寝起きの人でも

分かる git

おはよう!!

git とは？

履歴管理が

できる技術

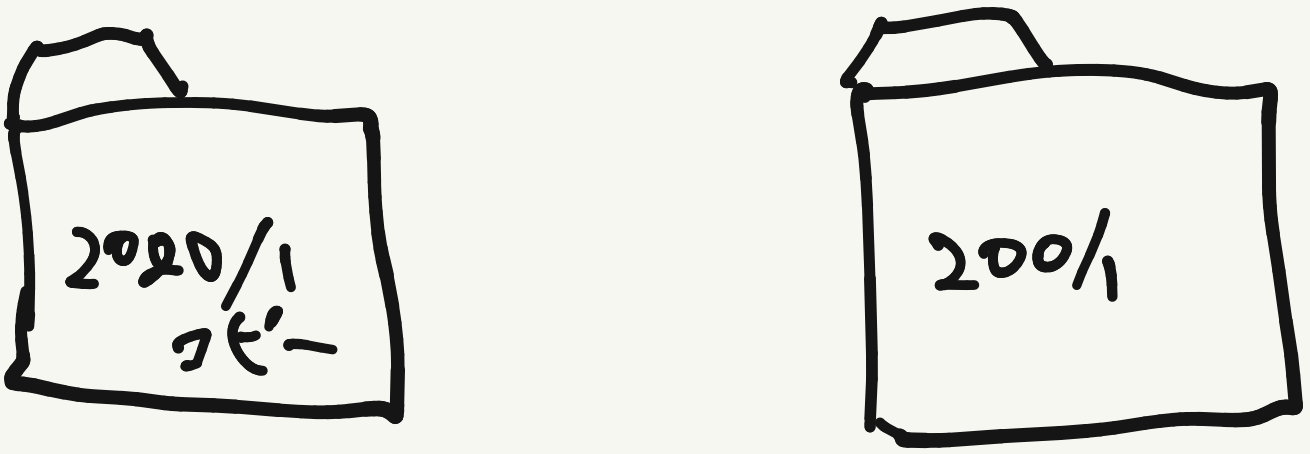
（ざっくり）

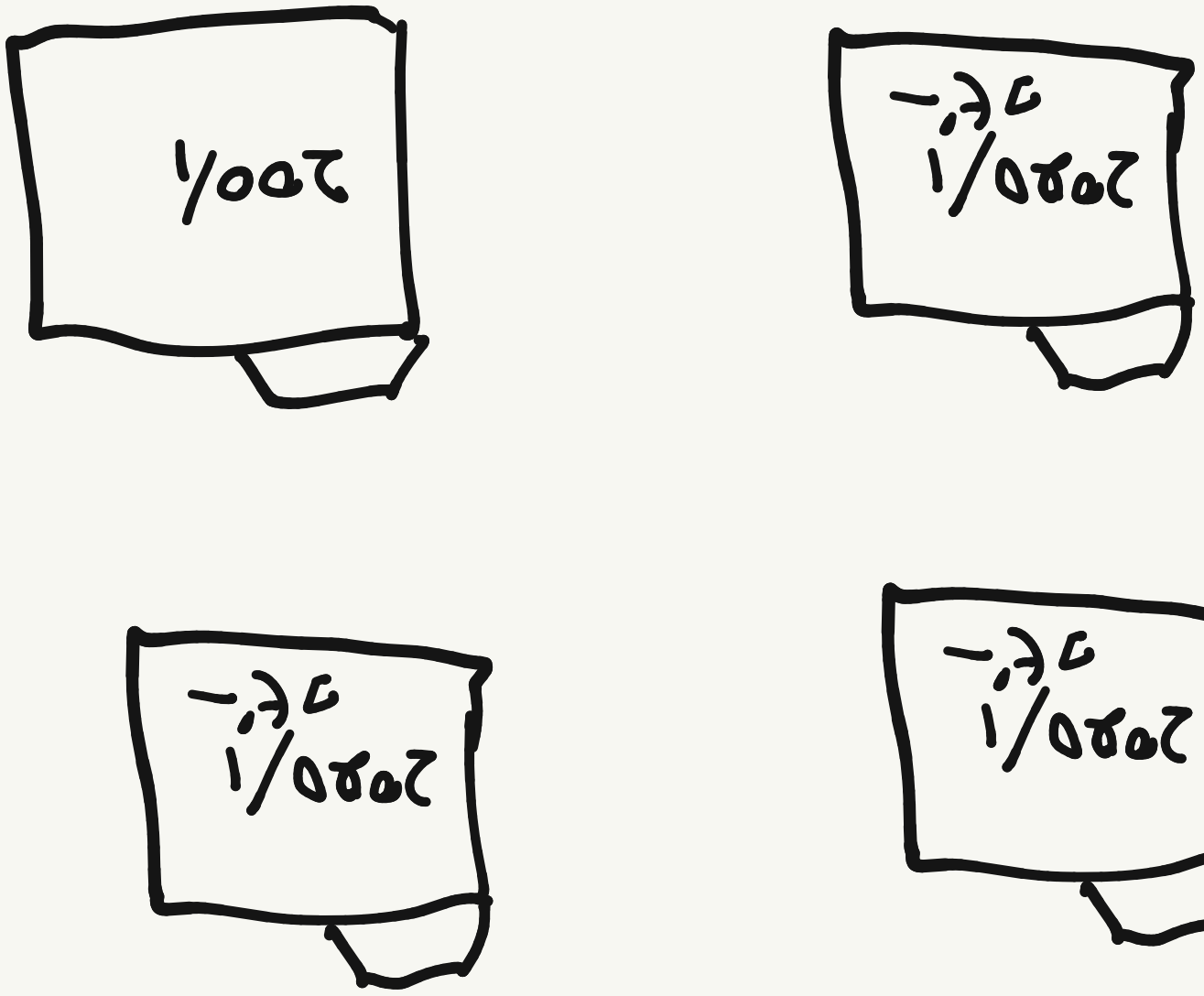
フォルダや

ファイルの

履歴

・バックアップがとりたい
場合

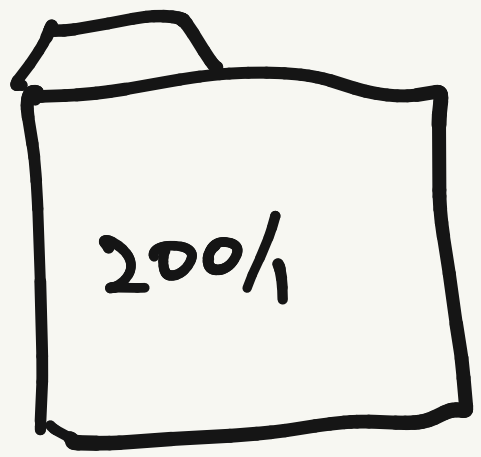



2020/1
ד"ב
2020/1
ד"ב
2020/1
ד"ב
2020/1
ד"ב

つづい


2020/1 コピー
2020/1 コピー
2020/1 コピー
2020/1 コピー
2020/1 コピー
200/1
2020/1 コピー
200/1
2020/1 コピー
200/1
2020/1 コピー
200/1
2020/1 コピー
200/1
2020/1 コピー
200/1
2020/1 コピー
200/1

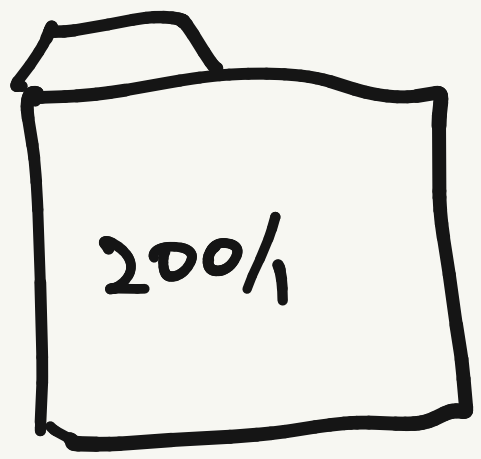
gitを使うと



これ自体が過去のある日点..
のフォルダ内の状態に戻れます

gitを使うと

履歴をもてる

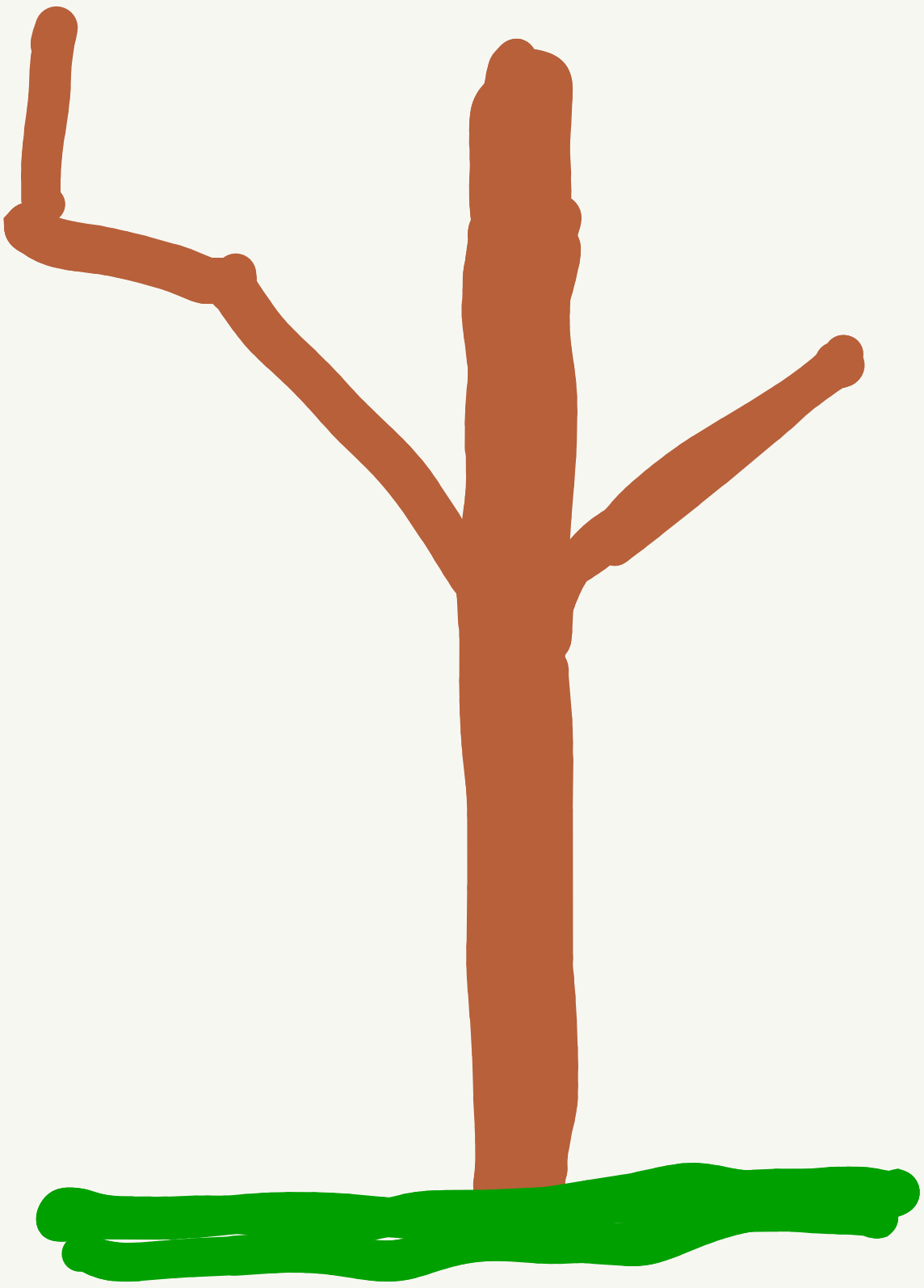


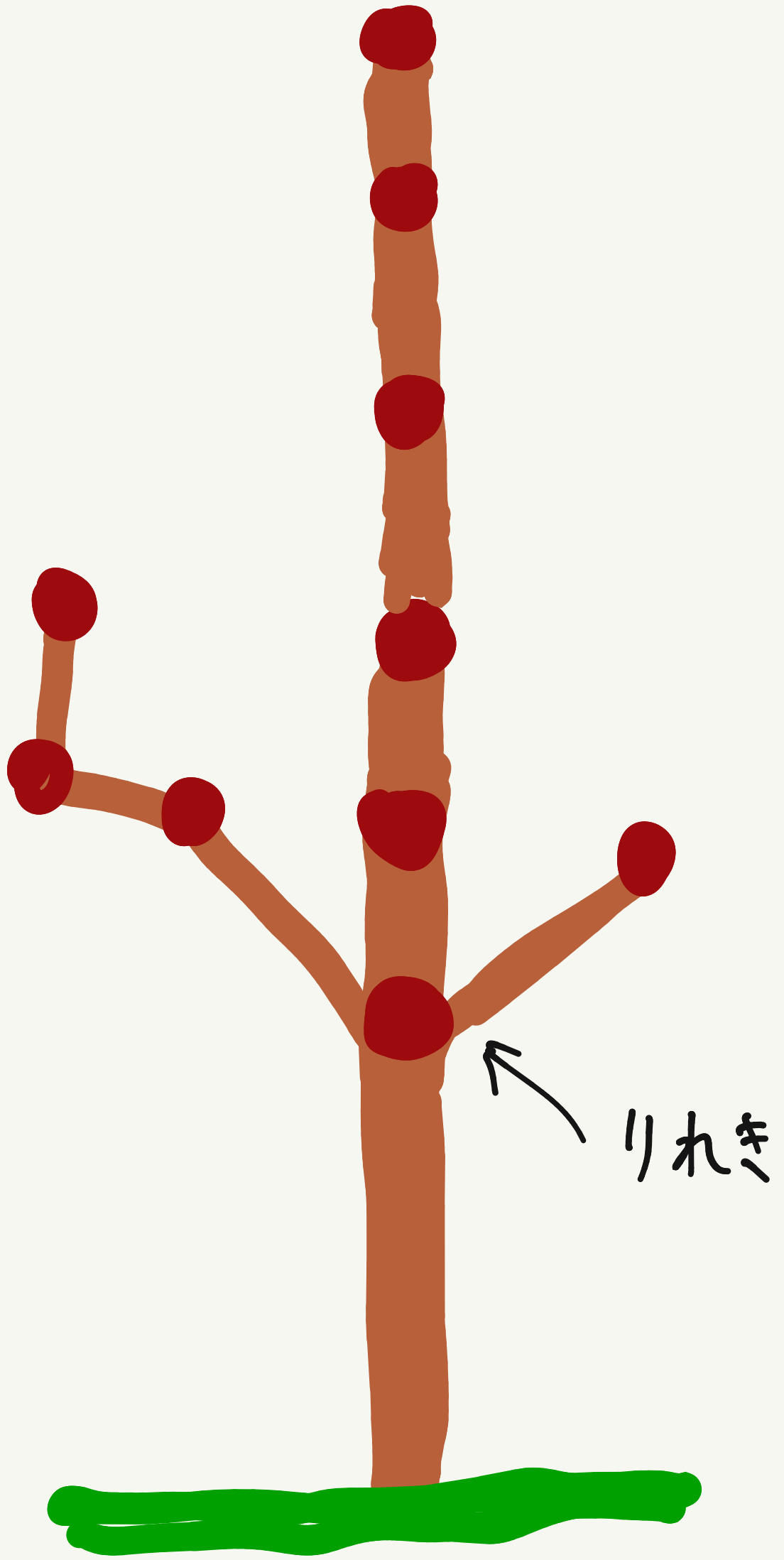
履歴のことを

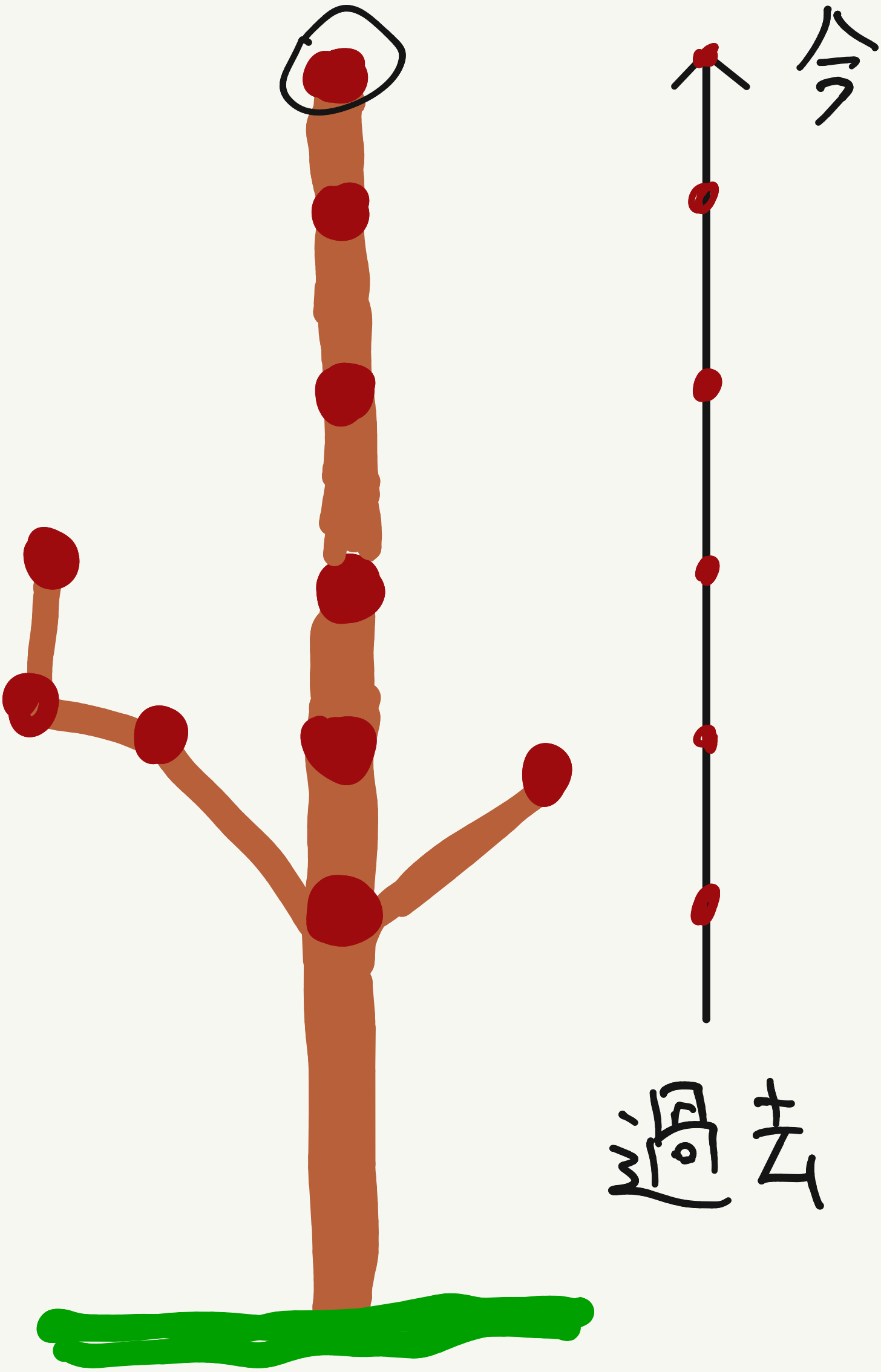
ユニット

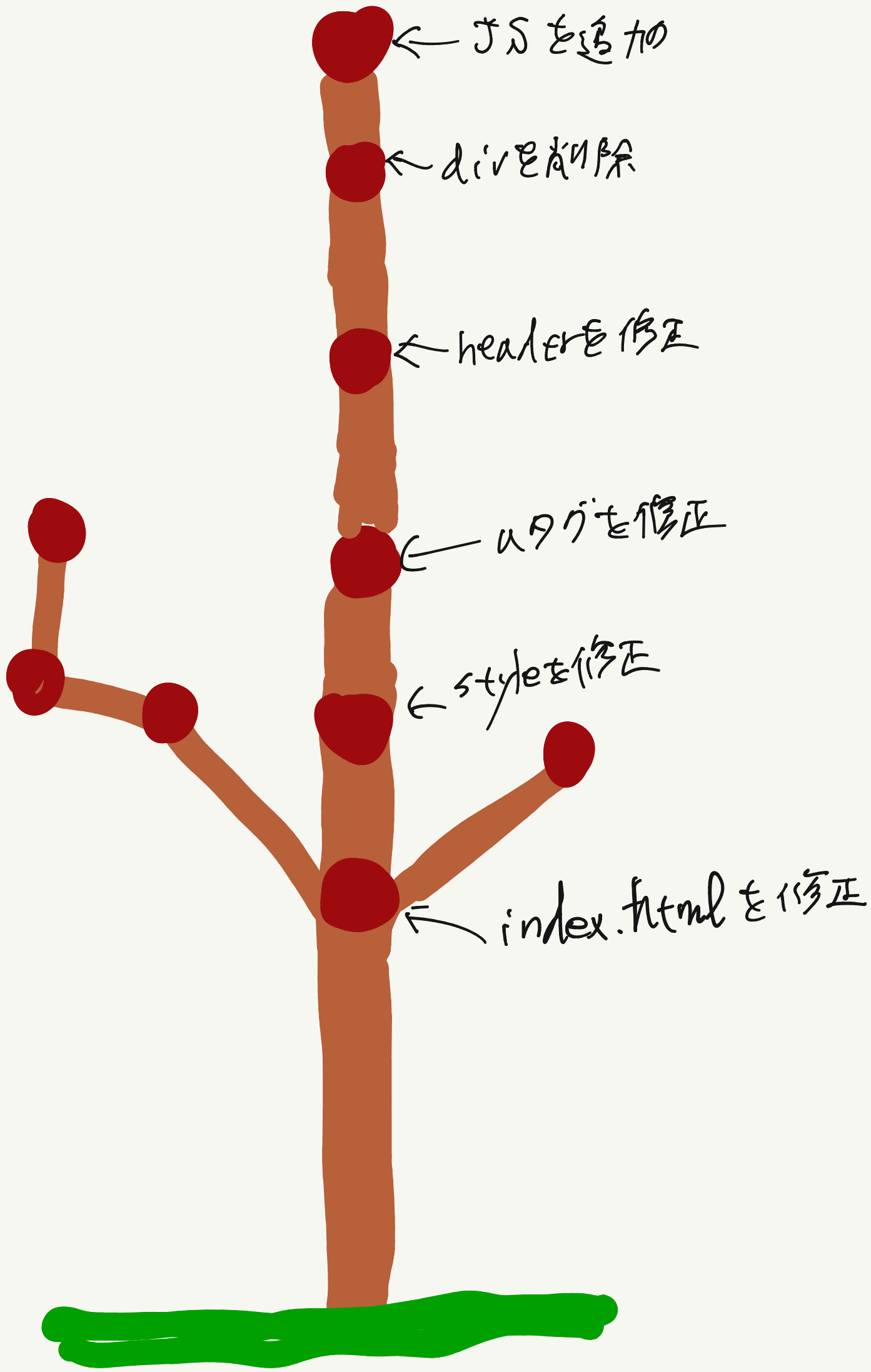
といいます

履歴（コミット）のつみ重ねは
branch（枝）と呼ばれています

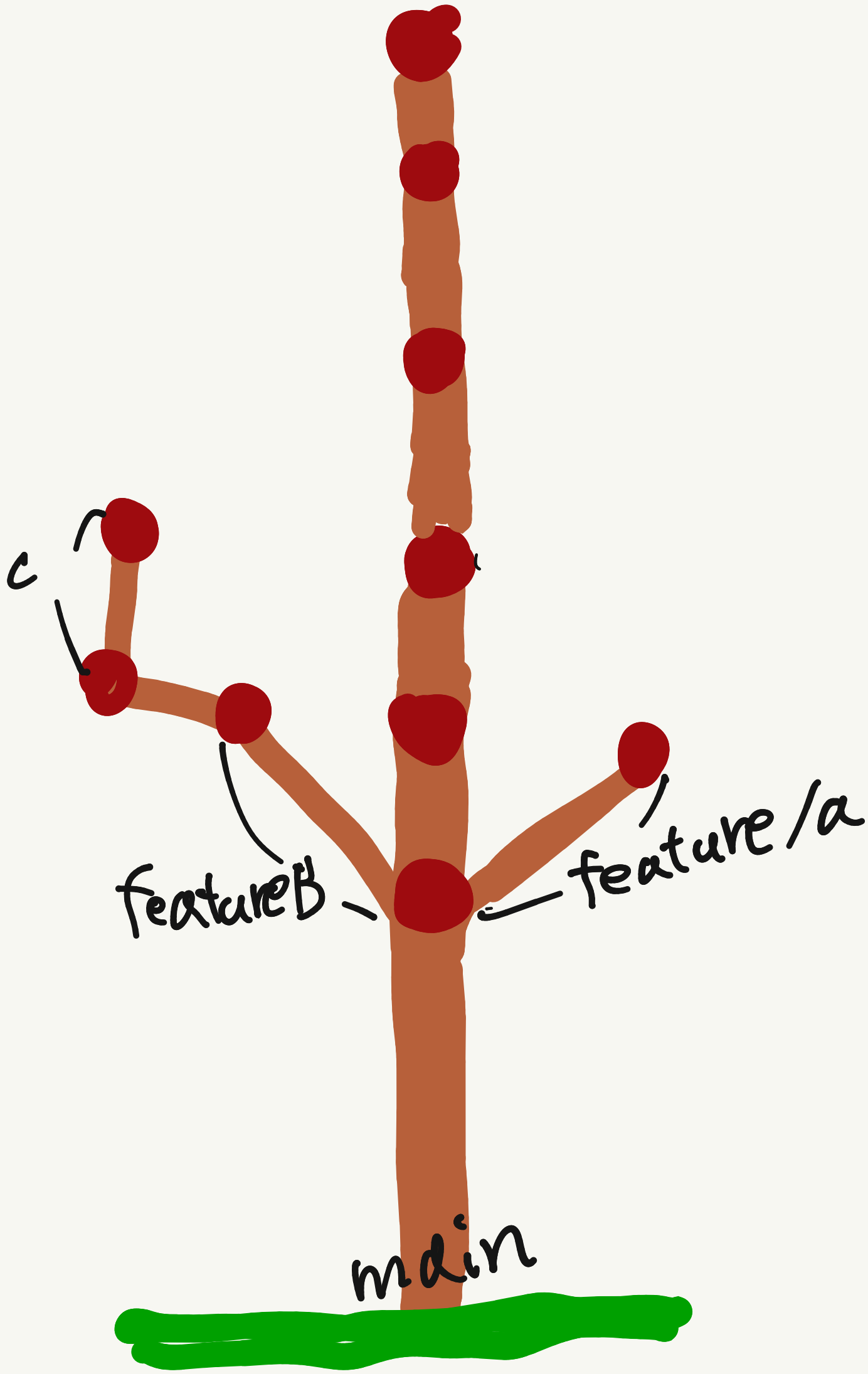



りれき


今
過去


JSを追加
divを削除
headerを修正
hタグを修正
styleを修正
index.htmlを修正

ブランチには好きな
名前がつけれます



一般的には

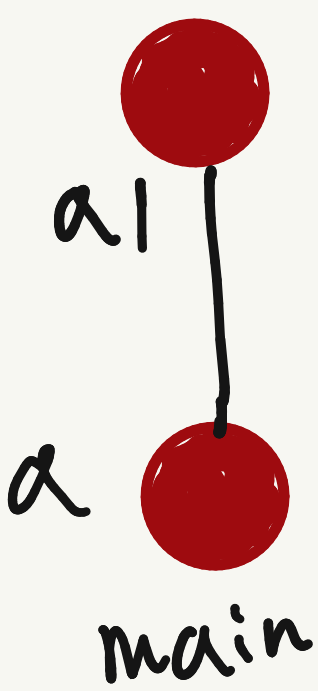
- master(main)
- release
- develope
- feature/[name]

一般的には

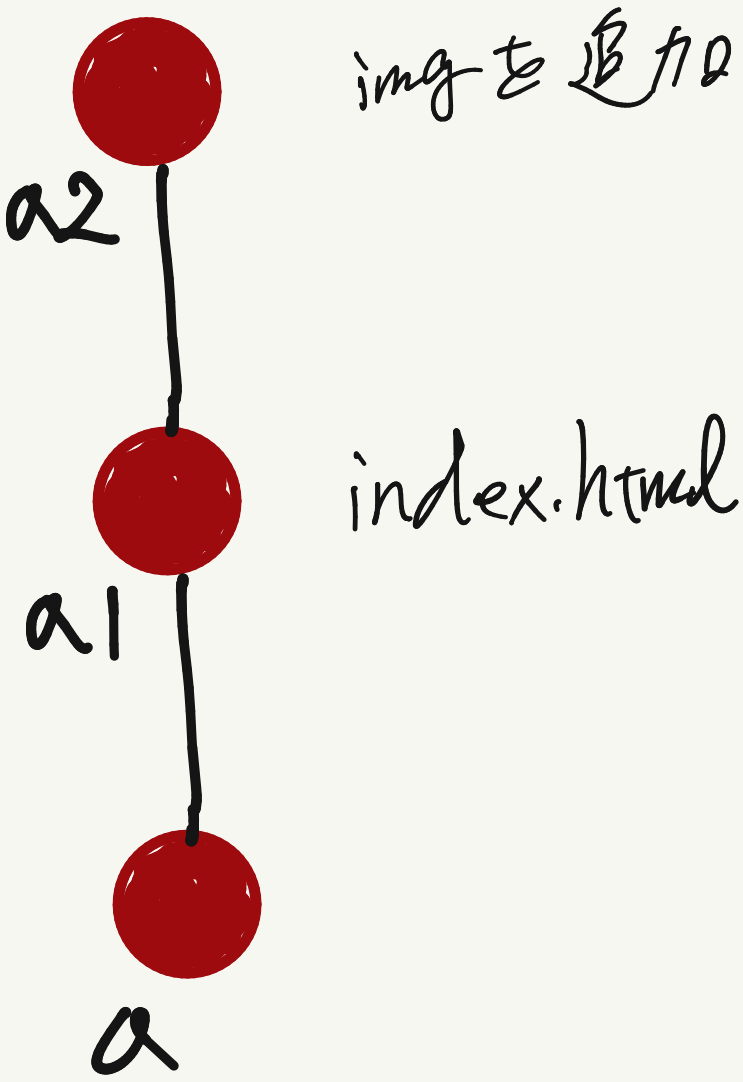
- master(main)
- release
- develope
- feature/[name]

大体これでok

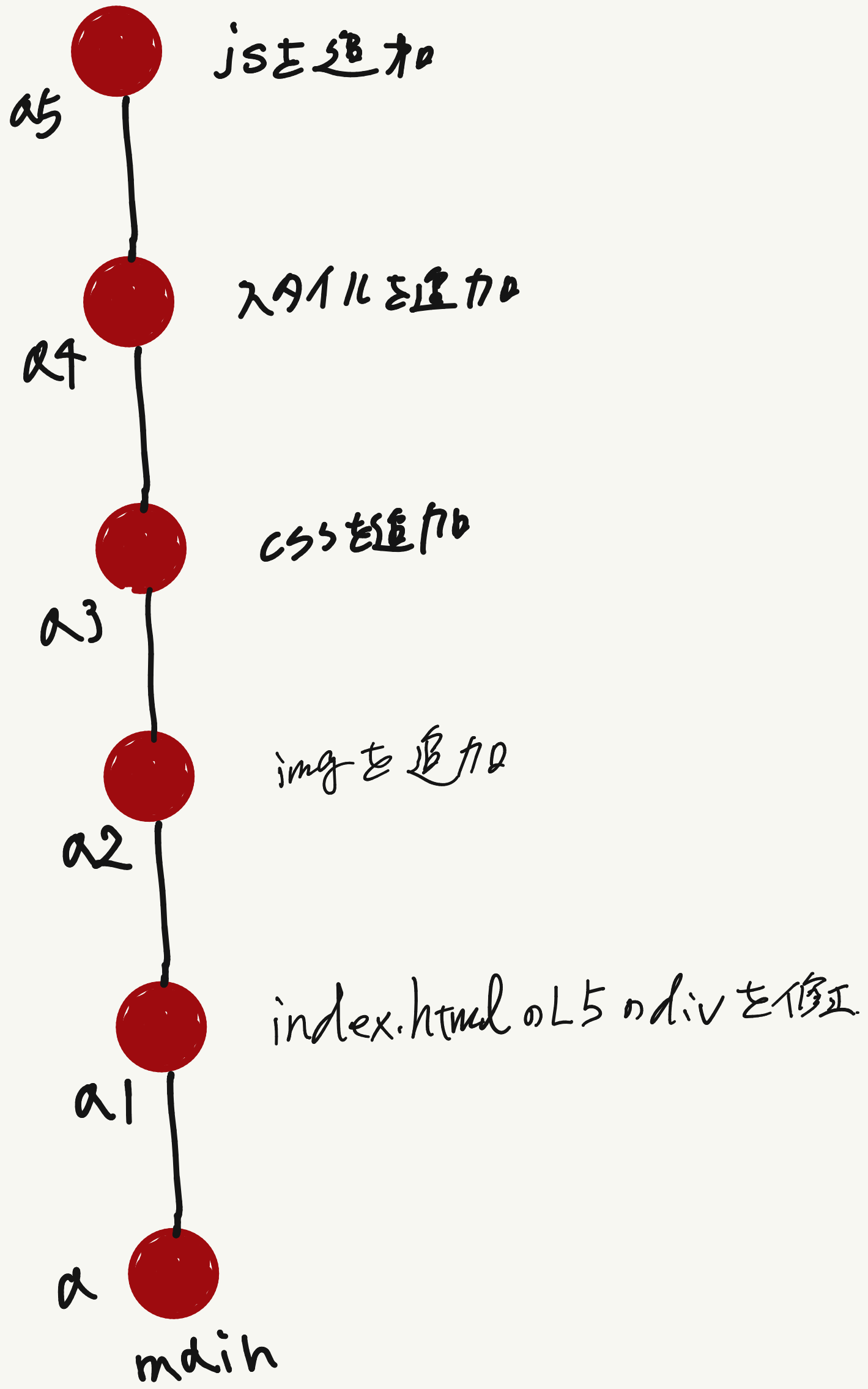
(詳しくはgit-flow
githubflow)



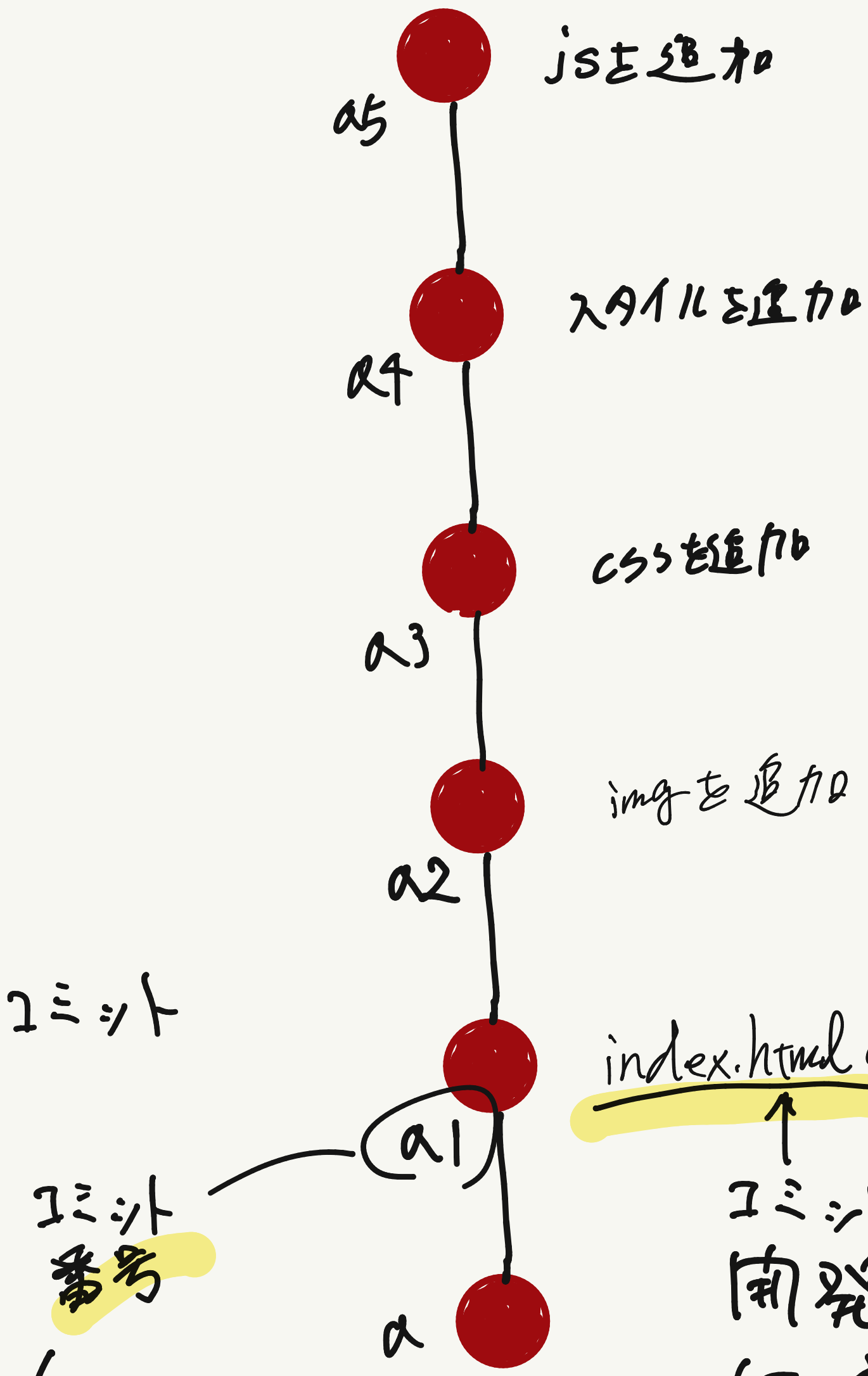
index.htmlの内容を修正


a2
a1
a
img を追加
index.html の L5 の div を修正

ロールプレイングゲームの「セーブ」に似ている



コミットには番号とメッセージがあり
メッセージは何を修正したかを書く.



index.html のL5のdivを修正

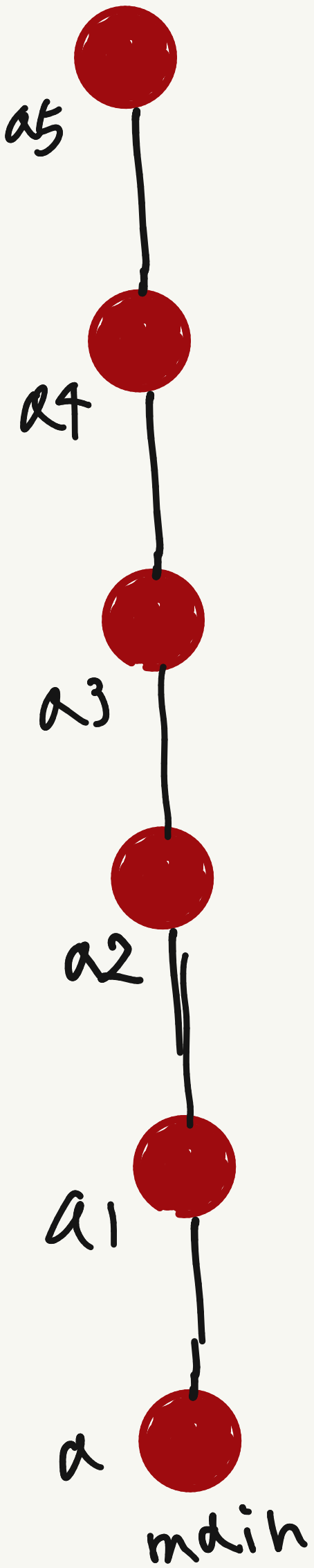
↑

コミットメッセージ

開発者に
何の変更があったか
知らせるため

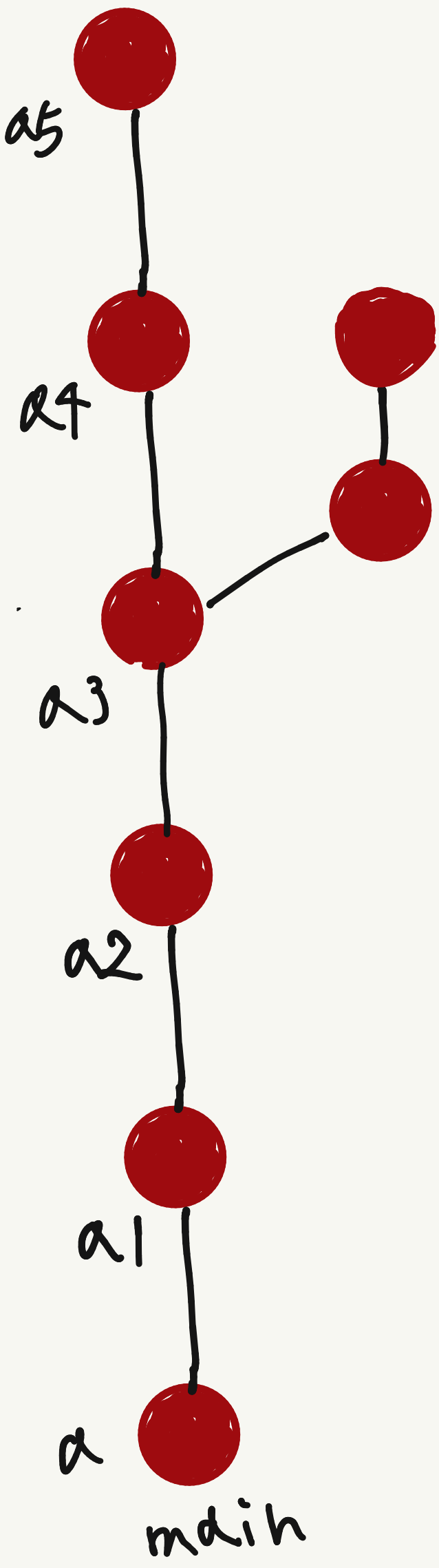
これ。
なにが嬉しいの？

このようにして履歴をセーブすることで
(コミット)



ある時点のセーブ時に戻れる

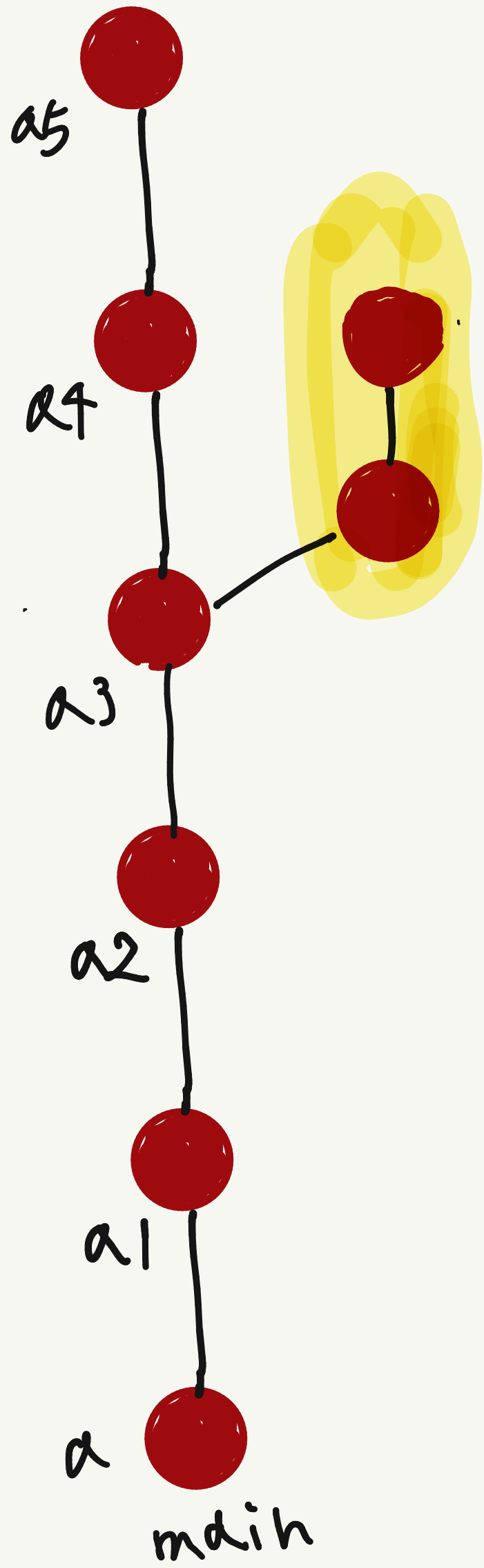
やり直して違う履歴をつみ上げる



← 今ココ

HEAD
といいます
(現在のブランチの先頭)

ブランチを作ってmainとは別の道を歩む



人生と一緒

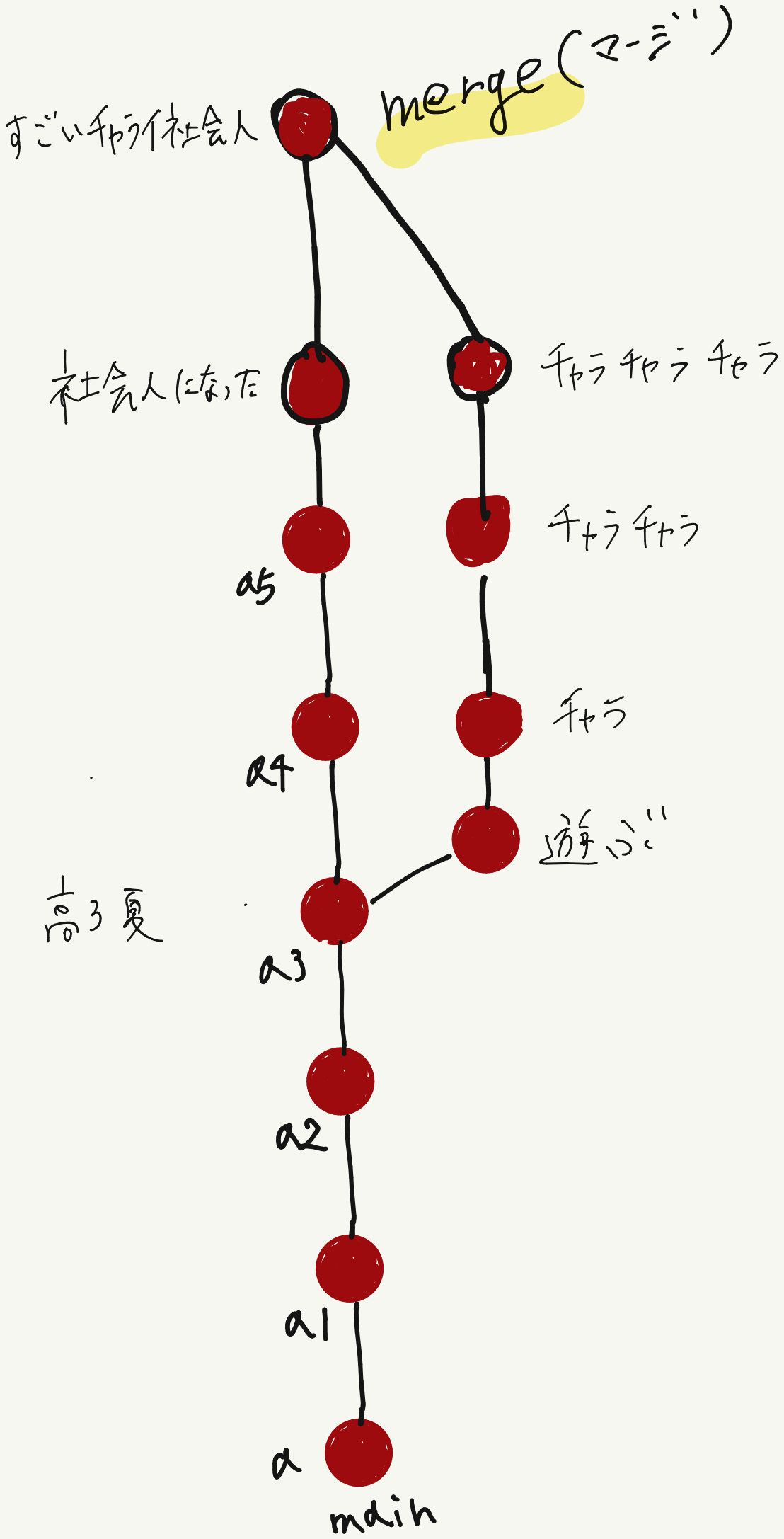
ですね

あの頃に戻って

違う選択をした

道を進んでいく

Gitは違う人生も今の人生の履歴に取り込む
ことができる

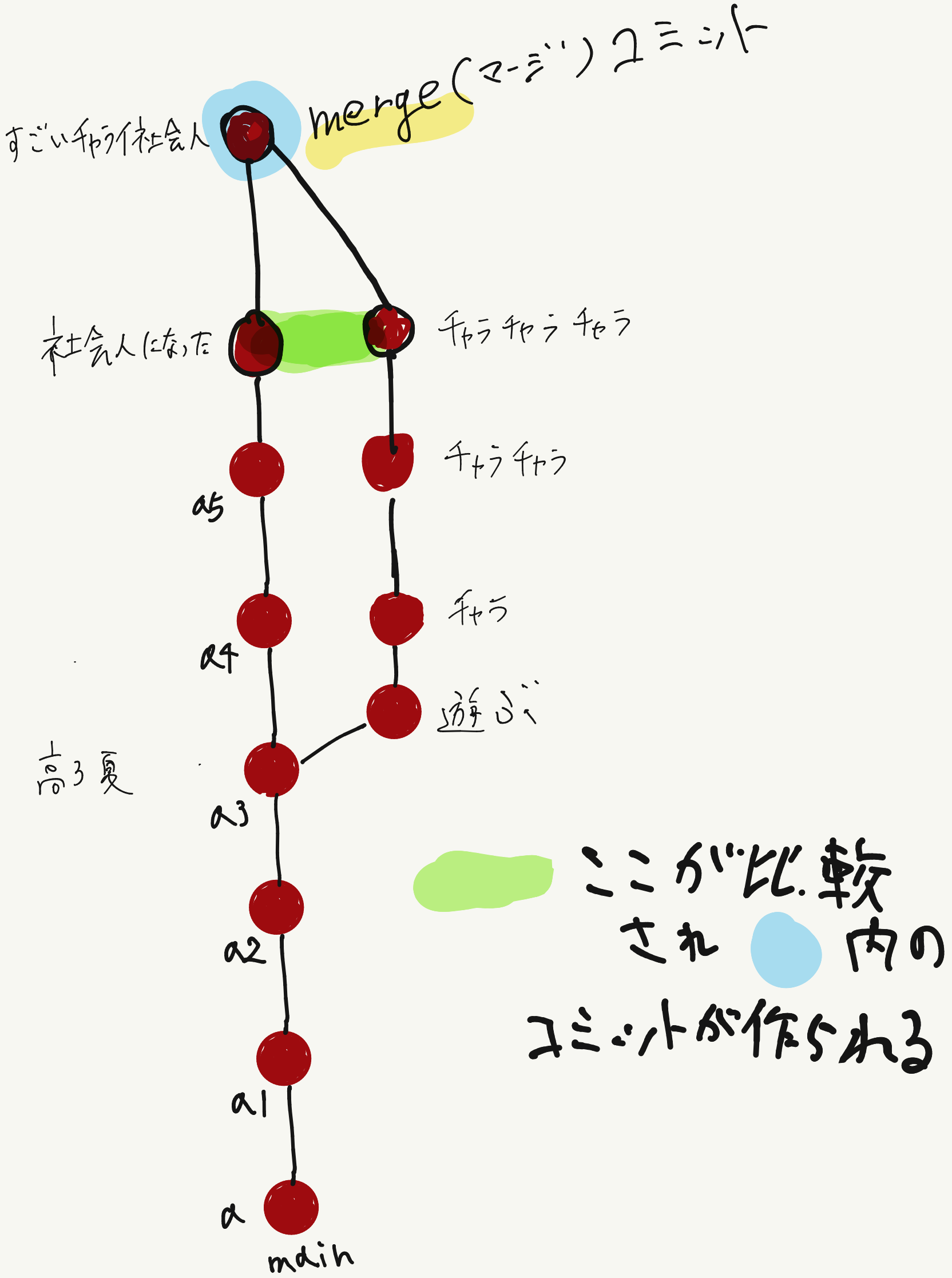


merge

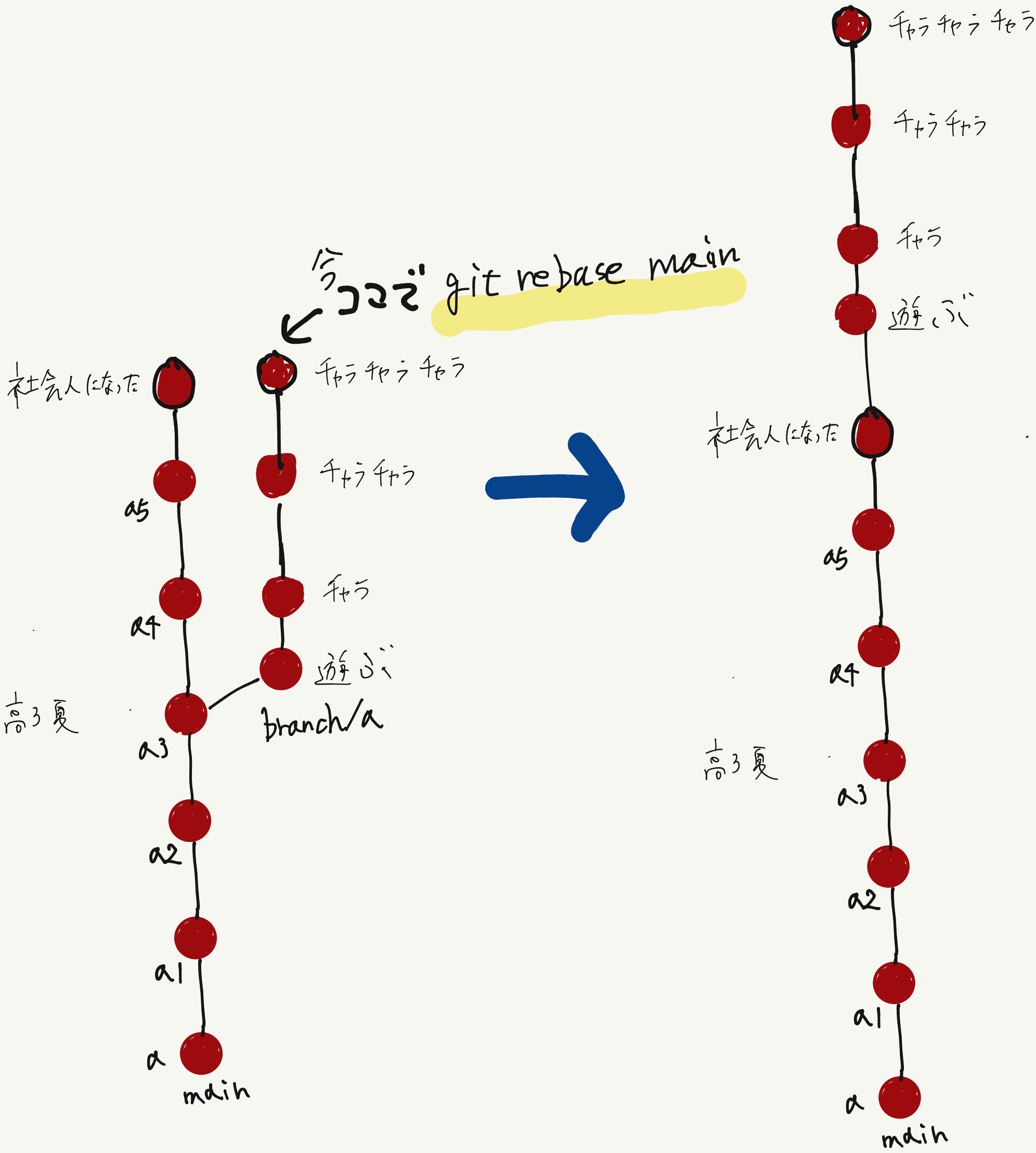
y

rebase

mergeは履歴を残す



rebaseは相手の枝の先頭に今の枝の履歴をつける

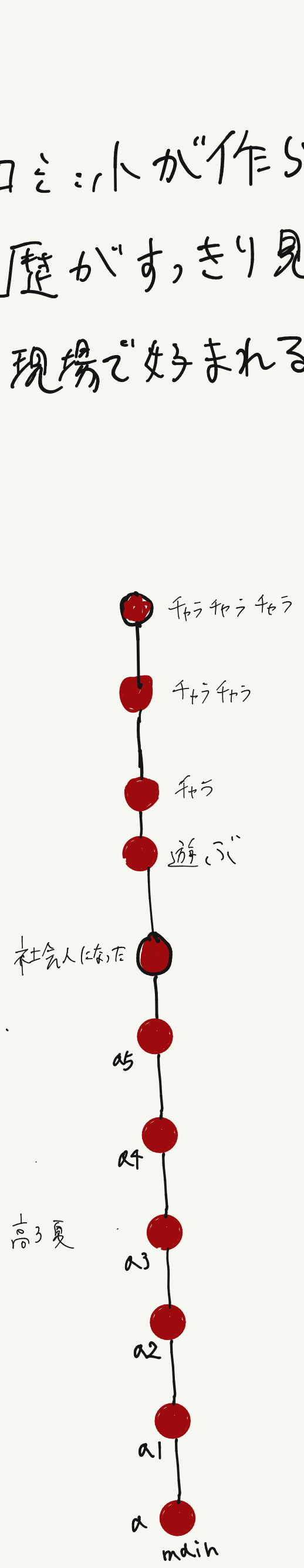


rebase

・マージコミットが作られない

→ 履歴がすっきり見える

→ よく現場で好まれる



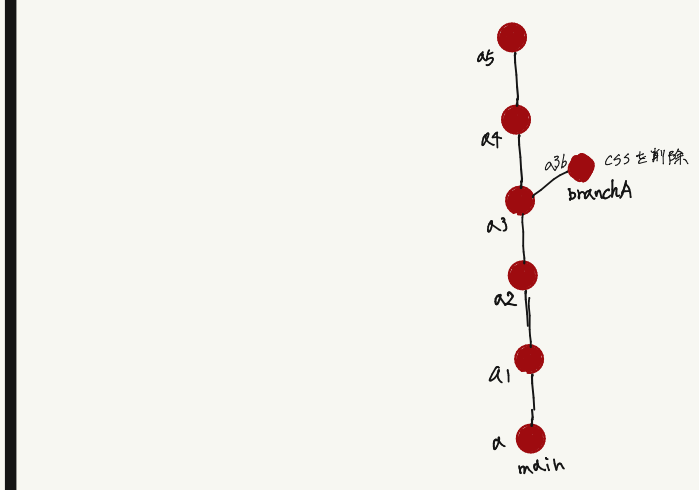
がらっと

話は変わって

リポジトリとは？

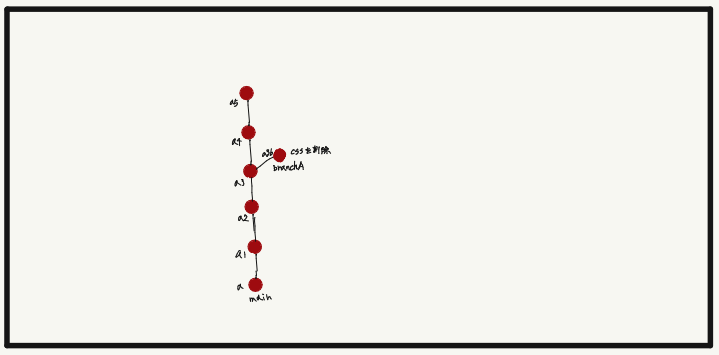
リポジトリは先程のような
ブランチの履歴をもっている箱（ざっくり）

リポジトリA



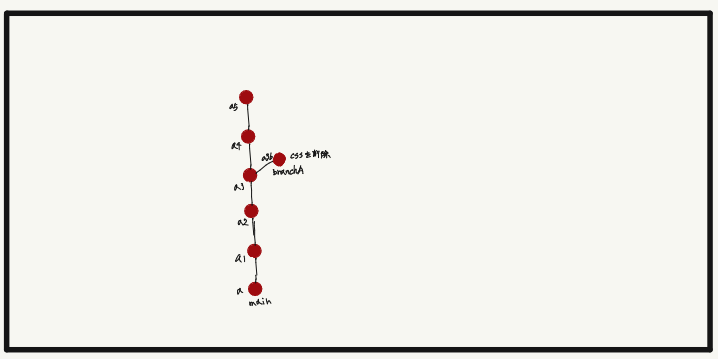
このリポジトリがweb上にある

＝リモートリポジトリ



自分のPC(ローカル)にある

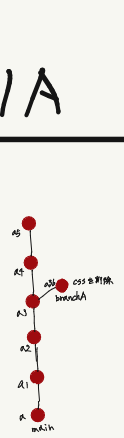
＝ローカルリポジトリ



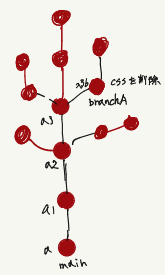
GitHubとは？

Web上でリポジトリを管理できるサービス

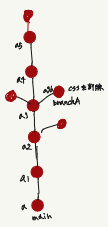
リポジトリA



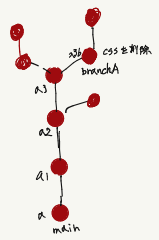
リポジトリB



リポジトリC



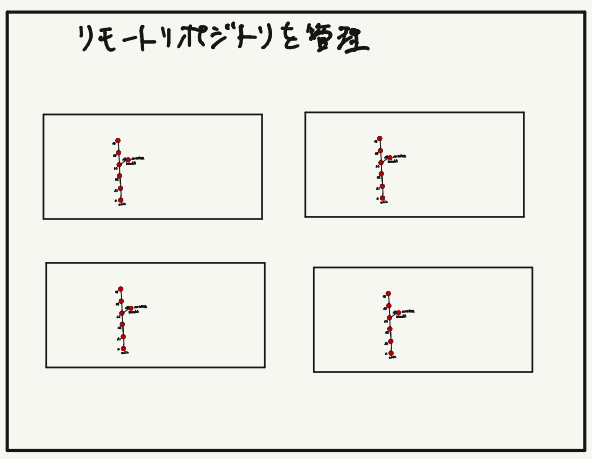
リポジトリD

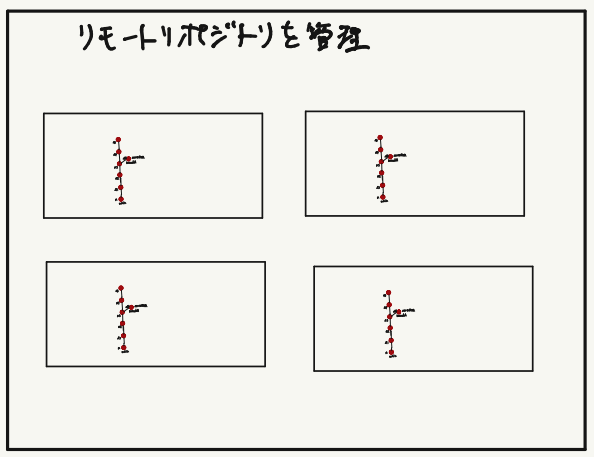


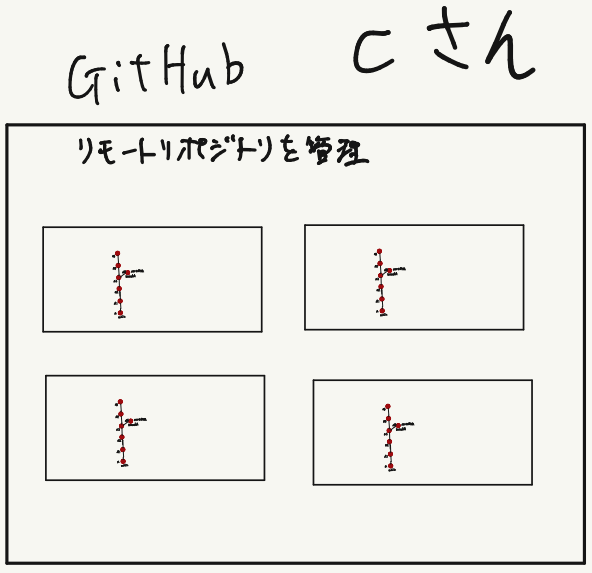
ちょっと見てみましょう

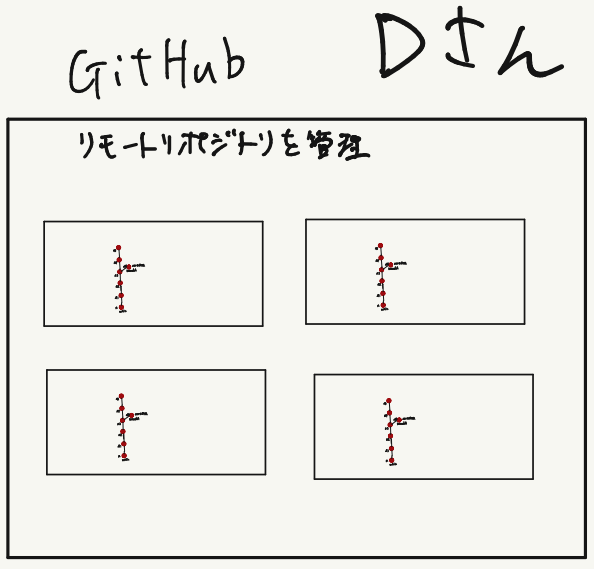
GitHub画面へ

それぞれが作っているものを自分のGitHub上で管理できる





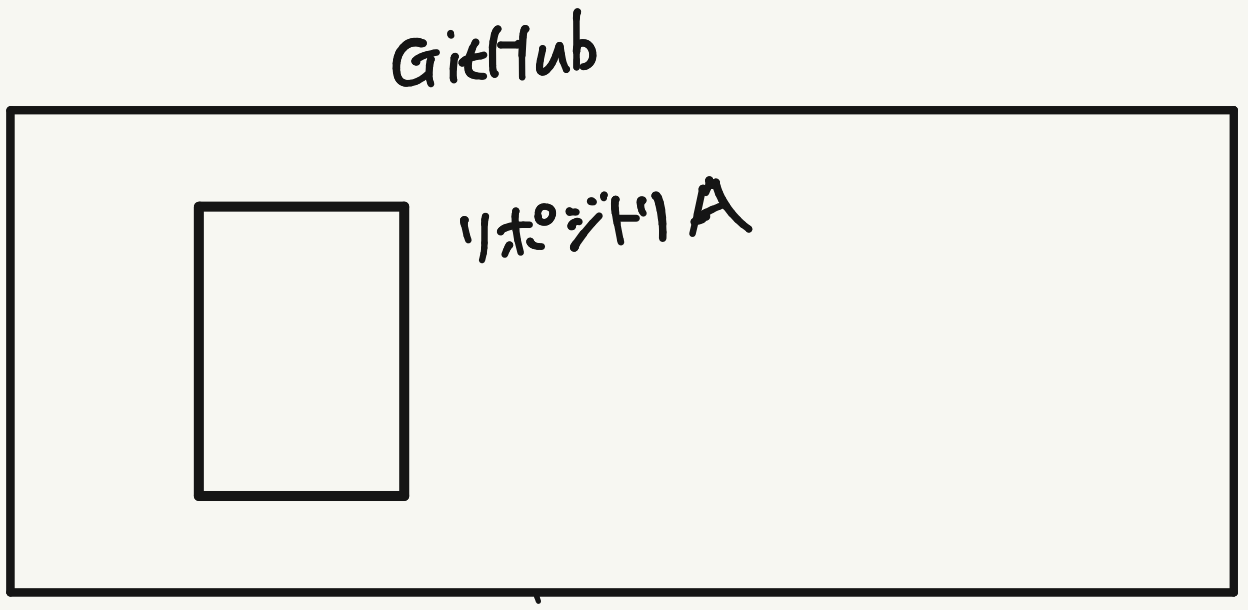




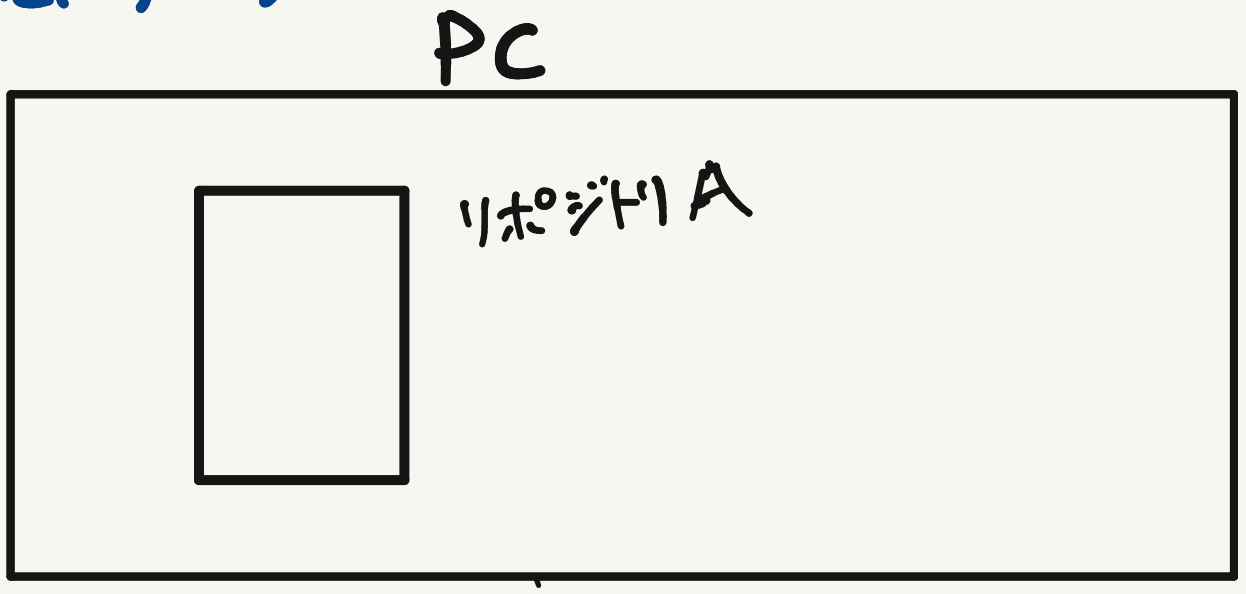
リポジトリを作る

リポジトリはどちらから作ってもいい

- リモートリポジトリから作る（web上）

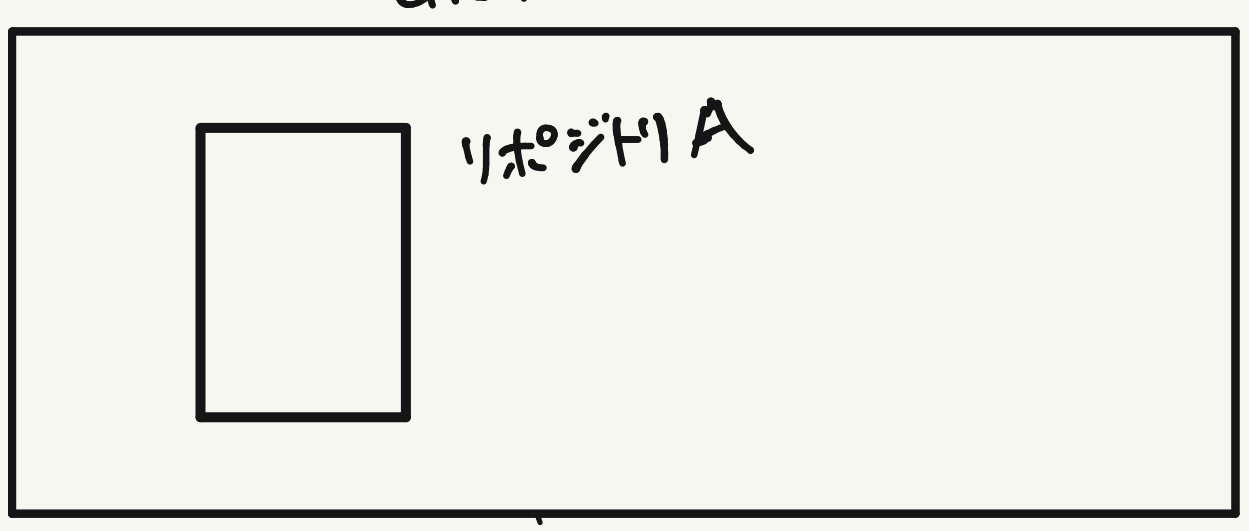


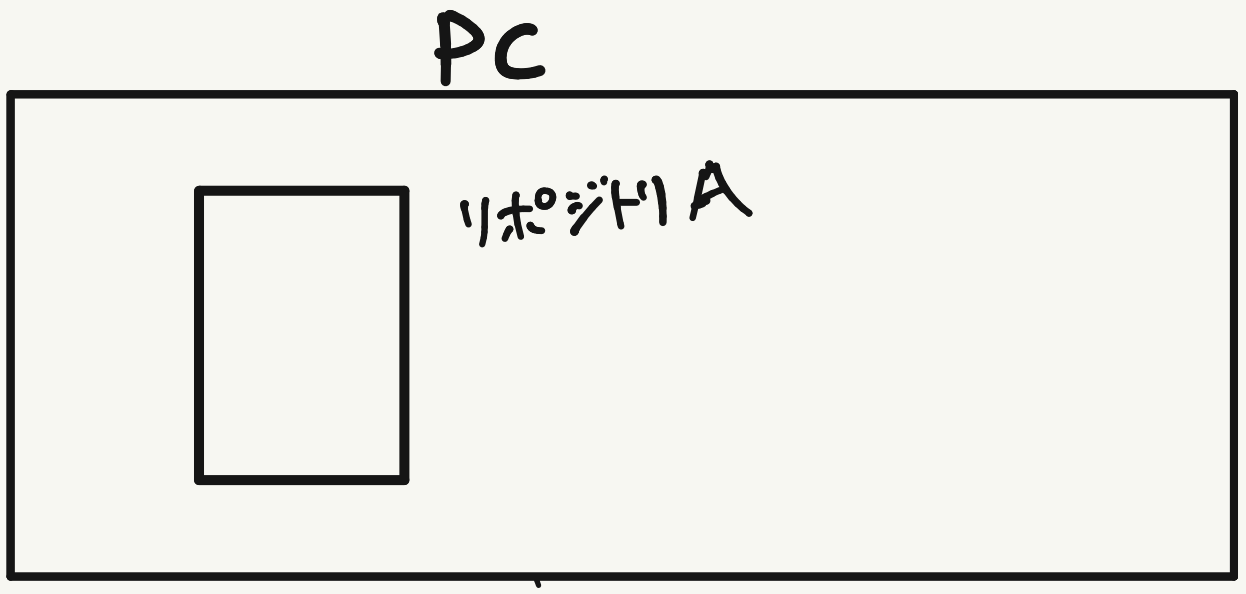
- 自分のローカル（PC）から作る



クローンとは？

リモートリポジトリを自分のPC上に複製すること





GitHub

A

クローン

D-カル(PC)

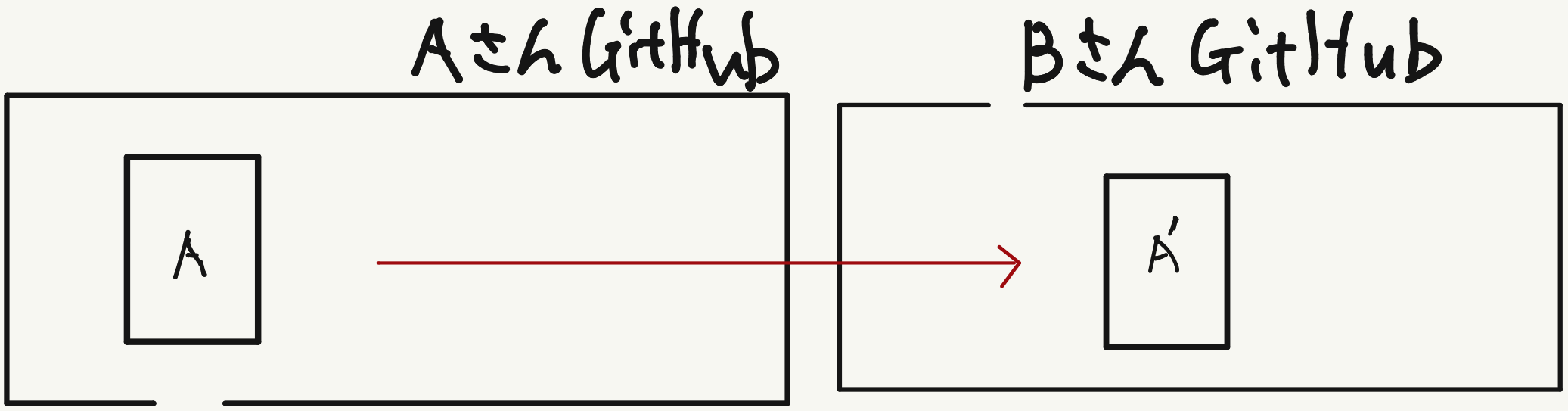
A'

(GitHub上のリポジトリをもってくる)

ちなみに

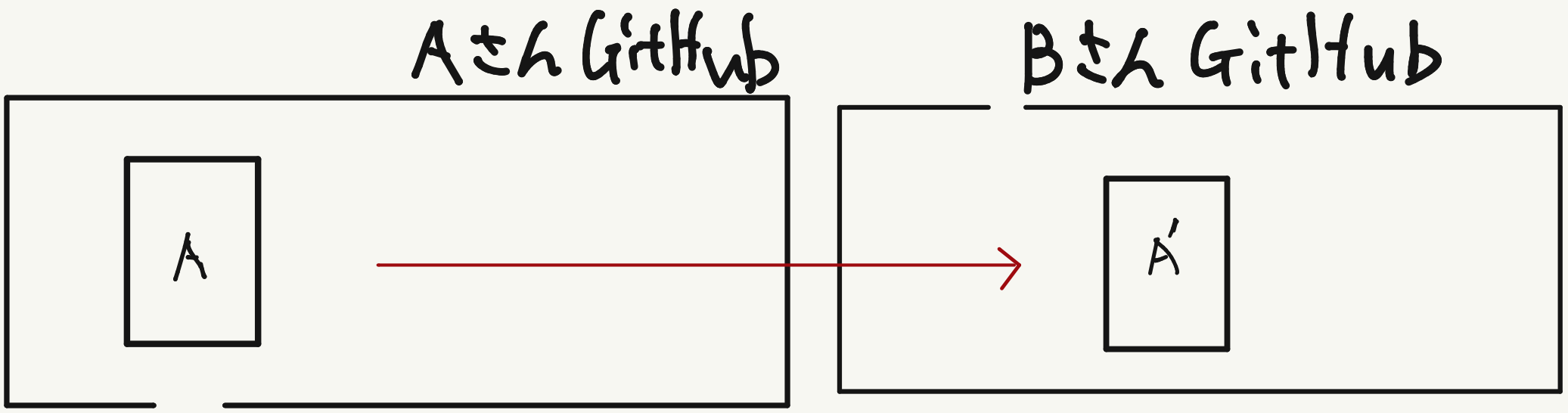
fork(フォーク)

とは



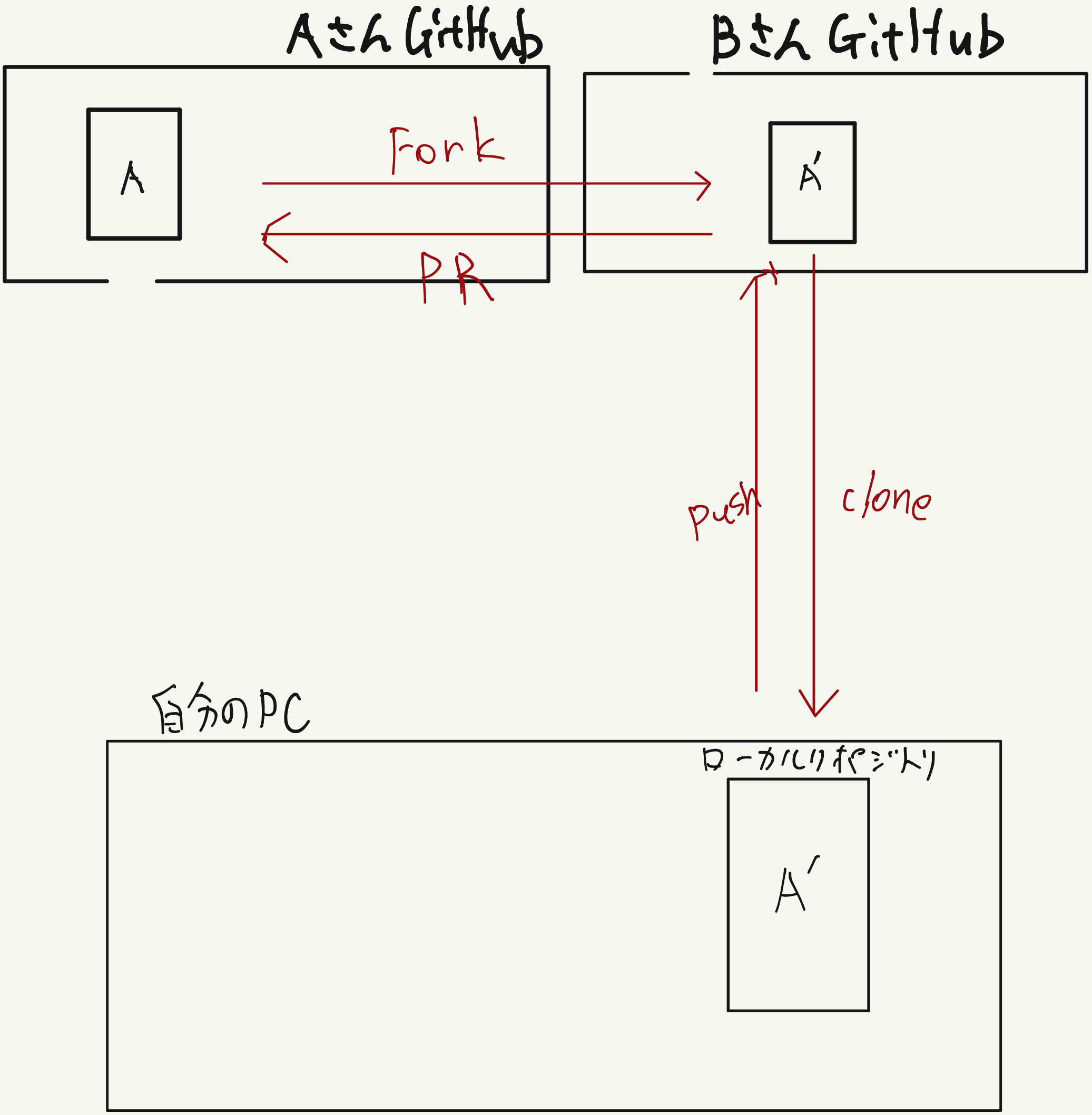
AさんのGitHub上にあるリポジトリを

自分のGitHub上に取り込むこと。



- Fork は Aさんのリポジトリに対して貢献することが前提
- 通知が行く
- このリポジトリのアイデアをつかって独自のものを作るとき
- 書き込み権現がないため fork する. cloneでもOK

Forkの流れ


Aさん Github
Bさん GitHub
A
Fork
A'
PR
push
clone
自分のPC
ローカルリポジトリ
A'

では

クローンすると

どうなるか

GitHub

A

ローカル(PC)

ローカルリポジトリ

A'

・リモート追跡ブランチ

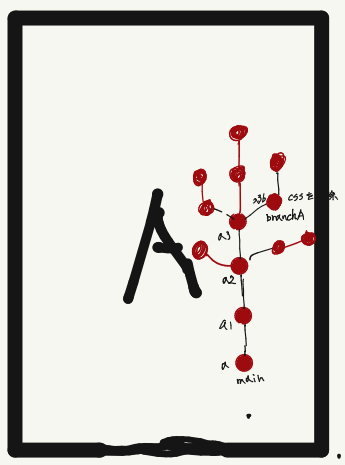
・ローカルブランチ

が作られる

・リモート追跡ブランチ

・ローカルブランチ

GitHub



watch

クローンすると作られる

ローカルブランチ

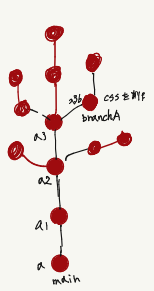
リモート追跡ブランチ

- リモートブランチの変更を watch している
- 参照のみ

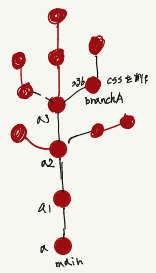
git branch -a

で remote/ と出てくるやつ

A'



ローカルブランチ



リモート追跡ブランチ

詳しい理解は後程

今は

ローカルブランチ

→コミットりれきを積んでいく所

リモート追跡ブランチ

→リモートリポジトリ内のブランチを参照、
しているもの

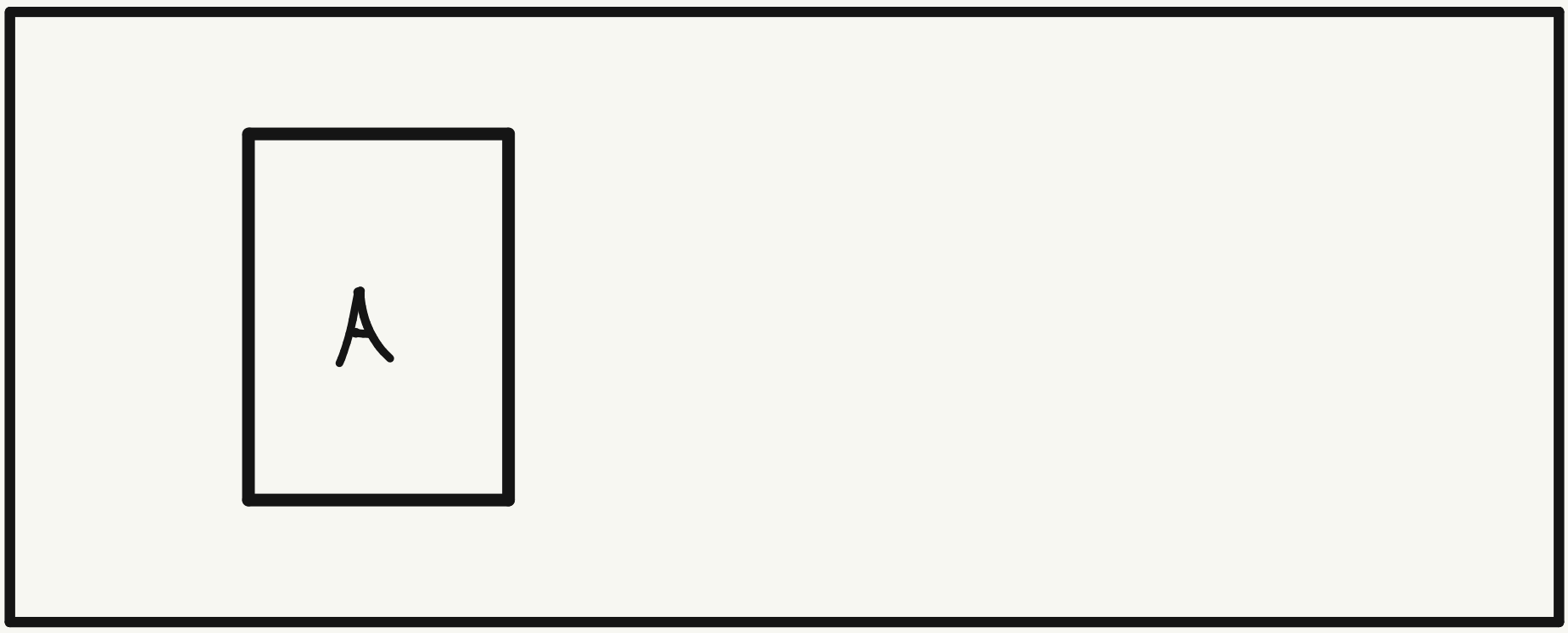
と覚えましょう

ではなぜ

クローンする必要が

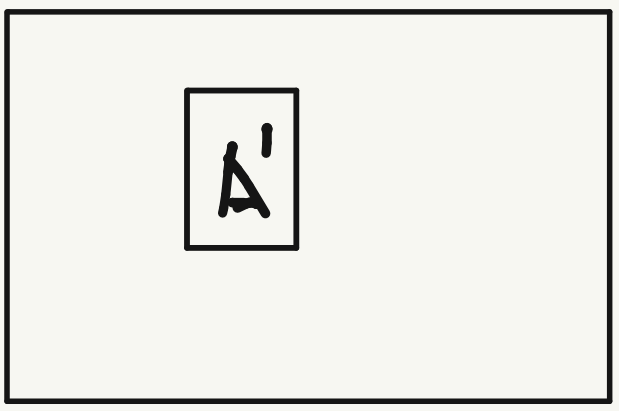
あるのでしょう

GitHub

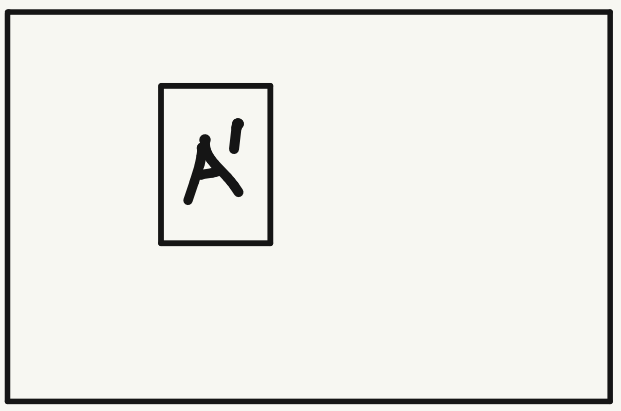


「もっとよくしたい！」
「貢献して名前を世に知らしめたい」

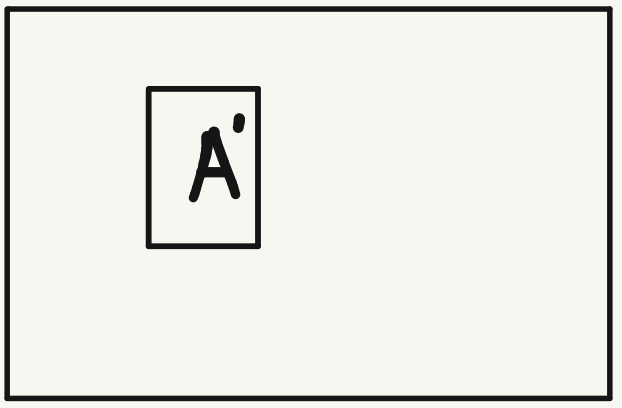
Aさん



Bさん



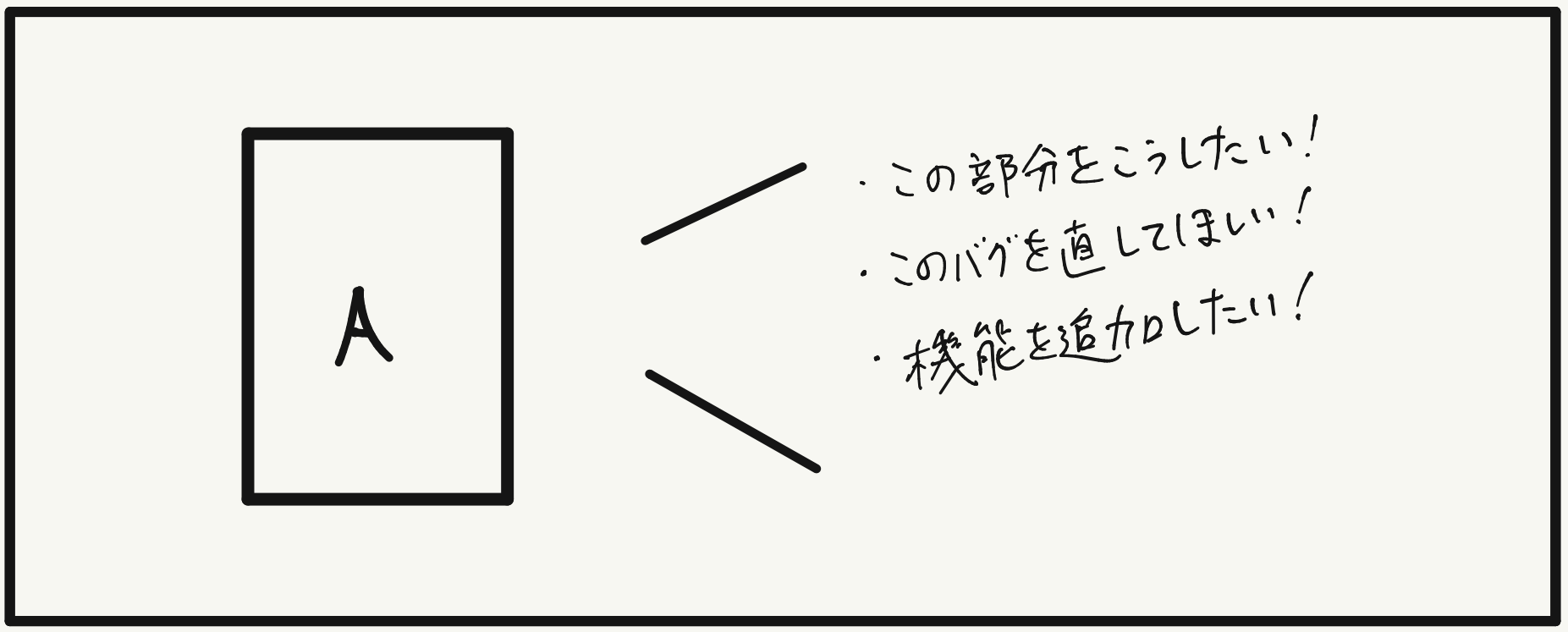
Cさん



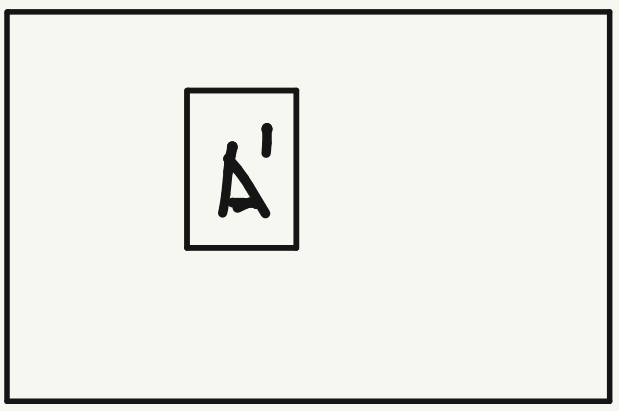
誰でもAのファイル群を
PC上で編集できる

たとえば

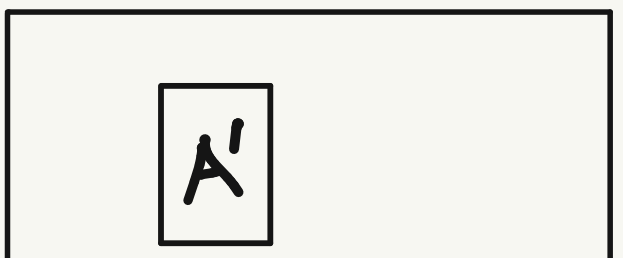
GitHub



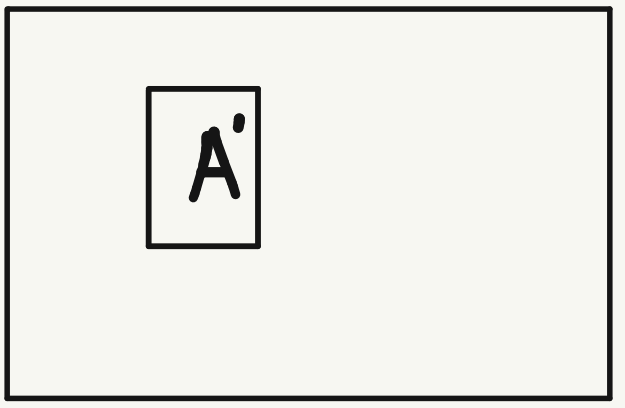
Aさん



Bさん



Cさん



issue

イシュー

GitHub

A

issue
・この部分をこうしたい！
・このバグを直してほしい！
・機能を追加したい！

直しました！ index.html
どうですか！これ！

Aさん

A'+

index.html
① 1行目を削除

Bさん

A'

Cさん

A'

Pull Request

プルリクエスト

（PR）

GitHub

PR

Aさんの PR
それれは!!
とりこんでくださぃ!!

直しました!

index.html
……
どうですか!これ!

Aさんの PR

Aさん

A'

Reviwe

レビュー

GitHub

A

PR

チェック
・このコードをAに入れてもよいか？
・バグはないコードか
・直っているか？
・変な書き方はないか

Bさん
（・ここをもうちょっと…）

Cさん
（・なんでこのような書き方なのですか？！）

A'

merge

マージ

（変更をとりこむ）

GitHub

A+

PR

okなら
オーナーがPRを
取り込む（merge）

A'+

この時の状態

GitHub

A+

PR

古い状態のまま・

Aさん

A'+

Bさん

A'

Cさん

A'

pull

(変更を取り込む

ざっくり言うと

・※正しくはfetch + merge

GitHub

A+

PR

変更を
取込んで
最新にして下さい!!

変更されましたよ!!

Aさん

A'+

Bさん

A'

Cさん

A'

GitHub

A+

PR

変更を
取込んで
最新にして下さい!!

変更されましたよ!!

Aさん

Bさん わかりました!

Cさん OK!!

A'+

A'

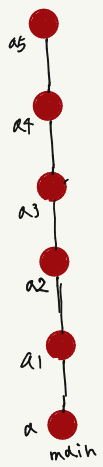
git pull

A'

git pull

Bさん

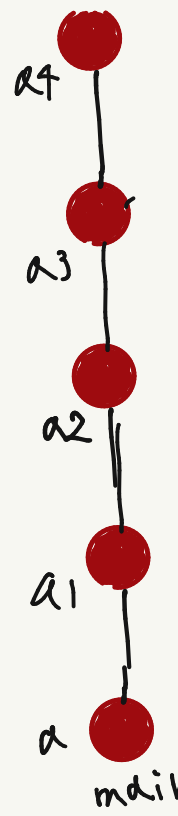
がする事



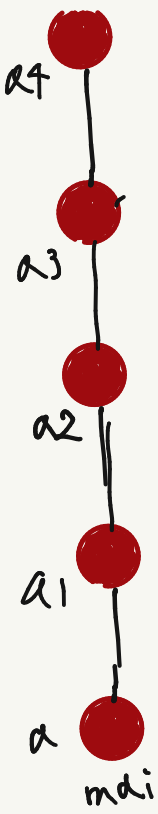
6つのコミット

変更ありますよ!!

Bさんのローカルリポジトリ A'



ローカルブランチ

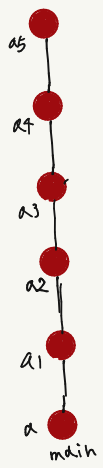


(5つのコミット)
リモート追跡ブランチ

リモート追跡ブランチに遅れがあるのでそれを取り込む

pull

(git fetch + git merge



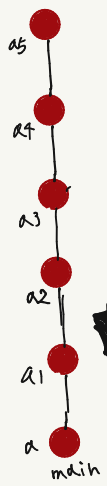
6コのコミット

変更ありますよ!!

BさんのローカルリポジトリA'

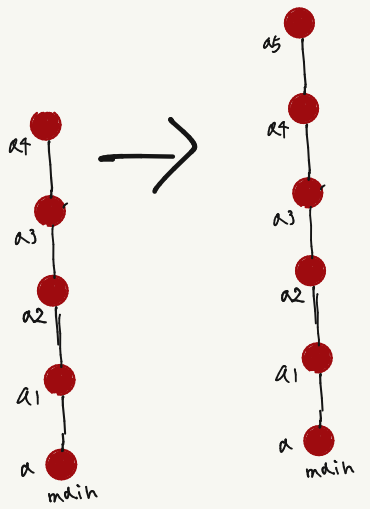
git pull すると…

ローカルブランチ



git merge

リモート追跡ブランチ



6コのコミット
に更新される

git fetch

現在の
できた追跡ブランチをローカルブランチにmerge.

リモートブランチと同じ状態になる

GitHub

A+

PR

issue

・この部分をこうしたい！

・このバグを直してほしい！

・機能を追加したい！

次は
これ!!

私やります！

Aさん

A'+

Bさん

A'+
PR

Cさん

A'+
PR

じゃあ

一人の場合は？

GitHub

自分が作ったリポジトリ

A

① これもっとこうしたい（自分）

PR

③（自分に対してチェックを要求）

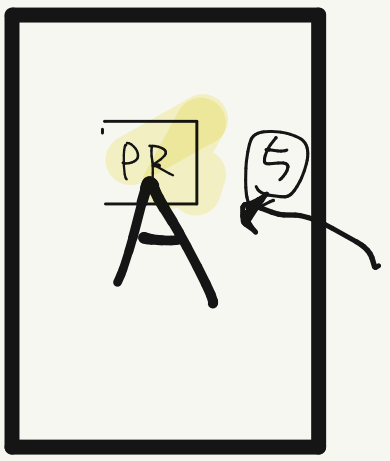
ローカル(PC)

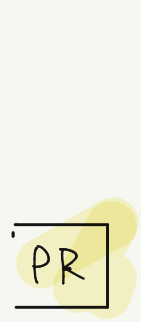
② コードを変更（自分）

A'

GitHub

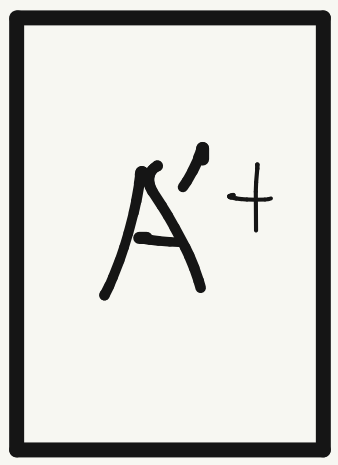
自分が作ったリポジトリ





4
自分がチェック、
問題なければマージ

ローカル(PC)



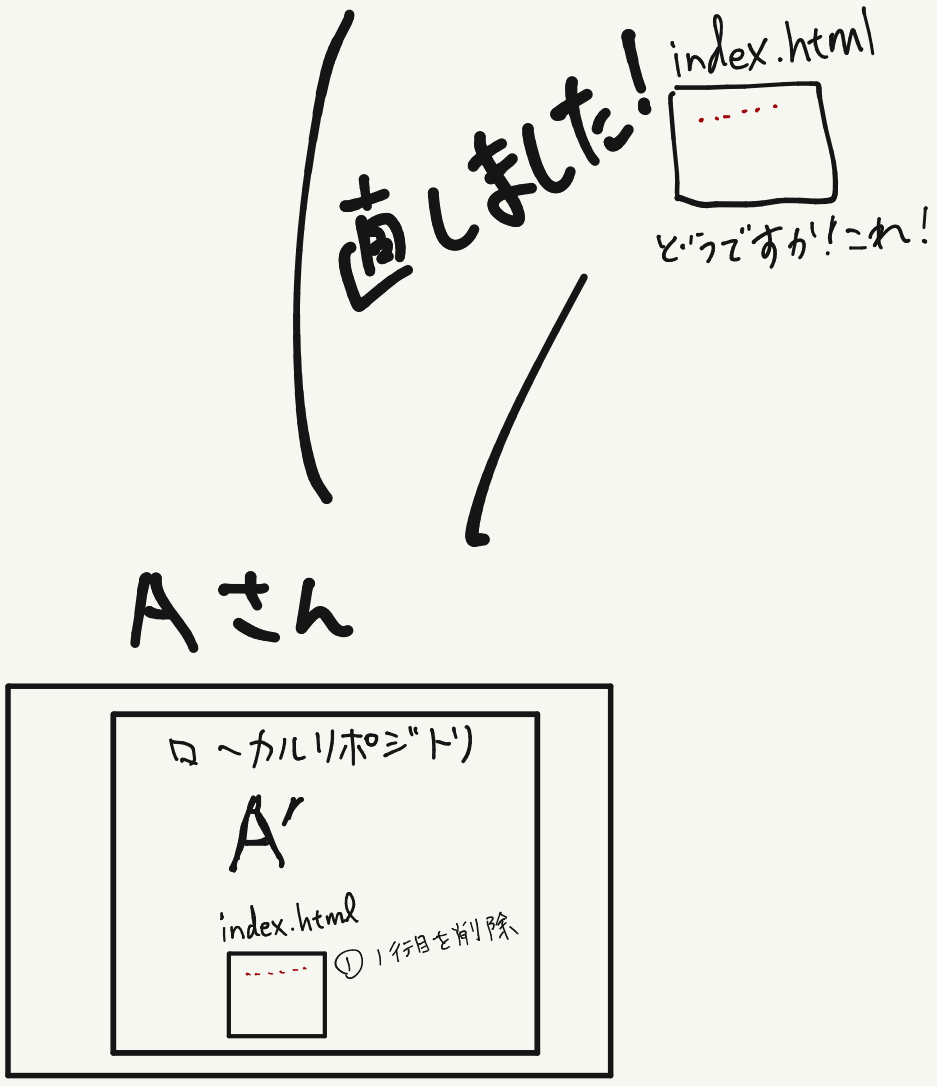
pullの必要はない

リモート追跡ブランチが更新されて
リモートブランチが更新されたから

pullRequest

を送るまで

具体的にどうやれば
いいのか



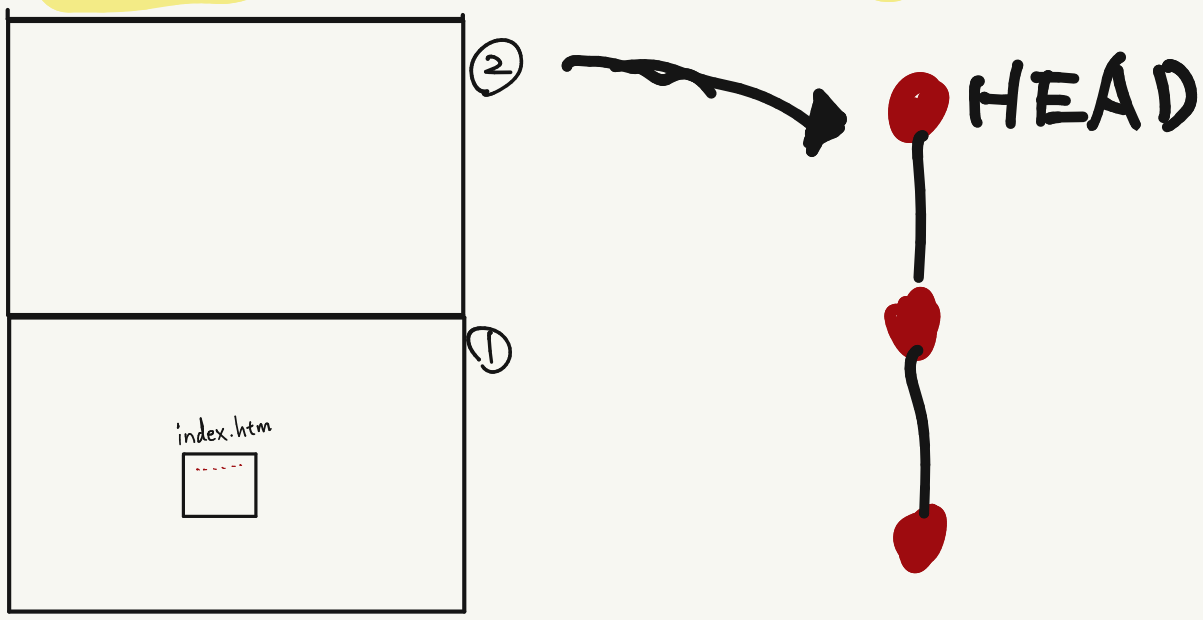
履歴を更新する

ローカルリポジトリA'

ローカルブランチ

HEADの状態

ローカルブランチの参照



HEADの状態（今）

① ワーキングツリー

変更中の場所
作業中 ：

② インデックス

- 最終チェックする場所
- 選別する所

① ワーキングツリー

変更中の場所
作業中 ：

例えるなら

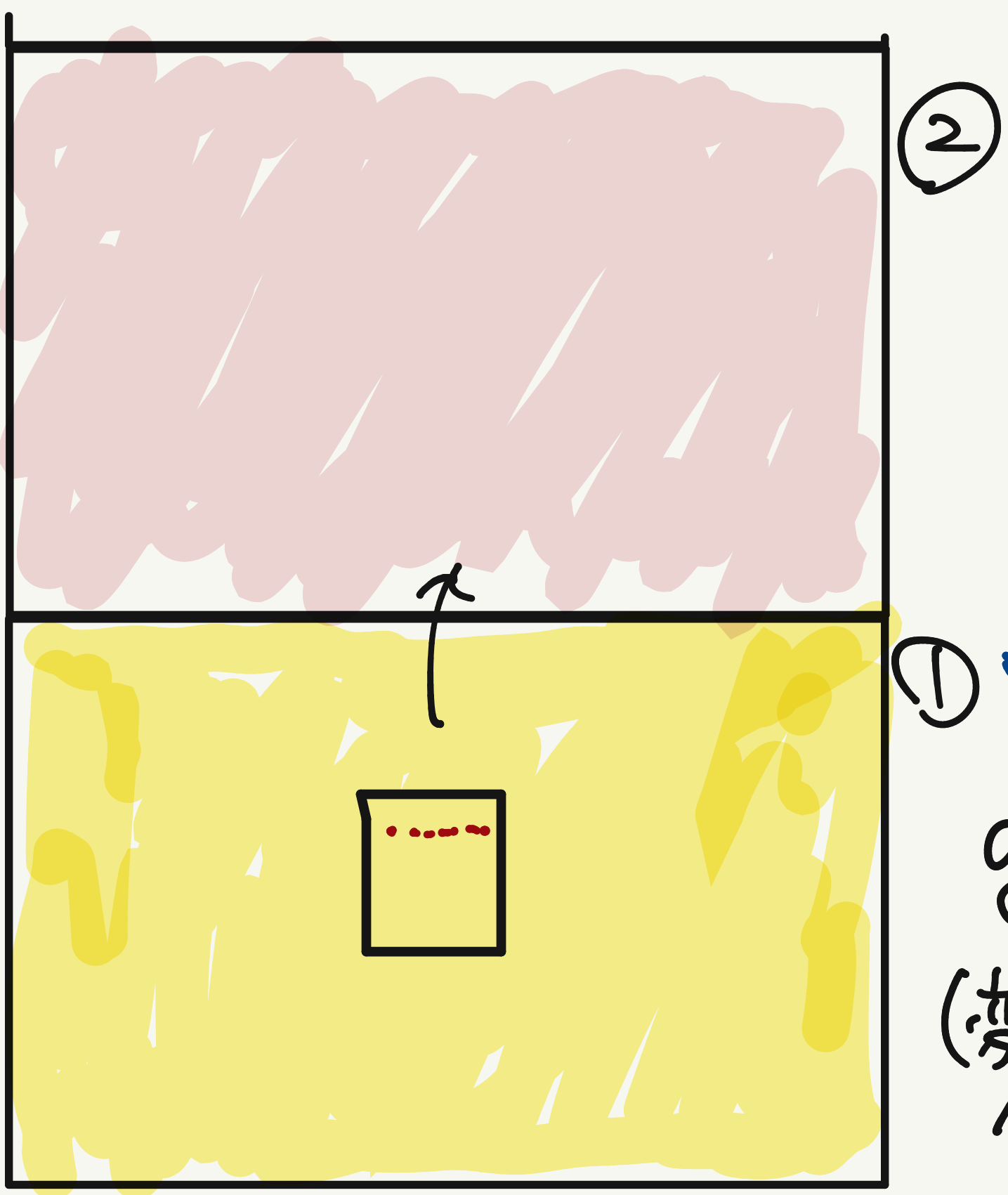
俳優

② インデックス

舞台・現場で確認（記録する前）

① ワーキングツリー

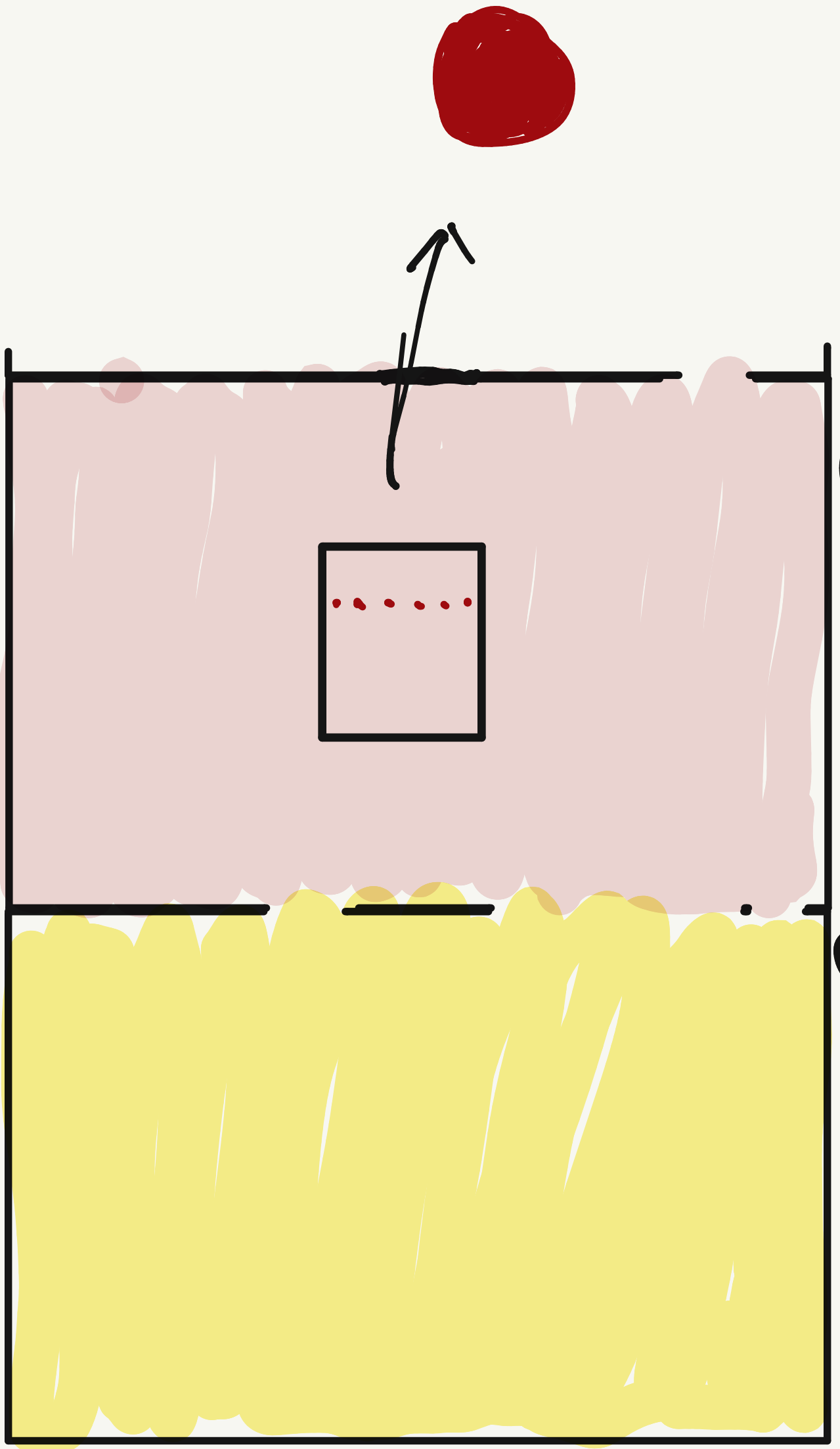
準備・自宅・練習



①ワーキングツリー

git add

(変更を最終チェック
へ↑舞台上へ)



② インデックス

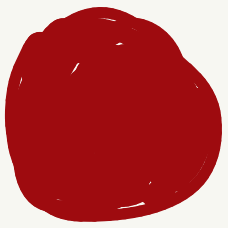
okだと思ったら
記録する

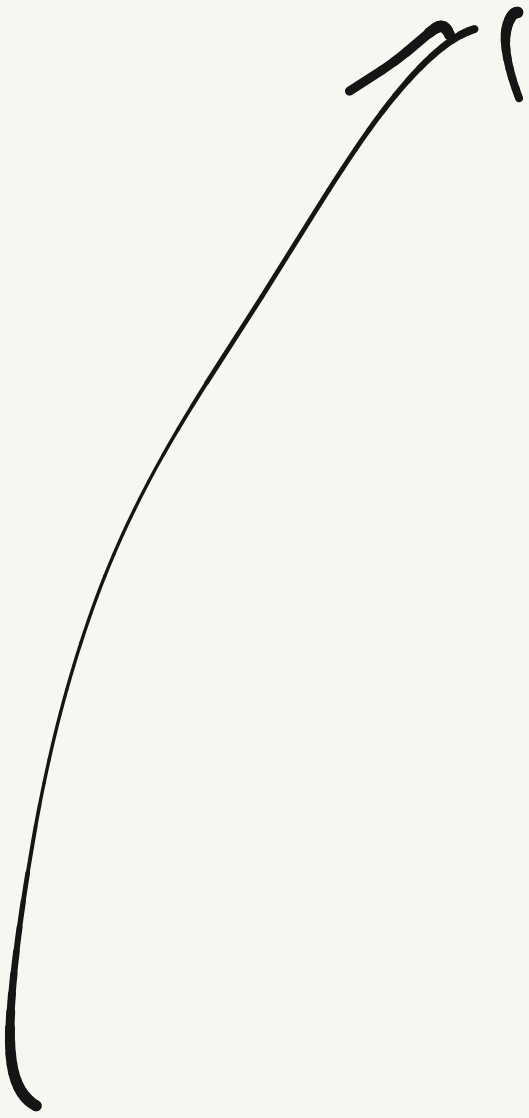
git commit -m
[message]

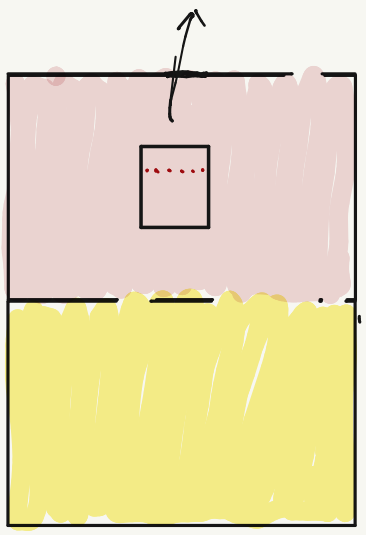
① ワーキングツリー

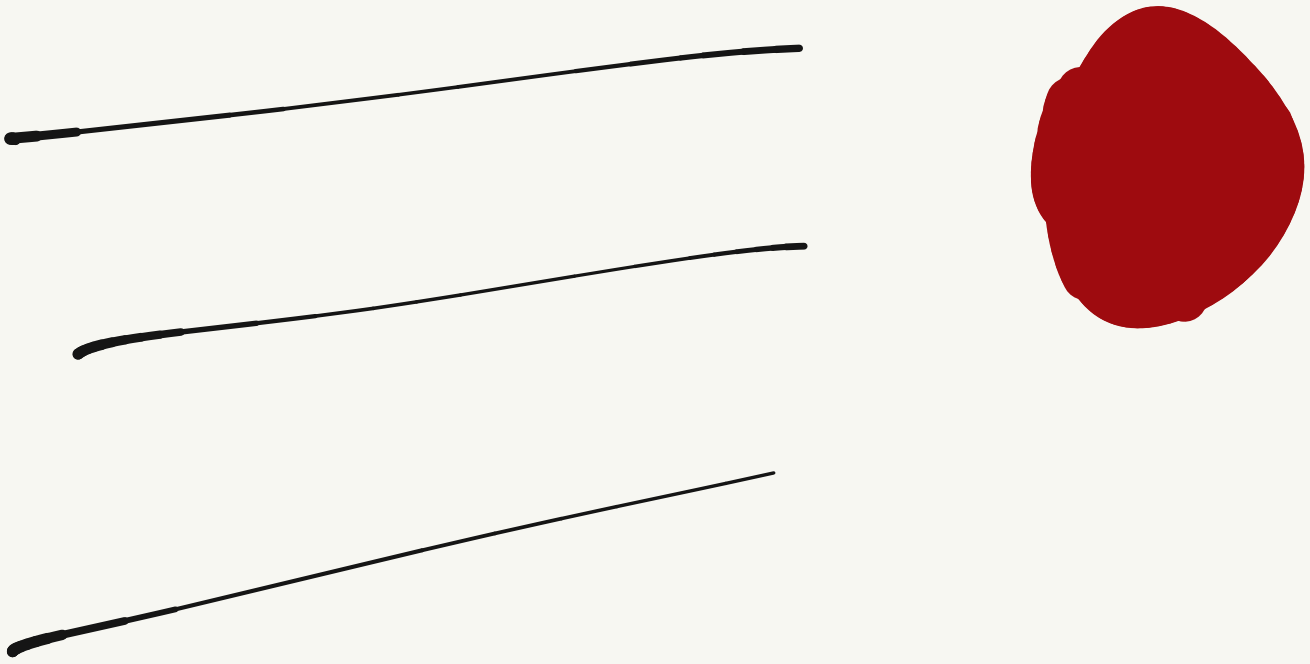
git add

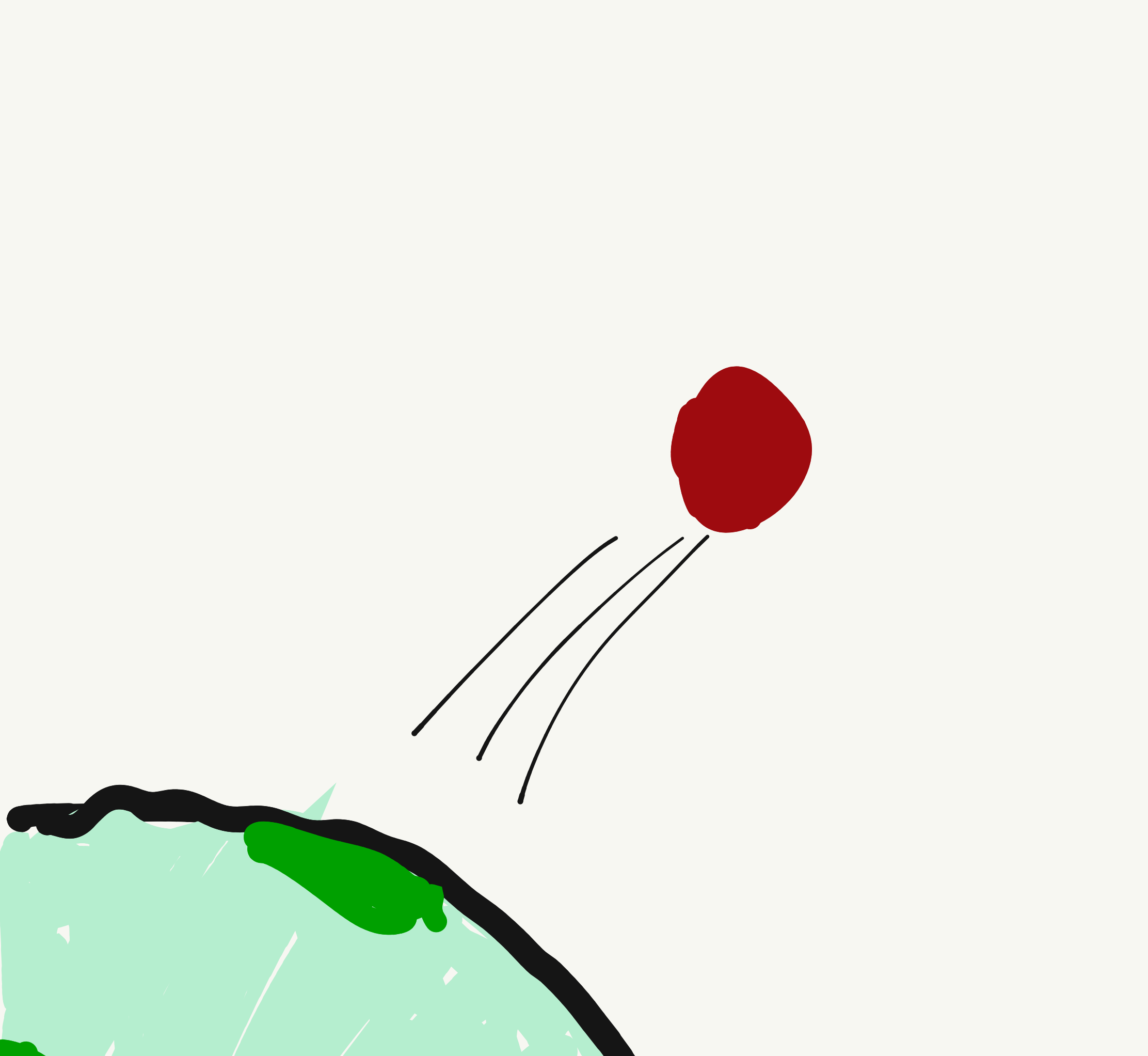
(変更を最終チェック
へ)舞台上へ

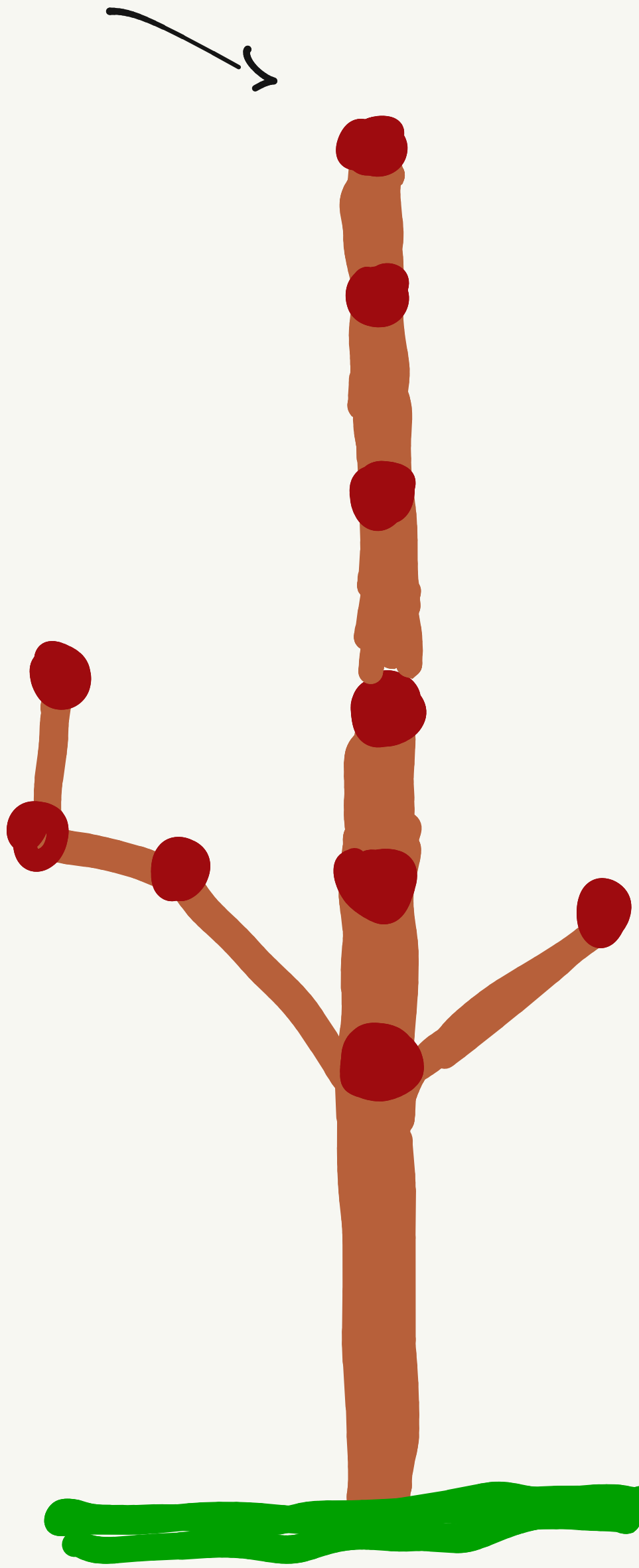




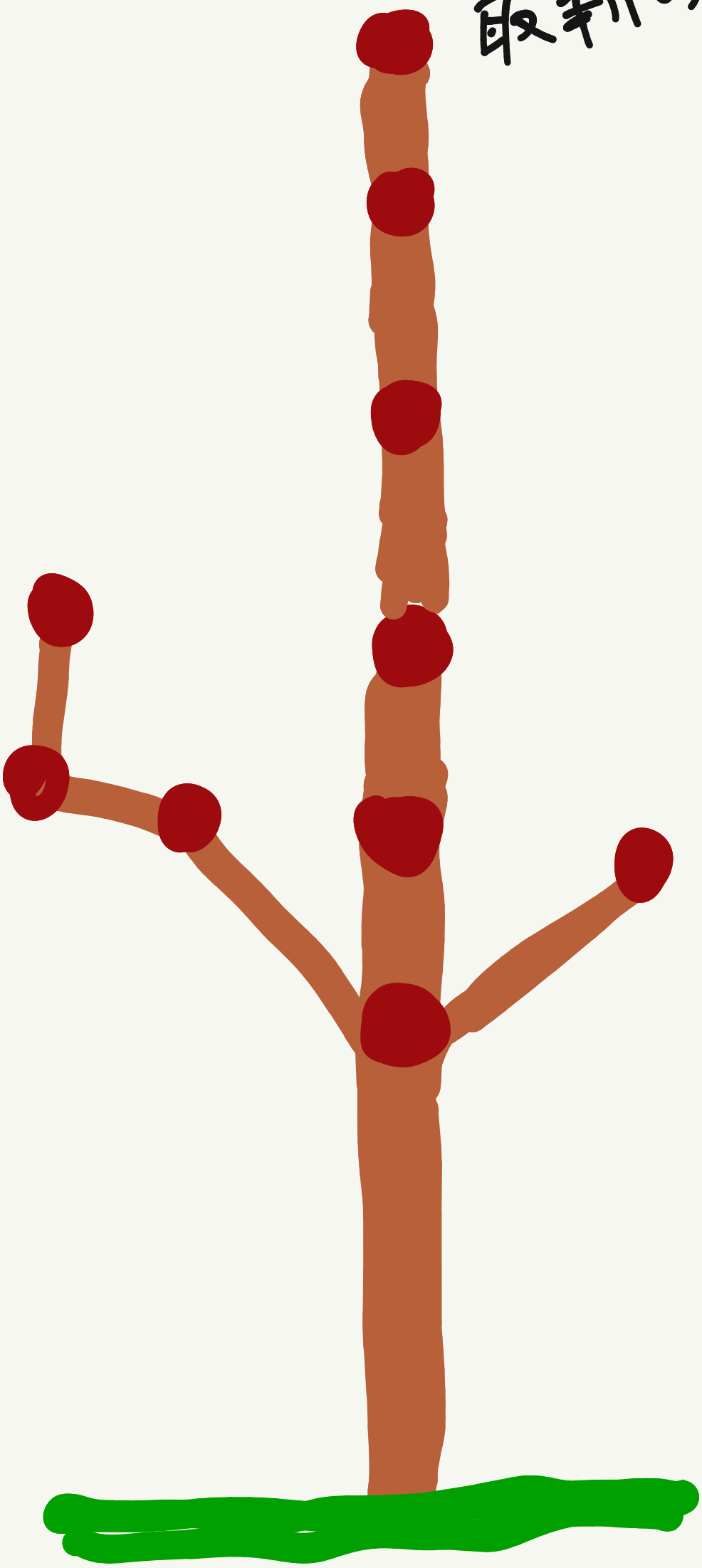








最新のコミットになった



Github

リモートリポジトリ

A'

リモート
ブランチ

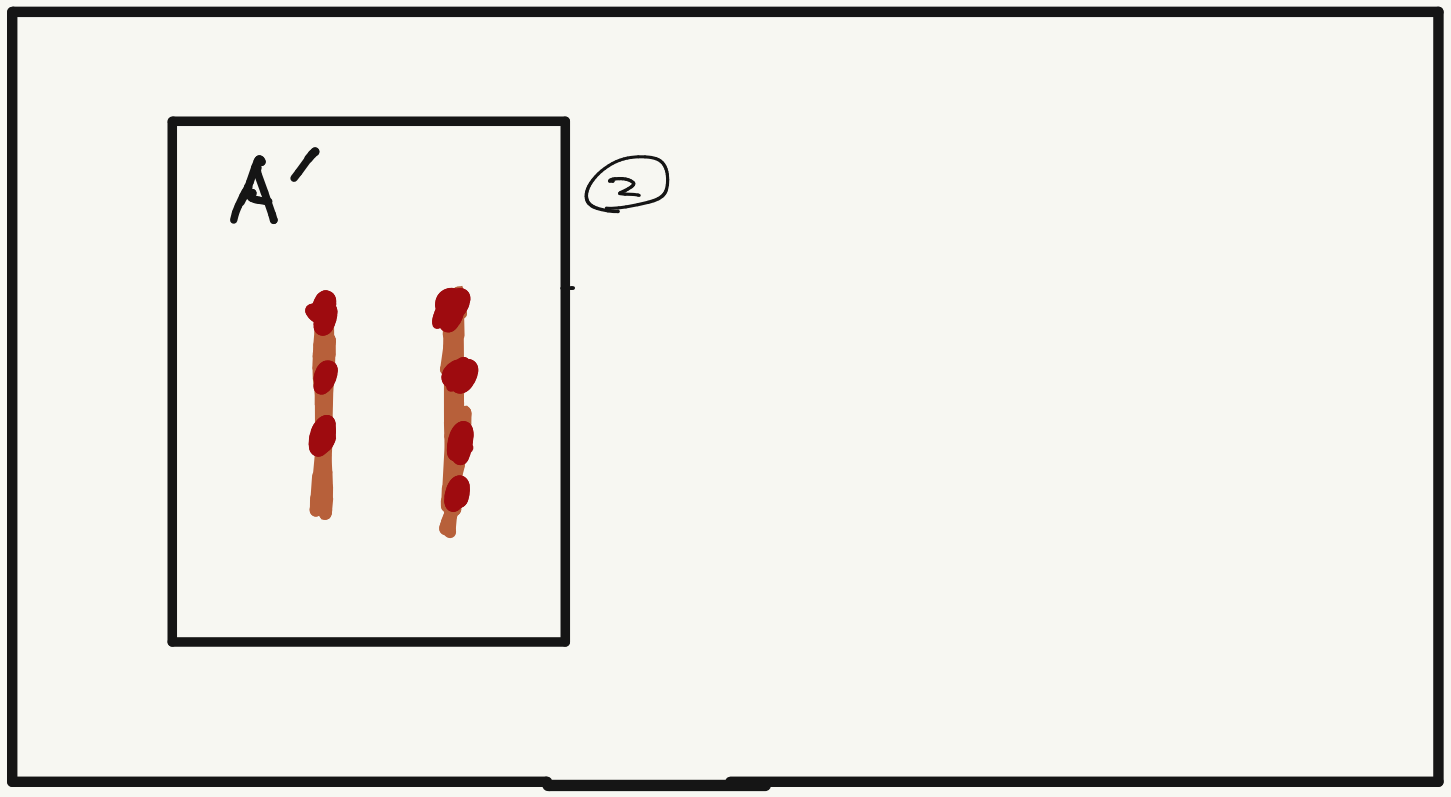
Github内で
PRを出す

ローカルリポジトリ

A'++

push

丸ごと



誰もがこの更新を見れる状態に
なった。
（撮影されたものが配信された）

GitHub

A

issue
・この部分をこうしたい！
・このバグを直してほしい！
・機能を追加したい！

GitHub内でPRを出しレビューされる

直しました！

git push

どうですか！これ！

お!!! 変更された

お!! Aさんが更新した！

Aさん

ローカルリポジトリ

A'

Bさん

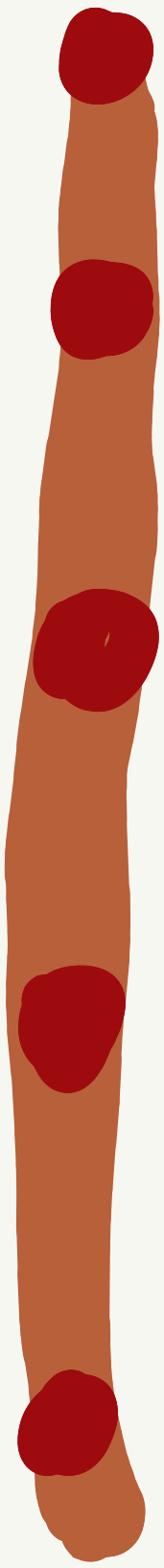
A'

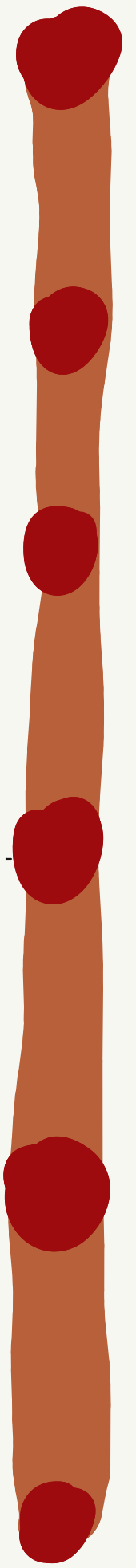
Cさん

A'

Pushした後・・・

どこのリモートブランチ
に
マージを求めるか
を決める





master vs リモートブランチ

HEAD

diff

リモートリポジトリ内にある

ファイル・フォルダ

すべてのリソース

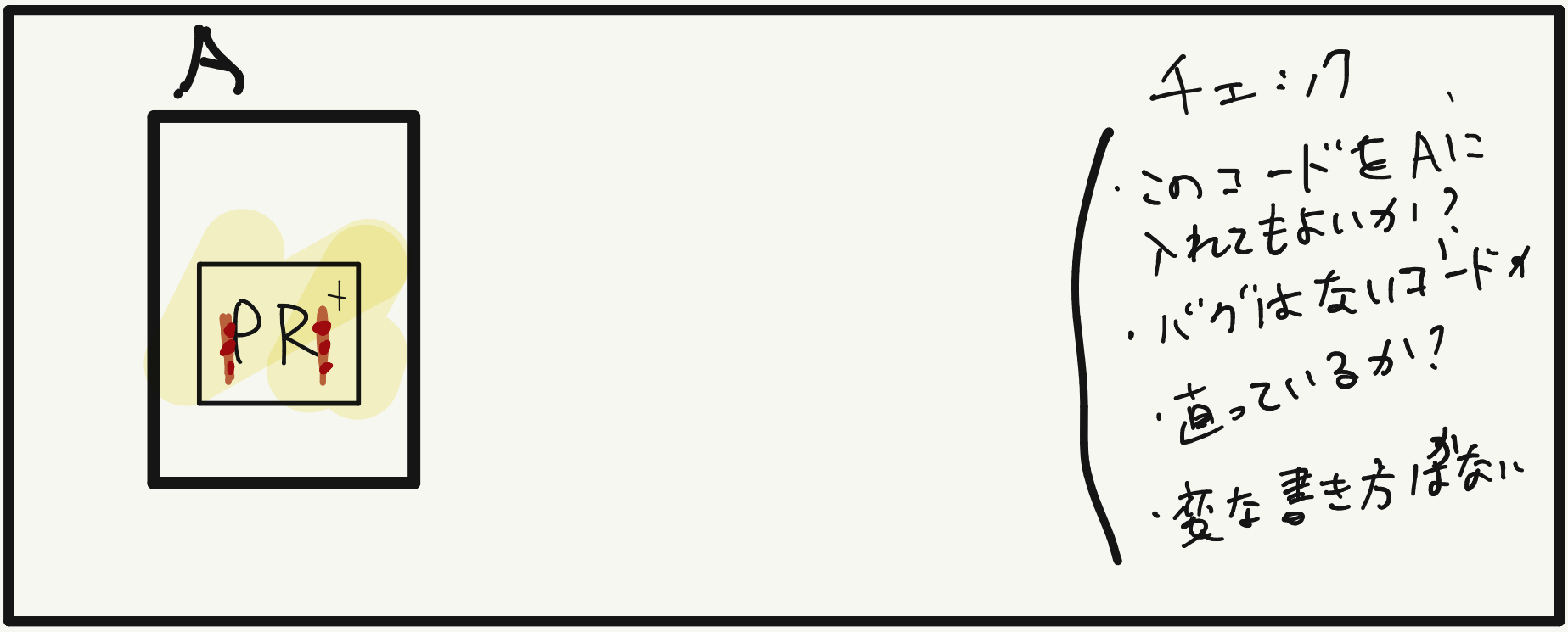
の差分が

検出される

master

リモート
pushしたブランチ

GitHub



GitHub

A+

PR

変更を
取込んで
最新にして下さい!!

変更されましたよ!!

Aさん

Bさん わかりました!

Cさん OK!!

A'+

A'

git pull

A'

git pull

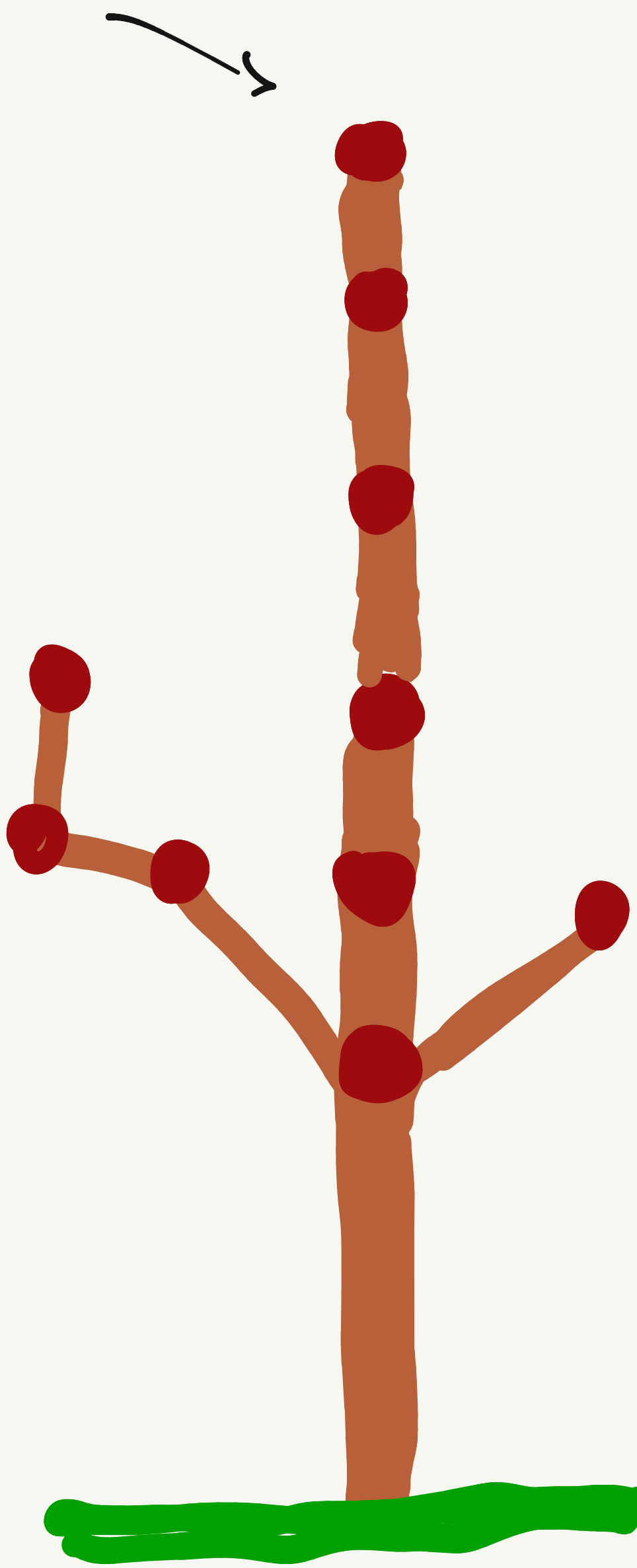
これのくり返し

まとめ

このようにローカルリポジトリの
コミット履歴の連続(branch)
をリモートリポジトリにのブランチに反映させ

GitHub上でPRを出し、
レビューをうけ、リモートリポジトリを更新
他の開発者がpullで変更を
自分のPC上にあるローカルリポジトリ
に受けとり、
ブランチを更新して開発を進めて
いきます

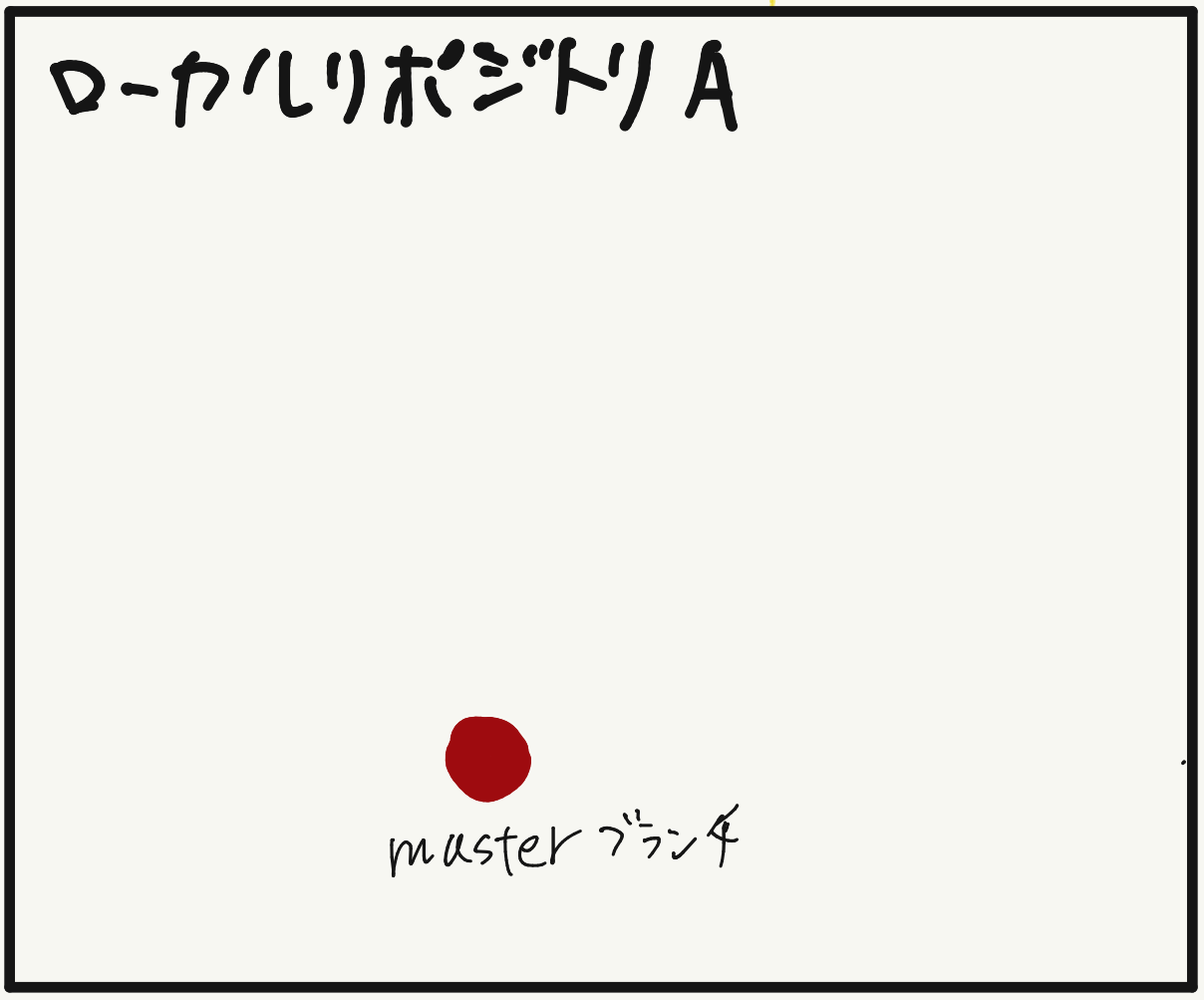
おしまい



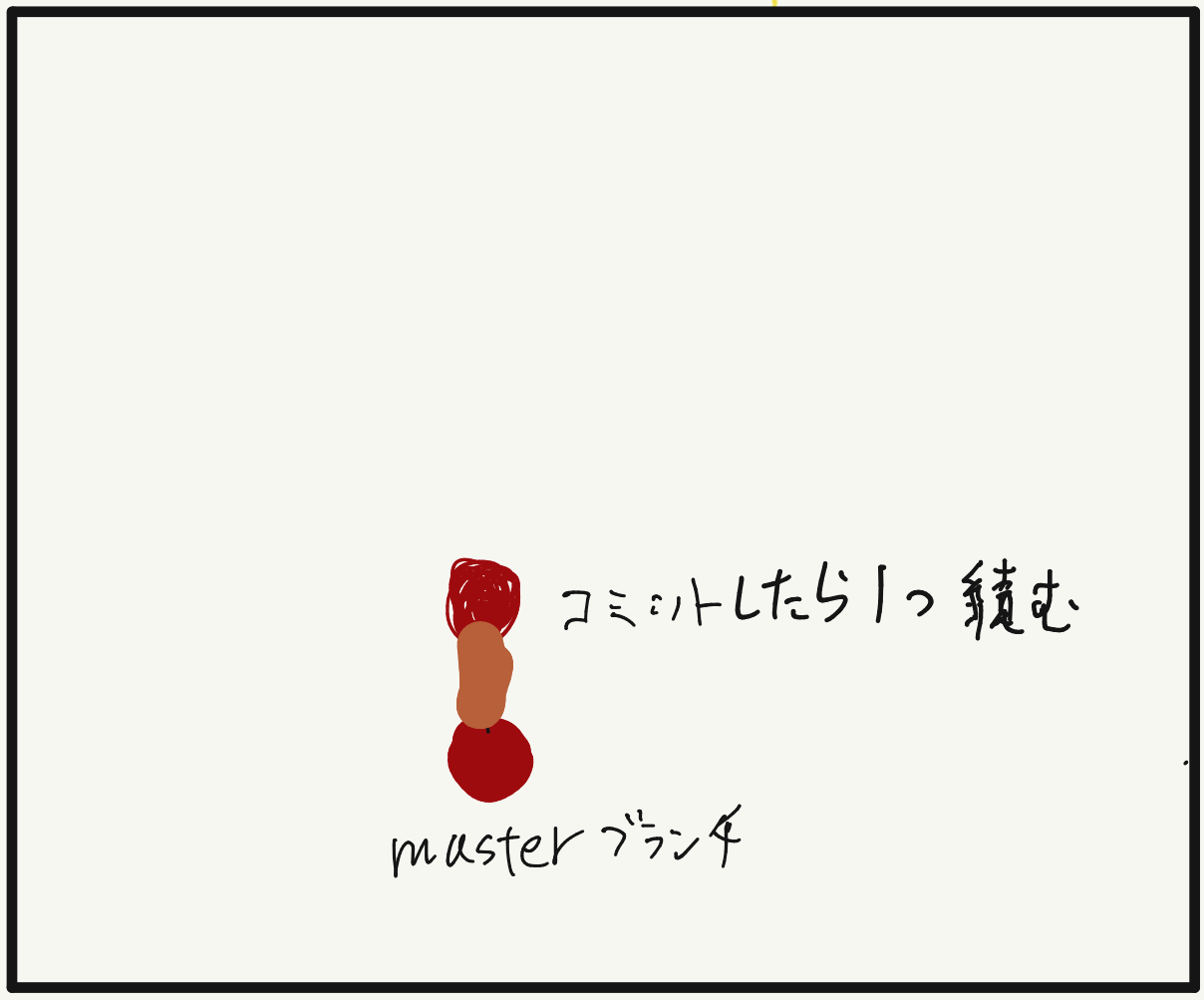
ここからは

wip

gitHub上でリポジトリを作って
git cloneした直後



何もファイルを
いじっていないので
ワーキングツリー
がクリーンな状態


コミットしたら1つ積む
masterブランチ

id を修正

文章を追加

master ブランチ

ブランチを作る

idを修正

文章を追加

masterブランチ

① GitHub上にリポジトリを作る.

② ローカルにクローンする. → 移動する

③ git branch で確認.
(ブランチの状態を確認)

④ git checkout -b [branch name]
(ブランチの作成)

⑤ index.htmlを書く.

⑥ git status
(リポジトリの状態を確認)

⑦ git add → git staus

⑧ git commit → git status

⑨ git push origin head → git Status

⑩ githubを確認

⑪ pullRequestを出す

⑫ okなら mergeする

⑬ git checkout master
(main)

⑭ git pull

⑮ git checkout -b [branch name]

⋮

続く

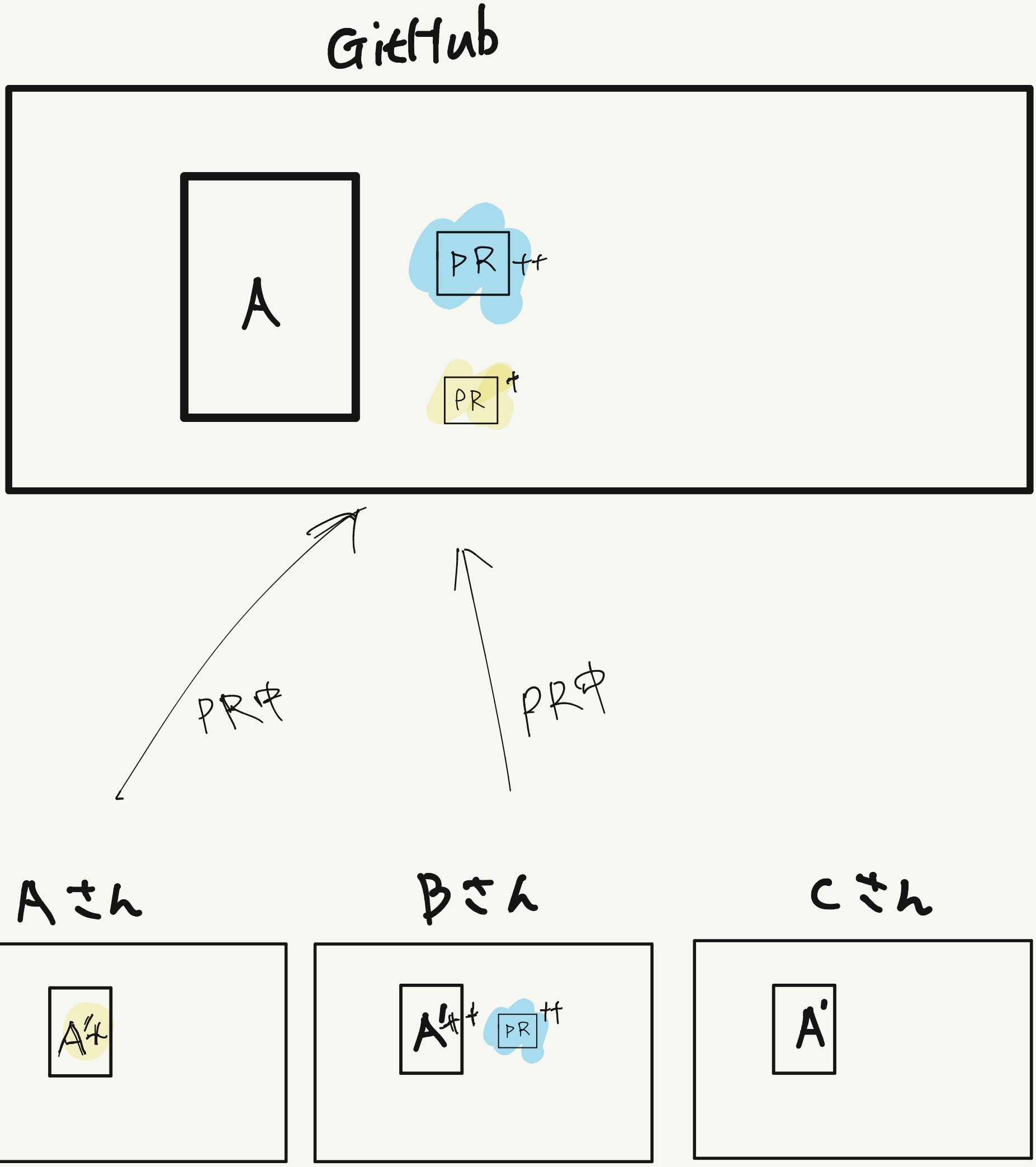
最後に

「コンフリクト」

とはどういう

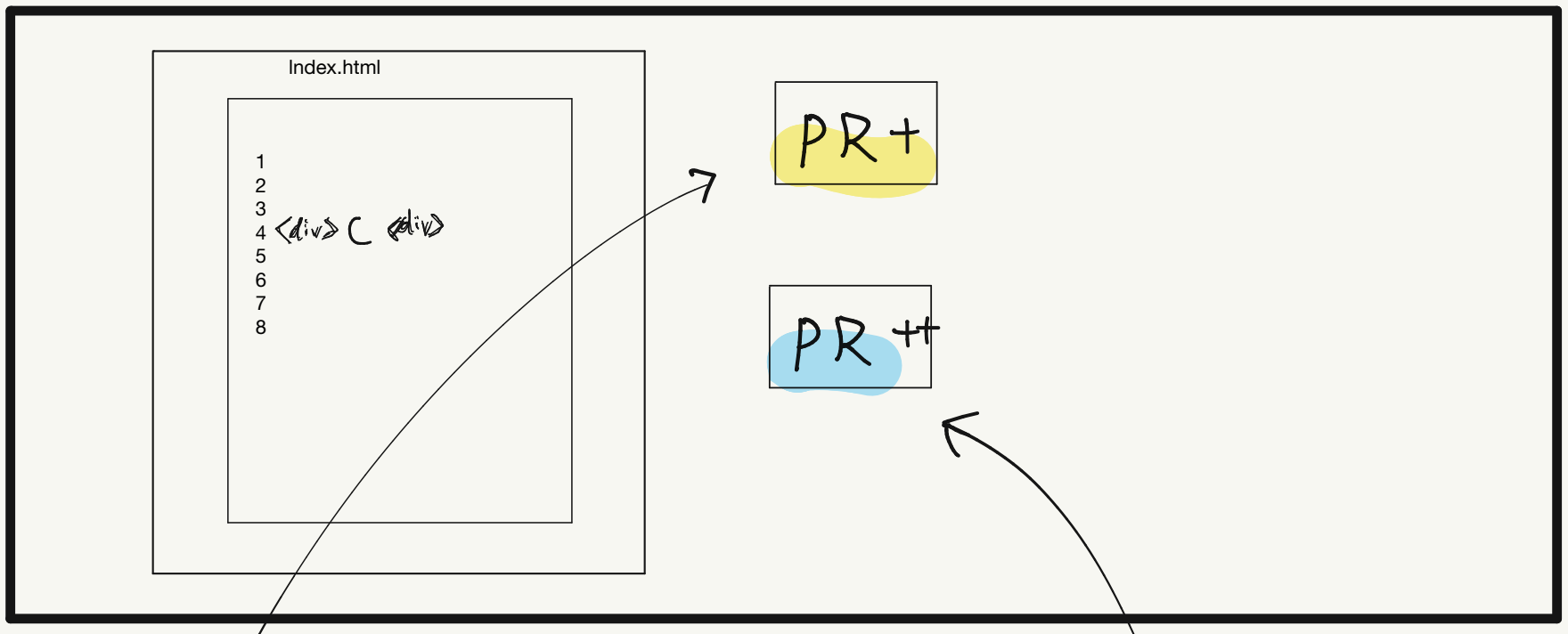
状態か？

コンフリクトパターン①



この時の2人の変更箇所

同じ行をそれぞれ違うように修正してたら.



Aさんの

Index.html

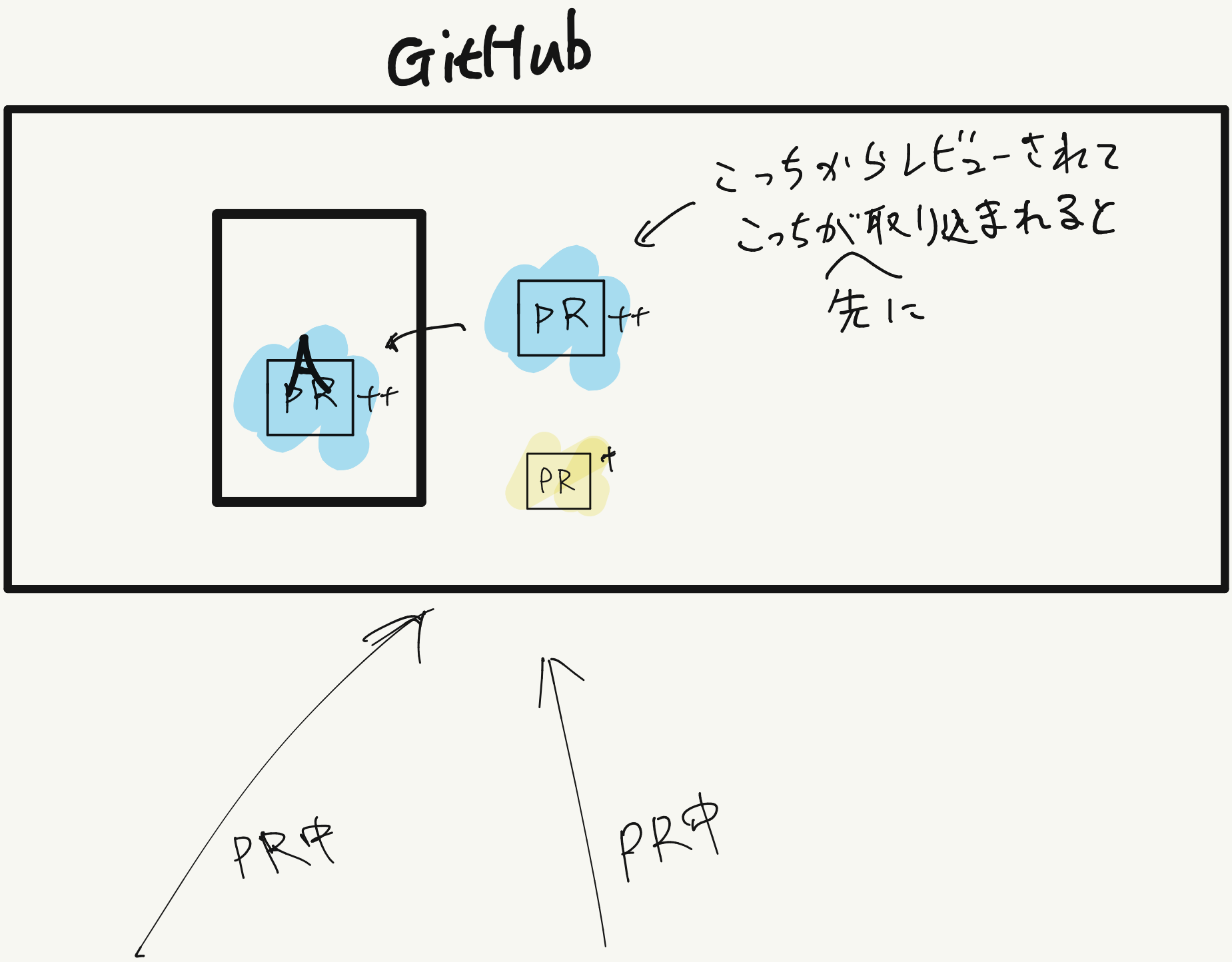
```
1
2
3
4 <div> A </div>
5
6
7
8
```

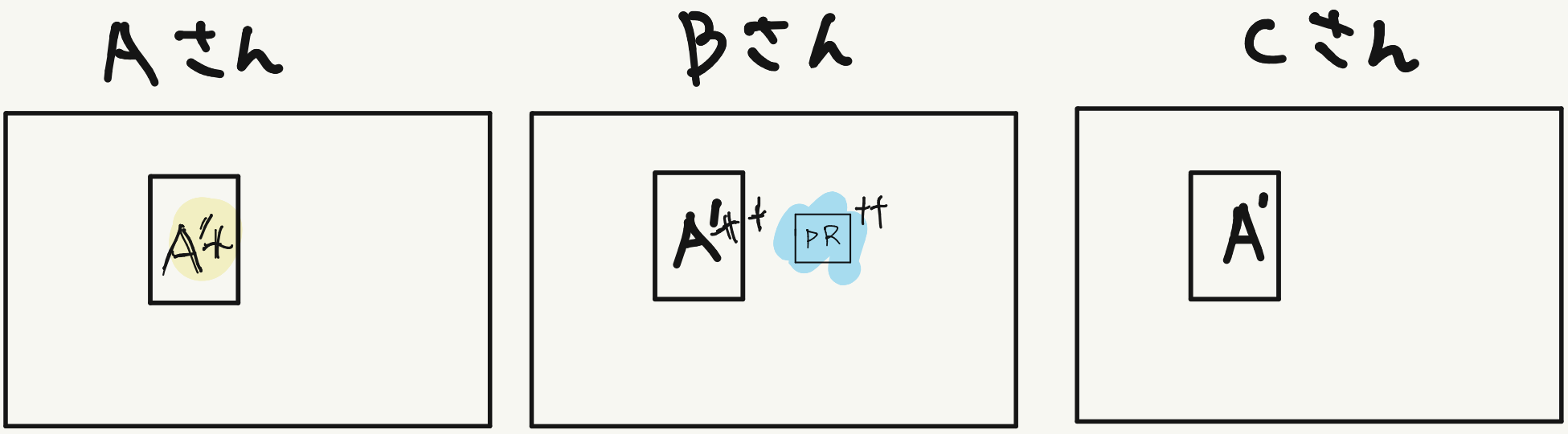
Bさんの

Index.html

```
1
2
3
4 <span> B </span>
5
6
7
8
```

コンフリクトパターン①





Index.html

1
2
3
4 C
5
6
7
8

Bさんの が最新になり

Index.html

1
2
3
4 B
5
6
7
8

PR +

バッティングする

なぜな、

Aさんの変更は最新のものと比較して

Index.html

1
2
3
4 A
5
6
7
8

どちらの変更を

生かすか

を決めて

diffのしょうとつを

解消する