Gitによるバージョン管理入門

田中 健策（株式会社ぺあのしすてむ）今日が誕生日！

第四回

あらすじ、方針、反響

前回までのあらすじ

- master ブランチに push すると LaTeX を分割した LaTeX ファイルをまとめて自動コンパイルするようにしようとした

今後の方針

- せっかく Github を使ってるのだから、pull request を使ってみよう

この講義のことを Twitter で呟いたところ、500RT 程されて好意的な声がたくさん寄せられたが、「git は難しい。svn は何も考えなくてもよかった」という意見もあった。
実際 git は git 以前のバージョン管理システムより複雑な部分がある。
それは git 以前のバージョン管理システムは「**中央管理的**」で git は「**分散型**」だからだ。

確認：分散型の長所と短所

長所

- インターネットが繋がっていなくてもローカルだけで変更のコミットができる
- ローカルだけでログ確認や過去の状態への巻き戻しができる
- たとえリモートリポジトリが消失しても、ローカルにも履歴があるので大部分の再現できる（単一障害点がない）
- 複数のリモートリポジトリを持つこともできる
- たくさんの人が自由に開発に参加しやすい

短所

- コンフリクトが起こりやすく、ブランチを細かく分ける必要がある
- 今のところ、まだまだワークフローが複雑である

そのためにいくつかのベストプラクティクスに「○○ flow」という名前がついている。

フローとは

git での開発におけるワークフローのベストプラクティクスに名前をつけたもの。
様々なものが開発されている。
有名なものに **git flow**, **github flow**, **gitlab flow** などがある。
git flow は Github 等の高機能なホスティングサービスを使わず、git だけで大規模開発するために作られたフローでそれを支援するツールも存在する。
しかし、その分非常に複雑になっている。
現在では Github 等のホスティングサービスを使わずに大規模開発することは滅多にない。もし外部のホスティングサービスを使いたくなければ、Gitlab を自分で立てるべき。

フローの使い方

実際には様々なフローを参考にして、自分たちのプロジェクトごとに調整してベストプラクティクスを探していくことになる。
プロジェクトの規模によって、使い勝手は変わる。
必要もないのに複雑すぎると、苦痛になり、結局雑な管理になる。
またソフトウェア以外の執筆に使う場合も、考え方は参考になるが、そのままでは複雑すぎるフローもある。
将来的にはローカルのツール（IDE や TeX の編集ツール）とリモートのツール（Github などのホスト）がこのようなフローを自動化する必要もあるだろう（そもそも自動化のためのツールなのに、手でやっていては本末転倒である）。
全てのフローに共通する重要なアイディアは**トピックブランチ**を分けること、である。

トピックブランチとは

大雑把に「そこだけ独立して書ける部分」で分けると考えるとわかりやすい。
ソフトウェアなら「一つの機能の追加」、複数人で書いてるドキュメントなら「自分の書いてる章」でブランチを分ける。
その際ブランチに「説明的な名前」をつけることが求められる。
トピックブランチの目的は、書きかけの物は master から分離しておき、master をいつも「綺麗」にしておくとともに、そのブランチでは他のブランチを一切気にせず執筆ができるようにする。
「綺麗」とは、大雑把に下記を意味する。

- ソフトウェアなら「リリース可能」
- ドキュメントなら「全体として読むことが可能」

トピックブランチを別のブランチへマージするときは通常レビューを通過する必要がある。

github flow

github flow とは git flow を Github を使うことを前提に簡略化したものである。

- master ブランチはいつでもリリースできる状態になってる
- 新しい何かをするときは master ブランチからトピックブランチを作る
- リモートの同名のブランチに変更を push する
- フィードバックや助言が欲しいときや、ブランチをマージしたいと思ったときは、**プルリクエスト**を作成する
- レビューで OK が出たら、master へマージする
- マージした master を push したら、リリースする。

小規模の場合はこれでも複雑すぎる場合もある。

プルリクエスト

プルリクエストは git ではなく Github の機能。
レビュー・マージ作業をタスク化して、やりとりを記録できる。

- Github のリポジトリの上部のボタンでプルリクエストを作成
- 担当者がレビューして、修正が不要ならマージされる
- 修正が必要なら修正を依頼し、修正を push 後またレビュー
- そもそもプルリクエスト自体が不要ならクローズされる

このような流れが関係者にオープンになり、また記録される。
またオープンなリポジトリなら、自由にプルリクエストが作成できるので、自由に開発に参加することができる（もちろん担当者がプルリクエストのレビューを捌けるならではあるが）