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．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 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われるなどの利点があるため，注目を集めている [1)]

バージョン管理システムには，サーバ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 1 つである git の概要，内部構造，および利点について述べる．

2　git

2.1　git の構成

git の構成を以下の Fig. 1 に示す．

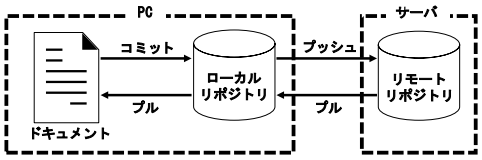

Fig.1　git の構成

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操作をコミットと呼び，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

2.2.1　概要

git の特徴の 1 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．

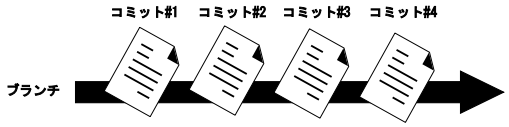

Fig.2　ブランチの概念

ブランチとは，蓄積されたコミットの時系列と，系列を分岐する操作を指す．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．

2.2.2　バグ修正におけるブランチの有用性

ブランチを利用した一例を以下の Fig. 3 に示す．

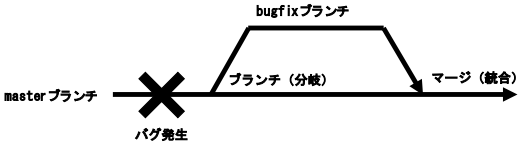

Fig.3　ブランチの一例

ブランチを用いることで，様々な開発やバグ修正などを同時並行で行うことができる．具体的には，Fig. 3 において既に運用されているソフトウエ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，バグ修正が完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ソフトウエ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えば，ソフトウエアの運用が行われたまま，マージの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エアを運用しながら新機能の追加を行い，さらにバグ修正も同時並行で行うといった運用が可能である．

2.3　内部構造

2.3.1　概要

本項では，git がどのようにバージョン管理を行っているかについて述べる．git で管理されているディレクトリの最上層には.git ディレクトリが生成されており，.git ディレクトリがローカルリポジトリの本体である．.git ディレ

クトリの構成を以下の Fig. 4 に示す．

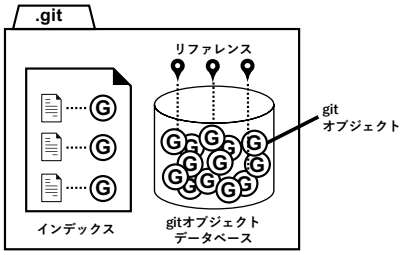

Fig.4　.git ディレクトリの構成

.git ディレクトリ内には，git オブジェクト，リファレンス，およびインデックスが格納されている．git オブジェクトの集合を git オブジェクトデータベースと呼ぶ．リファレンスは各 git オブジェクトに 1 つずつ割り当てられる目印であり，インデックスは git オブジェクトのデータベース内の格納位置を示している．git オブジェクトの内容を以下の Fig. 5 に示す．

Fig.5　git オブジェクト

git オブジェクトは，ツリーオブジェクト，ブロップオブジェクト，コミットオブジェクト，およびタグオブジェクトの 4 種類がある．ツリーオブジェクトはディレクトリに，ブロップオブジェクトはファイルに相当する．git で管理するディレクトリとファイルを，ツリーオブジェクトとブロップオブジェクトに変換し管理する．ここで，ツリーの関係を以下の Fig. 6 に示す．

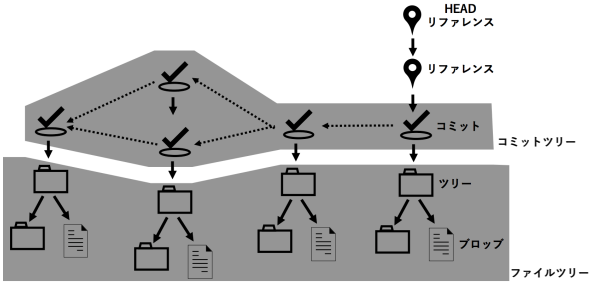

Fig.6　ツリーの関係

リファレンスは特定のコミットオブジェクトに対応し，コミットオブジェクトにはツリーオブジェクト，およびブロップオブジェクトが対応する．すなわち，リファレンスを目次として，特定のコミットの対象ファイルやディレクトリに芋づる式にアクセスできる．他にも，HEAD リファレンスと呼ばれる，現在どのブランチを管理しているのかを示すリファレンスが存在する．HEAD リファレンスは，リファレンスに付けられる．さらに，コミットオブジェクトには，1 つ前のコミットオブジェクトへのポインタが含まれるため，コミットオブジェクトをたどることでブランチを表現できる．

2.3.2　解説

git では，コミットの時系列順の管理は，コミットオブジェクトのポインタで繋がれたコミットツリーと呼ばれるツリー構造で行う．また，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もファイルツリーと呼ばれるツリー構造で表される．コミットごとにディレクトリやファイルがデータベースに蓄積し，リファレンスを付けて管理する．さらに，コミットツリーをたどることで，過去のいかなるコミットにもアクセスすることができる．このことから，任意のコミット時点の状態に戻す場合は，リファレンス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．

ブランチのような複雑な管理を行う場合でも，リファレンスを用いて管理することにより，処理を軽減し，git 全体の動作を高速化している．差分抽出（変更点の表示）を行う場合も，対象の 2 つのファイルを取り出して比較を行うことで，他サービスより高速に動作する．

2.4　利点と欠点

git は，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エ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．この利点は他の分散型バージョン管理システムにも当てはまる．

さらに，ブランチ機能が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．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．

欠点は，git で扱えるファイルの種類がテキストベースのファイルに限られる点である．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．

3　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)