本コースの特徴

Gitの知識を長期記憶に焼き付けて忘れないようにする

1. **「300枚の図解スライド」**で動作をイメージできる
2. **「100問以上」**のフラッシュカード式の演習問題で記憶にこびりつく
3. **「暗記がいらなくなるレベルの理解」**を目指した、丁寧でごまかしのない解説

Gitとは？

世界でいちばん使われている**バージョン管理システム**

世界中ほとんどの開発現場に導入されているので、

Gitを知らずに開発チームに参加することはほぼ不可能

→ **Gitはすべてのエンジニアの必須スキル！**

Gitを使いこなすのはむずかしい 🥺

- コマンド操作が**直感的じゃない**
- コマンドの裏で起きていることが**イメージしづらい**
- ブランチなどの**仕組みが複雑**で理解できない
- Gitに加えて**GitHubも使うから混乱する**

間違った理解で書かれているWeb記事もとても多い！

本コースの特徴

Gitの知識を長期記憶に焼き付けて忘れないようにする

1. **「300枚の図解スライド」**で動作をイメージできる
2. **「100問以上」**のフラッシュカード式の演習問題で記憶にこびりつく
3. **「暗記がいらなくなるレベルの理解」**を目指した、丁寧でごまかしのない解説

300枚の図解スライドで動作をイメージできる

Gitの壁「何が起こっているかイメージできない」

→ コマンドの動作イメージがわくように、コマンドの動きを**300枚の図解スライド**で解説

イメージで理解して、**自信を持ってGitを操作**しよう

100問以上のフラッシュカード式問題集

長期記憶に残す最強の学習方法、「**検索練習**」の実践

すべての内容をFlash Card式の演習問題として用意

100問以上の演習問題で確実に記憶に焼き付ける

確実に記憶に残すための取り組み方も掲載

暗記が不要になるレベルの理解へ導く丁寧な解説

とことん深い理解は暗記を不要にする

- なぜこのコマンドが必要なのか？
- いつ使うのか？いつ使わないのか？

を曖昧にせず、ていねいに解説

深い理解で、**暗記の労力を最小化**しましょう

本コースの特徴

Gitの知識を長期記憶に焼き付けて忘れないようにする

1. **「300枚の図解スライド」**で動作をイメージできる
2. **「100問以上」**のフラッシュカード式の演習問題で記憶にこびりつく
3. **「暗記がいらなくなるレベルの理解」**を目指した、丁寧でごまかしのない解説

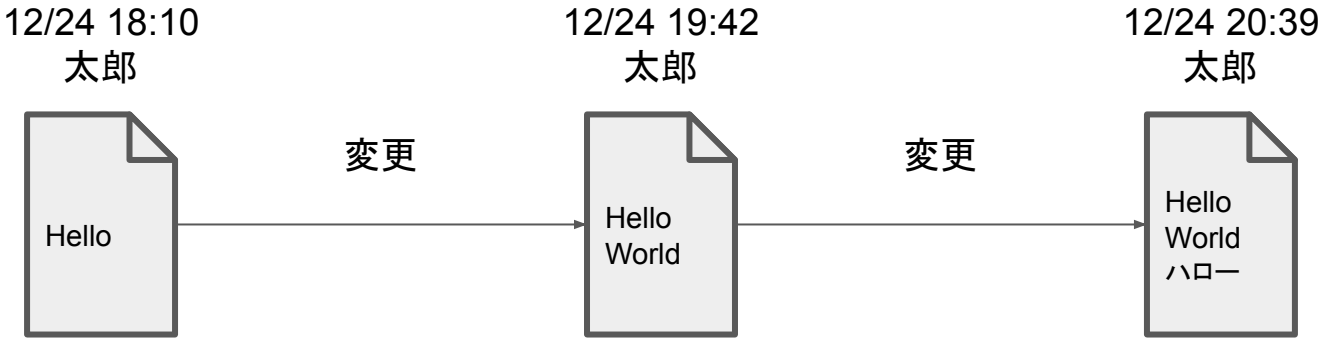
Gitの知識は長く使える「高コスパ」知識

Gitは一度学んでしまえば長い間使える知識です。

付け焼刃の知識ではなく、長期記憶に焼き付けて

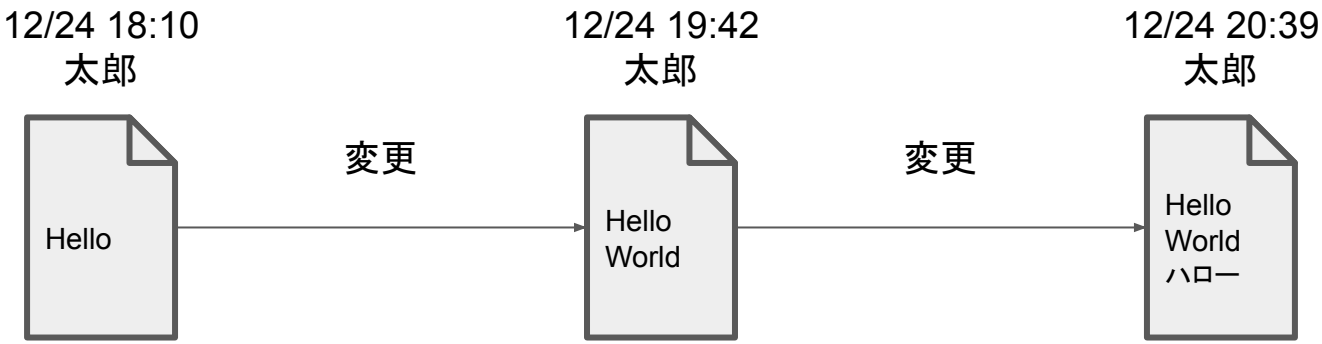
一生モノのスキルを手に入れませんか？

Gitによるバージョン管理のイメージ



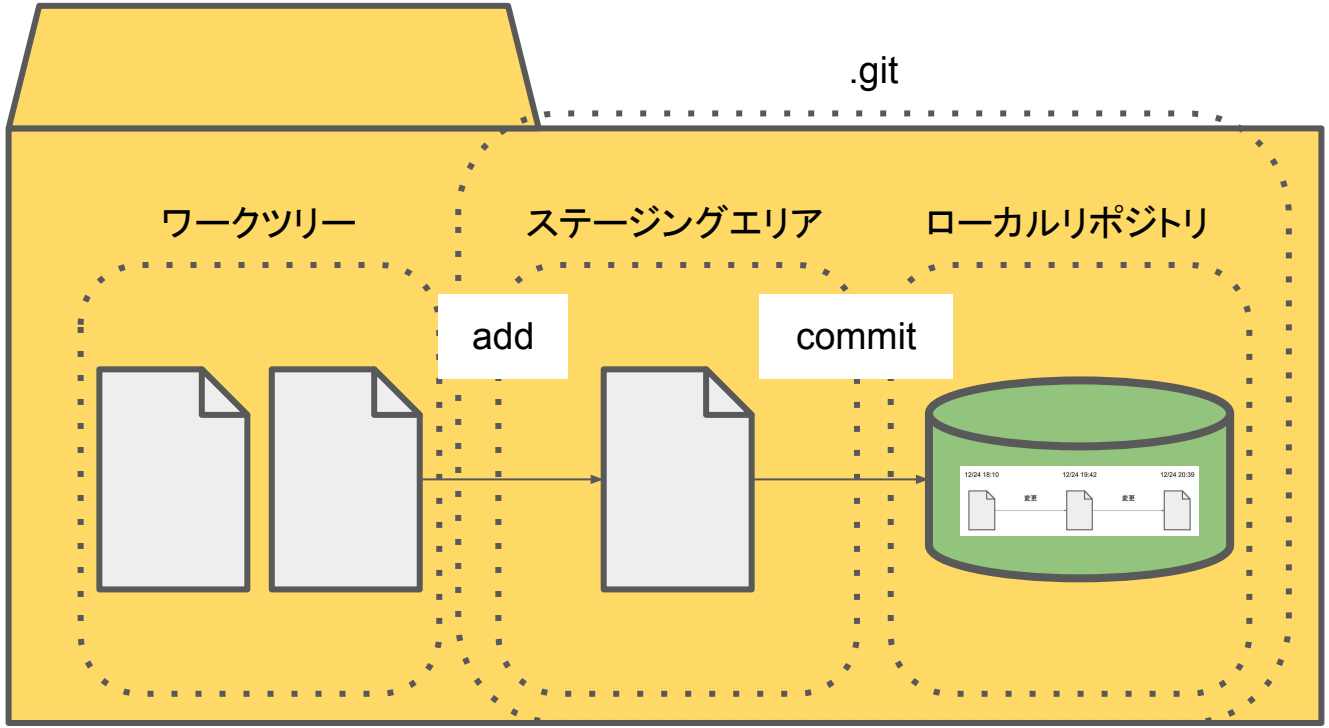
ファイルの変更の履歴を保存しておいて、
いつでもその時点のファイルに戻れるようにする
それぞれは各時点での**スナップショット**

Gitによるバージョン管理のイメージ

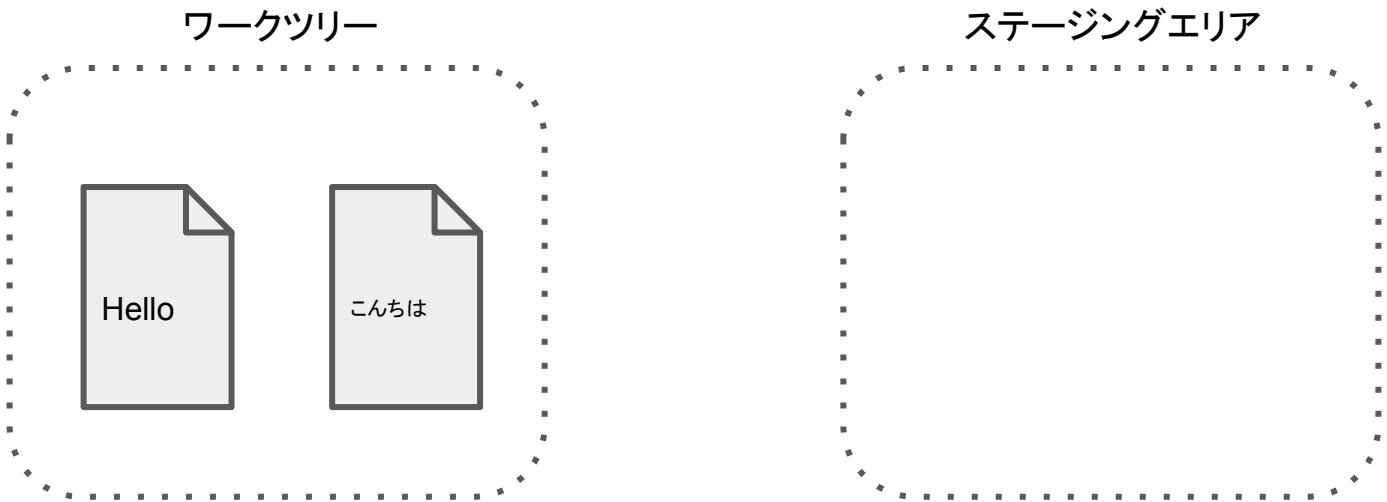


いつ、誰が、どのファイルに、どんな変更を加えたか
わかるようになっている

Gitによるバージョン管理（全体像）

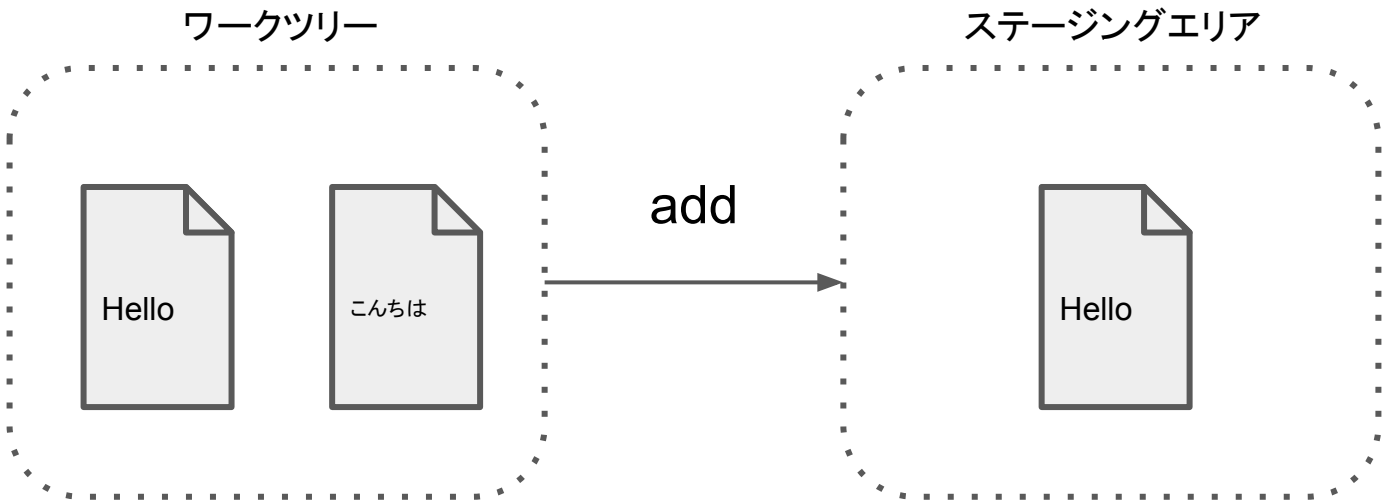


Gitによるバージョン管理



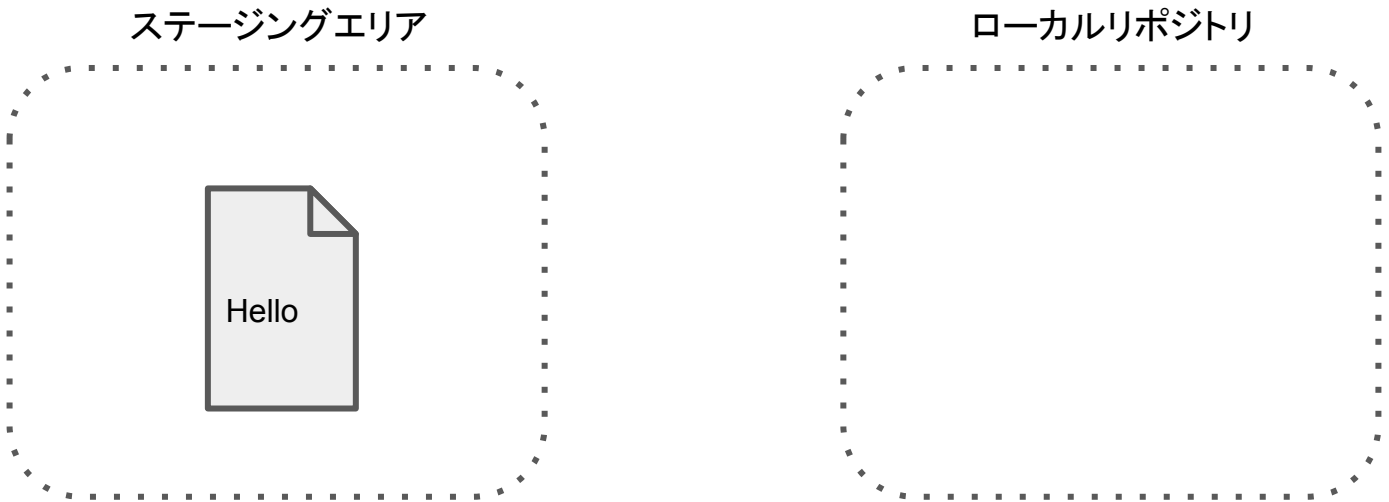
ワークツリーの中でバージョン管理したいファイルを
ステージングエリアにaddする

Gitによるバージョン管理



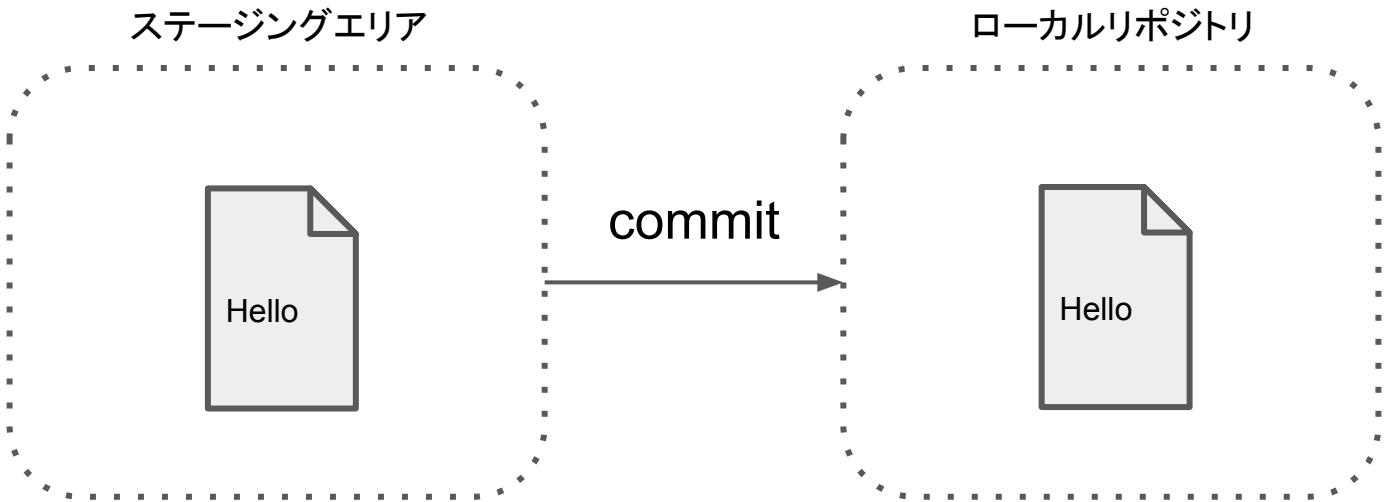
ワークツリーの中でバージョン管理したいファイルを
ステージングエリアにaddする

Gitによるバージョン管理



ステージングエリアのスナップショットを
ローカルリポジトリにコミット（commit）する

Gitによるバージョン管理



ステージングエリアのスナップショットを
ローカルリポジトリにコミット（commit）する

Gitによるバージョン管理

ワークツリー→ステージングエリア→ローカルリポジトリ
という流れでファイルの内容が保存される

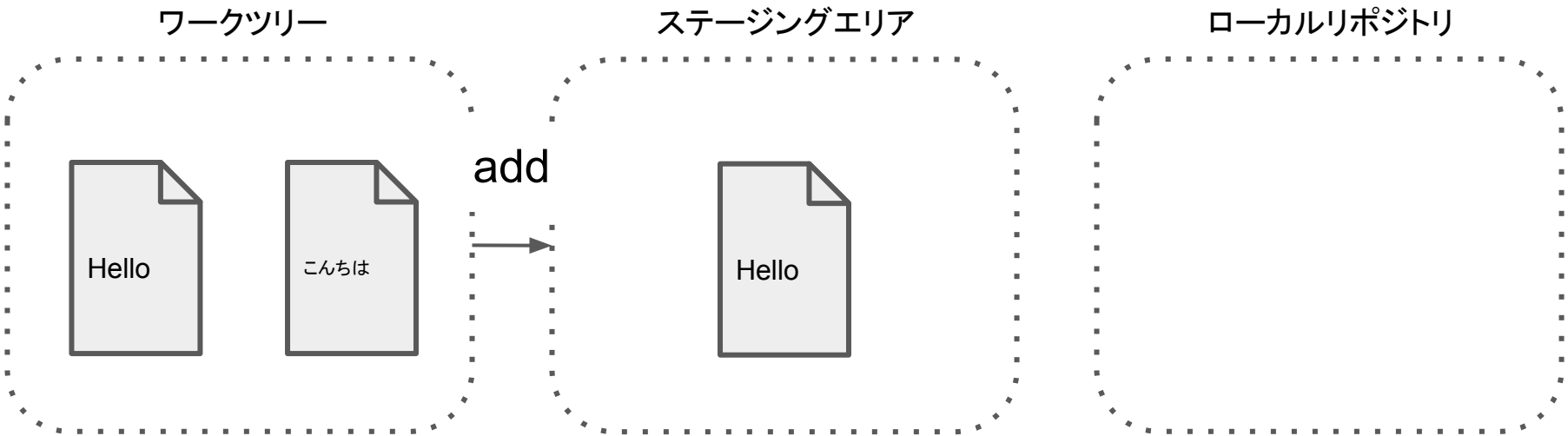
Gitによるバージョン管理

ワークツリー→ステージングエリア→ローカルリポジトリ
という流れでファイルの内容が保存される

Gitによるバージョン管理

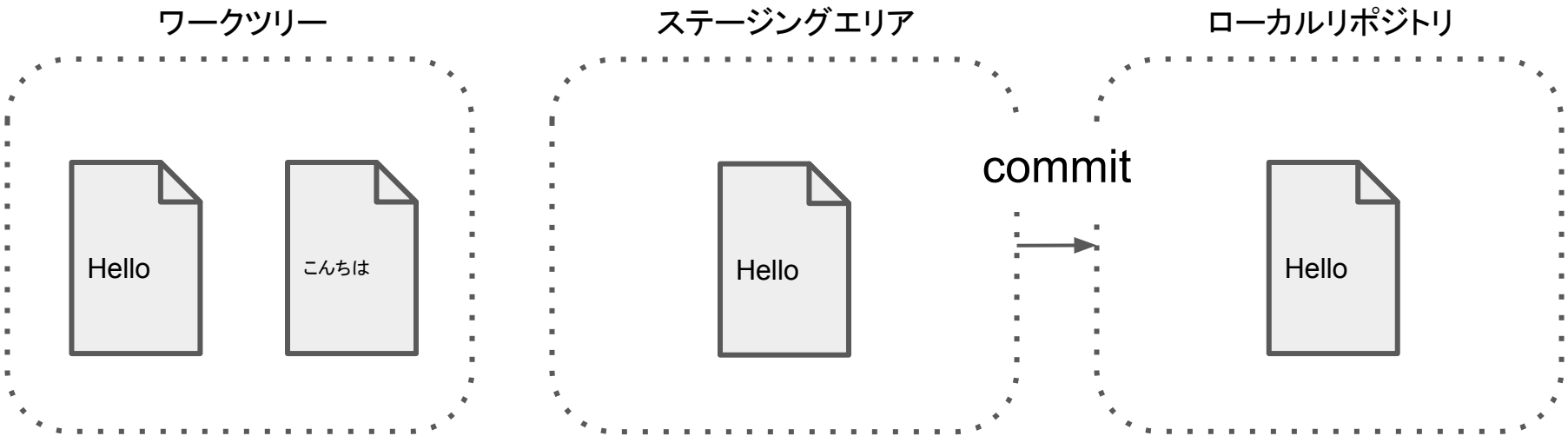
ワークツリー→ステージングエリア→ローカルリポジトリ
という流れでファイルの内容が保存される

Gitによるバージョン管理



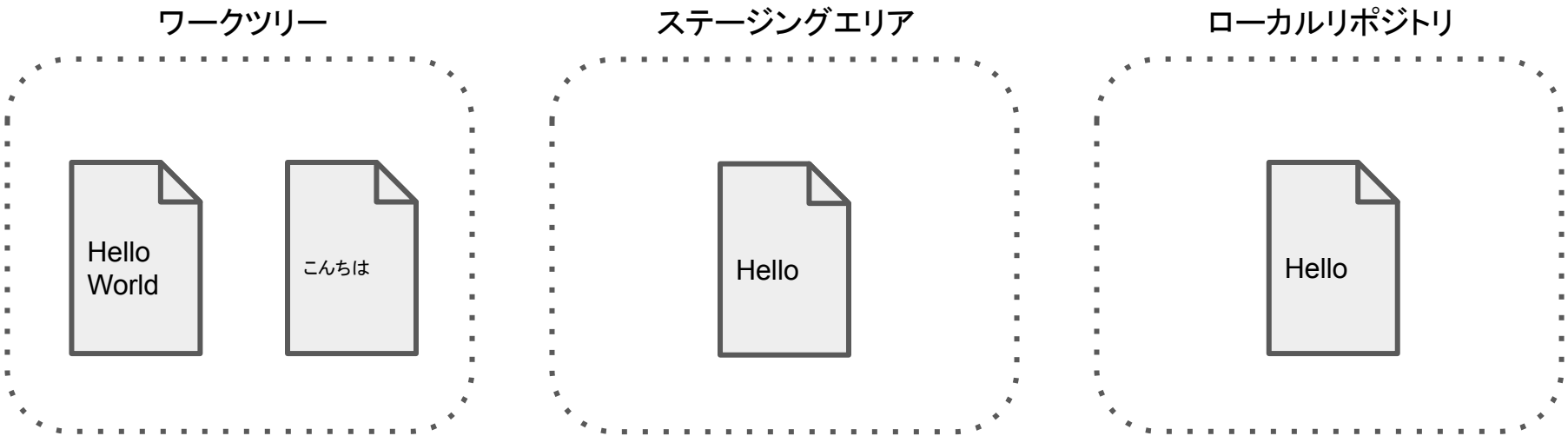
ワークツリー→ステージングエリア→ローカルリポジトリ
という流れでファイルの内容が保存される

Gitによるバージョン管理



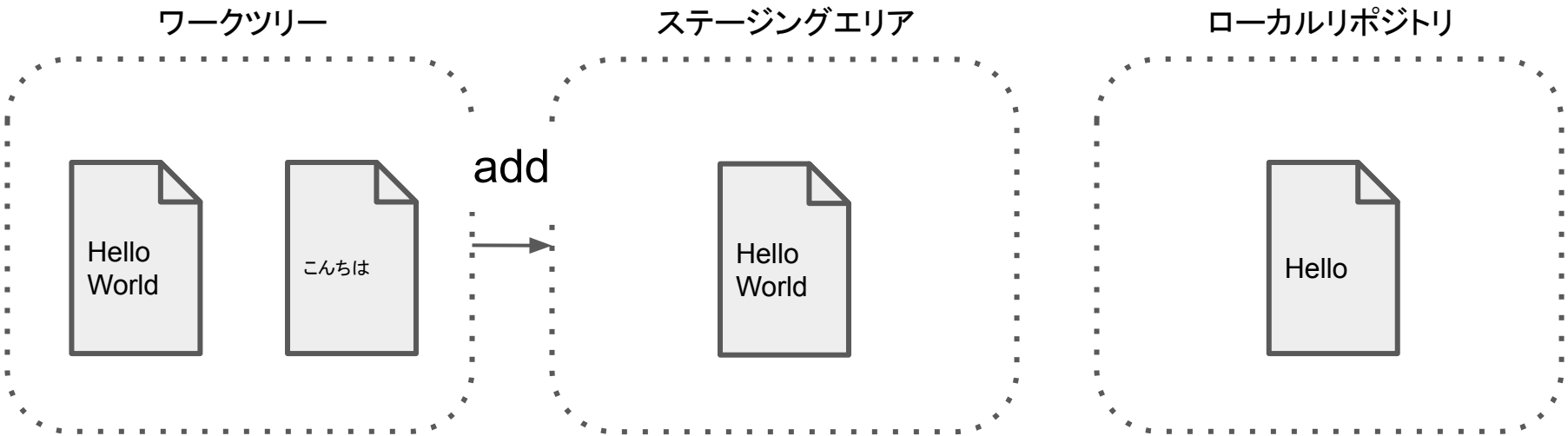
ワークツリー→ステージングエリア→ローカルリポジトリ
という流れでファイルの内容が保存される

Gitによるバージョン管理



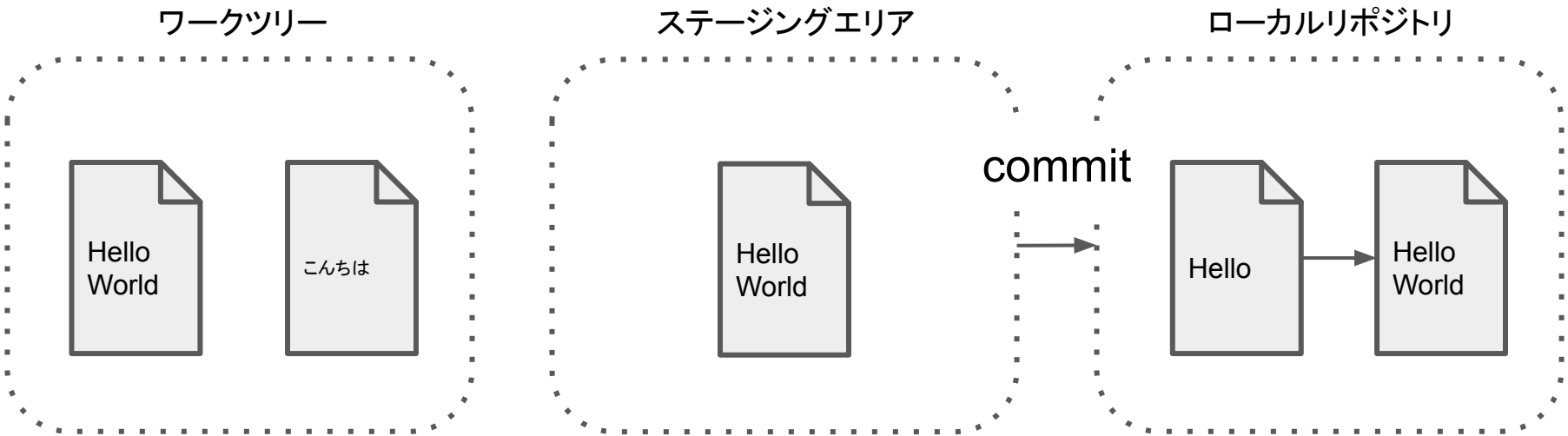
ワークツリー→ステージングエリア→ローカルリポジトリ
という流れでファイルの内容が保存される

Gitによるバージョン管理



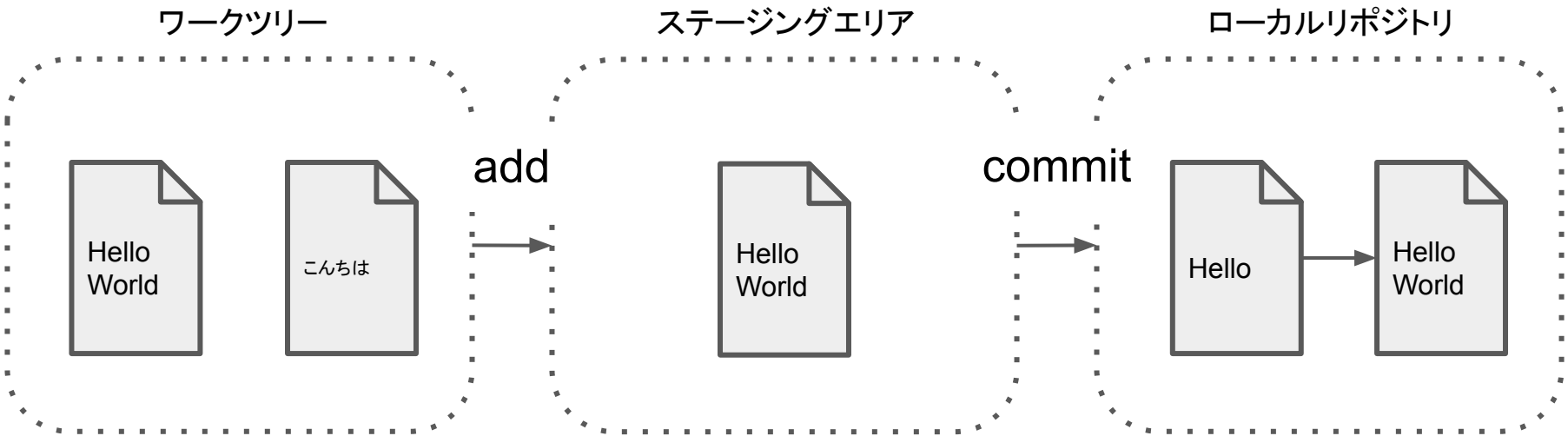
ワークツリー→ステージングエリア→ローカルリポジトリ
という流れでファイルの内容が保存される

Gitによるバージョン管理



ワークツリー→ステージングエリア→ローカルリポジトリ
という流れでファイルの内容が保存される

Gitによるバージョン管理



このような仕組みによって、
ファイルの数を増やさず、複雑な命名ルールも使わずに
任意の時点のファイルの状態に戻ることができる

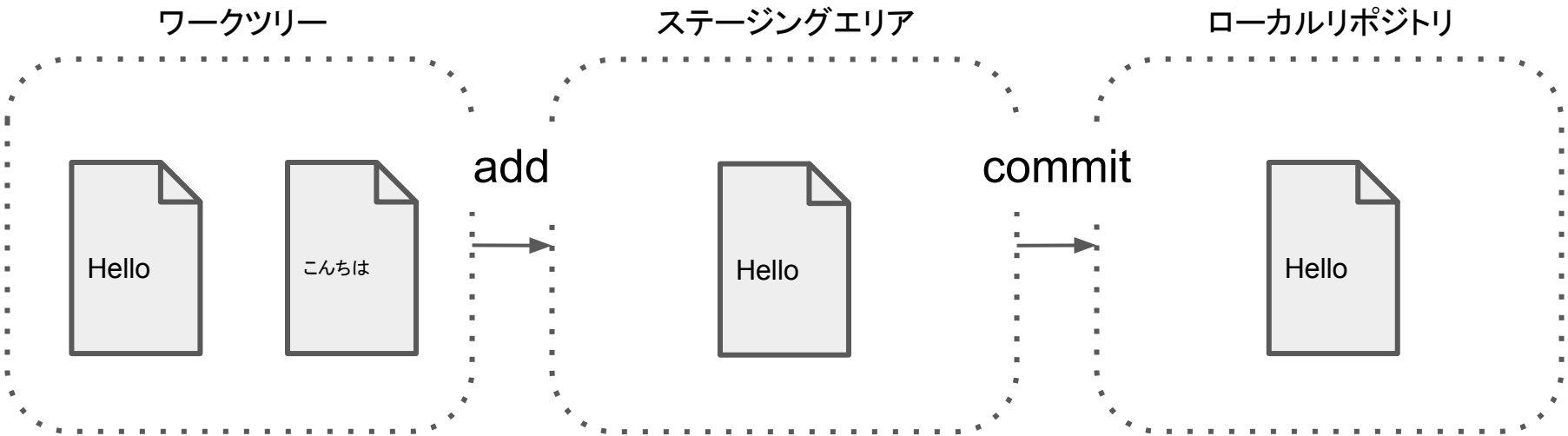
Gitによるバージョン管理

ワークツリー→ステージングエリア→ローカルリポジトリ
という流れでファイルの内容が保存される

Gitによるバージョン管理

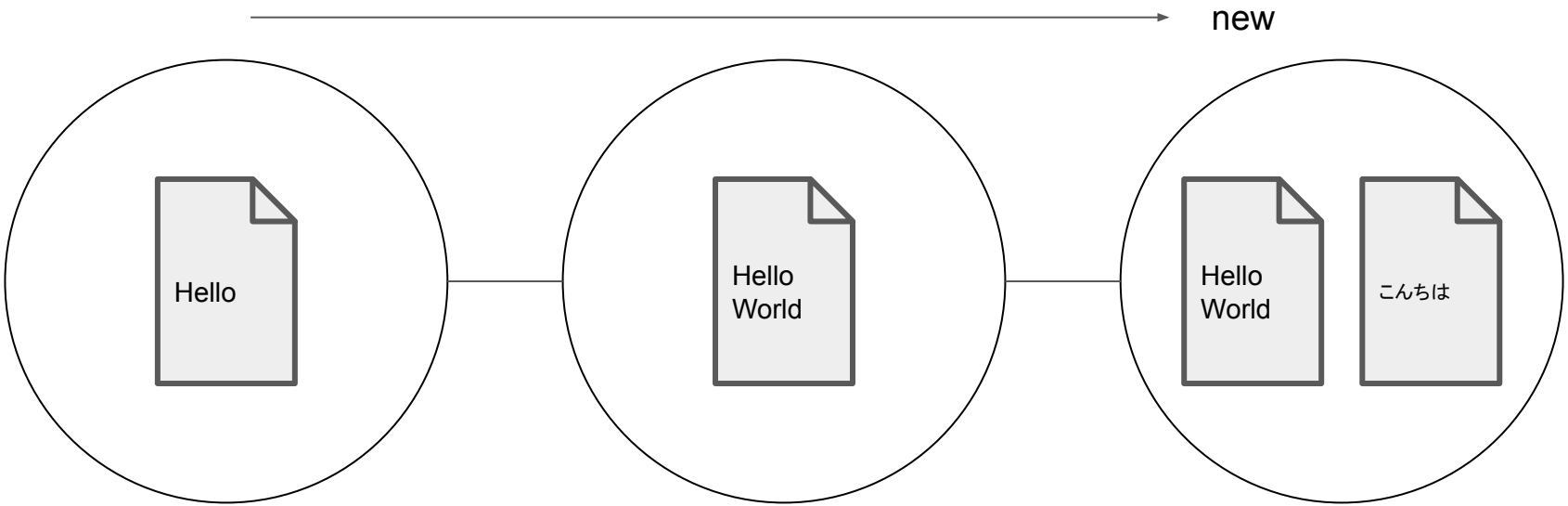
ワークツリー→ステージングエリア→ローカルリポジトリ
という流れでファイルの内容が保存される

Gitによるバージョン管理



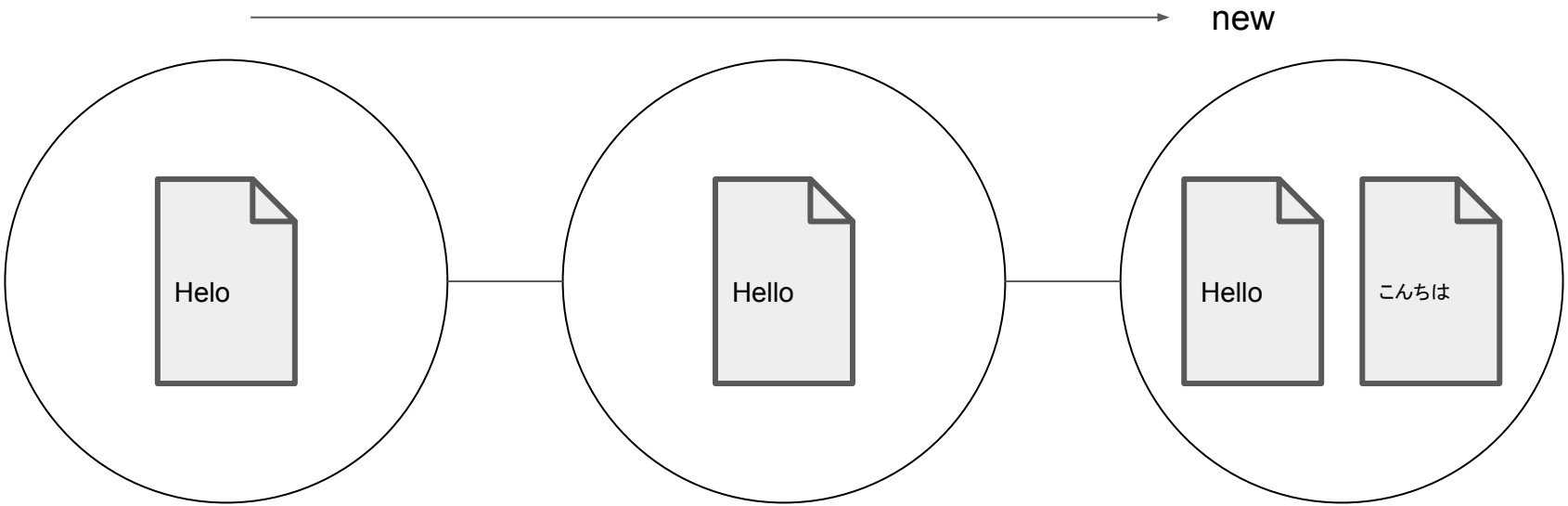
この仕組みでファイルの数を増やさない

１つのコミットに複数のファイルがあることもある



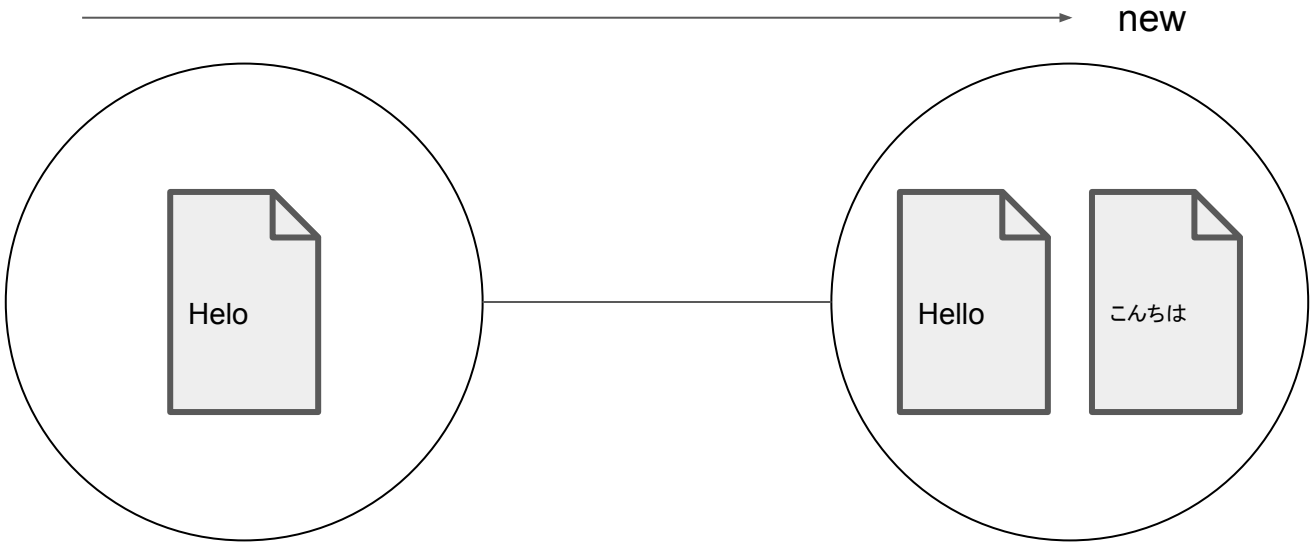
ステージングエリアに複数のファイルがあれば
コミットも複数ファイルになる

どっちの方がコミット履歴がわかりやすい？



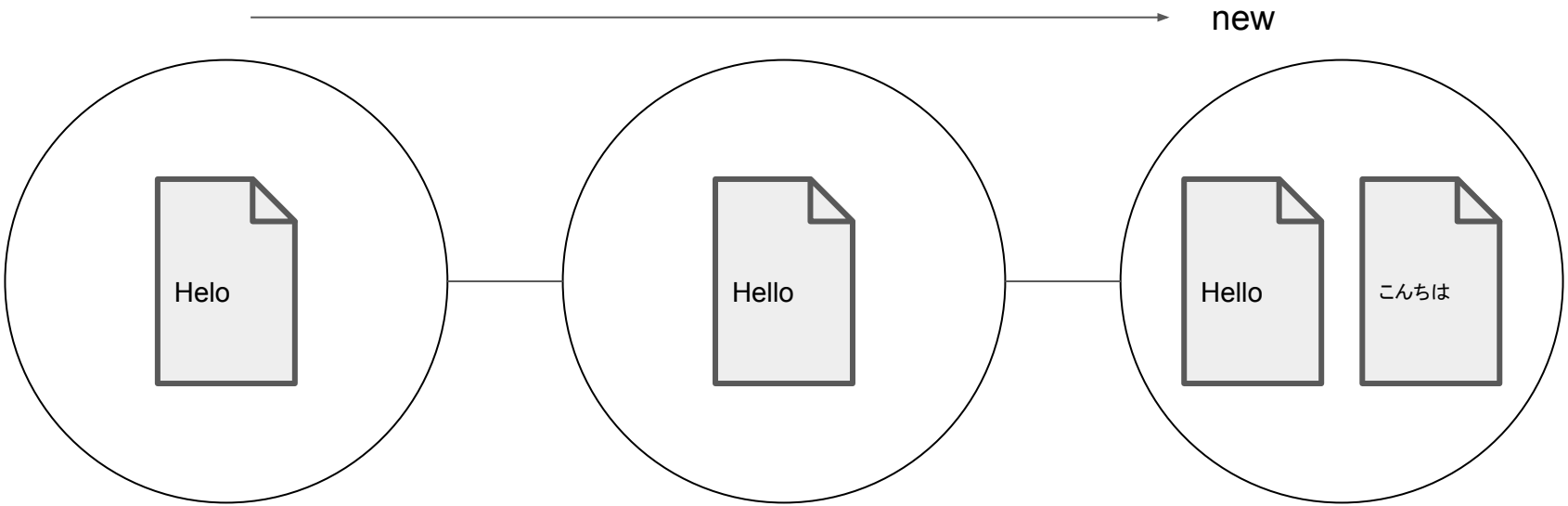
①バグ修正とファイル（機能）追加を分けてコミットする

どっちの方がコミット履歴がわかりやすい？



②バグ修正とファイル（機能）追加をまとめてコミットする

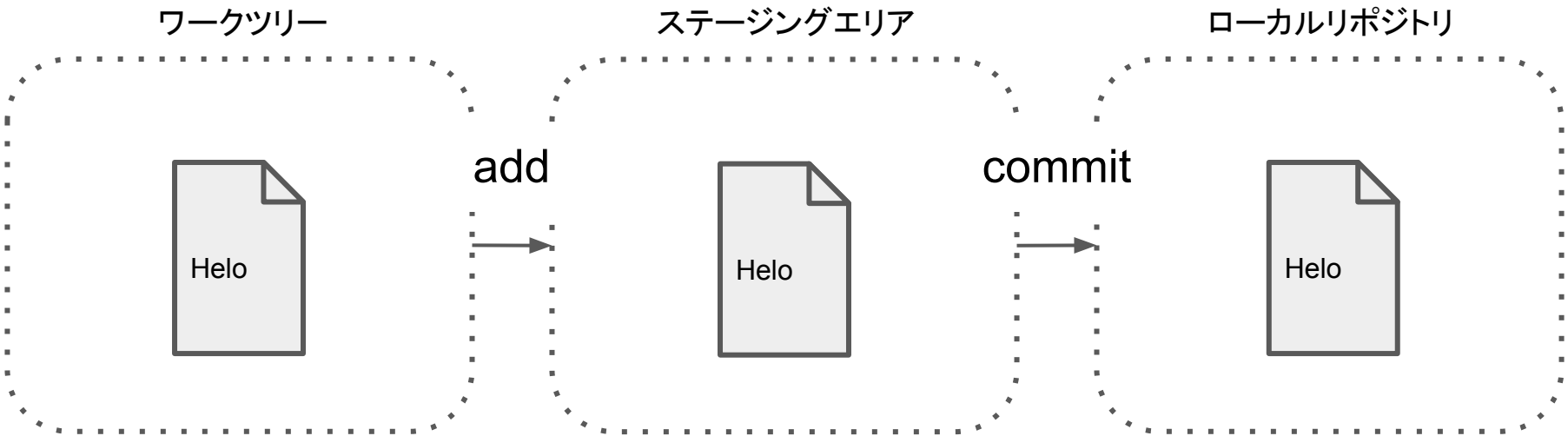
どっちの方がコミット履歴がわかりやすい？



①バグ修正とファイル（機能）追加を分けてコミットする

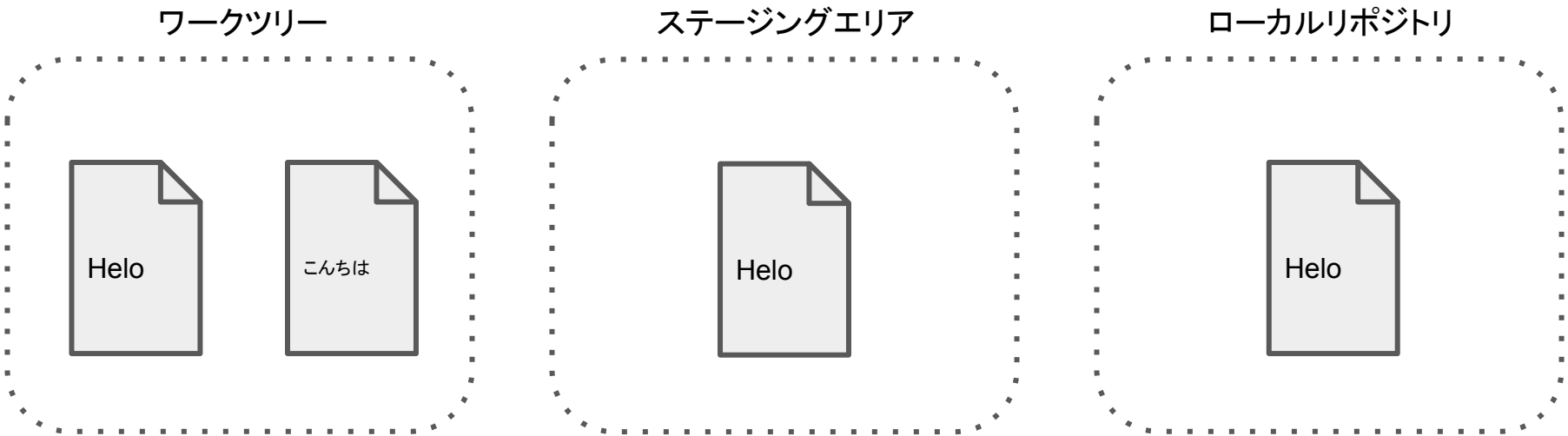
⇒ **コミットは局所化した方がわかりやすい！**

ステージングエリアが必要な理由



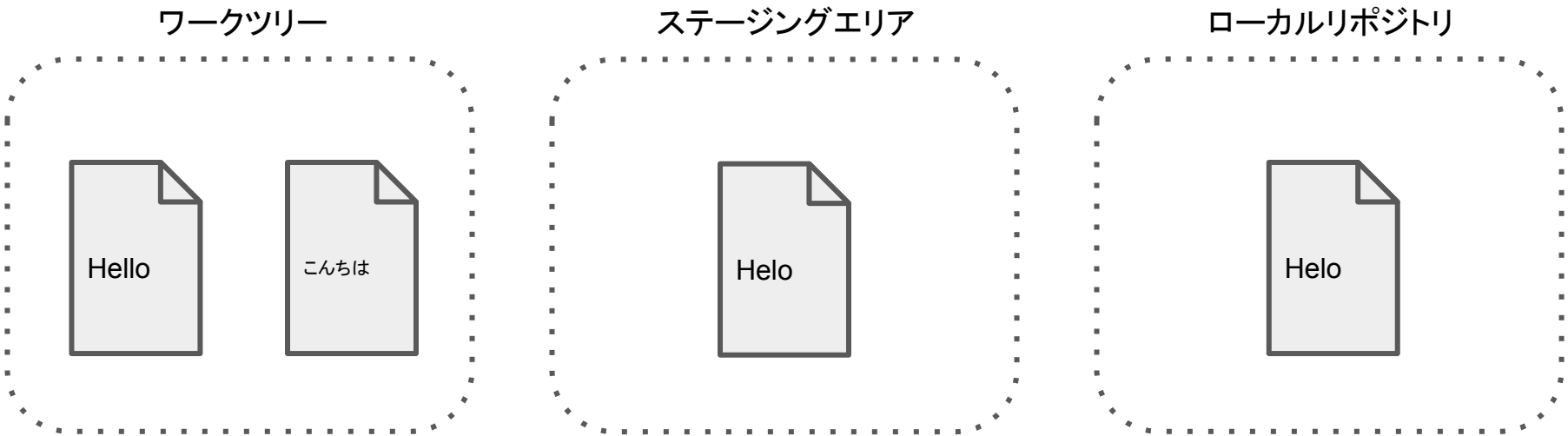
バグを含めてコミットしてしまった
（HelloをHeloと書いてしまっている）

ステージングエリアが必要な理由



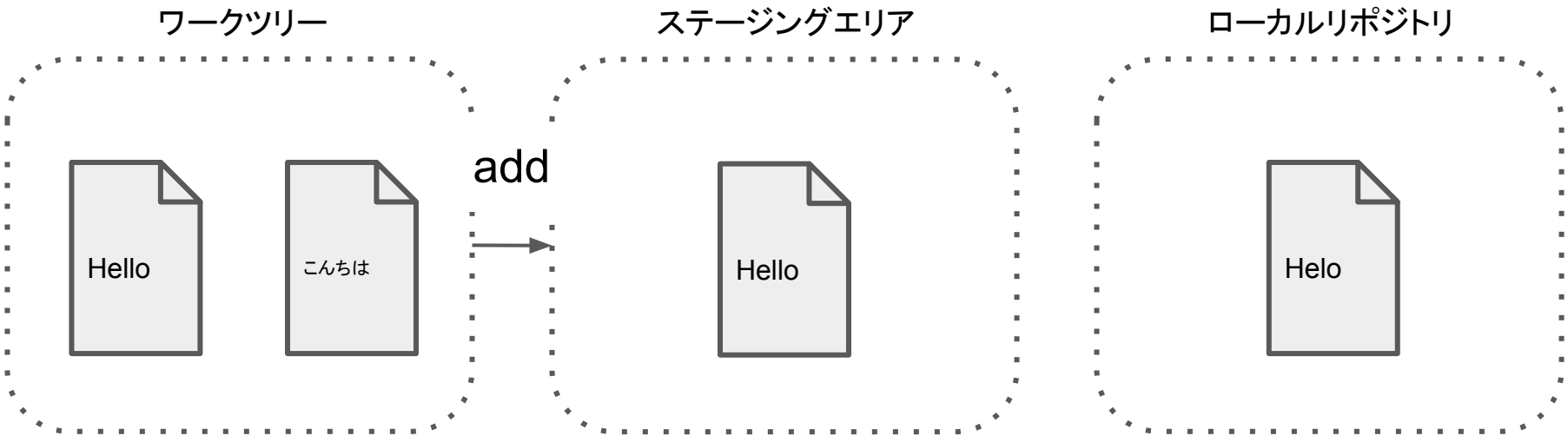
新機能を追加した
（こんちは、のファイル）

ステージングエリアが必要な理由



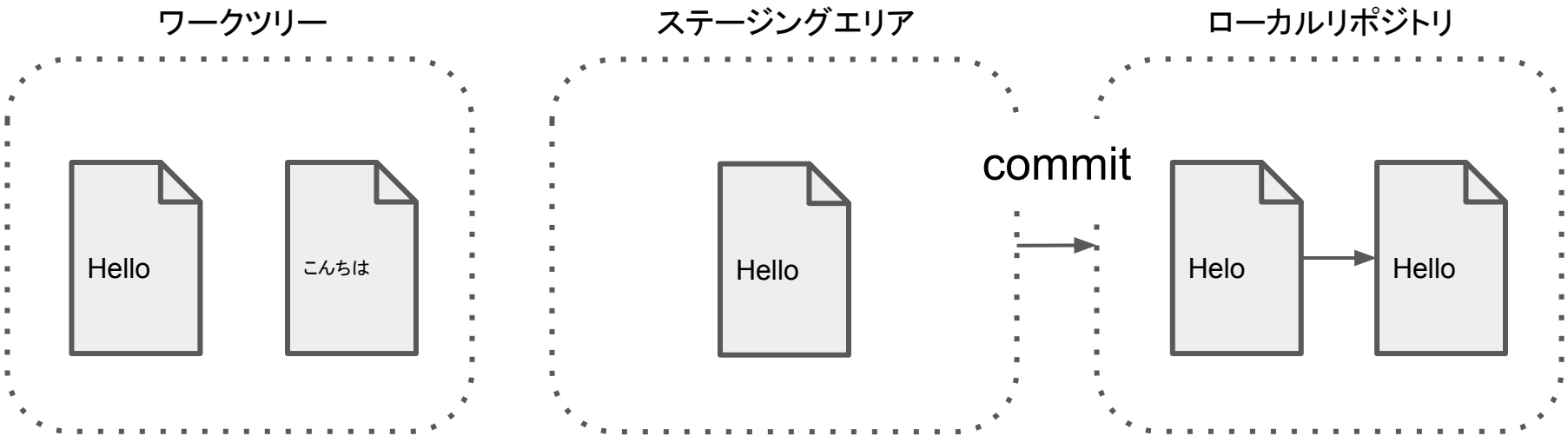
Heloのバグに気づいて修正した

ステージングエリアが必要な理由



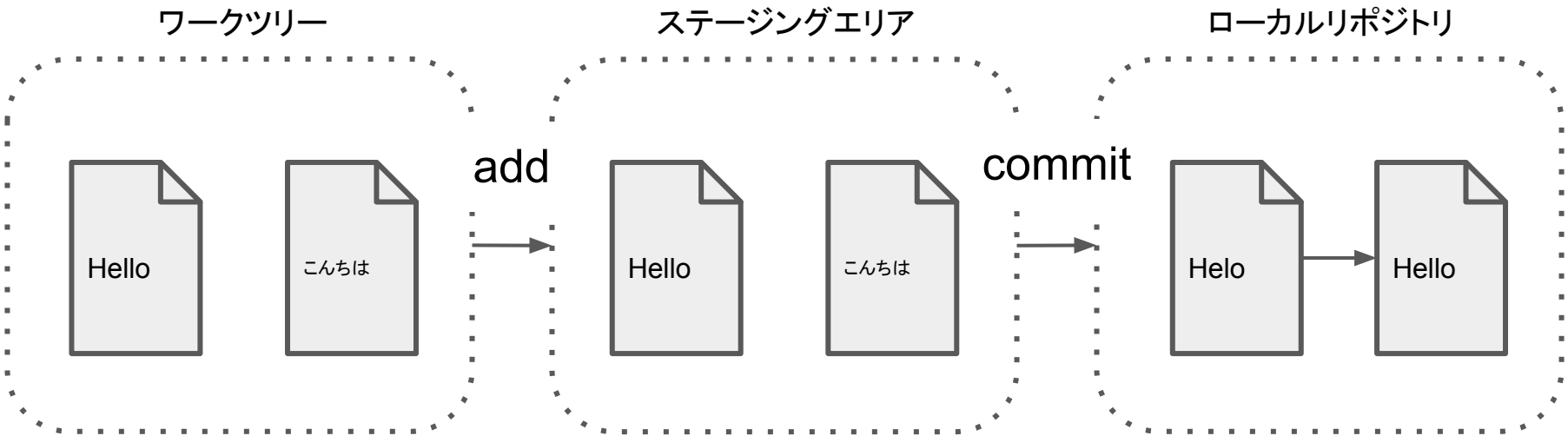
Helloのバグ修正だけをまずコミットすれば
コミット履歴がわかりやすくなる

ステージングエリアが必要な理由



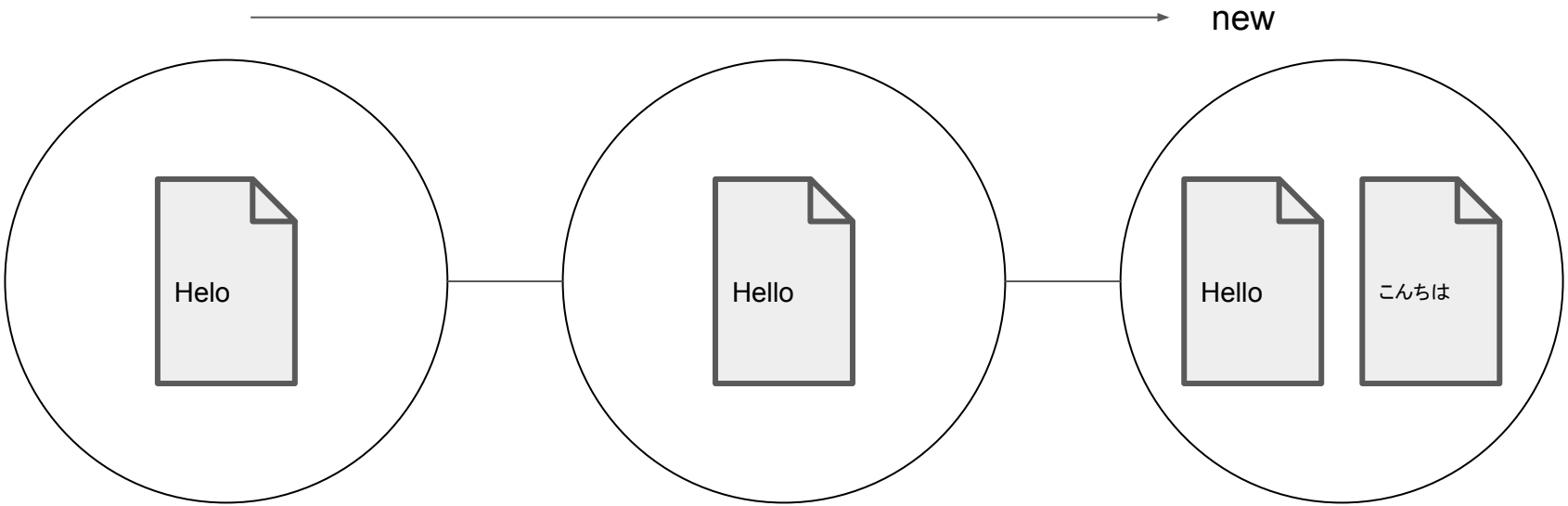
Helloのバグ修正だけをまずコミットすれば
コミット履歴がわかりやすくなる

ステージングエリアが必要な理由



そのあとに新機能追加のコミットをする

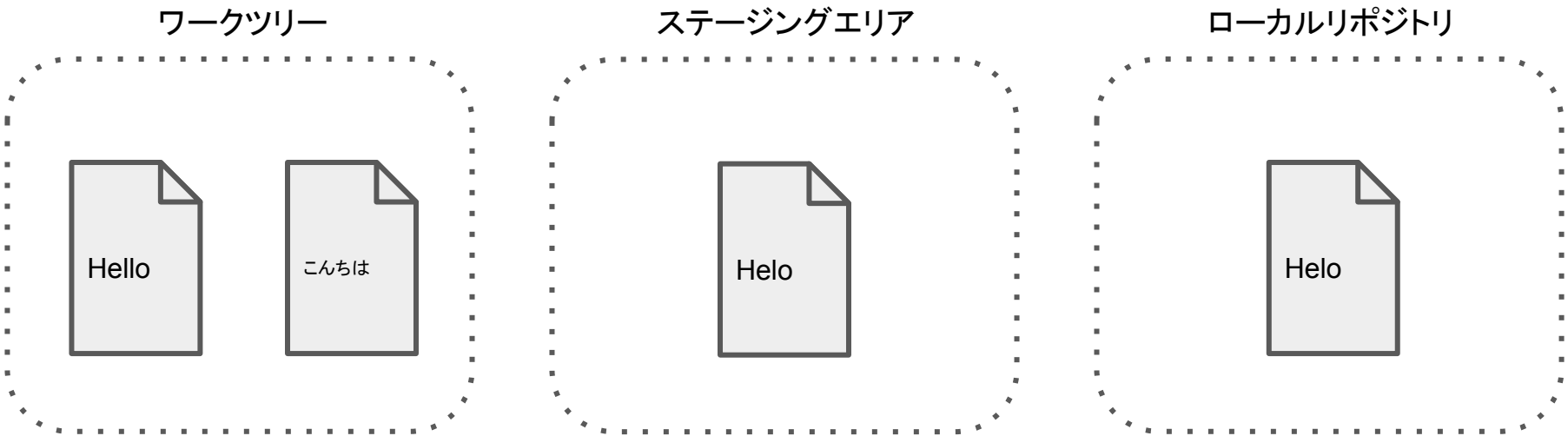
どっちの方がコミット履歴がわかりやすい？



①バグ修正とファイル（機能）追加を分けてコミットする

⇒ **コミットは局所化した方がわかりやすい！**

ステージングエリアが必要な理由

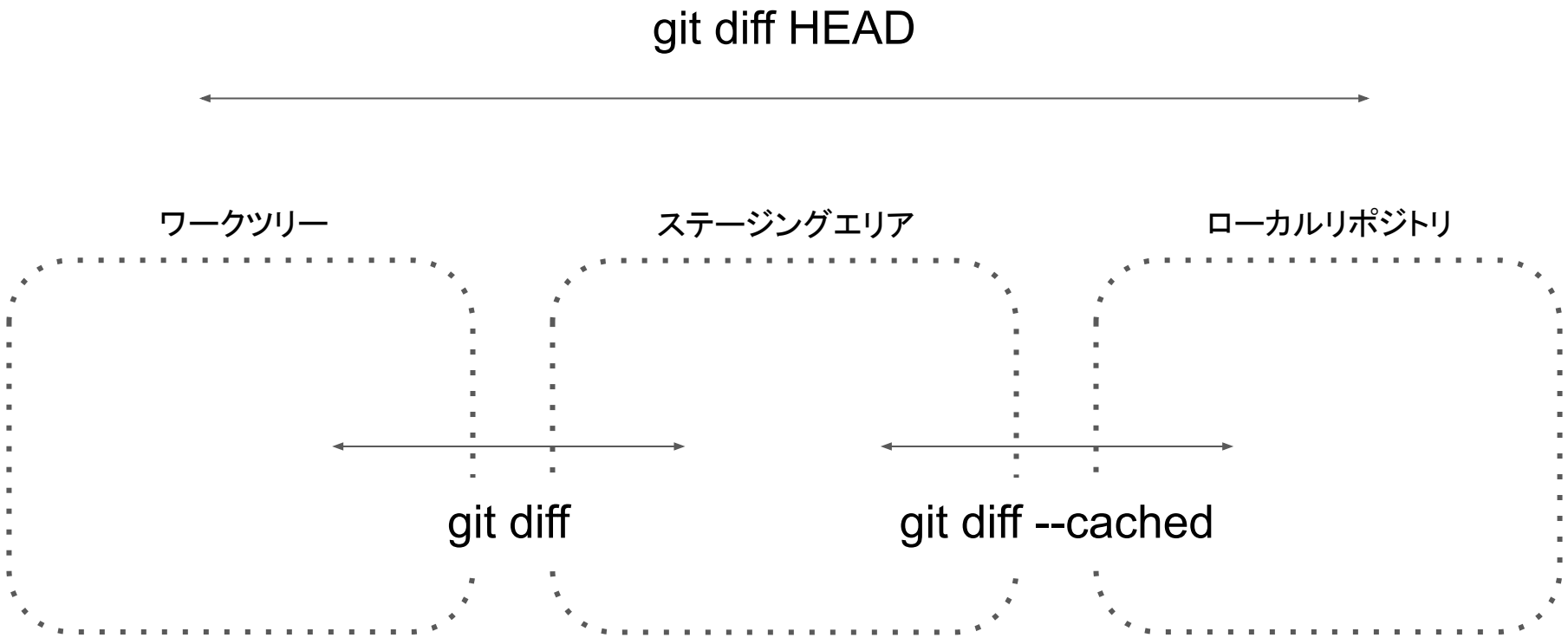


もしステージングエリアがなくて
ワークツリーのスナップショットが保存される仕組みなら
わかりづらいコミット履歴にせざるを得なくなる

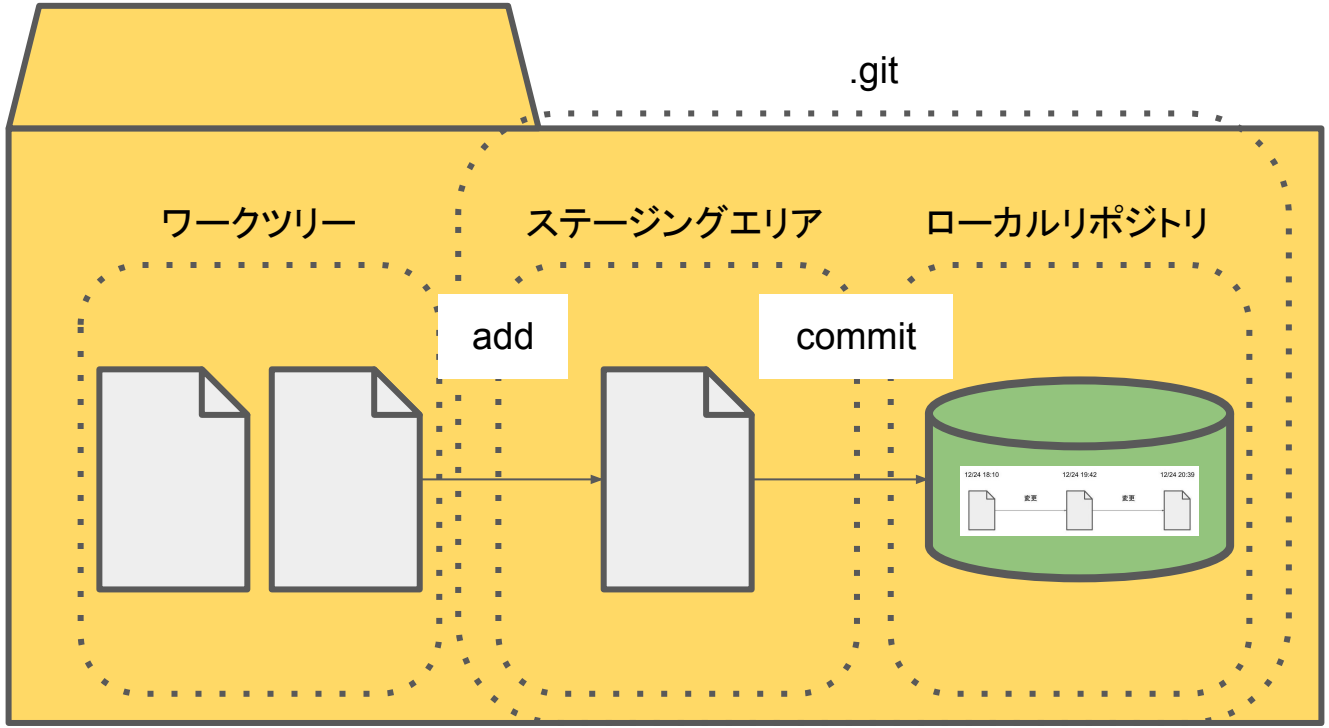
ステージングエリアが必要な理由

もしステージングエリアがなくて
ワークツリーのスナップショットが保存される仕組みなら
わかりづらいコミット履歴にせざるを得なくなる

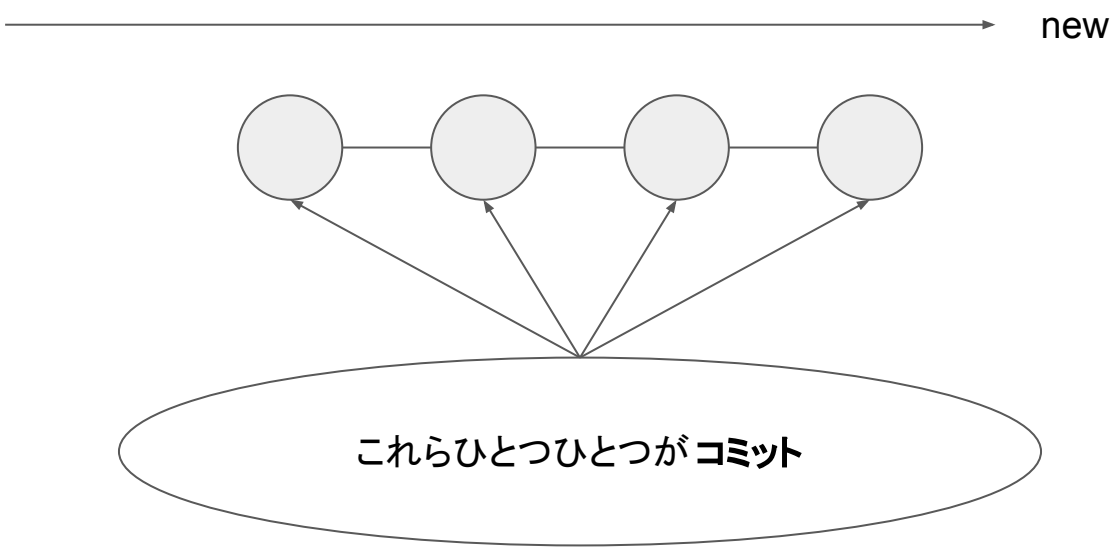
git diff の3種類



Gitによるバージョン管理（全体像）

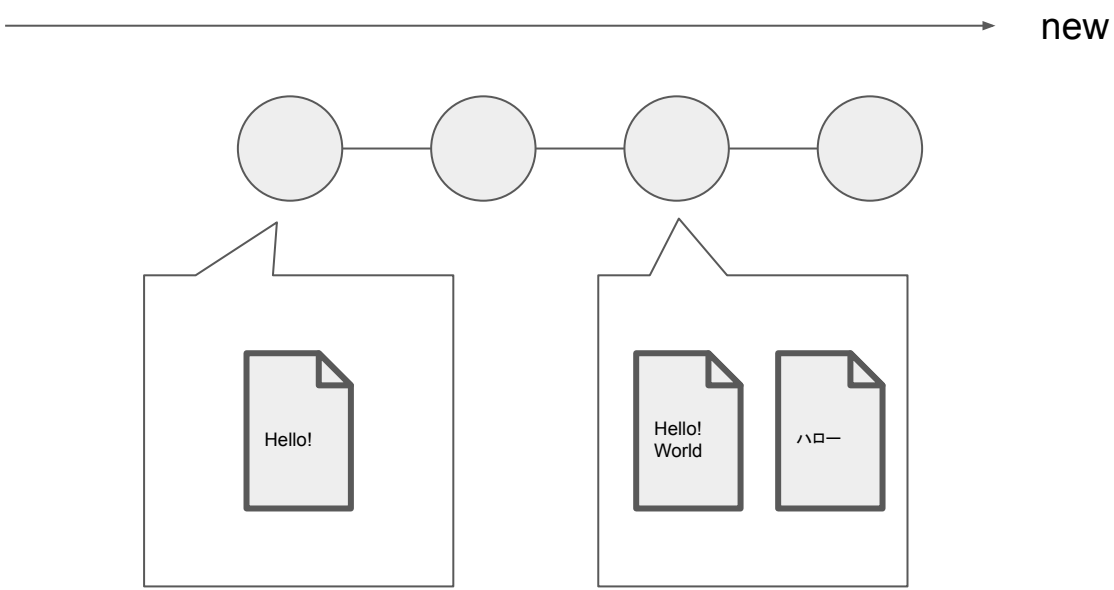


コミット（commit）とは



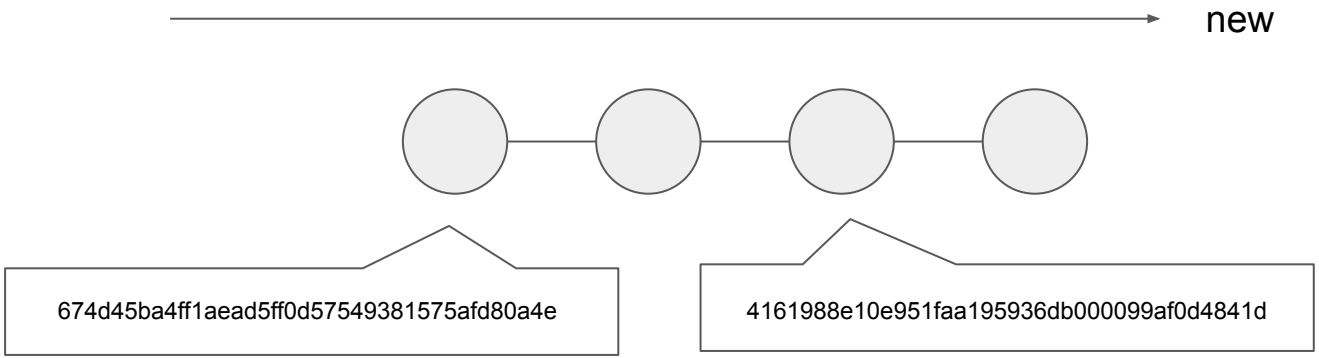
リポジトリに保存された変更の単位のこと

コミット（commit）とは



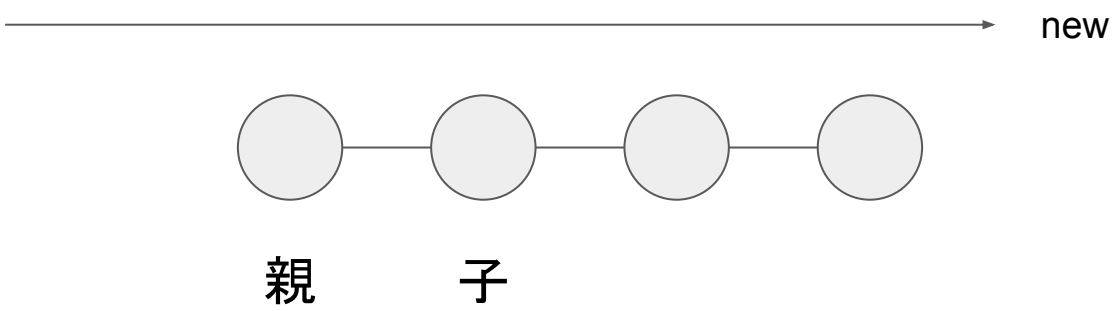
それぞれのコミットには
ステージングエリアのスナップショットが登録されている

SHA-1ハッシュ値とは



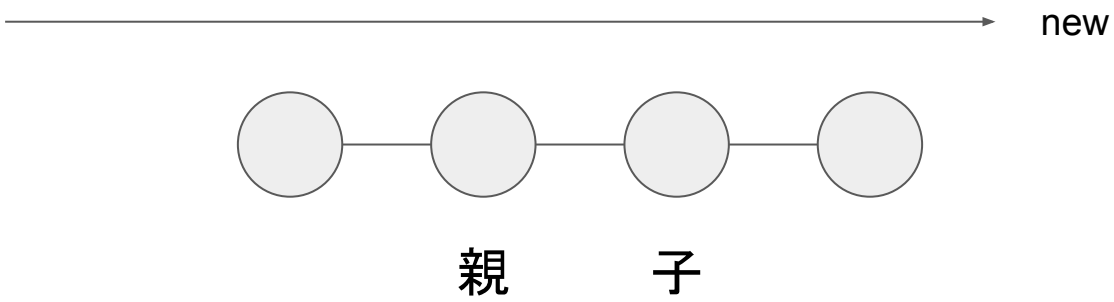
それぞれのコミットの内容から計算される16進数40文字の文字列
これを指定することでコミットを一意に特定することができる
先頭の7文字だけ表示することも可能

コミット（commit）の親子関係



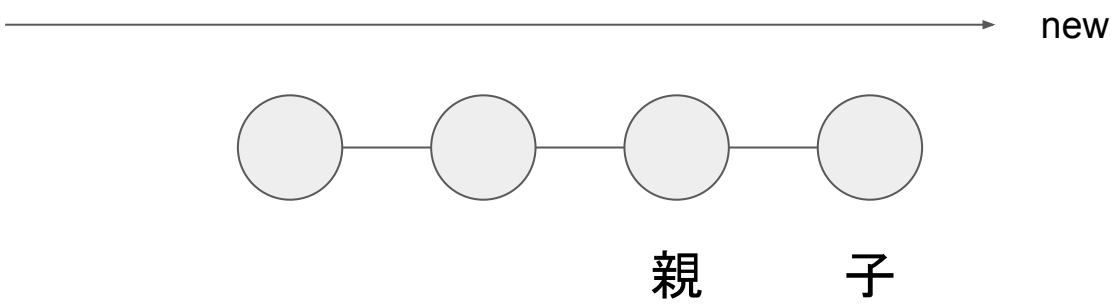
コミットの起点を親、追加したコミットを子という
親や子は複数ある場合もある（ブランチを切る・マージする場合）

コミット（commit）の親子関係



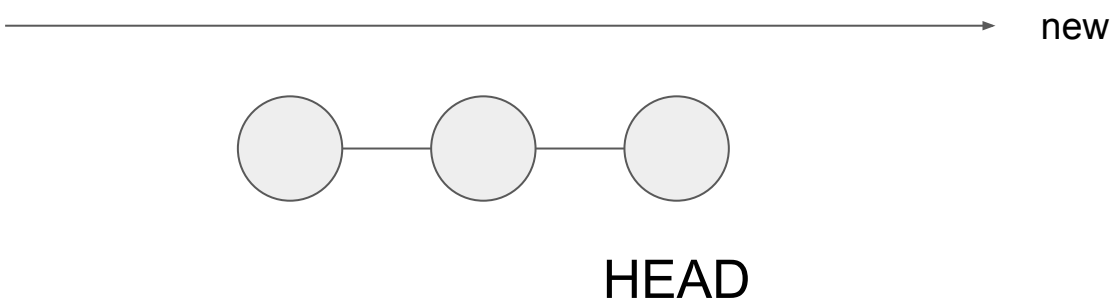
コミットの起点を親、追加したコミットを子という
親や子は複数ある場合もある（ブランチを切る・マージする場合）

コミット（commit）の親子関係



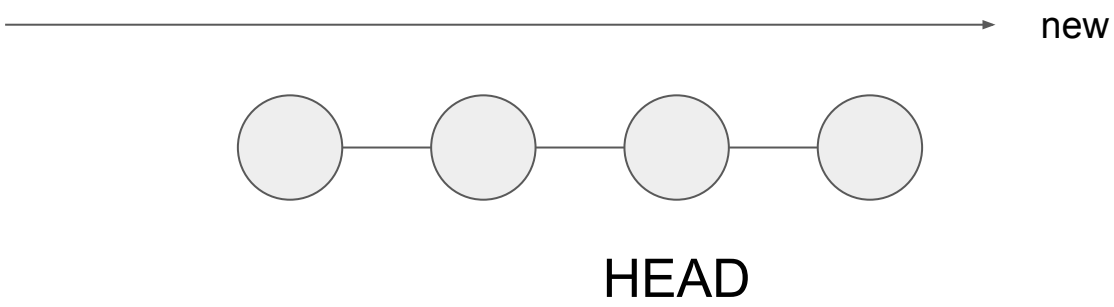
コミットの起点を親、追加したコミットを子という
親や子は複数ある場合もある（ブランチを切る・マージする場合）

コミット（commit）の親子関係



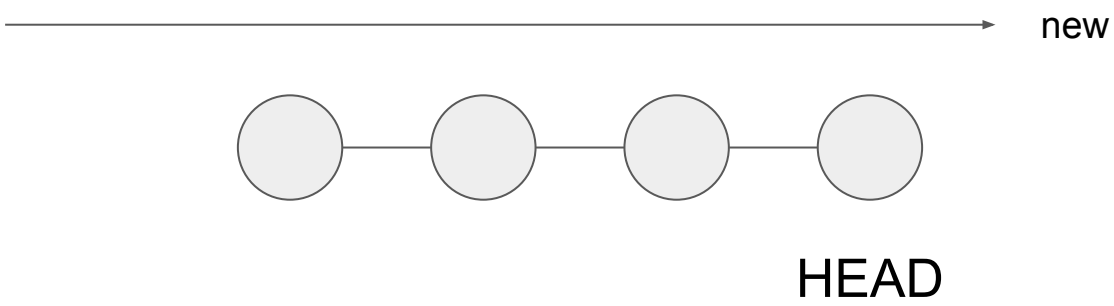
git commitを実行すると、
HEADの子にコミットが追加され、HEADは子に移動する

コミット（commit）の親子関係



git commitを実行すると、
HEADの子にコミットが追加され、HEADは子に移動する

コミット（commit）の親子関係



git commitを実行すると、
HEADの子にコミットが追加され、HEADは子に移動する

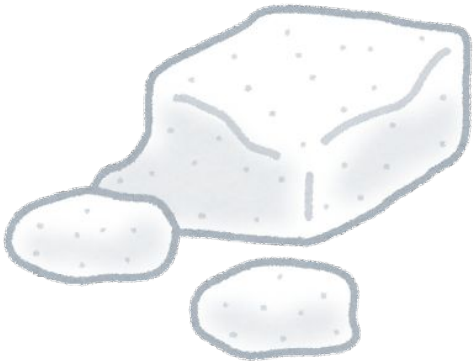
コミットするべきタイミング

- 1日の作業の終わり
- しばらく仕事から離れそうな時
- ファイルを追加、削除した時
- ファイル名を変更した時
- 機能追加など、あるタスクが完了した時
- 問題を引き起こしそうなコードを追加する前

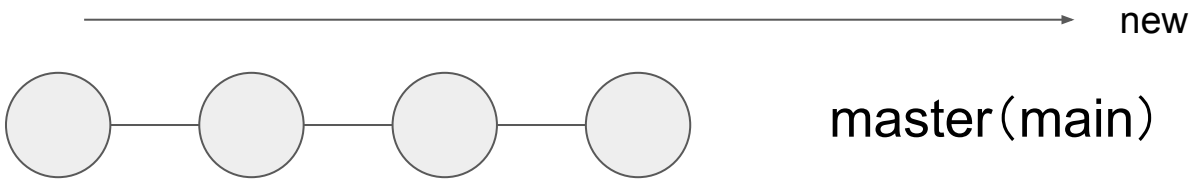
基本的な方針としては以下

- コミットは目的が明確になるように局所化する
- コミットはきりのいいところでなるべく頻繁に行う

紙粘土から人間サイズの像を複数人でつくる場合

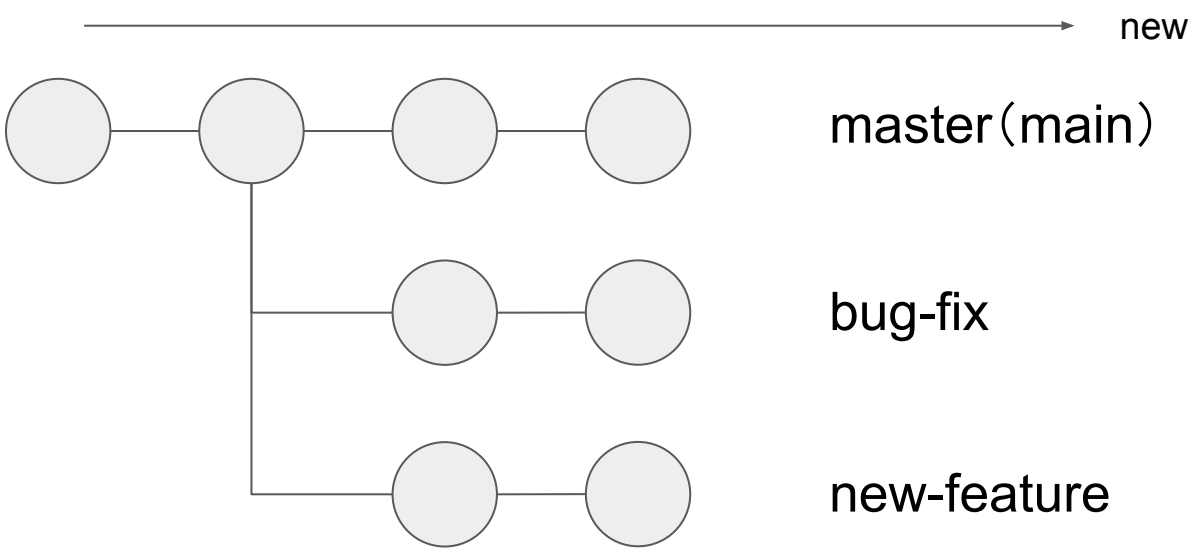


ブランチ（branch）とは



コミットの一連の履歴のこと
デフォルトではmaster（main）ブランチが生成される

ブランチ（branch）とは



開発の目的ごとにブランチを分けることで
お互いに変更の影響を与えないようにできる

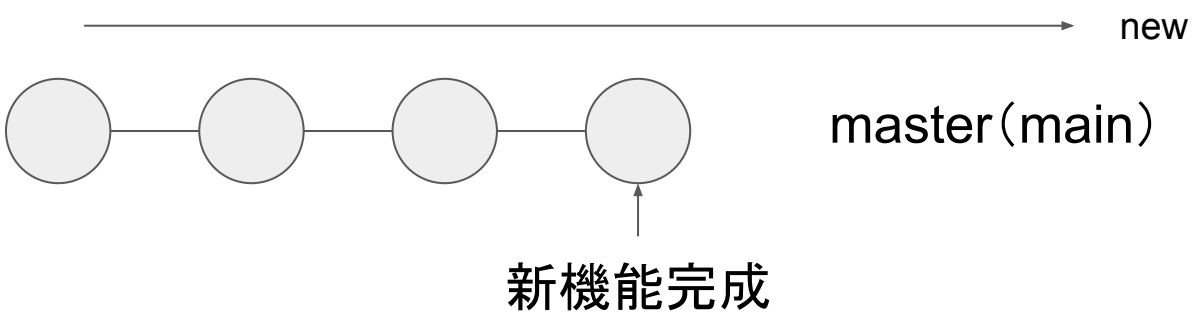
トピックブランチが必要な理由

トピックブランチを用意せずに
トランクに新機能追加のコミットをするとする

トピックブランチが必要な理由

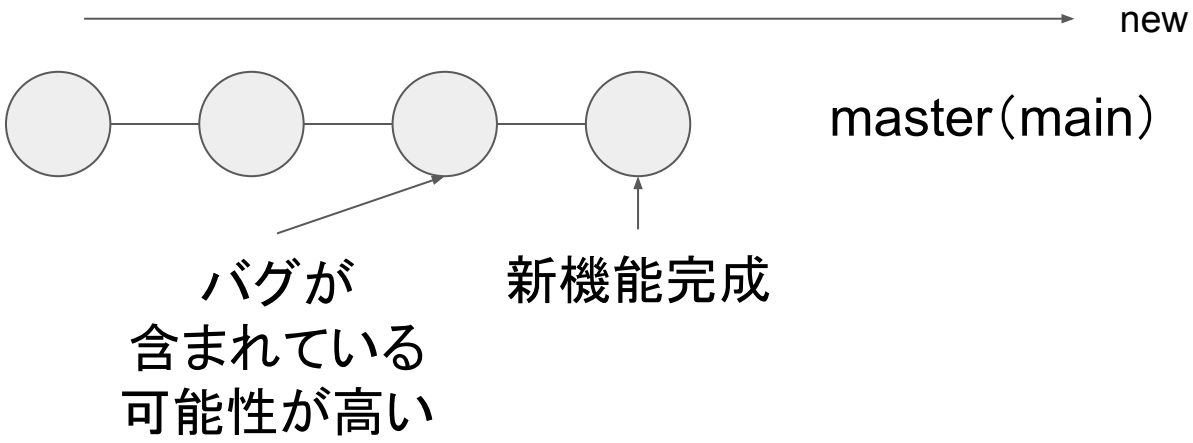
トピックブランチを用意せずに
トランクに新機能追加のコミットをするとする

トピックブランチが必要な理由



トピックブランチを用意せずに
トランクに新機能追加のコミットをするとする

トピックブランチが必要な理由

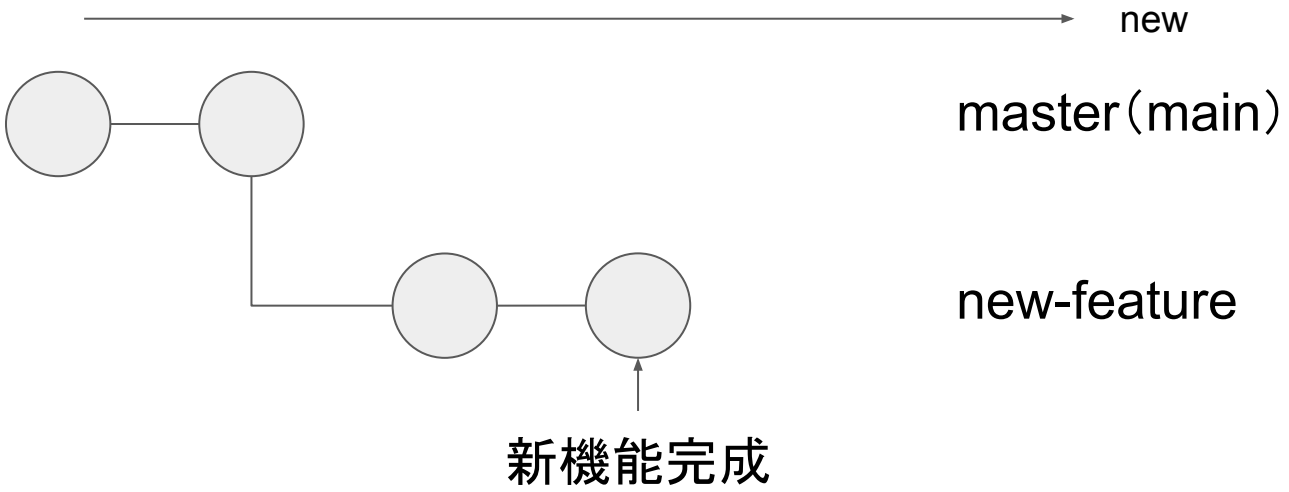


新機能追加の過程のコミットでは
バグが含まれている可能性が高いので
トランクがリリース可能ではなくなってしまう

トピックブランチが必要な理由

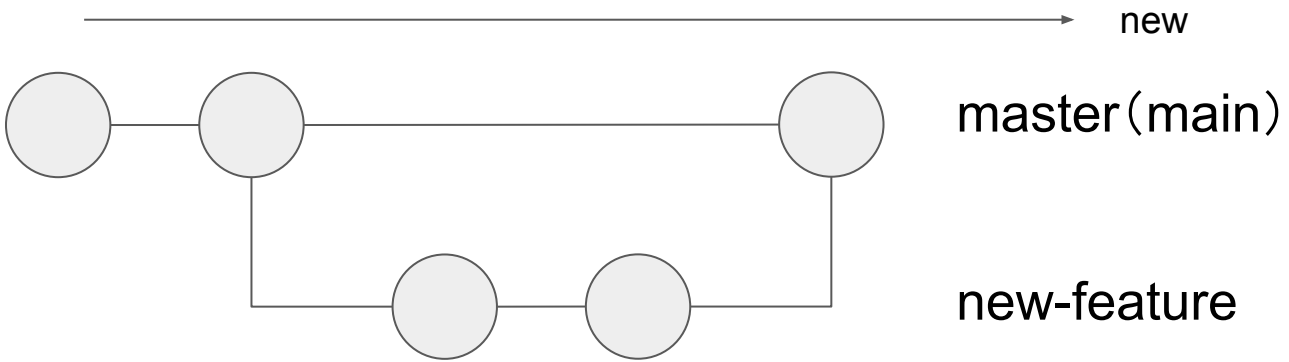
新機能の開発が成功したら、変更内容をmasterに取り入れる
こうすればリスクを最小化しつつ開発を進められる

トピックブランチが必要な理由



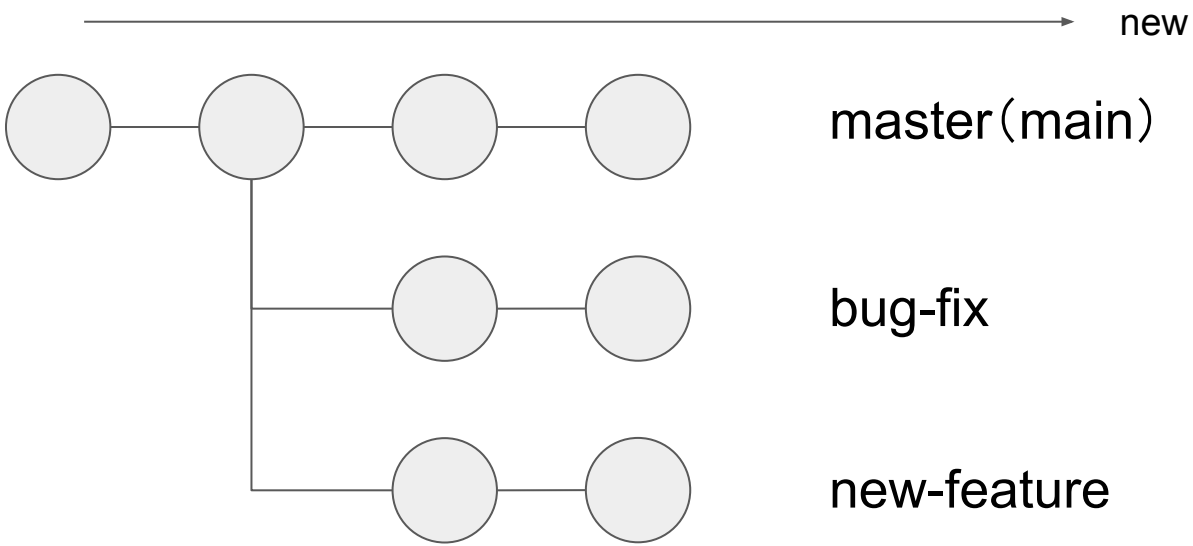
新機能の開発が成功したら、変更内容をmasterに取り入れる
こうすればリスクを最小化しつつ開発を進められる

トピックブランチが必要な理由



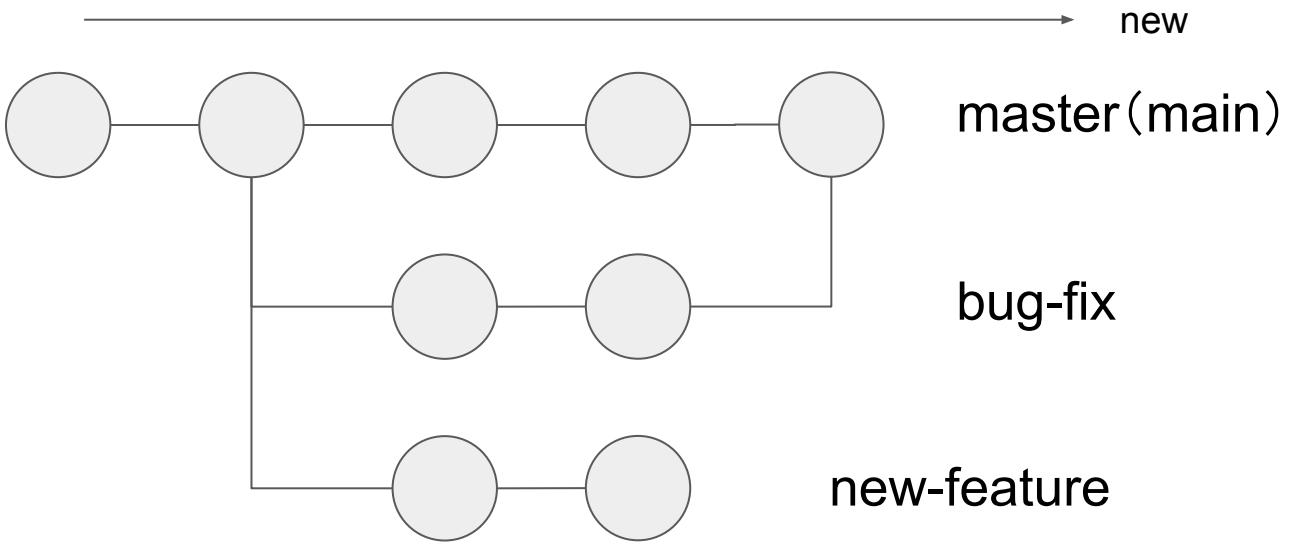
新機能の開発が成功したら、変更内容をmasterに取り入れる
こうすればリスクを最小化しつつ開発を進められる

マージ（merge）とは



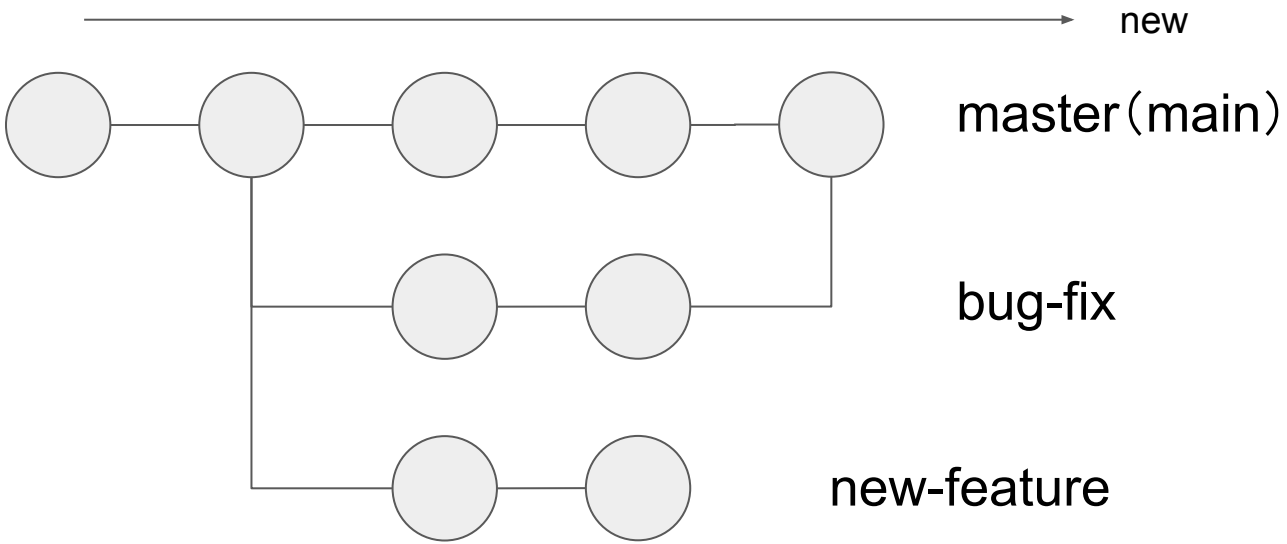
トピックブランチで行った変更を
メインブランチに取り込むことでメインブランチを伸ばしていく

マージ（merge）とは



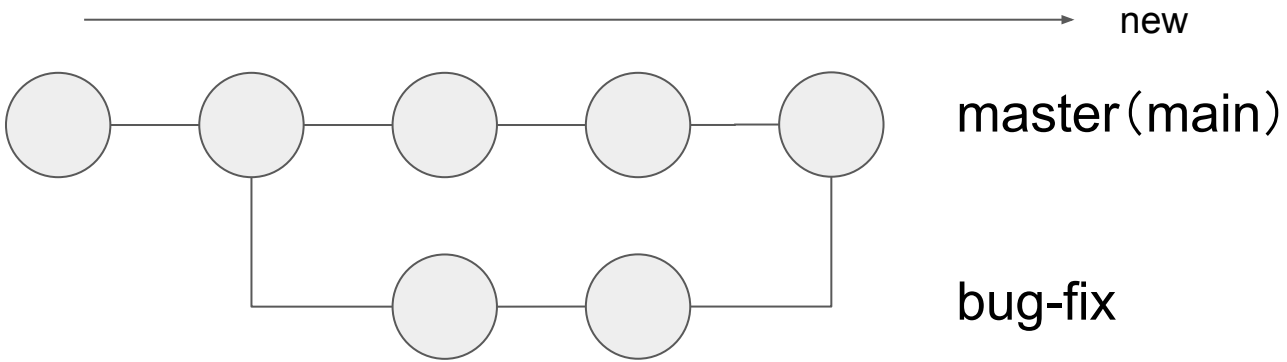
bug-fixの開発が成功したら、変更内容をmasterに取り入れる
こうすればリスクを最小化しつつ開発を進められる

マージ（merge）とは



新機能の開発が失敗したり、機能が不要になったら
new-featureブランチを削除すれば良い

マージ（merge）とは



新機能の開発が失敗したり、機能が不要になったら
new-featureブランチを削除すれば良い

これまでの流れ

最初の状態

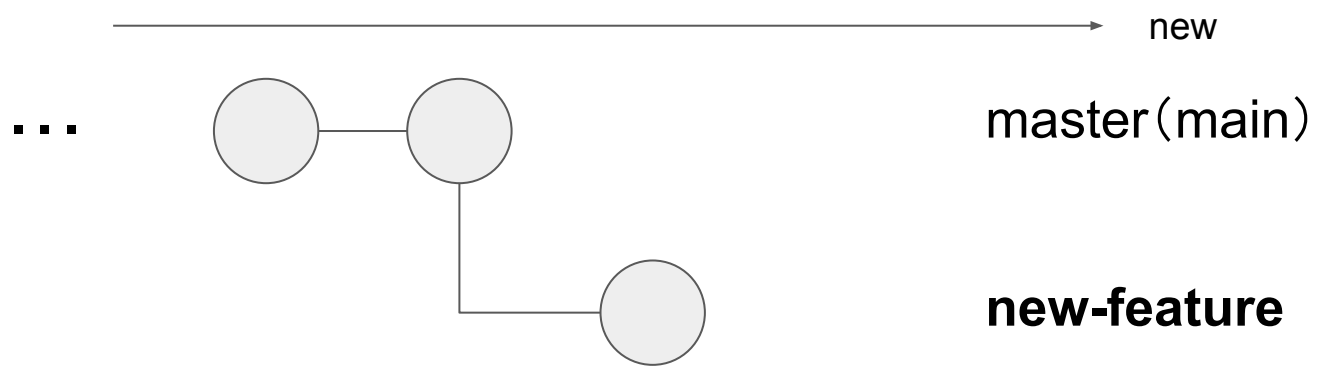
これまでの流れ

「git branch new-feature」で
new-featureブランチを新規作成

これまでの流れ

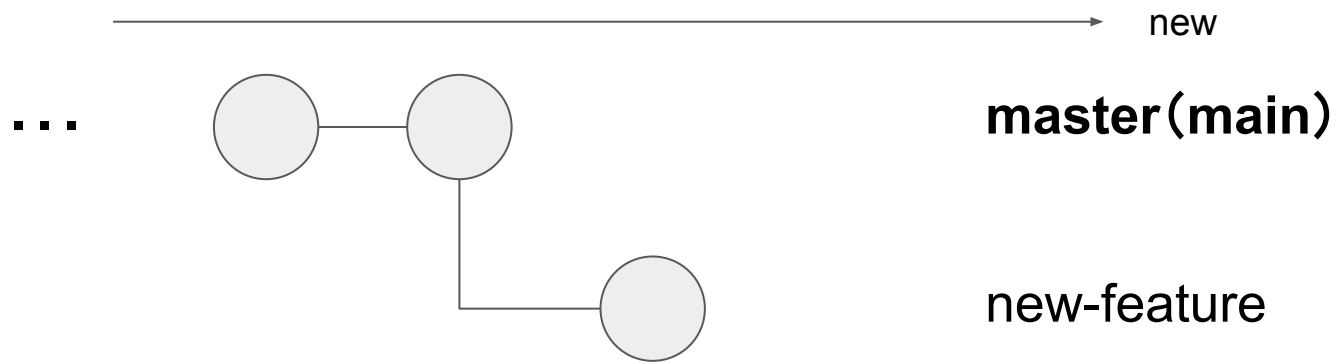
「git checkout new-feature」で
new-featureブランチに切り替え

これまでの流れ



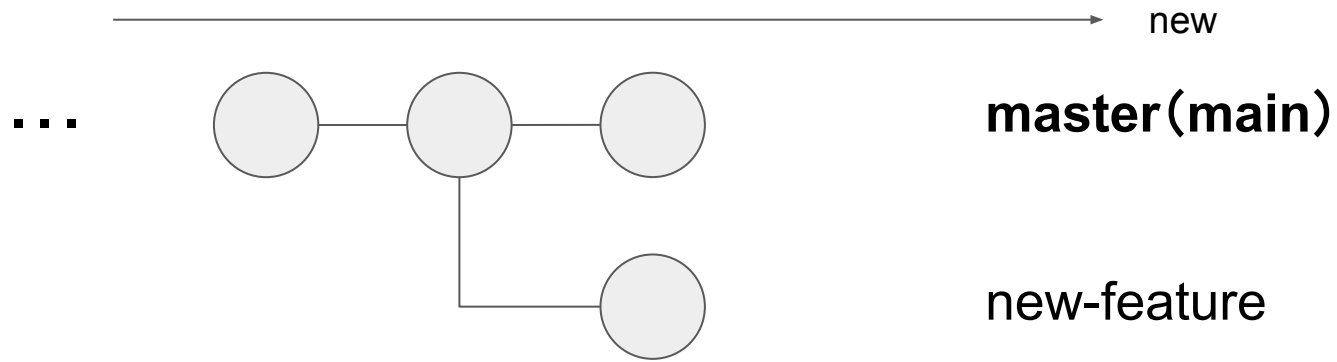
new-featureブランチにコミットを追加

これまでの流れ



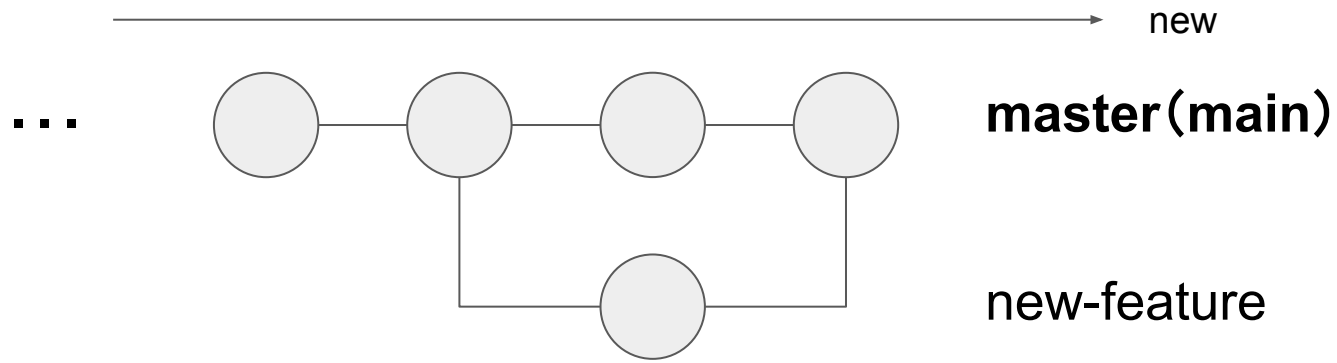
「git checkout master」で
masterブランチに切り替え

これまでの流れ



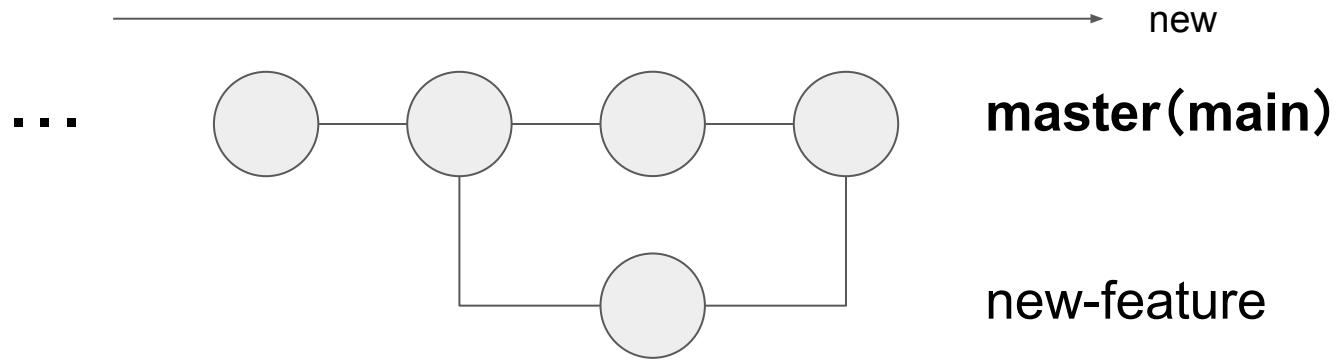
masterブランチにコミット追加

これまでの流れ



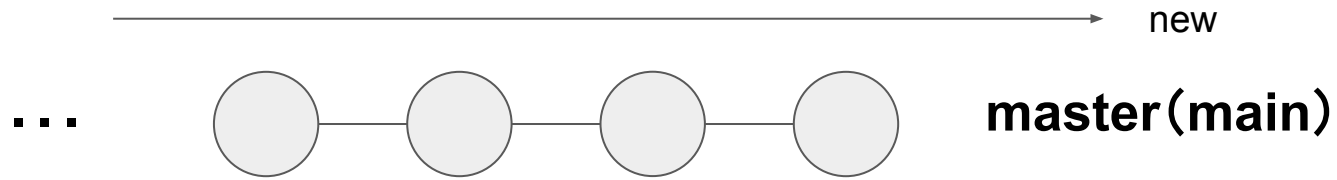
「git merge new-feature」で、現在いるmasterブランチに、
引数で指定したnew-featureブランチのコミットを取り込む

これまでの流れ



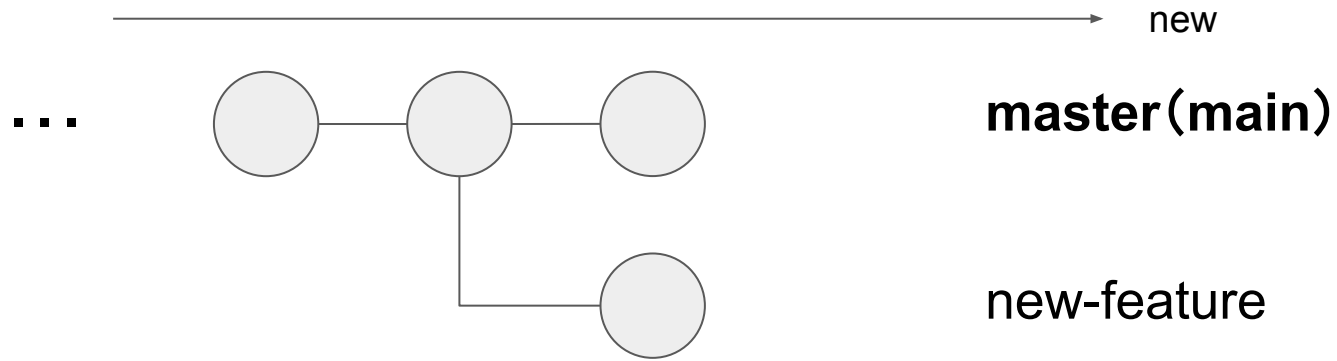
「git branch -d new-feature」を実行することで
マージを終えて不要になったトピックブランチは削除する

これまでの流れ



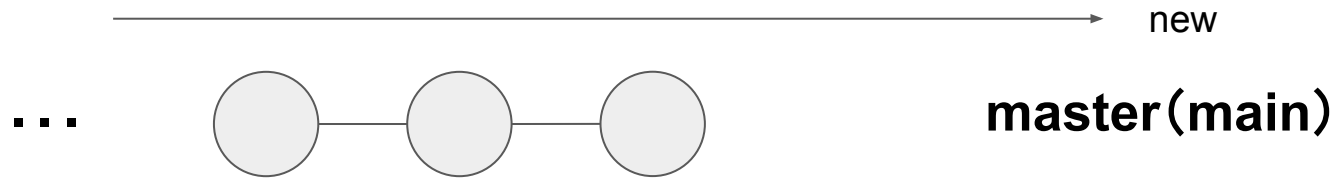
「git branch -d new-feature」を実行することで
マージを終えて不要になったトピックブランチは削除する

これまでの流れ



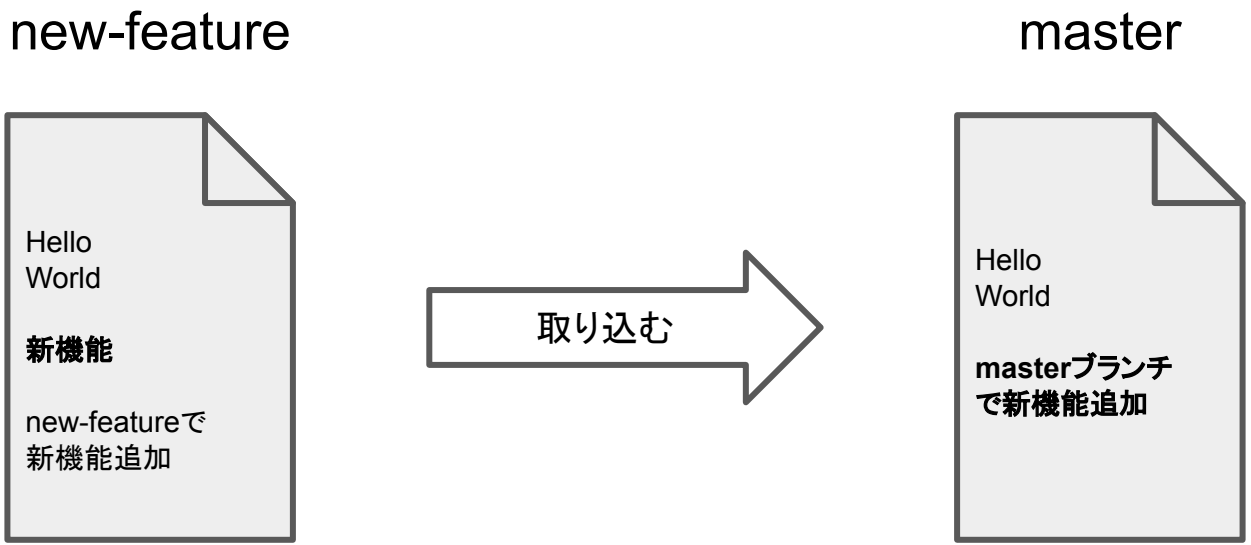
「git branch -D new-feature」を実行すると
マージされていないコミットがあるブランチを削除できる

これまでの流れ



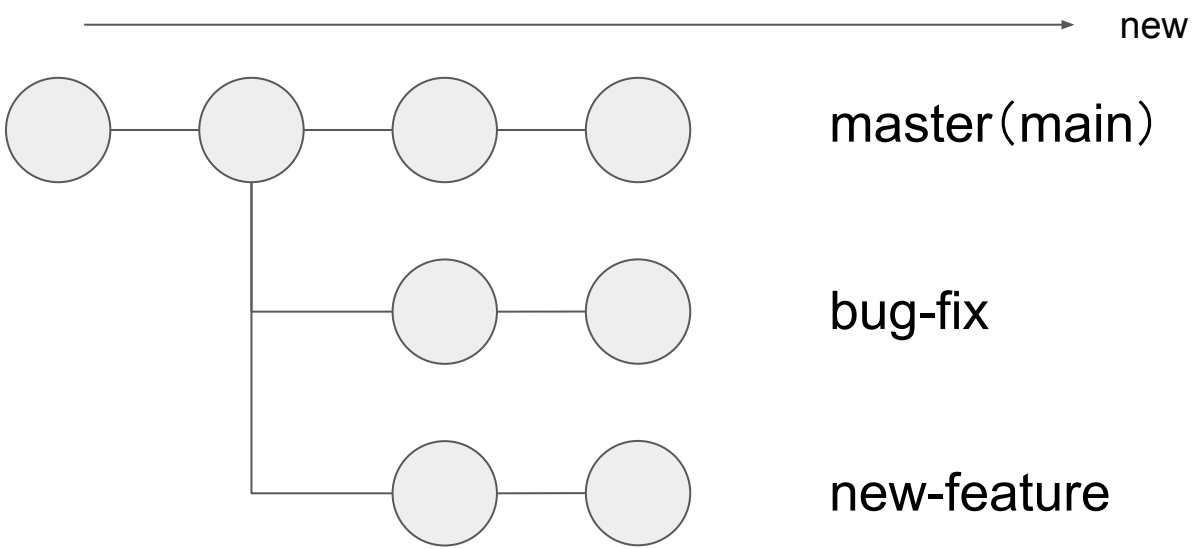
「git branch -D new-feature」を実行すると
マージされていないコミットがあるブランチを削除できる

コンフリクト（conflict）とは



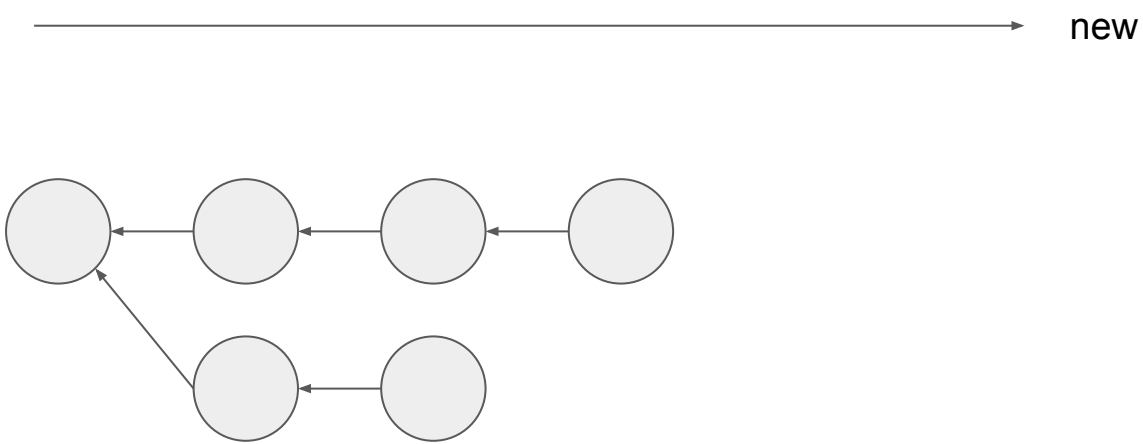
同じ箇所に別の内容が書かれていた場合、
どちらの内容を取り込めばいいかGitは判断できない

ここまで見てきたブランチ



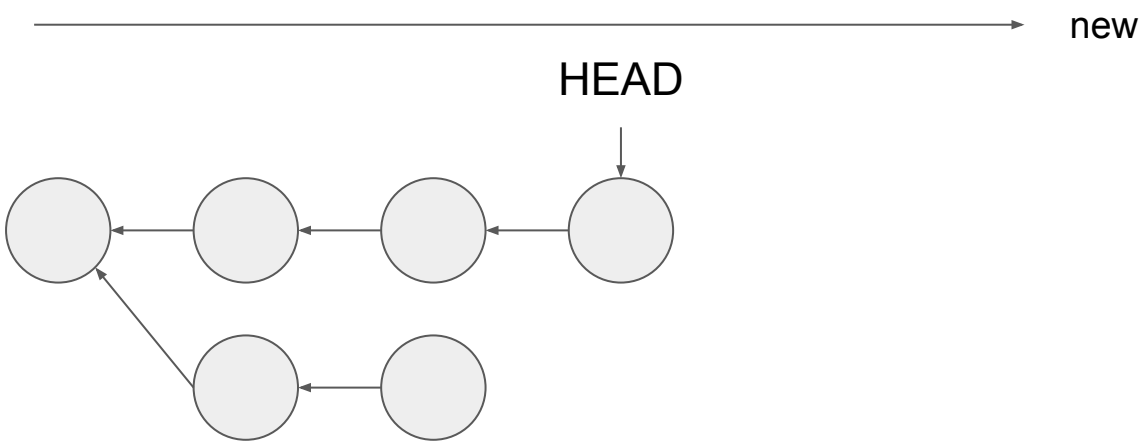
実は正確な表現ではない

コミットとブランチの仕組み



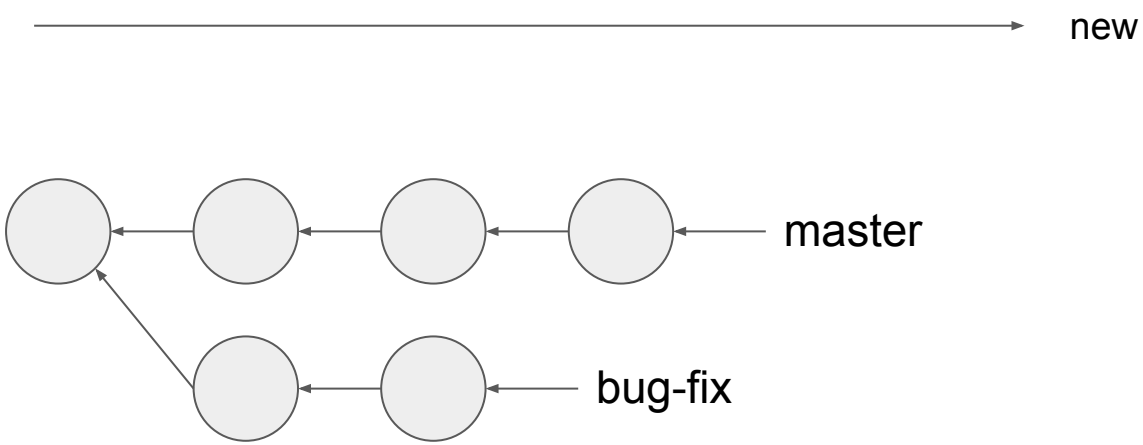
それぞれのコミットが前のコミットへのポインタを持つことで
コミットの履歴をたどれるようになっている

コミットとブランチの仕組み



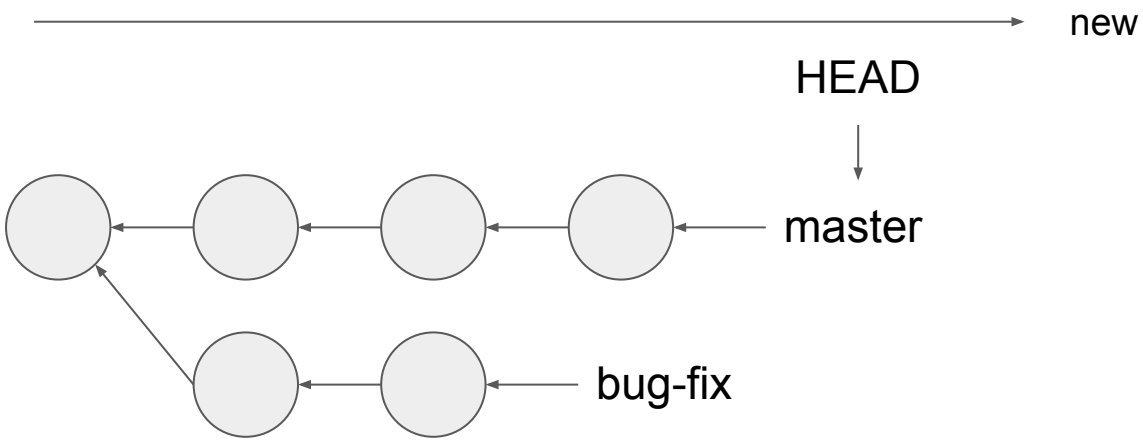
現在参照しているコミットを指すポインタがHEAD

コミットとブランチの仕組み



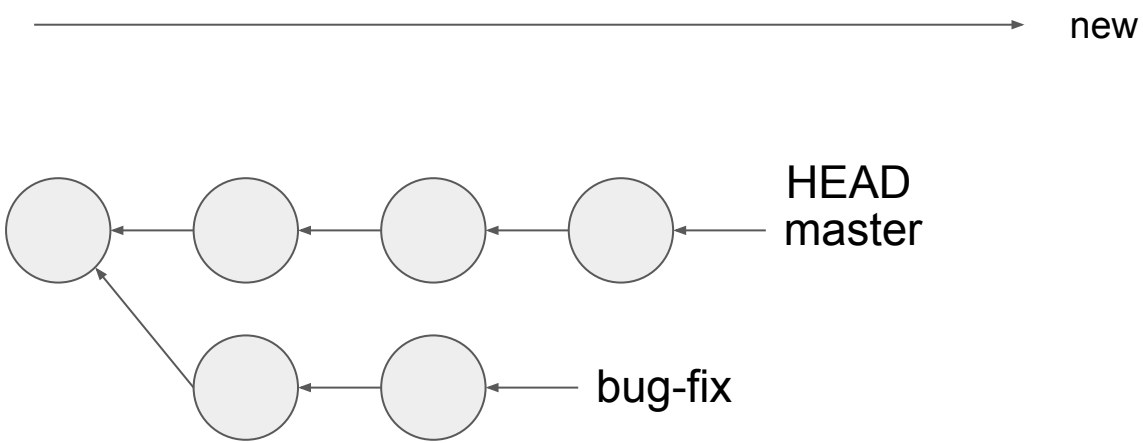
ブランチの実体は**一連のコミットの先頭を指すポインタ**

コミットとブランチの仕組み



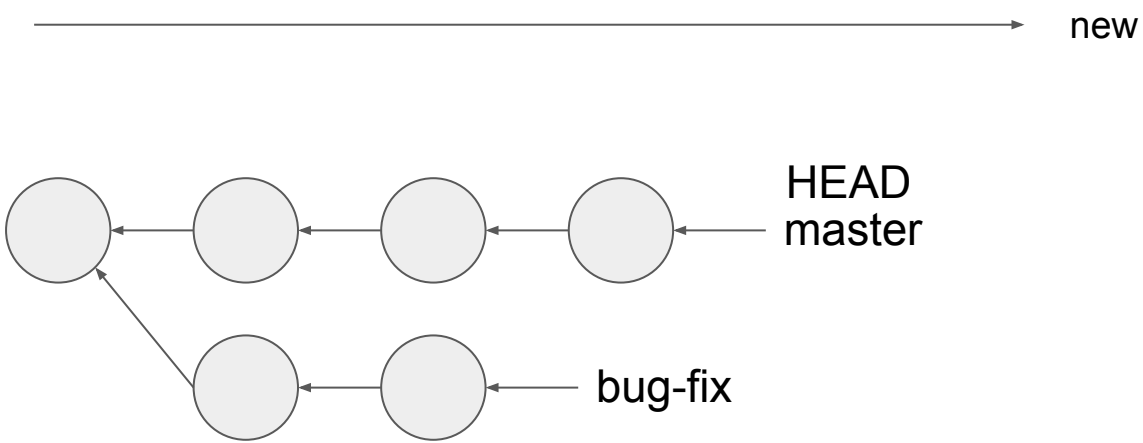
HEADはコミットだけでなく
ブランチにも移動することができる

コミットとブランチの仕組み



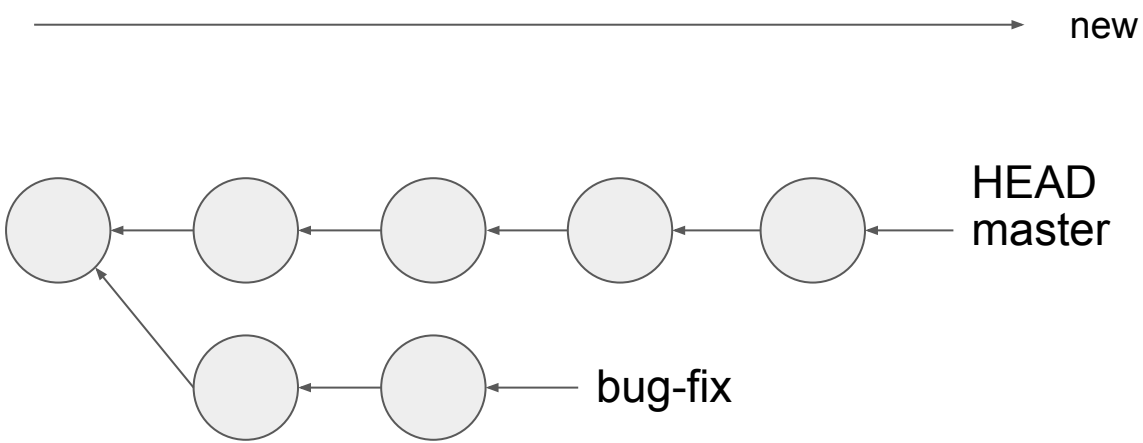
HEADはコミットだけでなく
ブランチにも移動することができる

コミットとブランチの仕組み



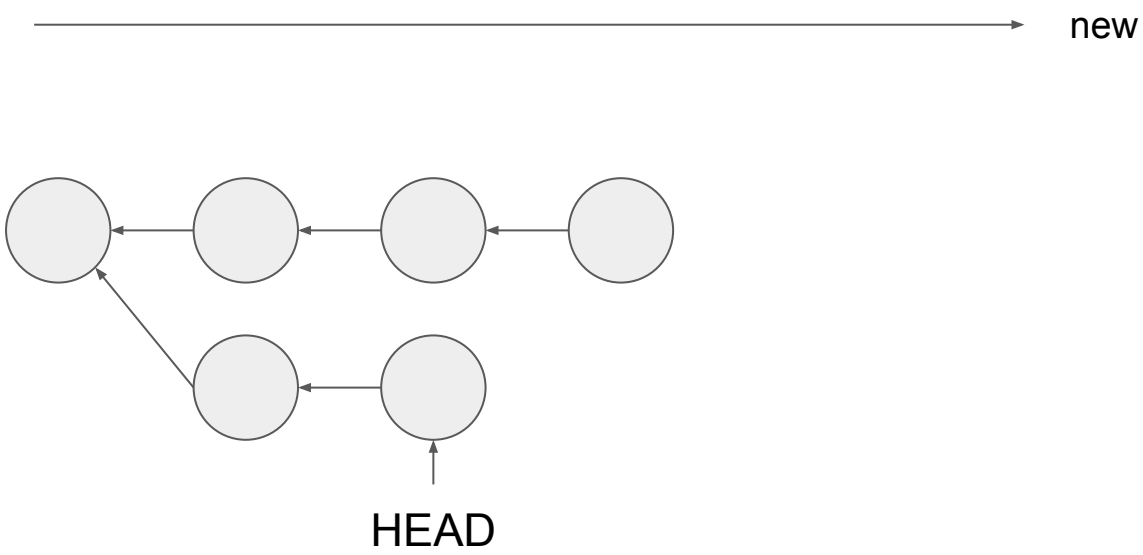
HEADがブランチを指す場合にコミットを行うと
そのブランチを伸ばすことができる

コミットとブランチの仕組み



コミットが追加されるとブランチのポインタも移動する

コミットとブランチの仕組み

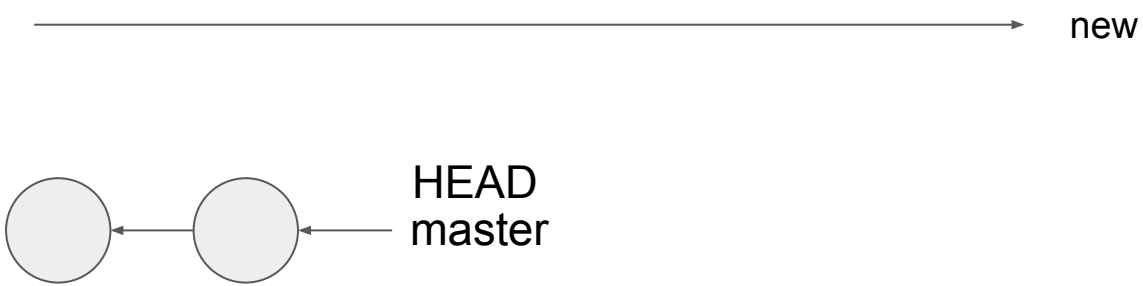


チェックアウトするとHEADが移動する

git commit

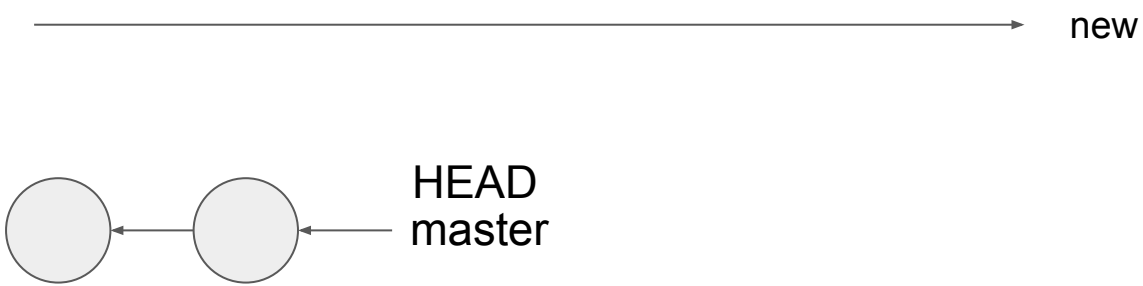
HEADが指すブランチが指すコミットの子に
新たなコミットを追加する

git commit



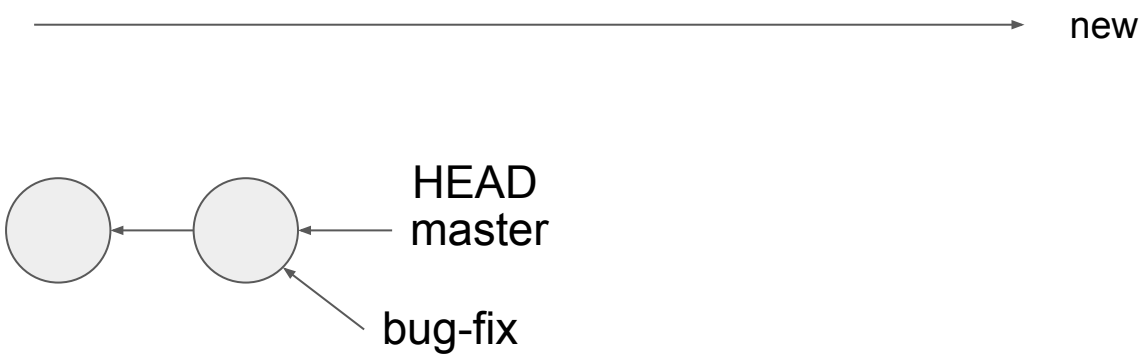
HEADが指すブランチが指すコミットの子に
新たなコミットを追加する

git branch bug-fix



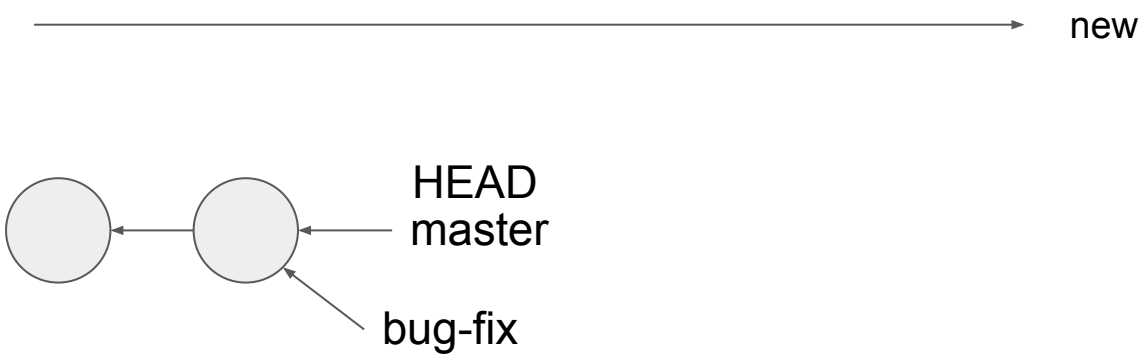
HEADが指すコミットにブランチを追加する
HEADがブランチを指す場合は、そのブランチが指すコミット

git branch bug-fix



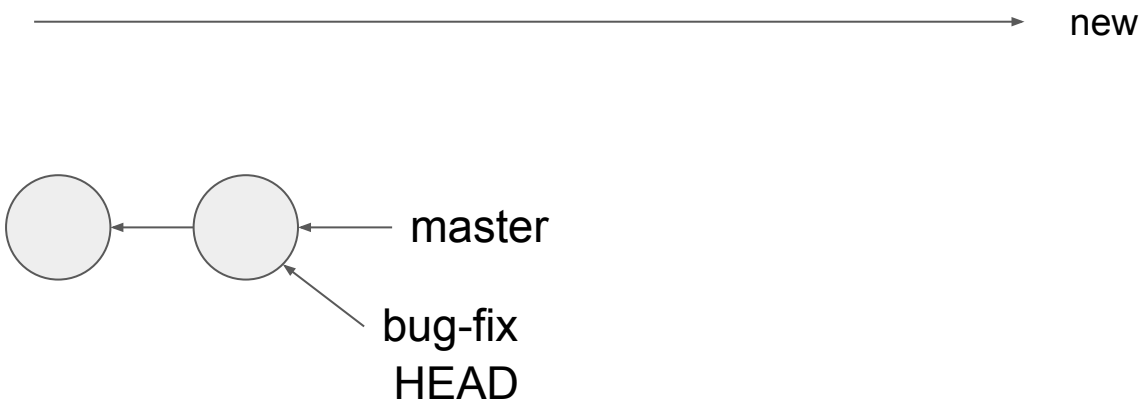
HEADが指すコミットにブランチを追加する
HEADがブランチを指す場合は、そのブランチが指すコミット

git checkout bug-fix



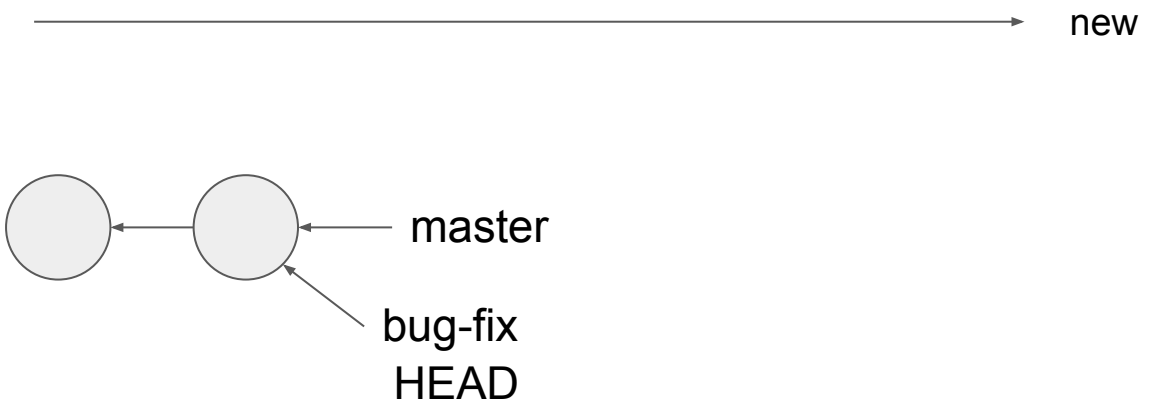
指定したブランチやコミットにHEADを移動する

git checkout bug-fix



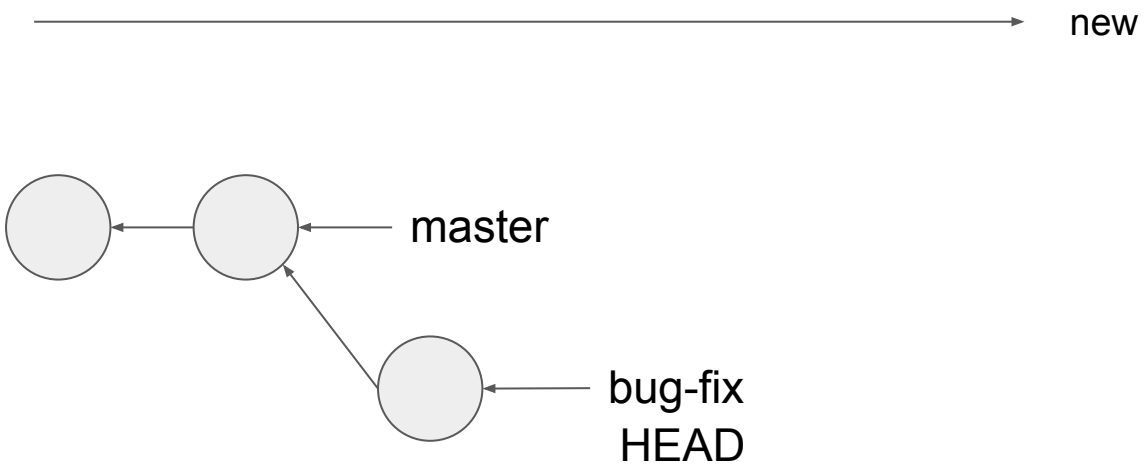
指定したブランチやコミットにHEADを移動する

git commit



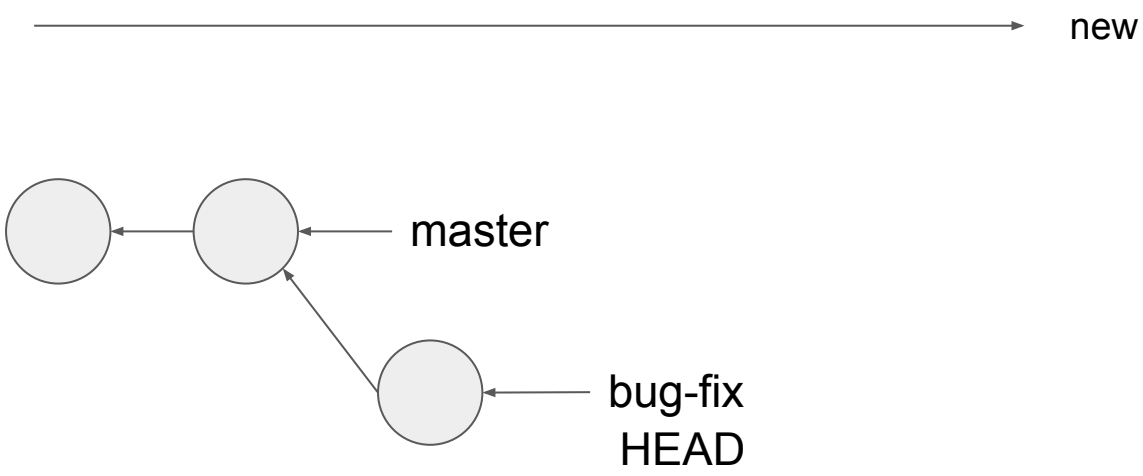
bug-fixをHEADが指しているのでコミットすると
bug-fixが指すコミットの子に新たなコミットが追加される

git commit



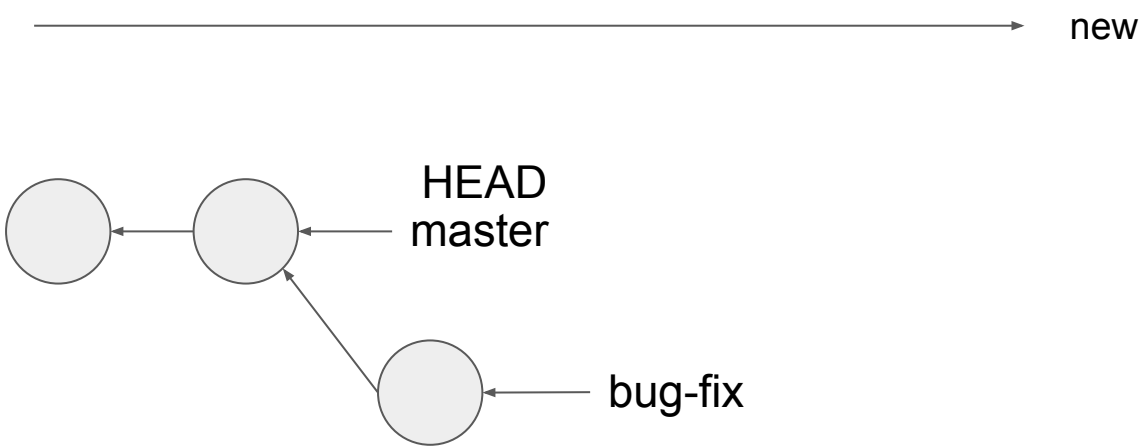
bug-fixをHEADが指しているのでコミットすると
bug-fixが指すコミットの子に新たなコミットが追加される

git checkout master



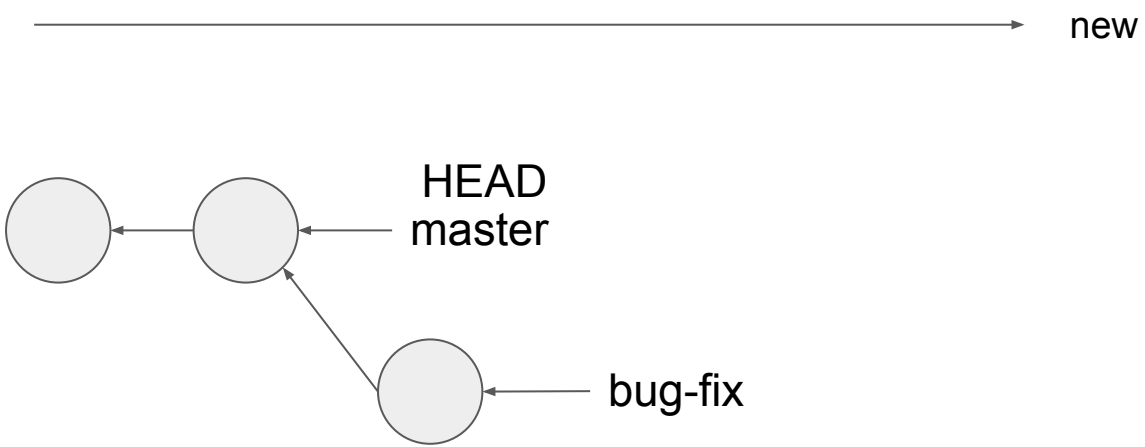
指定したmasterブランチにHEADが移動する

git checkout master



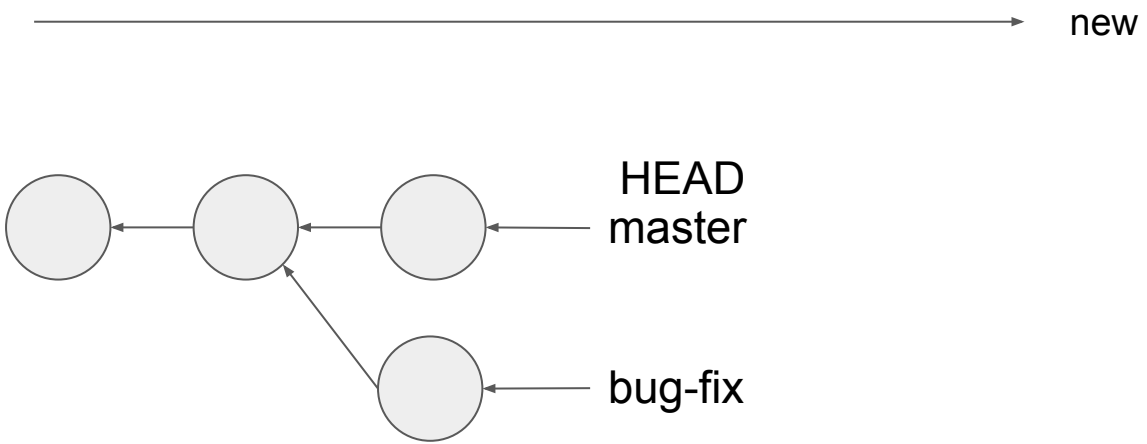
指定したmasterブランチにHEADが移動する

git commit



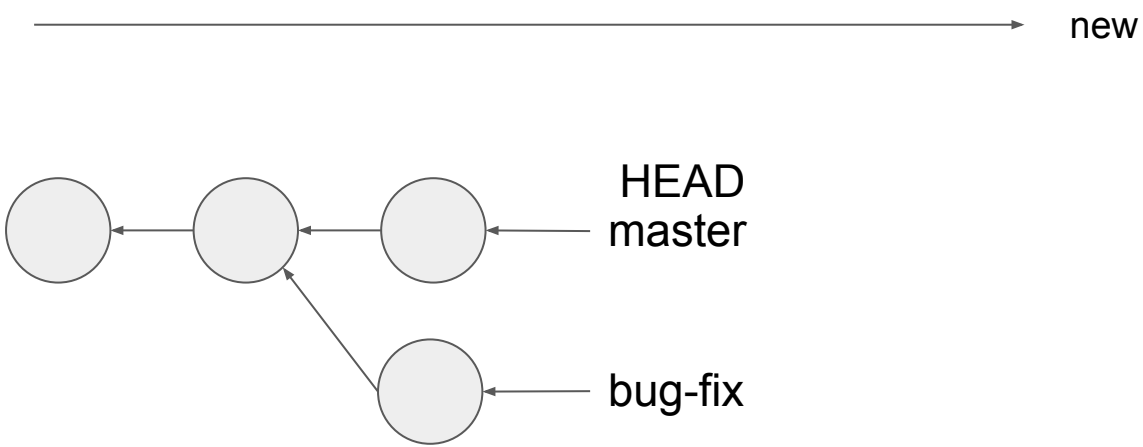
masterをHEADが指しているのでコミットすると
masterが指すコミットの子に新たなコミットが追加される

git commit



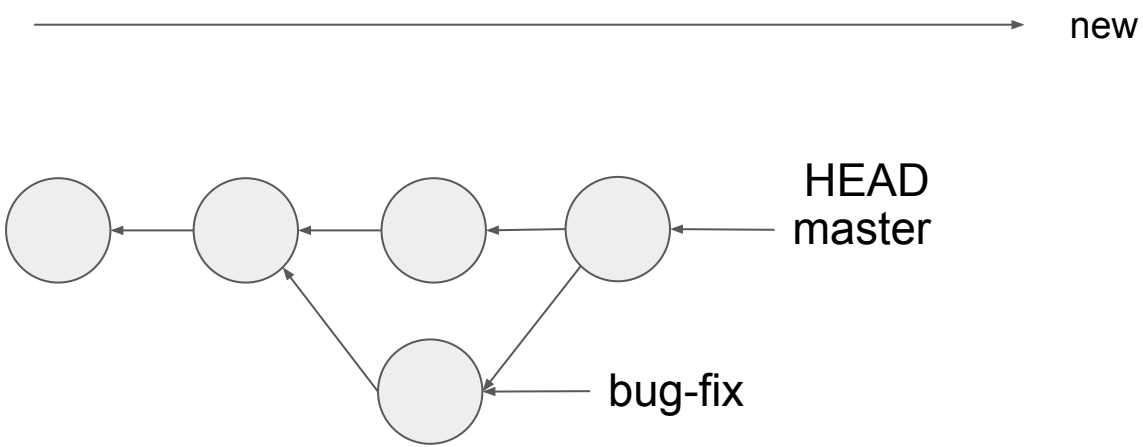
masterをHEADが指しているのでコミットすると
masterが指すコミットの子に新たなコミットが追加される

git merge bug-fix



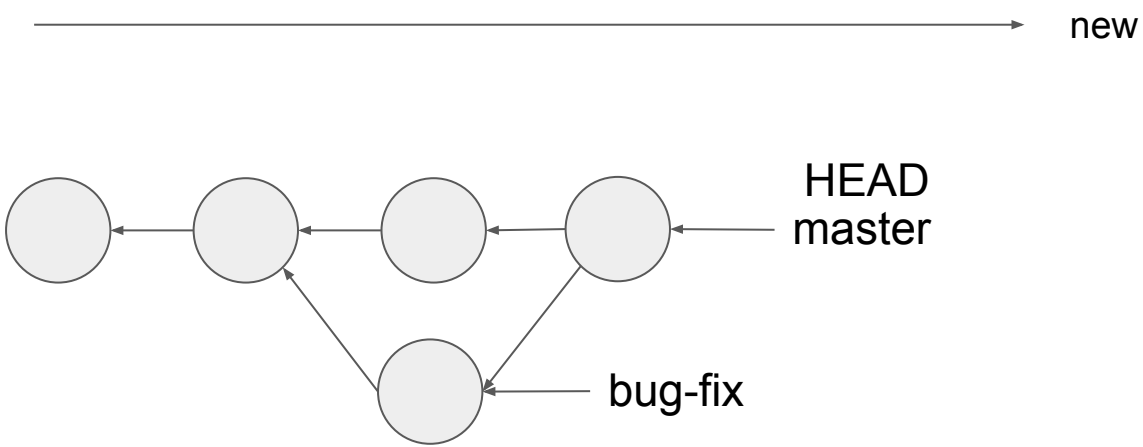
HEADが指すmasterブランチが指すコミットの子に
bug-fixブランチが指すコミットの内容を
取り込んだコミット追加する

git merge bug-fix



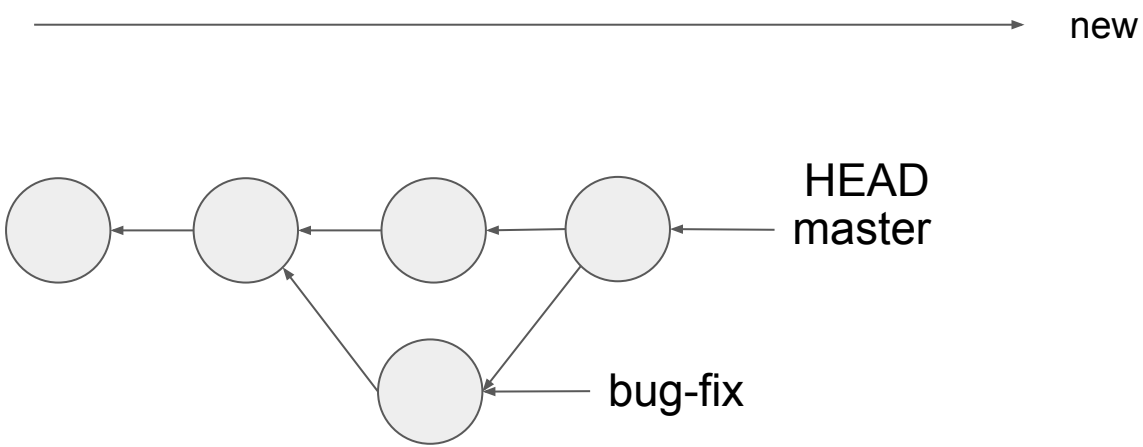
HEADが指すmasterブランチが指すコミットの子に
bug-fixブランチが指すコミットの内容を
取り込んだコミット追加する

git merge bug-fix



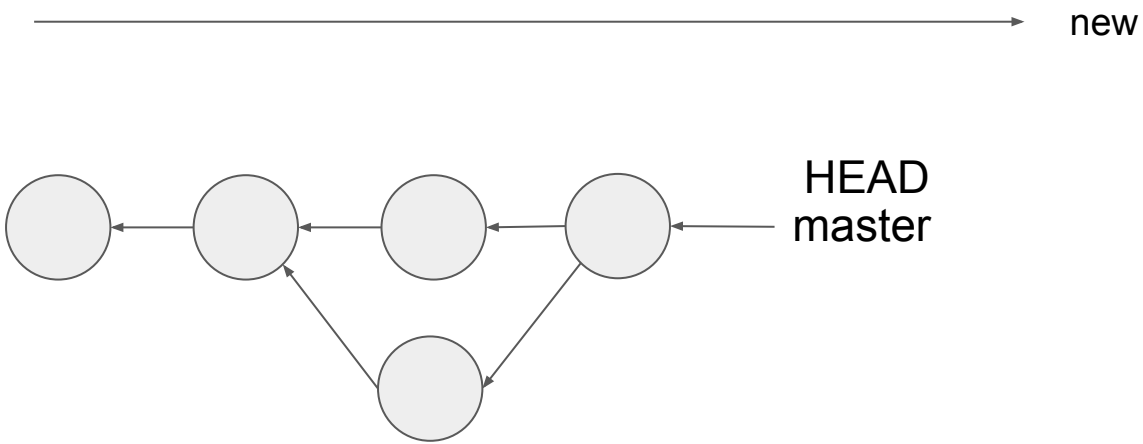
bug-fixブランチの位置は変わっていないことに注目

git branch -d bug-fix



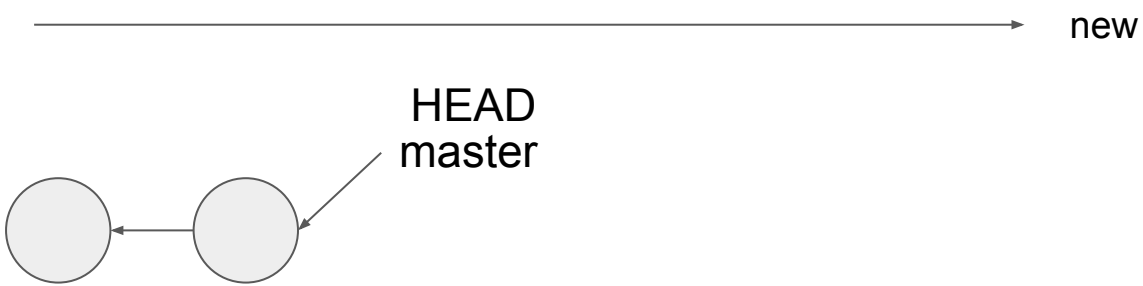
bug-fixブランチが削除される
bug-fixブランチが指すコミット自体が削除されるわけではない

git branch -d bug-fix



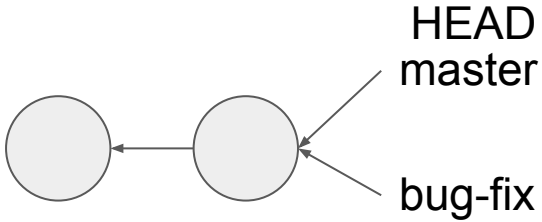
bug-fixブランチが削除される
bug-fixブランチが指すコミット自体が削除されるわけではない

ブランチの仕組み



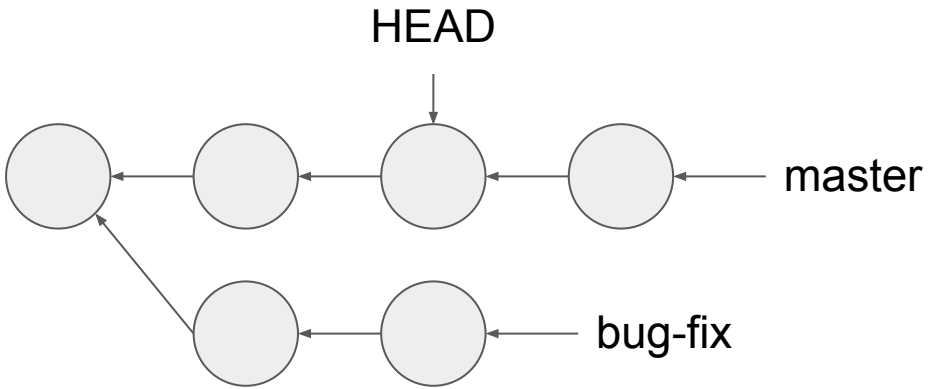
「git branch bug-fix」を実行するとHEADが指すコミットに
bug-fixブランチのポインタが追加される

ブランチの仕組み



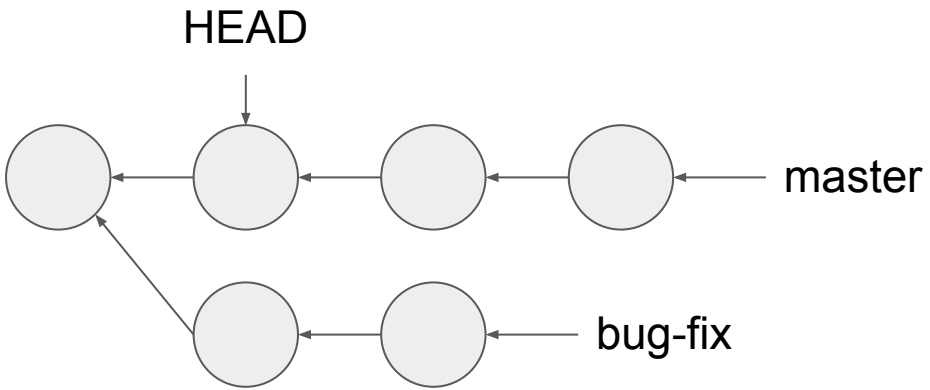
「git branch bug-fix」を実行するとHEADが指すコミットに
bug-fixブランチのポインタが追加される

detached HEADとは



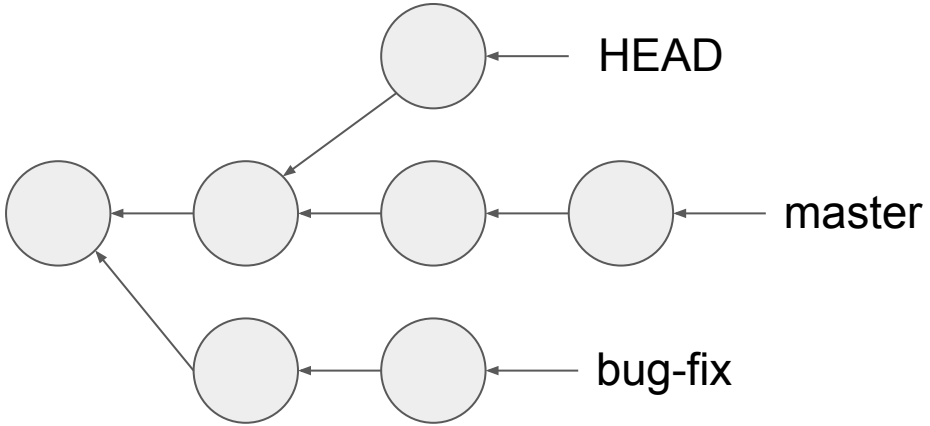
HEADがどのブランチも指していない状態を
detached HEAD（切り離されたHEAD）と呼ぶ

detached HEADとは



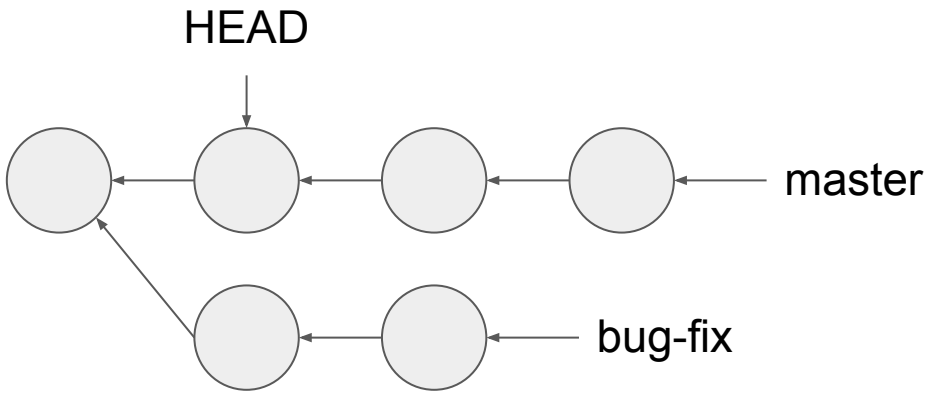
HEADがどのブランチも指していない状態を
detached HEAD（切り離されたHEAD）と呼ぶ

detached HEADでのコミット



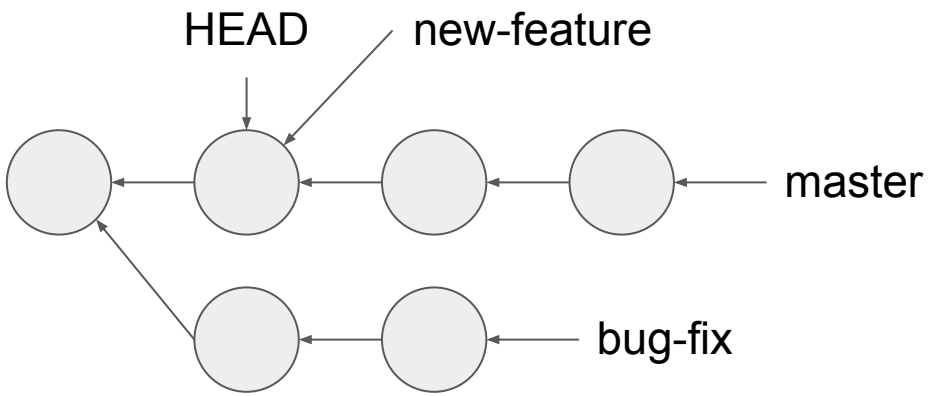
detached HEADでのコミットは、HEADの子にコミットを作るが、
どのブランチにも属さないコミットになってしまう

detached HEADでのコミットでブランチを伸ばす方法



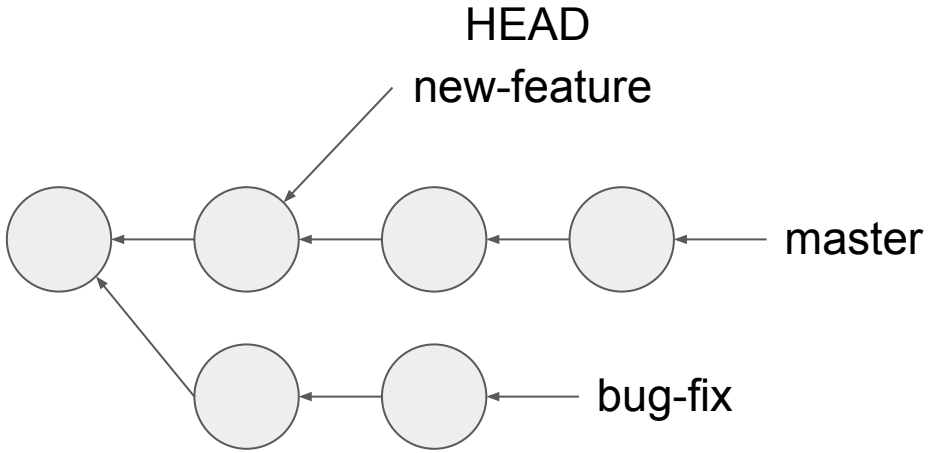
「git branch new-feature」を実行して
HEADのコミットにブランチを作成する

detached HEADでのコミットでブランチを伸ばす方法



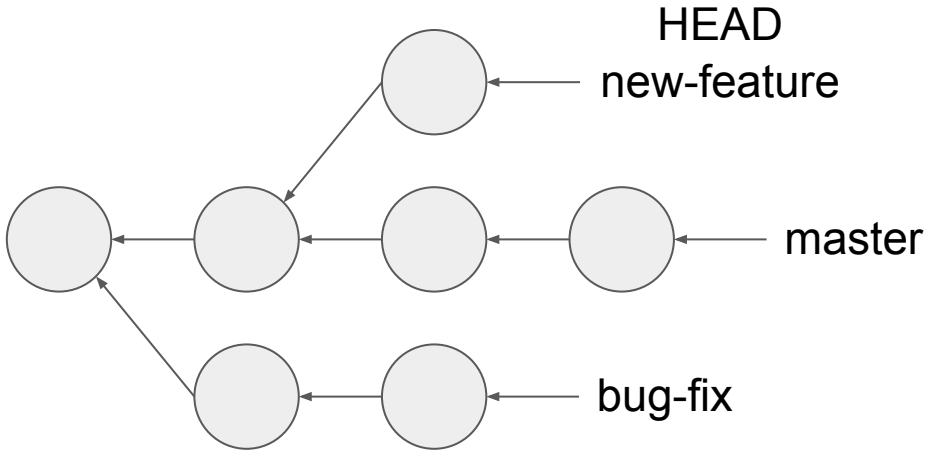
「git checkout new-feature」を実行して
HEADをnew-featureブランチに移動する

detached HEADでのコミットでブランチを伸ばす方法



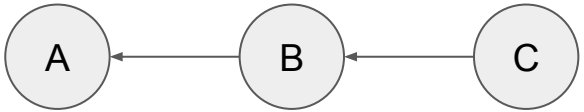
「git commit -m "Message"」を実行して
HEADの子にコミットを追加する

detached HEADでのコミットでブランチを伸ばす方法



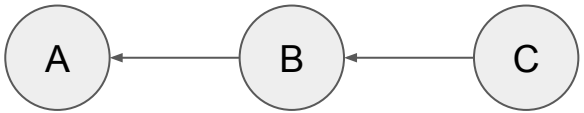
こうすれば、new-featureブランチから
コミットにアクセスできるようになる

コミットの取り消し方法は2つある



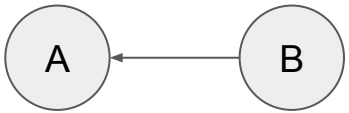
コミットCで行った変更を取り消して
コミットBの状態に戻したい

コミットの取り消し方法は2つある



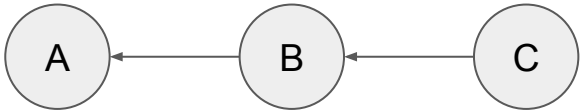
方法①：コミットCを削除する

コミットの取り消し方法は2つある



方法①：コミットCを削除する

コミットの取り消し方法は2つある

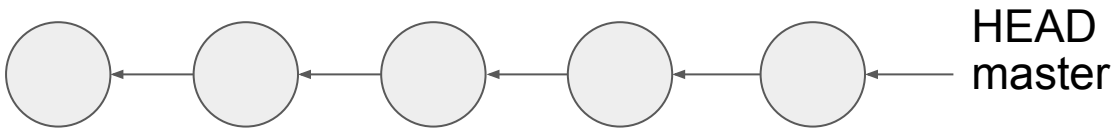


方法②：コミットCの変更を打ち消すコミットを追加する

コミットの取り消し方法は2つある

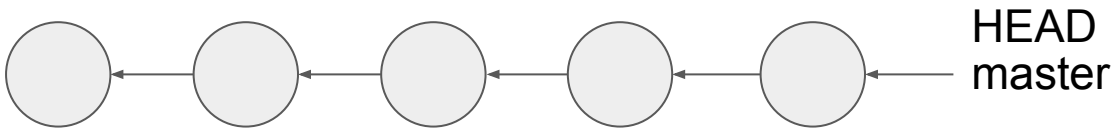
方法②：コミットCの変更を打ち消すコミットを追加する
コミットC’はコミットBと同じ内容

git resetとは



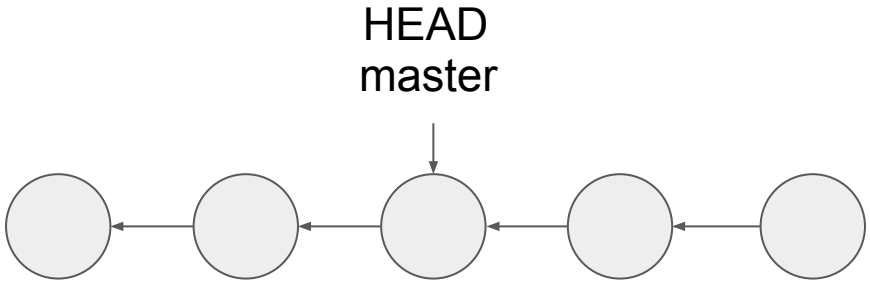
指定したコミットにHEADを移動するコマンド
HEADがブランチを指している場合は
HEADとともにブランチも移動するのが特徴

git resetとは



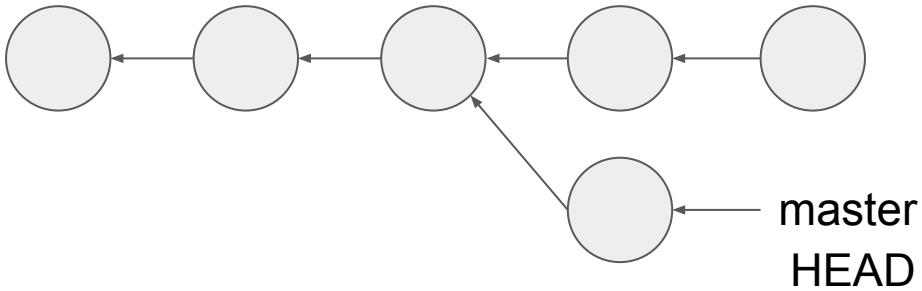
「git reset 移動先のコミット」を実行すると
指定したコミットにHEADとmasterが移動する

git resetとは



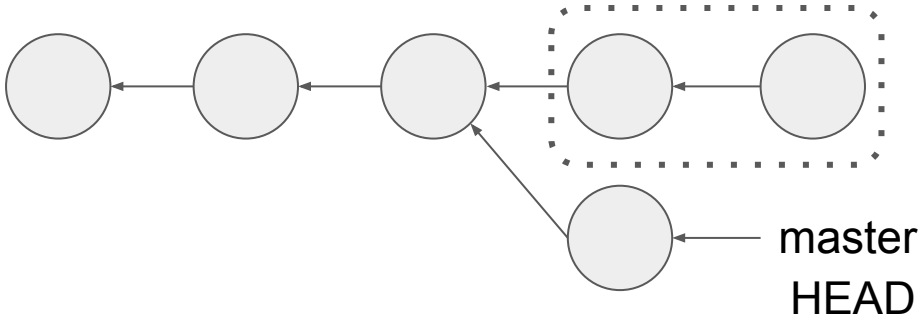
「git reset 移動先のコミット」を実行すると
指定したコミットにHEADとmasterが移動する

git resetとは



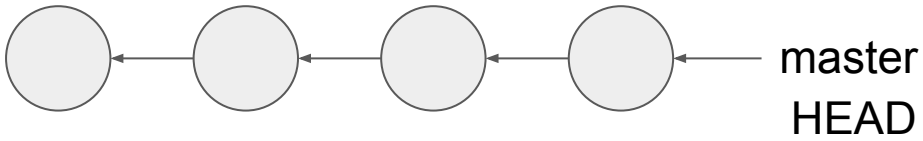
この状態でコミットすると新しくブランチを伸ばす

git resetとは



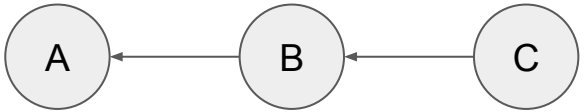
点線部分のコミットは捨てられる

git resetとは



点線部分のコミットは捨てられる

git revertとは

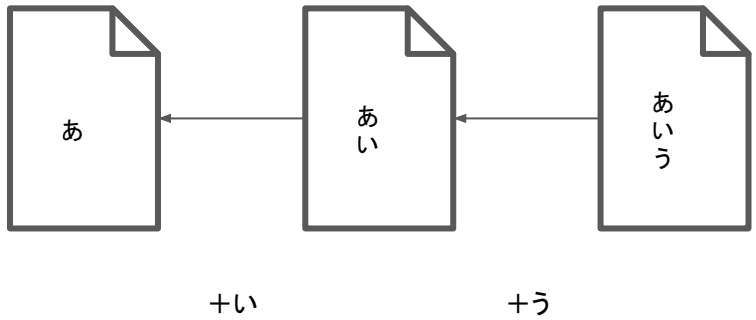


指定したコミットの変更を打ち消すコミットを追加することで
コミットを取り消すコマンド

git revertとは

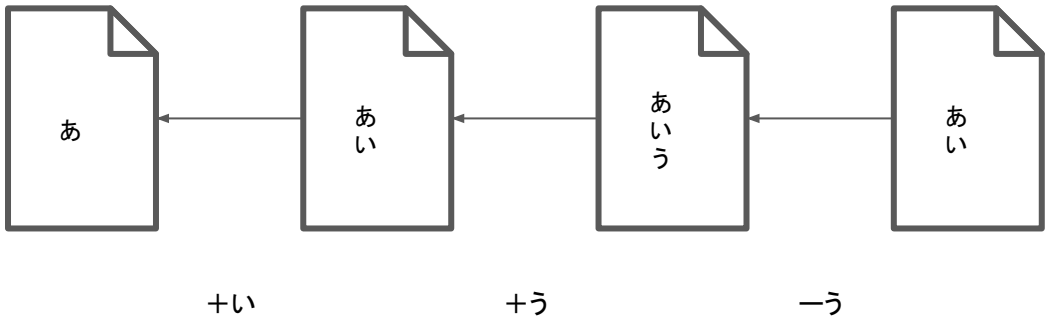
指定したコミットの変更を打ち消すコミットを追加することで
コミットを取り消すコマンド

git revertとは



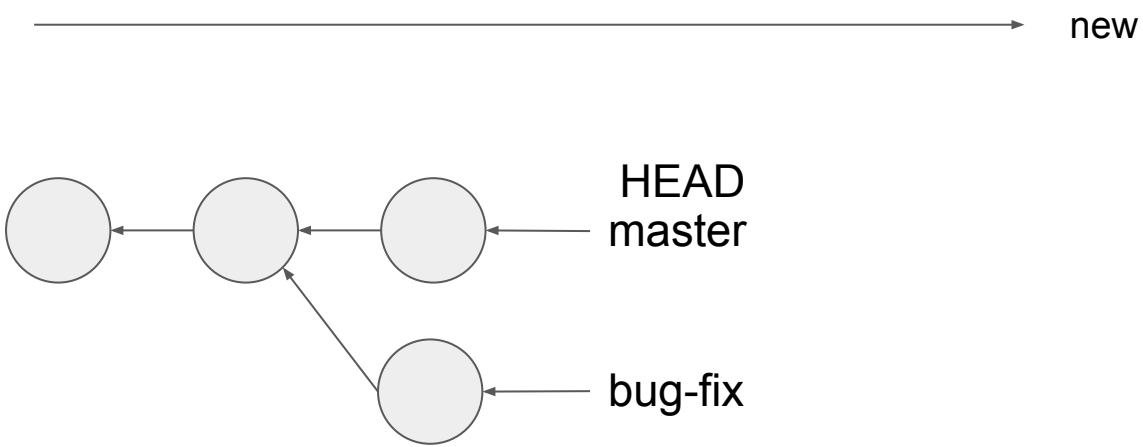
指定したコミットの変更を打ち消すコミットを追加することで
コミットを取り消すコマンド

git revertとは



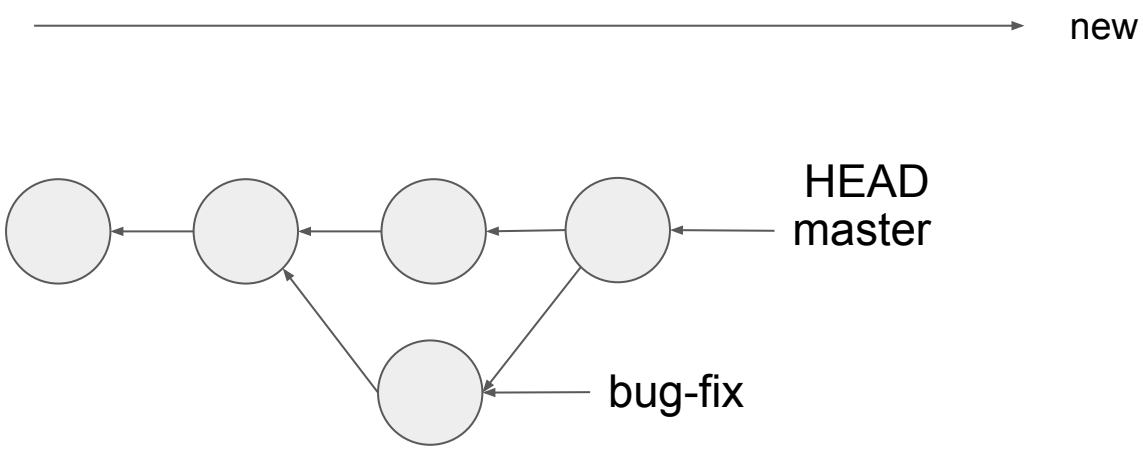
指定したコミットの変更を打ち消すコミットを追加することで
コミットを取り消すコマンド

3方向マージ（3-way merge）とは



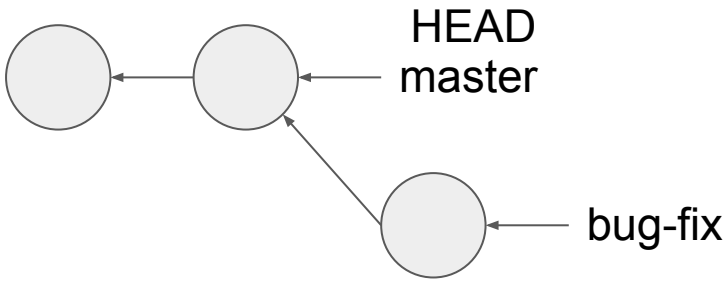
これまで見てきた通常のマージは「3-way merge」という
3つのコミットの内容を踏まえてマージされるため

3方向マージ（3-way merge）とは



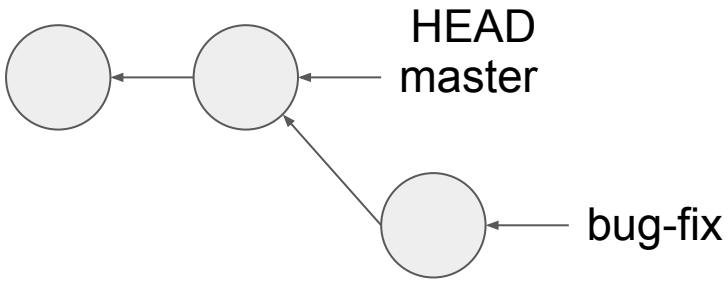
これまで見てきた通常のマージは「3-way merge」という
3つのコミットの内容を踏まえてマージされるため

早送りマージ（fast-foward merge）とは



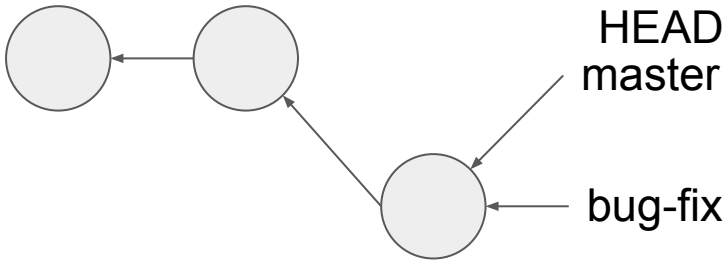
変更を取り込むブランチ（master）が指すコミットが
取り込まれるブランチ（bug-fix）の
祖先である場合のみ可能なマージ

早送りマージ（fast-foward merge）とは



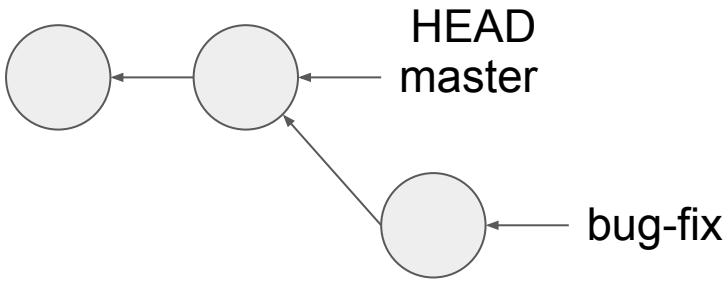
早送りマージは新たなコミットを作らずに
変更を取り込むブランチを移動するだけで済ませるマージ

早送りマージ（fast-foward merge）とは



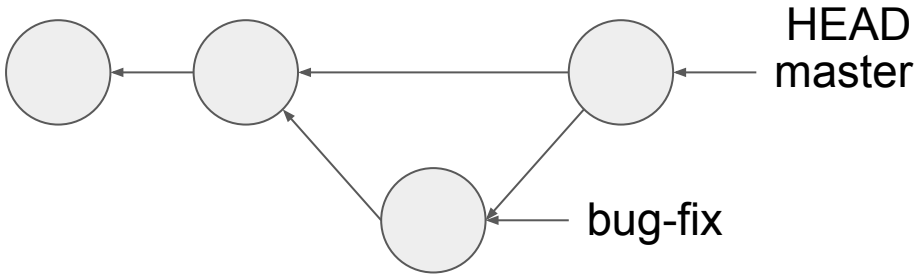
早送りマージは新たなコミットを作らずに
変更を取り込むブランチを移動するだけで済ませるマージ

早送りマージ（fast-foward merge）とは



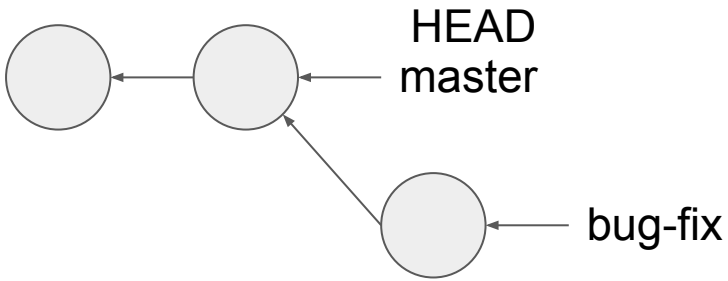
通常のマージを行うと新たなコミットが
変更を取り込むブランチに生成される

早送りマージ（fast-foward merge）とは



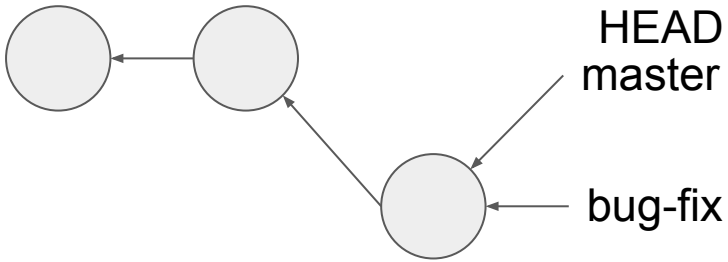
通常のマージを行うと新たなコミットが
変更を取り込むブランチに生成される

早送りマージ（fast-foward merge）とは



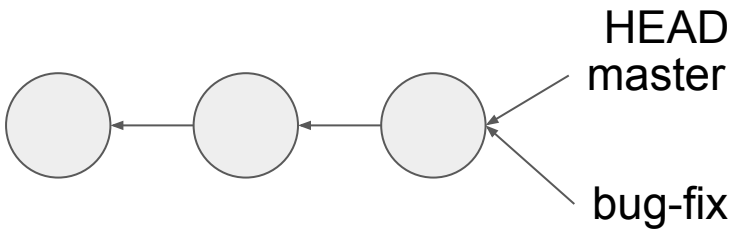
早送りマージは新たなコミットを作らずに
変更を取り込むブランチを移動するだけで済ませるマージ

早送りマージ（fast-foward merge）とは



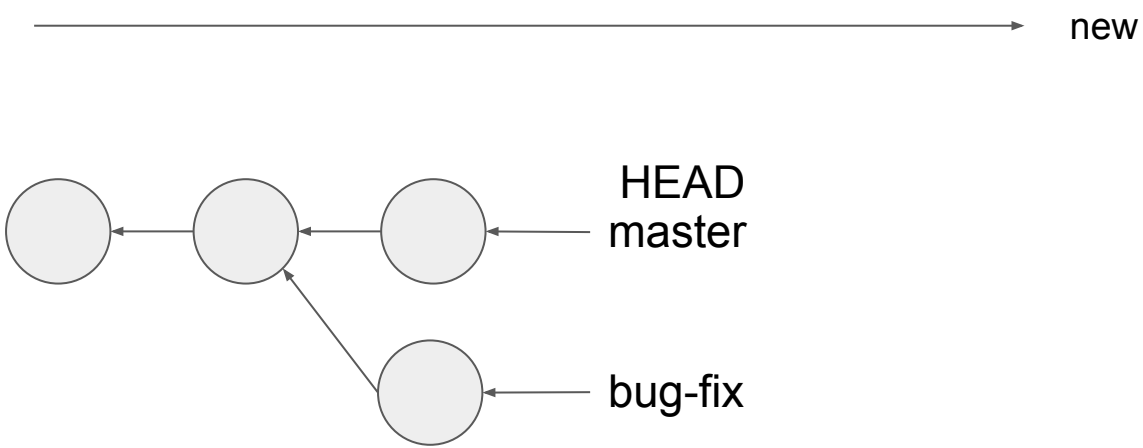
早送りマージは新たなコミットを作らずに
変更を取り込むブランチを移動するだけで済ませるマージ

早送りマージ（fast-foward merge）とは



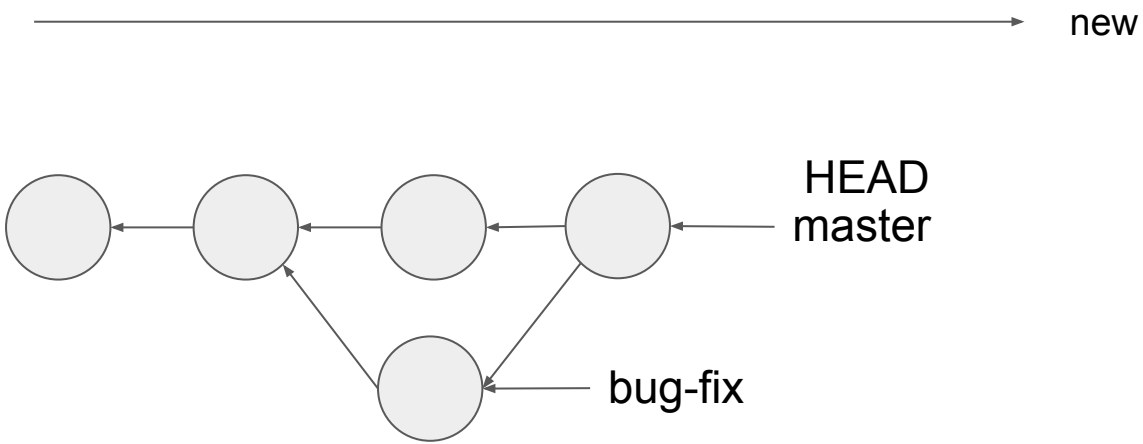
早送りマージは新たなコミットを作らずに
変更を取り込むブランチを移動するだけで済ませるマージ

ブランチを削除したらコミットも消える？



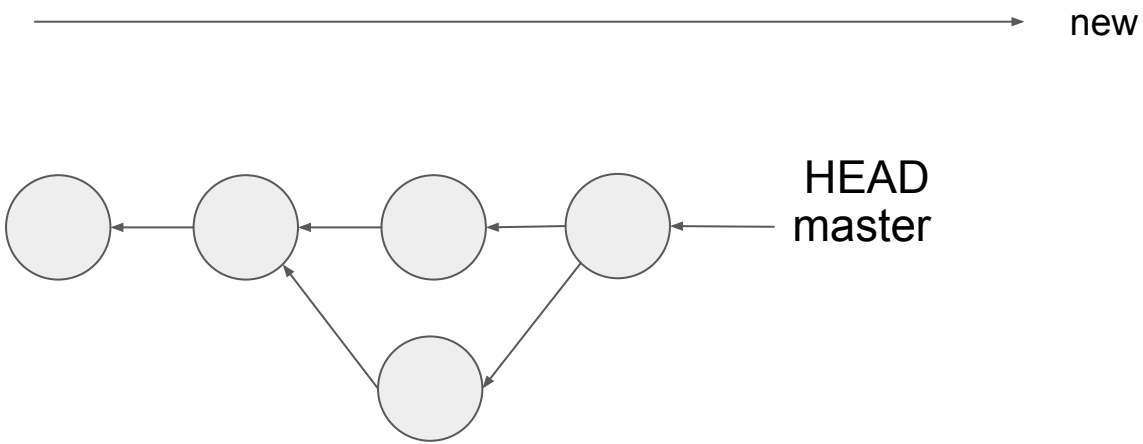
マージ後にbug-fixブランチを削除したとき、
bug-fixブランチのコミットにはアクセス出来なくなる？
それともアクセス出来る？

ブランチを削除したらコミットも消える？



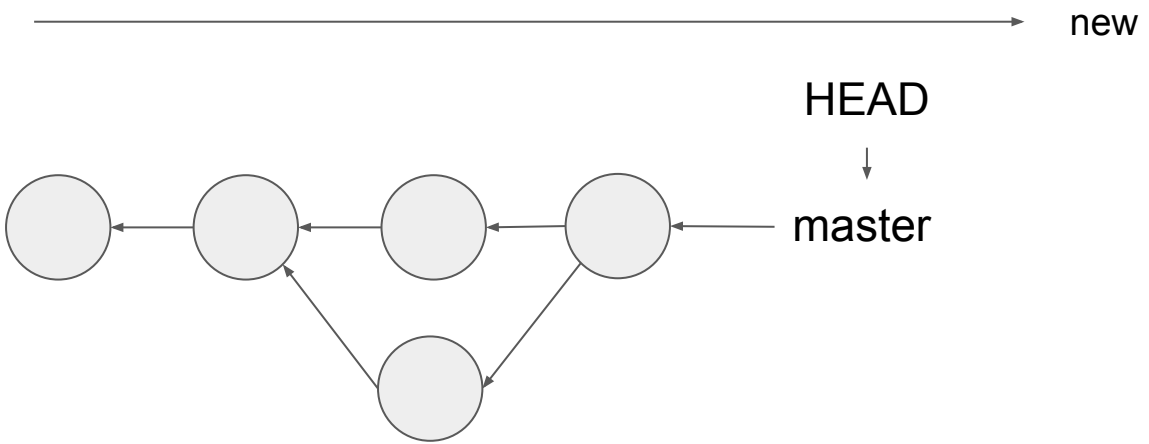
マージ後にbug-fixブランチを削除したとき、
bug-fixブランチのコミットにはアクセス出来なくなる？
それともアクセス出来る？

ブランチを削除したらコミットも消える？



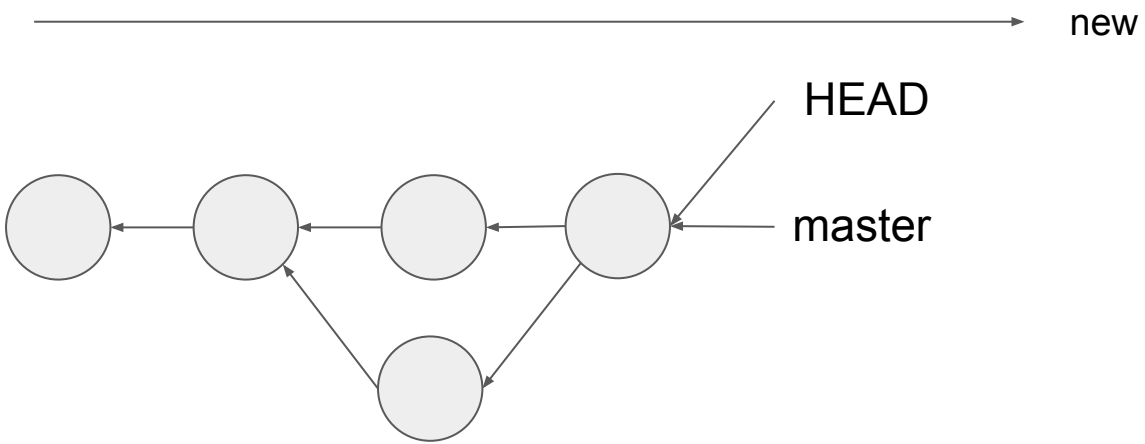
マージ後にbug-fixブランチを削除したとき、
bug-fixブランチのコミットにはアクセス出来なくなる？
それともアクセス出来る？

ブランチとHEADが同じコミットを指すと？



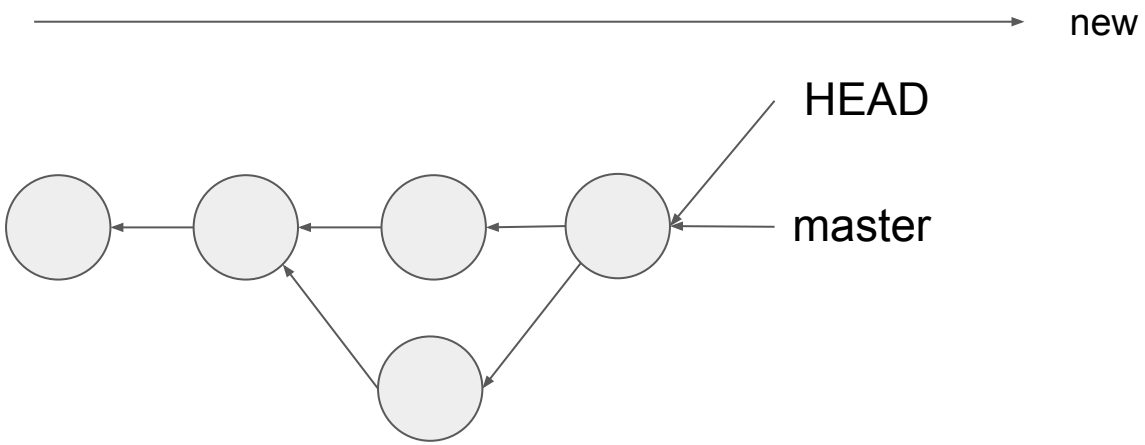
HEADがブランチを指すのではなく、
HEADがブランチと同じコミットを指している場合、
これはdetached HEADかどうか？

ブランチとHEADが同じコミットを指すと？



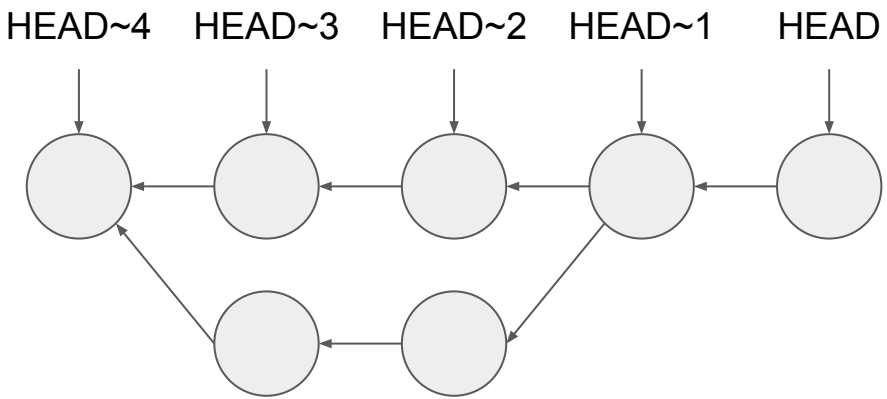
HEADがブランチを指すのではなく、
HEADがブランチと同じコミットを指している場合、
これはdetached HEADかどうか？

ブランチとHEADが同じコミットを指すと？



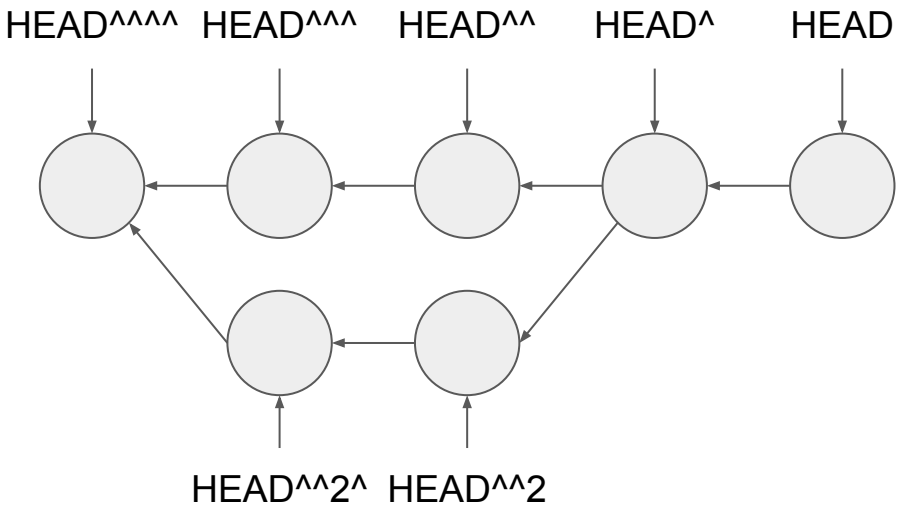
答え：detached HEADになる
HEADがどのブランチも指していない場合はdetached HEAD

コミットの指定方法：~（チルダ）と^（キャレット）



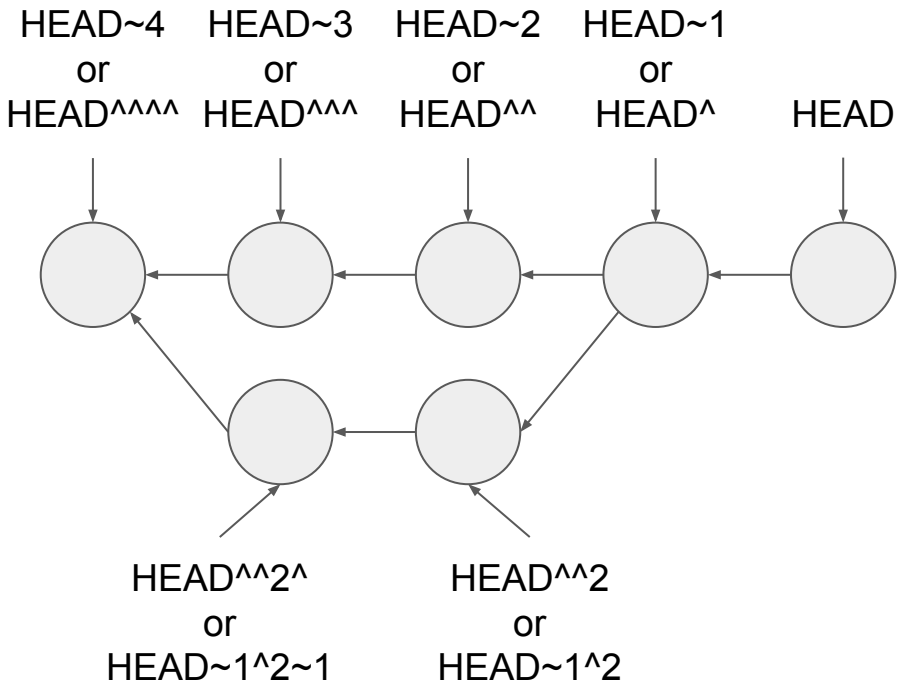
~（チルダ）は**HEADの何世代前の親か？**を表す
「~2」で2世代前、「~3」で3世代前

コミットの指定方法：~（チルダ）と^（キャレット）

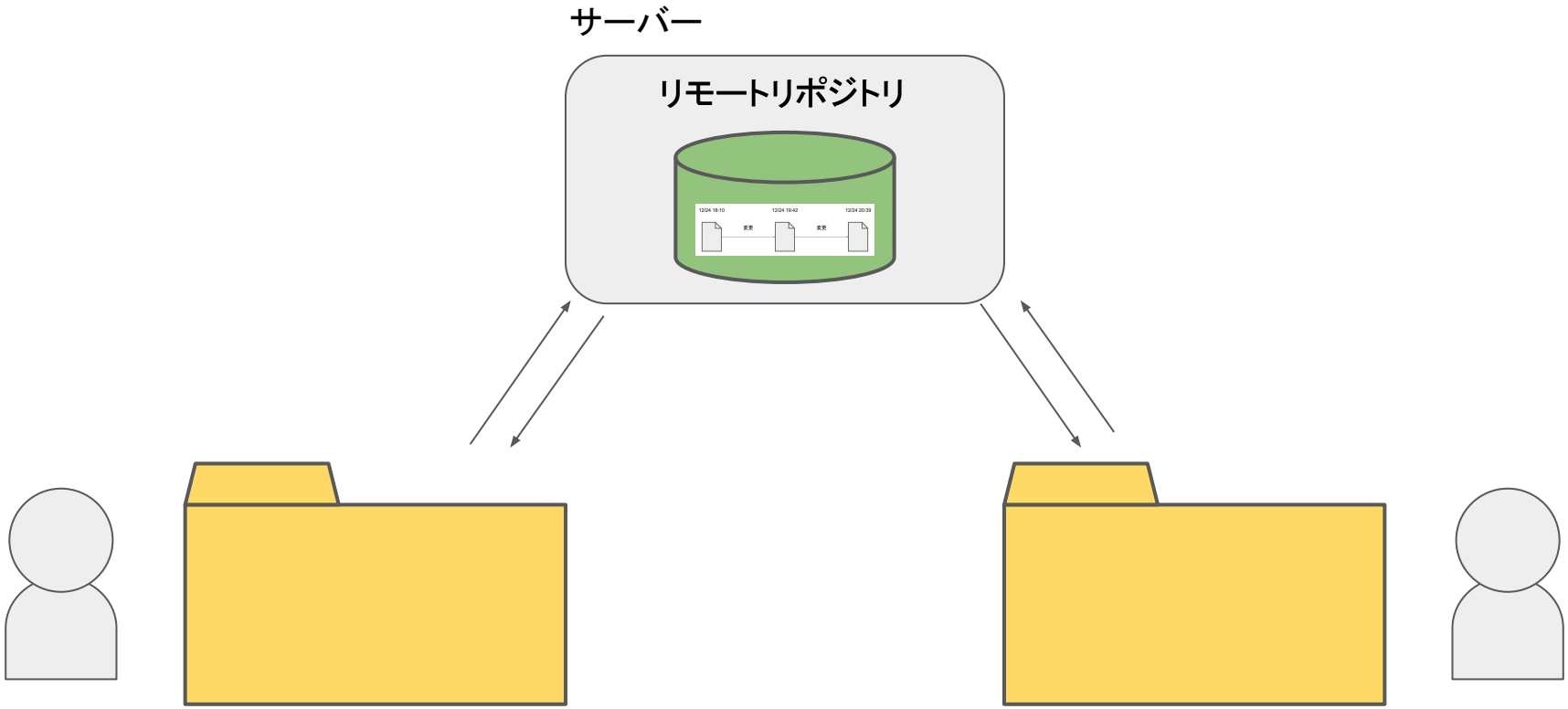


^（キャレット）は**親が複数ある場合何番目の親か？**を表す
「^」もしくは「^1」で1番目の親、「^2」で2番目の親

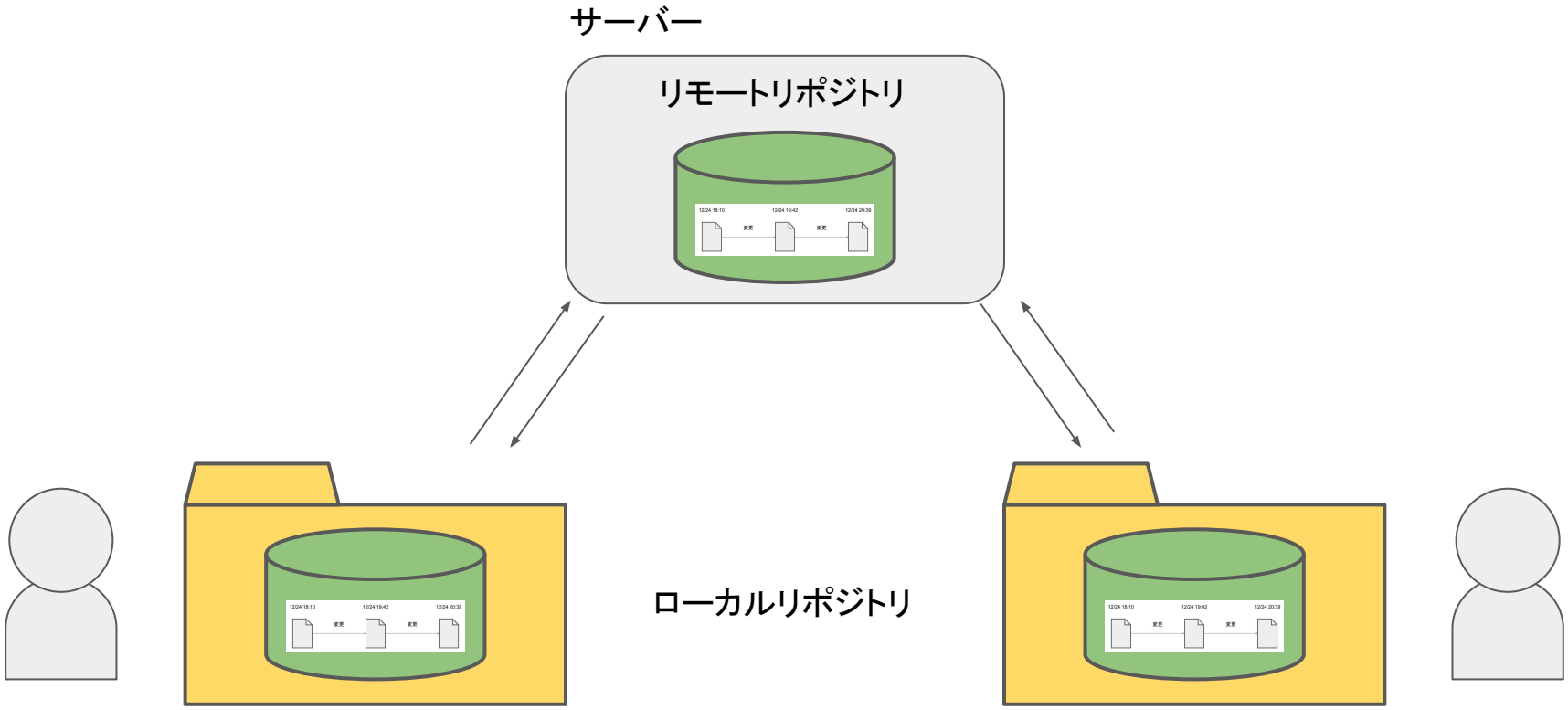
コミットの指定方法：~（チルダ）と^（キャレット）



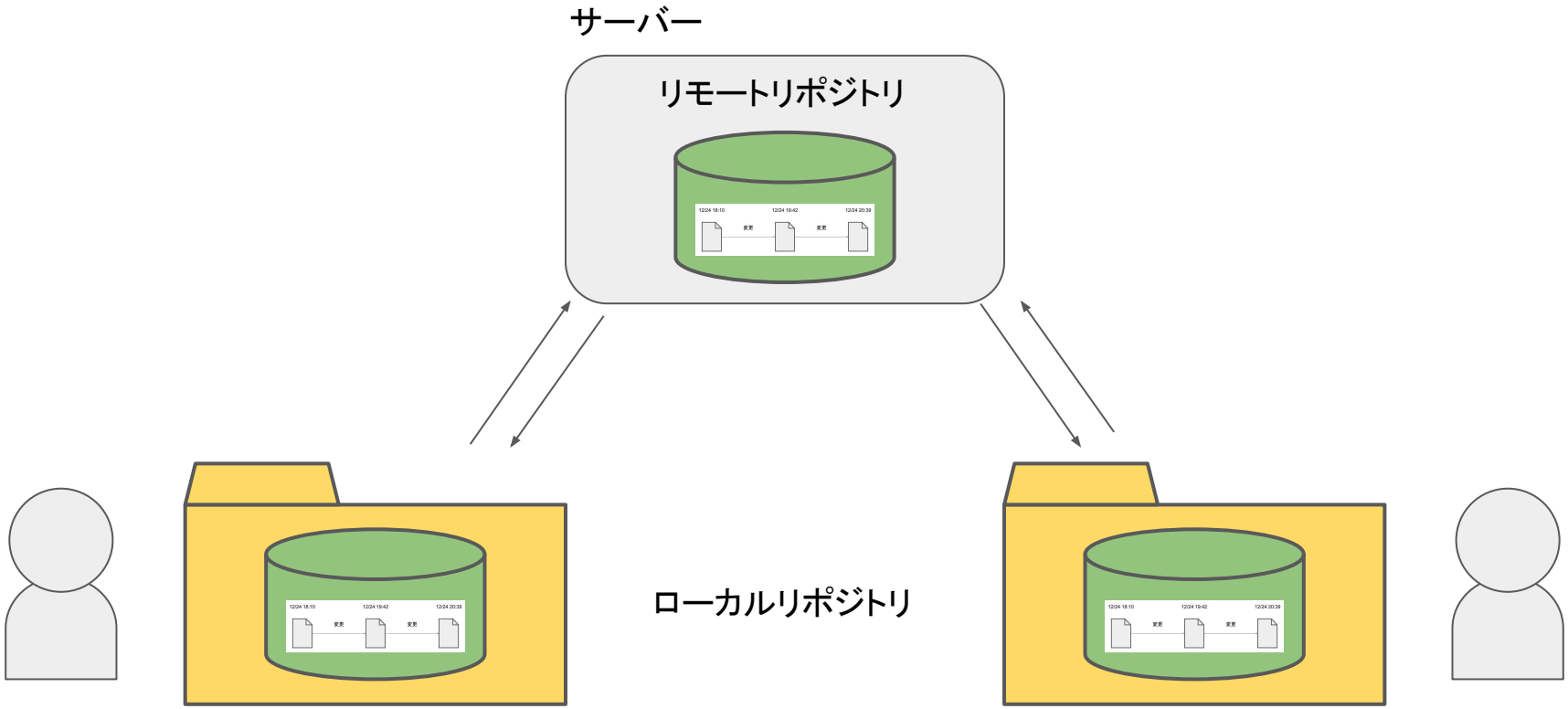
集中型のバージョン管理



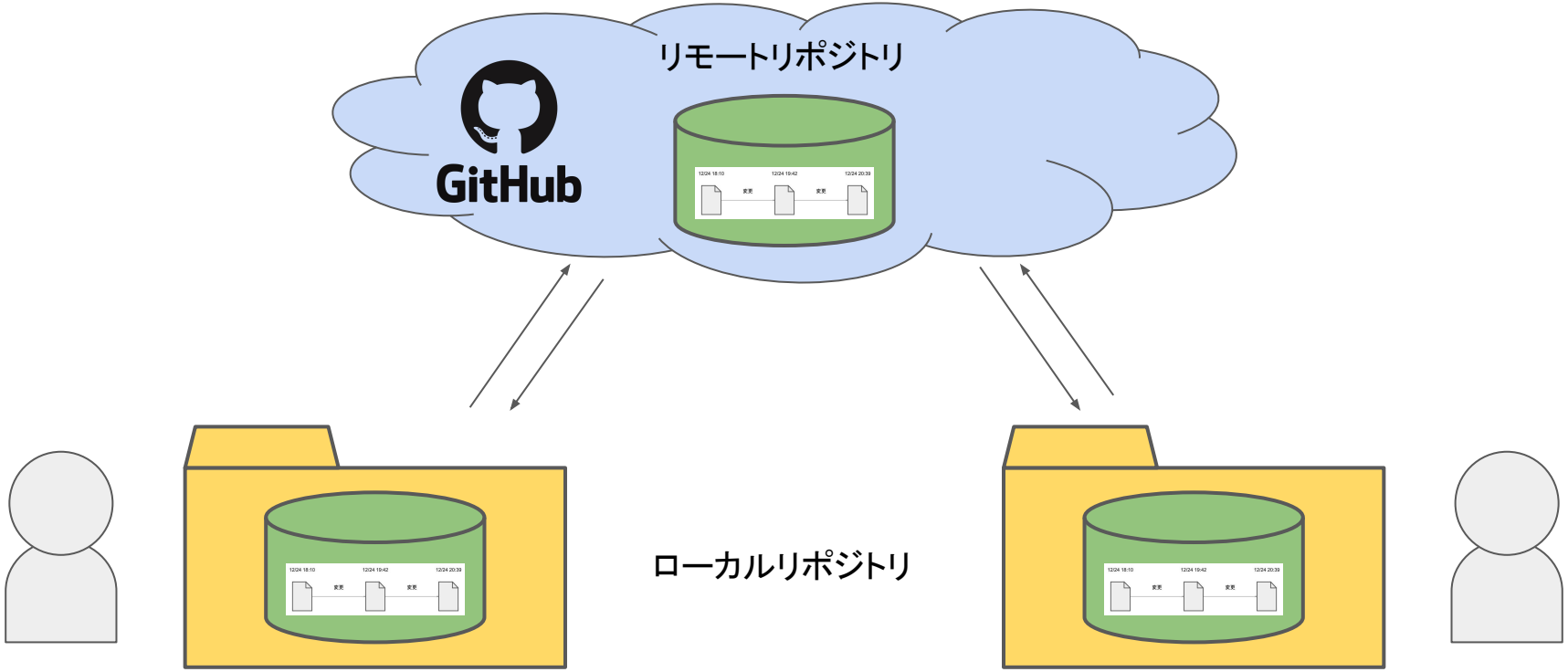
分散型のバージョン管理



GitHub上にリモートリポジトリを配置する

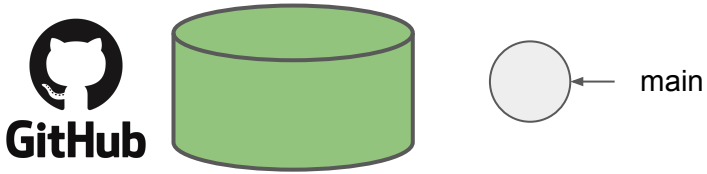


GitHub上にリモートリポジトリを配置する



GitHubを用いた開発フロー

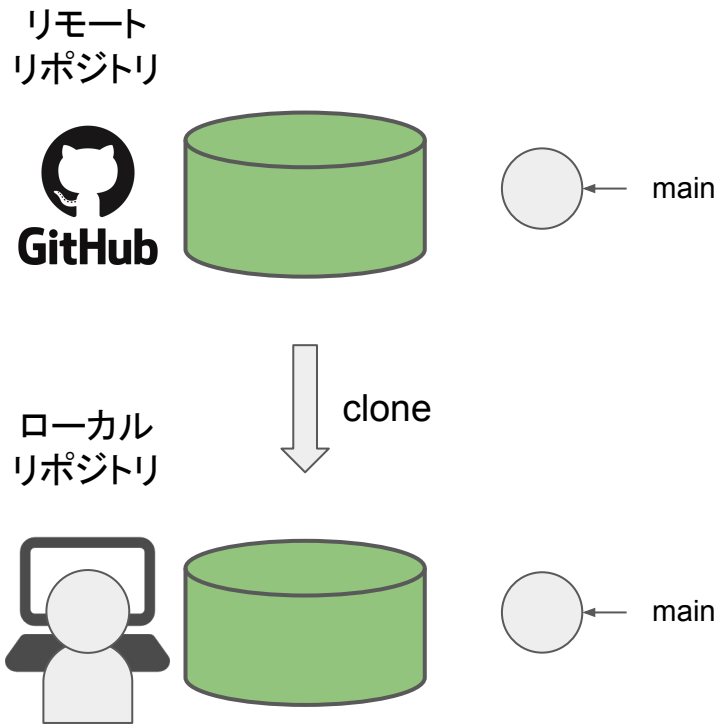
リモート
リポジトリ



ローカル
リポジトリ

1) リモートリポジトリをGitHubで作成

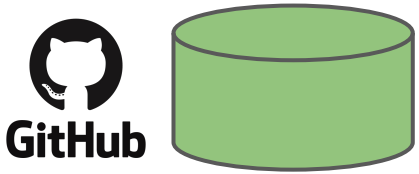
GitHubを用いた開発フロー

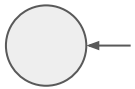


1) リモートリポジトリをGitHubで作成
2) リモートリポジトリをコピーして、ローカルリポジトリ作（clone）

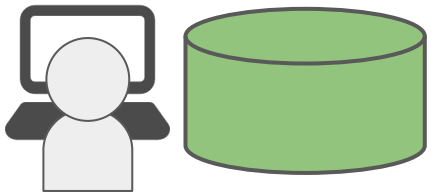
GitHubを用いた開発フロー

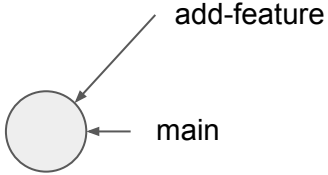
リモート
リポジトリ





ローカル
リポジトリ

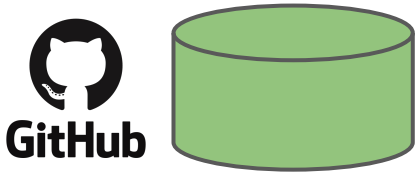


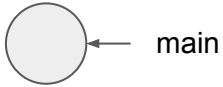


1) リモートリポジトリをGitHubで作成
2) リモートリポジトリをコピーして、ローカルリポジトリ作（clone）
3) ローカルリポジトリでブランチ（add-feature）を作成する

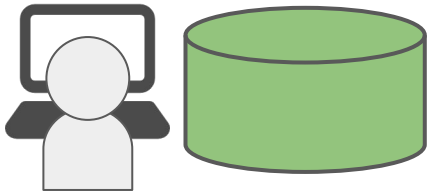
GitHubを用いた開発フロー

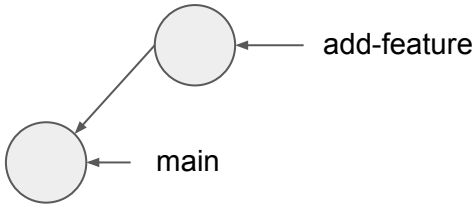
リモート
リポジトリ





ローカル
リポジトリ

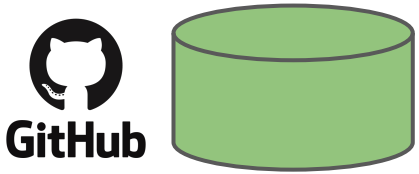


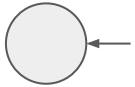


1) リモートリポジトリをGitHubで作成
2) リモートリポジトリをコピーして、ローカルリポジトリ作（clone）
3) ローカルリポジトリでブランチ（add-feature）を作成する
4) ファイルを編集して、作成したブランチにコミットする

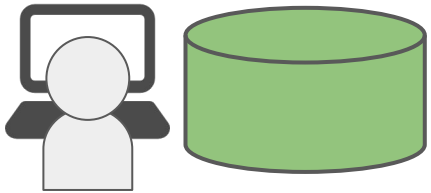
GitHubを用いた開発フロー

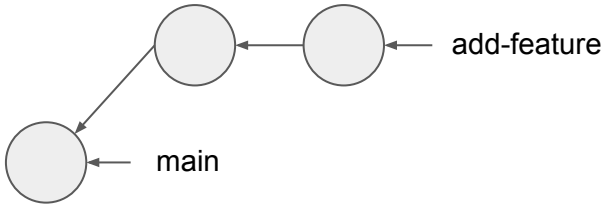
リモート
リポジトリ





ローカル
リポジトリ

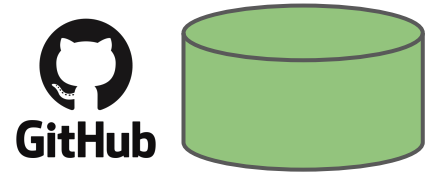


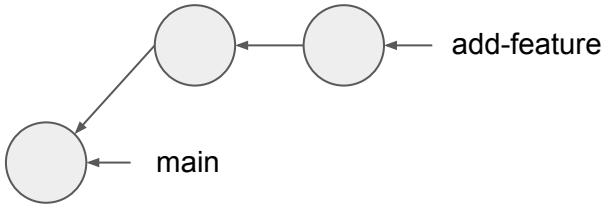


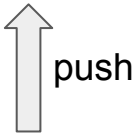
1) リモートリポジトリをGitHubで作成
2) リモートリポジトリをコピーして、ローカルリポジトリ作（clone）
3) ローカルリポジトリでブランチ（add-feature）を作成する
4) ファイルを編集して、作成したブランチにコミットする

GitHubを用いた開発フロー

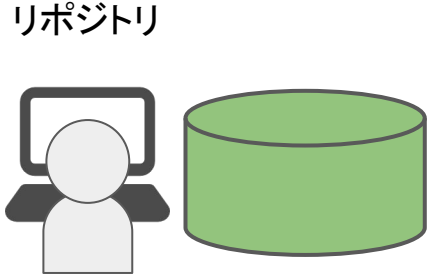
リモート
リポジトリ

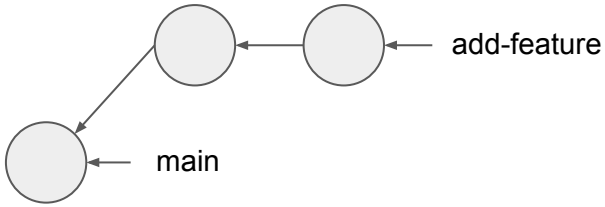






ローカル
リポジトリ

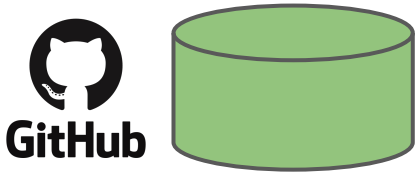


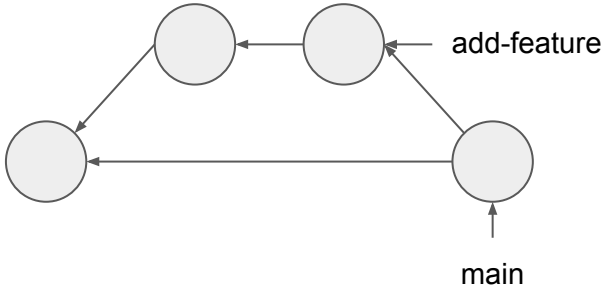


1) リモートリポジトリをGitHubで作成
2) リモートリポジトリをコピーして、ローカルリポジトリ作（clone）
3) ローカルリポジトリでブランチ（add-feature）を作成する
4) ファイルを編集して、作成したブランチにコミットする
5) ローカルリポジトリの内容をリモートリポジトリに反映する（push）

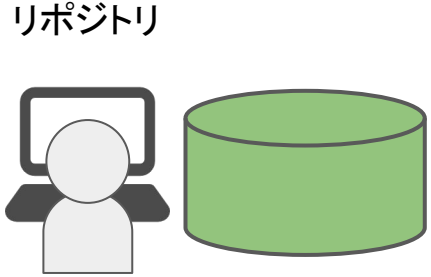
GitHubを用いた開発フロー

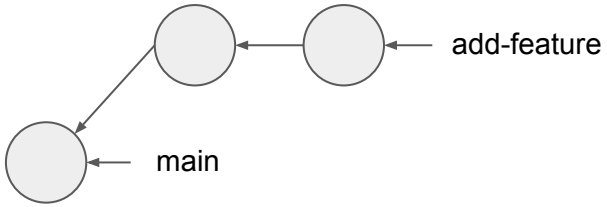
リモート
リポジトリ





ローカル
リポジトリ

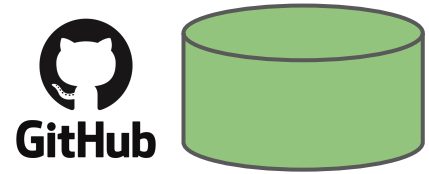


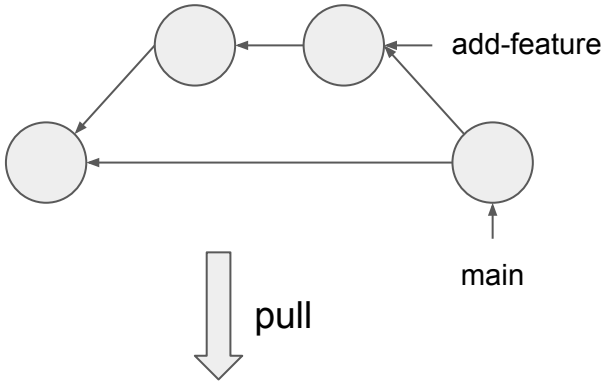


1) リモートリポジトリをGitHubで作成
2) リモートリポジトリをコピーして、ローカルリポジトリ作(clone)
3) ローカルリポジトリでブランチ(add-feature)を作成する
4) ファイルを編集して、作成したブランチにコミットする
5) ローカルリポジトリの内容をリモートリポジトリに反映する(push)
6) GitHubでブランチの内容を確認して良ければマージする

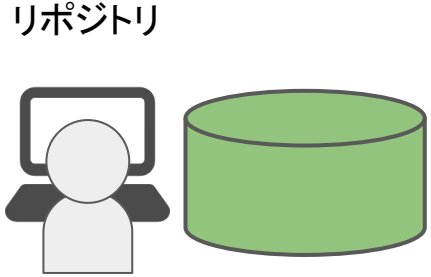
GitHubを用いた開発フロー

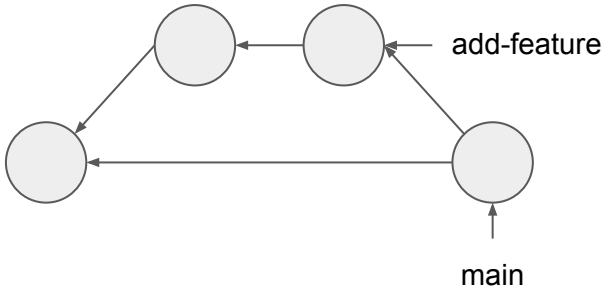
リモート
リポジトリ





ローカル
リポジトリ

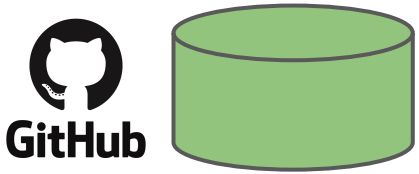


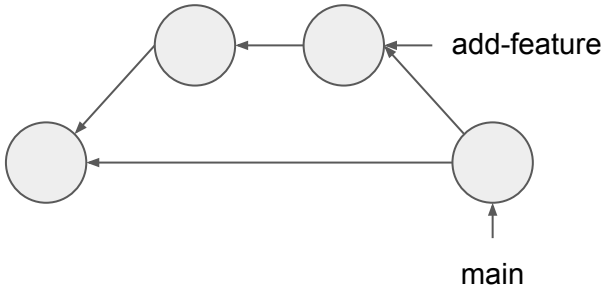


1) リモートリポジトリをGitHubで作成
2) リモートリポジトリをコピーして、ローカルリポジトリ作(clone)
3) ローカルリポジトリでブランチ(add-feature)を作成する
4) ファイルを編集して、作成したブランチにコミットする
5) ローカルリポジトリの内容をリモートリポジトリに反映する(push)
6) GitHubでブランチの内容を確認して良ければマージする
7) リモートリポジトリの内容をローカルリポジトリに反映する(pull)

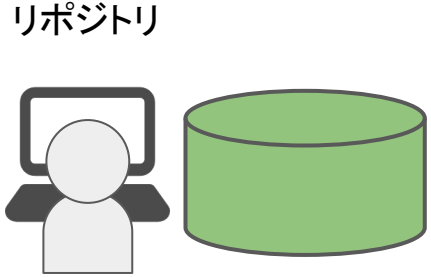
GitHubを用いた開発フロー

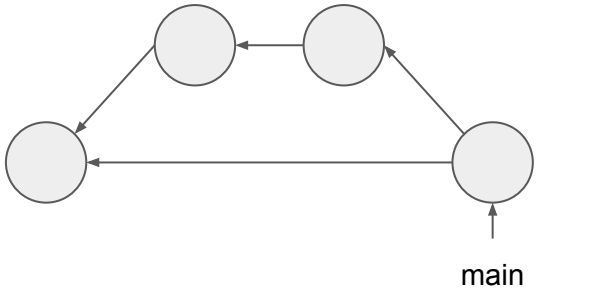
リモート
リポジトリ





ローカル
リポジトリ

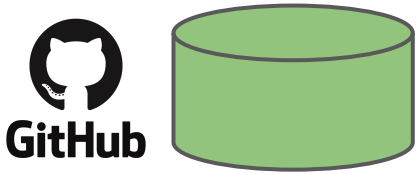


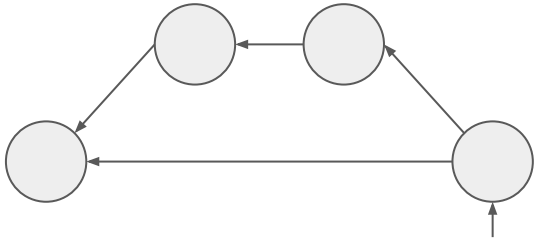


1) リモートリポジトリをGitHubで作成
2) リモートリポジトリをコピーして、ローカルリポジトリ作(clone)
3) ローカルリポジトリでブランチ(add-feature)を作成する
4) ファイルを編集して、作成したブランチにコミットする
5) ローカルリポジトリの内容をリモートリポジトリに反映する(push)
6) GitHubでブランチの内容を確認して良ければマージする
7) リモートリポジトリの内容をローカルリポジトリに反映する(pull)
8) 不要になったトピックブランチを削除する

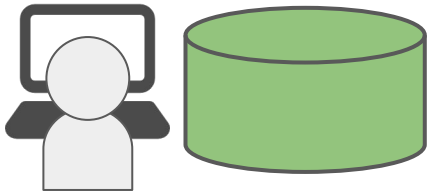
GitHubを用いた開発フロー

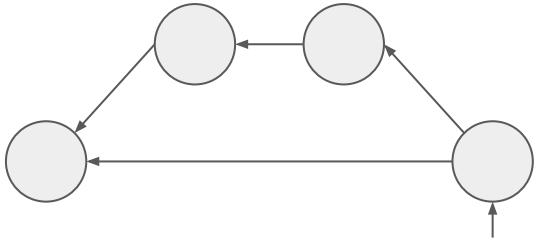
リモート
リポジトリ





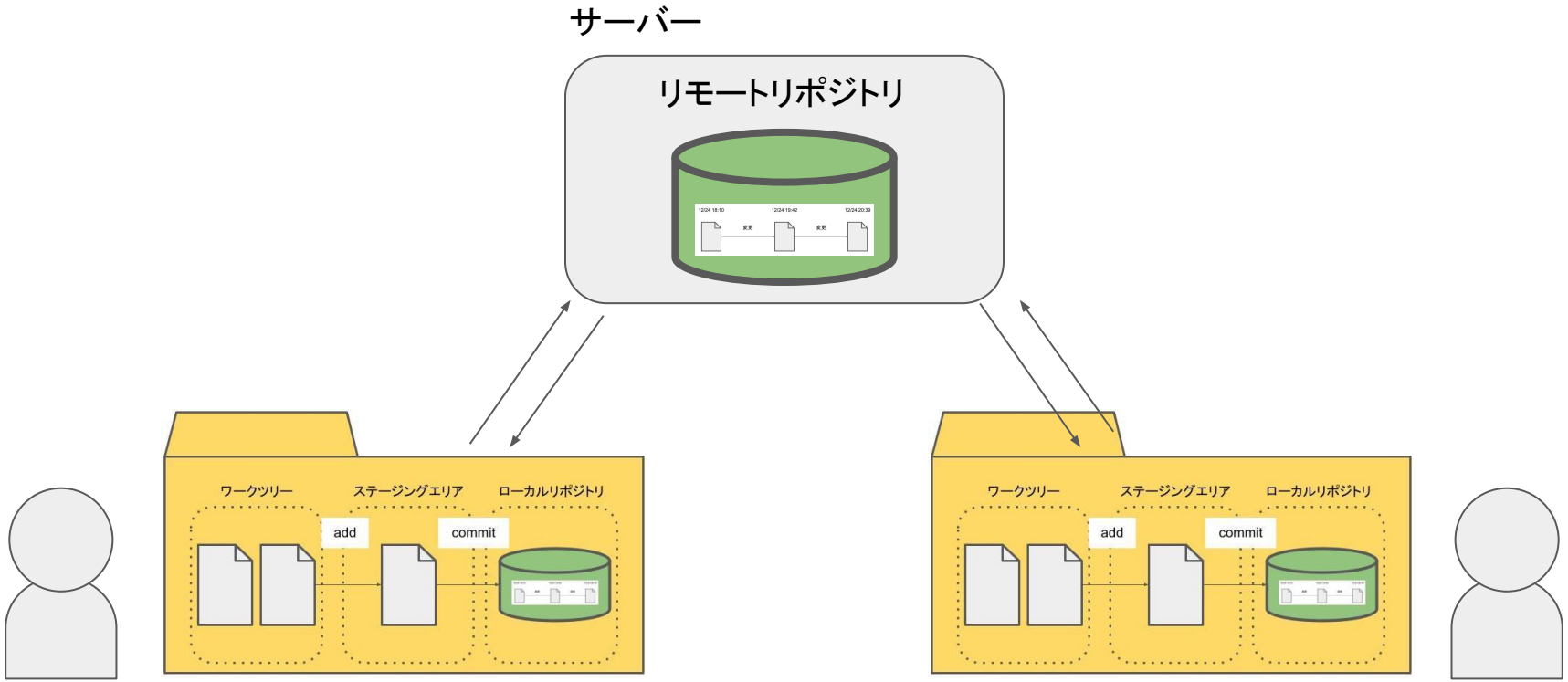
ローカル
リポジトリ



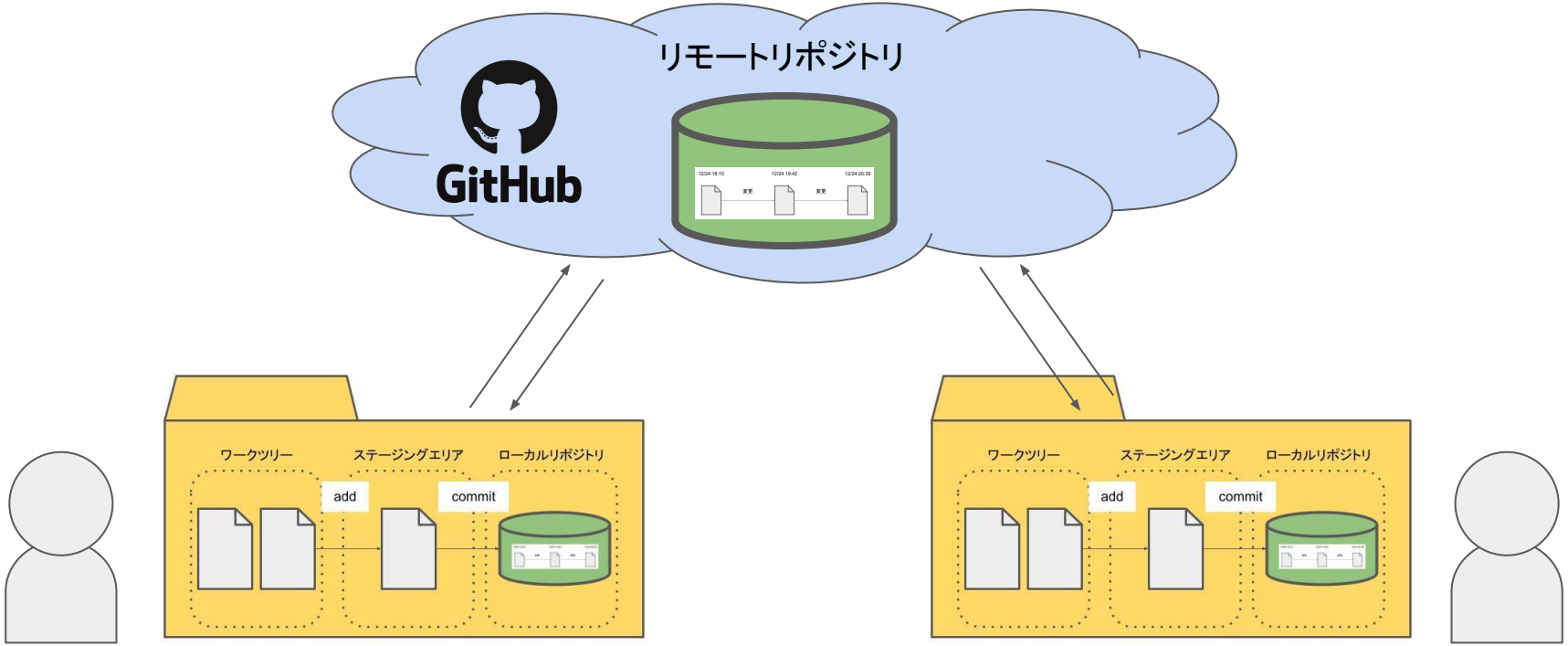


1) リモートリポジトリをGitHubで作成
2) リモートリポジトリをコピーして、ローカルリポジトリ作(clone)
3) ローカルリポジトリでブランチ(add-feature)を作成する
4) ファイルを編集して、作成したブランチにコミットする
5) ローカルリポジトリの内容をリモートリポジトリに反映する(push)
6) GitHubでブランチの内容を確認して良ければマージする
7) リモートリポジトリの内容をローカルリポジトリに反映する(pull)
8) 不要になったトピックブランチを削除する

Gitのリモートリポジトリ



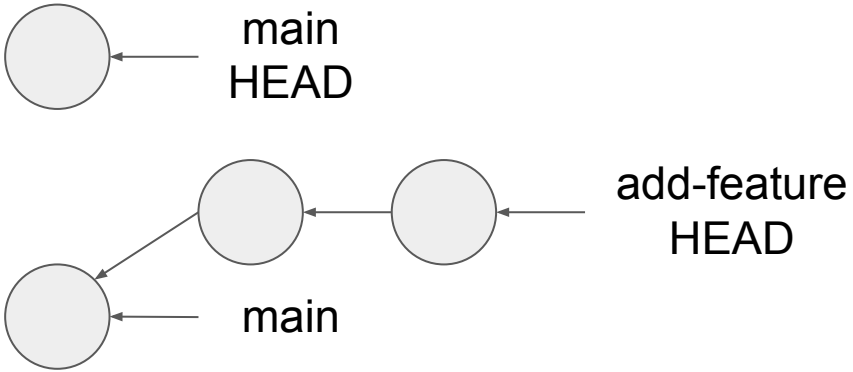
GitとGitHubによる共同作業



プッシュ（push）とは

リモート
リポジトリ

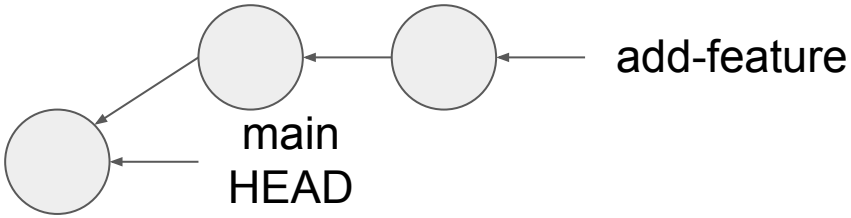
ローカル
リポジトリ



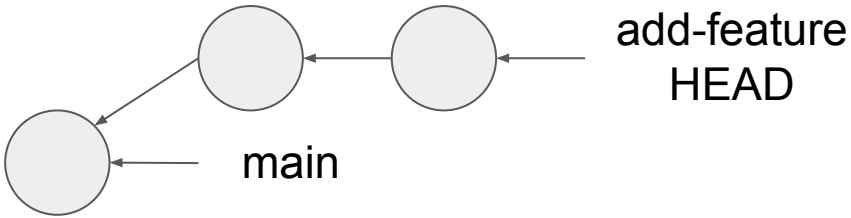
ローカルリポジトリのコミット履歴を
リモートリポジトリに反映する

プッシュ（push）とは

リモート
リポジトリ



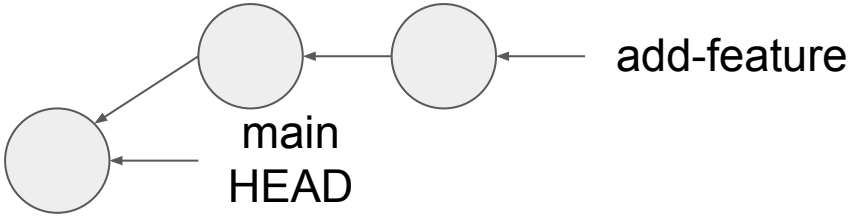
ローカル
リポジトリ



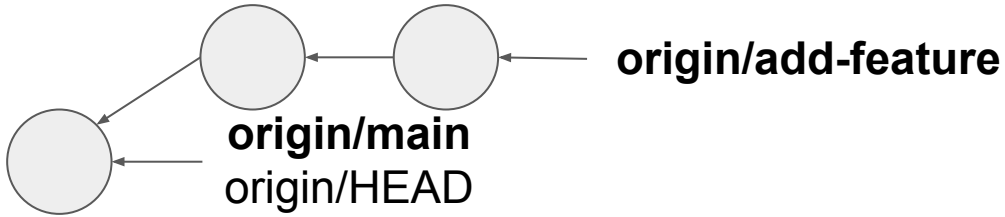
ローカルリポジトリのコミット履歴を
リモートリポジトリに反映する

リモート追跡ブランチとは

リモート
（origin）



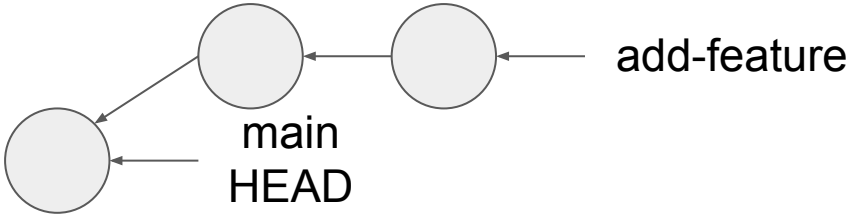
ローカル



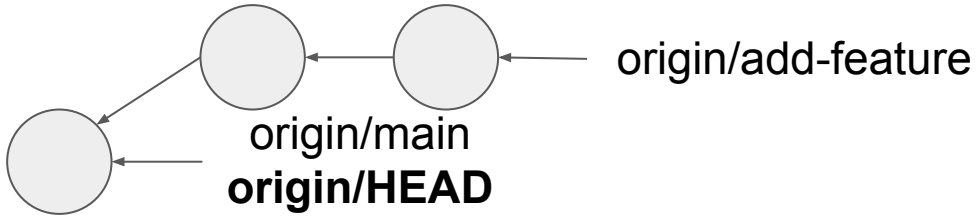
リモートリポジトリのブランチの位置を記憶する
ローカルリポジトリにあるブランチのこと

リモート追跡ブランチとは

リモート
（origin）



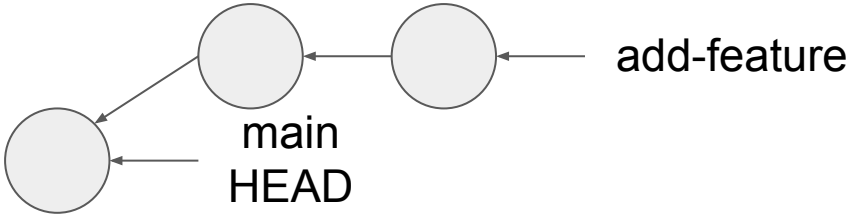
ローカル



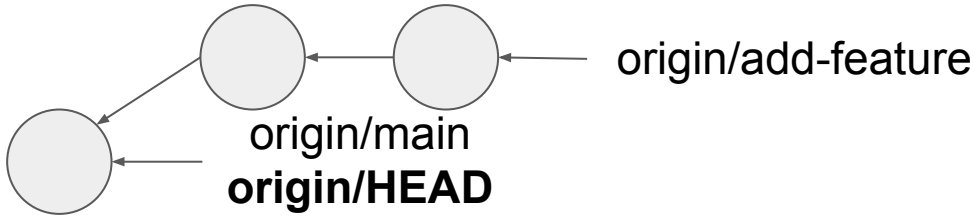
リモートリポジトリのHEADの位置を記憶する
origin/HEADというポインタもある

リモート追跡ブランチとは

リモート
（origin）



ローカル

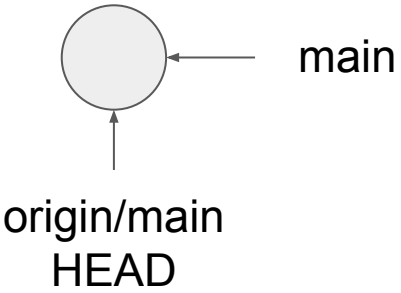


通常、origin/HEADはメインブランチの先頭のコミットを指す
このポインタを意識する必要のあるケースはほぼない

リモート追跡ブランチはコミットしても伸ばせない

リモート
（origin）

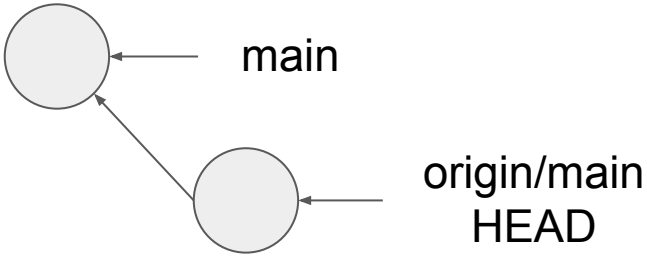
ローカル



リモート追跡ブランチはコミットしても伸ばせない

リモート
（origin）

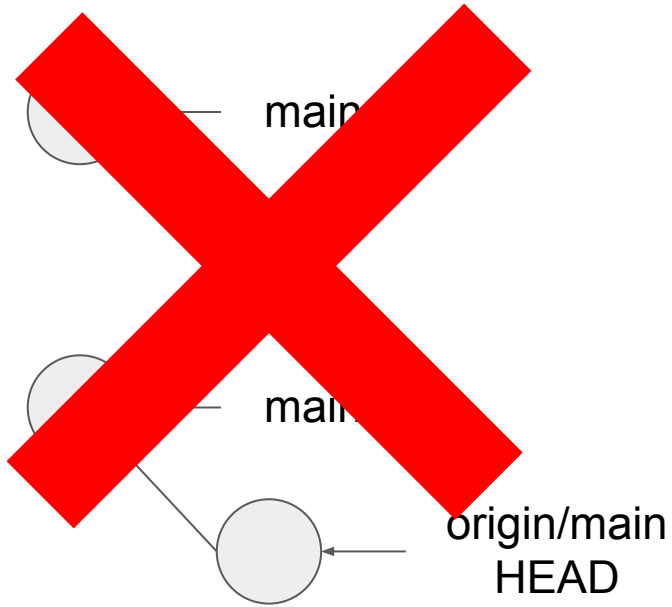
ローカル



リモート追跡ブランチはコミットしても伸ばせない

リモート
（origin）

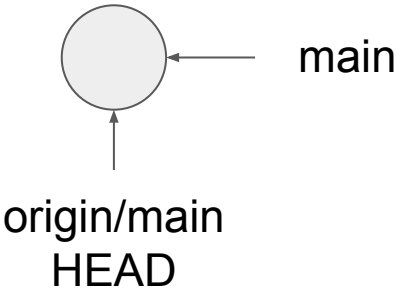
ローカル



リモート追跡ブランチはコミットしても伸ばせない

リモート
（origin）

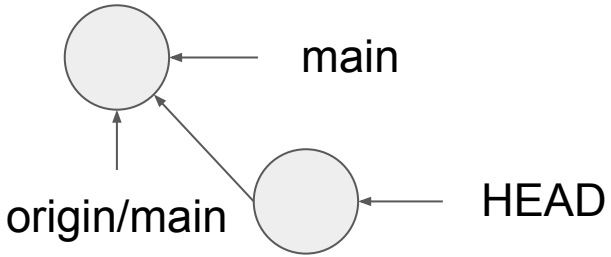
ローカル



リモート追跡ブランチはコミットしても伸ばせない

リモート
（origin）

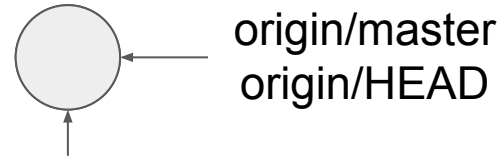
ローカル



フェッチ（fetch）とは

リモート
リポジトリ

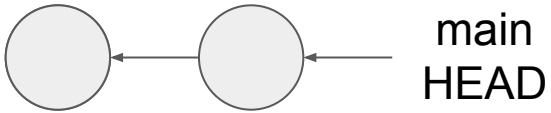
ローカル
リポジトリ



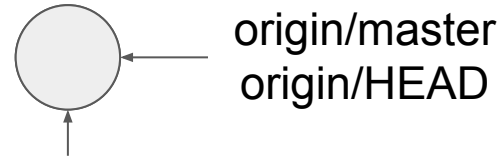
リモートリポジトリのコミット履歴を
ローカルリポジトリに反映する

フェッチ（fetch）とは

リモート
リポジトリ



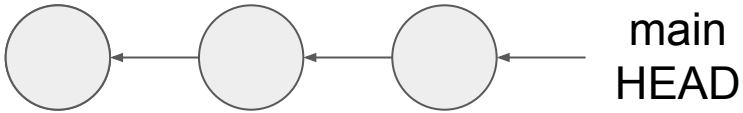
ローカル
リポジトリ



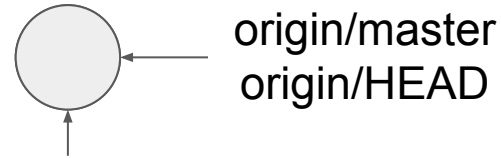
リモートリポジトリのコミット履歴を
ローカルリポジトリに反映する

フェッチ（fetch）とは

リモート
リポジトリ



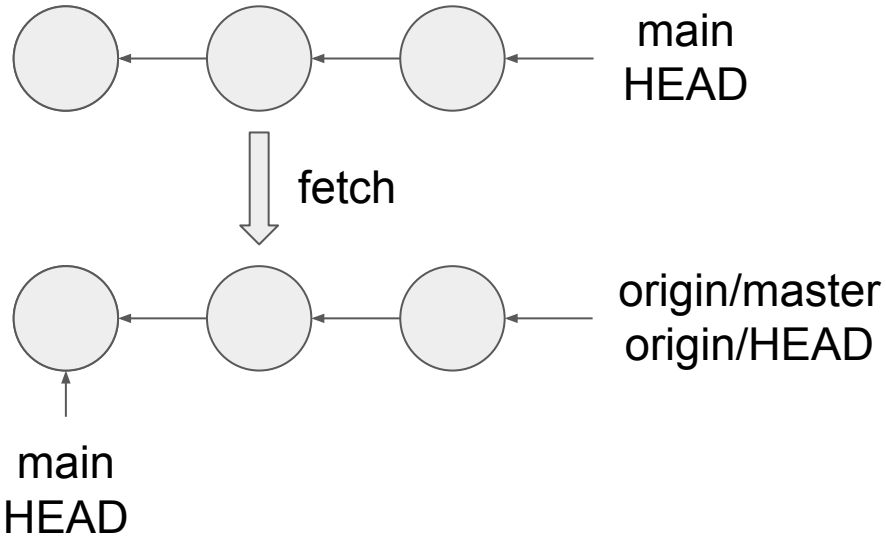
ローカル
リポジトリ



リモートリポジトリのコミット履歴を
ローカルリポジトリに反映する

フェッチ（fetch）とは

リモート
リポジトリ

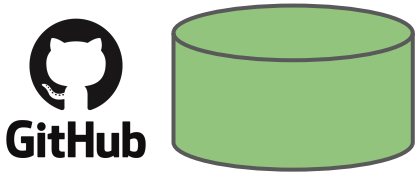


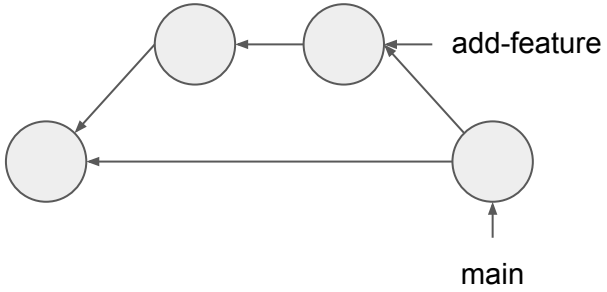
ローカル
リポジトリ

リモートリポジトリのコミット履歴を
ローカルリポジトリに反映する

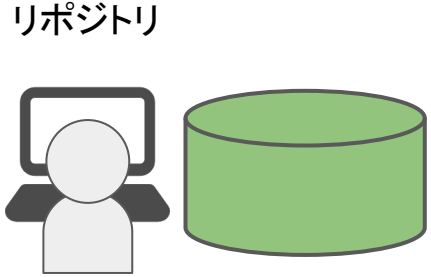
GitHubを用いた開発フロー

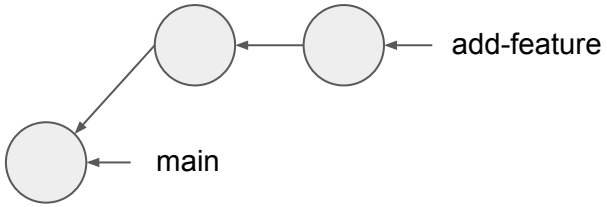
リモート
リポジトリ





ローカル
リポジトリ

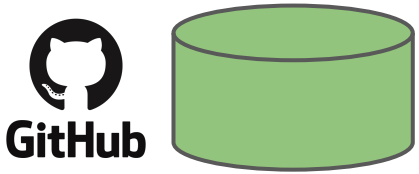


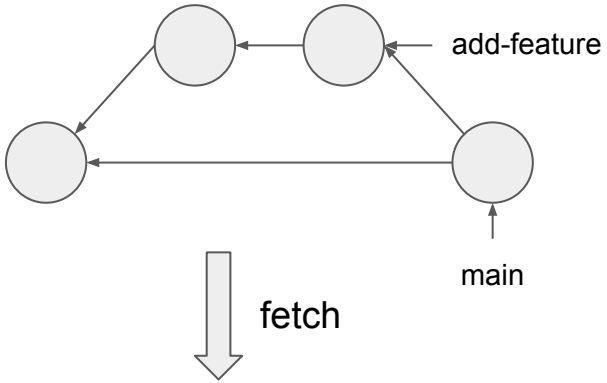


1) リモートリポジトリをGitHubで作成
2) リモートリポジトリをコピーして、ローカルリポジトリ作(clone)
3) ローカルリポジトリでブランチ(add-feature)を作成する
4) ファイルを編集して、作成したブランチにコミットする
5) ローカルリポジトリの内容をリモートリポジトリに反映する(push)
6) GitHubでブランチの内容を確認して良ければマージする

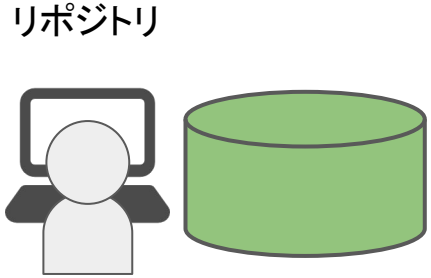
GitHubを用いた開発フロー

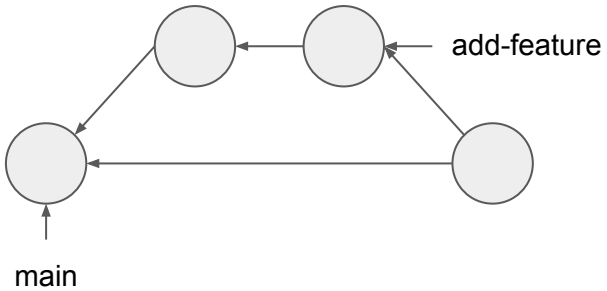
リモート
リポジトリ





ローカル
リポジトリ

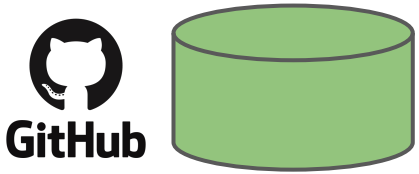


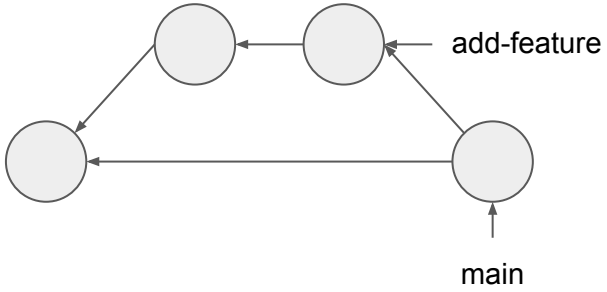


1) リモートリポジトリをGitHubで作成
2) リモートリポジトリをコピーして、ローカルリポジトリ作(clone)
3) ローカルリポジトリでブランチ(add-feature)を作成する
4) ファイルを編集して、作成したブランチにコミットする
5) ローカルリポジトリの内容をリモートリポジトリに反映する(push)
6) GitHubでブランチの内容を確認して良ければマージする
7) リモートリポジトリの内容をローカルリポジトリに反映する(pull)

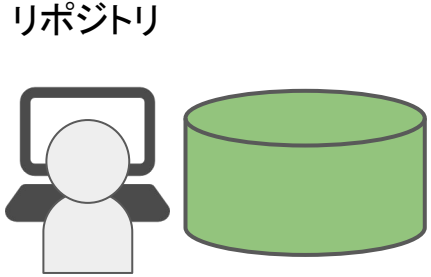
GitHubを用いた開発フロー

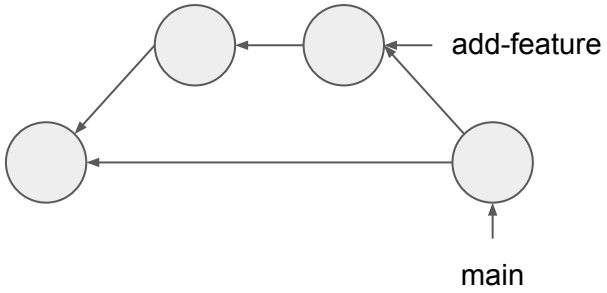
リモート
リポジトリ





ローカル
リポジトリ

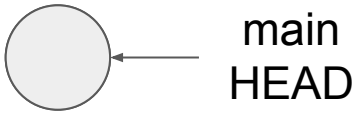




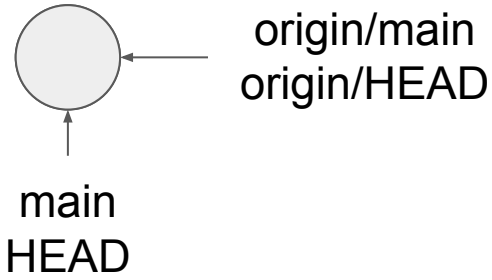
1) リモートリポジトリをGitHubで作成
2) リモートリポジトリをコピーして、ローカルリポジトリ作(clone)
3) ローカルリポジトリでブランチ(add-feature)を作成する
4) ファイルを編集して、作成したブランチにコミットする
5) ローカルリポジトリの内容をリモートリポジトリに反映する(push)
6) GitHubでブランチの内容を確認して良ければマージする
7) リモートリポジトリの内容をローカルリポジトリに反映する(pull)

プル（pull）とは

リモート
リポジトリ



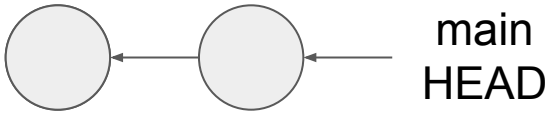
ローカル
リポジトリ



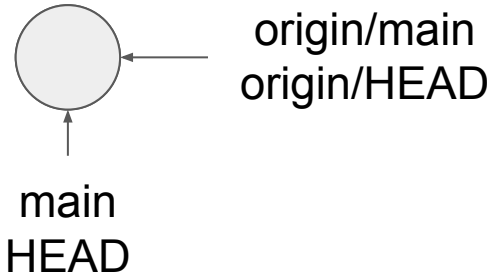
pull は fetch ＋ merge

プル（pull）とは

リモート
リポジトリ



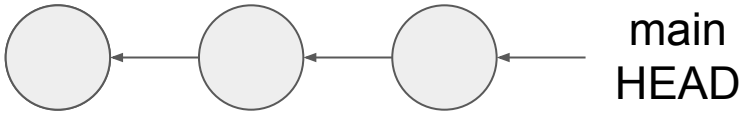
ローカル
リポジトリ



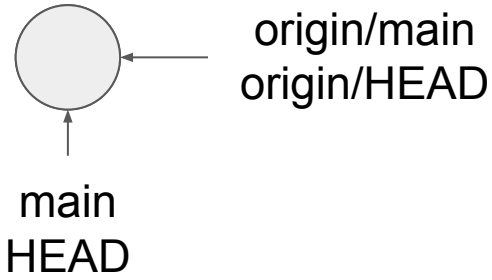
pull は fetch ＋ merge

プル（pull）とは

リモート
リポジトリ



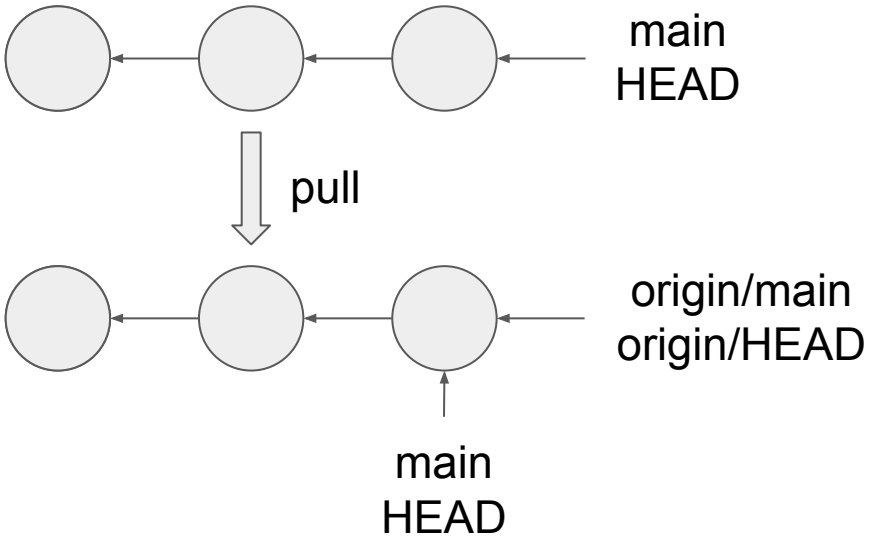
ローカル
リポジトリ



pull は fetch ＋ merge

プル（pull）とは

リモート
リポジトリ

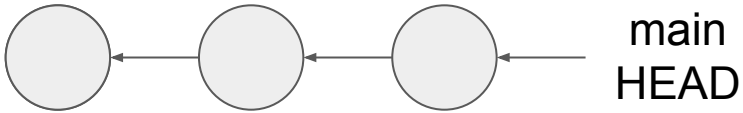


ローカル
リポジトリ

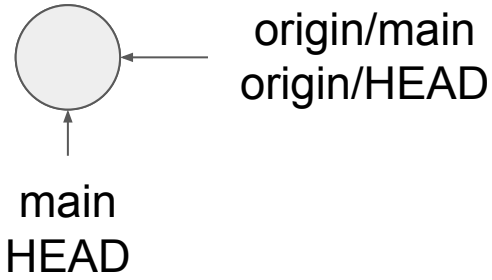
pull は fetch ＋ merge

プル（pull）とは

リモート
リポジトリ



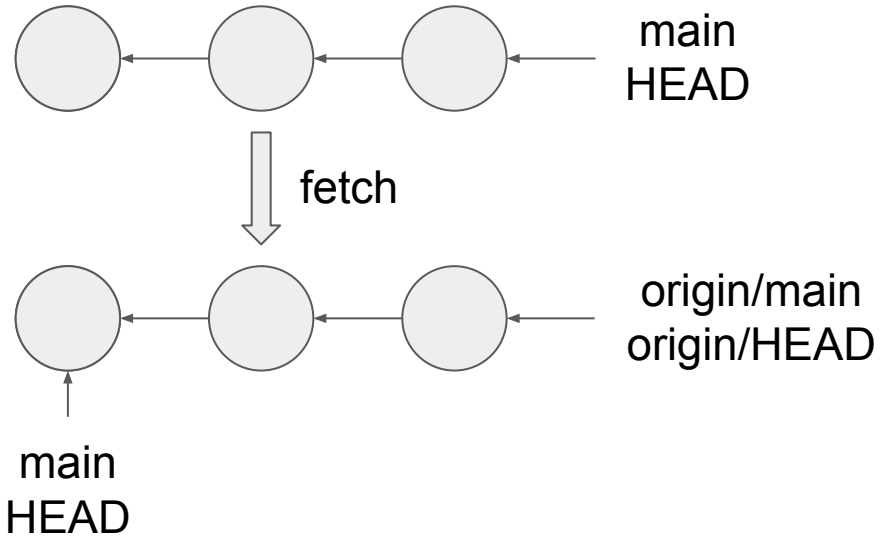
ローカル
リポジトリ



pull は fetch ＋ merge

プル（pull）とは

リモート
リポジトリ

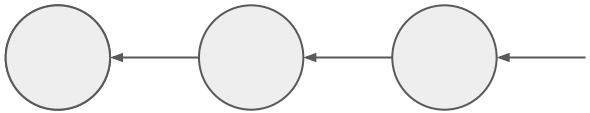


ローカル
リポジトリ

pull は fetch ＋ merge

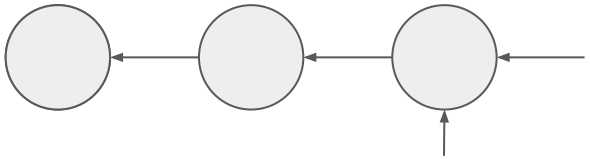
プル（pull）とは

リモート
リポジトリ



main
HEAD

ローカル
リポジトリ



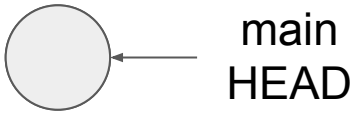
origin/main
origin/HEAD

main
HEAD

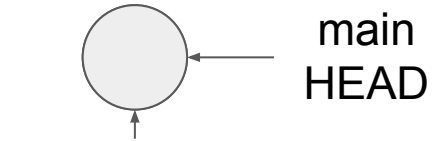
pull は fetch ＋ merge

早送りマージが出来ない場合

リモート
リポジトリ



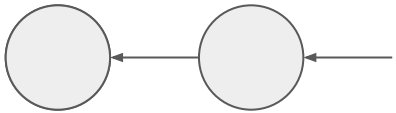
ローカル
リポジトリ



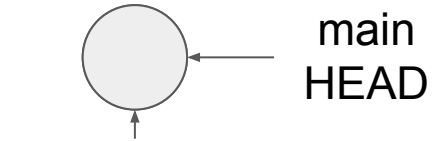
ローカルのHEADがあるブランチで
早送りマージができない場合は通常のマージになる

早送りマージが出来ない場合

リモート
リポジトリ



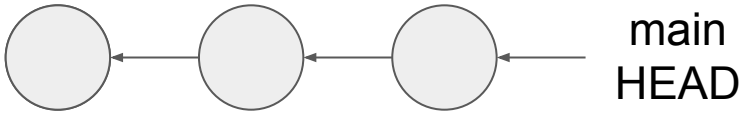
ローカル
リポジトリ



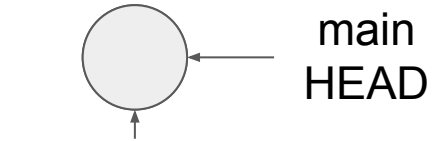
ローカルのHEADがあるブランチで
早送りマージができない場合は通常のマージになる

早送りマージが出来ない場合

リモート
リポジトリ



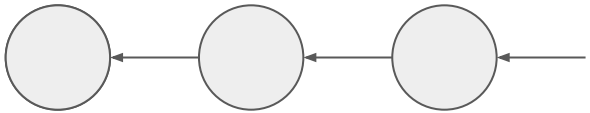
ローカル
リポジトリ



ローカルのHEADがあるブランチで
早送りマージができない場合は通常のマージになる

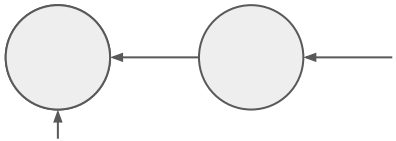
早送りマージが出来ない場合

リモート
リポジトリ



main
HEAD

ローカル
リポジトリ



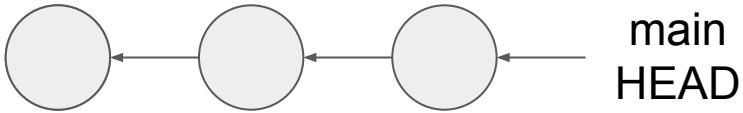
main
HEAD

origin/master
origin/HEAD

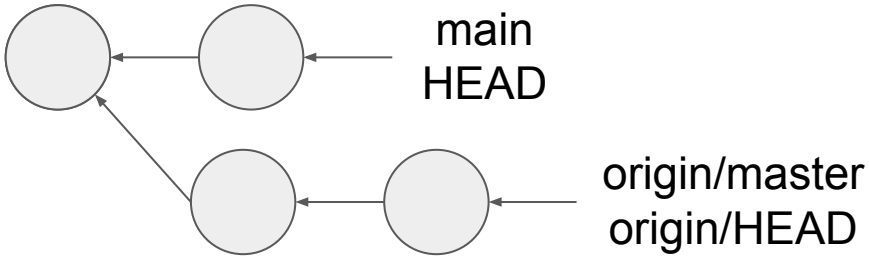
ローカルのHEADがあるブランチで
早送りマージができない場合は通常のマージになる

早送りマージが出来ない場合

リモート
リポジトリ



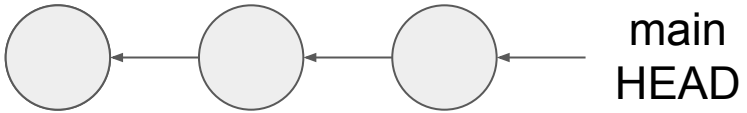
ローカル
リポジトリ



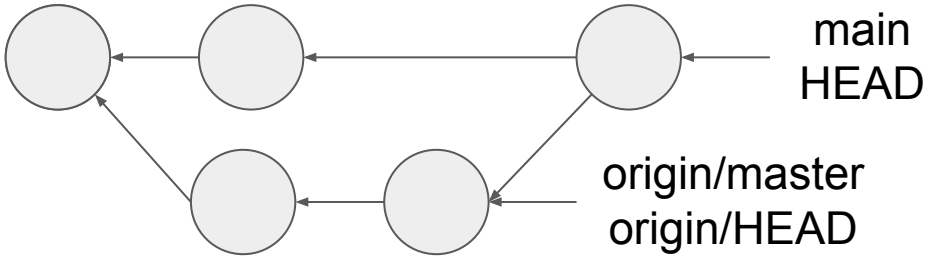
ローカルのHEADがあるブランチで
早送りマージができない場合は通常のマージになる

早送りマージが出来ない場合

リモート
リポジトリ



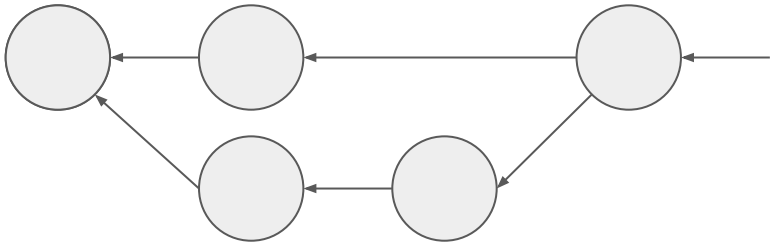
ローカル
リポジトリ



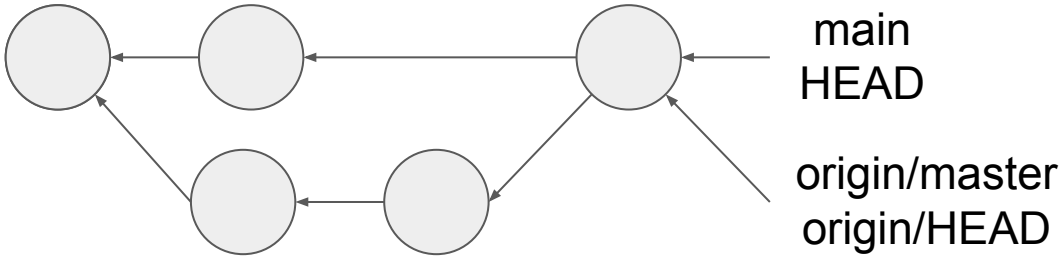
ローカルのHEADがあるブランチで
早送りマージができない場合は通常のマージになる

早送りマージが出来ない場合

リモート
リポジトリ



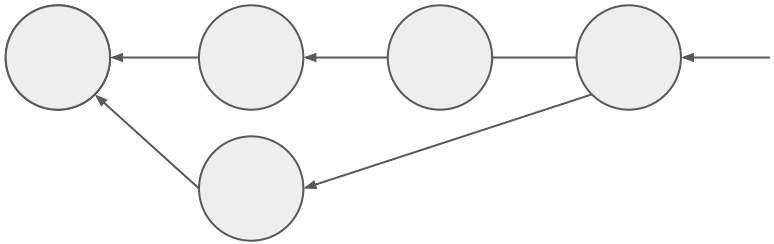
ローカル
リポジトリ



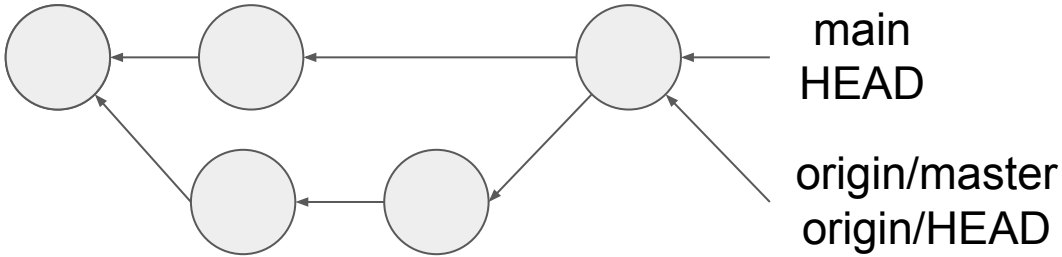
ローカルのHEADがあるブランチで
早送りマージができない場合は通常のマージになる

早送りマージが出来ない場合

リモート
リポジトリ



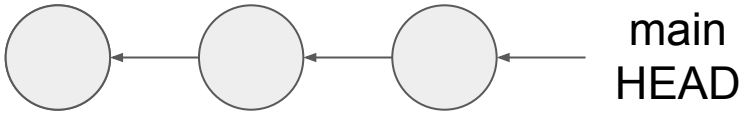
ローカル
リポジトリ



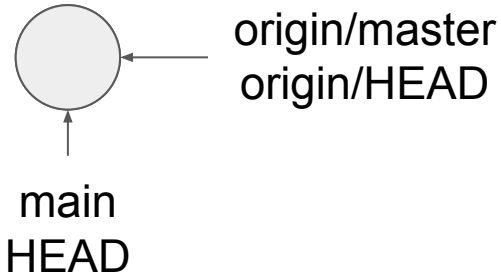
ローカルのHEADがあるブランチで
早送りマージができない場合は通常のマージになる

FETCH_HEADとは

リモート
リポジトリ



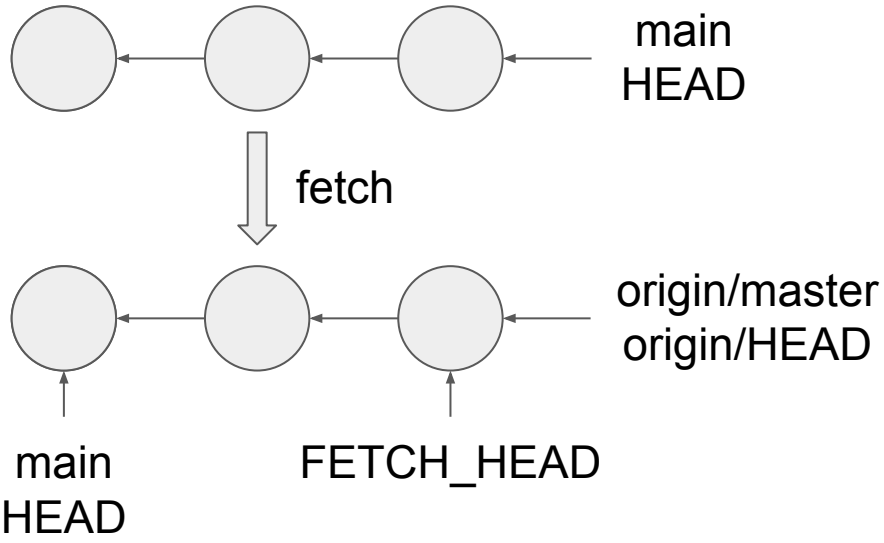
ローカル
リポジトリ



fetchしたブランチと同じコミットを指す
ローカルリポジトリにあるポインタ

FETCH_HEADとは

リモート
リポジトリ

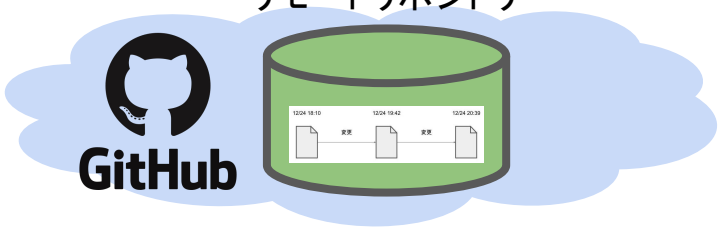


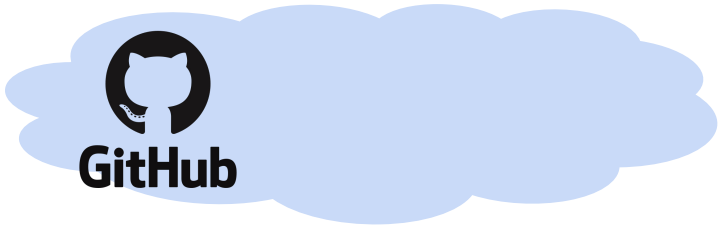
ローカル
リポジトリ

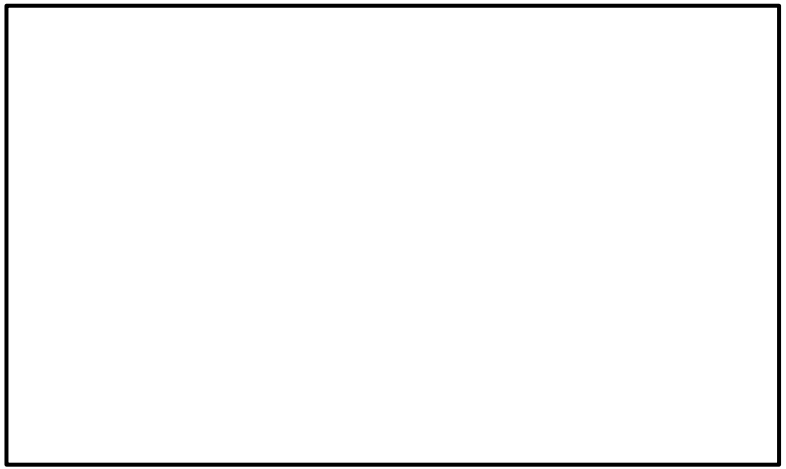
fetchしたブランチと同じコミットを指す
ローカルリポジトリにあるポインタ

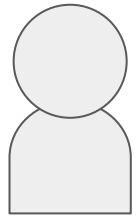
GitHubでのOSSの開発の流れ（暫定版）

OSSの
リモートリポジトリ

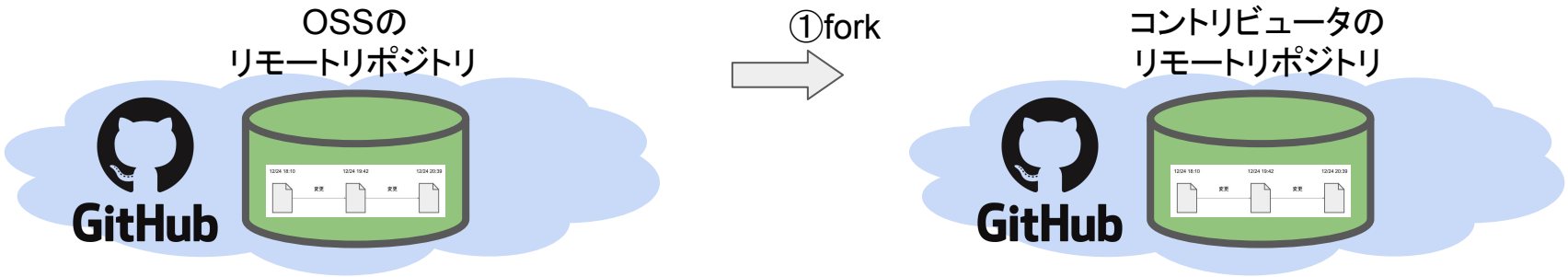




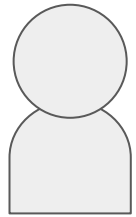




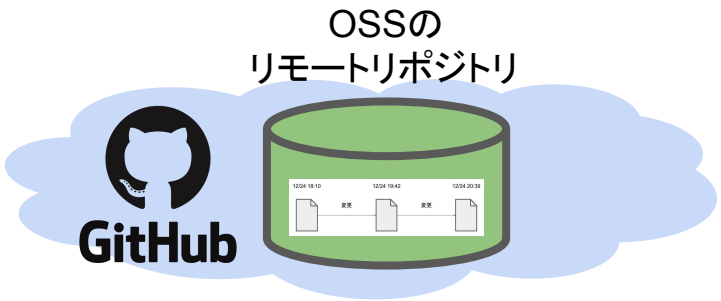
GitHubでのOSSの開発の流れ（暫定版）



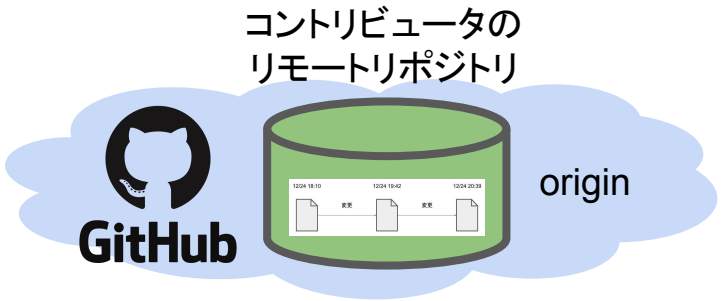
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする



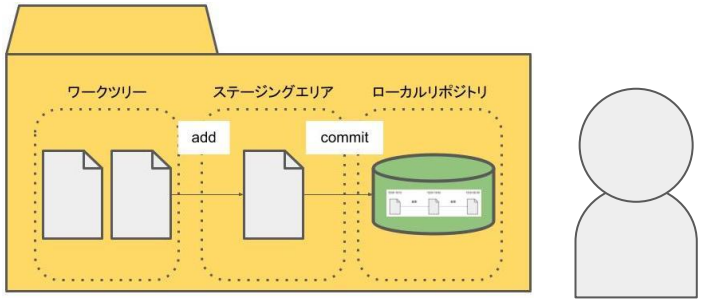
GitHubでのOSSの開発の流れ（暫定版）



①fork

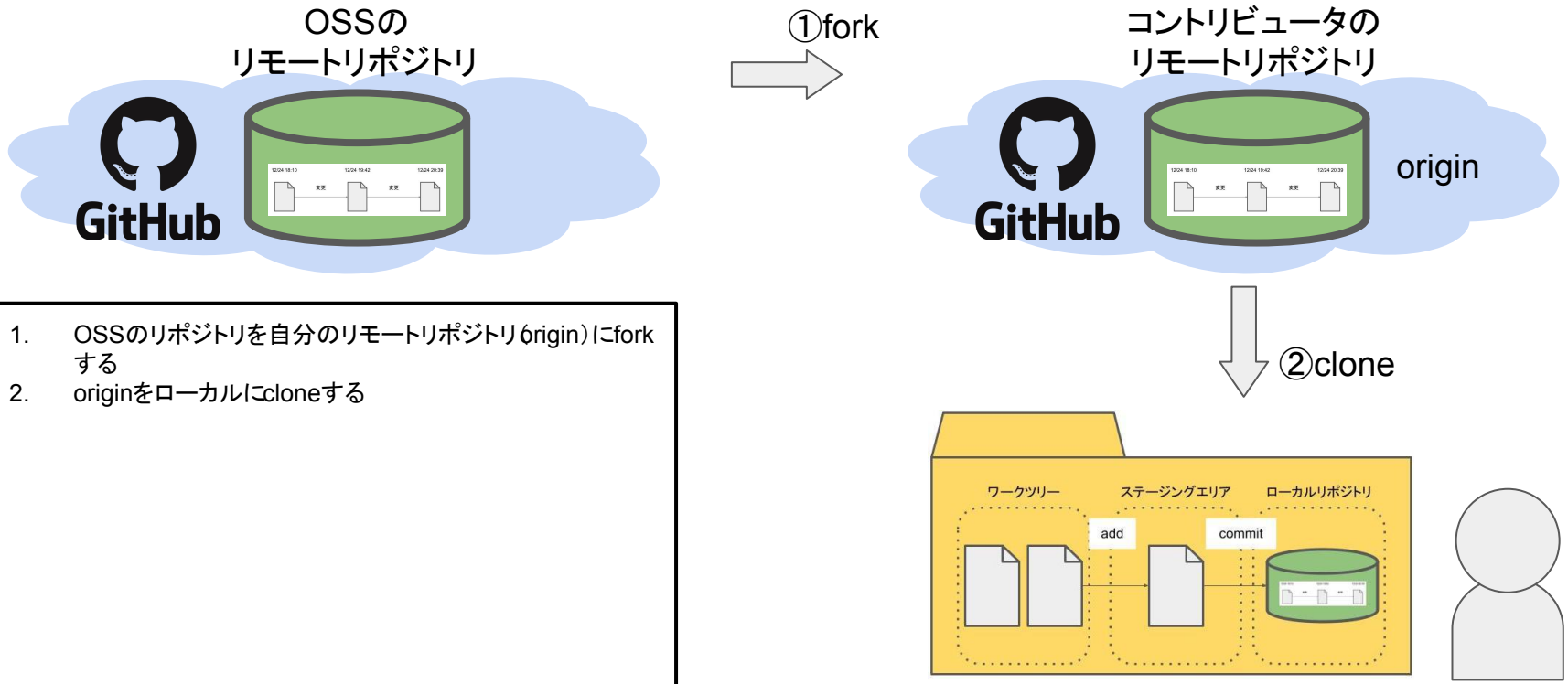


②clone



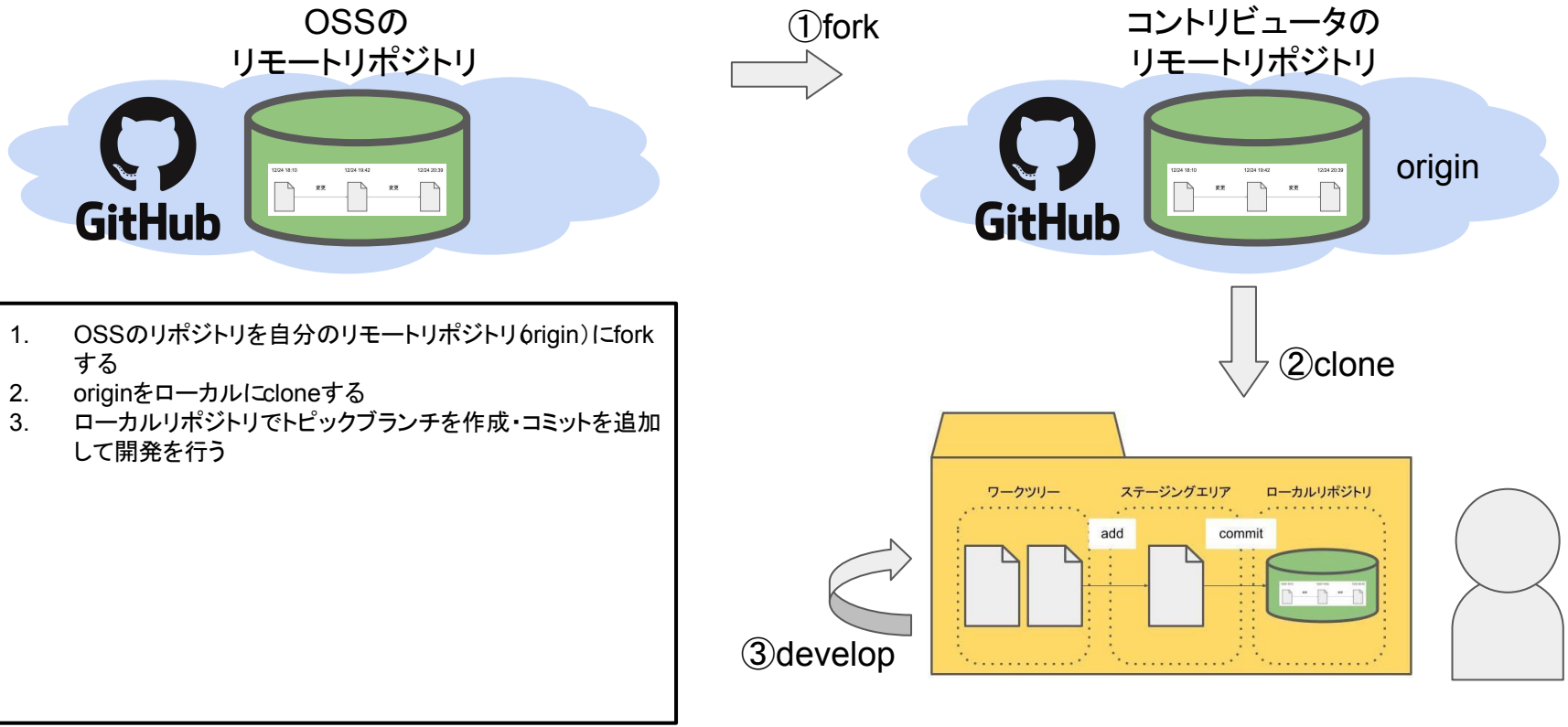
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする

GitHubでのOSSの開発の流れ（暫定版）



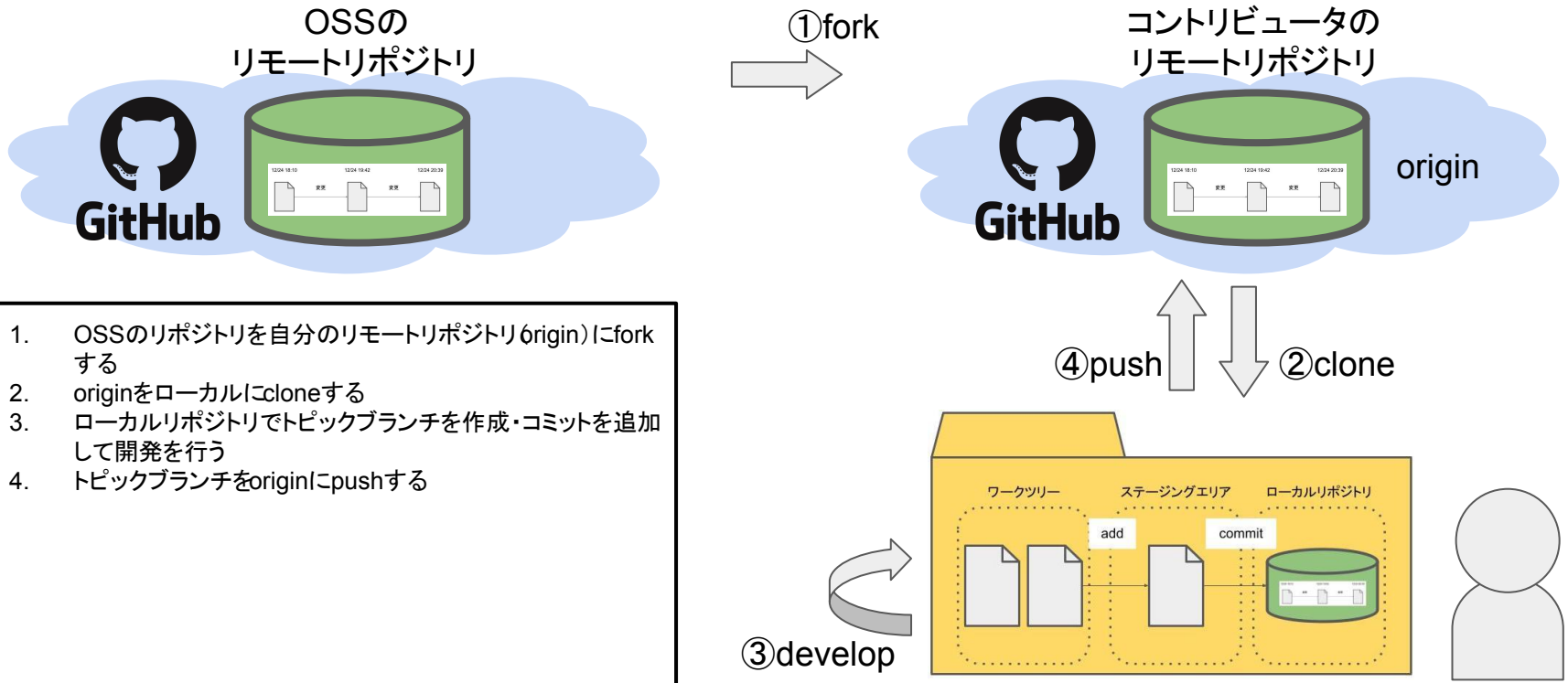
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする

GitHubでのOSSの開発の流れ（暫定版）



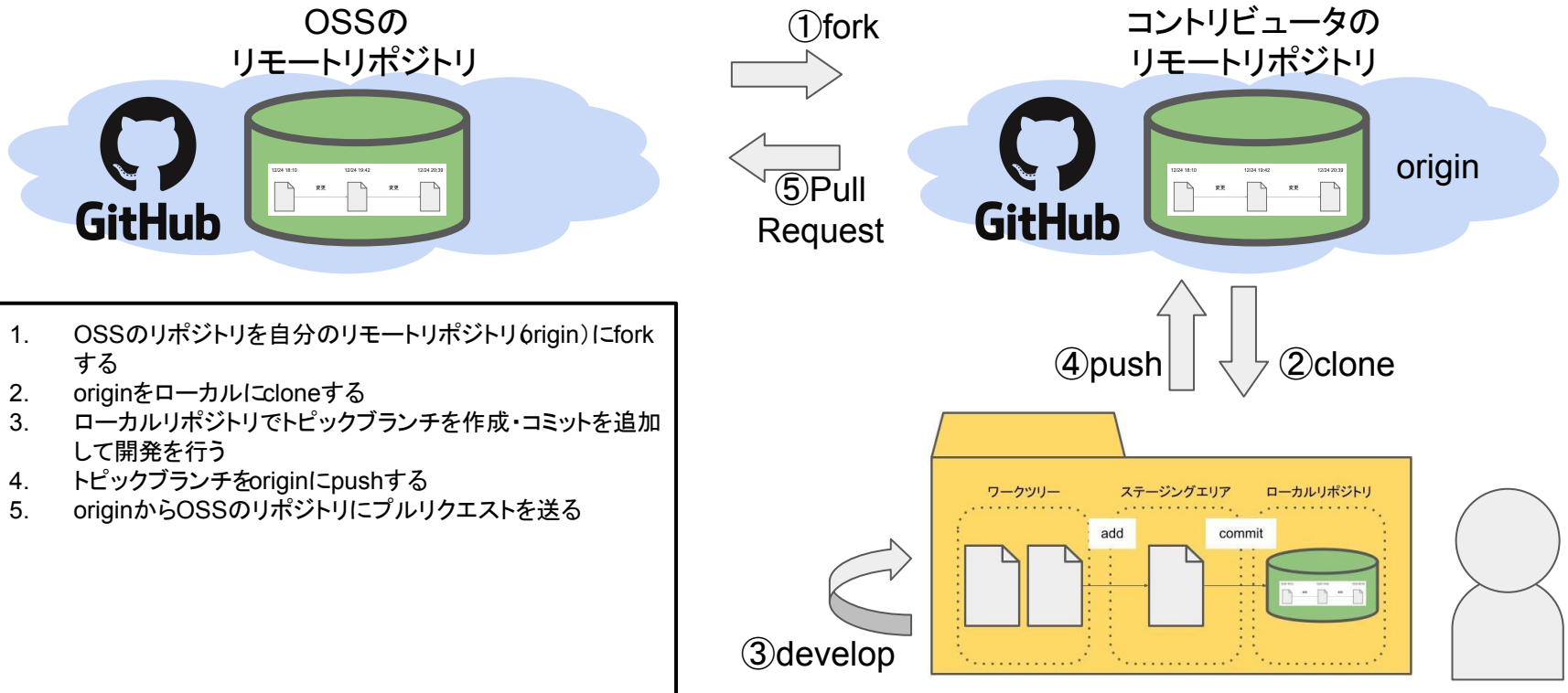
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする
3. ローカルリポジトリでトピックブランチを作成・コミットを追加して開発を行う

GitHubでのOSSの開発の流れ（暫定版）



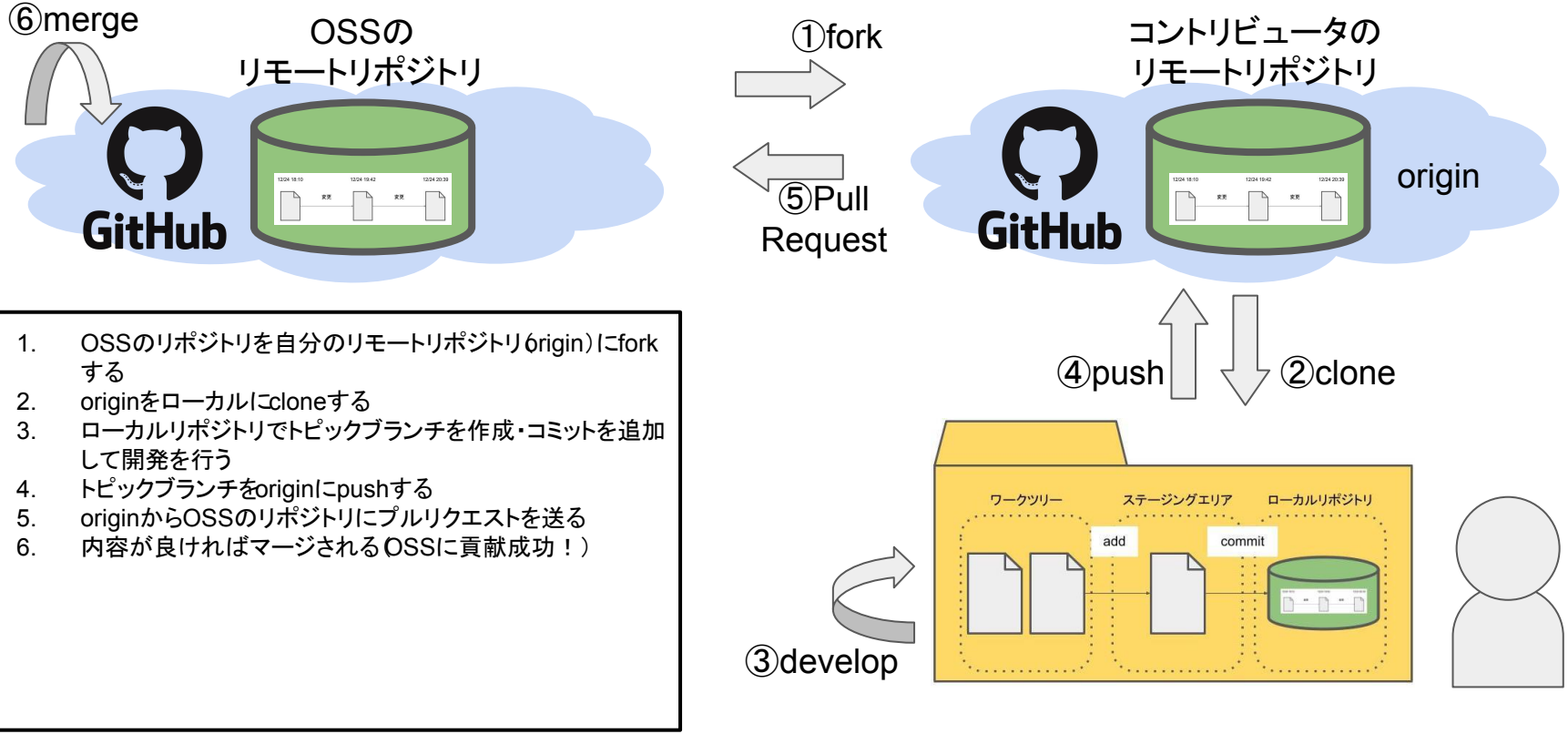
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする
3. ローカルリポジトリでトピックブランチを作成・コミットを追加して開発を行う
4. トピックブランチをoriginにpushする

GitHubでのOSSの開発の流れ（暫定版）



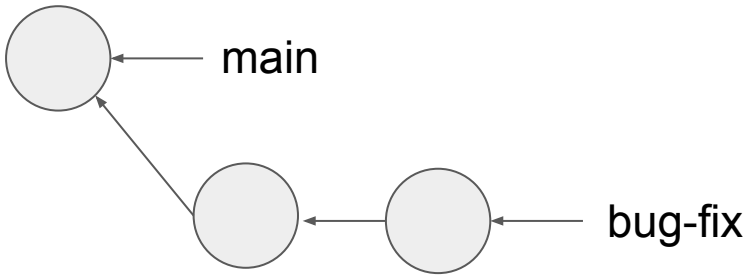
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする
3. ローカルリポジトリでトピックブランチを作成・コミットを追加して開発を行う
4. トピックブランチをoriginにpushする
5. originからOSSのリポジトリにプルリクエストを送る

GitHubでのOSSの開発の流れ（暫定版）



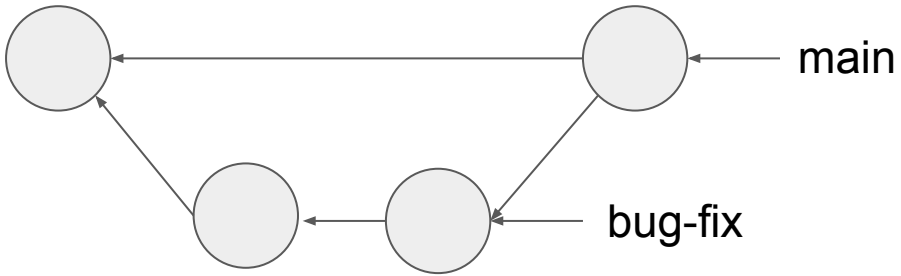
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする
3. ローカルリポジトリでトピックブランチを作成・コミットを追加して開発を行う
4. トピックブランチをoriginにpushする
5. originからOSSのリポジトリにプルリクエストを送る
6. 内容が良ければマージされる（OSSに貢献成功！）

OSSリポジトリのメインブランチが伸びた場合の問題点



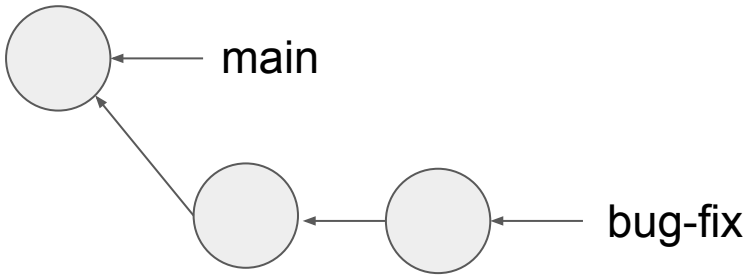
メインブランチが伸びていないなら
Pull Requestをマージする際にコンフリクトは発生しない

OSSリポジトリのメインブランチが伸びた場合の問題点



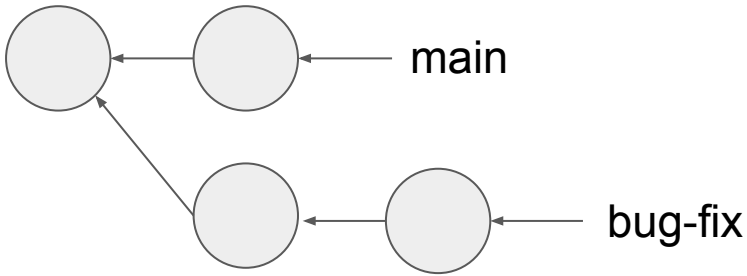
メインブランチが伸びていないなら
Pull Requestをマージする際にコンフリクトは発生しない

OSSリポジトリのメインブランチが伸びた場合の問題点



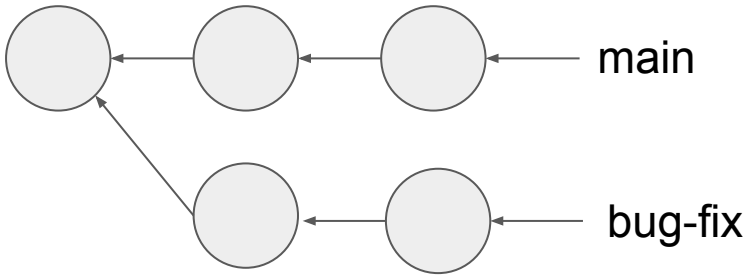
メインブランチが伸びていると、Pull Requestをマージする際に
コンフリクトが発生してしまう可能性がある

OSSリポジトリのメインブランチが伸びた場合の問題点



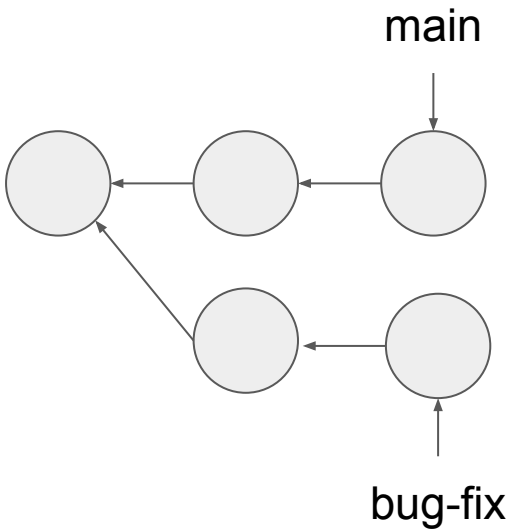
メインブランチが伸びていると、Pull Requestをマージする際に
コンフリクトが発生してしまう可能性がある

OSSリポジトリのメインブランチが伸びた場合の問題点



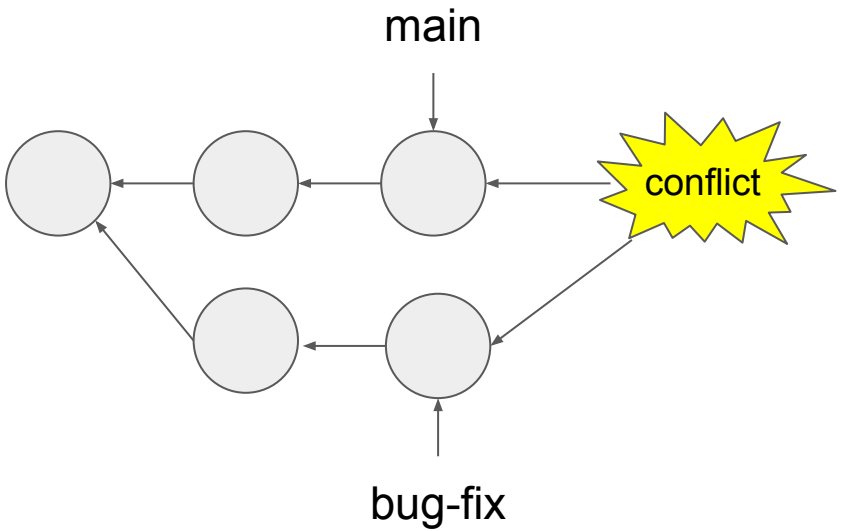
メインブランチが伸びていると、Pull Requestをマージする際に
コンフリクトが発生してしまう可能性がある

OSSリポジトリのメインブランチが伸びた場合の問題点



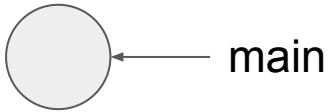
メインブランチが伸びていると、Pull Requestをマージする際に
コンフリクトが発生してしまう可能性がある

OSSリポジトリのメインブランチが伸びた場合の問題点



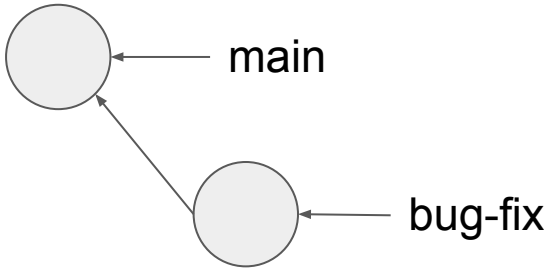
メインブランチが伸びていると、Pull Requestをマージする際に
コンフリクトが発生してしまう可能性がある

OSSリポジトリのメインブランチが伸びた場合の問題点



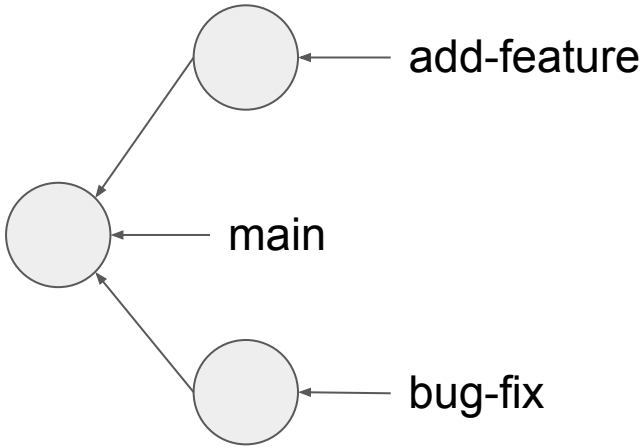
マージが入り乱れてコミット履歴がわかりづらくなる

OSSリポジトリのメインブランチが伸びた場合の問題点



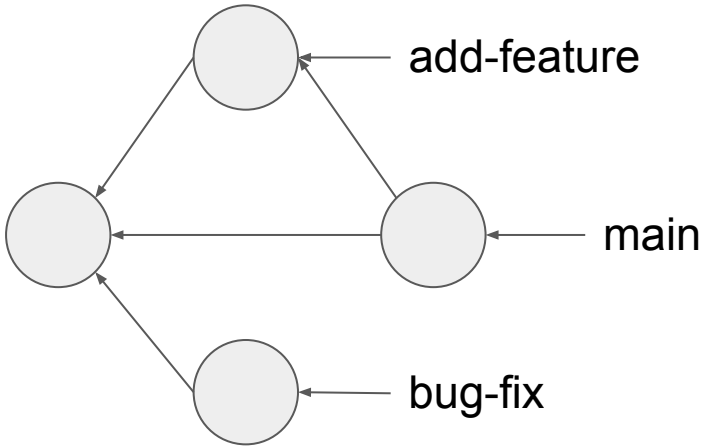
マージが入り乱れてコミット履歴がわかりづらくなる

OSSリポジトリのメインブランチが伸びた場合の問題点



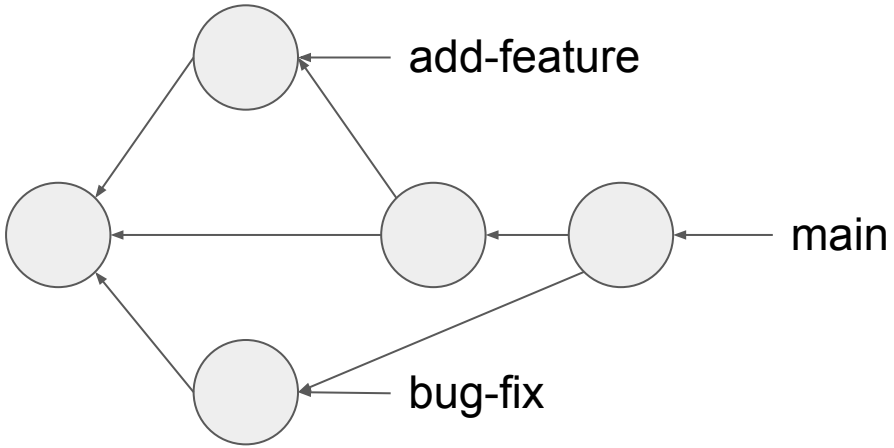
マージが入り乱れてコミット履歴がわかりづらくなる

OSSリポジトリのメインブランチが伸びた場合の問題点



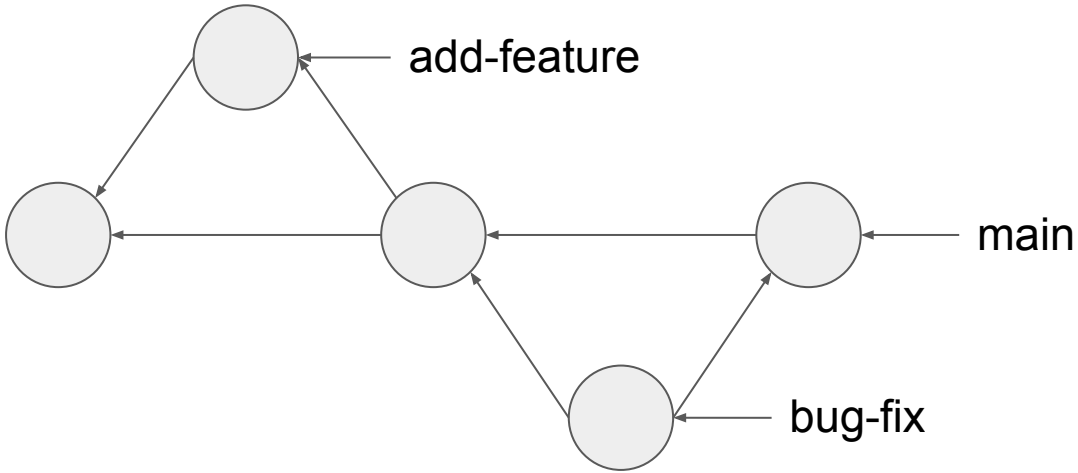
マージが入り乱れてコミット履歴がわかりづらくなる

OSSリポジトリのメインブランチが伸びた場合の問題点



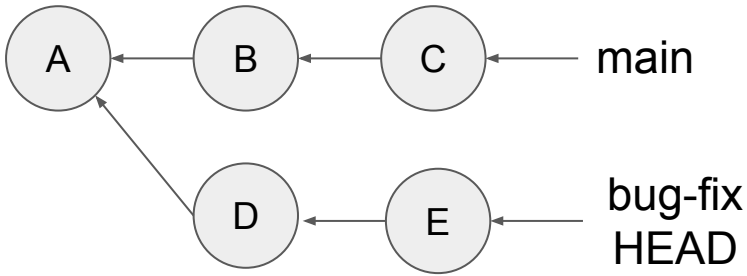
マージが入り乱れてコミット履歴がわかりづらくなる

OSSリポジトリのメインブランチが伸びた場合の問題点



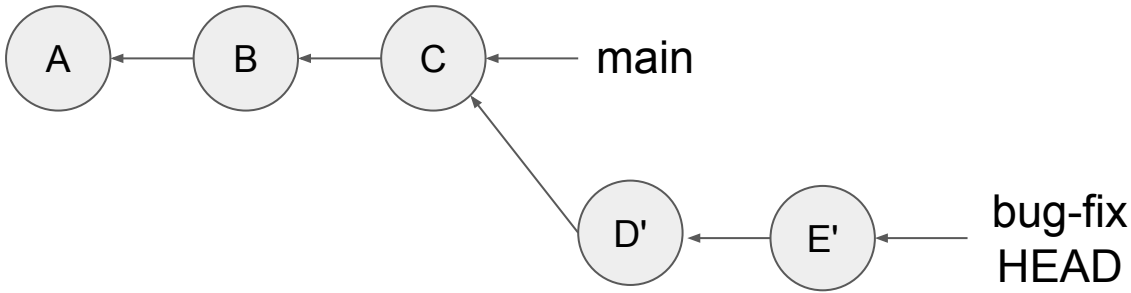
bug-fixブランチの根元を移動したほうが
コミット履歴としてかなりわかりやすい

rebaseとは



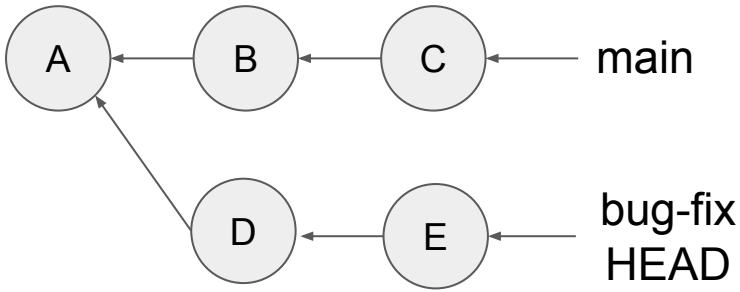
ブランチの根元を移動するコマンド

rebaseとは



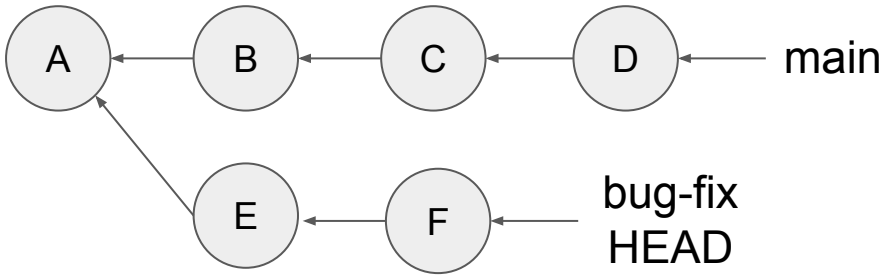
ブランチの根元を移動するコマンド

rebaseの仕組み



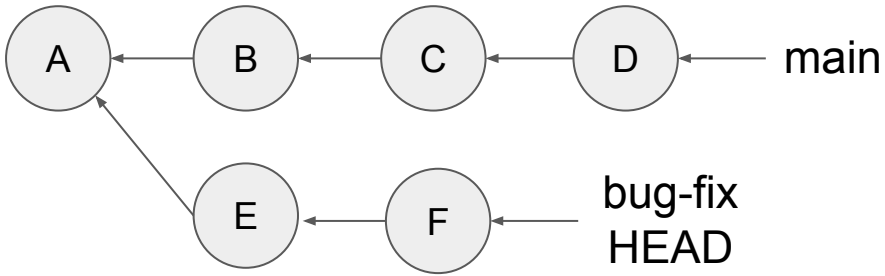
rebaseはcherry-pickを繰り返すことで行われる

cherry-pickとは



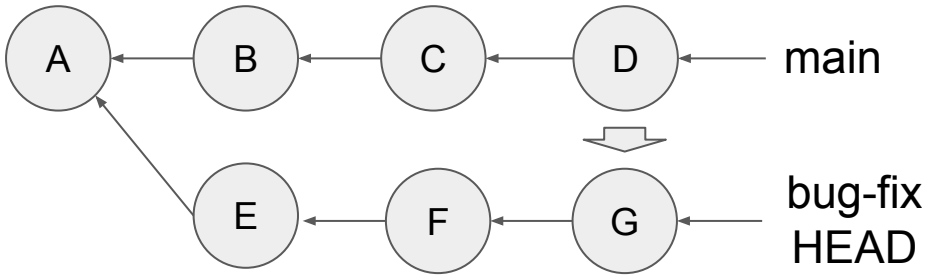
指定したコミットの内容を
取り入れることができる機能

cherry-pickとは



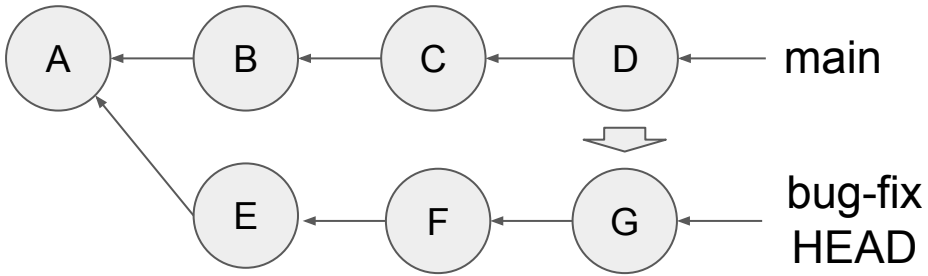
「git cherry-pick main」と実行すると
コミットFとコミットDの内容を
組み合わせたコミットGが追加される

cherry-pickとは



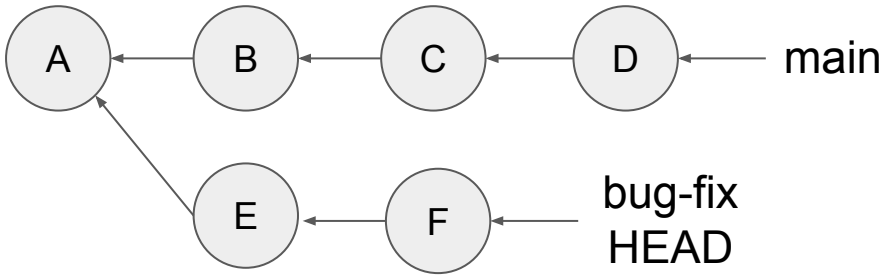
「git cherry-pick main」と実行すると
コミットFとコミットDの内容を
組み合わせたコミットGが追加される

cherry-pickとは



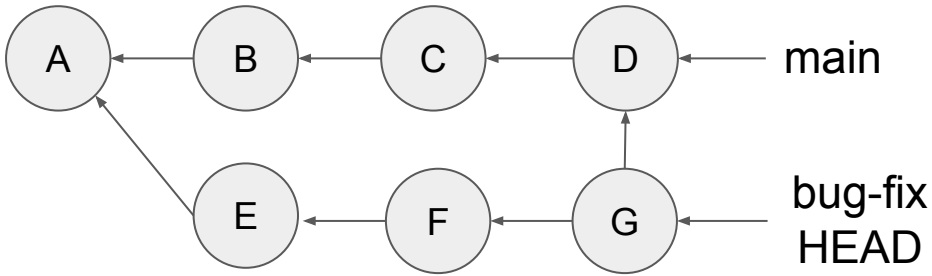
コミットGはmainブランチとbug-fixブランチの
マージコミットと同じ内容になる

cherry-pickとは



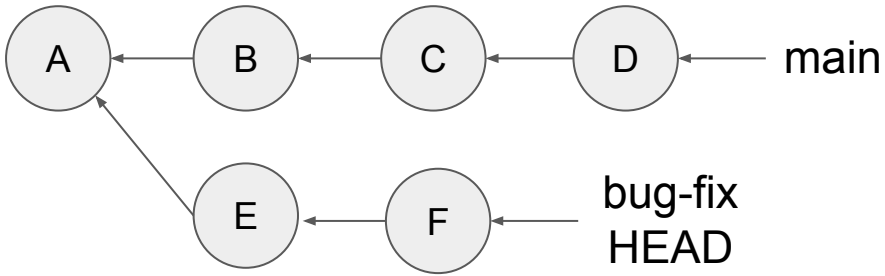
コミットGはmainブランチとbug-fixブランチの
マージコミットと同じ内容になる

cherry-pickとは



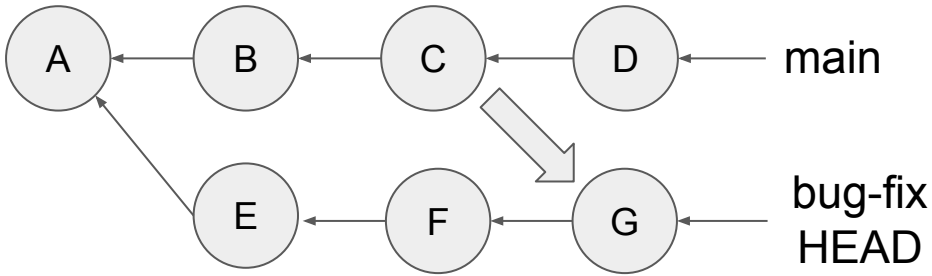
コミットGはmainブランチとbug-fixブランチの
マージコミットと同じ内容になる

cherry-pickとは



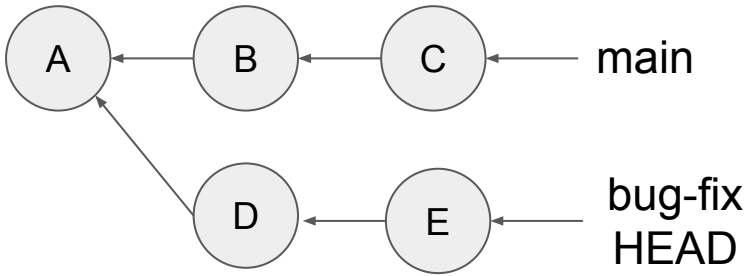
「git cherry-pick コミット」と実行すれば
ブランチが指す以外のコミットの内容を取り入れることもできる

cherry-pickとは



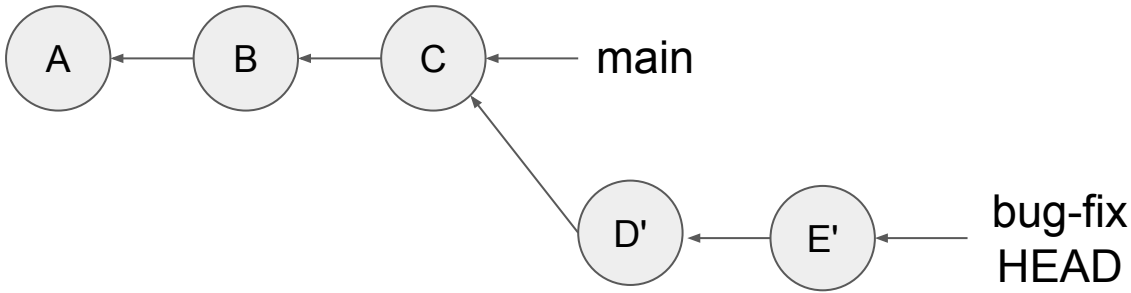
「git cherry-pick コミット」と実行すれば
ブランチが指す以外のコミットの内容を取り入れることもできる
G = F + C

rebaseとは



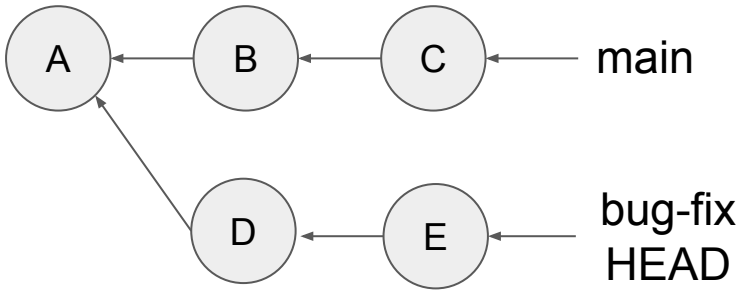
ブランチの根元を移動するコマンド

rebaseとは



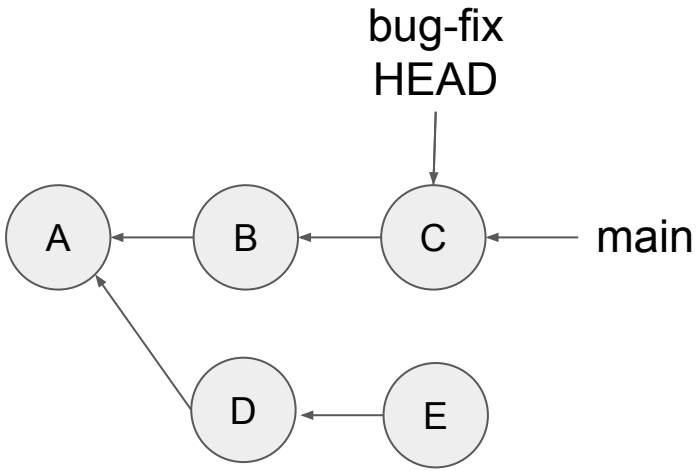
ブランチの根元を移動するコマンド

rebaseの仕組み



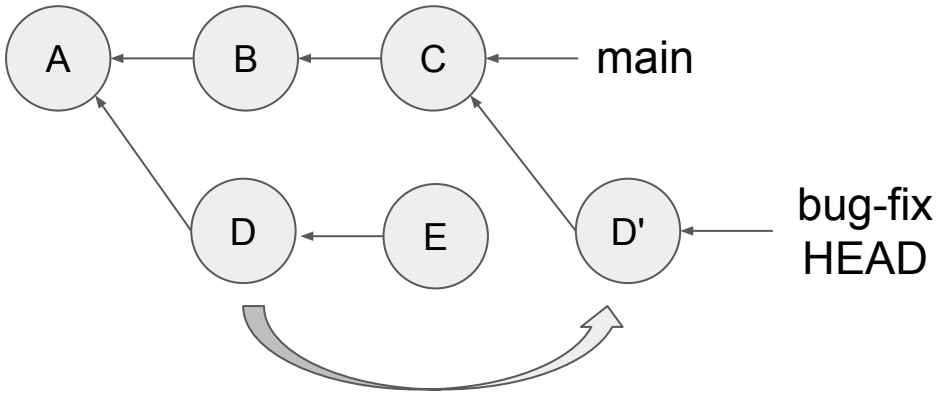
rebaseはcherry-pickを繰り返すことで行われる

rebaseの仕組み



rebaseはcherry-pickを繰り返すことで行われる

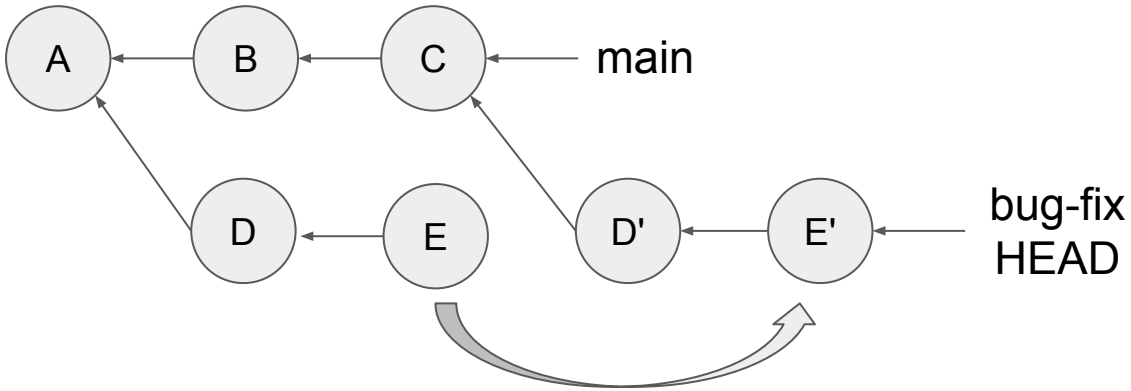
rebaseの仕組み



rebaseはcherry-pickを繰り返すことで行われる

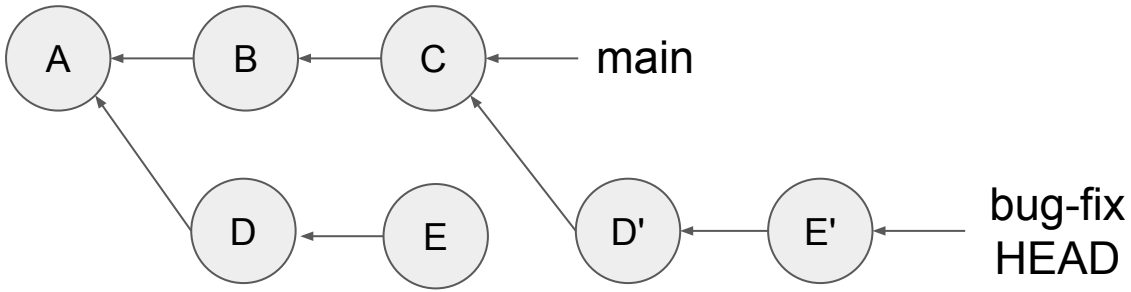
D’ = C + D

rebaseの仕組み



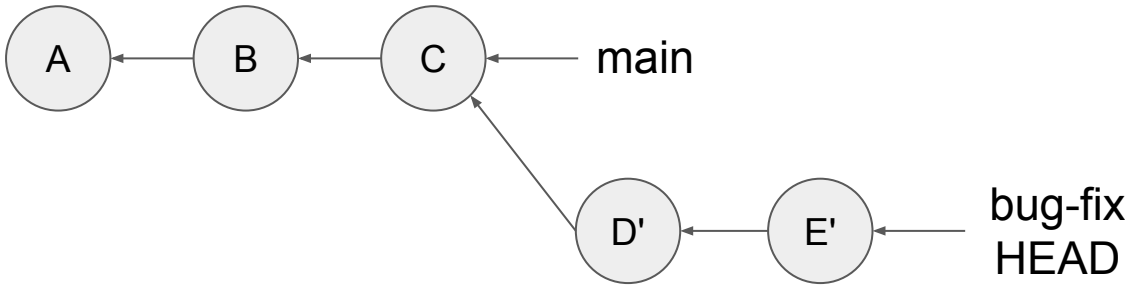
rebaseはcherry-pickを繰り返すことで行われる
E’ = D’ + E

rebaseの仕組み



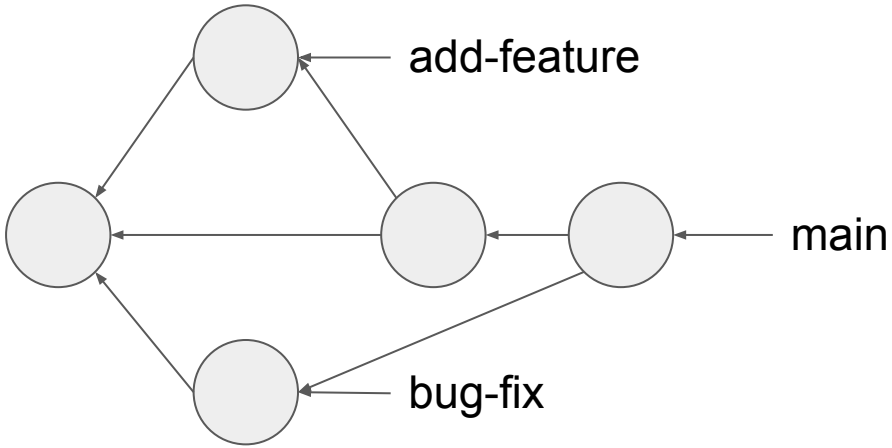
元のコミットとは別のコミットが作成されていることに注目
元のコミットはブランチからたどれなくなるので削除される

rebaseの仕組み



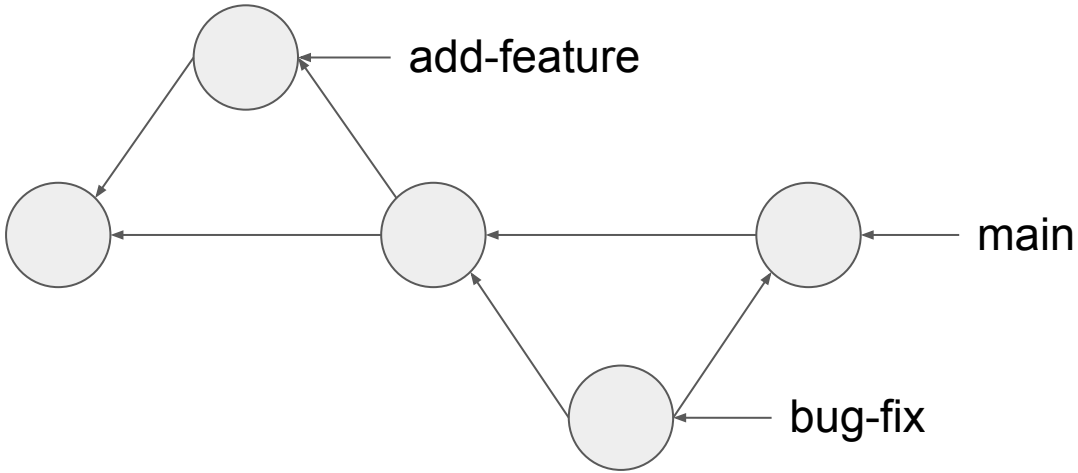
元のコミットとは別のコミットが作成されていることに注目
元のコミットはブランチからたどれなくなるので削除される

rebaseの用途①：コミット履歴をきれいにしたい



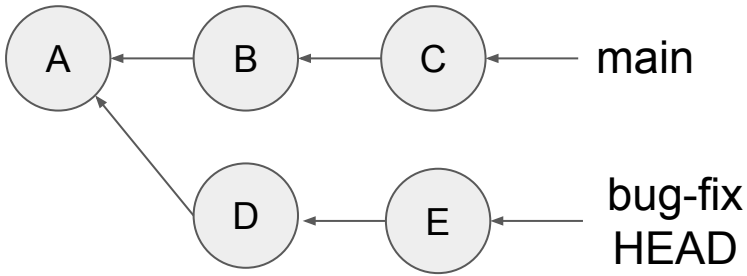
マージが入り乱れてコミット履歴がわかりづらくなる

rebaseの用途①：コミット履歴をきれいにしたい



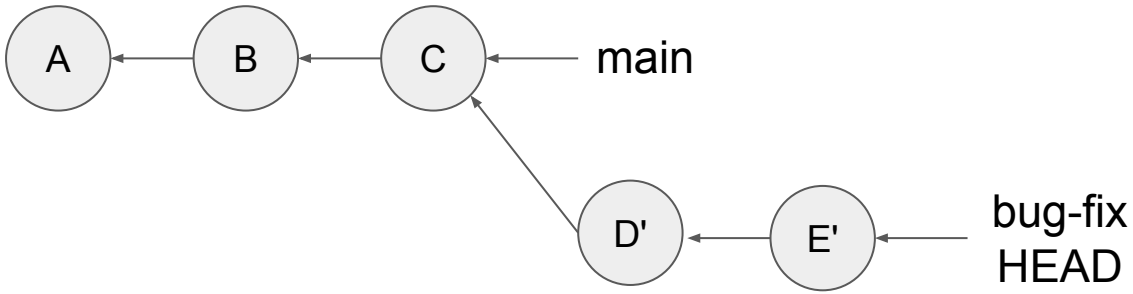
bug-fixブランチの根元を移動したほうが
コミット履歴としてかなりわかりやすい

rebaseの用途②：別のブランチの内容を取り込みたい



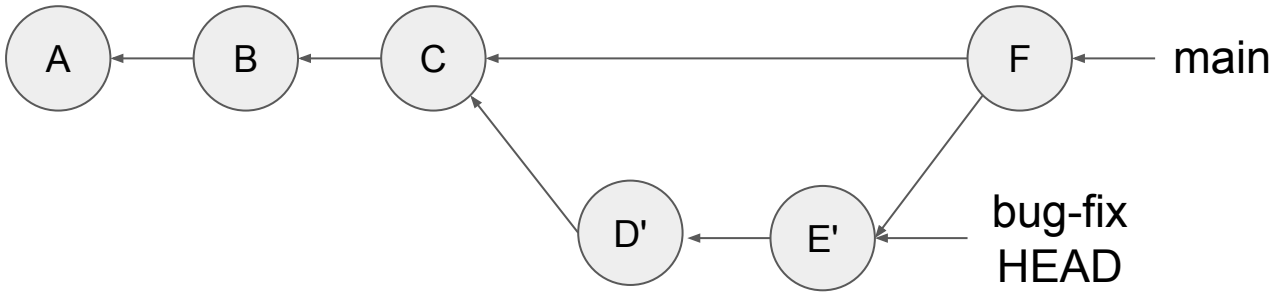
トピックブランチを開発している間にメインブランチが伸びたので
トピックブランチにメインブランチの内容を取り込みたい

rebaseの用途②：別のブランチの内容を取り込みたい



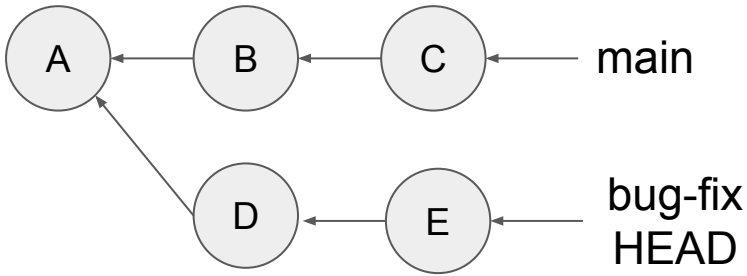
rebaseすればメインブランチのコミットを
トピックブランチに取り込むことができる

rebaseの用途②：別のブランチの内容を取り込みたい



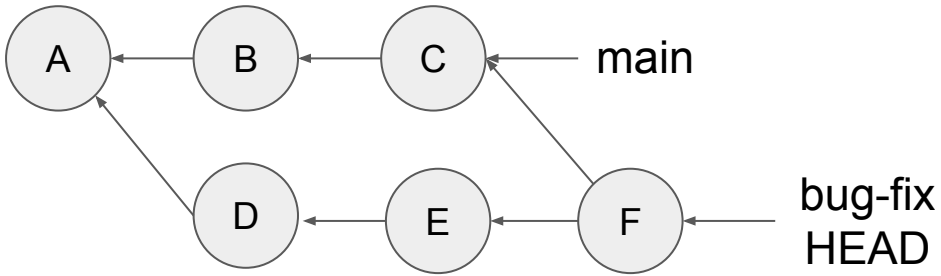
rebaseすればmainブランチにマージする際に
コンフリクトが発生しなくなる

rebaseを使わない方法



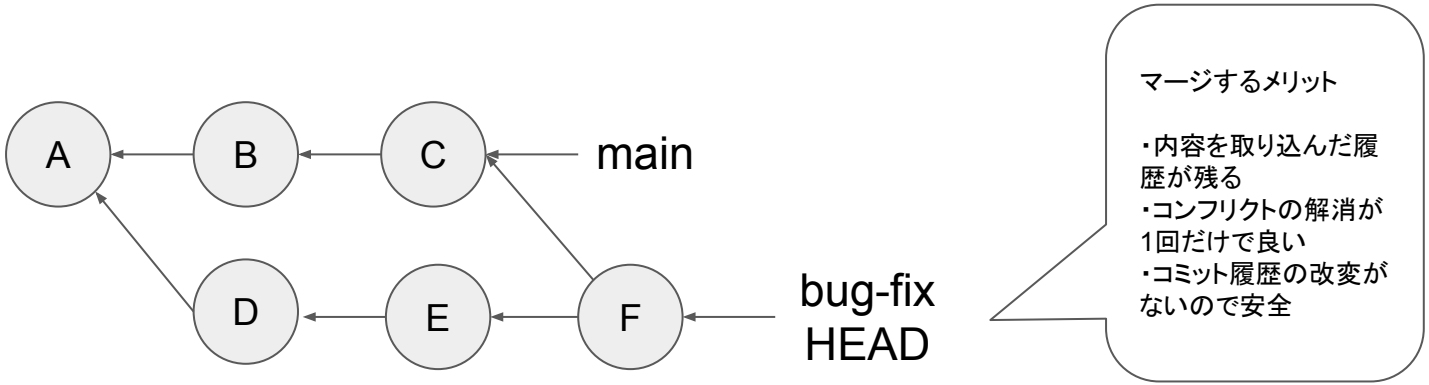
HEADがbug-fixを指す状態で「git merge main」として
トピックブランチにメインブランチを取り込む形でマージすれば、
rebaseを使わずともメインブランチの内容を取り込める

rebaseを使わない方法



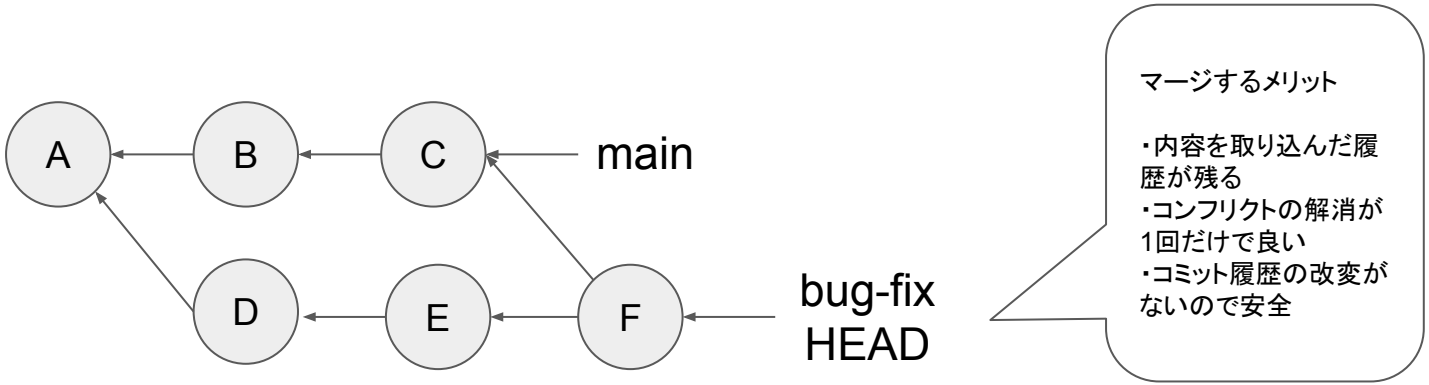
HEADがbug-fixを指す状態で「git merge main」として
トピックブランチにメインブランチを取り込む形でマージすれば、
rebaseを使わずともメインブランチの内容を取り込める

rebaseを使わない方法



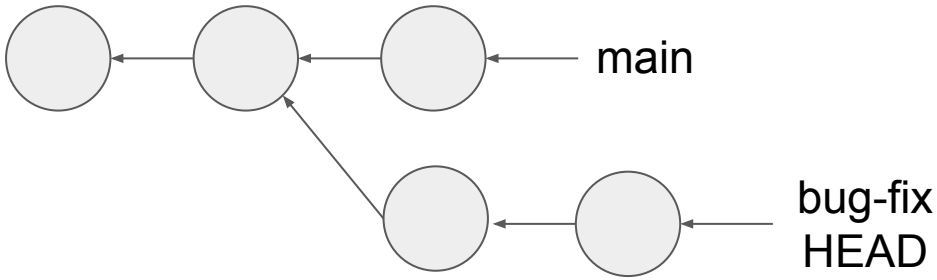
HEADがbug-fixを指す状態で「git merge main」として
トピックブランチにメインブランチを取り込む形でマージすれば、
rebaseを使わずともメインブランチの内容を取り込める

rebaseの使いどころ



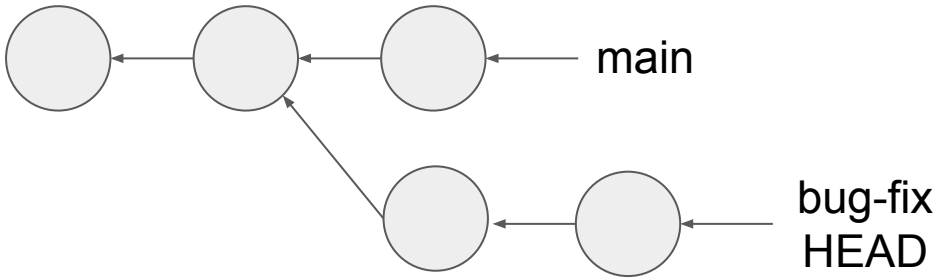
・マージの手間よりもコミット履歴のきれいさを優先したい
・まだrebaseするブランチをリモートにpushしていない
・リモートリポジトリを他の開発者と共有していない

rebaseは内容を取り込んだことが履歴として残らない



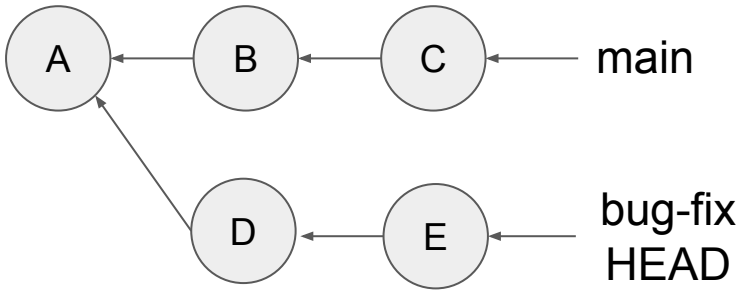
bug-fixブランチは、rebaseされてる？されていない？

rebaseは内容を取り込んだことが履歴として残らない



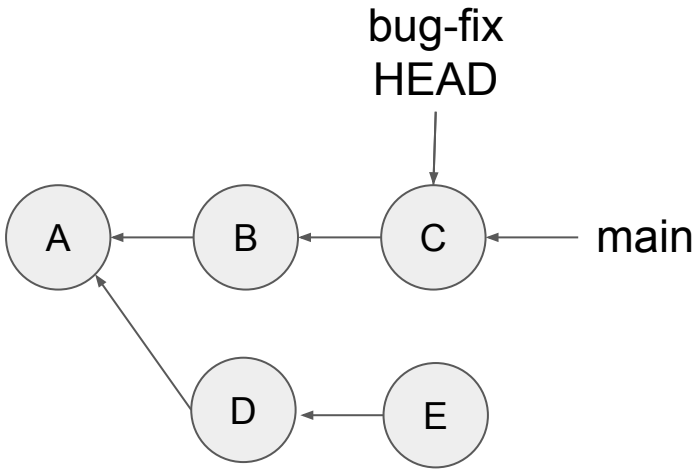
コミット履歴だけを見たときに、
rebaseが行なわれたかどうか区別が出来ない

rebaseのコンフリクトの解消は大変



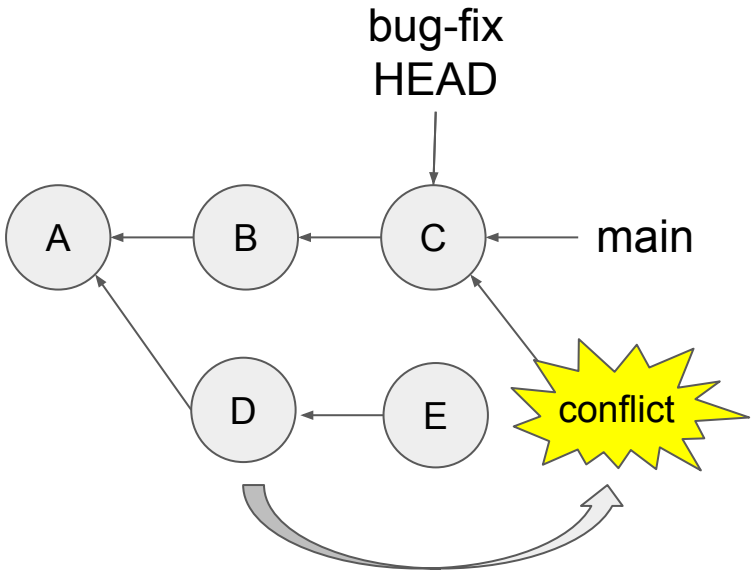
rebaseはcherry-pickの数だけコンフリクトが発生する
可能性があるので、マージよりも手間がかかる

rebaseのコンフリクトの解消は大変



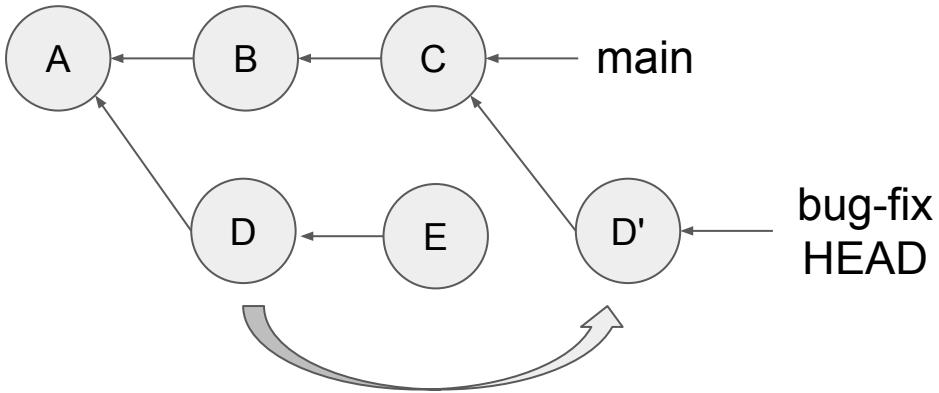
rebaseはcherry-pickの数だけコンフリクトが発生する
可能性があるので、マージよりも手間がかかる

rebaseのコンフリクトの解消は大変



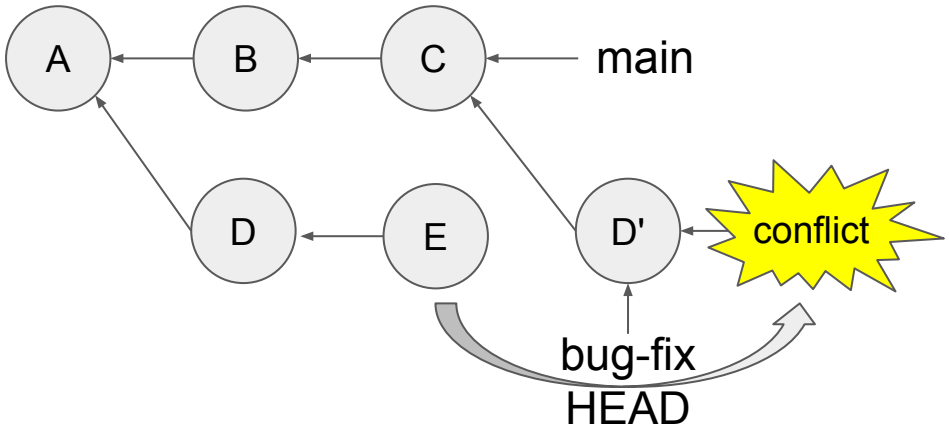
rebaseはcherry-pickの数だけコンフリクトが発生する
可能性があるので、マージよりも手間がかかる

rebaseのコンフリクトの解消は大変



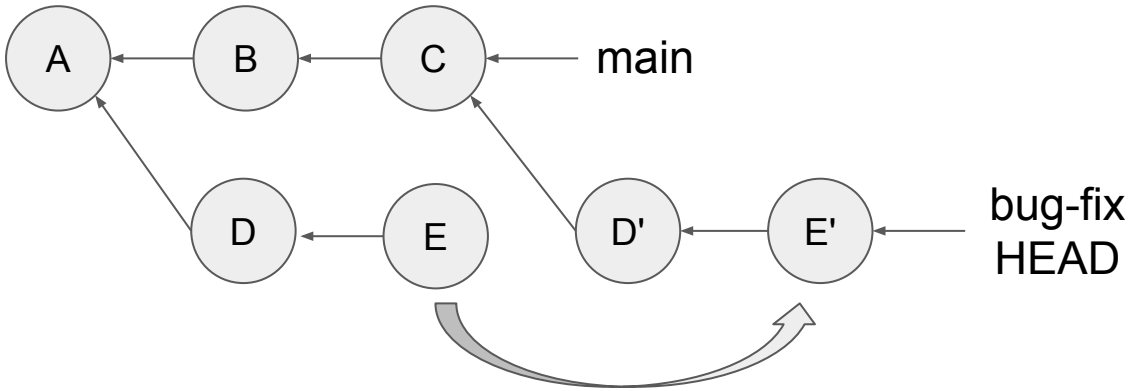
rebaseはcherry-pickの数だけコンフリクトが発生する
可能性があるので、マージよりも手間がかかる

rebaseのコンフリクトの解消は大変



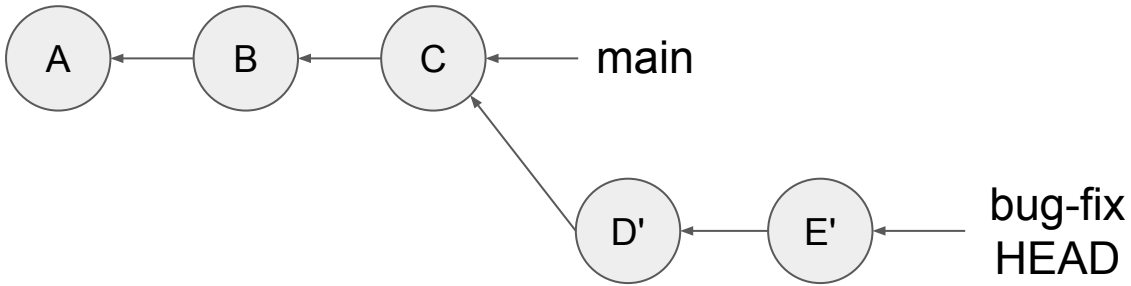
rebaseはcherry-pickの数だけコンフリクトが発生する
可能性があるので、マージよりも手間がかかる

rebaseのコンフリクトの解消は大変



rebaseはcherry-pickの数だけコンフリクトが発生する
可能性があるので、マージよりも手間がかかる

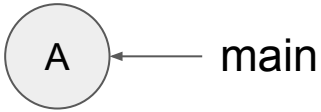
rebaseのコンフリクトの解消は大変



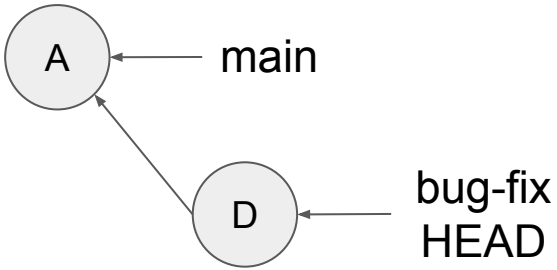
rebaseはcherry-pickの数だけコンフリクトが発生する
可能性があるので、マージよりも手間がかかる

pushやpullの仕組み

リモート

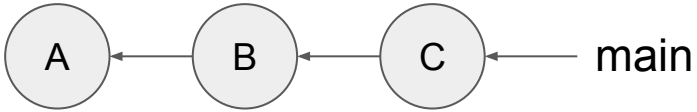


ローカル

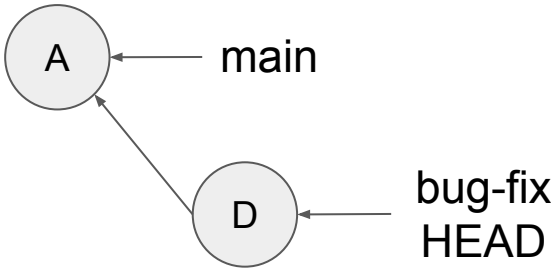


pushやpullの仕組み

リモート

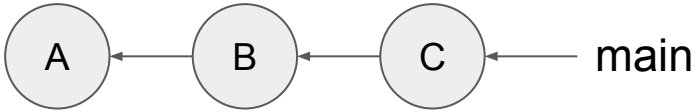


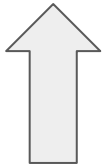
ローカル



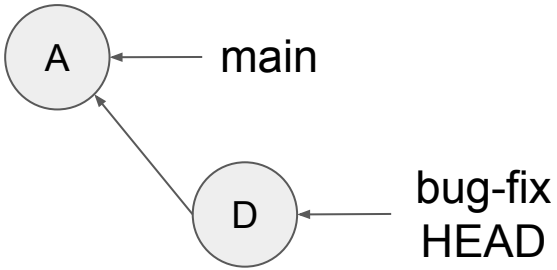
pushやpullの仕組み

リモート





ローカル

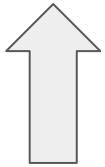


リモート:

ローカル:bug-fix→D→A

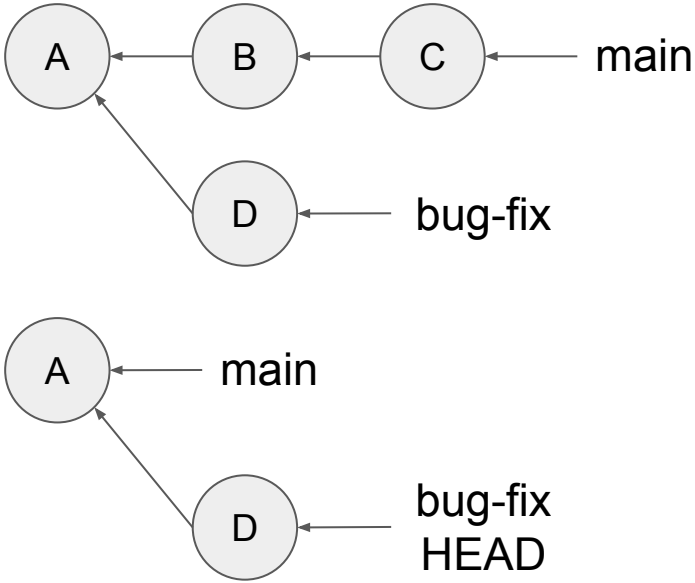
pushやpullの仕組み

リモート



push

ローカル

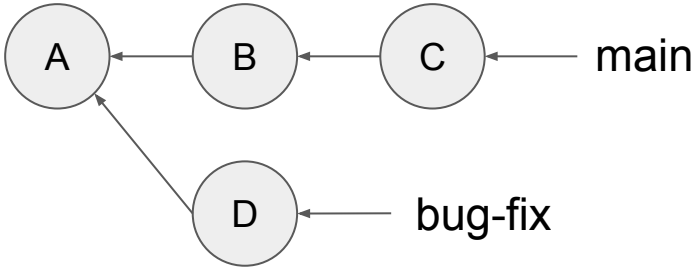


リモート：**A**

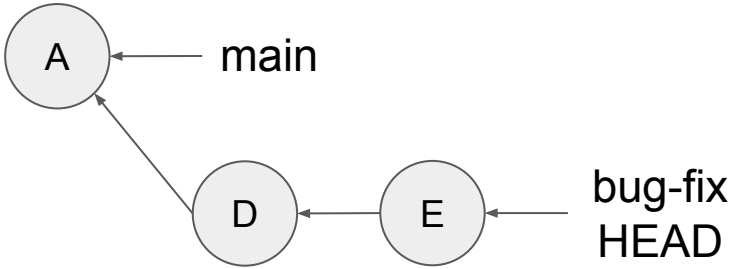
ローカル：bug-fix→D→**A**

pushやpullの仕組み

リモート

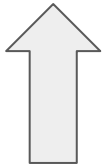


ローカル



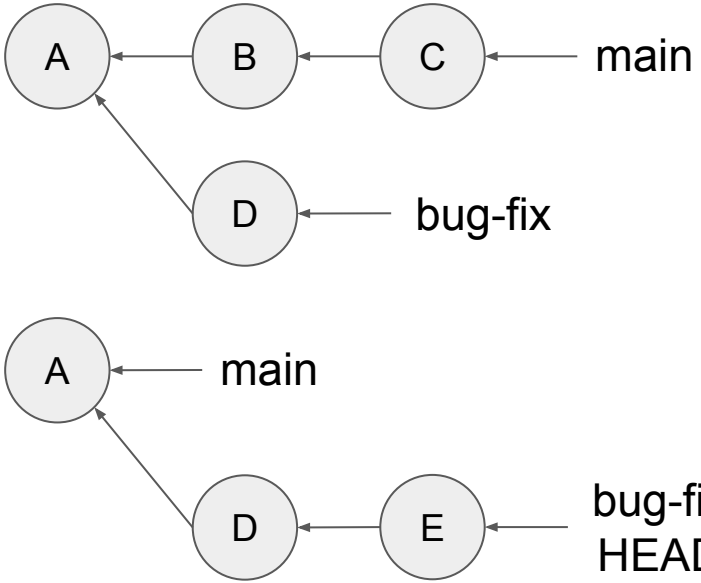
pushやpullの仕組み

リモート



push

ローカル

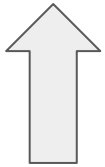


リモート：bug-fix→D→A

ローカル：bug-fix→E→D→A

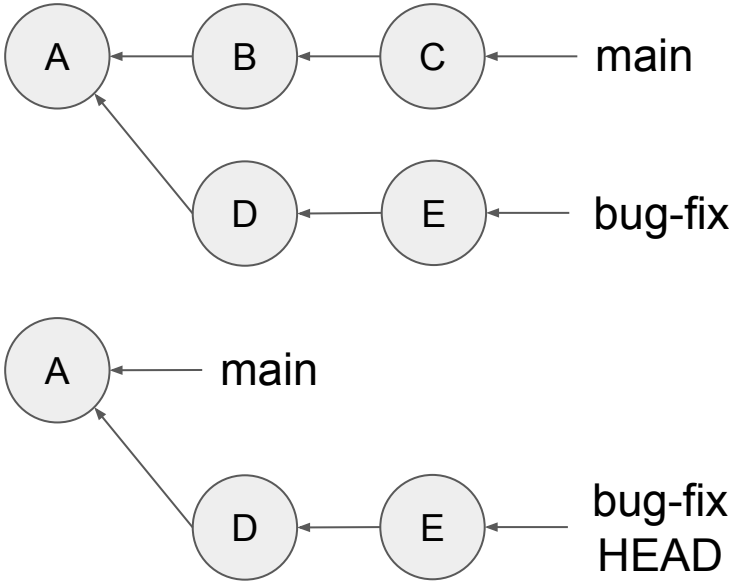
pushやpullの仕組み

リモート



push

ローカル

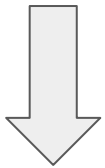


リモート：bug-fix→**D**→**A**

ローカル：bug-fix→E→**D**→**A**

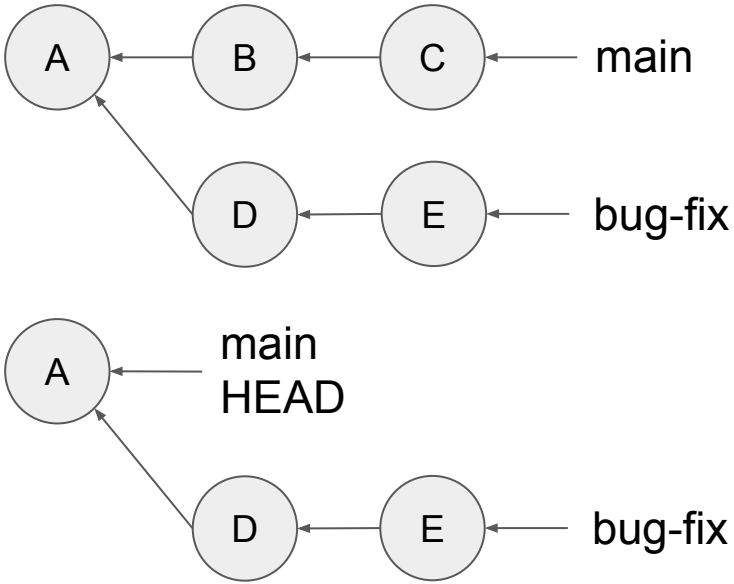
pushやpullの仕組み

リモート



pull

ローカル

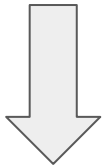


リモート：main→C→B→A

ローカル：main→A

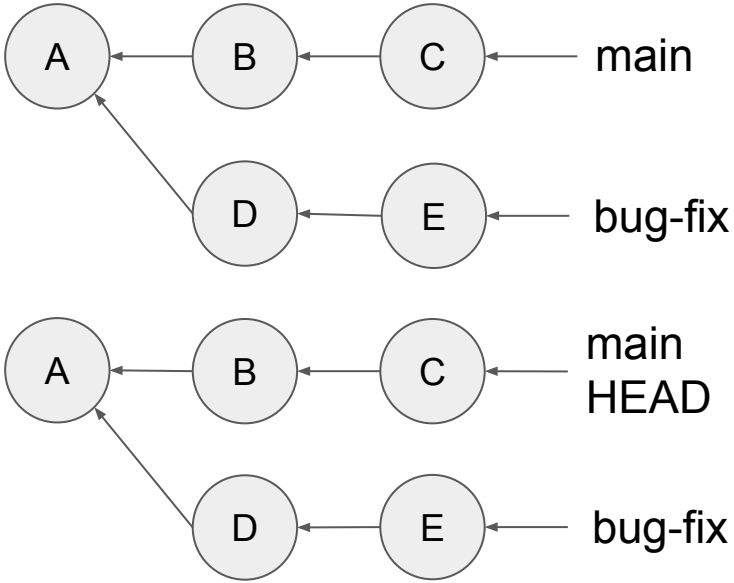
pushやpullの仕組み

リモート



pull

ローカル

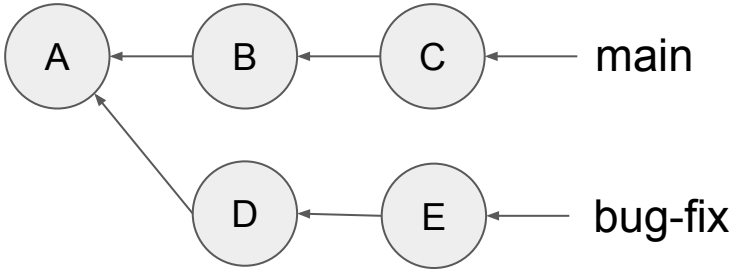


リモート：main→C→B→**A**

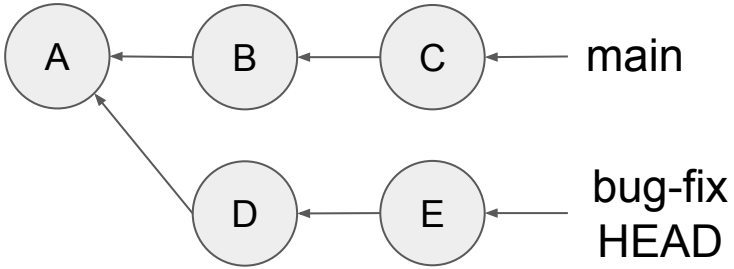
ローカル：main→**A**

pushやpullの仕組み

リモート

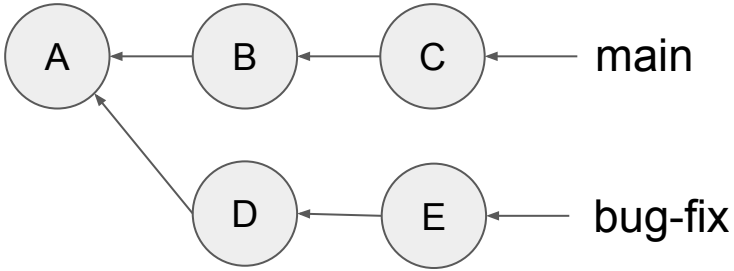


ローカル

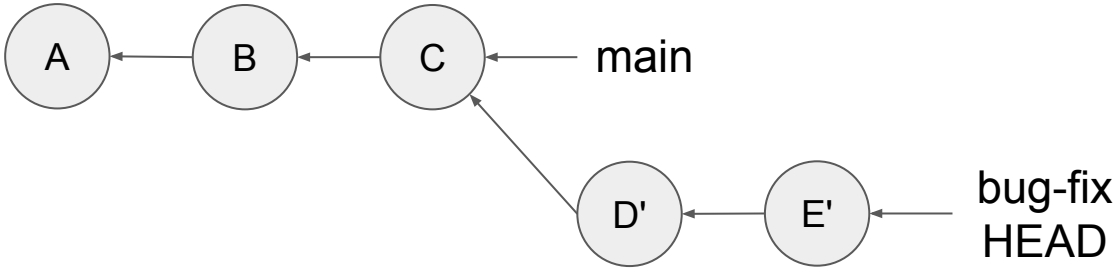


pushやpullの仕組み

リモート

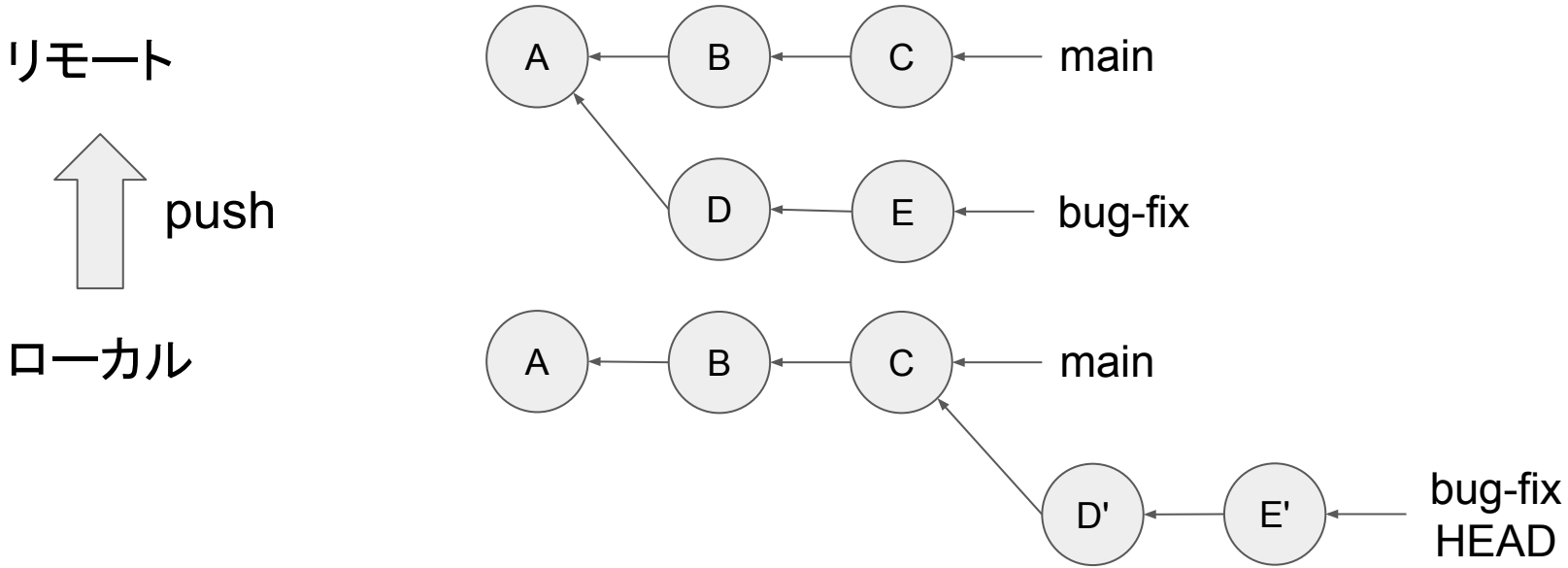


ローカル



rebase

pushやpullの仕組み

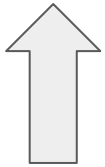


リモート：bug-fix→E→D→A

ローカル：bug-fix→E'→D'→C→B→A

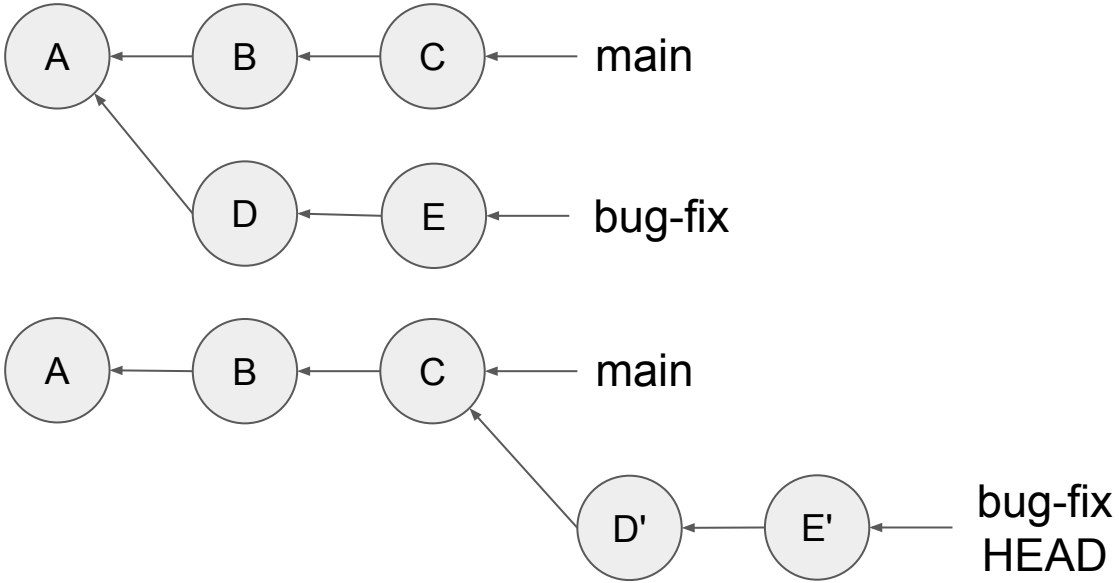
pushやpullの仕組み

リモート



push

ローカル

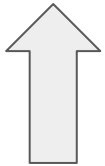


リモート：bug-fix→**E**→**D**→**A**

ローカル：bug-fix→E'→D'→C→B→A

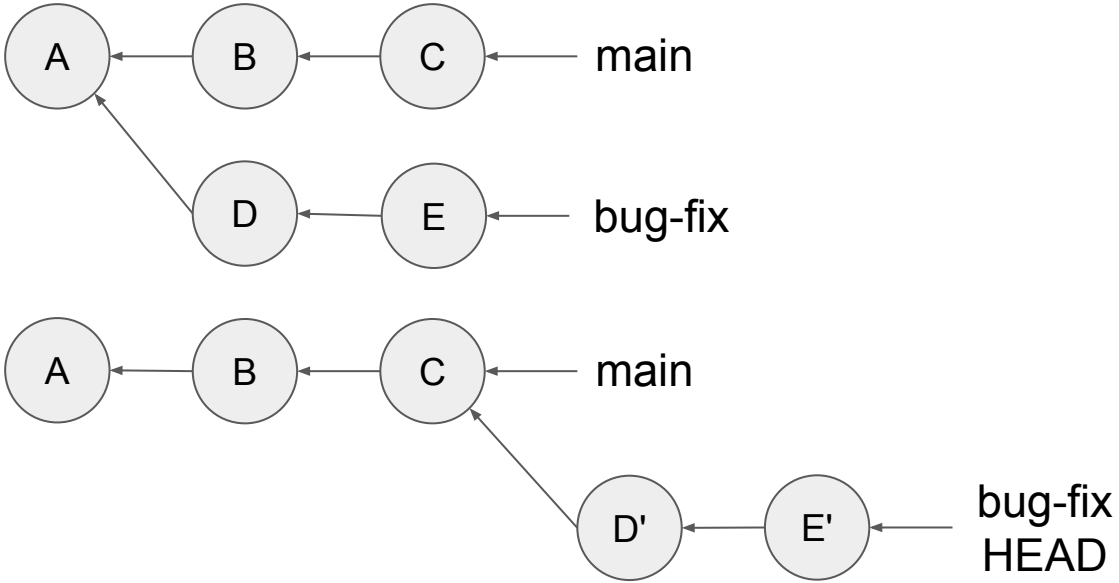
pushやpullの仕組み

リモート



push

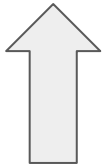
ローカル



一度pushしたブランチをrebaseすると
それ以降pushできなくなってしまう

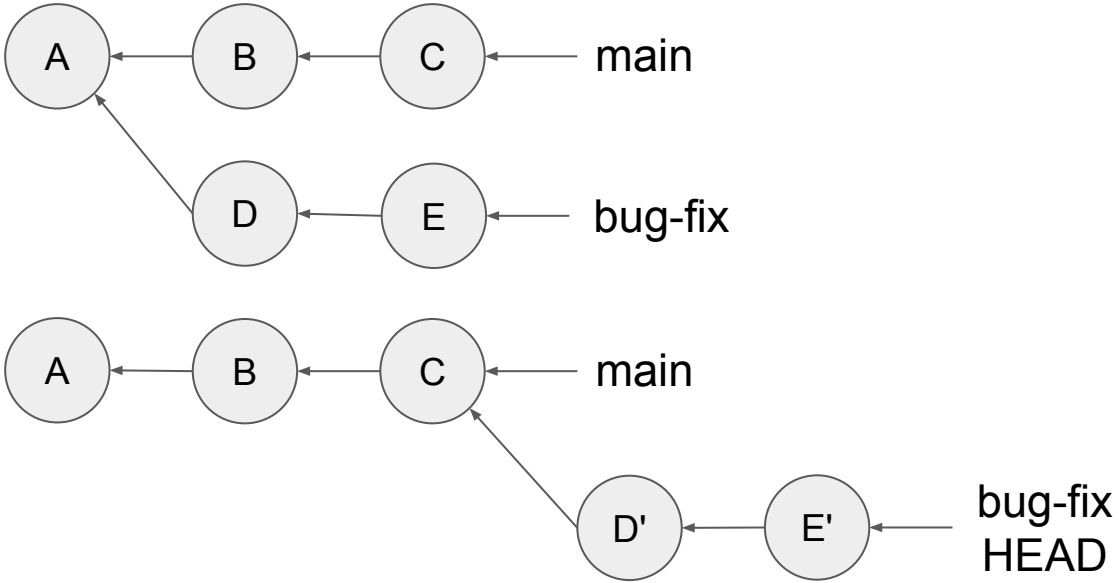
push -f オプション

リモート



push -f

ローカル

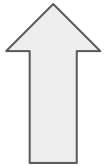


リモート：bug-fix→E→D→A

ローカル：bug-fix→E'→D'→C→B→A

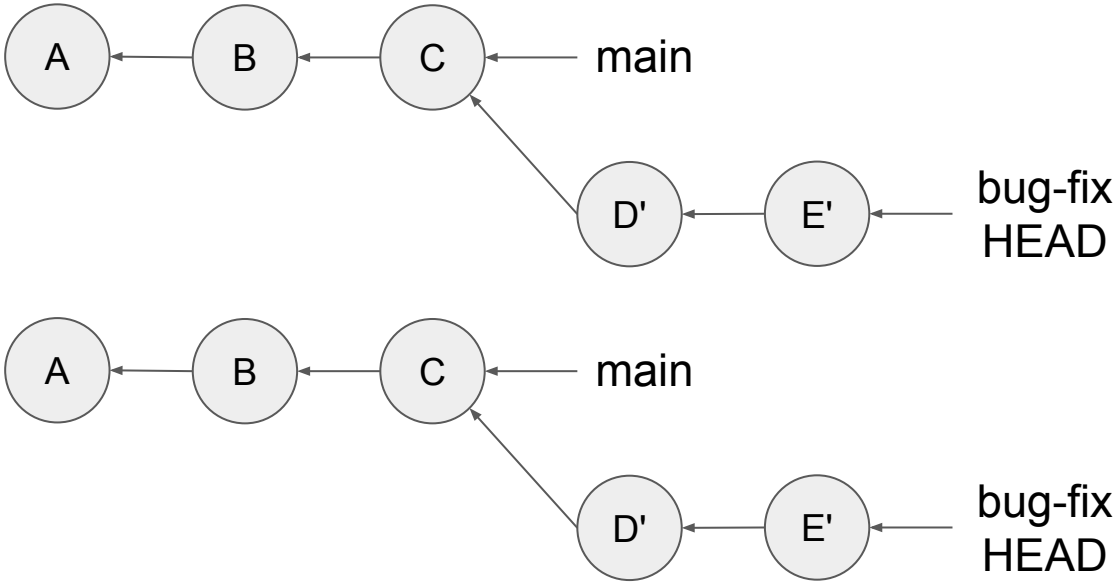
push -f オプション

リモート



push -f

ローカル

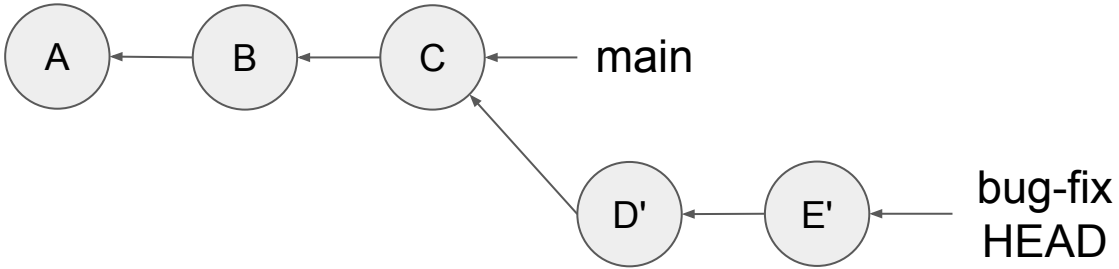


リモート：bug-fix→E'→D'→C→B→A

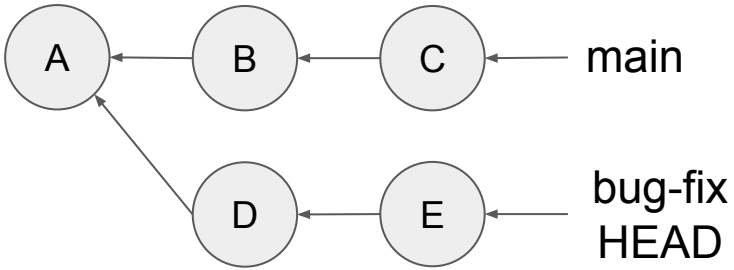
ローカル：bug-fix→E'→D'→C→B→A

push -f オプションを使うと他の開発者に影響する

リモート

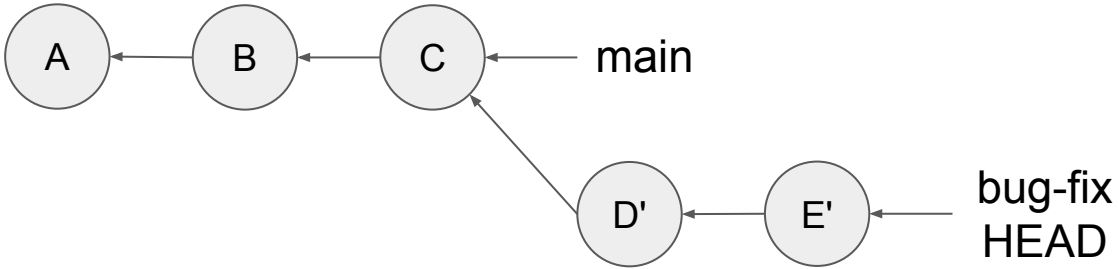


ローカル

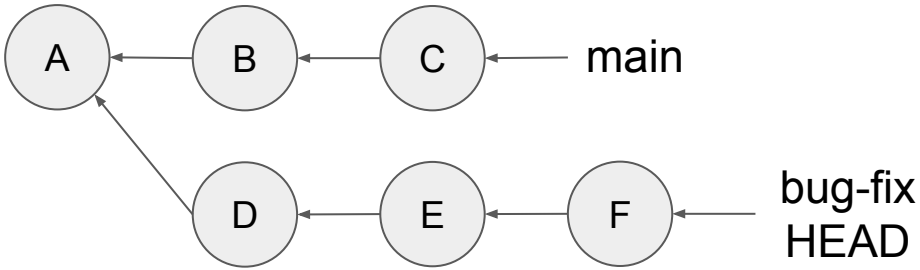


push -f オプションを使うと他の開発者に影響する

リモート

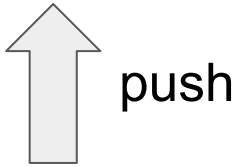


ローカル

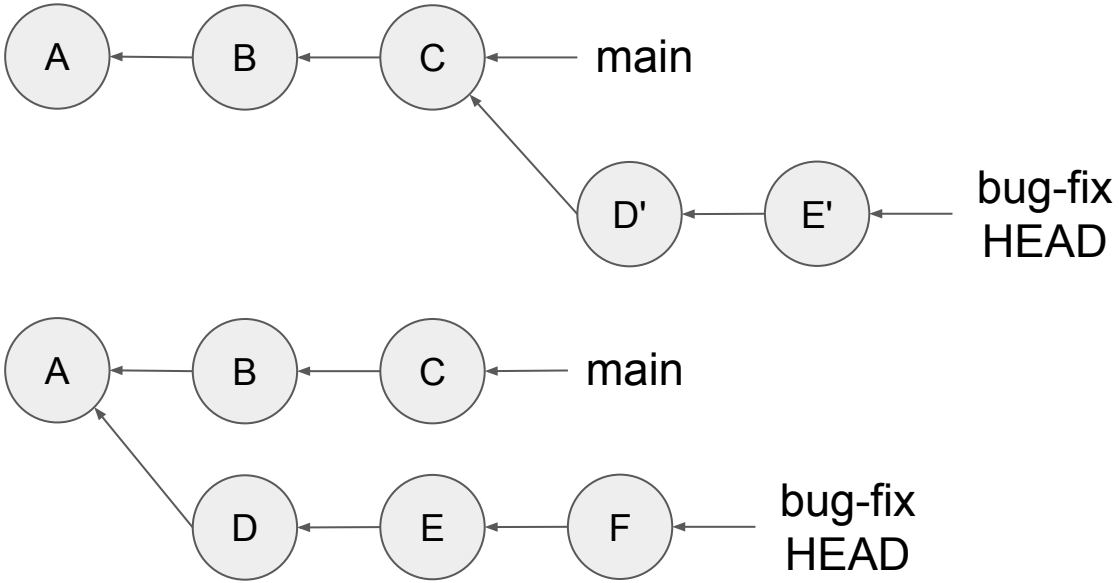


push -f オプションを使うと他の開発者に影響する

リモート



ローカル

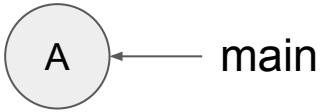


リモート：bug-fix→E'→D'→C→B→A

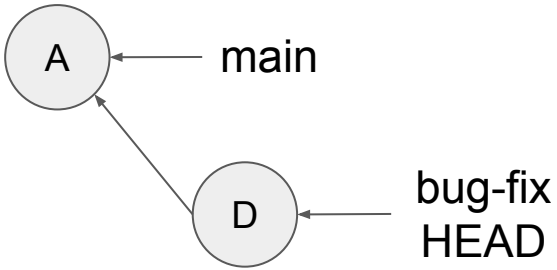
ローカル：bug-fix→F→E→D→A

resetよりもrevertの方が安全

リモート

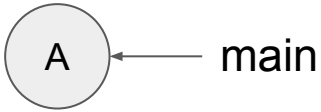


ローカル

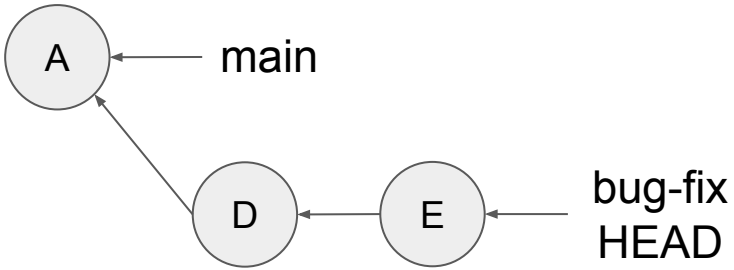


resetよりもrevertの方が安全

リモート

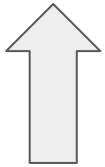


ローカル



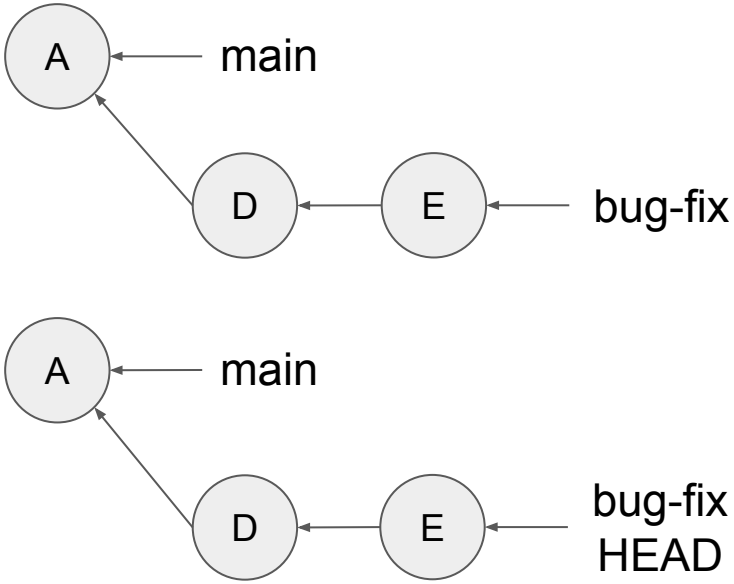
resetよりもrevertの方が安全

リモート



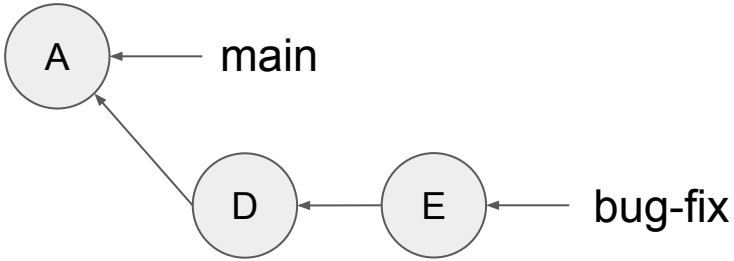
push

ローカル

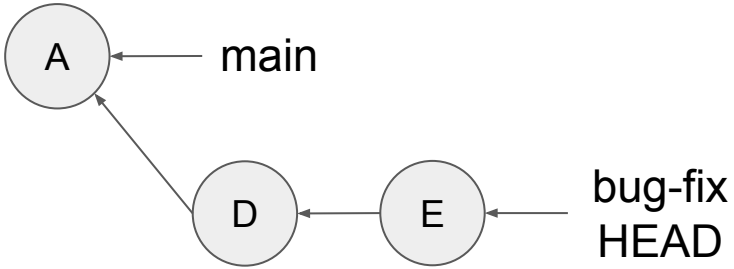


resetよりもrevertの方が安全

リモート



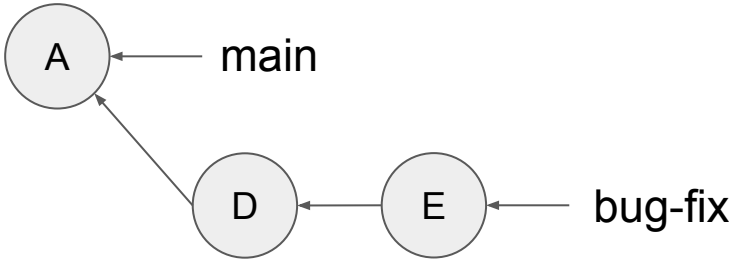
ローカル



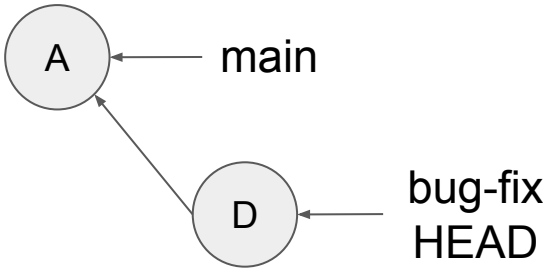
pushした後にEのコミットが不要なことに気づいた

resetよりもrevertの方が安全

リモート



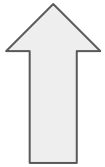
ローカル



reset

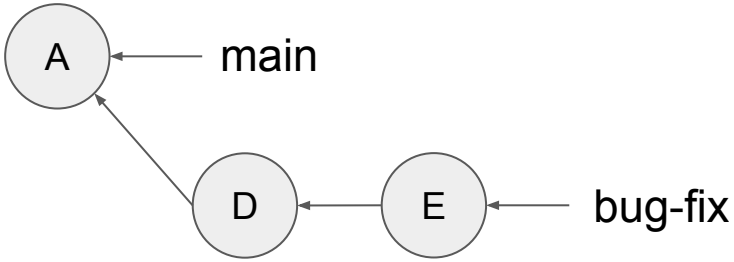
resetよりもrevertの方が安全

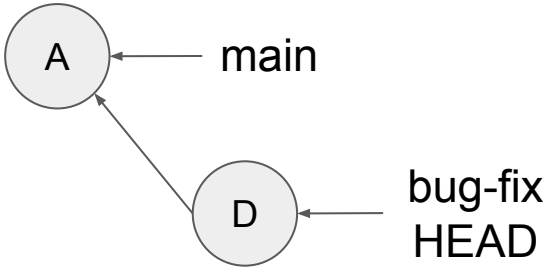
リモート



push

ローカル



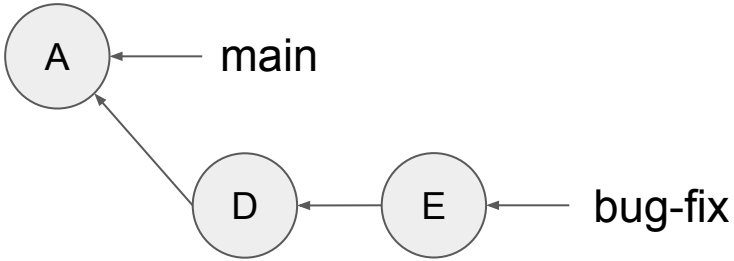


リモート：bug-fix→**E**→**D**→**A**

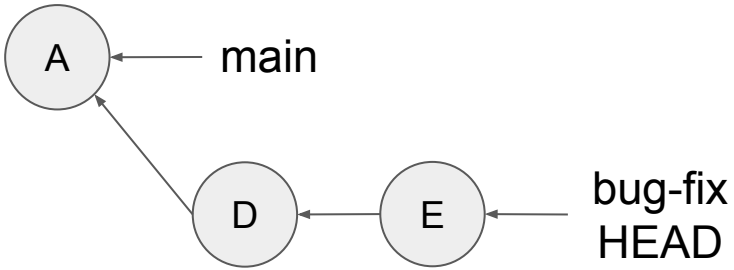
ローカル：bug-fix→D→A

resetよりもrevertの方が安全

リモート



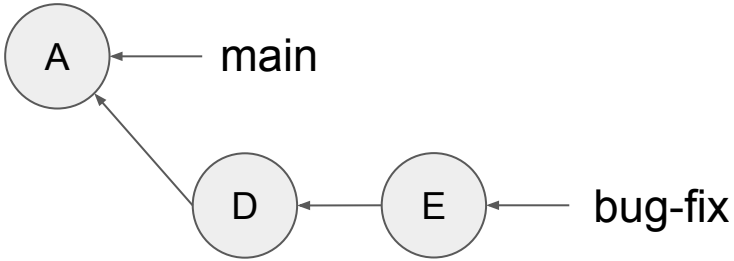
ローカル



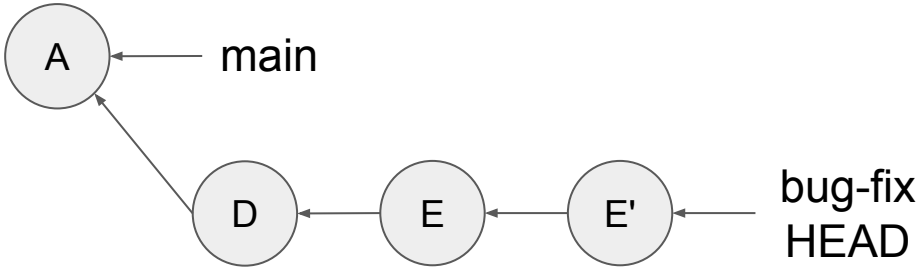
revertなら安全にコミットEを消せる

resetよりもrevertの方が安全

リモート



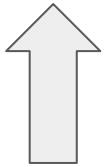
ローカル



revertなら安全にコミットEを消せる

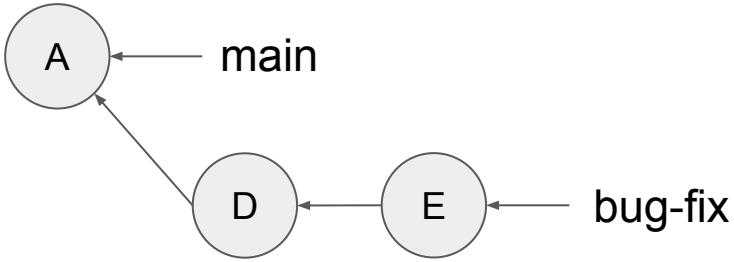
resetよりもrevertの方が安全

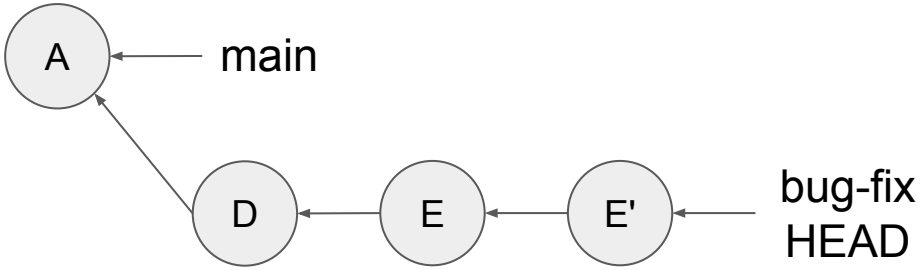
リモート



push

ローカル

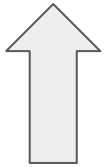




revertなら安全にコミットEを消せる

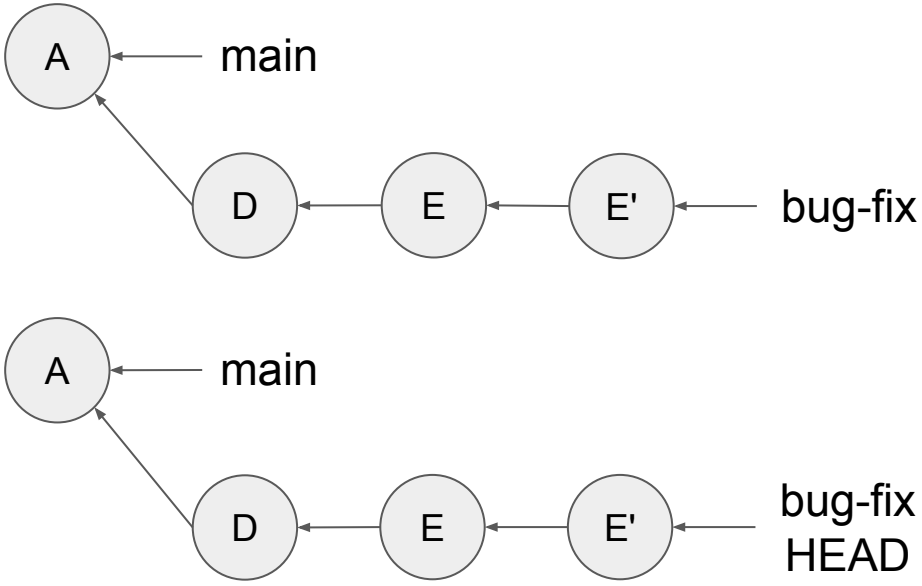
resetよりもrevertの方が安全

リモート



push

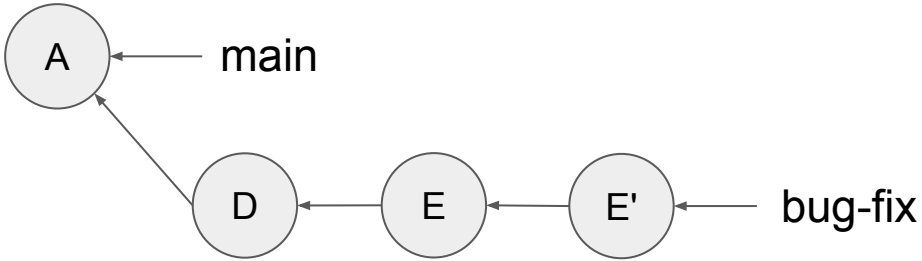
ローカル



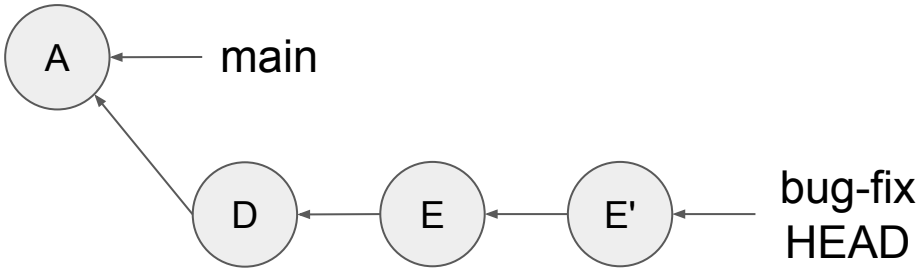
revertなら安全にコミットEを消せる

resetよりもrevertの方が安全

リモート

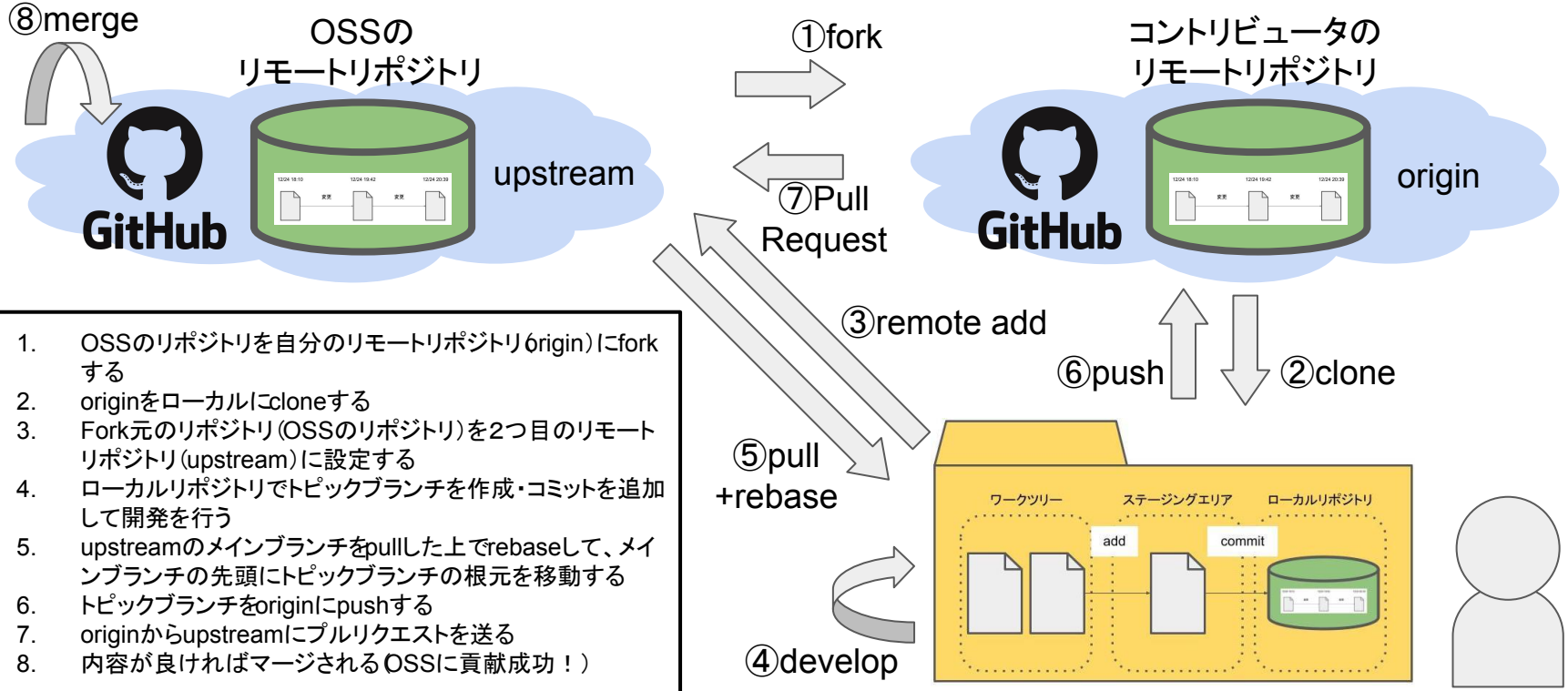


ローカル



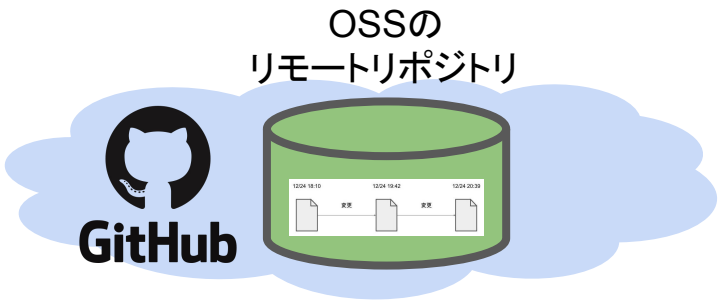
コミットDとコミットE'は同じ内容のコミット

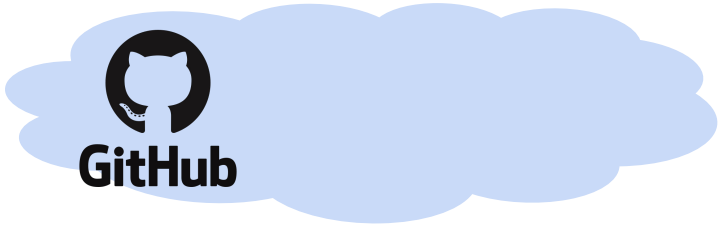
GitHubでのOSSの開発の流れ（完全版）

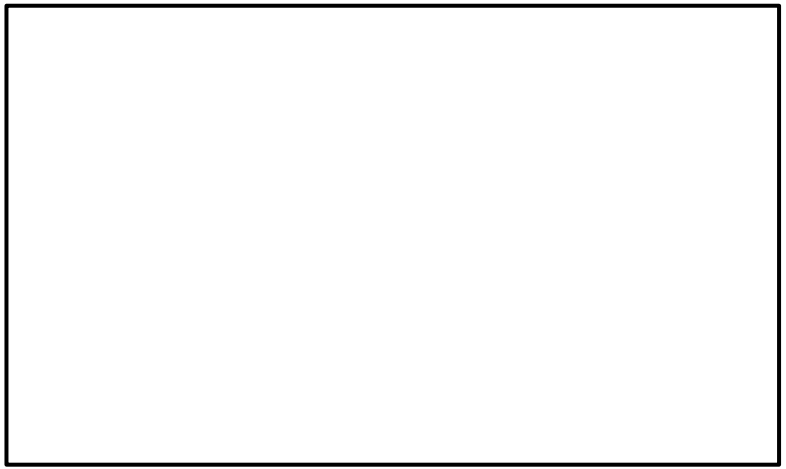


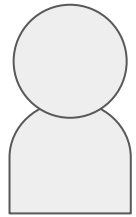
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする
3. Fork元のリポジトリ（OSSのリポジトリ）を2つ目のリモートリポジトリ（upstream）に設定する
4. ローカルリポジトリでトピックブランチを作成・コミットを追加して開発を行う
5. upstreamのメインブランチをpullした上でrebaseして、メインブランチの先頭にトピックブランチの根元を移動する
6. トピックブランチをoriginにpushする
7. originからupstreamにプルリクエストを送る
8. 内容が良ければマージされる（OSSに貢献成功！）

GitHubでのOSSの開発の流れ（完全版）

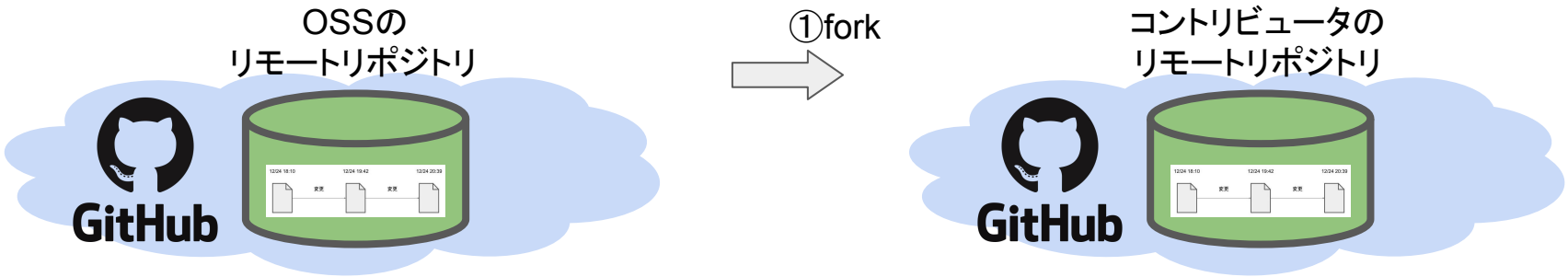




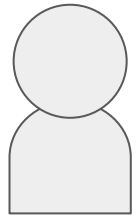




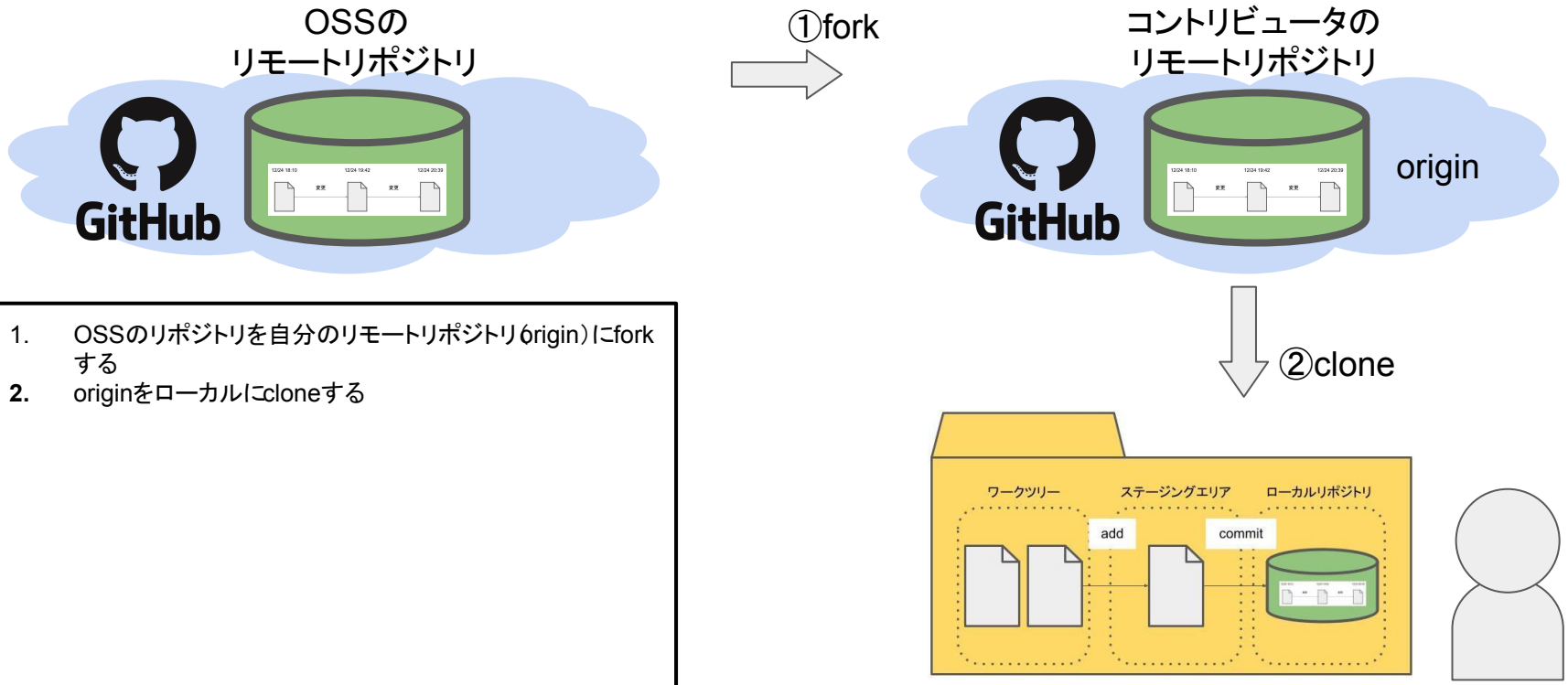
GitHubでのOSSの開発の流れ（完全版）



1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする

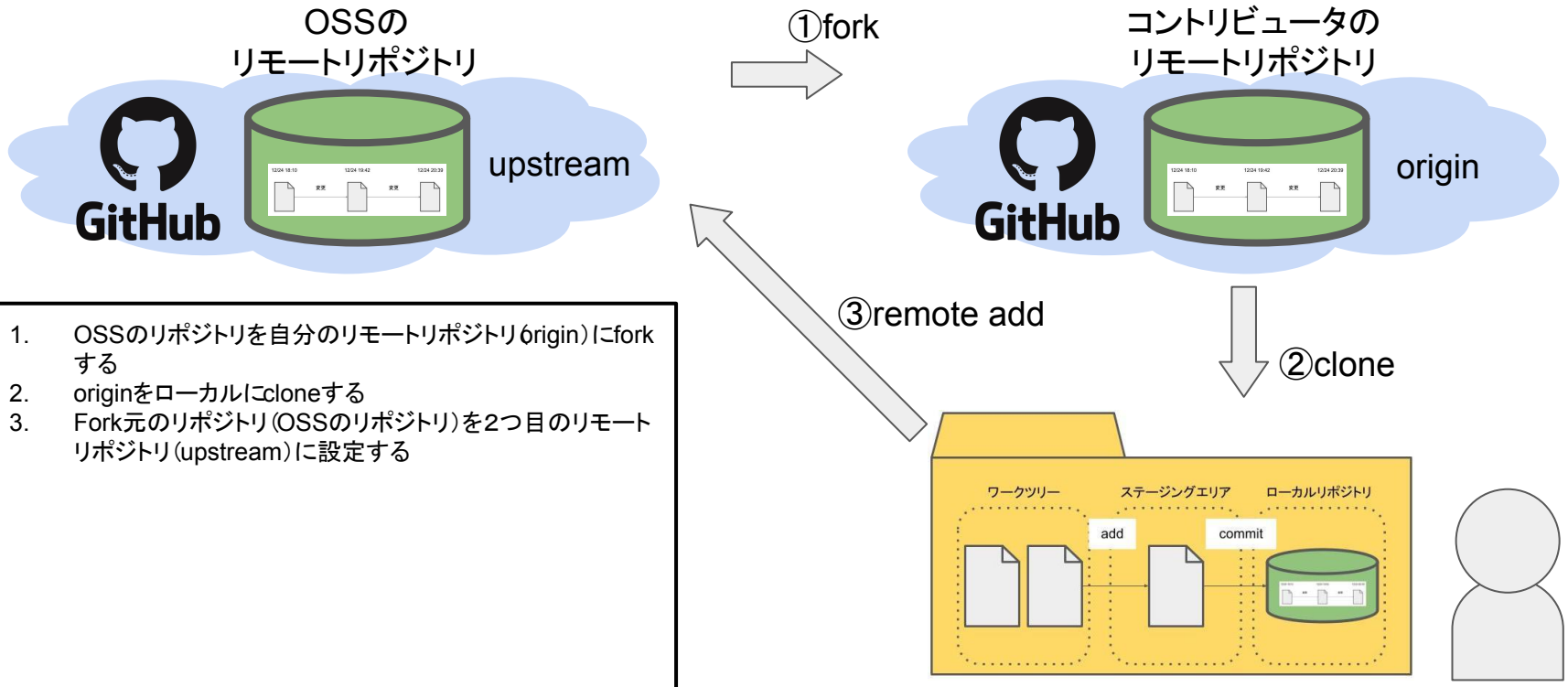


GitHubでのOSSの開発の流れ（完全版）



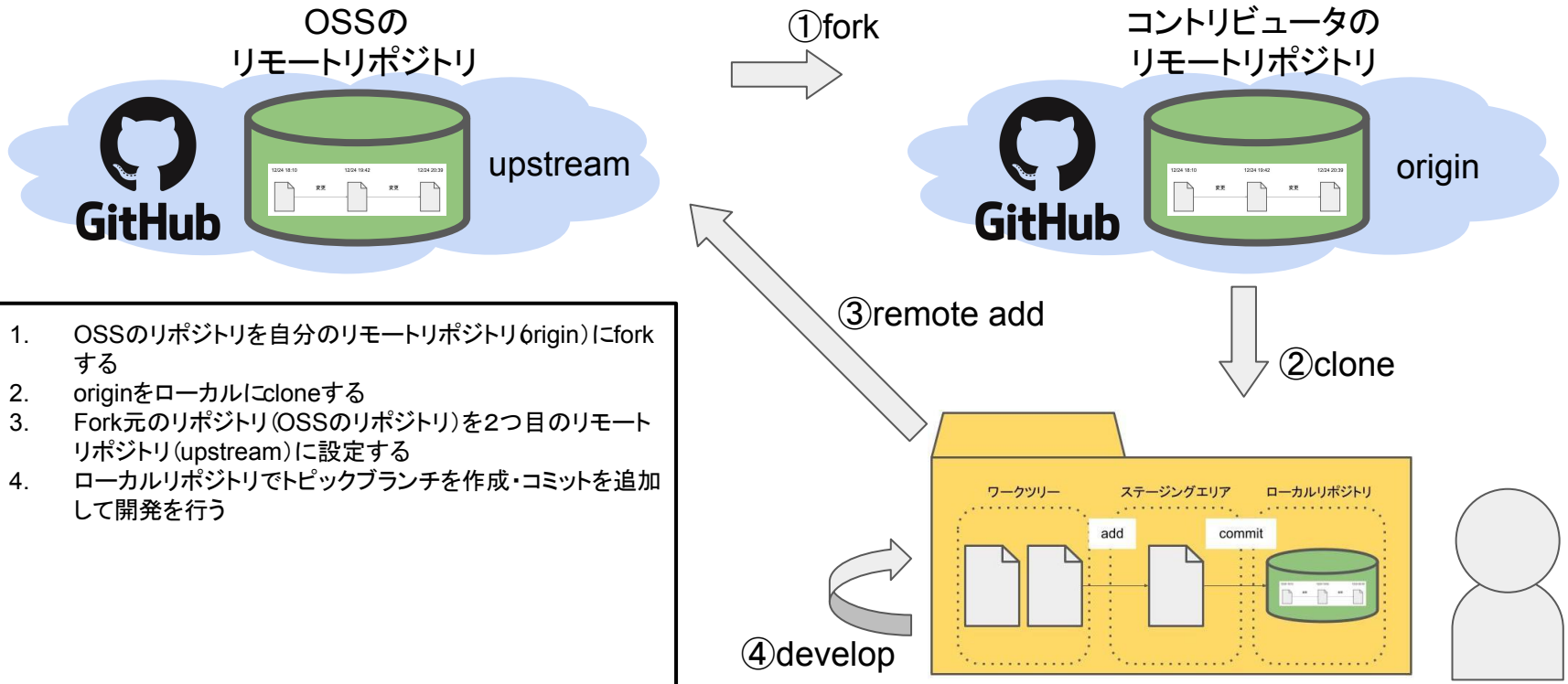
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする

GitHubでのOSSの開発の流れ（完全版）



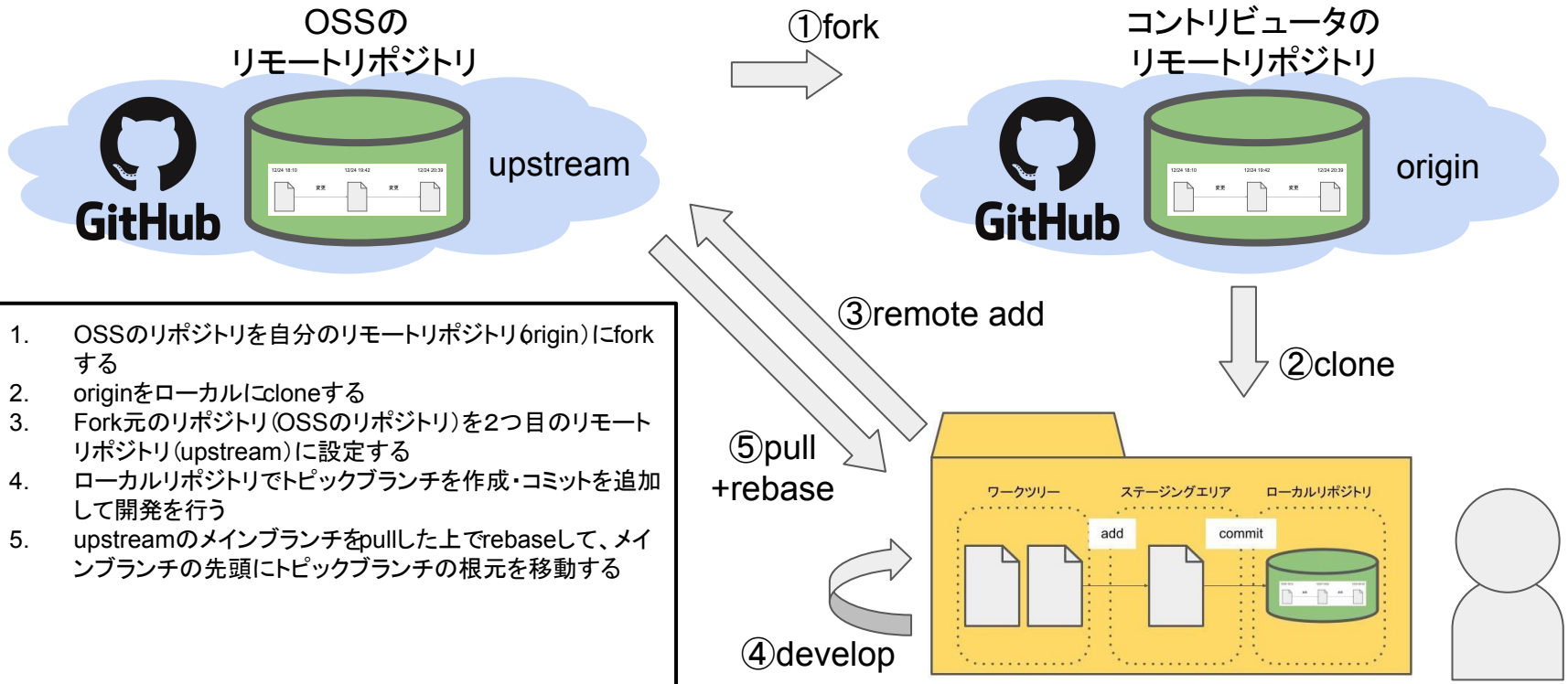
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする
3. Fork元のリポジトリ（OSSのリポジトリ）を2つ目のリモートリポジトリ（upstream）に設定する

GitHubでのOSSの開発の流れ（完全版）



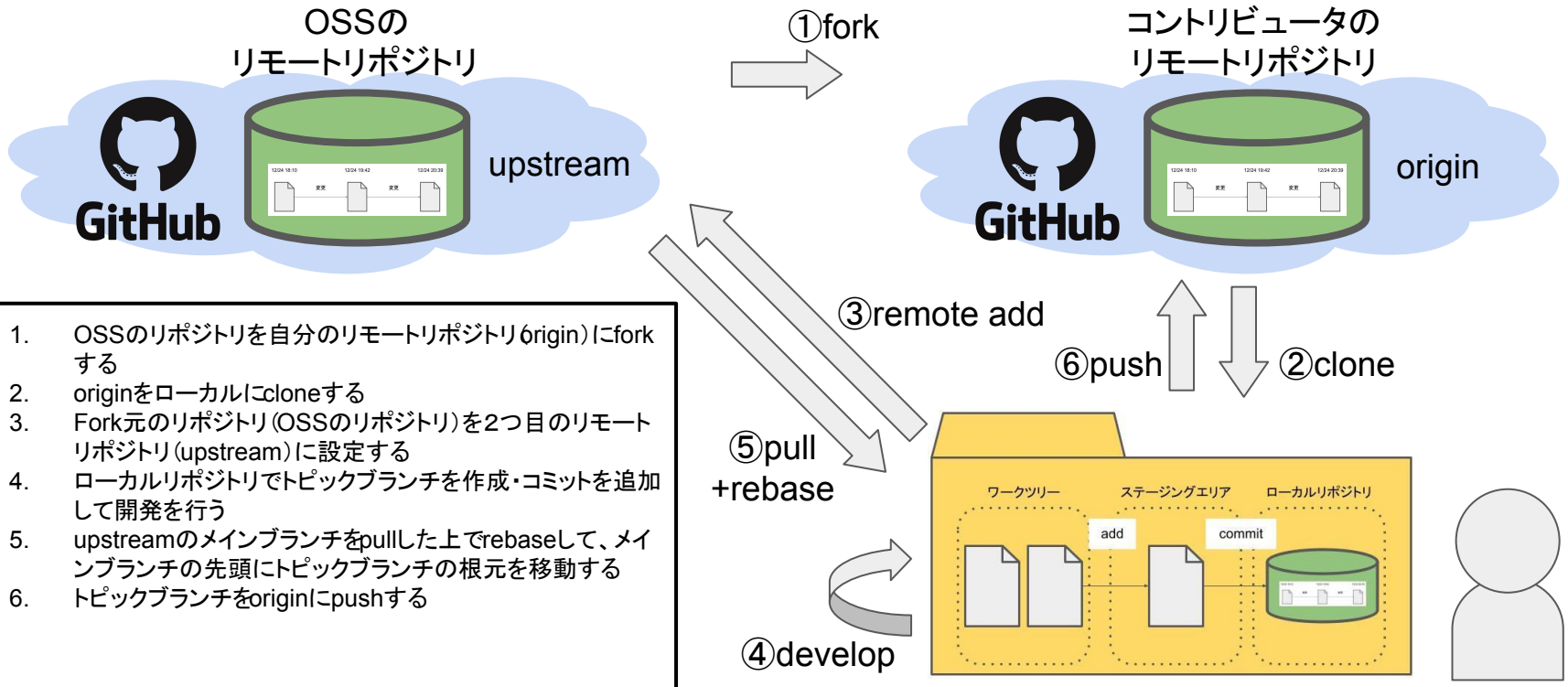
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする
3. Fork元のリポジトリ（OSSのリポジトリ）を2つ目のリモートリポジトリ（upstream）に設定する
4. ローカルリポジトリでトピックブランチを作成・コミットを追加して開発を行う

GitHubでのOSSの開発の流れ（完全版）



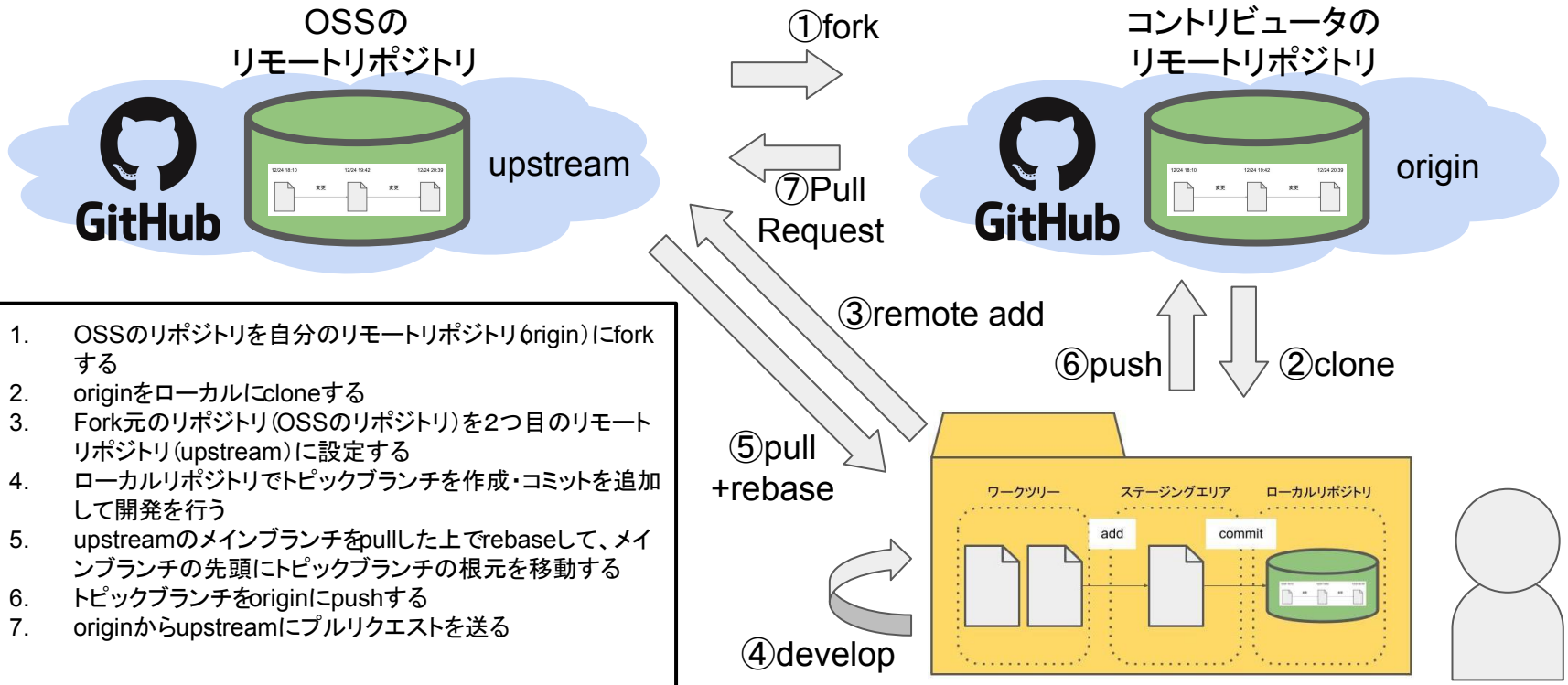
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする
3. Fork元のリポジトリ（OSSのリポジトリ）を2つ目のリモートリポジトリ（upstream）に設定する
4. ローカルリポジトリでトピックブランチを作成・コミットを追加して開発を行う
5. upstreamのメインブランチをpullした上でrebaseして、メインブランチの先頭にトピックブランチの根元を移動する

GitHubでのOSSの開発の流れ（完全版）



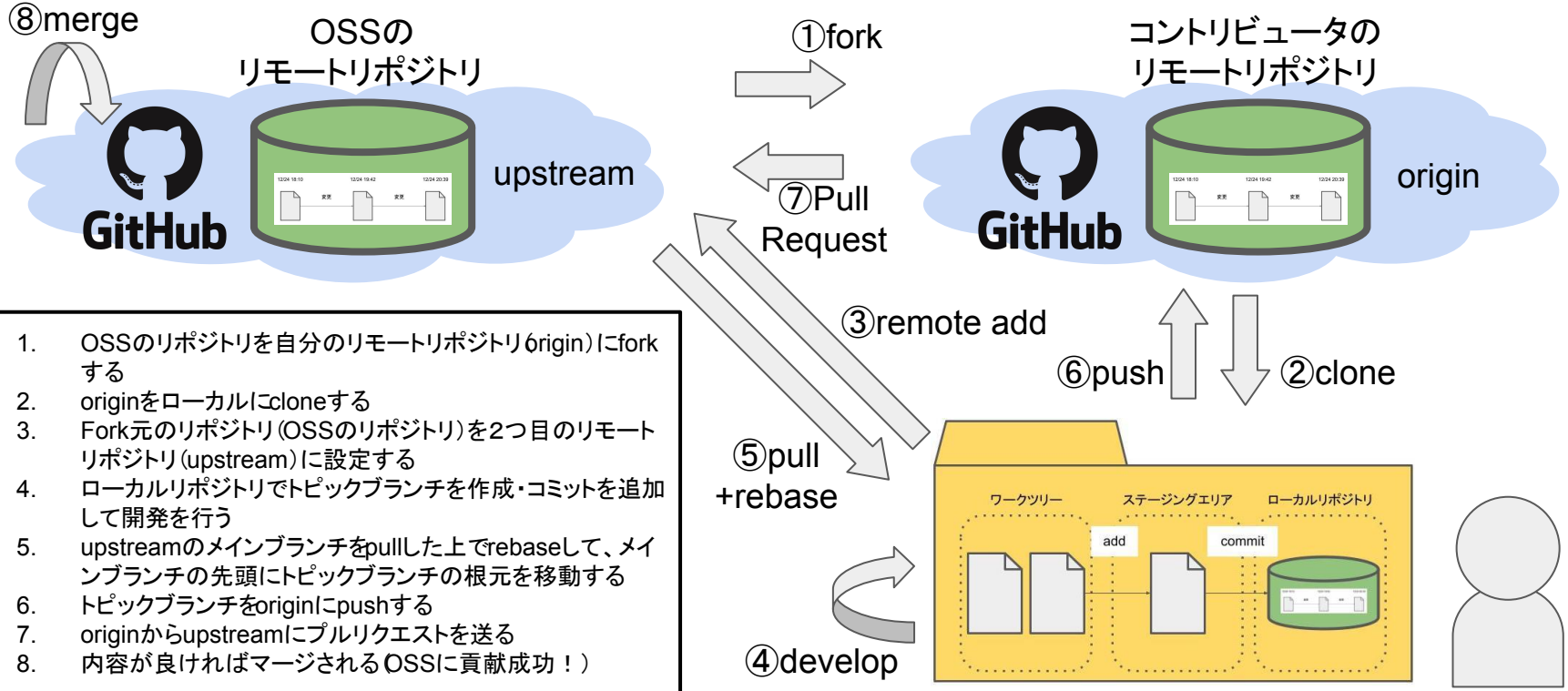
1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする
3. Fork元のリポジトリ（OSSのリポジトリ）を2つ目のリモートリポジトリ（upstream）に設定する
4. ローカルリポジトリでトピックブランチを作成・コミットを追加して開発を行う
5. upstreamのメインブランチをpullした上でrebaseして、メインブランチの先頭にトピックブランチの根元を移動する
6. トピックブランチをoriginにpushする

GitHubでのOSSの開発の流れ（完全版）



1. OSSのリポジトリを自分のリモートリポジトリ(origin)にforkする
2. originをローカルにcloneする
3. Fork元のリポジトリ(OSSのリポジトリ)を2つ目のリモートリポジトリ(upstream)に設定する
4. ローカルリポジトリでトピックブランチを作成・コミットを追加して開発を行う
5. upstreamのメインブランチをpullした上でrebaseして、メインブランチの先頭にトピックブランチの根元を移動する
6. トピックブランチをoriginにpushする
7. originからupstreamにプルリクエストを送る

GitHubでのOSSの開発の流れ（完全版）



1. OSSのリポジトリを自分のリモートリポジトリ（origin）にforkする
2. originをローカルにcloneする
3. Fork元のリポジトリ（OSSのリポジトリ）を2つ目のリモートリポジトリ（upstream）に設定する
4. ローカルリポジトリでトピックブランチを作成・コミットを追加して開発を行う
5. upstreamのメインブランチをpullした上でrebaseして、メインブランチの先頭にトピックブランチの根元を移動する
6. トピックブランチをoriginにpushする
7. originからupstreamにプルリクエストを送る
8. 内容が良ければマージされる（OSSに貢献成功！）