人と技術で次代を拓く

MEITEC

Engineering Firm at The Core

Git入門

目次

バージョン管理とは

- **ファイルへの変更履歴をファイルの内容も含めて記録すること。**
  このシステムをバージョン管理システム
  (VersionControl System =VCS**)という。**

【バージョン管理システムの種類】

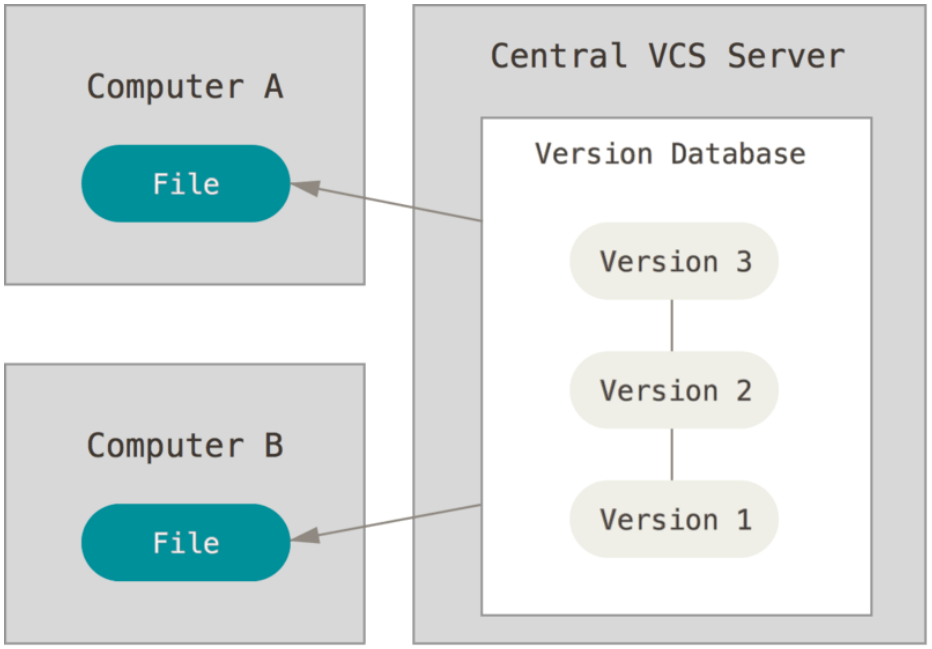
- **ローカル型**:　編集者が自分のディレクトリに別の名前で保管していく
  例:Revision Control System(RCS)
- **集中型**：　次スライドに記載
- **分散型**：　次スライドに記載

バージョン管理システムの種類

●集中型

バージョン管理対象のファイルを1つのサーバで一元管理する。

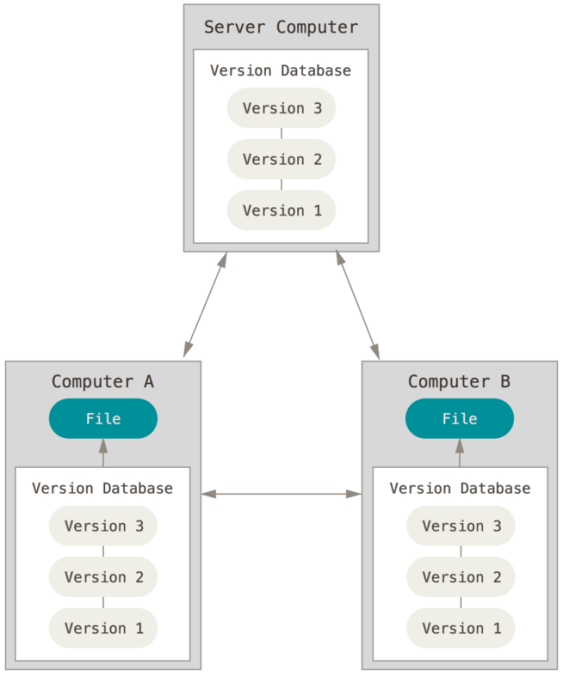
例：Concurrent Version System(CVS)
Apache Subversion(SVN)



●分散型

サーバ上のバージョン管理対象のファイルのコピーを作業者が手元に作る。

例：Git、Mercurial、Bazaar



バージョン管理ソフトの存在意義

● 背景/課題

- 現代のソフトウェアは、便利さや機能の進化と共に複雑さも増し、設計・開発においてソースコードを含めた複数のファイルになっていることが多い。
- 開発においてファイルの変更が、複数人によって頻繁に行われるため、それぞれの変更を全て管理する事は困難である。

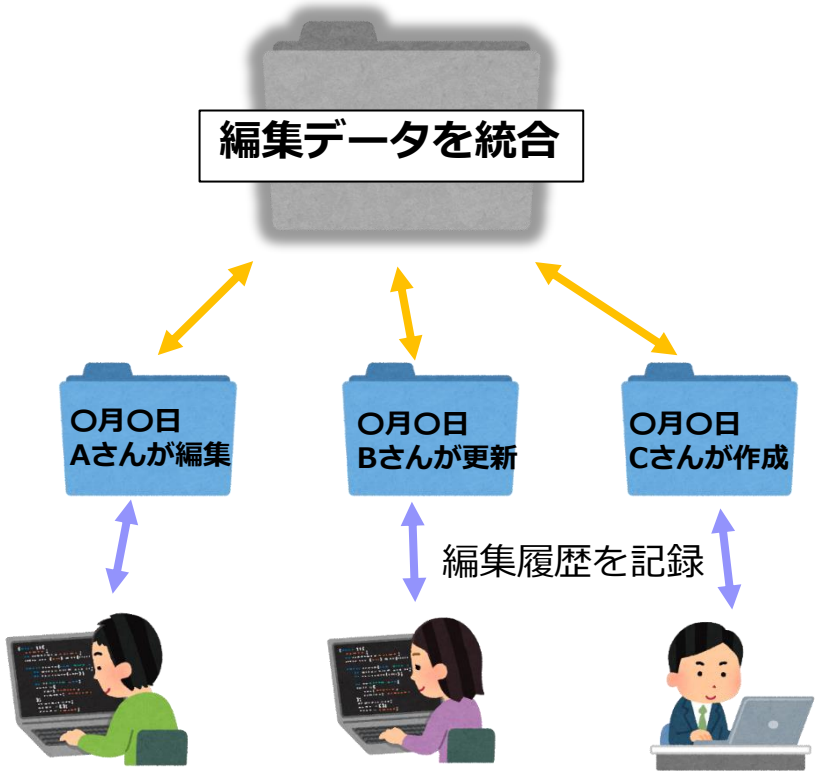
バージョン管理ソフトを導入して、個々の変更を管理

Gitとは

●分散型のバージョン管理システム

特徴

- 高速・柔軟性・堅牢性に突出しており、
  大規模プロジェクトに対応しやすい。
- ブランチ、 マージなどの機能により、
  変更履歴の管理や統合を行いやすい。
- 規模の大きさを問わず様々な
  プロジェクトで利用されている。



複数人が分担して編集

Gitで使用される用語

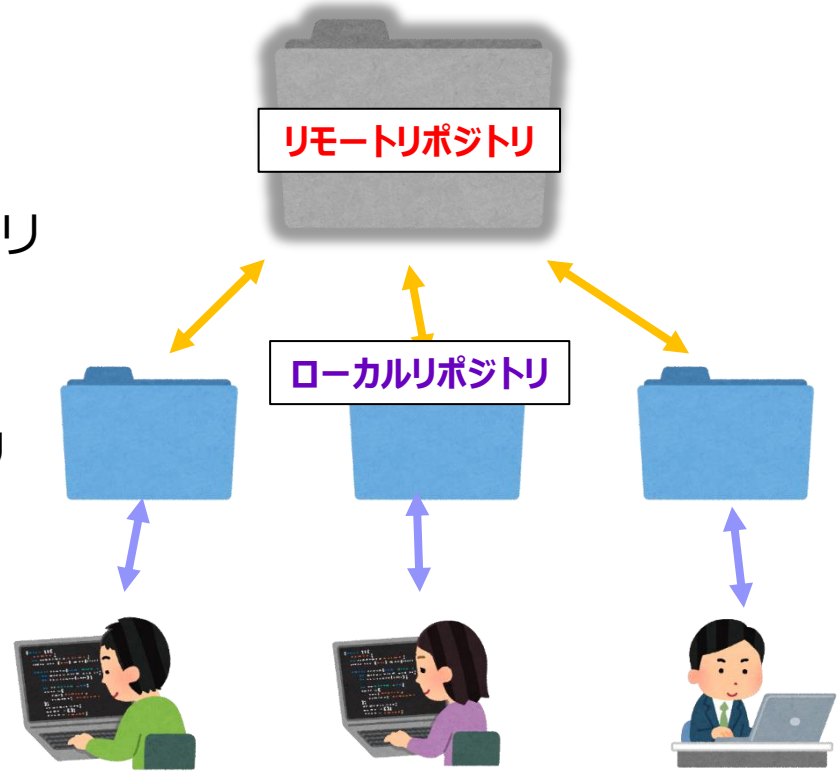
repository（リポジトリ）とは

- **ソースコードやディレクトリ構造のデータを格納するデータ構造のこと**

  ファイル、ディレクトリの状態、変更履歴を保存する場所

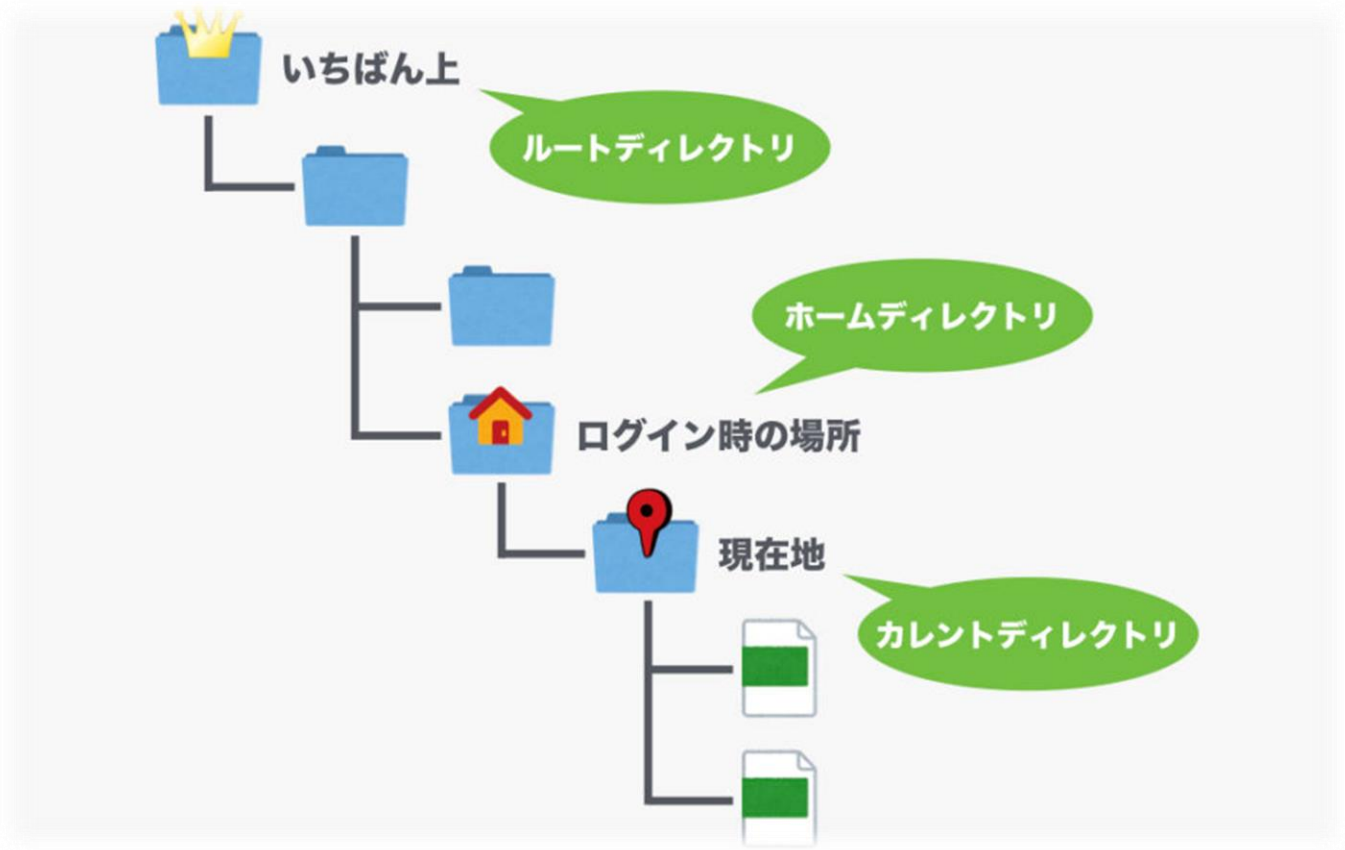
- **リモートリポジトリ**
  共有して使用するマスターリポジトリ

- **ローカルリポジトリ**
  自分のPC内にある作業用リポジトリ



ワークツリーとは

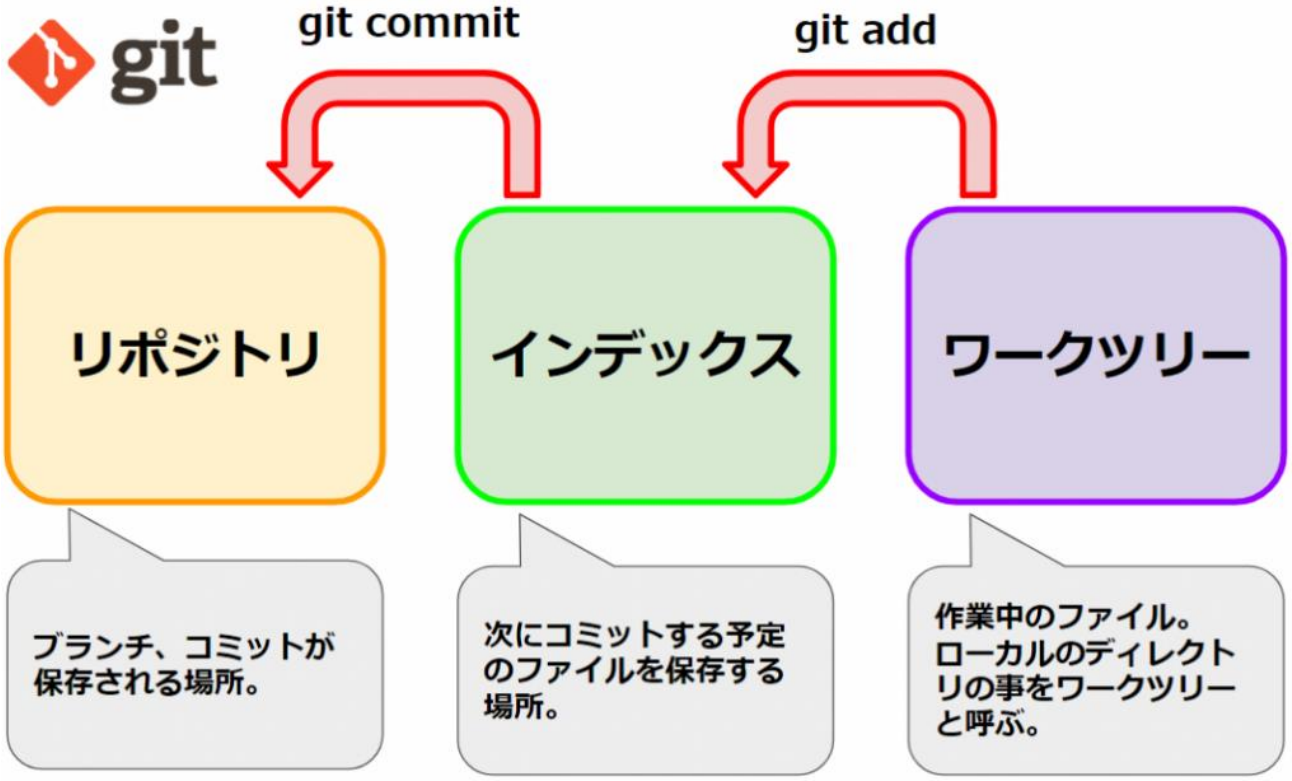
- 作業を行なうローカルのディレクトリのこと



インデックスとは

- **リポジトリにコミットするための準備をする場所**

※インデックスへ移動することを「ステージする」と呼ばれる。



Bareリポジトリとは

- **作業ディレクトリを持たず、更新情報のみを持つリポジトリ（更新管理用リポジトリ）**

⇔**ノンベア（Nonbare）**リポジトリ

ワーキングディレクトリを持つリポジトリ（作業用リポジトリ）

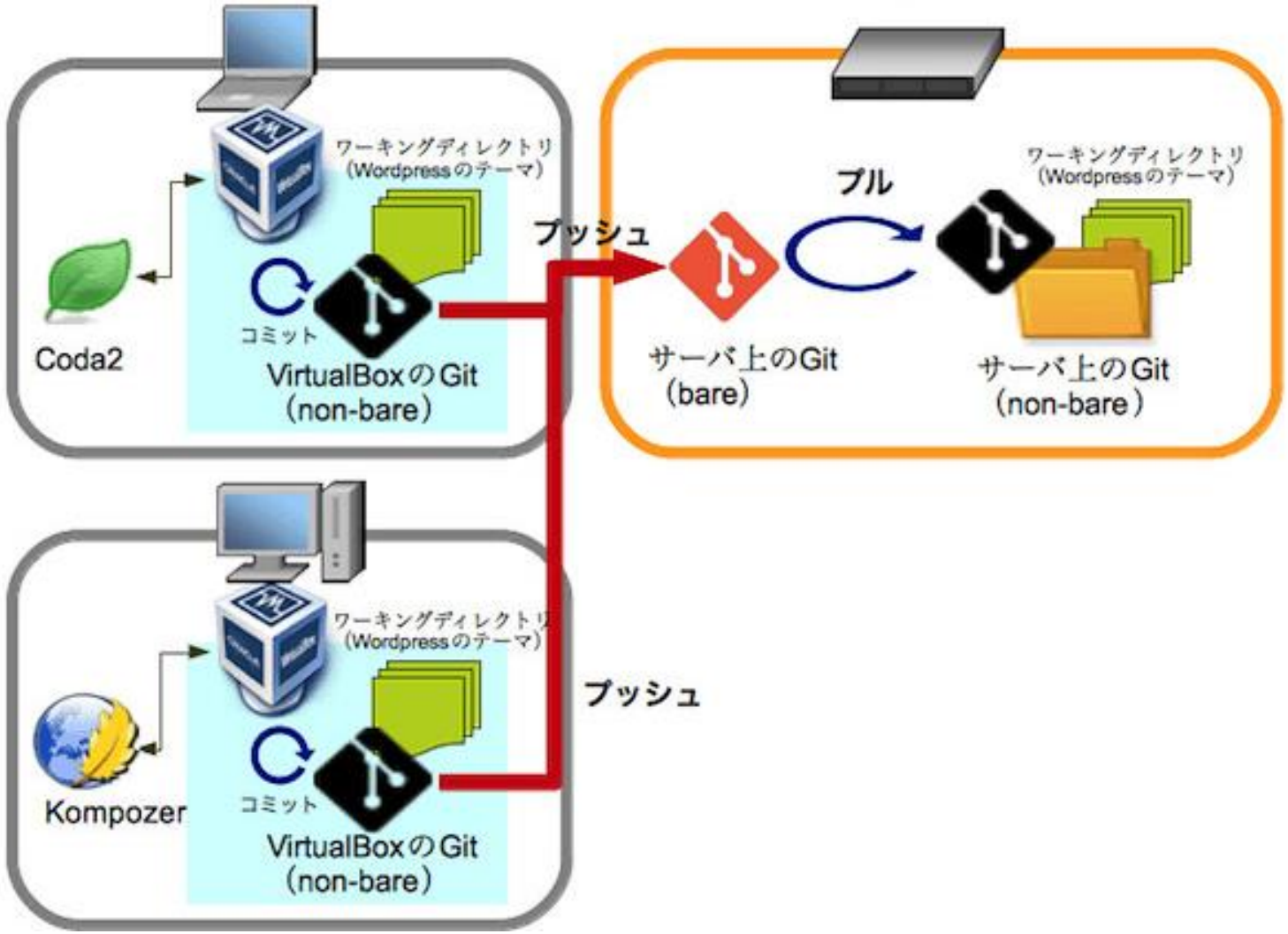
オプションを付けずにコマンドを実行するとノンベアリポジトリが作成される。

コマンド例：

$git init --bare –shared

⇒　複数のユーザで共有可能なベアリポジトリを作成

Bareリポジトリ構成例



Gitの基本操作コマンドの説明

initとは

- **「リポジトリを新規に作成」する時に使用するコマンド**

現在または指定したディレクトリに「.git」（隠し属性）という
リポジトリを構成するディレクトリが作成される。
「.git」にはGitで使用するファイルが新規に作成される。

コマンド例：

$git init [ディレクトリ名]

↓

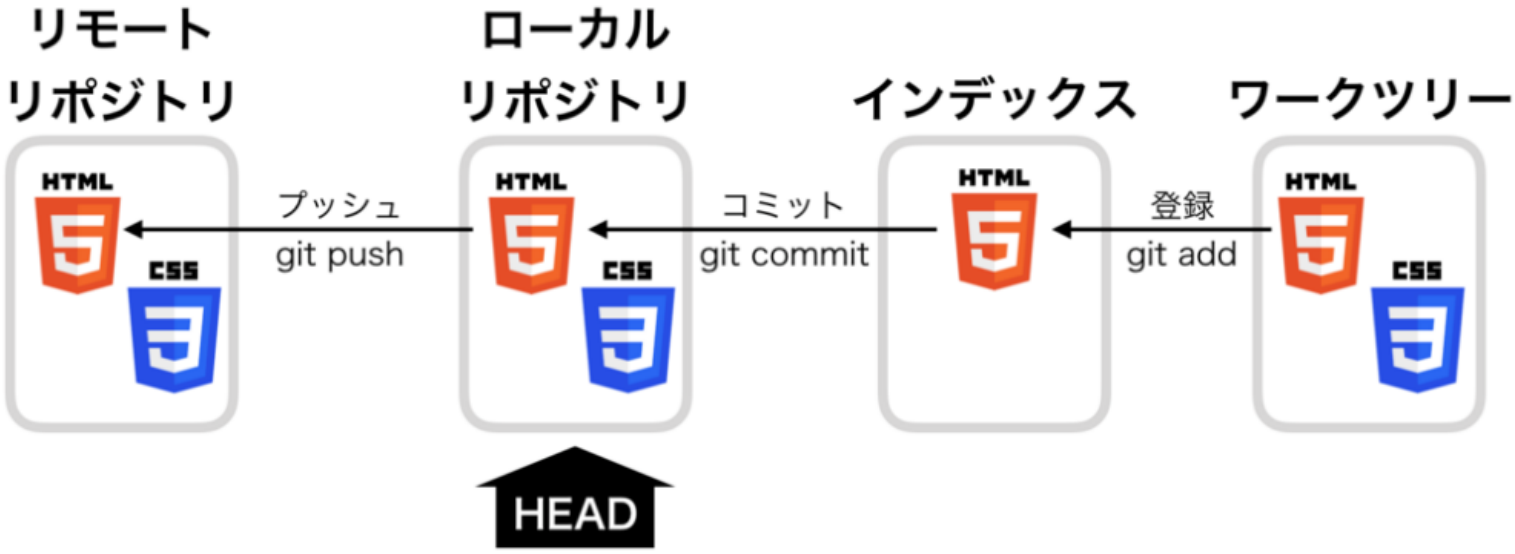
Gitが管理対象とするディレクトリ

addとは

- **ファイルをインデックスに追加するコマンド**

コマンド例：

$git add　カレントディレクトリの全てのファイルを
インデックスに追加する。

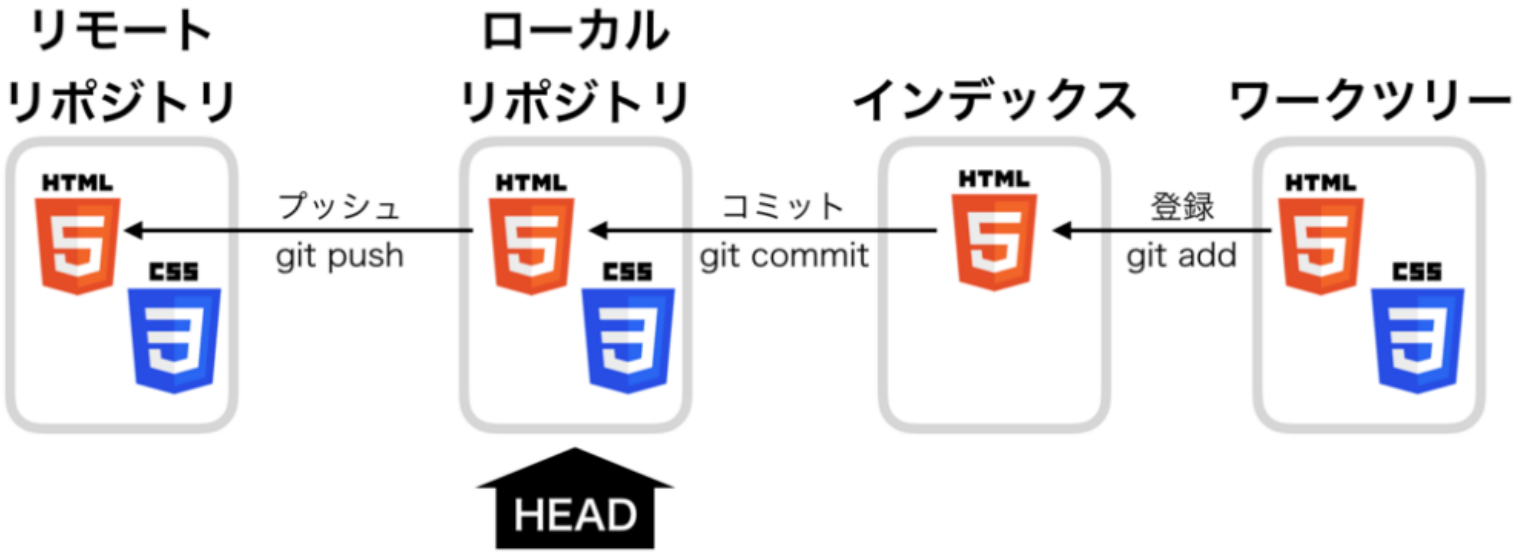


commitとは

●追加・変更したファイルをGitに登録ためのコマンド

コマンド例：

$git commit　　インデックスの内容を記録する。



commit メッセージとは

- **コミットした際のメッセージを残す機能**

  なぜ変更したのかなどを記載する。

  5W1Hに基づいてメッセージ作成が推奨される。

コマンド例：　$git commit –m “コミットテスト”

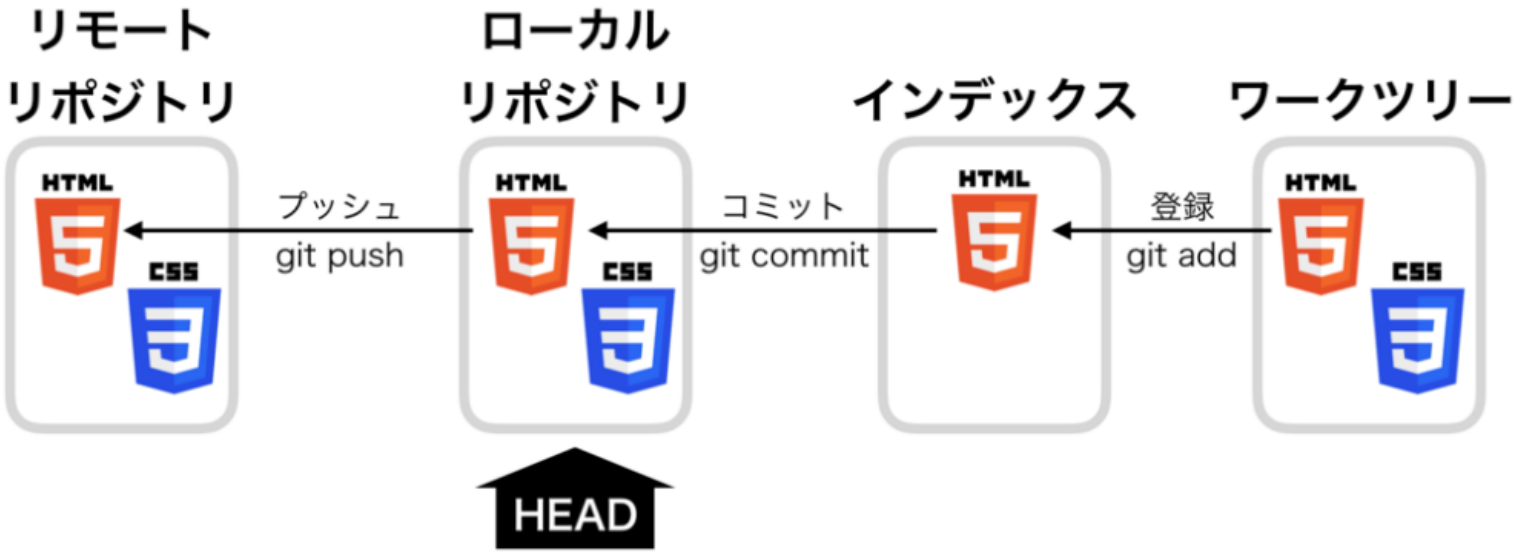
↓

commit メッセージ

pushとは

- **ローカルリポジトリの内容をリモートリポジトリにアップロードするコマンド**

コマンド例：　$git push



clone とは

- **リモートリポジトリをコピーしてローカルリポジトリを作成するコマンド**

コマンド例：

$git clone [リポジトリ名]
→ディレクトリ直下にクローンを作成する。

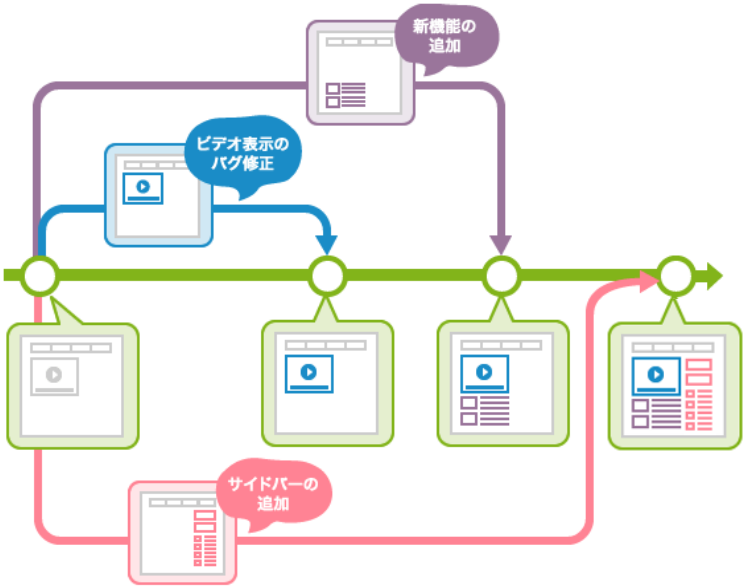
$git clone [リポジトリ名] [ディレクトリ名]
→指定したディレクトリにクローンを作成する。

branchとは

- **1つのプロジェクトから分岐させることで、プロジェクト本体に影響を与えずに並行して開発を行える機能**

コマンド例：

$git branch [ブランチ名]	ブランチを作成する。
$git branch	ブランチ一覧を表示する。

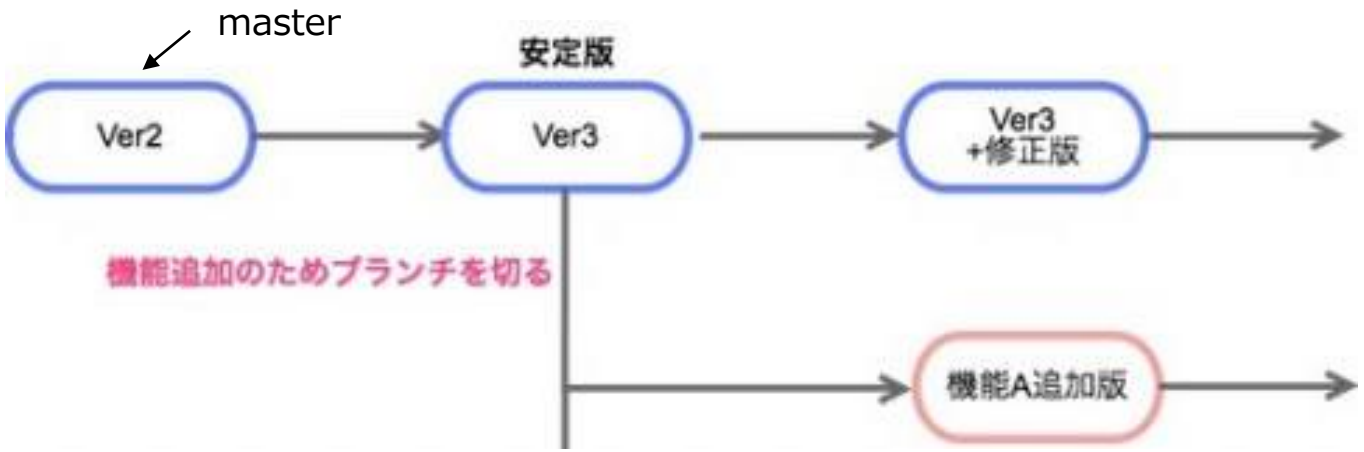


branchの出来ること

- 分岐したブランチは、他のブランチの影響を受けない。
- 同じリポジトリ内で、変更作業の同時進行が可能。
- 分岐したブランチは、別のブランチにマージ可能。

➢ master とは：

安定したバージョンのブランチ=メインのブランチ



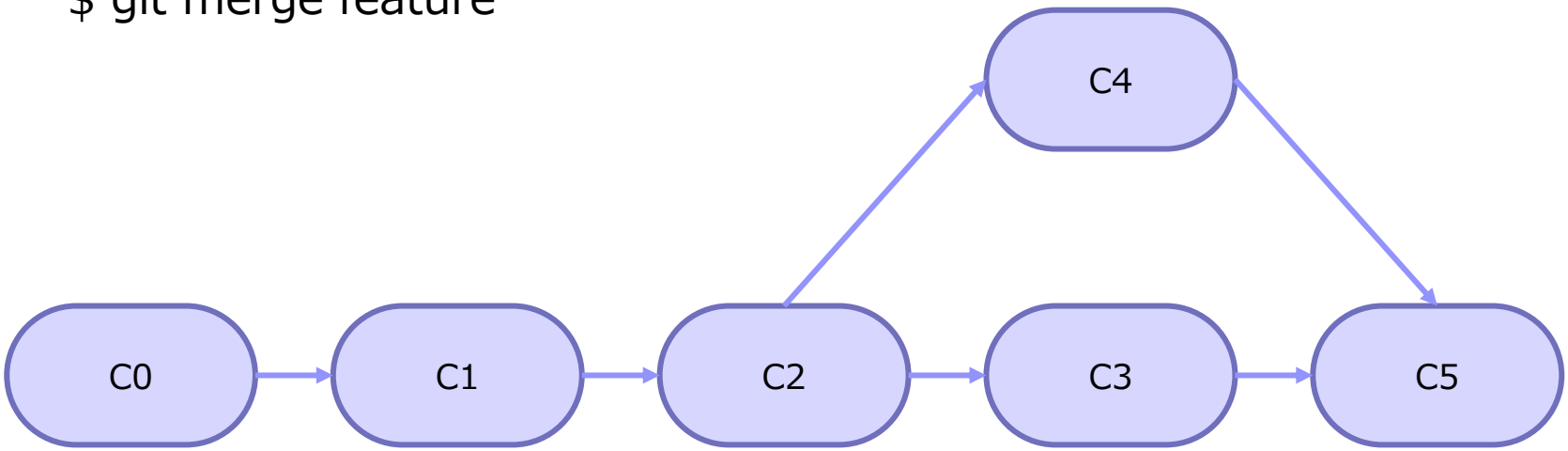
mergeとは

- **2つのブランチを結合するコマンド**

  mergeコマンドは、結合先のブランチで実行すること。

コマンド例：　$git merge [結合するブランチ名]

```
//masterブランチにいる状態でmergeする
$ git merge feature
```



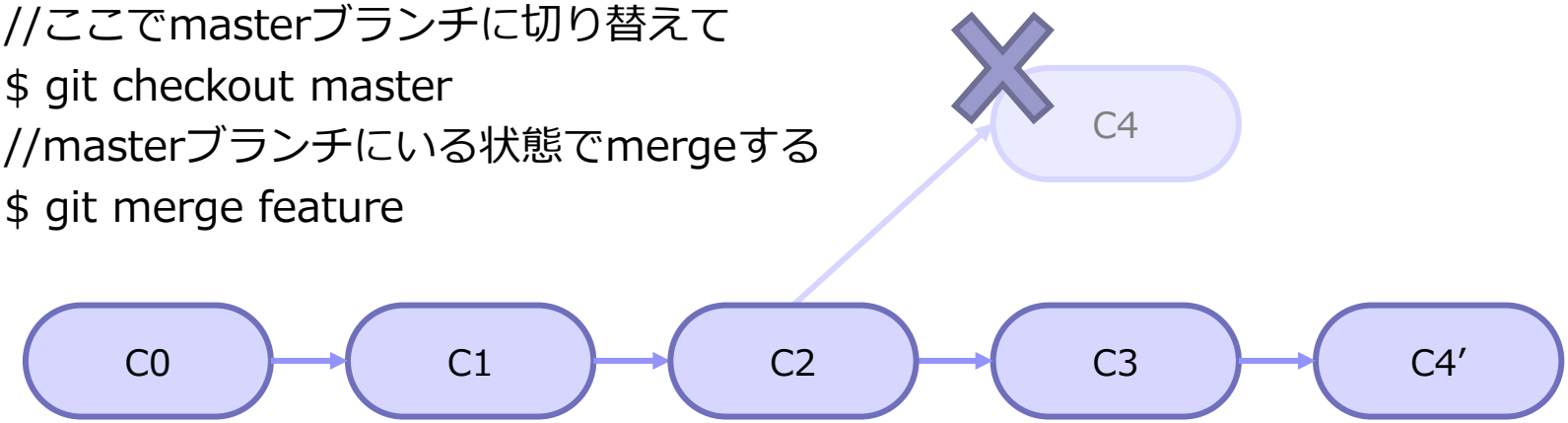
rebaseとは

- **2つのブランチを結合するコマンド**

  リベースされたブランチは消える。

コマンド例：　$git rebase [ブランチ名]

```
//まずはfeatureブランチに移動する
$ git checkout feature
//featureブランチにいる状態でrebaseする
$ git rebase master
//ここでmasterブランチに切り替えて
$ git checkout master
//masterブランチにいる状態でmergeする
$ git merge feature
```



mergeとrebaseの比較

- **merge**
  - メリット
    コンフリクトが1回しか発生しない。
  - デメリット
    マージコミットがたくさんあると、履歴が複雑化する。
- **rebase**
  - メリット
    履歴を綺麗（一直線）に保つことができる。
  - デメリット
    コミットそれぞれにコンフリクトの解消が必要。
- **使い分けのコツ**

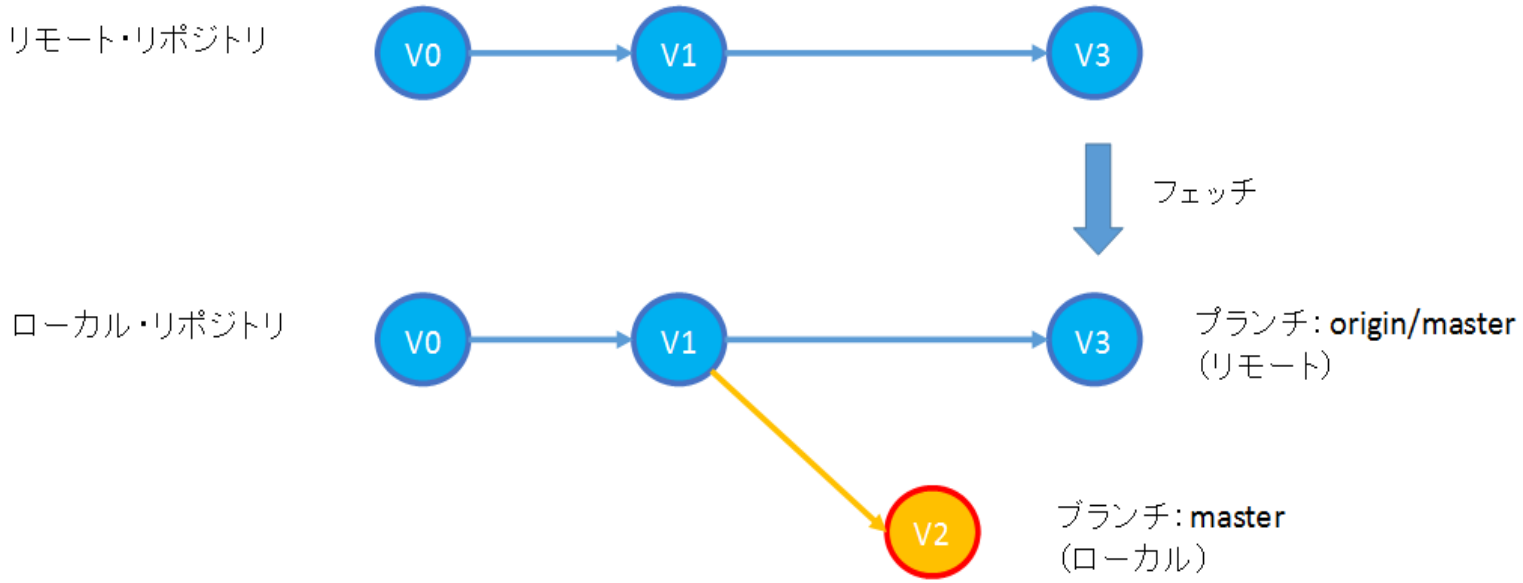
  プッシュしていないローカルの変更　→　rebase
  プッシュした後　→　merge
  コンフリクトしそう　→　merge

（注意）rebaseは必ず自分だけが使うブランチのみで使用すること

fetch とは

- **ローカルリポジトリで作成したブランチに影響を与えずリモートリポジトリで更新された最新の情報にリモートブランチの内容を更新するコマンド**

コマンド例：　$git fetch [リポジトリ名]

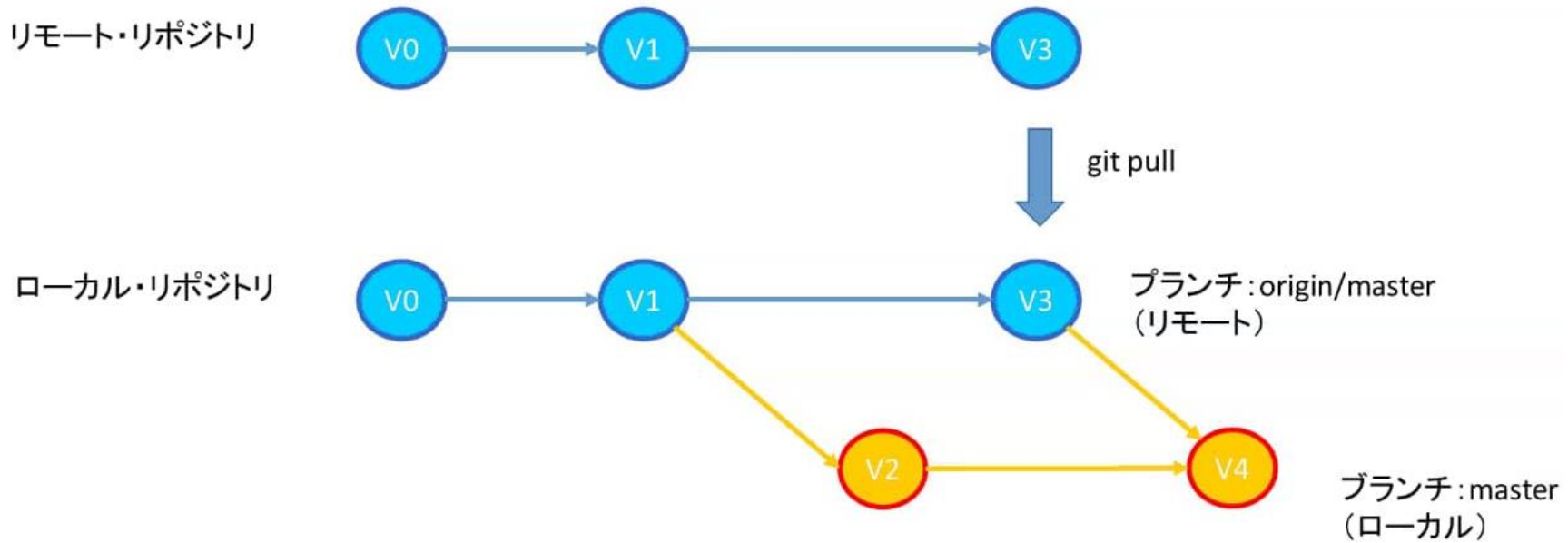


pull とは

- **リモートリポジトリのデータを取得、ワークツリーに統合するコマンド**

  pull＝fetch＋merge

コマンド例：　$git pull　[他のリポジトリ名]　[ブランチ名]



remove(rm) とは

- **ワークツリー、インデックスからファイルを削除するコマンド**

コマンド例：　$git rm [ファイル名]

ワークツリー、インデックスからファイルを削除する。
コミットした際に削除したことをリポジトリに記録する。

ブランチモデルについて

ブランチモデルとは

- Gitを活用して開発を行う際のブランチ運用ルール。
  ブランチモデルの中でも歴史の長いGit-flowの
  開発手順を紹介。

- Git-flow

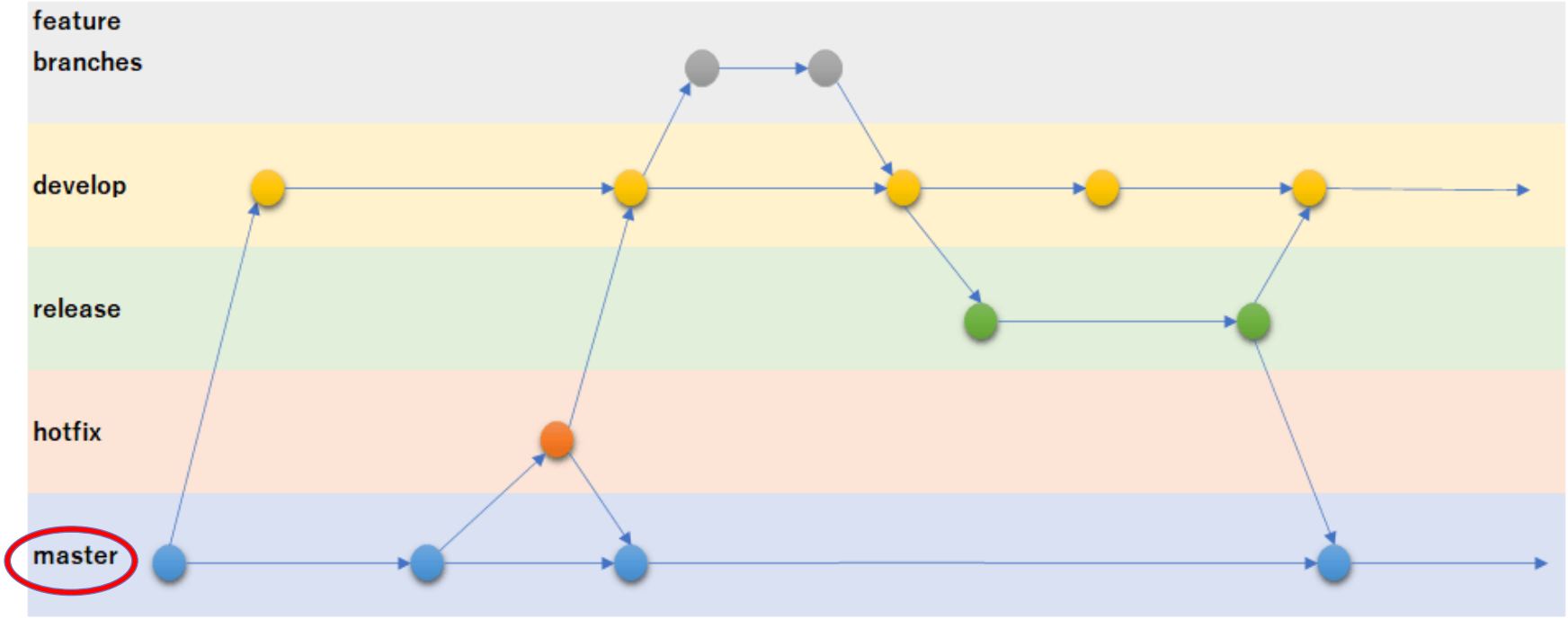
  ブランチの作成やマージに決まりを設け不用意なマージによる
  問題を避ける。

  master, develop, feature, release, hotfix,の５つを作業ごとに
  使い分けて運用。

Master branchとは

- **製品として出荷可能な状態であり、アプリケーションが安定して動くもののブランチ**

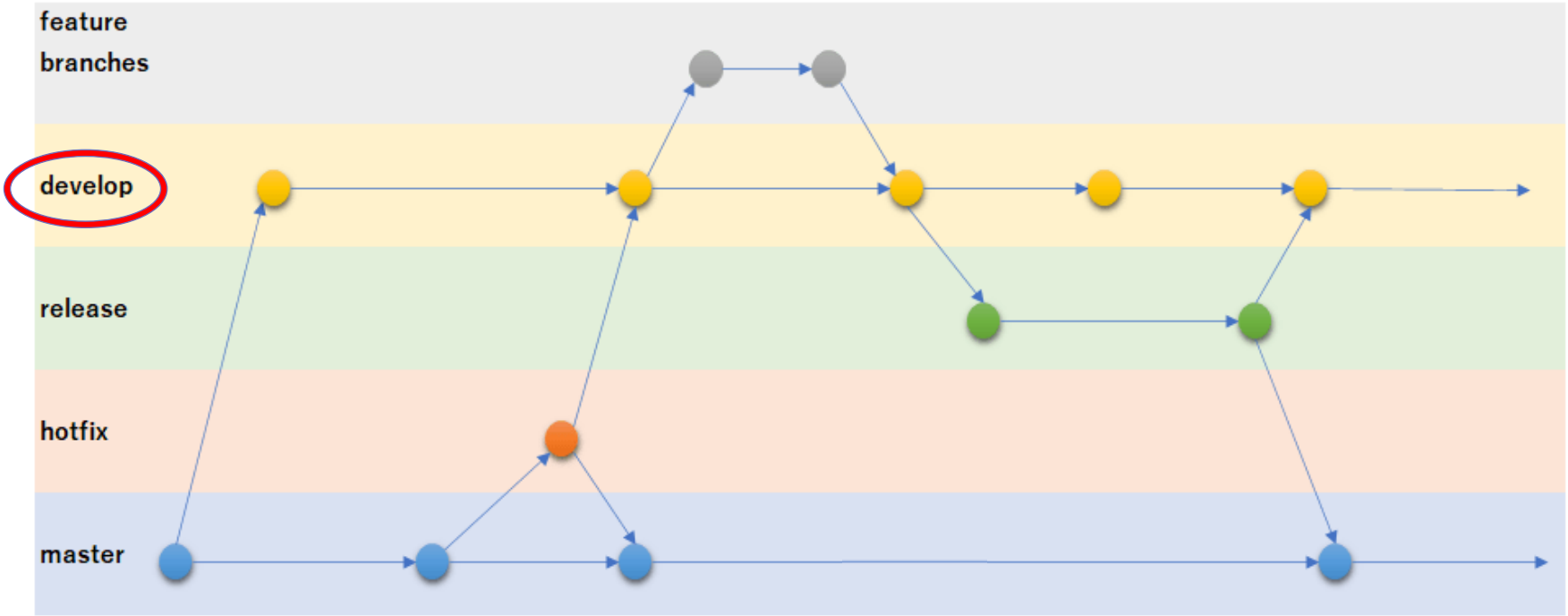
開発作業は行わない。
Master branchに直接コミットしない。（マージする）



develop branchとは

- **実際に開発作業、バグ修正を行うブランチ**

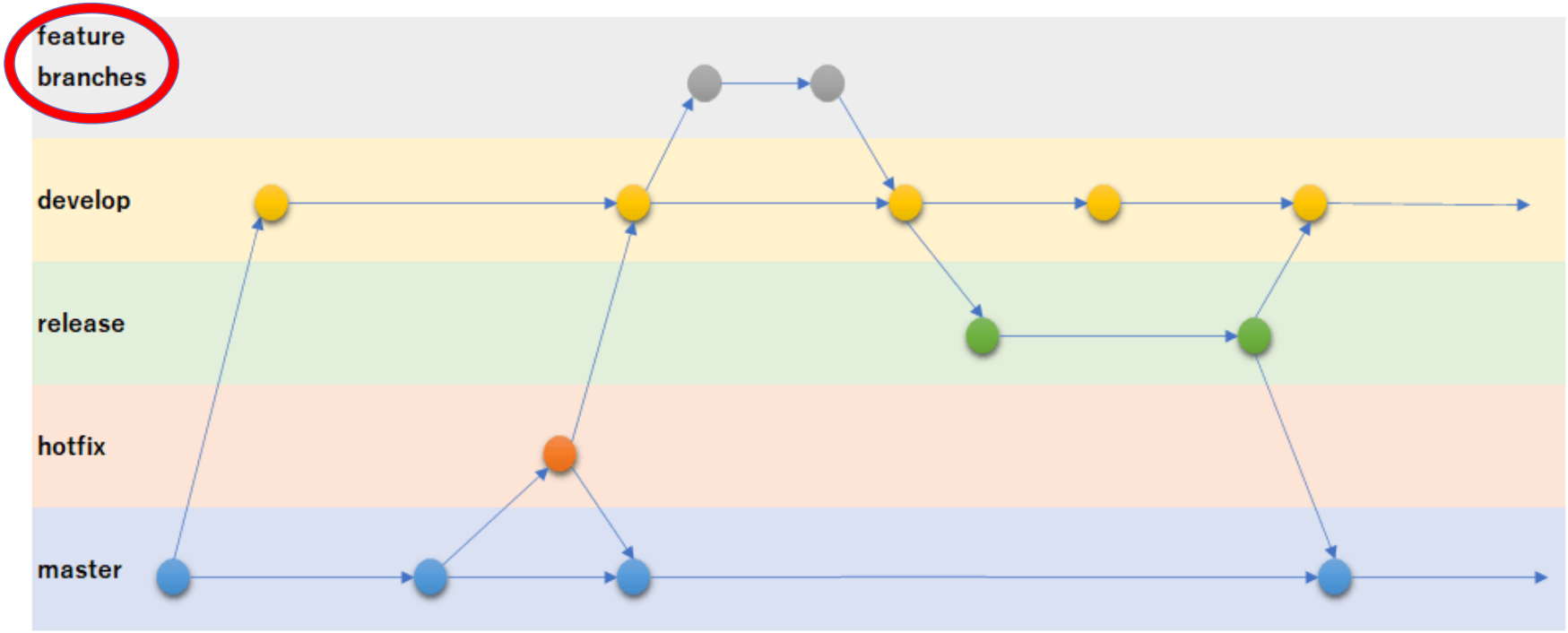
  基本的に新しい開発用のコードが置かれるため最も頻繁に更新が行われる。



feature branchとは

●新しい機能を開発を行うブランチ

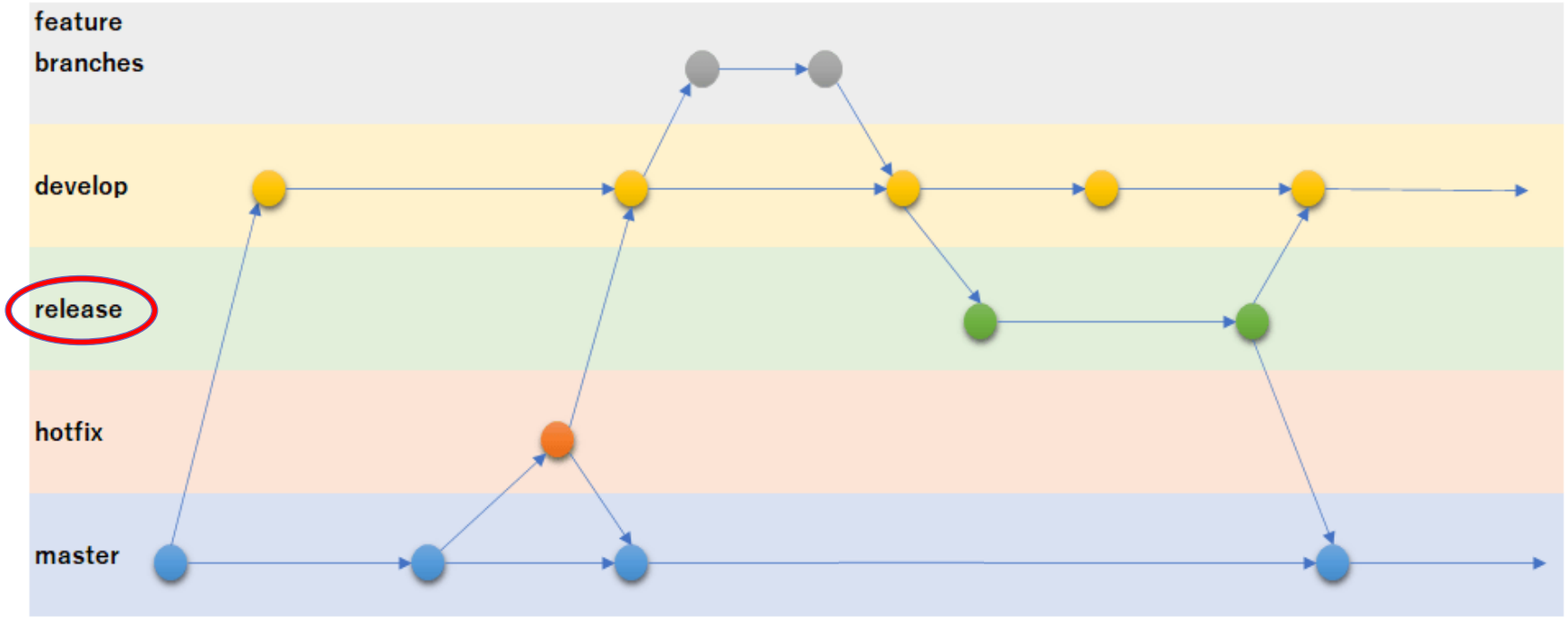
一つのタスクごとにdevelopブランチから作成され、
開発が終われば、developブランチにマージする。



release branchとは

- developブランチを基盤に作成。リリース直前にバグ修正などの微調整（QA）を行うブランチ

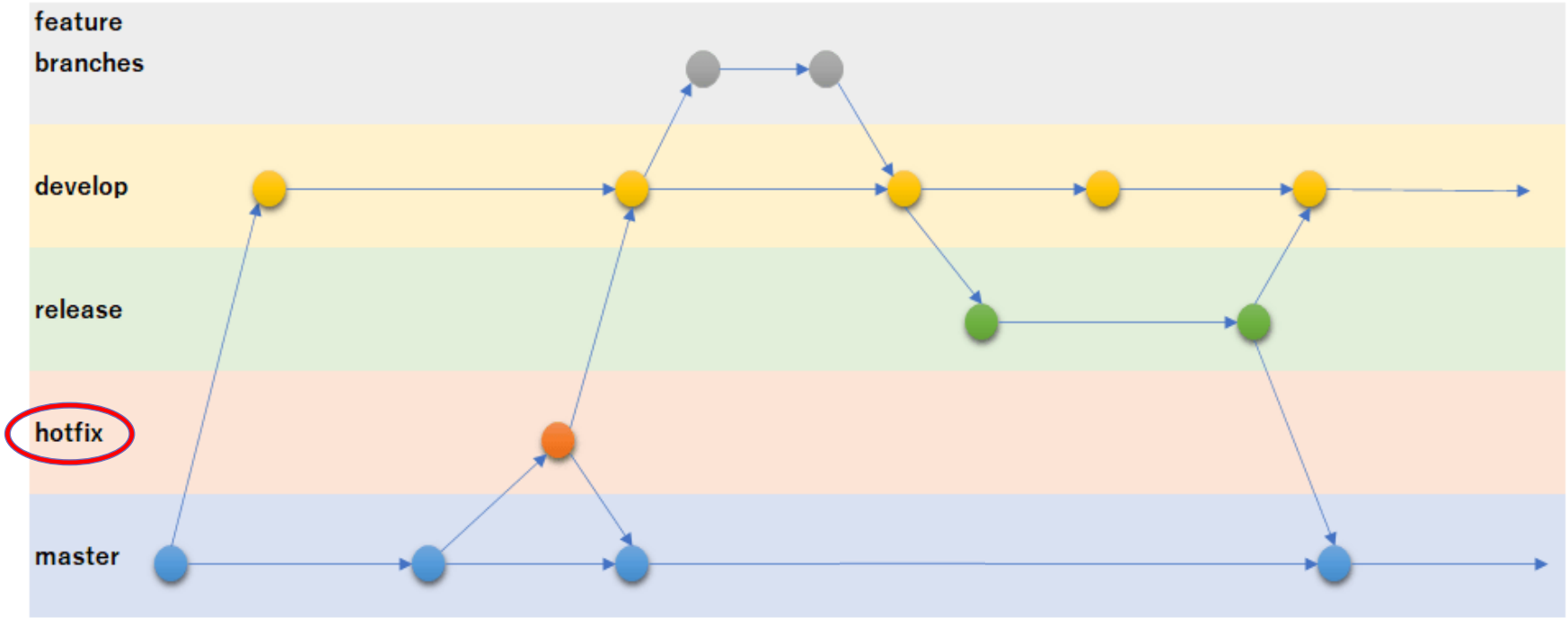
完了後、develop及びmasterにマージする。



hotfix branchとは

- **リリースされたバージョンで発生したバグを速やかに修正するブランチ**

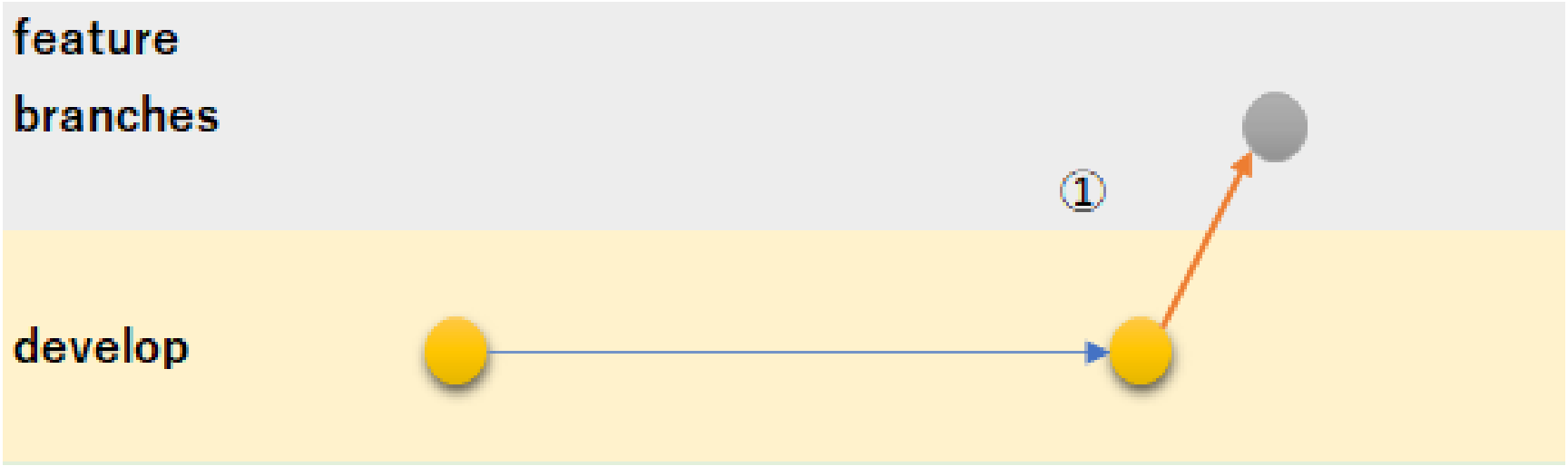
  修正後すぐmaster、developブランチにマージする。



Git flowを用いた開発手順

●開発手順①

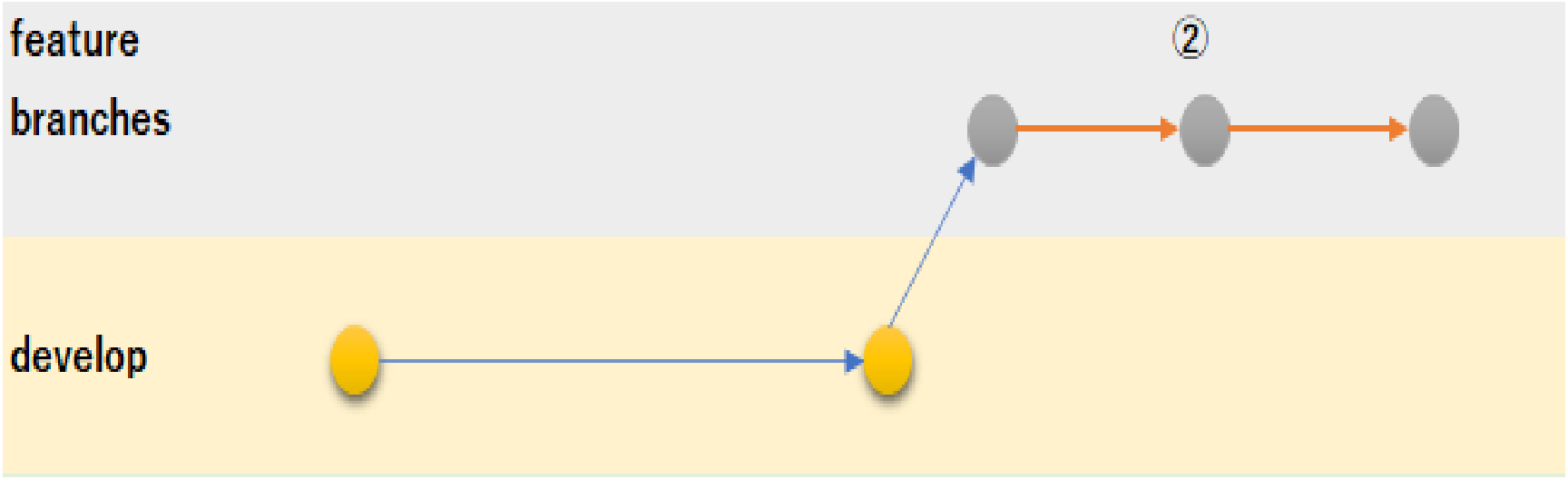
developブランチからfeatureブランチを作成する。



Git flowを用いた開発手順

●開発手順②

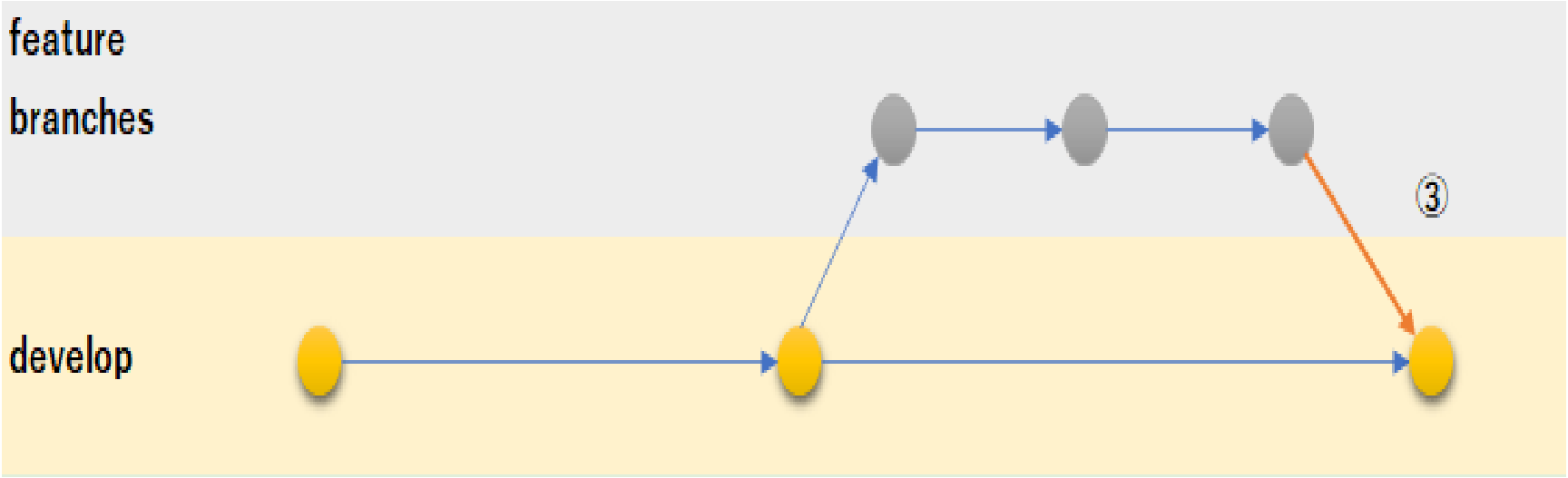
featureブランチで開発、バグ修正を行う。



Git flowを用いた開発手順

●開発手順③

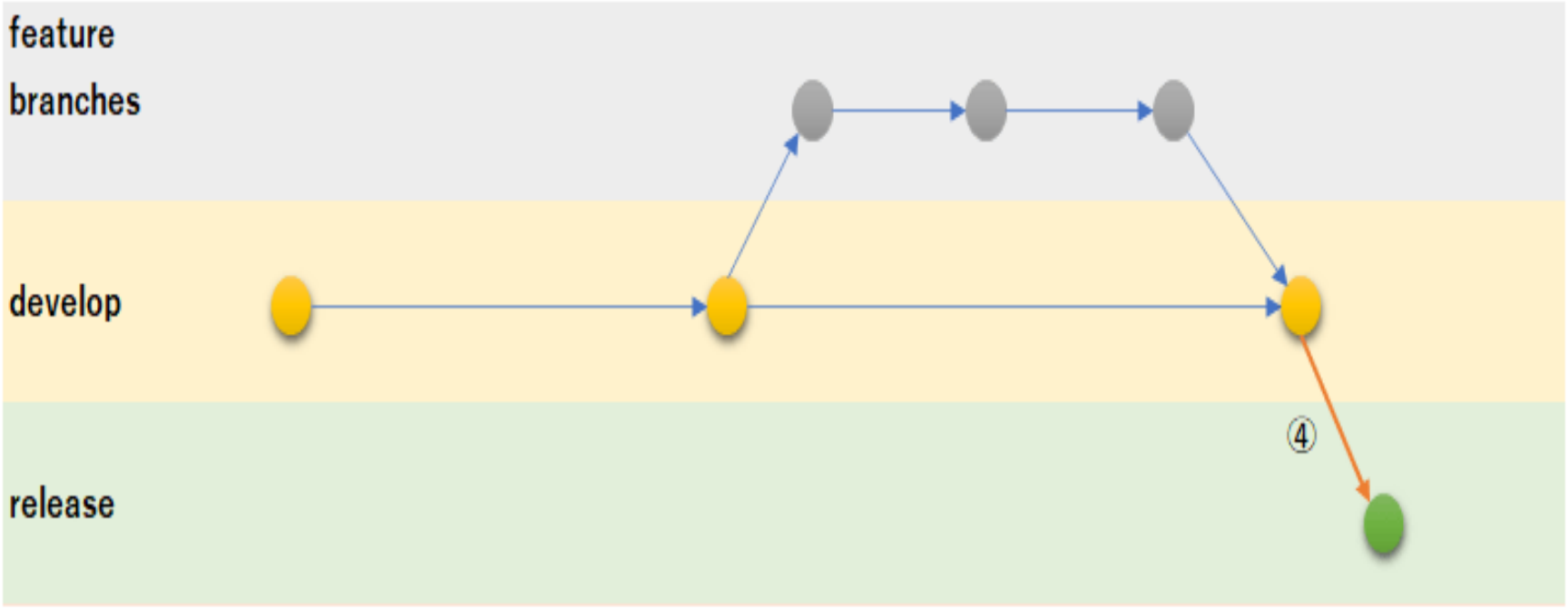
修正完了後、developブランチにマージする。



Git flowを用いた開発手順

●開発手順④

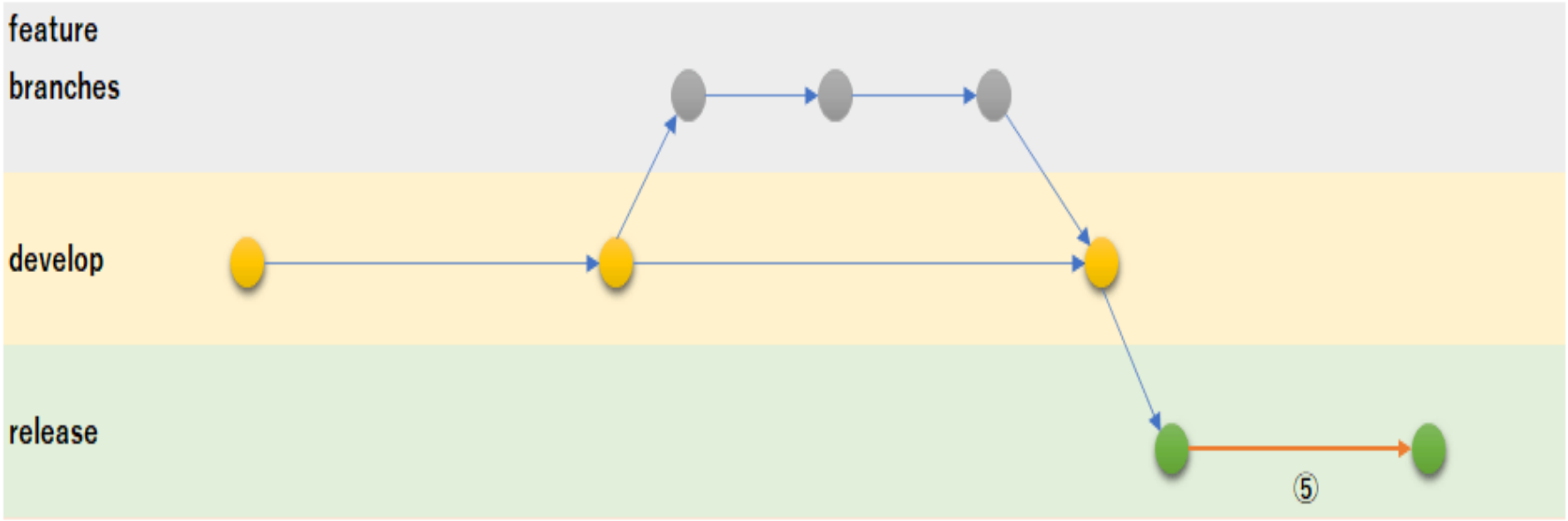
リリース作業を行うためdevelopブランチから
releaseブランチを作成する。



Git flowを用いた開発手順

●開発手順⑤

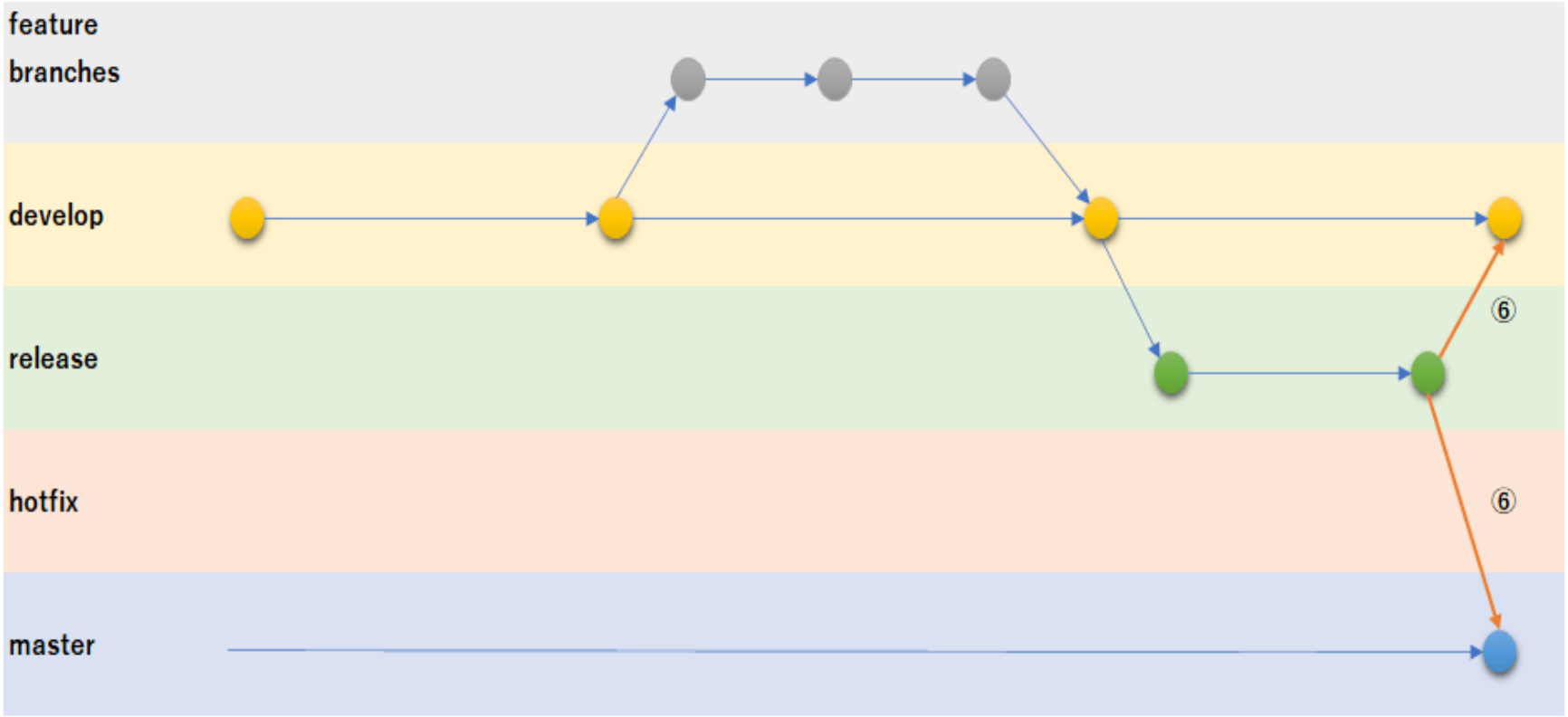
バージョン管理、ドキュメント作成などの変更を行う。



Git flowを用いた開発手順

●開発手順⑥

変更後、developブランチとmasterブランチにマージする。



参考文献・引用文献

- **「Git - Book」**
  https://git-scm.com/book/ja/v2, (参照 2021-07-08)
- **「主要なバージョン管理システム（Git、Mercurial、Subversion）それぞれの特徴を解説」**
  https://tracpath.com/works/development/git-mercurial-subversion, (参照 2021-07-08)
- **バージョン管理システム入門（初心者向け）「Git-flow ～Gitのブランチモデルを知る～」**
  https://tracpath.com/bootcamp/learning_git_git_flow.html, (参照：2021-07-09)
- **「Gitの実践的運用について～git-flow/GitHub flow～」**
  https://tracpath.com/works/development/git_flow_and_github-flow/, (参照：2021-07-09)
- **Git-flowをざっと整理してみた**
  https://dev.classmethod.jp/articles/introduce-git-flow/,（参照：2021-06-29）
- **サル先生のGit入門**
  https://backlog.com/ja/git-tutorial/,（参照：2023-11-09）