SCMBC Git 資料

第二回 SCMBC Git 班

Git のリポジトリ

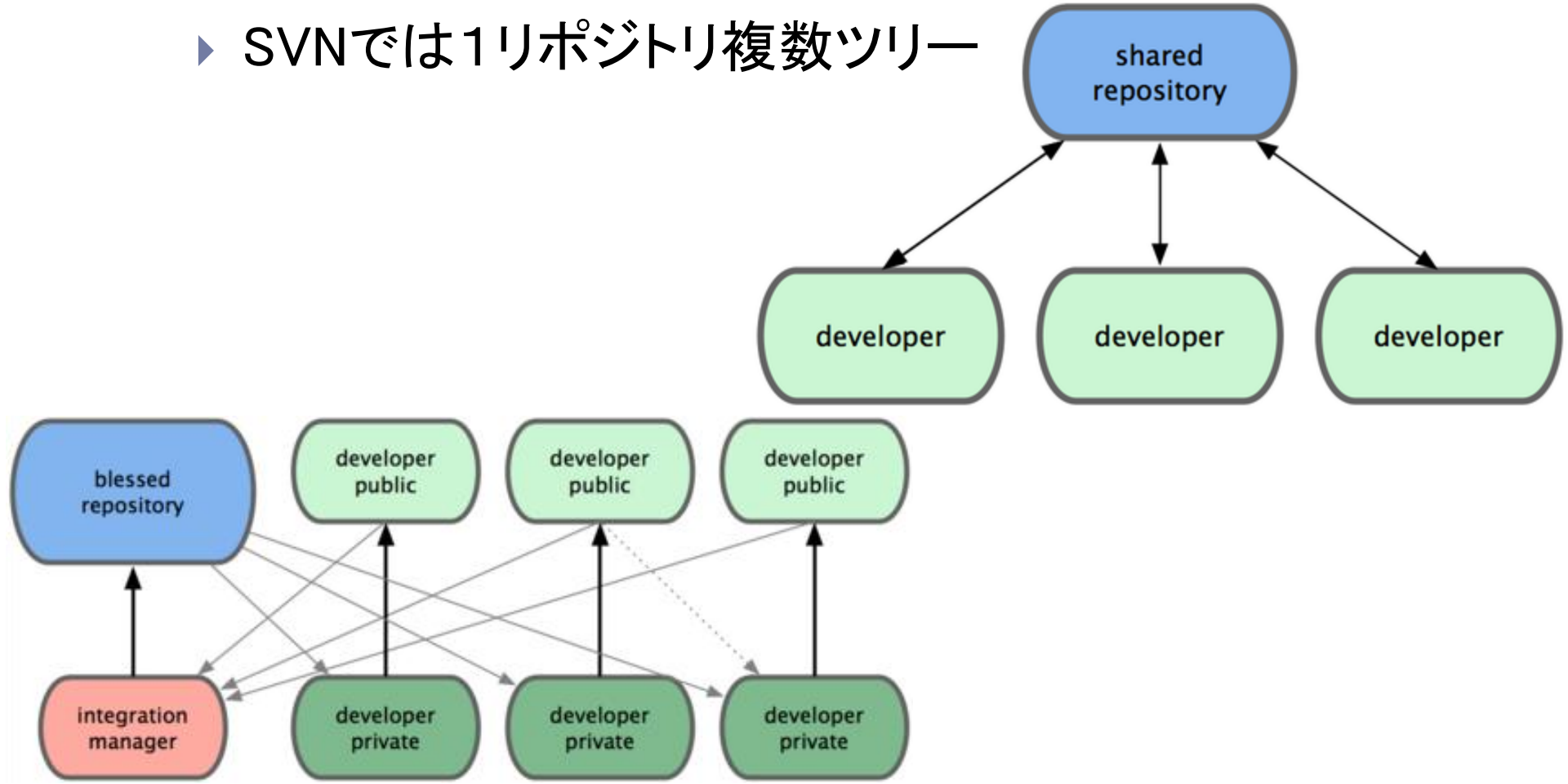
- リポジトリ　＝　データを貯めるところ

- Git ではリポジトリがローカルにある
  - SVNではローカルにないことが多い
  - ローカルのリポジトリに対する操作は高速（通信不要）
  - push, pull などを使って同期を取る（通信がここで発生）

- 手元のリポジトリではコンフリクトしない

多人数開発

- SVNでは1リポジトリ複数ツリー

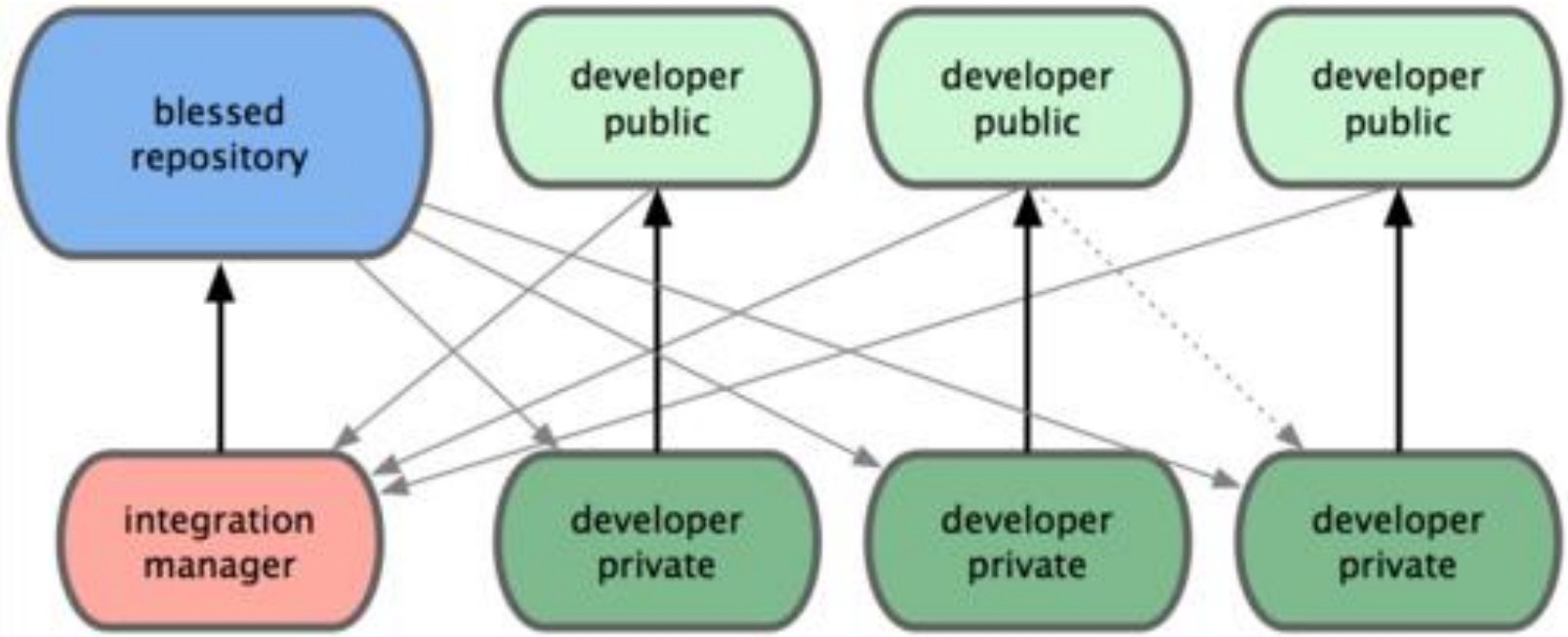


- Gitでは個人がリポジトリを持つ

Figures from Pro Git http://progit.org/book/ja/ch5-1.html

多人数開発

- 共有リポジトリに pull, push をする
- 共有リポジトリは複数ある場合も
  - CIサーバとステージング用と、、、



Figures from Pro Git http://progit.org/book/ja/ch5-1.html

Git のオブジェクト

- すべて Immutable
  - 作成されたら破棄されないかぎり変更されない

- Blob : ファイルの中身
- Tree : ディレクトリ構成
- Commit : コミット内容
- （Tag）

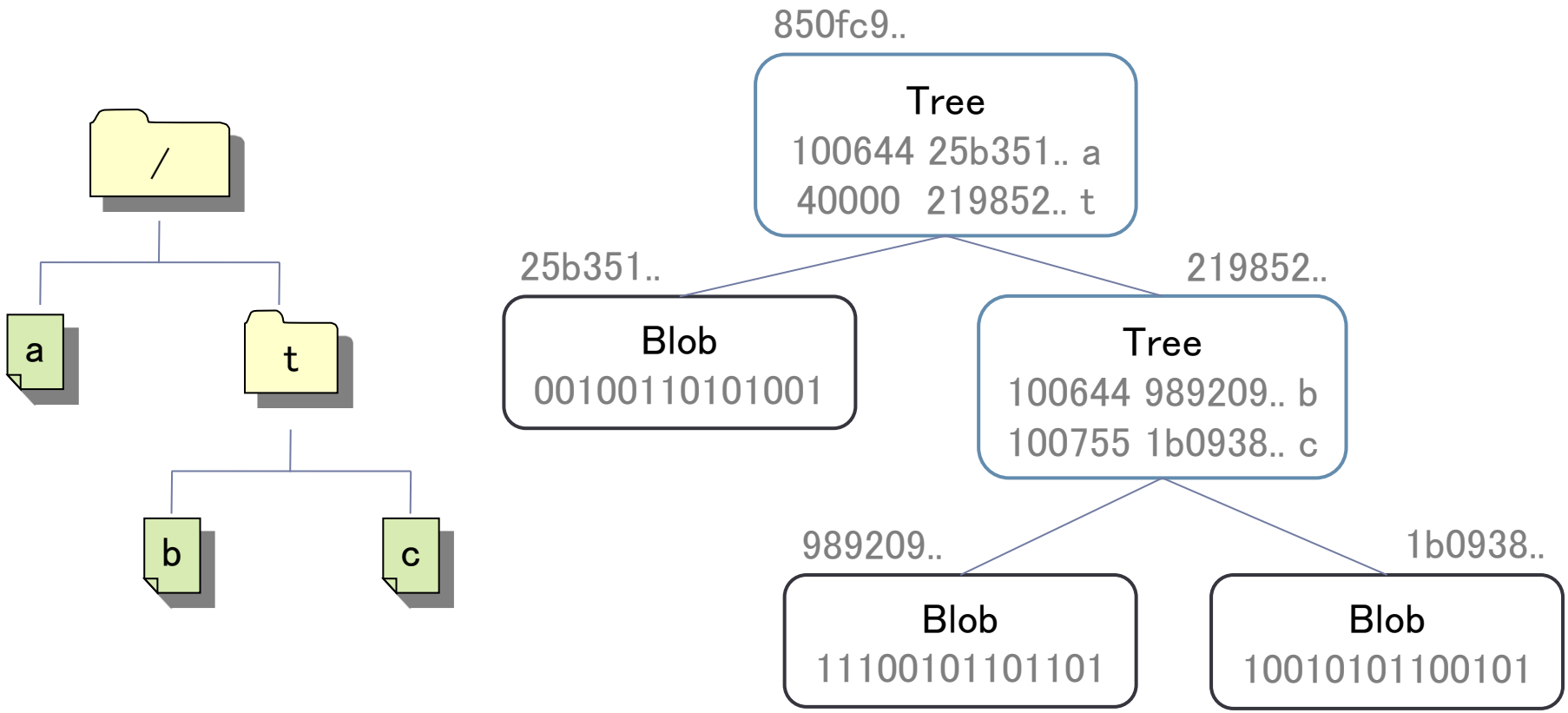
Blob オブジェクト

- ファイルの中身だけを表す
- ファイル名などは Tree オブジェクトが保持
- Tree や Commit をまたいで参照される
  - このために Immutable になっている
- 差分ではなく、スナップショット

Tree オブジェクト

- ディレクトリ構成を表す
  - 子ファイル
  - 子ディレクトリ
- 同一のオブジェクトは複数のツリーから参照される
  - ディスクの空間効率をよくするため

実ファイルと Git オブジェクト



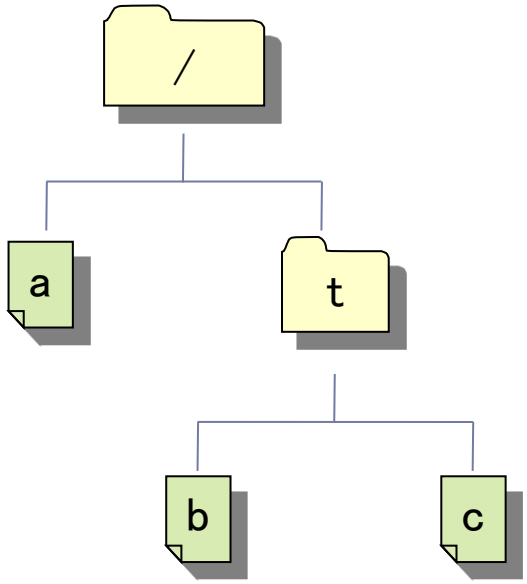
Commit オブジェクト

- コミット（リビジョンの記録）
  - コミットした人、時間、メッセージ
  - 親コミット
  - ルート Tree …
- 親コミット
  - 通常ひとつ
  - マージした場合、複数
  - 初回コミットにはない
- 親コミットを順にたどることで歴史がわかる

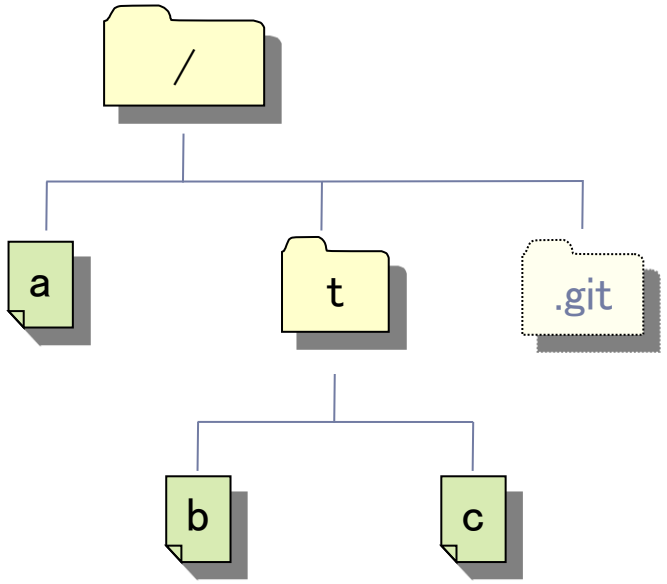
コミットメッセージ

- 普通にGitを使うとコミットメッセージは必須
  - 空だとエラーになる
- 一行目に概要、二行目を空白にして、三行目以降に詳細
  - 色々なコマンド（主にログ系）がこのフォーマット前提
  - 詳細が不要な時は一行目だけ

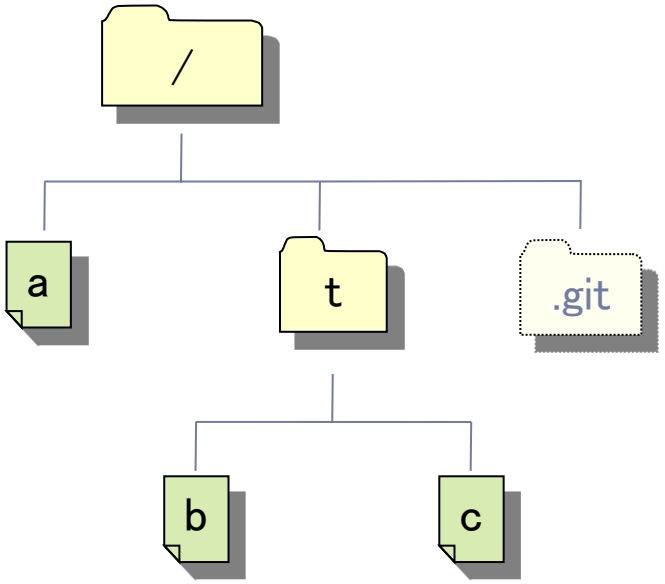
コミットの様子

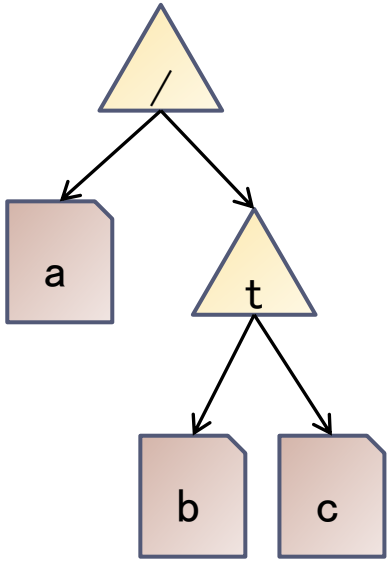


git init



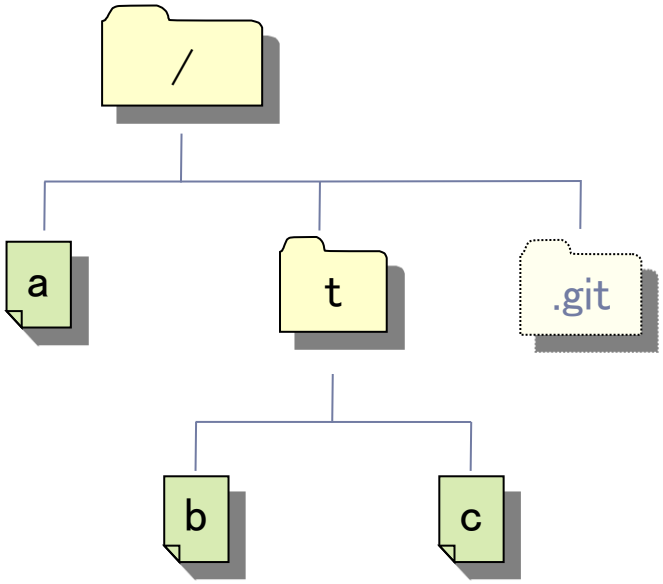
git add .

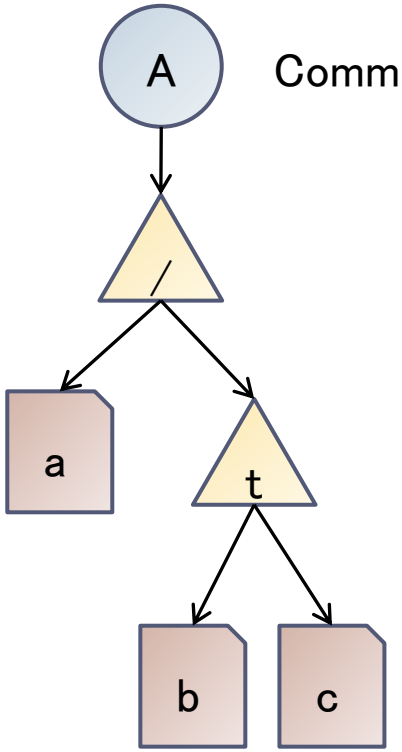




Tree オブジェクトや
Blob オブジェクトが作られる

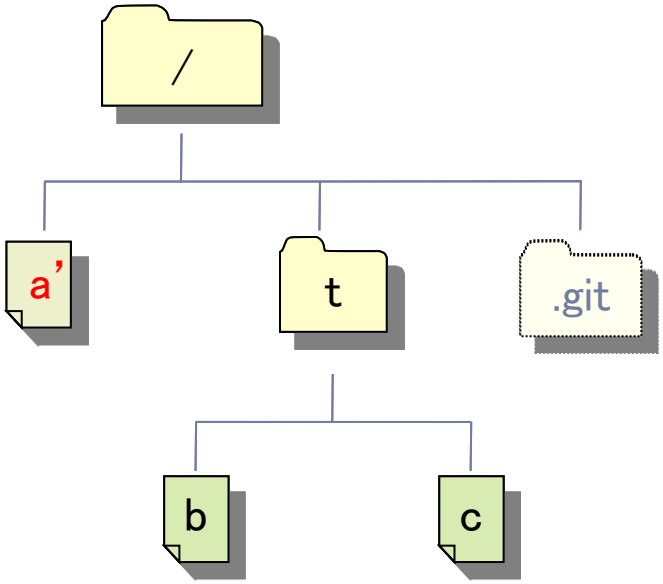
git commit

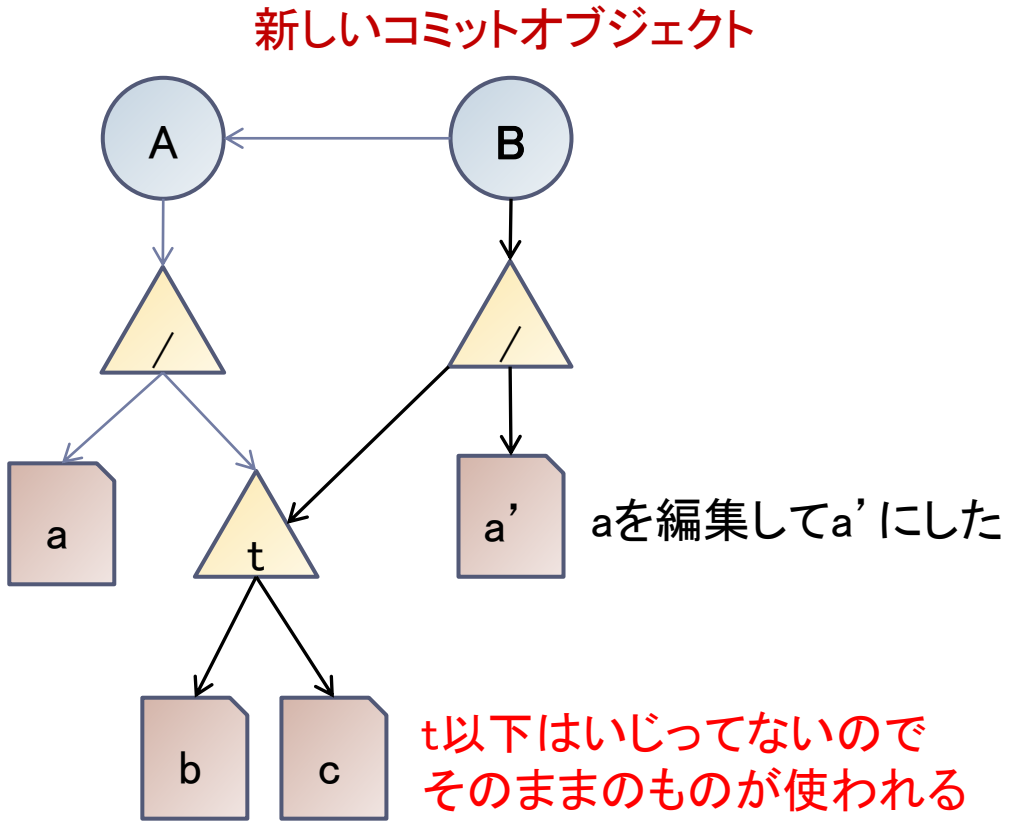




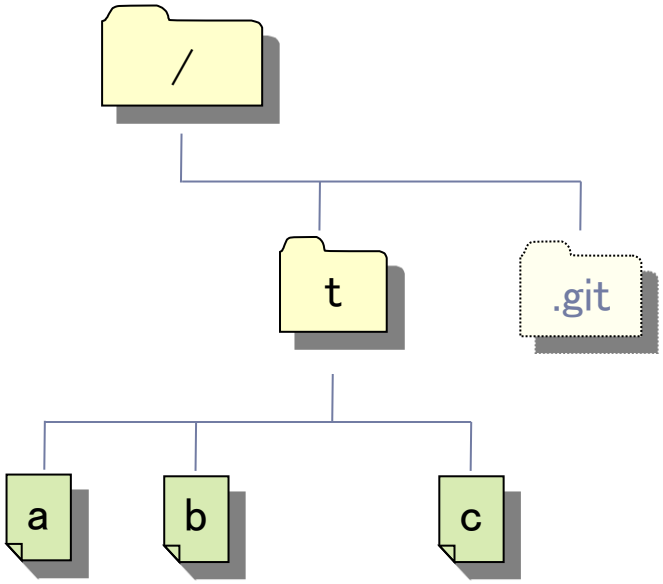
Commit オブジェクトが作られる

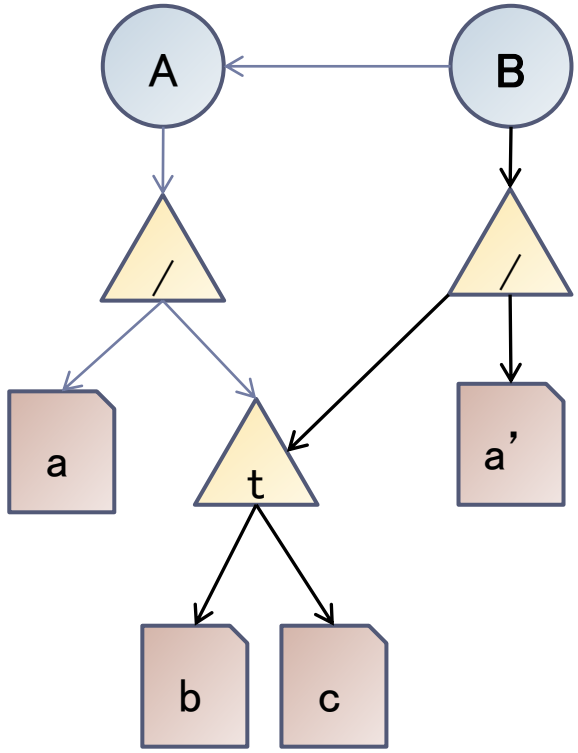
edit a; git add a; git commit



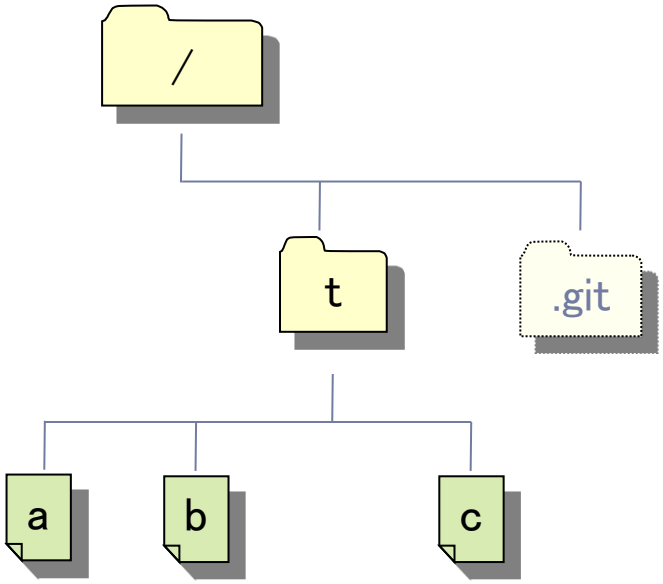


a を t 配下に移動 (mv a t/a)

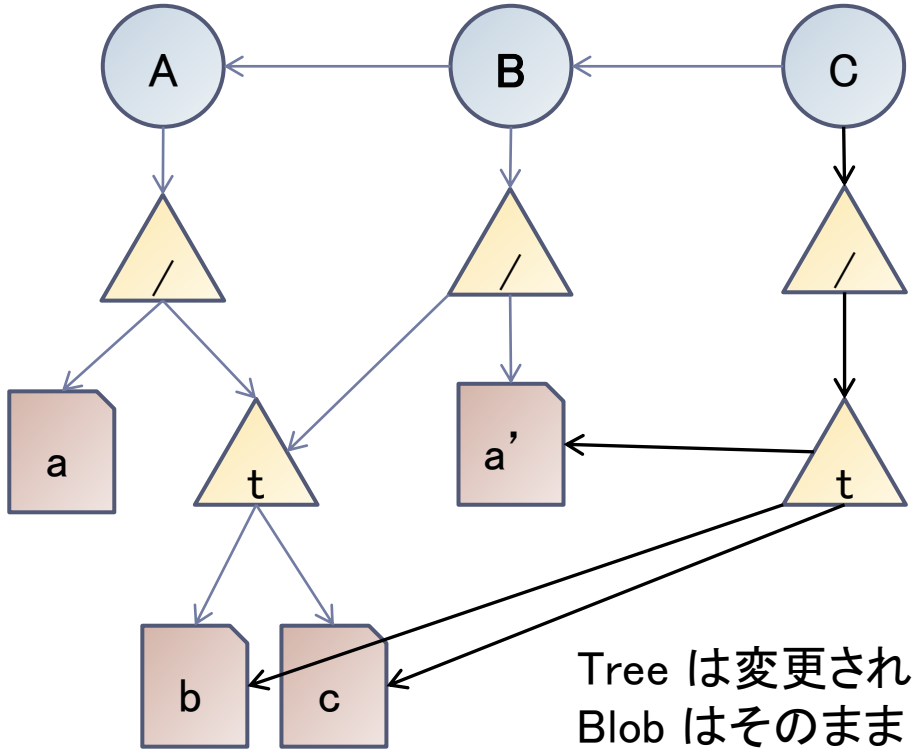




git add -A; git commit



新しいコミットオブジェクト



Tree は変更されたが
Blob はそのまま

オブジェクトのハッシュ値

- すべてのオブジェクトの SHA-1 ハッシュ
- 比較はすべてハッシュ値で行う

- 世界中で(事実上の)一意性が担保される
  - SVNなど連番リビジョン番号との違い
  - リポジトリが分散しても安心(後述)

- リモートとの通信でもハッシュ値でオブジェクトについて判断できるので高速、低負荷

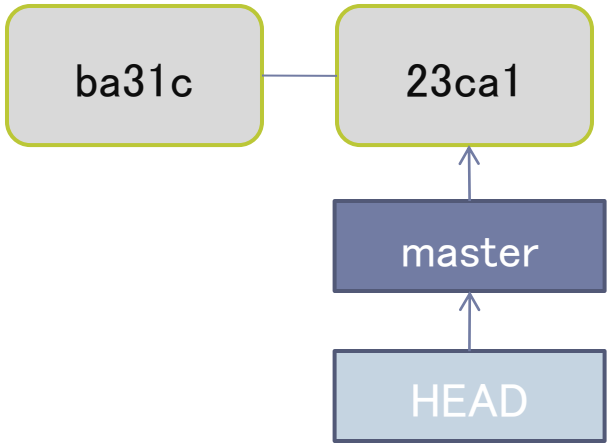
ブランチ

- Commit オブジェクト（ハッシュ値）へのポインタ
  - 作成、削除が高速
- Commit オブジェクトの親コミットをたどることでブランチが表現できる
- ブランチの切り替え
  - 重複しているオブジェクトをハッシュ値で区別

ブランチの使い方

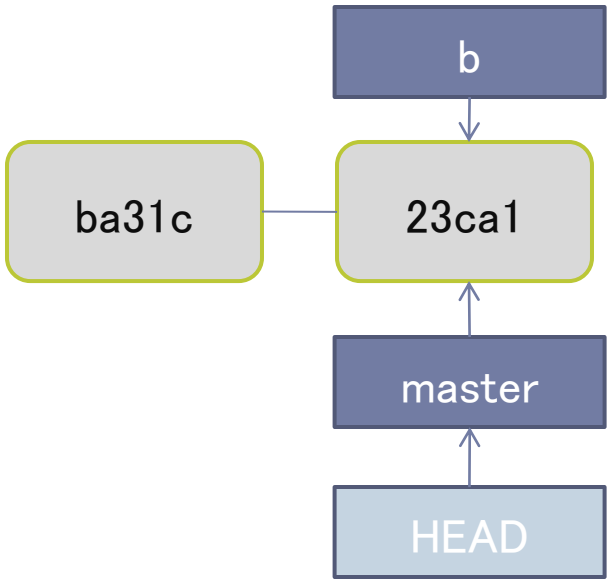
- 最初は master
  - git branch で作成
  - git checkout で移動
- フィーチャブランチ(トピックブランチ)
  - 機能ごとにブランチをきる
  - 短命なブランチ
- さまざまなプラクティス
  - **A successful Git branching model**
    - 英語: http://nvie.com/posts/a-successful-git-branching-model/
    - 日本語:http://keijinsonyaban.blogspot.com/2010/10/successful-git-branching-model.html

ブランチのイメージ

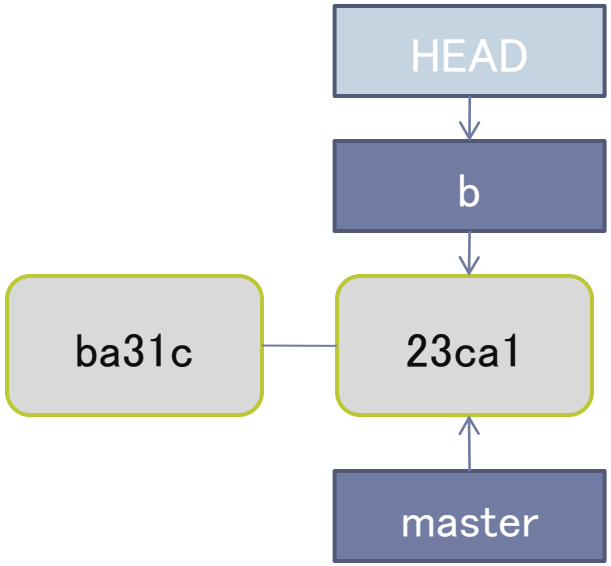


HEAD は現在のブランチを表す

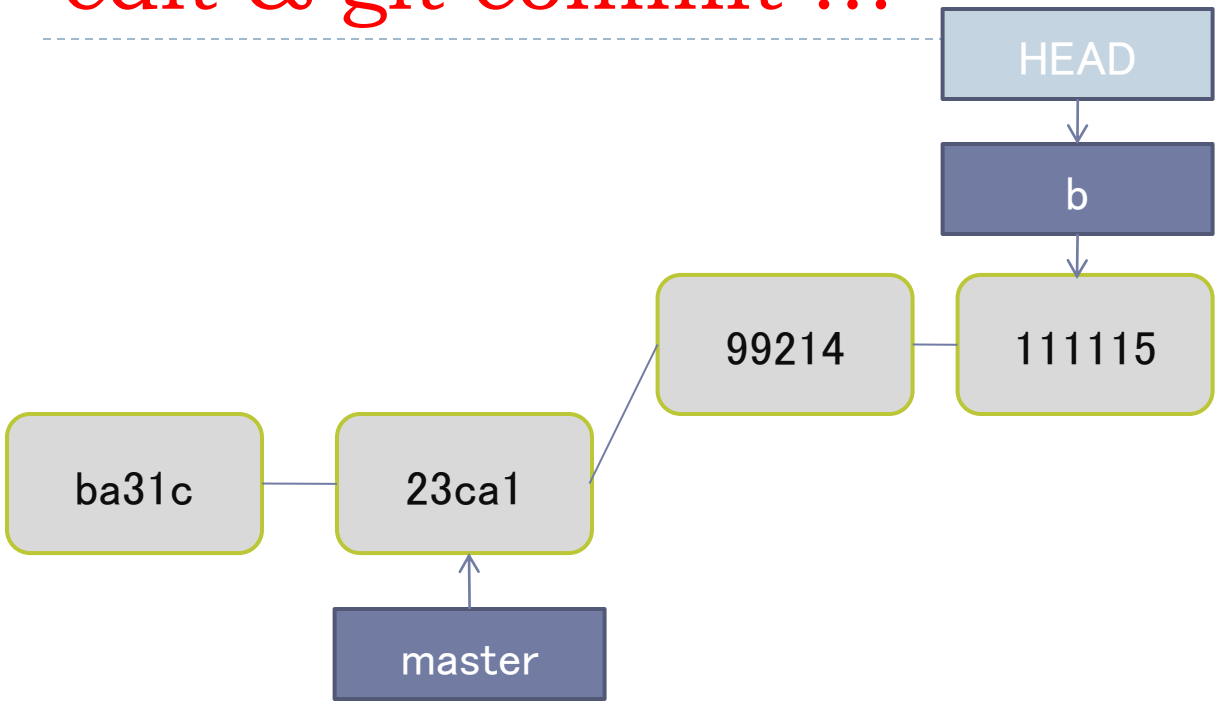
git branch b



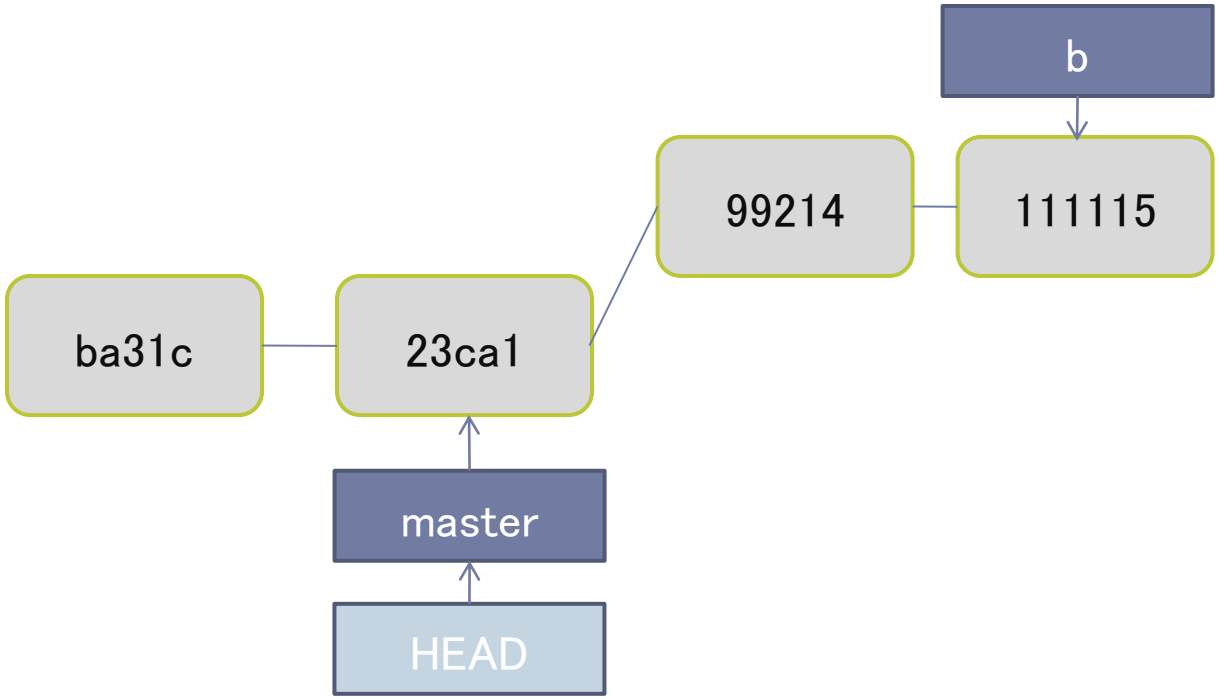
git checkout b



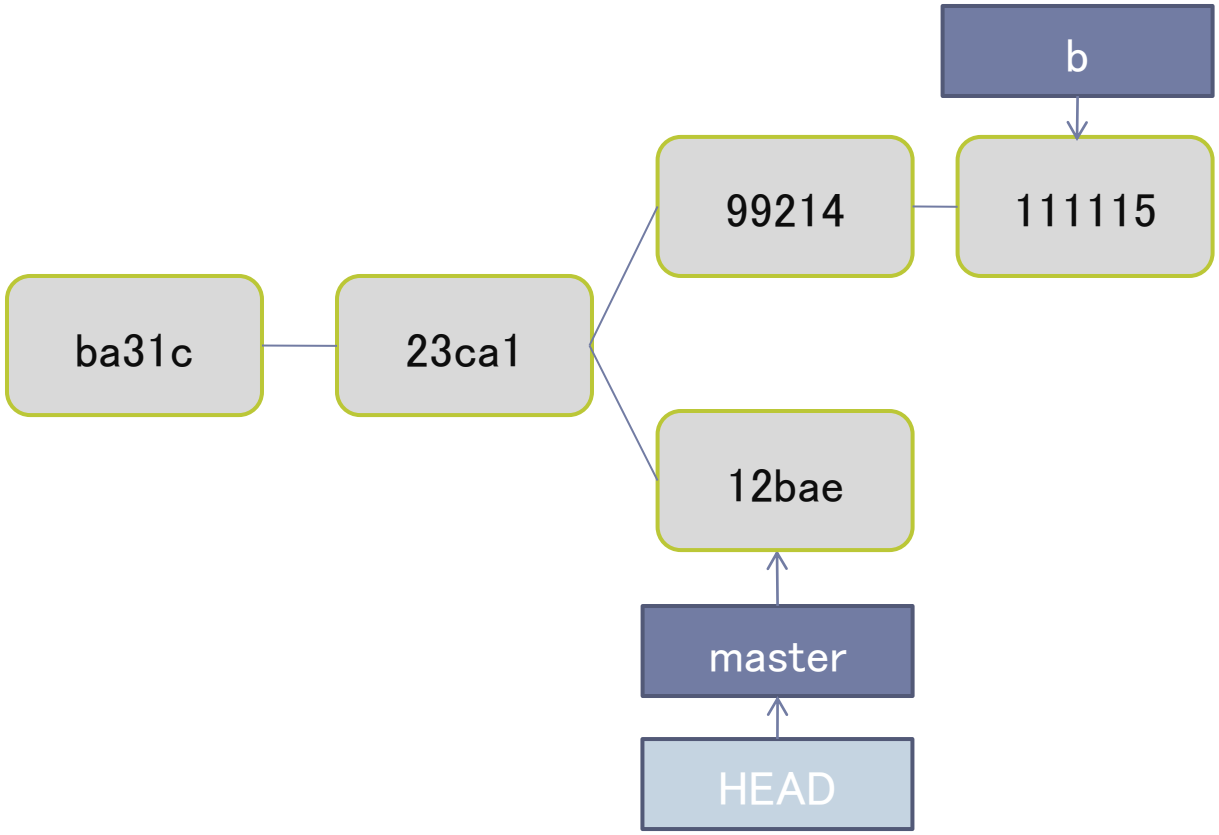
edit & git commit …



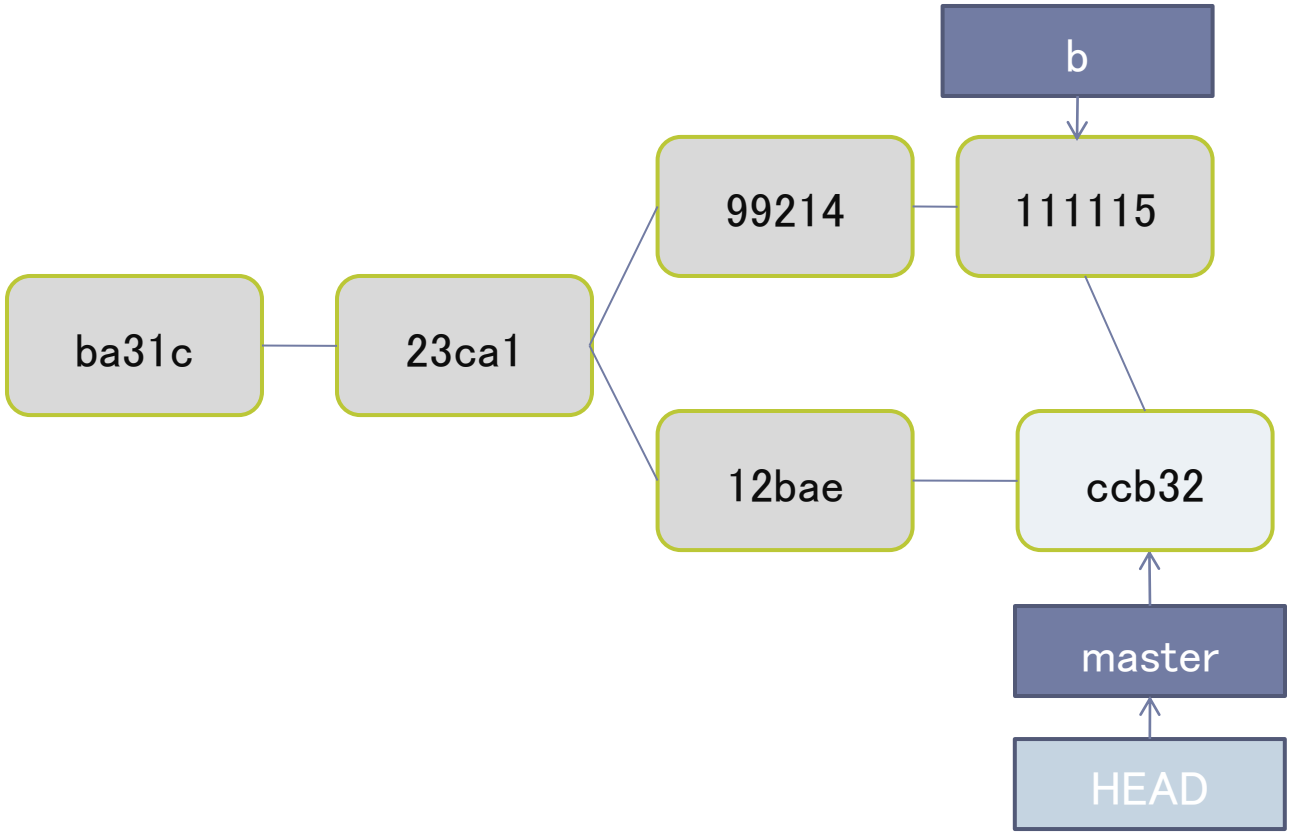
git checkout master



edit; git commit

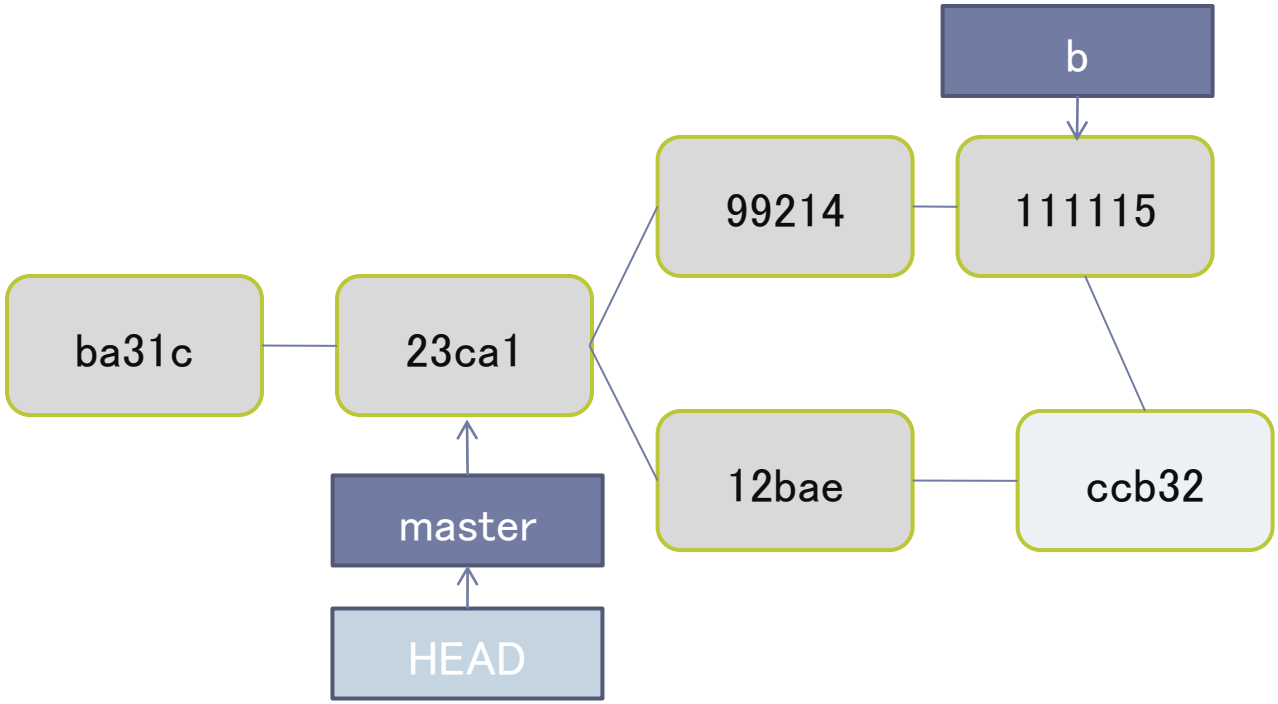


git merge b

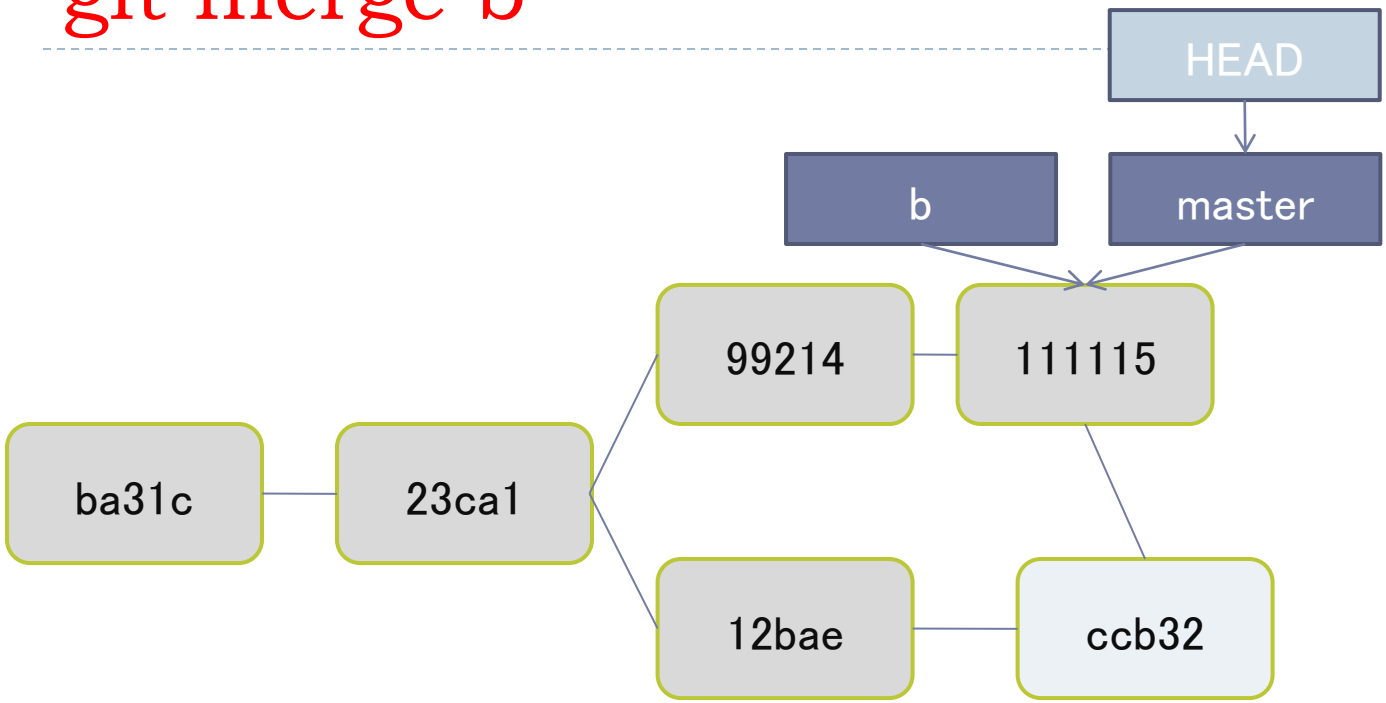


両方の変更点を問題ない形で持つ

git reset --hard 23ca1



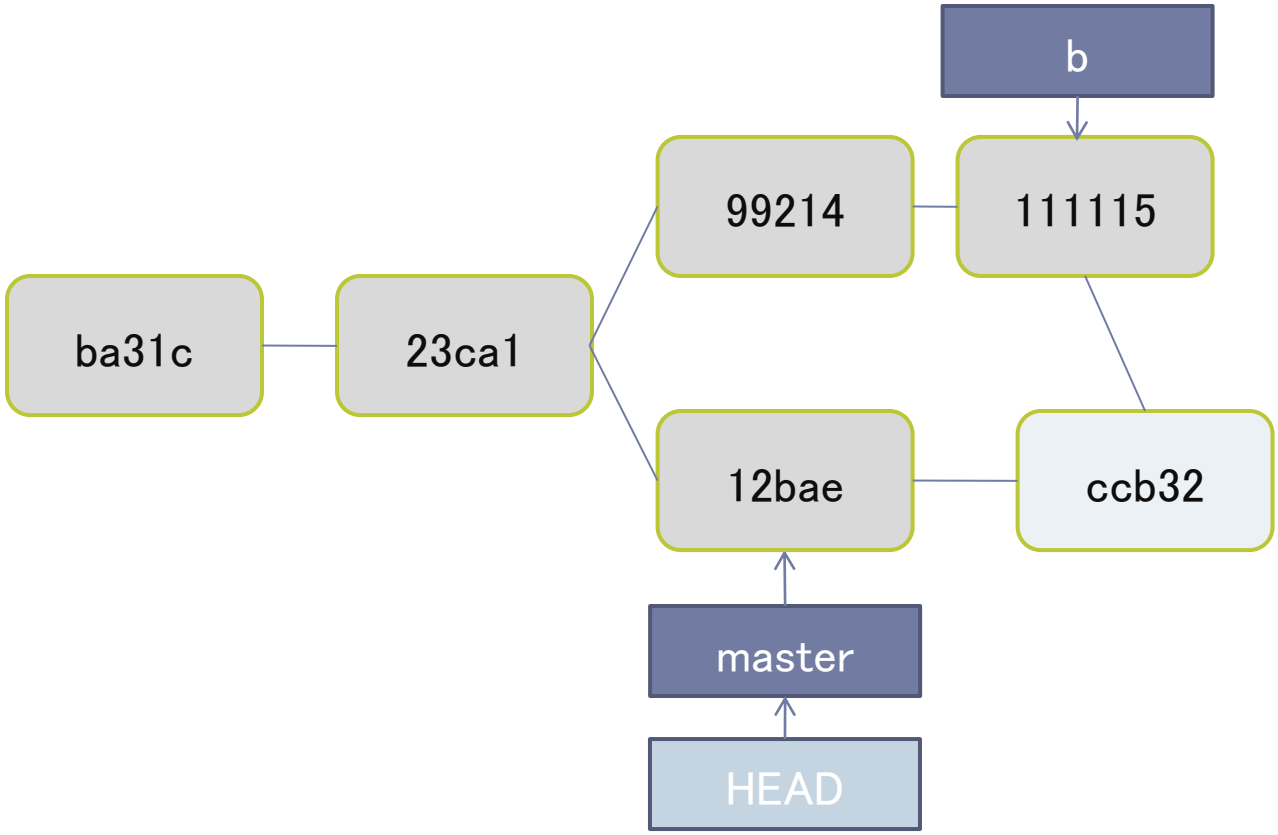
git merge b



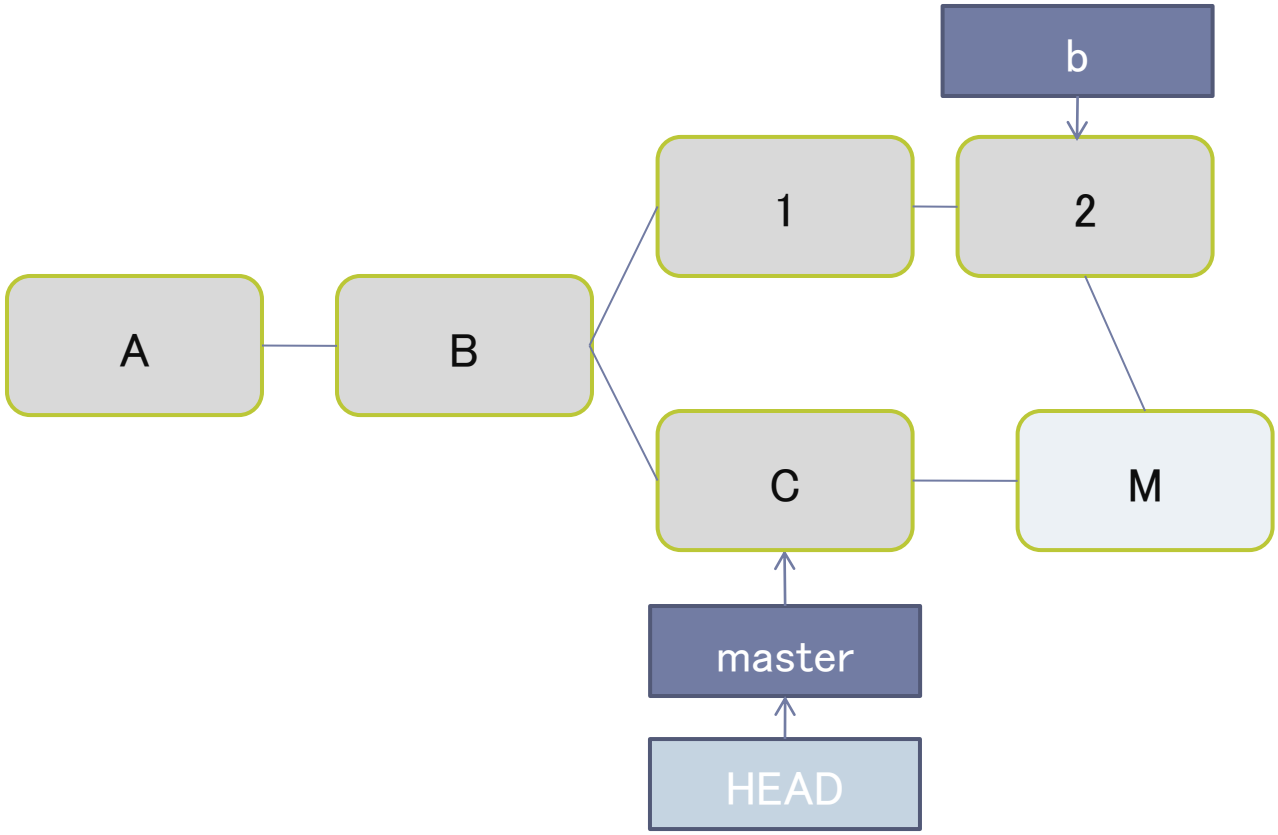
ブランチを移動するだけでマージ完了

→fast forward merge

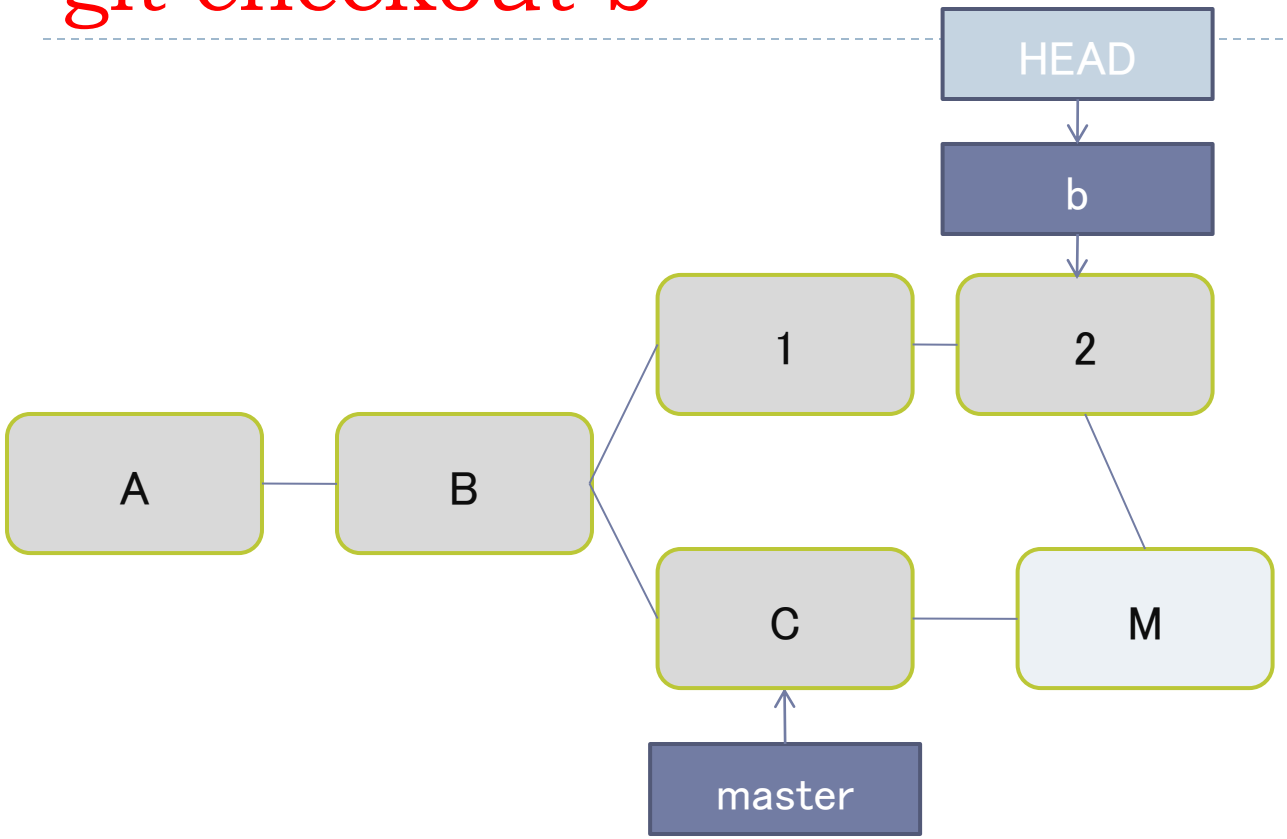
git reset --hard 12bae



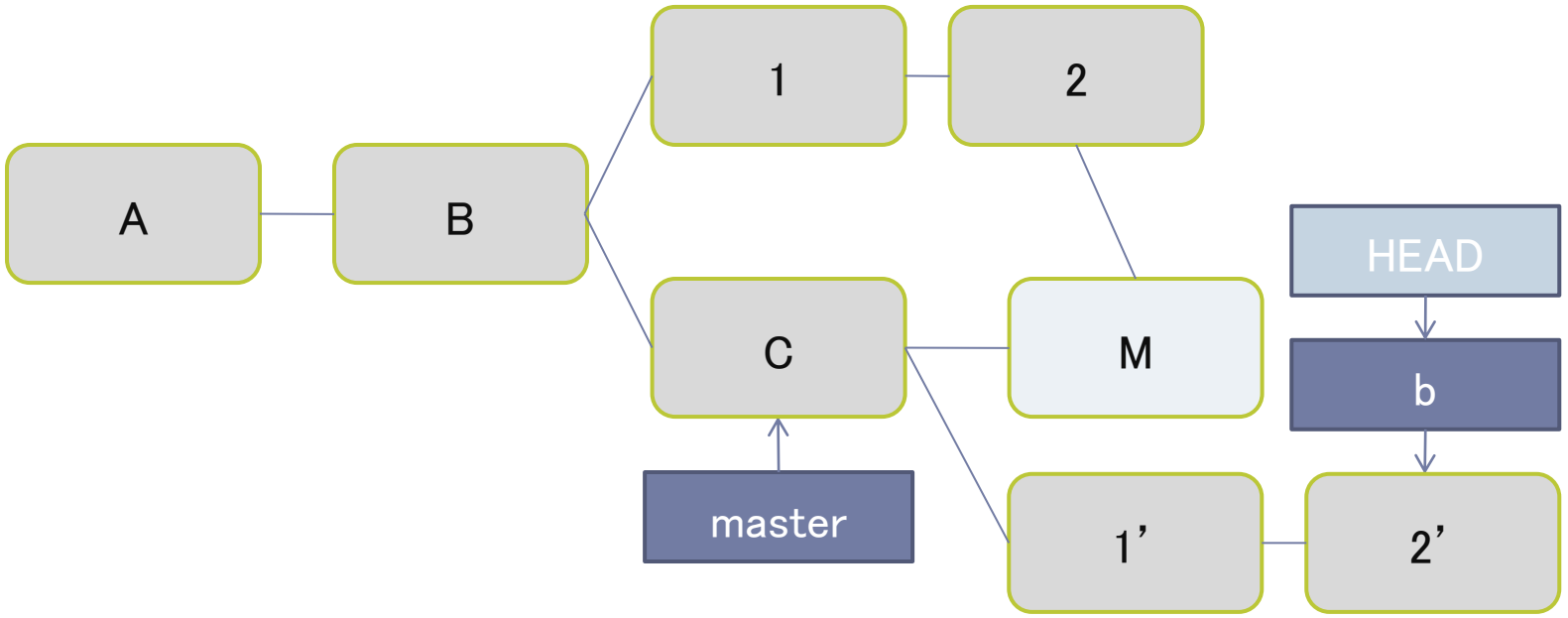
ちょっと表記を変更



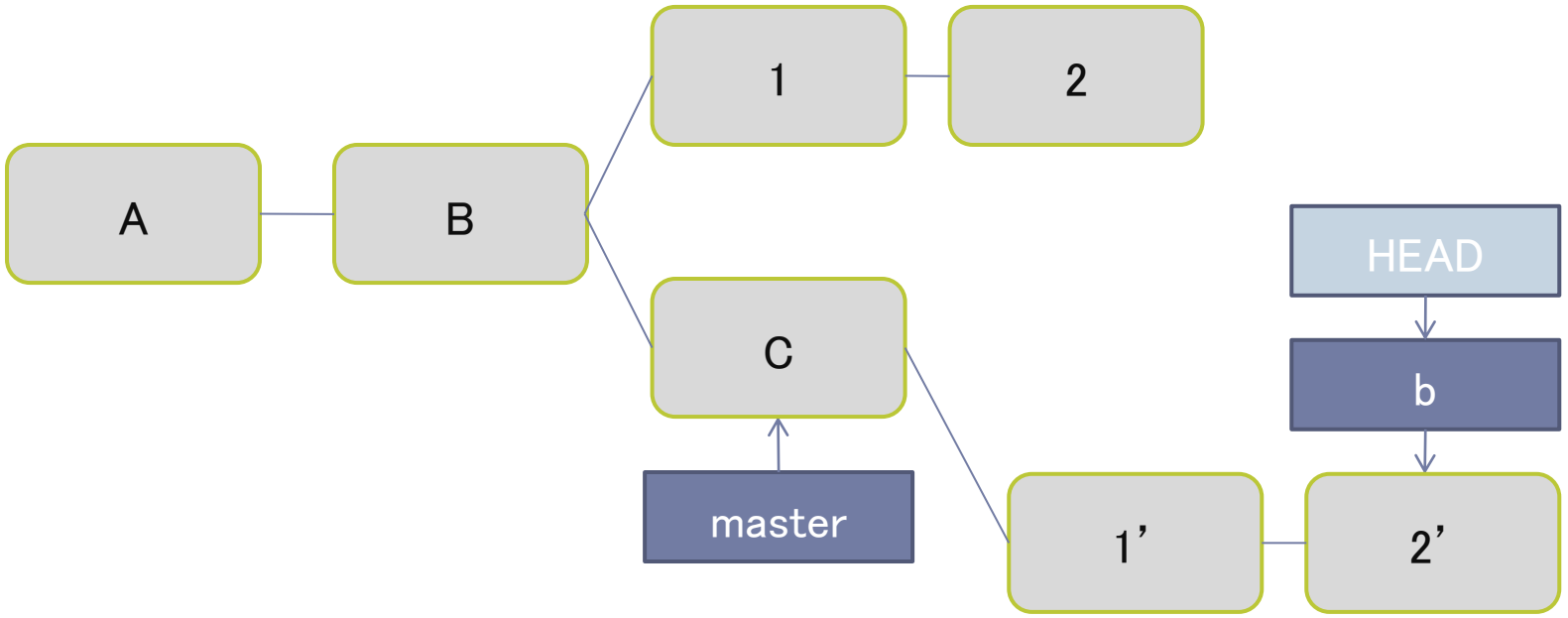
git checkout b



git rebase master

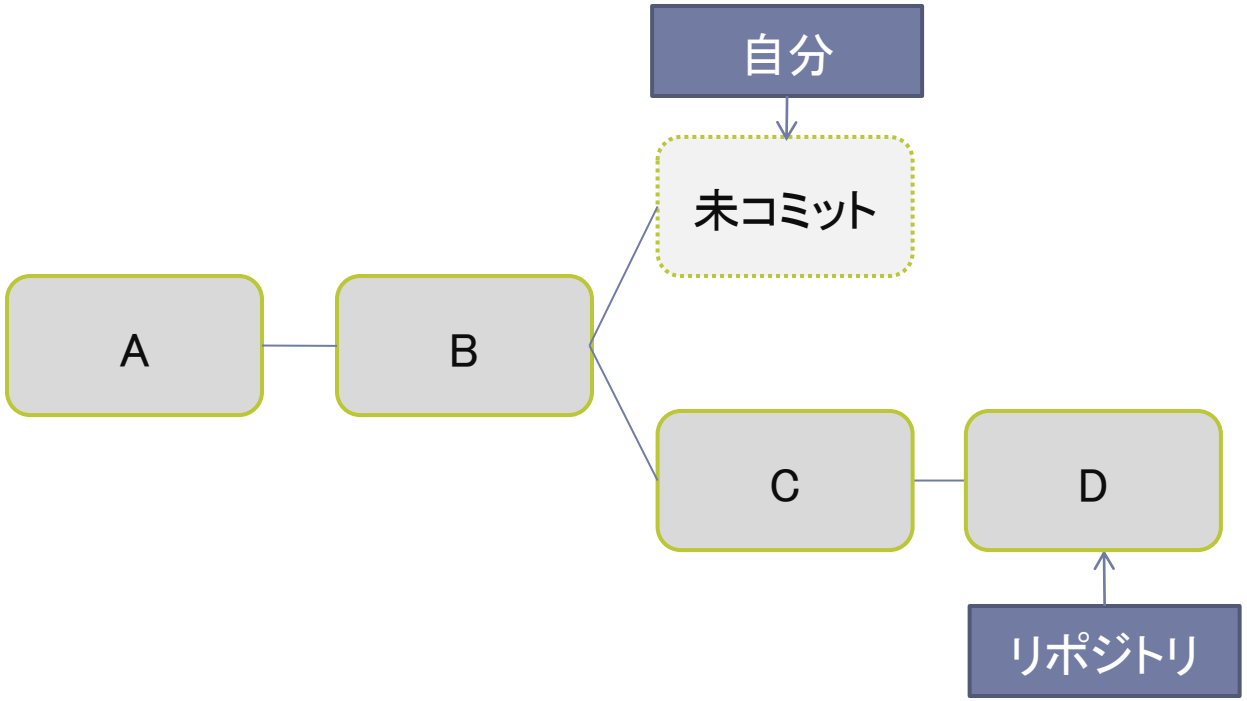


図が見にくいのでマージコミットを消す



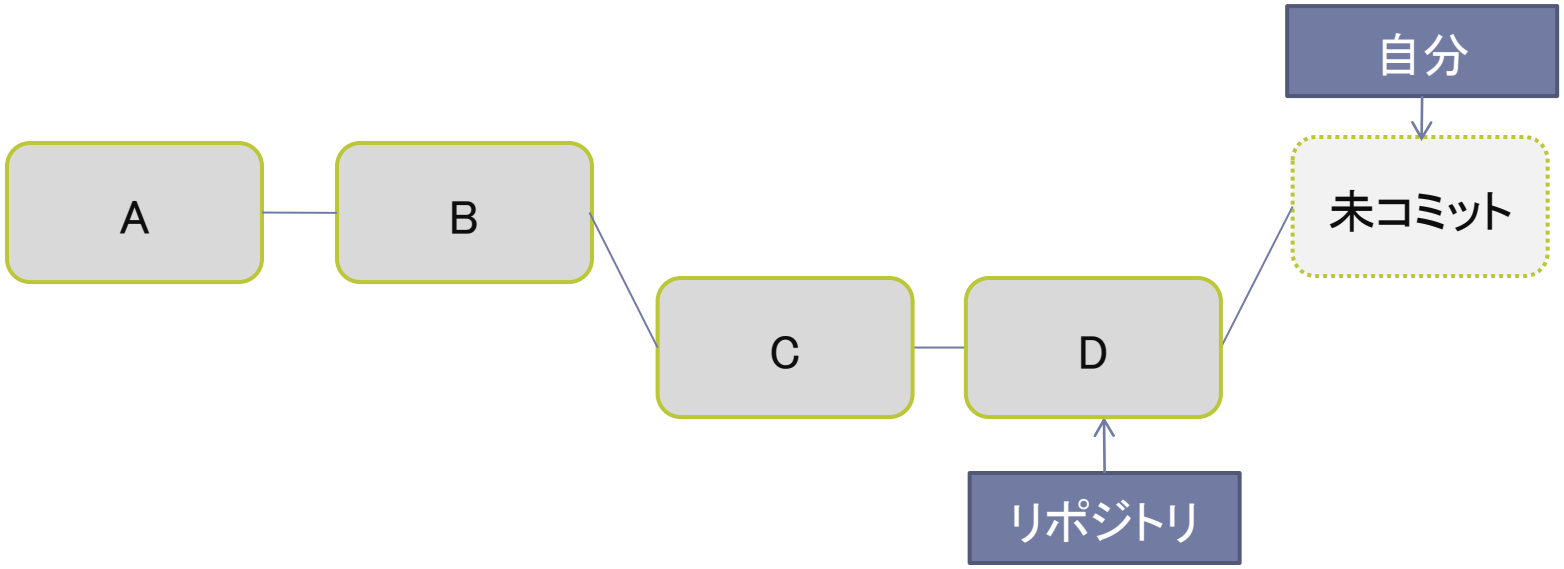
fast forward merge可能！

実はSVNでもやってた



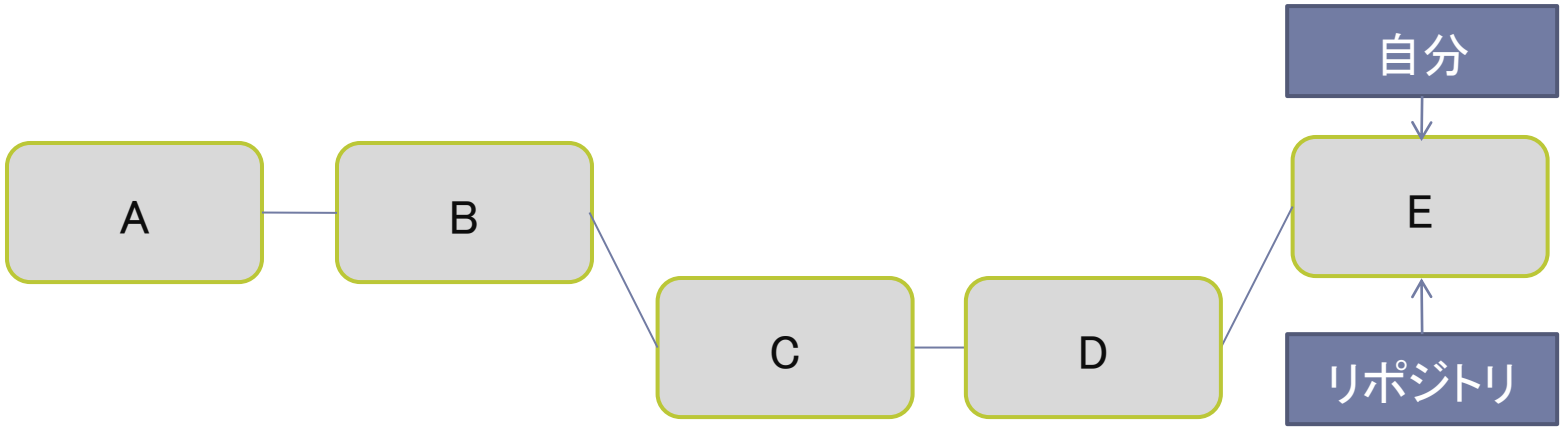
競合が発生してコミットできない・・・

SVNでのUpdate時の競合の解決≒rebase



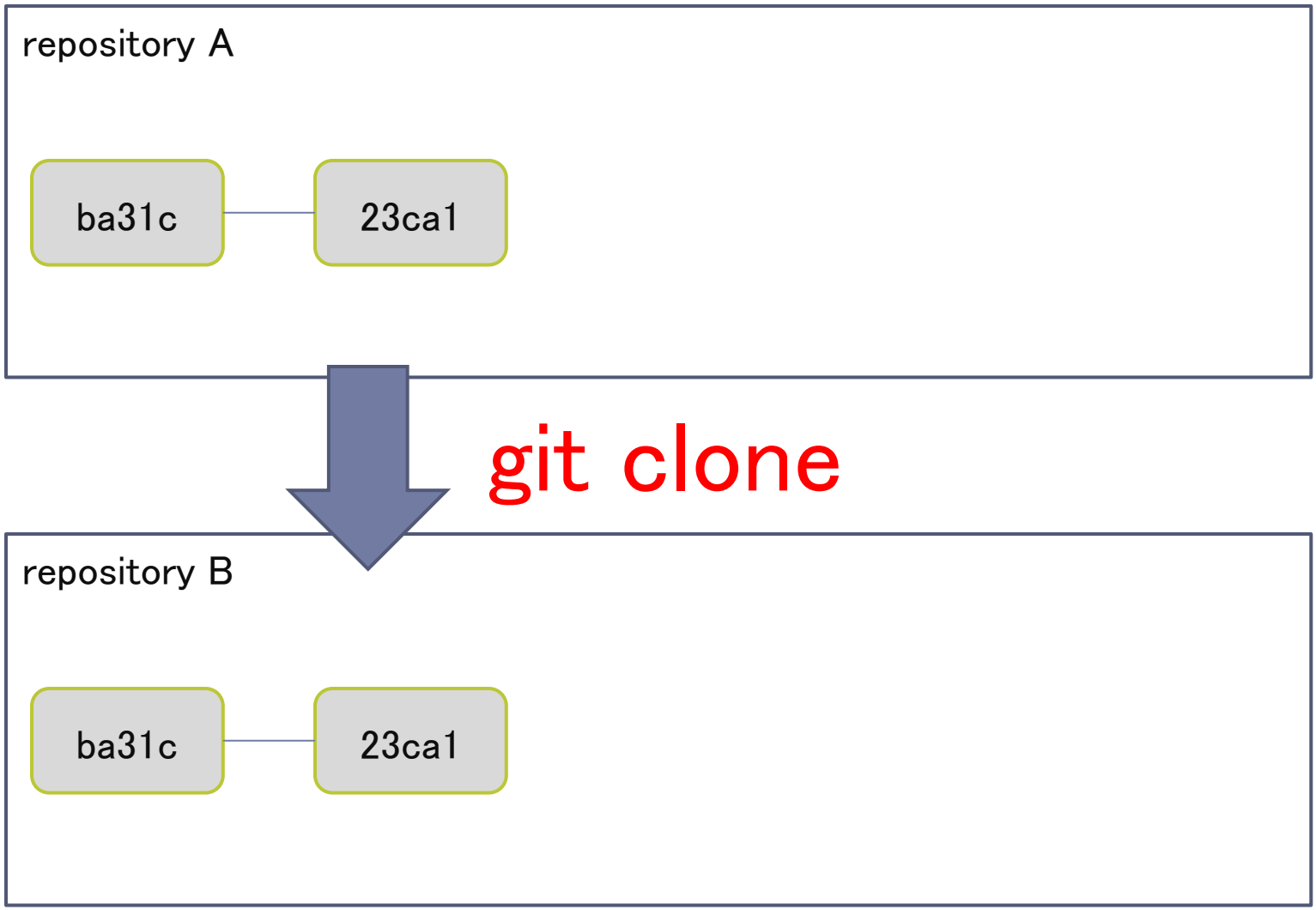
競合を解決・・・ここがrebaseっぽい

SVNでのUpdate時の競合の解決≒rebase

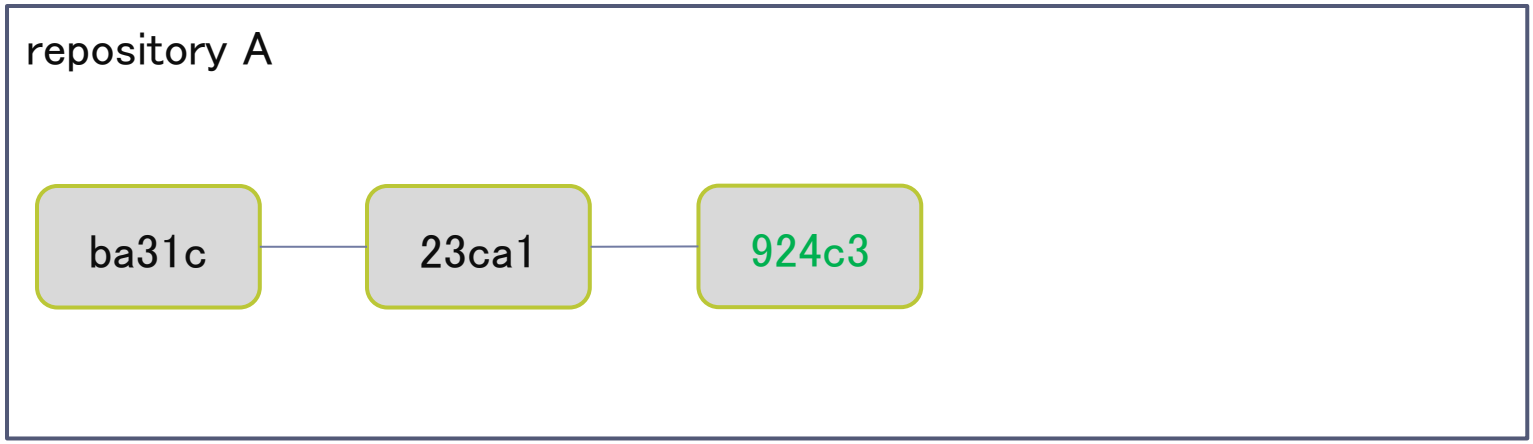


そしてコミット！ただしこの作業はやり直し不可

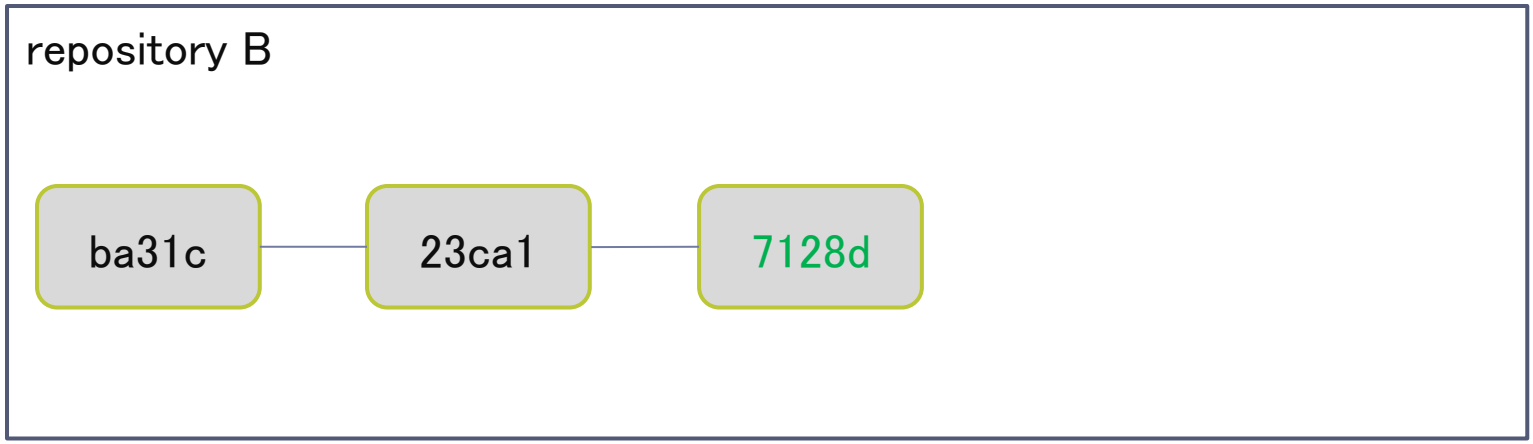
分散リポジトリの例


repository A
ba31c
23ca1
git clone
repository B
ba31c
23ca1

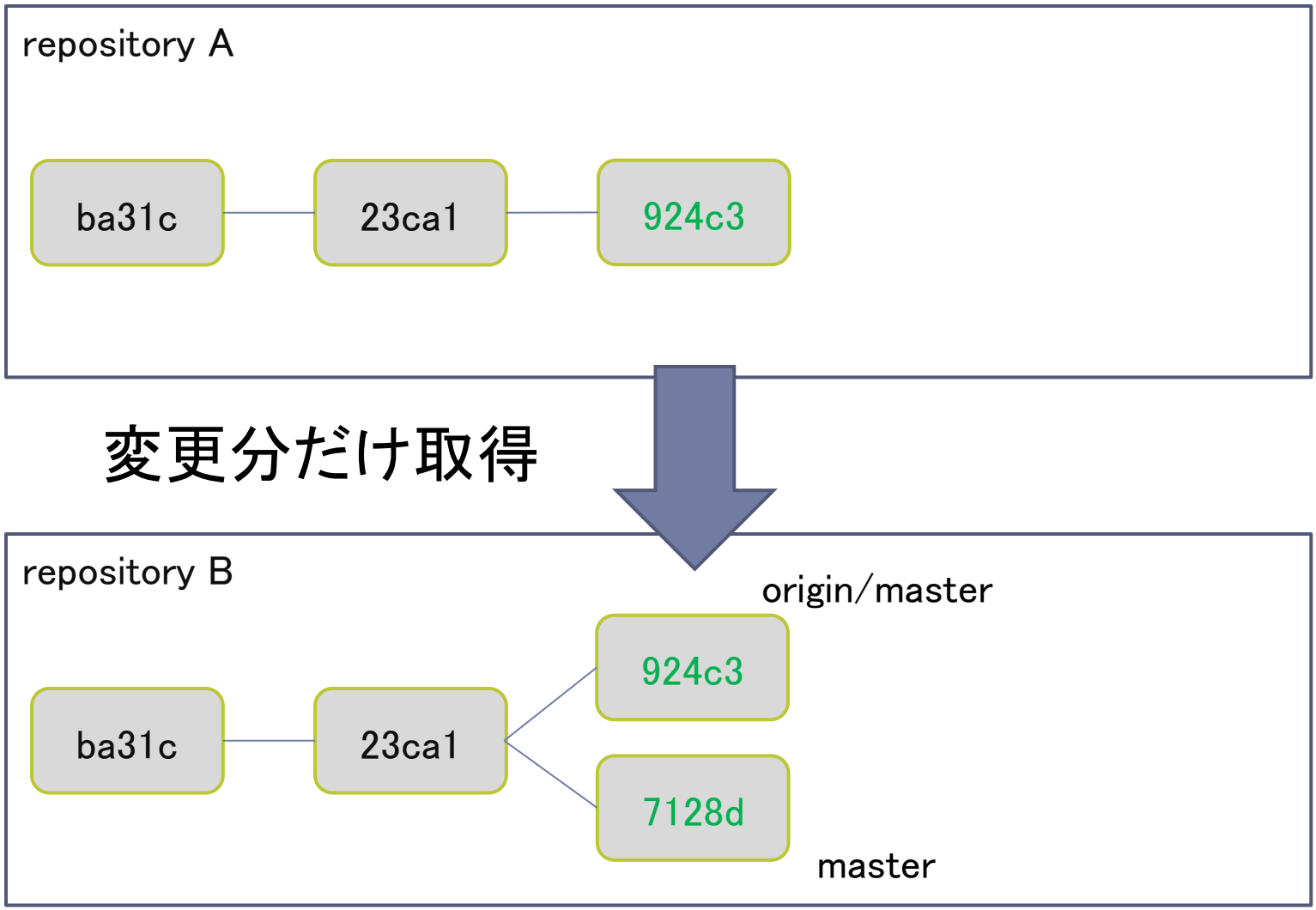
分散リポジトリの例



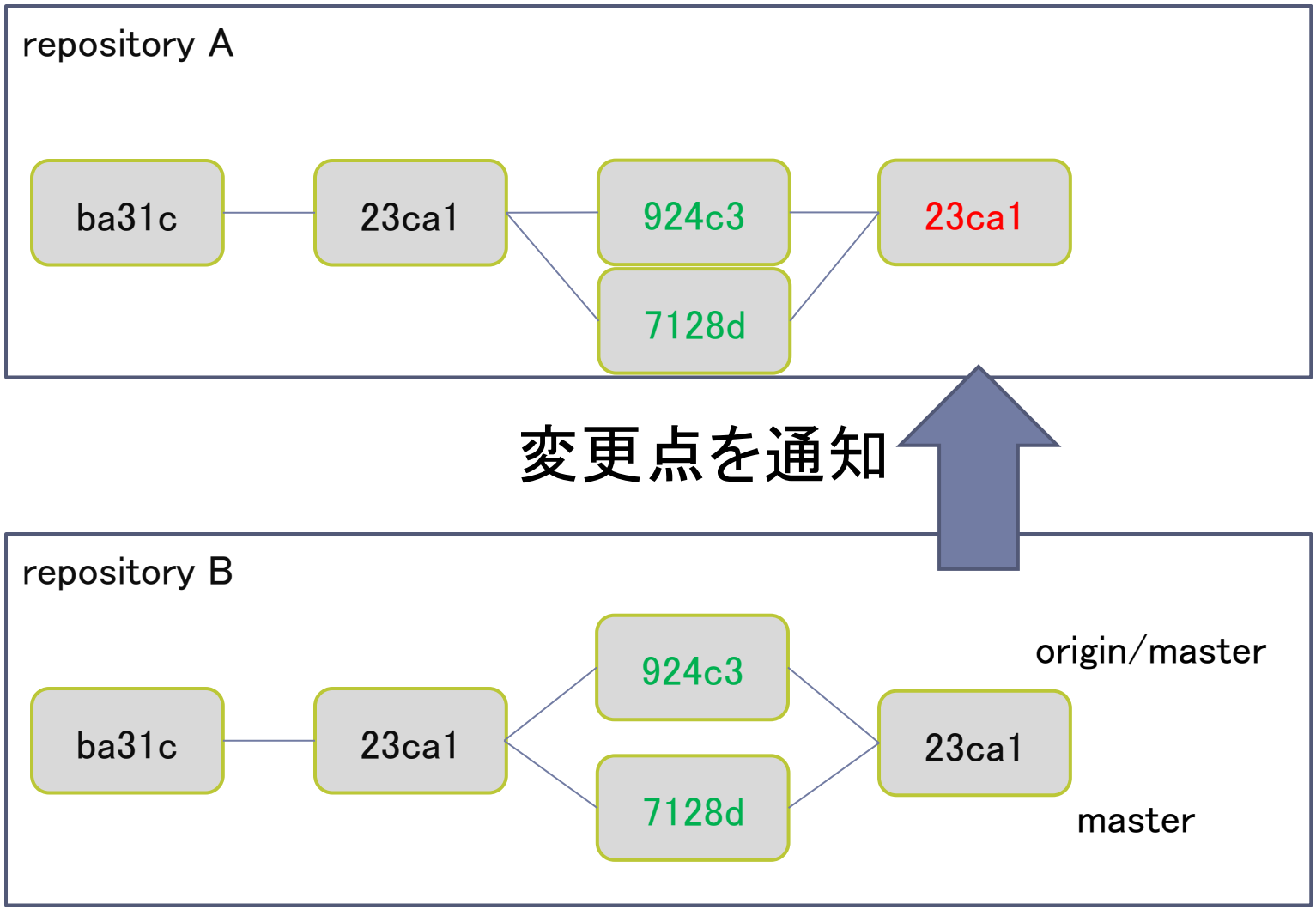
リポジトリはバラバラに成長するが、区別できる



git fetch origin



git merge origin/master; git push



さいごに

- オブジェクトを理解し、
- ブランチの考え方を理解し、
- コミットグラフを頭に思い浮かべることができれば勝てる

そうすればresetとかrebaseも理解しやすいよ！

みんなでreset/rebaseしまくろう！