NOTE: This is a version of Documentation/process/howto.rst translated into Japanese. This document is maintained by Tsugikazu Shibata <tshibata@ab.jp.nec.com> If you find any difference between this document and the original file or a problem with the translation, please contact the maintainer of this file.

Please also note that the purpose of this file is to be easier to read for non English (read: Japanese) speakers and is not intended as a fork. So if you have any comments or updates for this file, please try to update the original English file first.

この文書は、Documentation/process/howto.rst の和訳です。

翻訳者： Tsugikazu Shibata <tshibata@ab.jp.nec.com>

Linux カーネル開発のやり方

これは上のトピック(Linux カーネル開発のやり方)の重要な事柄を網羅した ドキュメントです。ここには Linux カーネル開発者になるための方法とLinux カーネル開発コミュニティと共に活動するやり方を学ぶ方法が含まれています。カーネルプログラミングに関する技術的な項目に関することは何も含めないよ うにしていますが、カーネル開発者となるための正しい方向に向かう手助けに なります。

もし、このドキュメントのどこかが古くなっていた場合には、このドキュメント の最後にリストしたメンテナにパッチを送ってください。

はじめに

あなたは Linux カーネルの開発者になる方法を学びたいのでしょうか？　そ れとも上司から「このデバイスの Linux ドライバを書くように」と言われた のかもしれません。この文書の目的は、あなたが踏むべき手順と、コミュニティと一緒にうまく働くヒントを書き下すことで、あなたが知るべき全てのことを 教えることです。また、このコミュニティがなぜ今うまくまわっているのかと いう理由も説明しようと試みています。

カーネルは少量のアーキテクチャ依存部分がアセンブリ言語で書かれている以 外の大部分は C 言語で書かれています。C言語をよく理解していることはカー ネル開発に必要です。低レベルのアーキテクチャ開発をするのでなければ、(どんなアーキテクチャでも)アセンブリ(訳注: 言語)は必要ありません。以下 の本は、C 言語の十分な知識や何年もの経験に取って代わるものではありませ んが、少なくともリファレンスとしては良い本です。

- "The C Programming Language" by Kernighan and Ritchie [Prentice Hall]
- 『プログラミング言語C第2版』(B.W. カーニハン/D.M. リッチー著 石田晴久訳) [共立出版]
- "Practical C Programming" by Steve Oualline [O'Reilly]
- 『C実践プログラミング第3版』(Steve Oualline著 望月康司監訳 谷口功訳) [オライリージャパン]
- "C: A Reference Manual" by Harbison and Steele [Prentice Hall]
- 『新・詳説 C 言語 H&S リファレンス』(サミュエル P ハービソン/ガイ L スティール共著 斉藤 信男監訳)[ソフトバンク]

カーネルは GNU C と GNU ツールチェインを使って書かれています。カーネル は ISO C89 仕様に準拠して書く一方で、標準には無い言語拡張を多く使って います。カーネルは標準 C ライブラリに依存しない、C 言語非依存環境です。 そのため、C の標準の中で使えないものもあります。特に任意の long long の除算や浮動小数点は使えません。カーネルがツールチェインや C 言語拡張に置いている前提がどうなっているのかわかりにくいことが時々あり、また、 残念なことに決定的なリファレンスは存在しません。情報を得るには、gcc の info ページ(info gcc)を見てください。

あなたは既存の開発コミュニティと一緒に作業する方法を学ぼうとしているこ とに思い出してください。そのコミュニティは、コーディング、スタイル、開 発手順について高度な標準を持つ、多様な人の集まりです。地理的に分散した 大規模なチームに対してもっともうまくいくとわかったことをベースにしなが ら、これらの標準は長い時間をかけて築かれてきました。これらはきちんと文 書化されていますから、事前にこれらの標準について事前にできるだけたくさ ん学んでください。また皆があなたやあなたの会社のやり方に合わせてくれる と思わないでください。

法的問題

Linux カーネルのソースコードは GPL ライセンスの下でリリースされていま す。ライセンスの詳細については、ソースツリーのメインディレクトリに存在 する、COPYING のファイルを見てください。もしライセンスについてさらに質 問があれば、Linux Kernel メーリングリストに質問するのではなく、どうぞ 法律家に相談してください。メーリングリストの人達は法律家ではなく、法的 問題については彼らの声明はあてにするべきではありません。

GPL に関する共通の質問や回答については、以下を参照してください-

https://www.gnu.org/licenses/gpl-faq.html

ドキュメント

Linux カーネルソースツリーは幅広い範囲のドキュメントを含んでおり、それ らはカーネルコミュニティと会話する方法を学ぶのに非常に貴重なものです。 新しい機能がカーネルに追加される場合、その機能の使い方について説明した 新しいドキュメントファイルも追加することを勧めます。カーネルの変更が、カーネルがユーザ空間に公開しているインターフェイスの 変更を引き起こす場合、その変更を説明するマニュアルページのパッチや情報 をマニュアルページのメンテナ mtk.manpages@gmail.com に送り、CC を linux-api@vger.kernel.org に送ることを勧めます。

以下はカーネルソースツリーに含まれている読んでおくべきファイルの一覧です-

README

このファイルは Linuxカーネルの簡単な背景とカーネルを設定(訳注 configure)し、生成(訳注 build)するために必要なことは何かが書かれています。カーネルに関して初めての人はここからスタートすると良いでしょう。

:ref:`Documentation/process/changes.rst <changes>`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\translations\ja_JP\(linux-master)(Documentation)(translations)(ja_JP)howto.rst, line 123); backlink

Unknown interpreted text role "ref".

このファイルはカーネルをうまく生成(訳注 build)し、走らせるのに最小限のレベルで必要な数々のソフトウェアパッケージの一覧を示しています。

:ref:`Documentation/process/coding-style.rst <codingstyle>`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\translations\ja_JP\(linux-master)(Documentation)(translations)(ja_JP)howto.rst, line 130); backlink

Unknown interpreted text role "ref".

これは Linux カーネルのコーディングスタイルと背景にある理由を記述しています。全ての新しいコードはこのドキュメントにあるガイドラインに従っていることを期待されています。大部分のメンテナはこれらのルールに従っているものだけを受け付け、多くの人は正しいスタイルのコードだけをレビューします。

:ref:`Documentation/process/submitting-patches.rst <codingstyle>` と :ref:`Documentation/process/submitting-drivers.rst <submittingdrivers>`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\translations\ja_JP\(linux-master)(Documentation)(translations)(ja_JP)howto.rst, line 150); backlink

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\translations\ja_JP\(linux-master)(Documentation)(translations)(ja_JP)howto.rst, line 150); backlink

Unknown interpreted text role "ref".

これらのファイルには、どうやってうまくパッチを作って投稿するかについて非常に詳しく書かれており、以下を含みます (これだけに限らないけれども)

- Email に含むこと
- Email の形式
- だれに送るか

これらのルールに従えばうまくいくことを保証することではありませんが (すべてのパッチは内容とスタイルについて精査を受けるので)、ルールに従わなければ間違いなくうまくいかないでしょう。

この他にパッチを作る方法についてのよくできた記述は-

"The Perfect Patch"
http://www.ozlabs.org/~akpm/stuff/tpp.txt
"Linux kernel patch submission format"
https://web.archive.org/web/20180829112450/http://linux.yyz.us/patch-format.html

:ref:`Documentation/process/stable-api-nonsense.rst <stable_api_nonsense>`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\translations\ja_JP\(linux-master)(Documentation)(translations)(ja_JP)howto.rst, line 162); backlink

Unknown interpreted text role "ref".

このファイルはカーネルの中に不変の API を持たないことにした意識的 な決断の背景にある理由について書かれています。以下のようなことを含 んでいます-

- サブシステムとの間に層を作ること(コンパチビリティのため?)
- オペレーティングシステム間のドライバの移植性
- カーネルソースツリーの素早い変更を遅らせる(もしくは素早い変更を妨げる)

このドキュメントは Linux 開発の思想を理解するのに非常に重要です。そして、他のOSでの開発者が Linux に移る時にとても重要です。

:ref:`Documentation/admin-guide/security-bugs.rst <securitybugs>`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\translations\ja_JP\(linux-master)(Documentation)(translations)(ja_JP)howto.rst, line 167); *backlink*

Unknown interpreted text role "ref".

もし Linux カーネルでセキュリティ問題を発見したように思ったら、こ のドキュメントのステップに従ってカーネル開発者に連絡し、問題解決を 支援してください。

:ref:`Documentation/process/management-style.rst <managementstyle>`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\translations\ja_JP\(linux-master)(Documentation)(translations)(ja_JP)howto.rst, line 174); *backlink*

Unknown interpreted text role "ref".

このドキュメントは Linux カーネルのメンテナ達がどう行動するか、彼らの手法の背景にある共有されている精神について記述しています。こ れはカーネル開発の初心者なら(もしくは、単に興味があるだけの人でも) 重要です。なぜならこのドキュメントは、カーネルメンテナ達の独特な 行動についての多くの誤解や混乱を解消するからです。

:ref:`Documentation/process/stable-kernel-rules.rst <stable_kernel_rules>`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\translations\ja_JP\(linux-master)(Documentation)(translations)(ja_JP)howto.rst, line 179); *backlink*

Unknown interpreted text role "ref".

このファイルはどのように stable カーネルのリリースが行われるかのルー ルが記述されています。そしてこれらのリリースの中のどこかで変更を取 り入れてもらいたい場合に何をすれば良いかが示されています。

:Ref:`Documentation/process/kernel-docs.rst <kernel_docs>`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\translations\ja_JP\(linux-master)(Documentation)(translations)(ja_JP)howto.rst, line 184); *backlink*

Unknown interpreted text role "Ref".

カーネル開発に付随する外部ドキュメントのリストです。もしあなたが探 しているものがカーネル内のドキュメントでみつからなかった場合、この リストをあたってみてください。

:ref:`Documentation/process/applying-patches.rst <applying_patches>`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\translations\ja_JP\(linux-master)(Documentation)(translations)(ja_JP)howto.rst, line 188); *backlink*

Unknown interpreted text role "ref".

パッチとはなにか、パッチをどうやって様々なカーネルの開発ブランチに 適用するのかについて正確に記述した良い入門書です。

カーネルはソースコードそのものや、このファイルのようなリストラクチャー ドテキストマークアップ(ReST)から自動的に生成可能な多数のドキュメントを もっています。これにはカーネル内APIの完全な記述や、正しくロックをかけ るための規則などが含まれます。

これら全てのドキュメントを PDF や HTML で生成するには以下を実行します -

```
make pdfdocs
make htmldocs
```

それぞれメインカーネルのソースディレクトリから実行します。

ReSTマークアップを使ったドキュメントは Documentation/outputに生成され ます。Latex とePub 形式で生成するには -

```
make latexdocs
make epubdocs
```

カーネル開発者になるには

もしあなたが、Linux カーネル開発について何も知らないのならば、KernelNewbies プロジェクトを見るべきです

https://kernelnewbies.org

このサイトには役に立つメーリングリストがあり、基本的なカーネル開発に関 するほとんどどんな種類の質問もできます (既に回答されているようなことを 聞く前にまずはアーカイブを調べてください)。またここには、リアルタイム で質問を聞くことができる IRC チャネルや、Linuxカーネルの開発に関して学 ぶのに便利なたくさんの役に立つドキュメントがあります。

Web サイトには、コードの構成、サブシステム、現在存在するプロジェクト (ツリーにあるもの無いものの両方)の基本的な管理情報があります。ここには、また、カーネルのコンパイルのやり方やパッチの当て方などの間接的な基本情 報も記述されています。

あなたがどこからスタートして良いかわからないが、Linux カーネル開発コミュ ニティに参加して何かすることをさがしているのであれば、Linux kernel Janitor's プロジェクトにいけば良いでしょう -

https://kernelnewbies.org/KernelJanitors

ここはそのようなスタートをするのにうってつけの場所です。ここには、Linux カーネルソースツリーの中に含まれる、きれいにし、修正しなければな らない、単純な問題のリストが記述されています。このプロジェクトに関わる 開発者と一緒に作業することで、あなたのパッチを Linuxカーネルツリーに入 れるための基礎を学ぶことができ、そしてもしあなたがまだアイディアを持っ ていない場合には、次にやる仕事の方向性が見えてくるかもしれません。

もしあなたが、すでにひとまとまりコードを書いていて、カーネルツリーに入 れたいと思っていたり、それに関する適切な支援を求めたい場合、カーネルメ ンターズプロジェクトはそのような皆さんを助けるためにできました。ここに はメーリングリストがあり、以下から参照できます -

https://selenic.com/mailman/listinfo/kernel-mentors

実際に Linux カーネルのコードについて修正を加える前に、どうやってその コードが動作するのかを理解することが必要です。そのためには、特別なツー ルの助けを借りてでも、それを直接よく読むことが最良の方法です(ほとんど のトリッキーな部分は十分にコメントしてありますから)。そういうツールで 特におすすめなのは、Linux クロスリファレンスプロジェクトです。これは、自己参照方式で、索引がついた web 形式で、ソースコードを参照することが できます。この最新の素晴しいカーネルコードのリポジトリは以下で見つかります -

https://elixir.bootlin.com/

開発プロセス

Linux カーネルの開発プロセスは現在幾つかの異なるメインカーネル「ブラン チ」と多数のサブシステム毎のカーネルブランチから構成されます。これらの ブランチとは -

- メインの 4.x カーネルツリー
- 4.x.y -stable カーネルツリー
- サブシステム毎のカーネルツリーとパッチ
- 統合テストのための 4.x -next カーネルツリー

4.x カーネルツリー

4.x カーネルは Linus Torvalds によってメンテナンスされ、https://kernel.org の pub/linux/kernel/v4.x/ ディレクトリに存在します。この開発プロセスは以下のとおり -

- 新しいカーネルがリリースされた直後に、2週間の特別期間が設けられ、この期間中に、メンテナ達は Linus に大きな差分を送ることができます。このような差分は通常 -next カーネルに数週間含まれてきたパッチです。大きな変更は git(カーネルのソース管理ツール、詳細は http://git-scm.com/ 参照) を使って送るのが好ましいやり方ですが、パッ チファイルの形式のまま送るのでも十分です。
- 2週間後、-rc1 カーネルがリリースされ、この後にはカーネル全体の安定 性に影響をあたえるような新機能は含まない類のパッチしか取り込むこと はできません。新しいドライバ(もしくはファイルシステム)のパッチは -rc1 の後で受け付けられることもあることを覚えておいてください。な ぜなら、変更が独立していて、追加されたコードの外の領域に影響を与え ない限り、退行のリスクは無いからです。-rc1 がリリースされた後、Linus へパッチを送付するのに git を

使うこともできますが、パッチは レビューのために、パブリックなメーリングリストへも同時に送る必要が あります。
- 新しい -rc は Linus が、最新の git ツリーがテスト目的であれば十分 に安定した状態にあると判断したときにリリースされます。目標は毎週新 しい -rc カーネルをリリースすることです。
- このプロセスはカーネルが「準備ができた」と考えられるまで継続しま す。このプロセスはだいたい 6週間継続します。

Andrew Morton が Linux-kernel メーリングリストにカーネルリリースについ て書いたことをここで言っておくことは価値があります -

> *「カーネルがいつリリースされるかは誰も知りません。なぜなら、これは現実に認識されたバグの状況によりリリースされるのであり、前もって決められた計画によってリリースされるものではないから です。」*

4.x.y -stable カーネルツリー

バージョン番号が3つの数字に分かれているカーネルは -stable カーネルです。これには、4.x カーネルで見つかったセキュリティ問題や重大な後戻りに対す る比較的小さい重要な修正が含まれます。

これは、開発/実験的バージョンのテストに協力することに興味が無く、最新 の安定したカーネルを使いたいユーザに推奨するブランチです。

もし、4.x.y カーネルが存在しない場合には、番号が一番大きい 4.x が最新 の安定版カーネルです。

4.x.y は "stable" チーム <stable@vger.kernel.org> でメンテされており、必要に応じてリリースされます。通常のリリース期間は 2週間毎ですが、差 し迫った問題がなければもう少し長くなることもあります。セキュリティ関 連の問題の場合はこれに対してだいたいの場合、すぐにリリースがされます。

カーネルツリーに入っている、Documentation/process/stable-kernel-rules.rst ファイルにはどのような種 類の変更が -stable ツリーに受け入れ可能か、またリリースプロセスがどう 動くかが記述されています。

サブシステム毎のカーネルツリーとパッチ

それぞれのカーネルサブシステムのメンテナ達は --- そして多くのカーネル サブシステムの開発者達も --- 各自の最新の開発状況をソースリポジトリに 公開しています。そのため、自分とは異なる領域のカーネルで何が起きている かを他の人が見られるようになっています。開発が早く進んでいる領域では、 開発者は自身の投稿がどのサブシステムカーネルツリーを元にしているか質問 されるので、その投稿とすでに進行中の他の作業との衝突が避けられます。

大部分のこれらのリポジトリは git ツリーです。しかしその他の SCM や quilt シリーズとして公開されているパッチキューも使われています。これら のサブシステムリポジトリのアドレスは MAINTAINERS ファイルにリストされ ています。これらの多くは https://git.kernel.org/ で参照することができま す。

提案されたパッチがこのようなサブシステムツリーにコミットされる前に、メー リングリストで事前にレビューにかけられます（以下の対応するセクションを 参照）。いくつかのカーネルサブシステムでは、このレビューは patchworkと いうツールによって追跡されます。Patchwork は web インターフェイスによっ てパッチ投稿の表示、パッチへのコメント付けや改訂などができ、そしてメン テナはパッチに対して、レビュー中、受付済み、拒否というようなマークをつ けることができます。大部分のこれらの patchwork のサイトは https://patchwork.kernel.org/ でリストされています。

統合テストのための 4.x -next カーネルツリー

サブシステムツリーの更新内容がメインラインの 4.x ツリーにマージされる 前に、それらは統合テストされる必要があります。この目的のため、実質的に 全サブシステムツリーからほぼ毎日プルされてできる特別なテスト用のリポジ トリが存在します-

> https://git.kernel.org/?p=linux/kernel/git/next/linux-next.git

このやり方によって、-next カーネルは次のマージ機会でどんなものがメイン ラインカーネルにマージされるか、おおまかなの展望を提供します。-next カー ネルの実行テストを行う冒険好きなテスターは大いに歓迎されます。

バグレポート

https://bugzilla.kernel.org は Linux カーネル開発者がカーネルのバグを追跡する 場所です。ユーザは見つけたバグの全てをこのツールで報告すべきです。どう kernel bugzilla を使うかの詳細は、以下を参照してください -

> https://bugzilla.kernel.org/page.cgi?id=faq.html

メインカーネルソースディレクトリにあるファイル admin-guide/reporting-bugs.rstはカーネルバグらしいものについてどうレポー トするかの良いテンプレートであり、問題の追跡を助けるためにカーネル開発 者にとってどんな情報が必要なのかの詳細が書かれています。

バグレポートの管理

あなたのハッキングのスキルを訓練する最高の方法のひとつに、他人がレポー トしたバグを修正することがあります。あなたがカーネルをより安定化させる こに寄与するということだけでなく、あなたは 現実の問題を修正することを 学び、自分のスキルも強化でき、また他の開発者があなたの存在に気がつきま す。バグを修正することは、多くの開発者の中から自分が功績をあげる最善の 道です、なぜなら多くの人は他人のバグの修正に時間を浪費することを好まな いからです。

すでにレポートされたバグのために仕事をするためには、https://bugzilla.kernel.org に行ってください。もし今後のバグレポートに つい

てアドバイスを受けたいのであれば、bugme-new メーリングリスト(新し いバグレポートだけがここにメールされる) または bugme-janitor メーリン グリスト(bugzilla の変更毎にここにメールされる)を購読できます。

https://lists.linux-foundation.org/mailman/listinfo/bugme-new

https://lists.linux-foundation.org/mailman/listinfo/bugme-janitors

メーリングリスト

上のいくつかのドキュメントで述べていますが、コアカーネル開発者の大部分 は Linux kernel メーリングリストに参加しています。このリストの登録/脱 退の方法については以下を参照してください-

http://vger.kernel.org/vger-lists.html#linux-kernel

このメーリングリストのアーカイブは web 上の多数の場所に存在します。こ れらのアーカイブを探すにはサーチエンジンを使いましょう。例えば-

http://dir.gmane.org/gmane.linux.kernel

リストに投稿する前にすでにその話題がアーカイブに存在するかどうかを検索 することを是非やってください。多数の事がすでに詳細に渡って議論されてお り、アーカイブにのみ記録されています。

大部分のカーネルサブシステムも自分の個別の開発を実施するメーリングリス トを持っています。個々のグループがどんなリストを持っているかは、MAINTAINERS ファイルにリストがありますので参照してください。

多くのリストは kernel.org でホストされています。これらの情報は以下にあ ります -

http://vger.kernel.org/vger-lists.html

メーリングリストを使う場合、良い行動習慣に従うようにしましょう。少し安っ ぽいが、以下の URL は上のリスト(や他のリスト)で会話する場合のシンプル なガイドラインを示しています -

http://www.albion.com/netiquette/

もし複数の人があなたのメールに返事をした場合、CC: で受ける人のリストは だいぶ多くなるでしょう。正当な理由がない限り、CC: リストから誰かを削除 をしないように、また、メーリングリストのアドレスだけにリプライすること のないようにしましょう。1つは送信者から、もう1つはリストからのように、メールを2回受けることになってもそれに慣れ、しゃれたメールヘッダーを追 加してこの状態を変えようとしないように。人々はそのようなことは好みませ ん。

今までのメールでのやりとりとその間のあなたの発言はそのまま残し、"John Kernelhacker wrote ...:" の行をあなたのリプライの先頭行にして、メールの先頭でなく、各引用行の間にあなたの言いたいことを追加するべきで す。

もしパッチをメールに付ける場合は、Documentation/process/submitting-patches.rst に提示されているように、そ れは プレーンな可読テキストにすることを忘れないようにしましょう。カー ネル開発者は 添付や圧縮したパッチを扱いたがりません。彼らはあなたのパッチの行毎にコメントを入れたいので、そうするしかありません。あなたのメー ルプログラムが空白やタブを圧縮しないように確認しましょう。最初の良いテ ストとしては、自分にメールを送ってみて、そのパッチを自分で当ててみることです。もしそれがうまく行かないなら、あなたのメールプログラムを直して もらうか、正しく動くように変えるべきです。

何をおいても、他の購読者に対する敬意を表すことを忘れないでください。

コミュニティと共に働くこと

カーネルコミュニティのゴールは可能なかぎり最高のカーネルを提供すること です。あなたがパッチを受け入れてもらうために投稿した場合、それは、技術 的メリットだけがレビューされます。その際、あなたは何を予想すべきでしょ うか?

- 批判
- コメント
- 変更の要求
- パッチの正当性の証明要求
- 沈黙

思い出してください、これはあなたのパッチをカーネルに入れる話です。あな たは、あなたのパッチに対する批判とコメントを受け入れるべきで、それらを 技術的レベルで評価して、パッチを再作成するか、なぜそれらの変更をすべき でないかを明確で簡潔な理由の説明を提供してください。もし、あなたのパッ チに何も反応がない場合、たまにはメールの山に埋もれて見逃され、あなたの 投稿が忘れられてしまうこともあるので、数日待って再度投稿してください。

あなたがやるべきでないことは?

- 質問なしにあなたのパッチが受け入れられると想像すること
- 守りに入ること
- コメントを無視すること
- 要求された変更を何もしないでパッチを出し直すこと

可能な限り最高の技術的解決を求めているコミュニティでは、パッチがどのくらい有益なのかについては常に異なる意見があります。あなたは協調的である べきですし、また、あなたのアイディアをカーネルに対してうまく合わせるようにすることが望まれています。もしくは、最低限あなたのアイディアがそれ だけの価値があるとすすんで証明するようにしなければなりません。正しい解決に向かって進もうという意志がある限り、間違うことがあっても許 容されることを忘れないでください。

あなたの最初のパッチに単に 1ダースもの修正を求めるリストの返答になることも普通のことです。これはあなたのパッチが受け入れられないということで は **ありません**、そしてあなた自身に反対することを意味するのでも **あ りません**。単に自分のパッチに対して指摘された問題を全て修正して再送す れば良いのです。

カーネルコミュニティと企業組織のちがい

カーネルコミュニティは大部分の伝統的な会社の開発環境とは異ったやり方で 動いています。以下は問題を避けるためにできると良いことのリストです。

あなたの提案する変更について言うときのうまい言い方 -

- "これは複数の問題を解決します"
- "これは2000行のコードを削除します"
- "以下のパッチは、私が言おうとしていることを説明するものです"
- "私はこれを5つの異なるアーキテクチャでテストしたのですが..."
- "以下は一連の小さなパッチ群ですが..."
- "これは典型的なマシンでの性能を向上させます..."

やめた方が良い悪い言い方 -

- "このやり方で AIX/ptx/Solaris ではできたので、できるはずだ..."
- "私はこれを20年もの間やってきた、だから..."
- "これは私の会社が金儲けをするために必要だ"
- "これは我々のエンタープライズ向け商品ラインのためである"
- "これは私が自分のアイディアを記述した、1000ページの設計資料である"
- "私はこれについて、6ケ月作業している..."
- "以下は ... に関する5000行のパッチです"
- "私は現在のぐちゃぐちゃを全部書き直した、それが以下です..."
- "私は〆切がある、そのためこのパッチは今すぐ適用される必要がある"

カーネルコミュニティが大部分の伝統的なソフトウェアエンジニアリングの労 働環境と異なるもう一つの点は、やりとりに顔を合わせないということです。 email と irc を第一のコミュニケーションの形とする一つの利点は、性別や 民族の差別がないことです。Linux カーネルの職場環境は女性や少数民族を受 容します。なぜなら、email アドレスによってのみあなたが認識されるからで す。 国際的な側面からも活動領域を均等にするようにします。なぜならば、あなた は人の名前で性別を想像できないからです。ある男性が アンドレアという名 前で、女性の名前は パット かもしれません (訳注 Andrea は米国では女性、それ以外(欧州など)では男性名として使われることが多い。同様に、Pat は Patricia (主に女性名)や Patrick (主に男性名)の略称)。Linux カーネルの活動をして、意見を表明したことがある大部分の女性は、前 向きな経験をもっています。

言葉の壁は英語が得意でない一部の人には問題になります。メーリングリスト の中で、きちんとアイディアを交換するには、相当うまく英語を操れる必要が あることもあります。そのため、自分のメールを送る前に英語で意味が通じて いるかをチェックすることをお薦めします。

変更を分割する

Linux カーネルコミュニティは、一度に大量のコードの塊を喜んで受容することはありません。変更は正確に説明される必要があり、議論され、小さい、個 別の部分に分割する必要があります。これはこれまで多くの会社がやり慣れて きたことと全く正反対のことです。あなたのプロポーザルは、開発プロセスのと ても早い段階から紹介されるべきです。そうすれば あなたは自分のやっていることにフィードバックを得られます。これは、コミュニティからみれば、あ なたが彼らと一緒にやっているように感じられ、単にあなたの提案する機能の ゴミ捨て場として使っているのではない、と感じられるでしょう。しかし、一度に 50 もの email をメーリングリストに送りつけるようなことは やってはいけません、あなたのパッチ群はいつもどんな時でもそれよりは小さくなければなりません。

パッチを分割する理由は以下 -

1. 小さいパッチはあなたのパッチが適用される見込みを大きくします、カー ネルの人達はパッチが正しいかどうかを確認する時間や労力をかけないか らです。5行のパッチはメンテナがたった1秒見るだけで適用できます。しかし、500行のパッチは、正しいことをレビューするのに数時間かかるか もしれません(時間はパッチのサイズなどにより指数関数に比例してかかります)

   小さいパッチは何かあったときにデバッグもとても簡単になります。パッ チを1個1個取り除くのは、とても大きなパッチを当てた後に(かつ、何かお かしくなった後で)解剖するのに比べればとても簡単です。

2. 小さいパッチを送るだけでなく、送るまえに、書き直して、シンプルにす る(もしくは、単に順番を変えるだけでも)ことも、とても重要です。

以下はカーネル開発者の Al Viro のたとえ話です -

"生徒の数学の宿題を採点する先生のことを考えてみてください、先生は生徒が解に到達するまでの試行錯誤を見た

いとは思わないでしょう。先生は簡潔な最高の解を見たいのです。良い生徒はこれを知っており、そして最終解の前の中間作業を提出することは決してないの です

カーネル開発でもこれは同じです。メンテナ達とレビューア達は、問題を解決する解の背後になる思考プロセスを見たいとは思いません。彼らは単純であざやかな解決方法を見たいのです。"

あざやかな解を説明するのと、コミュニティと共に仕事をし、未解決の仕事を 議論することのバランスをキープするのは難しいかもしれません。ですから、開発プロセスの早期段階で改善のためのフィードバックをもらうようにするの も良いですが、変更点を小さい部分に分割して全体ではまだ完成していない仕 事を(部分的に)取り込んでもらえるようにすることも良いことです。

また、でき上がっていないものや、"将来直す" ようなパッチを、本流に含め てもらうように送っても、それは受け付けられないことを理解してください。

あなたの変更を正当化する

あなたのパッチを分割するのと同時に、なぜその変更を追加しなければならな いかを Linux コミュニティに知らせることはとても重要です。新機能は必要 性と有用性で正当化されなければなりません。

あなたの変更を説明する

あなたのパッチを送付する場合には、メールの中のテキストで何を言うかにつ いて、特別に注意を払ってください。この情報はパッチの ChangeLog に使わ れ、いつも皆がみられるように保管されます。これは次のような項目を含め、パッチを完全に記述するべきです -

- なぜ変更が必要か
- パッチ全体の設計アプローチ
- 実装の詳細
- テスト結果

これについて全てがどのようにあるべきかについての詳細は、以下のドキュメントの ChangeLog セクションを見てください -

"The Perfect Patch"
http://www.ozlabs.org/~akpm/stuff/tpp.txt

これらはどれも、実行することが時にはとても困難です。これらの例を完璧に 実施するには数年かかるかもしれません。これは継続的な改善のプロセスであり、多くの忍耐と決意を必要とするものです。でも諦めないで、実現は可能で す。多数の人がすでにできていますし、彼らも最初はあなたと同じところから スタートしたのですから。

Paolo Ciarrocchi に感謝、彼は彼の書いた "Development Process" (https://lwn.net/Articles/94386/) セクションをこのテキストの原型にする ことを許可してくれました。Rundy Dunlap と Gerrit Huizenga はメーリング リストでやるべきこととやってはいけないことのリストを提供してくれました。以下の人々のレビュー、コメント、貢献に感謝。Pat Mochel, Hanna Linder, Randy Dunlap, Kay Sievers, Vojtech Pavlik, Jan Kara, Josh Boyer, Kees Cook, Andrew Morton, Andi Kleen, Vadim Lobanov, Jesper Juhl, Adrian Bunk, Keri Harris, Frans Pop, David A. Wheeler, Junio Hamano, Michael Kerrisk, と Alex Shepard 彼らの支援なしでは、このドキュメントはできなかったでしょう。

Maintainer: Greg Kroah-Hartman <greg@kroah.com>