

# Git 研修

kota.nara @ミクシィ (2019)

# 研修の目標

---

- ・ 想定レベル
  - ・ 個人プロジェクトで commit, push などを使ったことがある

# 研修の目標

---

- ・ 想定レベル
  - ・ 個人プロジェクトで commit, push などを使ったことがある
- ・ 目標
  - ・ チーム開発特有の git の使い方 (PR など) を知っている
  - ・ 典型的なトラブルで「なにが起きているのか」想像がつく

# 実習① GitHub Flow 体験

# 実習① GitHub Flow 体験

---

- ・の前に

Git とは

# Git とは

---

- ・の前に、まずは Git の概要をざっくり復習

# Git とは

---

- ・の前に、まずは Git の概要をざっくり復習
  - ・バージョン管理ツール (VCS)



# Git とは

---

- ・の前に、まずは Git の概要をざっくり復習
  - ・バージョン管理ツール (VCS)
    - ・たくさんのバージョンを効率よくバックアップ

# Git とは

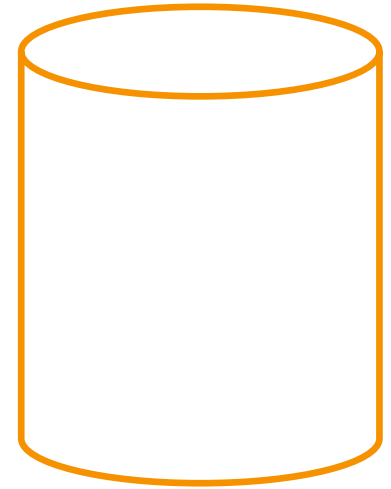
---

- ・の前に、まずは Git の概要をざっくり復習
  - ・バージョン管理ツール (VCS)
    - ・たくさんのバージョンを効率よくバックアップ
      - ・バックアップを取るたびに容量が 2 倍 3 倍 … にならない
      - ・変更履歴を簡単に辿れる
      - ・改竄を検知できる
      - ・など …

# Git とは

---

- Git の基本概念①

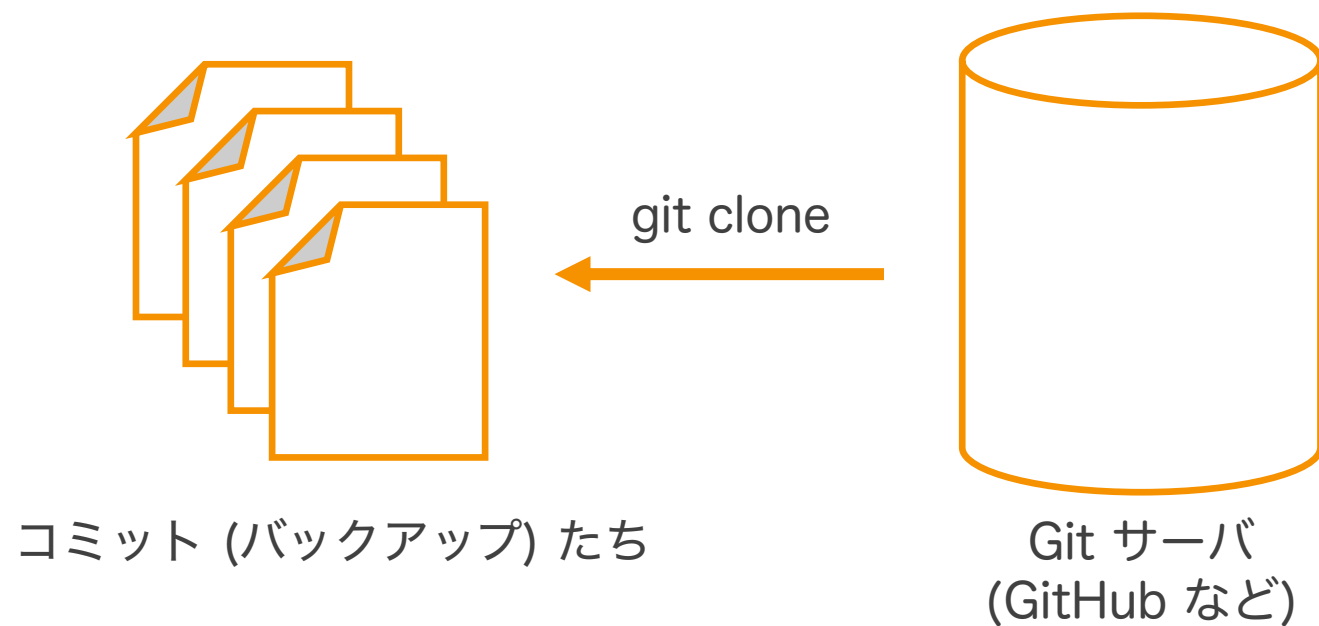


Git サーバ  
(GitHub など)

# Git とは

---

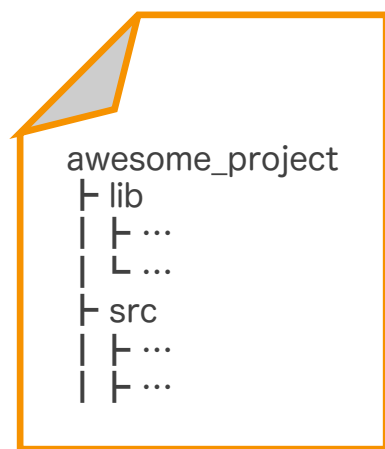
- Git の基本概念①



# Git とは

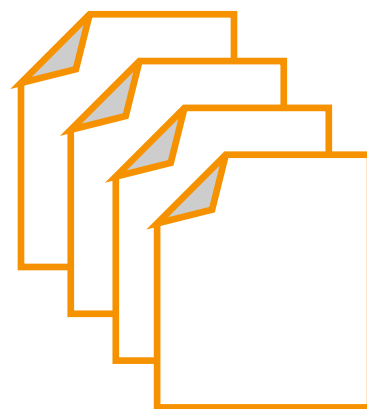
---

- Git の基本概念①



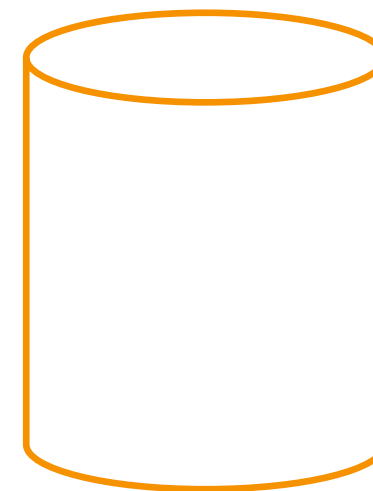
作業ディレクトリ  
(ワークツリー)

(checkout)



コミット (バックアップ) たち

git clone



Git サーバ  
(GitHub など)

# Git とは

---

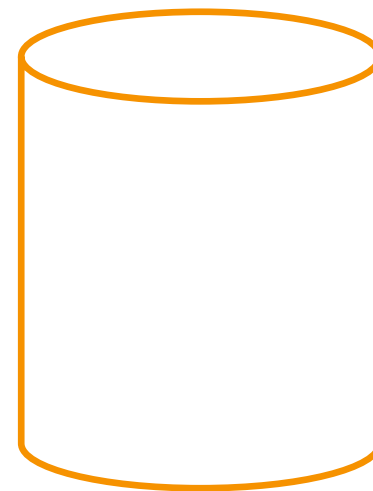
- Git の基本概念①



作業ディレクトリ  
(ワークツリー)



コミット (バックアップ) たち



Git サーバ  
(GitHub など)

# Git とは

---

- Git の基本概念①

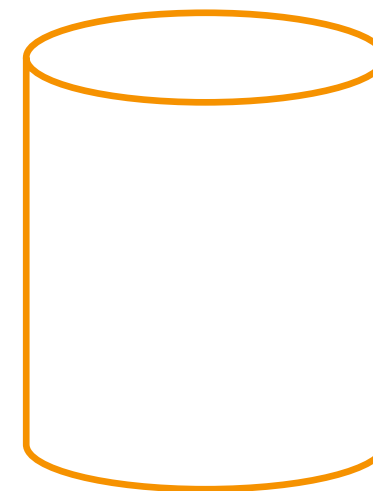


作業ディレクトリ  
(ワークツリー)

git commit



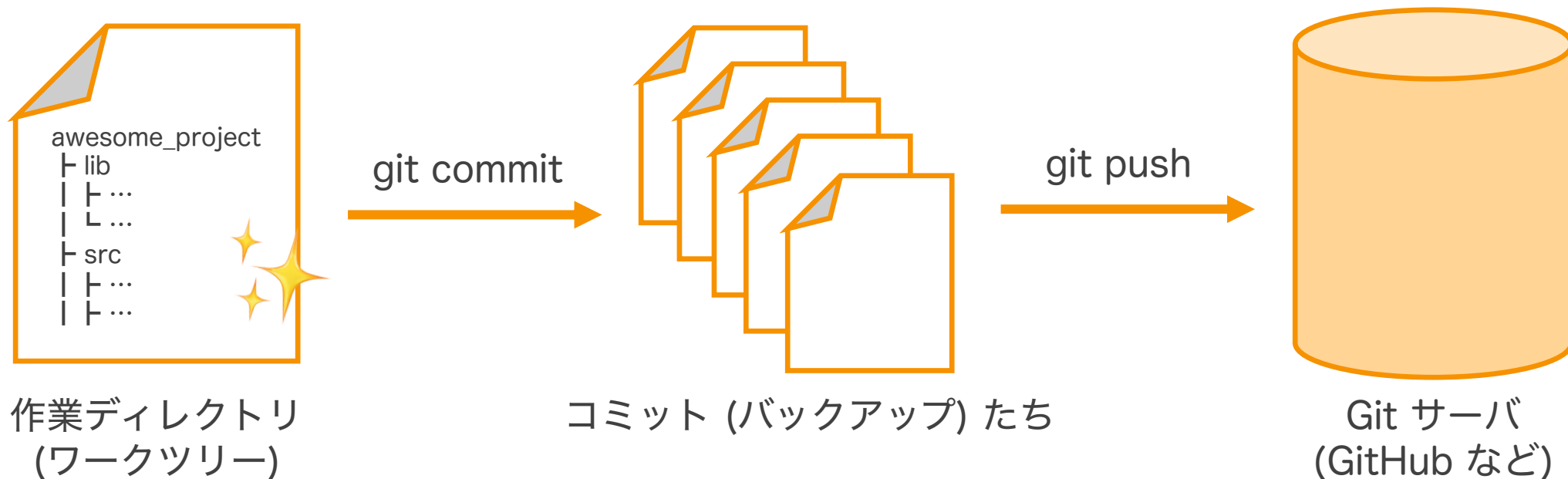
コミット (バックアップ) たち



Git サーバ  
(GitHub など)

# Git とは

- Git の基本概念①

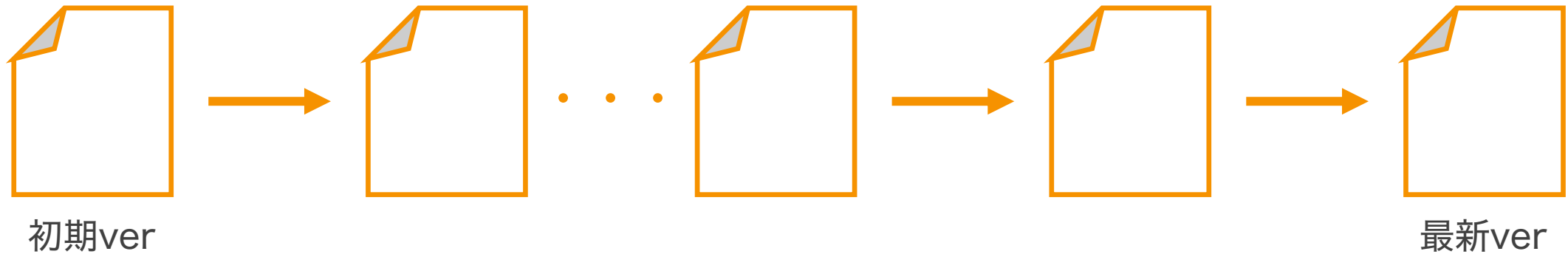




# Git とは

---

- Git の基本概念②



- ・コミット (バックアップ) 間には親子関係がある
- ・バージョン履歴を簡単に辿れる

# Git とは

---

- Git の基本概念②

```
$ git log --oneline  
3a943dc ぴよ機能を追加  
7fb3958 ほげ機能を追加  
d193ec3 Initial commit
```

- コミット (バックアップ) 間には親子関係がある
- バージョン履歴を簡単に辿れる

# Git とは

---

- Git の基本概念②

```
$ git log --oneline  
3a943dc ぴよ機能を追加  
7fb3958 ほげ機能を追加  
d193ec3 Initial commit
```

- コミット (バックアップ) 間には親子関係がある
- バージョン履歴を簡単に辿れる

# Git とは

---

- Git の基本概念②

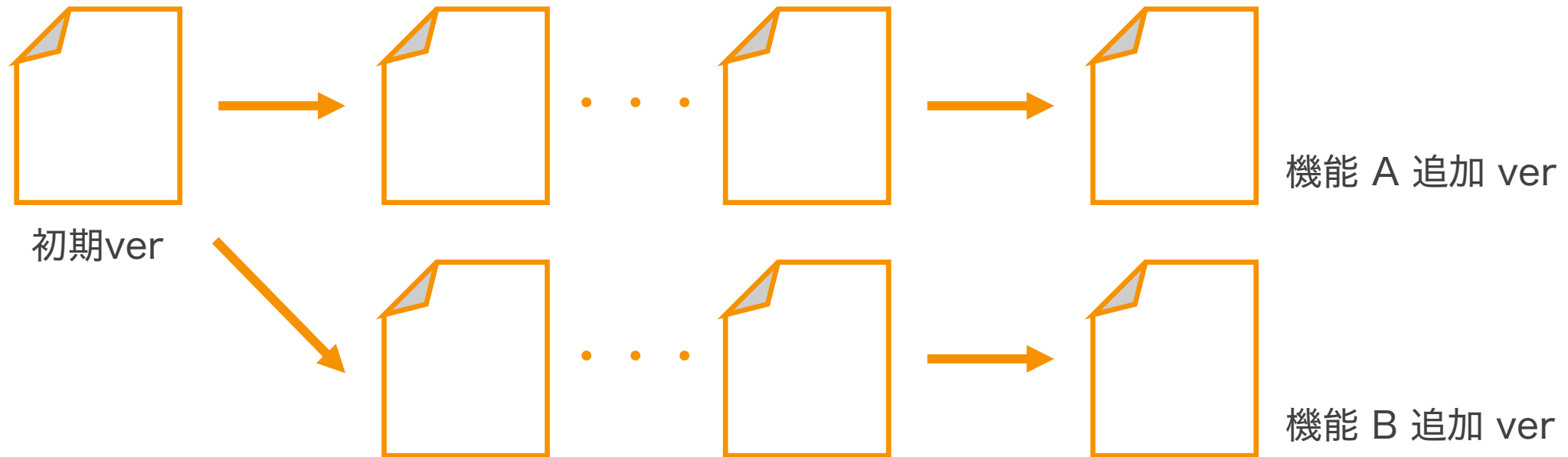
```
$ git log --oneline
3a943dc ぴよ機能を追加
7fb3958 ほげ機能を追加
d193ec3 Initial commit
$ git checkout 7fb3958
```

- コミット (バックアップ) 間には親子関係がある
- バージョン履歴を簡単に辿れる

# Git とは

---

- Git の基本概念②

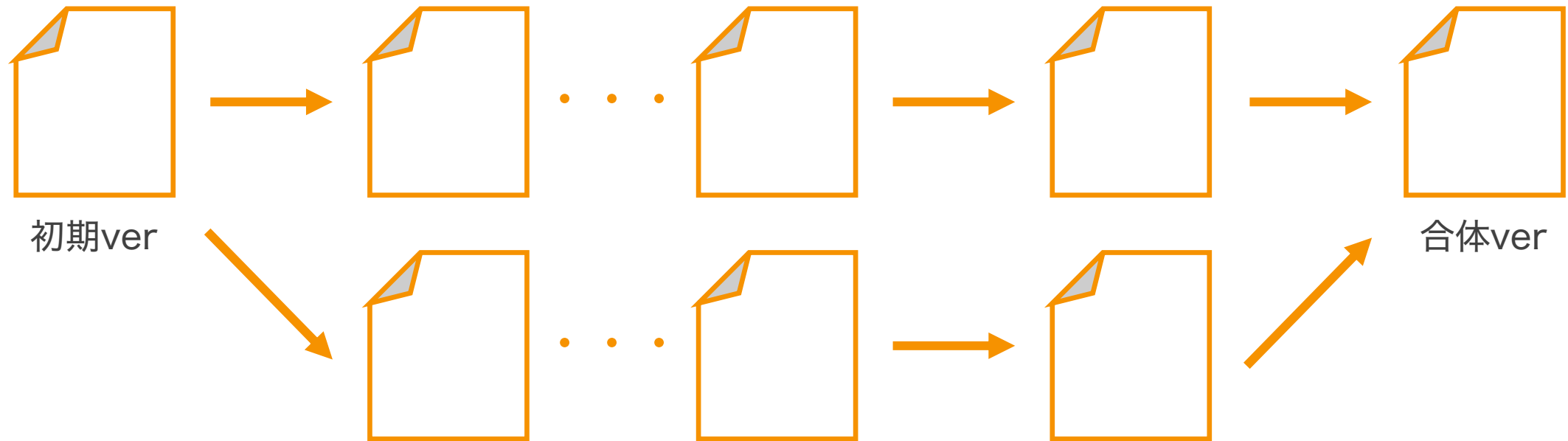


- 一つの親から複数の派生バージョン (ブランチ) を作れる

# Git とは

---

- Git の基本概念②



- 複数のブランチを合算したバージョンを自動で作れる (マージ)

# 実習① GitHub Flow 体験

# 実習① GitHub Flow 体験

---

- ・ みんなでいっぺんに一つのものを作る体験をします



## 実習① GitHub Flow 体験

---

- ・ みんなでいっぺんに一つのものを作る体験をします
  - ・ GitHub Flow ... Git を使ったチーム開発の一宗派

準備

# 準備

---

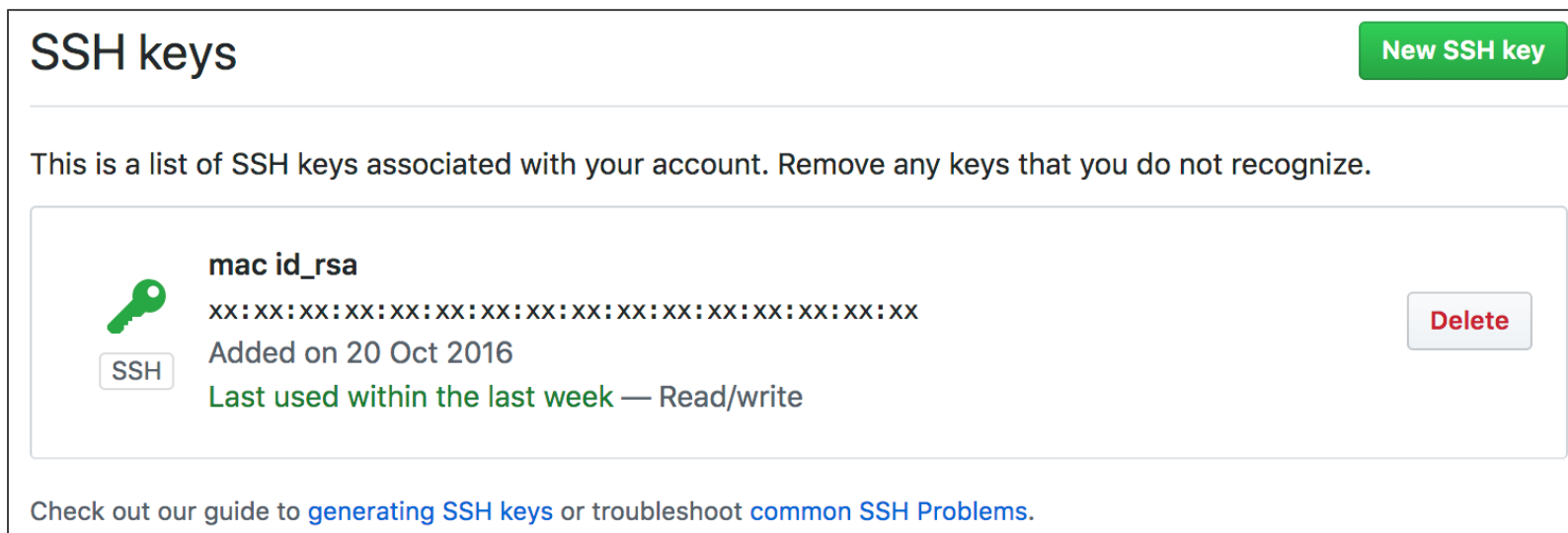
- git が入っていることを確認

```
$ which git  
/usr/local/bin/git
```

- まだの人がいたら気軽に止めてください 🙋🙋

# 準備

- GitHub に公開鍵を設置 (Settings > SSH Keys)



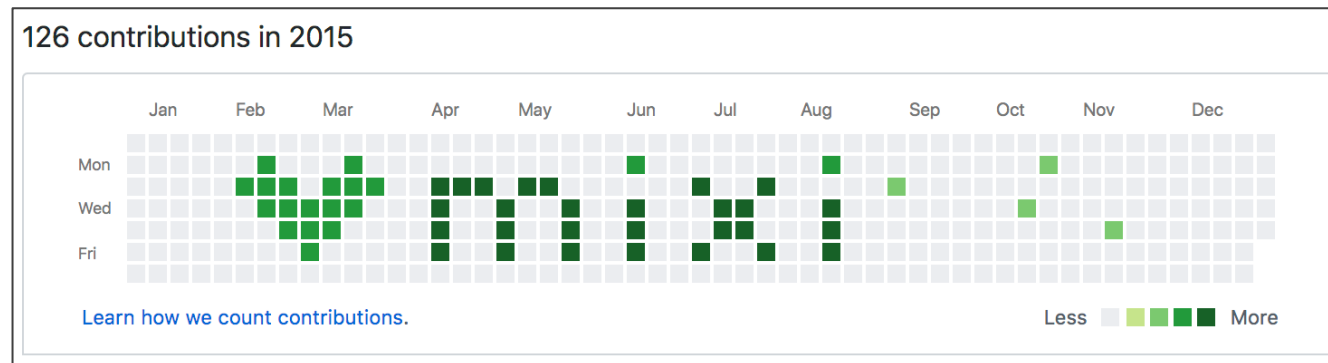
- 鍵がまだない場合は `ssh-keygen -t rsa` などで作る

# 準備

- git の署名を設定

```
$ git config --global user.name kota.nara  
$ git config --global user.email xxxxxx@xxxxxx.xx
```

- メアドが正しくないと草が生えません 😓



リモートリポジトリの取得

# リモートリポジトリの取得

---

- ・ GitHub 上で管理されているリポジトリをローカルで使う

# リモートリポジトリの取得

---

- ・ まずは適当なディレクトリをこしらえて git init

```
$ mkdir github-tutorial  
$ cd github-tutorial  
$ git init  
Initialized empty Git repository in ~/Documents/git-tutorial/.git/
```



# リモートリポジトリの取得

---

- ・ まずは適当なディレクトリをこしらえて git init

```
$ ls -a .  
.  
..  
.git
```

- ・ .git ディレクトリができる (内部データが格納される)

# リモートリポジトリの取得

---

- ・ リモートリポジトリの url を設定する

```
$ git remote add origin <url>
```

- ・ ”origin” は url につける名前 (複数追加できるので)

# リモートリポジトリの取得

---

- ・ リモートリポジトリの url を設定する

```
$ git remote -v  
origin <url> (fetch)  
origin <url> (push)
```

- ・ 間違っていたら `git remote set-url` で修正できる

# リモートリポジトリの取得

---

- "origin" の最新バージョンを手元にダウンロード

```
$ git pull origin master  
$ ls  
README.markdown    CONTRIBUTORS.markdown
```

- "master" はデフォルトのブランチ名
- つながらない人👤👤

# リモートリポジトリの取得

---

- ”origin” の最新バージョンを手元にダウンロード

```
$ cat CONTRIBUTORS.markdown
# kota.nara

- 得意 ... Lisp系, Perl, Vue/React.js, デザインシステム, Emacs
- 趣味 ... キーボード, ビール, 散歩, スーパードンキーコング

# ...
...
```

- みんなで一斉に自分の欄を埋めて、いっぺんに完成させてみる

# リモートリポジトリの取得

---

- ・ ちなみに
  - ・ ここまでの作業を一発でやってくれる便利コマンド

```
$ git clone <url>
```

ブランチを作る

# ブランチを作る

---

- ・ みんなで並行して作業するために、自分の作業に名前をつける



# ブランチを作る

---

- ・ みんなで並行して作業するために、自分の作業に名前をつける

```
$ git branch section-kota-nara
```

- ・ 他とかぶることがなければ基本的になんでもいい
- ・ 名前である程度どんな作業をしているかわかるとベター

# ブランチを作る

---

- ・ みんなで並行して作業するために、自分の作業に名前をつける

```
$ git branch section-kota-nara
```

- ・ 他とかぶることがなければ基本的になんでもいい
- ・ 名前である程度どんな作業をしているかわかるとベター
- ・ おそらくチームごとに文化があるので空気を读もう

# ブランチを作る

---

- ・ みんなで並行して作業するために、自分の作業に名前をつける

```
$ git branch  
* master  
  section-kota-nara
```

- ・ 新しいブランチができた
- ・ まだ master がアクティブ (“checkout されている”)

# ブランチを作る

---

- ・新しく作ったブランチを checkout

```
$ git checkout section-kota-nara  
$ git branch  
master  
* section-kota-nara
```

# ブランチを作る

---

- ・ ちなみに
  - ・ ブランチを作って checkout までしてくれる便利コマンド

```
$ git checkout -b section-kota-nara
```

コミットを作る

# コミットを作る

---

- ・ ファイルをよしなに編集してください

```
$ vim CONTRIBUTORS.markdown
```

```
...
```

```
:wq
```

# コミットを作る

---

- ・ ファイルをよしなに編集してください

```
$ vim CONTRIBUTORS.markdown
```

```
...
```

```
:wq
```

- ・ (5分くらいでここはひとつ…)



# コミットを作る

---

- ・ 編集後の内容でコミットを作る

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)

        modified:   CONTRIBUTORS.markdown

no changes added to commit (use "git add" and/or "git commit -a")
```

# コミットを作る

---

- ・ 編集後の内容でコミットを作る

```
$ git add CONTRIBUTORS.markdown
$ git status
On branch master
Changes to be committed:
  (use "git reset <file>..." to unstage)

        modified:   CONTRIBUTORS.markdown
```

# コミットを作る

---

- ・ 編集後の内容でコミットを作る

```
$ git diff --staged
diff --git a/CONTRIBUTORS.markdown b/CONTRIBUTORS.markdown
index 47155f3..87b19a8 100644
--- a/CONTRIBUTORS.markdown
+++ b/CONTRIBUTORS.markdown
@@ -1,7 +1,7 @@
 # kota.nara

- 得意 ... Lisp系, Perl, Vue/React.js, デザインシステム, Emacs
-- 趣味 ... キーボード, ビール, 散歩, スーパードンキーコング
+- 趣味 ... キーボード, ビール, 散歩, 温泉, スーパードンキーコング
```

# コミットを作る

---

- ・編集後の内容でコミットを作る

```
$ git commit -m "kota.nara に温泉を追加する"  
[master 972100d] kota.nara に温泉を追加する  
1 file changed, 1 insertion(+), 1 deletion(-)
```

# コミットを作る

---

- ・編集後の内容でコミットを作る

```
$ git log --oneline
08df9b4 (HEAD -> section-kota-nara) kota.nara に温泉を追加する
f616693 (origin/master, master) Initial commit
```

Pull Request を作る

# Pull Request を作る

---

- Pull Request
  - 「差分できたから master に取り込んで欲しい～」という表明

# Pull Request を作る

---

- Pull Request
  - 「差分できたから master に取り込んで欲しい～」という表明
  - リリース前に誰かのレビューをもらうための仕組み



# Pull Request を作る

---

- Pull Request
  - 「差分できたから master に取り込んで欲しい～」という表明
  - リリース前に誰かのレビューをもらうための仕組み
  - Git ではなく GitHub (や Bitbucket, GitLab) の機能

# Pull Request を作る

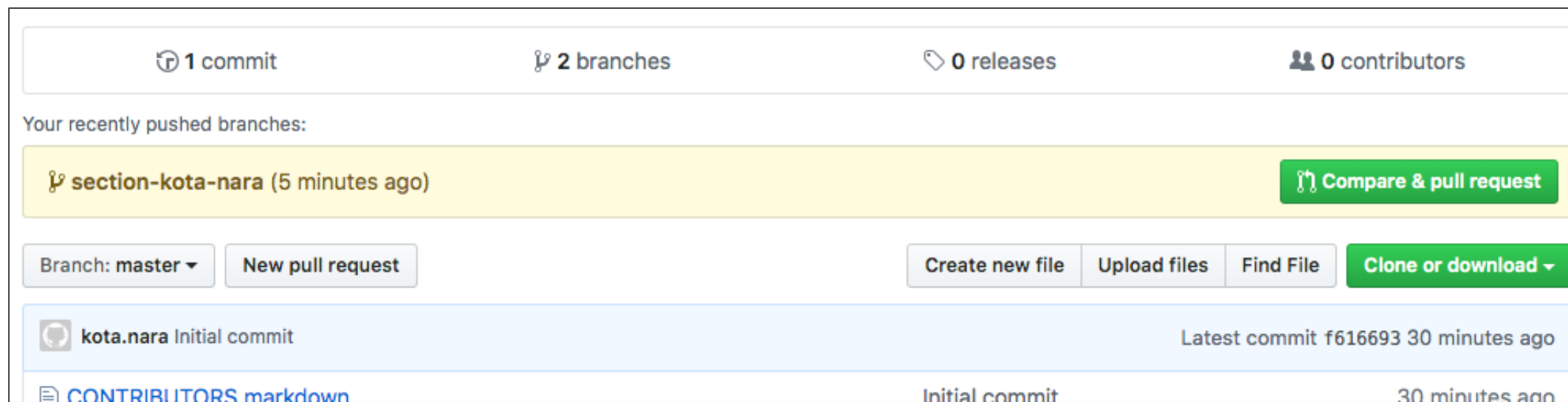
---

- ・ 自分の作業を GitHub にアップロードする

```
$ git push origin section-kota-nara
```

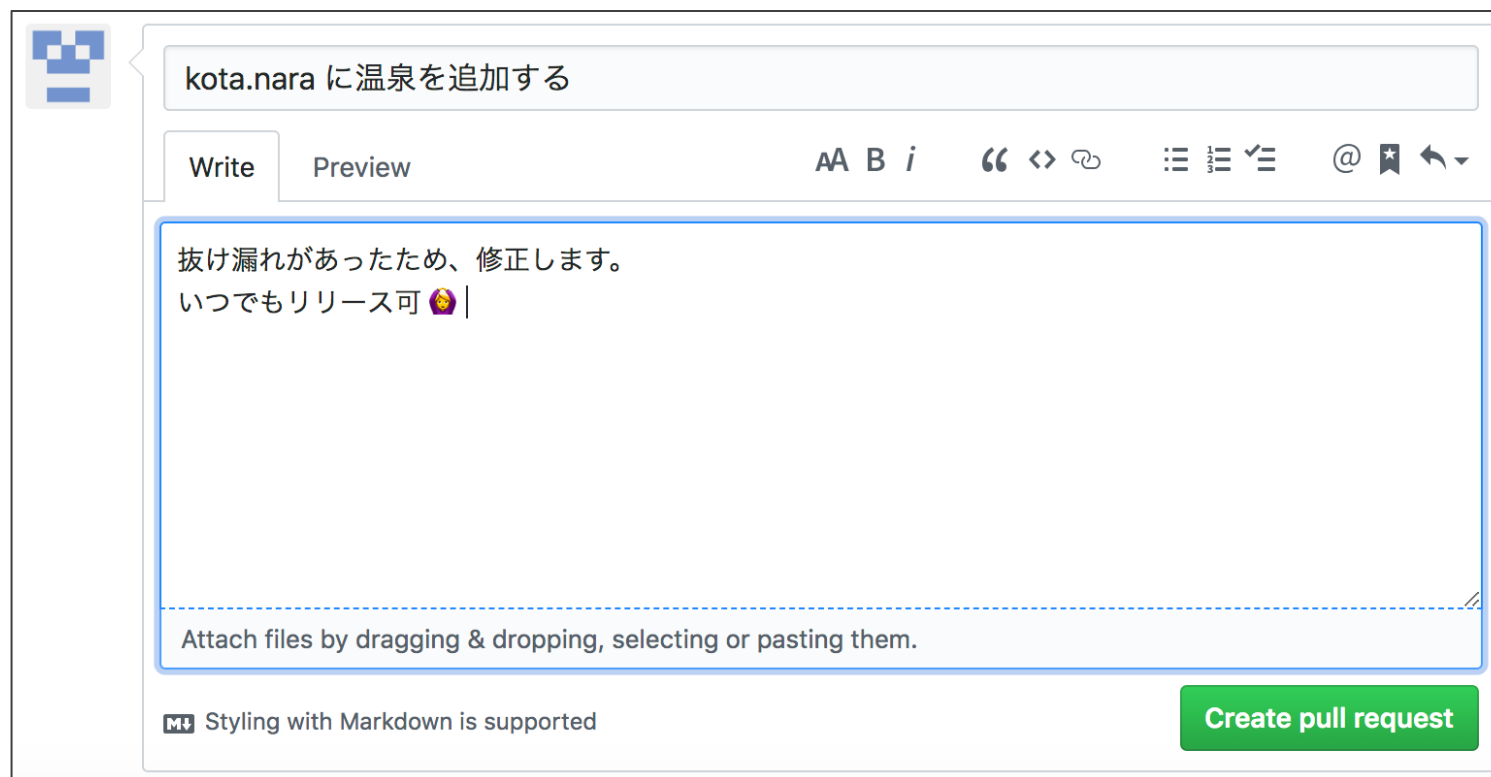
# Pull Request を作る

- GitHub 上で確認



# Pull Request を作る

- Pull Request を作成



The screenshot shows the GitHub interface for creating a pull request. At the top left is the GitHub logo. Below it is a text input field containing "kota.nara に温泉を追加する". Below the input field are two tabs: "Write" (selected) and "Preview". To the right of the tabs is a rich text editor toolbar with icons for bold (AA), italic (i), quote (”), code (<>), link (🔗), list (☰), ordered list (☰), and undo (↶). Below the toolbar is a large text area containing the text: "抜け漏れがあったため、修正します。いつでもリリース可 🍷 |". Below the text area is a dashed line and the text: "Attach files by dragging & dropping, selecting or pasting them." At the bottom left is a small icon and the text: "Styling with Markdown is supported". At the bottom right is a green button with the text: "Create pull request".

kota.nara に温泉を追加する

Write Preview

抜け漏れがあったため、修正します。  
いつでもリリース可 🍷 |

Attach files by dragging & dropping, selecting or pasting them.

Styling with Markdown is supported

Create pull request

## Pull Request を作る

- Pull Request を作成

🔗

1 Open

✓ 0 Closed

Author ▾Labels ▾Projects ▾Milestones ▾Reviews ▾Assignee ▾Sort ▾

🔗

kota.nara に温泉を追加する

#1 opened 13 hours ago by k-nara

Pull Request をマージする

# Pull Request をマージする

---

- ・ ペアのプルリクを相互チェックしよう

# Pull Request をマージする

- ・ ペアのプルリクを相互チェックしよう
- ・ 相方の作ったプルリクのページを開く





# Pull Request をマージする

- ・ ペアのプルリクを相互チェックしよう
  - ・ コミットの内容を確認

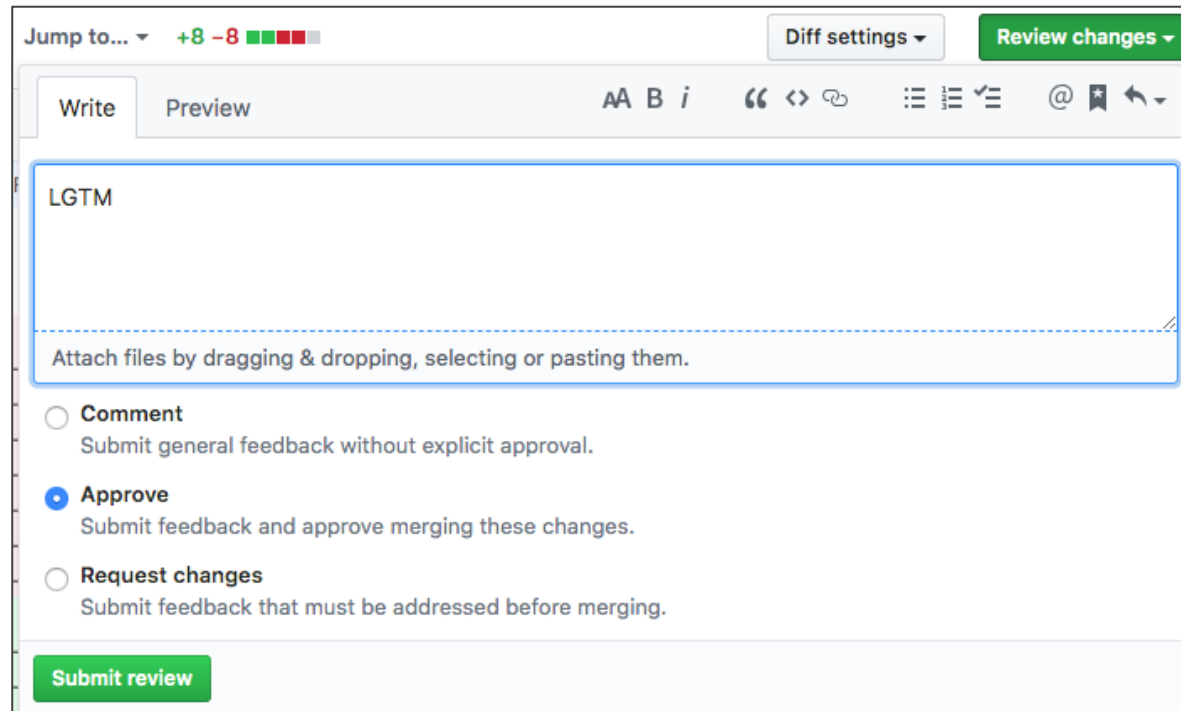


The screenshot shows a code diff interface for the file `CONTRIBUTORS.markdown`. The diff highlights a change on line 4. The original content (line 4) is shown in a red background, and the new content (line 4) is shown in a green background. The change is a modification to the 'kota.nara' entry, adding '温泉' (Onsen) to the list of interests.

```
2  @@ -1,7 +1,7 @@  
1  1      # kota.nara  
2  2  
3  3      - 得意 ... Lisp系, Perl, Vue/React.js, デザインシステム, Emacs  
4  4      - 趣味 ... キーボード, ビール, 散歩, スーパードンキーコング  
4  4      + 趣味 ... キーボード, ビール, 散歩, 温泉, スーパードンキーコング  
5  5
```

# Pull Request をマージする

- ペアのプルリクを相互チェックしよう
  - 特に気になるところがなければ Approve を選ぶ



The screenshot shows the GitHub Pull Request review interface. At the top, there's a 'Jump to...' dropdown, a diff summary '+8 -8' with colored squares, a 'Diff settings' dropdown, and a green 'Review changes' button. Below this is a tabbed interface with 'Write' and 'Preview' tabs. The 'Write' tab is active, showing a text area with 'LGTM' entered. Below the text area is a dashed line and a prompt: 'Attach files by dragging & dropping, selecting or pasting them.' At the bottom, there are three radio button options: 'Comment' (Submit general feedback without explicit approval.), 'Approve' (Submit feedback and approve merging these changes.), and 'Request changes' (Submit feedback that must be addressed before merging.). The 'Approve' option is selected. At the very bottom is a green 'Submit review' button.

# Pull Request をマージする

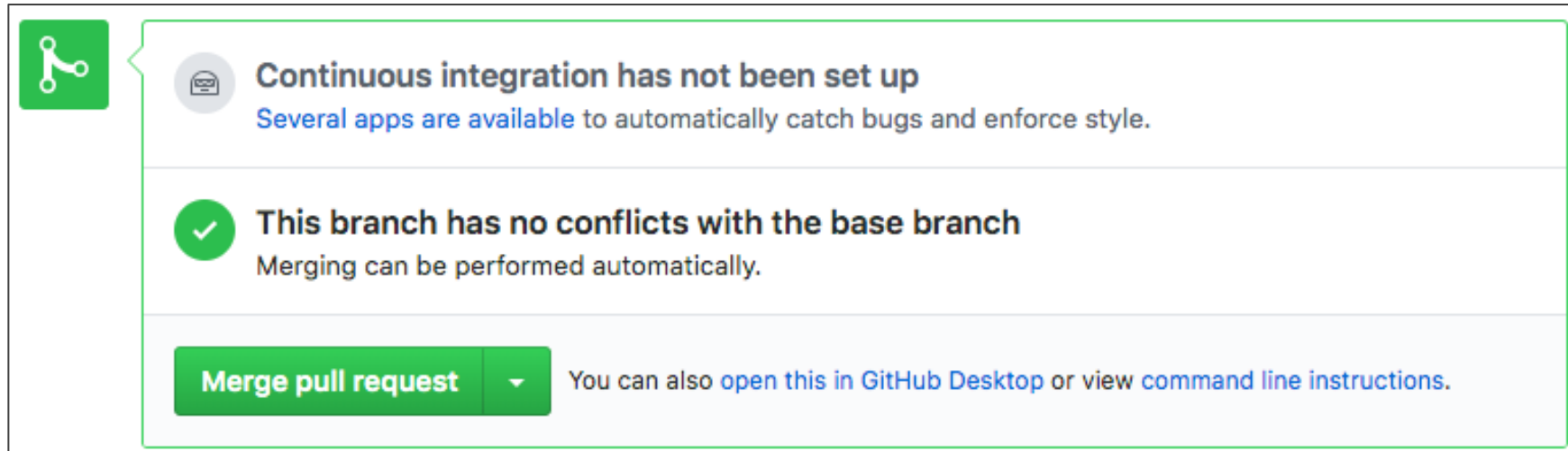
---

- ・ ペアのプルリクを相互チェックしよう
  - ・ 特に気になるところがなければ Approve を選ぶ
  - ・ Approve されなかった場合、修正コミットを追加して再 push

# Pull Request をマージする

---

- ・ 自分の PR がレビューに通ったら Merge (master に取り込む)



- ・ Merge されたブランチは、都度削除する場合が多い

# Pull Request をマージする

---

- ・ ちなみに：
  - ・ GitHub Review は比較的最近の機能
  - ・ 単に「LGTM」コメントをつける運用のチームもあるかも

ローカルリポジトリを追従する

# ローカルリポジトリを追従する

---

- ・ master に無事みんなの作業結果が入りました 🎉

# ローカルリポジトリを追従する

---

- ・ master に無事みんなの作業結果が入りました 🎉
- ・ 手元のリポジトリも更新しよう



# ローカルリポジトリを追従する

---

- ・ master に無事みんなの作業結果が入りました 🎉
- ・ 手元のリポジトリも更新しよう

```
$ git checkout master  
$ git pull origin master
```

- ・ リモートの master に入った差分を、手元の master に取込む

# ローカルリポジトリを追従する

---

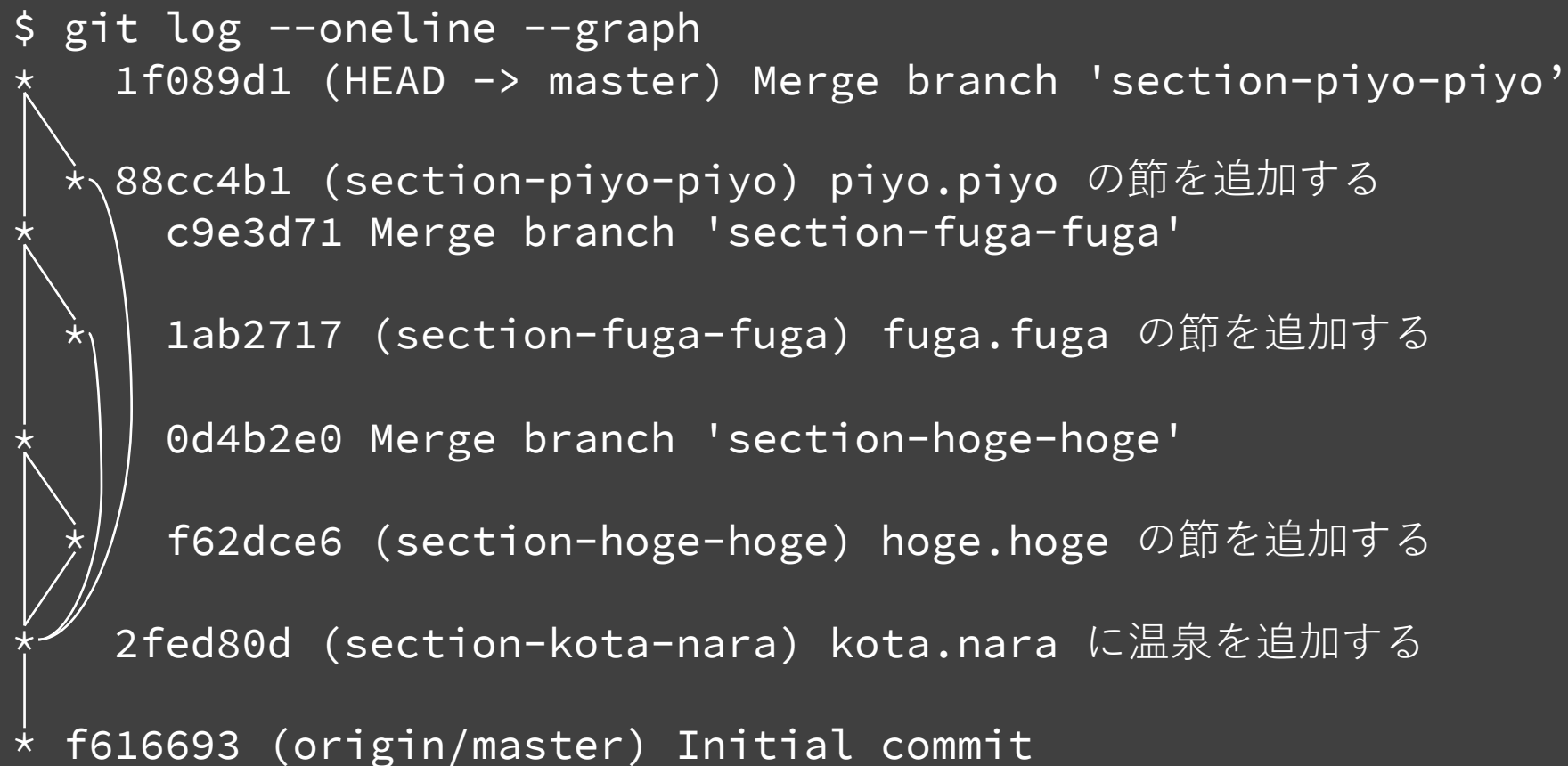
- master に無事みんなの作業結果が入りました 🎉
- 手元のリポジトリも更新しよう

```
$ cat CONTRIBUTORS.markdown  
# kota.nara  
...
```



# ローカルリポジトリを追従する

```
$ git log --oneline --graph
* 1f089d1 (HEAD -> master) Merge branch 'section-piyo-piyo'
* 88cc4b1 (section-piyo-piyo) piyo.piyo の節を追加する
* c9e3d71 Merge branch 'section-fuga-fuga'
* 1ab2717 (section-fuga-fuga) fuga.fuga の節を追加する
* 0d4b2e0 Merge branch 'section-hoge-hoge'
* f62dce6 (section-hoge-hoge) hoge.hoge の節を追加する
* 2fed80d (section-kota-nara) kota.nara に温泉を追加する
* f616693 (origin/master) Initial commit
```



The diagram illustrates a Git commit history. It starts with an initial commit 'f616693 (origin/master) Initial commit'. A vertical line of commits follows: '2fed80d (section-kota-nara) kota.nara に温泉を追加する', 'f62dce6 (section-hoge-hoge) hoge.hoge の節を追加する', '0d4b2e0 Merge branch 'section-hoge-hoge'', '1ab2717 (section-fuga-fuga) fuga.fuga の節を追加する', and 'c9e3d71 Merge branch 'section-fuga-fuga''. From 'c9e3d71', a branch 'section-piyo-piyo' is created, leading to '88cc4b1 (section-piyo-piyo) piyo.piyo の節を追加する'. Finally, this branch is merged back into the main line, resulting in '1f089d1 (HEAD -> master) Merge branch 'section-piyo-piyo''. The graph uses asterisks to mark each commit and lines to show the parent-child relationships and merges.

Pro Tip 💡

## Pro Tip 💡

---

- ・ おまけ：気をつけるとちょっといいかもしれないこと

## Pro Tip 💡

---

- ・サクッとレビューできる PR を作る：
  - ・巨大な PR はレビューにもまとまった時間が必要
  - ・大きくなりそうな場合は
    - ・未完成でも早めに方針を見てもらおう
      - ・(タイトルに「WIP」とつけて区別したりする)
  - ・コミットログを綺麗にしておこう
    - ・巨大なコミットを避ける、意味のある単位で分ける

## Pro Tip 💡

---

- ・コミットメッセージについて：
  - ・原則、チームの文化に従う
  - ・一般に良いとされるのは、
    - ・一行の簡潔な要約を現在形で書く
    - ・○○テーブルを追加する
    - ・△△キャンペーンのコードを削除する
  - ・補足が必要な場合は、空行を置いて三行目以降に

## Pro Tip

---

- ・ レビューのポイント：
  - ・ 人間が見るところ
    - ・ バグ (見つかったらラッキー、テストを書くのが原則)
    - ・ 仕様の穴、セキュリティ/法的にヤバそうなところ
    - ・ 負債になりそうなところ
  - ・ 機械に見てほしいところ
    - ・ テストに通るか
    - ・ コードのフォーマット (インデント崩れなど)



## Pro Tip 💡

---

- ・実装者の心理的負担を下げる：
  - ・指摘の重要度をわかりやすくする
    - ・そのままリリースすると死ぬ
    - ・直しておいた方があとで十中八九ハッピーになれる
  - ・自分なら別の方法で実装するけど、どちらが正解かは微妙

# 実習① GitHub Flow 体験

# 実習① GitHub Flow 体験

---

- やったこと
  - リモートリポジトリの追加 (git remote)
  - ローカルリポジトリとの同期 (git push / pull)
  - ブランチを切ってコミット (git branch / commit)
  - GitHub 上でのレビュー&マージ

Git をしくみから理解する

# Git をしくみから理解する

---

- ・ Git 、ふわっとした理解になりがち問題

# Git をしくみから理解する

---

- Git 、 ふわっとした理解になりがち問題
  - コミットって前回のバージョンからの差分のことでしょ
  - ブランチって枝分かれのことでしょ

# Git をしくみから理解する

---

- Git、ふわっとした理解になりがち問題
  - コミットって前回のバージョンからの差分のことでしょ
  - ブランチって枝分かれのことでしょ

ちがう

とは？

「ブランチを削除」したら  
具体的にどんなデータがディスクから消える？

# Git をしくみから理解する

---

- Git 、ふわっとした理解になりがち問題
  - コミットって前回のバージョンからの差分のことでしょ
  - ブランチって枝分かれのことでしょ
- 普通使いには問題なくても、複雑な操作をするとわからなくなる
  - rebase したら意図しないコミットログに… とか



# Git をしくみから理解する

---

- このコーナーの目標
  - Git の基本機能を「自信を持って」使える
    - commit
    - branch
    - rebase
    - merge

# Git をしくみから理解する

---

- このコーナーの目標
  - Git の基本機能を「自信を持って」使える
    - commit
    - branch
    - rebase
    - merge
  - 実用上の小ワザなどは実習でググりながら練習します

内部データを覗いてみる

# 内部データを覗いてみる

---

- Git の (主な) 登場人物は四人
  - Commit
  - Tree
  - Blob
  - Ref (Branch, Tag など)

# 内部データを覗いてみる

---

- Git の (主な) 登場人物は四人
  - Commit
  - Tree
  - Blob
  - Ref (Branch, Tag など)

# 内部データを覗いてみる

---

- ・ Commit ... Git でもっとも重要なオブジェクト

# 内部データを覗いてみる

---

- Commit ... Git でもっとも重要なオブジェクト
  - 固有の ID で管理されている

```
$ git show
commit 7fb395858e7aa6fe5d5f2bfabcb9231ae589f9be (HEAD -> master)
Author: kota.nara <xxxxxxx@xxxxxxx.xx>
Date:   Mon Mar 18 18:01:08 2019 +0900

    hoge
```

# 内部データを覗いてみる

---

- ・ 実は ID がそのまま内部ファイルのパスになっている

```
$ cat .git/objects/7f/b395858e7aa6fe5d5f2bfabcd92321ae589f9be
```



# 内部データを覗いてみる

---

- ・ 実は ID がそのまま内部ファイルのパスになっている

```
$ cat .git/objects/7f/b395858e7aa6fe5d5f2bfabcd92321ae589f9be
x\xI
1=\x
\xwO\x%
```

- ・ zlib 圧縮されている

# 内部データを覗いてみる

---

- ・ 伸長してみた

```
$ cat .git/objects/7f/b395... | zlib --decompress
commit 257tree 4b358f9a7387a2e8ebcb4e335bbd4a79fab91443
parent d193ec366176225f60320865026ab3b8f3998b32
author kota.nara <xxxxxxx@xxxxxxx.xx> 1552899668 +0900
committer kota.nara <xxxxxxx@xxxxxxx.xx> 1552899668 +0900

hoge
```

# 内部データを覗いてみる

---

- ・ 伸長してみた

```
$ cat .git/objects/7f/b395... | zlib --decompress  
commit 257tree 4b358f9a7387a2e8ebcb4e335bbd4a79fab91443  
parent 7f/b395...  
author k... 1552899668 +0900  
committer kota.nara <xxxxxx@xxxxxx.xx> 1552899668 +0900  
  
hoge
```

これは 257 バイトのコミットオブジェクトですよ

# 内部データを覗いてみる

---

- ・ 伸長してみた

```
$ cat .git/objects/7f/b395... | zlib --decompress  
commit 257tree 4b358f9a7387a2e8ebcb4e335bbd4a79fab91443  
parent d193ec366176225f603208650  
author kota.nara <xxxxxxx@xxxxxxx>  
committer kota.nara <xxxxxxx@xxxxxxx.xx> 1552899668 +0900  
  
hoge
```

tree オブジェクトの ID (後述)

# 内部データを覗いてみる

---

- ・ 伸長してみた

```
$ cat .git/objects/7f/b395... | zlib --decompress
commit 257tree 4b358f9a7387a2e8ebcb4e335bbd4a79fab91443
parent d193ec366176225f60320865026ab3b8f3998b32
author kota.nara <xxxxxxx@xxxxxxx.xxxxx> 00
committer kota.nara <xxxxxxx@xxxxxxx> +0900

hoge
```

親コミットの ID

# 内部データを覗いてみる

---

- ・ 伸長してみた

```
$ cat .git/objects/7f/b395... | zlib --decompress  
commit 257tree 4b358f9a7387a2e8ebcb4e335bbd4a79fab91443  
parent d193ec366176225f60320865026ab3b8f3998b32  
author kota.nara <xxxxxxx@xxxxxxx.xx> 1552899668 +0900  
committer kota.nara <xxxxxxx@xxxxxxx.xx> 1552899668 +0900
```

hoge

コミット日時と署名

# 内部データを覗いてみる

---

- ・ 伸長してみた

```
$ cat .git/objects/7f/b395... | zlib --decompress  
commit 257tree 4b358f9a7387a2e8ebcb4e335bbd4a79fab91443  
parent d193ec366176225f60320865026ab3b8f3998b32  
author kota.nara <xxxxxxx@xxxxxxx.xx> 1552899668 +0900  
committer kota.nara <xxxxxxx@xxxxxxx.xx> 1552899668 +0900
```

hoge

コミットメッセージ

# 内部データを覗いてみる

---

- ・ コミットオブジェクトの中身
  - ・ Tree オブジェクトの ID
  - ・ 親コミットの ID
  - ・ 日時と署名
  - ・ メッセージ



# 内部データを覗いてみる

---

- ・ コミットオブジェクトの中身
  - ・ Tree オブジェクトの ID
  - ・ 親コミットの ID
  - ・ 日時と署名
  - ・ メッセージ
- ・ 肝心のバックアップの中身は Tree オブジェクトにある

# 内部データを覗いてみる

---

- ・ ちなみに

# 内部データを覗いてみる

---

- ・ ちなみに

```
$ cat .git/objects/7f/b395... | zlib --decompress | shasum  
7fb395858e7aa6fe5d5f2bfabcb9231ae589f9be
```

- ・ コミットの ID はコミットオブジェクトの SHA1 ハッシュ

# 内部データを覗いてみる

---

- ・ ちなみに

```
$ cat .git/objects/7f/b395... | zlib --decompress | shasum  
7fb395858e7aa6fe5d5f2bfabcb9231ae589f9be
```

- ・ コミットの ID はコミットオブジェクトの SHA1 ハッシュ
- ・ つまり ...

# 内部データを覗いてみる

---

- ・ コミットオブジェクトの中身
  - ・ Tree オブジェクトの ID
  - ・ 親コミットの ID
  - ・ 日時と署名
  - ・ メッセージ
- ・ どれか一つでも違えば、必ず別の ID になる

# 内部データを覗いてみる

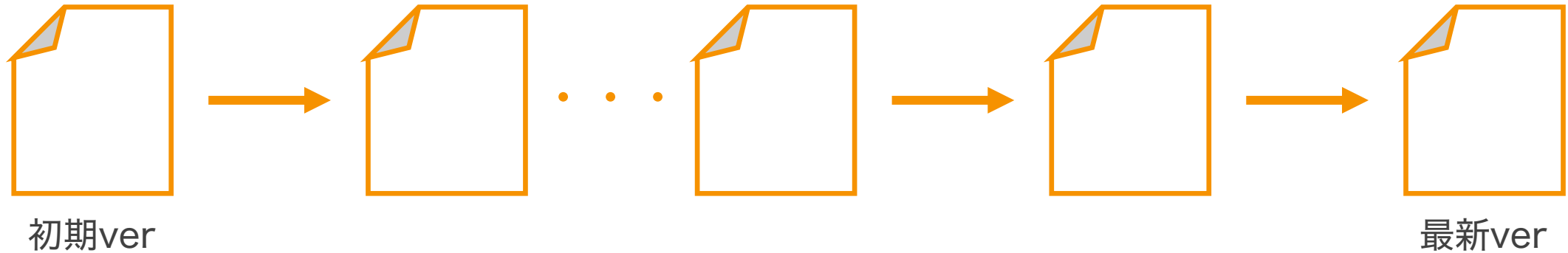
---

- ・ コミットオブジェクトの中身
  - ・ Tree オブジェクトの ID
  - ・ 親コミットの ID
  - ・ 日時と署名
  - ・ メッセージ
- ・ どれか一つでも違えば、必ず別の ID になる
- ・ 改竄がないことが保証される (cf. ブロックチェーン)

# 内部データを覗いてみる

---

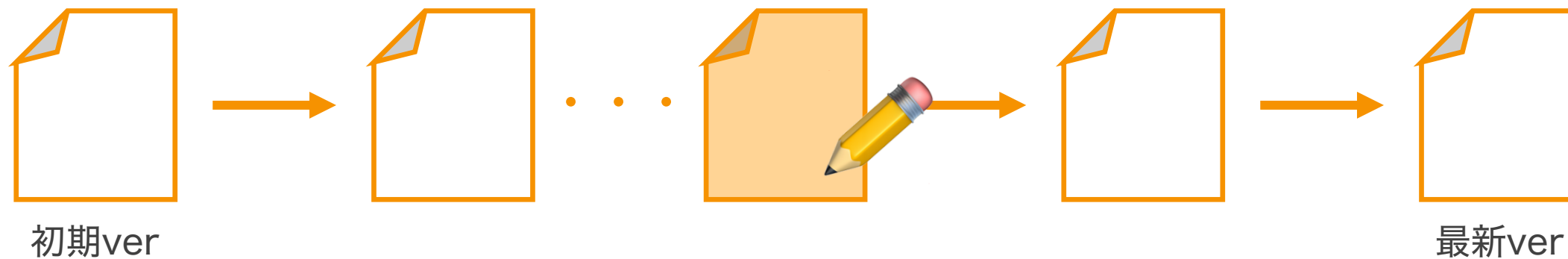
- ・改竄がないことが保証される (cf. ブロックチェーン)



# 内部データを覗いてみる

---

- ・改竄がないことが保証される (cf. ブロックチェーン)

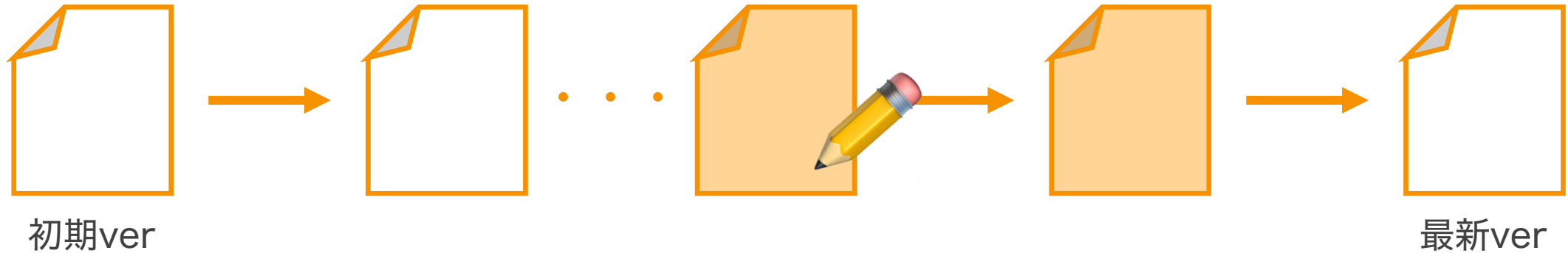




# 内部データを覗いてみる

---

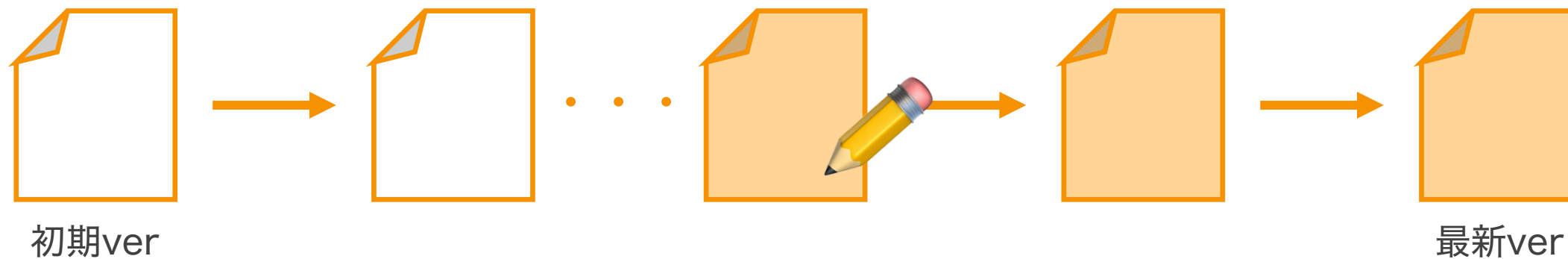
- ・改竄がないことが保証される (cf. ブロックチェーン)



# 内部データを覗いてみる

---

- ・改竄がないことが保証される (cf. ブロックチェーン)



Tree, Blob オブジェクト

# Tree, Blob オブジェクト

---

- Git の (主な) 登場人物は四人
  - Commit
  - Tree
  - Blob
  - Ref (Branch, Tag など)

# Tree, Blob オブジェクト

---

- ・ 肝心のバックアップデータが入っているオブジェクト

# Tree, Blob オブジェクト

---

- ・ 肝心のバックアップデータが入っているオブジェクト

```
$ cat .git/objects/4b/358f... | zlib --decompress
tree 103100644 Readme.markdown¥(¥¥¥¥¥
                                     ¶V(d¥¥_40000
doc¥5¥¥¥¥|¥¥¥¥T¥¥¥6¥¥¥¥$¥}40000 src¥¥¥¥g¥¥¥m¥¥¥t2v"S?¥¥¥^%
```

# Tree, Blob オブジェクト

---

- ・ 肝心のバックアップデータが入っているオブジェクト

```
$ cat .git/objects/4b/358f... | zlib --decompress
tree 103100644 Readme.markdown\x00\x00\x00\x00
                                     \x00V(d\x00\x00_40000
doc\x005\x00\x00|\x00\x00T\x00\x006\x00\x00$\x00}40000 src\x00\x00g\x00\x00m\x00\x00t2v"S?\x00\x00^%
```

# Tree, Blob オブジェクト

---

- ・ 肝心のバックアップデータが入っているオブジェクト

```
$ cat .git/objects/4b/358f... | zlib -decompress | xxd
00000000: 7472 6565 2031 3033 0031 3030 3634 3420  tree 103.100644
00000010: 5265 6164 6d65 2e6d 6172 6b64 6f77 6e00  Readme.markdown.
00000020: e9bc 1102 5c28 829e edf6 d30c d3b6 5628  ....¥(.....V(
00000030: 648c ad5f 3430 3030 3020 646f 6300 01fa  d.._40000 doc...
00000040: 358e b5e2 7ca2 e5f0 5486 a436 85e5 f524  5...|...T..6...$
00000050: 957d 3430 3030 3020 7372 6300 8a92 ef67  .}40000 src....g
00000060: eff9 6de1 0ff7 7432 7622 533f c7d5 125e  ..m...t2v"S?...^
```



# Tree, Blob オブジェクト

- 肝心のバックアップデータが入っているオブジェクト

```
$ cat .git/objects/4b/358f... | zlib -decompress | xxd
00000000: 7472 6565 2031 3033 0031 3030 3634 3420 tree 103.100644
00000010: 5265 6164 6d65 2e6d 6172 6b65 2031 3033
00000020: e9bc 1102 5c28 829e edf6 d309 2031 3033
00000030: 648c ad5f 3430 3030 3020 646f 6300 01fa d..._40000 doc...
00000040: 358e b5e2 7ca2 e5f0 5486 a436 85e5 f524 5...|...T..6...$
00000050: 957d 3430 3030 3020 7372 6300 8a92 ef67 .}40000 src....g
00000060: eff9 6de1 0ff7 7432 7622 533f c7d5 125e ..m...t2v"S?...^
```

これは 103 バイトの Tree オブジェクトですよ

# Tree, Blob オブジェクト

- 肝心のバックアップデータが入っているオブジェクト

```
$ cat .git/objects/4b/358f... | zlib -decompress | xxd
00000000: 7472 6565 2031 3033 0031 3030 3634 3420  tree 103.100644
00000010: 5265 6164 6d65 2e6d 6172 6b64 6f77 6e00  Readme.markdown.
00000020: e9bc 1102 5c28 829e edf6 d30c d3b6 5
00000030: 648c ad5f 3430 3030 3020 646f 6300 6
00000040: 358e b5e2 7ca2 e5f0 5486 a436 85e5 f524  5...|...T..6...$
00000050: 957d 3430 3030 3020 7372 6300 8a92 ef67  .}40000 src....g
00000060: eff9 6de1 0ff7 7432 7622 533f c7d5 125e  ..m...t2v"S?...^
```

一つ目のファイル名とパーミッション

# Tree, Blob オブジェクト

- ・ 肝心のバックアップデータが入っているオブジェクト

```
$ cat .git/objects/4b/358f... | zlib -decompress | xxd
00000000: 7472 6565 2031 3033 0031 3030 3634 3420  tree 103.100644
00000010: 5265 6164 6d65 2e6d 6172 6b64 6f77 6e00  Readme.markdown.
00000020: e9bc 1102 5c28 829e edf6 d30c d3b6 5628  ....¥(.....V(
00000030: 648c ad5f 3430 3030 3030 343f 3030 31fa  d..._40000 doc...
00000040: 358e b5e2 3430 3030 3030 343f 3030 31fa  5...|...T..6...$
00000050: 957d 3430 3030 3030 3030 343f 3030 31fa  .}40000 src....g
00000060: eff9 6de1 0ff7 7432 7622 533f c7d5 125e  ..m...t2v"S?...^
```

Git オブジェクトの ID (後述)

# Tree, Blob オブジェクト

- 肝心のバックアップデータが入っているオブジェクト

```
$ cat .git/objects/4b/358f... | zlib -decompress | xxd
00000000: 7472 6565 2031 3033 0031 3030 3634 3420  tree 103.100644
00000010: 5265 6164 6d65 2e6d 6172 6b64 6f77 6e00  Readme.markdown.
00000020: e9bc 1102 5c28 829e edf6 d30c d3b6 5628  ....¥(.....V(
00000030: 648c ad5f 3430 3030 3020 646f 6300 01fa  d.._40000 doc...
00000040: 358e b5e2 7ca2 e5f0 5486 a436 85e5 f500  =58=5486a43685e5f500
00000050: 957d 3430 3030 3020 7372 6300 8a92 e500  _957d343030303020737263008a92e500
00000060: eff9 6de1 0ff7 7432 7622 533f c7d5 125e  ..m...t2v"S?...^
```

二つ目のファイル名とパーミッション

# Tree, Blob オブジェクト

- 肝心のバックアップデータが入っているオブジェクト

```
$ cat .git/objects/4b/358f... | zlib -decompress | xxd
00000000: 7472 6565 2031 3033 0031 3030 3634 3420  tree 103.100644
00000010: 5265 6164 6d65 2e6d 6172 6b64 6f77 6e00  Readme.markdown.
00000020: e9bc 1102 5c28 829e edf6 d30c d3b6 5628  ....¥(.....V(
00000030: 648c ad5f 3430 3030 3020 646f 6300 01fa  d.._40000 doc...
00000040: 358e b5e2 7ca2 e5f0 5486 a436 85e5 f524  5...|...T..6...$
00000050: 957d 3430 3030 3030 3030 3030 3030 3030  .}40000 src....g
00000060: eff9 6de1 0ff7 0ff7 0ff7 0ff7 0ff7 0ff7  ..m...t2v"S?...^
```

Git オブジェクトの ID (後述)

# Tree, Blob オブジェクト

- 肝心のバックアップデータが入っているオブジェクト

```
$ cat .git/objects/4b/358f... | zlib -decompress | xxd
00000000: 7472 6565 2031 3033 0031 3030 3634 3420  tree 103.100644
00000010: 5265 6164 6d65 2e6d 6172 6b64 6f77 6e00  Readme.markdown.
00000020: e9bc 1102 5c28 829e edf6 d30c d3b6 5628  ....¥(.....V(
00000030: 648c ad5f 3430 3030 3020 646f 6300 01fa  d.._40000 doc...
00000040: 358e b5e2 7ca2 e5f0 5486 a436 85e5 f524  5...|...T..6...$
00000050: 957d 3430 3030 3020 7372 6300 8a92 ef67  .}40000 src....g
00000060: eff9 6de1 0ff7 7432 7622 533f c7d5 125...
```

最後のファイル名とパーミッション

# Tree, Blob オブジェクト

- ・ 肝心のバックアップデータが入っているオブジェクト

```
$ cat .git/objects/4b/358f... | zlib -decompress | xxd
00000000: 7472 6565 2031 3033 0031 3030 3634 3420  tree 103.100644
00000010: 5265 6164 6d65 2e6d 6172 6b64 6f77 6e00  Readme.markdown.
00000020: e9bc 1102 5c28 829e edf6 d30c d3b6 5628  ....¥(.....V(
00000030: 648c ad5f 3430 3030 3020 646f 6300 01fa  d.._40000 doc...
00000040: 358e b5e2 7ca2 e5f0 5486 a436 85e5 f524  5...|...T..6...$
00000050: 957d 3430 3030 3020 7372 6300 8a92 ef67  .}40000 src....g
00000060: eff9 6de1 0ff7 7432 7622 533f c7d5 125e  ..m...t2v"S?...^
```

Git オブジェクトの ID (後述)

# Tree, Blob オブジェクト

---

- Tree オブジェクト
  - ファイルの一覧が入っている
  - パーミッション、ファイル名、 Git オブジェクト ID
- 今回見つけたのは：
  - 100644 Readme.markdown e9bc1102...
  - 040000 doc 01fa3582...
  - 040000 src 8a92ef67...



## Commit

Tree ...

親 ...

署名 ...

メッセージ ...

d193ec

## Tree

Readme.markdown e9bc1102...

doc 01fa3582...

src 8a92ef67...

4b358f

## Commit

Tree 4b358f...

親 d193ec...

署名 kota.nara

メッセージ hoge

7fb395

# Tree, Blob オブジェクト

---

- "Readme.markdown" の Git オブジェクトを試してみる

```
$ cat .git/objects/e9/bc1102... | zlib --decompress
```

```
blob 36# hoge
```

```
This is a hogehoge project
```

# Tree, Blob オブジェクト

---

- "Readme.markdown" の Git オブジェクトをしてみる

```
$ cat .git/objects/e9/bc1102... | zlib --decompress  
blob 36# hoge
```

T

これは 36 バイトの Blob オブジェクトですよ

# Tree, Blob オブジェクト

---

- "Readme.markdown" の Git オブジェクトを見してみる

```
$ cat .git/objects/e9/bc1102... | zlib --decompress  
blob 36# hoge
```

This is a hogehoge project

Readme.markdown の内容

# Tree, Blob オブジェクト

---

- "Readme.markdown" の Git オブジェクトを試してみる

```
$ cat .git/objects/e9/bc1102... | zlib --decompress
```

```
blob 36# hoge
```

```
This is a hogehoge project
```

- Blob オブジェクトにはファイルの中身が入っている

## Commit

Tree ...

親 ...

署名 ...

メッセージ ...

d193ec

## Tree

Readme.markdown e9bc1102...

doc 01fa3582...

src 8a92ef67...

4b358f

## Blob

...

e9bc1102

## Commit

Tree 4b358f...

親 d193ec...

署名 kota.nara

メッセージ hoge

7fb395

# Tree, Blob オブジェクト

---

- ”doc” の Git オブジェクトも見てみる

```
$ cat .git/objects/01/fa358e... | zlib --decompress  
tree 48100644 hoge hogedoc.markdown🚢CK)wZSS%
```

# Tree, Blob オブジェクト

---

- "doc" の Git オブジェクトも見てみる

```
$ cat .git/objects/01/fa358e... | zlib --decompress  
tree 48100644 hoge hogedoc.markdown 🚢 CK) wZ s %
```

!!!!



# Tree, Blob オブジェクト

---

- Tree オブジェクトの中身
  - ファイル → Blob オブジェクト
  - ディレクトリ → Tree オブジェクト

## Commit

Tree ...

親 ...

署名 ...

メッセージ ...

d193ec

## Commit

Tree 4b358f...

親 d193ec...

署名 kota.nara

メッセージ hoge

7fb395

## Tree

Readme.markdown e9bc1102...

doc 01fa3582...

src 8a92ef67...

4b358f

## Blob

...

e9bc1102

## Tree

hoge hogedoc.markdown e69de29b...

01fa3582

## Commit

Tree ...

親 ...

署名 ...

メッセージ ...

d193ec

## Commit

Tree 4b358f...

親 d193ec...

署名 kota.nara

メッセージ hoge

7fb395

## Tree

Readme.markdown e9bc1102...

doc 01fa3582...

src 8a92ef67...

4b358f

## Blob

...

e9bc1102

## Tree

hogehogedoc.markdown e69de29b...

01fa3582

## Blob

...

e69de29b

## Tree

hogehoge.js 4a83cbe2...

8a92ef67

## Blob

...

4a83cbe2

# Tree, Blob オブジェクト

---

- ・ Tree, Blob が独自のファイルシステムを形成している

# Tree, Blob オブジェクト

---

- Tree, Blob が独自のファイルシステムを形成している
  - たんに前のコミットとの diff を保存しているわけではない

# Tree, Blob オブジェクト

---

- ・ ちなみに
  - ・ Tree, Blob オブジェクトも ID は SHA1 ハッシュ

# Tree, Blob オブジェクト

---

- ちなみに
  - Tree, Blob オブジェクトも ID は SHA1 ハッシュ
- ルートの Tree ID だけでディレクトリ全体の内容を保証できる

## Commit

Tree ...

親 ...

署名 ...

メッセージ ...

d193ec

## Commit

Tree 4b358f...

親 d193ec...

署名 kota.nara

メッセージ hoge

7fb395

## Tree

Readme.markdown e9bc1102...

doc 01fa3582...

src 8a92ef67...

4b358f

## Blob

...

e9bc1102

## Tree

hogehogedoc.markdown e69de29b...

01fa3582

## Blob

...

e69de29b

## Tree

hogehoge.js 4a83cbe2...

8a92ef67

## Blob

...

4a83cbe2



## Commit

Tree ...

親 ...

署名 ...

メッセージ ...

d193ec

## Commit

Tree 4b358f...

親 d193ec...

署名 kota.nara

メッセージ hoge

7fb395

## Tree

Readme.markdown e9bc1102...

doc 01fa3582...

src 8a92ef67...

4b358f

## Blob

...

e9bc1102

## Tree

hogehogedoc.markdown e69de29b...

01fa3582

## Blob

...

e69de29b

## Tree

hogehoge.js 4a83cbe2...

8a92ef67

## Blob

...

4a83cbe2

## Commit

Tree ...

親 ...

署名 ...

メッセージ ...

d193ec

## Commit

Tree 4b358f...

親 d193ec...

署名 kota.nara

メッセージ hoge

7fb395

## Tree

Readme.markdown e9bc1102...

doc 01fa3582...

src 8a92ef67...

4b358f

## Blob

...

e9bc1102

## Tree

hogehogedoc.markdown xxxxxxxx...

01fa3582

## Blob

...

xxxxxxxx

## Tree

hogehoge.js 4a83cbe2...

8a92ef67

## Blob

...

4a83cbe2

## Commit

Tree ...

親 ...

署名 ...

メッセージ ...

d193ec

## Commit

Tree 4b358f...

親 d193ec...

署名 kota.nara

メッセージ hoge

7fb395

## Tree

Readme.markdown e9bc1102...

doc xxxxxxxx...

src 8a92ef67...

4b358f

## Blob

...

e9bc1102

## Tree

hogehogedoc.markdown xxxxxxxx...

xxxxxxx

## Blob

...

xxxxxxx

## Tree

hogehoge.js 4a83cbe2...

8a92ef67

## Blob

...

4a83cbe2

**Commit**

Tree ...

親 ...

署名 ...

メッセージ ...

d193ec

**Commit**

Tree xxxxxxxx...

親 d193ec...

署名 kota.nara

メッセージ hoge

7fb395

**Tree**

Readme.markdown e9bc1102...

doc xxxxxxxx...

src 8a92ef67...

xxxxxxx

**Blob**

...

e9bc1102

**Tree**

hogehogedoc.markdown xxxxxxxx...

xxxxxxx

**Blob**

...

xxxxxxx

**Tree**

hogehoge.js 4a83cbe2...

8a92ef67

**Blob**

...

4a83cbe2

**Commit**

Tree ...

親 ...

署名 ...

メッセージ ...

d193ec

**Commit**

Tree xxxxxxxx...

親 d193ec...

署名 kota.nara

メッセージ hoge

xxxxxxx

**Tree**

Readme.markdown e9bc1102...

doc xxxxxxxx...

src 8a92ef67...

xxxxxxx

**Blob**

...

e9bc1102

**Tree**

hogehogedoc.markdown xxxxxxxx...

xxxxxxx

**Blob**

...

xxxxxxx

**Tree**

hogehoge.js 4a83cbe2...

8a92ef67

**Blob**

...

4a83cbe2

# Tree, Blob オブジェクト

---

- ちなみに
  - Tree, Blob オブジェクトも ID は SHA1 ハッシュ
- ルートの Tree ID だけでディレクトリ全体の内容を保証できる

# Tree, Blob オブジェクト

---

- ・ ちなみに
  - ・ Tree, Blob オブジェクトも ID は SHA1 ハッシュ
- ・ ルートの Tree ID だけでディレクトリ全体の内容を保証できる
- ・ 内容が同じオブジェクトは同じ ID になって区別されない
  - 重複してバックアップする必要がない

Ref



# Ref

---

- Git の (主な) 登場人物は四人
  - Commit
  - Tree
  - Blob
  - Ref (Branch, Tag など)

# Ref

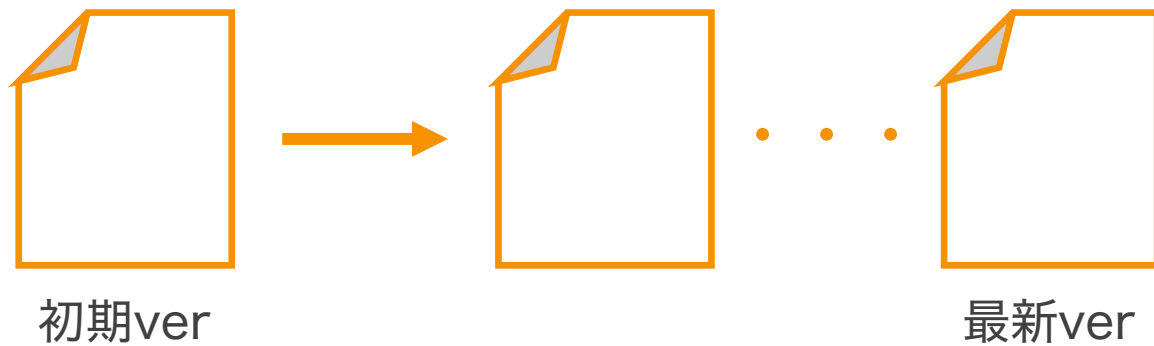
---

- ・コミットにタグをつける

# Ref

---

- ・コミットにタグをつける



# Ref

---

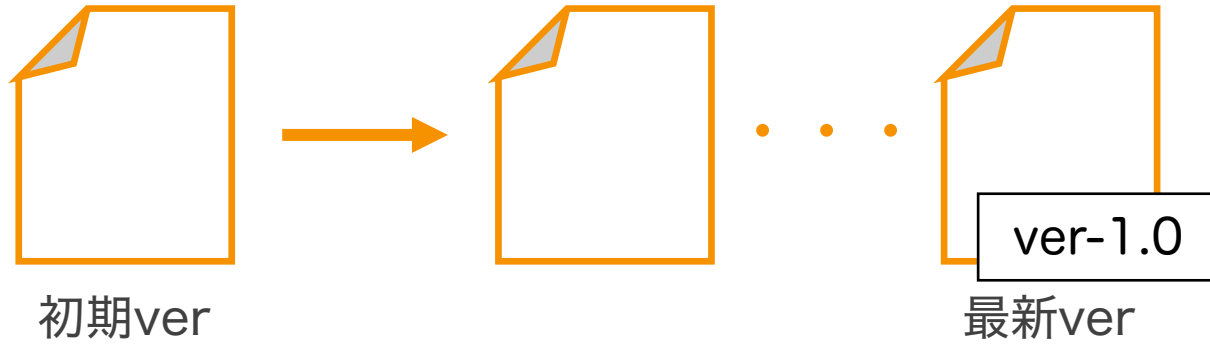
- ・コミットにタグをつける

```
$ git tag ver-1.0
```

# Ref

---

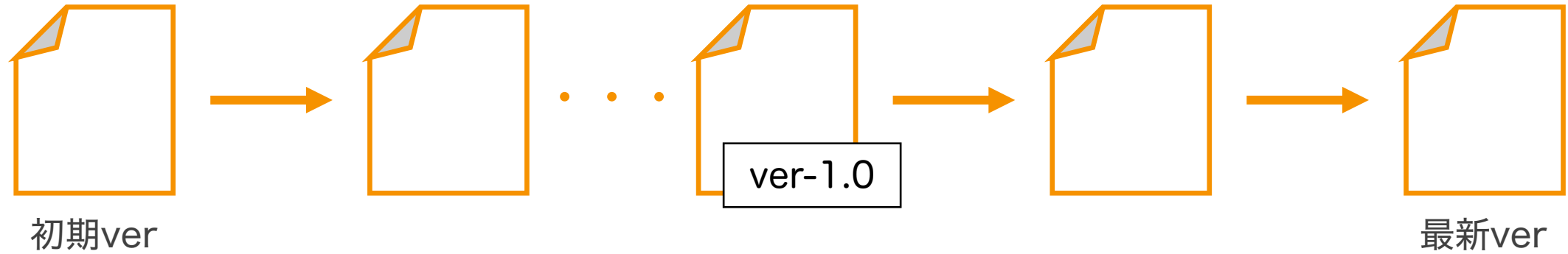
- ・ コミットにタグをつける



# Ref

---

- ・ コミットにタグをつける



# Ref

---

- ・コミットにタグをつける

```
$ git checkout ver-1.0
```

- ・いつでもコミットをタグ名で呼び出せる

# Ref

---

- ・ タグの実体はシンプル



# Ref

---

- ・タグの実体はシンプル

```
$ cat .git/refs/tags/ver-1.0  
c5d57bc23a343c226e4970835444c6bd16829878
```

- ・コミットの ID が書いてあるだけのテキストファイル

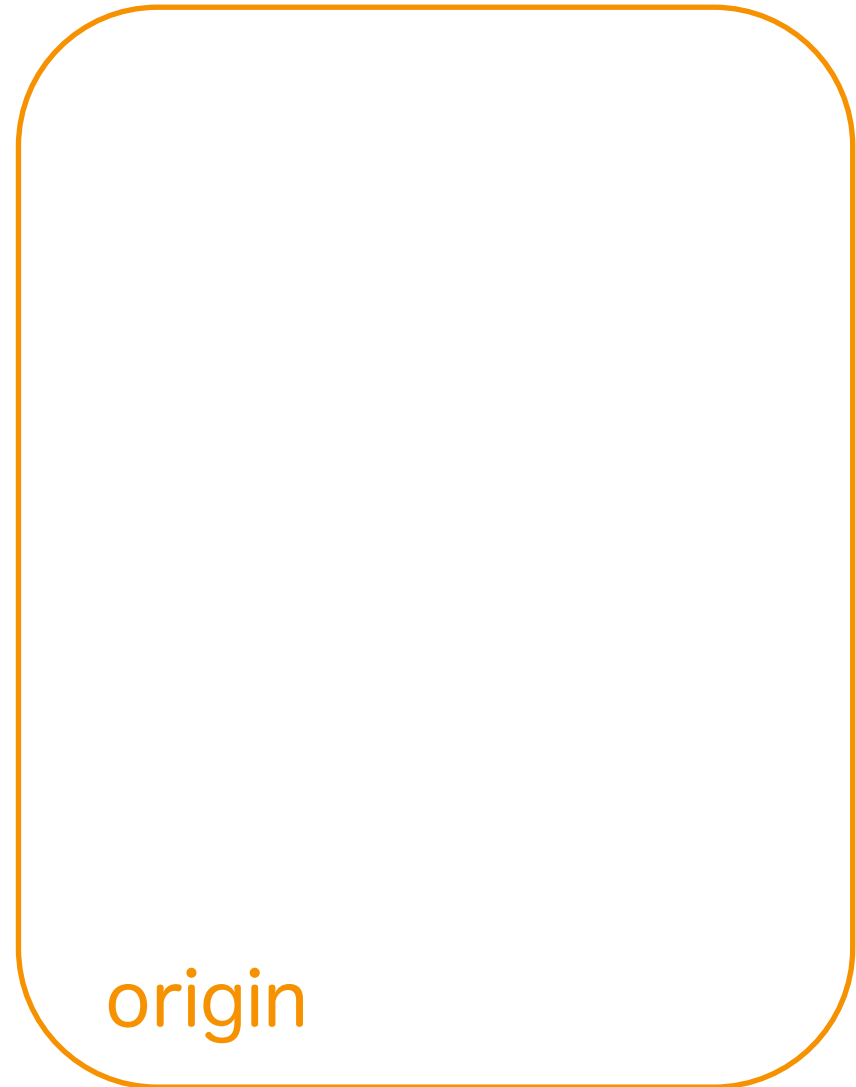
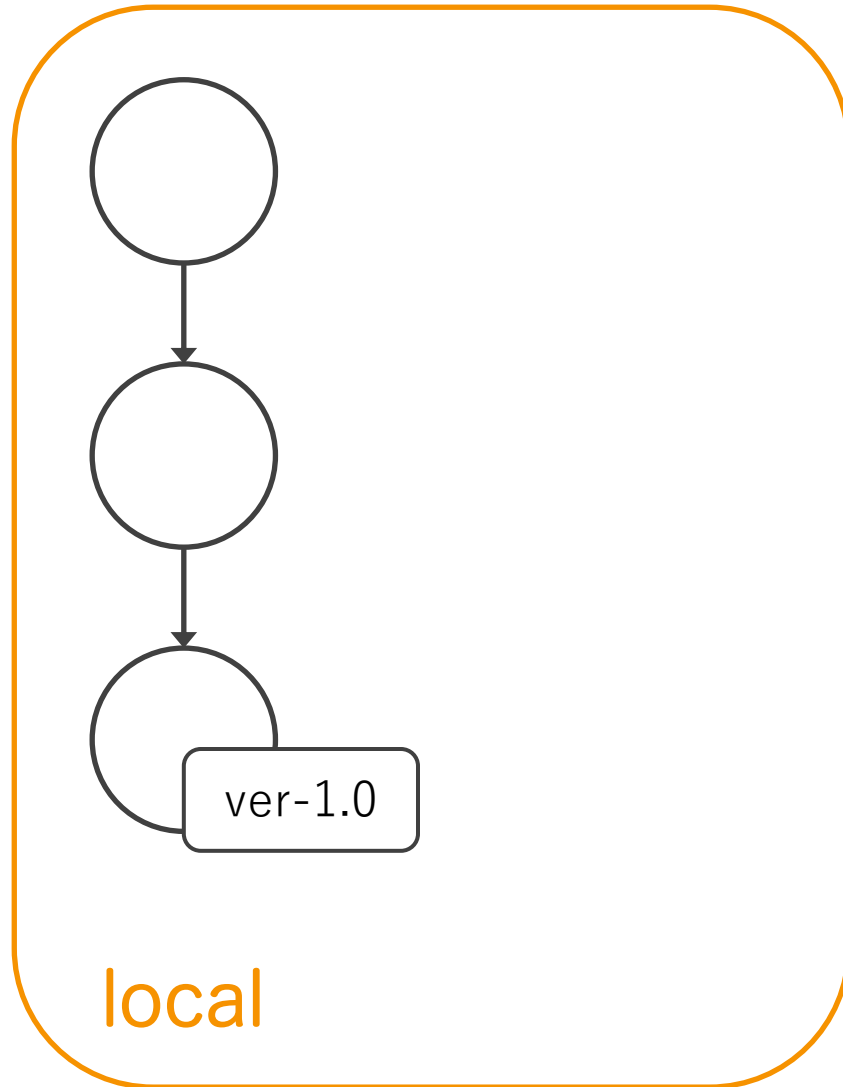
# Ref

---

- Ref の push

# Ref

---



# Ref

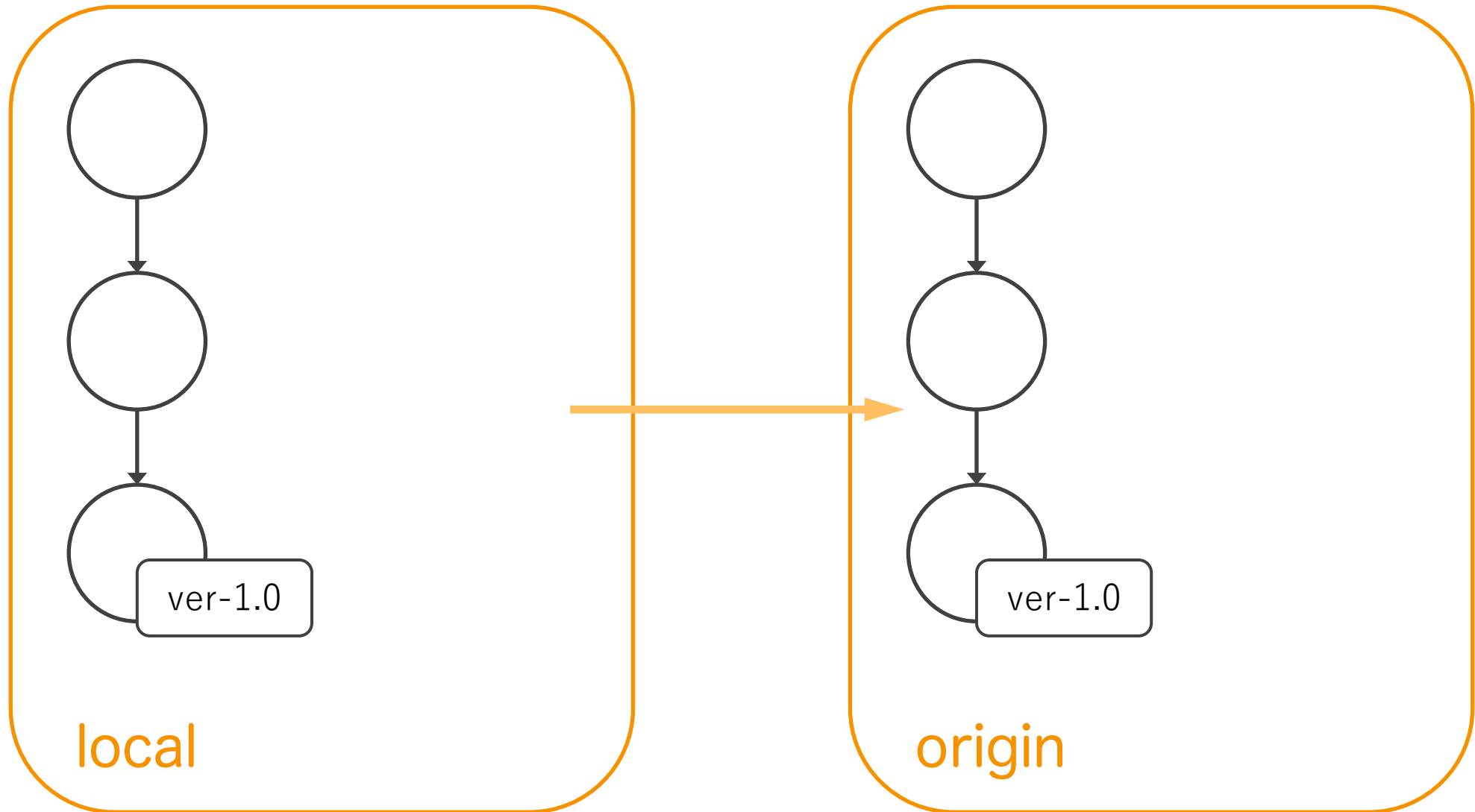
---

- Ref の push

```
$ git push origin ver-1.0
```

# Ref

---



# Ref

---

- Ref の push

```
$ git push origin ver-1.0
```

- ref と、コミットと、その祖先がアップロードされる
  - GitHub (など) 上で確認したり、他のマシンから DL できる

Branch

# Branch

---

- ・ ブランチも Ref の一種



# Branch

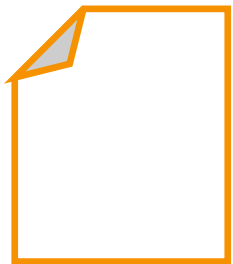
---

- ・ ブランチも Ref の一種
  - ・ タグはコミットを追加しても移動しない
  - ・ ブランチはコミットを追加すると追従する

# Branch

---

- ・ ブランチも Ref の一種



初期ver

# Branch

---

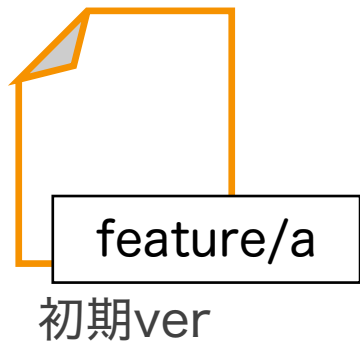
- ・ ブランチも Ref の一種

```
$ git branch feature/a  
$ git checkout feature/a
```

# Branch

---

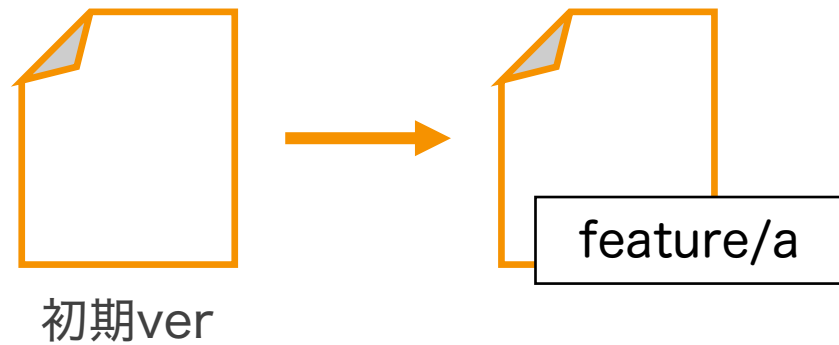
- ・ ブランチも Ref の一種



# Branch

---

- ・ ブランチも Ref の一種



# Branch

---

- ・何が嬉しいの？

# Branch

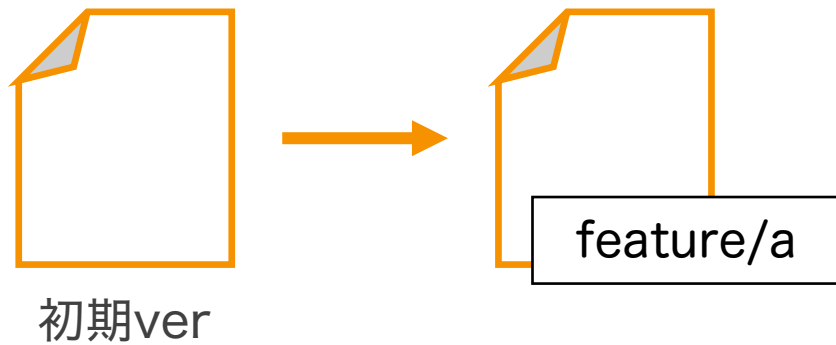
---

- ・何が嬉しいの？
  - ・複数の作業を並行してやる時に便利

# Branch

---

- ・何が嬉しいの？

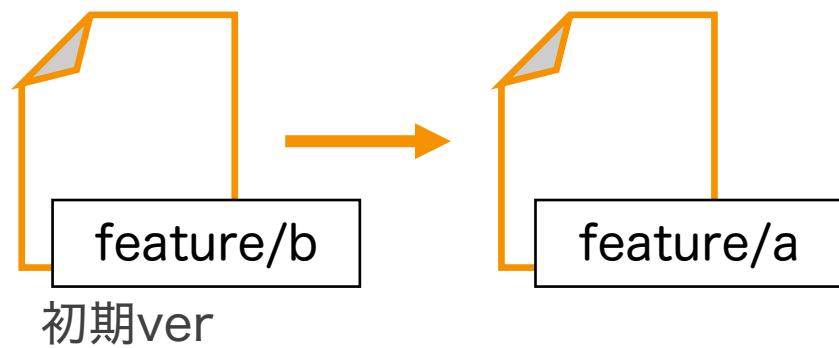




# Branch

---

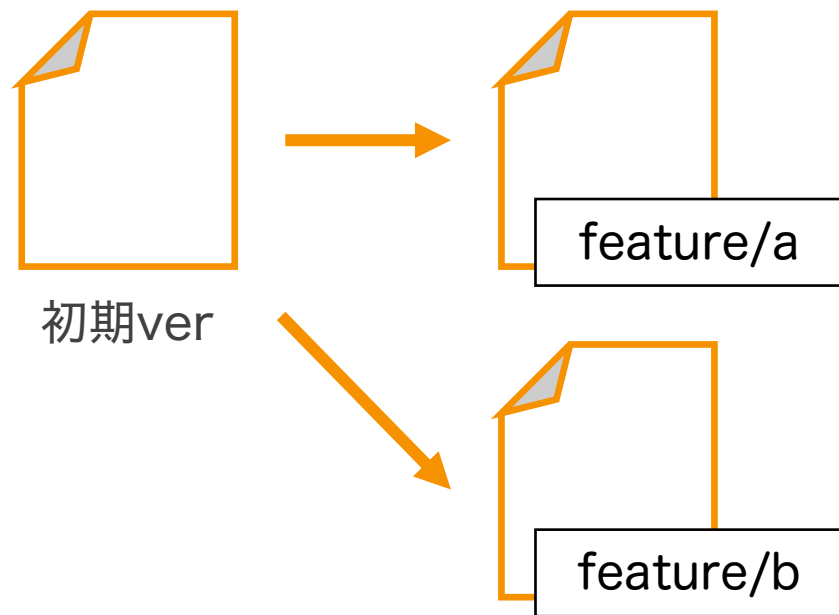
- ・何が嬉しいの？



# Branch

---

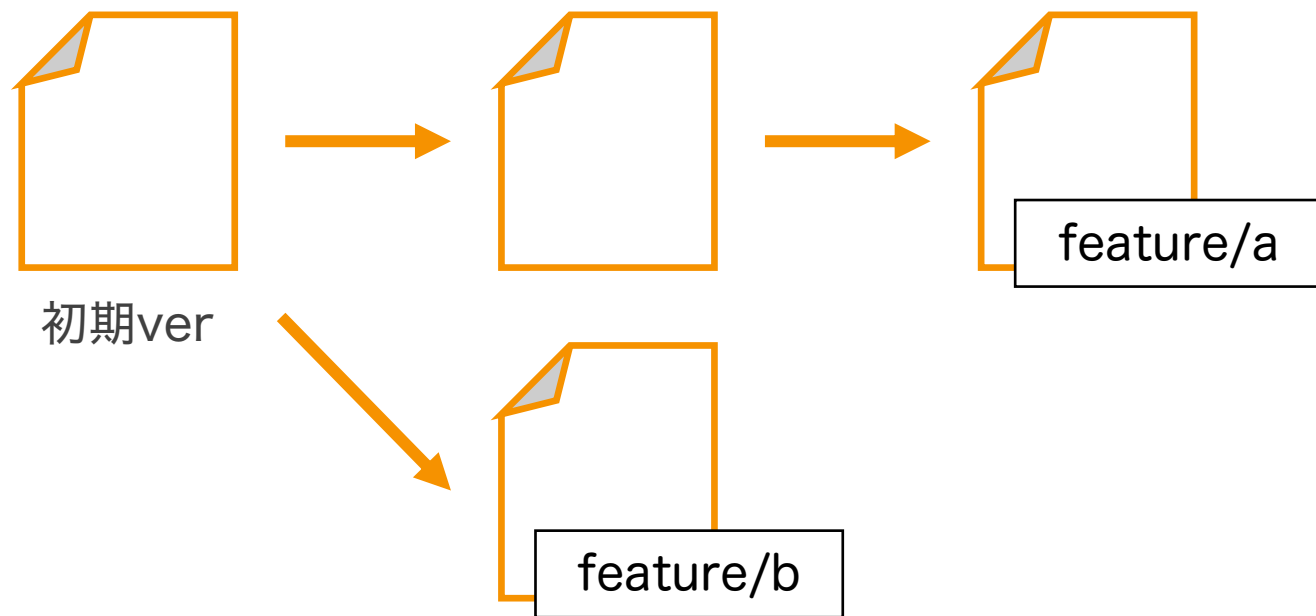
- ・何が嬉しいの？



# Branch

---

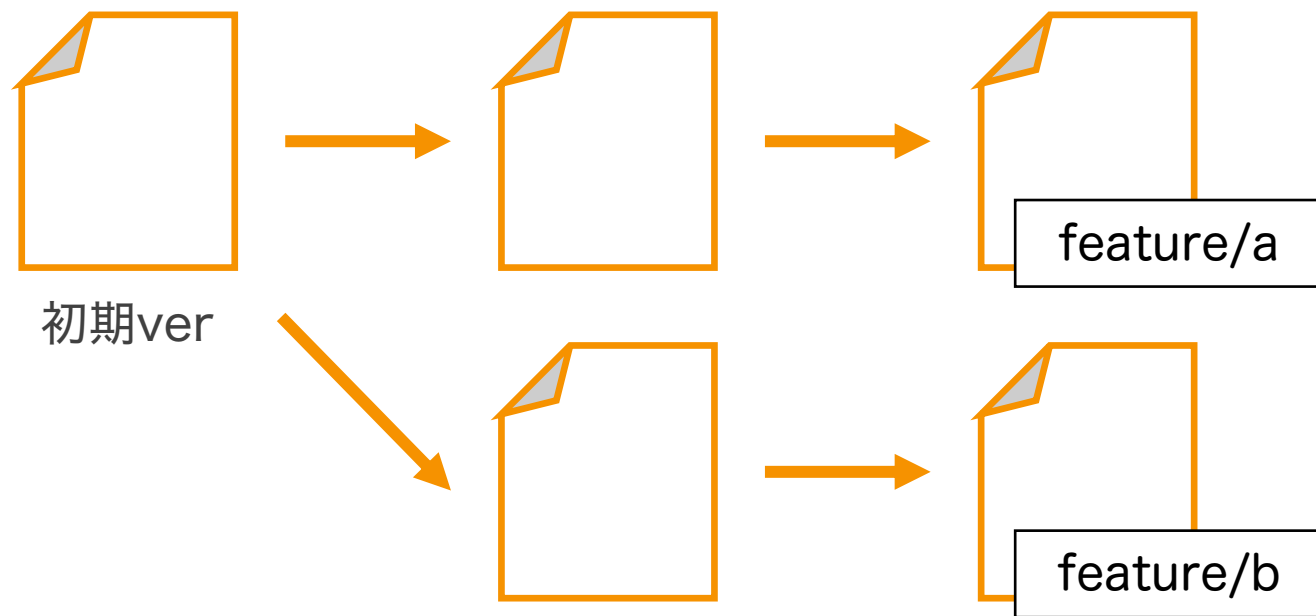
- ・何が嬉しいの？



# Branch

---

- ・何が嬉しいの？



# Branch

---

- ・ Branch の実体

# Branch

---

- Branch の実体

```
$ cat .git/refs/heads/feature/a  
c5d57bc23a343c226e4970835444c6bd16829878
```

- やっぱりコミット ID が書いてあるだけのテキストファイル

# Branch

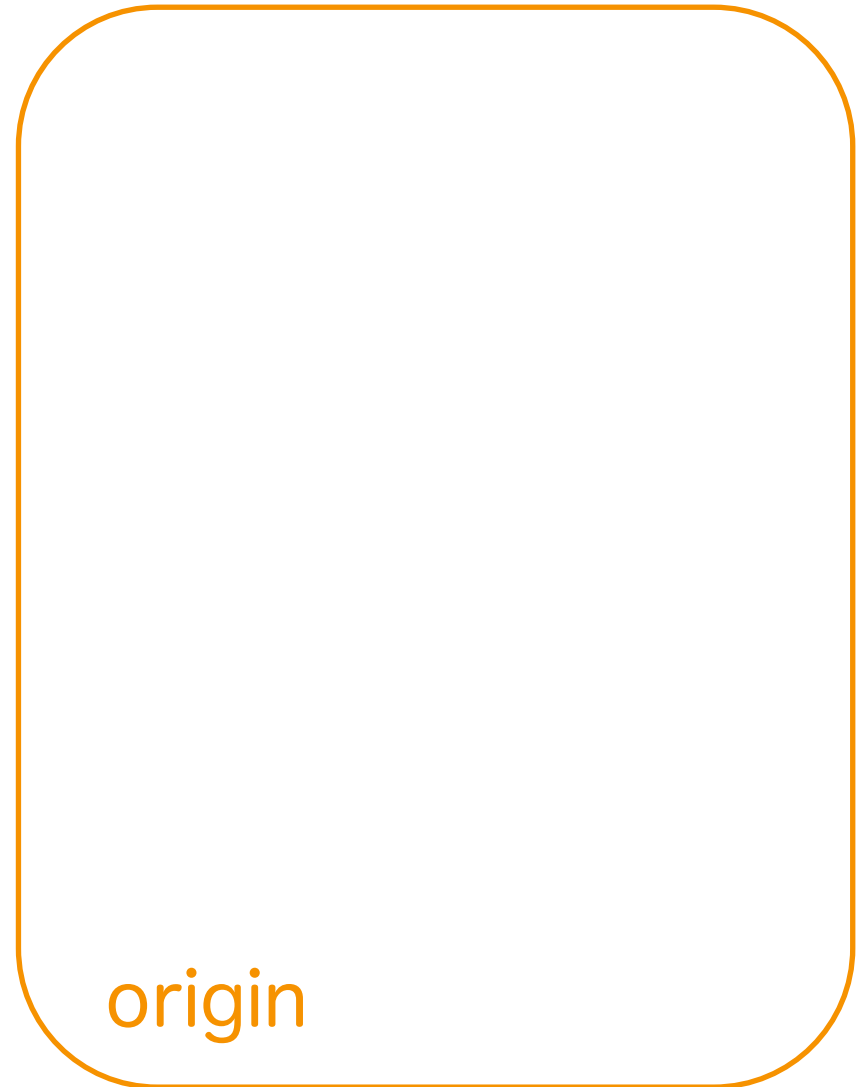
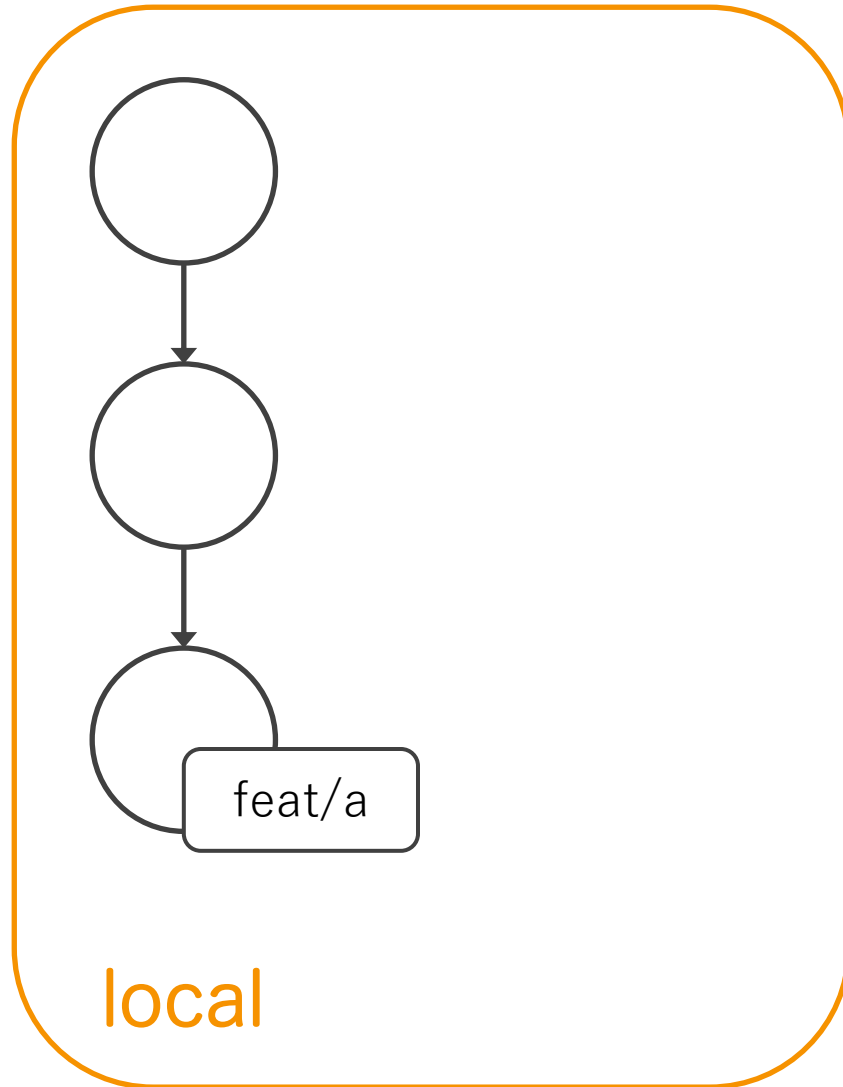
---

- Branch の push

```
$ git push origin feature/a
```

# Branch

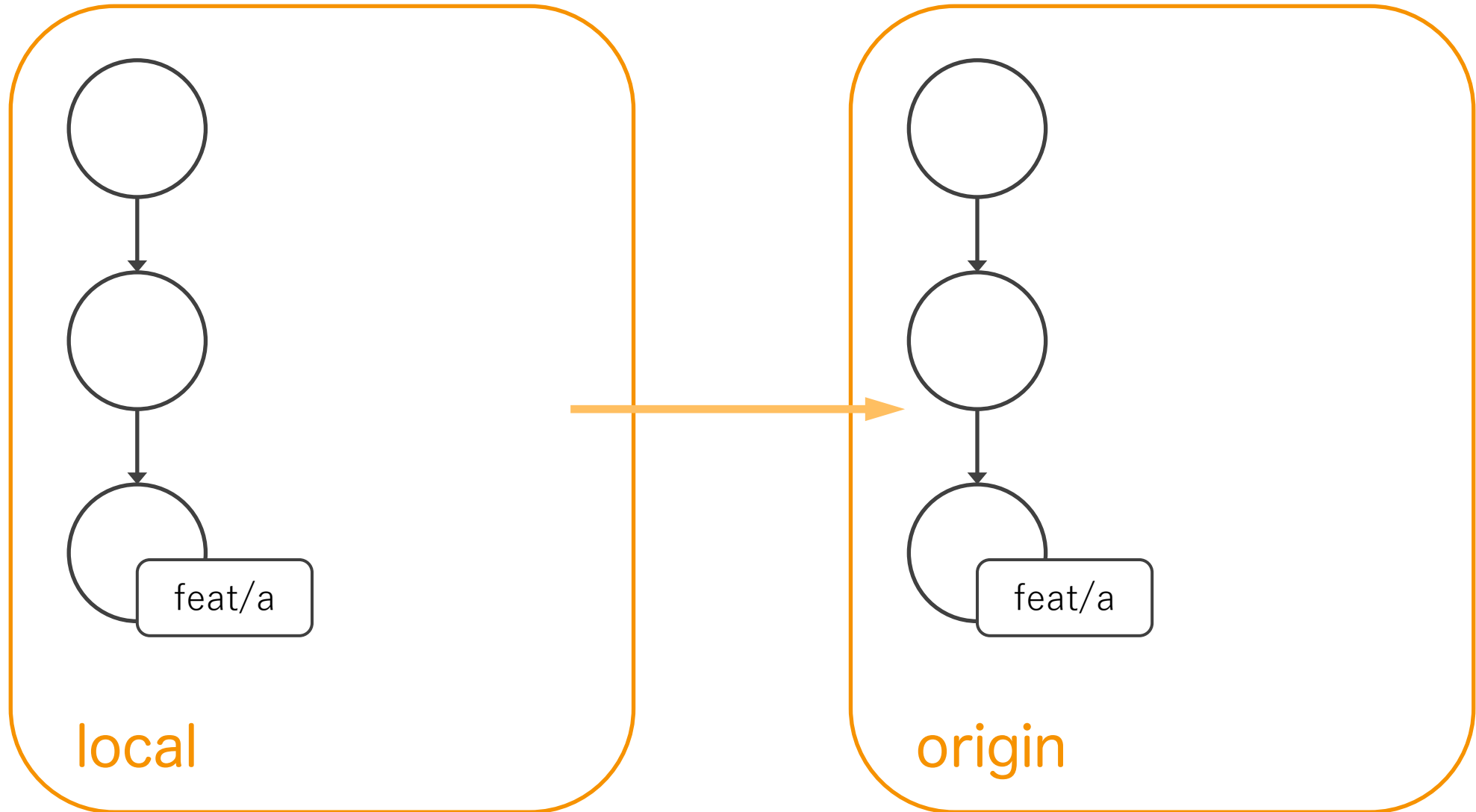
---



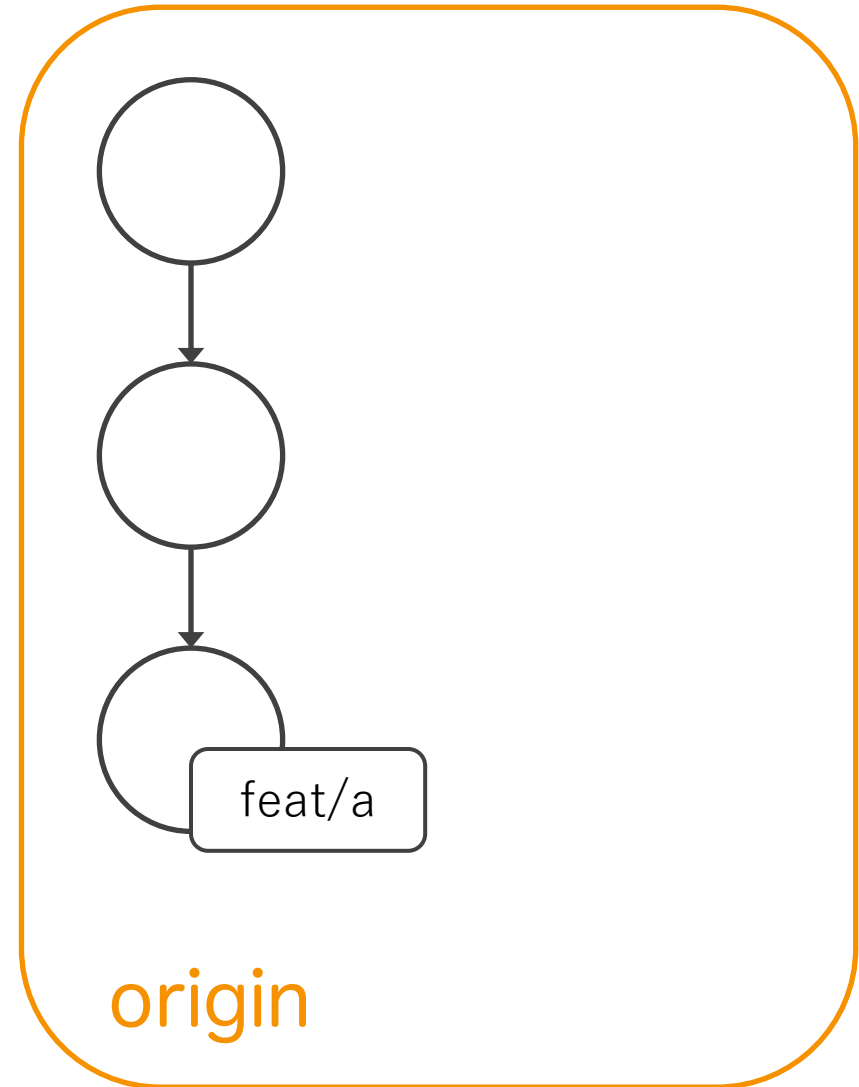
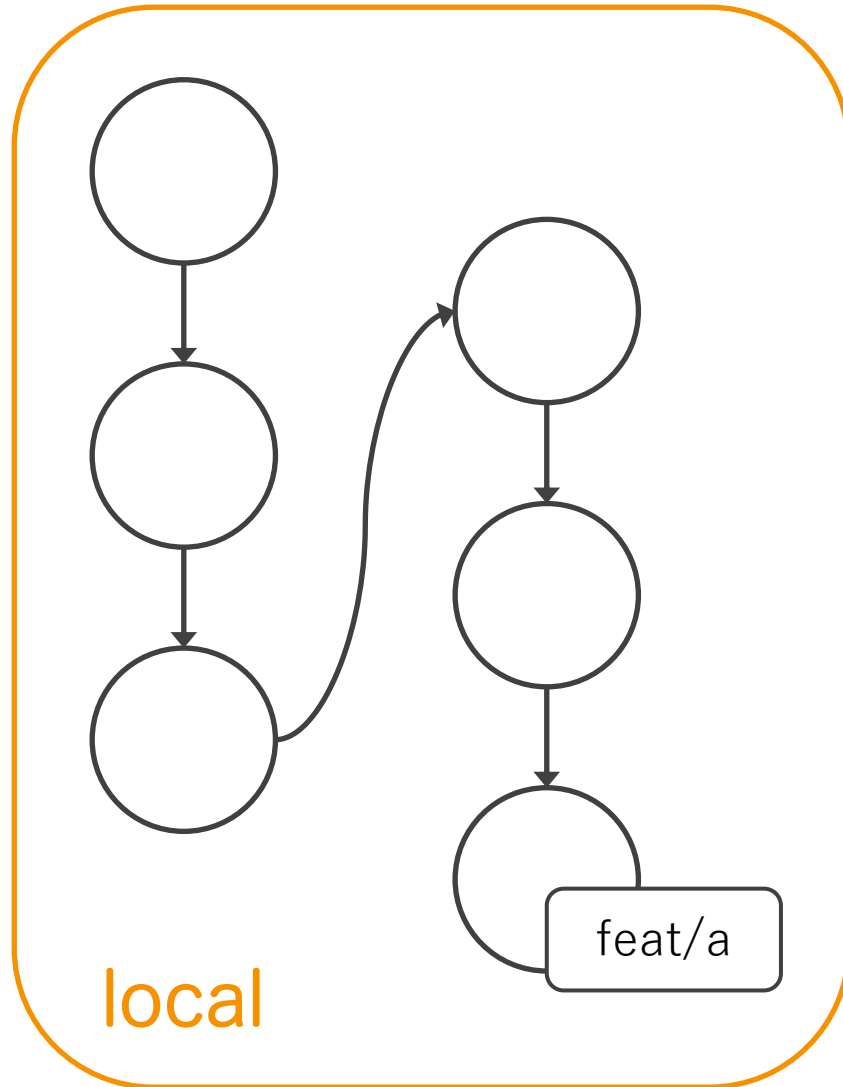


# Branch

---



# Branch



# Branch

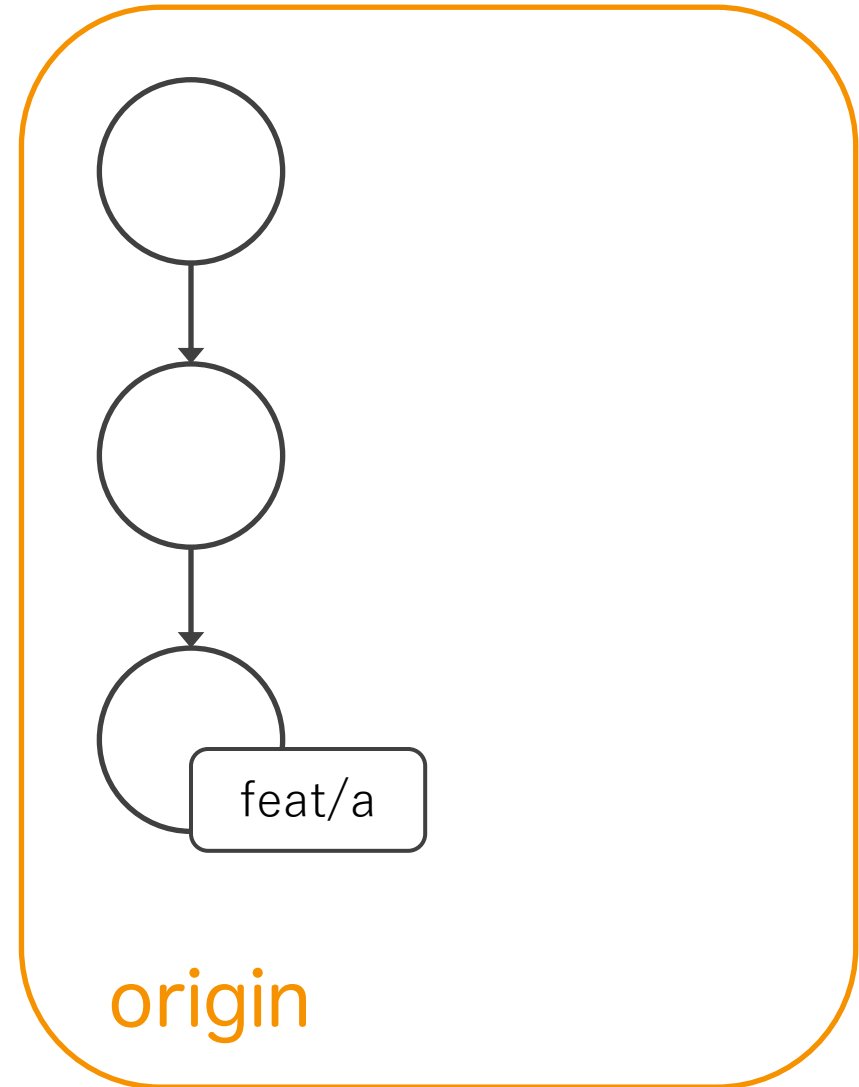
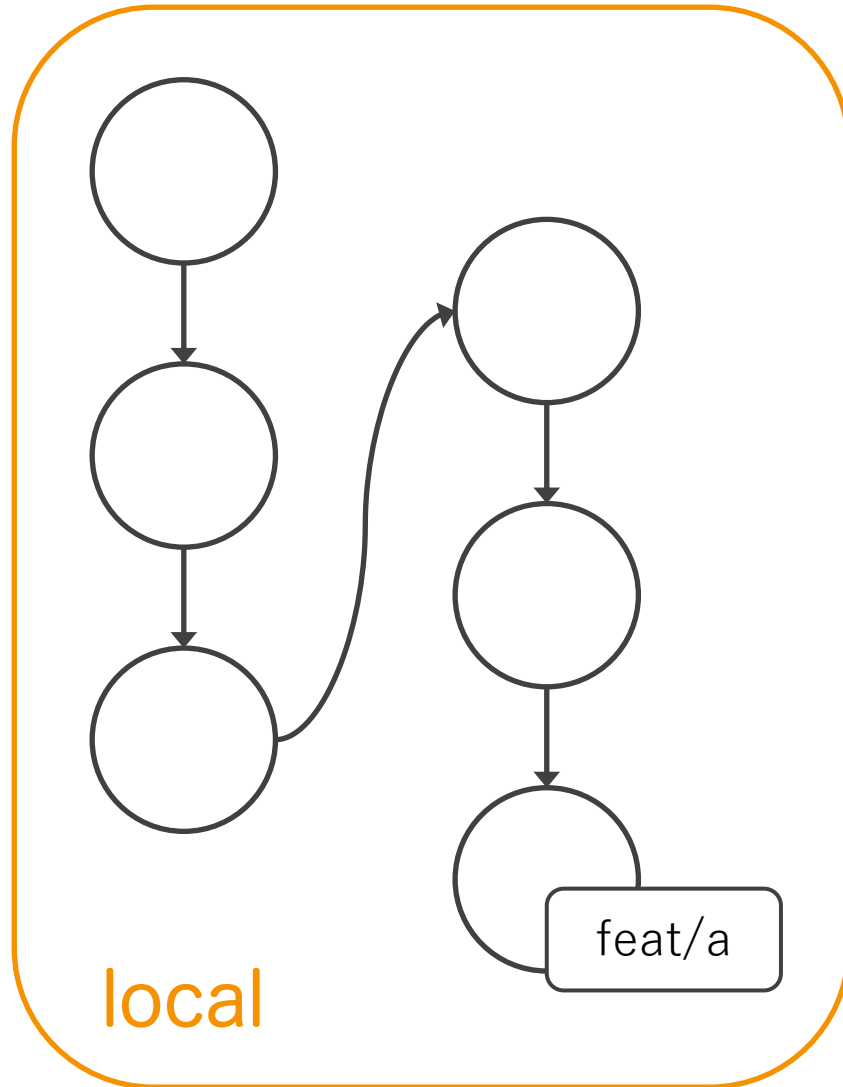
---

- Branch の push

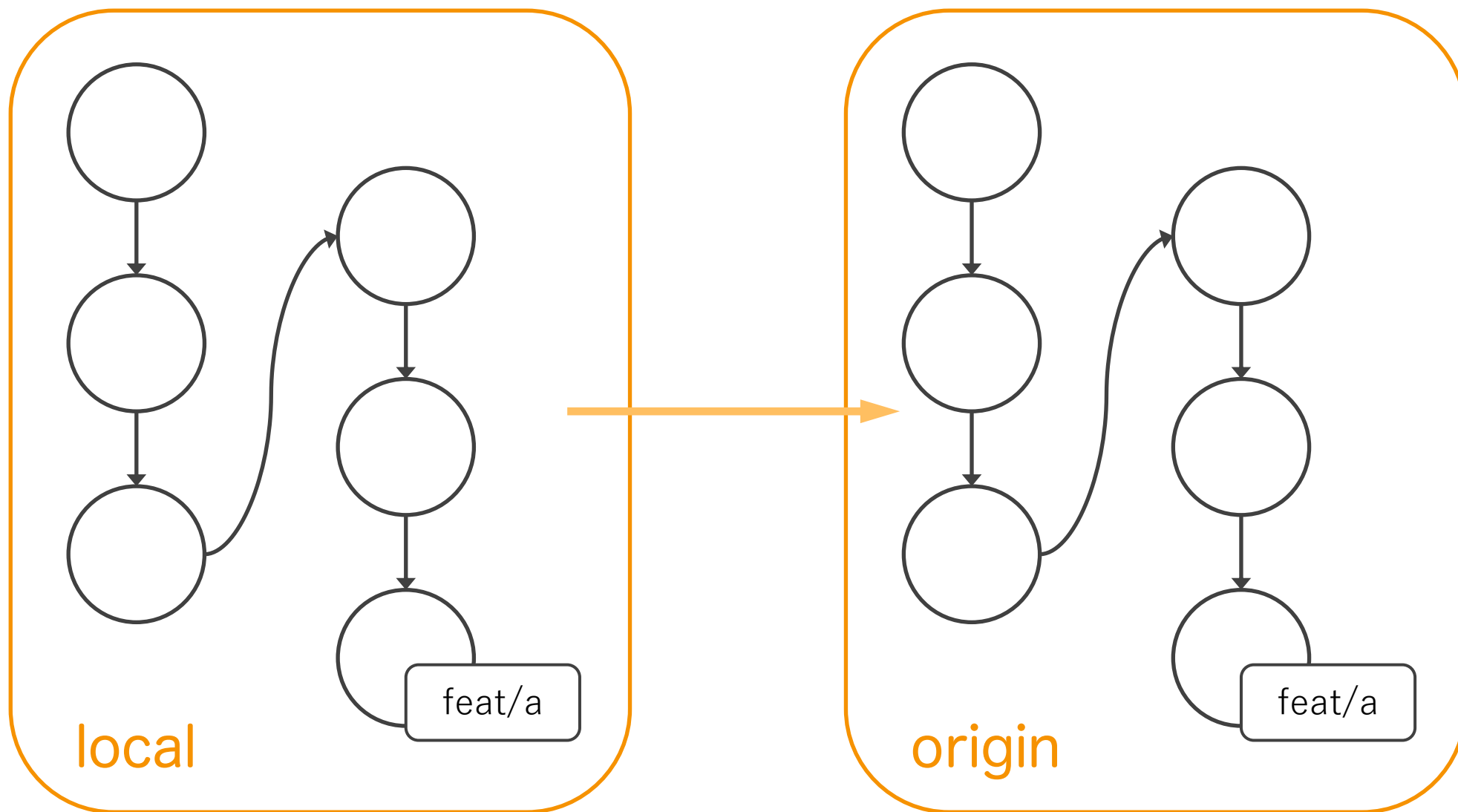
```
$ git push origin feature/a
```

- 作業後、再度 push するとリモートのブランチも更新される

# Branch



# Branch



組み込み Ref

## 組み込み Ref

---

- ・ Git には始めから 2 つの ref が用意されている

# 組み込み Ref

---

- Git には始めから 2 つの ref が用意されている
  - HEAD
  - master



# 組み込み Ref

---

- HEAD
  - 今 checkout されているものを表す特別な ref

# 組み込み Ref

---

- HEAD
  - 今 checkout されているものを表す特別な ref
    - 「コミット ID かブランチ名」を記録する (symbolic-ref)

# 組み込み Ref

---

- HEAD

```
$ git checkout c5d57bc...
$ cat .git/HEAD
C5d57bc23a343c226e4970835444c6bd16829878

$ git checkout hoge
$ cat .git/HEAD
ref: refs/heads/hoge
```

- コミット ID or タグで checkout するとコミット ID
- ブランチ名で checkout するとブランチ名

# 組み込み Ref

---

- HEAD
  - ブランチ名で checkout するとブランチ名
    - この状態でコミットすると、そのブランチが進む

# 組み込み Ref

---

- HEAD
  - ブランチ名で checkout するとブランチ名
    - この状態でコミットすると、そのブランチが進む
- コミットが直接 checkout されている状態 = Detached HEAD
  - Git ではむしろこちらがレアケース

# 組み込み Ref

---

- master

# 組み込み Ref

---

- master
  - デフォルトで checkout されているブランチ
  - 基本的には master の最新コミット = 最新 ver になる

`git commit` 再訪



# `git commit` 再訪

---

- Git の内部構造はわかった
  - Commit
  - Tree
  - Blob
  - Ref (Branch, Tag など)

## `git commit` 再訪

---

- ここからは、それぞれの Git コマンドの挙動を見ていく
  - commit (, add)
  - checkout, reset
  - merge
  - rebase (, cherry-pick, revert)

## `git commit` 再訪

---

- ここからは、それぞれの Git コマンドの挙動を見ていく
  - `commit` (`,` `add`)
  - `checkout`, `reset`
  - `merge`
  - `rebase` (`,` `cherry-pick`, `revert`)

## `git commit` 再訪

---

- commit を作る前に叩くコマンド `git add`

## `git commit` 再訪

---

- commit を作る前に叩くコマンド `git add`
  - 直感的には、次のコミットに反映する差分を選ぶコマンド

## `git commit` 再訪

---

- commit を作る前に叩くコマンド `git add`
  - 直感的には、次のコミットに反映する差分を選ぶコマンド
    - 実行すると、実際には何が起こるのか？

## `git commit` 再訪

---

- ・ 次のコミットを組み立てる場所「インデックス」

## `git commit` 再訪

- ・ 次のコミットを組み立てる場所「インデックス」

```
$ cat .git/index
DIRC¥[QlC[QlC[ [3/S ¥(
 V(d_Readme.markdown¥[Q[U¥[Q[U[ [3/S    CK)
 wZ   S doc/hogehogedoc.markdown¥[Q[+¥[Q[+      [3/S    CK) wZ   S src/hogehoge.js
```

- tree オブジェクトによく似たバイナリ



# `git commit` 再訪

- ・ 次のコミットを組み立てる場所「インデックス」

```
$ cat .git/index
```

```
DIRC¥QlC¥QlC[3/S ¥(¥¥¥¥¥
```

これは ver.2 形式のインデックスですよ

```
¥Q¥u¥[3/S¥¥CK¥)  
¥wZ¥¥¥S¥doc/hogehogedoc.markdown¥¥Q¥¥+¥¥Q¥¥+¥¥¥¥¥[3/¥  
S¥¥CK¥)¥wZ¥¥¥S¥src/hogehoge.js
```

- ・ tree オブジェクトによく似たバイナリ

## `git commit` 再訪

- ・ 次のコミットを組み立てる場所「インデックス」

```
$ cat .git/index
DIRC¥[QlC¥[QlC[[[3/S ¥([)]
[V(d[_Readme.markdown¥cuycoy[[[?/(S[CK)
wZ[Sdoc/hogehoged [[[3/
S[CK)wZ[Ssrc/nogenoge.js
```

- tree オブジェクトによく似たバイナリ

## `git commit` 再訪

- ・ 次のコミットを組み立てる場所「インデックス」

```
$ cat .git/index
DIRC¥[QlC[3/S ¥([
V(d_Readme.markdown¥[Q[U¥[Q[U[3/S🚢CK)
wZ[Sdoc/hogehogedoc.markdown¥[c[.y[c[.f[c[.v(
S🚢CK)wZ[Ssrc/hogehoge.
```

- tree オブジェクトによく似たバイナリ

# `git commit` 再訪

- ・ 次のコミットを組み立てる場所「インデックス」

```
$ cat .git/index
DIRC¥[QlC¥[QlC[3/S ¥([
[V(d_Readme.markdown¥[Q[U¥[Q[U[3/S🚢CK)
[wZ[Sdoc/hogehogedoc.markdown¥[Q[+¥[Q[+[3/S
S🚢CK)[wZ[Ssrc/hogehoge.js
```

三つ目のファイル名, blob ID, メタデータ

- ・ tree オブジェクトによく似たバイナリ

## `git commit` 再訪

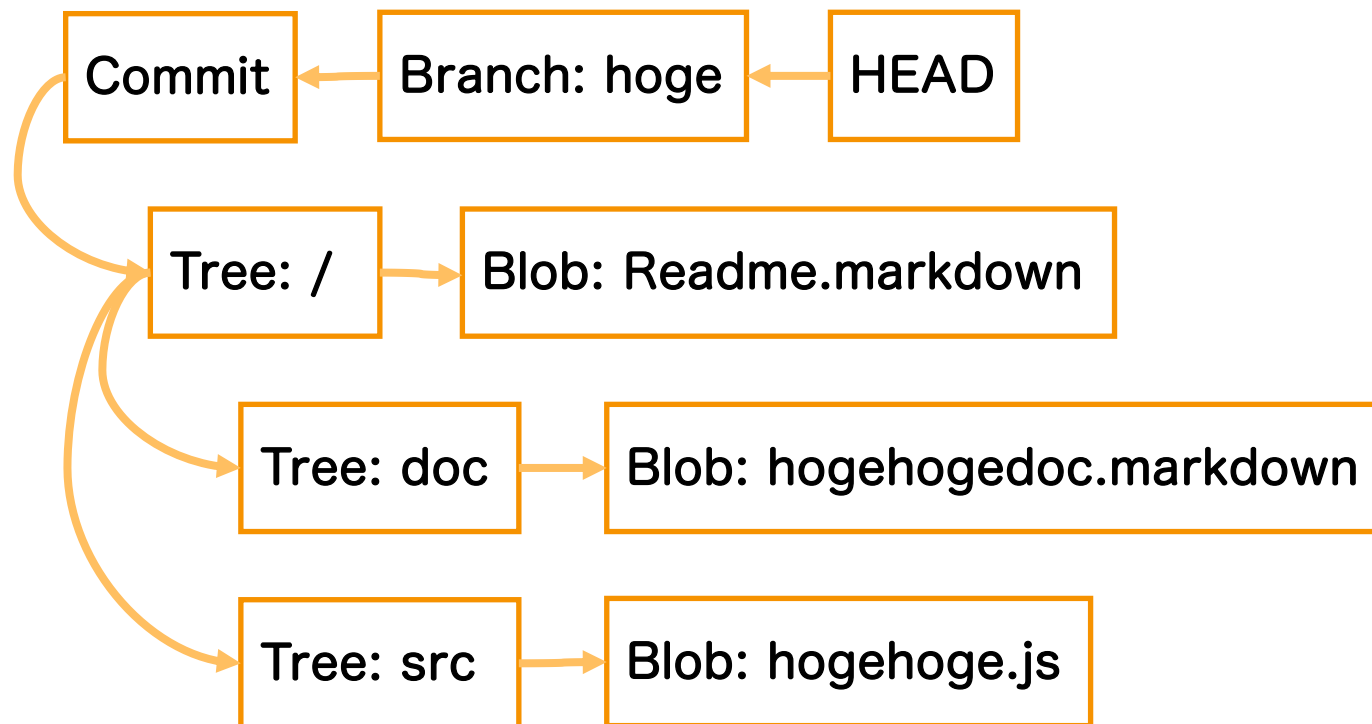
---

- ・ git リポジトリの全体像

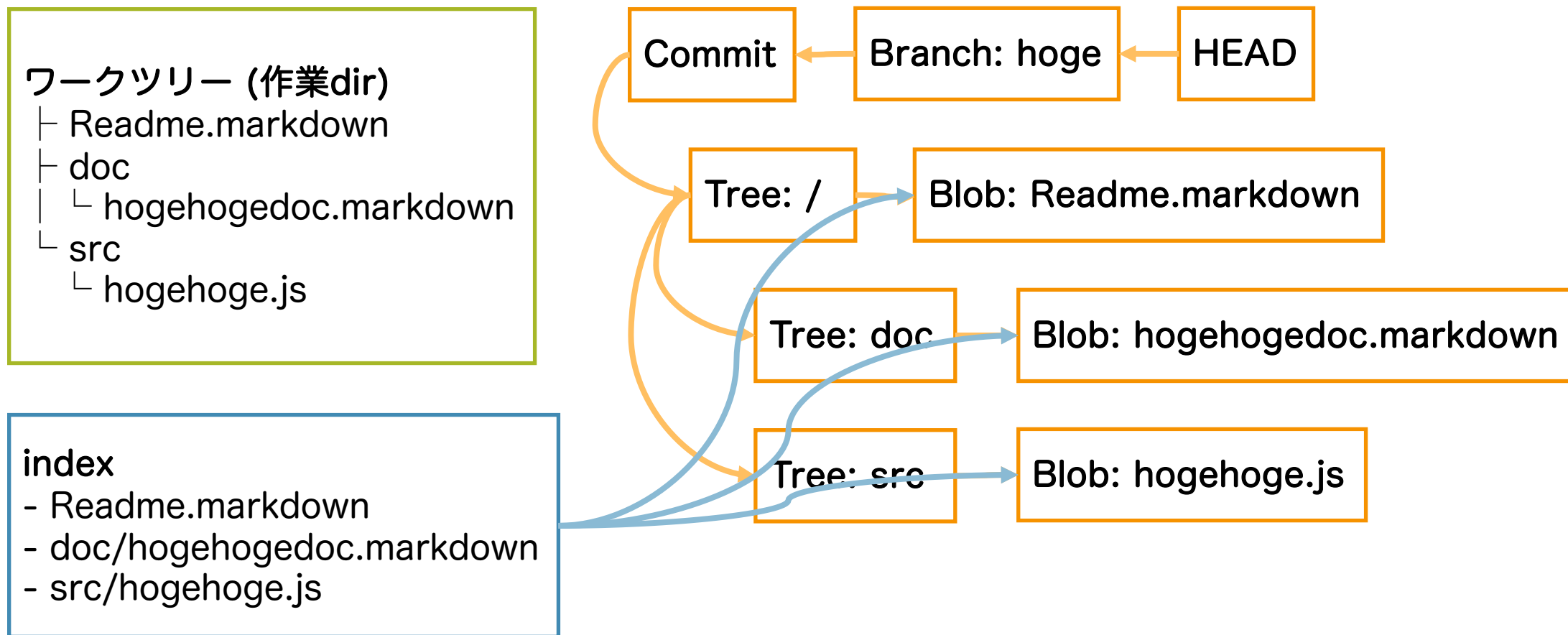
# `git commit` 再訪

ワークツリー (作業dir)

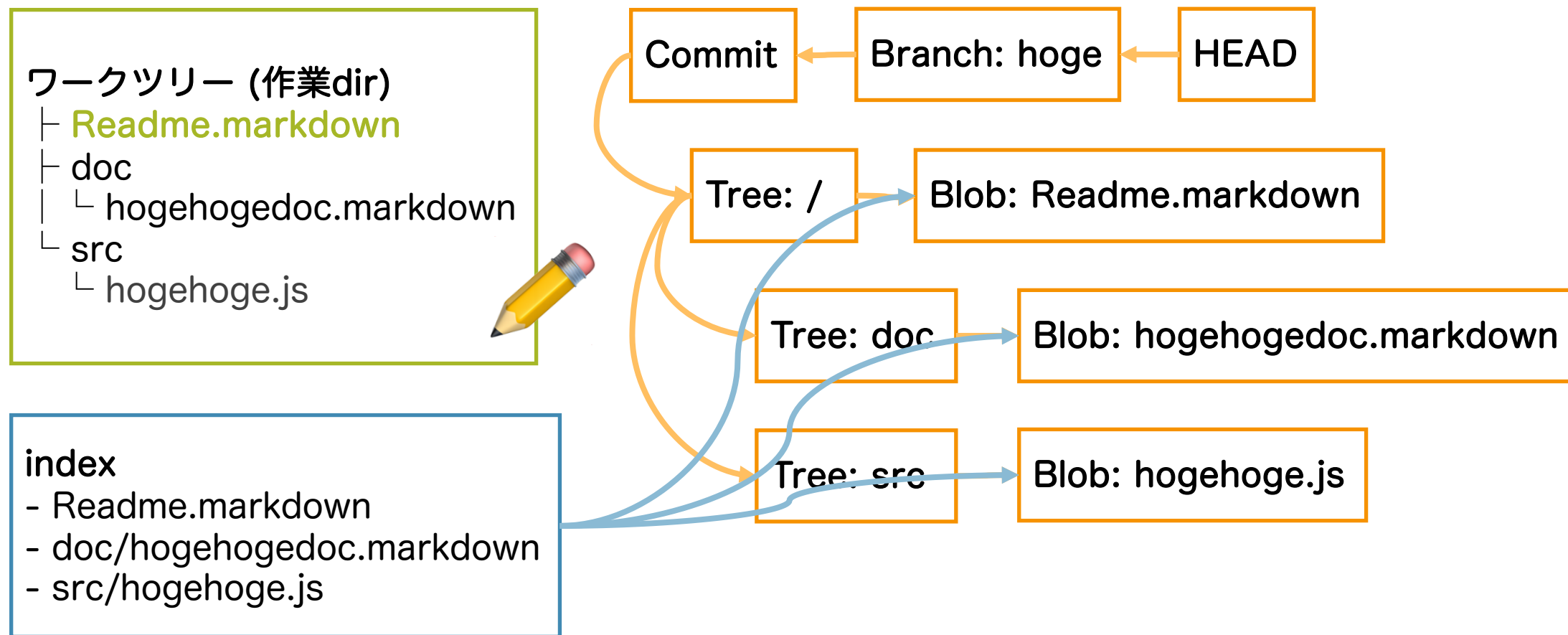
- └ Readme.markdown
- └ doc
  - └ hogehogedoc.markdown
- └ src
  - └ hogehoge.js



# `git commit` 再訪



# `git commit` 再訪





# `git commit` 再訪

---

- ・コミットができるまでの流れ

```
$ git status
On branch master
Changes not staged for commit:

    modified:   Readme.markdown

No changes added to commit
```

# `git commit` 再訪

---

- ・コミットができるまでの流れ

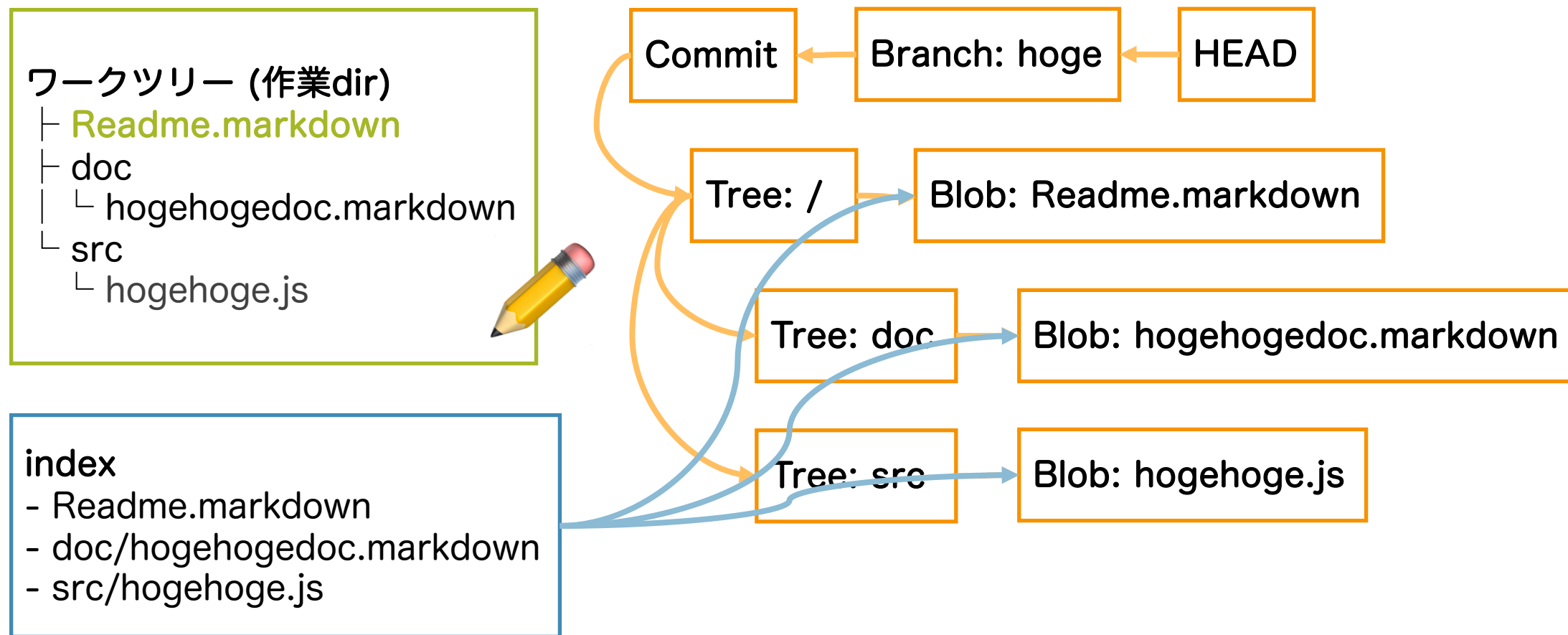
```
$ git add README.markdown
```

## `git commit` 再訪

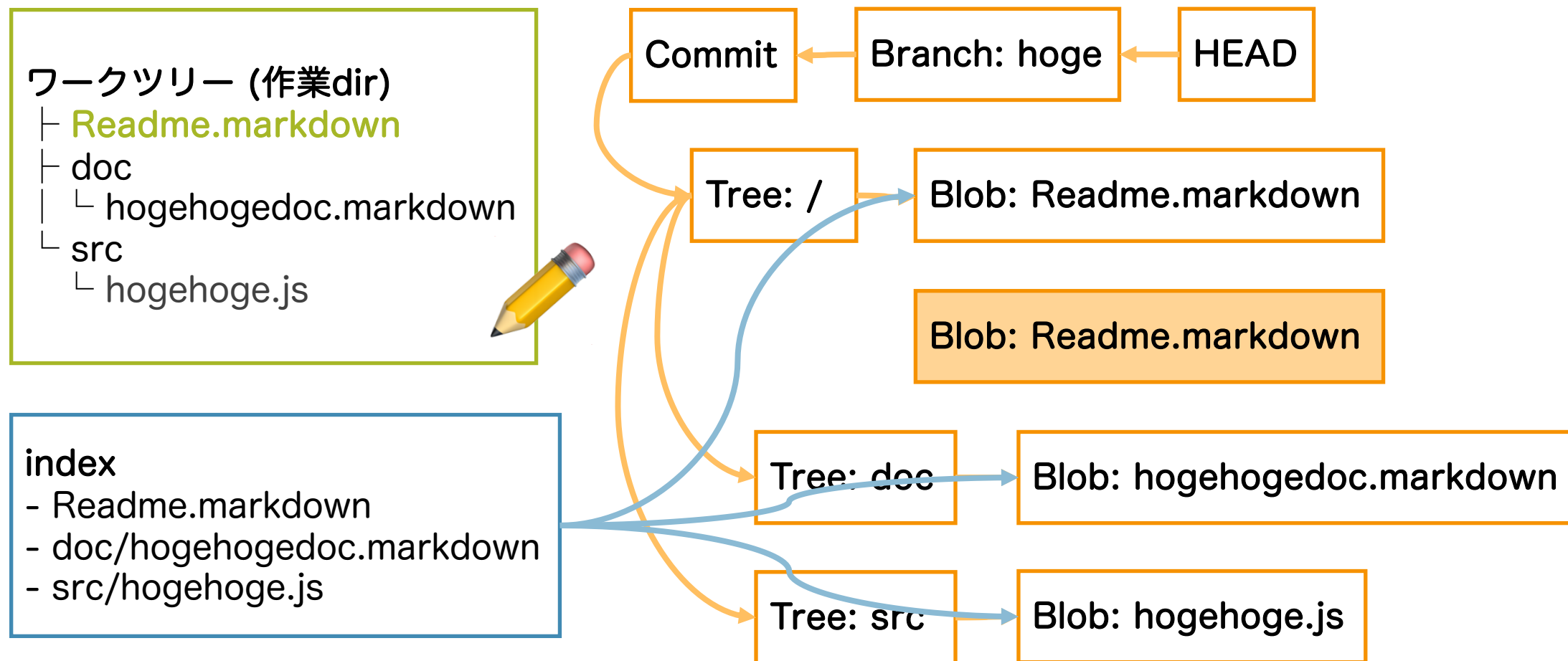
---

- ・ コミットができるまでの流れ
  - ・ `git add`
    - ・ Blob を作って、 index を更新する

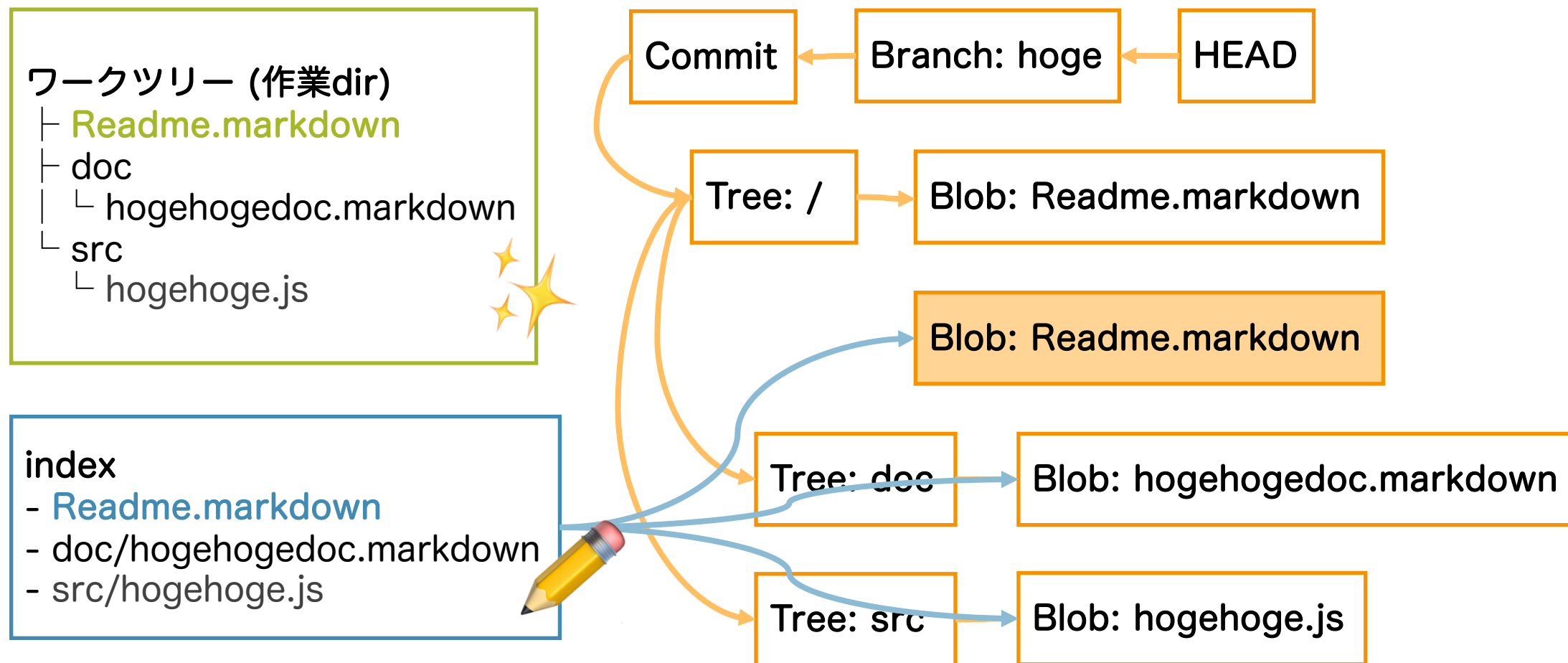
# `git commit` 再訪



# `git commit` 再訪



# `git commit` 再訪



# `git commit` 再訪

---

- ・コミットができるまでの流れ

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   README.markdown
```

# `git commit` 再訪

---

- ・コミットができるまでの流れ

```
$ git commit -m "Update Readme.markdown"
```

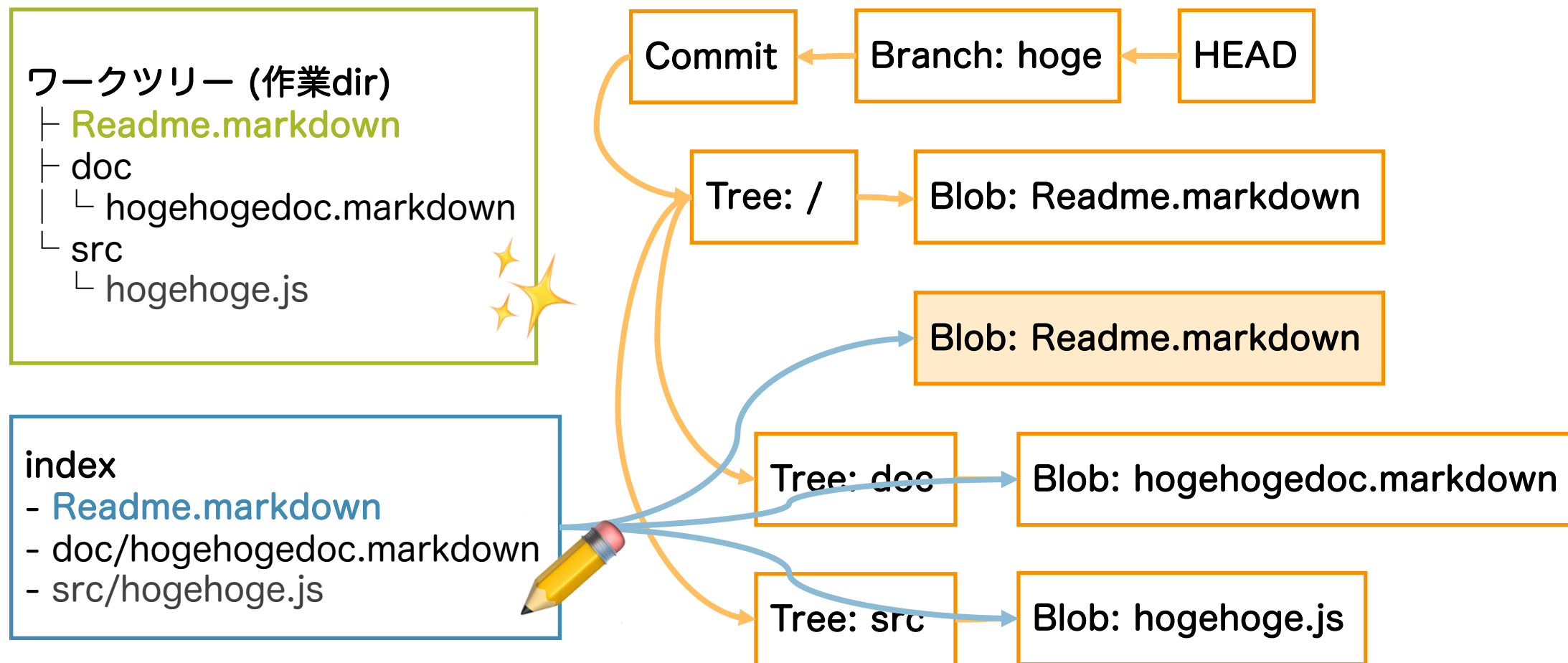


## `git commit` 再訪

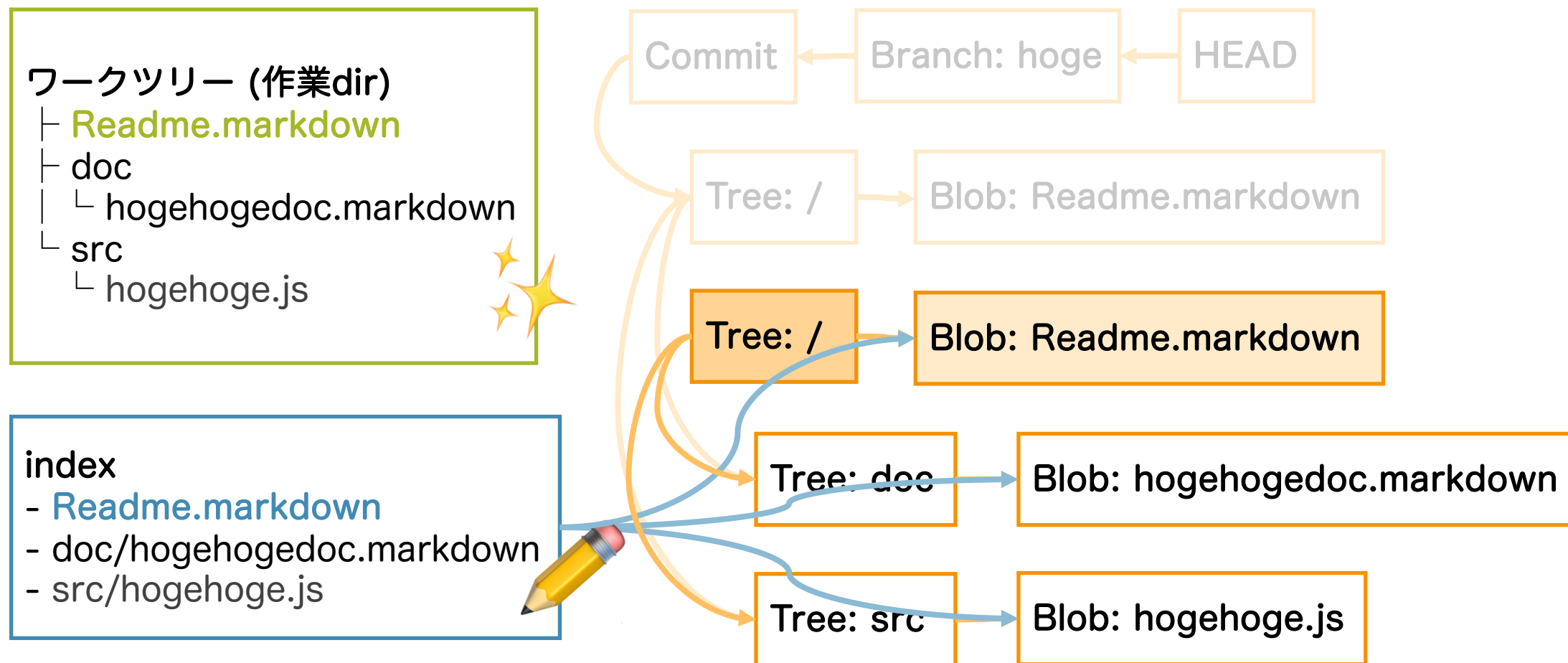
---

- コミットができるまでの流れ
  - `git add`
    - Blob を作って、index を更新する
  - `git commit`
    - 1. index をもとに Tree オブジェクトを作る

# `git commit` 再訪



# `git commit` 再訪

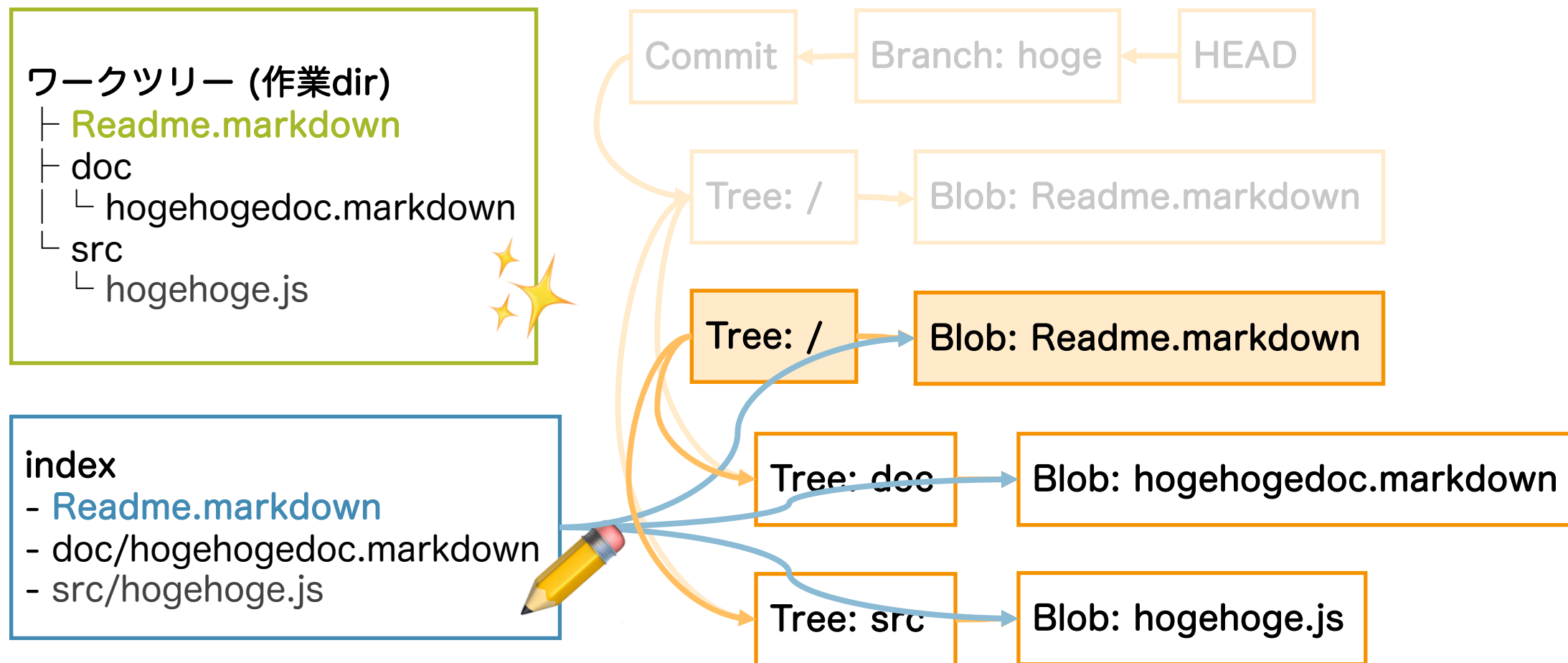


## `git commit` 再訪

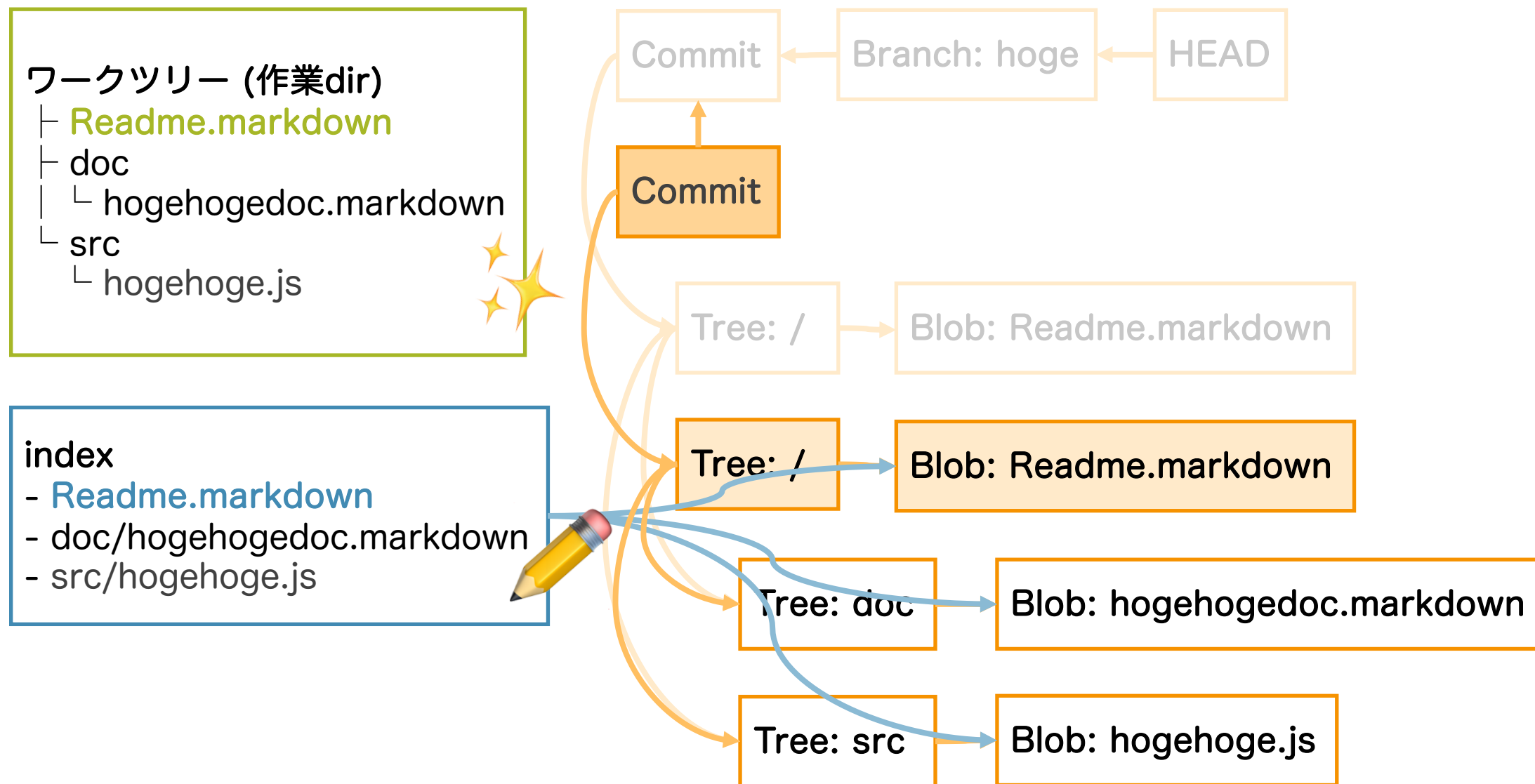
---

- コミットができるまでの流れ
  - `git add`
    - Blob を作って、index を更新する
  - `git commit`
    - 1. index をもとに Tree オブジェクトを作る
    - 2. Tree オブジェクトを指すコミットを作る

# `git commit` 再訪



# `git commit` 再訪

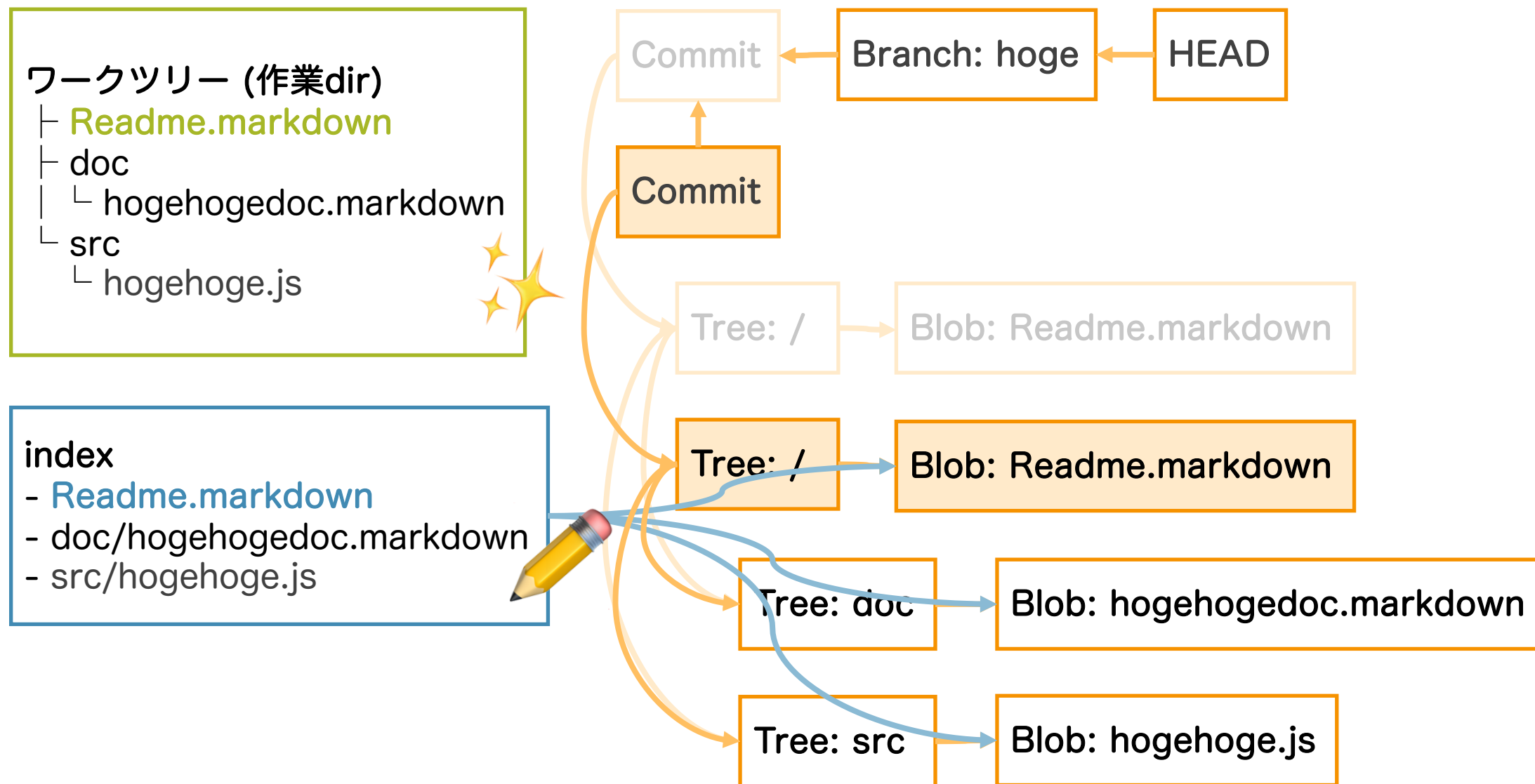


# `git commit` 再訪

---

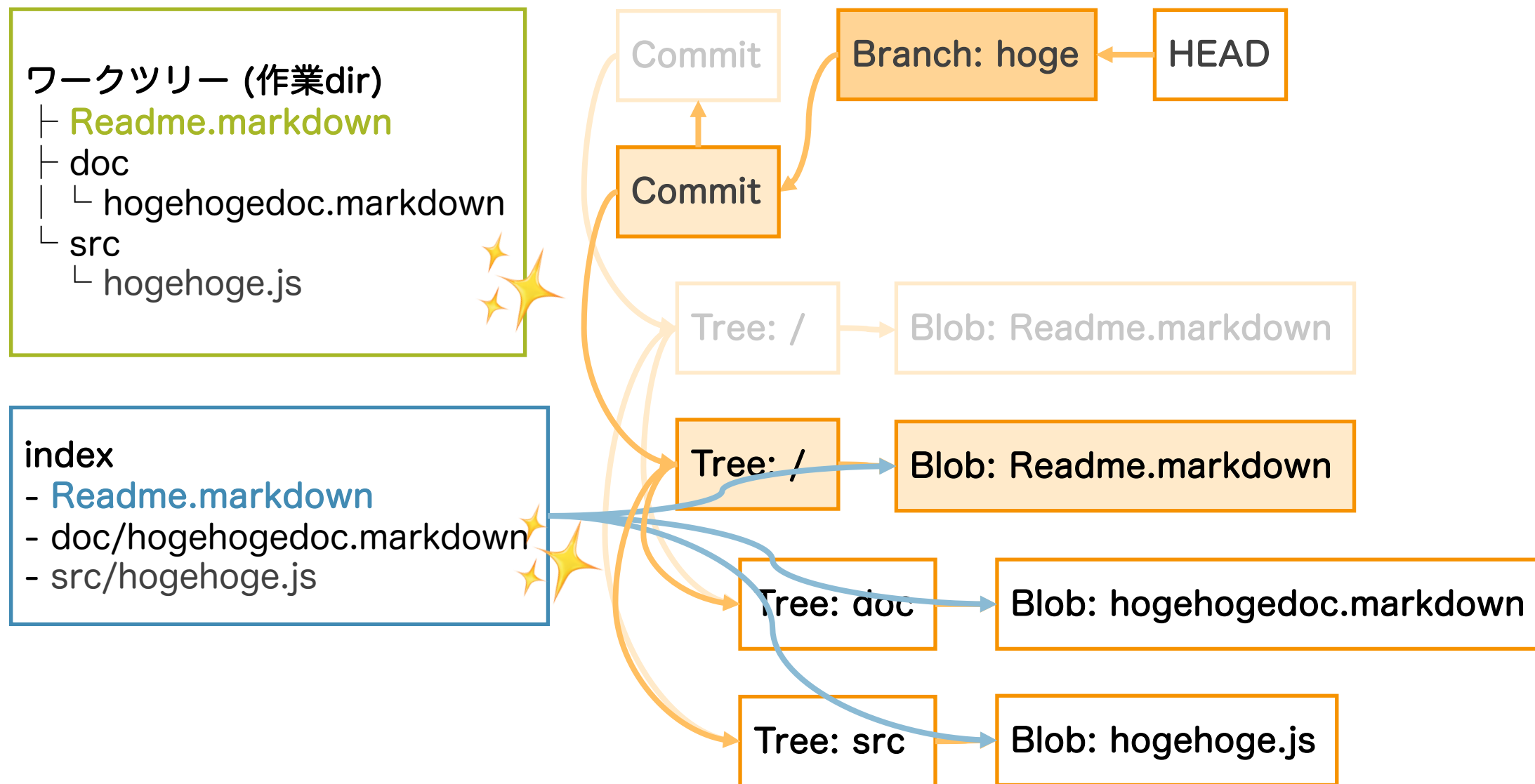
- コミットができるまでの流れ
  - `git add`
    - Blob を作って、index を更新する
  - `git commit`
    - 1. index をもとに Tree オブジェクトを作る
    - 2. この Tree オブジェクトを指すコミットを作る
    - 3. ブランチ (または HEAD) の向き先を更新する

# `git commit` 再訪





# `git commit` 再訪



コミットを取り出す

# コミットを取り出す

---

- ここからは、それぞれの Git コマンドの挙動を見ていく
  - commit (, add)
  - checkout, reset
  - merge
  - rebase (, cherry-pick, revert)

# コミットを取り出す

---

- ・意外と難しい `git checkout`, `git reset`

# コミットを取り出す

---

- ・ 意外と難しい `git checkout`, `git reset`
  - ・ checkout ... ブランチを切り替える？コミットを復元する？
  - ・ reset ... `git add` を元に戻す？ブランチを巻き戻す？

## コミットを取り出す

---

- ・ ``git checkout``, ``git reset`` の違いは実は更新の対象だけ

## コミットを取り出す

---

- `git checkout`, `git reset` の違いは実は更新の対象だけ
  - checkout ... HEAD, インデックス, ワークツリー を更新
  - reset ... branch, インデックス を更新

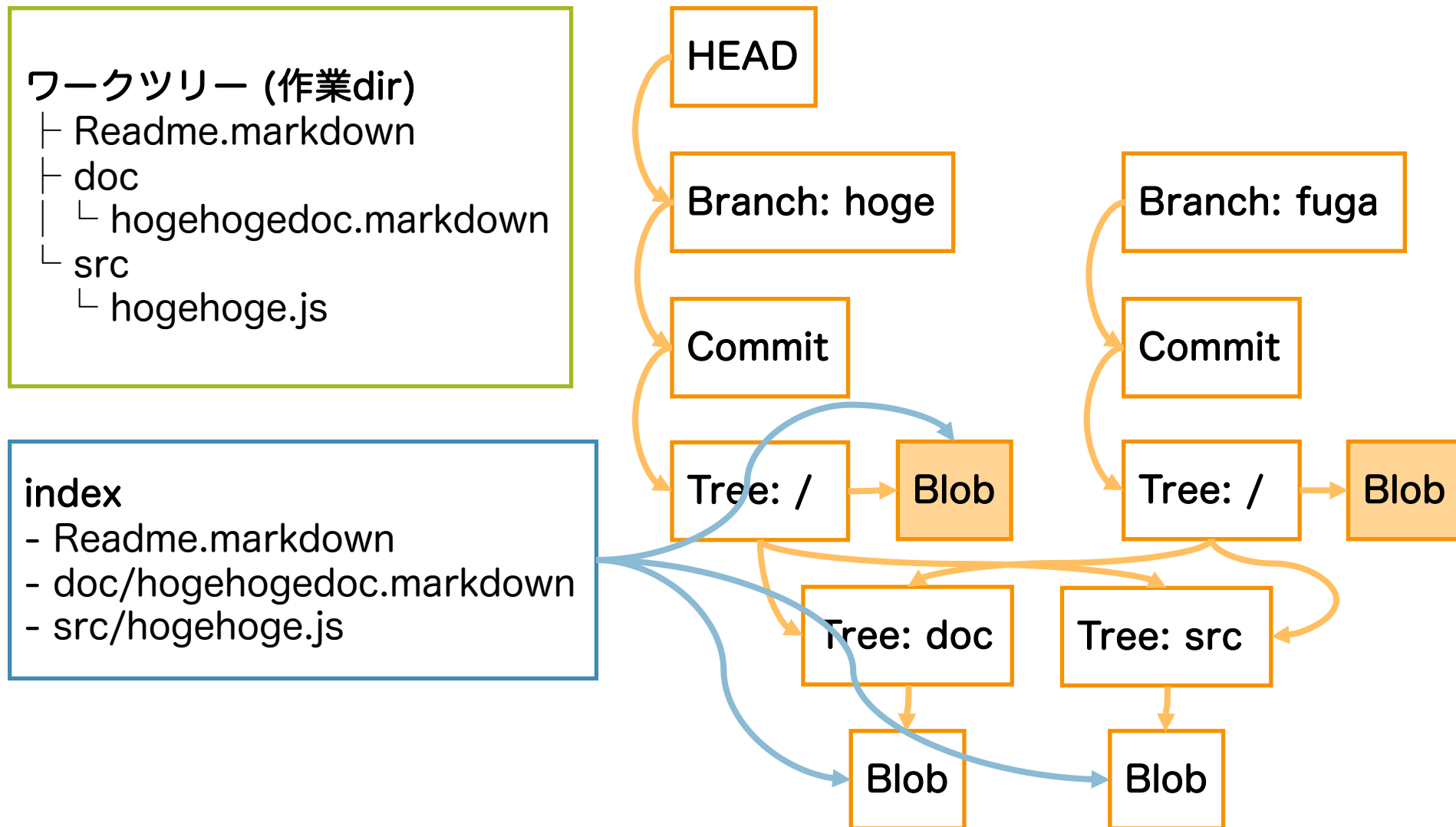
## コミットを取り出す

---

- ・ `git checkout` ... HEAD, インデックス, ワークツリー を更新



# コミットを取り出す



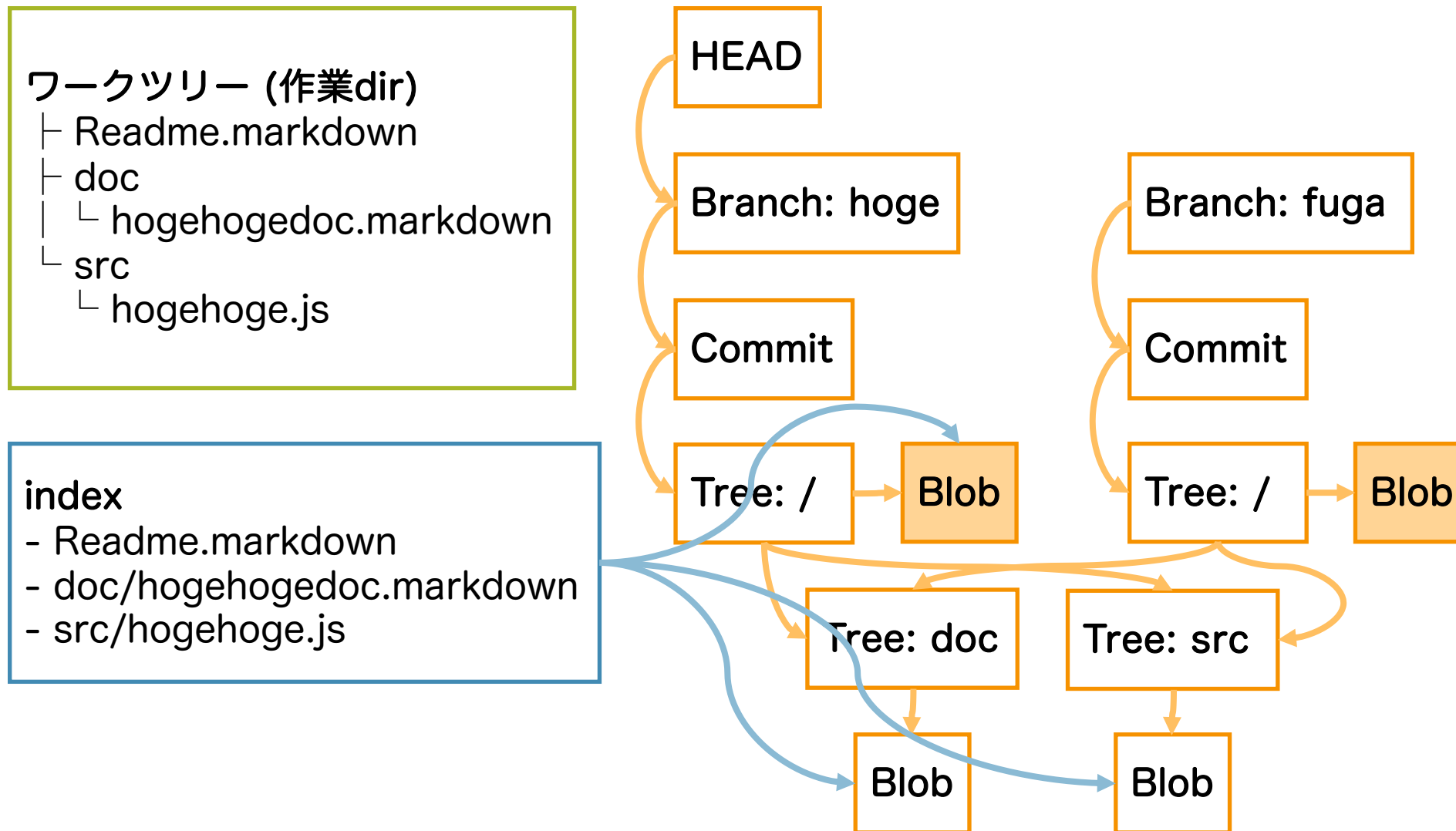
# コミットを取り出す

---

- `git checkout` ... HEAD, インデックス, ワークツリー を更新

```
$ git checkout fuga
```

# コミットを取り出す



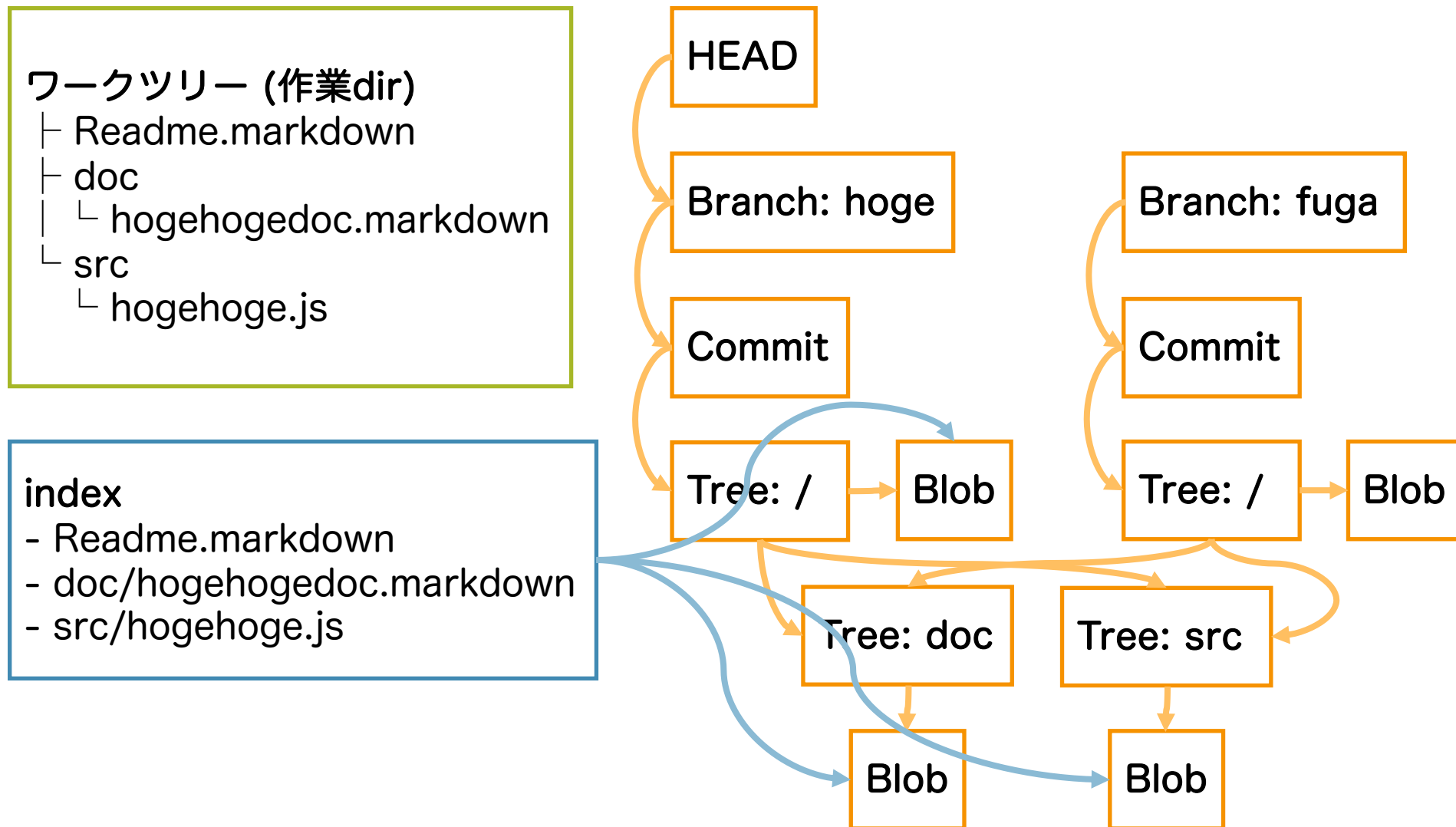
# コミットを取り出す

---

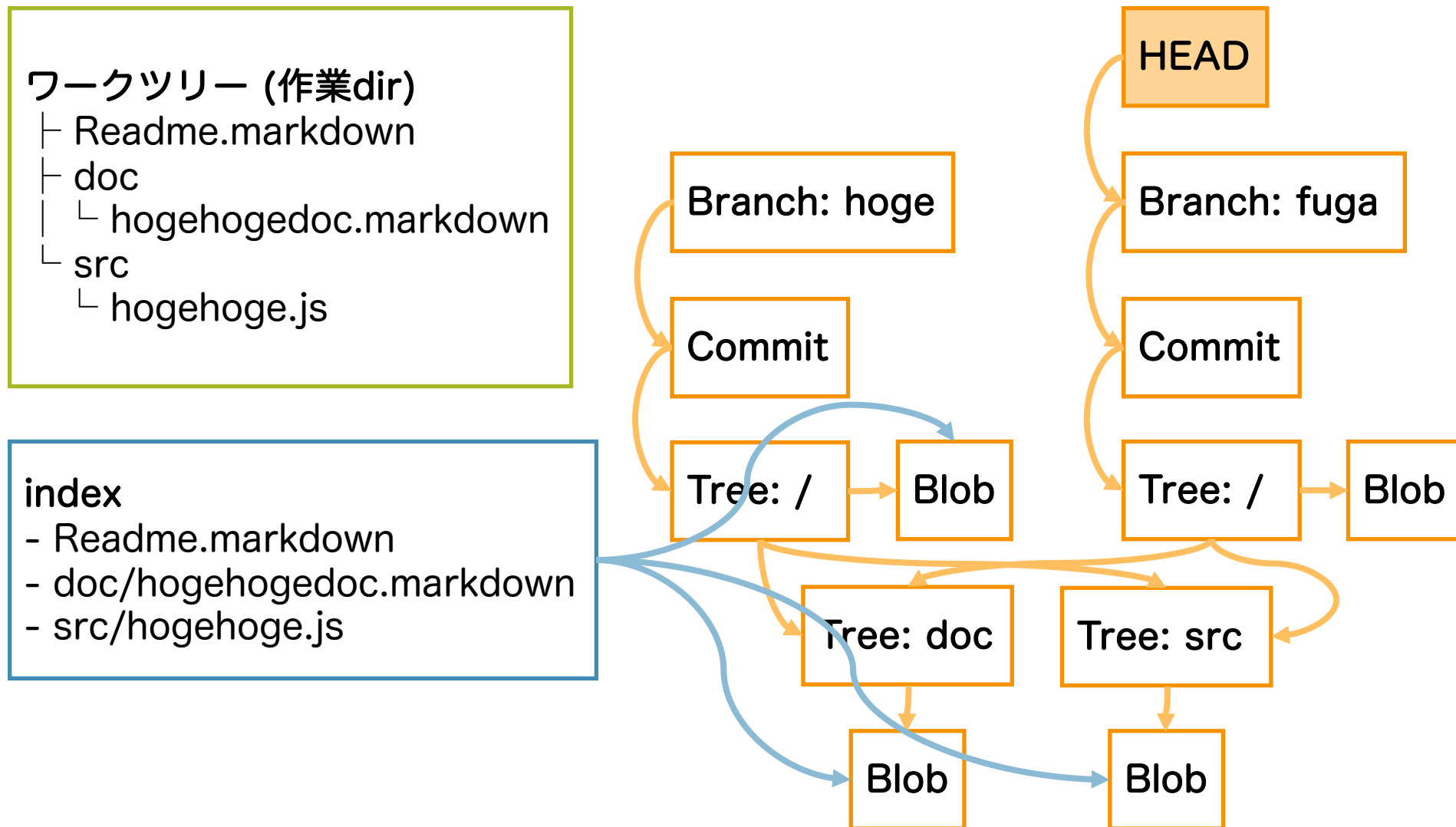
- `git checkout` ... **HEAD**, インデックス, ワークツリー を更新

```
$ git checkout fuga
```

# コミットを取り出す



# コミットを取り出す



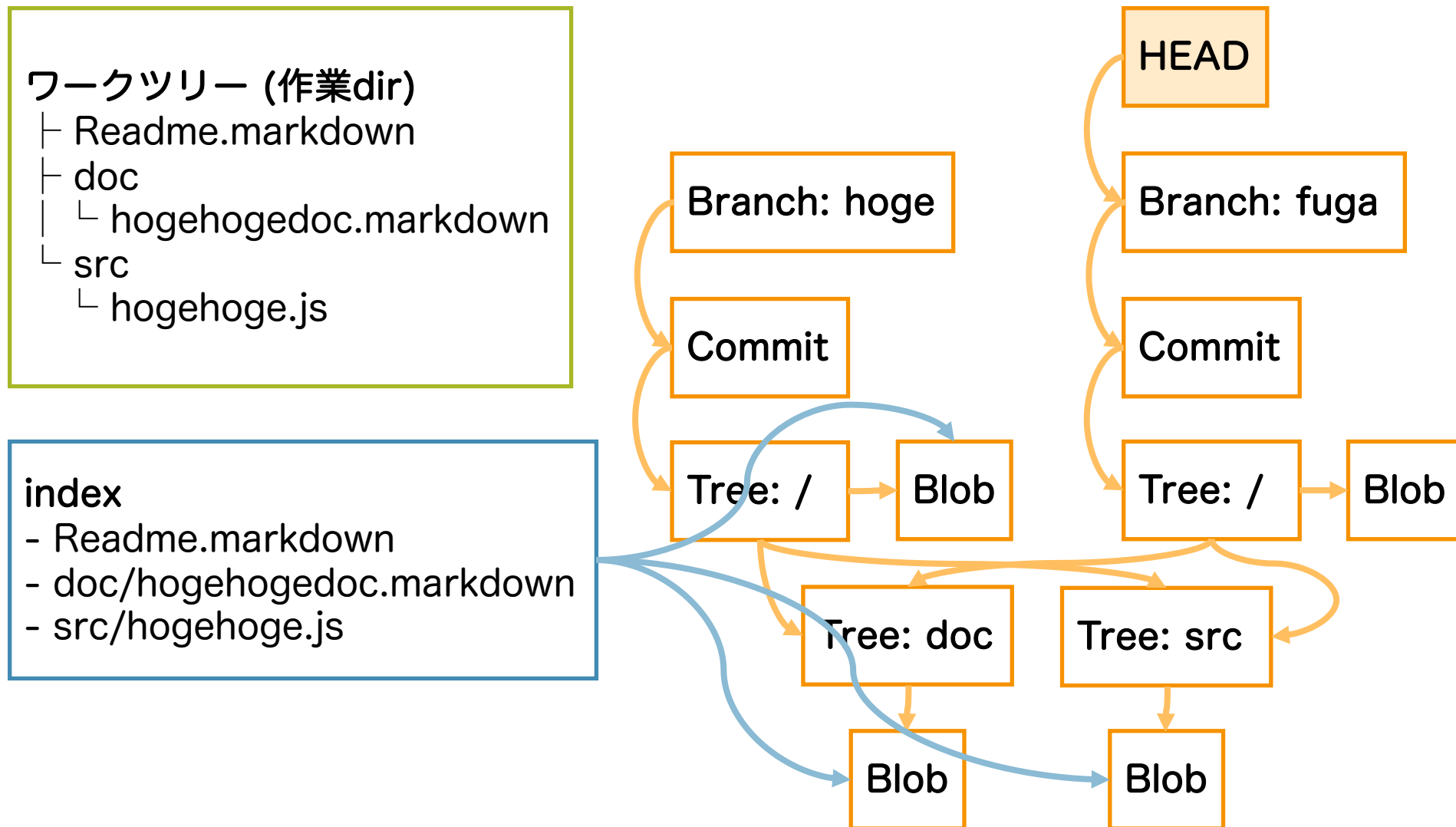
# コミットを取り出す

---

- `git checkout` ... HEAD, インデックス, ワークツリー を更新

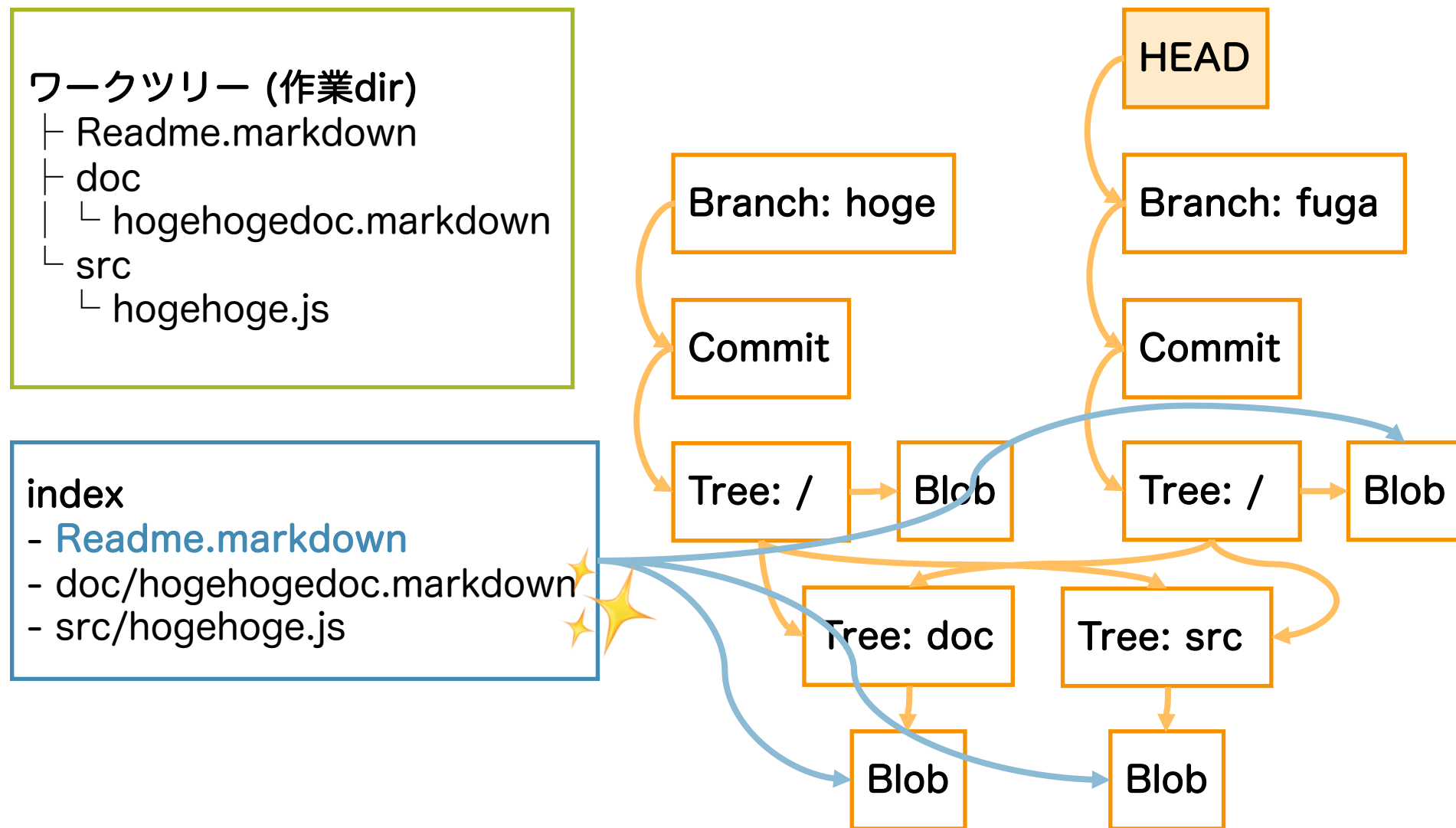
```
$ git checkout fuga
```

# コミットを取り出す





## コミットを取り出す





## コミットを取り出す

---

- ・ `git checkout` ... HEAD, インデックス, ワークツリー を更新

```
$ git checkout fuga
```

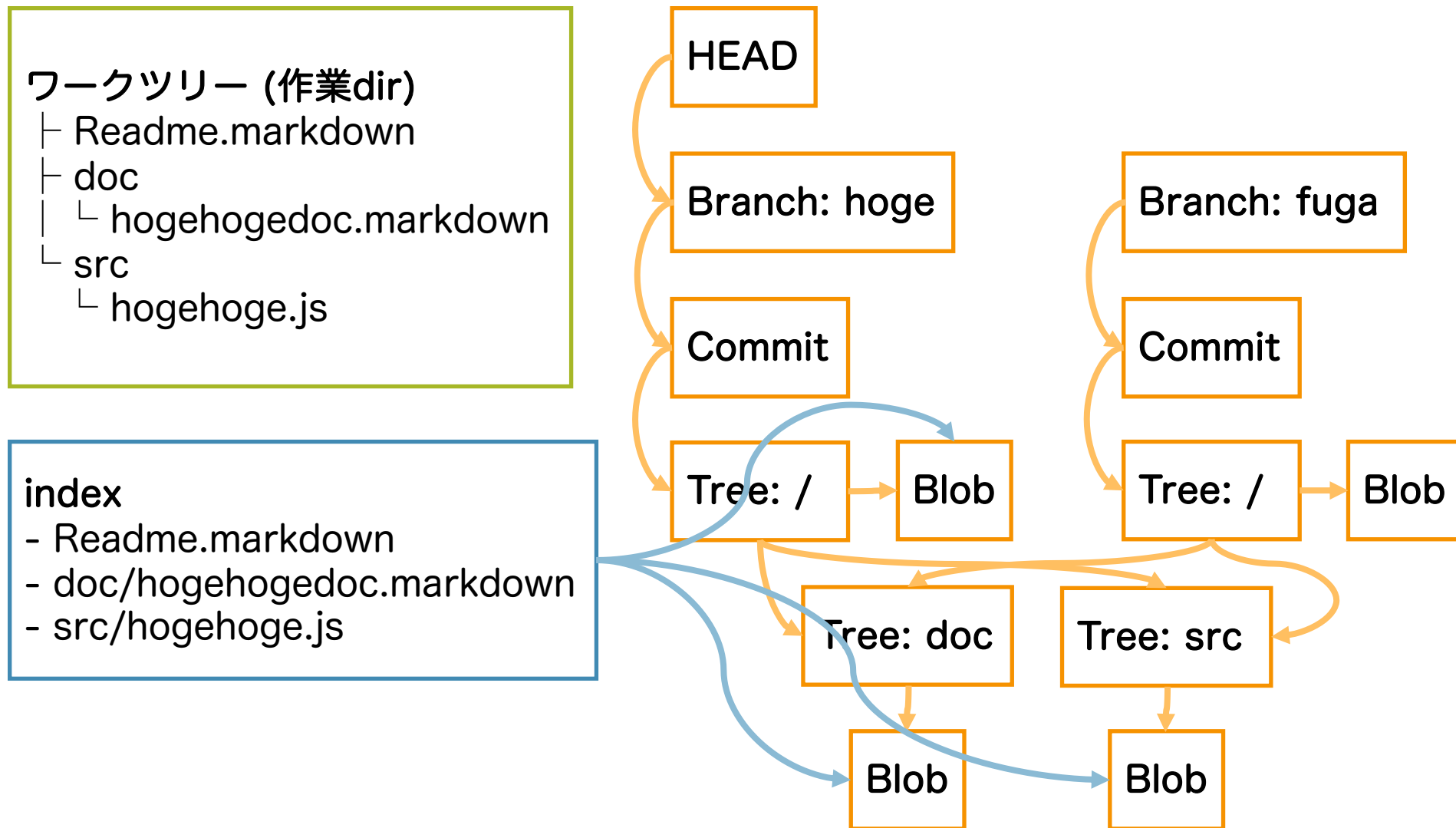
- ・ HEAD が fuga ブランチを向く
- ・ インデックス、ワークツリーの内容も fuga 時点のものになる

## コミットを取り出す

---

- ・ `git reset` ... branch, インデックス を更新

# コミットを取り出す



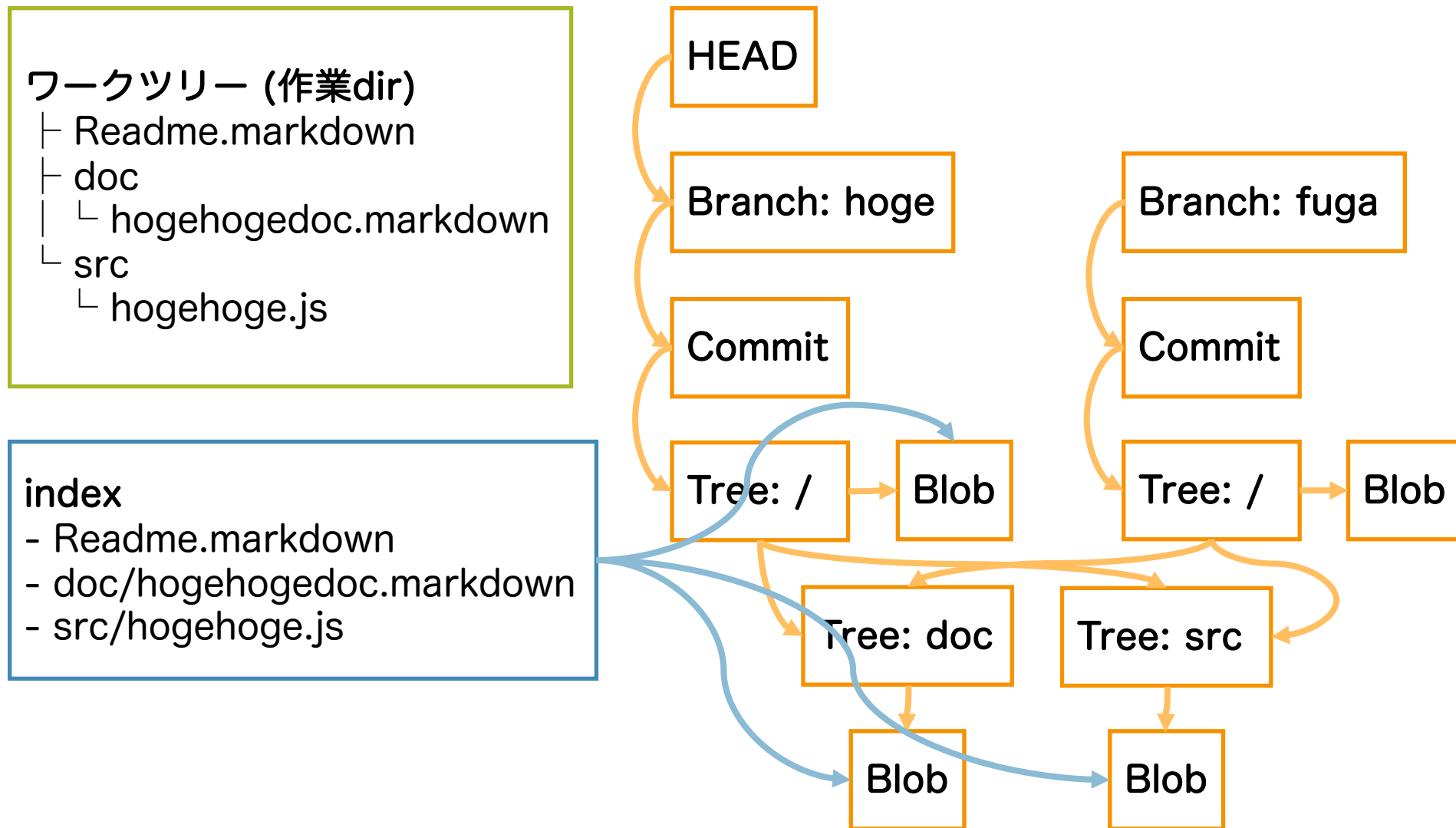
# コミットを取り出す

---

- `git reset` ... branch, インデックス を更新

```
$ git reset fuga
```

# コミットを取り出す



# コミットを取り出す

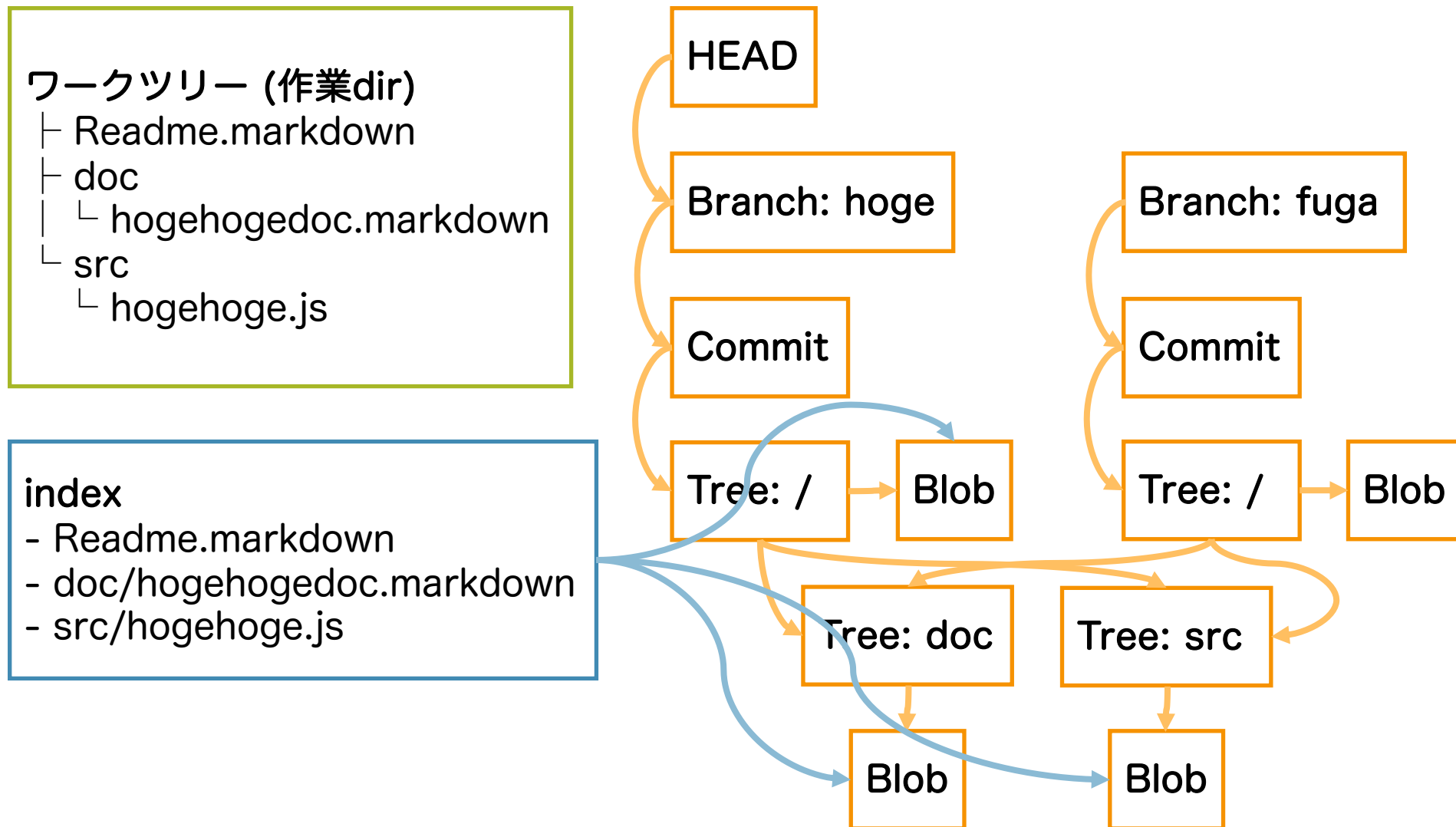
---

- `git reset` ... **branch**, インデックス を更新

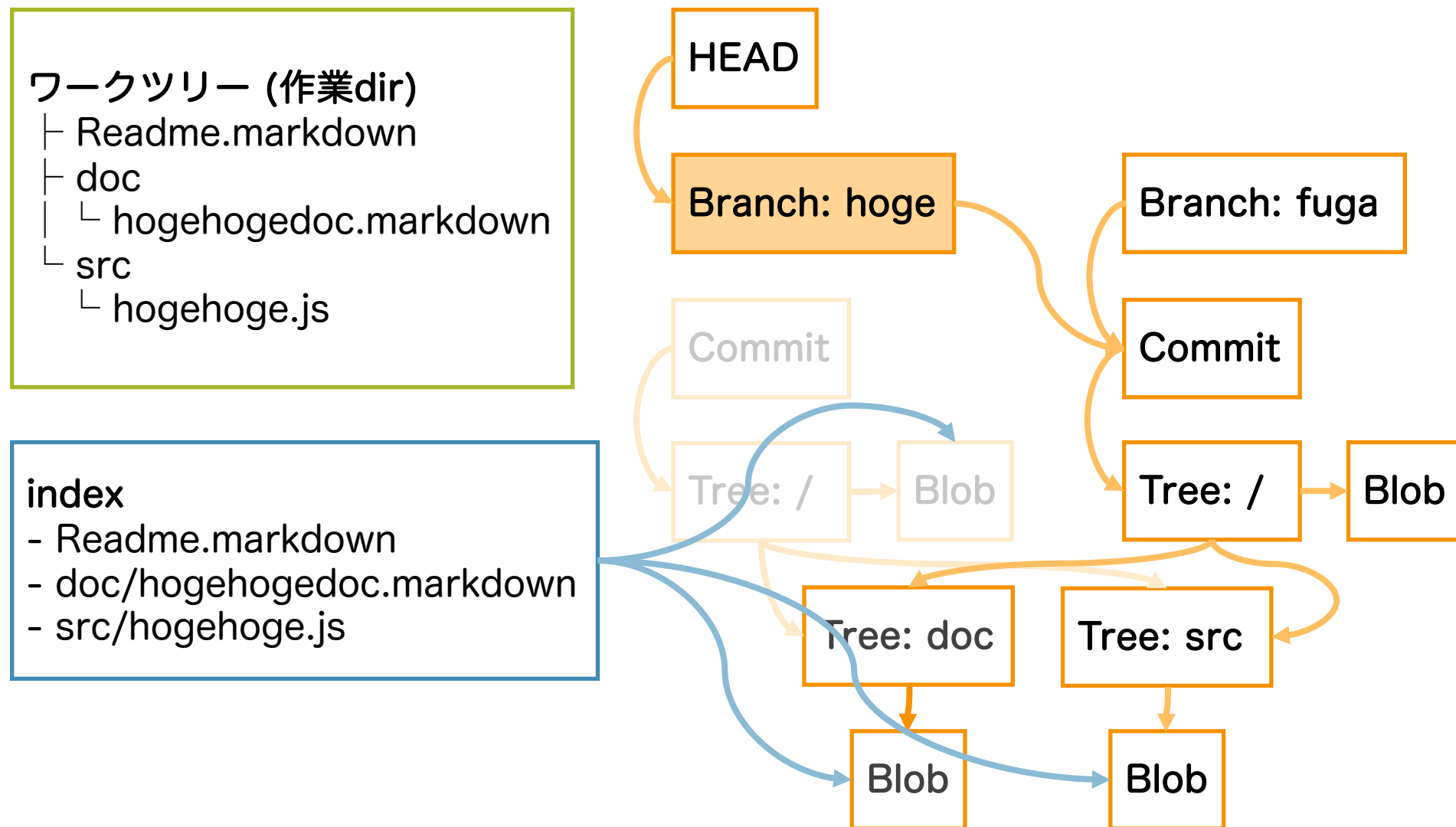
```
$ git reset fuga
```



# コミットを取り出す



# コミットを取り出す



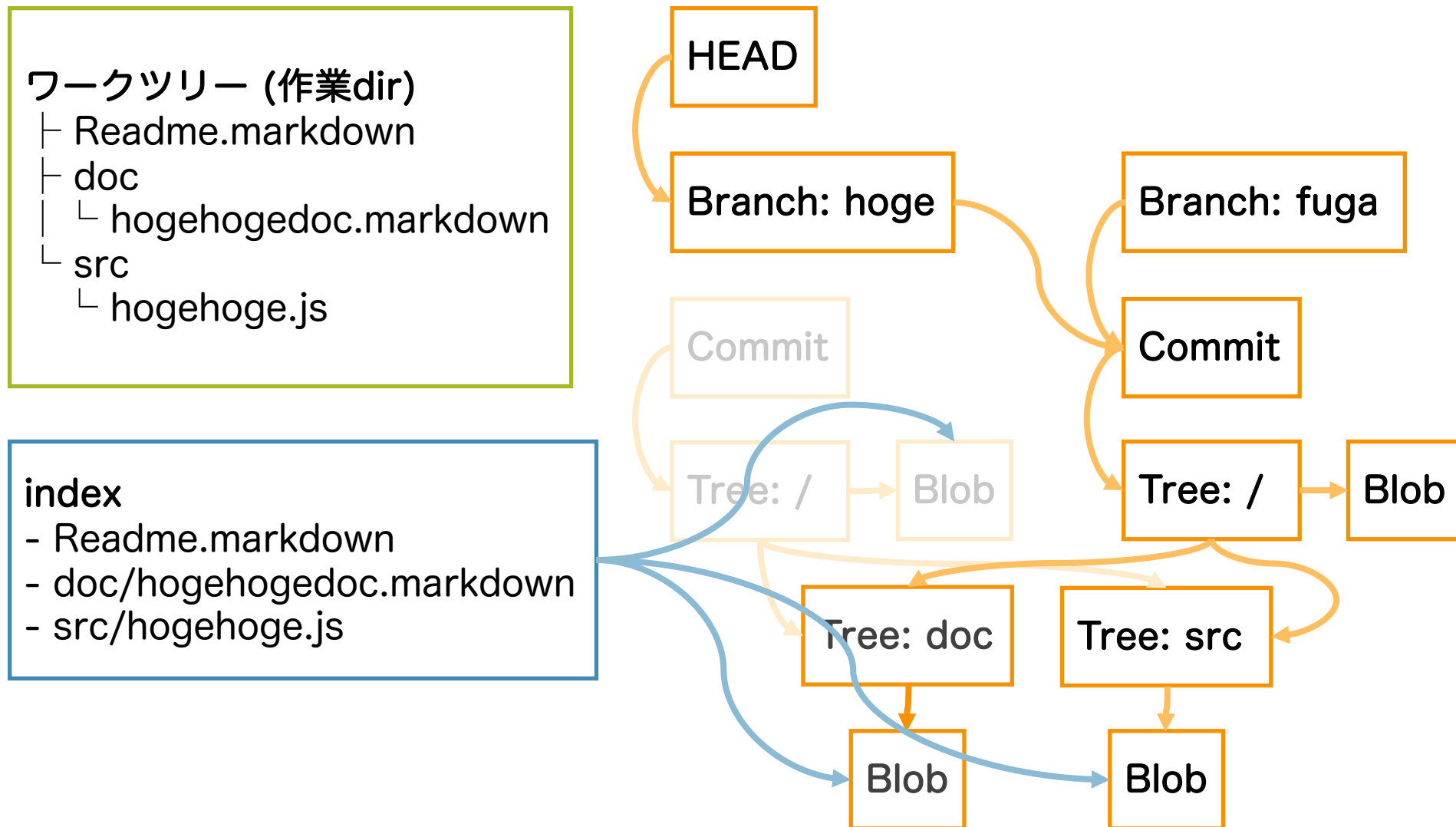
# コミットを取り出す

---

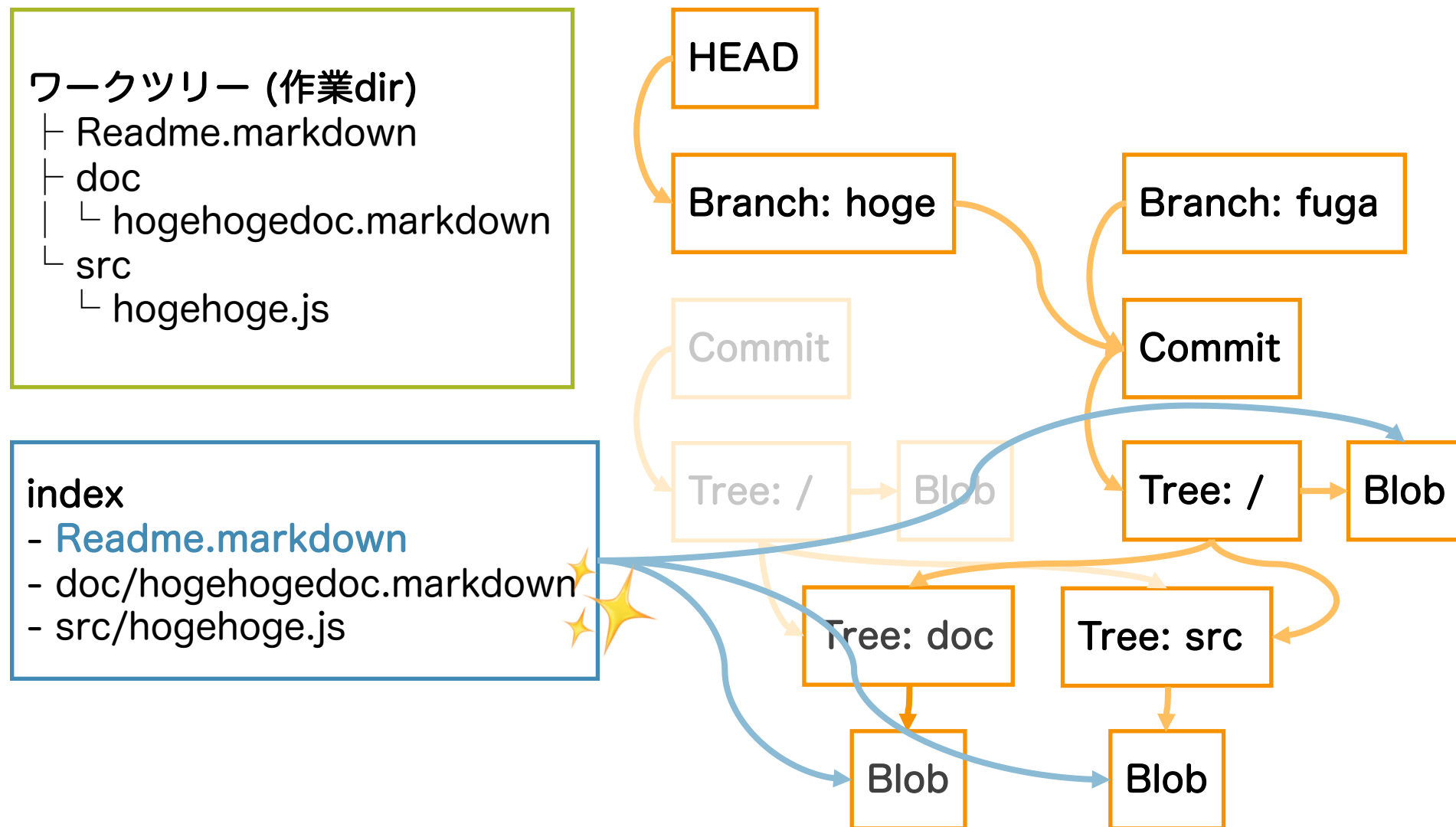
- `git reset` ... branch, インデックス を更新

```
$ git reset fuga
```

# コミットを取り出す



# コミットを取り出す



# コミットを取り出す

---

- ・ `git reset` ... branch, インデックス を更新

```
$ git reset fuga
```

- ・ hoge ブランチが fuga ブランチと同一になる

# コミットを取り出す

---

- `git reset` ... branch, インデックス を更新

```
$ git reset fuga
```

- hoge ブランチが fuga ブランチと同一になる
- ただし、ワークツリーは hoge 時点のまま放置

# コミットを取り出す

---

- `git reset` ... branch, インデックス を更新

```
$ git status
On branch master
Changes not staged for commit:

    modified:   Readme.markdown

No changes added to commit
```

- ワークツリーに Readme.markdown の差分が残っている

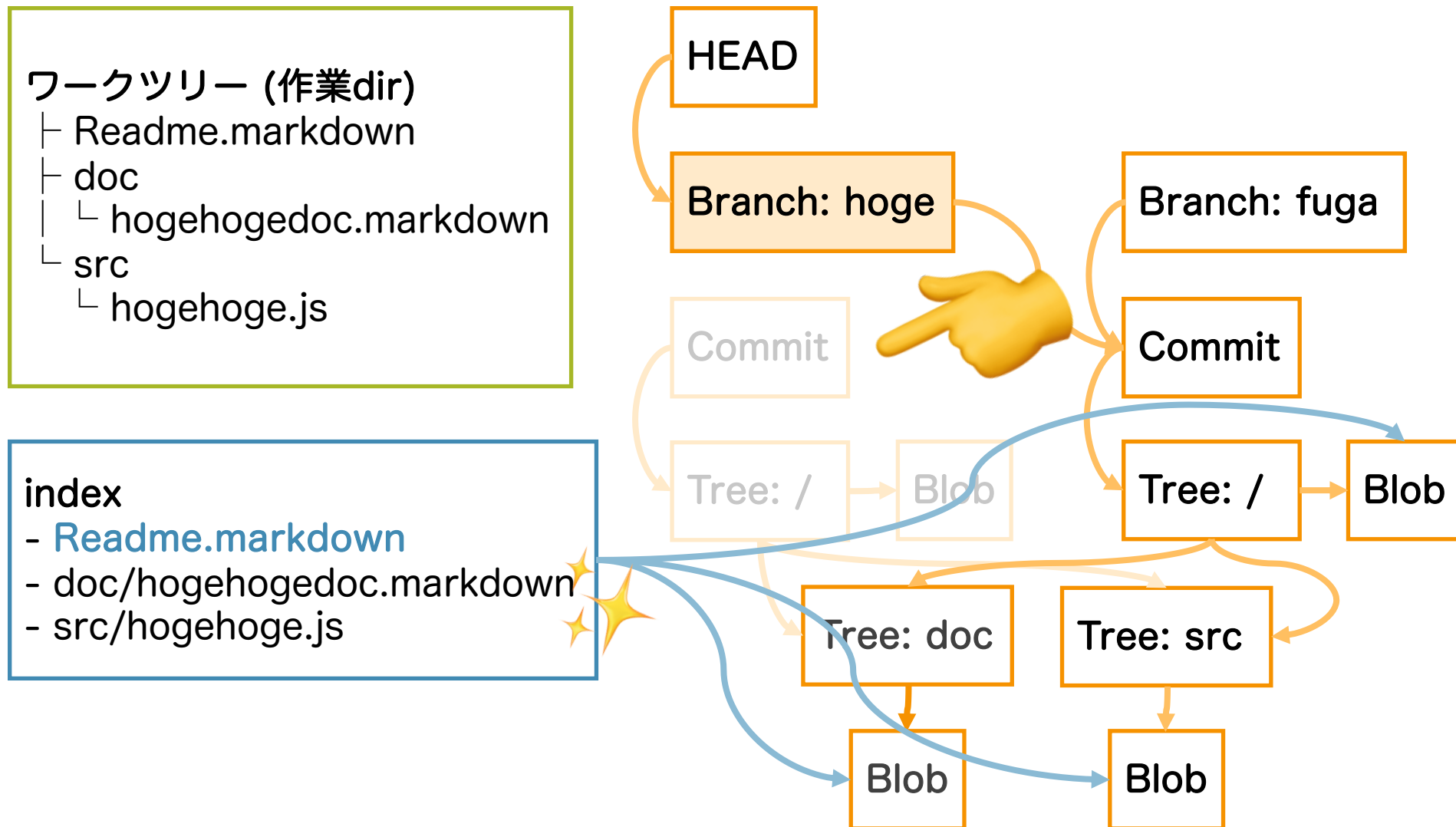


# コミットを取り出す

---

- ・ ちなみに

# コミットを取り出す



# コミットを取り出す

---

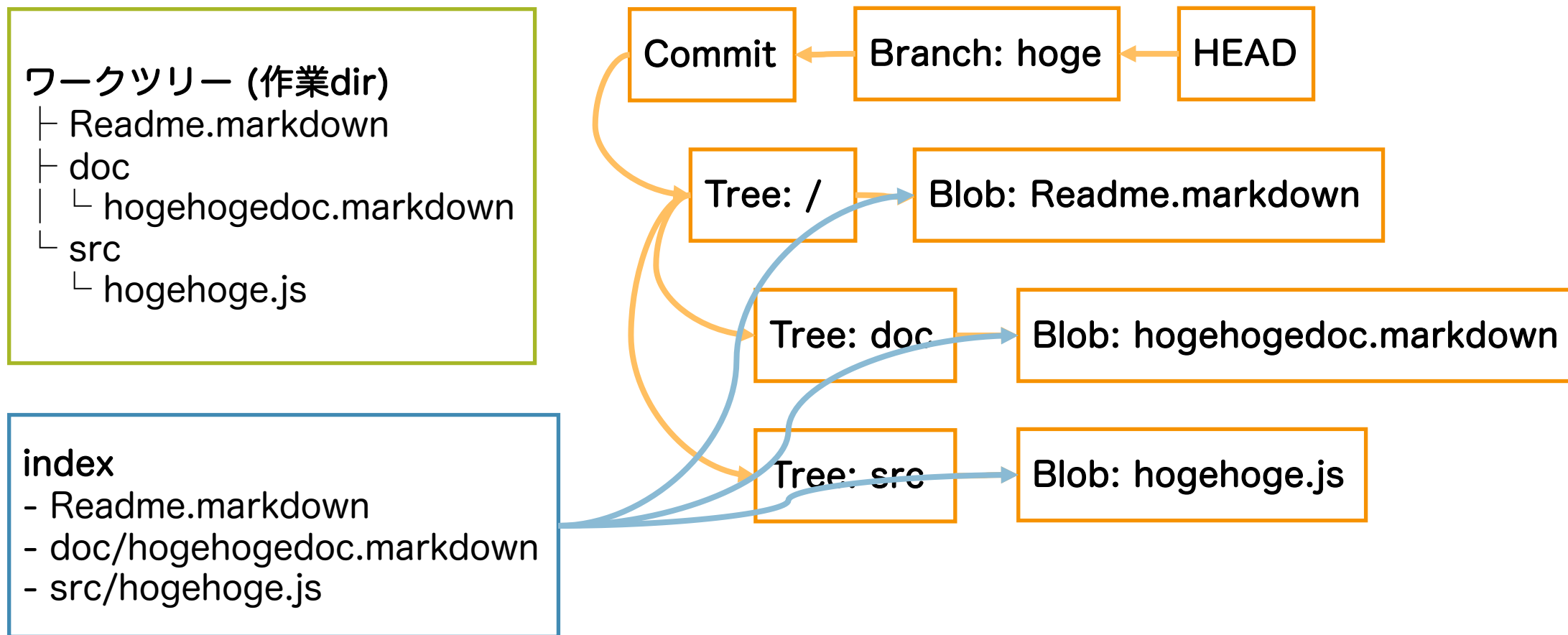
- ・ ちなみに
  - ・ ごみオブジェクトは一定期間後に削除 (GC) される

# コミットを取り出す

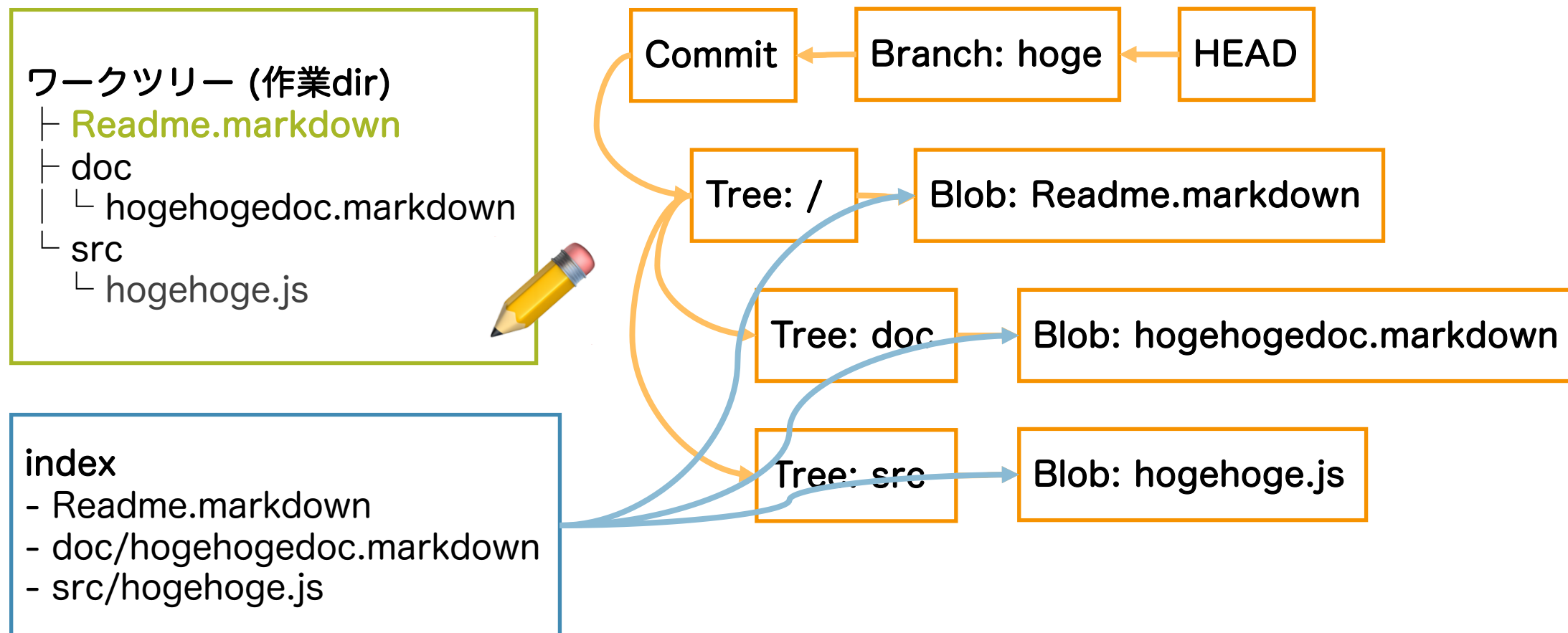
---

- ・`git reset` の活用例 (1) `git add` のキャンセル

# コミットを取り出す



# コミットを取り出す



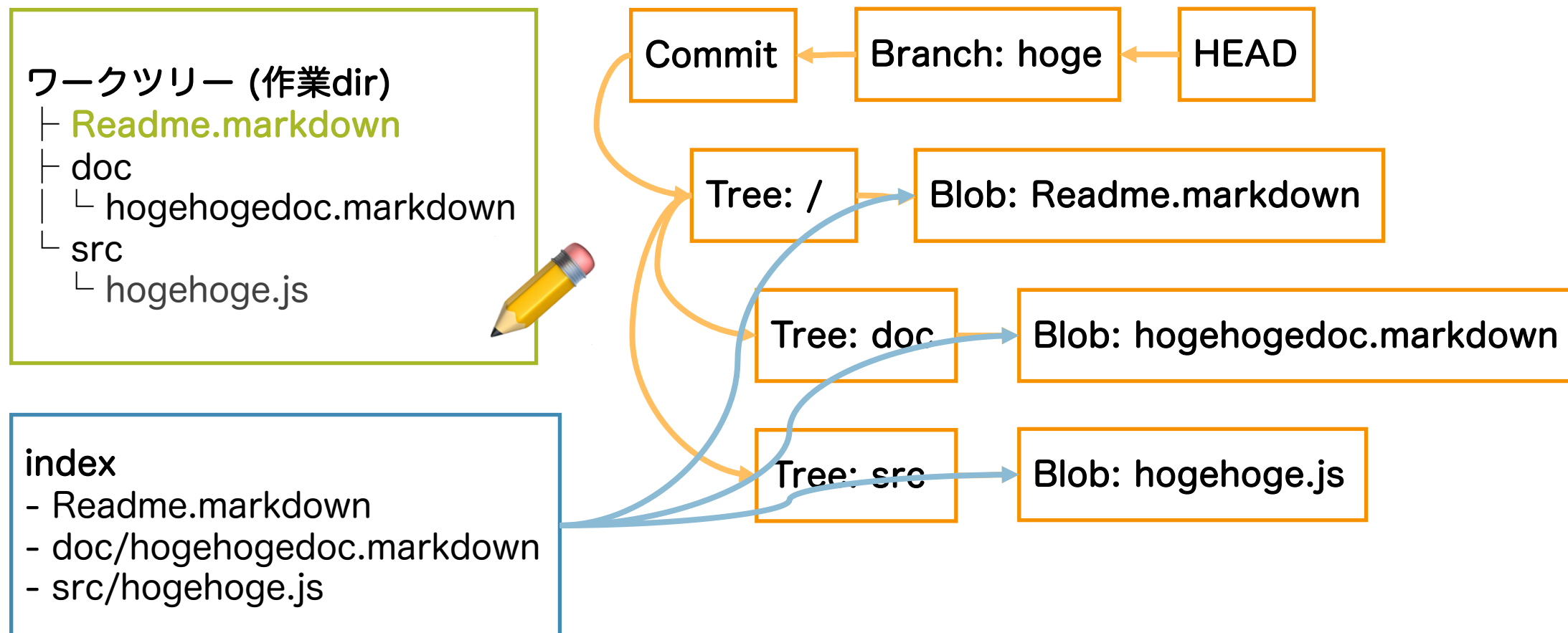
# コミットを取り出す

---

- ・ `git reset` の活用例 (1) `git add` のキャンセル

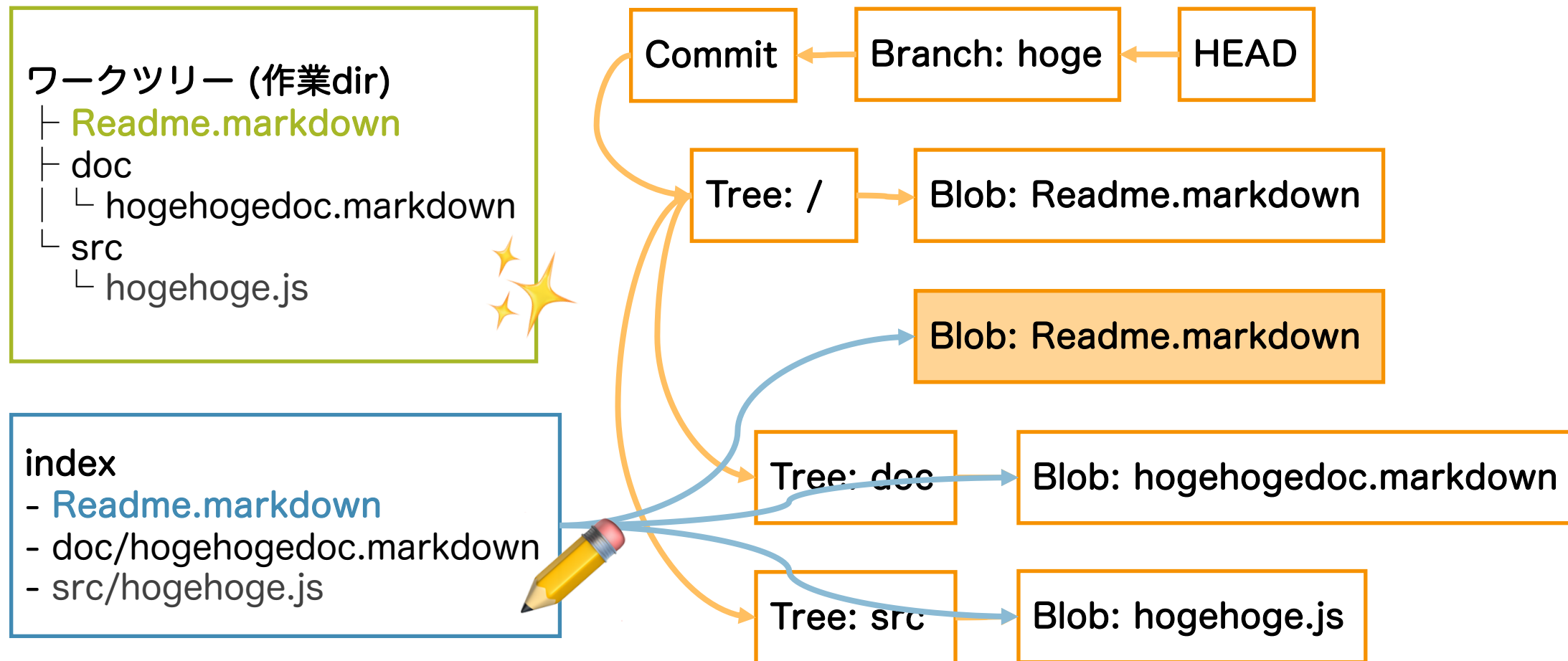
```
$ git add Readme.markdown
```

# コミットを取り出す





# コミットを取り出す



# コミットを取り出す

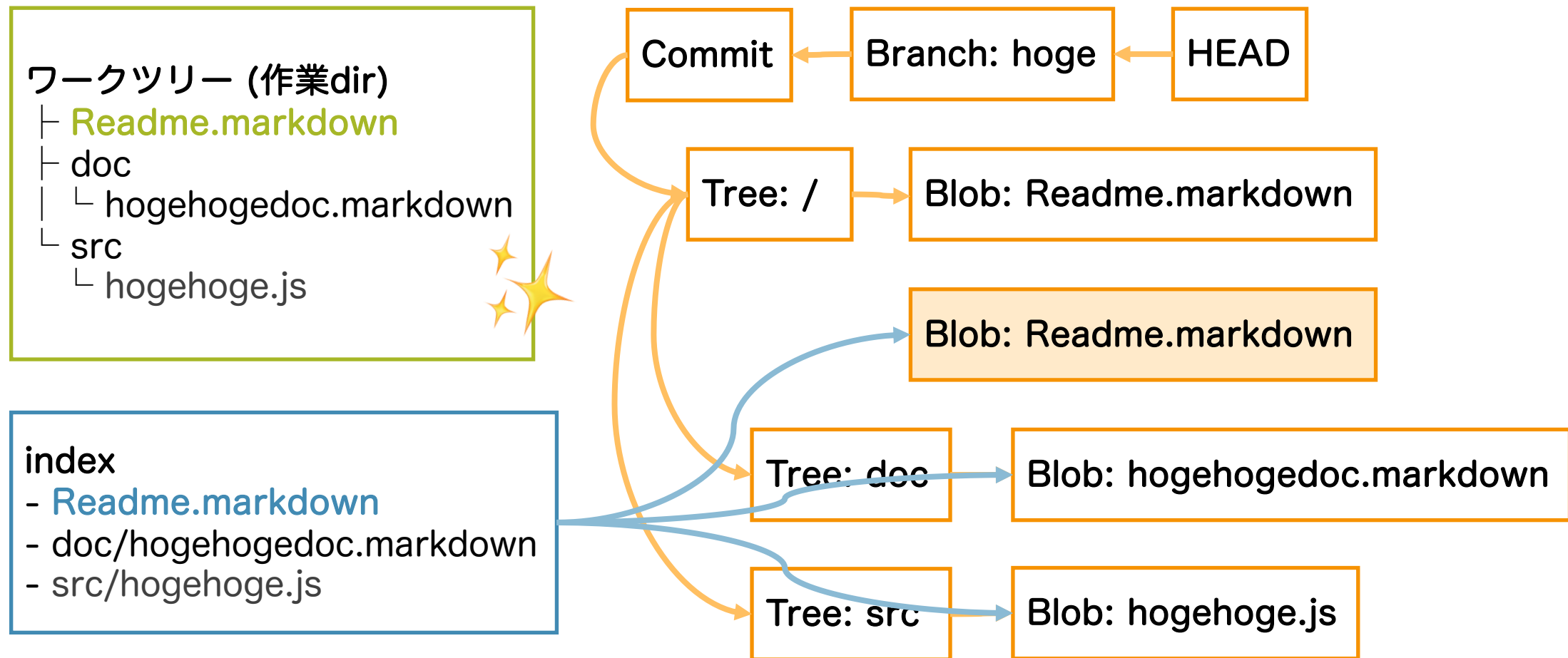
---

- ・ `git reset` の活用例 (1) `git add` のキャンセル

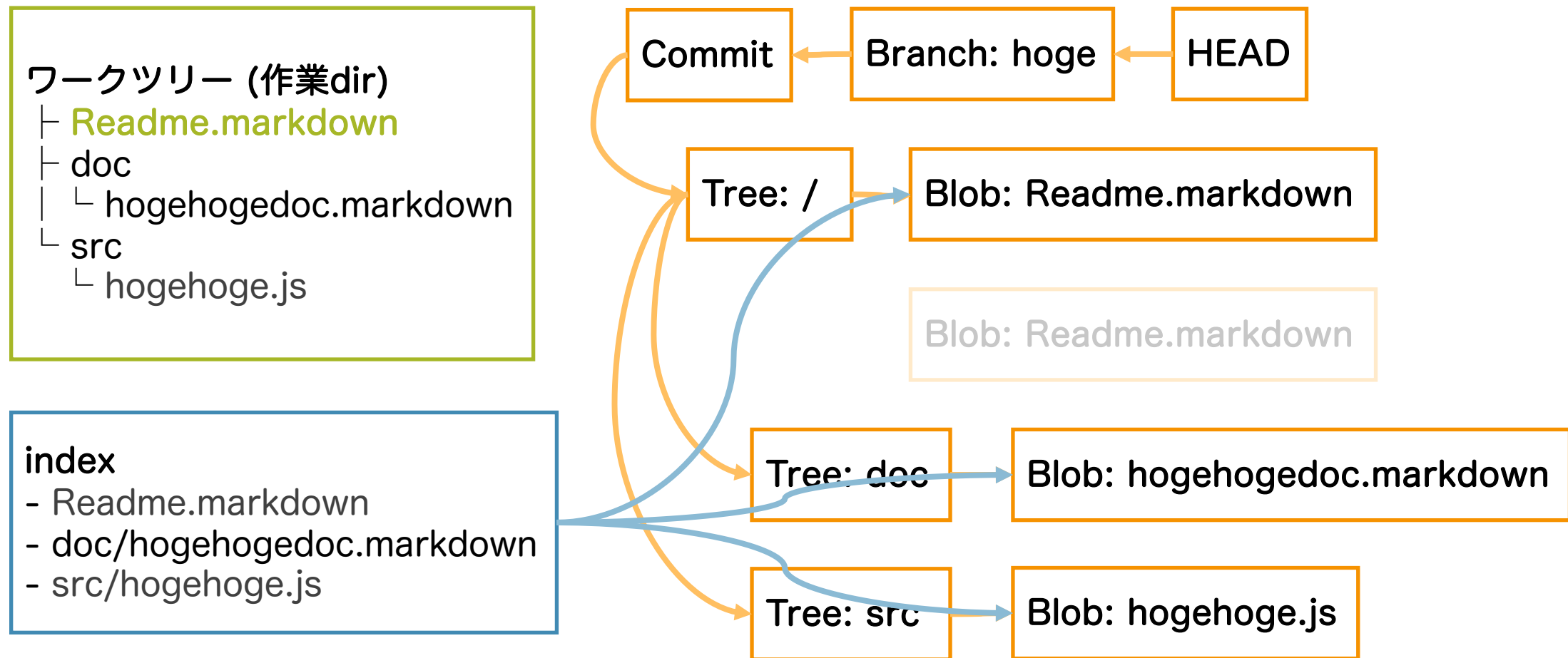
```
$ git reset HEAD
```

- ・ branch の向き先が同じなので、インデックスだけ復元される

# コミットを取り出す



# コミットを取り出す



## コミットを取り出す

---

- ・ ``git reset`` の活用例 (1) ``git add`` のキャンセル
  - ・ ``git add`` で更新されたインデックスが元に戻る

# コミットを取り出す

---

- ・ `git reset` の活用例 (1) `git add` のキャンセル
  - ・ ちなみに

```
$ git reset
```

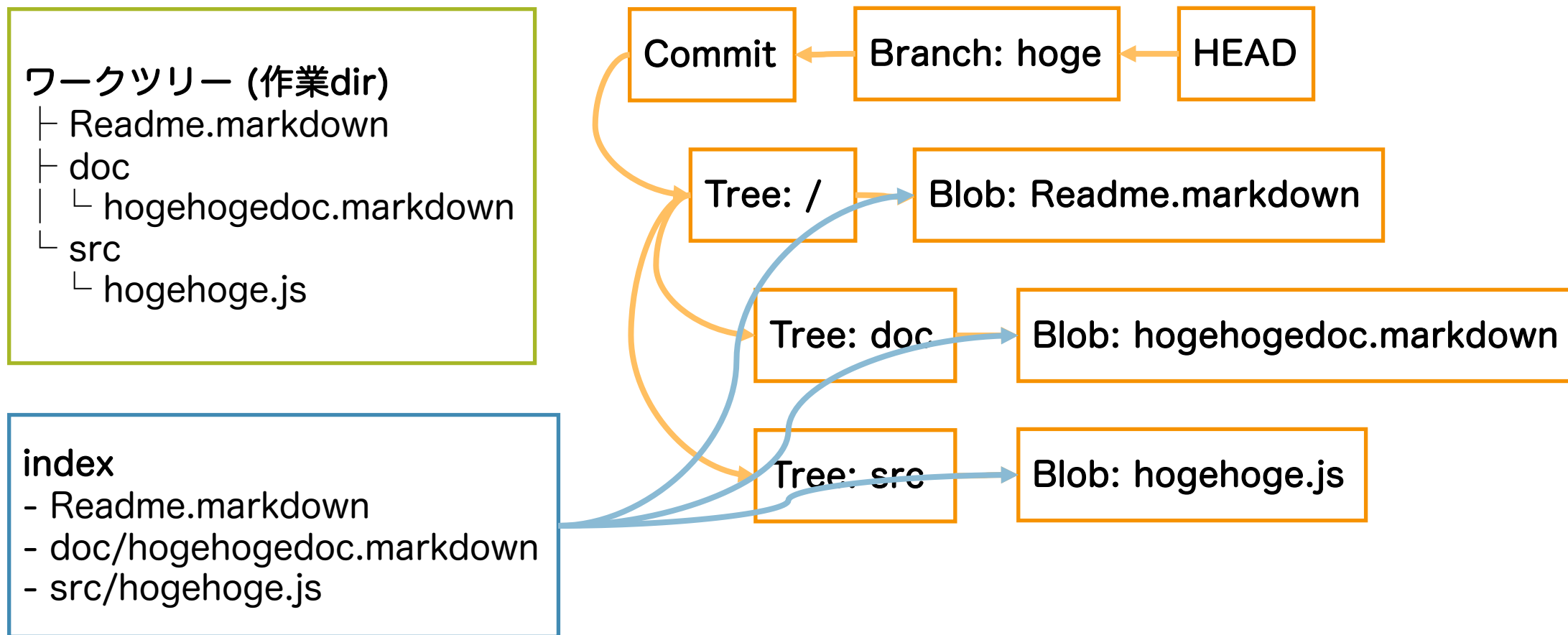
- ・ `git reset HEAD` はよく使うので省略できる

# コミットを取り出す

---

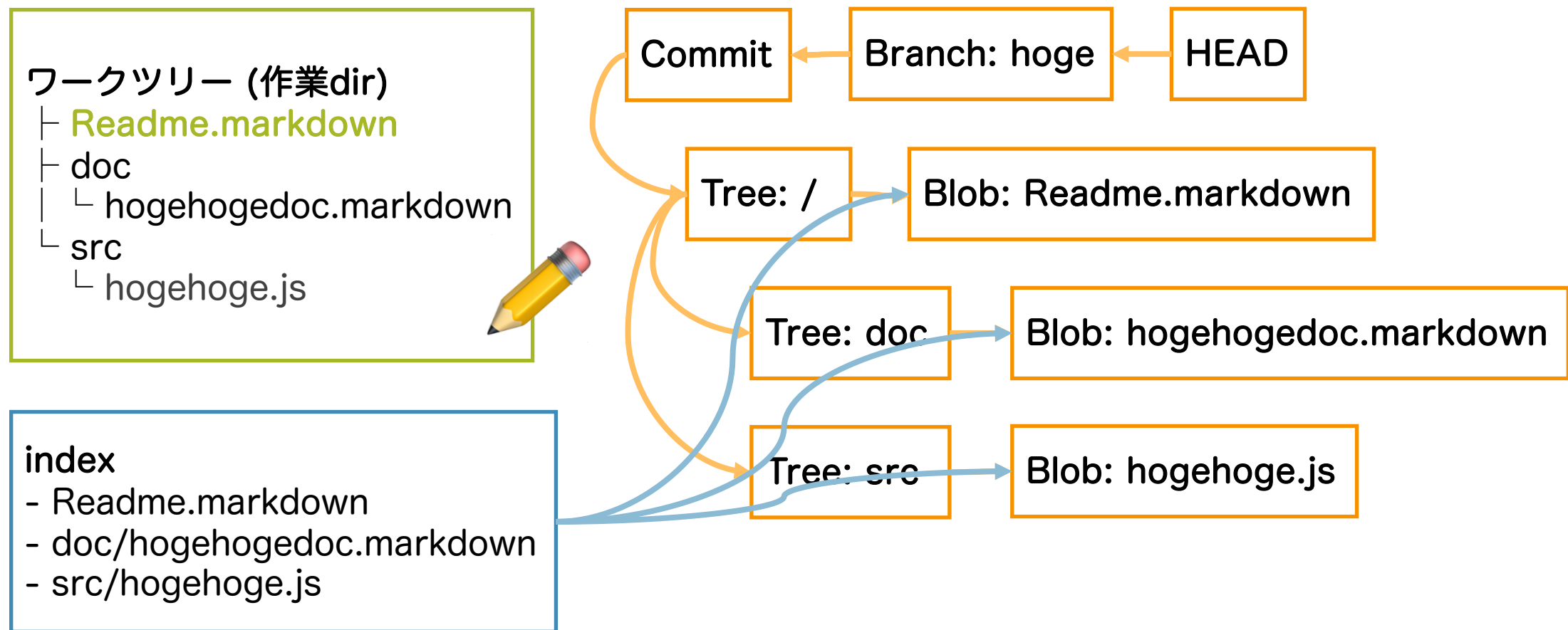
- ・ `git reset` の活用例 (2) `git commit` のキャンセル

# コミットを取り出す

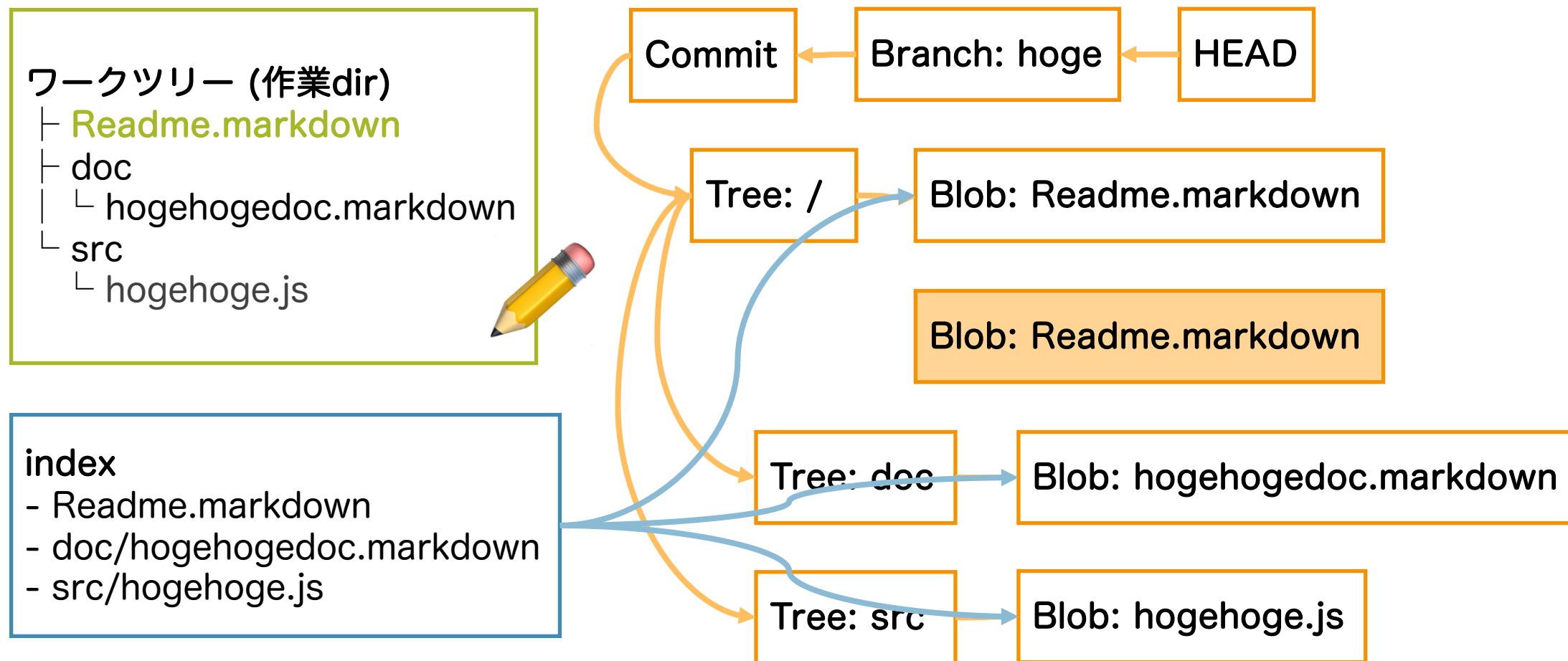




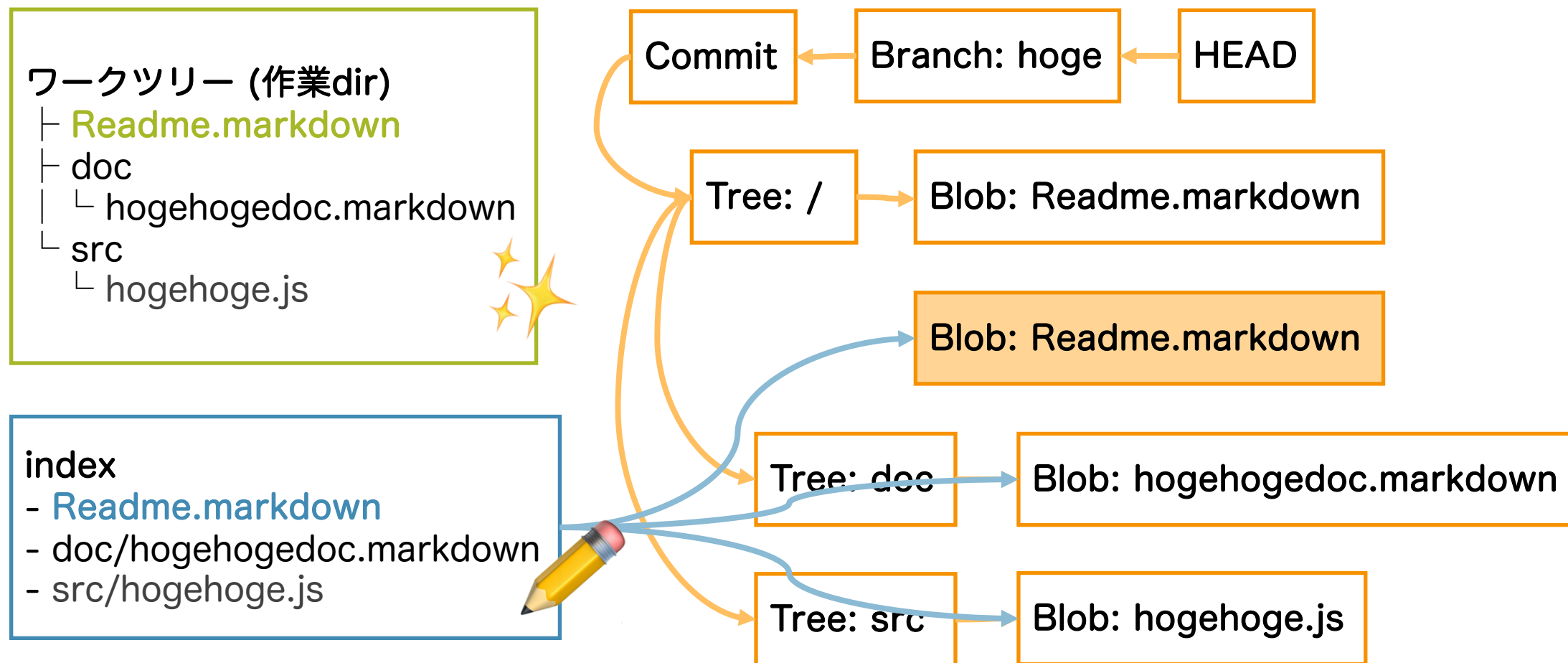
# コミットを取り出す



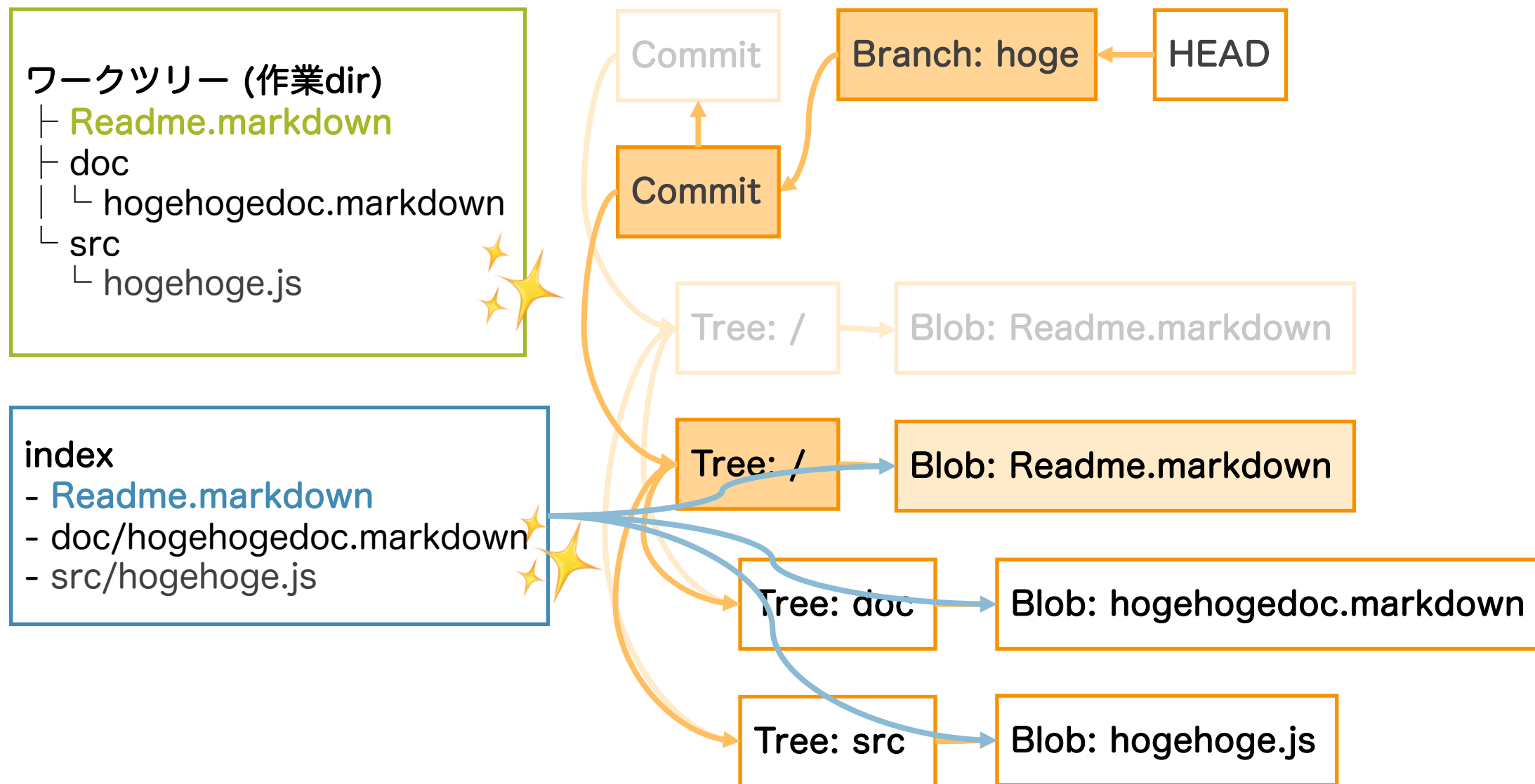
# コミットを取り出す



# コミットを取り出す



# コミットを取り出す



# コミットを取り出す

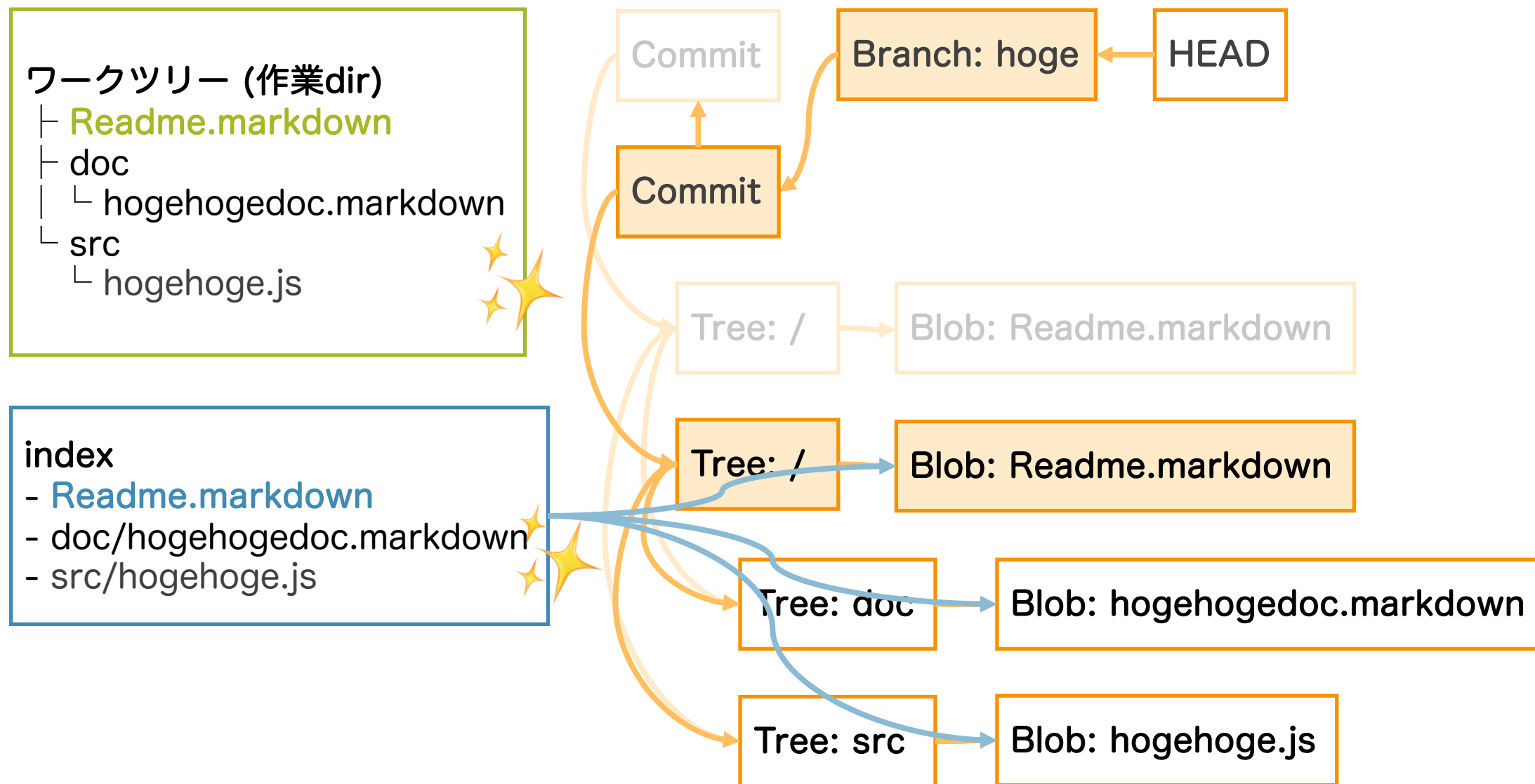
---

- ・ `git reset` の活用例 (2) `git commit` のキャンセル

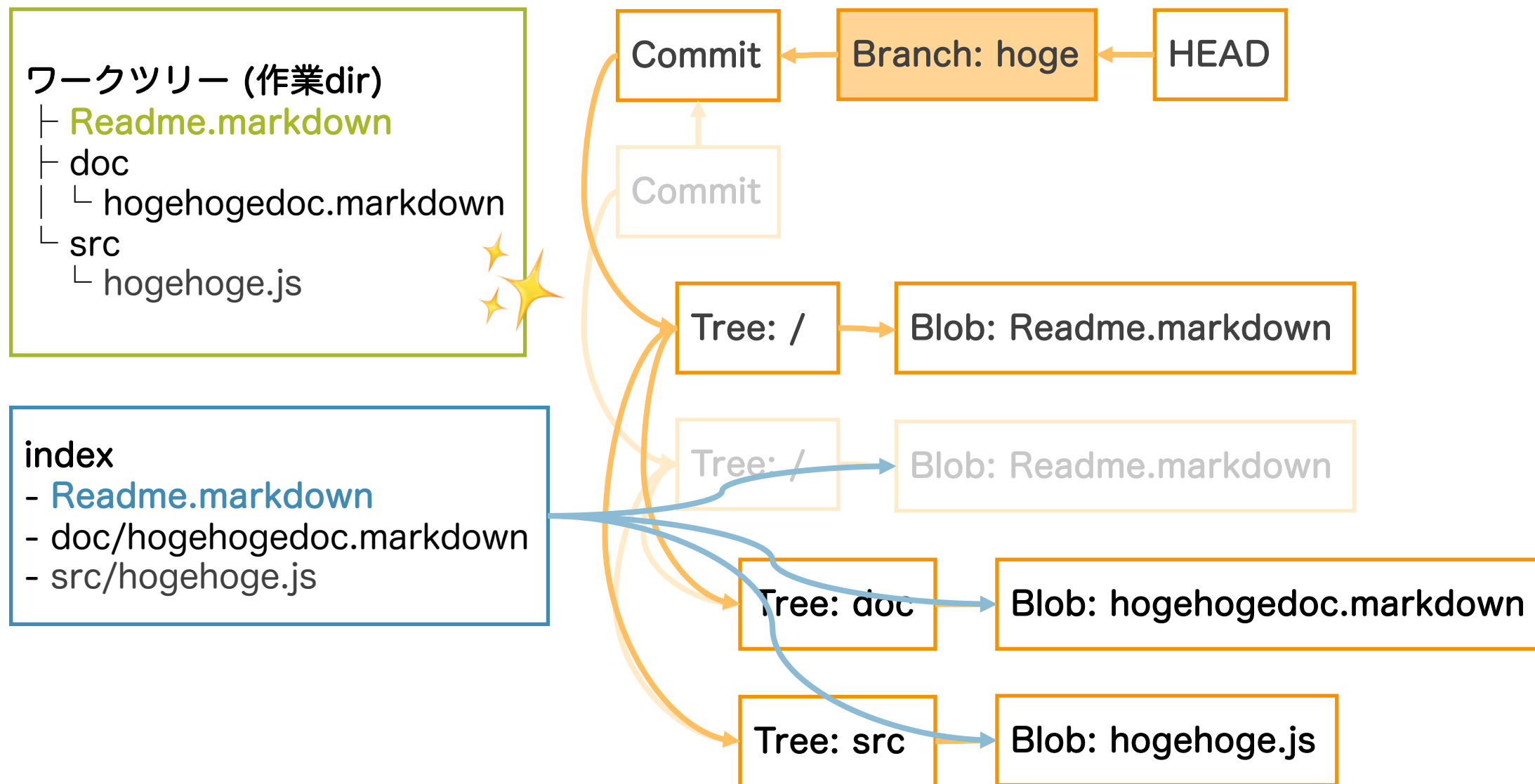
```
$ git reset HEAD^
```

- ・ HEAD^ ... HEAD の一つ前 (ID ベタ書きでも もちろん ok)

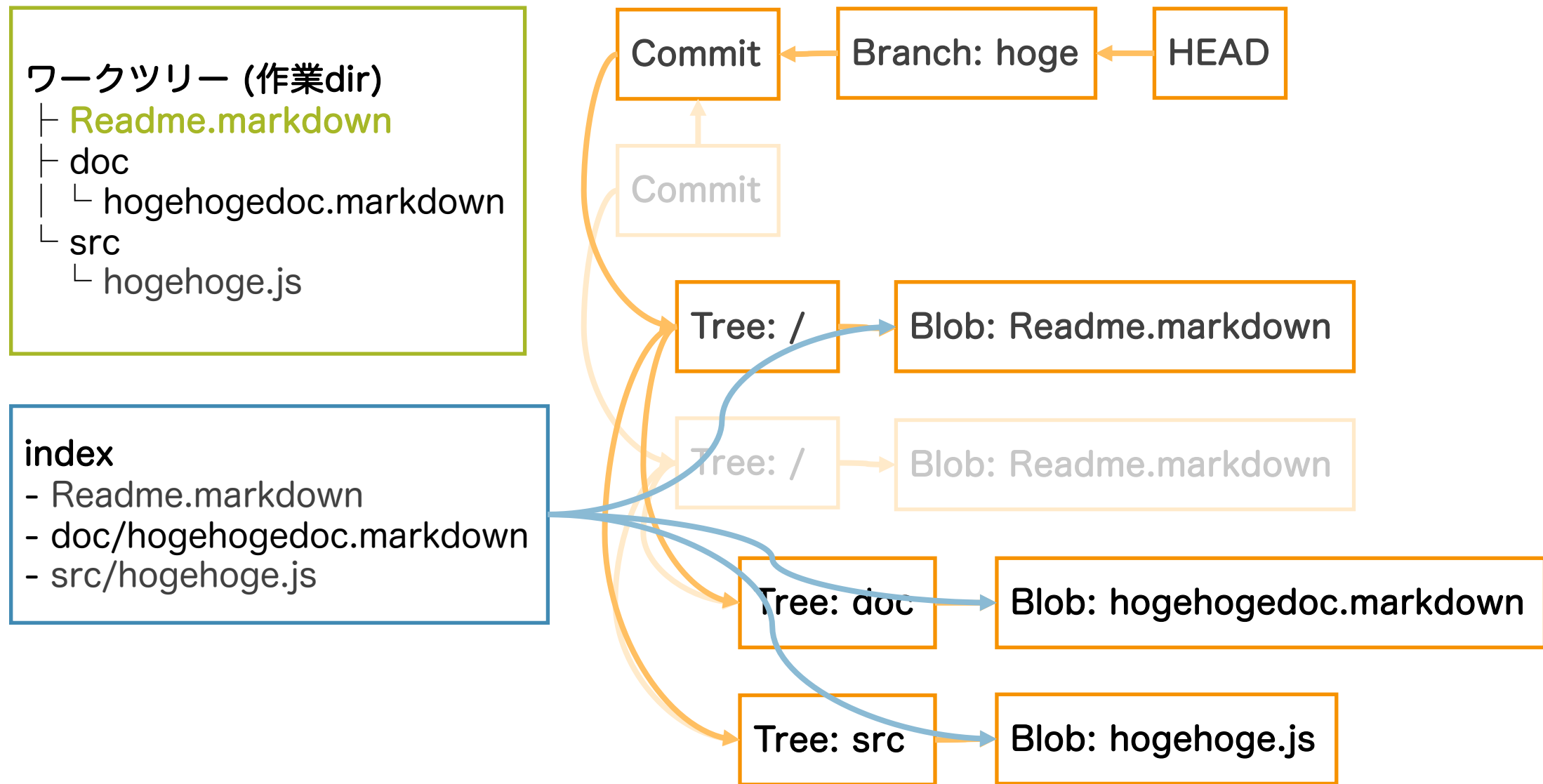
# コミットを取り出す



# コミットを取り出す



# コミットを取り出す





# コミットを取り出す

---

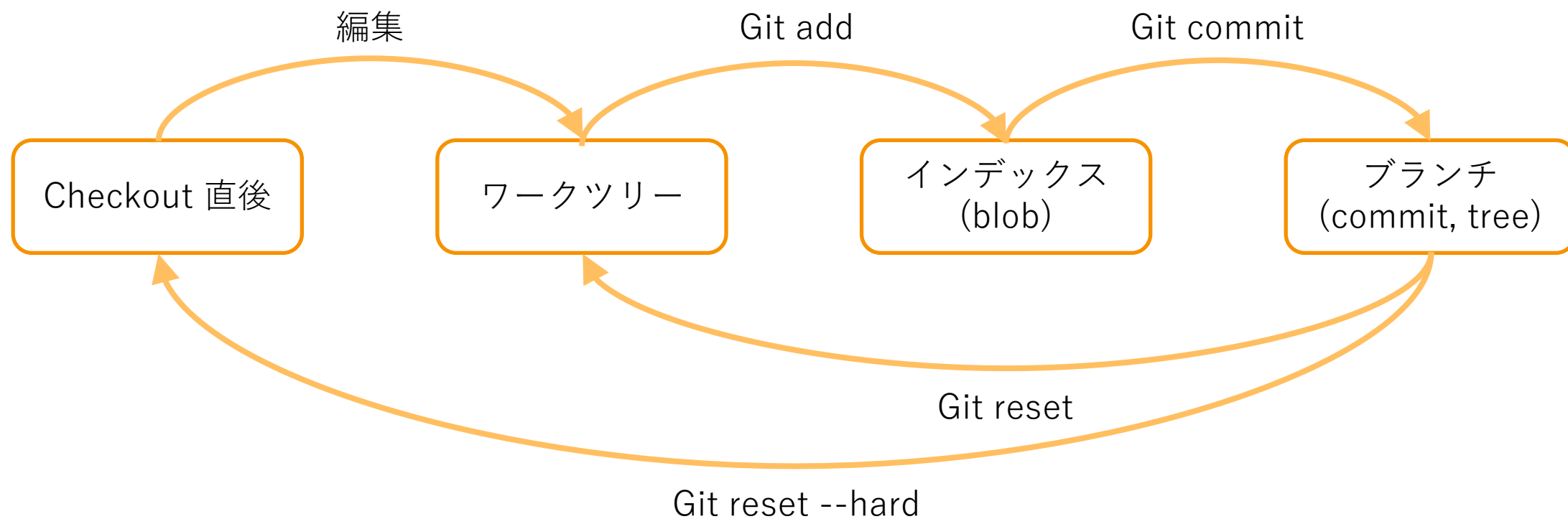
- ・危険だけど便利な --hard オプション

```
$ git reset --hard HEAD^
```

- ・ワークツリーも上書きする
- ・途中まで編集したけどやっぱりいらなくなった場合などに

# コミットを取り出す

---



歴史を合成する

# 歴史を合成する

---

- ここからは、それぞれの Git コマンドの挙動を見ていく
  - commit (, add)
  - checkout, reset
  - merge
  - rebase (, cherry-pick, revert)

# 歴史を合成する

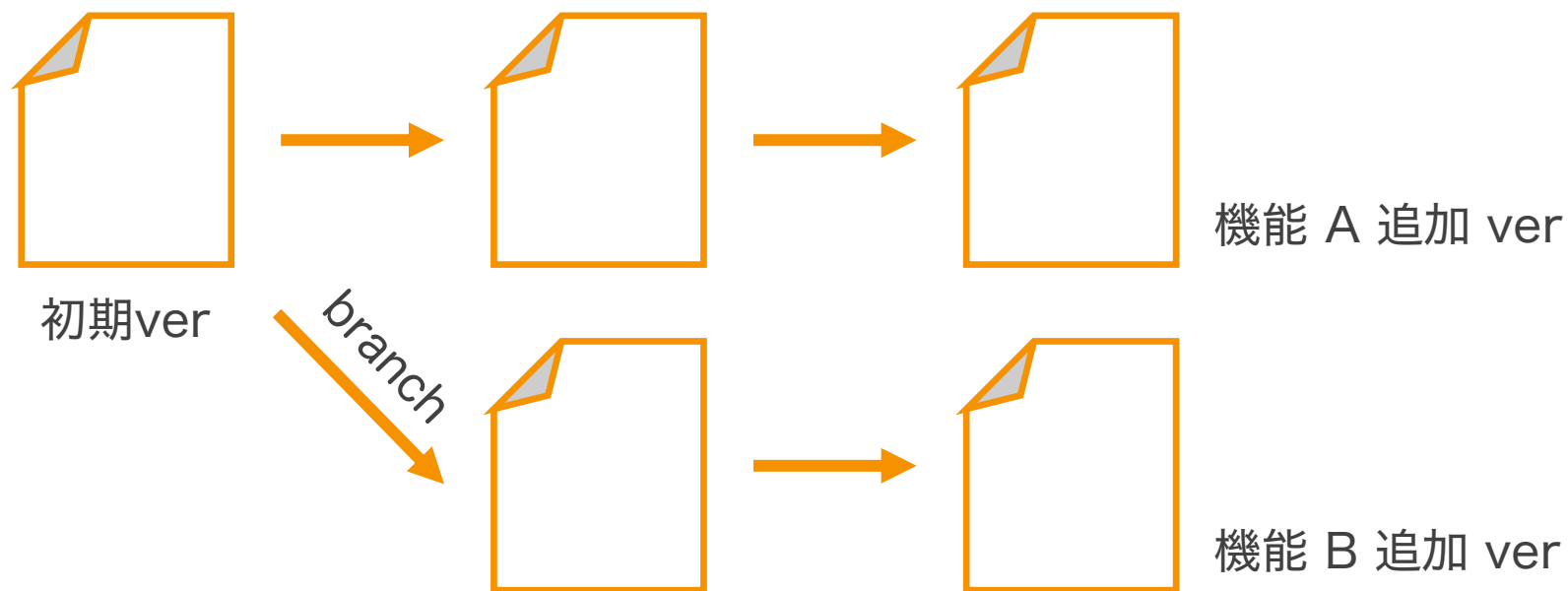
---

- ・ 複数のブランチで並行してやった作業を統合する (してくれる)

# 歴史を合成する

---

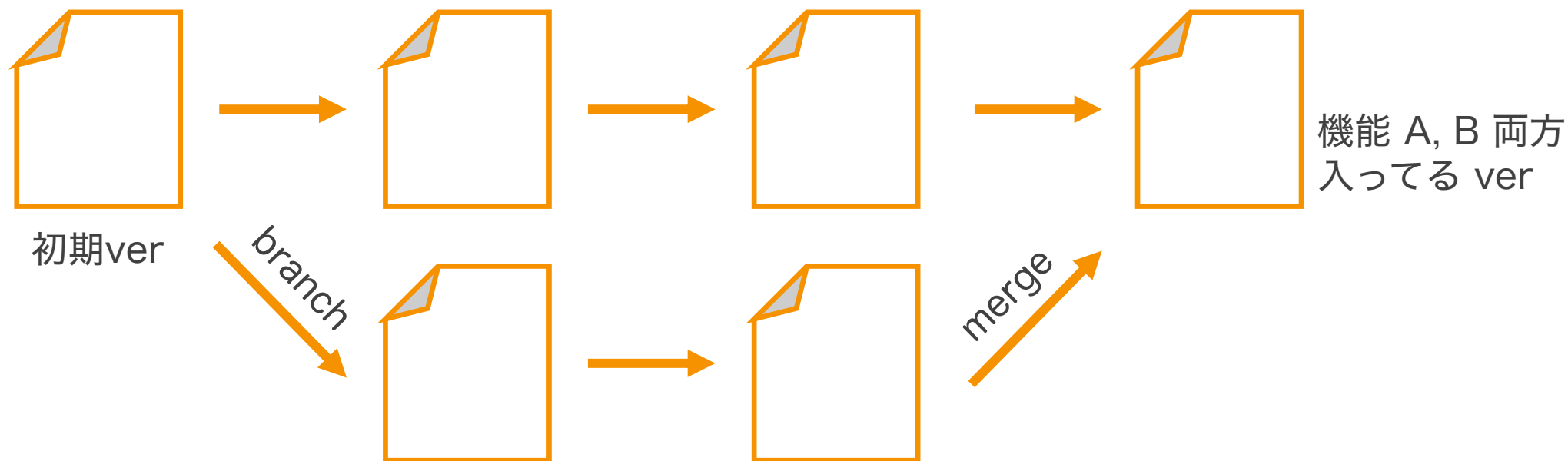
- ・ 複数のブランチで並行してやった作業を統合する (してくれる)



# 歴史を合成する

---

- 複数のブランチで並行してやった作業を統合する (してくれる)



# 歴史を合成する

---

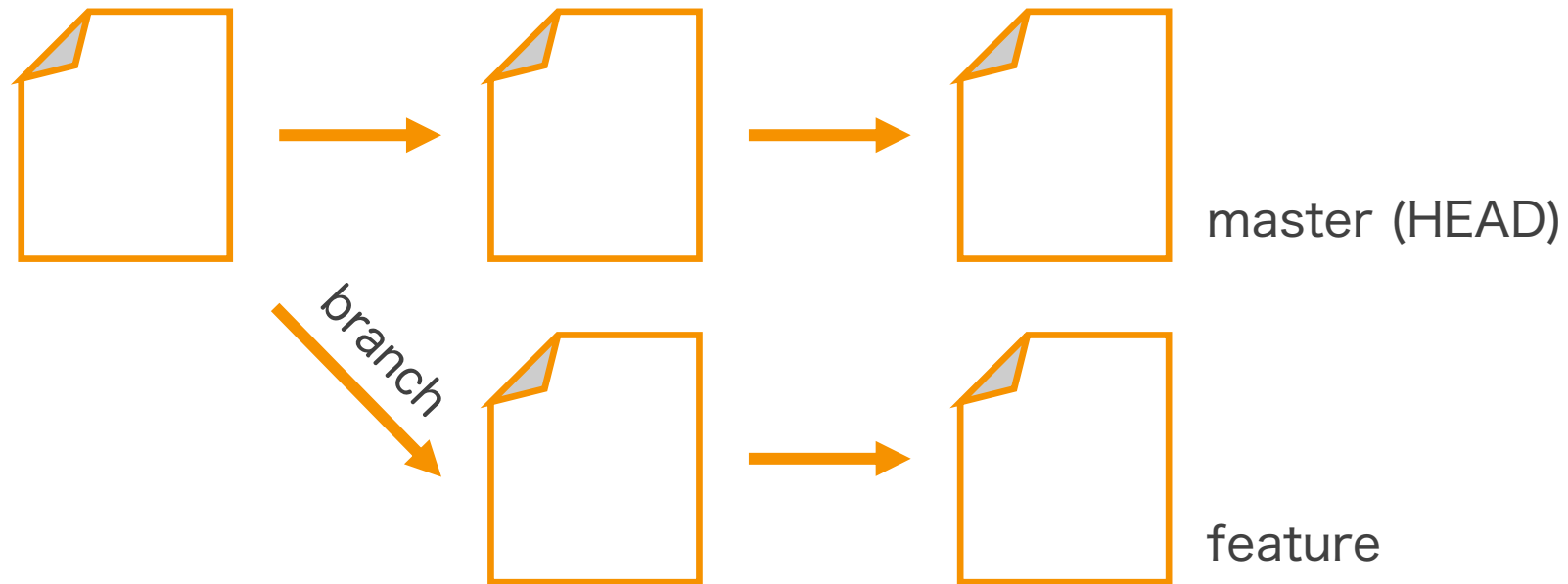
- ・ Merge コマンドの挙動



# 歴史を合成する

---

- Merge コマンドの挙動



# 歴史を合成する

---

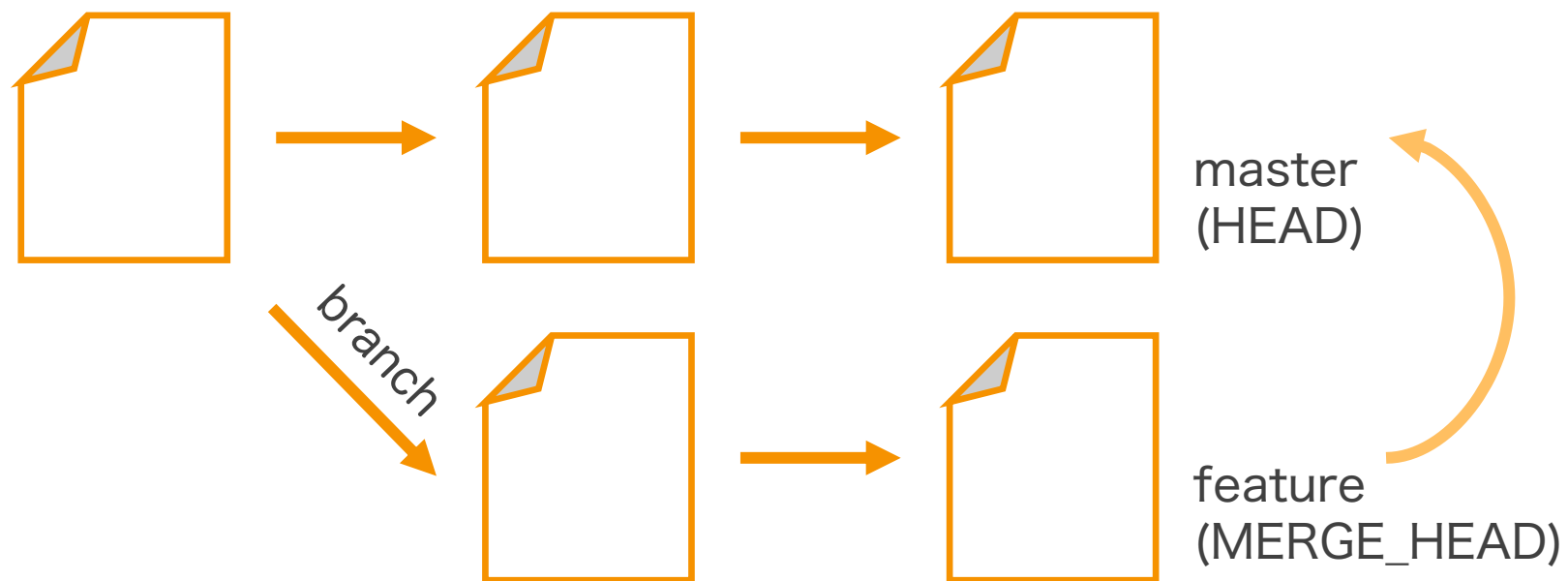
- Merge コマンドの挙動

```
$ git merge feature
```

# 歴史を合成する

---

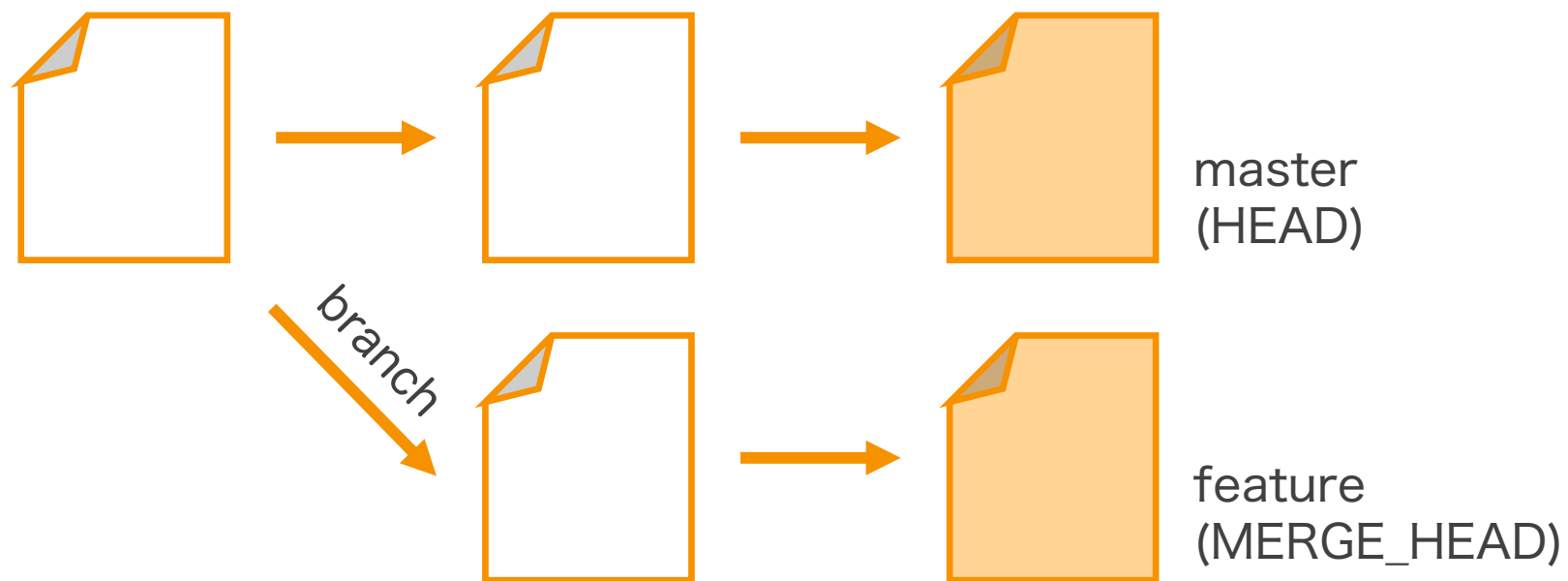
- Merge コマンドの挙動



# 歴史を合成する

---

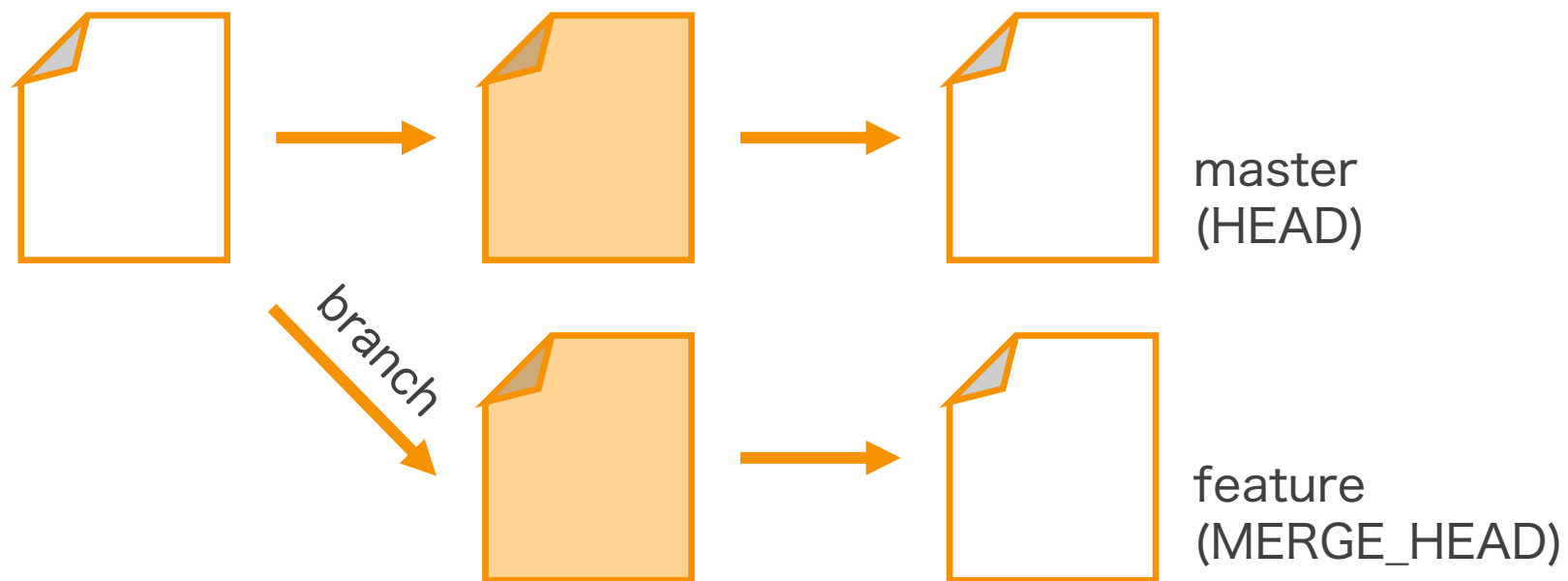
- Merge コマンドの挙動



# 歴史を合成する

---

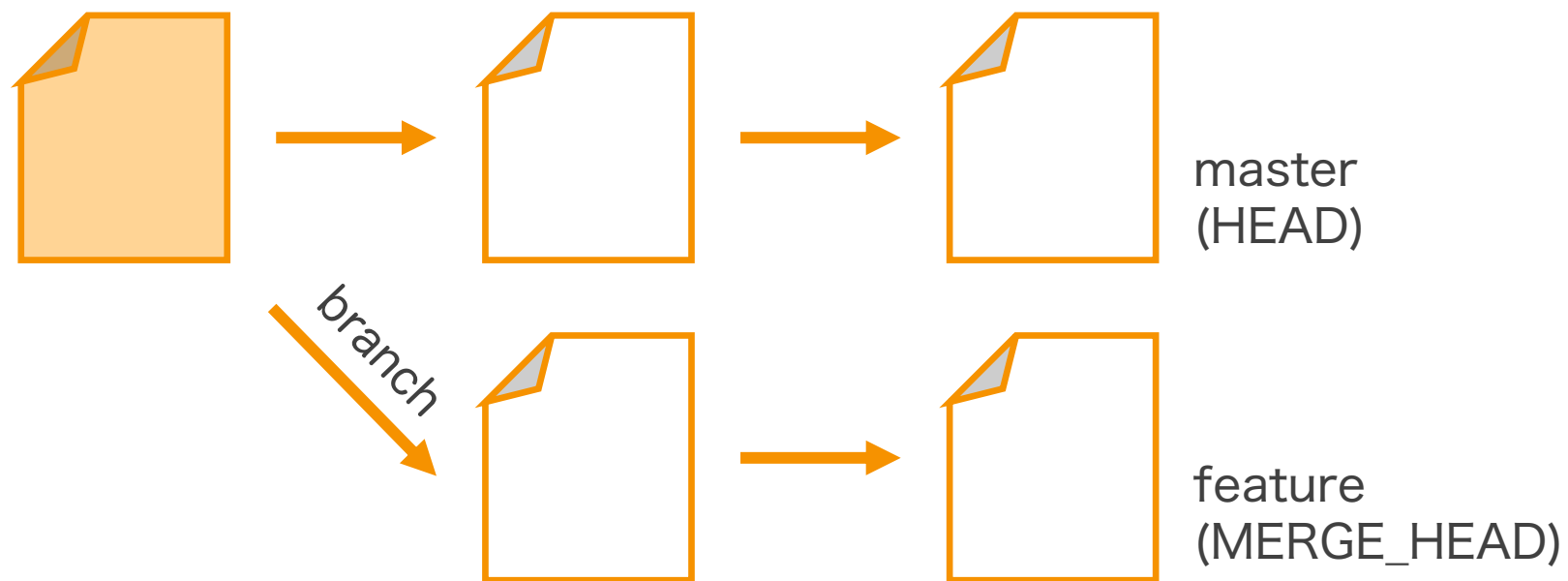
- Merge コマンドの挙動



# 歴史を合成する

---

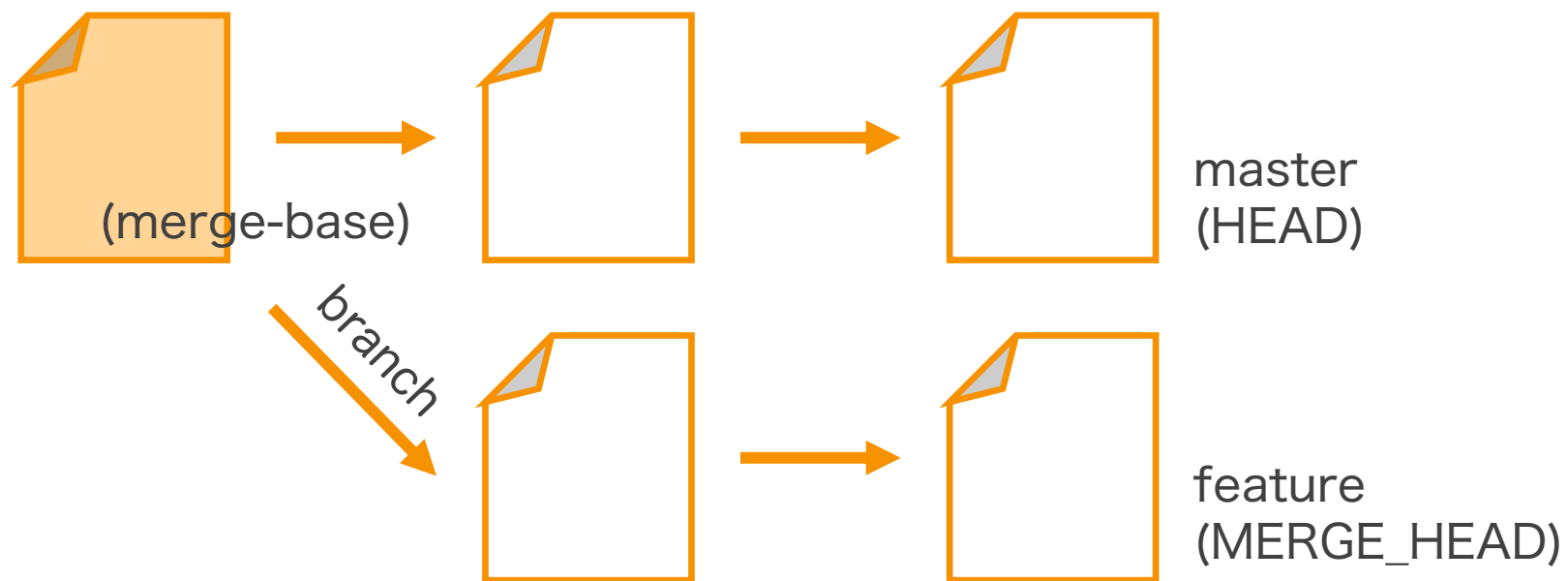
- Merge コマンドの挙動



# 歴史を合成する

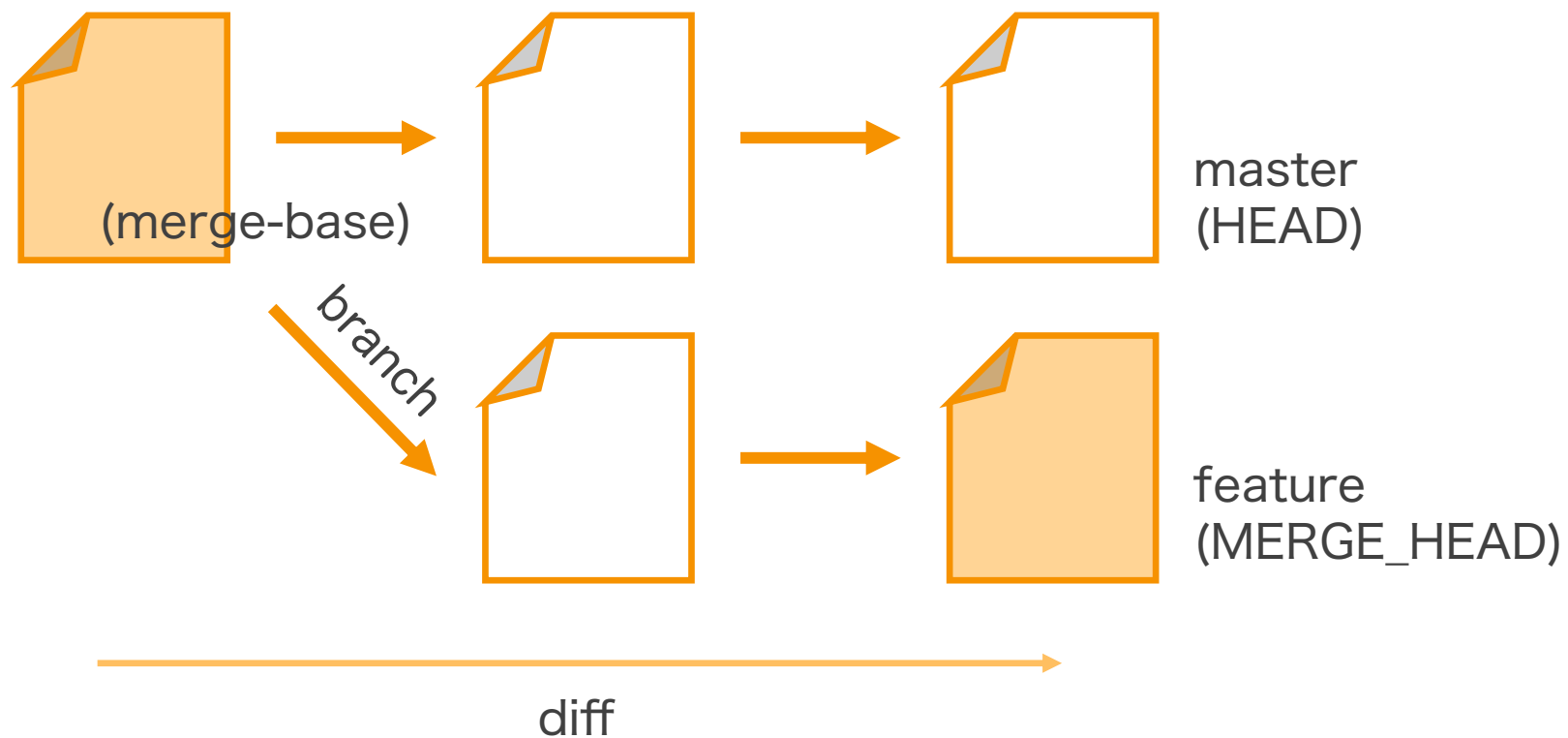
---

- Merge コマンドの挙動



# 歴史を合成する

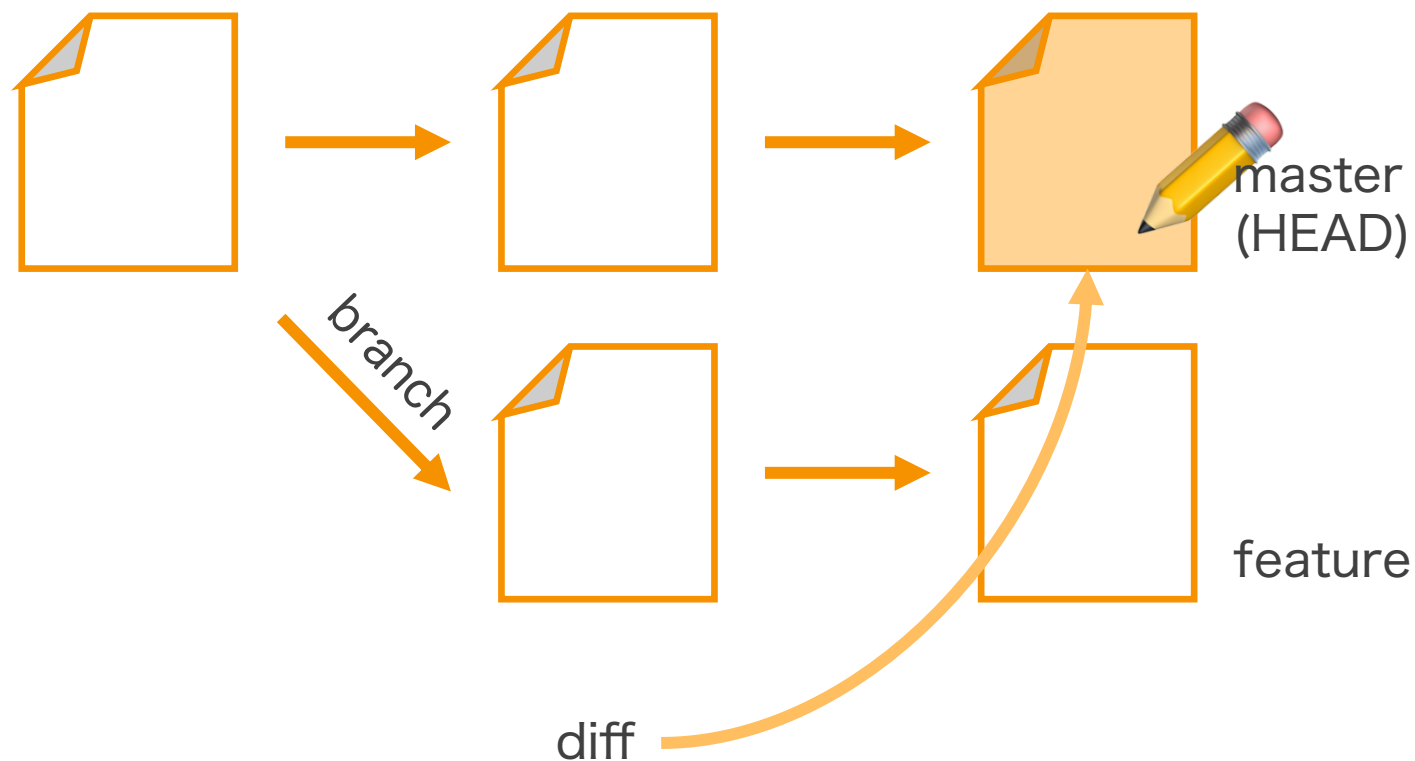
- Merge コマンドの挙動





# 歴史を合成する

- Merge コマンドの挙動

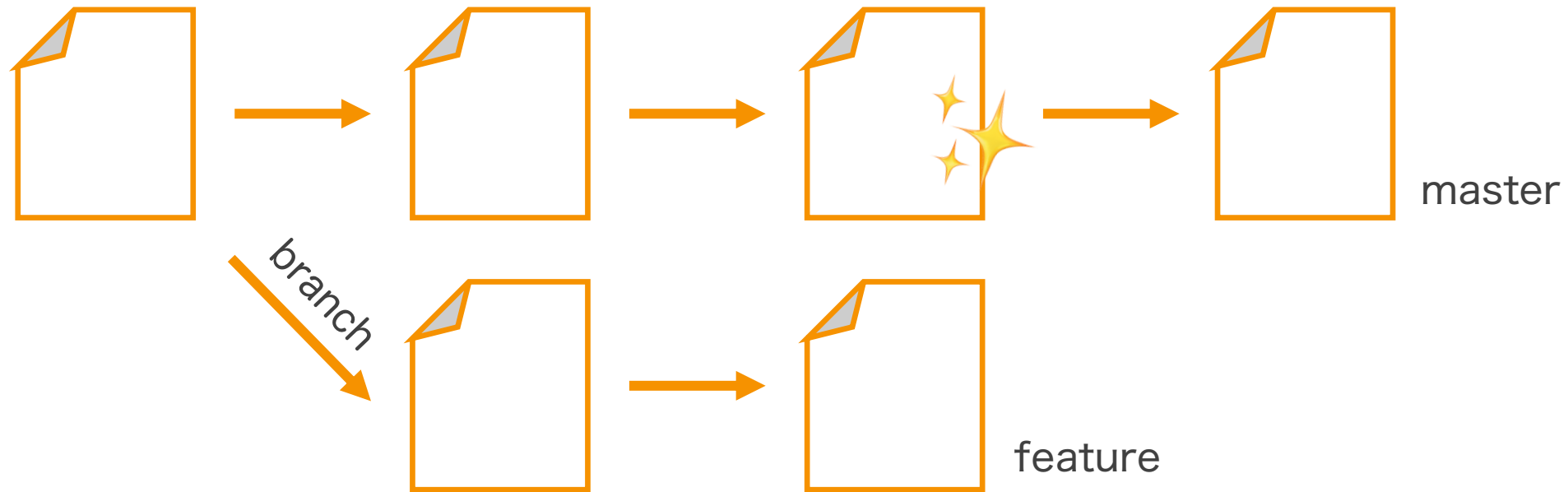


※実際には単なるパッチの適用ではなく、3-way merge というもう少し賢いアルゴリズムを使います

# 歴史を合成する

---

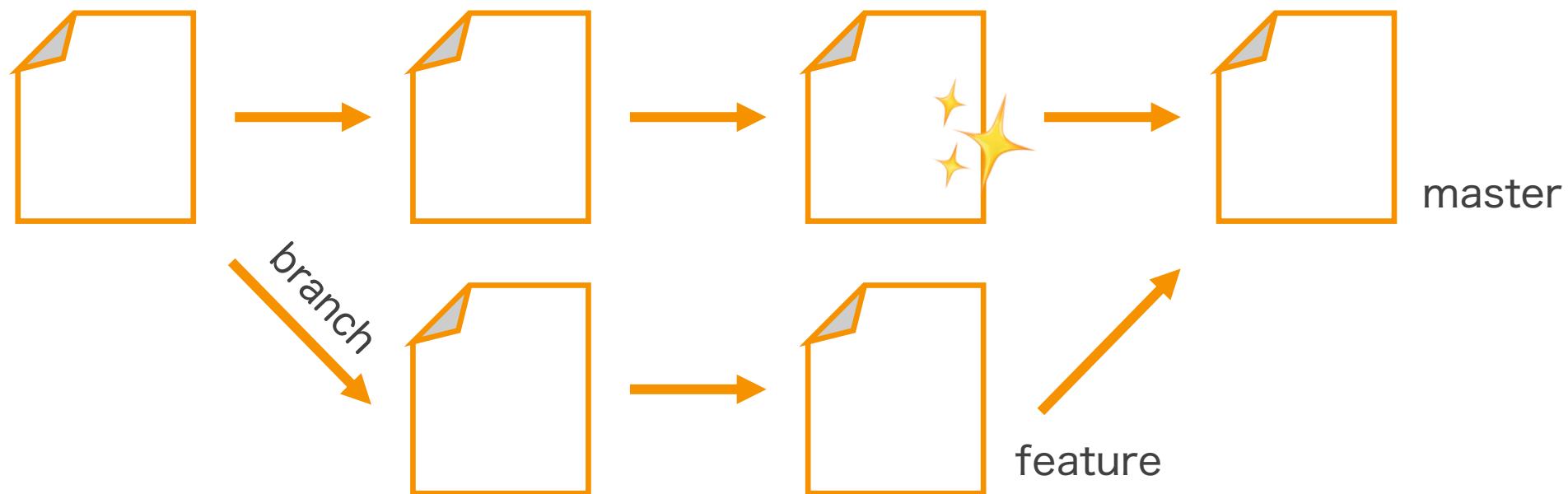
- Merge コマンドの挙動



# 歴史を合成する

---

- Merge コマンドの挙動



# 歴史を合成する

---

- ・ 親の二人いる Commit オブジェクト

```
$ cat .git/objects/64/e7f45... | zlib --decompress
commit 296tree 6826aac350de1ebddc6de03c4d8e424ea7735ed6
parent c5d57bc23a343c226e4970835444c6bd16829878
parent bf390ebccb910fd3a3ddd37f844b90adfce46b1f
author kota.nara <xxxxxxx@xxxxxxx.xx> 1553149094 +0900
committer kota.nara <xxxxxxx@xxxxxxx.xx> 1553149094 +0900

Merge branch 'feature'
```

# 歴史を合成する

---

- ・ ちなみに

# 歴史を合成する

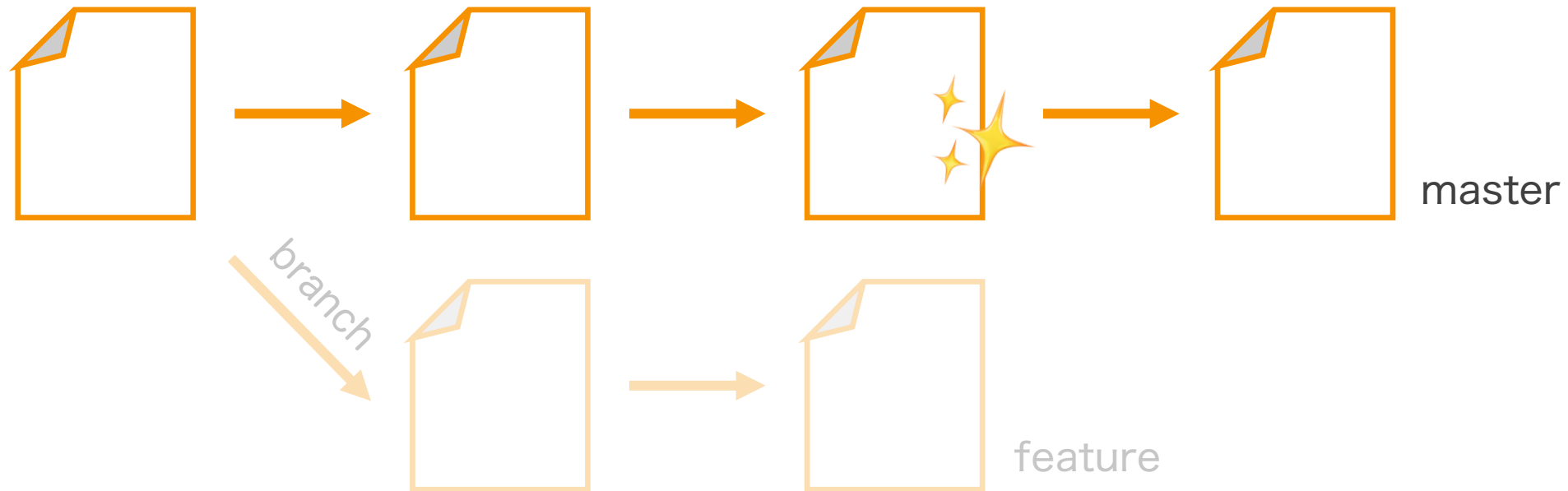
---

- ・ ちなみに
  - ・ `--squash` をつけると二人目の親は記録されない

# 歴史を合成する

---

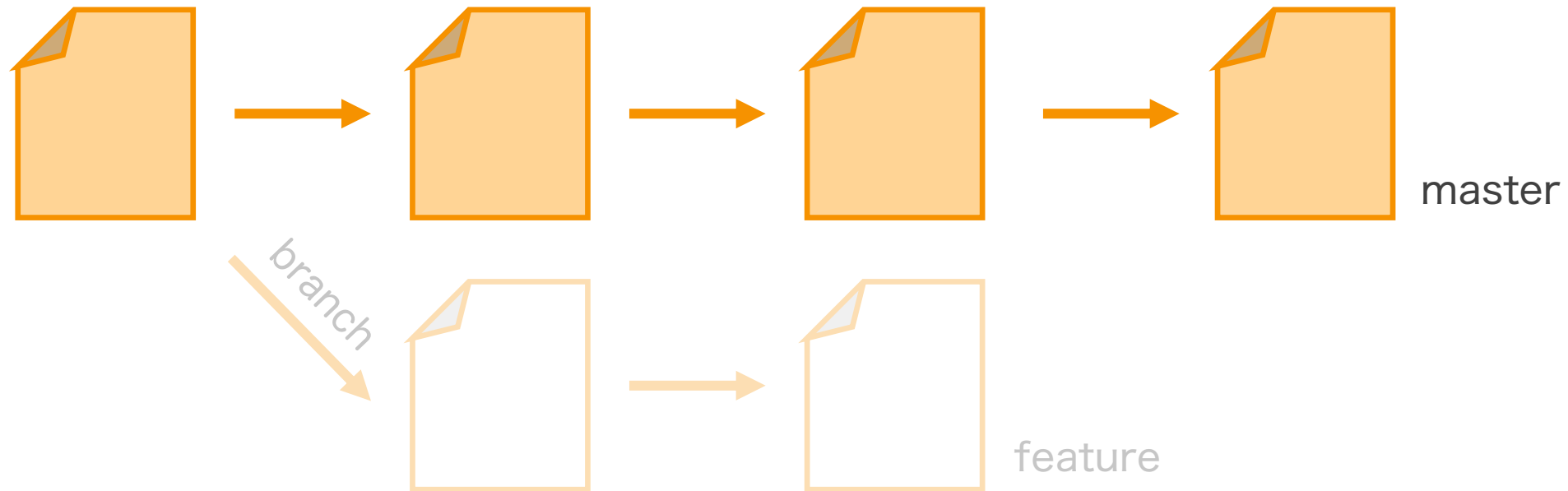
- ・ ちなみに



# 歴史を合成する

---

- ・ ちなみに





# 歴史を合成する

---

- ・ ちなみに
  - ・ `--squash` をつけると二人目の親は記録されない
  - ・ 1 コミットで全作業を完了したかのようなログになる

# 歴史を合成する

---

- ・ ちなみに
  - ・ `--squash` をつけると二人目の親は記録されない
  - ・ 1 コミットで全作業を完了したかのようなログになる
    - ・ ログがスッキリするので、こちらを使うチームもある

# 歴史を合成する

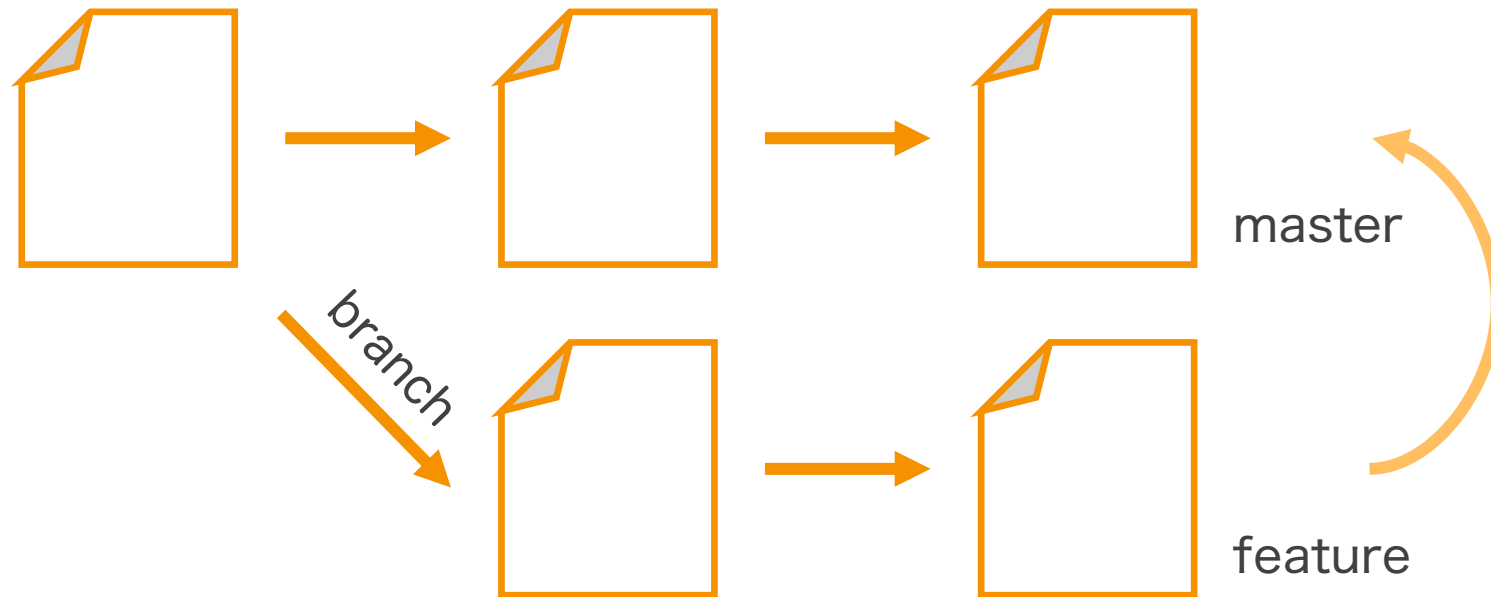
---

- ・ Merge を自動でできない場合

# 歴史を合成する

---

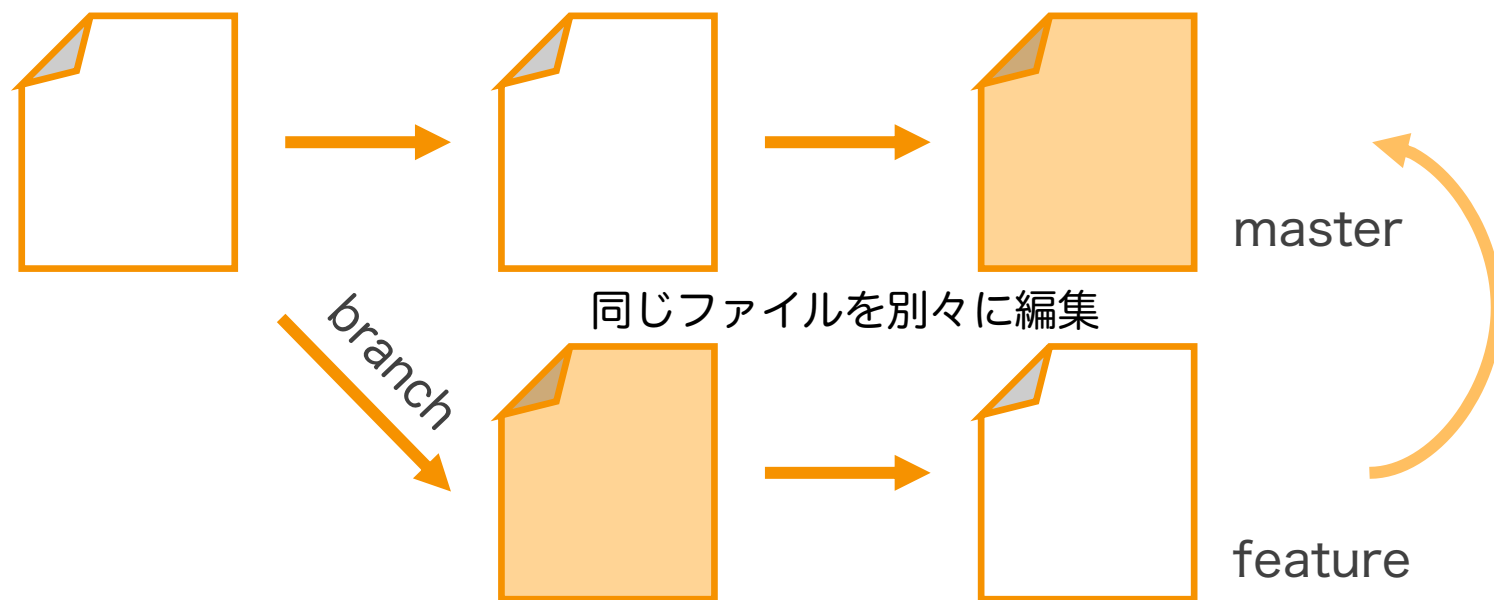
- Merge を自動でできない場合



# 歴史を合成する

---

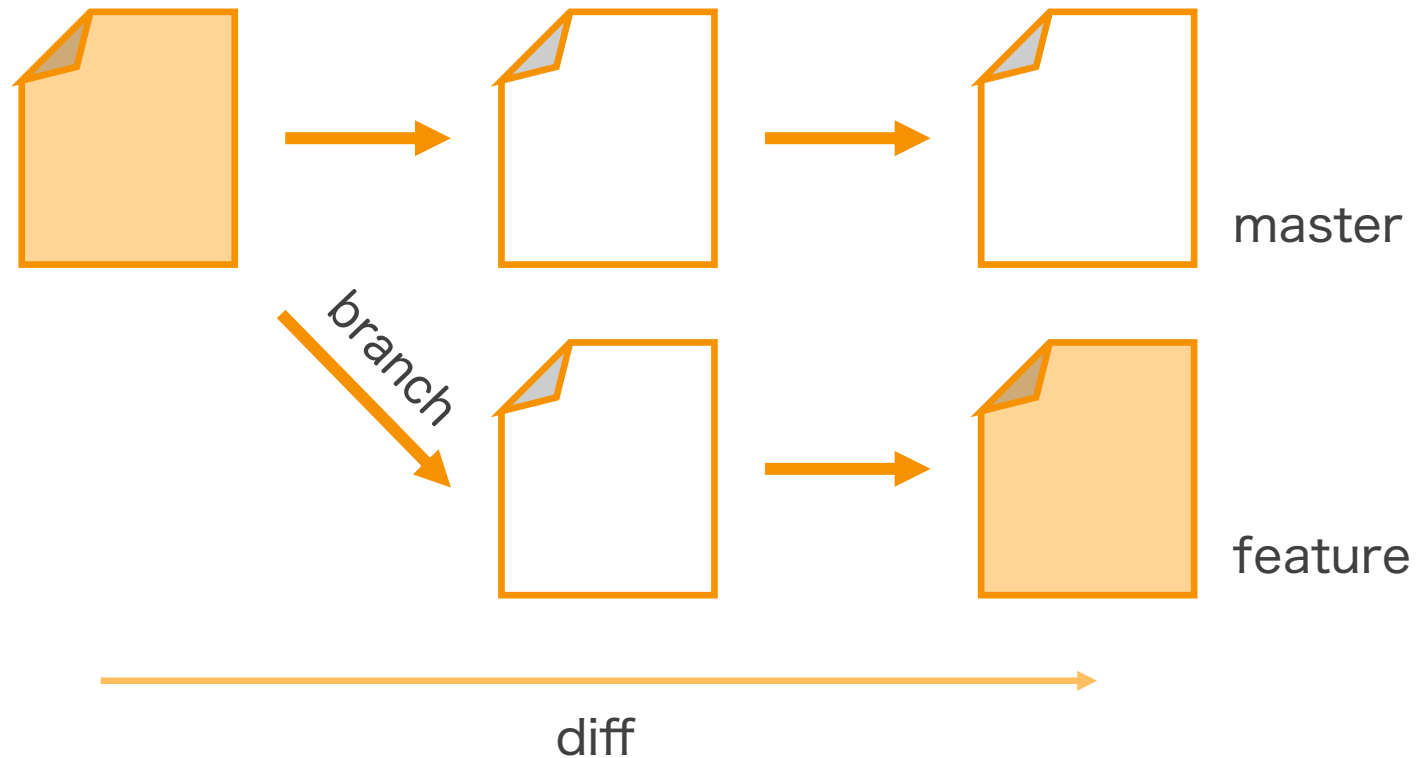
- Merge を自動でできない場合



# 歴史を合成する

---

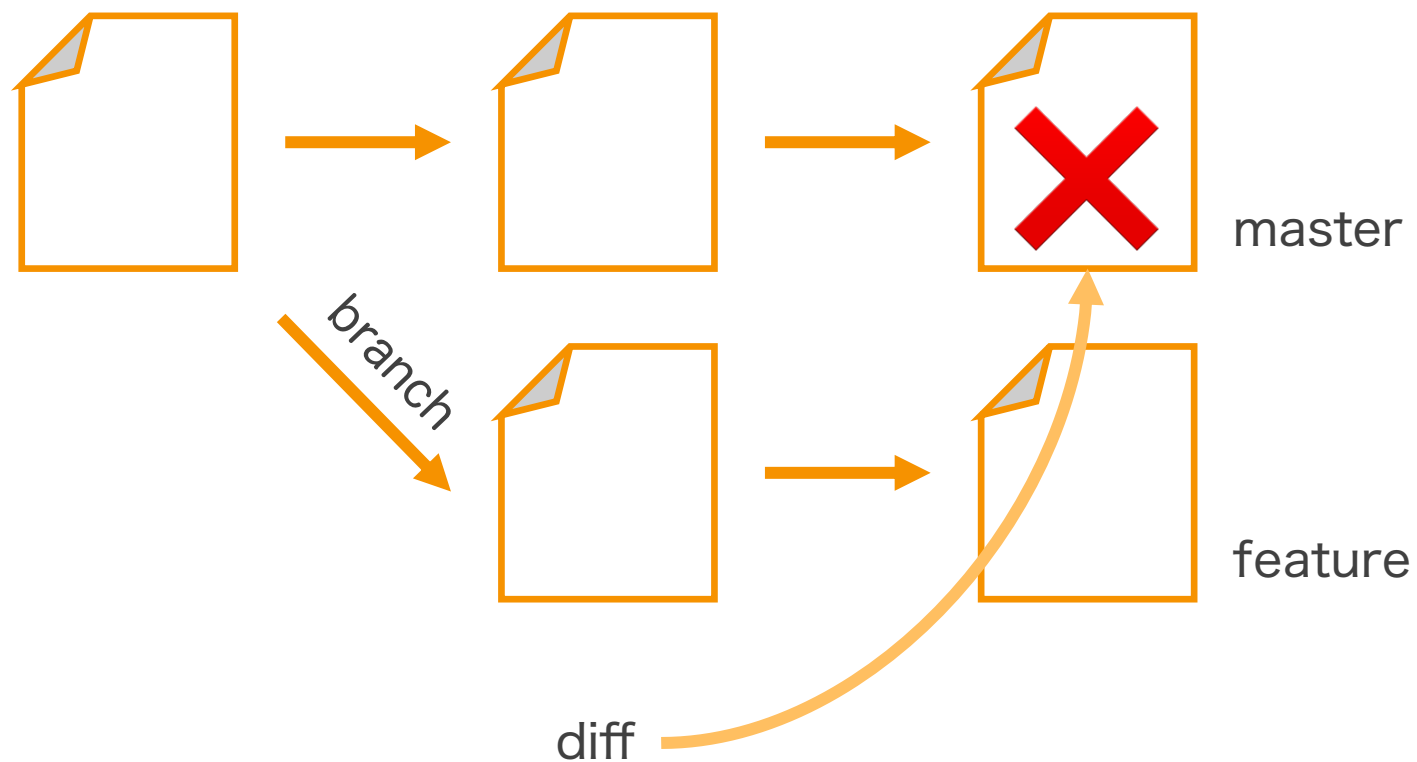
- Merge を自動でできない場合



# 歴史を合成する

---

- Merge を自動でできない場合



# 歴史を合成する

---

- Merge を自動でできない場合

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



# 歴史を合成する

---

- ・ みんな大好きコンフリクト

# 歴史を合成する

---

- ・ みんな大好きコンフリクト
  - ・ 二つのブランチで同じ箇所が別々に編集されている
  - ・ マージ結果は人間が教えてあげないといけない

# 歴史を合成する

---

- ・ みんな大好きコンフリクト

```
$ cat hoge
Hogehoge hoge
<<<<<< HEAD
piyo
=====
fuga
>>>>>> feature
hogehoge
```

- ・ コンフリクトしたファイルには「マーカー」が書き込まれる

# 歴史を合成する

---

- ・ みんな大好きコンフリクト

```
$ cat hoge  
Hogehoge hoge
```

```
<<<<<< HEAD
```

```
piyo
```

```
=====
```

```
fuga
```

```
>>>>>> feature
```

```
hogehoge
```

HEAD (master) は piyo って言ってるけど

- ・ コンフリクトしたファイルには「マーカー」が書き込まれる

# 歴史を合成する

---

- ・ みんな大好きコンフリクト

```
$ cat hoge
Hogehoge hoge
<<<<<< HEAD
piyo
=====
fuga
>>>>>> feature
hogehoge
```

feature は fuga って言ってるぞ

- ・ コンフリクトしたファイルには「マーカー」が書き込まれる

# 歴史を合成する

---

- ・ みんな大好きコンフリクト

```
$ vim hoge
```

- ・ コンフリクトしたファイルは手動でマージする

# 歴史を合成する

---

- ・ みんな大好きコンフリクト

```
$ vim hoge
Hogehoge hoge
<<<<<< HEAD
piyo
=====
fuga
>>>>>> feature
hogehoge
```

- ・ コンフリクトしたファイルは手動でマージする

# 歴史を合成する

---

- ・ みんな大好きコンフリクト

```
$ vim hoge  
Hogehoge hoge  
fugapiyo  
Hogehoge
```

fugapiyo でよろしくな

- ・ コンフリクトしたファイルは手動でマージする



# 歴史を合成する

---

- ・ みんな大好きコンフリクト

```
$ vim hoge
Hogehoge hoge
fugapiyo
Hogehoge

$ git add hoge
$ git commit
```

- ・ コンフリクトしたファイルは手動でマージする

# Fast-forward Merge

# Fast-forward Merge

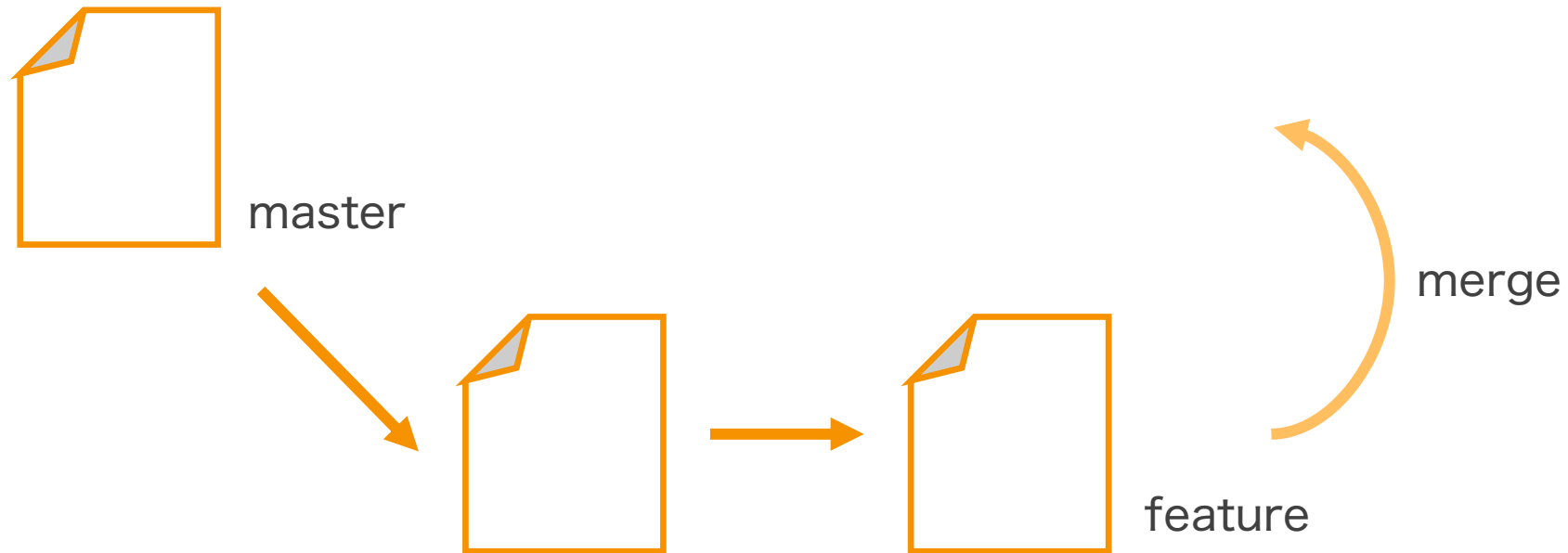
---

- ・ 特別な状況では Git は merge 作業をサボる

# Fast-forward Merge

---

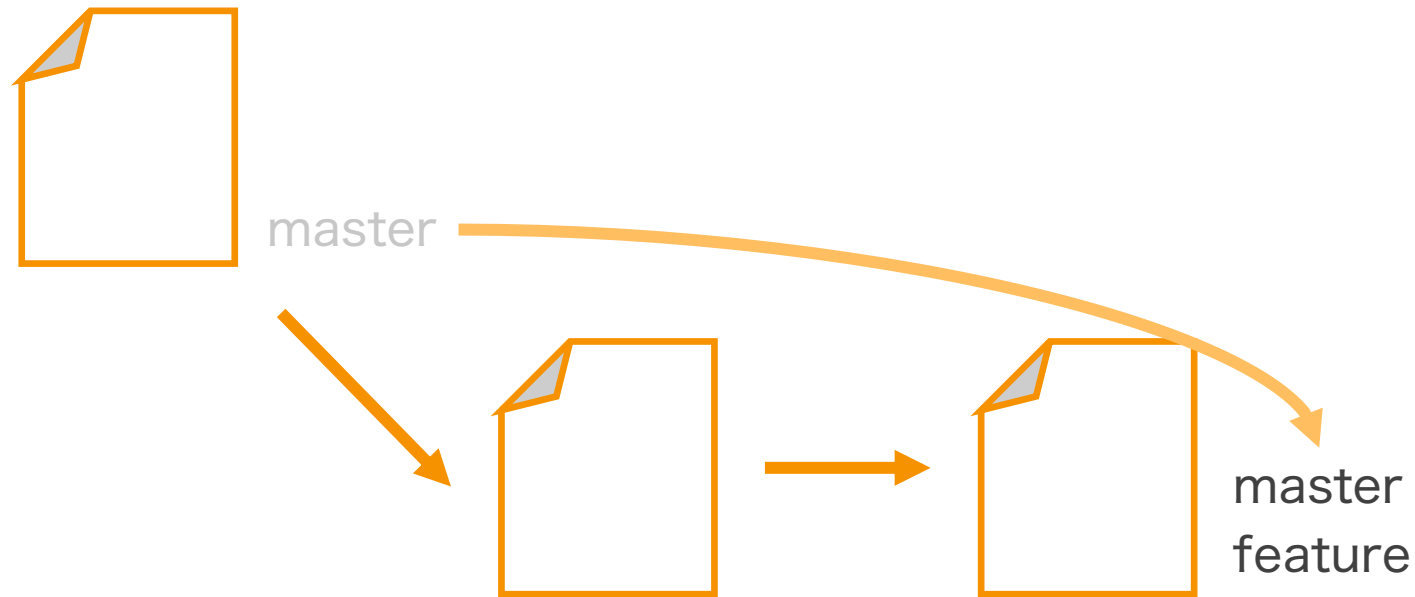
- ・ 特別な状況では Git は merge 作業をサボる



# Fast-forward Merge

---

- ・ 特別な状況では Git は merge 作業をサボる



# Fast-forward Merge

---

- ・ 特別な状況では Git は merge 作業をサボる
  - ・ 枝分かれがない → Fast-forward (早送り) マージ

# Fast-forward Merge

---

- merge の記録を明示的に残したい場合は --no-ff

```
$ git merge --no-ff feature
```

Commit --amend



# Commit --amend

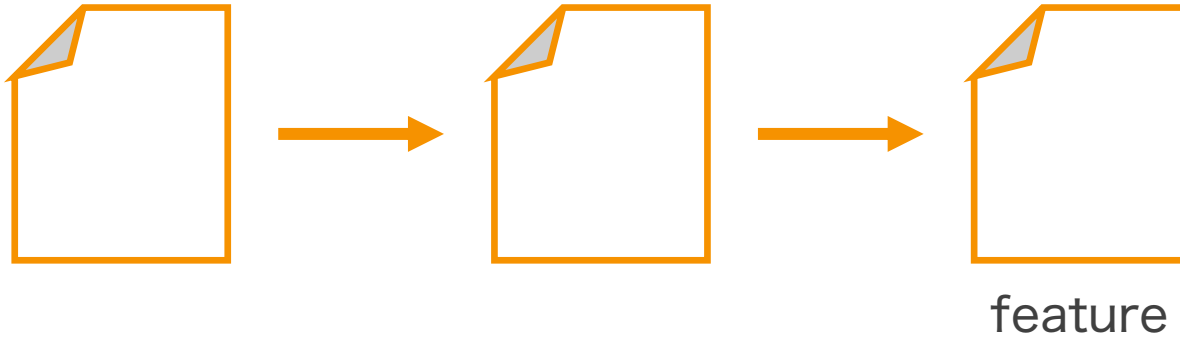
---

- ・ 直前のコミットを修正する

# Commit --amend

---

- ・ 直前のコミットを修正する



# Commit --amend

---

- ・直前のコミットを修正する

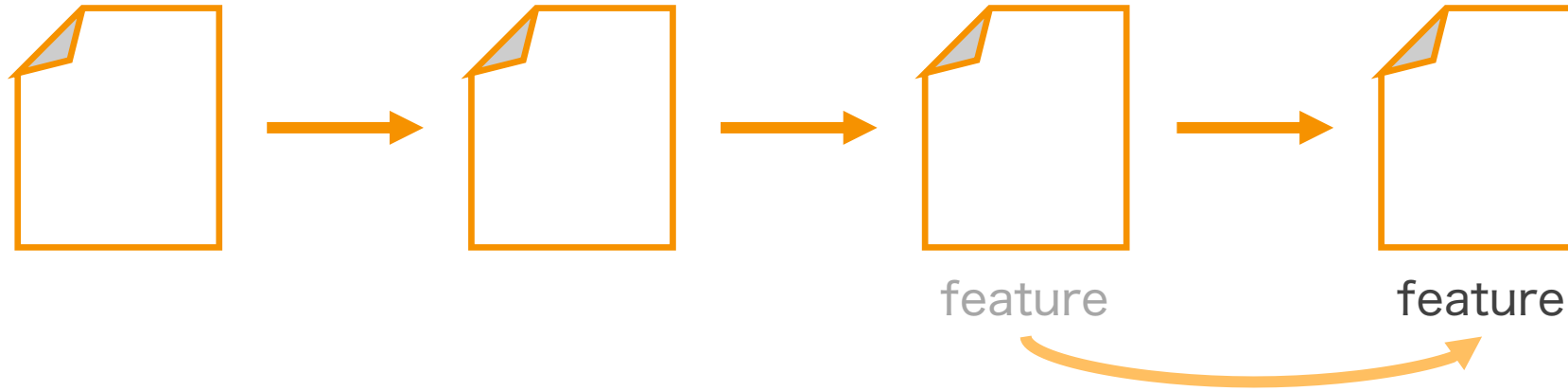
```
$ git commit -m "hogehoge"
```

- ・普通に commit すると…

# Commit --amend

---

- ・ 直前のコミットを修正する



# Commit --amend

---

- ・ 直前のコミットを修正する

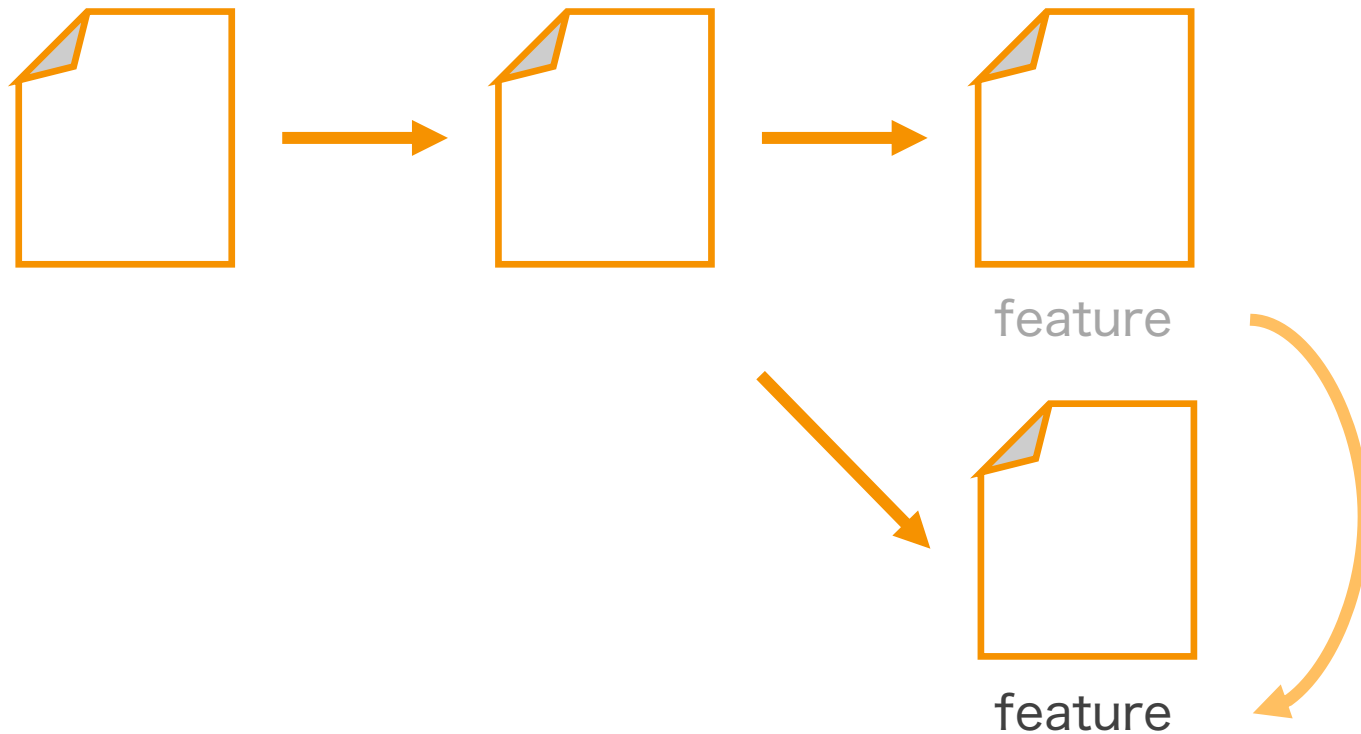
```
$ git commit --amend -m "hogehoge"
```

- ・ --amend をつけて commit すると…

# Commit --amend

---

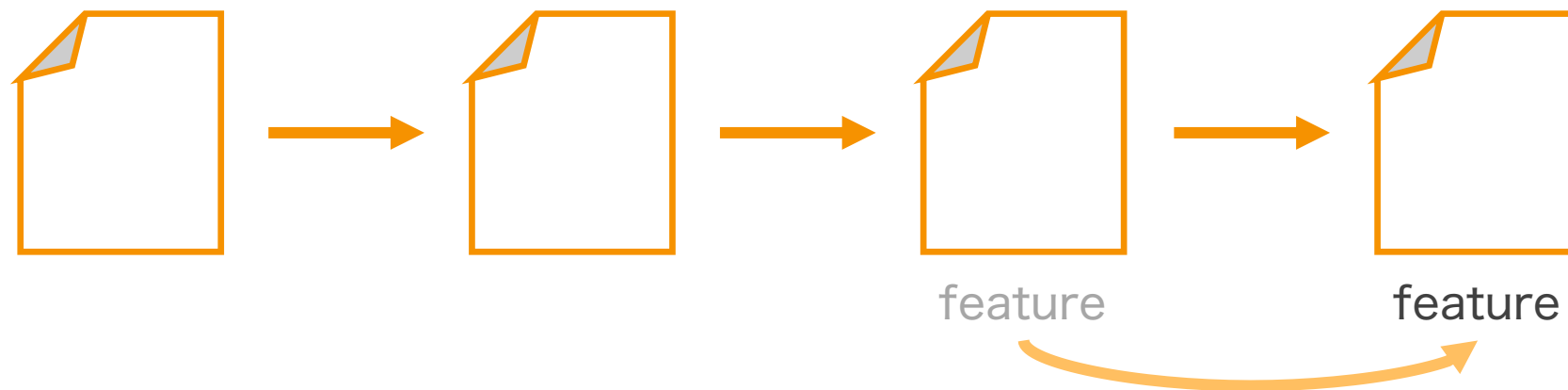
- ・ 直前のコミットを修正する



# Commit --amend

---

- ・ 直前のコミットを修正する

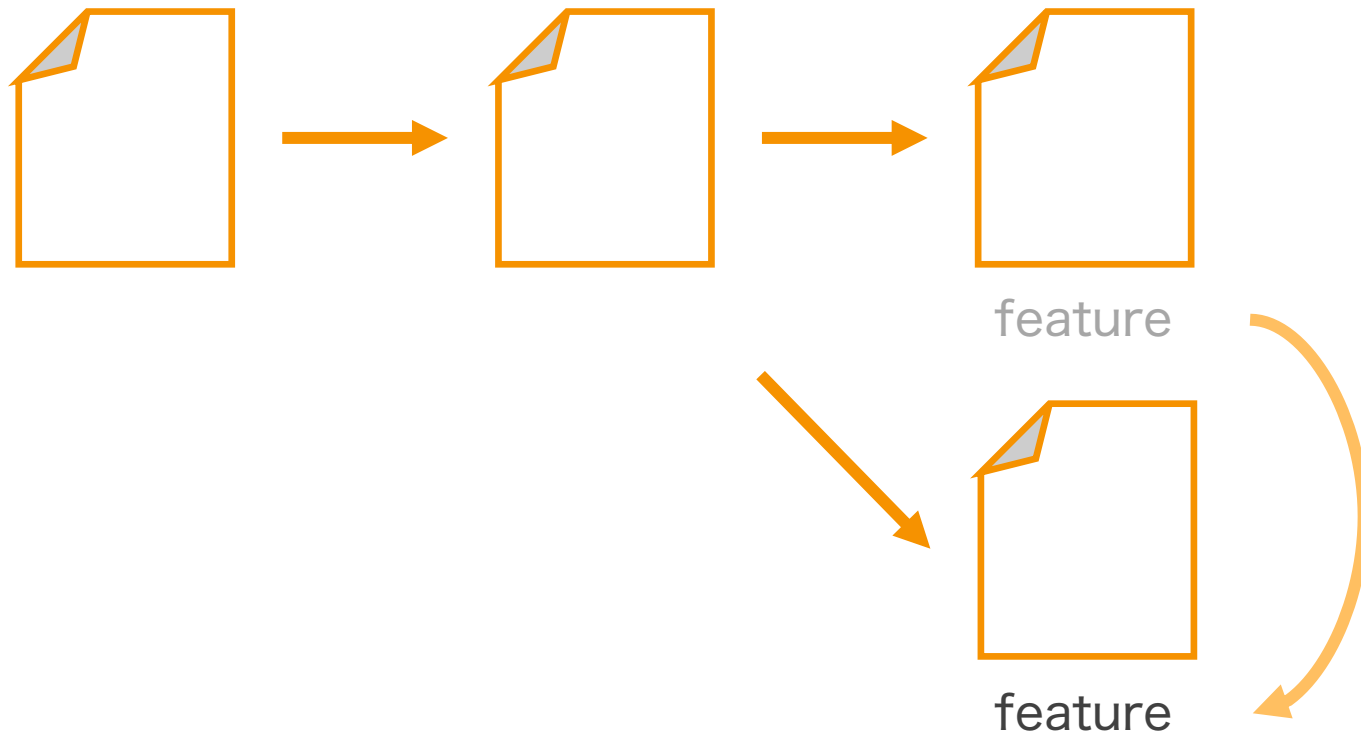


- ・ 普通にコミットした場合とは親が違う

# Commit --amend

---

- ・ 直前のコミットを修正する





# Commit --amend

---

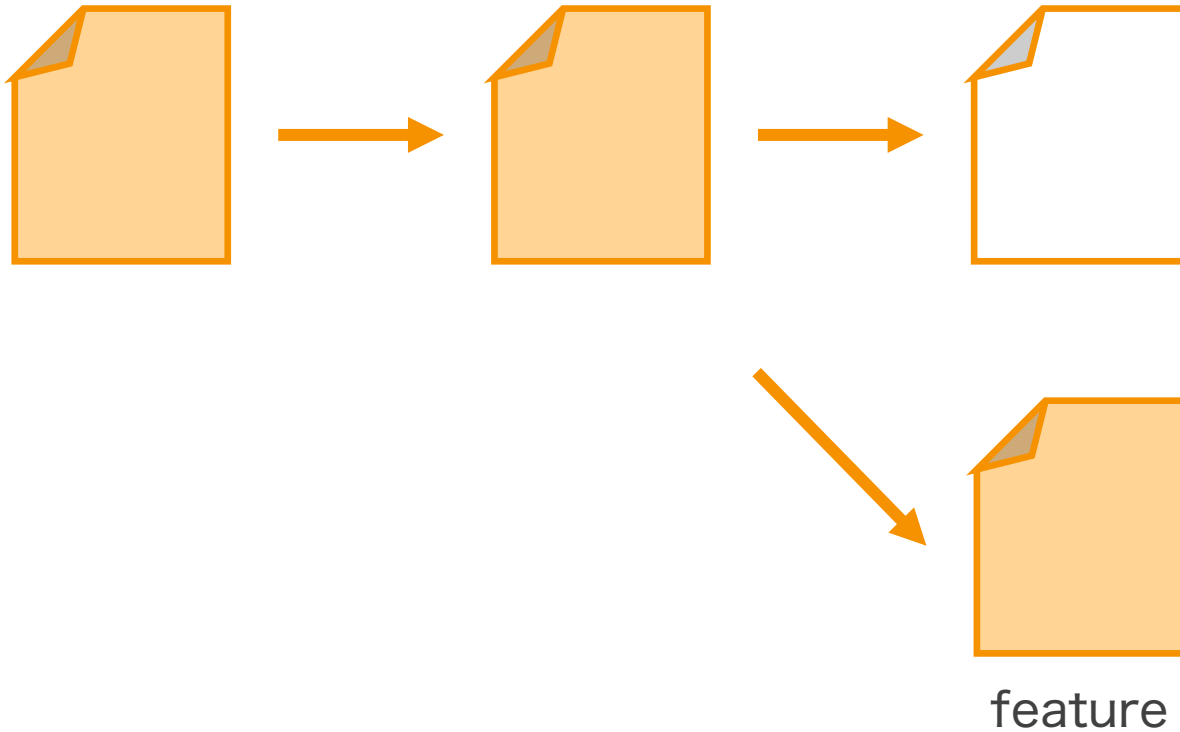
- ・ 直前のコミットを修正する

```
$ git log feature
```

# Commit --amend

---

- ・ 直前のコミットを修正する



# Commit --amend

---

- ・ 直前のコミットを修正する
  - ・ 元のコミットがなかったことになる (コミットログから外れる)

# Commit --amend

---

- ・ 直前のコミットを修正する
  - ・ 元のコミットがなかったことになる (コミットログから外れる)

※元のコミットを破壊的に書き換えるわけではない

- ・ 内容が変われば SHA1 が変わるので、改ざんはできない

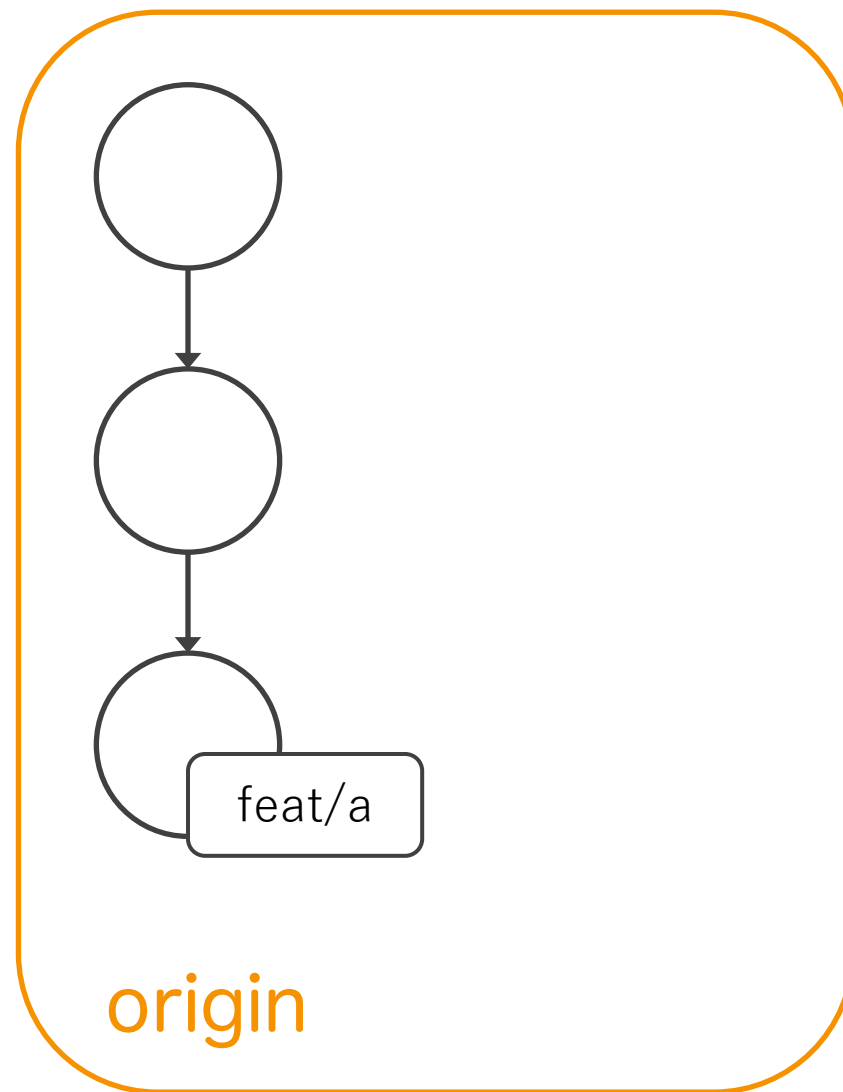
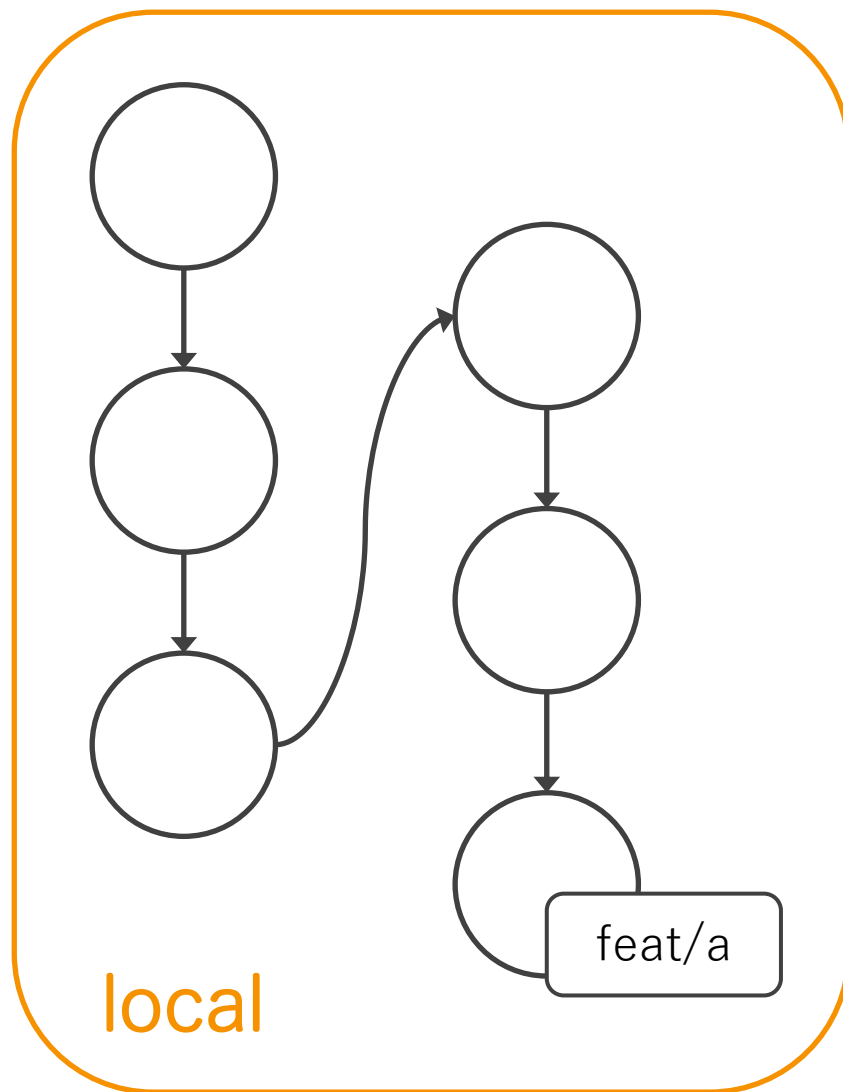
歴史改変と push

## 歴史改変と push

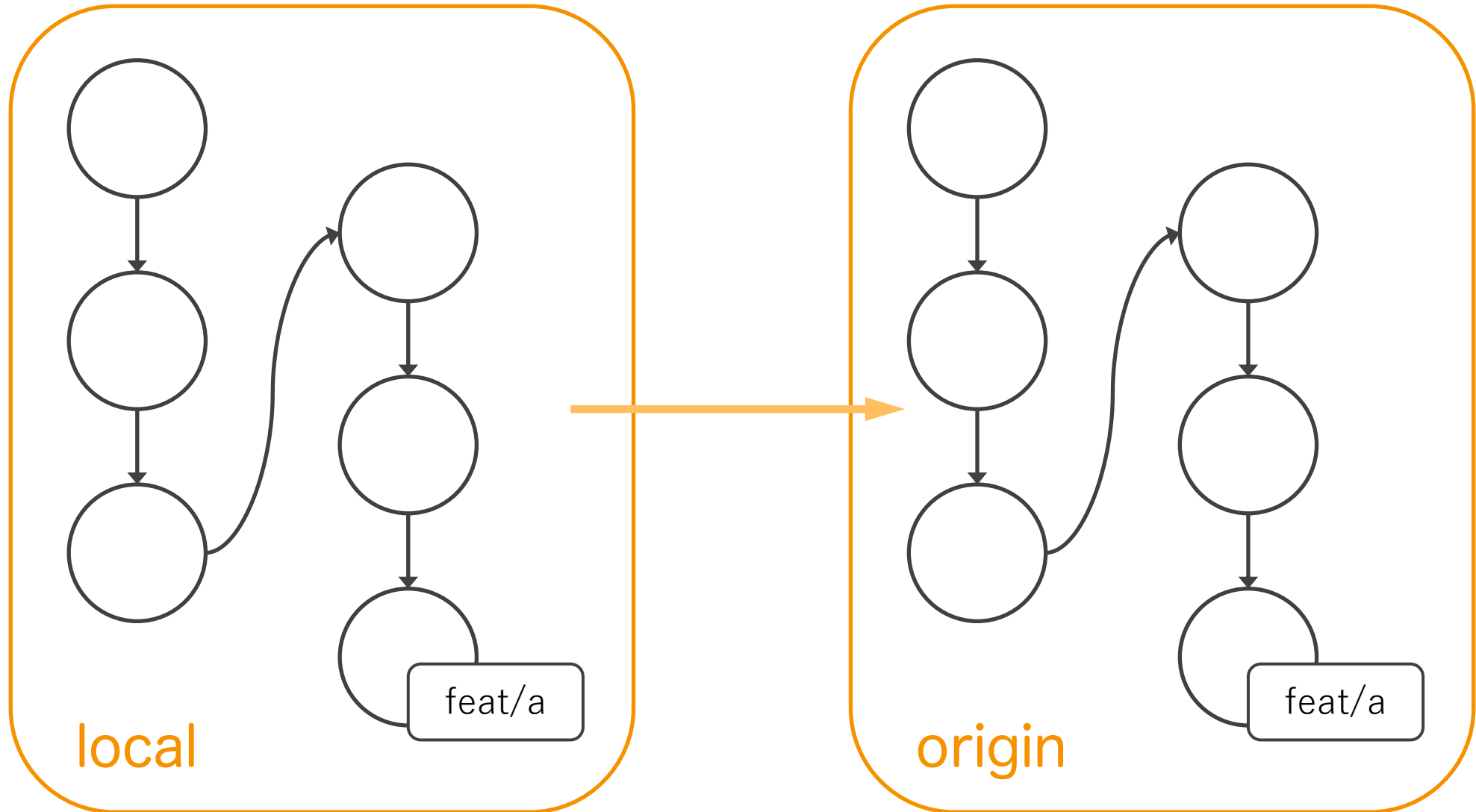
---

- ・ `commit --amend` してはいけないタイミングがある

# 歴史改変と push

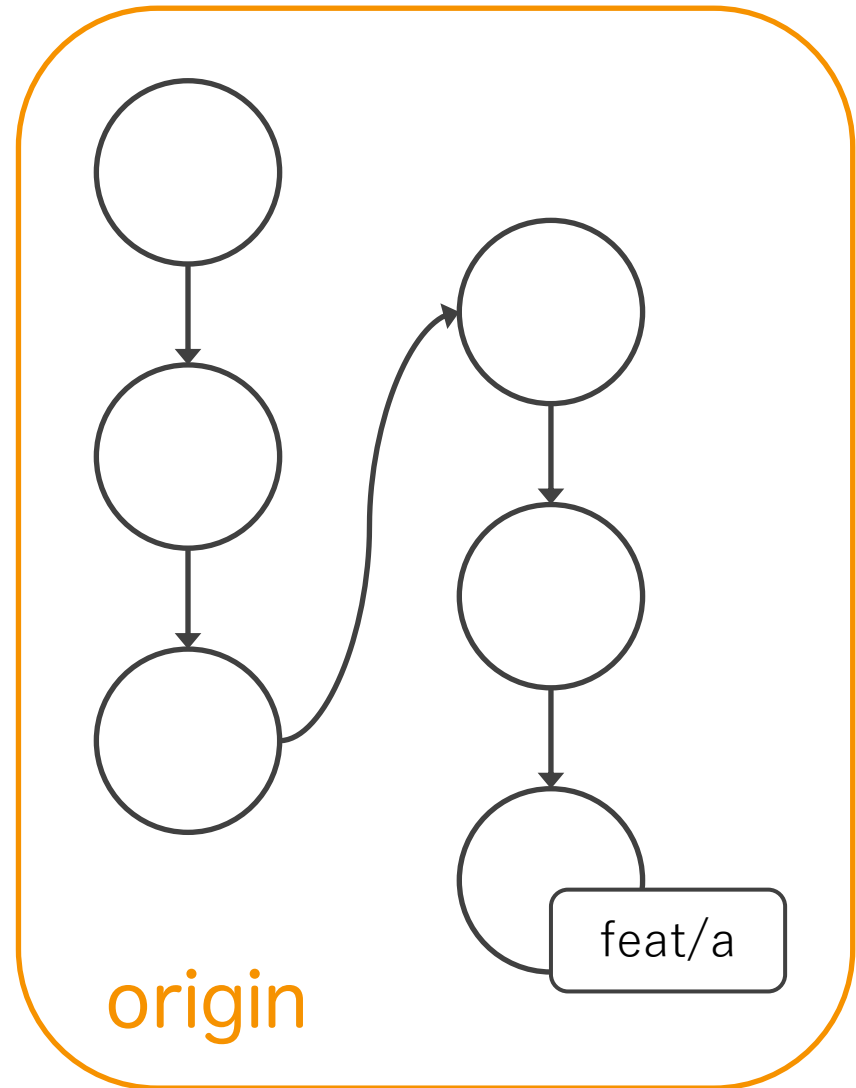
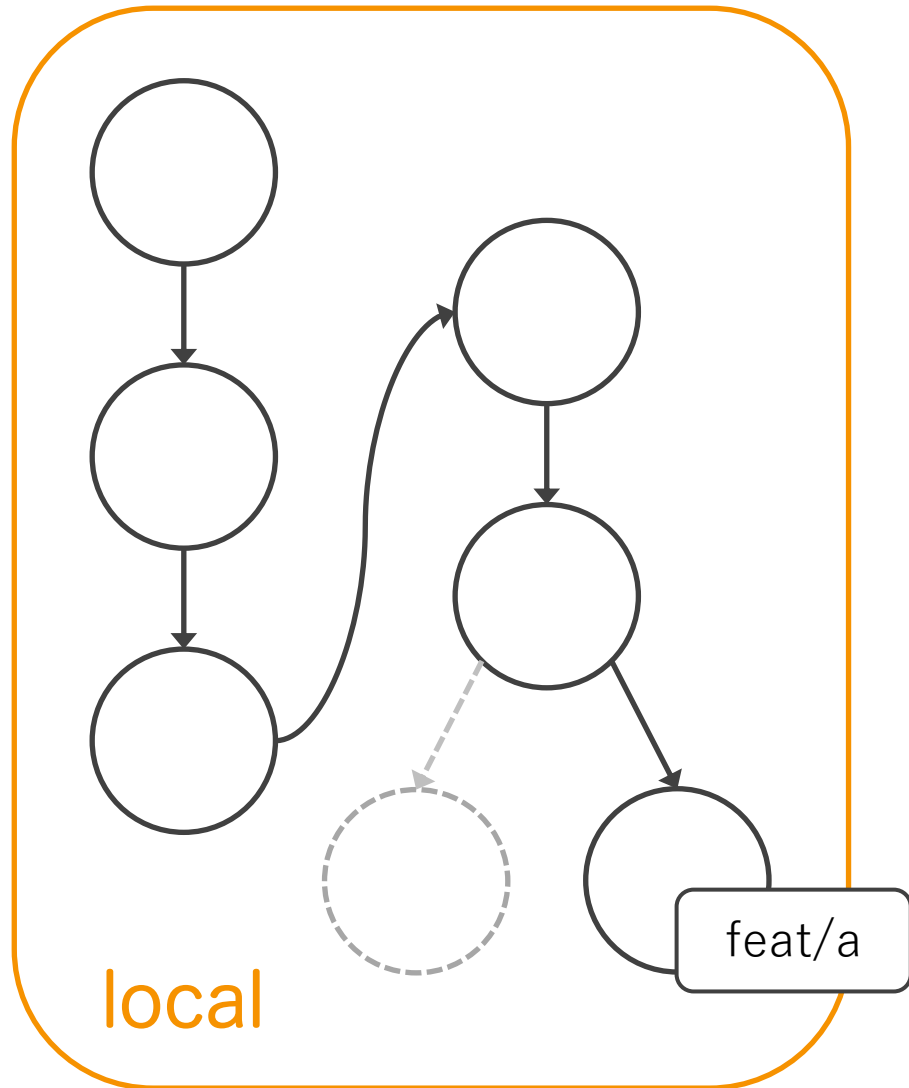


# 歴史改変と push

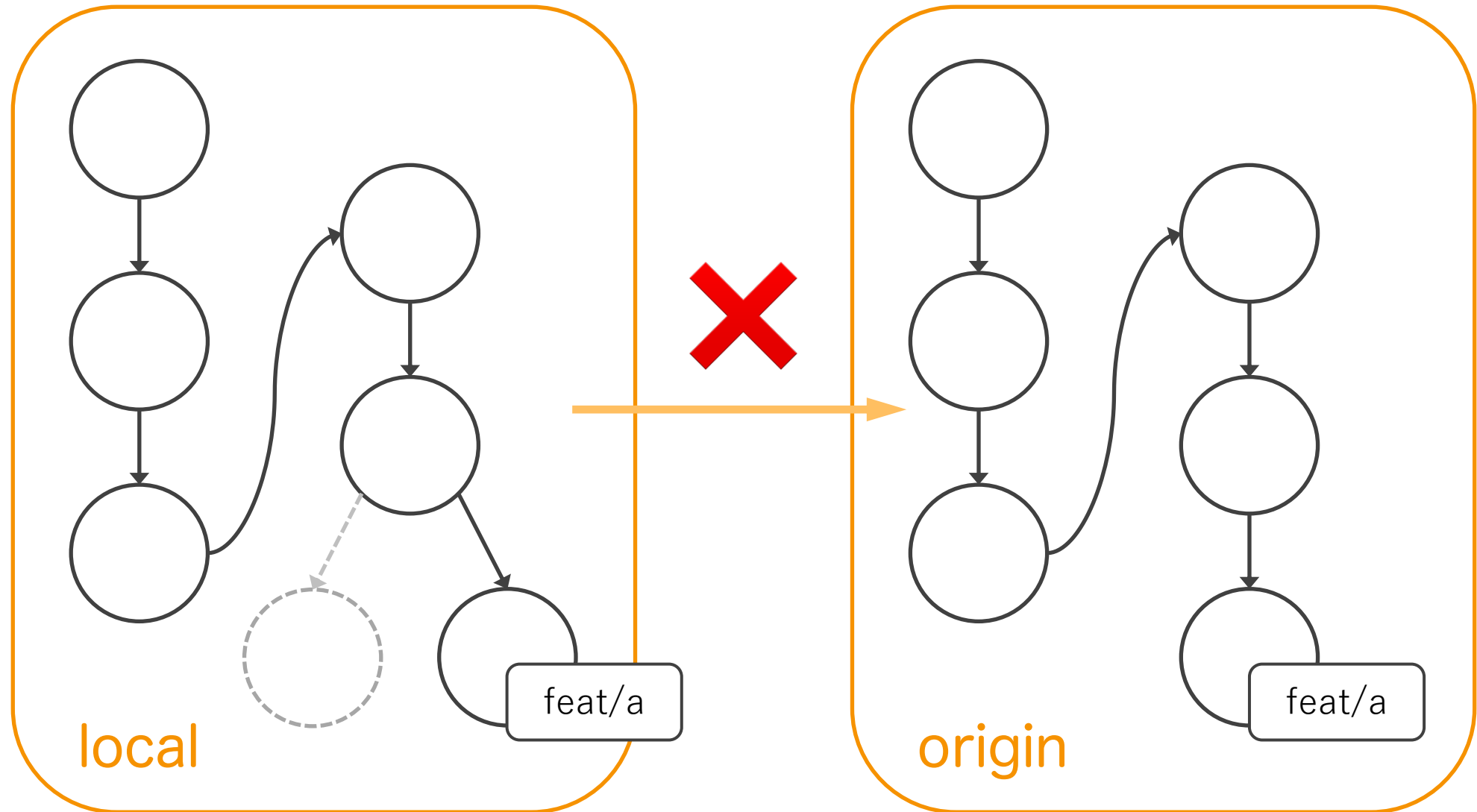




# 歴史改変と push



# 歴史改変と push



# 歴史改変と push

---

- commit --amend してはいけないタイミングがある

```
$ git push origin feature/a  
! [rejected]      feature/a -> feature/a (non-fast-forward)  
Error: failed to push some refs
```

# 歴史改変と push

---

- commit --amend してはいけないタイミングがある

```
$ git push origin feature/a  
! [rejected]      feature/a -> feature/a (non-fast-forward)  
Error: failed to push some refs
```

- リモート側は fast-forward でブランチの更新を取り込む

# 歴史改変と push

---

- commit --amend してはいけないタイミングがある

```
$ git push origin feature/a  
! [rejected]      feature/a -> feature/a (non-fast-forward)  
Error: failed to push some refs
```

- リモート側は fast-forward でブランチの更新を取り込む
- ローカルと食い違い (枝分かれ) があると push できない

# 歴史改変と push

---

-   必殺 force push

# 歴史改変と push

---

-   必殺 force push

```
$ git push -f origin feature/a
```

- リモートの ref を強制上書き

# 歴史改変と push

---

-   必殺 force push

```
$ git push -f origin feature/a
```

- リモートの ref を強制上書き
- 他のメンバーがすごく困るので原則やってはいけない



便利コマンドたち

## `git commit` 再訪

---

- ここからは、それぞれの Git コマンドの挙動を見ていく
  - commit (, add)
  - checkout, reset
  - merge
  - rebase (, cherry-pick, revert)

# 便利コマンドたち

---

- ・ 典型的な作業は Git で自動化できる

# 便利コマンドたち

---

- ・ 典型的な作業は Git で自動化できる
  - ・ ある差分を別のブランチに移植
  - ・ ある差分を元に戻す
  - ・ 一連の差分をまるごと引っ越す

# 便利コマンドたち

---

- ・ 典型的な作業は Git で自動化できる
  - ・ ある差分を別のブランチに移植
  - ・ ある差分を元に戻す
  - ・ 一連の差分をまるごと引っ越す

# 便利コマンドたち

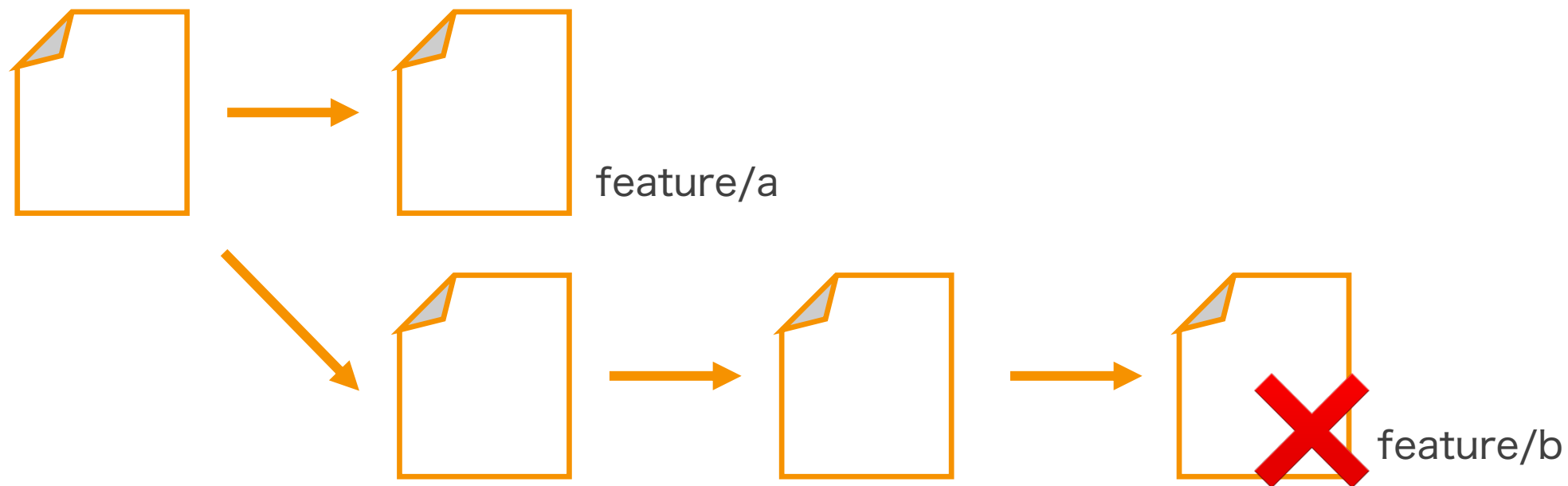
---

- ・ ある差分を別のブランチに移植する

# 便利コマンドたち

---

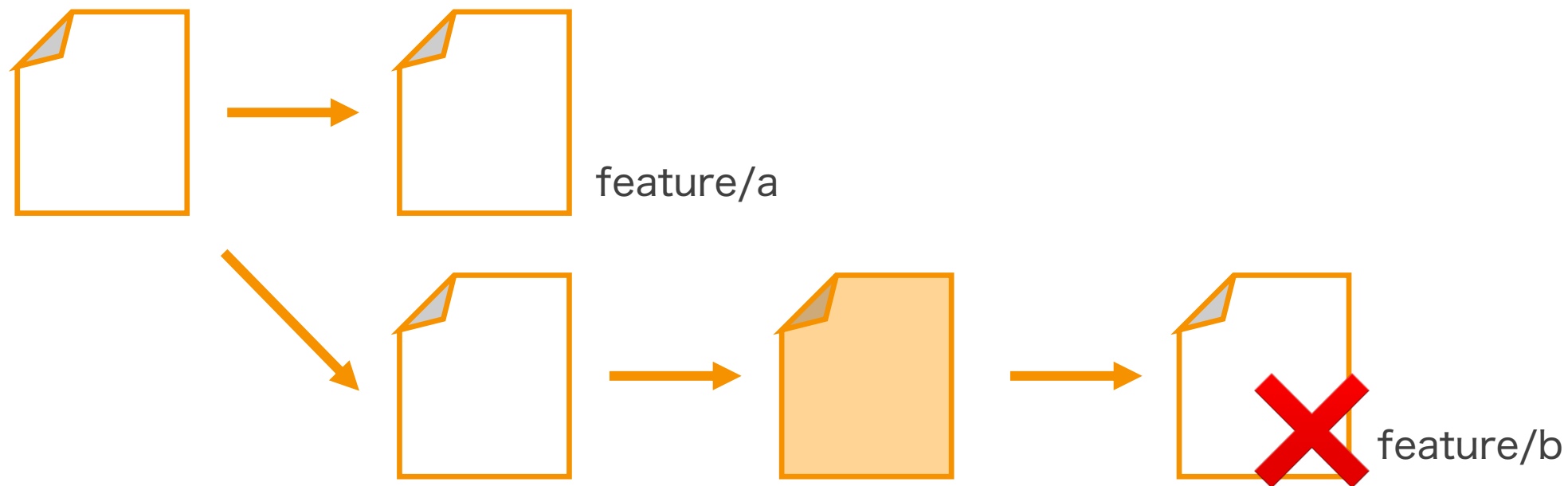
- ・ ある差分を別のブランチに移植する



# 便利コマンドたち

---

- ・ ある差分を別のブランチに移植する





# 便利コマンドたち

---

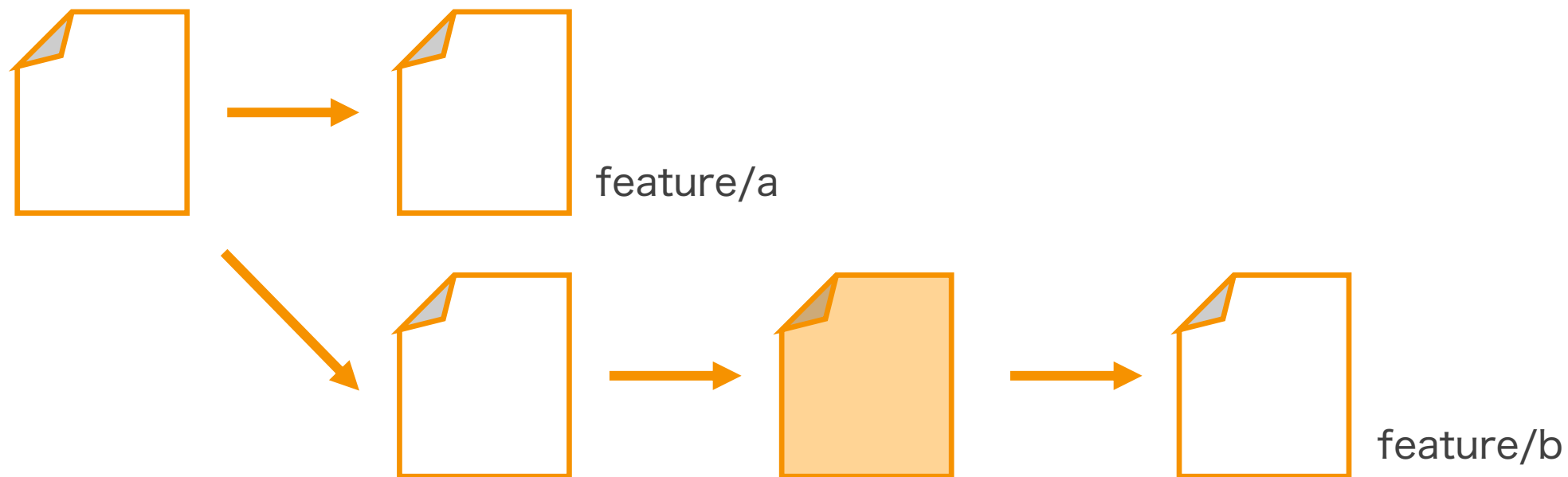
- ・ ある差分を別のブランチに移植する

```
$ git checkout feature/a  
$ git cherry-pick <commit-id>
```

# 便利コマンドたち

---

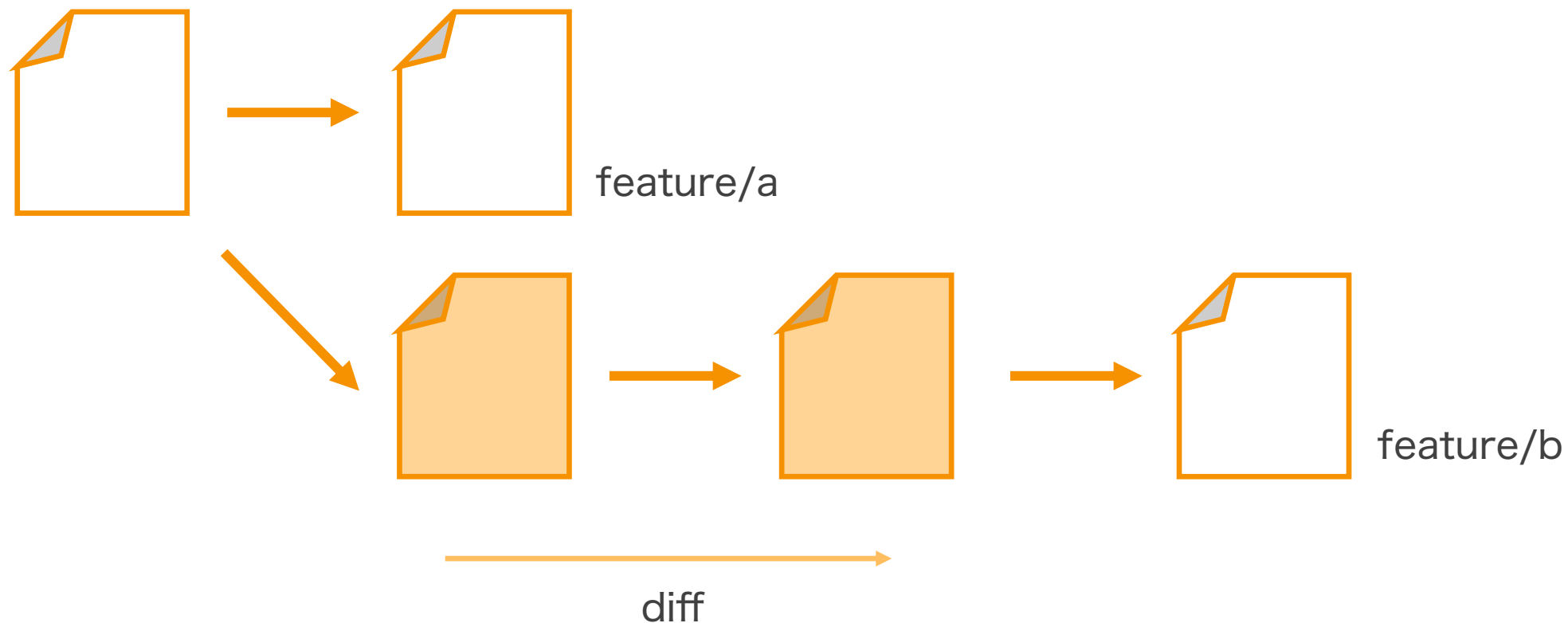
- ・ ある差分を別のブランチに移植する



# 便利コマンドたち

---

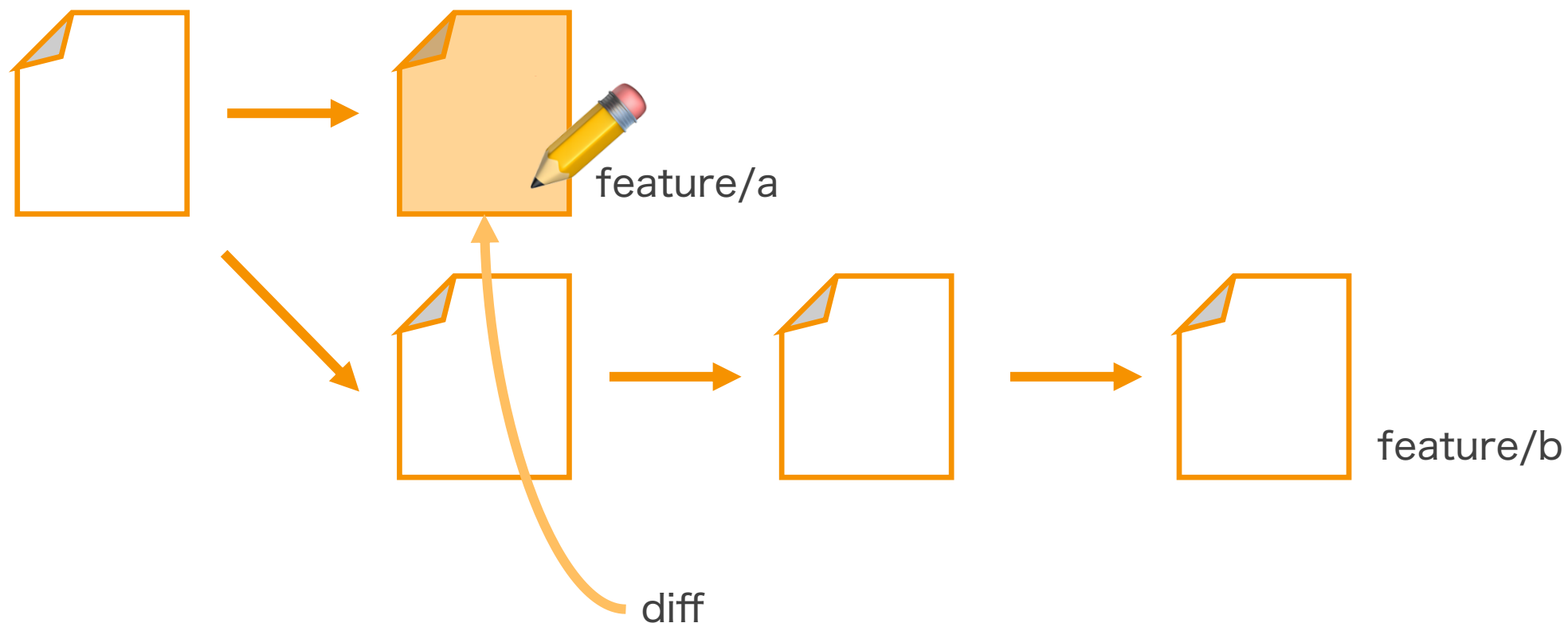
- ・ ある差分を別のブランチに移植する



# 便利コマンドたち

---

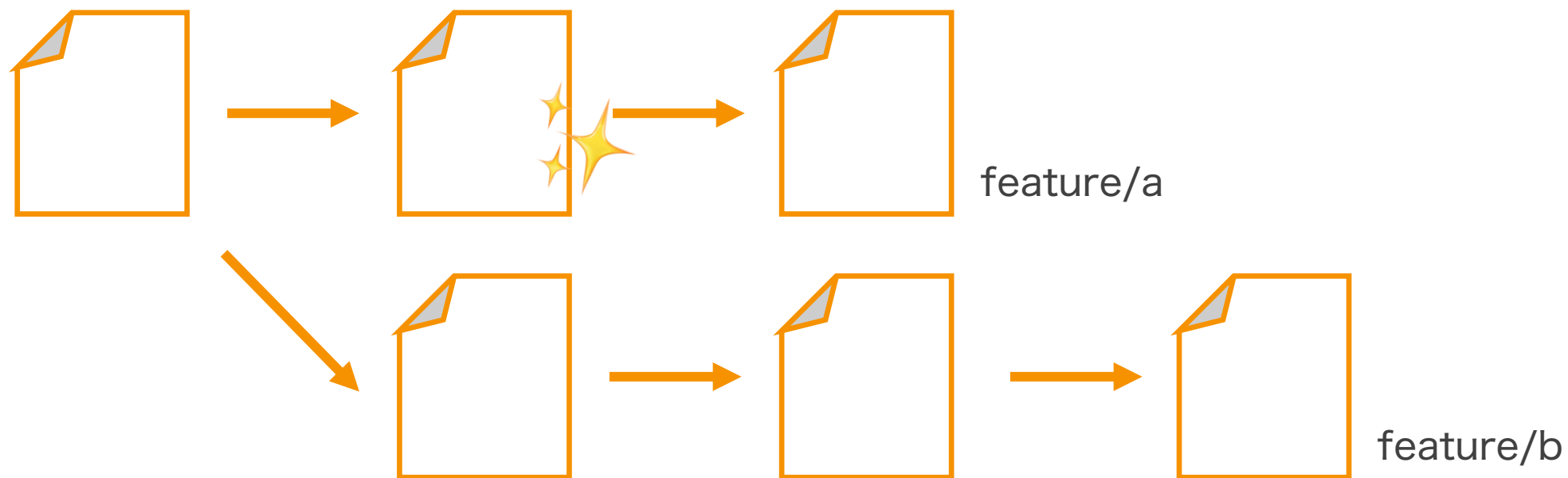
- ・ ある差分を別のブランチに移植する



# 便利コマンドたち

---

- ・ ある差分を別のブランチに移植する



# 便利コマンドたち

---

- ・ 典型的な作業は Git で自動化できる
  - ・ ある差分を別のブランチに移植 -> cherry-pick
  - ・ ある差分を元に戻す
  - ・ 一連の差分をまるごと引っ越す

# 便利コマンドたち

---

- ・ 典型的な作業は Git で自動化できる
  - ・ ある差分を別のブランチに移植 -> cherry-pick
  - ・ ある差分を元に戻す
  - ・ 一連の差分をまるごと引っ越す

# 便利コマンドたち

---

- ・ ある差分を元に戻す



# 便利コマンドたち

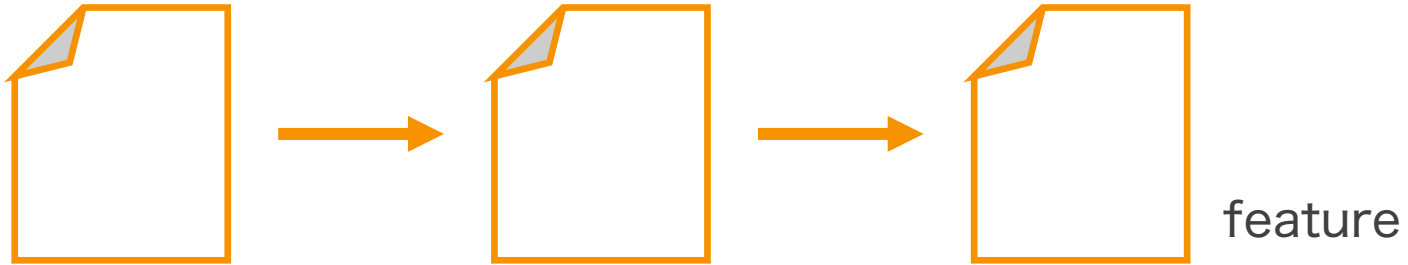
---

- ・ ある差分を元に戻す
  - ・ 歴史を改変しないように戻さないと、 push できなくなる

# 便利コマンドたち

---

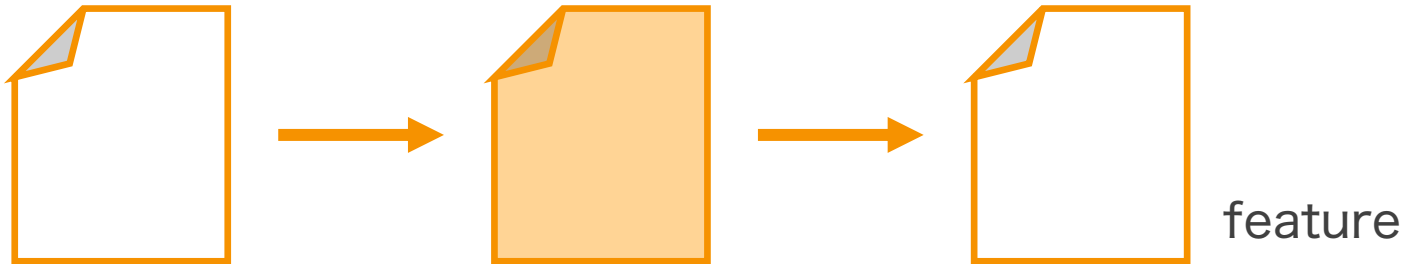
- ・ ある差分を元に戻す



# 便利コマンドたち

---

- ・ ある差分を元に戻す



# 便利コマンドたち

---

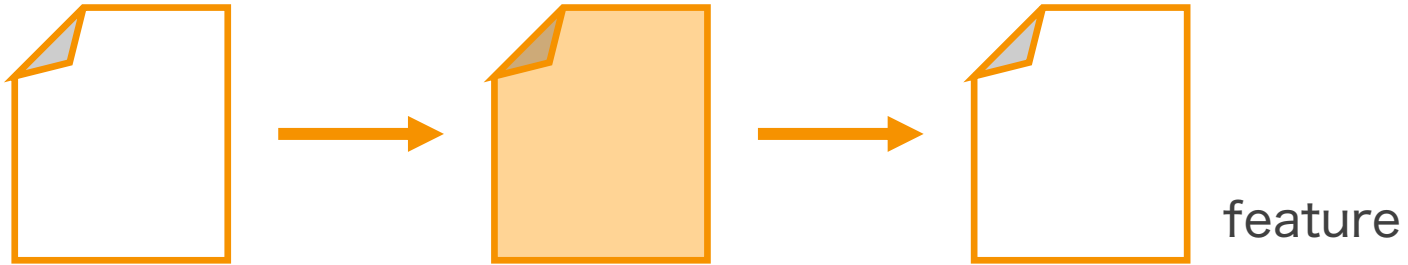
- ・ ある差分を元に戻す

```
$ git revert <commit-id>
```

# 便利コマンドたち

---

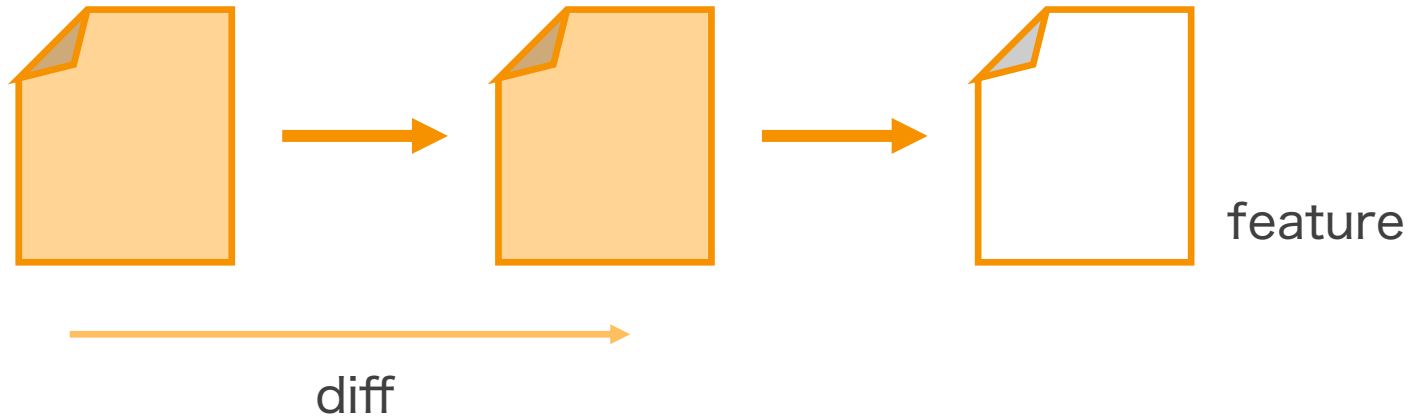
- ・ ある差分を元に戻す



# 便利コマンドたち

---

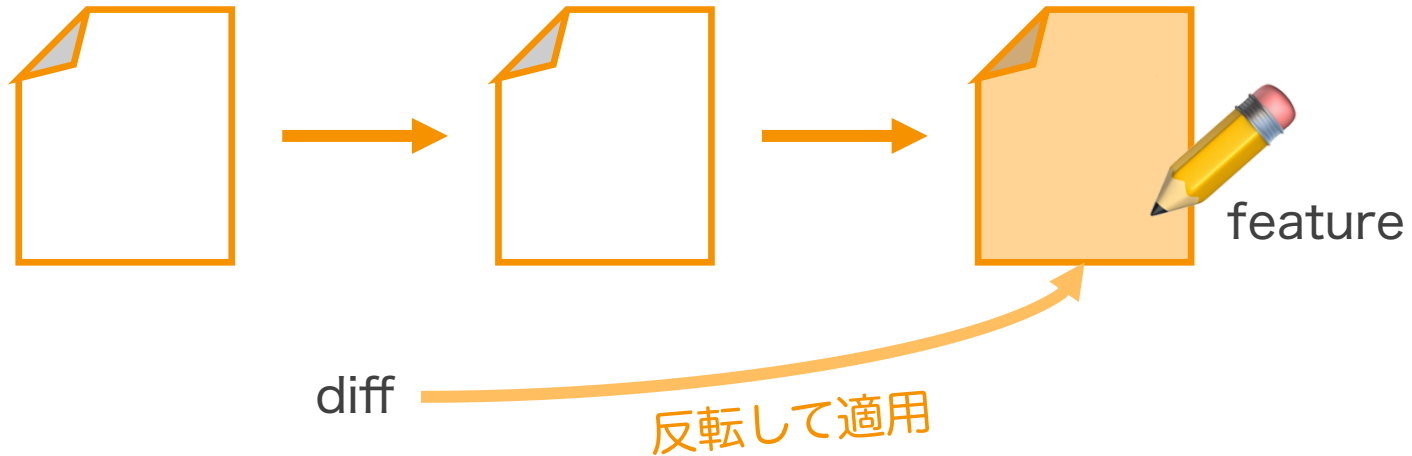
- ・ ある差分を元に戻す



# 便利コマンドたち

---

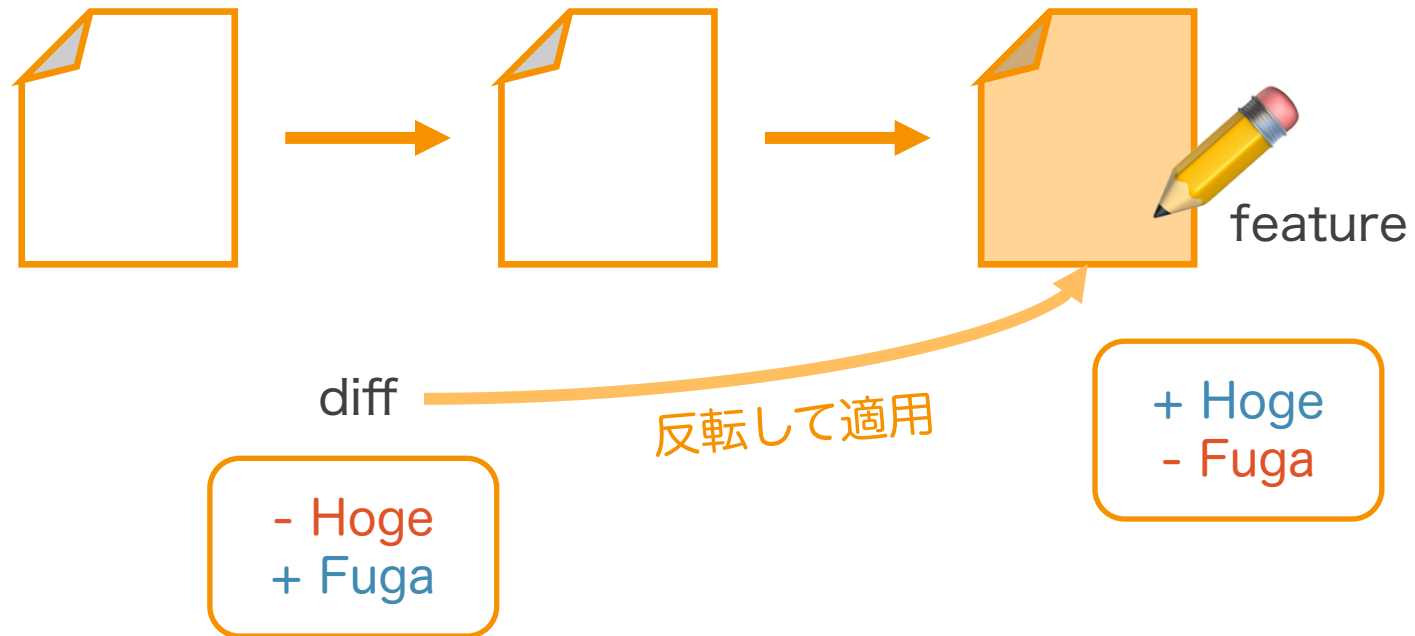
- ・ ある差分を元に戻す



# 便利コマンドたち

---

- ・ ある差分を元に戻す

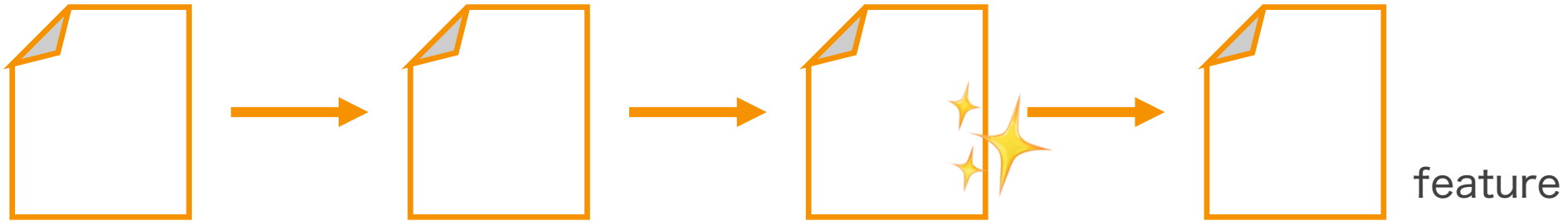




# 便利コマンドたち

---

- ・ ある差分を元に戻す



# 便利コマンドたち

---

- ・ 典型的な作業は Git で自動化できる
  - ・ ある差分を別のブランチに移植 -> cherry-pick
  - ・ ある差分を元に戻す -> revert
  - ・ 一連の差分をまるごと引っ越す

# 便利コマンドたち

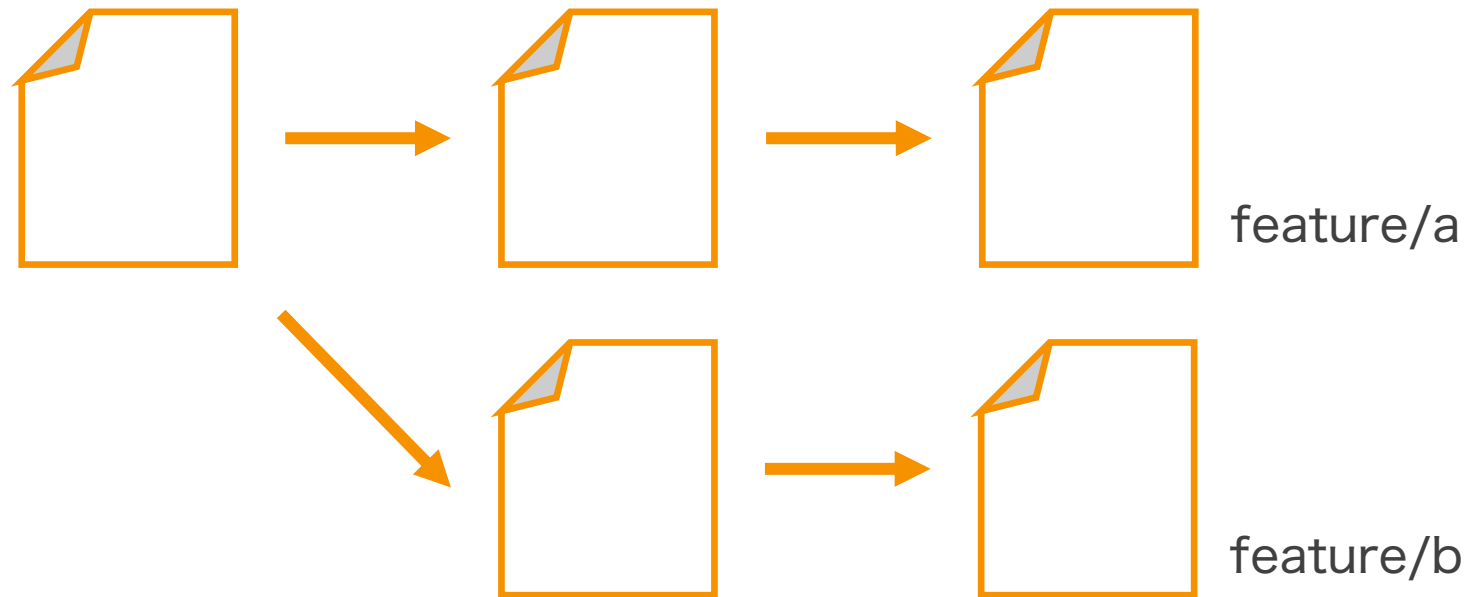
---

- ・ 典型的な作業は Git で自動化できる
  - ・ ある差分を別のブランチに移植 -> cherry-pick
  - ・ ある差分を元に戻す -> revert
  - ・ 一連の差分をまるごと引っ越す

# 便利コマンドたち

---

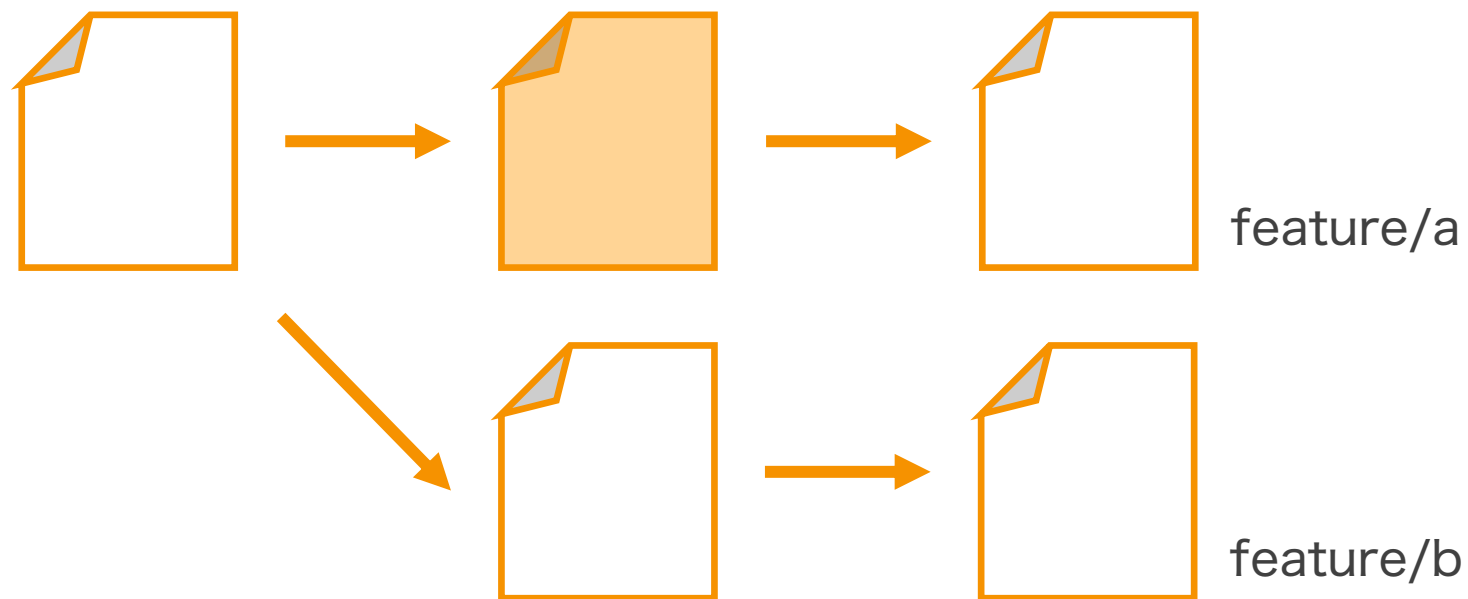
- ・一連の差分をまるごと引っ越す



# 便利コマンドたち

---

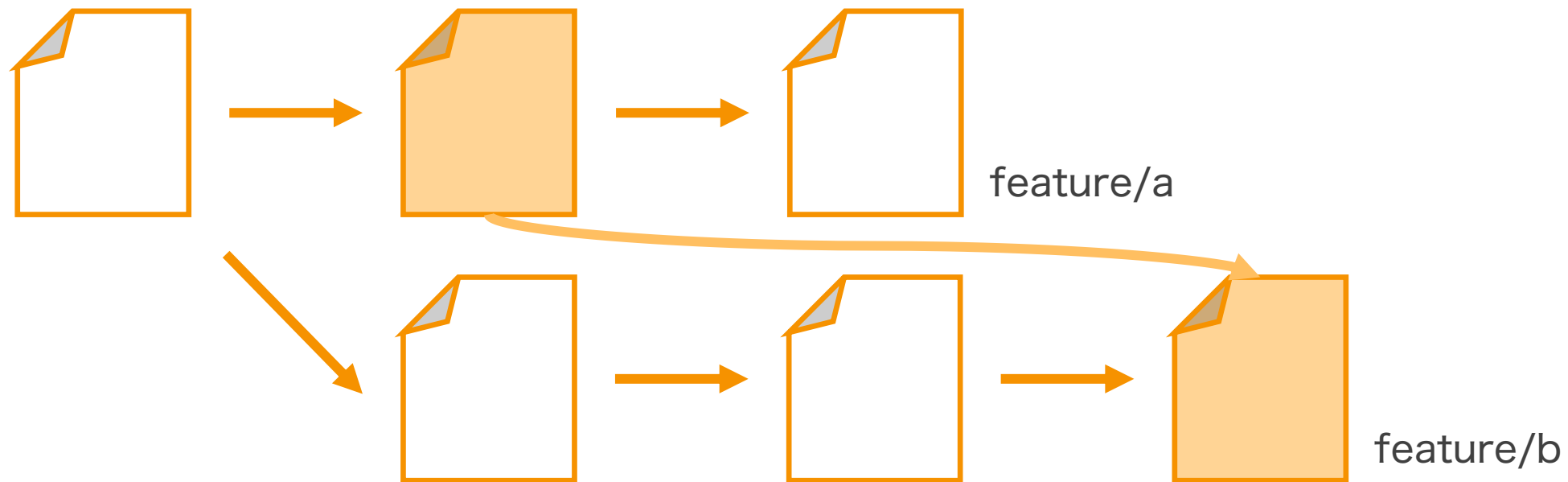
- ・一連の差分をまるごと引っ越す



# 便利コマンドたち

---

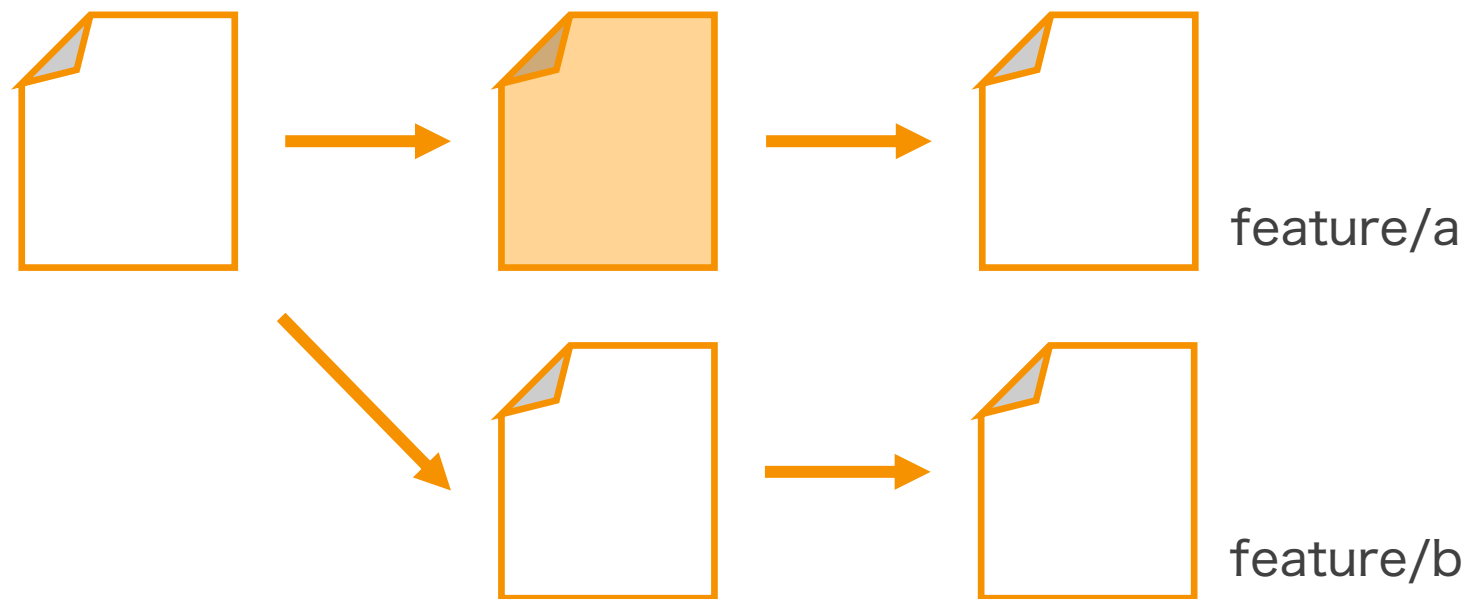
- ・一連の差分をまるごと引っ越す



# 便利コマンドたち

---

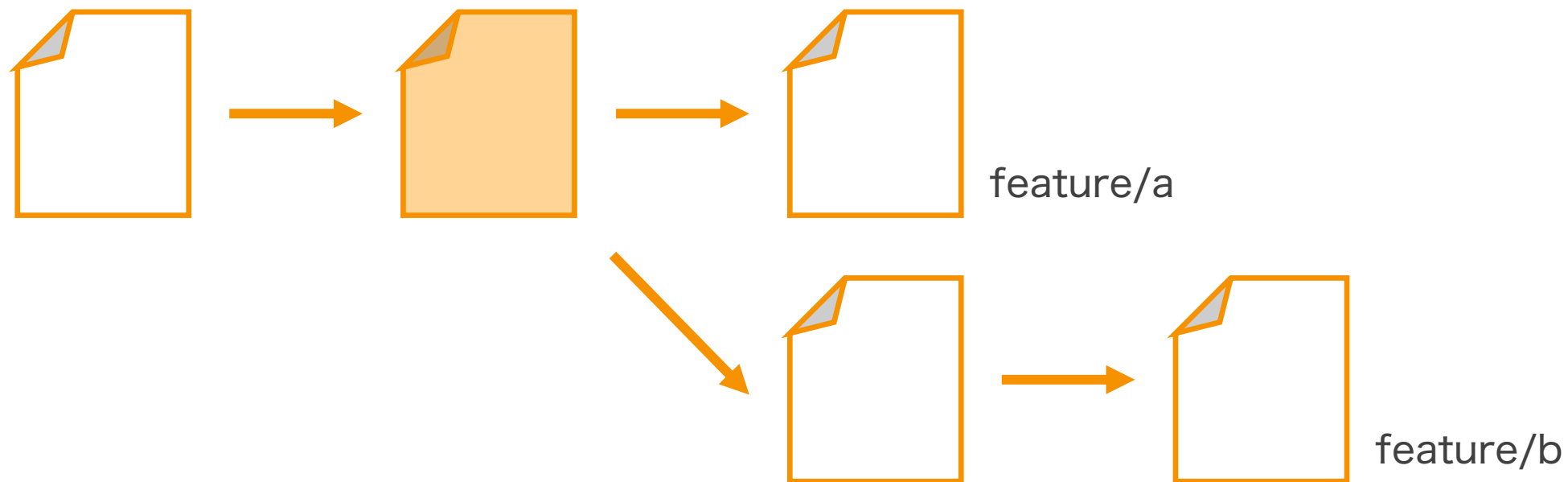
- ・一連の差分をまるごと引っ越す



# 便利コマンドたち

---

- ・一連の差分をまるごと引っ越す

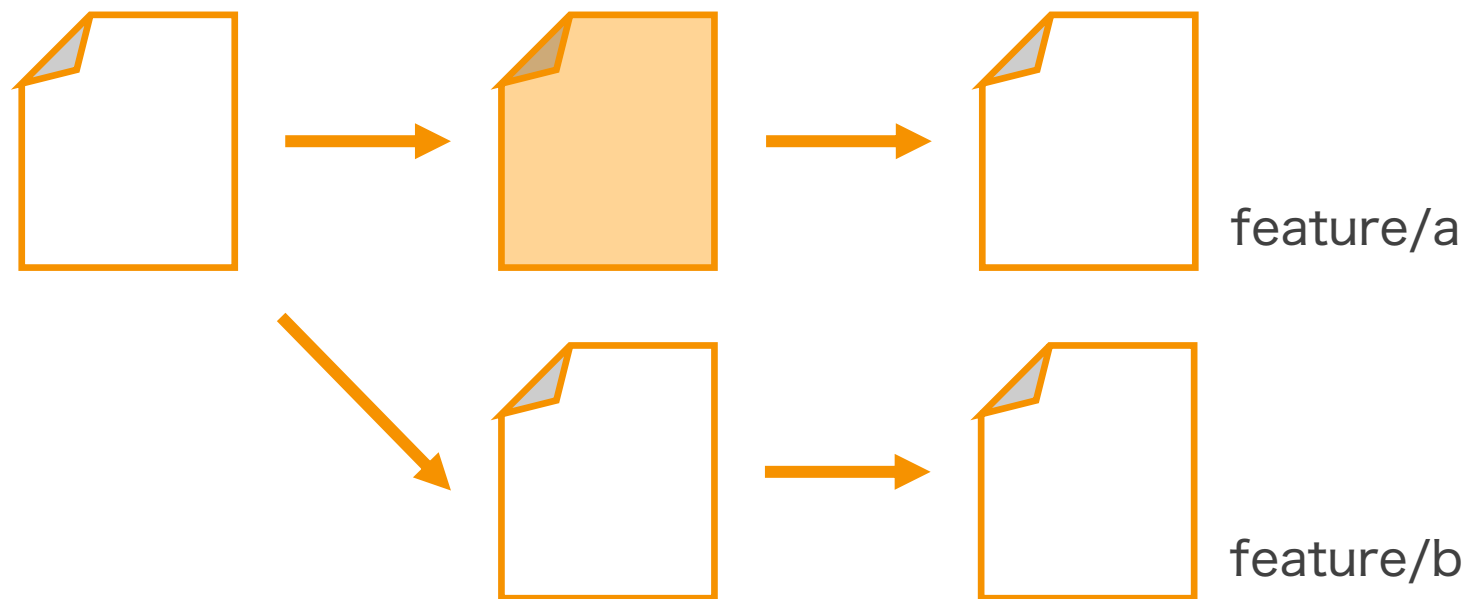




# 便利コマンドたち

---

- ・一連の差分をまるごと引っ越す



# 便利コマンドたち

---

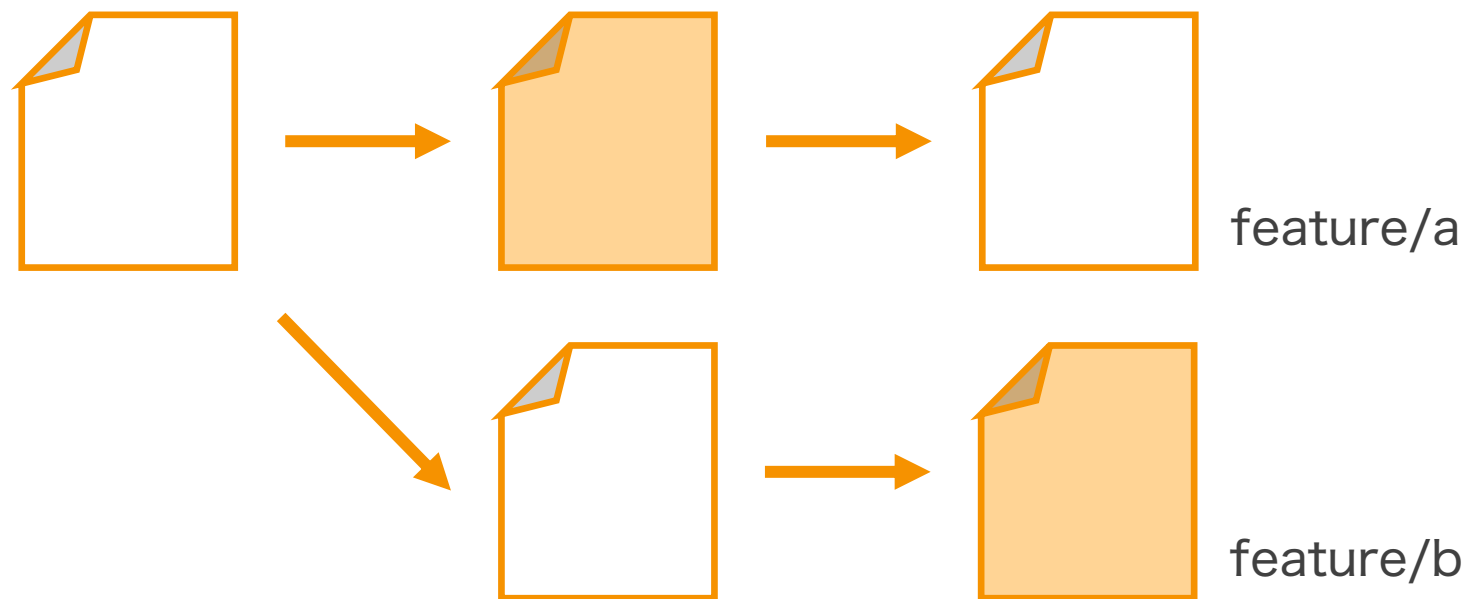
- ・一連の差分をまるごと引っ越す

```
$ git rebase <commit-id>
```

# 便利コマンドたち

---

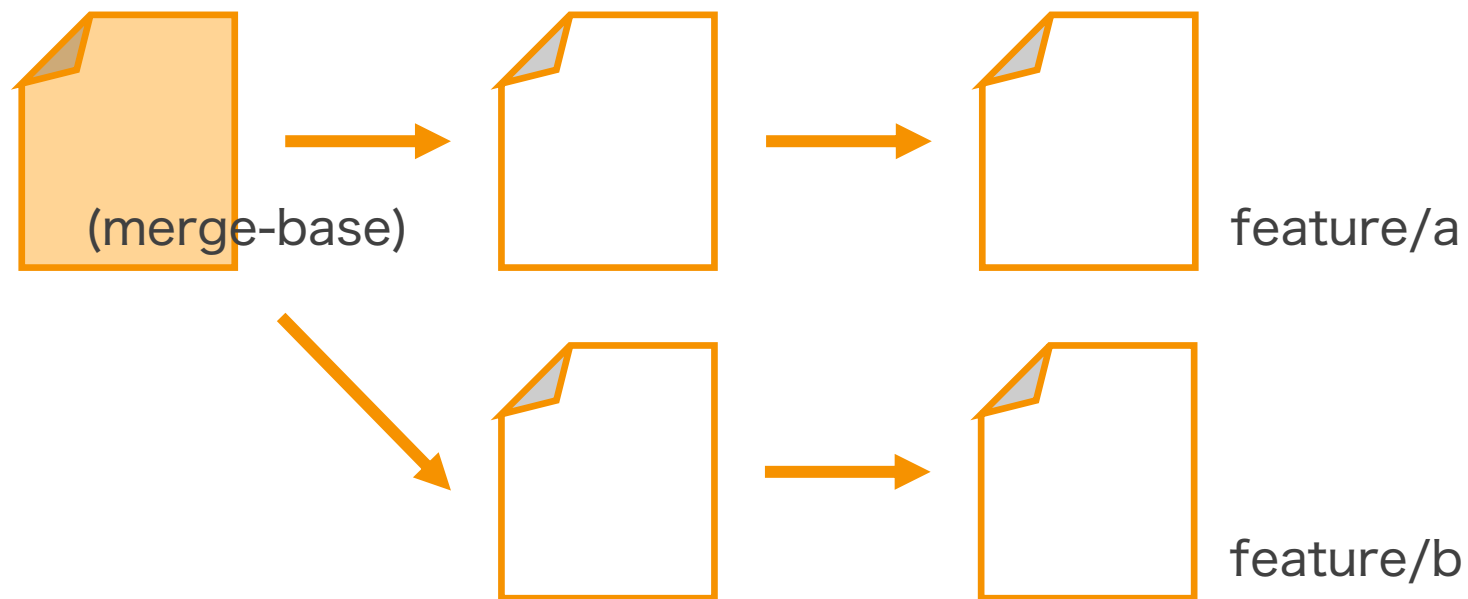
- ・一連の差分をまるごと引っ越す



# 便利コマンドたち

---

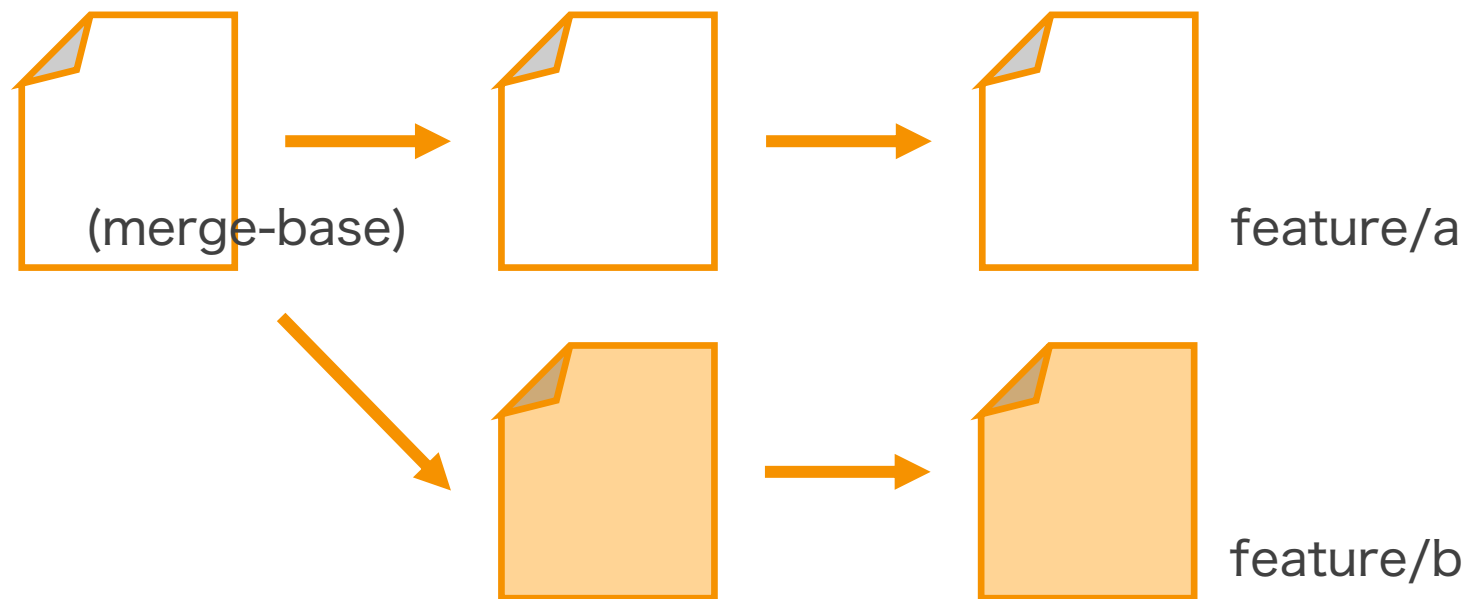
- 一連の差分をまるごと引っ越す



# 便利コマンドたち

---

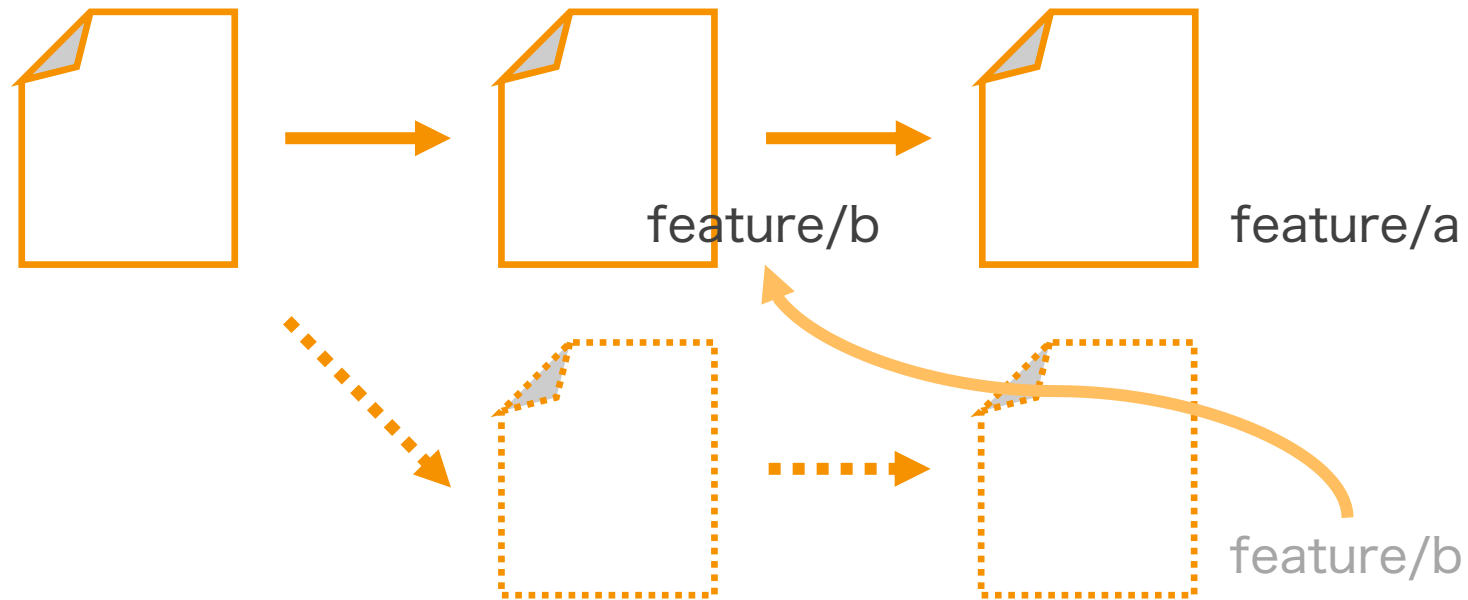
- ・ 一連の差分をまるごと引っ越す



# 便利コマンドたち

---

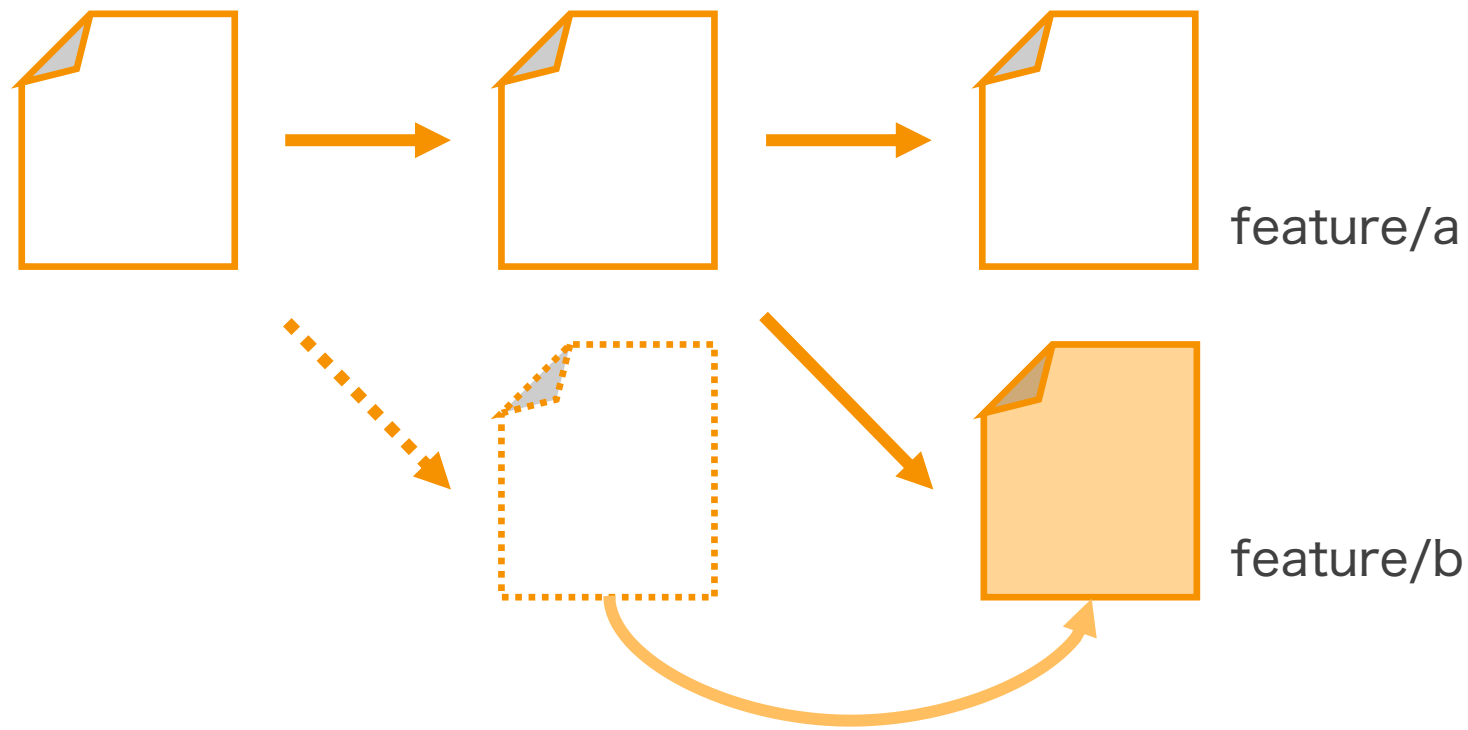
- ・一連の差分をまるごと引っ越す



# 便利コマンドたち

---

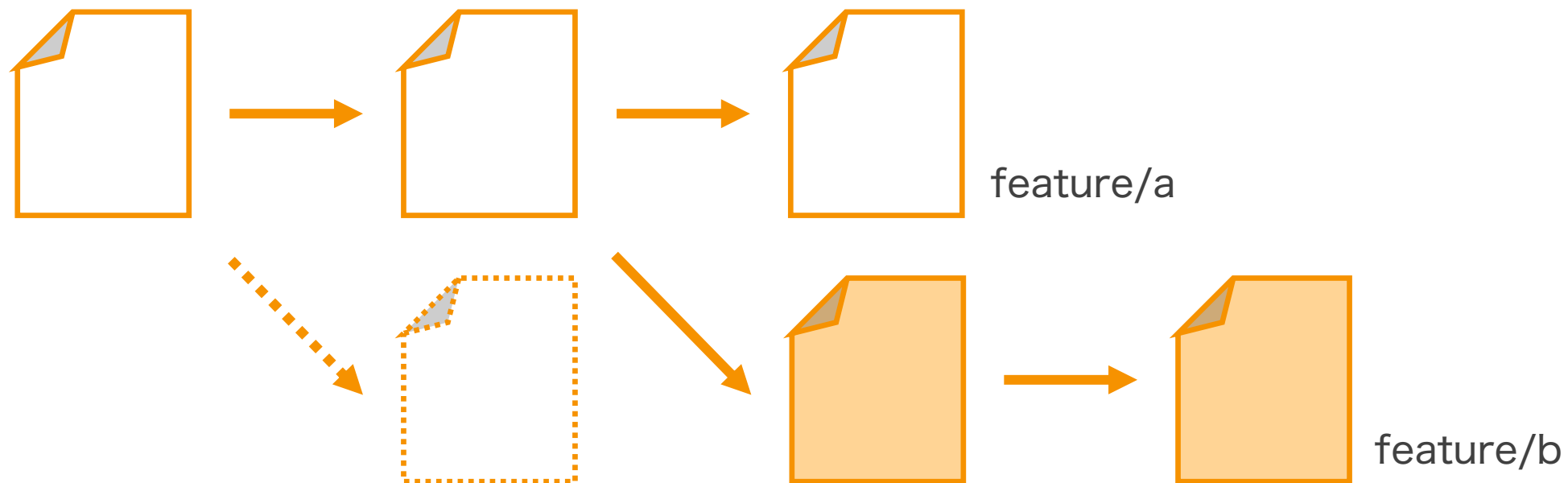
- ・一連の差分をまるごと引っ越す



# 便利コマンドたち

---

- ・一連の差分をまるごと引っ越す





# 便利コマンドたち

---

- ・ ちなみに

# 便利コマンドたち

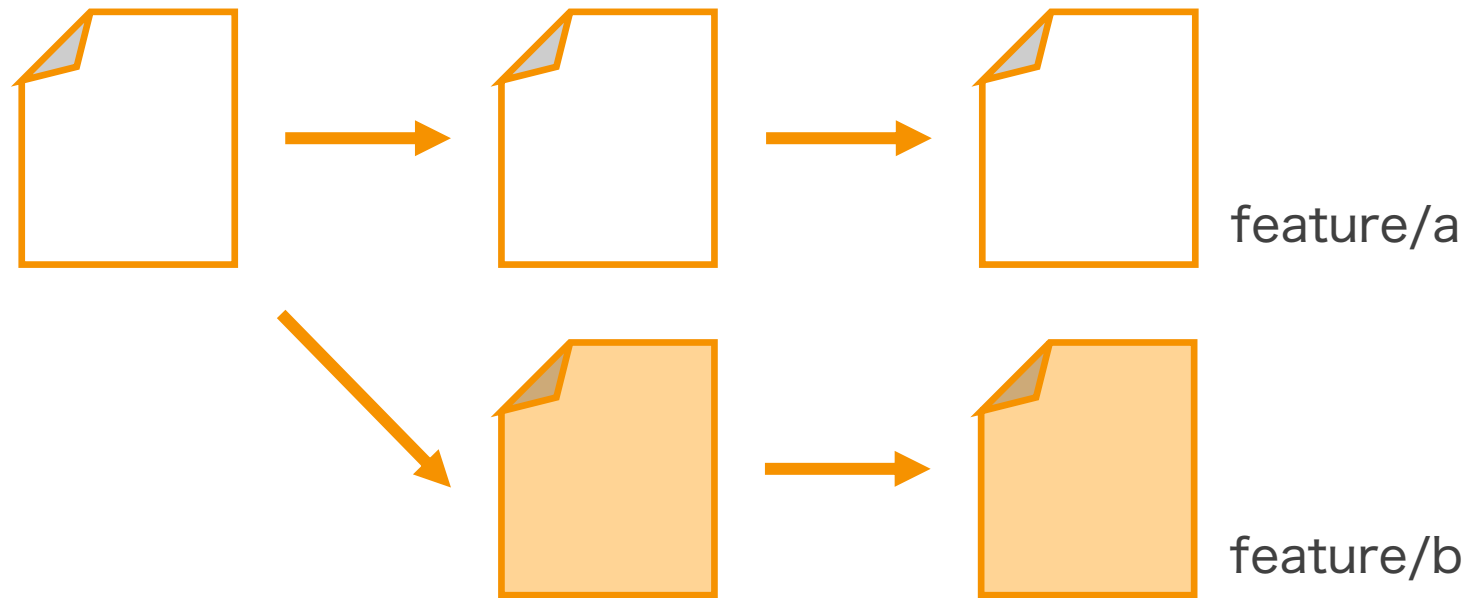
---

- ・ ちなみに
  - ・ 引っ越し先は自分自身でもよい

# 便利コマンドたち

---

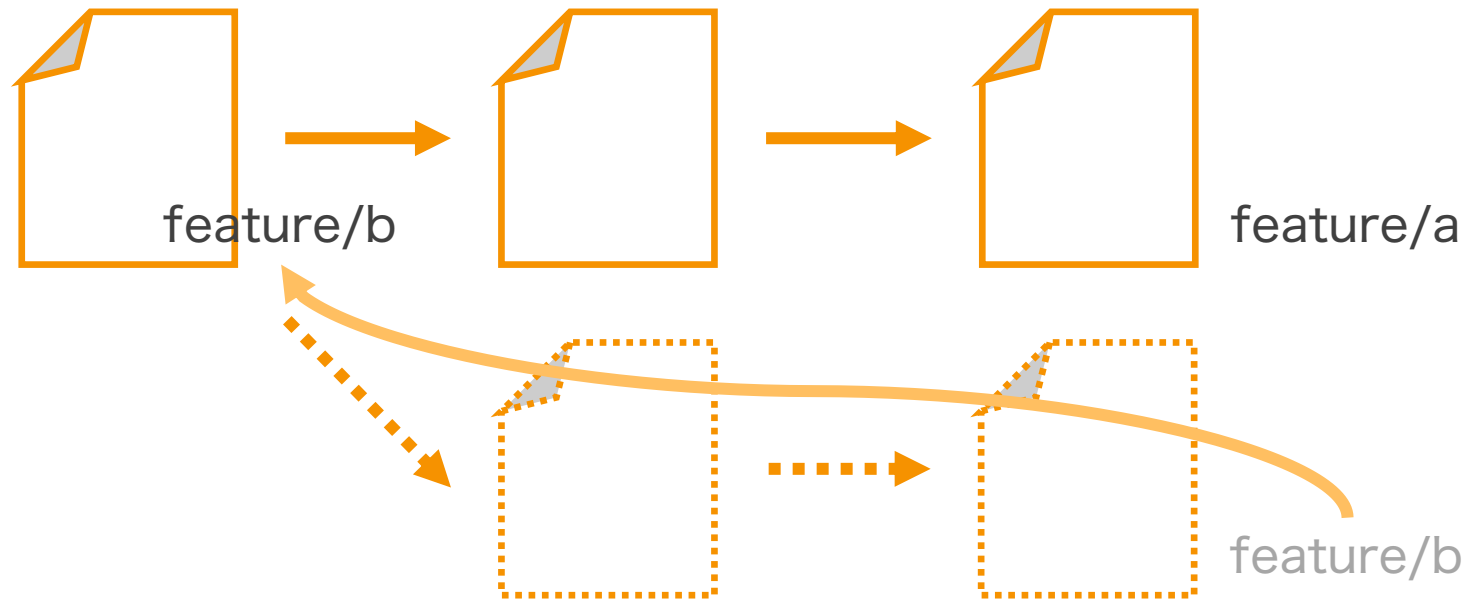
- ・ ちなみに



# 便利コマンドたち

---

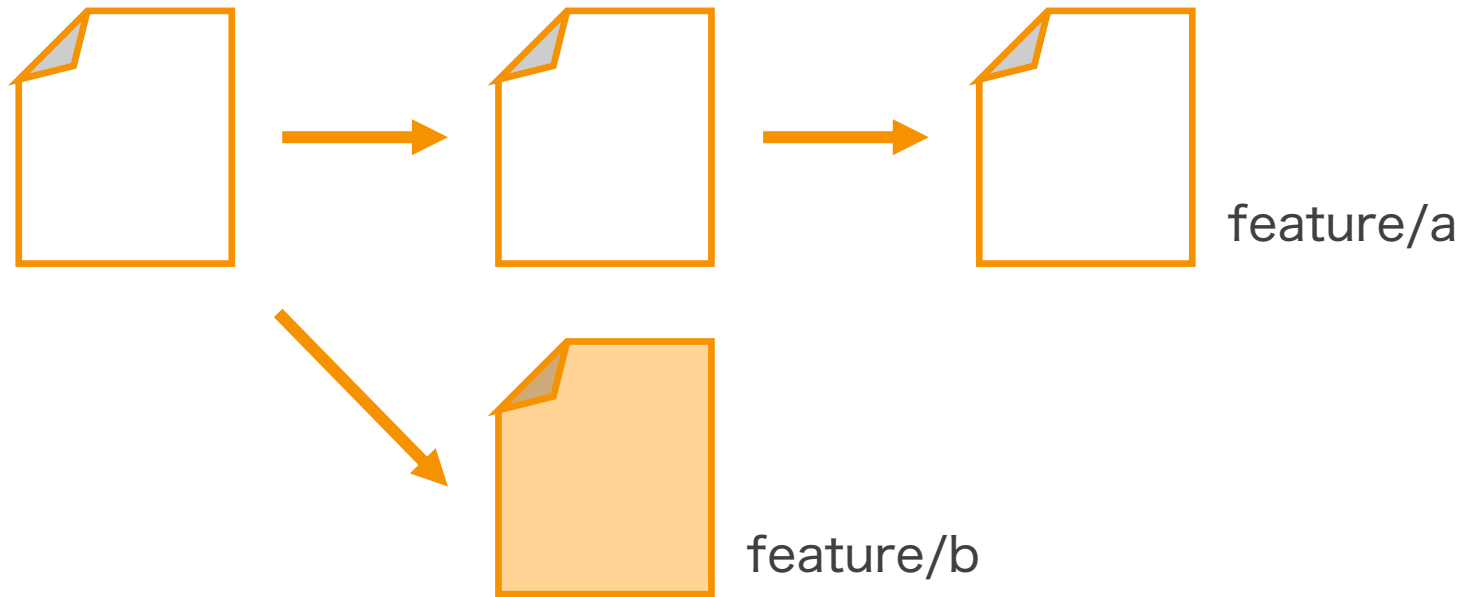
- ・ ちなみに



# 便利コマンドたち

---

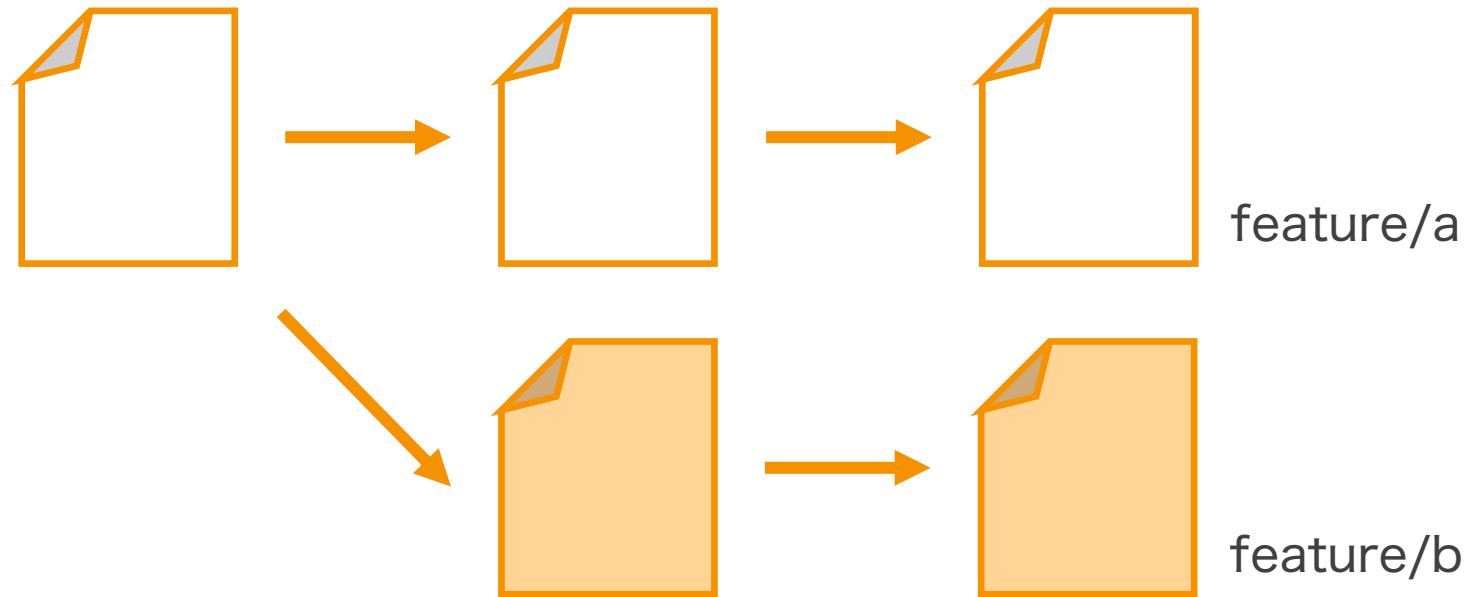
- ・ ちなみに



# 便利コマンドたち

---

- ・ ちなみに



# 便利コマンドたち

---

- ・ ちなみに
  - ・ 引っ越し先は自分自身でもよい
    - ・ なんか意味あるの？

# 便利コマンドたち

---

- ・ ちなみに
  - ・ 引っ越し先は自分自身でもよい
    - ・ なんか意味あるの？
      - ・ 引っ越しついでに簡単な修正をする -i オプションが便利



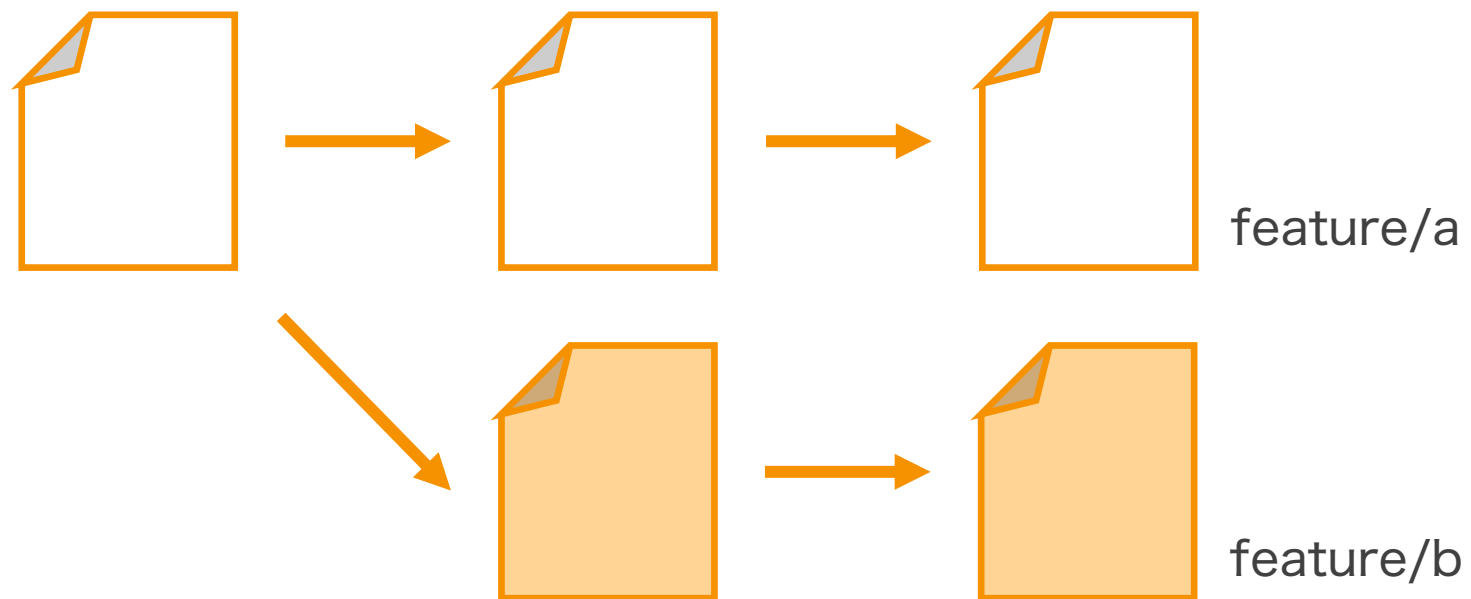
# 便利コマンドたち

---

- ・ ちなみに
  - ・ 引っ越し先は自分自身でもよい
    - ・ なんか意味あるの？
      - ・ 引っ越しついでに簡単な修正をする -i オプションが便利
        - ・ コミットを並べ替える
        - ・ いくつかのコミットを一つに合体
        - ・ コミットメッセージを変更
        - ・ … など

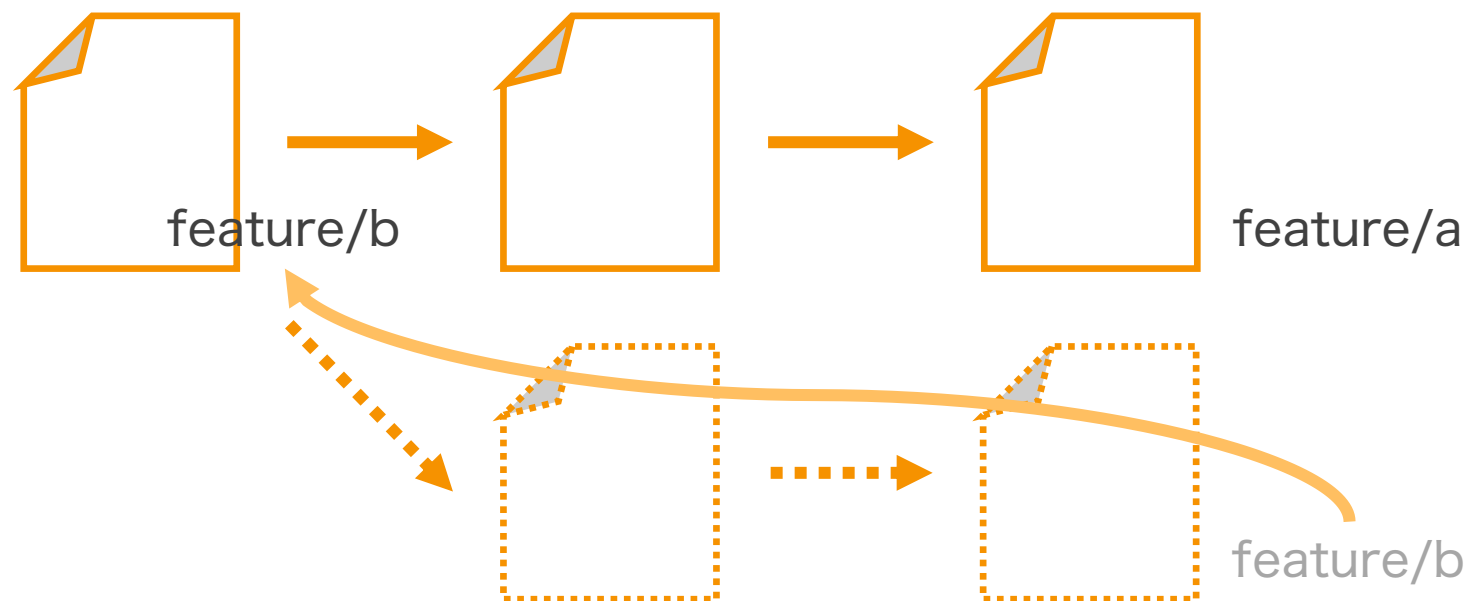
# 便利コマンドたち

---



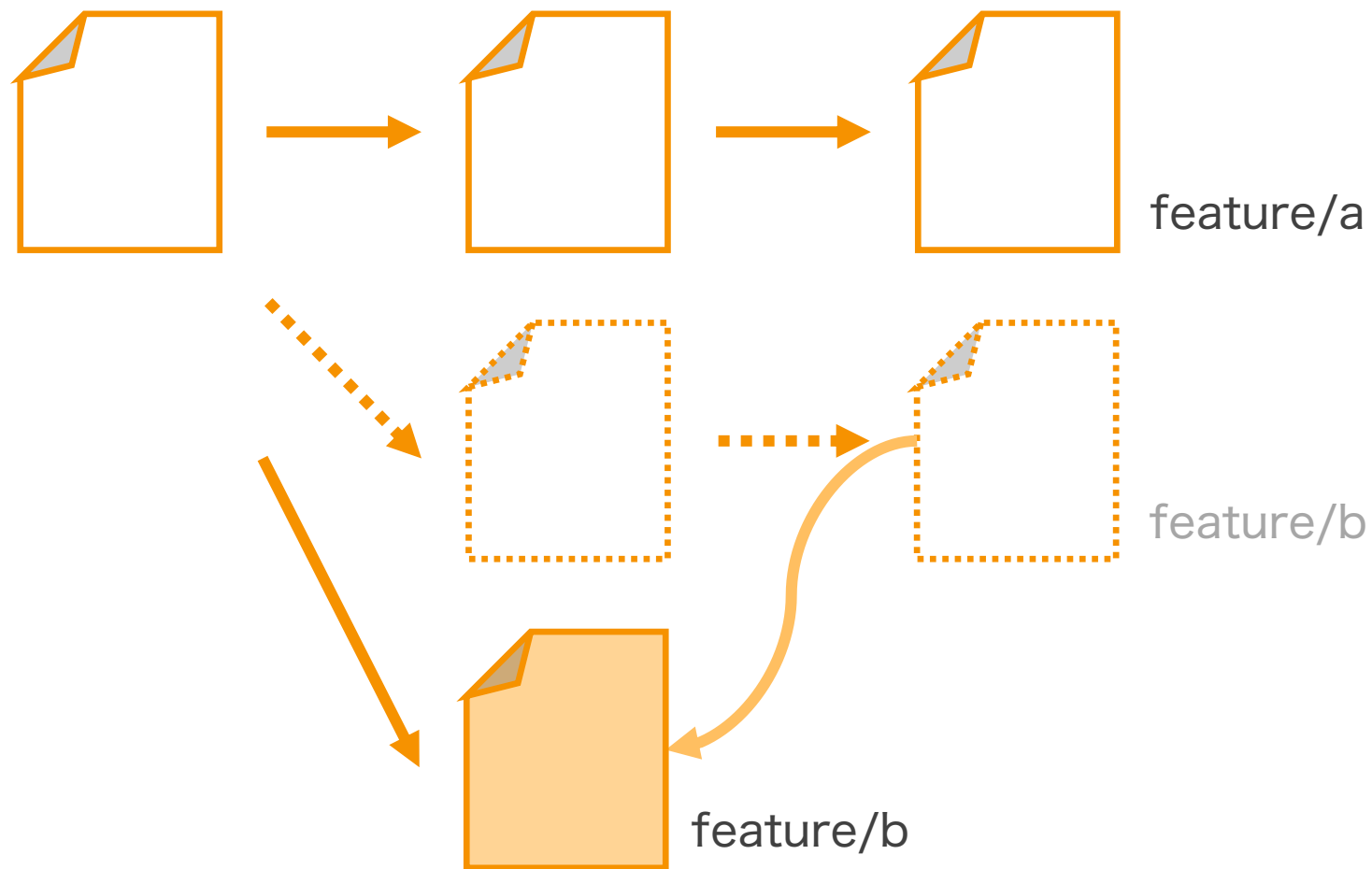
# 便利コマンドたち

---



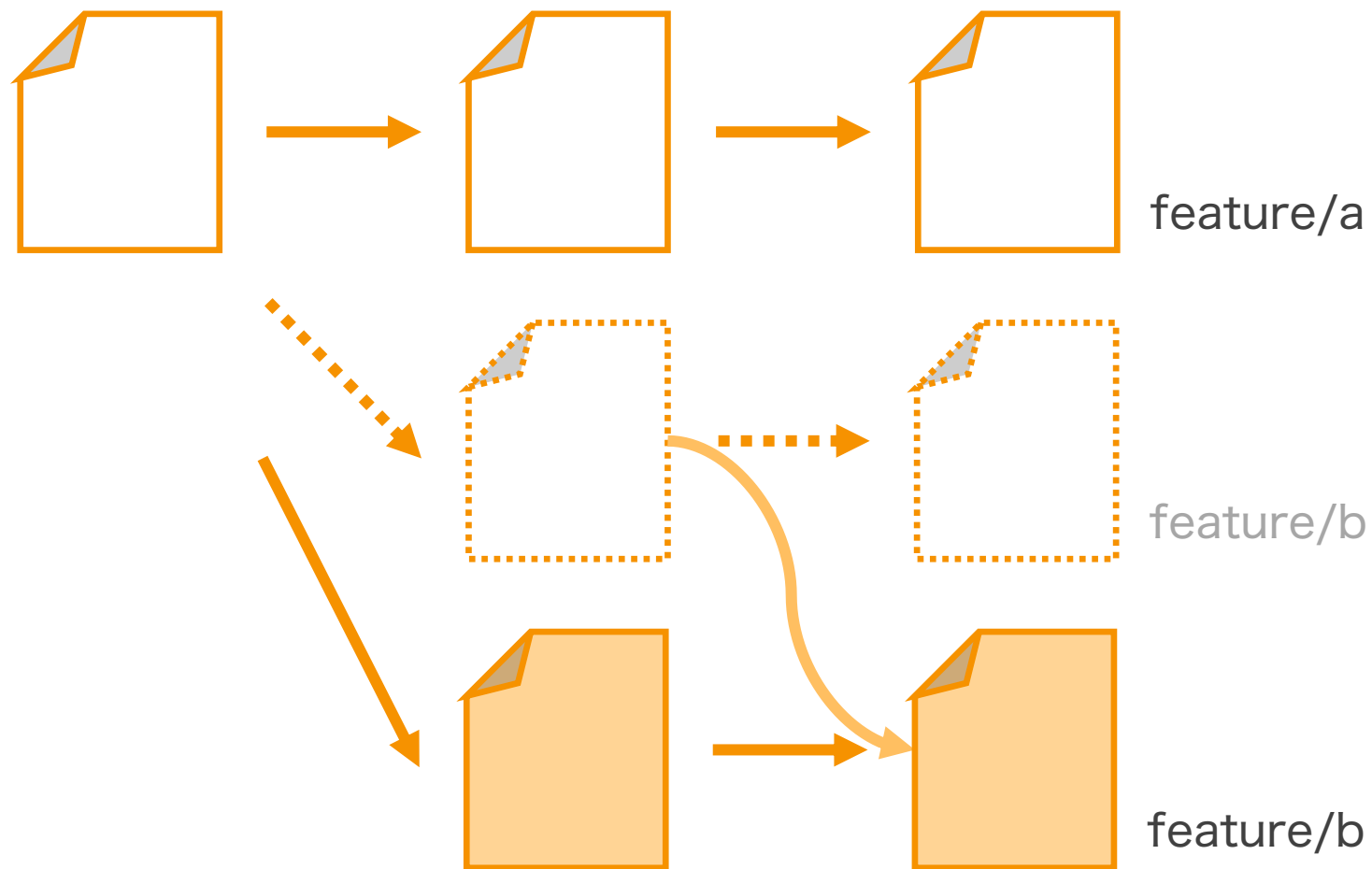
# 便利コマンドたち

---



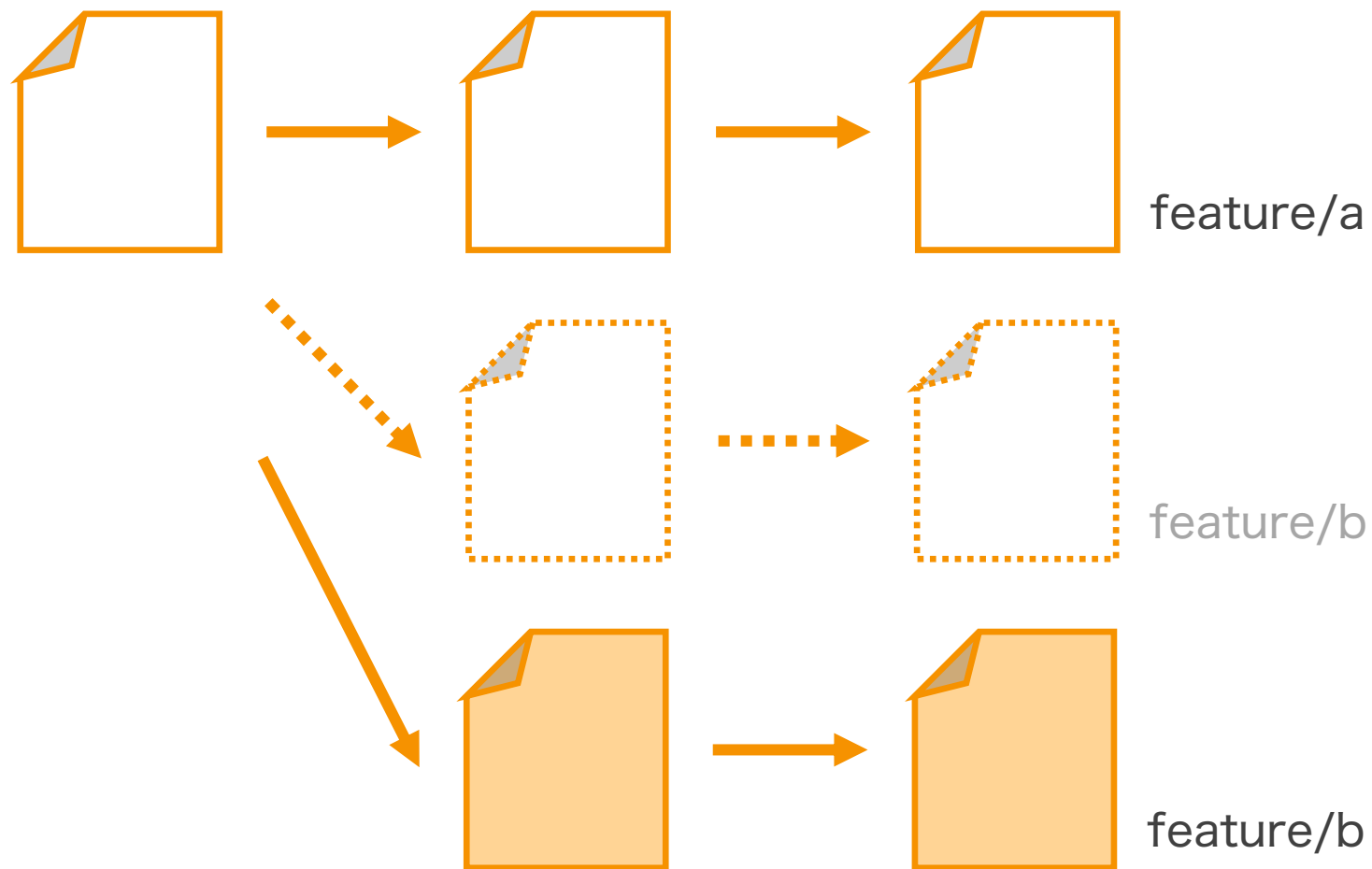
# 便利コマンドたち

---



# 便利コマンドたち

---



# 便利コマンドたち

---

- ・ rebase の罫

# 便利コマンドたち

---

- rebase の罨
  - コミットログが大胆に書き換わる
  - push 済みのブランチに使うと、もちろん push できなくなる



Git をしくみから理解する

# Git をしくみから理解する

---

- このコーナーの目標
  - Git の基本機能を「自信を持って」使える
    - commit
    - branch
    - rebase
    - merge
  - 実用上の小ワザなどは実習でググりながら練習します

# Git をしくみから理解する

---

- このコーナーの目標
  - Git の基本機能を「自信を持って」使える
    - commit
    - branch
    - rebase
    - merge
  - 実用上の小ワザなどは実習でググりながら練習します

# 実習② プチ Git Challenge

## 実習② プチ Git Challenge

---

- ・の前に休憩 time ...



## 実習② プチ Git Challenge

---

- ・ プチ Git Challenge   




## 実習② プチ Git Challenge

---

- ・ プチ Git Challenge   
- ・ 悲しみを背負った Git リポジトリたちを救出していく大会

## 実習② プチ Git Challenge

---

- ・ プチ Git Challenge   
  - ・ 悲しみを背負った Git リポジトリたちを救出していく大会
  - ・ 過去問から教育的な問題をピックアップしました
    - ・ 参加者にしか公開していないので、**社外秘**です