Git入門

概要

分散型バージョン管理システム

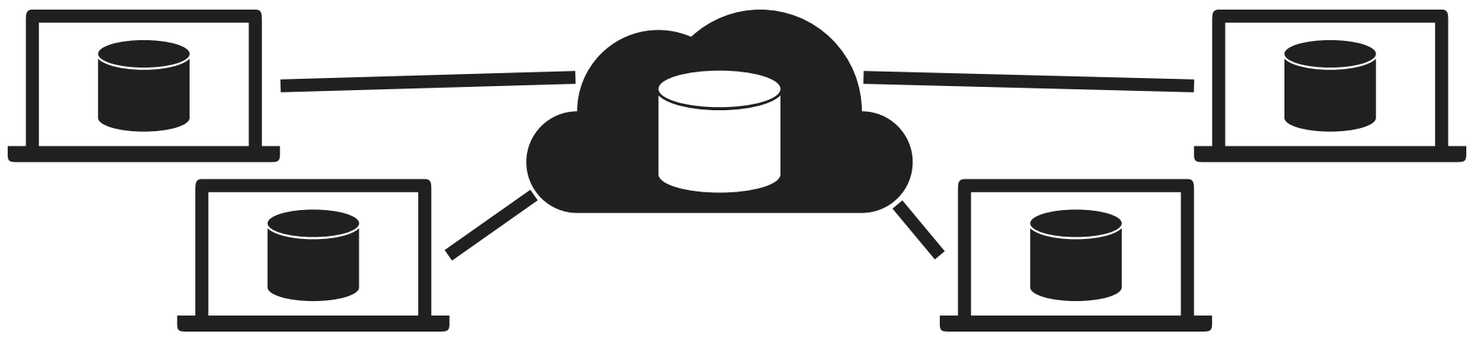
- **ファイルの変更履歴（差分）を管理するシステム**
- **使用例：**チーム開発、履歴管理、コード管理

▶ 分散型

- 複数の開発者が並行して作業を行い、統合が可能

▶ バージョン管理

- 変更履歴の追跡が可能
- 変更適用前など、過去の状態に戻すことが可能



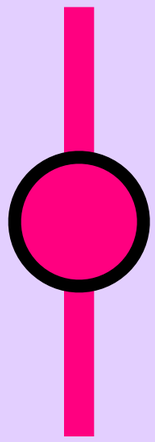
用語

リポジトリの作成

- Repository
- init
- fork
- clone
- shallow clone

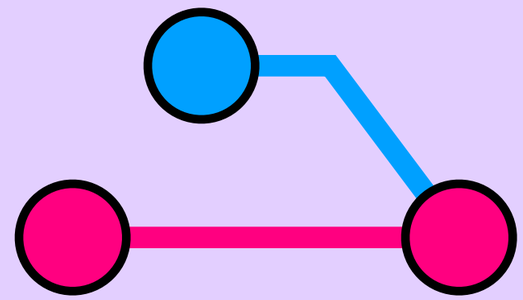
変更内容の登録

- hunk
- index
- staging
- commit



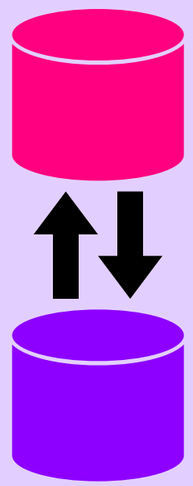
分岐と統合

- branch
- merge
- conflict



リポジトリ間の同期

- fetch
- pull
- push
- pull request



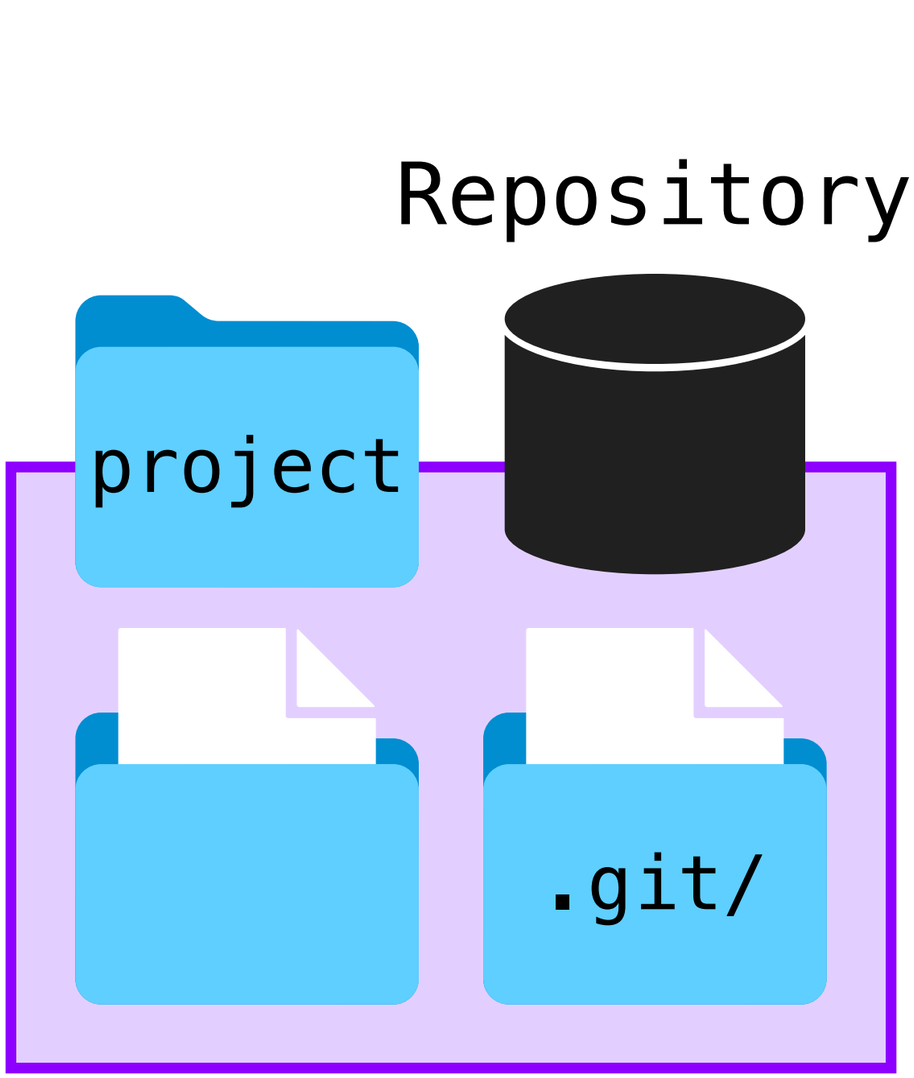
おまけ

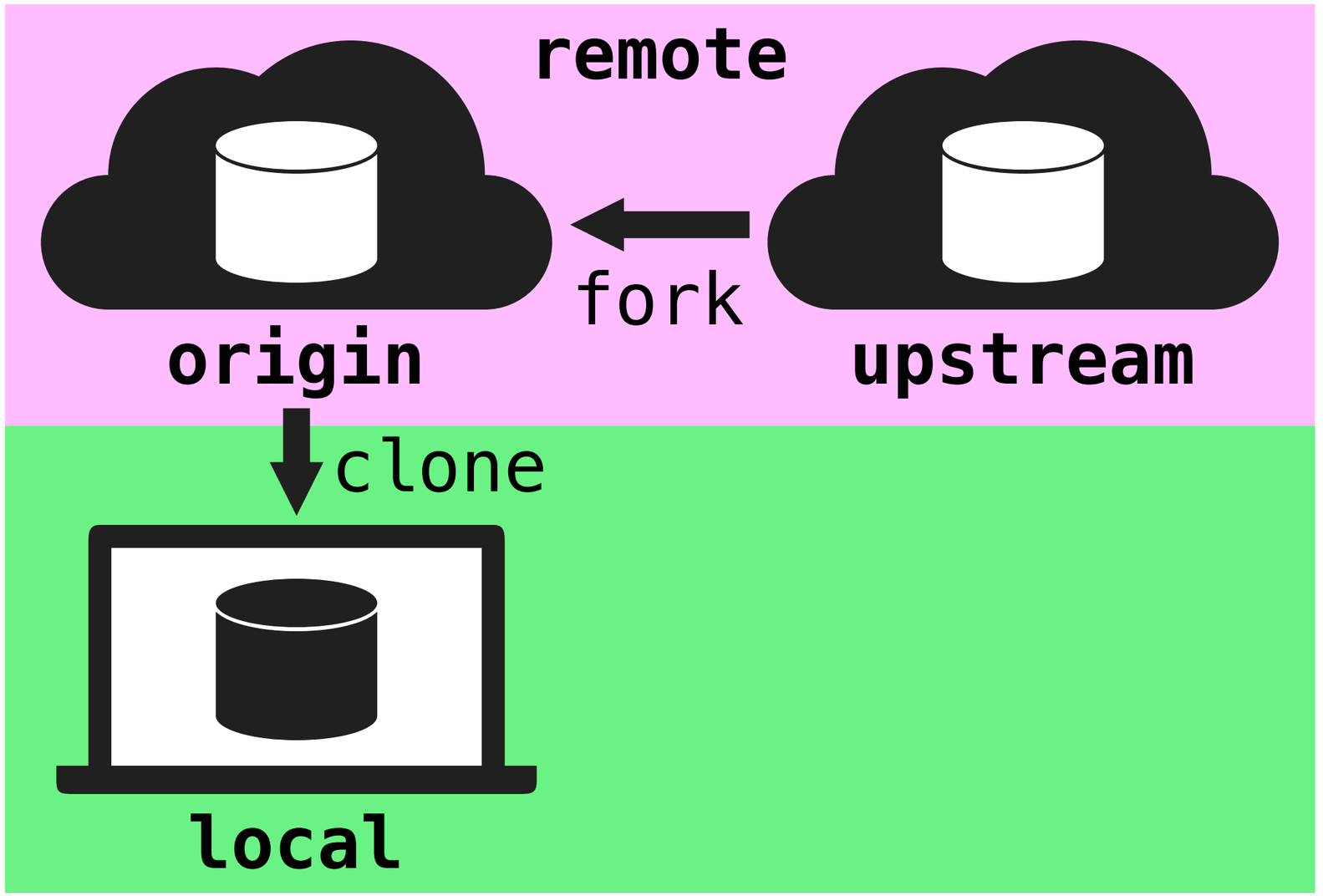
- Tag
- .gitignore

リポジトリの作成

[remote, local] Repository

- **Gitで管理するプロジェクト全体の保存場所**
- 全てのファイル、変更履歴、ブランチ、メタデータ、プロジェクトの設定などを含む





リポジトリの作成

init `git init`

- **ディレクトリをリポジトリとして初期化する操作**
- `.git/`を作成し、Gitの管理情報（履歴や設定）を格納

```
$ mkdir project
$ cd project
$ git init
```

`project/` に `.git/` が作られリポジトリとして初期化される

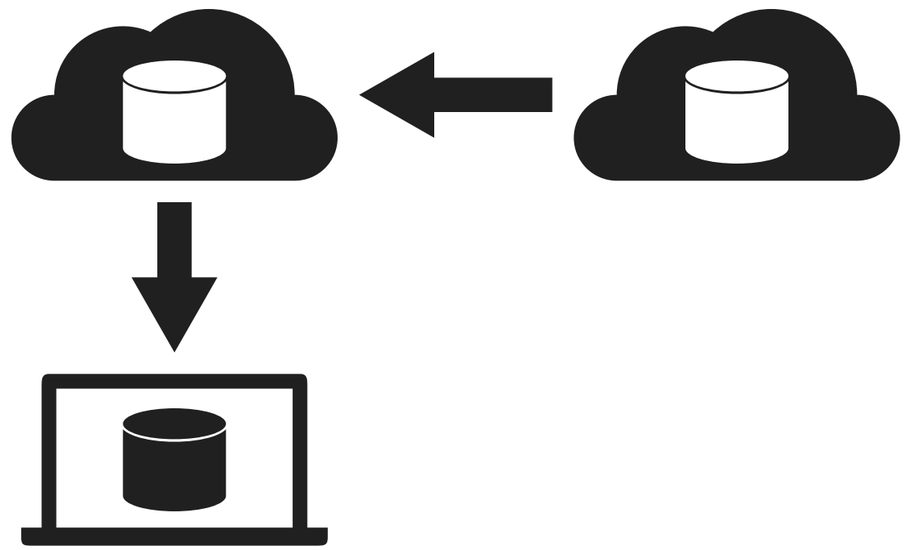
- **リモートの追加：**リモートリポジトリを（GitHubやGitLab等で）作成→リモートの登録→ブランチの登録

```
$ git remote add origin <remote_rep_URL>
$ git push (-u|--set-upstream) origin main
```

リポジトリの作成

fork

- 他人のリモートリポジトリを
  自分のリモートにコピーする操作



clone

```
git clone
```

- リモートリポジトリをローカルにコピーする操作

shallow clone

```
git clone --depth 1 <rep>
```

- 最新の履歴のみをコピー（クローン）する操作

利点：リポジトリのサイズ削減、クローン時間の短縮

欠点：古いコミット履歴は参照不可となる

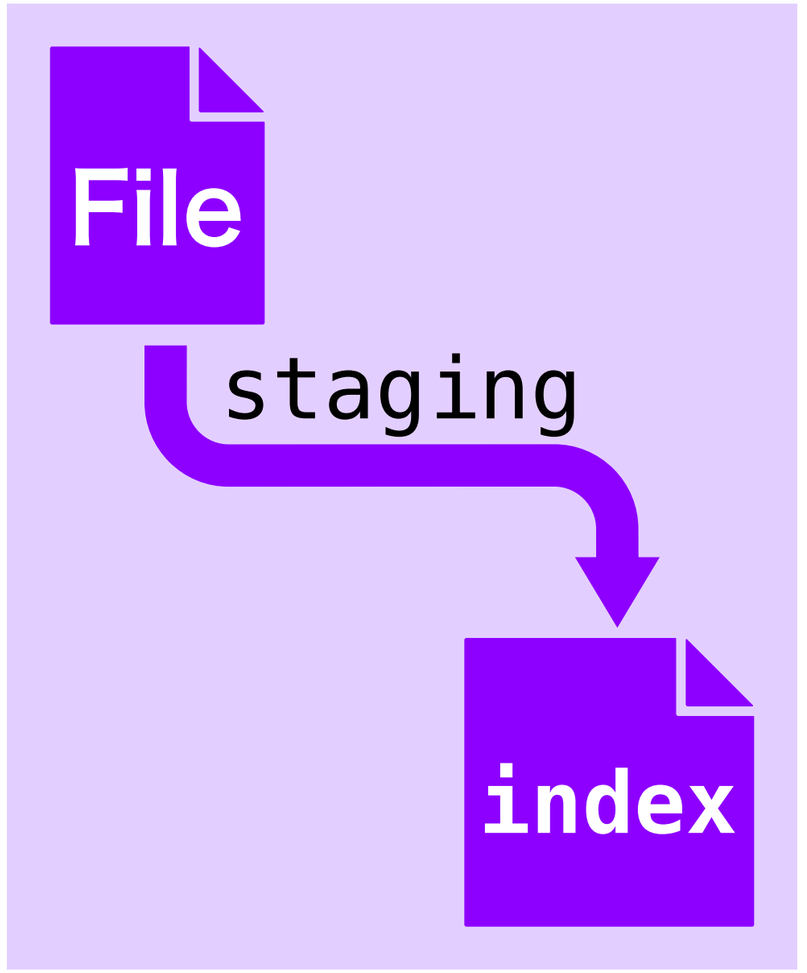
変更内容の登録

hunk `git add (-p|--patch)`

- **変更の単位**
- staging内容を対話的に選択

index `.git/index`

- **変更内容をstagingするファイル**



staging `git add`

- **コミットする内容を選別する操作**
- ファイルなどをindex (staging area) に追加すること

commit `git commit`

- **変更を記録する単位、及び操作**（indexの内容を記録）
- ID、説明、作成者、作成日時などのメタデータを含む

分岐と統合

branch　`git branch`

- **変更を分岐して管理する機能**

独立した
作業空間を用意

→

- 本番環境と独立して開発が可能
- 並列作業が可能

→ チーム開発で有用

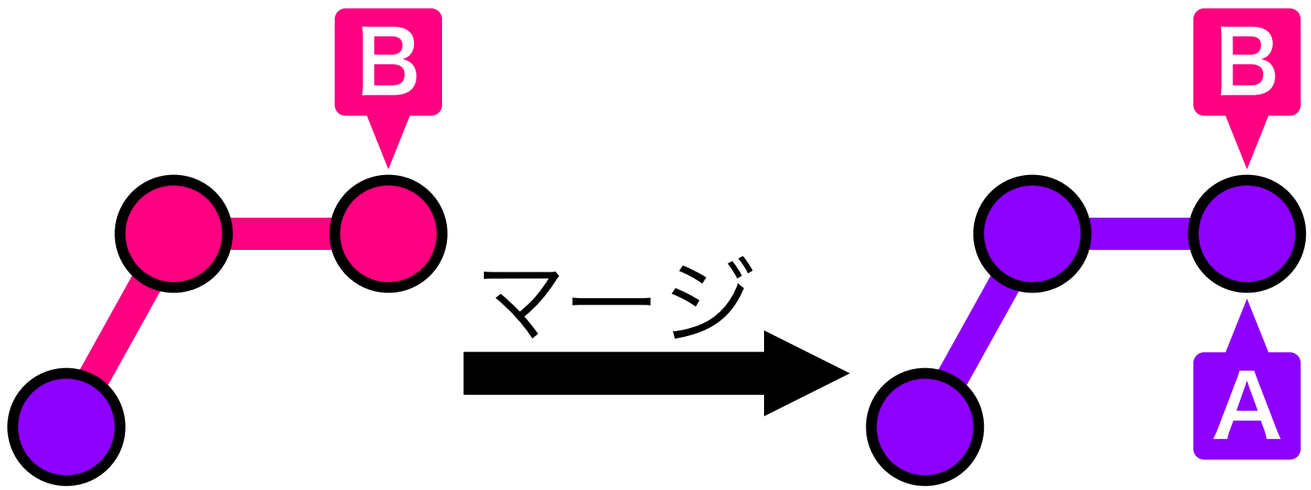
- ブランチを削除しても、含まれるコミットは削除されず、ガベージコレクションの対象となり、参照のないコミットは一定期間で削除される可能性がある

- **保護されたブランチ：**リポジトリ管理者が保護対象とした、Gitのいくつかの機能（強制プッシュや削除）がブロックされたブランチ

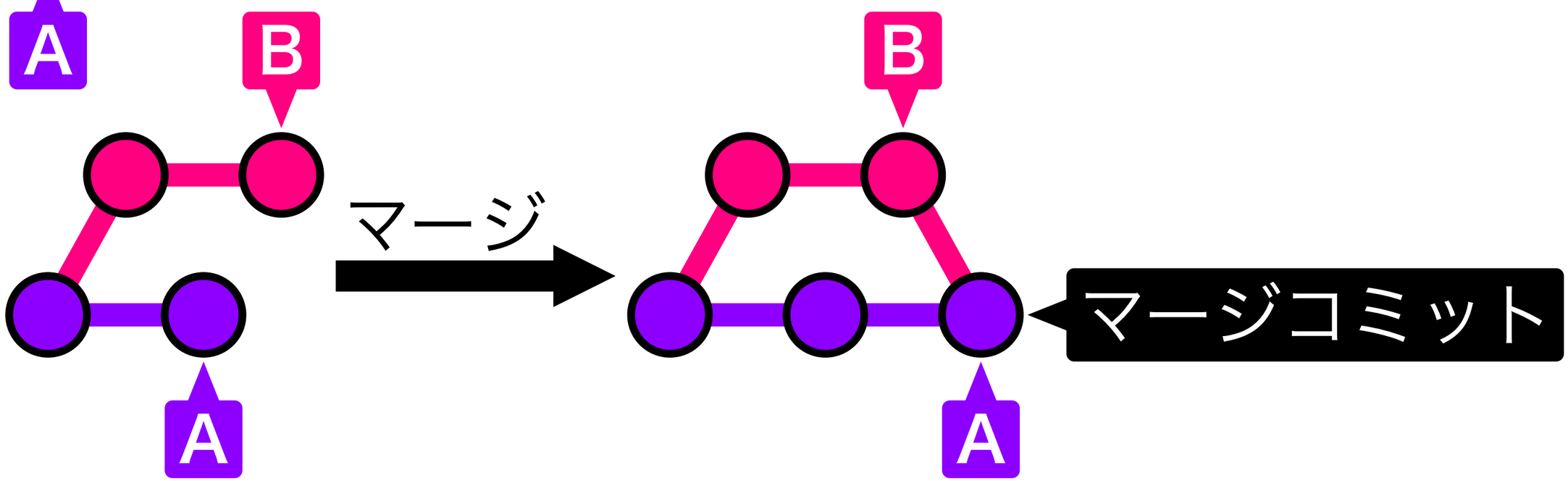
分岐と統合

merge　`git merge`

- **異なるブランチの変更を統合する操作**
- 通常はマージコミットを生成



fast-forwardマージ



分岐と統合

conflict

- **複数の変更が競合している状況**
- **マージコンフリクト：**異なるブランチで、
  同じファイルの同じ箇所が別々に変更され、
  Gitが自動でマージできない場合に発生

- **解決方法：**競合マーカーで囲まれた部分を手動で修正

リポジトリ間の同期

fetch `git fetch`

- **リモートリポジトリから最新の変更を取得する操作**
  （作業ツリーへの反映はない）
- 最新の変更を確認し、マージの準備に利用
- 必要に応じて `git merge` や `git rebase` を使用し、ローカルブランチに変更内容を取り込む

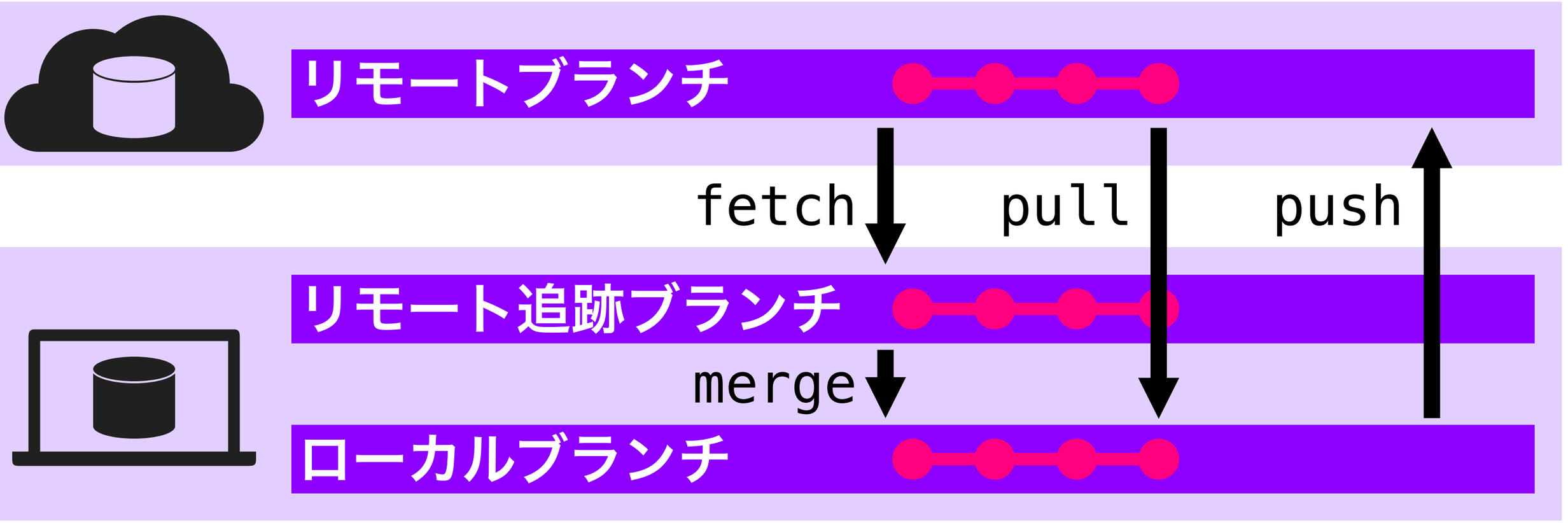
pull `git pull`

- **リモートリポジトリから最新の変更を取得（フェッチ）し、ローカルブランチに統合（マージ）する操作**

リポジトリ間の同期

push `git push`

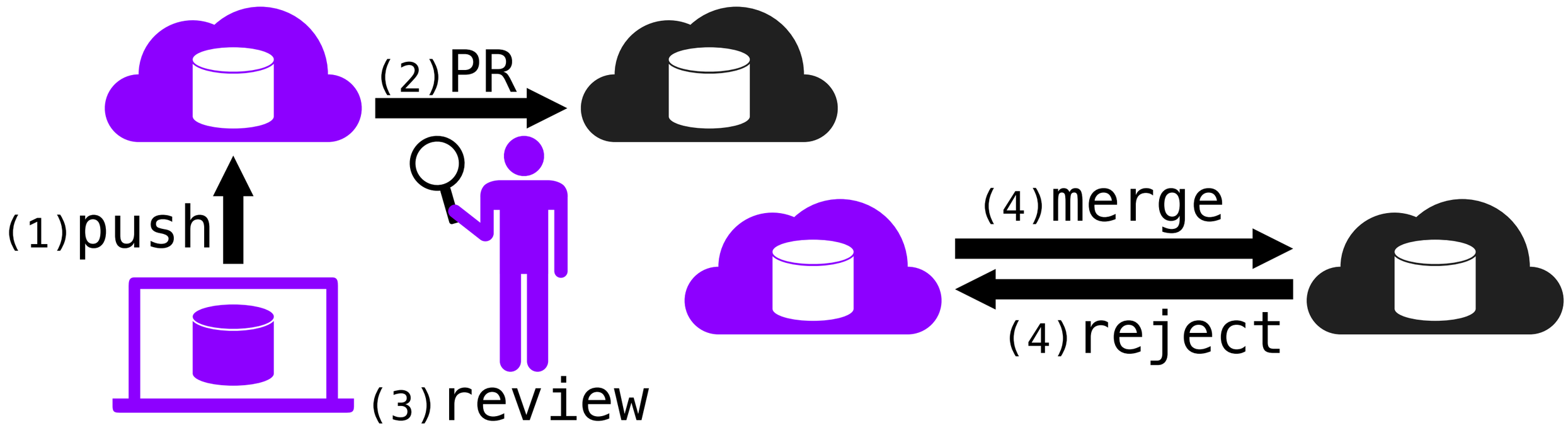
- **ローカルの変更をリモートに反映させる操作**
- アクセス権限やリモートの設定に応じて認証が必要
- **ブランチ更新の例：** `git push origin main`



リポジトリ間の同期

pull request

- **相手にプルを要求する機能**
- GitHubやGitLabなどで使用される
- コード変更後、リポジトリ管理者やチームメンバーにレビューやマージを依頼するために使用
- **役割**：コードの品質維持、フィードバック



おまけ

Tag　`git tag`

- **特定のコミットの識別子**
- **リリースのマーク：**Versionや安定版を示すために使用
- Version番号やリリース日付などを含むことが多い
  （例：`v1.0.0, security-patch2024`）

種類

軽量タグ：メタデータを持たず、名前のみを持つ

注釈付きタグ：作成者、作成日時、メッセージなどの情報を持つ。リリースなどの重要なコミットで使用

おまけ

.gitignore　.gitignore

- **Gitが無視すべきファイルを記述するファイル**
- ファイルをステージング不可にし、
  誤ってコミットするのを防ぐ

使用例

- 自動生成される
  ビルドファイルやログファイル
- パスワードや秘密情報を
  含むファイル

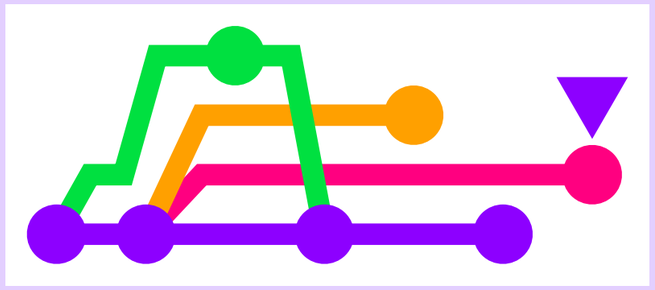
```
# 特定のファイル
secret.txt
# 特定のディレクトリ
logs/
# 特定の拡張子
*.log
# 特定のパターン
debug-*
# 例外指定
!important.log
```

Git入門

用語

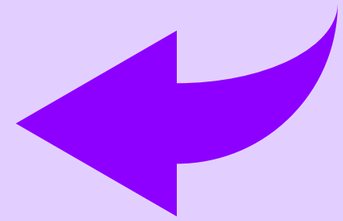
ツリーと移動

- working tree
- HEAD
- checkout
- stash



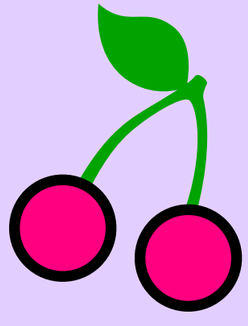
やり直し機能

- reset
- revert
- abort



整理整頓

- rebase
- squash
- cherry-pick



便利機能

- blame
- issue

おまけ

- README
- Markdown

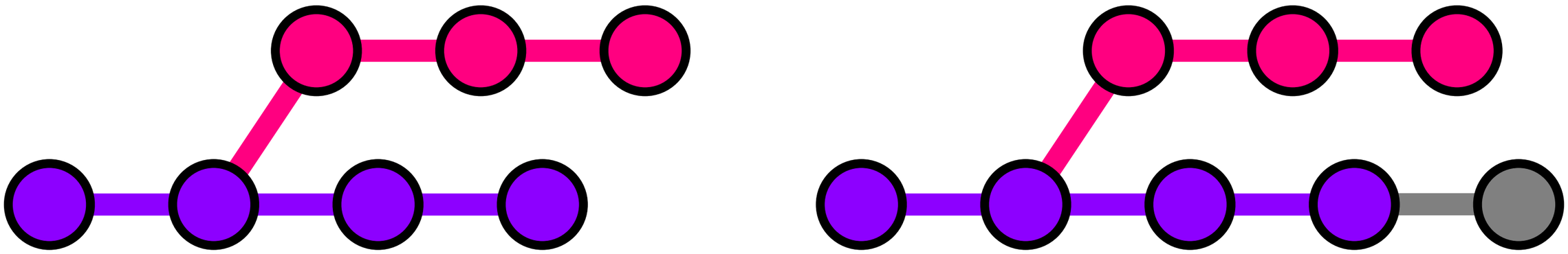
ツリーと移動

working tree

- **現在の作業ディレクトリ**
  （最後のコミットからの変更内容を含む）

クリーン：全ての変更がコミットされている状態

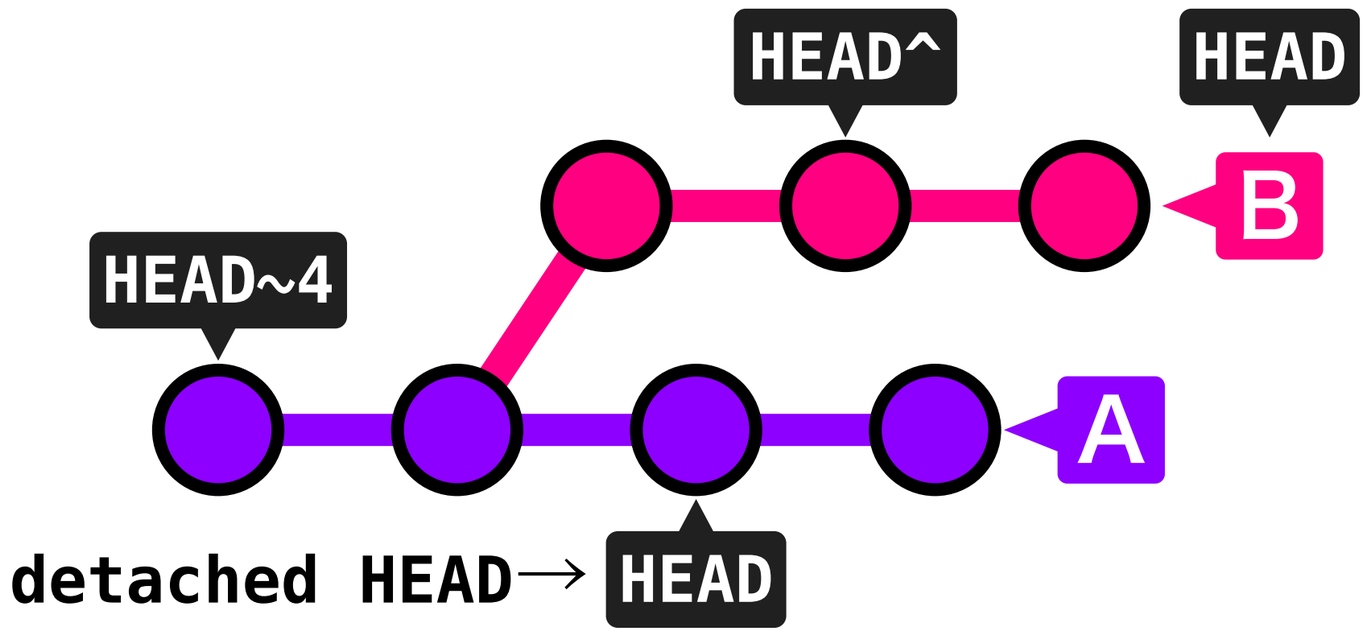
ダーティ：コミットされていない変更を含む状態



ツリーと移動

[detached] HEAD

- **作業中のコミットを指すポインタ**
- checkoutでブランチやコミットを移動すると、HEADが指すコミットも移動
- **直前のコミット：HEAD^, HEAD~で指定可能**
- **n個前のコミット：HEAD^^^, HEAD~nで指定可能**
- **detached HEAD**：どのブランチにも所属していないHEAD



ツリーと移動

checkout (switch, restore)　`git checkout`

- 主に、下記の３つの目的で使用する機能
- **ブランチの切り替え（switch）**

```
git (checkout|switch) <branch>
```

- **コミット、タグへの移動**

```
git checkout (<commit>|<tag>)
```

- **ファイルの復元（restore）**：作業ツリーやindexのファイルを最後のコミットの状態に復元

```
git (checkout --|restore) [<file>]
```

ツリーと移動

stash　`git stash`

- **作業ツリーの変更を一時的に退避させる操作**
- ブランチの切り替え、マージの競合回避などで使用

操作

- 作業ツリーの変更内容をstashに保存

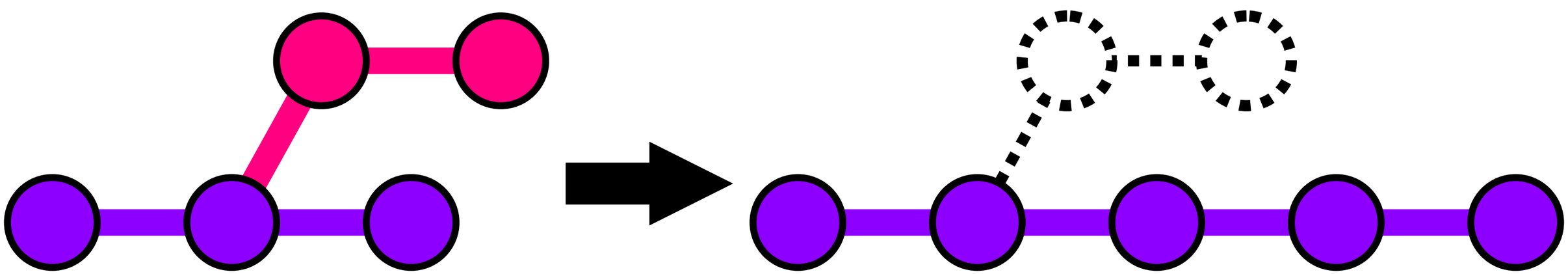
```
git stash save “Message”
```

- stashの一覧を表示 `git stash list`
- 最新のstashを適用 `git stash apply`
- 特定のstashを適用 `git stash apply stash@{<index>}`
- 全てのstashを削除 `git stash clear`
- 特定のstashを削除 `git stash drop stash@{<index>}`

整理整頓

rebase　`git rebase <branch>`

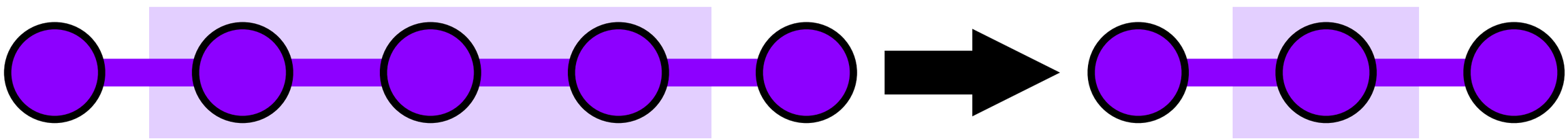
- **コミット履歴を整理するための機能**
- 現在のブランチのコミット履歴を<branch>の先頭に再適用
- `-i`（インタラクティブモード）の使用で、コミットの順序変更、結合、コミットメッセージの編集が可能



整理整頓

squash　`git rebase -i (<commit>|HEAD~n)`

- **複数のコミットを一つにまとめる操作**
- 多くの小さなコミットを意味のある一つのコミットにまとめることで、メンテナンス性が向上
- 指定したコミットからHEADまでのコミットに対し、`pick`, `squash`, `edit`などを選択して操作可能
- **squash**：選択したコミットを直前のコミットに結合



整理整頓

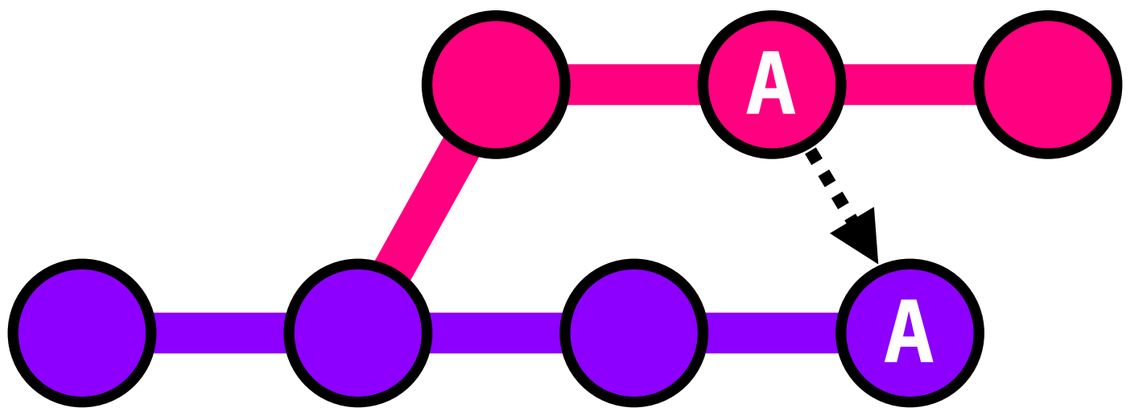
cherry-pick　`git cherry-pick`

- **特定のコミットを選んで取り込む操作**
- 特定のコミットを現在のブランチに取り込む

```
git cherry-pick <commit>
```

- 複数のコミットを順番に取り込む

```
git cherry-pick <commit1> <commit2> ...
```



やり直し機能

reset　`git reset`

- **コミット履歴や作業ツリーの状態を復元する機能**
- コミット履歴のリセット

```
git reset --soft (<commit>|HEAD~n)
```

- 上記＋indexのリセット

```
git reset [--mixed] (<commit>|HEAD~n) [<file>]
```

- 上記＋作業ツリーのリセット

```
git reset --hard (<commit>|HEAD~n)
```

- <commit>の省略値はHEAD

コミット履歴 `--soft`　**＋ index** `--mixed`　**＋ 作業ツリー** `--hard`

やり直し機能

revert　`git revert`

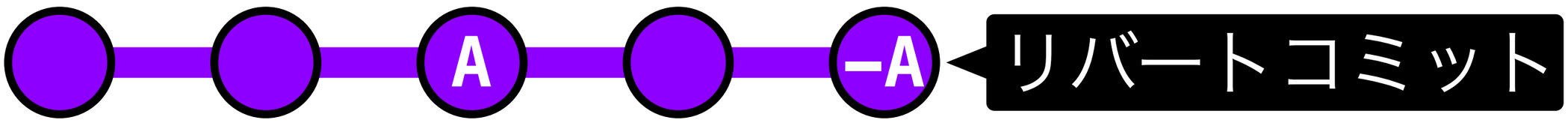
- **過去の変更を取り消す操作**
- **リバートコミット：**特定のコミットを打ち消すコミット
- **コミットの取り消し：**リバートコミットの作成
  （※元のコミットは残る）

```
git revert <commit>
```

- **マージの取り消し：**マージコミットの親コミットを指定
  （二つの親のうち、一つを指定）

```
git revert -m 1 <merge-commit>
```

- -m 1で1番目の親コミットを反映



やり直し機能

abort `git (merge|rebase) --abort`

- **実行中の操作（マージやリベース）を中止する操作**
- **マージの中止：**マージ中に競合が発生した場合や誤ってマージを開始した場合にマージを中止

```
git merge --abort
```

- **リベースの中止：**リベース中に競合が発生した場合や誤ってリベースを開始した場合にリベースを中止

```
git rebase --abort
```

- コンフリクトの解決が困難な場合に有用（バイナリファイルなど）

便利機能

blame　`git blame <file>`

- **ファイルの各行が最後に影響を受けたコミットの情報を表示する操作**（作成者、作成日時など）

用途

変更の追跡：ファイルの各行が、どの時点でどのように変更されたかの確認

原因の特定：バグや問題が、どのコミットで発生したかの特定

```
a390cf (John 2020-11-16 1) def func():
a390cf (John 2020-11-16 2)     a = 10
4gow0s (Mike 2020-11-18 3)     a *= 2
a390cf (John 2020-11-16 4)     return a
```

便利機能

issue

- **GitHubで提供されるバグ報告などを行うための仕組み**
- バグ・改善点・新機能のリクエストなどをissue
  として記録し、追跡する。開発者はissueを見て、
  作業の割り当てや解決策の議論などが可能
- ラベルを付けて分類が可能
- プルリクエストと関連付けてissueを解決

おまけ

README

- **プロジェクトの概要を記述したドキュメント**
  （使用方法・インストール手順・ライセンス情報など）
- ディレクトリ訪問者が最初に見るファイル
- プロジェクトの透明性・利便性向上に重要

Markdown

- **軽量なマークアップ言語**
- プレーンテキスト形式で記述し、HTMLに変換される
- シンプルな記法で読み書きが容易
- GitHub、Stack Overflow、Reddit、Qiitaなど
  広く採用されている

おまけ

Markdown

```
### 見出し
通常 _斜体_ **太字**
- 箇条書きリスト項目
1. 順序付きリスト項目
[text](http:///foo.com)
> 引用
`コード`
```

HTMLより簡素

HTML

```
<h3>見出し</h3>
通常 <em>斜体</em> <strong>太字
</strong>
<ul>
  <li>箇条書きリスト項目</li>
</ul>
<ol>
  <li>順序付きリスト項目</li>
</ol>
<a href="http://foo.com">text</a>
<blockquote>引用</blockquote>
<code>コード</code>
```

Gitflow

`main` リリース履歴の記録

`hotfix` 緊急対応（バグ修正）

`release` リリースの準備

`develop` 機能の統合

`feature` 機能の開発