. `dh_make`, from the `dh-make` package (see *dh-make*), can be used to convert a vanilla source package

to a debhelperized package. This shortcut, though, should not convince you that you do not need to bother understanding the individual dh_* helpers. If you are going to use a helper, you do need to take the time to learn to use that helper, to learn its expectations and behavior.

6.1.2 複数のバイナリパッケージ

A single source package will often build several binary packages, either to provide several flavors of the same software (e.g., the vim source package) or to make several small packages instead of a big one (e.g., so the user can install only the subset needed, and thus save some disk space, see for example the libxml2 source package).

二つ目の例は、debian/rules で簡単に扱うことができます。ビルドディレクトリからパッケージの一時ツリーへ、適切なファイルを移動する必要があるだけです。これは、install または debhelper の dh_install を使ってできます。パッケージ間の依存関係を debian/control 内で正しく設定したのを忘れずに確認してください。

The first case is a bit more difficult since it involves multiple recompiles of the same software but with different configuration options. The vim source package is an example of how to manage this using a hand-crafted debian/rules file.

6.2 debian/control のベストプラクティス

以下のプラクティスは、debian/control ファイルに関するものです。パッケージ説明文についてのポリシーを補完します。

パッケージの説明文は、control ファイルの対応するフィールドにて定義されている様に、パッケージの概要とパッケージに関する長い説明文の両方を含んでいます。**パッケージ説明文の基本的なガイドライン** では、パッケージ説明文の双方の部分についての一般的なガイドラインが記述されています。それによると、**パッケージの概要、あるいは短い説明文** が概要に特化したガイドラインを提供しており、そして **長い説明文** *(long description)* が説明文 (description) に特化したガイドラインを含んでいます。

6.2.1 The package name

The package name:

- Must be consistent with widely established conventions, e.g.
  - C and C++ libraries are typically prefixed with lib
  - for many other languages, package names are prefixed or suffixed with the language name (the actual convention depends on the language).
- Should not be a common, unqualified word. In particular, words that represent a whole class of applications (e.g. reader, browser) should be reserved for virtual packages.
- Should ideally be at least 4 characters long, unless the upstream name is shorter than that *and* the project is already widely recognized by that name (e.g. fzf).

6.2.2 パッケージ説明文の基本的なガイドライン

パッケージの説明文は平均的なユーザーに向けて書く必要があります。平均的な人というのは、パッケージを使って得をするであろう人のことです。例えば、開発用パッケージであれば開発者向けですし、彼ら向けの言葉でテクニカルに記述することができます。より汎用的なアプリケーション、例えばエディタなどであれば、あまり技術的ではないユーザ向けに書く必要があります。

パッケージ説明文のレビューを行った結果、ほとんどのものがテクニカルである、つまり、技術に詳しくはないユーザに通じるようには書かれてはいないという結論に達しました。あなたのパッケージが、本当に技術に精通したユーザ用のみではない限り、これは問題です。

どうやって技術に詳しくはないユーザに対して書けばいいのでしょう? ジャーゴンを避けましょう。ユーザが詳しくないであろう他のアプリケーションやフレームワークへの参照を避けましょうー GNOME や KDE については、おそらくユーザはその言葉について知っているでしょうから構いませんが、GTK はおそらくダメです。まったく知識がないと仮定してみましょう。技術用語を使わねばならない場合は、説明しましょう。

客観的になりましょう。パッケージ説明文はあなたのパッケージの宣伝場所ではありません。あなたがそのパッケージをどんなに愛しているかは関係ありません。その説明文を読む人は、あなたが気にすることと同じことを気にはしないであろうことを覚えておいてください。

他のソフトウェアパッケージ、プロトコル名、標準規格、仕様の名前を参照する場合には、もしあれば正規名称を使いましょう。X Windows や X-Windows や X Window ではなく、X Window System あるいは X11 または X を使いましょう。GTK+ や gtk ではなく GTK を使いましょう。Gnome ではなく GNOME を使いましょう。Postscript や postscript ではなく PostScript を使いましょう。

説明文を書くことに問題があれば、`debian-l10n-english@lists.debian.org` へそれを送ってフィードバックを求めるとよいでしょう。

6.2.3 パッケージの概要、あるいは短い説明文

ポリシーでは、概要行 (短い説明文) はパッケージ名を繰り返すのではなく、簡潔かつ有益なものである必要がある、となっています。

概要は、完全な文章ではなくパッケージを記述するフレーズとして機能します。ですので、句読点は不適切です: 追加の大文字や最後のピリオドは不要です。また、最初の不定冠詞や定冠詞– "a", "an", or "the" を削る必要があります。つまり、例えば以下のようになります:

```
Package: libeg0
Description: exemplification support library
```

技術的に言えば、動詞のフレーズに対して、これは名詞のフレーズから文章を差し引いたものです。パッケージ**名**と**要約**をこの決まり文句に代入できるのがよい見つけ方です:

パッケージの**名前**は**概要**を提供します。

関連パッケージ群は、概要を 2 つに分けた別の書き方をした方が良いでしょう。最初はその組一式の説明文で、その次はその組内でのパッケージの役割のサマリにします:

```
Package: eg-tools
Description: simple exemplification system (utilities)
```

(次のページに続く)

(前のページからの続き)

```
Package: eg-doc
Description: simple exemplification system - documentation
```

これらの要約が、手が加えられた決まり文句に続きます。パッケージ "**名**" が、"**プログラム一式** (**役割**)" あるいは "**プログラム一式** - **役割**" という要約を持つ場合、要素はフレーズにすべきで、決まり文句に合うようになります:

パッケージ**名**は、**プログラム一式**に対する**役割**を表しています。

6.2.4 長い説明文 (long description)

長い説明文は、ユーザーがパッケージをインストールする前に利用可能な最初の情報です。ユーザーがインストールするか否かを決めるのに必要な情報を、すべて提供する必要があります。ユーザーがパッケージの概要を既に読んでいると仮定してください。

長い説明文は、完全な文章から成る必要があります。

長い説明文の最初の段落は、以下の質問に答えている必要があります: このパッケージは何をするの? ユーザが作業を完了するのに、どのタスクが役に立つの? — 技術寄りではない書き方でこれを記述するのが重要です。もちろんパッケージの利用者が必然的に技術者ではない限り、です。

Long descriptions of related packages, for example built from the same source, can share paragraphs in order to increase consistency and reduce the workload for translators, but you need at least one separate paragraph describing the package's specific role.

The following paragraphs should answer the following questions: Why do I as a user need this package? What other features does the package have? What outstanding features and deficiencies are there compared to other packages (e.g., if you need X, use Y instead)? Is this package related to other packages in some way that is not handled by the package manager (e.g., is this the client for the foo server)?

スペルミスや文法の間違いを避けるよう、注意してください。スペルチェックを確実に行ってください。`ispell` と `aspell` の双方に、`debian/control` ファイルをチェックするための特別なモードがあります:

```
ispell -d american -g debian/control
```

```
aspell -d en -D -c debian/control
```

通常、ユーザは以下のような疑問がパッケージ説明文で答えられることを期待しています:

- パッケージは何をするの? 他のパッケージのアドオンだった場合、パッケージがアドオンであるということを概要文に書く必要があります。
- なぜこのパッケージを使うべきなの? これは上記に関連しますが、同じではありません (これはメールユーザーエージェントです; クールで速く、PGP や LDAP や IMAP のインターフェイスがあり、X や Y や Z の機能があります)。
- パッケージが直接インストールされるべきではないが、他のパッケージから引っ張ってこられる時には、付記しておく必要があります。

- パッケージが実験的である、あるいは使われない方が良い他の理由がある場合、同様にここに記載する必要があります。
- How is this package different from the competition? Is it a better implementation? more features? different features? Why should I choose this package?

6.2.5 開発元のホームページ

`debian/control` 中の `Source` セクションの `Homepage` フィールドへ、パッケージのホームページの URL を追加することをお勧めします。この情報をパッケージ説明文自身に追加するのは推奨されない (deprecated) であると考えられています。

6.2.6 バージョン管理システムの場所

`debian/control` には、バージョン管理システムの場所についての追加フィールドがあります。

6.2.6.1 Vcs-Browser

このフィールドの値は、指定したパッケージのメンテナンスに使われているバージョン管理システムのリポジトリのコピーがもしあれば、それを指し示す `https://` URL である必要があります。

この情報は、パッケージに行われた最新の作業を閲覧したいエンドユーザにとって有用であるのが目的です (例: バグ追跡システムで `pending` とタグがつけられているバグを修正するパッチを探している場合)。

6.2.6.2 Vcs-*

Value of this field should be a string identifying unequivocally the location of the Version Control System repository used to maintain the given package, if available. * identifies the Version Control System; currently the following systems are supported by the package tracking system: `arch`, `bzr` (Bazaar), `cvs`, `darcs`, `git`, `hg` (Mercurial), `mtn` (Monotone), `svn` (Subversion).

この情報は、そのバージョン管理システムについて知識があり、VCS ソースから現在のバージョンパッケージを生成ユーザにとって有益であるよう意図されています。この情報の他の使い方としては、指定されたパッケージの最新の VCS バージョンを自動ビルドするなどがあるかもしれません。このため、フィールドによって指し示される場所は、バージョンに関係なく、(そのようなコンセプトをサポートしている VCS であれば) メインブランチを指すのが良いでしょう。また、指し示される場所は一般ユーザがアクセス可能である必要があります; この要求を満たすには SSH アクセス可能なリポジトリを指すのではなく、匿名アクセスが可能なリポジトリを指すようにすることを意味します。

In the following example, an instance of the field for a Git repository of the `vim` package is shown. Note how the URL is in the `https://` scheme (instead of `ssh://`). The use of the `Vcs-Browser` and `Homepage` fields described above is also shown.

```
Source: vim
<snip>
Vcs-Git: https://salsa.debian.org/vim-team/vim.git
Vcs-Browser: https://salsa.debian.org/vim-team/vim
Homepage: https://www.vim.org
```

Maintaining the packaging in a version control system, and setting a Vcs-* header is good practice and makes it easier for others to contribute changes.

Almost all packages in Debian that use a version control system use Git; if you create a new package, using Git is a good idea simply because it's the system that contributors will be familiar with.

DEP-14 defines a common layout for Debian packages.

6.3 debian/changelog のベストプラクティス

以下のプラクティスは changelog ファイルに対するポリシーを補完します。

6.3.1 役立つ changelog のエントリを書く

パッケージリビジョンに対する changelog エントリは、そのリビジョンでの変更それだけを記載します。最後のバージョンから行われた、重要な、そしてユーザに見える形の変更を記述することに集中しましょう。

何が変更されたかについて集中しましょう– 誰が、どうやって、何時なのか通常あまり重要ではありません。そうは言っても、パッケージ作成について明記すべき手助けをしてくれた人々 (例えば、パッチを送ってくれた人) を丁寧に明記するのを忘れないようにしましょう。

些細で明白な変更を詳細に書く必要はありません。複数の変更点を一つのエントリにまとめることもできます。逆に言えば、大きな変更をした場合には、あまりに簡潔にしすぎないようにしましょう。プログラムの動作に影響を与える変更がある場合には、特に確認しておきましょう。詳細な説明については、`README.Debian` ファイルを使ってください。

平易な英語を使いましょう。そうすれば読者の大半が理解できます。バグをクローズする変更を説明する際には略語や、テクニカルな言い方やジャーゴンを避けましょう。特に、技術的に精通していないと思われるユーザによって登録されたバグを閉じる際には気をつけましょう。礼儀正しく、断言をしないように。

時折、changelog エントリに変更したファイルの名前を頭に付けたくなることがあります。ですが、個々のすべての変更したファイルを一覧にする必要性はありません。特に変更点が小さくて繰り返される場合です。ワイルドカードを使いましょう。

バグに触れる際には、何も仮定しないようにしましょう。何が問題だったのか、どうやってそれが直されたのかについて言い、closes: #nnnnn の文字列を追加しましょう。詳細については **新規アップロードでバグがクローズされる時** を参照してください。

6.3.2 Selecting the upload urgency

The release team have indicated that they expect most uploads to `unstable` to use **urgency=medium**. That is, you should choose **urgency=medium** unless there is some particular reason for the upload to migrate to `testing` more quickly or slowly (see also **不安定版からの更新**). For example, you might select **urgency=low** if the changes since the last upload are large and might be disruptive in unanticipated ways.

The delays are currently 2, 5 or 10 days, depending on the urgency (high, medium or low). The actual numbers are actually controlled by the britney configuration which also includes accelerated migrations when Autopkgtest passes.

6.3.3 changelog のエントリに関するよくある誤解

changelog エントリは、一般的なパッケージ化の事柄 (ほら、foo.conf を探しているなら /etc/blah にあるよ) を記載するべきでは**ありません**。何故なら、管理者やユーザは少なくとも Debian システム上でそのようなことがどのように扱われるかを多少は知らされているでしょうから。ですが、設定ファイルの場所を変更したのであれば、それは一言添えておくべきです。

changelog エントリでクローズされるバグは、実際にそのパッケージリビジョンで修正されたものだけにすべきです。changelog で関係ないバグを閉じるのは良くない習慣です。**新規アップロードでバグがクローズされる時** を参照してください。

changelog エントリは、バグ報告者との各種の議論の場 (foo をオプション bar 付きで起動した際にはセグメンテーションフォルトは見られません; もっと詳しい情報を送ってください)、生命、宇宙、そして万物についての概要 (すいませんが、このアップロードに時間がかかったので風邪をひきました)、手助けの求め (このパッケージのバグ一覧は巨大です、手を貸してください) に使うべきでは**ありません**。そのようなことは、大抵の場合対象としている読者は気づくことが無く、パッケージで実際にあった変更点の情報について読みたい人々を悩ますことでしょう。どの様にバグ報告システムを使えばいいのかについて、詳細な情報は**バグへの対応**を参照してください。

正式なメンテナによるアップロードの changelog エントリの最初で、non-maintainer upload で修正されたバグを承認するのは、古い慣習です。今はバージョン管理を行っているので、NMU された changelog エントリを残しておいて自分の changelog エントリ中でその事実に触れるだけで十分です。

6.3.4 changelog のエントリ中のよくある間違い

以下の例で、changelog エントリ中のよくある間違いや間違ったスタイルの例を挙げます。

```
* Fixed all outstanding bugs.
```

これは、全く読み手に何も有用なことを教えてくれません。

```
* Applied patch from Jane Random.
```

何についてのパッチですか?

```
* Late night install target overhaul.
```

何をオーバーホールしてどうなったのですか? 深夜というのに言及しているのは、私たちにこのコードを信用するなと言いたいのですか?

```
* Fix vsync fw glitch w/ ancient CRTs.
```

Too many acronyms (what does "fw" mean, "firmware"?), and it's not overly clear what the glitch was actually about, or how it was fixed.

```
* This is not a bug, closes: #nnnnnn.
```

まず初めに、この情報を伝えるためにパッケージをアップロードする必要は、全くありません; 代わりにバグ追跡システムを使ってください。次に、何故この報告がバグではなかったのかについての説明がありません。

```
* Has been fixed for ages, but I forgot to close; closes: #54321.
```

If for some reason you didn't mention the bug number in a previous changelog entry, there's no problem, just close the bug normally in the BTS. There's no need to touch the changelog file, presuming the description of the fix is already in (this applies to the fixes by the upstream authors/maintainers as well; you don't have to track bugs that they fixed ages ago in your changelog).

```
* Closes: #12345, #12346, #15432
```

説明はどこ? 説明文を考えられないのなら、それぞれのバグのタイトルを入れるところから始めてください。

6.3.5 NEWS.Debian ファイルで changelog を補足する

パッケージの変更に関する重要なニュースは NEWS.Debian ファイルにも書くことができます。このニュースは apt-listchanges のようなツールで、残りすべての changelog の前に表示されます。これは、ユーザにパッケージ内の著しい変更点について知らせるのに好ましいやり方です。インストール後にユーザが一旦戻って NEWS.Debian ファイルを参照できるので、debconf の notes を使うより良いです。そして、目立った変更点を README.Debian に列挙するより良いです。何故ならば、ユーザは容易にそのような注意書きを見逃すからです。

ファイル形式は debian changelog ファイルと同じですが、アスタリスク (*) を取って各ニュースを changelog に書くような簡潔な要約ではなく、必要に応じて完全な段落で記述してください。changelog のようにビルド中に自動的にはチェックされないので、ファイルを dpkg-parsechangelog を実行してチェックするのは良い考えです。実際の NEWS.Debian ファイルの例が、以下になります:

```
cron (3.0pl1-74) unstable; urgency=low

    The checksecurity script is no longer included with the cron package:
    it now has its own package, checksecurity. If you liked the
    functionality provided with that script, please install the new
    package.

 -- Steve Greenland <stevegr@debian.org>  Sat,  6 Sep 2003 17:15:03 -0500
```

NEWS.Debian ファイルは /usr/share/doc/*package*/NEWS.Debian.gz ファイルとしてインストールされます。圧縮されていて、Debian ネイティブパッケージ中でも常にこの名前です。debhelper を使う場合は、dh_installchangelogs が debian/NEWS ファイルをインストールしてくれます。

changelog ファイルと違って、毎回のリリースごとに NEWS.Debian ファイルを更新する必要はありません。何か特にユーザが知るべき目新しいことがあったときにのみ、更新してください。全くニュースがない場合、NEWS.Debian ファイルをパッケージに入れてリリースする必要はありません。便りが無いのは良い知らせ、です (No news is good news!)。

6.4 セキュリティに関するベストプラクティス

When an upstream publishes a cryptographic signature for every new release, you should setup `uscan` to automatically verify the latter. For details, refer to the section on `Upstream source location: "debian/watch"` in the Debian Policy Manual.

https://wiki.debian.org/Hardening has suggestions on how to build security hardened executables.

6.5 メンテナスクリプトのベストプラクティス

Maintainer scripts include the files `debian/postinst`, `debian/preinst`, `debian/prerm` and `debian/postrm`. These scripts take care of any package installation or deinstallation setup that isn't handled merely by the creation or removal of files and directories. The following instructions supplement the Debian Policy.

6.5.1 Generic guidelines

メンテナスクリプトは冪等でなければなりません。これは、通常は 1 回呼ばれるスクリプトが 2 回呼ばれた場合、何も悪いことが起きないのを保証する必要があるという意味です。

標準入出力はログの取得のためにリダイレクトされることがあります (例: パイプへ向けられる)。ですので、これらが tty であることに依存してはいけません。

質問や対話的な設定はすべて最小限に止めておく必要があります。必要になった時は、インターフェイスに `debconf` パッケージを使いましょう。どのような場合でも、質問は `postinst` スクリプトの `configure` 段階にのみ、配置することができます。

メンテナスクリプトは、できる限りシンプルなものにしておきましょう。我々は、あなたは純粋な POSIX シェルスクリプトを使っているものだと考えています。覚えておいて欲しいのですが、何かしら bash の機能が必要になったら、メンテナスクリプトは bash のシェバン行にしておく必要があります。スクリプトへ簡単にちょっとした変更を追加するのに `debhelper` を使えるので、Perl より POSIX シェル、あるいは Bash の方が好まれます。

メンテナスクリプトを変更したら、パッケージの削除や二重インストール、purge のテストを確認してください。purge されたパッケージが完全に削除されたことを確認してください。つまり、メンテナスクリプト中で直接／間接を問わず作成されたファイルを削除する必要があるということです。

コマンドの存在をチェックする必要がある場合は、このような感じで行います

```
if command -v install-docs > /dev/null; then ...
```

コマンド名を引数として渡すことで、`$PATH` の検索するのにこの関数を使うことができます。コマンドが見つかった場合は true (ゼロ) を返し、そうでない場合は false を返します。`command -v` は POSIX に定義されていて、多くのシェルで使えるので、これがもっとも汎用性の高いやり方です。

Using `which` is an acceptable alternative, since it is from the required `debianutils` package.

6.5.2 Expected behavior of complex postinst scripts

`postinst` scripts must normally not fail, except for very rare cases. These are cases where postinst must succeed:

on fresh installs:

- the service configuration is straightforward and can be reasonably expected to work as-is in typical Debian setups
- the service has no external dependencies (e.g. a database which may not yet be configured, or unreachable at install time)

on upgrades:

- `postinst` can verify with high confidence (which may not always be feasible) that the service was running prior to the restart
- the service has no external dependencies or `postinst` can verify that they are functional
- the service configuration has not changed in backwards incompatible ways between the old and new package versions

6.6 debconf による設定管理

`Debconf` is a configuration management system that can be used by all the various packaging scripts (`postinst` mainly) to request feedback from the user concerning how to configure the package. Direct user interactions must now be avoided in favor of `debconf` interaction. This will enable non-interactive installations in the future.

Debconf is a great tool but it is often poorly used. Many common mistakes are listed in the debconf-devel(7) man page. It is something that you must read if you decide to use debconf. Also, we document some best practices here.

これらのガイドラインは、ディストリビューションの一部 (例えば、インストールシステム) に関する、より明確な推奨と同様に、幾つかの書き方と体裁に関する推奨、そして debconf の使い方についての一般的な考慮すべき事柄を含んでいます。

6.6.1 debconf を乱用しない

debconf が Debian に現れて以来、広く乱用され続けています。そして、debconf の乱用によって、ちょっとしたものをインストールする前に、大量の質問に答える必要があることに由来するいくつもの非難が Debian ディストリビューションに寄せられました。

Keep usage notes to where they belong: the `NEWS.Debian`, or `README.Debian` file. Only use notes for important notes that may directly affect the package usability. Remember that notes will always block the install until confirmed or bother the user by email.

Carefully choose the questions' priorities in maintainer scripts. See debconf-devel 7 for details about priorities. Most questions should use medium and low priorities.

6.6.2 作者と翻訳者に対する雑多な推奨

6.6.2.1 正しい英語を書く

ほとんどの Debian パッケージメンテナはネイティブの英語話者ではありません。ですので、正しいフレーズのテンプレートを書くのは彼らにとっては容易ではありません。

debian-l10n-english@lists.debian.org メーリングリストを利用してください (むしろ乱用してください)。テンプレートを査読してもらいましょう。

下手に書かれたテンプレートは、パッケージに対して、そしてあなたの作業に対して、さらには Debian それ自体にすら対して、悪い印象を与えます。

可能な限り技術的なジャーゴンを避けましょう。いくつかの用語があなたにとっては普通に聞こえても、他の人には理解不可能かもしれません。避けられない場合には、 (説明文を使って) 解説してみましょう。その場合には、冗長さとシンプルさのバランスを取るようにしましょう。

6.6.2.2 翻訳者へ丁寧に接する

Debconf templates may be translated. Debconf, along with its sister package po-debconf, offers a simple framework for getting templates translated by translation teams or even individuals.

gettext ベースのテンプレートを使ってください。開発用のシステムに po-debconf をインストールしてドキュメントを読みましょう (man po-debconf が取っ掛かりとして良いでしょう)。

Avoid changing templates too often. Changing template text induces more work for translators, which will get their translation fuzzied. A fuzzy translation is a string for which the original changed since it was translated, therefore requiring some update by a translator to be usable. When changes are small enough, the original translation is kept in PO files but marked as fuzzy.

大本のテンプレートを変更する予定がある場合、po-debconf パッケージで提供されている、podebconf-report-po という名の通知システムを使って翻訳作業者にコンタクトを取ってください。ほとんどのアクティブな翻訳作業者たちはとても反応が良く、変更を加えたテンプレートに対応するための作業をしてくれ、あなたが追加でアップロードする必要を減らしてくれます。gettext ベースのテンプレートを使っている場合、翻訳作業者の名前とメールアドレスは PO ファイルのヘッダに表示されており、podebconf-report-po によって使われます。

このユーティリティの使い方のお勧めの使い方:

```
cd debian/po && podebconf-report-po --call --languageteam --withtranslators --
↪deadline="+10 days"
```

This command will first synchronize the PO and POT files in debian/po with the template files listed in debian/po/POTFILES.in. Then, it will send a call for new translations, in the debian-i18n@lists.debian.org mailing list. Finally, it will also send a call for translation updates to the language team (mentioned in the Language-Team field of each PO file) as well as the last translator (mentioned in Last-translator).

翻訳作業者に締切りを伝えるのは常にお勧めです。それによって、彼らは作業を調整できます。いくつかの翻訳作業チームは形式化された翻訳／レビュープロセスを整えており、10 日未満の猶予は不合理であると考えられています。より短い猶予期間は強すぎるプレッシャーを翻訳チームに与えるので、非常に些細な変更点に対してのみに留めるべきです。

迷った場合は、該当の言語の翻訳チーム (debian-l10n-xxxxx@lists.debian.org) か debian-i18n@lists.debian.org にも問い合わせましょう。

6.6.2.3 誤字やミススペルを修正する際に fuzzy を取る

debconf テンプレートの文章が修正されて、その変更が翻訳に**影響しない**と**確信している**場合には、翻訳作業者を労って翻訳文を *fuzzy* **を取り除いて**ください。

そうしないと、翻訳作業者が更新をあなたに送るまでテンプレート全体は翻訳されていない状態になります。

翻訳を *fuzzy* **ではなくす**ために、(po4a パッケージの一部である)msguntypot を使うことができます。

1. POT ファイルと PO ファイルを再生成する。

```
debconf-updatepo
```

2. POT ファイルのコピーを作成する。

```
cp templates.pot templates.pot.orig
```

3. すべての PO ファイルのコピーを作成する。

```
mkdir po_fridge; cp *.po po_fridge
```

4. debconf テンプレートファイルを誤字修正のために変更する。
5. POT ファイルと PO ファイルを再生成する (もう一度)。

```
debconf-updatepo
```

この時点では、typo 修正はすべての翻訳を fuzzy にしており、この残念な変更はメインディレクトリの PO ファイルと fridge の PO ファイルのみに適用されている。ここではどの様にしてこれを解決するかを示す。

6. fuzzy になった翻訳を捨て、fridge から作り直す。

```
cp po_fridge/*.po .
```

7. 手動で PO ファイルと新しい POT ファイルをマージするが、不要な fuzzy を考慮に入れる。

```
msguntypot -o templates.pot.orig -n templates.pot *.po
```

8. ゴミ掃除。

```
rm -rf templates.pot.orig po_fridge
```

6.6.2.4 インターフェイスについて仮定をしない

Templates text should not make reference to widgets belonging to some debconf interfaces. Sentences like *If you answer Yes...* have no meaning for users of graphical interfaces that use checkboxes for boolean questions.

文字列テンプレートは、説明文中でのデフォルト値について言及することも避けましょう。まず、ユーザによってそして、デフォルト値はメンテナの考え方によって違う場合があるからです (たとえば、debconf データベースが preseed で指定されている場合など)。

より一般的に言うと、ユーザの行動を参照させるのを避けるようにしましょう。単に事実を与えましょう。

6.6.2.5 一人称を使わない

You should avoid the use of first person (*I will do this...* or *We recommend...*). The computer is not a person and the Debconf templates do not speak for the Debian developers. You should use neutral construction. Those of you who already wrote scientific publications, just write your templates like you would write a scientific paper. However, try using the active voice if still possible, like *Enable this if ...* instead of *This can be enabled if...*.

6.6.2.6 性差に対して中立であれ

As a way of showing our commitment to our diversity statement, please use gender-neutral constructions in your writing. This means avoiding pronouns like he/she when referring to a role (like "maintainer") whose gender is unknown. Instead, you should use the plural form (singular they).

6.6.3 テンプレートのフィールド定義

This part gives some information which is mostly taken from the debconf-devel(7) manual page.

6.6.3.1 Type

string

ユーザがどのような文字列でも記述可能な自由記述形式の入力フィールドの結果。

password

ユーザにパスワードの入力を求めます。これを使う場合は注意して使ってください: ユーザが入力したパスワードは debconf のデータベースに書き込まれることに注意してください。おそらく、この値をデータベースから可能な限り早く消す必要があります。

boolean

true/false の選択です。注意点: true/false であって、**yes/no ではありません**...

select

複数の値から一つを選びます。選択するものは 'Choices' というフィールド名で指定されている必要があります。利用可能な値をコンマとスペースで区切ってください。以下のようになります: `Choices: yes, no, maybe`

選択肢が翻訳可能な文字列である場合、'Choices' フィールドは __Choices を使って翻訳可能であることを示しておくと良いでしょう。2 つのアンダースコアは、各選択肢を分かれた文字列に分割してくれます。

po-debconf システムは、翻訳可能な**いくつかの**選択肢のみをマークする面白い機能を提供してくれます。例:

```
Template: foo/bar
Type: Select
#flag:translate:3
__Choices: PAL, SECAM, Other
_Description: TV standard:
 Please choose the TV standard used in your country.
```

この例では、他は頭文字から構成されていて翻訳できませんが、'Other' 文字列だけは翻訳可能です。上記では 'Other' だけが PO および POT ファイルに含めることができます。

The debconf templates flag system offers many such possibilities. The po-debconf(7) manual page lists all these possibilities.

multiselect

Like the select data type, except the user can choose any number of items from the choices list (or choose none of them).

note

本来質問ではありませんが、このデータ型が示すのはユーザに表示することができる覚え書きです。debconf はユーザが必ず参照するようにするため、多大な苦痛を与えることになる (キーを押すためにインストールを休止したり、ある場合にはメモをメールさえする) ので、ユーザが知っておく必要がある重要な記述にのみ使うべきです。

text

この type は現状では古すぎるものと考えられています: 使わないでください。

error

This type is designed to handle error messages. It is mostly similar to the note type. Front ends may present it differently (for instance, the dialog front end of cdebconf draws a red screen instead of the usual blue one).

何かを補正するためにユーザの注意を引く必要があるメッセージに対し、この type を使うのがお勧めです。

6.6.3.2 Description: short および extended 説明文

テンプレート説明文は 2 つのパートに分かれます: short と extended です。短い説明文 (short description) はテンプレートの Description: 行にあります。

短い説明文は、ほとんどの debconf インターフェイスに適用するように、短く (50 文字程度に) しておく必要があります。通常、翻訳はオリジナルよりも長くなる傾向にあるので、短くすることは翻訳作業者を助けます。

The short description should be able to stand on its own. Some interfaces do not show the long description by default, or only if the user explicitly asks for it or even do not show it at all. Avoid things like: "What do you want to do?"

短い説明文は完全な文章である必要はありません。これは文章を短くしておき、効率的に推奨を行うためです。

拡張された説明文 (extended description) は、短い説明文を一語一句繰り返しをしてはなりません。長い説明文章を思いつかなければ、まず、もっと考えてください。debian-devel に投稿しましょう。助けを求めましょう。文章の書き方講座を受講しましょう! この拡張された説明文は重要です。もし、まったく何も思いつかなければ、空のままにしておきましょう。

拡張された説明文は完全な文章である必要があります。段落を短くしておくのは可読性を高めます。同じ段落で 2 つの考えを混ぜてはいけません。代わりに別の段落を書きます。

Don't be too verbose. Users tend to ignore too long screens. 20 lines are by experience a border you shouldn't cross, because that means that in the classical dialog interface, people will need to scroll, and lot of people just don't do that.

拡張された説明文では、質問を含めては**決していけません**。

テンプレートの type (string、boolean など) に応じた特別なルールについては、以下を読んでください。

6.6.3.3 Choices

This field should be used for select and multiselect types. It contains the possible choices that will be presented to users. These choices should be separated by commas.

6.6.3.4 Default

このフィールドはオプションです。これには、string、select あるいは multiselect のデフォルトでの答えが含まれます。multiselect テンプレートの場合、コンマで区切られた選択肢一覧が含まれます。

6.6.4 Template fields specific style guide

6.6.4.1 Type フィールド

特別な指定はありません。一点だけ、その前のセクションを参照して適切な type を使ってください。

6.6.4.2 Description フィールド

以下は、テンプレートの型に応じて適切な Description (short および extended) を書くための特別な指示です。

String/password テンプレート

- 短い説明文は、プロンプトであってタイトルでは**ありません**。質問形式のプロンプト (IP アドレスは?) を避け、代わりに閉じていない形のプロンプト (IP アドレス:) を使ってください。コロン (:) の使用をお勧めします。

- 拡張された説明文は、短い説明文を補足します。拡張の部分では、長い文章を使って同じ質問を繰り返すのではなく、何を質問されているのかを説明します。完全な文章を書いてください。簡潔な書き方は強く忌避されます。

Boolean テンプレート

- The short description should be phrased in the form of a question, which should be kept short and should generally end with a question mark. Terse writing style is permitted and even encouraged if the question is rather long (remember that translations are often longer than original versions).
- 繰り返しますが、特定のインターフェイスのウィジェットを参照するのを避けてください。このようなテンプレートで良くある間違いは、「はい」と答える形かどうかです。

Select/Multiselect

- The short description is a prompt and **not** a title. Do **not** use useless "Please choose..." constructions. Users are clever enough to figure out they have to choose something... :)
- 拡張された説明文は、短い説明文を完備します。これでは、利用可能な選択肢に言及することがあります。テンプレートが multiselect なものの場合、ユーザが選べる選択肢が 1 つではないことについても言及するかもしれません (インターフェイスが大抵これを明確にはしてくれますが)。

Note

- 短い説明文は**タイトル**として扱われます。
- 拡張された説明文では、note のより詳細な説明を表示します。フレーズで、簡潔過ぎない書き方です。
- **Do not abuse debconf.** Notes are the most common way to abuse debconf. As written in the debconf-devel(7) manual page: it's best to use them only for warning about very serious problems. The `NEWS.Debian` or `README.Debian` files are the appropriate location for a lot of notes. If, by reading this, you consider converting your Note type templates to entries in `NEWS.Debian` or `README.Debian`, please consider keeping existing translations for the future.

6.6.4.3 Choices フィールド

もし Choise が頻繁に変わるようであれば、__Choices という使い方をするのを検討してください。これは個々の選択項目を単一の文字列に分割するので、翻訳作業者が作業を行うのを助けてくれると考えられています。

6.6.4.4 Default フィールド

If the default value for a select template is likely to vary depending on the user language (for instance, if the choice is a language choice), please use the _Default trick, documented in po-debconf 7.

This special field allows translators to put the most appropriate choice according to their own language. It will become the default choice when their language is used while your own mentioned Default Choice will be used when using English.

Do not use an empty default field. If you don't want to use default values, do not use Default at all.

If you use po-debconf (and you **should**; see **翻訳者へ丁寧に接する**), consider making this field translatable, if you think it may be translated.

geneweb パッケージのテンプレートを例にとってみましょう:

```
Template: geneweb/lang
Type: select
__Choices: Afrikaans (af), Bulgarian (bg), Catalan (ca), Chinese (zh), Czech (cs),␣
↪Danish (da), Dutch (nl), English (en), Esperanto (eo), Estonian (et), Finnish (fi),␣
↪French (fr), German (de), Hebrew (he), Icelandic (is), Italian (it), Latvian (lv),␣
↪Norwegian (no), Polish (pl), Portuguese (pt), Romanian (ro), Russian (ru), Spanish␣
↪(es), Swedish (sv)
# This is the default choice. Translators may put their own language here
# instead of the default.
# WARNING : you MUST use the ENGLISH NAME of your language
# For instance, the French translator will need to put French (fr) here.
_Default: English[ translators, please see comment in PO files]
_Description: Geneweb default language:
```

Note the use of brackets, which allow internal comments in debconf fields. Also note the use of comments, which will show up in files the translators will work with.

The comments are needed as the _Default trick is a bit confusing: the translators may put in their own choice.

6.7 国際化

This section contains global information for developers to make translators' lives easier. More information for translators and developers interested in internationalization are available in the Internationalisation and localisation in Debian documentation.

6.7.1 debconf の翻訳を取り扱う

移植作業者同様に、翻訳作業者は難しい課題を抱えています。多くのパッケージについて作業を行い、多くの異なったメンテナと共同作業をする必要があります。さらには、ほとんどの場合、彼らはネイティブな英語話者ではないので、あなたは特に忍耐強くあらねばいけないでしょう。

The goal of `debconf` was to make package configuration easier for maintainers and for users. Originally, translation of debconf templates was handled with `debconf-mergetemplate`. However, that technique is now deprecated; the best way to accomplish `debconf` internationalization is by using the `po-debconf` package. This method is easier both for maintainer and translators; transition scripts are provided.

Using `po-debconf`, the translation is stored in `.po` files (drawing from `gettext` translation techniques). Special template files contain the original messages and mark which fields are translatable. When you change the value of a translatable field, by calling `debconf-updatepo`, the translation is marked as needing attention from the translators. Then, at build time, the `dh_installdebconf` program takes care of all the needed magic to add the template along with the up-to-date translations into the binary packages. Refer to the po-debconf(7) manual page for details.

6.7.2 ドキュメントの国際化

ドキュメントの国際化はユーザにとって極めて重要ですが、多くの労力がかかります。この作業をすべて除去する方法はありませんが、翻訳作業者を気楽にはできます。

どのようなサイズであれドキュメントをメンテナンスしている場合、翻訳作業者がソースコントロールシステムにアクセスできるのであれば、彼らの作業が楽になるでしょう。翻訳作業者が、ドキュメントの 2 つのバージョン間の違いを見ることができるので、例えば、何を再翻訳すればいいのかがわかるようになります。翻訳されたドキュメントは、翻訳作業がどのソースコントロールリビジョンをベースにしているのかという記録を保持しておくことをお勧めします。`debian-installer` パッケージ中の doc-check では興味深いシステムが提供されています。これは、翻訳すべき現在のリビジョンのファイルに対する構造化されているコメントを使って、指定されたあらゆる言語の翻訳状況の概要を表示し、翻訳されたファイルについては、翻訳がベースにしているオリジナルのファイルのリビジョンを表示します。自分の VCS 領域でこれを導入して利用した方が良いでしょう。

If you maintain XML or SGML documentation, we suggest that you isolate any language-independent information and define those as entities in a separate file that is included by all the different translations. This makes it much easier, for instance, to keep URLs up to date across multiple files.

Some tools (e.g. `po4a`, `poxml`, or the `translate-toolkit`) are specialized in extracting the translatable material from different formats. They produce PO files, a format quite common to translators, which permits seeing what needs to be re-translated when the translated document is updated.

6.8 Best practices for `debian/patches`

Debian packages might suffer from bugs in the upstream code that you need to deal with. In the source format " 3.0 (quilt) "patches are stored in `debian/patches/` and automatically applied as listed in `debian/patches/series` when the source package is unpacked.

Patches should be documented following DEP-3.

Several tools exist to automate managing the patches. If you manage a source package outside of any Git repository, then your best option is likely `quilt`. Otherwise, you should consider to rely on Git's built-in features or on on the git packaging helper that you use (if any). In particular, for packages using `git-buildpackage`, you should use the `gbp pq` commands to manage the contents of the `debian/patches/` directory.

A single patch can be created with e.g. `git format-patch -1 d33286c` from a single commit. Avoid using `git show` as it lacks the full headers.

If the upstream fix is spread across multiple commits but makes sense to apply (and drop) in Debian as a single patch, one could use a command such as `git format-patch --stdout abc123..def456 > debian/patches/...` and append the `Bug` field only in the commit message of the first commit in the patch.

If one appends *.patch* to the url of a GitHub commit or Pull Request or GitLab commit or Merge Request, the resulting patch file is using this same format (as if it were generated by `git format-patch`).

Remember to always append a `Bug` header to the patch description so that a reader can follow the link to see where the bug was reported or patch submitted. If the purpose of the patch is to specifically divert from upstream permanently, append the header *Forwarded: not-needed* to the end of the description.

6.9 パッケージ化に於ける一般的なシチュエーション

6.9.1 autoconf/automake を使っているパッケージ

autoconf の config.sub および config.guess を最新に保ちつづけるのは、移植作業者、特により移行中の状況であるアーキテクチャの移植作業者にとって非常に重要です。autoconf や automake を使うあらゆるパッケージについてのとても素晴らしいパッケージ化における教訓が autotools-dev パッケージの /usr/share/doc/autotools-dev/README.Debian.gz にまとめられています。このファイルを読んで書かれている推奨に従うことを強くお勧めします。

6.9.2 ライブラリ

ライブラリは様々な理由から常にパッケージにするのが難しいです。ポリシーは、メンテナンスに楽にし、新しいバージョンが開発元から出た際にアップグレードを可能な限りシンプルであることを確保するため、多くの制約を課しています。ライブラリでの破損は、依存している何十ものパッケージの破損を招き得ます。

Good practices for library packaging have been grouped in the library packaging guide.

6.9.3 ドキュメント化

ドキュメント化のポリシーに忘れず従ってください。

あなたのパッケージが XML や SGML から生成されるドキュメントを含んでいる場合、XML や SGML のソースをバイナリパッケージに含めてリリースしないことをお勧めします。ユーザがドキュメントのソースを欲しい場合には、ソースパッケージを引っ張ってくれば良いのです。

ポリシーではドキュメントは HTML 形式でリリースされるべきであると定めています。我々は、もし手がかからないで問題ない品質の出力が可能であれば、ドキュメントを PDF 形式とテキスト形式でもリリースすることをお勧めしています。ですが、ソースの形式が HTML のドキュメントを普通のテキスト版でリリースするのは、大抵の場合は適切ではありません。

リリースされたメジャーなマニュアルは、インストール時に doc-base で登録されるべきです。詳細については、doc-base パッケージのドキュメントを参照してください。

Debian ポリシー (12.1 章) では、マニュアルページはすべてのプログラム・ユーティリティ・関数に対して付属すべきであり、設定ファイルのようなその他のものについては付属を提案を示しています。もし、あなたがパッケージングをしているものがそのようなマニュアルページを持っていない場合は、パッケージに含めるために記述を行い、開発元 (upstream) へ送付することを検討してください。

The manpages do not need to be written directly in the troff format. Popular source formats are DocBook, POD and reST, which can be converted using `xsltproc`, `pod2man` and `rst2man` respectively. To a lesser extent, the `help2man` program can also be used to write a stub.

6.9.4 特定の種類のパッケージ

いくつかの特定の種類のパッケージは、特別なサブポリシーと対応するパッケージ化ルールおよびプラクティスを持っています:

- Perl related packages have a Perl policy; some examples of packages following that policy are `libdbd-pg-perl` (binary perl module) or `libmldbm-perl` (arch independent perl module).

- Python related packages have their Python policy; see `/usr/share/doc/python3/python-policy.txt.gz` in the `python3` package.
- Emacs 関連パッケージには、emacs ポリシーがあります。
- Java 関連パッケージには、java ポリシーがあります。
- OCaml related packages have their own policy, found in `/usr/share/doc/ocaml/ocaml_packaging_policy.gz` from the `ocaml` package. A good example is the `camlzip` source package.
- XML や SGML DTD を提供しているパッケージは、`sgml-base-doc` パッケージ中の推奨に従わねばなりません。
- Rust packaging is described in the Debian Rust Team Book;.
- Packages providing services ("daemons") should be functional on a fresh install, to the extent that that is possible without compromising security (e.g. a web server should by default be up and running and serve a dummy page, but must otherwise not allow unauthenticated sensitive operations; consider whether to serve only on the localhost network interface, by default).
- Web application packages should aim to have their dependencies (including javascript) packaged separately, and should carry out whatever setup is necessary for basic and secure functionality out of the box (e.g. create a database, ship configs with reasonable defaults, install files in appropriate location with appropriate permissions, etc). For examples, look at how existing web applications are packaged, e.g. dfsg-new-queue for Go, gitlab for ruby on rails, node-shiny-server for NPM. diaspora-installer is a dummy package which downloads diaspora (also pulling in runtime dependencies as rubygems) and configures it to use PostgreSQL and Nginx.

6.9.5 アーキテクチャ非依存のデータ

大量のアーキテクチャ非依存データがプログラムと共にパッケージ化されるのは、良くあることではありません。例えば、音声ファイル、アイコン集、様々な模様の壁紙、その他一般的な画像ファイルです。このデータのサイズがパッケージの残りのサイズと比較して取るに足らないのであれば、おそらくは単一パッケージでひとまとめにしておくのがベストでしょう。

However, if the size of the data is considerable, consider splitting it out into a separate, architecture-independent package (`_all.deb`). By doing this, you avoid needless duplication of the same data into ten or more .debs, one per each architecture. While this adds some extra overhead into the `Packages` files, it saves a lot of disk space on Debian mirrors. Separating out architecture-independent data also reduces processing time of `lintian` (see **パッケージチェック *(lint)* 用ツール**) when run over the entire Debian archive.

6.9.6 ビルド中に特定のロケールが必要

ビルド中に特定のロケールを必要とする場合、こんな技を使えば一時ファイルを作成できます:

`LOCPATH` を `/usr/lib/locale` と同等のものに、そして `LC_ALL` を生成したロケールの名前に設定すれば、root にならなくても欲しいものが手に入ります。こんな感じです:

```
LOCALE_PATH=debian/tmpdir/usr/lib/locale
LOCALE_NAME=en_IN
LOCALE_CHARSET=UTF-8

mkdir -p $LOCALE_PATH
localedef -i $LOCALE_NAME.$LOCALE_CHARSET -f $LOCALE_CHARSET $LOCALE_PATH/$LOCALE_
↪NAME.$LOCALE_CHARSET

# Using the locale
LOCPATH=$LOCALE_PATH LC_ALL=$LOCALE_NAME.$LOCALE_CHARSET date
```

6.9.7 移行パッケージを deboprhan に適合するようにする

deborphan は、どのパッケージがシステムから安全に削除できるか、ユーザが検出するのを助けてくれるプログラムです; すなわち、どのパッケージも依存していないものです。デフォルトの動作は、使われていないライブラリを見つけ出すために libs と oldlibs セクションからのみ検索を行います。ですが、正しい引数を渡せば、他の使われていないパッケージを捕らえようとします。

For example, with `--guess-dummy`, `deborphan` tries to search all transitional packages which were needed for upgrade but which can now be removed. For that, it currently looks for the string dummy or transitional in their short description, though it would be better to search for both strings, as there are some dummy or transitional packages, which have other purposes.

ですので、あなたがそのようなパッケージを作る際には、 `transitional dummy package` を短い説明文に必ず追加してください。例を探す場合は、以下を実行してください: `apt-cache search .|grep dummy` または `apt-cache search .|grep transitional`

Also, it is recommended to adjust its section to `oldlibs` and its priority to `optional` in order to ease `deborphan`'s job.

6.9.8 `.orig.tar.{gz,bz2,xz}` についてのベストプラクティス

オリジナルのソース tarball には 2 種類あります: 手が入れられていない素のソース (Pristine source) と再パッケージした開発元のソース (repackaged upstream source) です。

6.9.8.1 手が入れられていないソース (Pristine source)

素のソース tarball (pristine source tarball) の特徴の定義は、`.orig.tar.{gz,bz2,xz}` は、開発元の作者によって公式に配布された tarball と 1 バイトたりとも違わない、というものです。[1] これは、Debian diff 内に含まれている Debian バージョンと開発元のバージョン間のすべての違いを簡単に比較するのにチェックサムを使えるようになります。また、オリジナルのソースが巨大な場合、既に upstream の tarball を持っている upstream の作者と他の者は、あなたのパッケージを詳細に調査したい場合、ダウンロード時間を節約できます。

[1] We cannot prevent upstream authors from changing the tarball they distribute without also incrementing the version number, so there can be no guarantee that a pristine tarball is identical to what upstream *currently* distributing at any point in time. All that can be expected is that it is identical to something that upstream once *did* distribute. If a difference arises later (say, if upstream notices that they weren't using maximal compression in their original distribution and then re-`gzip` it), that's just too bad. Since there is no good way to upload a new `.orig.tar.{gz,bz2,xz}` for the same version, there is not even any point in treating this situation as a bug.

There are no universally accepted guidelines that upstream authors follow regarding the directory structure inside their tarball, but `dpkg-source` is nevertheless able to deal with most upstream tarballs as pristine source. Its strategy is equivalent to the following:

1. 以下のようにして空の一時ディレクトリに tarball を展開します

```
zcat path/to/packagename_upstream-version.orig.tar.gz | tar xf -
```

2. もし、この後で、一時ディレクトリが 1 つのディレクトリ以外含まず他にどのファイルも無い場合、`dpkg-source` はそのディレクトリを **パッケージ名-開発元のバージョン** (.orig) にリネームします。tarball 中の最上位のディレクトリ名は問題にはされず、忘れられます。

3. それ以外の場合、upstream の tarball は共通の最上位ディレクトリ無しでパッケージされなくてはいけません (upstream の作者よ、恥を知りなさい!)。この場合、`dpkg-source` は一時ディレクトリ**そのもの**を **パッケージ名-開発元のバージョン** (.orig) へリネームします。

6.9.8.2 upstream のソースをパッケージしなおす

パッケージは手が入っていない素のソース tarball と共にアップロード**すべき**ですが、それが可能ではない場合が色々とあります。upstream がソースを gzip 圧縮した tarball を全く配布していない場合や、upstream の tarball が DFSG-free ではない、あなたがアップロード前に削除しなければならない素材を含んでいる場合がこれにあたります。

この様な場合、開発者は適切な `.orig.tar.{gz,bz2,xz}` ファイルを自身で準備する必要があります。この様な tarball を、再パッケージした開発元のソース (repackaged upstream source) と呼びます。再パッケージした開発元のソースでは Debian ネイティブパッケージとは違うことに注意してください。再パッケージしたソースは、Debian 固有の変更点は分割された `.diff.gz` または `.debian.tar.{gz,bz2,xz}` のままであり、バージョン番号は **開発元のバージョン** と *debian* **リビジョン** から構成されたままです。

開発元が、原則的にはそのままの形で使える `.tar.{gz,bz2,xz}` を配布していたとしても再パッケージをしたくなるという場合がありえます。最も明解なのは、tar アーカイブを再圧縮することや upstream のアーカイブから純粋に使われていないゴミを削除することで、**非常に**容量を節約できる時です。ここで慎重になって頂きたいのですが、ソースを再パッケージする場合は、決定を貫く用意をしてください。

パッケージしなおした `.orig.tar.{gz,bz2,xz}` では、

1. **should** be documented in the resulting source package. Detailed information on how the repackaged source was obtained, and on how this can be reproduced should be provided in `debian/copyright`, ideally in a way that can be done automatically with uscan. If that really doesn't work, at least provide a `get-orig-source` target in your `debian/rules` file that repeats the process, even though that was actually deprecated in the 4.1.4 version of the Debian policy.

2. 開発元の作者由来ではないファイルや、あなたが内容を変更したファイルを含める**べきではありません**。[2]

3. 法的理由から不可能であるものを除いて、開発元の作者が提供しているビルドおよび移植作業環境を完全に**保全すべき**です。例えば、ファイルを省略する理由として MS-DOS でのビルドにしか使われないから、というのは十分な理由にはなりません。同様に、開発元から提供されている `Makefile` を

[2] As a special exception, if the omission of non-free files would lead to the source failing to build without assistance from the Debian diff, it might be appropriate to instead edit the files, omitting only the non-free parts of them, and/or explain the situation in a `README.source` file in the root of the source tree. But in that case please also urge the upstream author to make the non-free components easier to separate from the rest of the source.

省略すべきではありません。たとえ debian/rules が最初にすることが configure スクリプトを実行してそれを上書きすることであったとしても、です。

(**理由**: Debian 以外のプラットフォームのためにソフトウェアをビルドする必要がある Debian ユーザが、開発元の一次配布先を探そうとせずに Debian ミラーからソースを取得する、というのは普通です)。

4. **may** use *packagename-upstream-version*+dfsg (or any other suffix which is added to the tarball name) as the name of the top-level directory in its tarball. This makes it possible to distinguish pristine tarballs from repackaged ones.

5. **should** be compressed with xz (or gzip or bzip2) with maximal compression.

6.9.8.3 バイナリファイルの変更

Sometimes it is necessary to change binary files contained in the original tarball, or to add binary files that are not in it. This is fully supported when using source packages in " 3.0 (quilt) " format; see the dpkg-source(1) manual page for details. When using the older format " 1.0 ", binary files can't be stored in the .diff.gz so you must store a uuencoded (or similar) version of the file(s) and decode it at build time in debian/rules (and move it in its official location).

6.9.9 デバッグパッケージのベストプラクティス

A debug package is a package that contains additional information that can be used by gdb. Since Debian binaries are stripped by default, debugging information, including function names and line numbers, is otherwise not available when running gdb on Debian binaries. Debug packages allow users who need this additional debugging information to install it without bloating a regular system with the information.

The debug packages contain separated debugging symbols that gdb can find and load on the fly when debugging a program or library. The convention in Debian is to keep these symbols in /usr/lib/debug/*path*, where *path* is the path to the executable or library. For example, debugging symbols for /usr/bin/foo go in /usr/lib/debug/usr/bin/foo, and debugging symbols for /usr/lib/libfoo.so.1 go in /usr/lib/debug/usr/lib/libfoo.so.1.

6.9.9.1 Automatically generated debug packages

Debug symbol packages can be generated automatically for any binary package that contains executable binaries, and except for corner cases, it should not be necessary to use the old manually generated ones anymore. The package name for a automatic generated debug symbol package ends in -dbgsym.

The dbgsym packages are not installed into the regular archives, but in dedicated archives. That means, if you need the debug symbols for debugging, you need to add this archives to your apt configuration and then install the dbgsym package you are interested in. Please read https://wiki.debian.org/HowToGetABacktrace on how to do that.

6.9.9.2 Manual -dbg packages

Before the advent of the automatic `dbgsym` packages, debug packages needed to be manually generated. The name of a manual debug packages ends in `-dbg`. It is recommended to migrate such old legacy packages to the new `dbgsym` packages whenever possible. The procedure to convert your package is described in https://wiki.debian.org/AutomaticDebugPackages but the gist is to use the `--dbgsym-migration='pkgname-dbg (<< currentversion~)'` switch of the `dh_strip` command.

However, sometimes it is not possible to convert to the new `dbgsym` packages, or you will encounter the old manual -dbg packages in the archives, so you might need to deal with them. It is not recommended to create manual -dbg packages for new packages, except if the automatic ones won't work for some reason.

One reason could be that debug packages contains an entire special debugging build of a library or other binary. However, usually separating debugging information from the already built binaries is sufficient and will also save space and build time.

This is the case, for example, for debugging symbols of Python extensions. For now the right way to package Python extension debug symbols is to use `-dbg` packages as described in https://wiki.debian.org/Python/DbgBuilds.

To create `-dbg` packages, the package maintainer has to explicitly specify them in `debian/control`.

The debugging symbols can be extracted from an object file using `objcopy --only-keep-debug`. Then the object file can be stripped, and `objcopy --add-gnu-debuglink` used to specify the path to the debugging symbol file. objcopy 1 explains in detail how this works.

デバッグパッケージは、そのパッケージがデバッグシンボルを提供するパッケージに依存する必要があり、この依存関係はバージョン指定が必要であるということに注意してください。以下のような例になります:

```
Depends: libfoo (= ${binary:Version})
```

The `dh_strip` command in `debhelper` supports creating debug packages, and can take care of using `objcopy` to separate out the debugging symbols for you. If your package uses `debhelper/9.20151219` or newer, you don't need to do anything. `debhelper` will generate debug symbol packages (as `package`-dbgsym) for you with no additional changes to your source package.

6.9.10 メタパッケージのベストプラクティス

メタパッケージは、時間がかかる一貫したセットのパッケージをインストールするのを楽にしてくれる、ほぼ空のパッケージです。そのセットの全パッケージに依存することで、これを実現しています。APT の力のおかげで、メタパッケージのメンテナは依存関係を調整すればユーザのシステムが自動的に追加パッケージを得ることができます。自動的にインストールされていてメタパッケージから落とされたパッケージは、削除候補としてマークされます (そして `aptitude` によって自動的に削除もされます)。`gnome` と `linux-image-amd64` は、メタパッケージの 2 つの例です (ソースパッケージ `meta-gnome2` and `linux-latest` から生成されています)。

The long description of the meta-package must clearly document its purpose so that the user knows what they will lose if they remove the package. Being explicit about the consequences is recommended. This is particularly important for meta-packages that are installed during initial installation and that have not been explicitly installed by the user. Those tend to be important to ensure smooth system upgrades and the user should be discouraged from uninstalling them to avoid potential breakages.

第7章　パッケージ化、そして…

Debian は、単にソフトウェアのパッケージを作ってメンテナンスをしているだけではありません。この章では、単にパッケージを作ってメンテナンスする以外で Debian へ協力・貢献するやり方、多くの場合とても重要となるやり方の情報を取り扱います。

ボランティア組織の例にたがわず、Debian の活動はメンバーが何をしたいのか、時間を割くのに最も重大だと思われることが何か、というメンバーの判断に依っています。

7.1 バグ報告

我々としては、Debian パッケージで見つけたバグを登録することを勧めています。実際のところ、大抵の場合は Debian 開発者が第一線でのテスト作業者となっています。他の開発者のパッケージで見つけたバグを報告することで Debian の品質が向上されています。

Debian バグ追跡システム (BTS) の バグ報告のやり方について (instructions for reporting bugs) を参照してください。

いつも使っているメールを受け取れるユーザアカウントからバグを送ってみてください。そうすることで、開発者がそのバグに関するより詳細な情報を必要とする場合にあなたに尋ねることができます。root ユーザでバグを報告しないでください。

バグを報告するには、reportbug 1 のようなツールが使えます。これによって作業を自動化し、かなり簡単なものにできます。

パッケージに対して既にバグが報告されていないことを確認しておいてください。個々のパッケージに対するバグのリストは `https://bugs.debian.org/`**パッケージ名** にて簡単に確認できます。querybts 1 のようなユーティリティでもこの情報を入手できます (なお、`reportbug` では大抵の場合、バグを送信する前に `querybts` の実行も行っています)。

正しい所にバグを報告する様に心がけてください。例えばあるパッケージが他のパッケージのファイルを上書きしてしまうバグの場合ですが、バグ報告が重複して登録されるのを避けるため、これらのパッケージの**両方**のバグリストを確認してください。

さらに言うと、他のパッケージについても、何度も報告されているバグをマージしたり既に修正されているバグに「fixed」タグをつけたりすることもできます。そのバグの報告者であったりパッケージメンテナではない場合は (メンテナから許可をもらっていなければ)、実際にバグをクローズしてはいけないことに注意してください。

時折、あなたが登録したバグ報告について何が起こっているのかをチェックしたくなることでしょう。これは、もう再現できないものをクローズするきっかけになります。報告した全てのバグ報告を確認するには、`https://bugs.debian.org/from:`*your-email-addr* を参照すればいいだけです。

7.1.1 一度に大量のバグを報告するには (mass bug filing)

大量の異なるパッケージに対して、同じ問題についての非常に多くのバグ (例えば 10 個以上) を報告するのは、推奨されていないやり方です。不要なバグ報告を避けるため、可能な限りの手続きを踏むようにしましょう。例えば、問題の確認を自動化できる場合は `lintian` に新しくチェック項目を追加すれば、エラーや警告が明確になります。

同じ問題について一度に 10 個以上のバグを報告する場合は、バグ報告を登録する前に `debian-devel@lists.debian.org` へ送ることをお勧めします。バグ報告を送る前に注意点を記述し、メールのサブジェクトに事実を挙げておきます。これで他の開発者がそのバグが本当に問題であるかどうかを確認できるようになります。さらに、これによって複数のメンテナが平行して同じバグ報告を登録するのを防止できるようになります。

`dd-list` プログラムを利用することと、明確になっているのであれば影響を受けるパッケージのリストを (`devscripts` パッケージの) `whodepends` を使って出力して、`debian-devel@lists.debian.org` へのメールに含めて下さい。

同じサブジェクトで大量のバグを送信する際は、バグ報告がバグ情報用メーリングリストへ転送されないように `maintonly@bugs.debian.org` へバグ報告を送るべきであるの注意してください。

The program `mass-bug` (from the package `devscripts`) can be used to file bug reports against a list of packages.

7.1.1.1 Usertag

多数のパッケージに対するバグを登録する際に BTS の usertag を使いたくなるかもしれません。usertag は 'patch' や 'wishlist' のような通常のタグに似ていますが、ユーザが定義する事と特定のユーザに対して一意な名前空間を占めるという点で違っています。これによって、同じバグについて衝突する事無しに、開発者がそれぞれ別のやり方で複数の設定ができるようになります。

バグを登録する際に usertag を追加するには、擬似ヘッダ (pseudo-header) `User` と `Usertags` を指定します。

```
To: submit@bugs.debian.org
Subject: title-of-bug

Package: pkgname
[ ... ]
User: email-addr
Usertags: tag-name [ tag-name ... ]

description-of-bug ...
```

Note that tags are separated by spaces and cannot contain underscores. If you are filing bugs for a particular group or team it is recommended that you set the `User` to an appropriate mailing list after describing your intention there.

特定の usertag でバグを参照する場合は `https://bugs.debian.org/cgi-bin/pkgreport.cgi?users=`**メールアドレス**`&tag=`**タグ名** を指定してください。

7.2 品質維持の努力

7.2.1 日々の作業

品質保証に割り当てられたグループの人々がいたとしても、QA 作業は彼らのみに課せられるものではありません。あなたもパッケージを可能な限りバグが無いように保ち、できるだけ lintian clean な状態 (*lintian* を参照) にすることで品質保証の作業に参加することができるのです。それが可能ではないように思えるなら、パッケージをいくつか「放棄 (orphan)」してください (**パッケージを放棄する** 参照)。または、溜まったバグ処理に追いつくため、他の人々に助力を願い出ることも可能です (`debian-qa@lists.debian.org` や `debian-devel@lists.debian.org` で助けを求めることができます)。同時に共同メンテナ (co-maintainer) を探すことも可能です (**共同メンテナンス** を参照してください)。

7.2.2 バグ潰しパーティ (BSP)

From time to time the QA group organizes bug squashing parties to get rid of as many problems as possible. They are announced on `debian-devel-announce@lists.debian.org` and the announcement explains which area will be the focus of the party: usually they focus on release critical bugs but it may happen that they decide to help finish a major upgrade (like a new `perl` version that requires recompilation of all the binary modules).

The rules for non-maintainer uploads differ during the parties because the announcement of the party is considered prior notice for NMU. If you have packages that may be affected by the party (because they have release critical bugs for example), you should send an update to each of the corresponding bugs to explain their current status and what you expect from the party. If you don't want an NMU, or if you're only interested in a patch, or if you will deal with the bug yourself, please explain that in the BTS.

People participating in the party have special rules for NMU; they can NMU without prior notice if they upload their NMU to DELAYED/3-day at least. All other NMU rules apply as usual; they should send the patch of the NMU to the BTS (to one of the open bugs fixed by the NMU, or to a new bug, tagged fixed). They should also respect any particular wishes of the maintainer.

NMU をする自信が無い場合は、BTS へパッチを投げるだけにしてください。NMU でパッケージを壊してしまうより、遥かに良いことです。

7.3 他のメンテナに連絡を取る

Debian と共に過ごす間、様々な理由で他のメンテナに連絡を取りたくなることでしょう。関連パッケージ間での共同作業の新たなやり方について協議したい場合や、開発元で自分が使いたい新しいバージョンが利用可能になっていることを単に知らせたい場合などです。

パッケージメンテナのメールアドレスを探しだすのは骨が折れます。幸いな事に、**パッケージ名**`@packages.debian.org` というシンプルなメールのエイリアスがあり、メンテナらの個人アドレスが何であれメンテナへメールを届ける手段となっています。**パッケージ名** はパッケージのソース名かバイナリパッケージ名に置き換えてください。

Debian **パッケージトラッカー** 経由でソースパッケージの購読を行っている人に連絡を取りたくなるかもしれません。その場合は **パッケージ名**`@packages.qa.debian.org` メールアドレスが使えます。

7.4 活動的でない、あるいは連絡が取れないメンテナに対応する

If you notice that a package is lacking maintenance, you should make sure that the maintainer is active and will continue to work on their packages. It is possible that they are not active anymore, but haven't registered out of the system, so to speak. On the other hand, it is also possible that they just need a reminder.

There is a simple system (the MIA database) in which information about maintainers who are deemed Missing In Action is recorded. When a member of the QA group contacts an inactive maintainer or finds more information about one, this is recorded in the MIA database. This system is available in the directory `/org/qa.debian.org/mia` on the host `qa.debian.org`. The directory contains the tool `mia-query` with which the database can be queried. Use `/org/qa.debian.org/mia/mia-query --help` to see how to query the database. For example, to find out information about Debian Developer named Maria you could run:

```
ssh qa.debian.org /org/qa.debian.org/mia/mia-query --name Maria
```

If you find that no information has been recorded about an inactive maintainer yet, or that you can add more information, you should generally proceed as follows.

最初の一歩はメンテナに丁寧にコンタクトを取り、応答するのに充分な時間待つことです。充分な時間というのを定義するのは非常に困難ですが、実生活では時折非常に多忙になるのを考慮に入れると重要なことです。一つのやり方としては、リマインダーを二週間後に送るという方法があります。

A non-functional e-mail address is a violation of Debian Policy. If an e-mail "bounces", please file a bug against the package and submit this information to the MIA database.

メンテナが４週間 (１ヶ月）応答をしない場合、おそらく反応がないと判断できます。この様な場合はより詳細に確認し、可能な限り問題となっているメンテナに関する有用な情報をかき集める必要があります。これには以下のようなものが含まれています。

- 開発者 LDAP データベース を通じて得られる `echelon` 情報は、開発者が最後に Debian メーリングリストに投稿したはいつなのかを示します (これには `debian-devel-changes@lists.debian.org` での配布物のアップロードのメールも含まれます)。また、データベースでメンテナが休暇中かどうかも確認してください
- このメンテナが対応しているパッケージ数やパッケージの状態。特に、長期間放置され続けている RC バグがあるかどうか? さらに通常どの程度の数のバグがあるか? もう一つの重要な情報はパッケージが NMU されているかどうか、されているとしたら誰によって行われているか、です。
- Debian 以外でメンテナの活動があるかどうか? 例えば、近頃 Debian 以外のメーリングリストや news グループへの投稿をしているなど。

パッケージがスポンサーされている、つまりメンテナが公式の Debian 開発者ではない場合はちょっとした問題となります。例えば `echelon` の情報は、スポンサーされている人は利用できません。そのため実際にパッケージをアップロードした Debian 開発者を探して確認を取る必要があります。彼らがパッケージにサインしたということは、アップロードについて何であれ責任を持ち、スポンサーした人に何が起こっているかを知っていそうだということです。

`debian-devel@lists.debian.org` に、活動が見えないメンテナの居所に誰か気づいているかという質問を投稿するのもありです。問題の人を Cc: に入れてください。

ここに書かれた全てを収集したなら、`mia@qa.debian.org` に連絡しましょう。この名前の宛先を担当している人は、あなたが供給した情報を使ってどう進めるかを判断します。例えば、そのメンテナのパッケージ

の一部または全てを放棄 (Orphan) するかもしれません。パッケージが NMU されていた場合は、パッケージを放棄 (Orphan) する前に NMU をした人に連絡する事を選ぶかもしれません– NMU をした人はきっとパッケージに関心があるでしょうから。

最後に一言: 礼儀正しく振る舞いましょう。我々は所詮ボランティアで、全ての時間を Debian に捧げられるわけではありません。また、関わっている人の状況がわかるわけでもありません。重い病気にかかっているかかもしれないし、あるいは死んでしまっているかもしれません - メッセージを受け取る側にどんな人がいるかは分かりません。亡くなった方のご親戚の方がメールを読んだ場合に、非常に無礼で怒った叱責調のメッセージを見つけてどうお感じになるかを想像してください。

On the other hand, although we are volunteers, a package maintainer has made a commitment and therefore has a responsibility to maintain the package. So you can stress the importance of the greater good– if a maintainer does not have the time or interest anymore, they should let go and give the package to someone with more time and/or interest.

If you are interested in working on the MIA team, please have a look at the `README` file in `/org/qa.debian.org/mia` on `qa.debian.org`, where the technical details and the MIA procedures are documented, and contact `mia@qa.debian.org`.

7.5 Dealing with repeatedly negligent Debian developers

Debian developers are expected to comply with a series of technical and non-technical policies (for example, respectively, the Debian Machine Usage Policy and the Debian Code of Conduct).

To err is human and one should generally assume good intent, but repeated technical mistakes and/or disrespectful interactions, combined with a dismissive attitude towards Debian policies should be referred to the Debian Community Team. Most escalations to the Community Team are resolved amicably, but extreme cases may escalate further to the Debian Project Leader (leader@debian.org) and the Debian Account Managers (da-manager@debian.org) and could ultimately lead to expulsion from the Debian project.

7.6 Debian 開発者候補に対応する

Debian の成功は新たな才能あるボランティアをどう魅了し確保するかにかかっています。あなたが経験豊かな開発者なら、新たな開発者を呼び込むプロセスに関与するべきです。このセクションでは新たな開発者候補者をどうやって手助けするのかについて記述します。

7.6.1 パッケージのスポンサーを行う

パッケージのスポンサーをするというのは、パッケージをアップロードする権限をもたないメンテナーのためにかわりにアップロードするということです。アップロードを軽く考えてはなりません。スポンサーはパッケージを検証し、Debian に含まれるのにふさわしい高水準の品質であることを担保すべきです。

Debian 開発者はパッケージをスポンサーできます。Debian メンテナはできません。

パッケージのスポンサー作業の流れは以下の通りです:

1. The maintainer prepares a source package (`.dsc`) and puts it online somewhere (like on mentors.debian.net) or even better, provides a link to a public VCS repository (see *salsa.debian.org: Git repositories and collaborative development platform*) where the package is maintained.

2. スポンサーはソースパッケージをダウンロード (もしくはチェックアウト) します。
3. スポンサーはソースパッケージをレビューします。問題を見つけたら、メンテナに知らせて修正版をくれるように尋ねます (作業は step 1 へやり直しされます)。
4. スポンサーは、何も問題が残っているのを見つけられませんでした。パッケージをビルドし、署名し、Debian へアップロードします。

どのようにパッケージをスポンサーするのかの詳細を掘り下げるまえに、提案されているパッケージを Debian に追加するメリットがあるのかを問うべきです。

この質問に答えるシンプルなルールは存在しません。様々な要素に依存しうるからです。アップストリームのコードが十分枯れており、脆弱性がまったくないか? 同じようなことができるパッケージがすでにパッケージ化されていないだろうか? そして新しいパッケージはそれらと比較してどんな違いがあるだろうか? 新しいパッケージはユーザーの要望があるだろうか? どれくらいのユーザー数がいるだろうか? アップストリームの開発者はどれくらい精力的に活動しているだろうか?

それから、メンテナ候補者が良いメンテナになるであろうことを保証する必要があります。他のパッケージでの経験がありますか? そうであれば、良い仕事をしていますか (バグを確認している)? パッケージと使われているプログラミング言語について詳しいですか? そのパッケージに必要なスキルを持っていますか? そうでなければ、学ぶことが可能でしょうか?

Debian についてどのような立ち位置にあるかを知るのもよいでしょう: 彼らは Debian の哲学に同意し、Debian コミュニティーへ参加するつもりがあるでしょうか? Debian メンバーになることは簡単なので、Debian メンバーになろうと思っている人だけをスポンサーしたいことでしょう。そのようにすれば無期限にスポンサーとして活動する必要がないことが最初からわかります。

7.6.1.1 新しいパッケージのスポンサーを行う

New maintainers usually have certain difficulties creating Debian packages– this is quite understandable. They will make mistakes. That's why sponsoring a brand new package into Debian requires a thorough review of the Debian packaging. Sometimes several iterations will be needed until the package is good enough to be uploaded to Debian. Thus being a sponsor implies being a mentor.

レビューをせずに新しいパッケージのスポンサーをしないでください。ftpmaster による新しいパッケージのレビューは、主にソフトウェアが本当にフリーなものであるかを確認するためです。もちろん、パッケージ化に関する問題に偶然気づくことはありますが、それを期待すべきではありません。アップロードされたパッケージが、Debian フリーソフトウェアガイドラインに適合し、良い品質であるのを保証するのは、あなたの仕事です。

パッケージをビルドし、ソフトウェアのテストを行うのはレビューの一部ではありますが、それだけでは十分ではありません。この章の残りの部分では、レビューでチェックするポイントの一覧を述べます (徹底的なものではありません)。[1]

- upstream の tarball として提供されているものが、upstream の作者が配布しているものと同じかどうかを確認する (ソースが Debian 用に再パッケージされている場合、修正した tarball を自分自身で生成する)。
- Run `lintian` (see *lintian*). It will catch many common problems. Be sure to verify that any `lintian` overrides set up by the maintainer are fully justified.

[1] You can find more checks in the wiki, where several developers share their own sponsorship checklists.

- Run `licensecheck` (see *licensecheck*) and verify that `debian/copyright` seems correct and complete. Look for license problems (like files with " All rights reserved " headers, or with a non-DFSG compliant license). `grep -ri` is your friend for this task.
- ビルドの依存関係が完全であるのを保証するため、パッケージを `pbuilder` (やその他類似のツール) でビルドする (*pbuilder* 参照)。
- `debian/control` を査読する: ベストプラクティスに従っている? (*debian/control* **のベストプラクティス** 参照) 依存関係は完璧ですか?
- `debian/rules` を査読する: ベストプラクティスに従っている? (*debian/rules* **についてのベストプラクティス** 参照) 改善可能な点がある?
- メンテナスクリプト (`preinst`, `postinst`, `prerm`, `postrm`, `config`) を査読する: 依存関係がインストールされていない時でも動作する? 全てのスクリプトが等冪 (idempotent、すなわち、問題無しに複数回実行できる)?
- 開発元のファイルに対する変更 (`.diff.gz`、`debian/patches/`、あるいは直接 `debian` tarball に埋め込まれているバイナリファイル) をレビューする。十分な根拠がありますか? (パッチに対し、DEP-3 に沿って) 正しくドキュメント化されている?
- すべてのファイルについて、そのファイルが何故そこにあるのか、望んでいる結果をもたらすためにそれが正しいやり方かどうかを自身に問いかけてください。メンテナはパッケージ化のベストプラクティスに従っていますか? (**パッケージ化のベストプラクティス** 参照)
- Build the packages, install them and try the software. Ensure that you can remove and purge the packages. Maybe test them with `piuparts`.

If the audit did not reveal any problems, you can build the package and upload it to Debian. Remember that even if you're not the maintainer, as a sponsor you are still responsible for what you upload to Debian. That's why you're encouraged to keep up with the package through *Debian* **パッケージトラッカー**.

Note that you should not need to modify the source package to put your name in the `changelog` or in the `control` file. The `Maintainer` field of the `control` file and the `changelog` should list the person who did the packaging, i.e. the sponsee. That way they will get all the BTS mail.

Instead, you should instruct `dpkg-buildpackage` to use your key for the signature. You do that with the `-k` option:

```
dpkg-buildpackage -kKEY-ID
```

`debuild` と `debsign` を使う場合は、`~/.devscripts` に設定を決め打ちで書いても構いません:

```
DEBSIGN_KEYID=KEY-ID
```

7.6.1.2 既存パッケージの更新をスポンサーする

You will usually assume that the package has already gone through a full review. So instead of doing it again, you will carefully analyze the difference between the current version and the new version prepared by the maintainer. If you have not done the initial review yourself, you might still want to have a deeper look just in case the initial reviewer was sloppy.

To be able to analyze the difference, you need both versions. Download the current version of the source package (with `apt-get source`) and rebuild it (or download the current binary packages with `aptitude download`). Download the source package to sponsor (usually with `dget`).

Read the new changelog entry; it should tell you what to expect during the review. The main tool you will use is `debdiff` (provided by the `devscripts` package); you can run it with two source packages (`.dsc` files), or two binary packages, or two `.changes` files (it will then compare all the binary packages listed in the `.changes`).

ソースパッケージを比較した場合 (新しい開発元のバージョンの場合には、例えば `debdiff` の出力を `filterdiff -i '*/debian/*'` などとして、開発元のファイルを除外します)、確認したすべての変更点を理解して、この変更点が Debian の changelog に正しく記載されている必要があります。

If everything is fine, build the package and compare the binary packages to verify that the changes on the source package have no unexpected consequences (some files dropped by mistake, missing dependencies, etc.).

You might want to check out the Package Tracking System (see *Debian* **パッケージトラッカー**) to verify if the maintainer has not missed something important. Maybe there are translation updates sitting in the BTS that could have been integrated. Maybe the package has been NMUed and the maintainer forgot to integrate the changes from the NMU into their package. Maybe there's a release critical bug that they have left unhandled and that's blocking migration to `testing`. If you find something that they could have done (better), it's time to tell them so that they can improve for next time, and so that they have a better understanding of their responsibilities.

何も大きな問題を見つけなければ、新しいバージョンをアップロードします。そうでなければ、メンテナに修正したバージョンをアップロードするよう要請します。

7.6.2 Granting upload permissions to DMs

Debian メンテナーの鍵は debian-maintainers キーリングに追加され、Debian 開発者は特定のパッケージに関するアップロード権限を DM に許可することがあります。これは署名した dak コマンドを FTP-Master による debian-devel へのアナウンス のとおりに ftp.upload.debian.org にアップロードすることで行います。

このプロセスは `dput-ng` パッケージに含まれる `dcut` コマンドにより簡略化できます。 `dput` パッケージの `dcut` では動作しないことに注意が必要です!

例:

```
dcut dm --uid 0xfedcba9876543210 --allow nano --deny bash
```

DM の鍵がキーリングパッケージに含まれていないが DD のローカルのキーリングに含まれている場合、`--force` オプションとスペースを含まないフィンガープリントを指定します。とりわけ 0x というプレフィクスをつけず、すべて大文字を指定します。

```
dcut --force dm --uid FEDCBA9876543210FEDCBA9876543210 --allow nano
```

7.6.3 新たな開発者を支持する (advocate)

Debian ウェブサイトの開発者志願者の支持者になる (advocating a prospective developer) のページを参照してください。

7.6.4 新規メンテナ申請 (new maintainer applications) を取り扱う

Debian のウェブサイトにある 申請管理者用チェックリスト (Checklist for Application Managers) を参照してください。

第8章　国際化と翻訳

Debian がサポートしている自然言語の数は未だ増え続けています。あなたが英語圏のネイティブスピーカーで他の言語を話さないとしても、国際化の問題について注意を払うことはメンテナとしてのあなたの責務です (internationalization の 'i' と 'n' の間に 18 文字があるので i18n と略されます)。つまり、あなたが英語のみのプログラムを扱っていて問題がない場合であっても、この章の大部分を読んでおく必要があるということです。

i18n means modification of software or related technologies so that software can potentially handle multiple languages, customs, and other differences, while L10N (localization) means implementation of a specific language for already-internationalized software.

l10n と i18n は関連していますが、それぞれ関連する難しさについては違います。プログラムをユーザの設定に応じて表示されるテキストの言語を変更するようにするのはあまり難しくはありませんが、実際にメッセージを翻訳するのはとても時間がかかります。一方、文字のエンコード設定は些細な事ですが、複数の文字エンコードを扱えるようなコードにするのはとても難しい問題です。

i18n の問題を横においたとしても、一般的なガイドラインは与えられておらず、移植作業用の buildd のメカニズムと比較できるような、Debian での l10n 用の中心となるインフラは実際のところ存在していません。そのため、多くの作業は手動で行わねばなりません。

8.1 どの様にして Debian では翻訳が取り扱われているか

パッケージに含まれている文章の翻訳の取り扱いは未だ手動であり、作業のやり方は翻訳を表示させたい文の種類に因ります。

For program messages, the gettext infrastructure is used most of the time. Often the translation is handled upstream within projects like the Free Translation Project, the GNOME Translation Project or the KDE Localization project. The only centralized resources within Debian are the Central Debian translation statistics, where you can find some statistics about the translation files found in the actual packages and download those files.

Package descriptions have translations since many years and Maintainers don't need to do anything special to support translated package descriptions; translators should use the Debian Description Translation Project (DDTP).

For `debconf` templates, maintainers should use the `po-debconf` package to ease the work of translators. Some statistics can be found on the Central Debian translation statistics site.

ウェブページについては、それぞれの l10n チームが対応する VCS にアクセスし、Debian の翻訳に関する統計サイトから統計情報が取得できます。

Debian についての一般的なドキュメントは、作業は多少の差はあれウェブページと同じです (翻訳者は VCS にアクセスします)。ですが、統計情報のページはありません。

Another part of i18n work is package-specific documentation (man pages, info documents, other formats). At least the man page translations are po-based as most other things mentioned above.

8.2 メンテナへの I18N & L10N FAQ

これはメンテナが i18n や l10n を考えるのにあたって直面するであろう問題の一覧です。読み進める間、Debian でこれらの点について実際のコンセンサスは得られていないことを念頭においてください。これは単にアドバイスです。出てきた問題についてもっと良い考えがある、あるいはいくかの点で賛同できないという場合は、連絡をして頂いて構いません。そのことによって、この文章の質をさらに高めることができます。

8.2.1 翻訳された文章を得るには

To translate package descriptions, you have nothing to do; the DDTP infrastructure will dispatch the material to translate to volunteers with no need for interaction on your part.

For all other material (`debconf` templates, gettext files, man pages, or other documentation), the best solution is to ask on debian-i18n for a translation in different languages. Some translation team members are subscribed to this list, and they will take care of the needed coordination, to get the material translated and reviewed. Once they are done, you will get your translated document from them in your mailbox or via a wishlist bugreport. It is also recommended, to use the `po-debconf` tools for i18n integration.

8.2.2 どの様にして提供された翻訳をレビューするか

時折、あなたのパッケージ内の文章を訳して翻訳をパッケージに含めるように依頼する人が出てきます。これはあなたがその言語に詳しくない場合、問題となり得ます。その文章を対応する l10n メーリングリストに投稿し、レビューを依頼するのが良い考えです。一旦レビューが終われば、翻訳の質に自信を持つでしょうし、パッケージに含めるのにも安心を覚えるでしょう。

8.2.3 どの様にして翻訳してもらった文章を更新するか

古いままになっていた文章に対して翻訳文がある場合、元の文章を更新する度に、以前翻訳した人に新たに変更した点に合わせて翻訳を更新してもらうように依頼する必要があります。この作業には時間がかかることを覚えておいてください—更新をレビューしてもらったりするには少なくとも 1 週間はかかります。

翻訳者が応答してこない場合、対応する l10n メーリングリストに助力を願い出ましょう。すべてうまくいかなかった場合は、翻訳した文中に翻訳がとにかく古い事の警告を入れておくの忘れないようにして、できれば読者がオリジナルの文章を参照するようにしましょう。

古くなっているからといって翻訳を全て削除するのは避けてください。非英語圏のユーザにとって何もドキュメントが無いよりは古いドキュメントがある方が有益であることが往々にしてあります。

8.2.4 どの様にして翻訳関連のバグ報告を取り扱うか

最も良い解決策は開発元のバグという印を付けておいて (forward)、以前の翻訳者と関連するチーム (対応する debian-l10n-XXX メーリングリスト) に転送することです。

8.3 翻訳者への I18N & L10N FAQ

これを読み進める間、Debian においてこれらの点に関する一般的な手続きは存在していないこと、そしていかなる場合でもチームやパッケージメンテナと協調して作業する必要があることを念頭においてください。

8.3.1 どの様にして翻訳作業を支援するか

翻訳したい文章を選び、誰もまだ作業をしていないことを確認し (debian-l10n-XXX メーリングリストを参照。日本語の場合は debian-doc@debian.or.jp を参照してください)、翻訳し、l10n メーリングリストで他のネイティブスピーカーにレビューをしてもらい、パッケージメンテナに提供します (次の段を参照)。

8.3.2 どの様にして提供した翻訳をパッケージに含めてもらうか

含めてもらう翻訳が正しいかどうかを提供する前に確認してください (l10n メーリングリストでレビューを依頼しましょう)。皆の時間を節約し、バグレポートに複数バージョンの同じ文章があるというカオス状態を避けることになります。

最も良いやり方は、パッケージに対して翻訳を含めて通常のバグとして登録することです。忘れずに「patch」や「l10n」タグを使い、翻訳が欠けていたとしてもプログラムの動作に支障は無いので「wishlist」以上の重要度を使わないようにしましょう。

8.4 I10n に関する現状でのベストプラクティス

- メンテナとしては、翻訳については関連の l10n メーリングリストに尋ねること無くどの様な方法であれいじらないこと (レイアウトを変えることでさえしないこと) です。もしいじってしまうと、例えばファイルのエンコーディングを破壊する危険があります。さらに、あなたが間違いだと思っていることがその言語では正解である (または必要ですらある) ことがあり得ます。
- 翻訳者としては、元の文章に間違いを見つけた場合は必ず報告することです。翻訳者はしばしばその文章の最も注意深い読者であり、翻訳者が見つけた間違いを報告しないのならば誰も報告しないでしょう。
- いずれの場合でも、l10n に関する最も大きな問題は複数人の協調であり、誤解から小さな問題でフレームウォーを起こすのはとても簡単だということです。ですから、もし、あなたの話し相手と問題が起こっている場合は、関連する l10n メーリングリストや debian-i18n メーリングリスト、さらにあるいは debian-devel メーリングリストに助けを求めてください (ですが、ご注意を。l10n 関連の議論は debian-devel では頻繁にフレームウォーになります :)
- 何にせよ、協調は**互いを尊敬しあうこと**によってのみ成し得ます。

第9章　Debian メンテナツールの概要

この章には、メンテナが利用できるツールについて大まかな概要が含まれています。以下は完全なものでも決定版的なものでもありませんが、よく使われているツールについての説明です。

Debian メンテナツールは、開発者を手助けし、重要な作業のために時間を作れるようにしてくれるものです。Larry Wall が言うように、やり方は一つではありません (there's more than one way to do it)。

Some people prefer to use high-level package maintenance tools and some do not. Debian is officially agnostic on this issue; any tool that gets the job done is fine. Therefore, this section is not meant to stipulate to anyone which tools they should use or how they should go about their duties of maintainership. Nor is it meant to endorse any particular tool to the exclusion of a competing tool.

パッケージの説明文のほとんどは実際のパッケージの説明から取ったものです。より詳細な情報はパッケージ内のドキュメントで確認できます。`apt-cache show` **パッケージ名** コマンドでも情報を得られます。

9.1 主要なツール

以下のツールはどのメンテナであっても、必ず必要とするものです。

9.1.1 dpkg-dev

`dpkg-dev` は、パッケージを展開、ビルド、Debian ソースパッケージをアップロードするのに必要なツールを含んでいます (`dpkg-source` を含む)。これらのユーティリティはパッケージを作成・操作するのに必要な基礎的で、低レイヤの機能を含んでいます。そのため、これらはあらゆる Debian メンテナにとって必要不可欠なものです。

9.1.2 debconf

`debconf` は、パッケージを対話形式で設定できる一貫したインターフェイスを提供します。これはユーザインターフェイスに依存せず、エンドユーザがテキストのみのインターフェイス、HTML インターフェイス、ダイアログ形式のインターフェイスでパッケージを設定できます。新たなインターフェイスはモジュールとして追加できます。

このパッケージに関するドキュメントは `debconf-doc` パッケージ中で確認できます。

Many feel that this system should be used for all packages that require interactive configuration; see *debconf* **による設定管理**. `debconf` is not currently required by Debian Policy, but that may change in the future.

9.1.3 fakeroot

fakeroot は root 特権をシミュレートします。これは root になること無しにパッケージをビルドできるようにしてくれます (パッケージは通常 root の所有権でファイルをインストールしようとします)。fakeroot をインストールしていれば、dpkg-buildpackage で自動的に利用します。

9.2 パッケージチェック (lint) 用ツール

According to the Free On-line Dictionary of Computing (FOLDOC), lint is: "A Unix C language processor which carries out more thorough checks on the code than is usual with C compilers." Package lint tools help package maintainers by automatically finding common problems and policy violations in their packages.

9.2.1 lintian

lintian は Debian パッケージを解剖してバグやポリシー違反の情報を出力します。一般的なエラーへのチェック同様に Debian ポリシーの多くの部分を自動チェックする機能を含んでいます。

定期的に最新の lintian を unstable から取得し、パッケージを全てチェックするべきです。-i オプションは、各エラーや警告が何を意味しているのか、ポリシーを元に、詳細な説明を提供してくれ、一般的に問題をどのように修正するべきかを説明してくれることに留意してください。

何時、どのようにして Lintian を使うのか、詳細については **パッケージをテストする** を参照してください。

You can also see a summary of all problems reported by Lintian on your packages at https://udd.debian.org/lintian/. The reports contain the latest lintian output for the whole development distribution (unstable).

9.2.2 lintian-brush

lintian-brush contains a set of scripts that can automatically fix more than 80 common lintian issues in Debian packages.

It comes with a wrapper script that invokes the scripts, updates the changelog (if desired) and commits each change to version control.

9.2.3 piuparts

piuparts is the .deb package installation, upgrading, and removal testing tool.

piuparts tests that .deb packages handle installation, upgrading, and removal correctly. It does this by creating a minimal Debian installation in a chroot, and installing, upgrading, and removing packages in that environment, and comparing the state of the directory tree before and after. piuparts reports any files that have been added, removed, or modified during this process.

piuparts is meant as a quality assurance tool for people who create .deb packages to test them before they upload them to the Debian archive.

9.2.4 debdiff

(devscripts パッケージ、*devscripts* より) debdiff は二つのパッケージのファイルのリストと control ファイルを比較します。前回のアップロードからバイナリパッケージ数が変わったことや、control ファイル内で何が変わったのかなどに気付く手助けをしてくれるなど、簡単なリグレッションテストとなります。もちろん、報告される変更の多くは問題ありませんが、様々なアクシデントを防止するのに役立ってくれるでしょう。

バイナリパッケージのペアに対して実行することができます:

```
debdiff package_1-1_arch.deb package_2-1_arch.deb
```

changes ファイルのペアに対してさえも実行できます:

```
debdiff package_1-1_arch.changes package_2-1_arch.changes
```

より詳細については、debdiff 1 を参照してください。

9.2.5 diffoscope

diffoscope provides in-depth comparison of files, archives, and directories.

diffoscope will try to get to the bottom of what makes files or directories different. It will recursively unpack archives of many kinds and transform various binary formats into more human readable form to compare them.

Originally developed to compare two .deb files or two changes files nowadays it can compare two tarballs, ISO images, or PDF just as easily and supports a huge variety of filetypes.

The differences can be shown in a text or HTML report or as JSON output.

9.2.6 duck

duck, the Debian Url ChecKer, processes several fields in the debian/control, debian/upstream, debian/copyright, debian/patches/* and systemd.unit files and checks if URLs, VCS links and email address domains found therein are valid.

9.2.7 adequate

adequate checks packages installed on the system and reports policy violations.

The following checks are currently implemented:

- broken symlinks
- missing copyright file
- obsolete conffiles
- Python modules not byte-compiled
- missing libraries, undefined symbols, symbol size mismatches
- program name collisions

- missing alternatives
- missing `binfmt` interpreters and detectors
- missing `pkg-config` dependencies
- invalid user/group values in systemd/D-Bus/sysvinit files.

See /usr/share/doc/adequate/README on how to have `adequate` be ran automatically using `autopkgtest`.

9.2.8 i18nspector

`i18nspector` is a tool for checking translation templates (POT), message catalogues (PO) and compiled message catalogues (MO) files for common problems.

9.2.9 cme

`cme` is a configuration editor. With `libconfig-model-dpkg-perl` installed, it can edit and validate Debian source package files.

9.2.10 licensecheck

`licensecheck` attempts to determine the license that applies to each file passed to it, by searching the start of the file for text belonging to various licenses.

9.2.11 blhc

`blhc` is a tool which checks build logs for missing hardening flags.

9.3 debian/rules の補助ツール

パッケージ構築ツールは `debian/rules` ファイルを書く作業を楽にしてくれます。これらが望ましい、あるいは望ましくない理由の詳細については **ヘルパースクリプト** を参照してください。

9.3.1 debhelper

`debhelper` is a collection of programs that can be used in `debian/rules` to automate common tasks related to building binary Debian packages. `debhelper` includes programs to install various files into your package, compress files, fix file permissions, and integrate your package with the Debian menu system.

Unlike some approaches, `debhelper` is broken into several small, simple commands, which act in a consistent manner. As such, it allows more fine-grained control than some of the other debian/rules tools.

ここに記すには一時的な、大量の小さな `debhelper` のアドオンパッケージがあります。`apt-cache search ^dh-` と実行することで一覧の多くを参照できます。

When choosing a `debhelper` compatibility level for your package, you should choose the highest compatibility level that is supported in the most recent stable release. Only use a higher compatibility level if you need specific features that are provided by that compatibility level that are not available in earlier levels.

In the past the compatibility level was defined in debian/compat, however nowadays it is much better to not use that but rather to use a versioned build-dependency like debhelper-compat (=12).

9.3.2 dh-make

The dh-make package contains dh_make, a program that creates a skeleton of files necessary to build a Debian package out of a source tree. As the name suggests, dh_make is a rewrite of debmake, and its template files use dh_* programs from debhelper.

dh_make によって生成された rules ファイルは、大抵の場合作業するパッケージに対して十分な基礎にはなりますが、まだこれは下地でしかありません。メンテナに残っている責務は、生成されたファイルをきれいに整理して、完全に動作してポリシーに準拠したパッケージにすることです。

9.3.3 equivs

equivs はパッケージ作成用のもう一つのパッケージです。単純に依存関係を満たしたいだけのパッケージを作成する必要がある場合に、しばしばローカルでの使用を勧められます。時折、他のパッケージに依存することだけが目的のパッケージ、「メタパッケージ (meta-packages)」を作る際にも使われます。

9.4 パッケージ作成ツール

The following packages help with the package building process, general driving of dpkg-buildpackage, as well as handling supporting tasks.

9.4.1 git-buildpackage

git-buildpackage は、Debian ソースパッケージを Git リポジトリに挿入あるいはインポートし、Debian パッケージを Git リポジトリから生成、そして開発元での変更をリポジトリに統合するのに役立つ機能を提供します。

これらのユーティリティは、Debian メンテナによる Git の利用を促進するインフラストラクチャを提供します。これは、バージョンコントロールシステムの他の利点と同様に、stable、unstable、おそらく experimental ディストリビューション用にパッケージに個々の Git ブランチを持つことができます。

9.4.2 debootstrap

debootstrap パッケージとスクリプトは、システムのどこででも Debian ベースシステムをブートストラップできるようにしてくれます。ベースシステムとは、操作するのに必要となる素の最小限パッケージ群を意味し、それに加えてシステムの残りの部分をインストールします。

この様なシステムを持つことは、様々な面で役に立つでしょう。例えば、ビルドの依存関係をテストしたい場合に chroot でそのシステムの中に入ることができます。あるいは素のベースシステムにインストールした際にパッケージがどのように振る舞うかをテストできます。chroot 作成ツールはこのパッケージを使います。以下を参照ください。

9.4.3 pbuilder

pbuilder constructs a chrooted system, and builds a package inside the chroot. It is very useful to check that a package's build dependencies are correct, and to be sure that unnecessary and wrong build dependencies will not exist in the resulting package.

A related package is cowbuilder, which speeds up the build process using a COW filesystem on any standard Linux filesystem.

9.4.4 sbuild

sbuild はもう一つの自動ビルドシステムです。同様に chroot された環境を使うことが出来ます。単独で使うことも、分散ビルド環境のネットワークの一部として使うこともできます。文字通り、移植者たちによって利用可能な全アーキテクチャのバイナリパッケージをビルドするのに使われているシステムの一部です。詳細については *wanna-build* を参照してください。それからシステムの動作については https://buildd.debian.org/ を参照してください。

9.5 パッケージのアップロード用ツール

以下のパッケージはパッケージを公式アーカイブにアップロードする作業を自動化、あるいは単純化してくれるのに役立ちます。

9.5.1 dupload

dupload is a package and a script to automatically upload Debian packages to the Debian archive, to log the upload, and to optionally send mail about the upload of a package. It supports various kinds of hooks to extend its functionality, and can be configured for new upload locations or methods, although by default it provides various hooks performing checks and comes configured with all Debian upload locations.

9.5.2 dput

The dput package and script do much the same thing as dupload, but in a different way. Out of the box it supports to run dinstall in dry-run mode after the upload.

9.5.3 dcut

dcut スクリプト (dput パッケージの一部、*dput* 参照) は、ftp アップロードディレクトリからファイルを削除するのに役立ちます。

9.6 メンテナンスの自動化

以下のツールは changelog のエントリや署名行の追加、Emacs 内でのバグの参照から最新かつ公式の config.sub を使うようにするまで、様々なメンテナンス作業を自動化するのに役立ちます。

9.6.1 devscripts

devscripts is a package containing wrappers and tools that are very helpful for maintaining your Debian packages. Example scripts include debchange (or its alias, dch), which manipulates your debian/changelog file from the command-line, and debuild, which is a wrapper around dpkg-buildpackage. The bts utility is also very helpful to update the state of bug reports on the command line. uscan can be used to watch for new upstream versions of your packages (see https://wiki.debian.org/debian/watch for more info on that). suspicious-source outputs a list of files which are not common source files.

See the devscripts(7) manual page for a complete list of available scripts.

9.6.2 reportbug

reportbug is a tool designed to make the reporting of bugs in Debian and derived distributions relatively painless. Its features include:

- Integration with mutt and mh/nmh mail readers.
- Access to outstanding bug reports to make it easier to identify whether problems have already been reported.
- Automatic checking for newer versions of packages.

reportbug is designed to be used on systems with an installed mail transport agent; however, you can edit the configuration file and send reports using any available mail server.

This package also includes the querybts script for browsing the Debian bug tracking system.

9.6.3 autotools-dev

autotools-dev contains best practices for people who maintain packages that use autoconf and/or automake. Also contains canonical config.sub and config.guess files, which are known to work on all Debian ports.

9.6.4 dpkg-repack

dpkg-repack creates a Debian package file out of a package that has already been installed. If any changes have been made to the package while it was unpacked (e.g., files in /etc were modified), the new package will inherit the changes.

This utility can make it easy to copy packages from one computer to another, or to recreate packages that are installed on your system but no longer available elsewhere, or to save the current state of a package before you upgrade it.

9.6.5 alien

alien は、Debian、RPM (RedHat)、LSB (Linux Standard Base)、Solaris、Slackware などの各種バイナリパッケージのパッケージ形式を変換します。

9.6.6 elpa-dpkg-dev-el

elpa-dpkg-dev-el is an Emacs lisp package that provides assistance when editing some of the files in the debian directory of your package. For instance, there are handy functions for listing a package's current bugs, and for finalizing the latest entry in a debian/changelog file.

9.6.7 dpkg-depcheck

(devscripts パッケージ、*devscripts* より) dpkg-depcheck は、指定されたコマンドによって使われた全てのパッケージを確認するため、コマンドを strace の下で実行します。

Debian パッケージについていうと、これは新しいパッケージの Build-Depends 行を構成するのが必要になった際に役立ちます。dpkg-depcheck を通してビルド作業を実行すると、最初の大まかなビルドの依存関係を良い形で得られます。例えば以下の様にします:

```
dpkg-depcheck -b debian/rules build
```

dpkg-depcheck は、特にパッケージが他のプログラムを実行するのに exec 2 を使っている場合に実行時の依存性を確認するのにも使えます。

より詳細については、dpkg-depcheck 1 を参照してください。

9.6.8 debputy

The debputy tools is new since 2024. While its main purpose is to offer a new Debian package build paradigm, it includes subcommands that can be used on any existing Debian package to validate the correctness of most of the files in debian/*, and in many cases also automatically fix them.

To check correctness of files in debian/* run:

```
debputy lint --spellcheck
```

To format debian/control and debian/tests/control files

```
debputy reformat --style black
```

Using the reformat command obsoletes using wrap-and-sort -ast.

The debputy tool also includes a language server which, when integrated with a code editor, can give real-time feedback on the correctness of files in debian/* while editing them.

For more information please see debputy 1.

9.7 移植用ツール

以下のツールが、移植作業者やクロスコンパイル作業に役立ちます。

9.7.1 dpkg-cross

dpkg-cross は、dpkg に似た方法でクロスコンパイルするためのライブラリとヘッダをインストールするツールです。さらに、dpkg-buildpackage および dpkg-shlibdeps の機能がクロスコンパイルをサポートするように拡張されます。

9.8 ドキュメントと情報について

以下のパッケージが、メンテナへの情報提供やドキュメントの作成に役立ちます。

9.8.1 debian-policy

The debian-policy package contains the Debian Policy Manual and related documents, which are:

- Debian Policy Manual
- Filesystem Hierarchy Standard (FHS)
- Debian Menu sub-policy
- Debian Perl sub-policy
- Debian configuration management specification
- Machine-readable debian/copyright specification
- Autopkgtest - automatic as-installed package testing
- Authoritative list of virtual package names
- Policy checklist for upgrading your packages

The Debian Policy Manual the policy relating to packages and details of the packaging mechanism. It covers everything from required gcc options to the way the maintainer scripts (postinst etc.) work, package sections and priorities, etc.

Also useful is the file /usr/share/doc/debian-policy/upgrading-checklist.txt.gz, which lists changes between versions of policy.

9.8.2 doc-debian

doc-debian contains lots of useful Debian-specific documentation:

- Debian Linux Manifesto
- Constitution for the Debian Project
- Debian Social Contract
- Debian Free Software Guidelines
- Debian Bug Tracking System documentation
- Introduction to the Debian mailing lists

9.8.3 developers-reference

The developers-reference package contains the document you are reading right now, the Debian Developer's Reference, a set of guidelines and best practices which has been established by and for the community of Debian developers.

9.8.4 maint-guide

The maint-guide package contains the Debian New Maintainers' Guide.

This document tries to describe the building of a Debian package to ordinary Debian users and prospective developers. It uses fairly non-technical language, and it's well covered with working examples.

9.8.5 debmake-doc

The debmake-doc package contains the Guide for Debian Maintainers.

This document is newer than Debian New Maintainers' Guide and intends to replace it. The Guide for Debian Maintainers caters to those learning Debian packaging and covers a wide range of topics and tools, along with plenty of examples about various types of packaging issues.

9.8.6 packaging-tutorial

This tutorial is an introduction to Debian packaging. It teaches prospective developers how to modify existing packages, how to create their own packages, and how to interact with the Debian community.

In addition to the main tutorial, it includes three practical sessions on modifying the grep package, and packaging the gnujump game and a Java library.

9.8.7 how-can-i-help

how-can-i-help shows opportunities for contributing to Debian. how-can-i-help hooks into APT to list opportunities for contributions to Debian (orphaned packages, bugs tagged 'newcomer') for packages installed locally, after each APT invocation. It can also be invoked directly, and then lists all opportunities for contribution (not just the new ones).

9.8.8 docbook-xml

docbook-xml provides the DocBook XML DTDs, which are commonly used for Debian documentation (as is the older debiandoc SGML DTD).

docbook-xsl パッケージは、ソースをビルドして様々な出力フォーマットに整形する XSL ファイルを提供します。XSL スタイルシートを使うには xsltproc のような XSLT プロセッサが必要になります。スタイルシートのドキュメントは各種 docbook-xsl-doc-* パッケージで確認できます。

To produce PDF from FO, you need an FO processor, such fop or dblatex.

9.8.9 debiandoc-sgml

debiandoc-sgml は Debian のドキュメントで一般的に使われている DebianDoc SGML DTD を提供します。しかし、現在は非推奨 (deprecated) となっています (代わりに docbook-xml か python3-sphinx を使うようにしてください)。これも、ソースをビルドして様々な出力フォーマットに整形するスクリプトを提供します。

9.8.10 debian-keyring

Debian 開発者および Debian メンテナの公開 GPG 鍵を含んでいます。詳細については **公開鍵をメンテナンスする** とパッケージ内のドキュメントを参照してください。

9.8.11 elpa-debian-el

elpa-debian-el provides an Emacs mode for viewing Debian binary packages. This lets you examine a package without unpacking it.