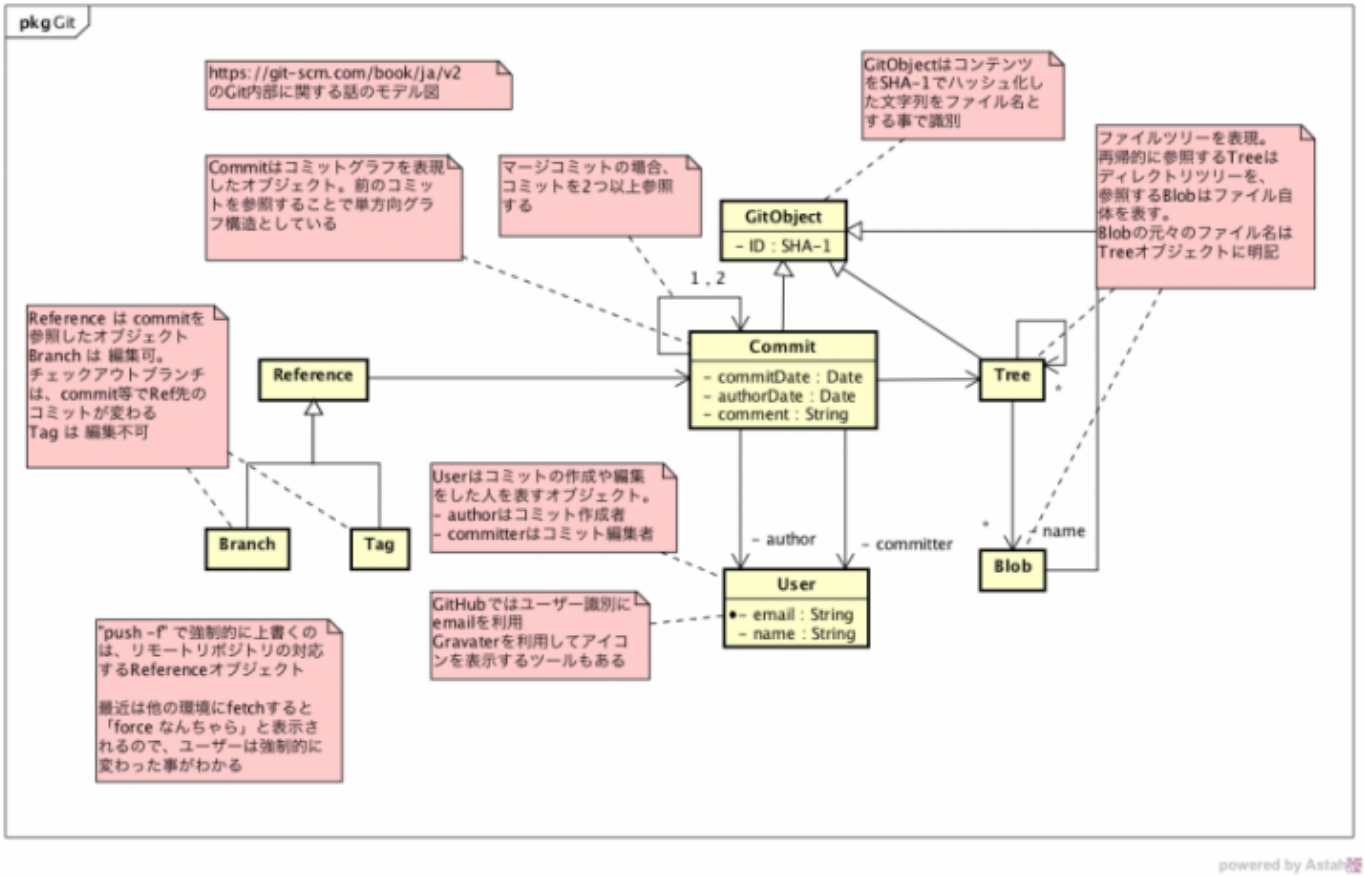
Gitのデータモデル

by kompiro July 16, 2015 event



近藤です。こんにちは。Gitは様々な利用の仕方ができますが、その基盤となるモデルは8個だけの簡単なモデルです。これらのモデルを理解していない状態でGitを利用すると、あたかもリポジトリが壊れたように見えてしまいます。`Git`は難しいと言われますが、そういう感想を持つ人はGitのモデルを理解していない事が多いようです。

今回はGitを構成する中心モデルと、基本的なコマンドを実行した時のオブジェクト関係を解説します。

基本概念

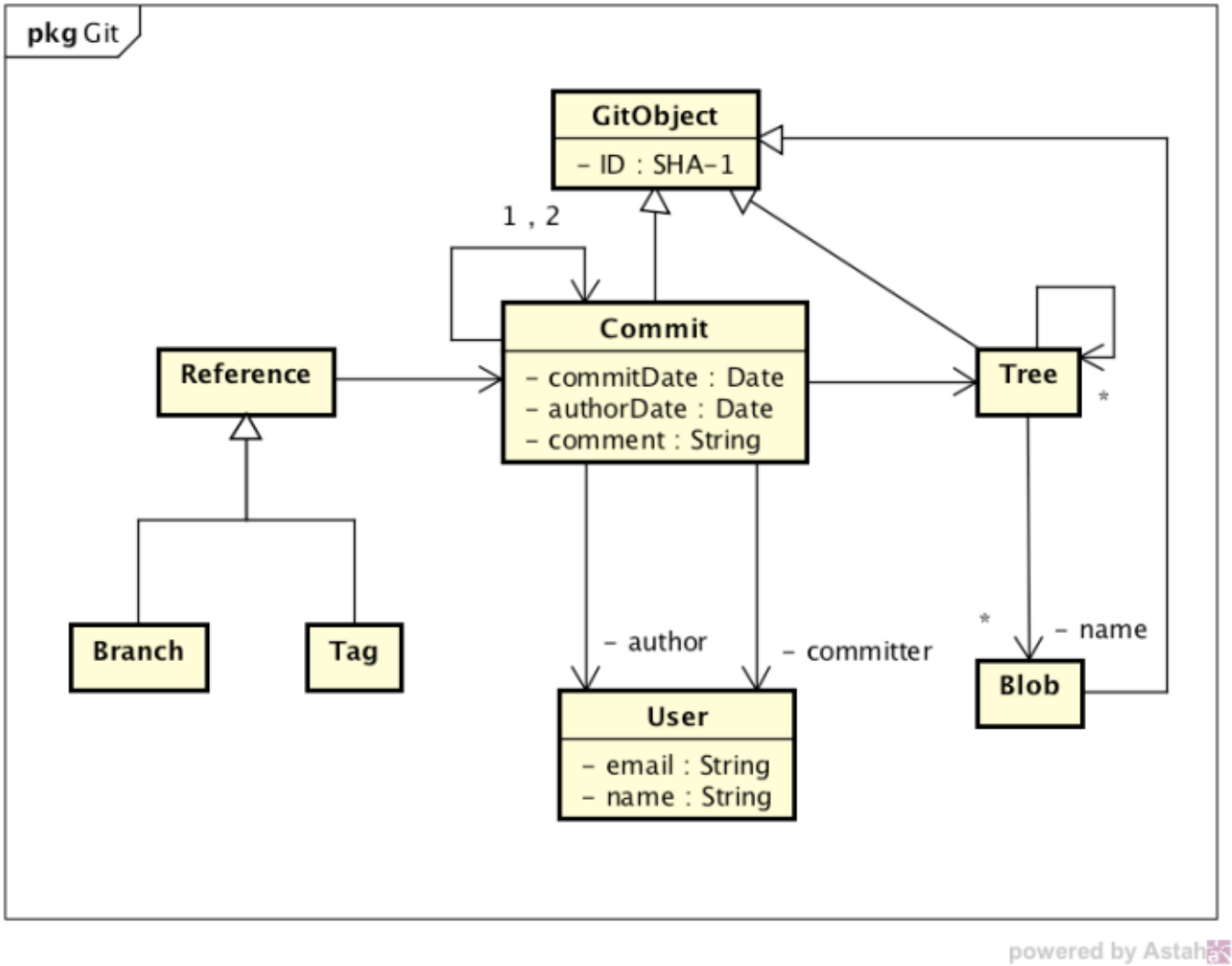
Gitの基本概念は大きく2つにわかれます。

- GitObject
- Reference

`GitObject`はGitで管理するオブジェクトです。Commitなどが`GitObject`です。Gitリポジトリである`.git`を開くと`objects`配下にあるファイルが`GitObject`です。`GitObject`はそのコンテンツをハッシュ化した文字列を元に、先頭2文字で配置フォルダ、残りの文字列でファイル名です。

`Reference`は`Commit`を参照しています。Branchなどが`Reference`です。Gitリポジトリである`.git`を開くと、`refs`配下にあるファイルが`Reference`です。`Reference`のコンテンツはCommitのIDです。

それぞれの概念をまとめたのが下記のモデル図です。それぞれの概念について解説します。

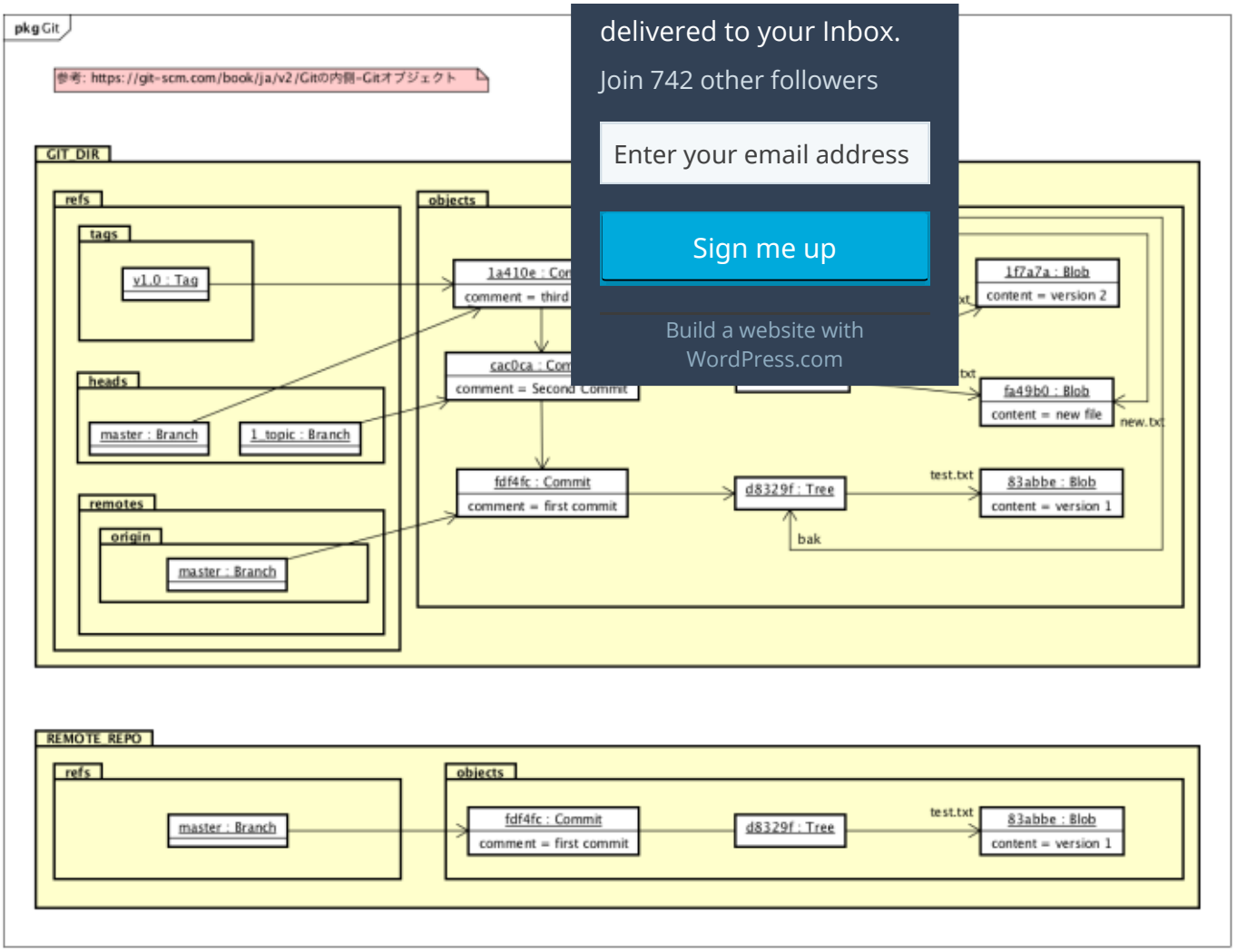


- Commit – コミットグラフを表現する概念です。`Commit`のコンテンツは`comment`だけでなく、1つ前のCommit、ファイルツリーを示すTree、作成者と編集者を示すUser、作成日時、編集日時がありま

す。`rebase`や`cherry-pick`、`commit --amend`を行うとCommitのIDが変更になりますが、それは1つ前のCommitが変更になったり、編集日時が変更になるためです。

- User – コミットを扱った人を表現する概念です。GitHub等ではユーザーアイコンを表示します。これは`email`を使って表示しています。
- Tree – フォルダツリーを表現する概念です。Treeは、子フォルダを示すTreeと、ファイルを示すBlobを持ちます。Treeへの参照、Blobへの参照共にnameがあります。
- Blob – ファイルを表現する概念です。ワーキングツリーのファイルそのものがコピーされます。`.git`の中には、これまで編集が少しでもあったファイルが全て入っています。
  Blobオブジェクトがファイルそのものであるため、`.git/objects`の容量は大きくなりがちです。少しでも抑えるため、全く同一のコンテンツであれば新たにファイルを作らなくても済むませるためのハッシュ文字列のファイル名だったり、ファイルの圧縮だったりします。
- Branch – ブランチそのものです。配置フォルダにより、ローカルリポジトリのブランチだったり、リモートリポジトリのブランチだったりします。`.git/refs/heads`配下にある`Reference`がローカルリポジトリのブランチです。`.git/refs/remotes`配下にある`Reference`がリモートリポジトリのブランチです。`.git/HEAD`に書かれたrefsが現在チェックアウト中のブランチを示します。ブランチをチェックアウトしている場合、コミットがあると、コミットに追従しますが、`.git/HEAD`から参照されたブランチのコンテンツを更新しているだけです。
- Tag – タグそのものです。`.git/refs/tags`配下にある`Reference`です。

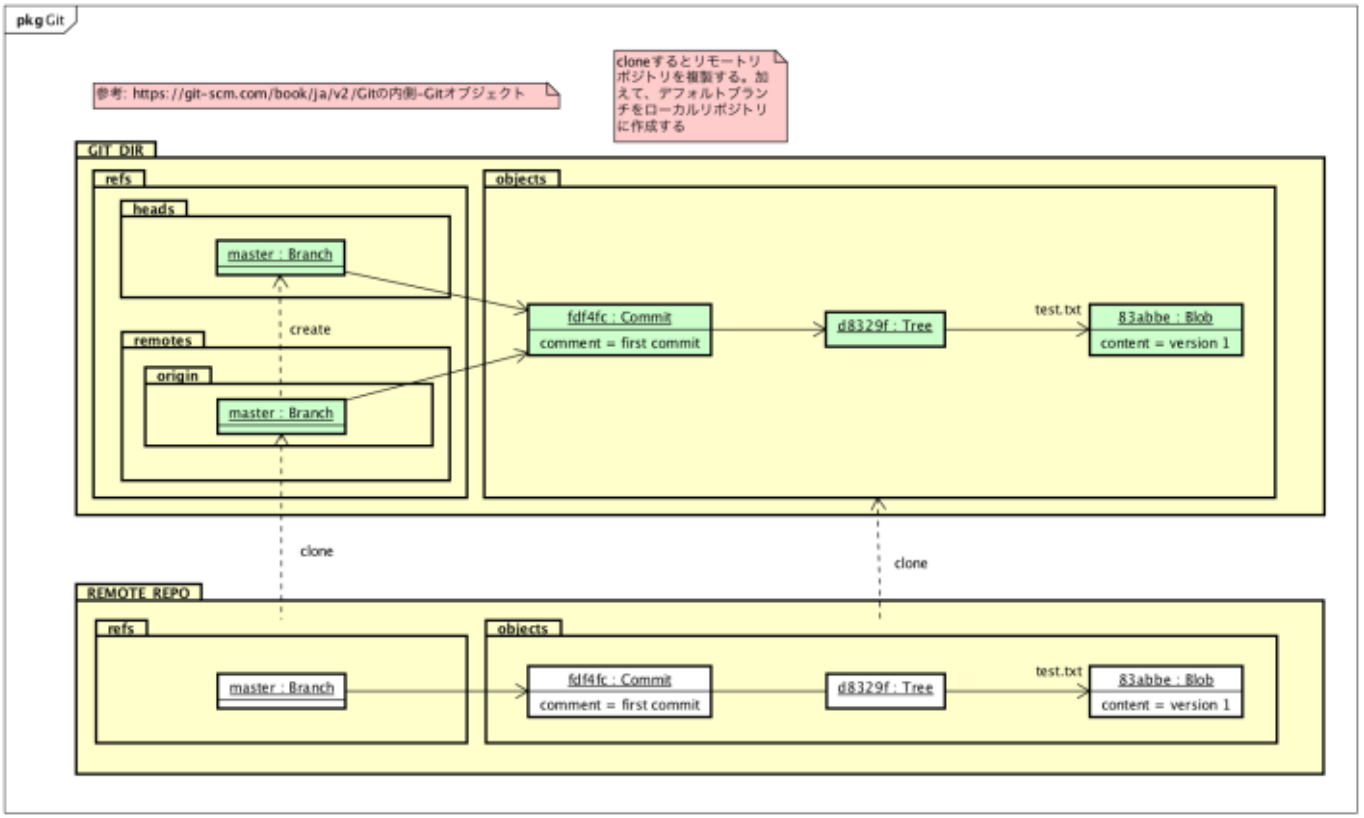
Gitは様々な機能を提供していますが、基本となるモデルはこれだけです。これらの概念を元に作成されたオブジェクトが下記のように関連しあっているのです。



コマンドで起きること

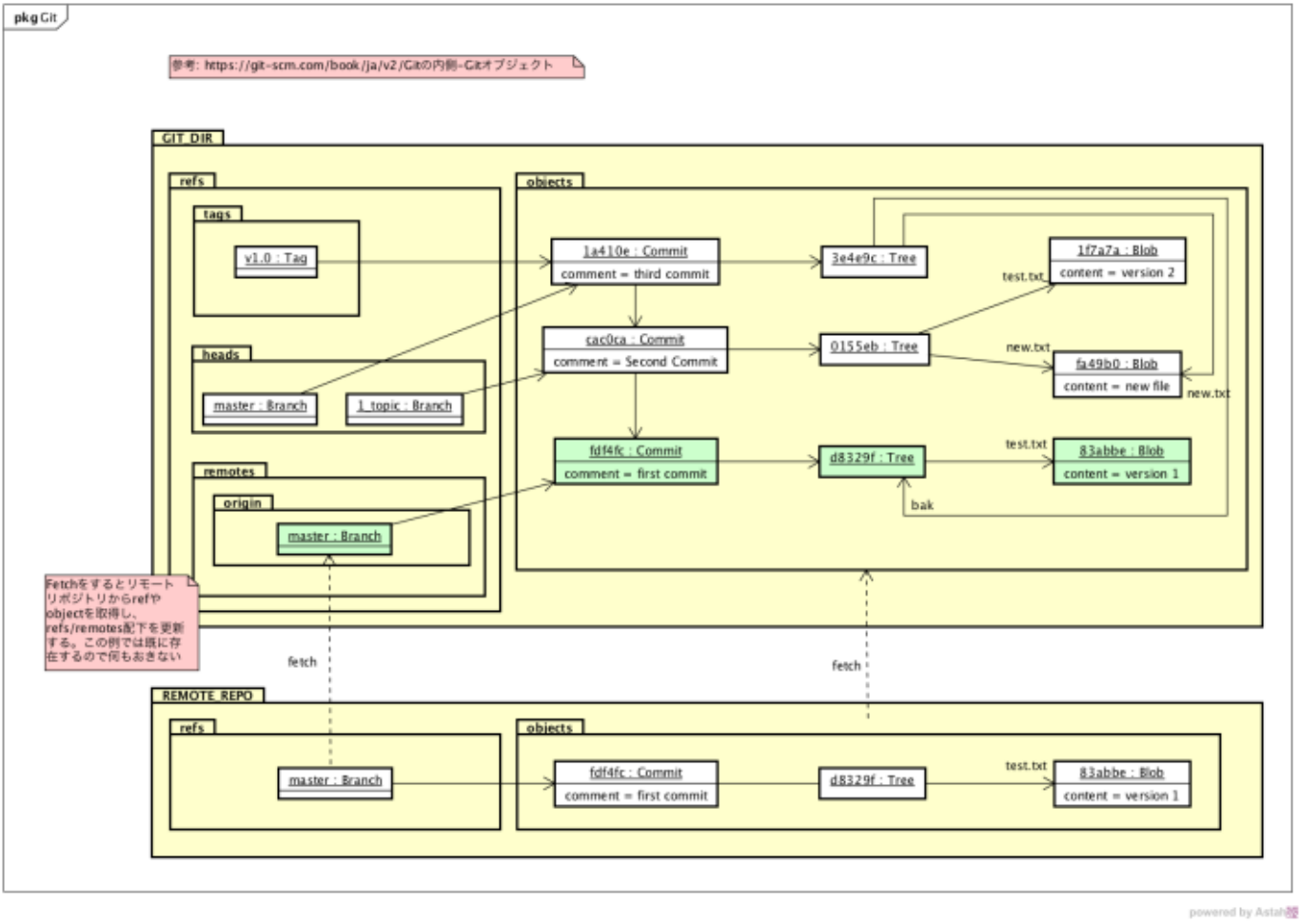
Gitはファイル名と配置フォルダにより、それぞれのオブジェクトを強く意味付けしています。Gitでよく使う`clone`、`fetch`、`push`,`pull`の4つのコマンドで起きることをモデル図にまとめてみました。

clone



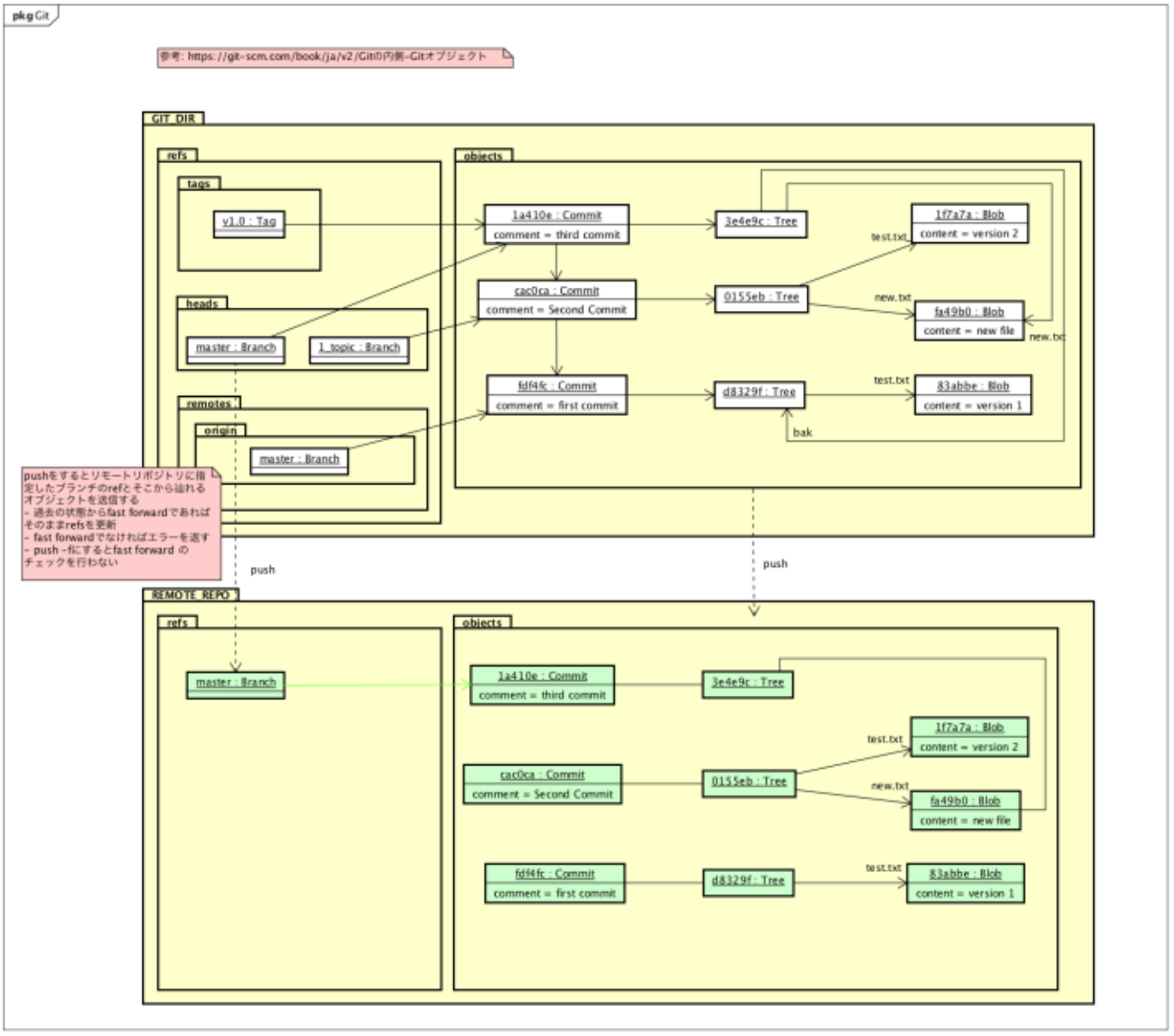
`clone`を実行すると指定したリモートリポジトリから`GitObject`と`Reference`を取得し、`master`ブランチをローカルブランチに作成します。

fetch



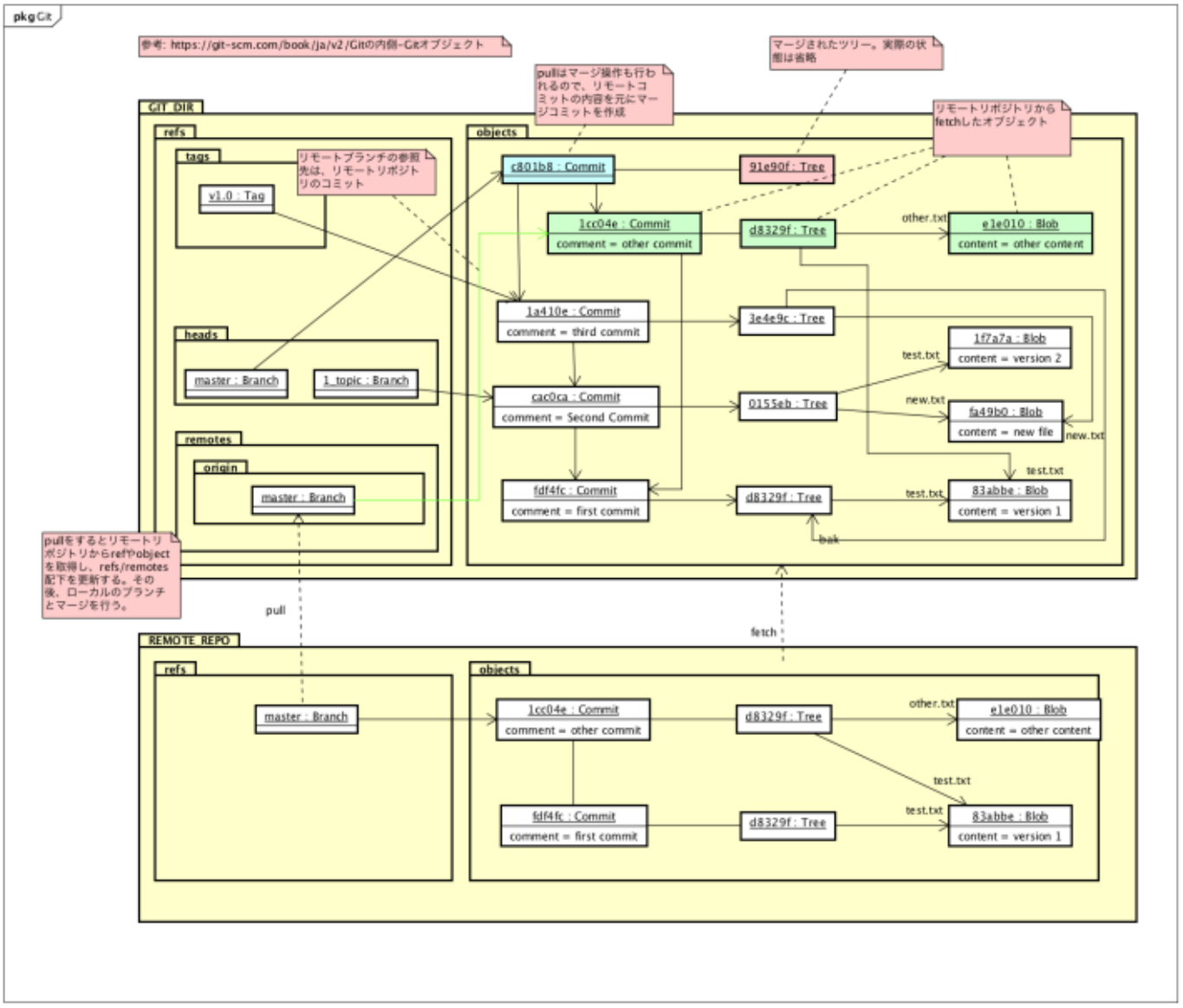
`fetch`を実行すると、指定したリモートリポジトリから`GitObject`と`Reference`を取得します。ローカルリポジトリのリモートブランチが更新されるだけです。ローカルブランチは更新されません。

push



`push`を実行すると、指定したリモートリポジトリに`GitObject`と`Reference`を送信します。リモートリポジトリのブランチとローカルブランチがFast Forwardではなかった場合、エラーが返ります。`push -f`はFast Forwardチェックを飛ばしているだけです。ブランチを指定している場合は、そのブランチと、ブランチに紐づく`GitObject`だけ送信します。

pull



powered by Astah

`pull`を実行すると、指定したリモートリポジトリから`GitObject`と`Reference`を取得し、ローカルブランチにリモートブランチをマージします。

リポジトリの設定

リポジトリの設定は`.git/config`にかかれています。開いてみると下記のようなファイルになっているでしょう。

```
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
        ignorecase = true
```

```
        precomposeunicode = false
[remote "origin"]
        url = git@github.com:ChangeVision/astah.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[ branch "master" ]
        remote = origin
        merge = refs/heads/master
```

この例では下記の内容が記述されています。

リポジトリ自体の設定

`[core]`の辺りです。共用のリポジトリではないので`bare = false`となっている、等がわかります。

リモートリポジトリの名前とURL

`[remote "origin"]`の辺りです。`origin`という名前のリモートリポジトリのURLは`git@github.com:ChangeVision/astah.git`、ブランチのfetch対象は`+refs/heads/*:refs/remotes/origin/*`と書かれてます。

ローカルブランチとリモートブランチをひもづけるUpstreamブランチ

`[ branch "master"]`の辺りです。`master`ブランチのUpstreamは`origin`の`refs/heads/master`と設定されています。

Gitの基本的な概念と設定についてざっくりと解説しました。案外単純だったのではないでしょうか？

こういった感じでモデル化してみると、文章だけでは見えなかった概念のつながりが見えるので、よりわかりやすかったのではないでしょうか？こういう感じで、利用しているツールの概念をモデル化してみると、よりよい使い方がわかるのでオススメです。

参考文献

GitBook