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，プログラムの更新頻度は増加傾向にある．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 つである．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できることや，1 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えることなどの利点がある [1)]．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データベースに蓄積される．

2.2　構成

git の構成を Fig. 1 に示す．

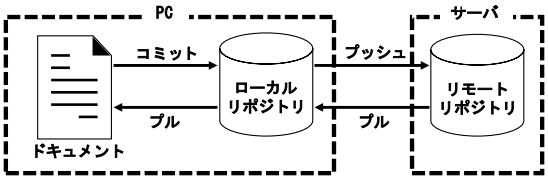

Fig.1　git の構成

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 種類に大別できる．1 つは，サーバ上に配置され複数ユーザで利用するリモートリポジトリである．もう 1 つは，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である．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操作をコミットと呼ぶ．コミットやプッシュを行うことで作業成果を記録する．また，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．

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，バグ修正のために個人のローカルリポジトリに適量のコミットを行い，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．

次に，バージョン管理手法について述べる．コミットを行うと，対象のディレクトリの内容がリポジトリに全て保存される．参照しているコミットには目印が付けられており，目印からコミットを辿ることで，いかなるコミットの対象ファイルやディレクトリにも芋づる式にアクセスできる．

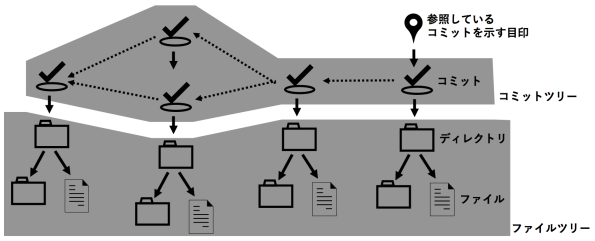

Fig.2　ツリー構造

コミットの時系列順の管理は，コミットツリーで行う．コミットツリーとは，コミット間のポインタで繋がれたツリー構造である．1 つのコミットで管理される複数のファイルやディレクトリは，ファイルツリーと呼ばれるツリー構造で表される．したがって，任意のコミット時点の状態に戻す場合は，目印からコミットツリーをたどり，対応するファイルを取り出せばよい．git の動作は目印による管理のために，高速である．

2.3　ブランチ

git の特徴の 1 つであるブランチは，蓄積されたコミットの時系列を指す．すなわち，複数回のコミットを作成順に並べた連なりをブランチと呼ぶ．また，ブランチを分岐する操作もブランチと呼ぶ．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呼ぶ．ブランチを利用した例を Fig. 3 に示す．

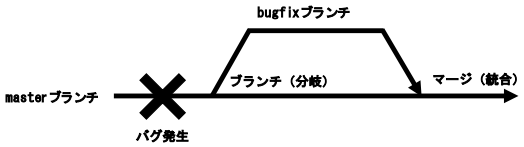

Fig.3　ブランチの例

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 master ブランチとし，そのソフトウェ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 ブランチでバグ修正を行うことができる．すなわち，ブランチを用いない場合

は，バグ修正が完了するまでソフトウェアの運用を停止しなければならない．それに対し，ブランチを用いれば，ソフトウェアの運用を止めることなく，平行してバグ修正を行うことができる．バグ修正が完了した後にマージを行った時点でバグが修正される．さらに，ブランチの本数を増やせば，リリースしたソフトウェアを運用しながら新機能の追加を行い，加えてバグ修正も同時に行うといった運用が可能である．

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明示的に入力する必要があるが，git は自動的に行うことができる．また，マージにかかる時間も他のシステムに比べて高速である．

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 ホスティングサイトであり，そのユーザ数は 1000 万人を超えている [3]．GitHub のユーザはリモートリポジトリを無料で持つことができるため，他のユーザ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．他のユーザと協力するための機能として，コードレビュー，コメント機能などがある．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，ユーザのソースコードが他者の目に触れ，コメントが付く．その結果，ユーザによる修正が行われ，ソースコードのバグが解消や可読性の向上などの利点を生む．

3.2 特徴

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．フォークによって，ユーザが他のユーザ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．利点は，ソフトウエアのデバッグを開発者以外のユーザが行えることである．さらに，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ことなどが挙げられる．

次に，チケットについて述べる．GitHub 内では Issue と呼ばれているが，本稿ではチケットと同義とする．チケットとは，作業をタスクに分割し，タスク 1 つ 1 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．この一連の流れでタスクを管理する．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，チームでの開発においてタスクの分配が行い易くなることなどの利点がある．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 デザインなど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，サーバ障害などが発生した場合に関連サービスが使用不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，多くのユーザや企業が被害を被った [4]．

3.3 関連サービス

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 を使用できるなどの利点がある．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や，GUI が使いやすいことなどによる．

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が，ある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などのビジネスも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).
4) 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).