

7.6 バージョン管理

- Git
- Git の利用
- Markdown



1

Git

2

Git の利用

3

Markdown

バージョン管理ツール

- **バージョン管理**[Version Control]ツールは、ディレクトリ（フォルダ）やファイルの変更履歴を完全に保存・追跡するバージョン管理機能を提供し、ソフトウェア開発者にとって必須のツールであり、ソフトウェア会社にとってはインフラとなるものとでも言えます。

バージョン管理ツールの機能

- バージョン管理の最も重要な機能は、**ファイルの変更を追跡**することです。
- バージョン管理ツールは、いつ、誰が、どのファイルのどこを変更したかを忠実に記録します。ファイルが変更されるたびに、そのファイルのバージョン番号が増加します。
- もう一つの重要な機能は、**並行開発**あります。
- ソフトウェア開発は、多くの場合、複数人の共同作業で行われます。バージョン管理は、異なる開発者間のバージョンの同期と開発コミュニケーションの問題を効果的に解決し、共同開発の効率を高めることができます。並行開発でよくあるバグ修正の問題も、バージョン管理ツールでブランチマージを行うことで解決することができます。

バージョン管理ツールの役割

- 共同開発: チームで同じプロジェクトに取り組む。
- バージョニング: プロジェクトのバージョンを継続的に増やしていく形で、段階的にプロジェクトを完成させる。
- データのバックアップ: 開発されたすべてのバージョンの履歴を保存している。
- 権限管理: チームの開発者に異なる権限を割り当てる。
- ブランチマネジメント: 開発チームが複数のラインで同時にタスクを進めることで、さらなる効率化を図ることができる。

Git とは

- Git はオープンソースの分散型バージョン管理システムで、規模の大小にかかわらず、あらゆるプロジェクトで俊敏かつ効率的に作業を行えます。
- Git はもともと、Linux カーネル開発を管理するためのオープンソースのバージョン管理ソフトウェアとして、Linus Torvalds 氏によって開発されました。
- CVS や Subversion などの従来の集中型なバージョン管理ツールとは異なり、Git は**分散型リポジトリ**を採用しており、サーバー側にはソフトウェアが要りません。

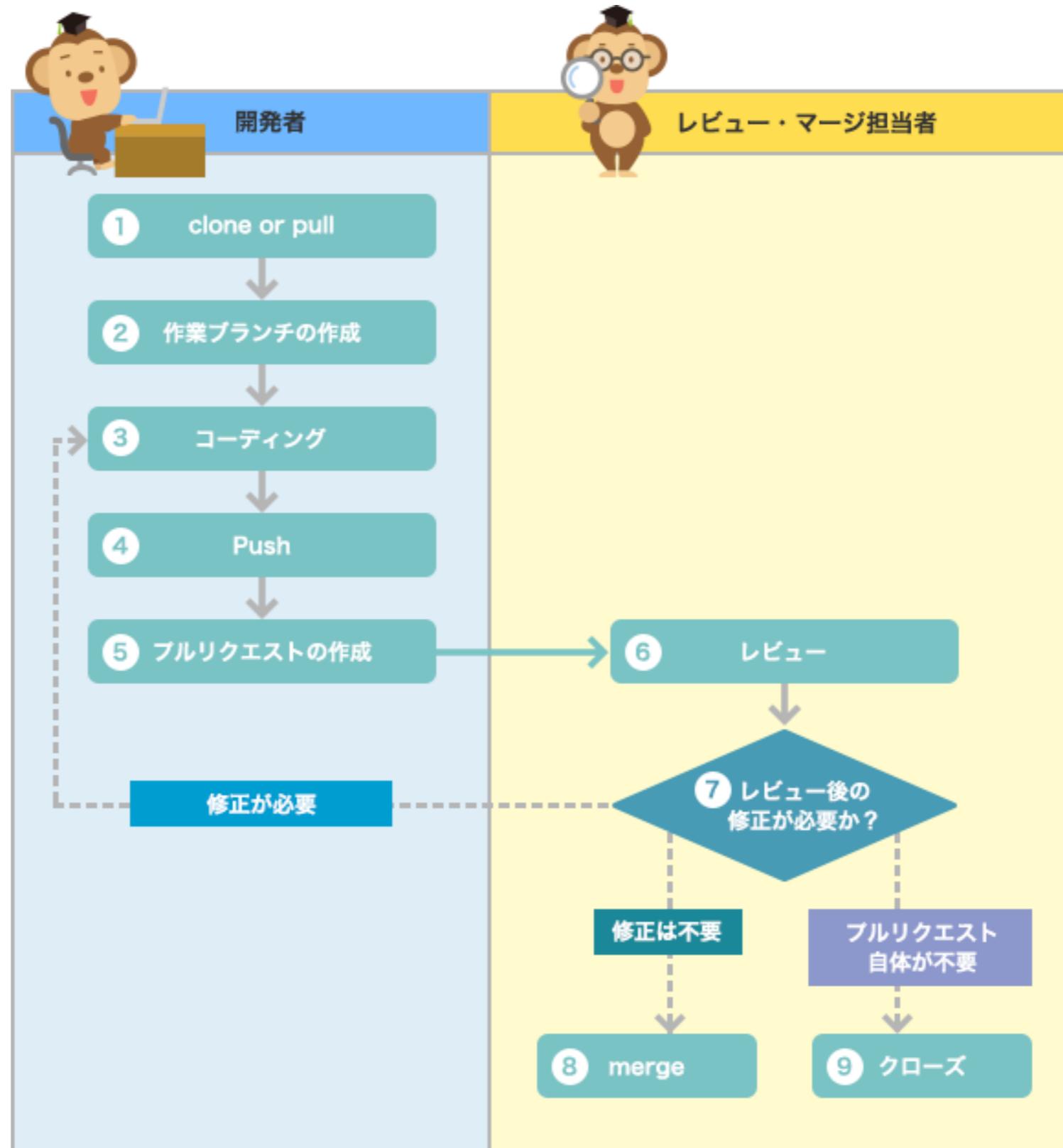


 <https://git-scm.com>

Git の作業フロー

- Git のファイルを作業ディレクトリにクローンする。
- 他の人がファイルを変更した場合、どのファイルがどう変更するかは確認できる。
- クローンしたファイルに**追加・変更**する。
- コミットする前に**変更点を確認**する。
- 変更を**コミット**する。
- 変更完了後、もしエラーが見つかったら、コミットを撤回し、変更し直してコミットすることができる。

Gitの作業プロセス



参考: <https://backlog.com/ja/git-tutorial/>

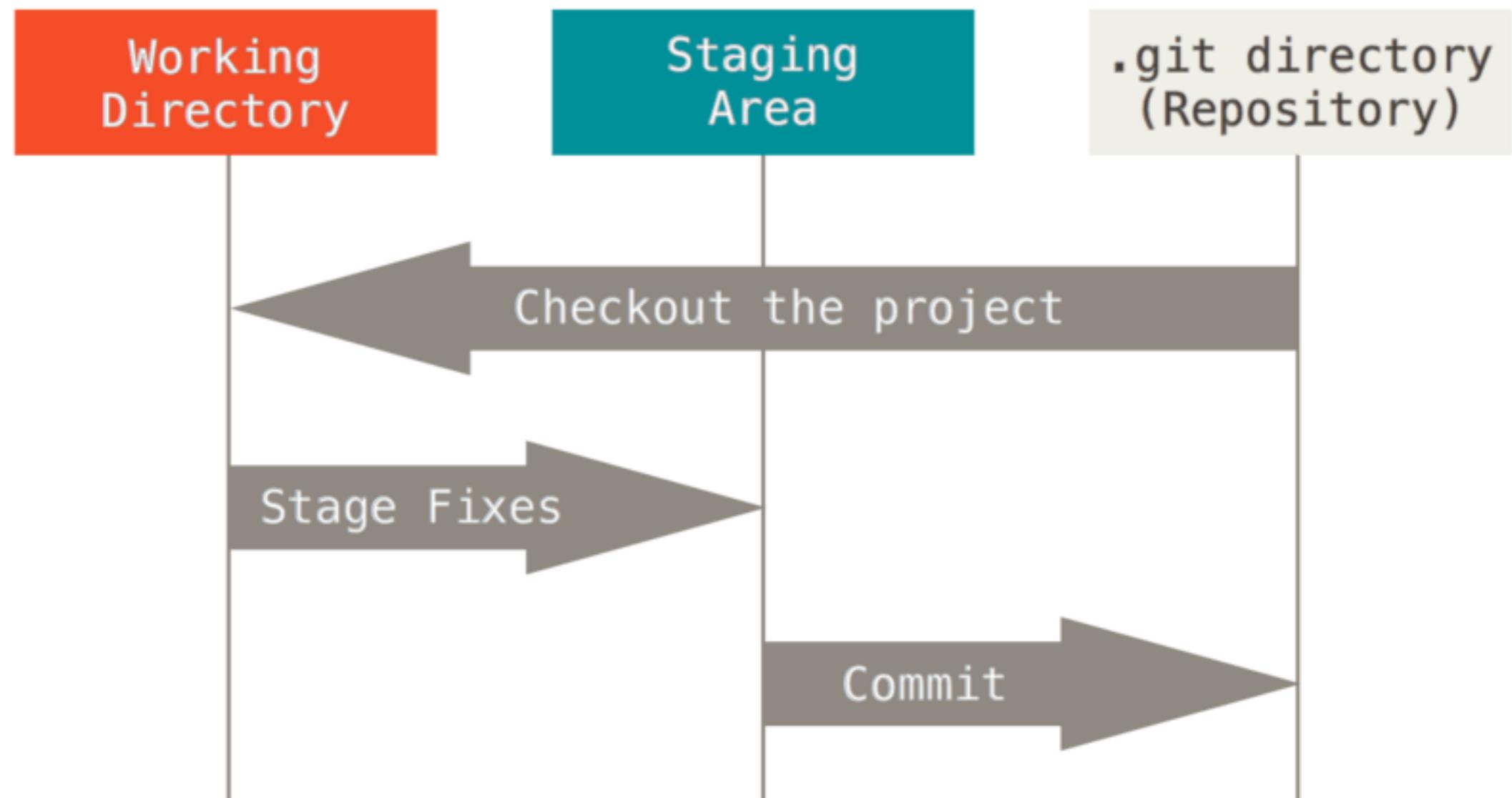
ファイルの 3 つの状態

- Git にはファイルは 3 つの状態があり、**コミット済**[Committed]、**変更済**[Modified]、**ステージ済**[Staged] のどちらかになっています。
- **コミット済**のファイルは、ローカルデータベースに安全に保存されています。
- **変更済**は、ファイルが変更されたが、まだデータベースに保存されていないことを意味します。
- **ステージ済**とは、変更されたファイルの最新バージョンが、次のコミットに含まれるようにマークされています。
- これらと関係するのは Git の 3 つの作業領域：Git リポジトリ、作業ディレクトリ、ステージングエリア。

3つの作業領域

- Git の**リポジトリ**[Repository]とは、Git がプロジェクトのメタデータとデータベースを保持する場所です。これは Git の最も重要な部分で、他のコンピュータからリポジトリをクローンするときにデータはここからコピーされます。
- **作業ディレクトリ**[Working Directory]は、あるバージョンのプロジェクトの内容を個別に抽出したものです。ここにあるファイルは、リポジトリから抽出され、我々は使用または変更できるように、ディスク上に配置ています。
- **ステージングエリア**[Staging Area]とは、次回にコミットされるファイルのリストに関する情報を保持するファイルです。通常は Git リポジトリに置かれます。「**インデックス**[index]」と呼ばれることもあります。

Git のワークフロー: 3 つの状態

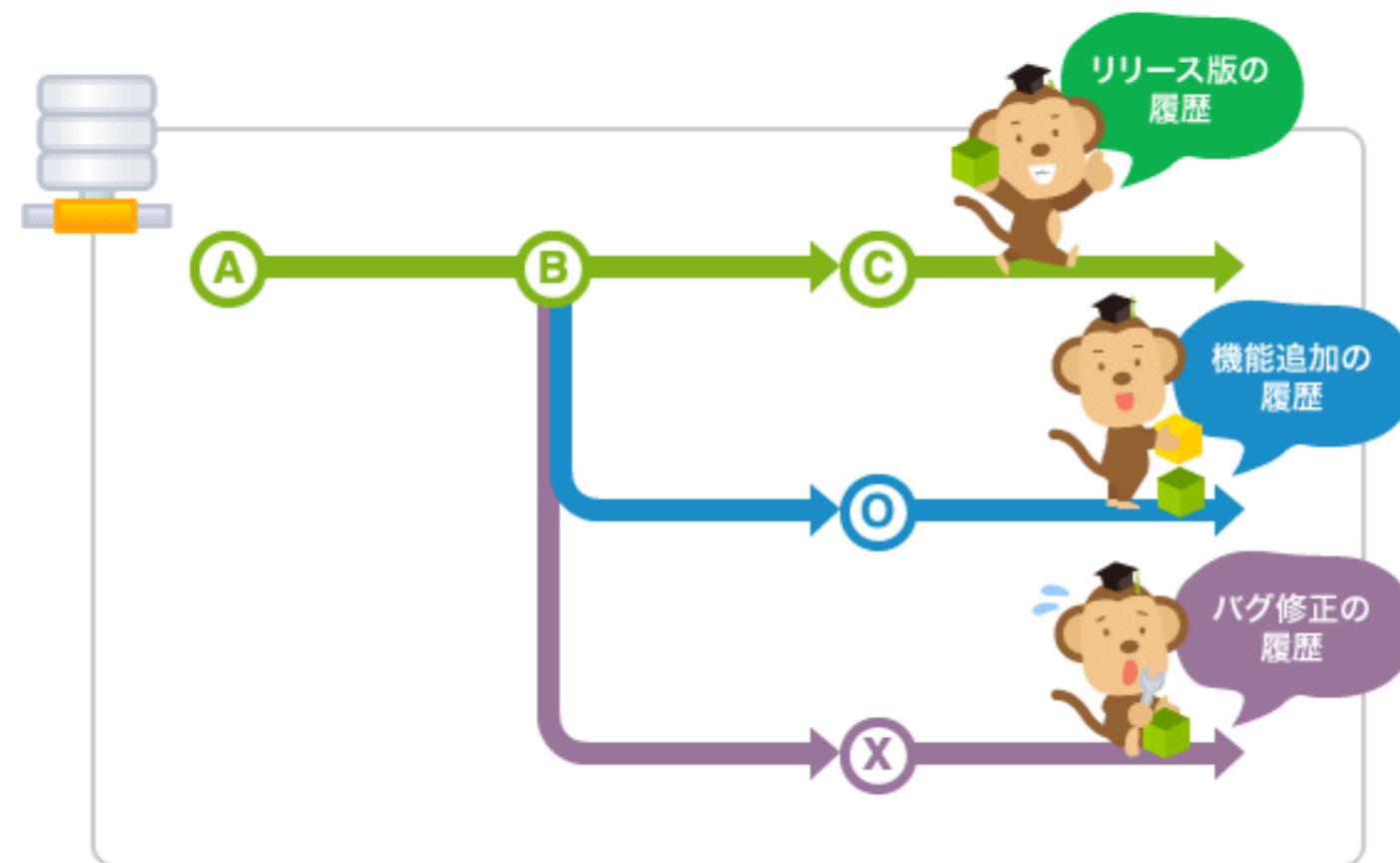


Git ブランチ

- ソフトウェアを開発する場合、同じソフトウェアの**機能開発は複数の人が同時にを行い、メンテナンスが必要なバージョンも複数存在**する場合があります。
- 幸いなことに、Git には**ブランチ[Branch]**機能があって、複数の機能を同時に開発しバージョン管理することをサポートしています。

ブランチの目的

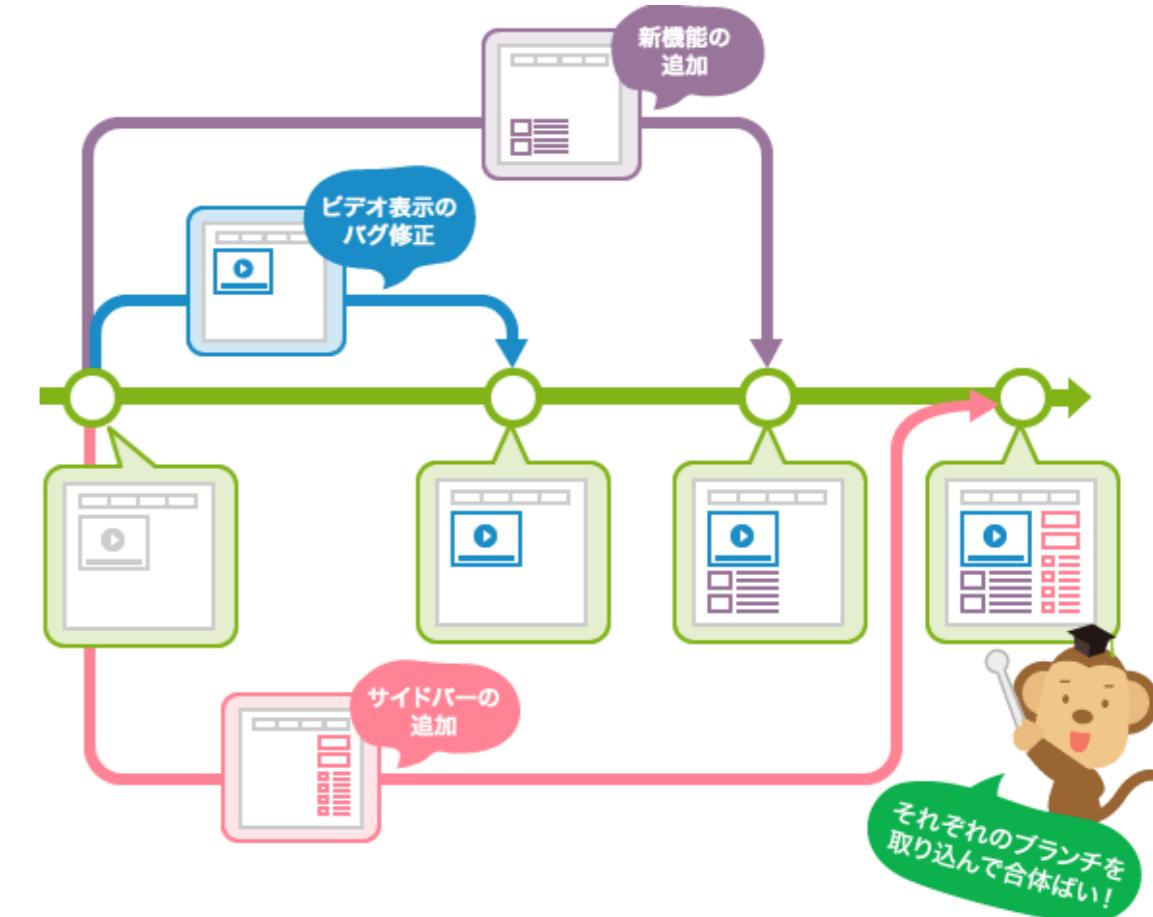
- ブランチ機能の目的は、変更の全体的な流れを分岐させること（**フォーク[Fork]**）です。分岐されたブランチは他のブランチの影響を受けないので、同じプロジェクトの複数のバージョン変更を行うことができます。



参考: <https://backlog.com/ja/git-tutorial/>

マージ

- フォークされたブランチを合併（マージ [Merge]）することができます。
- 他の開発者の影響を受けないように、`master` ブランチ上に自分専用のブランチを作成することができます。作業が終了したら、自分のブランチの変更をこの `master` ブランチにマージします。各コミットの履歴が保存されるため、問題が発生したときに原因となったコミットを探し出し、修正することが非常に容易になります。



参考: <https://backlog.com/ja/git-tutorial/>



Question and answer



1

Git

2

Git の利用

3

Markdown

Git のインストール

- Windows:  <https://gitforwindows.org>
- Mac (Terminal で実行) :
 - インストールパッケージの管理ツールをインストール:
`/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`
 - Git をインストール:
`brew install git`
- インストールを確認します。cmd または Terminal で:
`git --version`

リポジトリのクローン

- リポジトリを**クローン**するコマンドは以下の通り:

```
git clone [url]
```

- 例えば、「grit」というリポジトリをクローンする場合は、以下のコマンドを使用します:

```
git clone git://github.com/schacon/grit.git
```

- このコマンドを実行すると、カレントディレクトリに「grit」というディレクトリが作成されます。その中には、ダウンロードしたすべてのバージョンレコードを保持する「.git」ファイルを含むディレクトリが作成されます。

次へ 

 前へ

- クローンしたプロジェクトディレクトリに独自の名前を定義したい場合は、上記のコマンドの最後に新しい名前を指定します。例えば、クローンしたプロジェクトを「mygrit」と名付けて保存したい場合は、以下のコマンドを実行：

```
git clone git://github.com/schacon/grit.git mygrit
```

- いまのプロジェクトの最新バージョンを、リモートリポジトリからローカルリポジトリに取り込むには、以下のコマンドを実行：

```
git pull
```

ブランチ管理

- ブランチの一覧を表示する基本的なコマンドは:

```
git branch
```

- 手動でブランチを作成するには、「git branch [ブランチ名]」を実行:

```
git branch testing
```

- checkout コマンドを使って、変更したいブランチに切り替えます:

```
git checkout testing
```

次へ 

◀ 前へ

- 「git checkout -b [ブランチ名]」というコマンドで新しいブランチを作成し、すぐにそのブランチに切り替えてその中で操作できるようにすることも可能：

```
git checkout -b newtest
```

- ブランチを削除するには、「git branch -d [ブランチ名]」を実行：

```
git branch -d testing
```

リポジトリの表示と変更

- status コマンドで今の**リポジトリの状態を確認**できます:

```
git status
```

- add コマンドで、ステージングエリアに**ファイルを追加**できます:

```
git add a.txt
```

- 「add .」で**すべてのファイルを追加**することもできます:

```
git add .
```

- ファイルを追加できたら、status コマンドで**ステージングエリアにファイルの変更を確認**できます。

変更のコミットとプッシュ

- 変更を**コミット**するには、commit コマンドを：

```
git commit -m "Commit Message"
```

- 「-m」オプションを省略すると、Git はテキストエディターを開き、そこに複数行のコミットメッセージを書けます。
- コミットする前に、**必ず add** コマンドを実行してください。
- 現在のブランチをリモートリポジトリの対応するブランチに**プッシュ**（更新）するには：

```
git push
```

ブランチマージ

- push コマンドを実行すると、リモートリポジトリには変更がうまく反映されますが、master ブランチには反映されないので、マージを行う必要があります。
- 自分がプロジェクトのオーナーである場合、直接に merge コマンドを使ってローカルでマージし、マージした master ブランチをリモートリポジトリにプッシュすることができます。
- 自分がただのプロジェクトの貢献者で、マスター権限がない場合は、Git の「**pull request**」という機能を使ってプロジェクト管理者にマージを依頼することができます。

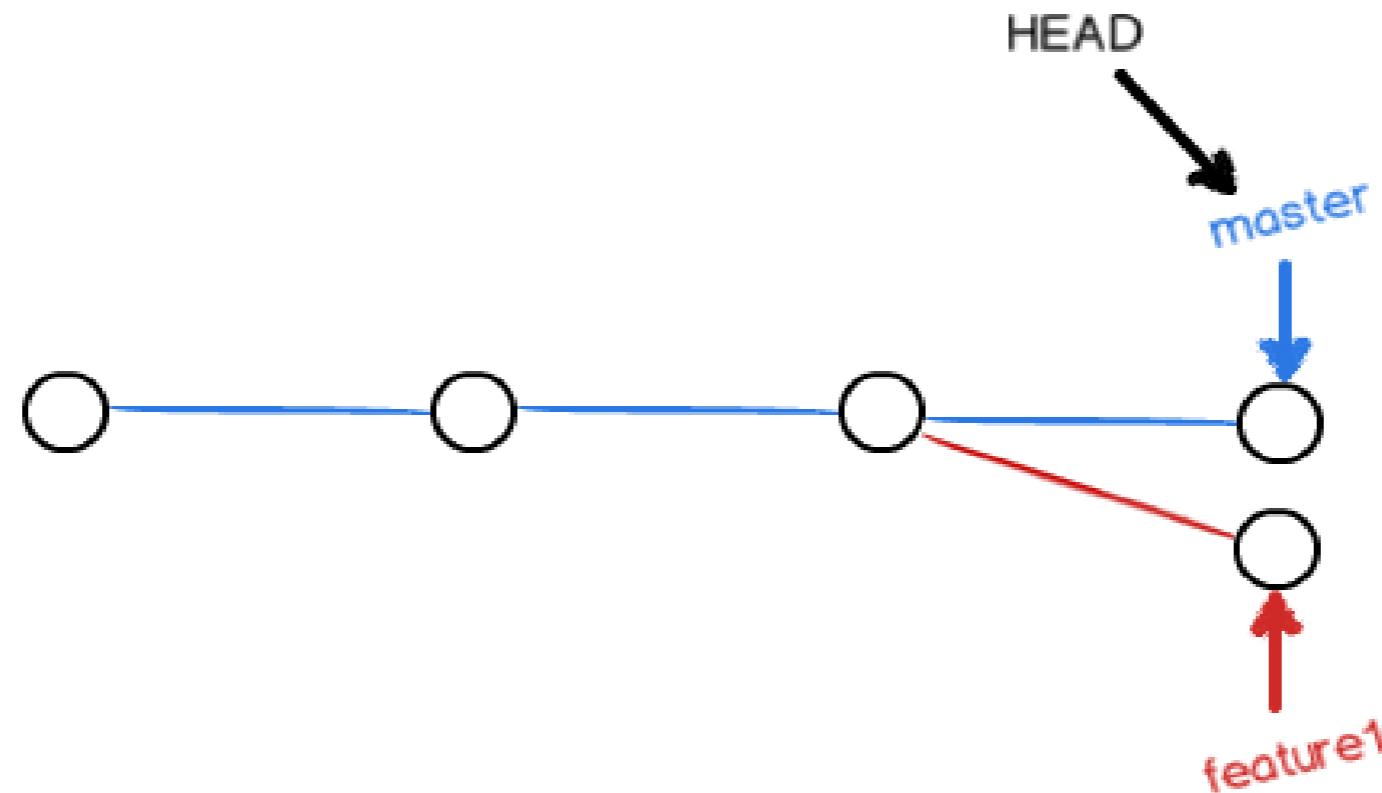
マージのコマンド

- まずブランチをターゲットのブランチに切り替え、次にマージしたいソースのブランチをマージします：

```
git checkout master  
git merge test
```

- この二つのコマンドで、test ブランチを master にマージしました。

Git 衝突



- 上図のように、feature1 を master にマージしたいときに、feature1 の変更している間に他の誰かが master に変更を加えた可能性があります。
- その後で master ブランチにマージしようとすると、ファイルの変更によって**衝突**[Conflict] が発生する可能性があります（たとえば、両方が同じ行のコードを変更した）。

衝突の発生

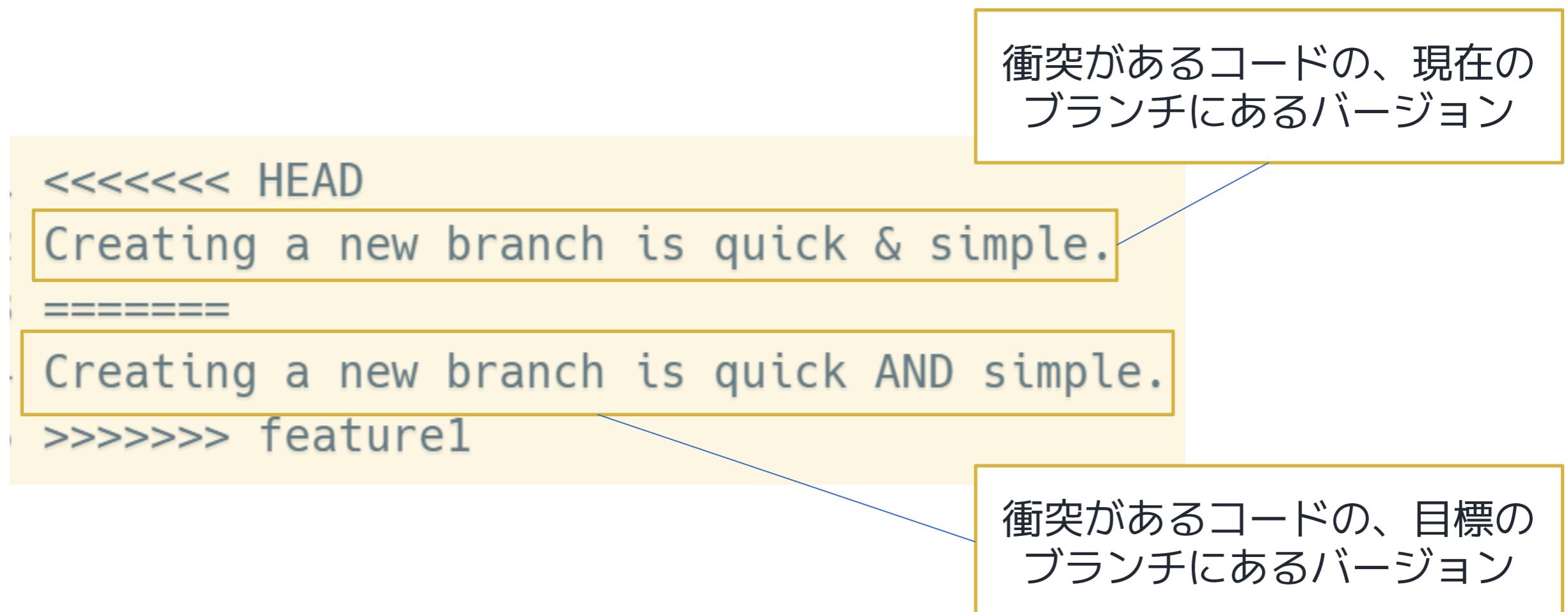
- 衝突の例 (Mac) :

```
$ git merge feature1
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the result.
```

- Git は「readme.txt に衝突があるので、コミットする前に衝突を手動で修正しろ」と指示しました。
- status コマンドで、Git は衝突があるファイルについての詳細を教えてくれます。

衝突の表示

- Git は「<<<<<<」 「=====」 「>>>>>>」のような特殊符号を使って、競合するファイル中の異なるブランチの内容をマークする。これにより、コンフリクトを含むファイルを開いて手動で解決することができます:



衝突の解消

- <<<<<<、=====、>>>>>> がマークした内容を手動で以下のように修正して、保存します：

Creating a new branch is quick AND simple.
- 上記のようにブランチの 1 つのみへの変更を保持して、<<<<<<、=====、>>>>>> の行も完全に削除しなければなりません。
- すべてのファイルの衝突を解決したら、各ファイルで add コマンドを使用してステージングエリアに追加します。衝突するファイルがステージされると、Git はそれらを解決済みとしてマークします。



Question and answer



1

Git

2

Git の利用

3

Markdown

Markdown

- **Markdown** 言語は、2004 年に John Gruber 氏によって作られた、読みやすく、書きやすいプレーンテキスト形式で文書を書くことができる軽量のマークアップ言語です。
- Markdown で書かれた文書は、HTML、Word 文書、画像、PDF、Epub などのさまざまな形式に変換できます。
- Markdown ファイルの拡張子は「.markdown」または「.md」になります。
- 書き方については、こちらのリンクを参考してください：
 <https://github.com/zhongtaoaki/lighthouse-july-java/blob/main/markdown.md>



Question and answer

まとめ

Sum Up

1. Git の基本原理:

- ① ローカルリポジトリとリモートリポジトリの概念。
- ② リポジトリにあるファイルの 3 つの状態。
- ③ ブランチ・マージ。

2. Git のコマンドラインでの基本操作:

- ① リポジトリの基本操作: 状態のチェック、クローン、ステージ、コミット、プッシュ。
- ② 衝突解消の仕方。

3. Markdown の基本的な構文。

THANK YOU!