git

山下 俊樹，外村 篤紀
Toshiki YAMASHITA，Atsuki TONOMURA

1　はじめに

近年，ソフトウエアは大規模化し，修正を行う対象の数が増加したため，その更新頻度は増加の一途を辿っている．そこで，コンピューター上で作成，および編集される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の重要性が増している．さらに，バージョン管理システムには，管理下のファイルを任意の記録時点の状態に復元することができる，および一つのファイルを複数人で編集する場合，競合が発生しないように管理が行われるなどの利点があるため，注目を集めている．

バージョン管理システムには，サーバー上のみでファイルの管理を行う集中型バージョン管理システムと，サーバーに加え，個人の PC でも管理を行う分散型バージョン管理システムがある．本報告では，分散型バージョン管理システムの一つである git の概要，内部処理，および関連サービスについて述べる．

2　git

2.1　git の構成

git の構成を以下の Fig. 1 に示す．

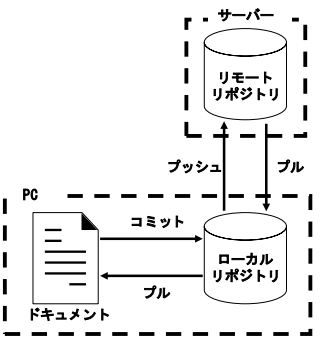

Fig.1　git の構成

git で管理されるファイルやディレクトリの変更内容は，リポジトリと呼ばれる一種の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，サーバー上に配置され，複数ユーザーで利用するリモートリポジトリと，個人の PC 内に配置され，その個人が利用するローカルリポジトリに分類される．ユーザーがファイルやディレクトリの変更をローカルリポジトリに記録する作業をコミットと呼び，これを実行すると最新の状態が記録される．過去のコミット時点の状態への復元操作，およびブランチを切り替える操作をチェックアウトと呼ぶ．コミットやプッシュを行うことで作業成果を記録し，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統合することができる．

2.2　ブランチ

git の特徴の一つであるブランチについて述べる．ブランチの概念図を以下の Fig. 2 に示す．

Fig.2　ブランチの概念

ブランチとは，蓄積されたコミットの時系列と，系列を分岐する作業を指す．分岐したブランチ同士は互いに独立しており，任意のブランチ内のコミットは他のブランチでのコミットの影響を受けない．ブランチ同士の結合はマージと呼ぶ．ここで，ブランチの一例を以下の Fig. 3 に示す．

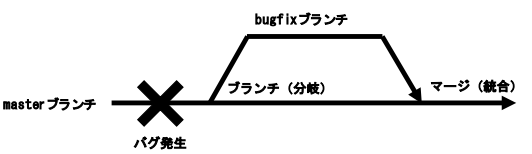

Fig.3　ブランチの一例

既に運用されているソフトウエアの開発を行っているブランチを master ブランチとし，そのソフトウエアのバグを修正するために用意したブランチを bugfix ブランチとする．この場合，二つのブランチは独立しているため，master ブランチに影響を与えることなく bugfix ブランチでバグの修正を行う

ことができる．すなわち，ブランチを使用しない場合はバグの修正が終了するまで，ソフトウエアの運用を停止しなければならないが，ブランチを用いることで運用を止めることなく，平行してバグの修正を行うことができる．

2.3 内部処理

内部処理・・・

2.4 利点

また，他のバージョン管理ソフトウエアに対するgit の利点は次の通りである．

- ローカルリポジトリを持つ
- ブランチ機能が強力
- 高速である

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．あるいは，バグの修正のために個人のローカルリポジトリにコミットを適量蓄積し，修正が完了した後にリモートリポジトリにプッシュし他者に公開する等の使用方法が可能である．ブランチについては，マージを行う際に自動化されている比率が高いこと，マージが高速である等が挙げられる．更に，プログラム全体の動作が他サービスに比べて高速である．これは，git が Linux カーネル開発に使用するために開発されたため，OS カーネルのような巨大なソースコードの集合を高速に処理する必要があった経緯がある．

一方，欠点として，git で扱えるファイルの種類はテキストベースのファイルに限られることが挙げられる．このため，書類作成のデファクト・スタンダードである，Microsoft Office で作成したファイルを改変なしでは管理できない等の不具合を生じる．

3 GitHub とは

GitHub とは，git を利用した SNS である．ユーザーはリモートリポジトリを無料で持つことができ，他のユーザーと協力してソースコードを管理できるサービスとして広まった．他のユーザーと協力するための機能として，コードレビュー，及びコメント機能等がある．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，ユーザーのソースコードが他者の目に触れ，それに対してのコメントが付与される．その結果，ユーザーによる修正が行われ，ソースコードのバグが減少する，及び可読性が向上する等の利点を生む．

次に，フォーク機能について述べる．GitHub におけるフォークはブランチの分岐操作を指す．この機能によって，ユーザーが他のユーザーのリポジトリからフォークしたブランチを用いて開発を行うことができる．利点としては，あるソフトウエアのデバッグを開発者以外のユーザーが行う事ができる，初心者がある程度完成されたソフトウエアをフォークすれば，フォーク元の本体には影響を与えず，様々な改変を行いながら学習ができる等が挙げられる．

更に，チケット機能についてである．GitHub 内では Issue と呼ばれているが，ここではチケットと同義として説明を行う．チケットとは，任意の作業をタスクに分割し，タスク一つ一つに対して割り振られるものである．開発者は発行されたチケットを取り，それに記されたタスク（バグ修正など）をこなし，タスク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．このように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，チームでの開発においてタスクの分配が行い易くなる等の利点がある．また，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ため，注目を集めている．

GitHub の利点は上記の通りである．一方，欠点として GitHub で作成したリモートリポジトリは全て公開されるため，ソースコードを公開しない場合の多い Web デザイン等のソースコード管理には不具合が発生する．また，GitHub は外部サービスであるため，サーバー障害の様な事態が発生した場合に関連サービスが使用不可能になる危険性がある．実際に，2016 年 1 月 28 日にサービス障害が発生し，多くのユーザーや企業が被害を被った [1)]．

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，非公開のリモートリポジトリを作成できる，及び git 以外の分散型バージョン管理システムである Mercurial を使用できる等の利点がある．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と言えるが，現時点でのユーザー数は GitHub が圧倒的に多い．また，GitHub は対応する周辺サービスが多いこともあり，更にユーザーを増やしている．他にも競合するサービスがあるが，現段階では Github が最も人気のある git ホスティングサイトであり，そのユーザー数は 1000 万人を超えている [2)]．

4　今後の展望

今日，ソフトウエアのリリース速度は増加の一途をたどり，その裏では開発の効率化，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エアやチケット駆動開発は更に普及すると考えられる．また，現在は GitHub のような SNS サービスは概ね無料で公開されているが，あるソフトウエアの開発において，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等のビジネスも近い将来に生まれると考える．

参考文献

1) Yukari Mitsuhashi，GitHub、1 月末のダウンの原因はデータセンターでの停電と説明，(http://www.itmedia.co.jp/news/articles/1602/01/news070.html)．
2) 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ーー日本からは国土地理院が登壇 [GitHub Universe]，(http://thebridge.jp/2015/10/github-universe-session-changing-lives-with-open-data)．