# git

山下 俊樹，外村 篤紀
Toshiki YAMASHITA，Atsuki TONOMURA

## 1　はじめに

近年，ソフトウェアの大規模化にともない，プログラムの開発頻度は増加傾向にある．システムによる管理を行わず，開発に使用するファイルを編集した場合を考える．この場合，ファイルを以前の状態に復元したい時に，復元したい状態のファイルが発見しづらい問題点がある．そこで，コンピュータ上で作成または編集したファイルの変更履歴を管理するバージョン管理システムは，より重要となっている．本稿では，バージョン管理システムの 1 つである git，および関連サービスの GitHub について述べる．

## 2　git

### 2.1　概要

git はバージョン管理システムの 1 つであり，CUI で動作する．バージョン管理システムは，管理しているファイルを以前の状態に戻す機能や，複数人で 1 つのファイルを編集するときに発生してしまう競合を解消する機能を持つ．これらの機能を用いることで，開発中にバグが発生した場合，ファイルをバグが発生する以前の状態に復元できる利点がある [1)]．

### 2.2　構成

git で管理されるファイルやフォルダの変更内容は，リポジトリとよばれるデータベースに蓄積される．分散型バージョン管理システムにおけるリポジトリは 2 種類に分類できる．git の構成を Fig. 1 に示す．

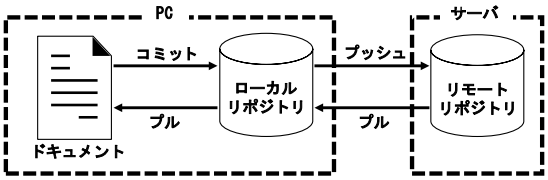


Fig.1　git の構成

ローカルリポジトリは，個人の PC 内に配置される．ファイルやフォルダの変更をローカルリポジトリに記録する操作をコミットとよぶ．コミットやプッシュを行うことで作業成果を記録する．ローカルリポジトリによって，ネットワークに接続されていない環境でもコミットを行うことができる．

リモートリポジトリは，サーバ上に配置され，複数ユーザで利用する．プルを行うことでリモートリポジトリから他者の作業成果をダウンロードしてローカルリポジトリに統合する．例えば，バグ修正のために個人のローカルリポジトリにコミットを行い，修正を完了したとする．その後，修正が完了したソフトウェアをリモートリポジトリにプッシュし，他者に公開する方法が可能である．

### 2.3　バージョン管理

git は，履歴を管理するファイルを全て蓄積しており，必要なファイルを適切に取り出すことによりバージョン管理を実現している．バージョン管理の例を Fig. 2 に示す．

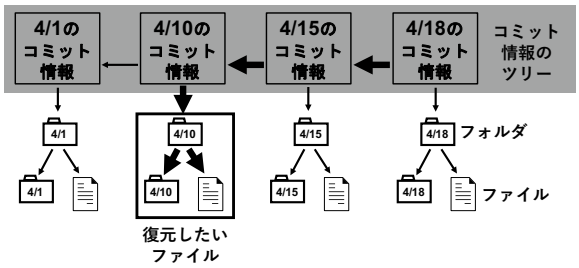


Fig.2　バージョン管理の例

コミットを行うと，コミット情報，および履歴を管理しているファイルがリポジトリに保存される．コミット情報は以下の 2 つで構成される．

- コミットしたファイルの保存場所
- 前回のコミット情報の保存場所

前回のコミット情報の保存場所を辿ることで，復元したいファイルやフォルダにアクセスできる．

### 2.4　ブランチ

ブランチは，コミットの履歴の流れを分岐して管理することを指す．分岐したブランチ同士は互いに独立しており，異なる内容の更新を同時に行える．ブランチ同士の結合はマージとよぶ．ブランチを利用した例を Fig. 3 に示す．

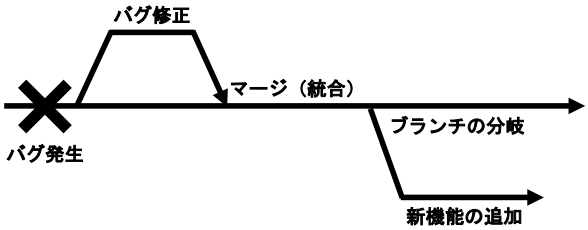


Fig.3　ブランチの例

ブランチを用いることで，様々な開発やバグ修正などを並行して行うことができる．Fig. 3 において，既に運用されているソフトウェアの管理を行っているブランチを分岐する場合を考える．ソフトウェアのバグ修正や，新機能を

追加するために，それぞれブランチを分岐する．この場合，3 つのブランチは独立しているため，運用中のソフトウェアに影響を与えることなく，バグ修正と新機能の追加を行える．ブランチを用いない場合，修正や開発を行うためには，ソフトウェアの運用を停止しなければならない．しかし，ブランチを用いることで，ソフトウェアの運用を継続しながら，他の修正や開発を行うことができる．

git のブランチは，他のバージョン管理システムに比べて優れている．他のシステムでは，マージの際にどのファイルをどのようにマージするか入力する必要があるが，git は自動的に行うことができる．管理するファイルが多い場合，手動でマージの指示を入力するには労力を要する．しかし，git を用いることでマージが自動的に行われ，労力を削減できる．

## 3 GitHub

### 3.1 概要

GitHub とは, git を利用した SNS である．現在, GitHub は最も人気のある git サービス提供サイトであり, そのユーザ数は 1000 万人を超えている [3]．GitHub の特徴の 1 つは，分かりやすい GUI である．GitHub のリモートリポジトリの内容は全て公開されるため，ソースコードを公開できない Web デザインや，個人情報を扱う開発などには利用できない．しかし，リモートリポジトリが無料で提供されるため，個人でリモートリポジトリ用のサーバを用意する必要がない．その結果，GitHub は手軽にリモートリポジトリを使用できるサービスとして広まった．

他のユーザと協力してソースコードを管理するための機能として，コードレビュー，およびコメント機能がある．コードレビューとは，ソースコードに含まれる誤りを検出，修正することを目的として行われるソースコードの査読を指す．これらの機能を用いることで，各ユーザが互いのソースコードを評価することができる．その結果，ソースコードの改善が行われ，ソースコードのバグの解消や可読性の向上といった利点を生む．

### 3.2 GitHub におけるブランチの活用例

git のブランチの分岐は，GitHub ではフォークとよばれる．GitHub のユーザは他のユーザのリポジトリからフォークしたアプリケーションを引き継いで開発することができる．フォークに 1 つのタスクを割り当てることでチケット駆動開発を行うことができる．

次に，チケットについて述べる．チケットとは，全体の作業を細かいタスクに分割し，タスク 1 つ 1 つに対して割り振られる，各タスクの作業指示である．開発者は発行されたチケットを取り，チケットに記されたタスク（バグ修正や機能追加など）をこなす．指示内容が完了するとコミットを行い，チケットを消去（クローズ）する．タスクをチケットで管理することにより，作業の全容が把握しやすいことや，チーム開発においてタスクの分配が行いやすくなる利点がある．

チケットを用いて行うチケット駆動開発は，アジャイル開発とも親和性が高い．アジャイル開発とは，開発対象を

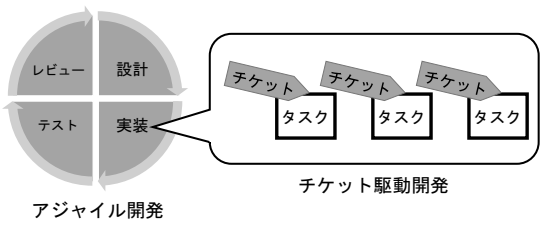


Fig.4　アジャイル開発とチケット駆動開発

多数の小さな機能に分割し，短い期間（1 週間から 4 週間）で 1 つの機能を開発する工程を反復しながら，開発を行う手法である．アジャイル開発における小さな機能は，チケット駆動開発のタスクに対応する．Fig. 3 に示したように，アジャイル開発におけるタスクにチケットを割り当てることで，チケット駆動開発との統合を実現する．チケット駆動開発を行う際には，オンラインプロジェクト管理ソフトウェアである Redmine を併用する場合が多い．

### 3.3 BitBucket との比較

GitHub に競合するサービスとして，BitBucket が挙げられる． GitHub に対して，無料で非公開のリモートリポジトリを作成できる利点があるが，1 つのリモートリポジトリを利用できるユーザ数は最大 5 人に限られる．そのため，ソースコードを共有して開発を行える人数が 5 人に限られる．その結果，大人数での開発には使用できず，開発規模が制限される欠点がある．

どちらのサービスもソーシャルコーディングの普及に貢献しているが，現時点でのユーザ数は GitHub が圧倒的に多い．これは，GitHub に対応する周辺サービスが多いこと，著名な開発プロジェクトが多数あること，および GUI が使いやすいことによる．

## 4 今後の展望

今日，ソフトウェアのリリース速度は増加の一途を辿っている．それにともない，開発の効率化，高速化が重要視されている．そのため，git のようなバージョン管理ソフトウェアは更に普及すると考えられる．また，GitHub のような git を用いたサービスは，現在概ね無料で公開されている．しかし，ソフトウェアの開発において，デバッグや開発の一部を他者に依頼し，一番良いソースコードを含むフォークに報酬を払うビジネスといった，商業利用が近い将来に生まれると考える．

## 参考文献

1) 岡本隆史：Git に潜む光と闇，入手先，(http://gihyo.jp/dev/column/01/prog/2012/git)
2) koseki2：Git の仕組み (1)，入手先 (http://koseki.hatenablog.com/entry/2014/04/22/inside-git-1)
3) Ken Nishimura，市民生活をオープンデータ活用で改善する政府や地域行政の事例 [GitHub Universe]，(http://thebridge.jp/2015/10/github-universe-session-changing-lives-with-open-data)．