

共同開発環境を構築しよう

檜内蒼太郎

自己紹介

- 檜内蒼太郎
- 和歌山工業高等専門学校 電気情報工学科 3年
- 実績
 - 「WRO全国大会」優勝
 - 「セキュリティ・キャンプ全国大会 2023」Cコンパイラゼミ受講生
- 趣味
 - 散歩等
 - 意外と外に出るのが好き
- 分野
 - セキュリティ
 - 低レイヤ

目的

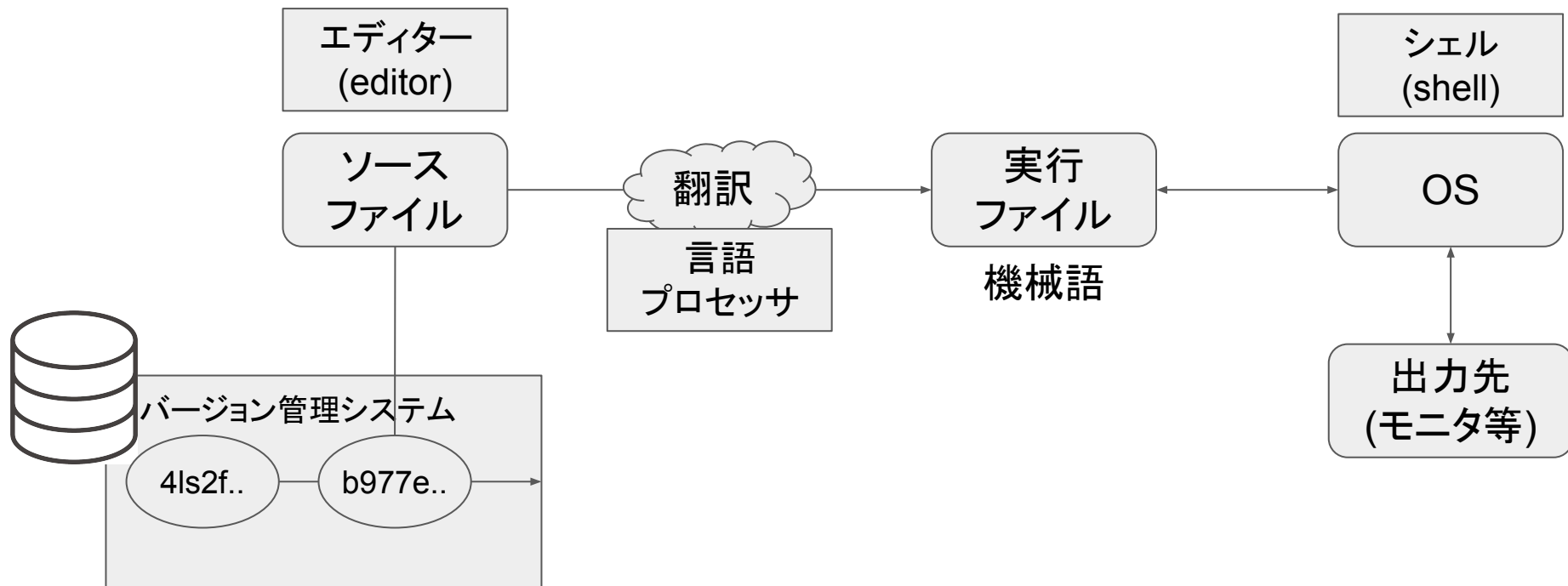
学びたい言語の環境を自分で構築することができるようになる

- C、C++、C#、Java、JavaScript、Python、Go、Rust等いろいろな言語がある
- これらの開発環境を自力で構築できるようになる
- 今回はGo言語を例に、全ての言語で共通なことを学ぶ

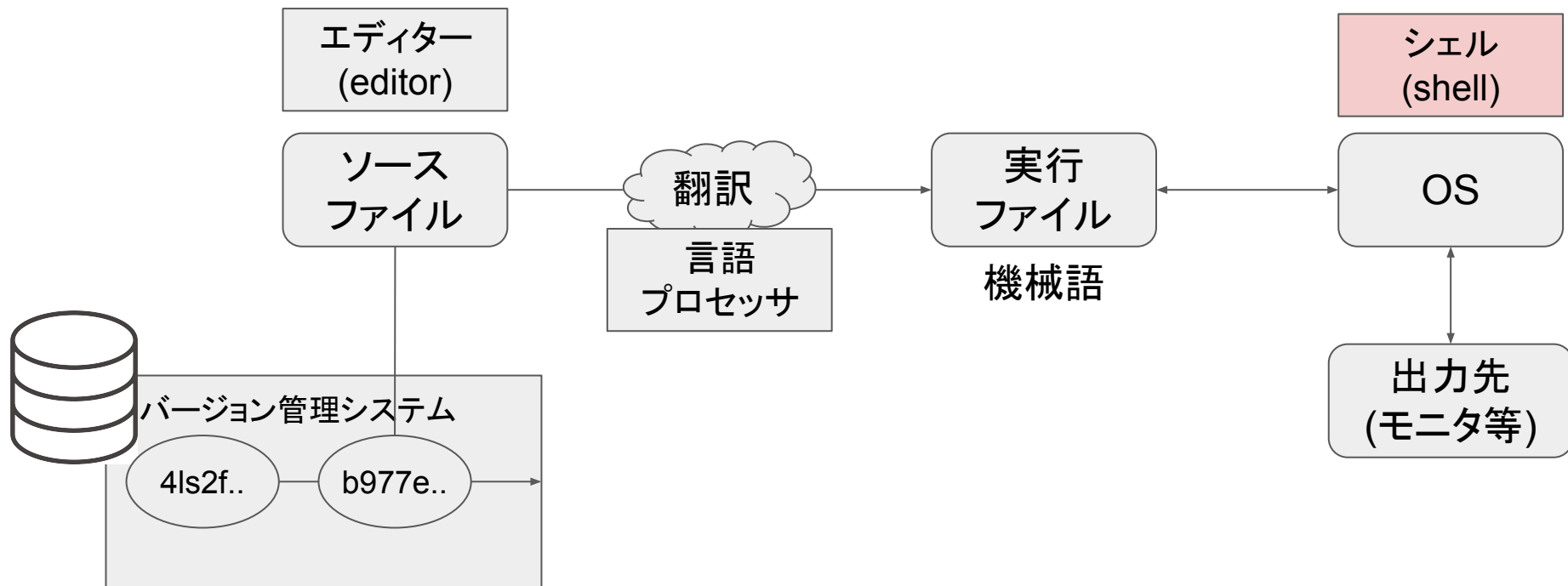
複数人で共同開発を行う環境を構築する

- 近年オンライン化が進んでいる
- プログラムをどうやって共同で開発するのか？
- コミュニケーションはどのように行うのか？
- 等の問題を解決する

個人開発での一般的な開発環境



個人開発での一般的な開発環境



シェル(shell)

シェルとは

- OSと受付窓口
- 人間はコンピュータに直接命令することができない
- シェルを介してOSに命令をする

有名なシェル

- Windos: コマンドプロンプト、PowerShell
- Mac: ターミナル(bash, zsh)
- Linux: bash, zsh, ksh, sh

シェル講座

ディレクトリ(directory)

- フォルダの別名と考えて問題ない

カレントディレクトリ

- 現在ユーザ自身がいるディレクトリ

シェル講座

パス(path)

- ファイルの場所を表すもの。「ディレクトリ名/ディレクトリ名/ディレクトリ名/ファイル名」の形式で表す。
- ファイルの住所のようなもの

絶対パス

- 階層構造の頂点(ルートディレクトリ)を基準に表したパス
- 住所で言えば、「日本/和歌山県/御坊市/名田町/野島77」

相対パス

- 現在ユーザ自身がいるディレクトリ(カレントディレクトリ)を基準に表したパス
- 住所で言えば「隣の隣の家」

シェル講座

相対パス

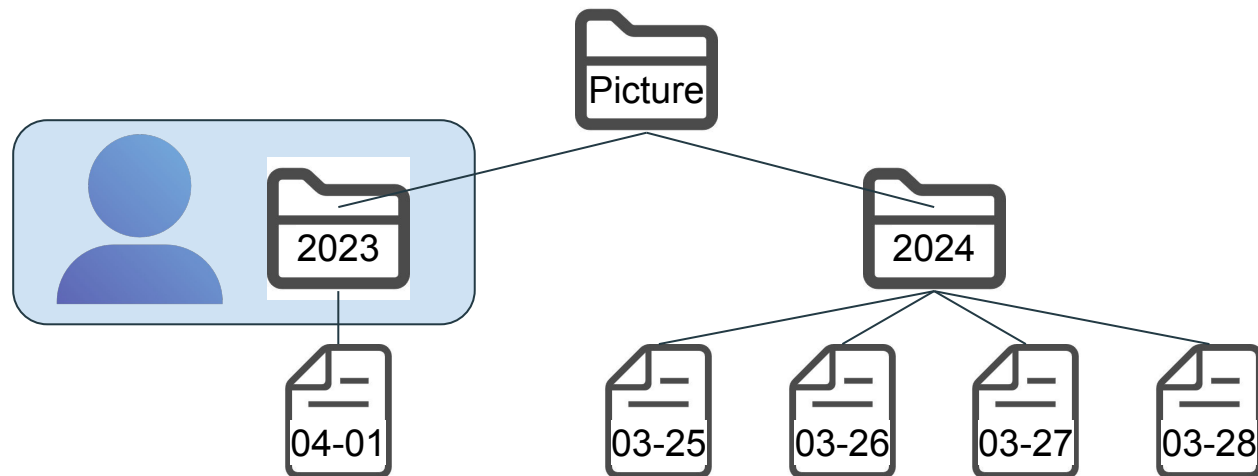
- 現在ユーザ自身がいるディレクトリ(カレントディレクトリ)を基準に表したパス
- 住所で言えば「隣の隣の家」

パスの表記	意味
.	カレントディレクトリを指す
..	1つ上のディレクトリを指す

課題1-1

1) カレントディレクトリが「2023」の時以下のファイル又はディレクトを表す相対パスを答えなさい

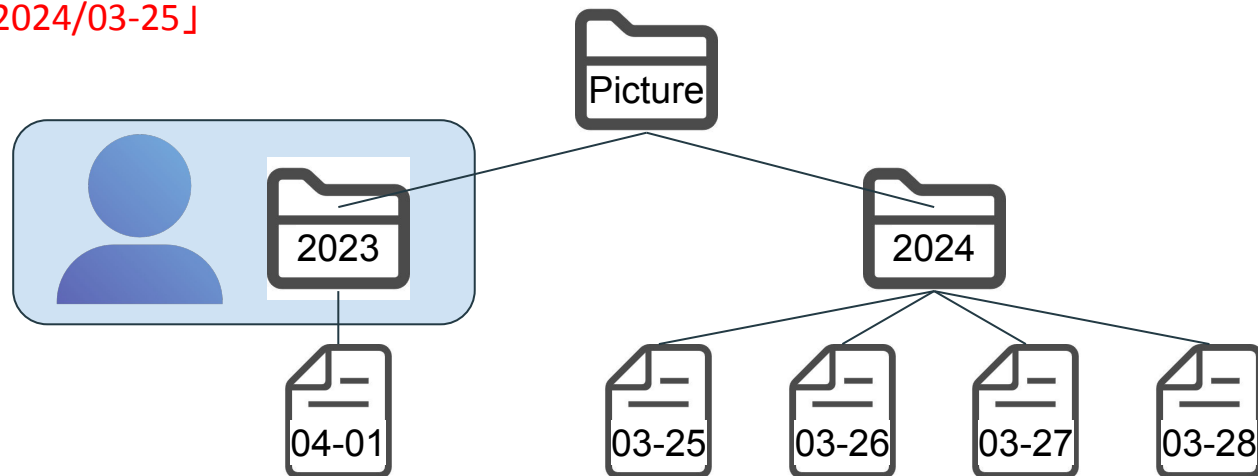
- a) 例)「04-01」:「./04-01」
- b) 「2023」:
- c) 「2024」:
- d) 「03-25」:



課題1-2

1) カレントディレクトリが「2023」の時以下のファイル又はディレクトを表す相対パスを答えなさい

- a) 例)「04-01」:「./04-01」
- b) 「2023」:「.」
- c) 「2024」:「../2024」
- d) 「03-25」:「../2024/03-25」



シェル講座

- コマンド: シェルを介して与えることができる命令のこと
- オプション: コマンドの機能を指定する命令のこと

書式

\$ コマンド名 -オプション [-オプション] <引数> ...

この講義で
\$ はシェルで操作を
表す記号

この講義で
[] は省略可能を
表す記号

この講義で
< > は適切な引数を
表す記号

シェル講座

Windows

1. 「PowerShell」シェルを起動

コマンド名	意味
ls [-al]	ディレクトリ内のファイルを表示
cd <directory>	ディレクトリを移動
mkdir <directory>	フォルダの作成
touch <file>	ファイルの作成 (Windowsの場合は「ni」)
cat <file>	ファイルの中身を表示
pwd	カレントディレクトリのフォルダのパスを表示

Mac

1. 「Terminal」を起動

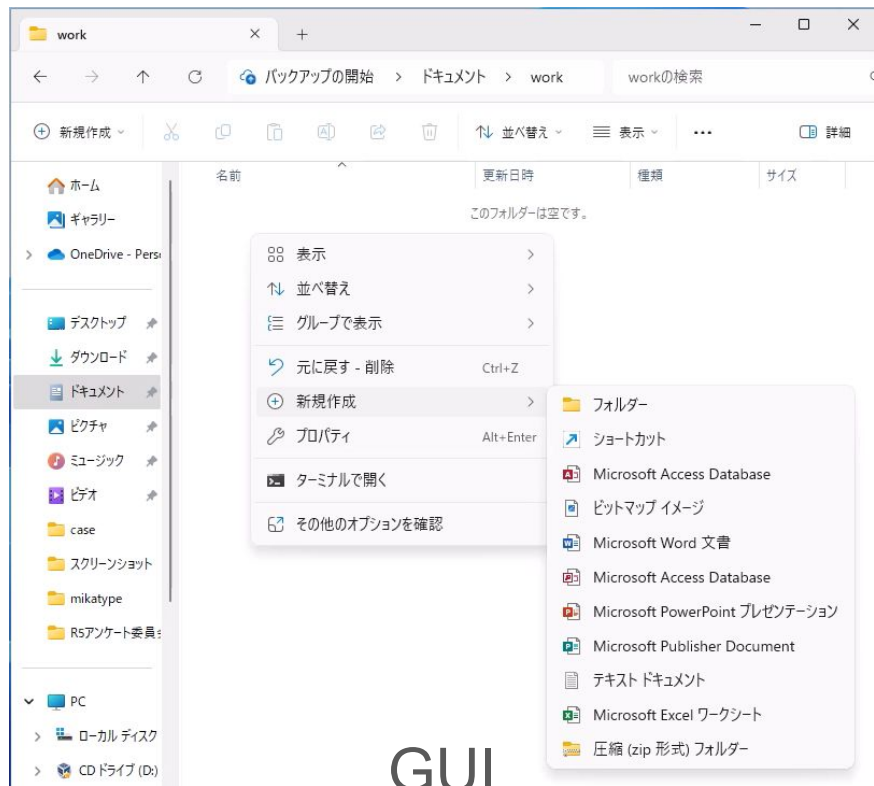
課題1-3

- 1) 「lsコマンド」を実行しなさい
- 2) 「cdコマンド」を実行しなさい

シェル講座

- CUI:Character User Interface
 - 文字で操作する方法
 - 細かな操作を行うことができる
 - シェルでの操作もCUIである
 - CUIでできて、GUIでできない操作は数多くある
- GUI:Graphical User Interface
 - 文字だけではなく、マウスや、クリックで操作する方法
 - 直感的な操作が行える
 - 内部でCUIの命令が動作していると考えて良い

シェル講座



GUI

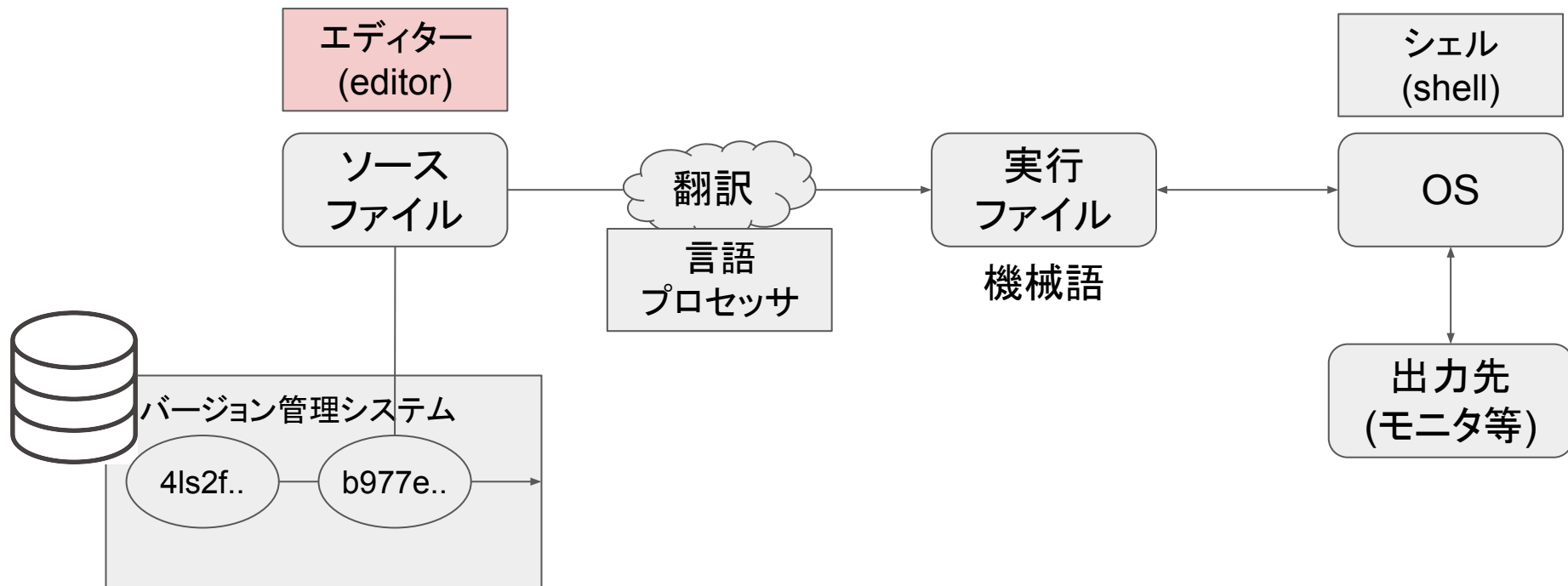
\$ mkdir <directory>

\$ ls

\$ cd <directory>

CUI

個人開発での一般的な開発環境



エディター(editor)

エディタとは

- 文字情報の入力・編集・保存を可能とするソフトウェア
- メモ帳やWordもその一つ
- Wordはレポートなどの文章を書くことを目的として作られている
- ということはプログラムを書くことを目的としたものもある

有名なエディタ

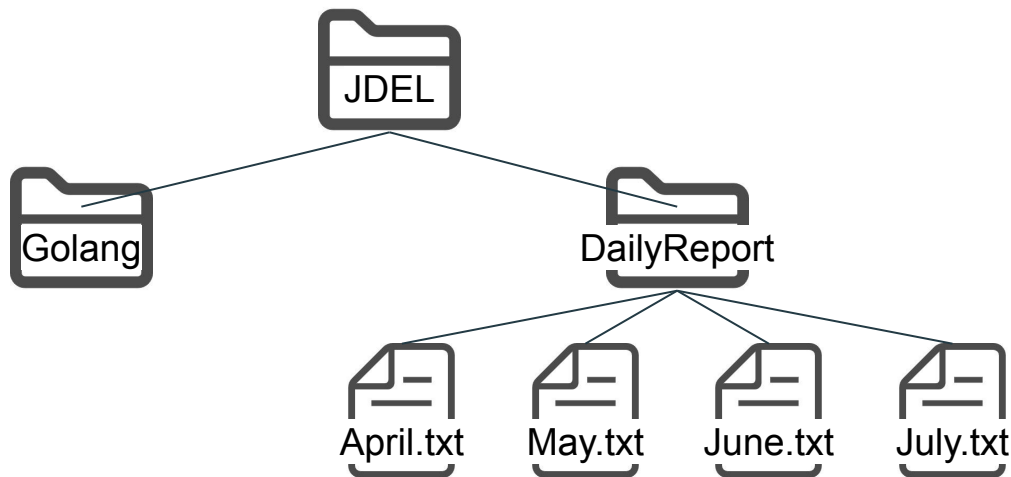
- メモ帳
- Visual Studio Code
- Notepad++
- Atom
- 秀丸エディタ

エディタ講座

- 今回は「Visual Studio Code (VSCode)」を使用する
- 「VSCode」のインストール
 - Windowsユーザは事前課題でダウンロードしたファイルをクリックする
- 「VSCode」の拡張機能のインストール
 - 「Japanese Language Pack for Visual Studio Code」を入れてもよい
 - 「Go」
 - 「Git Graph」

課題2-1

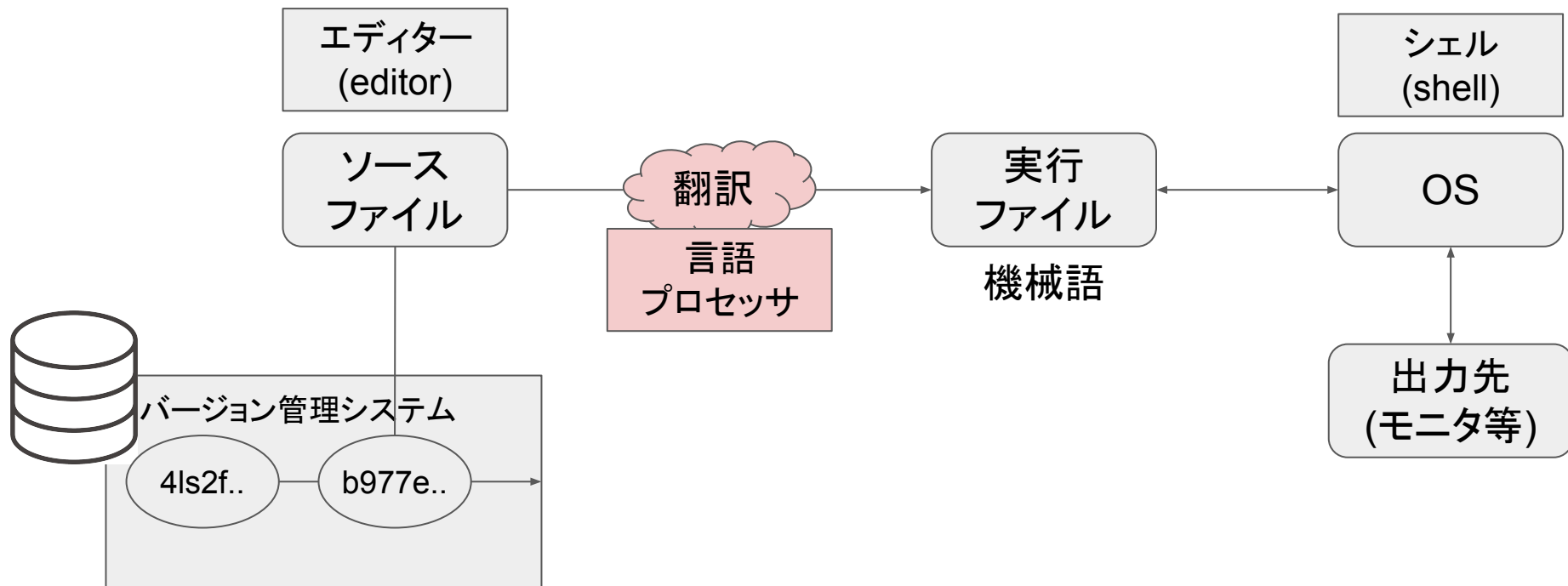
- 1) 「JDEL」というディレクトリを作成し、VSCodeで開きなさい
- 2) VSCodeから下のようなディレクトリとファイルを作りなさい
- 3) JDEL/DailyReport/April.txtに右下のような文を記述し保存しなさい



1日
今日は昼寝をした

2日
今日も昼寝をした

個人開発での一般的な開発環境



言語プロセッサ

言語プロセッサとは

- ソースコードを実行可能な形式に変換するソフトウェア
- コンパイラ: 言語プロセッサの内、全てのソースコードを一度に変換する
- インタプリタ: 言語プロセッサの内、部分的にソースコードを変換する

有名な言語プロセッサ

- コンパイラ
 - C、C++、C#、Java、Go等
- インタプリタ
 - Python、Ruby、PHP等

言語プロセッサ講座

- 今回は「Goコンパイラ」をインストールする
- Windowsユーザ、Macユーザは事前課題でダウンロードした「Go」をインストールする
- 「Go」が正しくインストールされたか確認
 - `$ go version`
 - 「go version gox.xx.x xxxxx/xxxx」と表示される
- 「VSCode」で、Goツールのインストールと更新を行う
 - 「View」->「command Palette」をクリックし、「Go: Install/Update tools」を検索し、全てにチェックを入れて、「OK」を押す

Golangの言語仕様

- 今回はGo言語を学ぶ目的ではない
- 開発環境を整えることが目的
- そのため、Go言語の必要最低限の言語仕様だけを紹介する
- 本来は自分で勉強する部分
- わからなくても問題ない！！
- 右はGo言語の基本的な書き方

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
}
```


Golangの言語仕様

機能	書式	備考
変数宣言	var 変数名 データ型	データ型には以下のようなものがある int(整数), float64(実数), string(文字列)
変数参照	変数名	C言語とほとんど同じように使える
変数代入	変数名 = 式	
関数宣言	func 関数名 (引数, ...) 戻り値のデータ型 { ... }	例) func sample (x int, y string) int { ... }
関数呼び出し	関数名(引数, ...)	C言語とほとんど同じように使える
if文	if 条件式 { ... }	演算子はC言語とほとんど同じ +, -, /, %, *, ==, !=, &&, ,
if - else文	if 条件式 { ... } else { ... }	
for文	for 初期化; 条件式; 後処理 { ... }	for 条件式 { ... } でwhile文を実現

Golangの標準ライブラリー

書式	機能
<code>fmt.Println(文字列又は、変数名)</code>	標準出力(画面)へ表示 C言語のprintf()に該当
<code>fmt.Scan(&変数名)</code>	標準入力(シェル)からデータを受け取る C言語のscanf()に該当

課題3-1

- 1) 「Hello World」と表示するプログラムを「JDEL/Golang/Hello.go」というファイルを作成し、Go言語で記述しなさい

```
package main

import "fmt"

func main ( ) {
    fmt.Println("HelloWorld")
    return
}
```

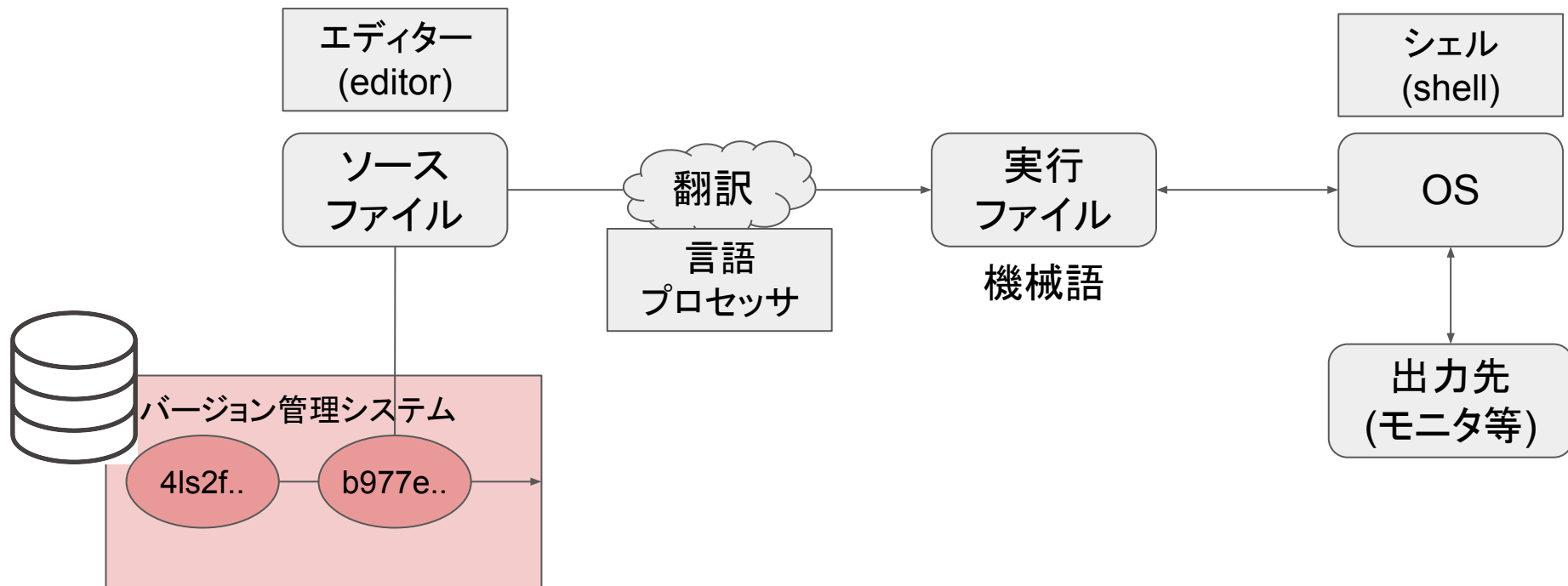
課題3-2

- 1) 「Hello World」を表示させなさい
 - a) `$ go build ファイル名`
 - b) `$./ファイル名`
 - c) ※「`$ go run ファイル名`」でコンパイルと実行をまとめて行える

課題3-3 (時間に余裕があれば)

- 1) 入力された整数が、偶数かどうかを判断するプログラムを作成せよ。偶数の場合は「偶数」と表示し、奇数の場合は「奇数」と表示しなさい。
- 2) 入力された整数が、素数かどうかを判断するプログラムを作成せよ。素数の場合は「素数」と表示し、素数でない場合は「素数でない」と表示しなさい。

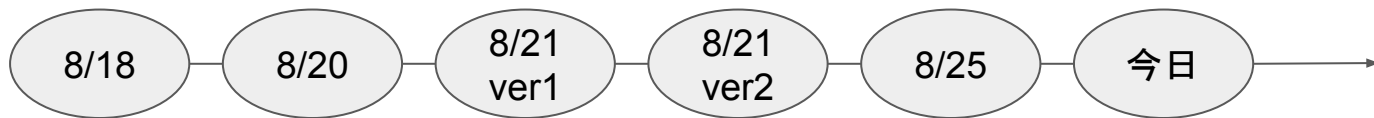
個人開発での一般的な開発環境



バージョン管理システム

バージョン管理システムとは

- ソースコードなどのファイルのバージョンを管理するツール
- ソースコードを変更する前の状態に戻すことなどができる



有名なバージョン管理システム

- Git
- Subversion

バージョン管理システム講座

- 今回は「Git」を使用する

Windows

1. 「Git」のインストール

- a. 事前課題でダウンロードしたインストーラを実行

Mac(事前課題で実施済み)

1. 「Git」のインストール

- a. Terminalを開き
- b. `$ brew install git`
- c. を実行

バージョン管理システム講座

1. 「Git」のインストール確認

- a. `$ git --version`
- b. 「git version x.xx.x」と表示される

2. Gitの初期設定

- a. eメールの登録
- b. `$ git config --global user.email "you@example.com"`
- c. 名前の登録
- d. `$ git config --global user.name "Your Name"`

3. Gitの初期設定の確認

- a. eメールの確認 (以下のコマンドを実行して登録した emailが表示されればよい)
- b. `$ git config user.email`
- c. 名前の確認 (以下のコマンドを実行して登録した nameが表示されればよい)
- d. `$ git config user.name`

バージョン管理システム講座

リポジトリ

- バージョンを管理するデータベースのこと

コミット

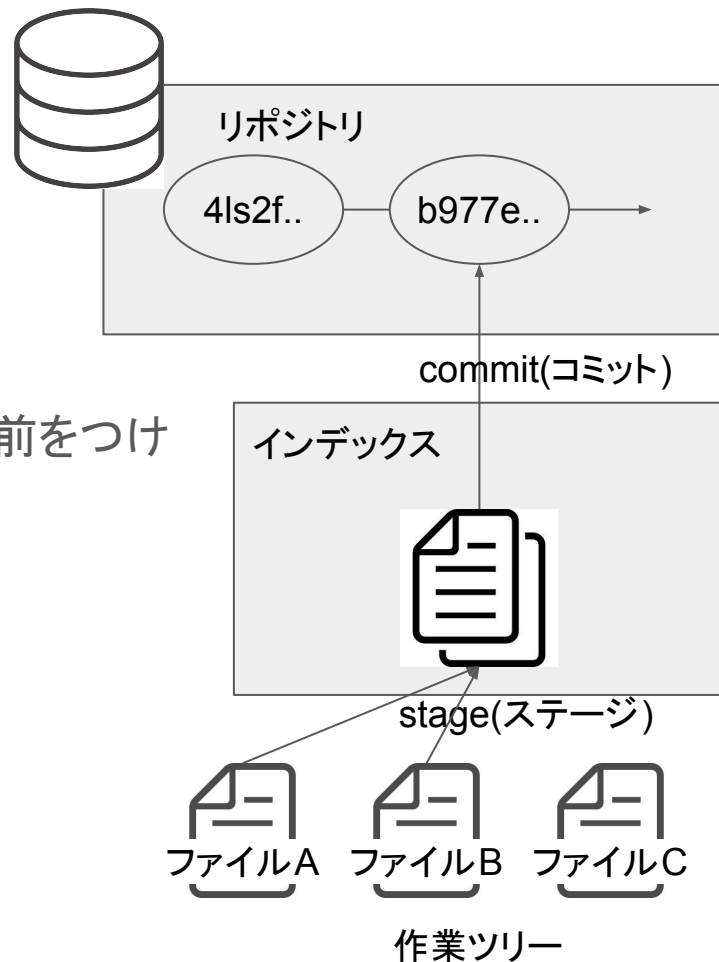
- バージョンを管理するための単位(バージョンに名前をつけるイメージ)

インデックス

- 一時的に変更内容を貯めておく場所
- コミットを行う時に取り込まれるファイル等の集まり

作業ツリー

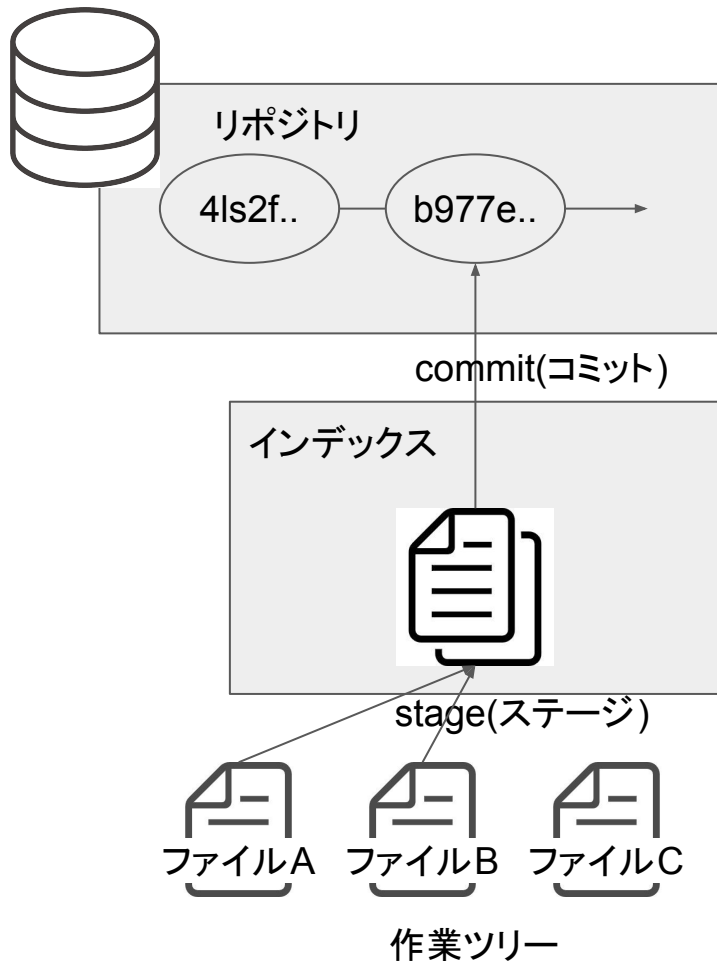
- 現在、作業をしているファイル
- 書き換え可能なファイル



バージョン管理システム講座

Gitリポジトリの作成

1. フォルダを開く
2. 「リポジトリを初期化する」をクリック



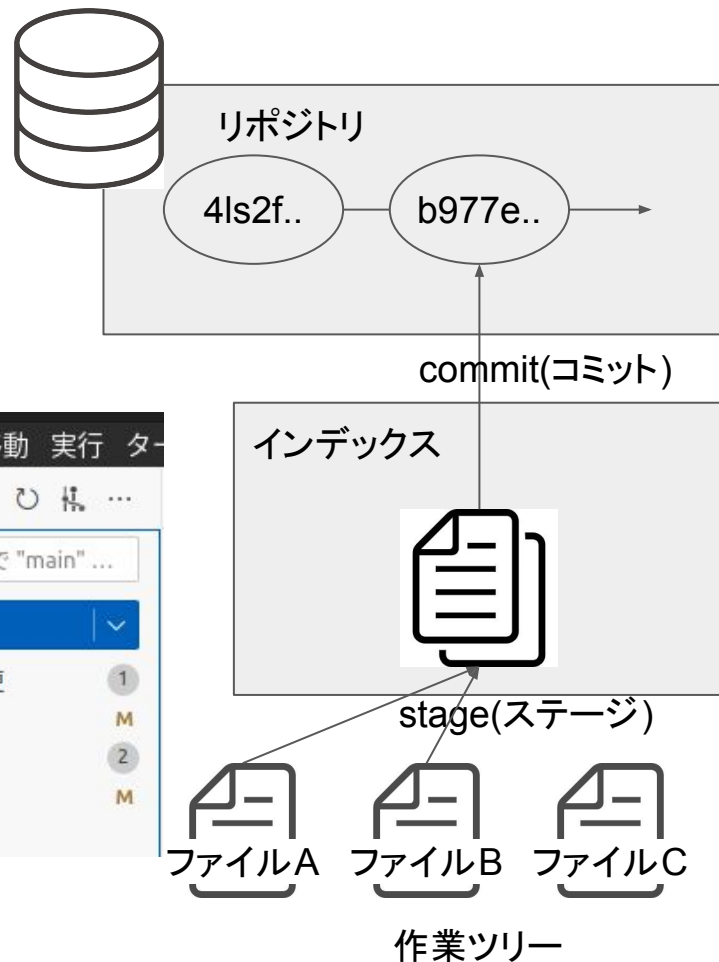
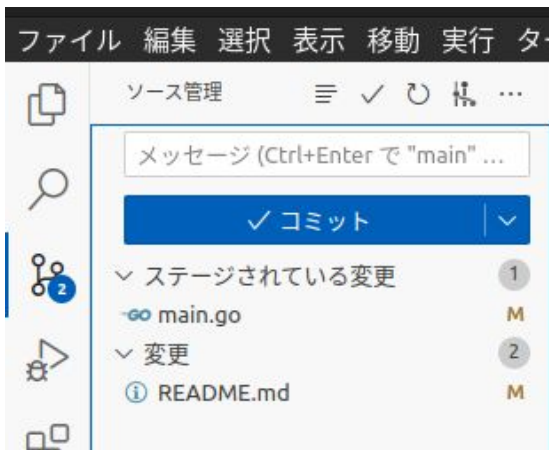
課題4-1

- 1) 「DailyReport」のリポジトリを作成しなさい
 - a) VSCodeで「DailyReport」フォルダを開く
 - b) 「ソース管理」から「リポジトリを初期化」をクリック

バージョン管理システム講座

インデックスに追加する(ステージする)

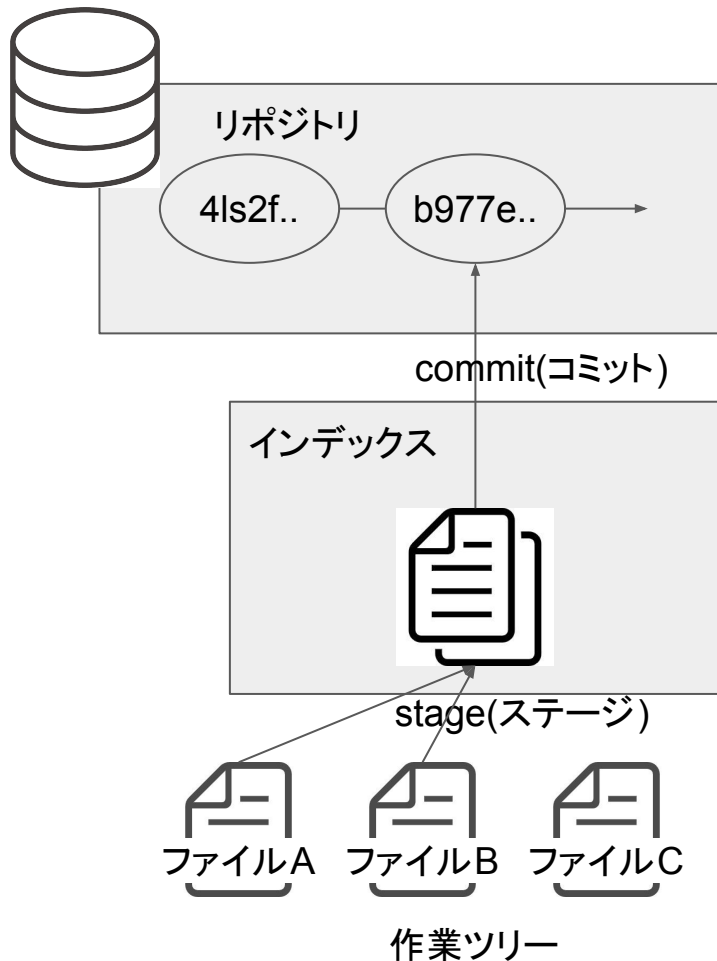
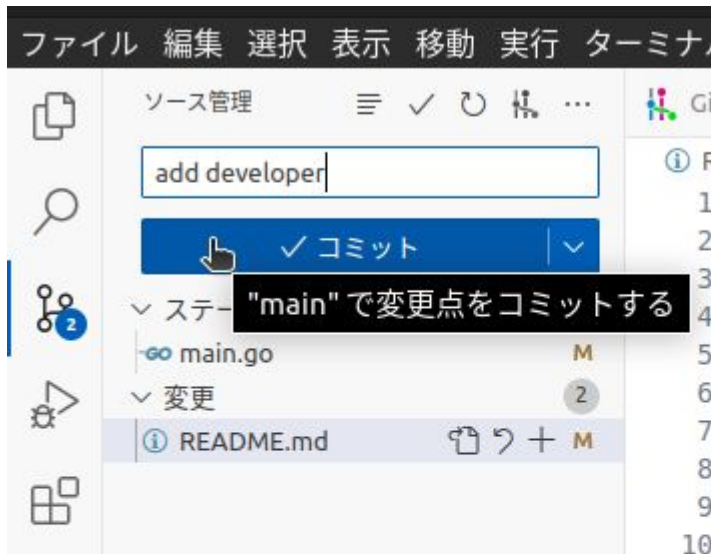
1. ソース管理の「+」をクリック



バージョン管理システム講座

コミットする

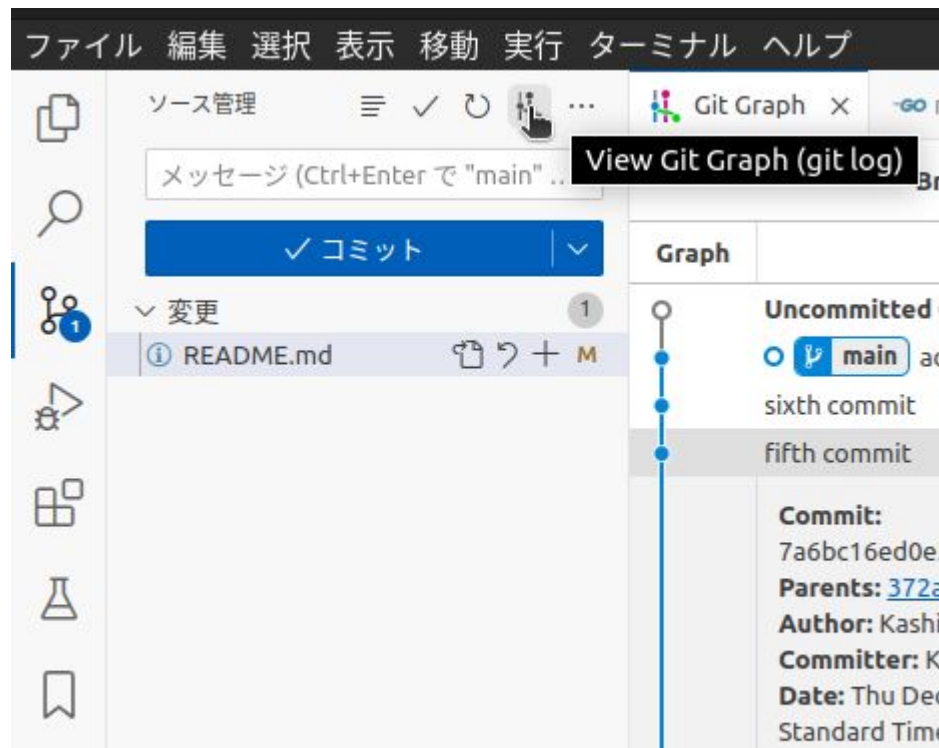
1. 「メッセージ」に任意のコミットメッセージを記入
2. 「コミット」ボタンをクリック



バージョン管理システム講座

ログを確認する

1. 「ソース管理」の「View Git Graph」をクリック



課題4-2

- 1) 「April.txt」をインデックスに追加し、コミットしなさい
 - a) 「ソース管理」で「April.txt」ファイルの「+」をクリック
 - b) 「メッセージ」にコミットメッセージを記入
 - c) 「コミット」をクリック
 - d) 「Git Graph」でログを確認
- 2) 「May.txt, June.txt, July.txt」をインデックスに追加し、コミットしなさい
 - a)
 - b)
 - c)
 - d)

バージョン管理システム講座

Git Graphの見方(コミットが持つ情報)

The screenshot shows the Git Graph interface with a table of commits. Red boxes with arrows point to specific elements, explaining their meaning:

- HEAD**: Points to the current branch tip (main).
- 親となるコミットのコミットハッシュ** (Commit hash of the parent commit): Points to the parent commit hash `c8db9c9938f54f26930f24099c70f1b0` in the **Parents** field.
- コミットした日時** (Commit date and time): Points to the **Date** column.
- コミットハッシュ
コミットを一意に識別するための文字列** (Commit hash: A string used to uniquely identify a commit): Points to the **Commit** column.
- コミットメッセージ** (Commit message): Points to the **Description** column.
- コミットした人** (Commit author): Points to the **Author** column.

Graph	Description	Date	Author	Commit
Uncommitted Changes (1)		8 Dec 2023 19:48	*	*
HEAD → main	4th commit	8 Dec 2023 19:45	Kashiuchi Sotaro	7785bf1f
	Commit: 7785bf1f1e05702bfafdb131886404f5dd...			
	Parents: c8db9c9938f54f26930f24099c70f1b0			
	Author: Kashiuchi Sotaro <sotaro0416@gmail.com>			
	Committer: Kashiuchi Sotaro <sotaro0416@gmail.com>			
	Date: Fri Dec 08 2023 19:45:37 GMT+0900 (Japan Standard Time)			
	4th commit			
3rd commit		8 Dec 2023 19:45	Kashiuchi Sotaro	c8db9c99

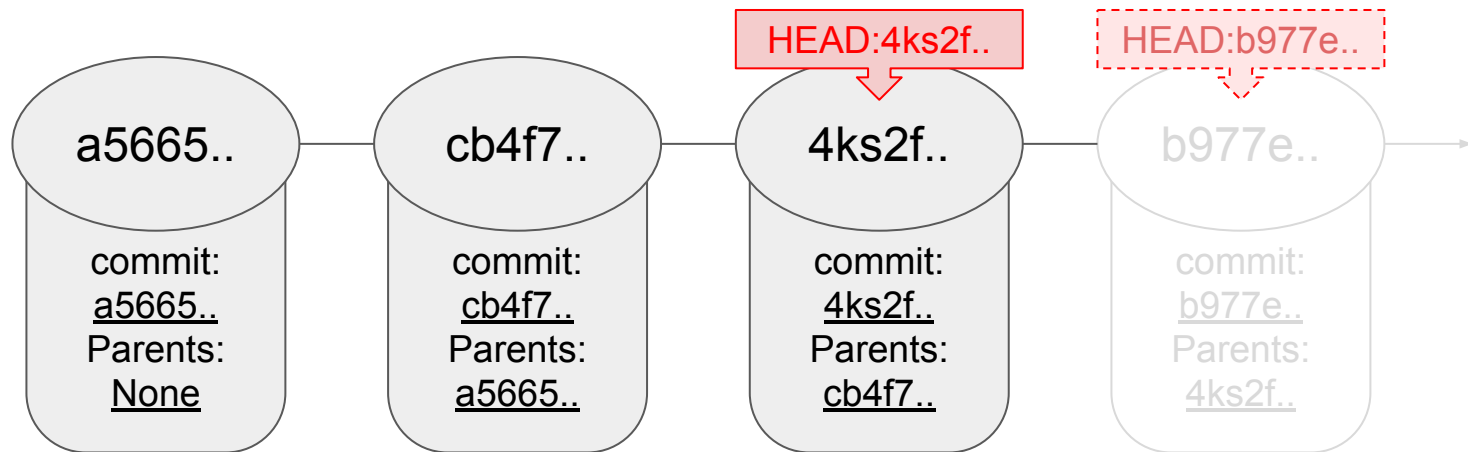
バージョン管理システム講座

コミットが持つ情報

- コミットハッシュ: コミットを一意に識別するための文字列
- 親コミット: そのコミットの一つ前のコミット

HEADとは

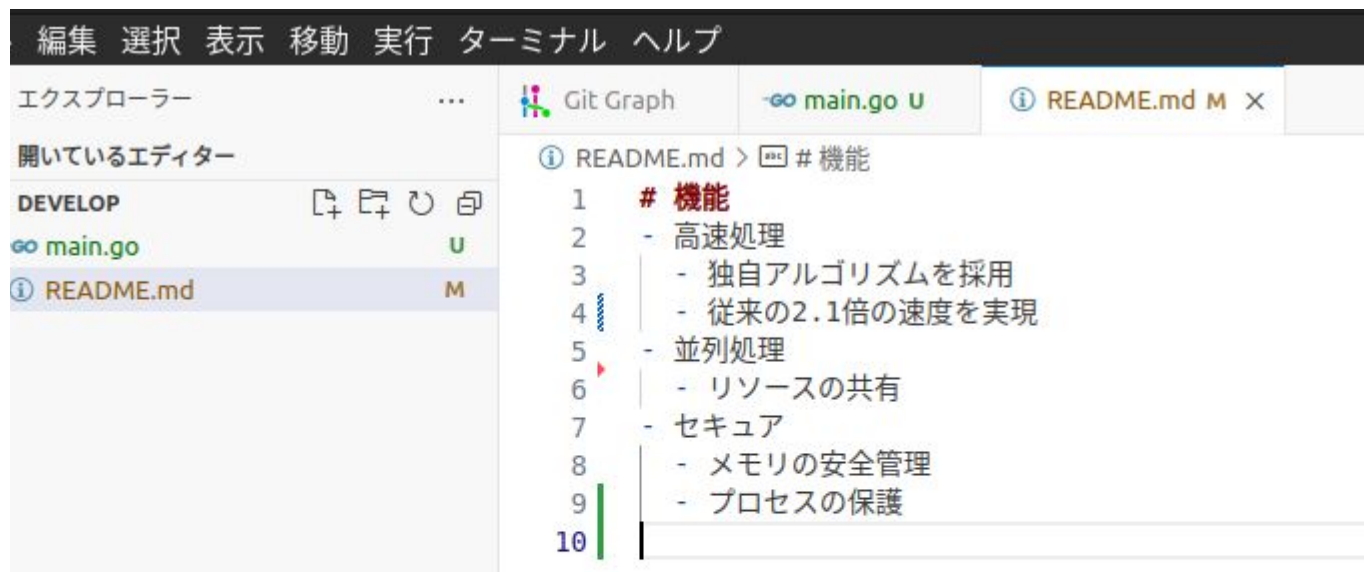
- 現在作業している場所の最後のコミットを参照(ポインタ)しているもの



バージョン管理システム講座

変更点を確認する

色	意味
青	変更した行
赤	削除した行
緑	追加した行



バージョン管理システム講座

差分を確認する

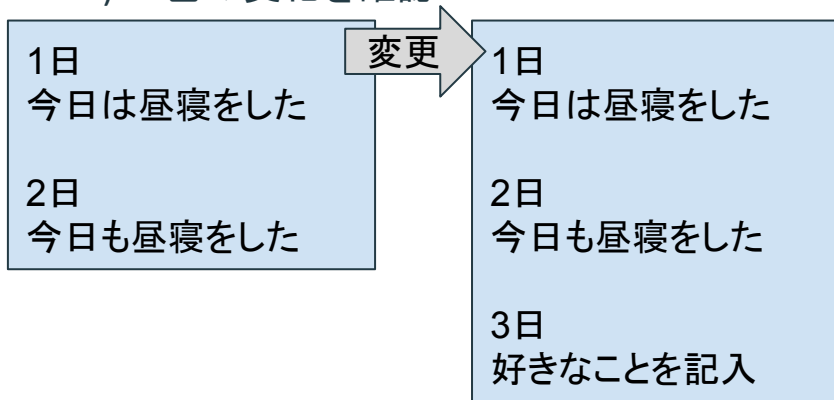
1. 「Git Graph」に表示されているファイルをクリック



課題4-3

1) 「April.txt」に以下のように変更し、差分を確認しなさい

- a) VSCodeでファイルを編集
- b) 色の変化を確認

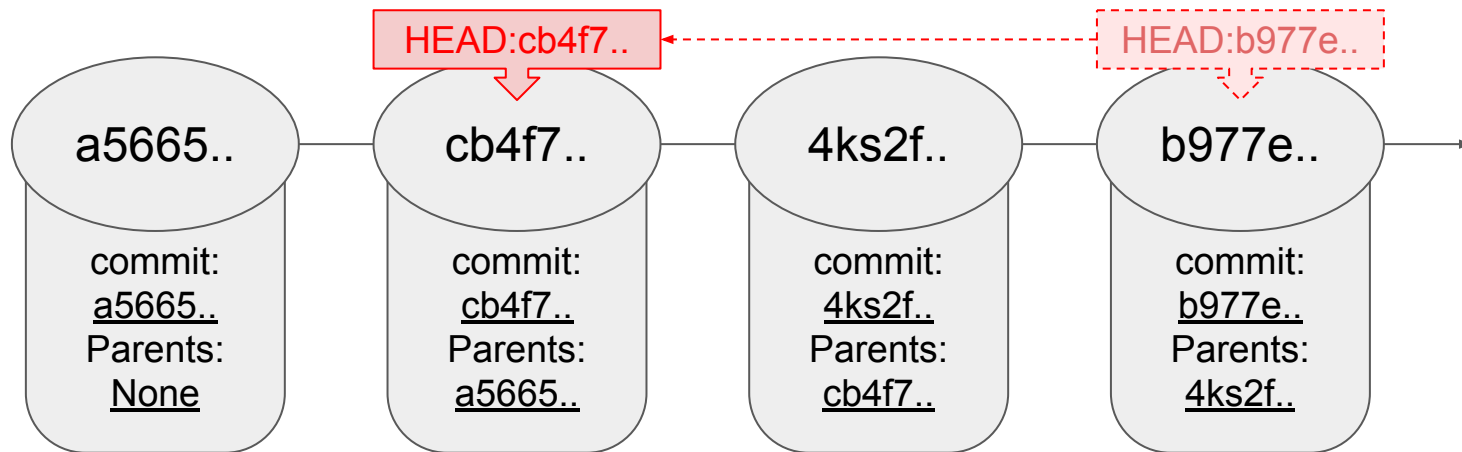


2) 変更した「April.txt」をコミットしなさい

バージョン管理システム講座

チェックアウト

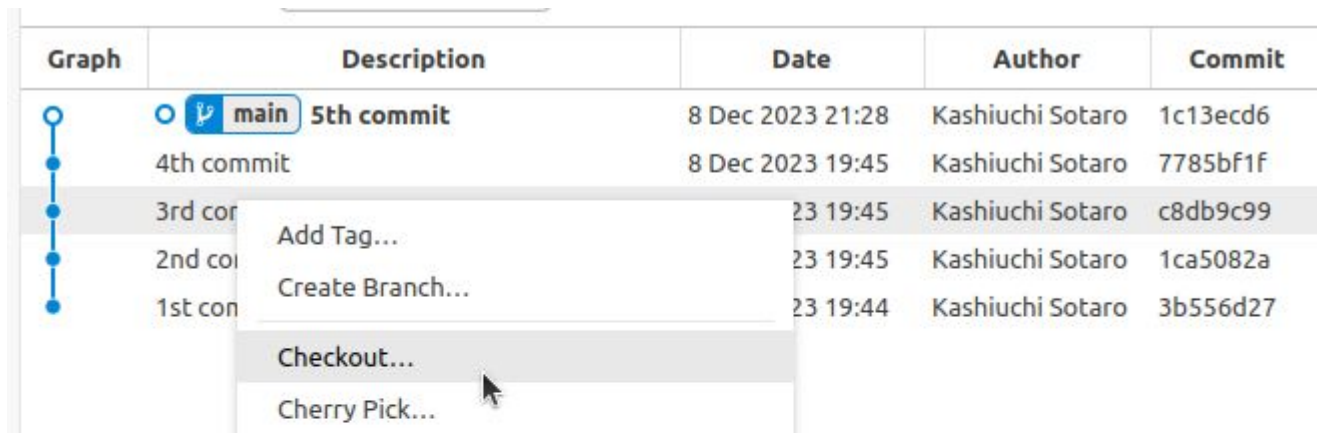
- 「HEAD」の参照(ポインタ)先を変更すること
- 過去のバージョンに戻る時などに使う
- 作業ツリーとインデックスが空の状態でないといけない









バージョン管理システム講座

チェックアウト

1. 「Git Graph」に表示されているコミットを選択する
2. 右クリックし、「Checkout...」をクリックする



Graph	Description	Date	Author	Commit
	 main 5th commit	8 Dec 2023 21:28	Kashiuchi Sotaro	1c13ecd6
	4th commit	8 Dec 2023 19:45	Kashiuchi Sotaro	7785bf1f
	3rd commit	23 19:45	Kashiuchi Sotaro	c8db9c99
	2nd commit	23 19:45	Kashiuchi Sotaro	1ca5082a
	1st commit	23 19:44	Kashiuchi Sotaro	3b556d27

Add Tag...

Create Branch...

Checkout...

Cherry Pick...

課題4-4

1) チェックアウトし、「April.txt」を以下の状態にしないさい

- a) 「Git Graph」で、一つ前のコミットを選択
- b) 右クリックし、「Checkout...」をクリックする

1日

今日は昼寝をした

2日

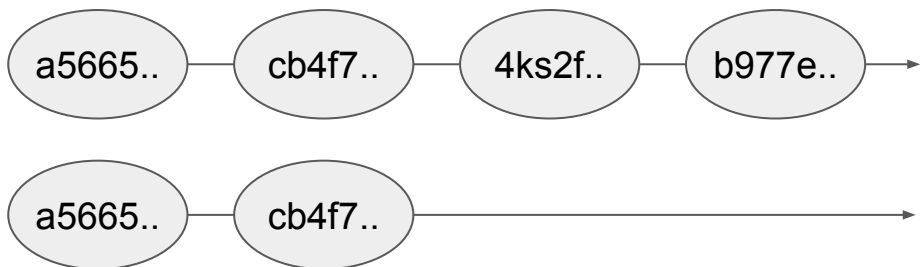
今日も昼寝をした

2) 「April.txt」を最新の状態に戻さないさい。

バージョン管理システム講座

コミットを削除する

1. 削除したいコミットの一つ前のコミット上を選択
2. 右クリックし「Reset current branch on this Commit...」をクリック
3. モードを選択し、「Yes」をクリック

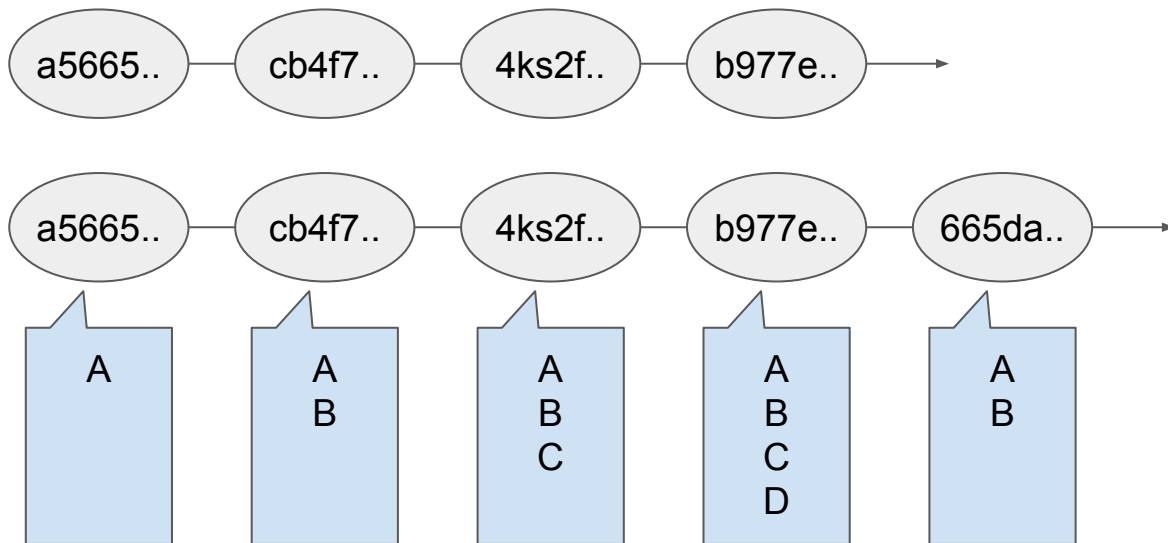


モード	コミット	インデックス	作業ツリー
soft	O	X	X
mixed	O	O	X
hard	O	O	O

バージョン管理システム講座

コミットの内容を打ち消すコミットを作成する

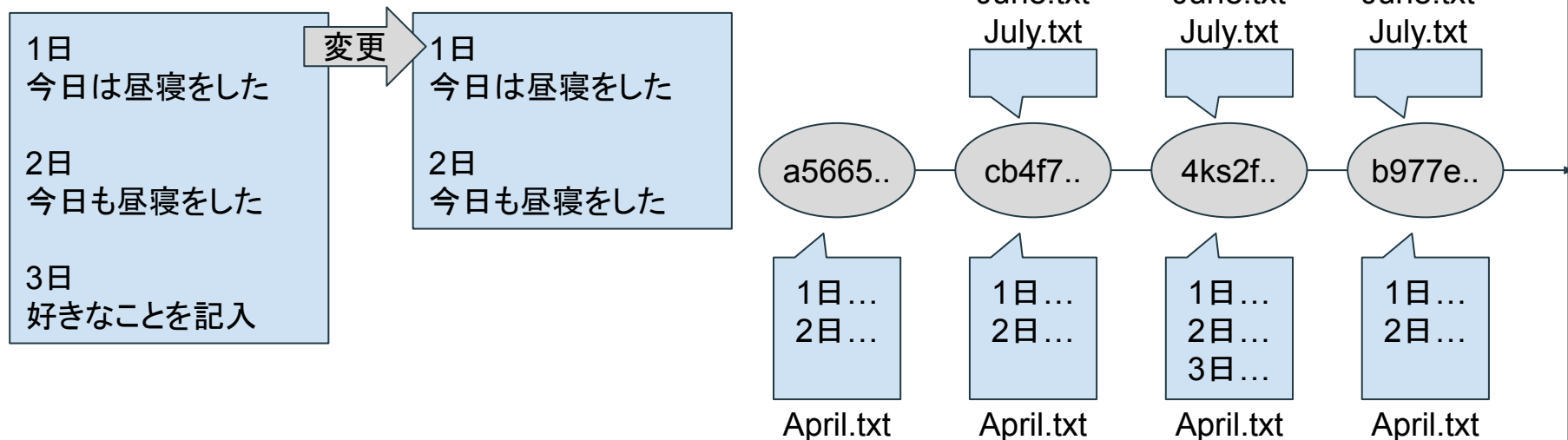
1. 打ち消したいコミットを選択
2. 右クリックし「Revert...」をクリック
3. 「Yes」をクリック



課題4-5

1) 「Revert」を用いて、右側のファイルの状態に戻さない

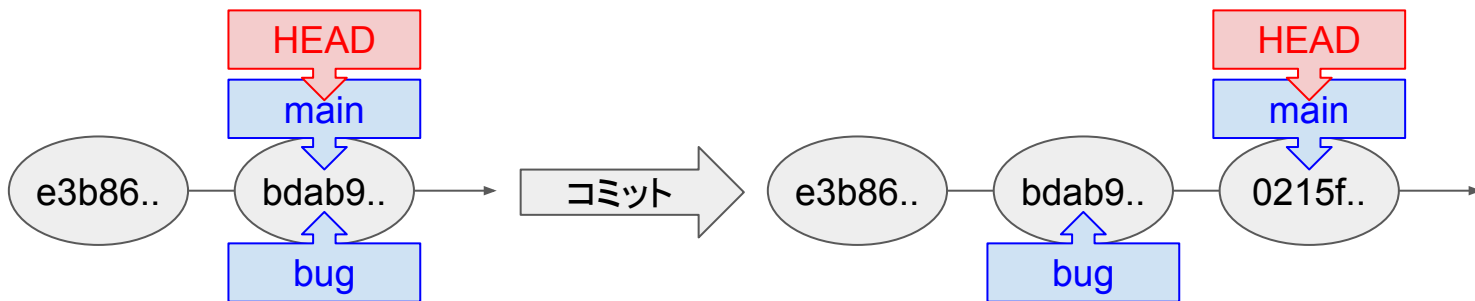
- 「Git Graph」から最新のコミットを選択
- 右クリック、「Revert...」をクリックし、ログを確認
- 「April.txt」が元に戻されていることを確認する



バージョン管理システム講座

ブランチ

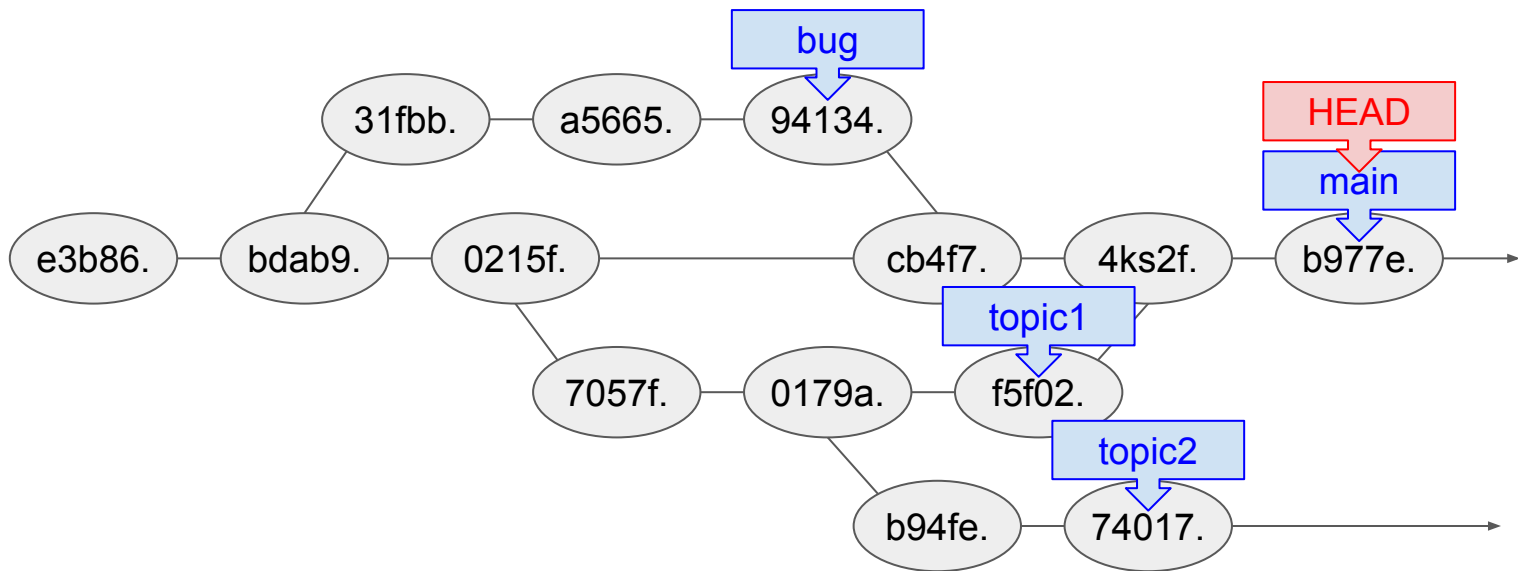
- 特定のコミットを参照し、別名を付けたもの
- 実はHEADはブランチを指している
- mainブランチでコミットすると、mainブランチとHEADは新しいコミットを追跡する



バージョン管理システム講座

ブランチ

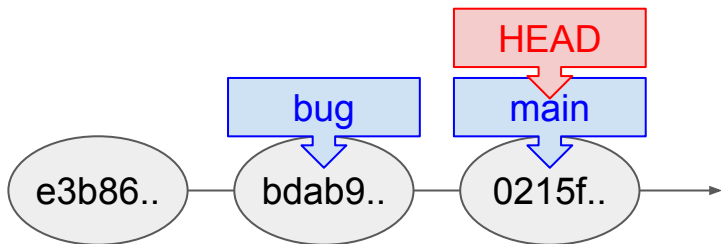
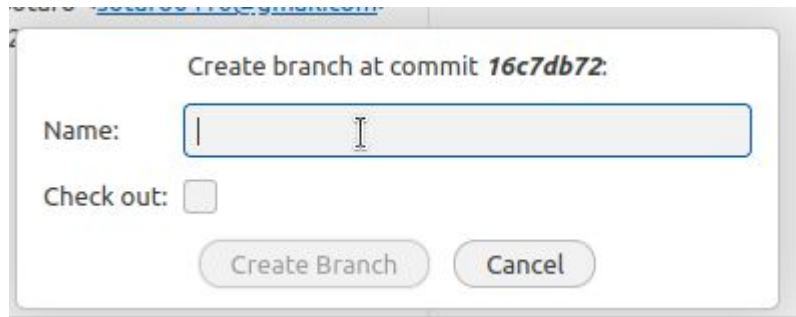
- 一般的に、幹のように並行にバージョンを管理するために使われる



バージョン管理システム講座

ブランチの作成

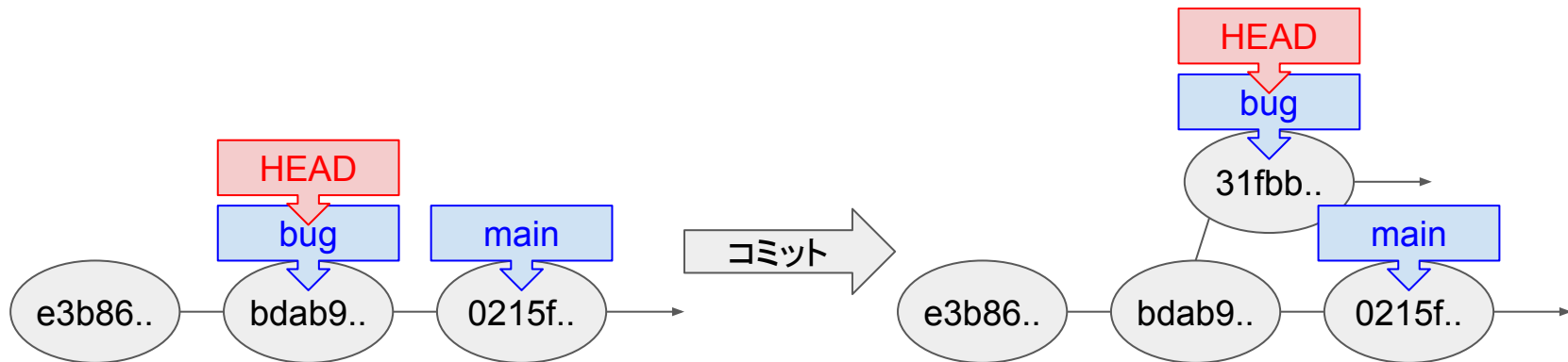
1. 分岐基となるコミットを選択
2. 右クリックし、「Create Branch...」をクリック
3. 「Name:」にブランチ名を記入し、「Create Branch」

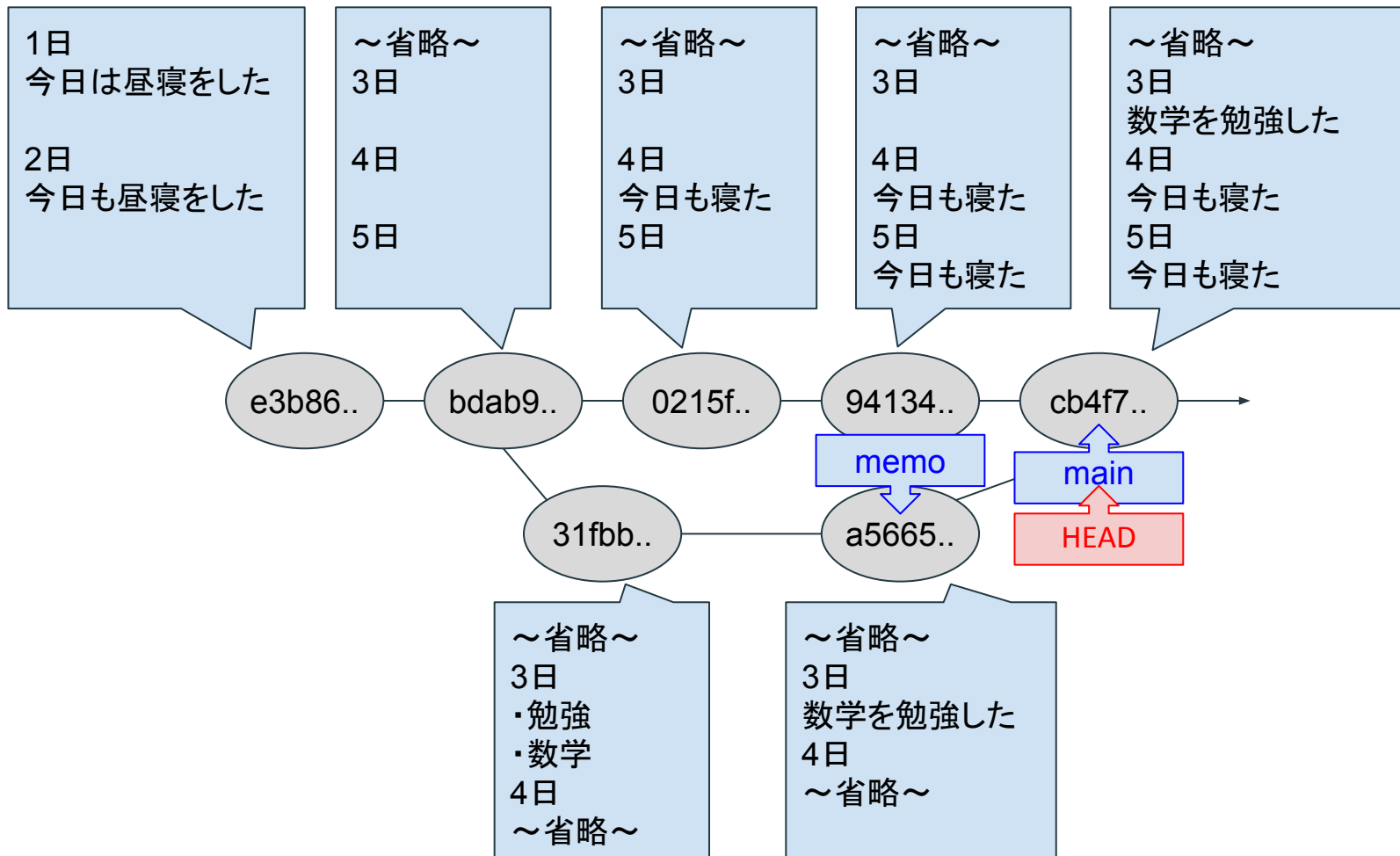


バージョン管理システム講座

ブランチの切り替え(チェックアウト)

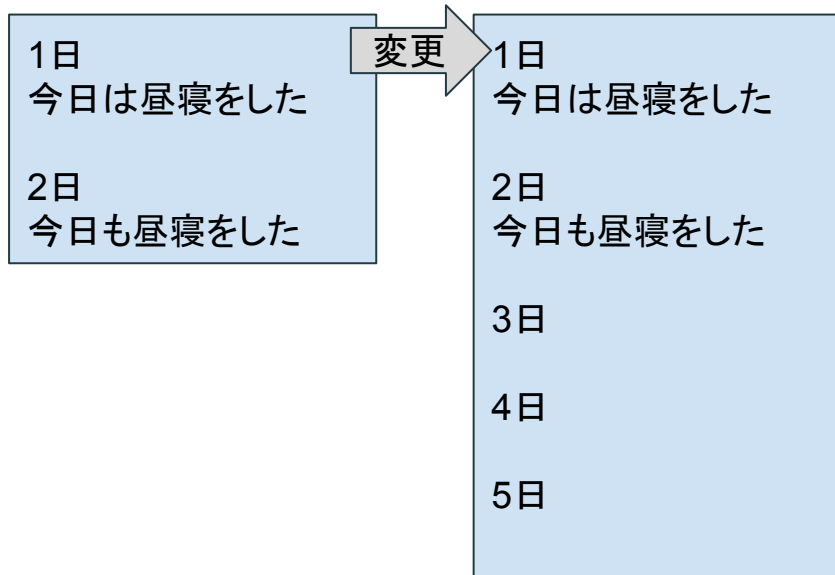
1. 「Git Graph」に表示されているブランチ名をダブルクリック
1. ステータスバー(下のバー)に表示されているブランチ名をクリックし変更

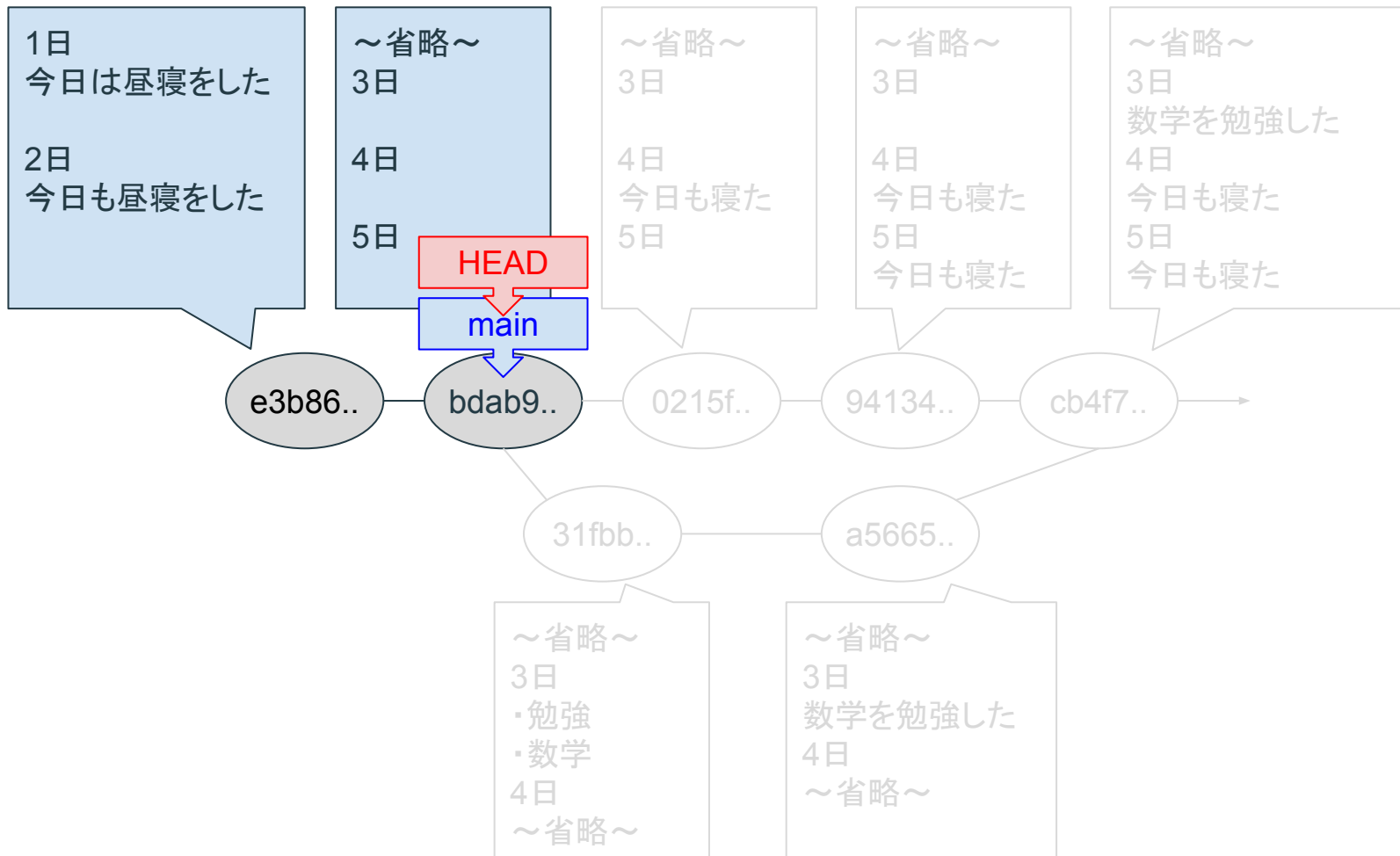




課題4-6

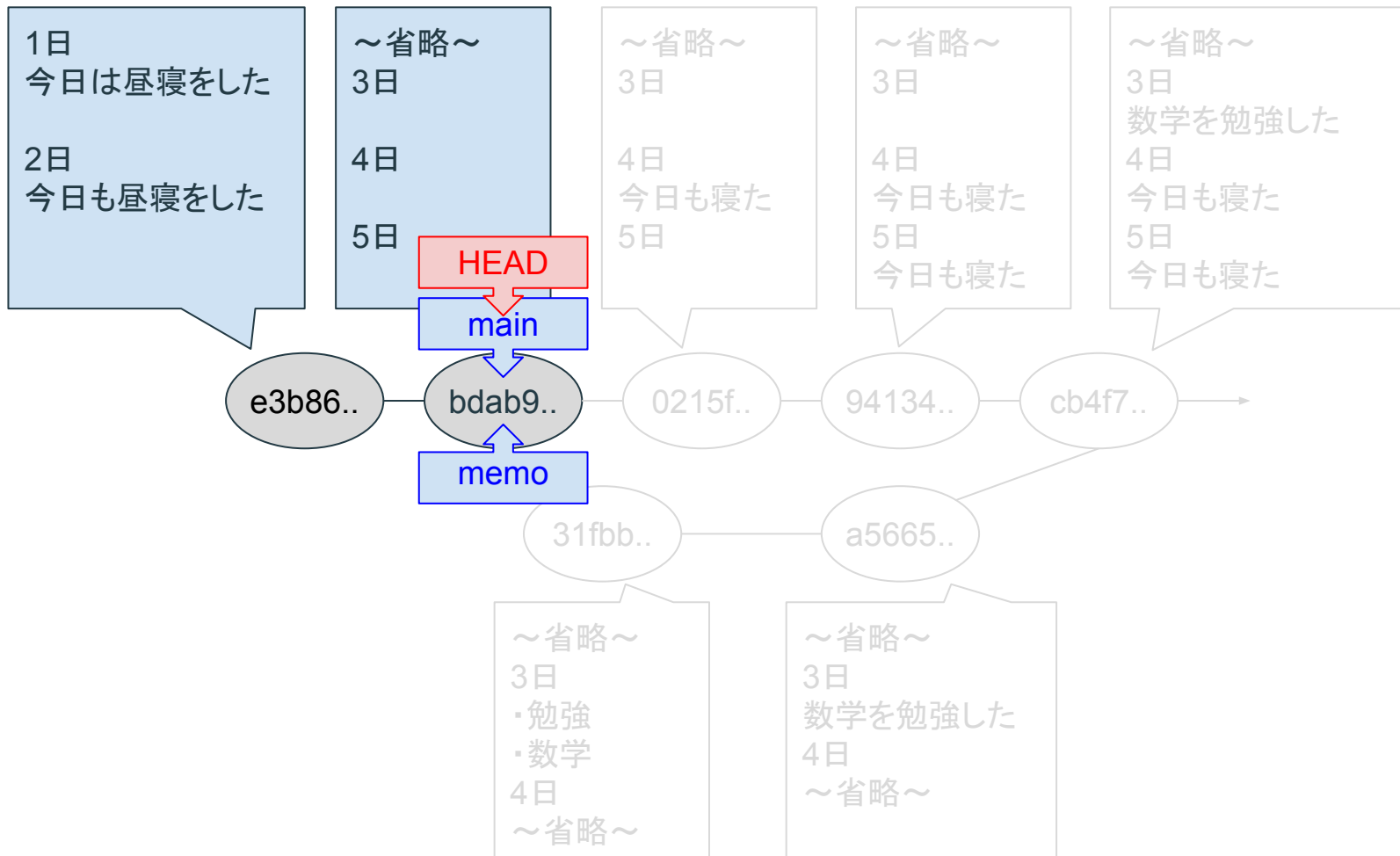
1) 「April.txt」を右側のように変更し、コミットなさい





課題4-7

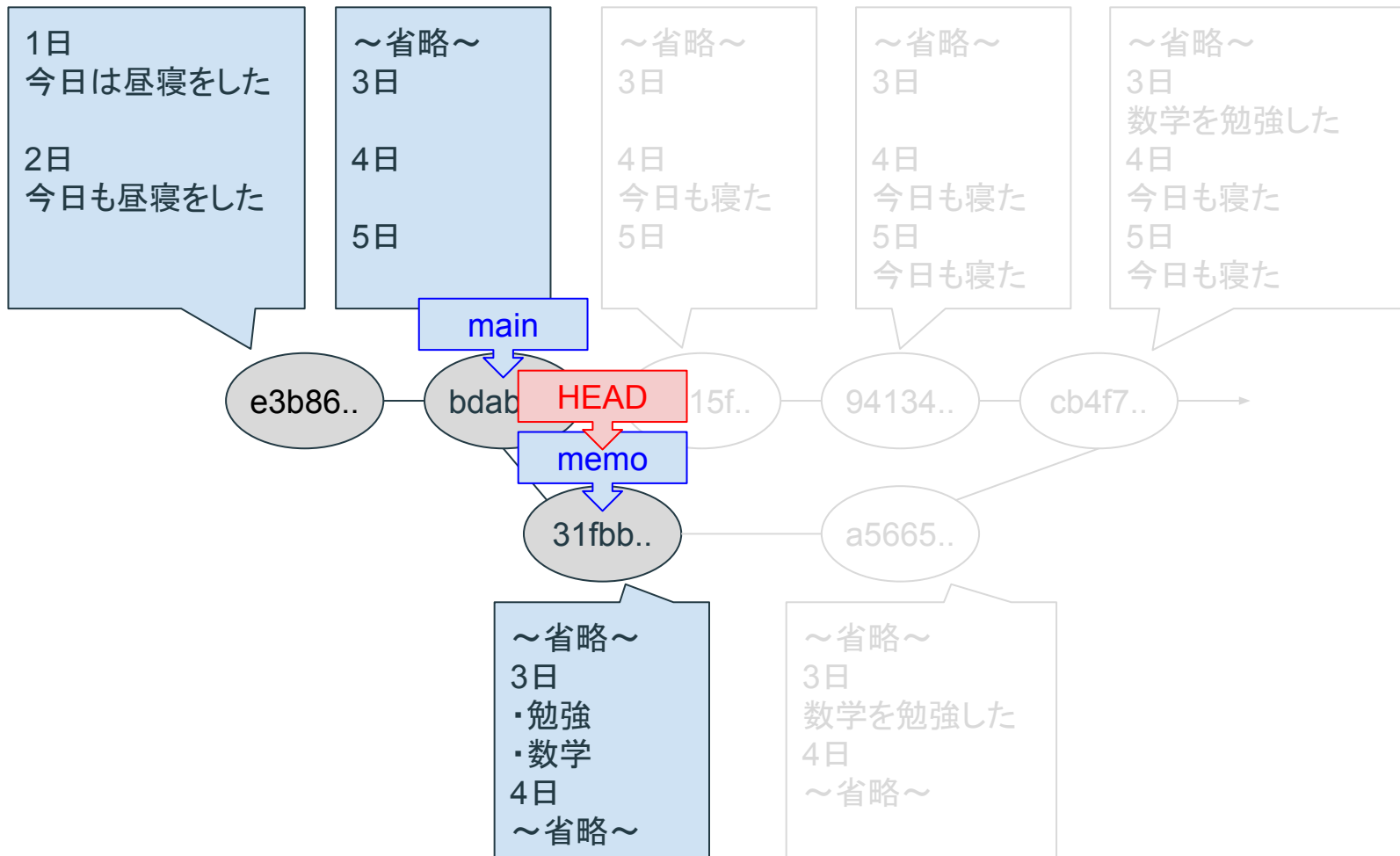
- 1) 現在のブランチを確認しなさい
 - a) ステータスバーを確認
- 2) 今日は4月4日である。4月3日の日記をつけ忘れたので、別の「memo」というブランチを作成しなさい
 - a) 分岐基となるコミットを選択
 - b) 右クリックし、「Create Branch...」をクリック
 - c) 「Name:」に「memo」を記入し、「Create Branch」をクリック



課題4-8

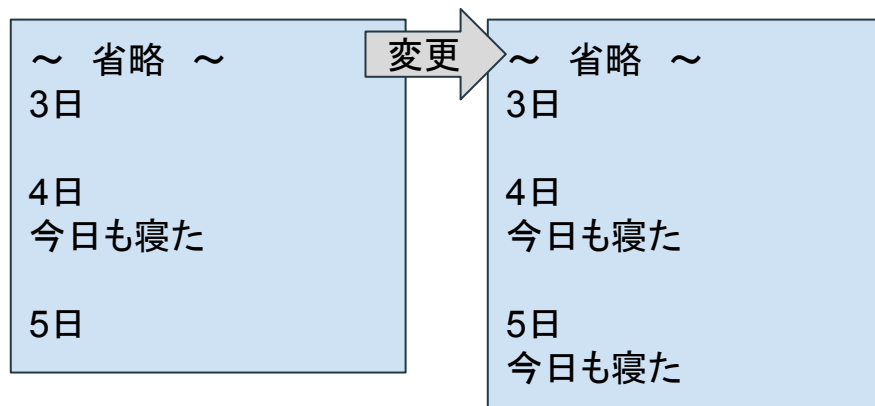
- 1) 「memo」ブランチで3日のメモを箇条書きで書き加え、コミットしなさい
 - a) ステータスバー(下のバー)に表示されているブランチ名をクリックし「memo」ブランチに変更
 - b) ファイルを編集
 - c) 「April.txt」をインデックスに追加し、コミットを行う

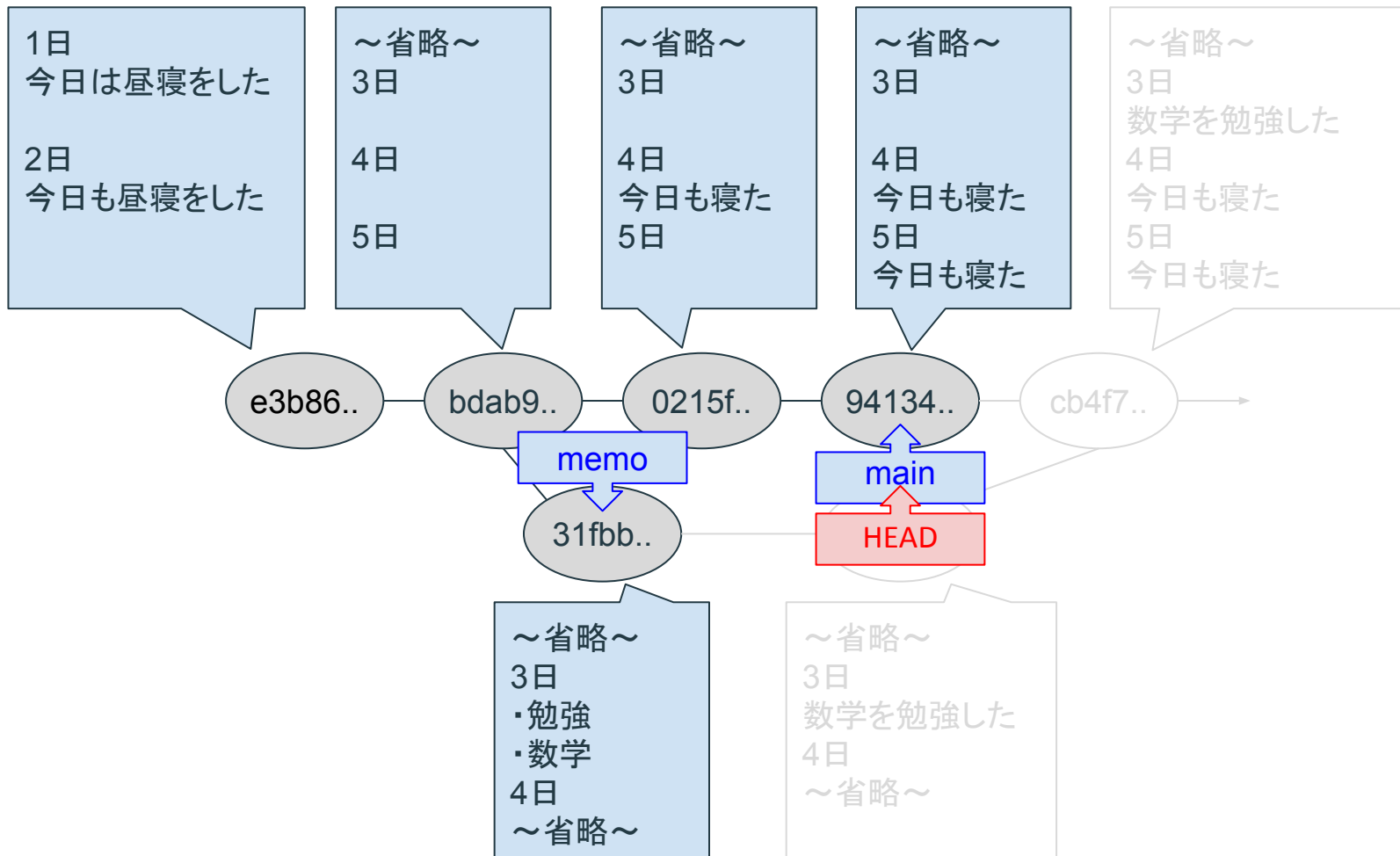
～ 省略 ～
3日
・勉強
・数学
4日
～ 省略 ～



課題4-9

- 1) 「main」ブランチに切り替え、4日の日記を書き加え、コミットしなさい
 - a) ステータスバー(下のバー)に表示されているブランチ名をクリックし「main」ブランチに変更
 - b) ファイルを編集
 - c) 「April.txt」をインデックスに追加し、コミットを行う
- 2) 「main」ブランチに5日の日記を書き加え、コミットしなさい

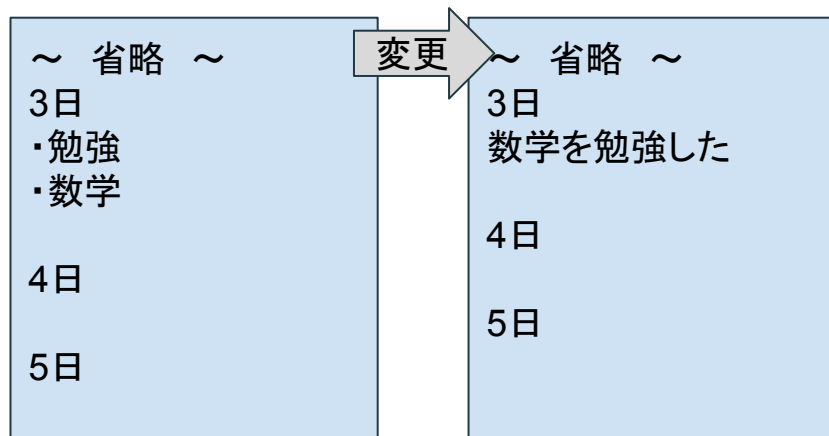


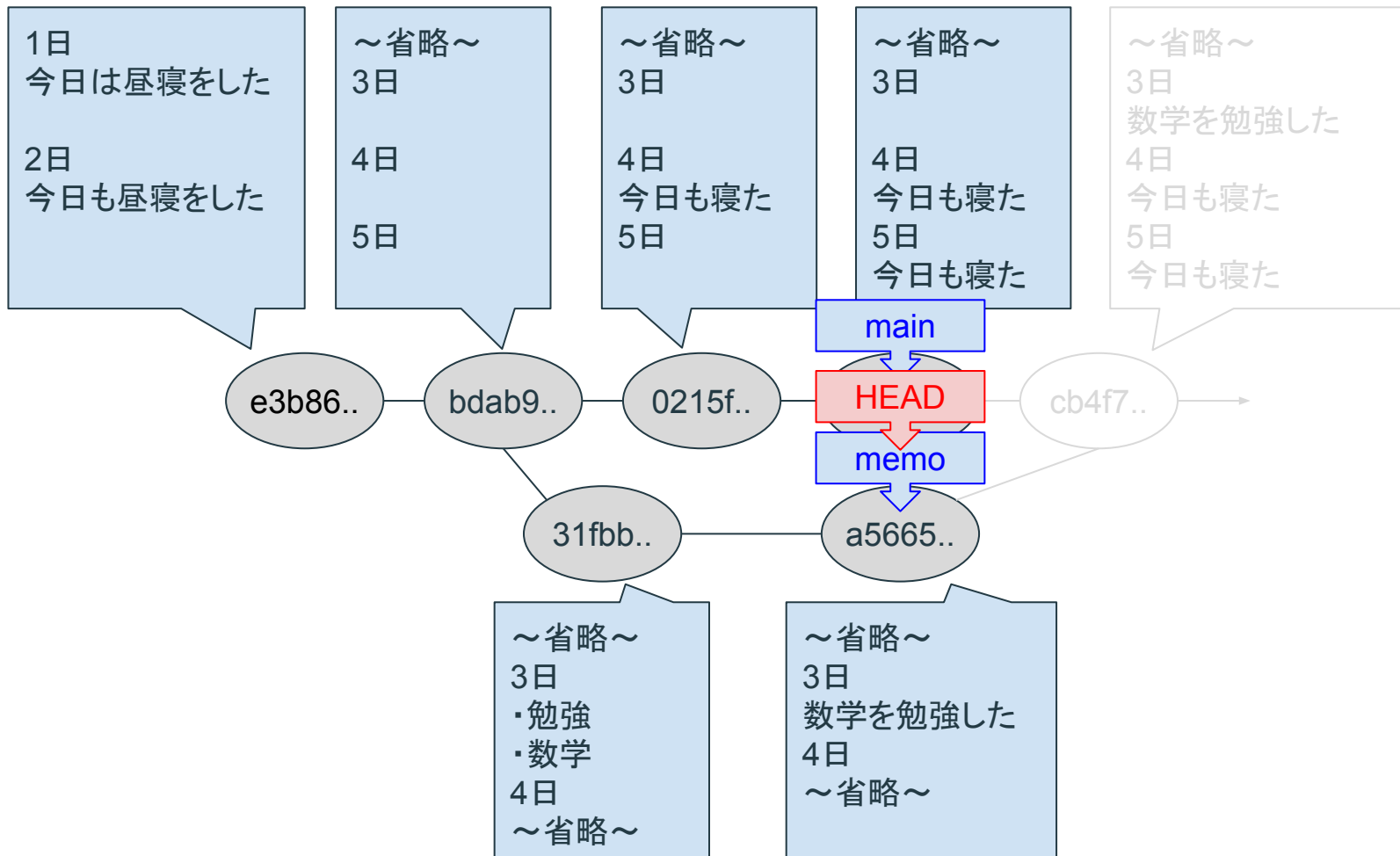


課題4-10

1) 「memo」ブランチに切り替え、3日のメモから文章を作成し、コミットしなさい

- a)
- b)
- c)
- d)
- e)

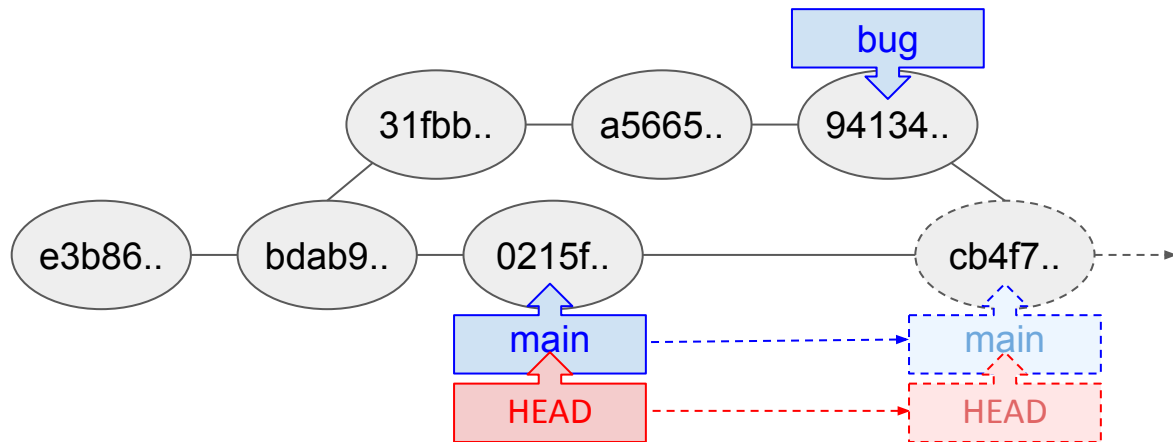




バージョン管理システム講座

ブランチの結合(マージ)

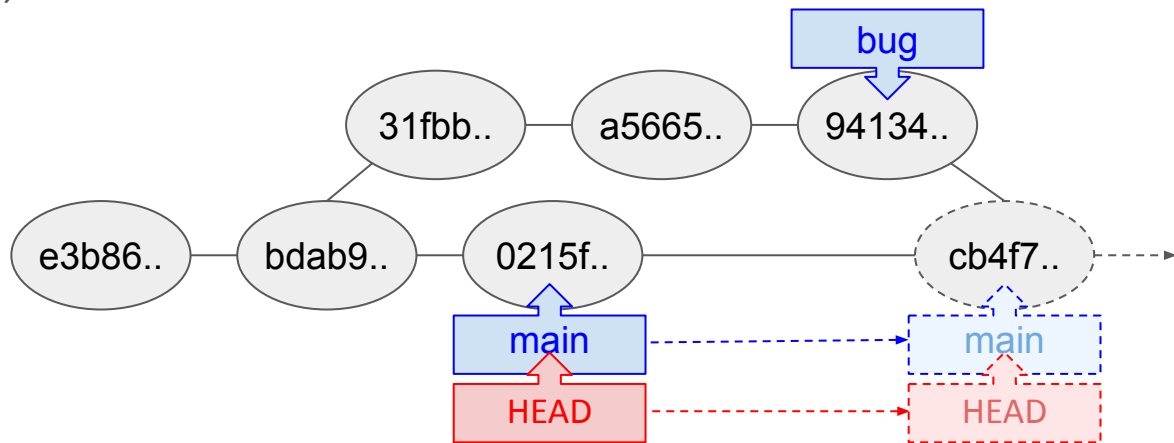
- 別のブランチの変更を取り込むこと
- 変更を取り込む側のブランチにチェックアウトしている必要がある
- マージにより作成されるコミット(マージコミット)の親コミットは2つ存在する
- 競合が発生する場合がある



バージョン管理システム講座

ブランチの結合(マージ)

1. ブランチの内容を取り込む側にチェックアウトする
2. マージしたいコミットを選択
3. 右クリックし、「Merge into current branch」をクリックする
4. マージのモードを選択し「Yes merge」をクリック(マージのモードは次の資料に記す)



バージョン管理システム講座

マージのモード

項目	意味
チェックなし	可能ならマージコミットを作らない (fast-forwardマージ)
Create a new commit ...	マージコミットを作成 (non-fast-forwardマージ)
Squash Commits	squashマージコミットを作成する
No Commit	マージコミットを作成する手前で、マージ処理を中断する

Are you sure you want to merge commit **96392bf1** into *main*
(the current branch)?

☒ Create a new commit even if fast-forward is possible

☐ Squash Commits ⓘ

☐ No Commit ⓘ

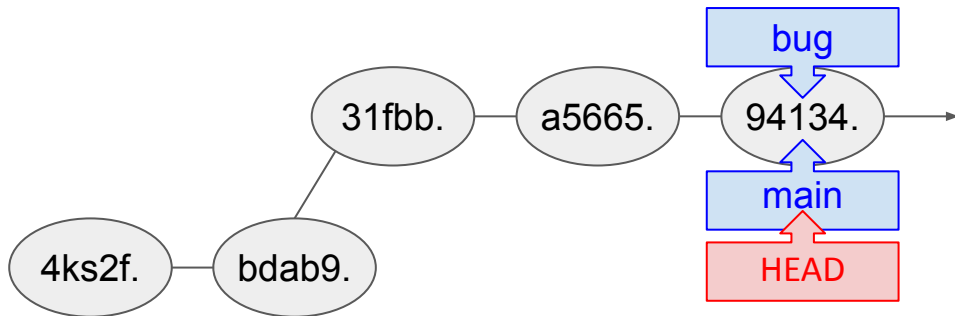
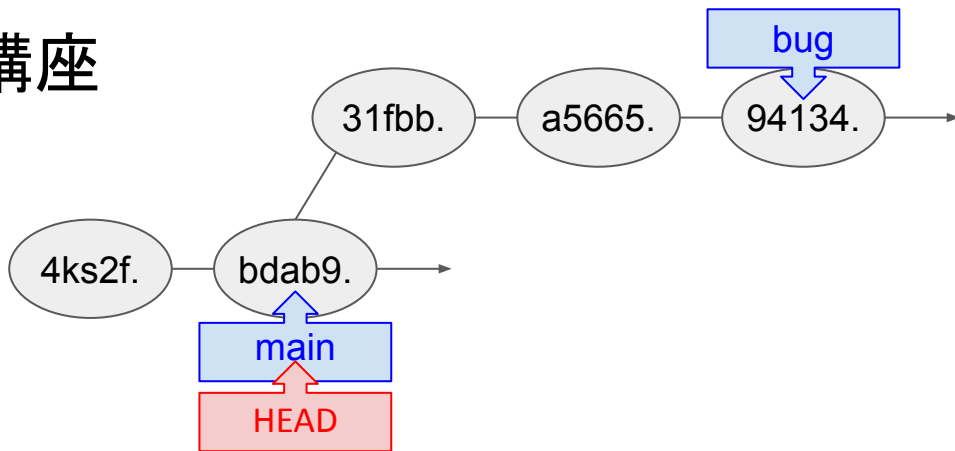
Yes, merge

Cancel

バージョン管理システム講座

fast-forwardマージ

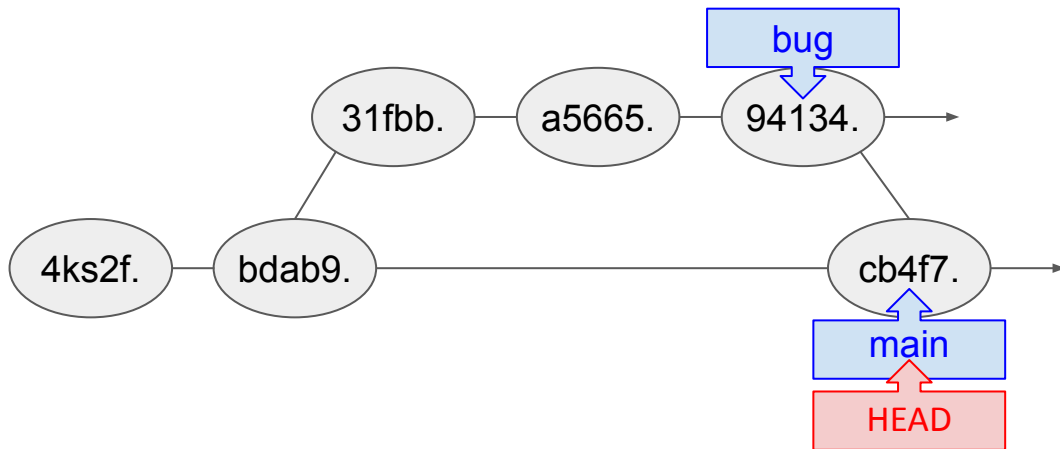
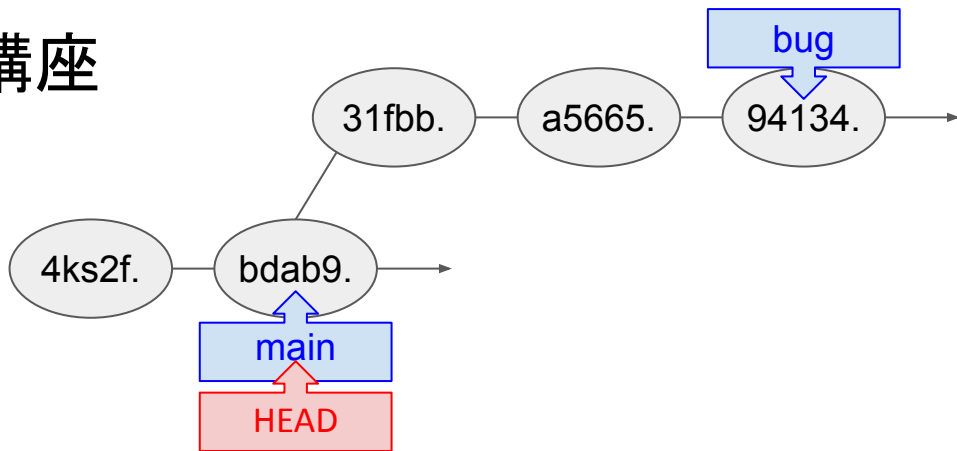
- マージされる側(main)のブランチを、マージする側(bug)のブランチに移動する
- Gitのログが一直線になる



バージョン管理システム講座

non-fast-forwardマージ

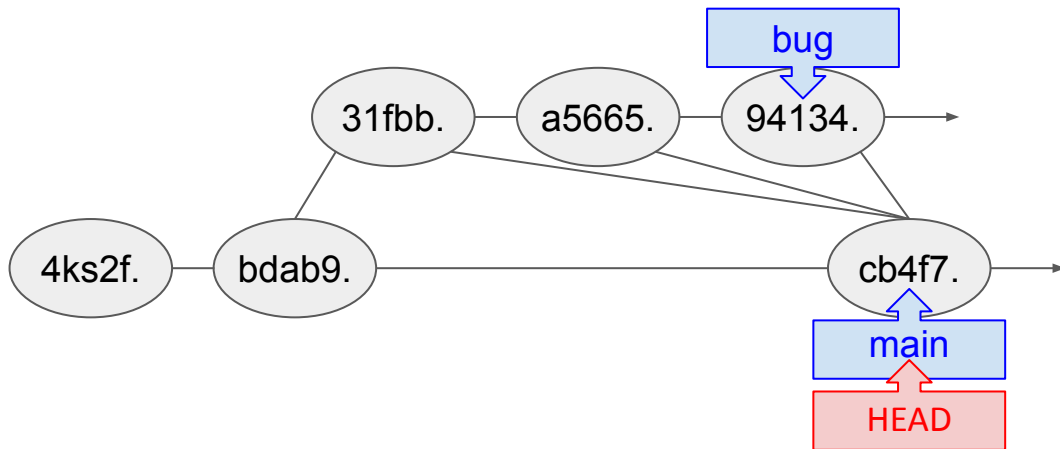
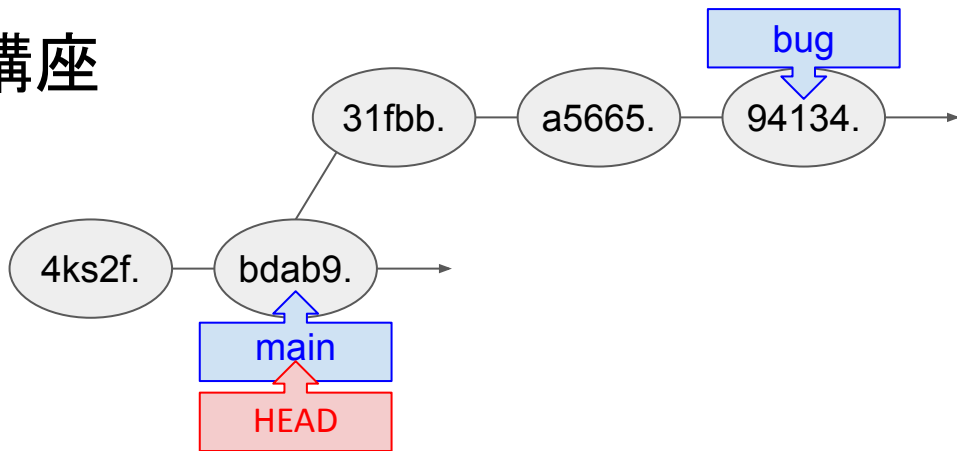
- マージコミットを作成する
- マージする側のブランチ(bug)のコミットが全て残る



バージョン管理システム講座

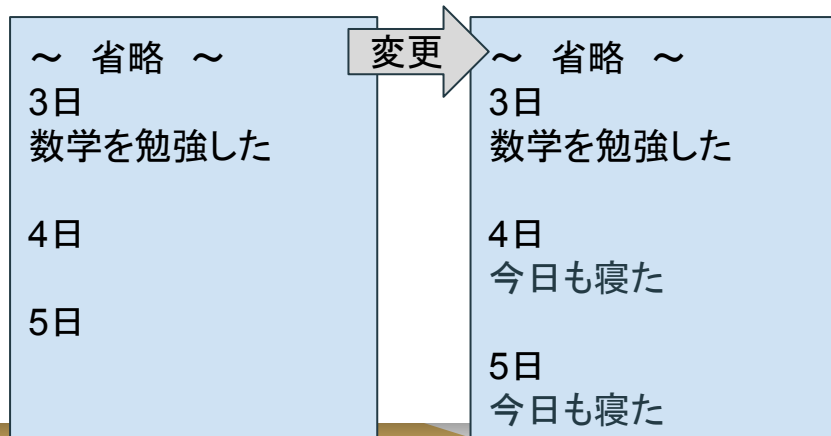
squashマージ

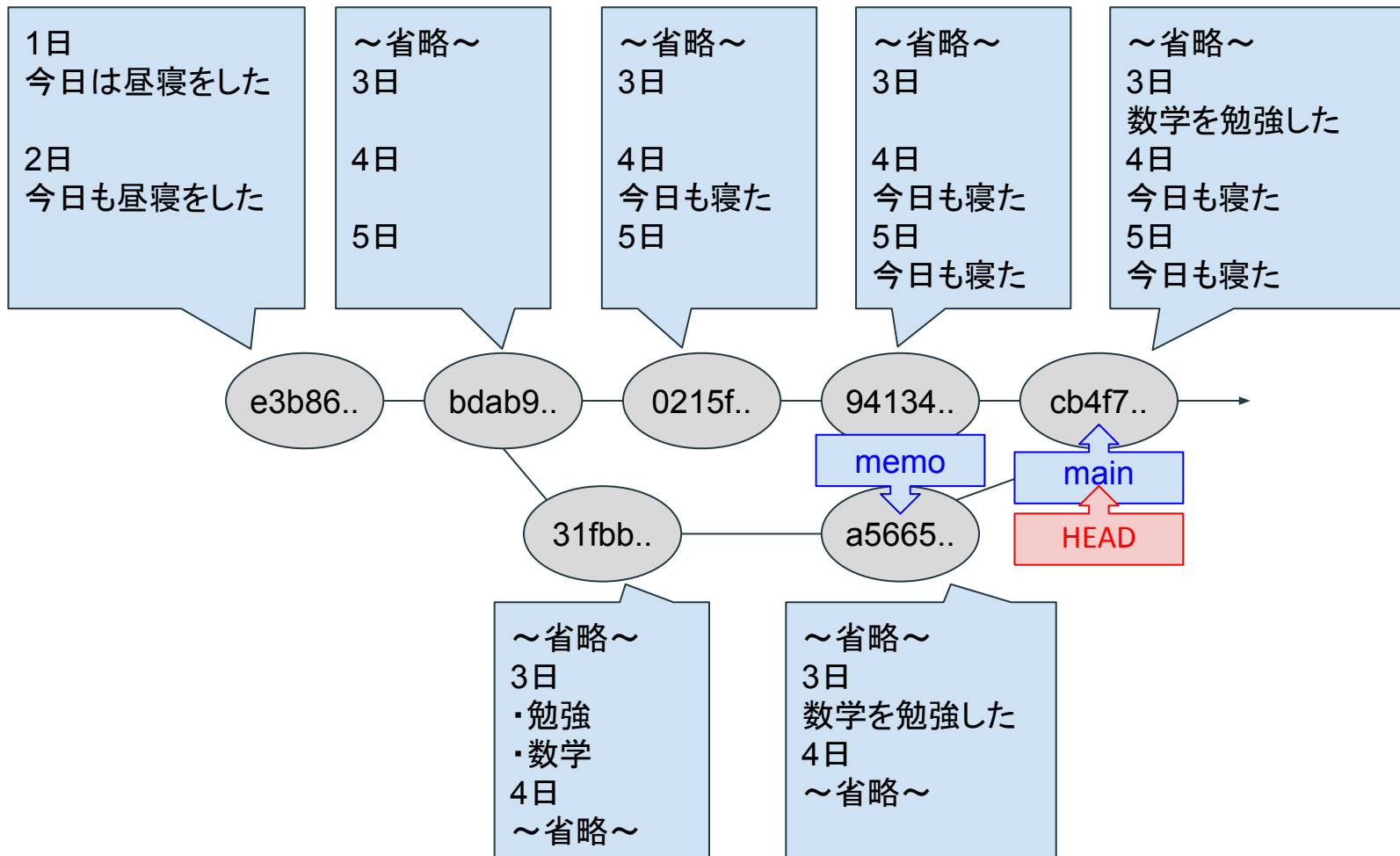
- コミットを一つ作る
- マージする側のブランチ(bug)のコミットが残らない
- Gitのログが一直線になる



課題4-11

- 1) 「main」ブランチに「memo」をマージしなさい
 - a) ステータスバー(下のバー)に表示されているブランチ名をクリックし「main」ブランチに変更
 - b) マージしたいコミットを選択
 - c) 右クリックし、「Merge into current branch」をクリックする
 - d) マージのモードを選択し「Yes merge」をクリック
- 2) マージした履歴を確認しなさい
 - a) 「Git Graph」で確認

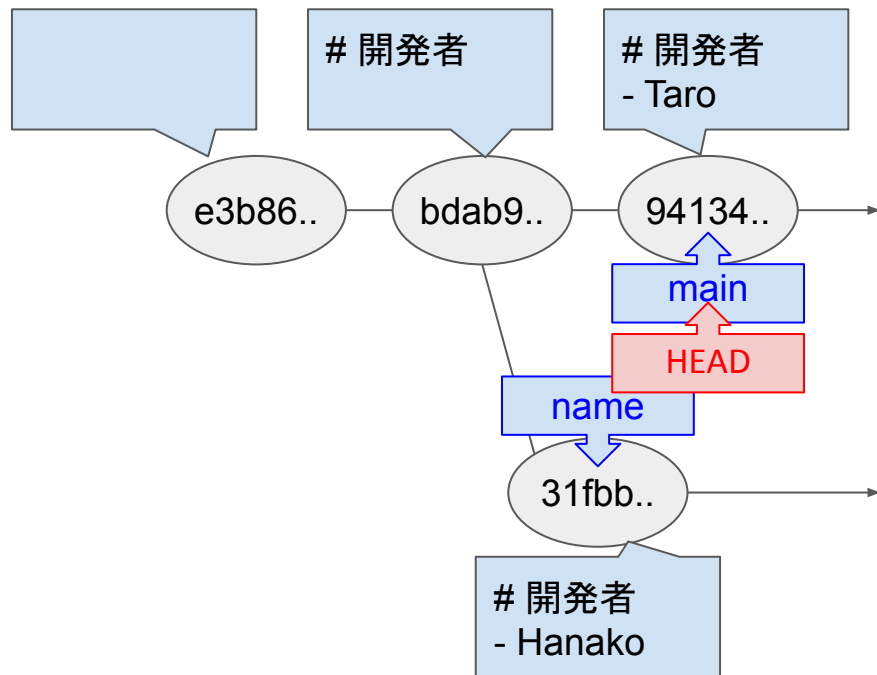




バージョン管理システム講座

競合

- マージをする時、それぞれのブランチで同じ箇所を変更している場合に発生する
- マージ時に競合が発生すると、マージが一時中断され、競合を解決する必要がある



バージョン管理システム講座

競合

1. 競合が発生したファイルを開き、どちらの変更を受け入れるかを選択する
2. コミットしなおし、マージコミットを作成する



```
C main.c > main(void)
1  #include <stdio.h>
2
3  int main(void){
  現在の変更を取り込む | 入力側の変更を取り込む | 両方の変更を取り込む | 変更の比較
4  <<<<<<< HEAD (現在の変更)
   // Hello world を表示
5
6  =====
7  // 文字列を表示
8  >>>>>>> comment (入力側の変更)
   printf("Hello World");
9
10 }
```

マージエディターで解決

バージョン管理システム講座

競合の解決方法

1. 「マージエディターで解決」から競合を解決してもよい
2. コミットしなおし、マージコミットを作成する

main.c > ...

受信中 φ b7de114 • comment

1 #include <stdio.h>

2

3 int main(void){

受信中を適用する | 組み合わせを受け入れる (受信中 First) | 無視する

4 // 文字列を表示

5 printf("Hello World");

6 }

現在のマシン φ a9f54fa • main

1 #include <stdio.h>

2

3 int main(void){

現在のマシンを適用する | 組み合わせを受け入れる (現在のマシン First)

4 // Hello world を表示

5 printf("Hello World");

6 }

結果 main.c

1 #include <stdio.h>

2

3 int main(void){

変更は承諾されませんでした

4 printf("Hello World");

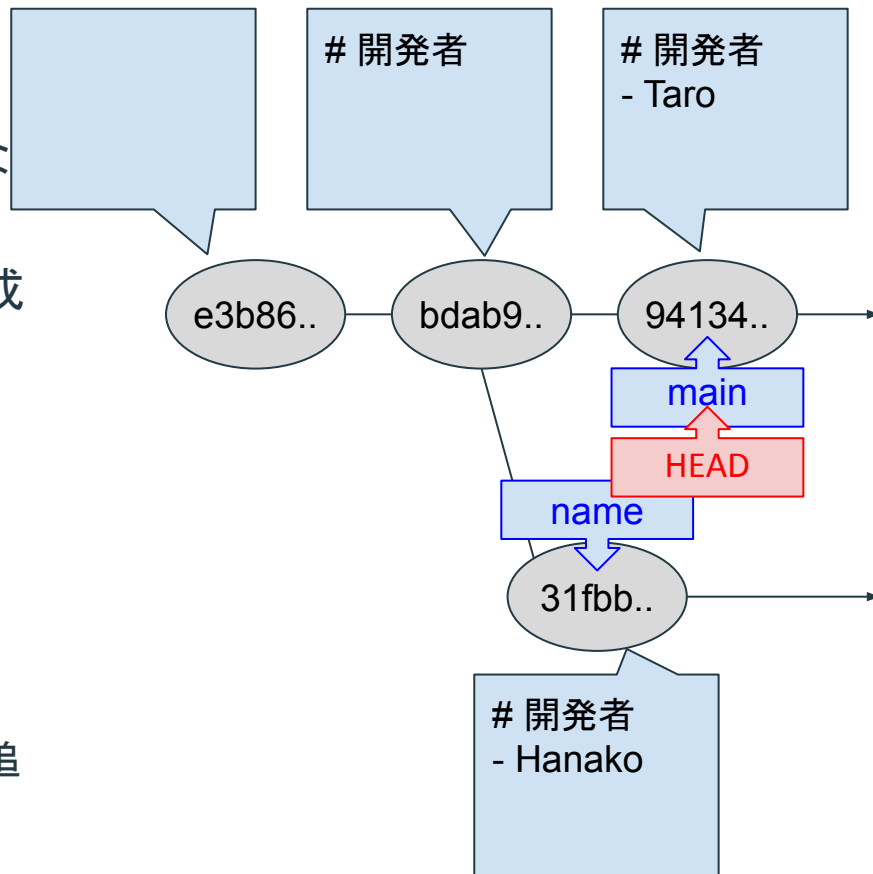
5 }

1 個の残りの競合 ...

課題4-12

1) 左のようなGitのログになるように、変更しなさい。
ただし、青枠は新しく作成した「Author.md」というファイルを表している。

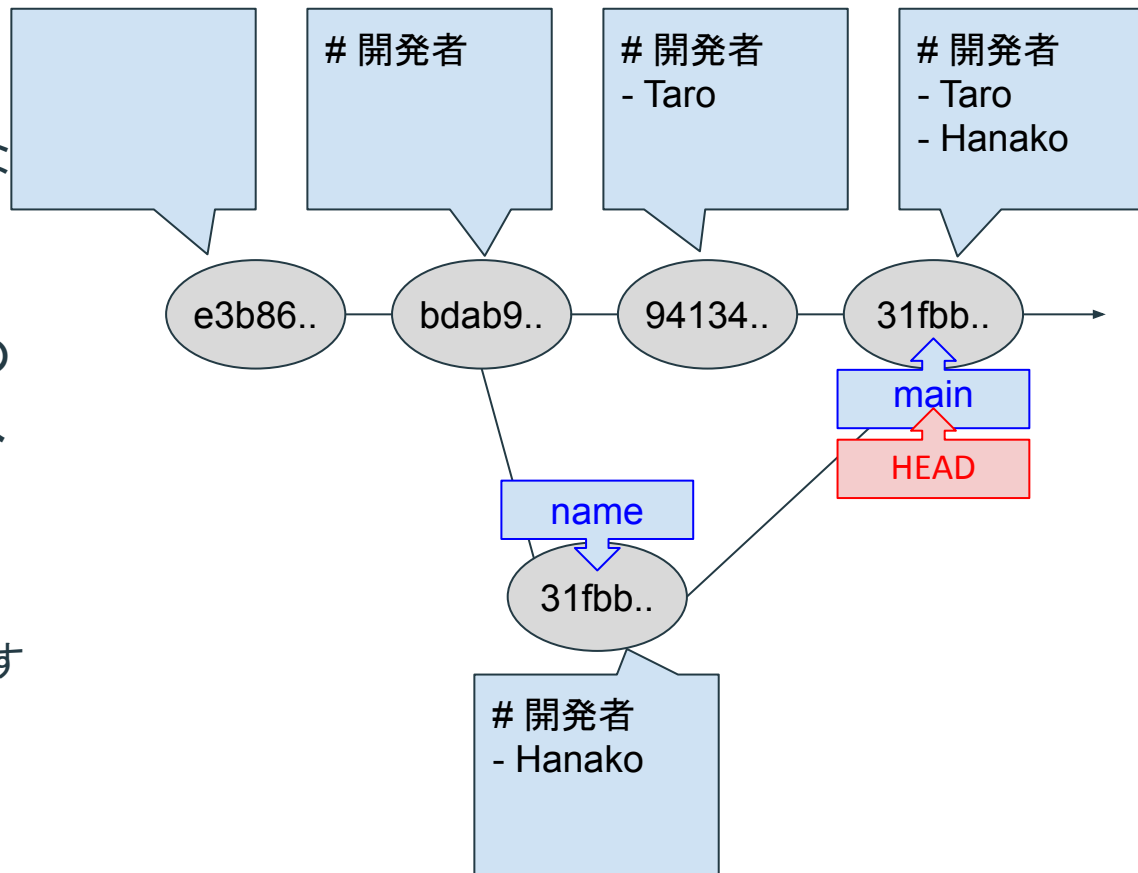
- Author.mdを作成
- コミットする
- nameブランチ作成
- nameブランチで名前を追加し、コミット
- mainブランチで名前を追加し、コミット



課題4-13

1) 左のようなGitのログになるように、変更しなさい。
ただし、マージ時に競合が発生した場合、両方のブランチの変更を受け入れなさい

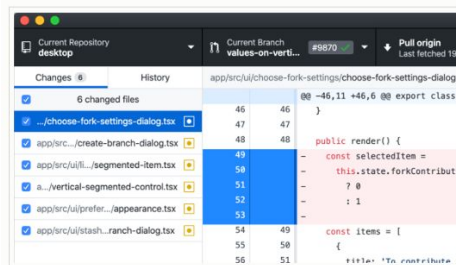
- mainブランチにチェックアウト
- nameブランチをマージする
- 競合を解決する
- コミットする



バージョン管理システム講座

GUI Clients

- GitをGUIで操作するためのソフトをGUI Clientsと呼ばれている
- GitそのものはCUIのソフト
- VSCodeのGitはGUIで操作できるが、内部でコマンドが呼び出されている

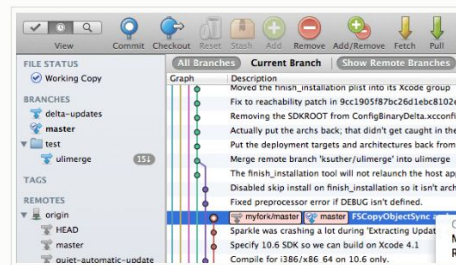


GitHub Desktop

Platforms: Mac, Windows

Price: Free

License: MIT

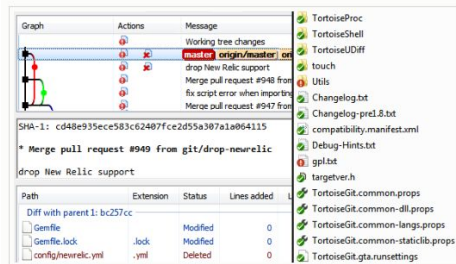


SourceTree

Platforms: Mac, Windows

Price: Free

License: Proprietary

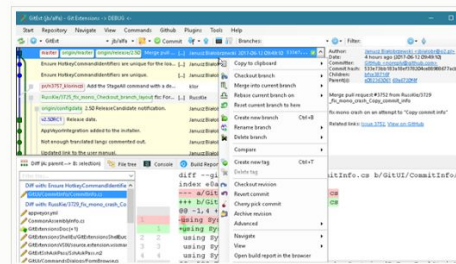


TortoiseGit

Platforms: Windows

Price: Free

License: GNU GPL



Git Extensions

Platforms: Windows

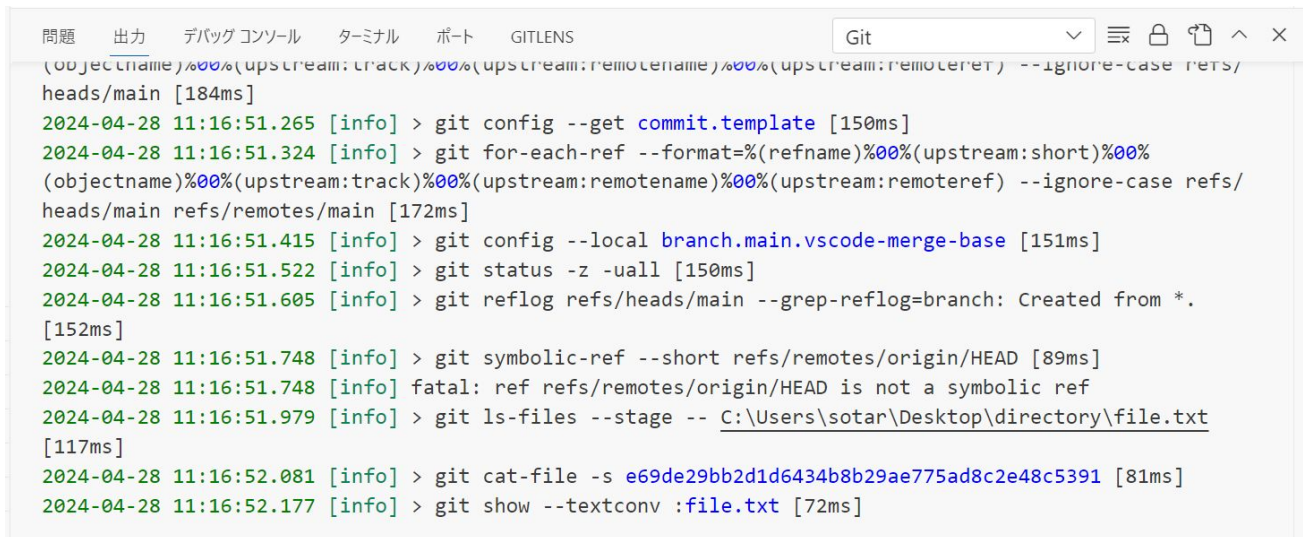
Price: Free

License: GNU GPL

バージョン管理システム講座

CUIの確認方法

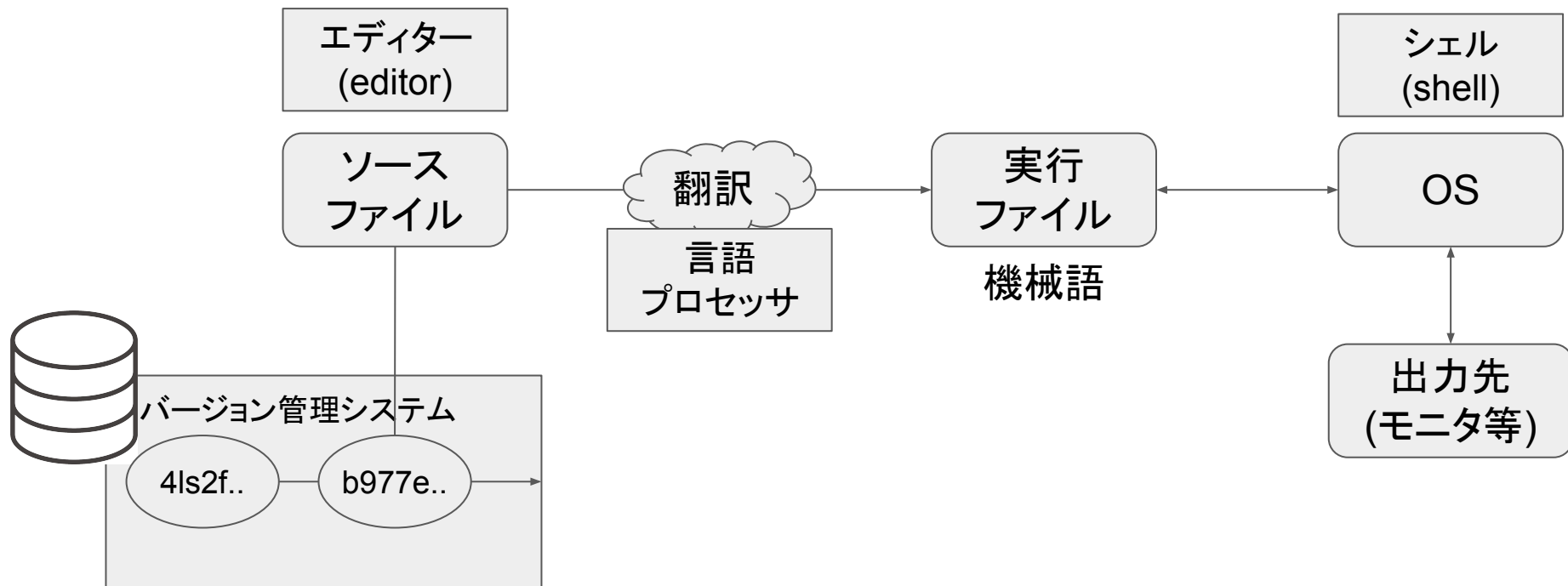
1. ターミナルを起動
2. 出力をクリック
3. 「Git」を選択



The screenshot shows a terminal window with a menu bar at the top containing '問題', '出力', 'デバッグコンソール', 'ターミナル', 'ポート', and 'GITLENS'. The '出力' (Output) tab is selected. The terminal displays a series of Git commands and their outputs, including configuration checks, status checks, and file operations. The output is color-coded, with green for informational messages and red for fatal errors.

```
(objectname)%00%(upstream:track)%00%(upstream:remotename)%00%(upstream:remote:ref) --ignore-case refs/  
heads/main [184ms]  
2024-04-28 11:16:51.265 [info] > git config --get commit.template [150ms]  
2024-04-28 11:16:51.324 [info] > git for-each-ref --format=%(refname)%00%(upstream:short)%00%  
(objectname)%00%(upstream:track)%00%(upstream:remotename)%00%(upstream:remote:ref) --ignore-case refs/  
heads/main refs/remotes/main [172ms]  
2024-04-28 11:16:51.415 [info] > git config --local branch.main.vscode-merge-base [151ms]  
2024-04-28 11:16:51.522 [info] > git status -z -uall [150ms]  
2024-04-28 11:16:51.605 [info] > git reflog refs/heads/main --grep-reflog=branch: Created from *.  
[152ms]  
2024-04-28 11:16:51.748 [info] > git symbolic-ref --short refs/remotes/origin/HEAD [89ms]  
2024-04-28 11:16:51.748 [info] fatal: ref refs/remotes/origin/HEAD is not a symbolic ref  
2024-04-28 11:16:51.979 [info] > git ls-files --stage -- C:\Users\sotar\Desktop\directory\file.txt  
[117ms]  
2024-04-28 11:16:52.081 [info] > git cat-file -s e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 [81ms]  
2024-04-28 11:16:52.177 [info] > git show --textconv :file.txt [72ms]
```

個人開発での一般的な開発環境



統合開発環境(IDE)

統合開発環境(IDE)とは

- プログラムを開発するのに必要な機能を全て含めたソフト
- エディタ、シェル、バージョン管理ツール等が1つのソフトで利用できる
- Visual Studio CodeもカスタマイズすればIDEと言える機能がたくさん存在する
 - シェルの呼び出し
 - Gitの可視化ツール (Git Graph)
 - プログラムの実行、デバッグ

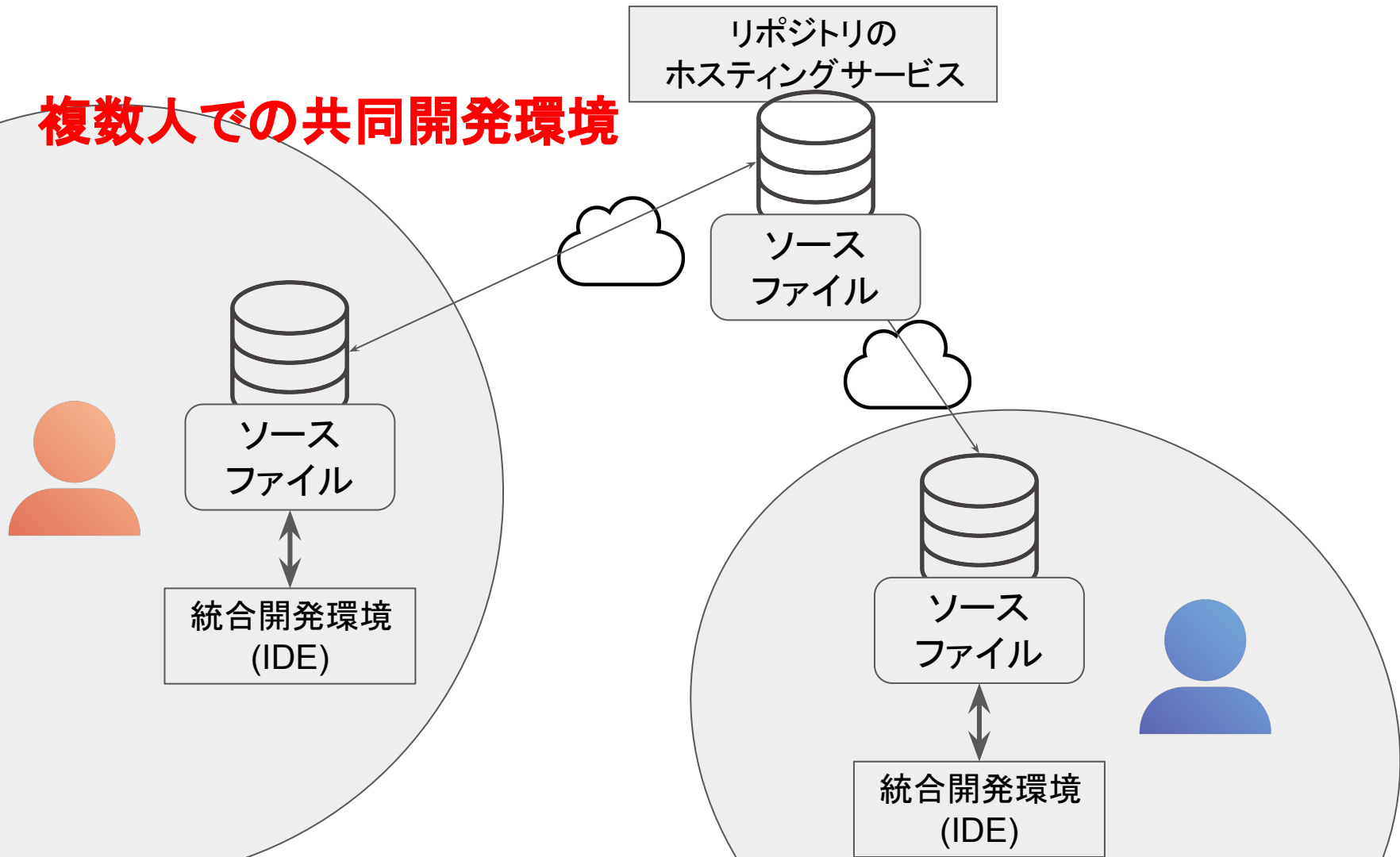
有名な統合開発環境(IDE)

- Xcode
- Unity
- Visual Studio

複数人での共同開発環境

- どうやってソースコードを共有するのか？
- どうやってソースコードを共同で編集するのか？
- どうやってコミュニケーションを取るのか？

複数人での共同開発環境



リポジトリのホスティングサービス

リポジトリのホスティングサービスとは

- バージョン管理システムで使ったリポジトリをインターネット上など、複数のユーザがアクセス可能な場所に置くことができるサービスのこと
- 今回はGitHubに限定して話を進める

GitHubとは

- Gitリポジトリをインターネット上に置くことができるサービス
- 複数のユーザがGitリポジトリにアクセスできる

GitHub講座～初期設定～

- GitHubアカウントの作成
 - <https://github.com>
 - 「Sign up」をクリックし作成する
- GitにGitHubアカウントを認証させる
 - 認証用ソフトウェア「GitHub CLI」のインストール
 - Windows:「PowerShell」を開き、「winget install --id GitHub.cli」を実行
 - Mac:「Terminal」を開き、「brew install gh」を実行
 - GitにGitHubアカウントを認証させる
 - 以下のコマンドを実行し、プロンプトに従う
 - `$ gh auth login`

GitHub講座

リモートリポジトリ(≡ 共有リポジトリ)

- インターネット上などにあるリポジトリ

ローカルリポジトリ

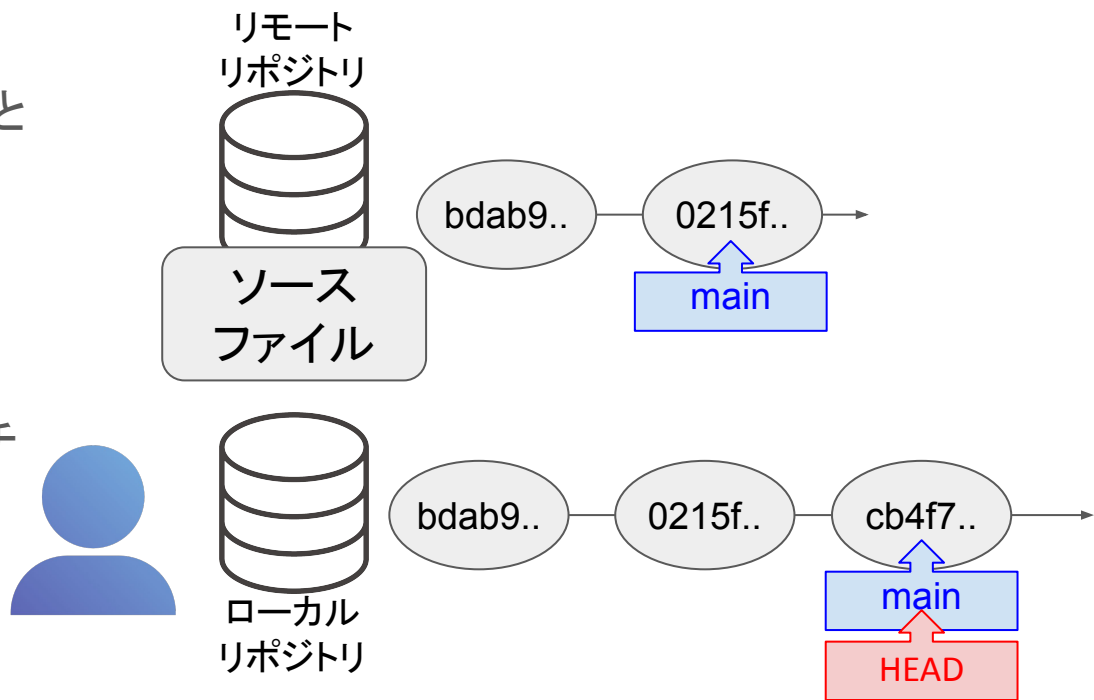
- 自分のパソコンのリポジトリのこと

リモートブランチ

- リモートリポジトリにあるブランチ

ローカルブランチ

- ローカルリポジトリにあるブランチ



GitHub講座

空のリモートリポジトリの作成方法

- GitHubにアクセスし、「NEW」をクリックする
- 「Repository name」にリポジトリの名前をつける
- 「Public」は全世界に公開される。「Private」は指定したユーザだけが見れる
- 「Create repository」で作成できる

リポジトリの編集権限の設定方法

- 「Settings」をクリックする
- 「Collaborators」をクリックする
- 「Add people」をクリックする

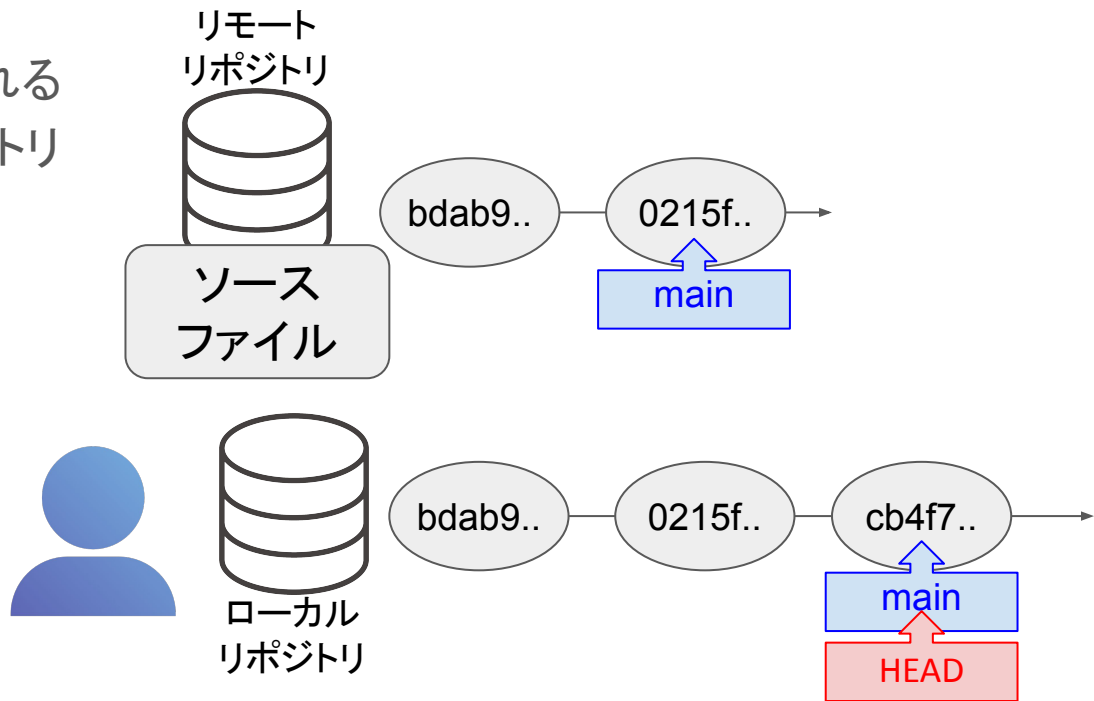
課題5-1

- 1) リモートリポジトリを以下の設定で作成なさい
 - a) Repository Name : DailyReport
 - b) Kind of Repository : Private

GitHub講座

リモートリポジトリを設定

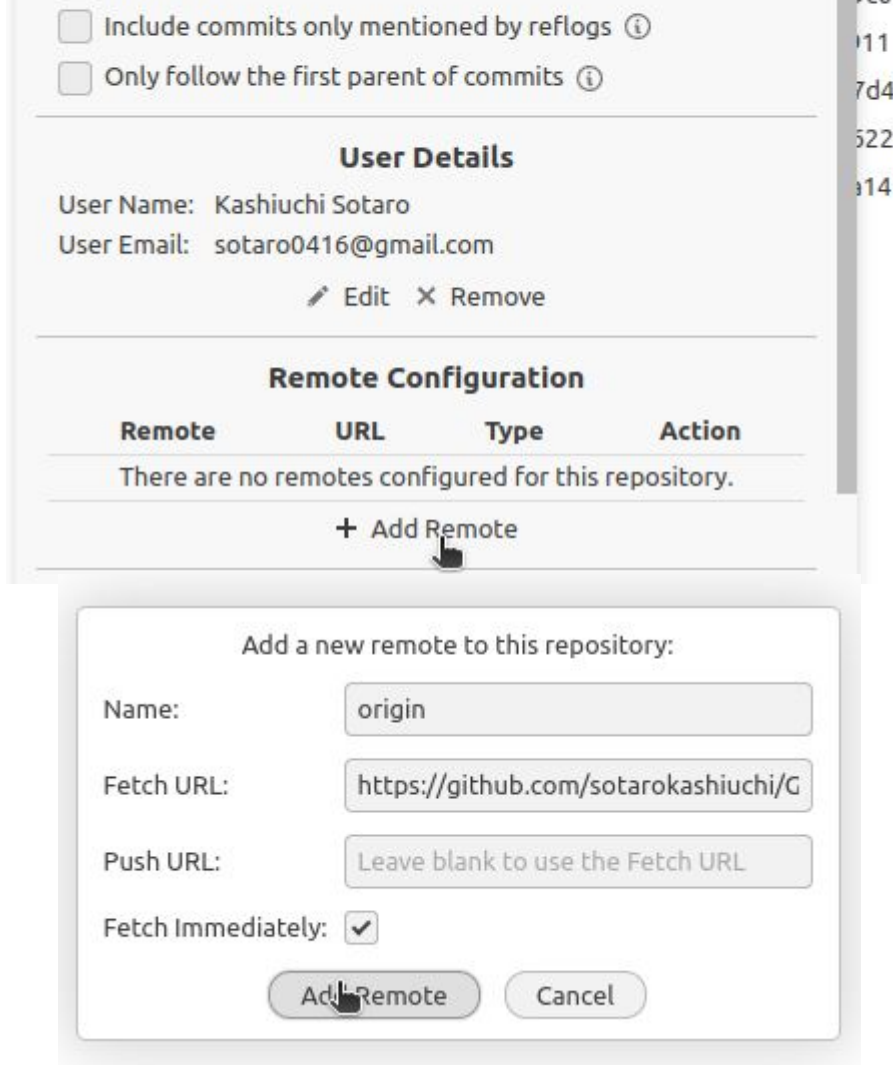
- ローカルリポジトリにリモートリポジトリを認識させる
- リモートリポジトリはURLで表される
- URLは長いので、リモートリポジトリに別名を付ける



GitHub講座

リモートリポジトリを設定

1. 「Git Graph」の右上にある「設定マーク」をクリック「Remote Configuration」の「Add Remote」をクリック
2. 「Name」と対応させたいリモートリポジトリの「URL」を入力



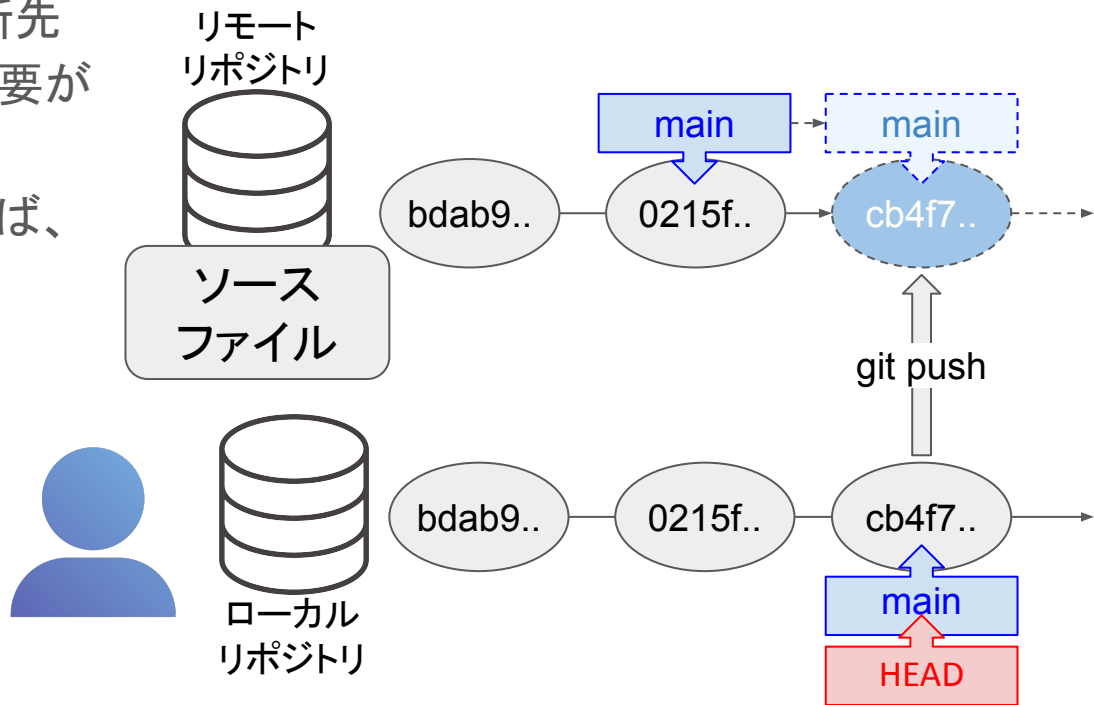
課題5-2

- 1) ローカルリポジトリにリモートリポジトリを「origin」として対応付けなさい
 - a) 「Git Graph」の右上にある「設定マーク」をクリック「Remote Configuration」の「Add Remote」をクリック
 - b) 「Name」に「origin」を、「Fetch URL」に先程作成したリモートリポジトリの「URL」を入力

GitHub講座

プッシュ

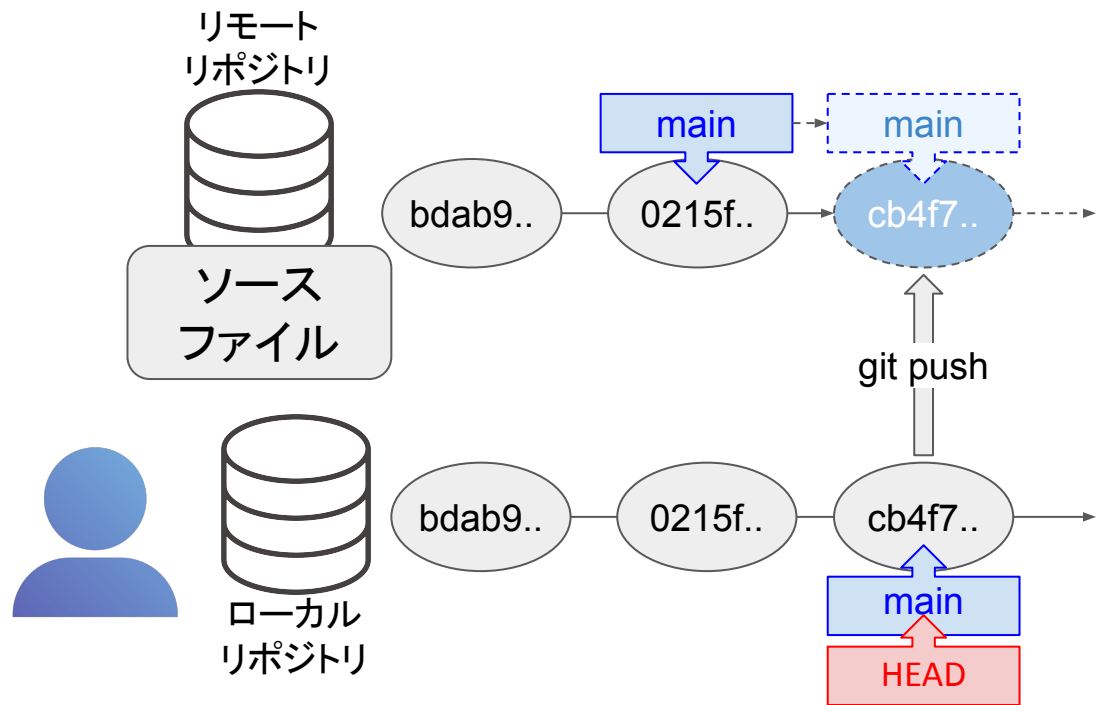
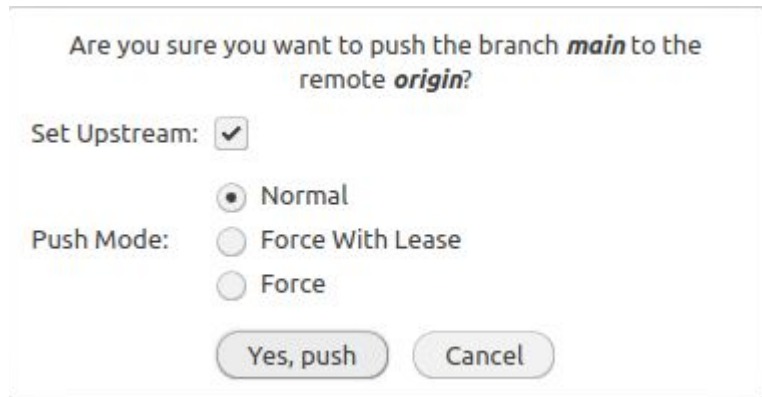
- リモートブランチを更新する
- 更新元のローカルブランチ、更新先のリモートブランチを指定する必要がある
- リモートブランチが存在しなければ、作られる



GitHub講座

プッシュ

1. 更新元のローカルブランチを右クリック
2. 「Push Branch...」をクリック
3. 「Yes,push」をクリック



課題5-3

- 1) ローカルリポジトリの「main」ブランチをリモートリポジトリにプッシュしなさい
 - a) 「main」ブランチを右クリック
 - b) 「Push Branch...」をクリック
 - c) 「Yes,push」をクリック
 - d) GitHubのリモートリポジトリのページを見てみよう！

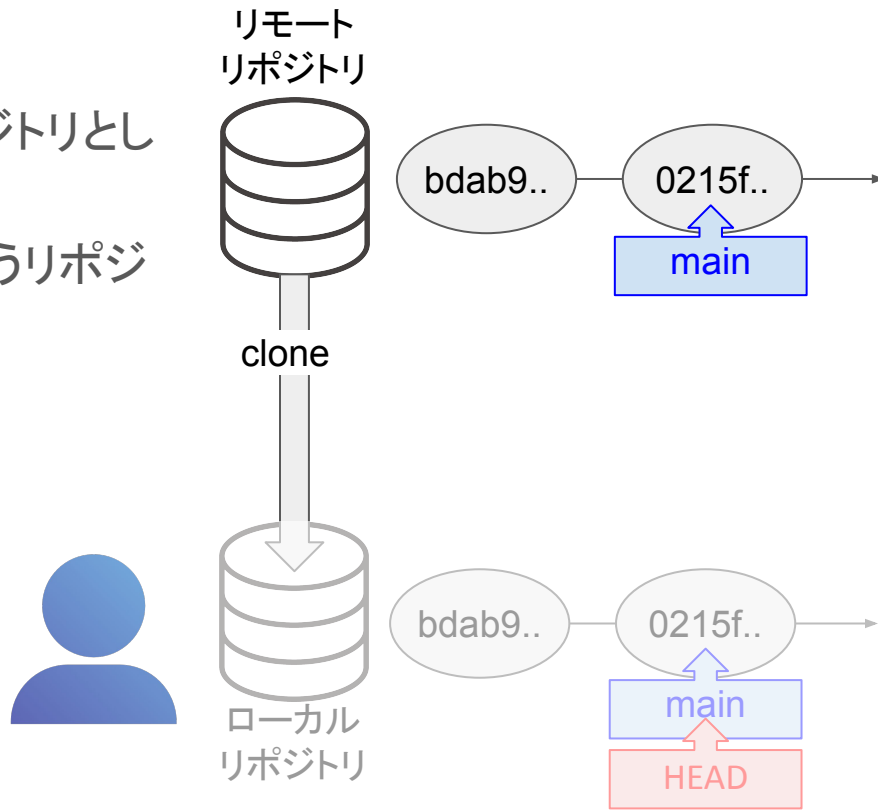
課題5-4

- 1) 「April.txt」の6日以降を自由に変更し、コミットし、リモートリポジトリに反映させなさい
 - a) <省略>
 - b) GitHubのリモートリポジトリのページを見てみよう！

GitHub講座

クローン

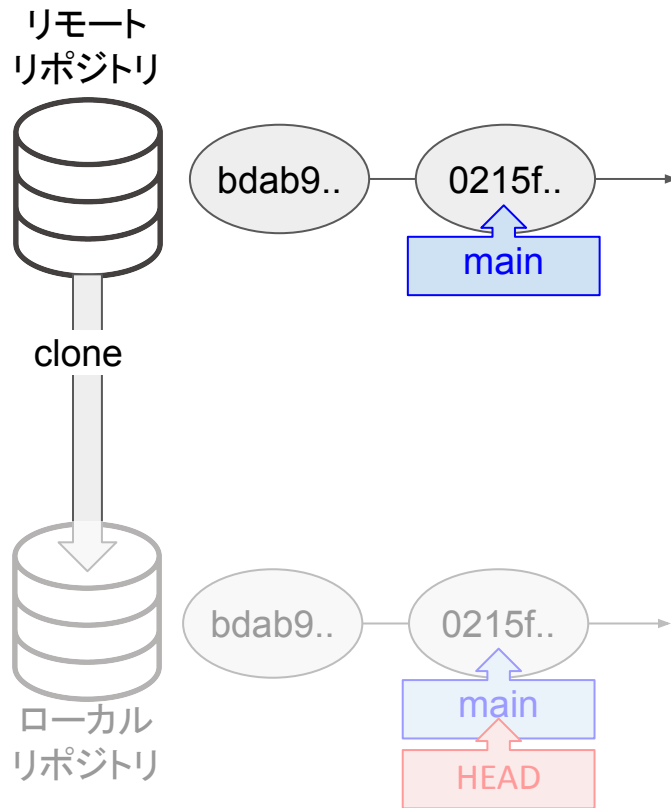
- リポジトリを複製すること
- 一般的にリモートリポジトリをローカルリポジトリとして複製する時に使用する
- クローンで複製すると、自動的にoriginというリポジトリ名が付けられる



GitHub講座

クローン

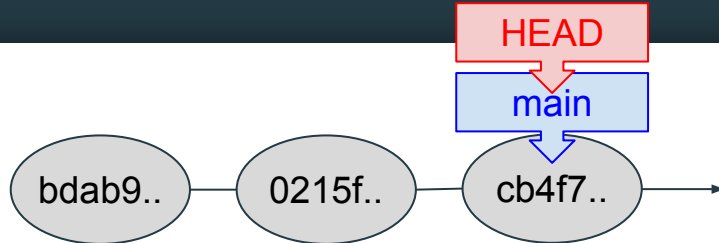
1. クローンするリモートリポジトリのURLを取得する。
(GitHubの場合、「Code」から取得できる)
2. VSCodeのコマンドパレットを開き、「Git: Clone」をクリック
3. 先程のURLを入力する
4. クローンするフォルダを指定する



課題5-5

- 1) 「RepoB」というフォルダを作成し、先程の「DailyReport」が含まれたリモトリポジトリをクローンしなさい。これをローカルリポジトリBとする
 - a) 「RepoB」というフォルダを作成する
 - b) 「DailyReport」が含まれたリモトリポジトリのクローン用リンクを取得
 - c) VSCodeのコマンドパレットを開き、「Git: Clone」をクリック
 - d) 先程のURLを入力する
 - e) クローンするフォルダを指定する
- 2) 初めの方のローカルリポジトリをローカルリポジトリAとする。ローカルリポジトリA内の「DailyReport/May.txt」ファイルを適当に変更し、コミット、プッシュしなさい

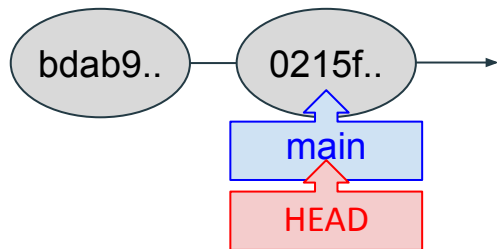
ローカル
リポジトリA



リモート
リポジトリ



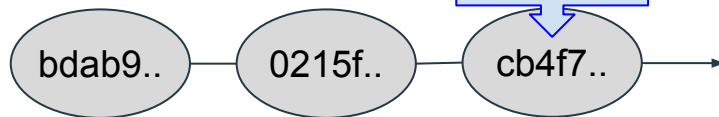
ローカル
リポジトリB



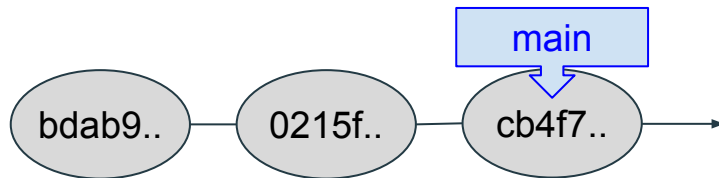
課題5-6

- 1) ローカルリポジトリB内の「DailyReport/June.txt」ファイルを適当に変更し、コミットしなさい

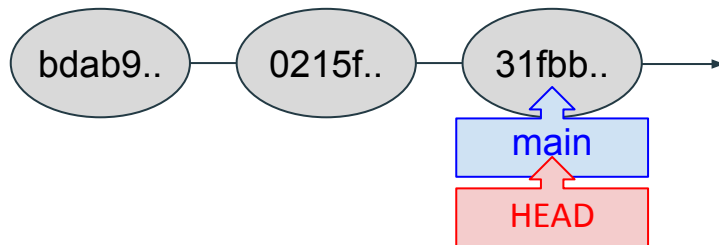
ローカル
リポジトリA



リモート
リポジトリ



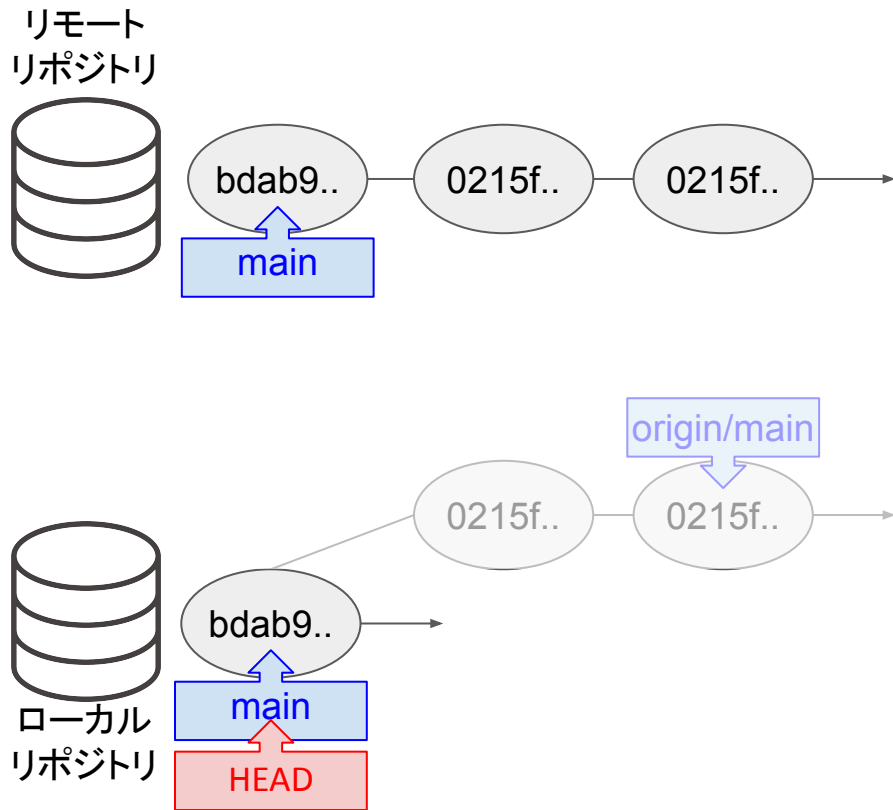
ローカル
リポジトリB



GitHub講座

フェッチ

- リモートリポジトリからコミットを複製する
- 一般的にリモートリポジトリからコミットを取得する時に使用する
- 共通の親コミットまで取得し、リモート追跡ブランチを作成する



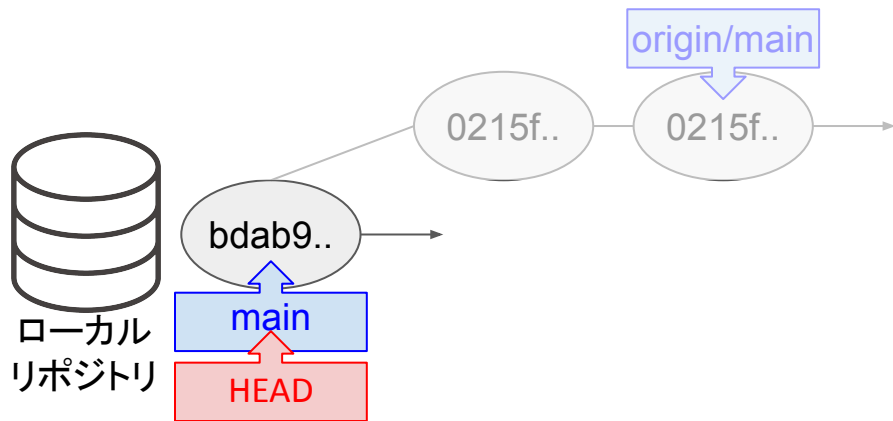
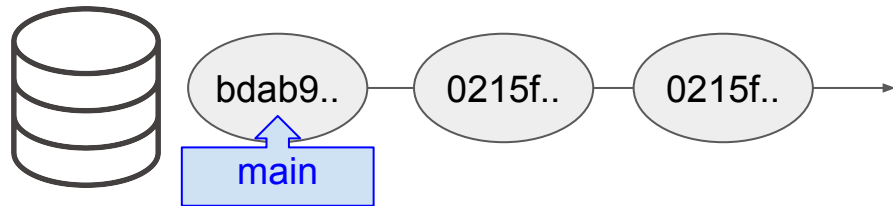
GitHub講座

フェッチ

1. 「Git Graph」を開く
2. 右上の雲のようなボタンを押す



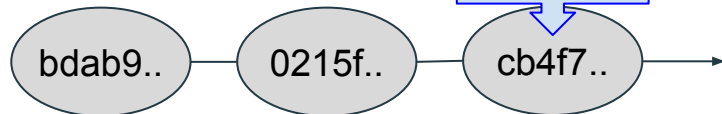
リモート
リポジトリ



課題5-7

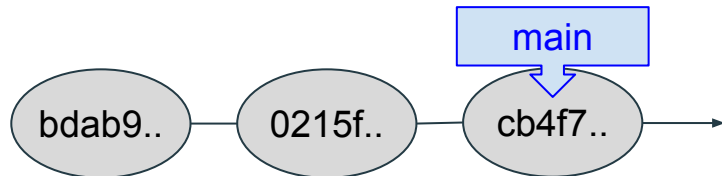
- 1) ローカルリポジトリBでフェッチをなさい
 - a) 「Git Graph」を開く
 - b) 右上の雲のようなボタンを押す

ローカル
リポジトリA

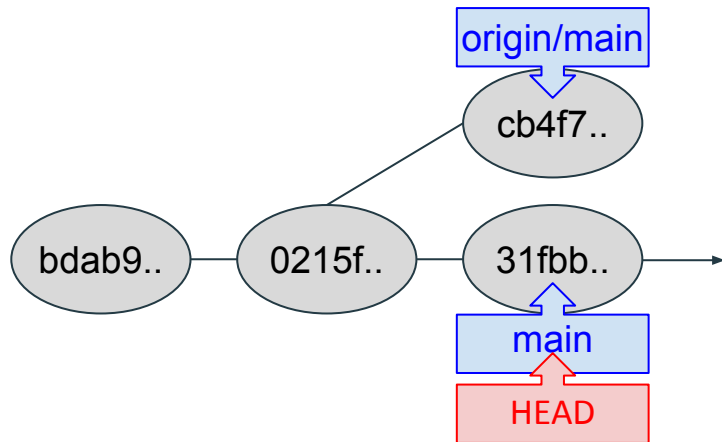


上から
main:Aのローカルブランチ
main:リモートブランチ
origin/main:リモート追跡ブランチ
main:Bのローカルブランチ

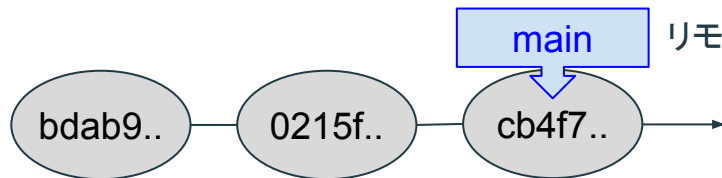
リモート
リポジトリ



ローカル
リポジトリB

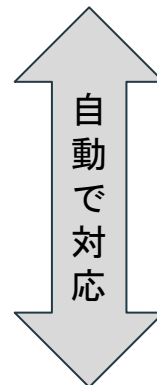


リモート
リポジトリ



main

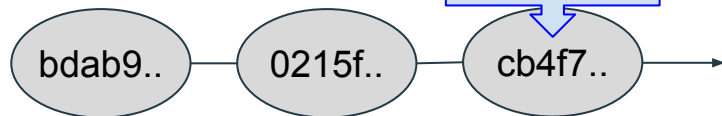
リモートブランチ



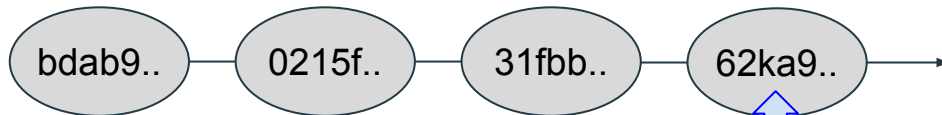
origin/HEAD

origin/main

リモート追跡ブランチ



ローカル
リポジトリB

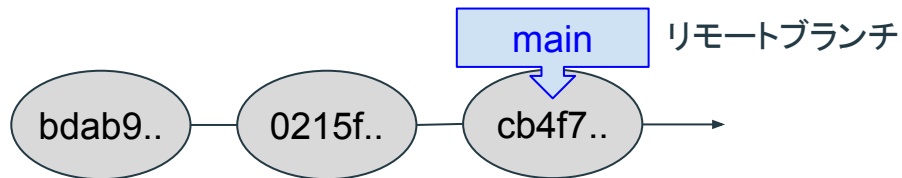


main

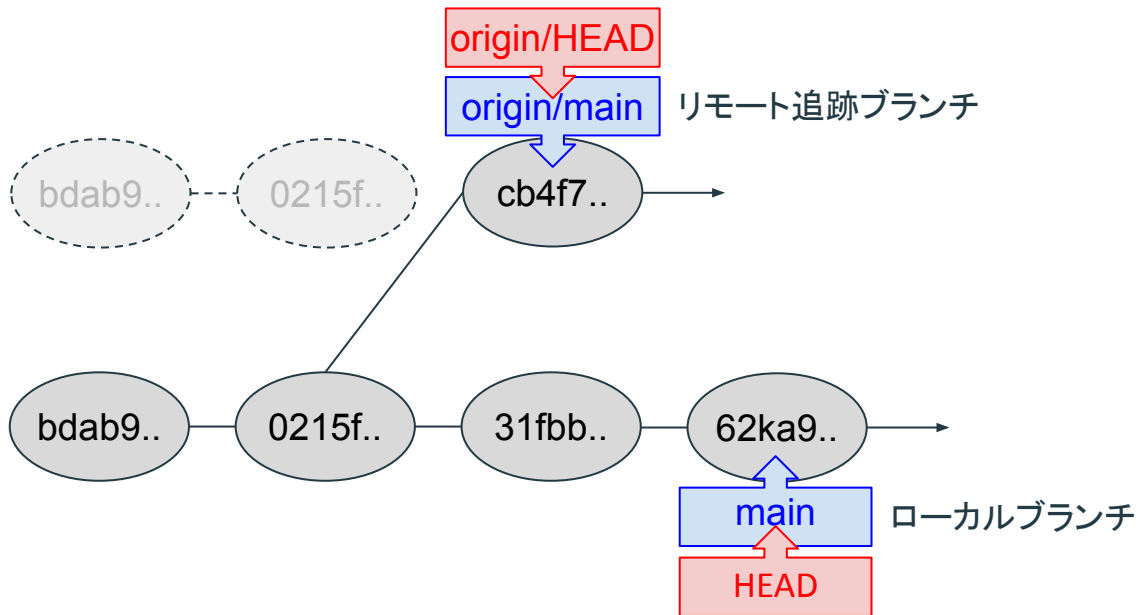
ローカルブランチ

HEAD

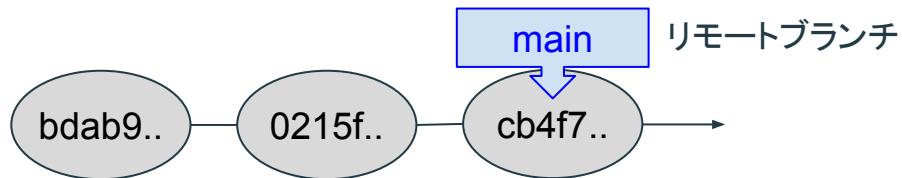
リモート
リポジトリ



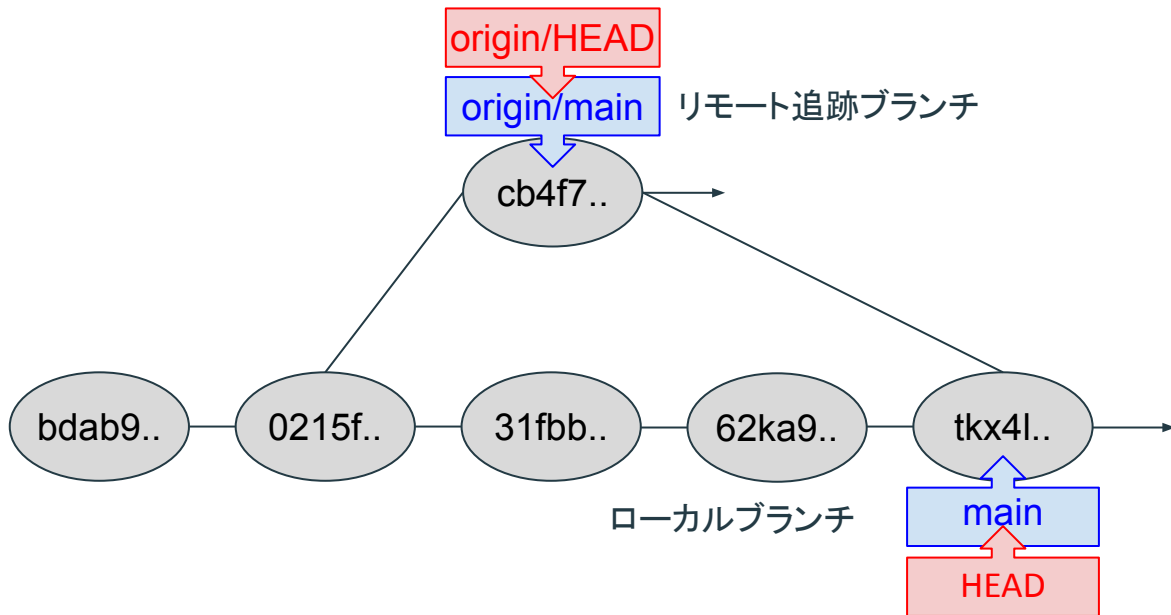
ローカル
リポジトリB



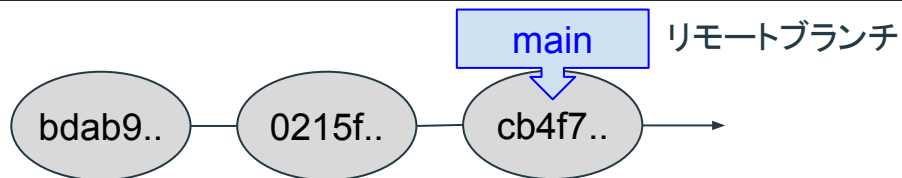
リモート
リポジトリ



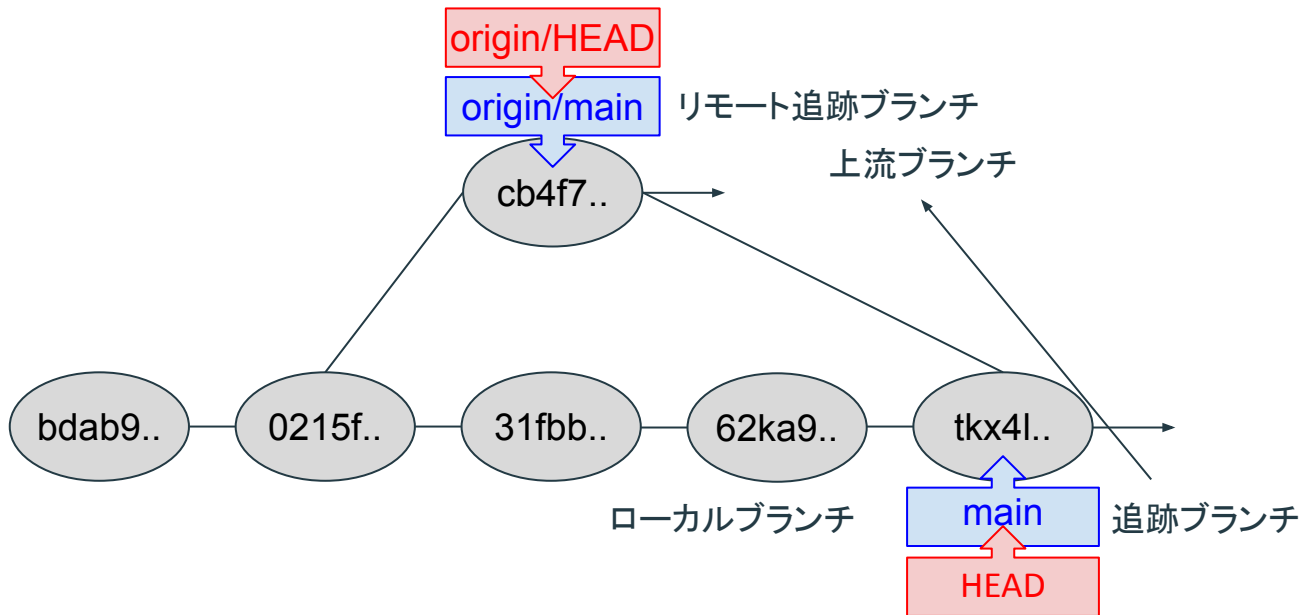
ローカル
リポジトリB



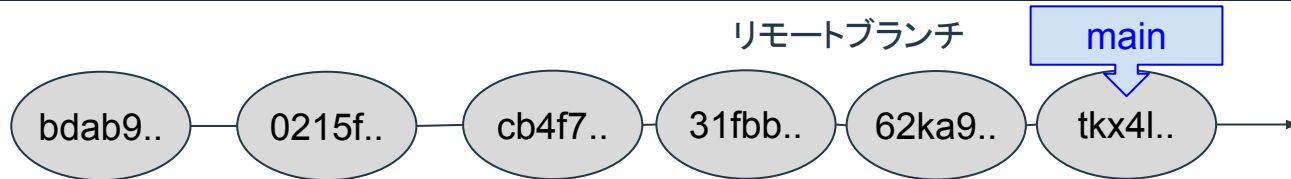
リモート
リポジトリ



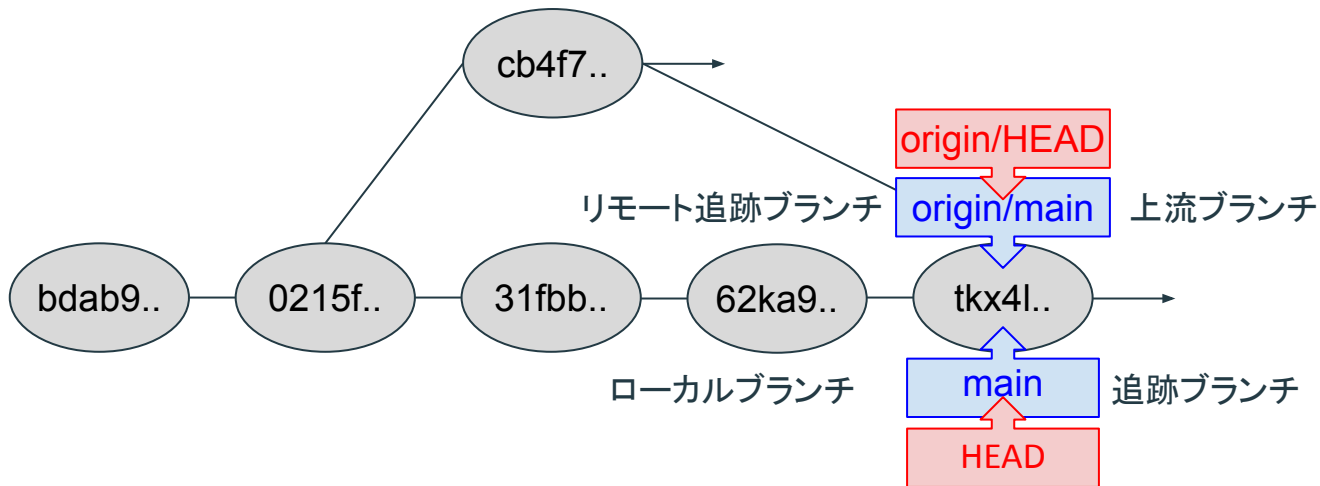
ローカル
リポジトリ



リモート
リポジトリ



ローカル
リポジトリ



課題5-8

- 1) ローカルリポジトリBで「origin/main」を「main」にマージしなさい
- 2) ローカルリポジトリBの「main」ブランチをプッシュしなさい

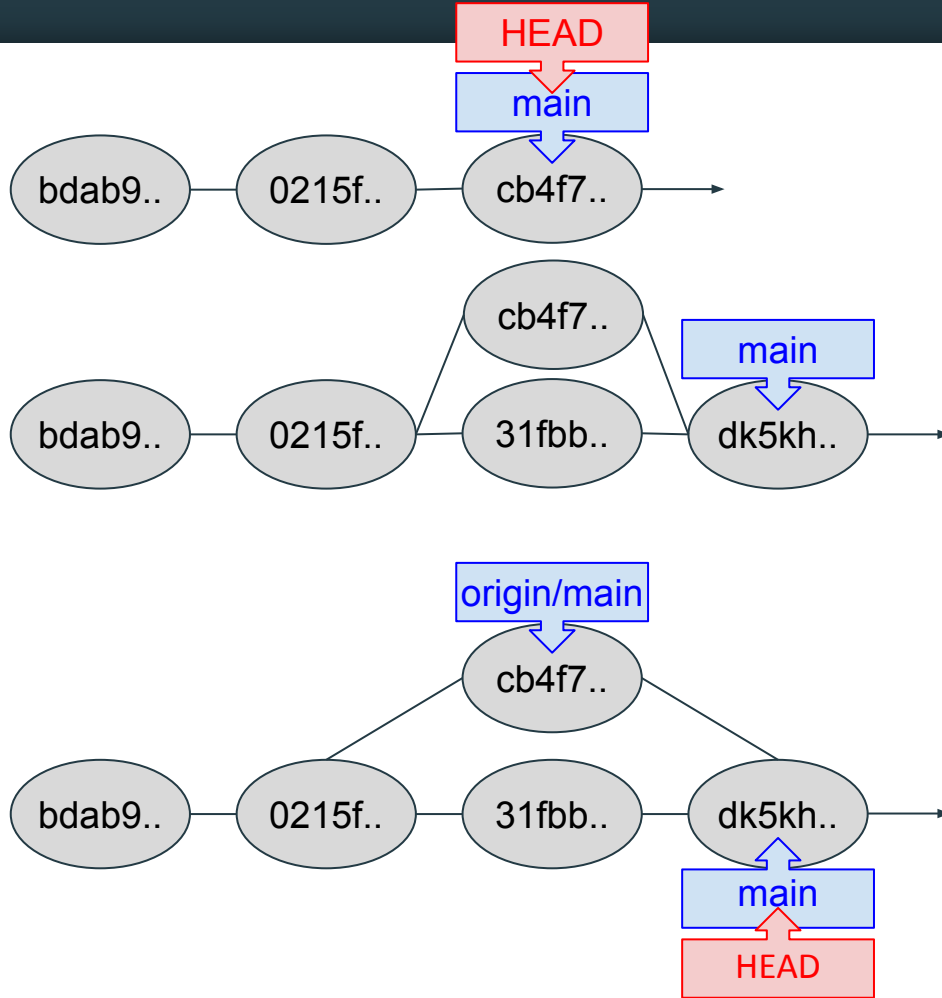
ローカル
リポジトリA



リモート
リポジトリ



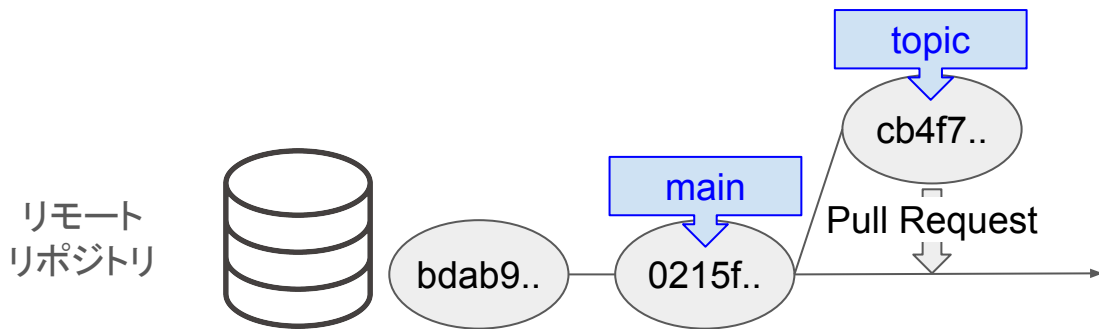
ローカル
リポジトリB



GitHub講座

Pull Requestとは

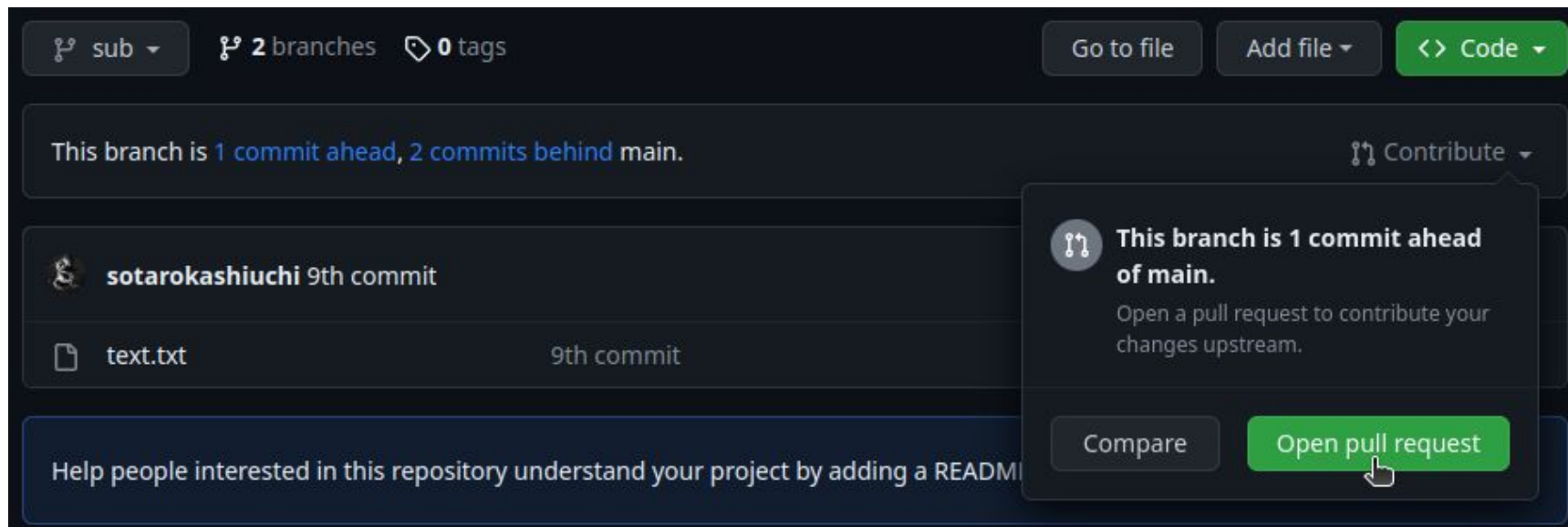
- mainブランチなどの重要なブランチへマージを行うためのGitHubの機能
- 重要なブランチにバグのあるコードを含めたり、動かないコードを含またりしたくない
- そのためレビュー(コード作成者以外の人がコードを確認すること)したコードしかマージをしたくない



GitHub講座

Pull Requestの送信方法

1. GitHub上で、自分が作成したブランチに移動
2. 「Compare & pull request」ボタン又は「Contribute」->「Open pull request」ボタンを押す



GitHub講座

Pull Requestの送信方法

3. 右側にある「Reviewers」にレビュアーを指定する
4. レビューアーに伝えたい情報を記入し、「Create pull request」ボタンを押す
5. レビューアーに「Approve」してもらい、「Merge pull request」ボタンを押す

The screenshot shows the GitHub interface for creating a pull request. At the top, it indicates the base branch is 'main' and the compare branch is 'sub', with a green checkmark stating 'Able to merge. These branches can be automatically merged.' Below this, there are two main sections: 'Add a title' and 'Add a description'. The 'Add a title' section has a text input field containing the text '高速化'. The 'Add a description' section has a text area with the text '# 行ったこと' and '- 並列処理を実装し、高速化を図った'. To the right of these sections, there are several metadata fields: 'Reviewers' with a dropdown menu showing 'KashiuchiSotaroSub', 'Assignees' with a dropdown menu showing 'No one—assign yourself', 'Labels' with a dropdown menu showing 'None yet', 'Projects' with a dropdown menu showing 'None yet', 'Milestone' with a dropdown menu showing 'No milestone', and 'Development' with a dropdown menu showing 'Use Closing keywords in the de automatically close issues'. At the bottom right, there is a green button labeled 'Create pull request'. At the bottom left, there is a footer note: 'Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).'

base: main ← compare: sub ✓ Able to merge. These branches can be automatically merged.

Add a title

高速化

Add a description

Write Preview H B I ≡ <> ↻ ≡ ≡ ≡ ...

行ったこと
- 並列処理を実装し、高速化を図った

Markdown is supported Paste, drop, or click to add files

Reviewers
KashiuchiSotaroSub

Assignees
No one—[assign yourself](#)

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Use [Closing keywords](#) in the de automatically close issues

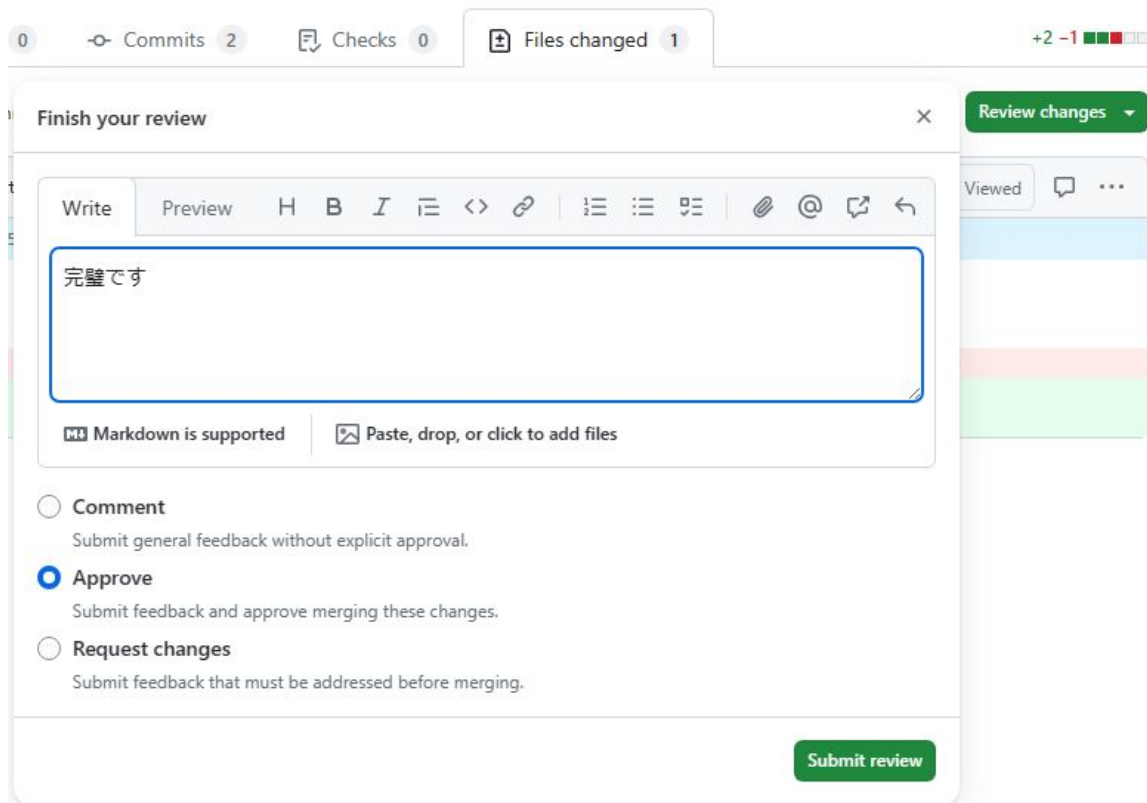
Create pull request

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

GitHub講座

レビューの対応方法

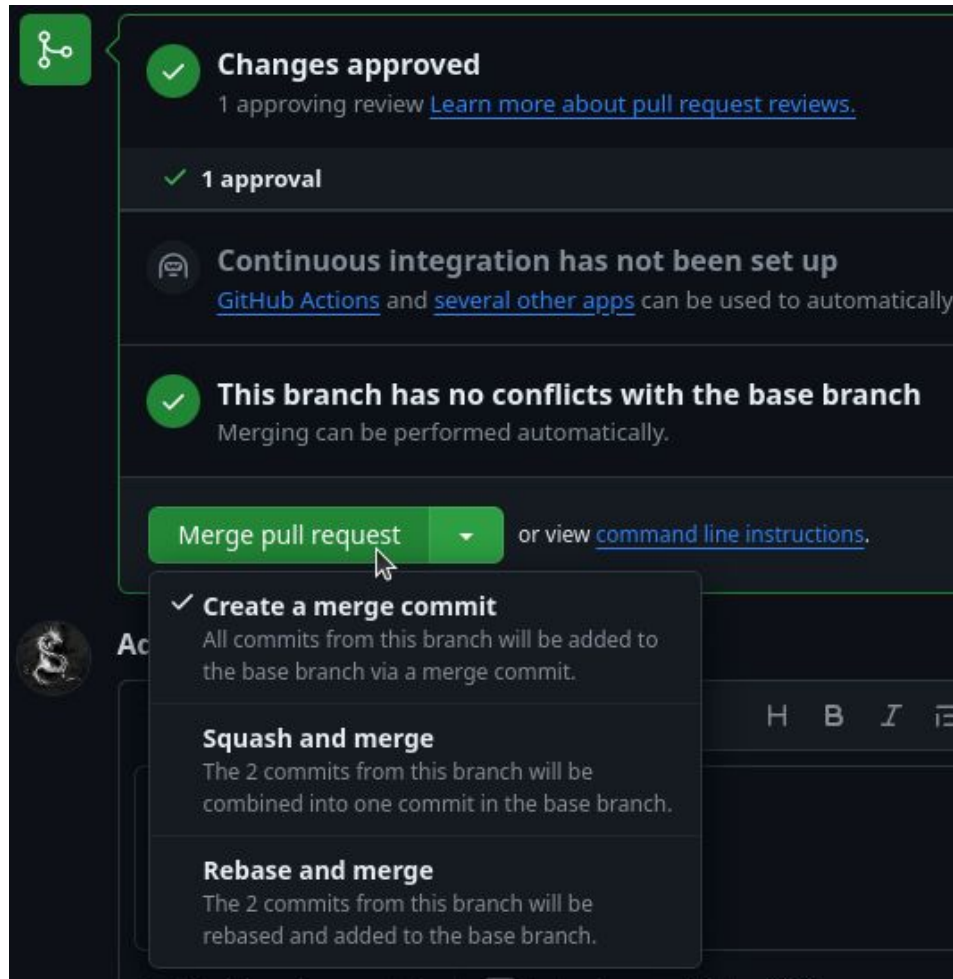
1. 「File changed」を開く
2. コードに問題がなければ、「Review changes」->「Approve」ボタンを押す
3. コードに問題があれば、コメントを記入する



GitHub講座

Merge Pull Request

1. マージのモードを選択
2. 「Merge pull request」をクリック

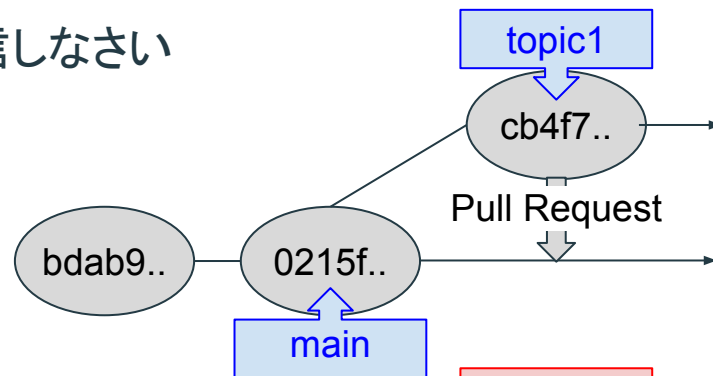


課題5-9 (時間があれば)

1) 以下のようにGitを操作し、Pull Requestを送信しなさい

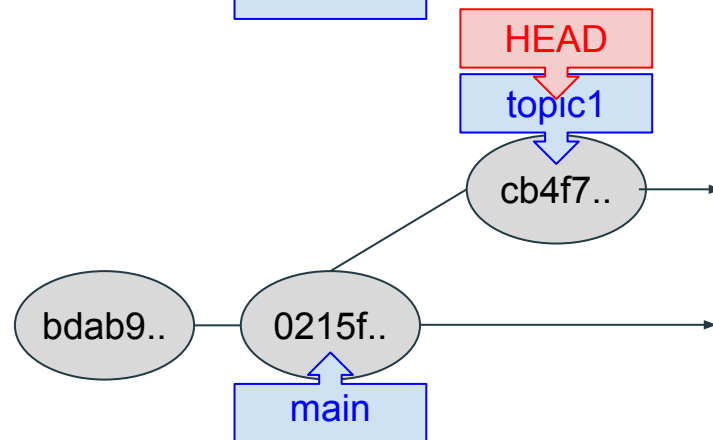
- ローカルリポジトリAでtopic1ブランチを作成
- ファイルをコミットしプッシュ
- Pull Requestを発行

リモート
リポジトリ



2) Merge Pull Requestをクリックしなさい

ローカル
リポジトリA



最終目標

「数値解析」が行えるプログラムをGo言語で作成しなさい

<https://github.com/sotarokashiuchi/JointDevelopmentEnviromentLesson>

GitHub講座

共同開発のルール(GitHub Flowを採用)

1. mainブランチは常に動作する状態にする(コンパイルが通る状態)
2. 新しい機能を追加するときはmainから新しい作業用ブランチを作成する
3. 新しい機能が追加でき、mainブランチに取り込んでほしいときは、作業用ブランチにmainブランチをマージし、PullRequestを送信する
4. チームメンバーの少なくとも一人がレビューを行う
5. レビューで問題がなければ、approveを押し、マージを行う
6. レビューで問題があれば、問題点を改善して3の工程からやり直す

GitHub講座

リモート
リポジトリ

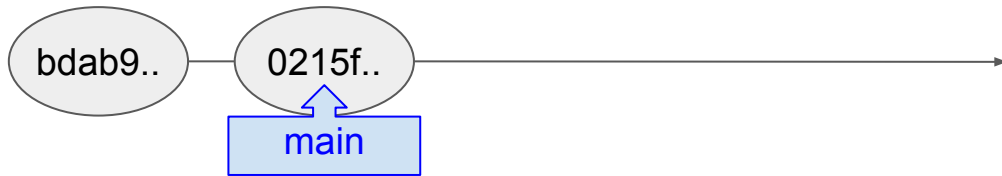


bdab9..

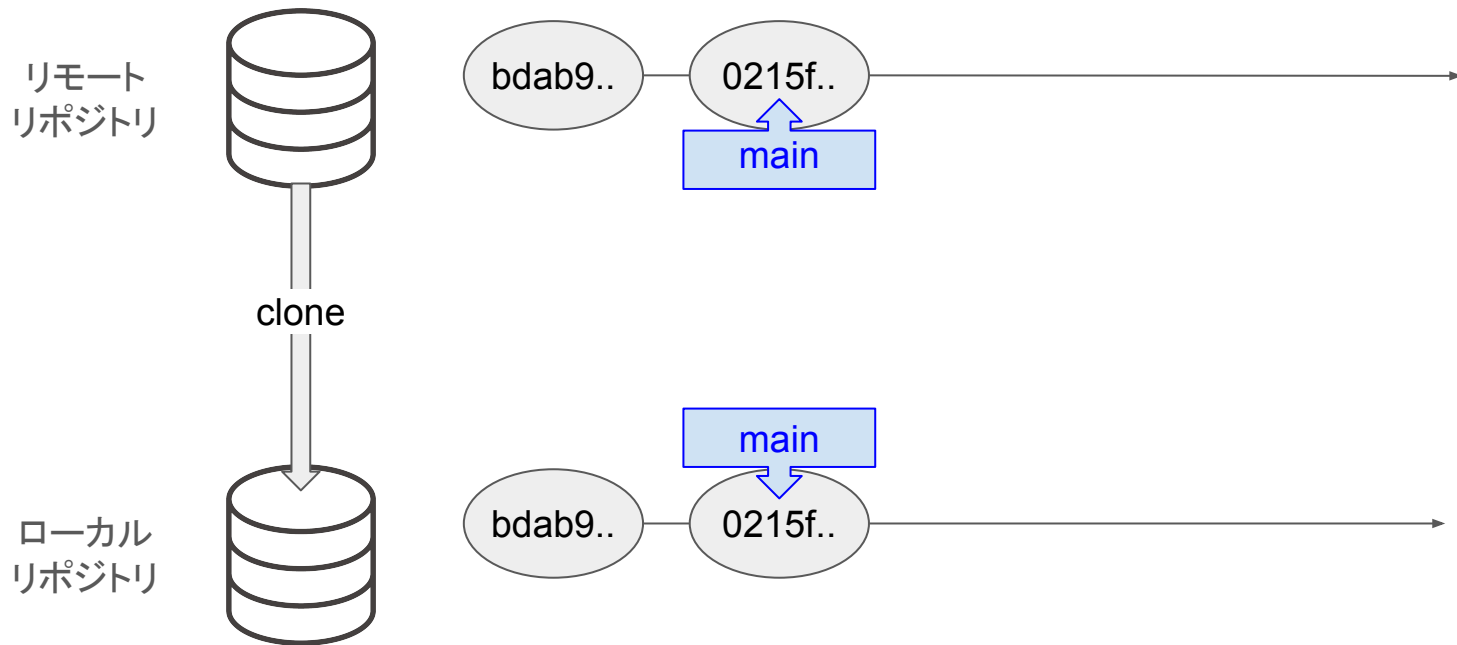
0215f..

main

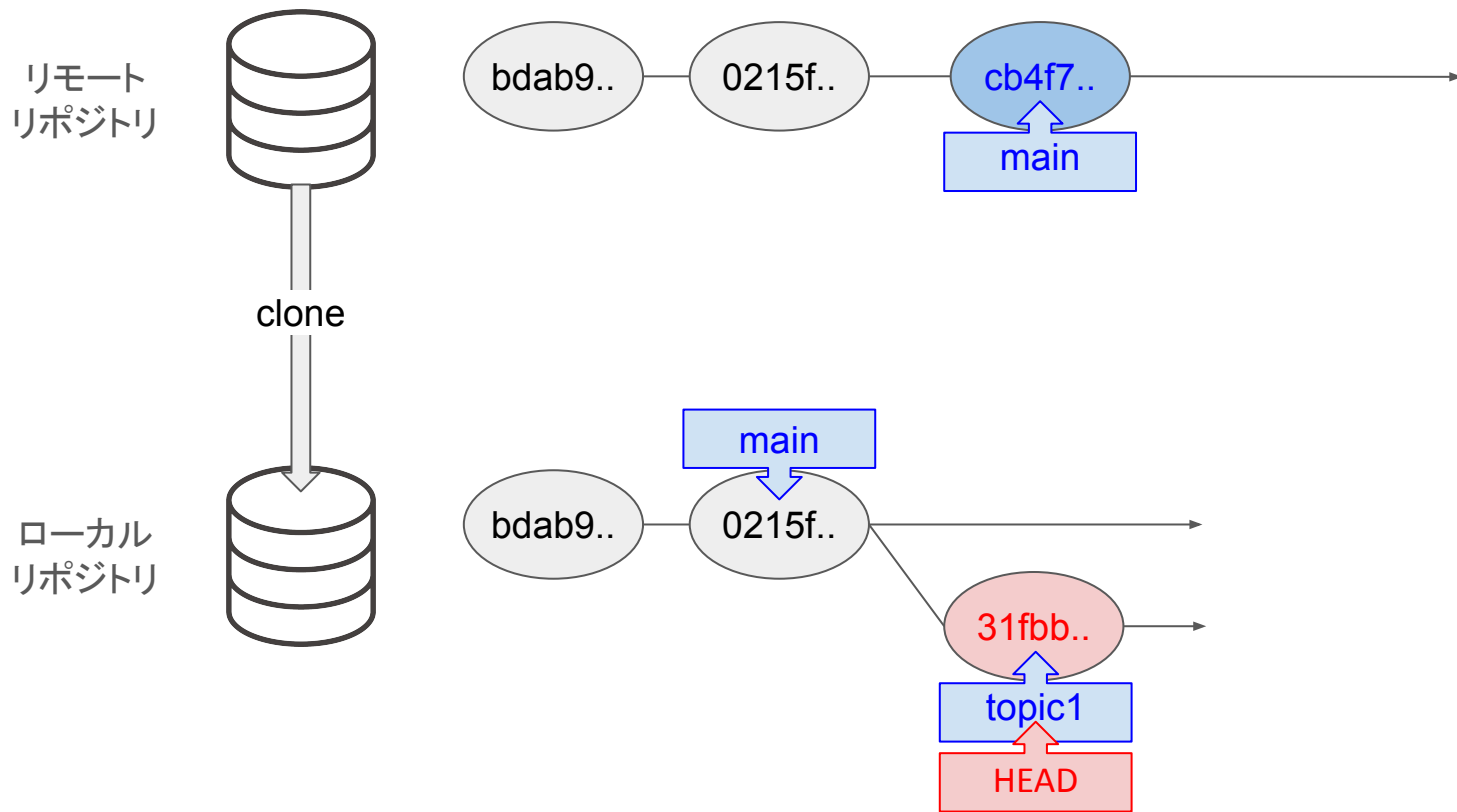
ローカル
リポジトリ



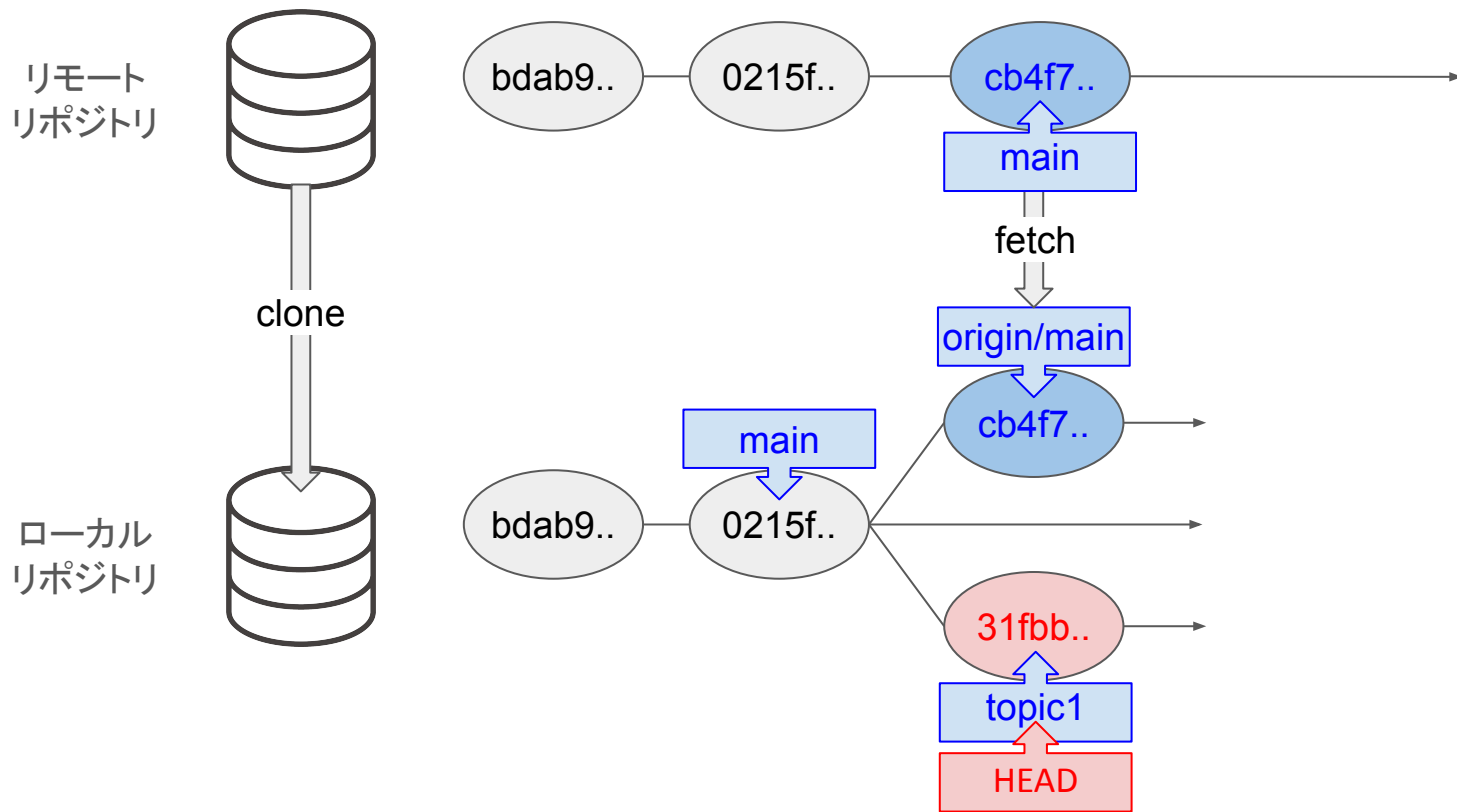
GitHub講座



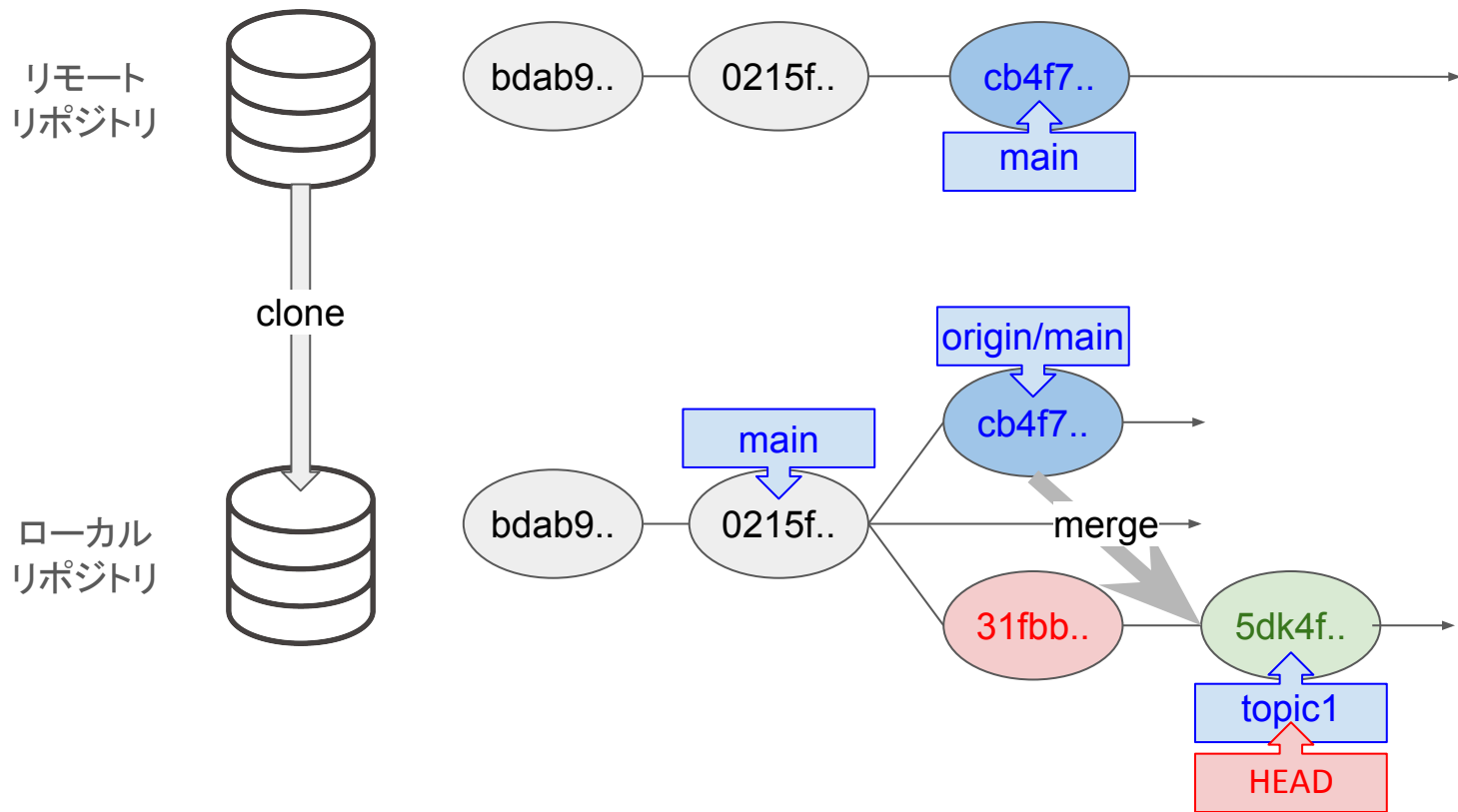
GitHub講座



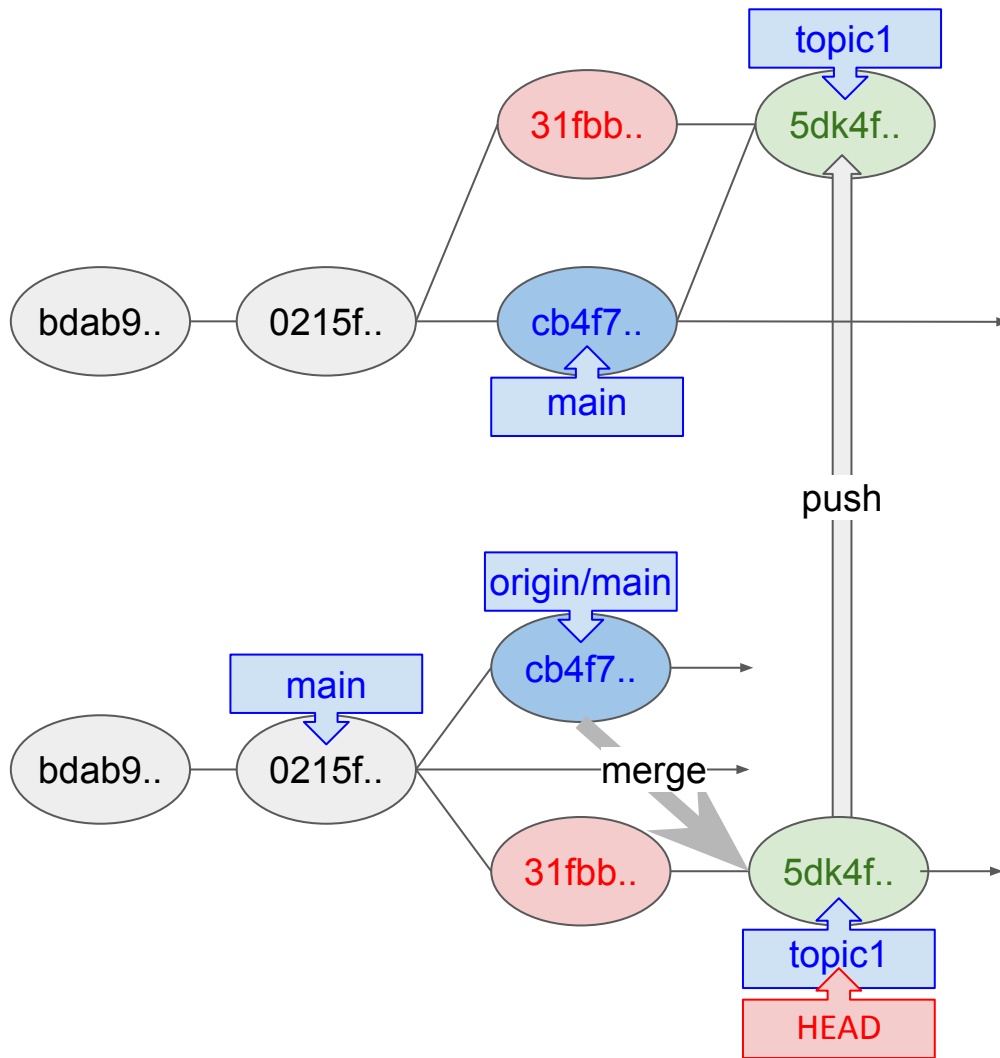
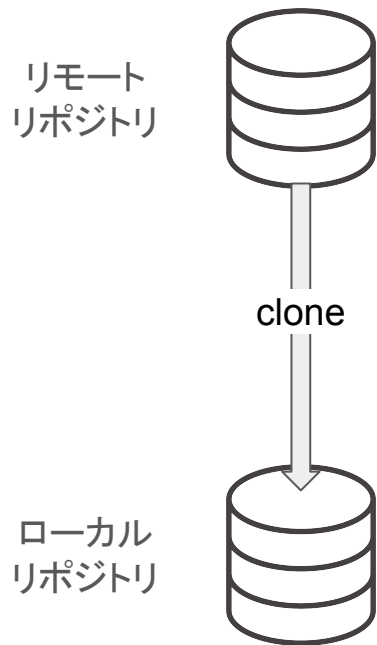
GitHub講座



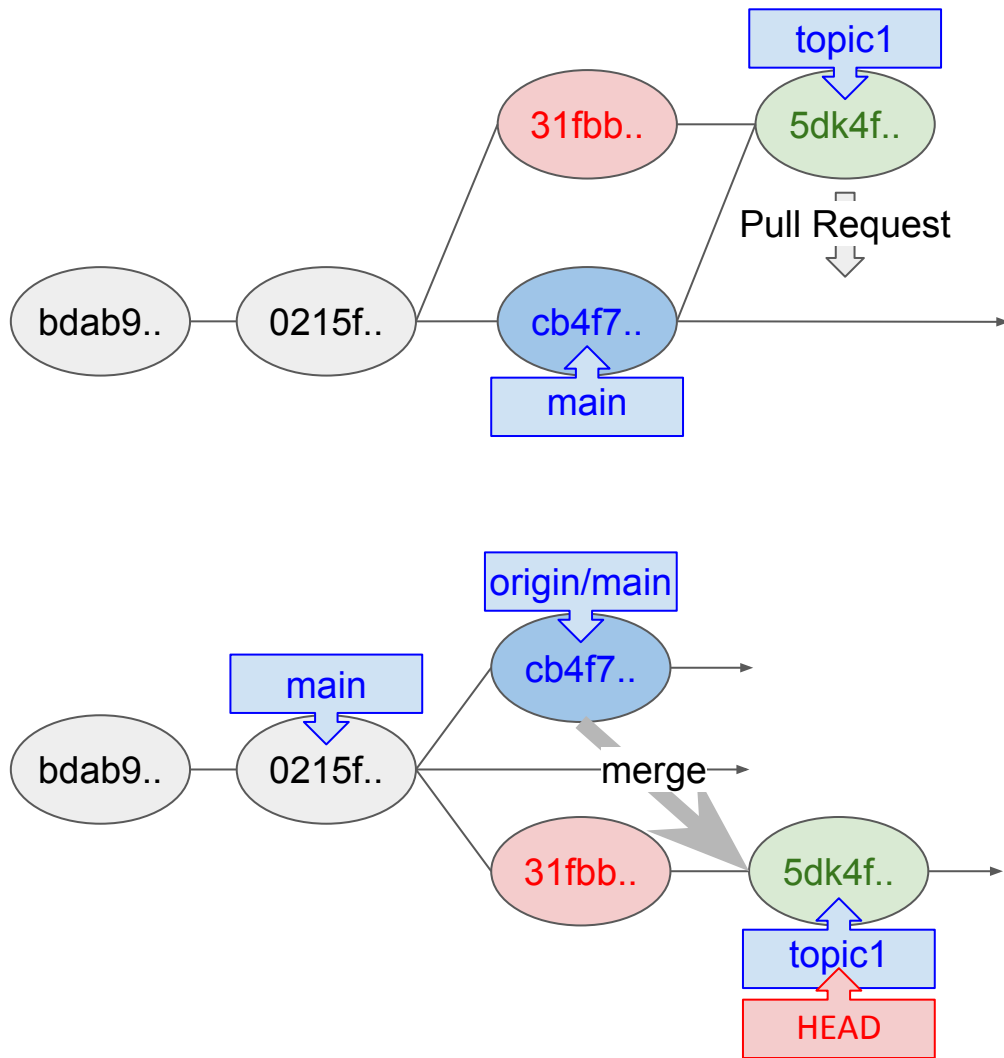
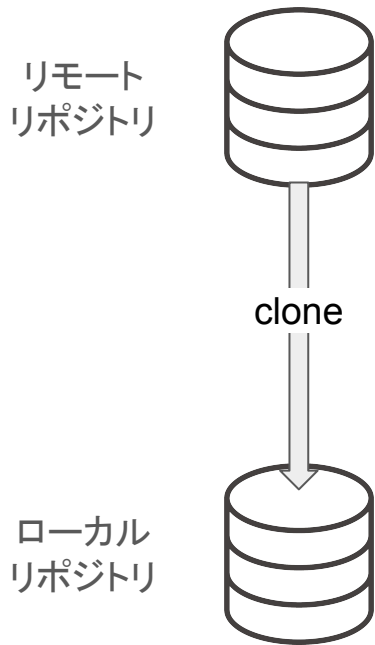
GitHub講座



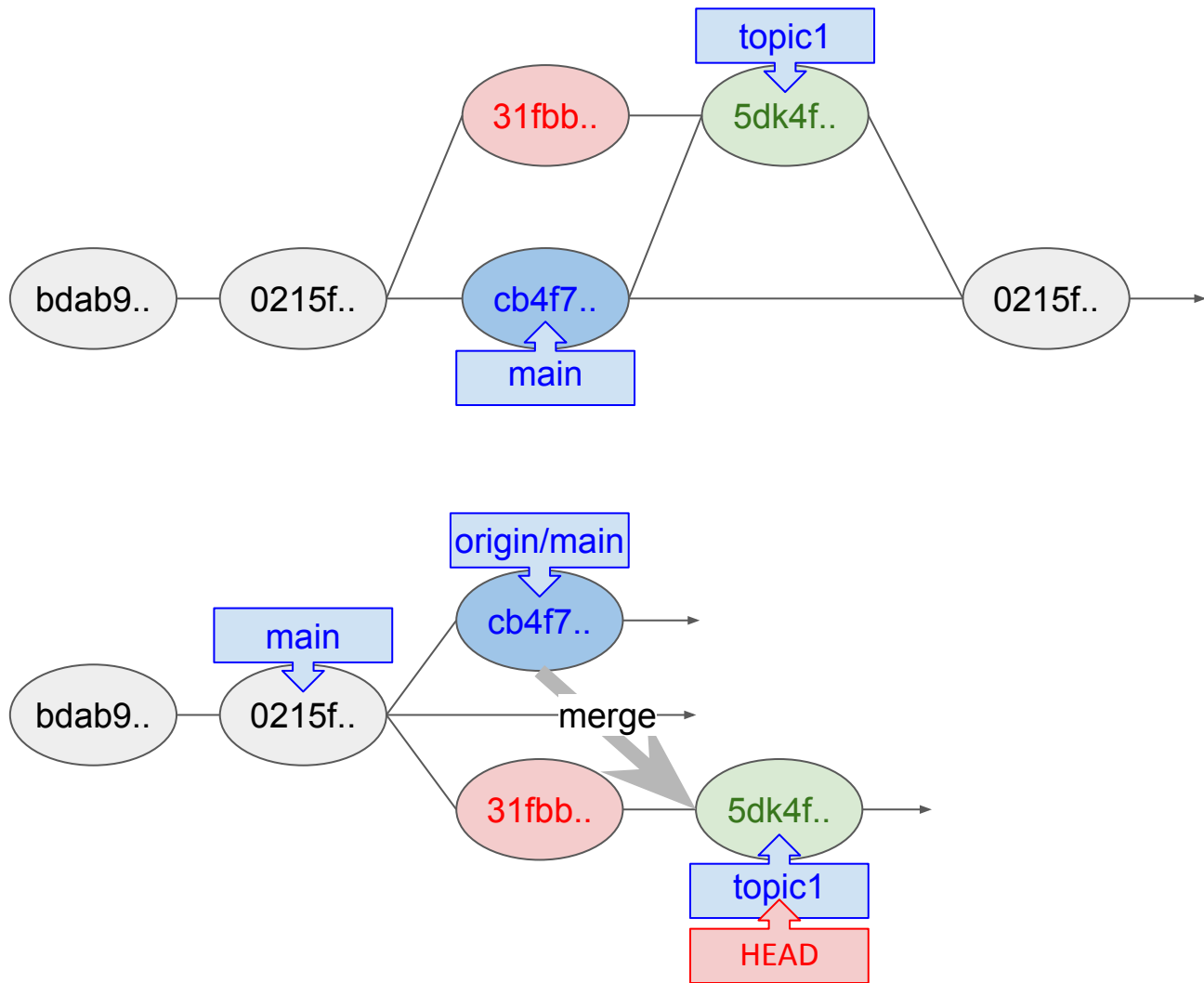
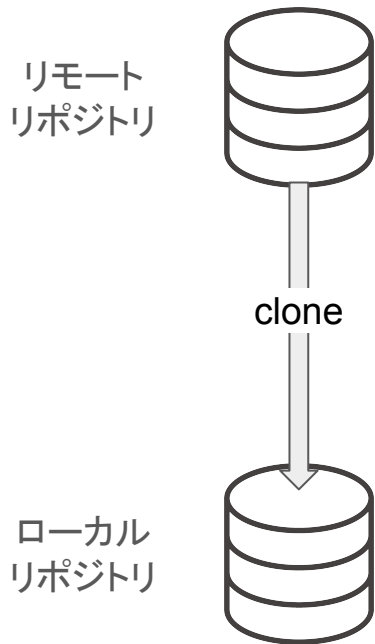
GitHub講座



GitHub講座



GitHub講座



課題6-1

- 1) 講師のGitHubのリモトリポジトリをcloneしなさい
 - a) VSCodeのコマンドパレットを開き、「Git: Clone」をクリック
 - b) 講師のGitHubのリモトリポジトリのURLを入力する
 - c) クローンするフォルダを指定する

課題6-2

- 1) 自分の「ニックネーム」のbranchを作成、チェックアウトし、「README.md」を以下のように変更し、「# 開発者」の下に行に「- 名前」を付け加え、コミットしなさい
 - a) 「main」ブランチを右クリックし、「Create Branch...」をクリックし。。。。
 - b) 新しく作ったブランチをダブルクリック
 - c) ファイルを編集し、コミットを行う

```
~省略~  
# 開発者  
-   Name  
  
~省略~
```

課題6-3

- 1) リモートブランチをフェッチで取得しなさい
- 2) リモートブランチとローカルの作業用ブランチ(ニックネームブランチ)を比べ、リモートブランチの方が進んでいた場合、リモートのmainブランチをマージし、プッシュしなさい。進んでいない場合は、そのままプッシュしなさい。
 - a) Git Graphの雲のようなFetchボタンをクリック
 - b) リモートブランチとローカルブランチを比較する
 - c) 必要であればマージする
 - d) プッシュしたブランチを右クリックし、「Push Branch...」をクリック

課題6-4

1) Pull Requestを作成しなさい

- a) レビューアーに「KashiuchiSotaro」を含めてPull Requestを送る
- b) 競合が発生した場合は、競合を解決してpushしなさい
- c) 既にPull Requestを開いているブランチにpushすると、自動的にPull Requestが更新される

課題6-5

- 1) 数値解析ソフトを実行しなさい
 - a) `$ go run cmd/main.go`
- 2) 「仕様書」に書かれている各関数を作成しなさい。なおコミット、プッシュ、フェッチ、マージは各自適宜行うこと
- 3) 上記の課題ができて時間がある人は、他の機能を付け足したりしてみてください

※マージを少なくするには、プルリクを早く送るのと、修正を少なくすること

先手必勝!?

Golangの言語仕様

機能	書式	備考
変数宣言	var 変数名 データ型	データ型には以下のようなものがある int(整数), float64(実数), string(文字列)
変数参照	変数名	C言語とほとんど同じように使える
変数代入	変数名 = 式	
関数宣言	func 関数名 (引数, ...) 戻り値のデータ型 { ... }	例) func sample (x int, y string) int { ... }
関数呼び出し	関数名(引数, ...)	C言語とほとんど同じように使える
if文	if 条件式 { ... }	演算子はC言語とほとんど同じ +, -, /, %, *, ==, !=, &&, ,
if - else文	if 条件式 { ... } else { ... }	
for文	for 初期化; 条件式; 後処理 { ... }	for 条件式 { ... } でwhile文を実現

Golangの標準ライブラリー

書式	機能
<code>fmt.Println(文字列又は、変数名)</code>	標準出力(画面)へ表示 C言語のprintf()に該当
<code>fmt.Scan(&変数名)</code>	標準入力(シェル)からデータを受け取る C言語のscanf()に該当

課題6-5

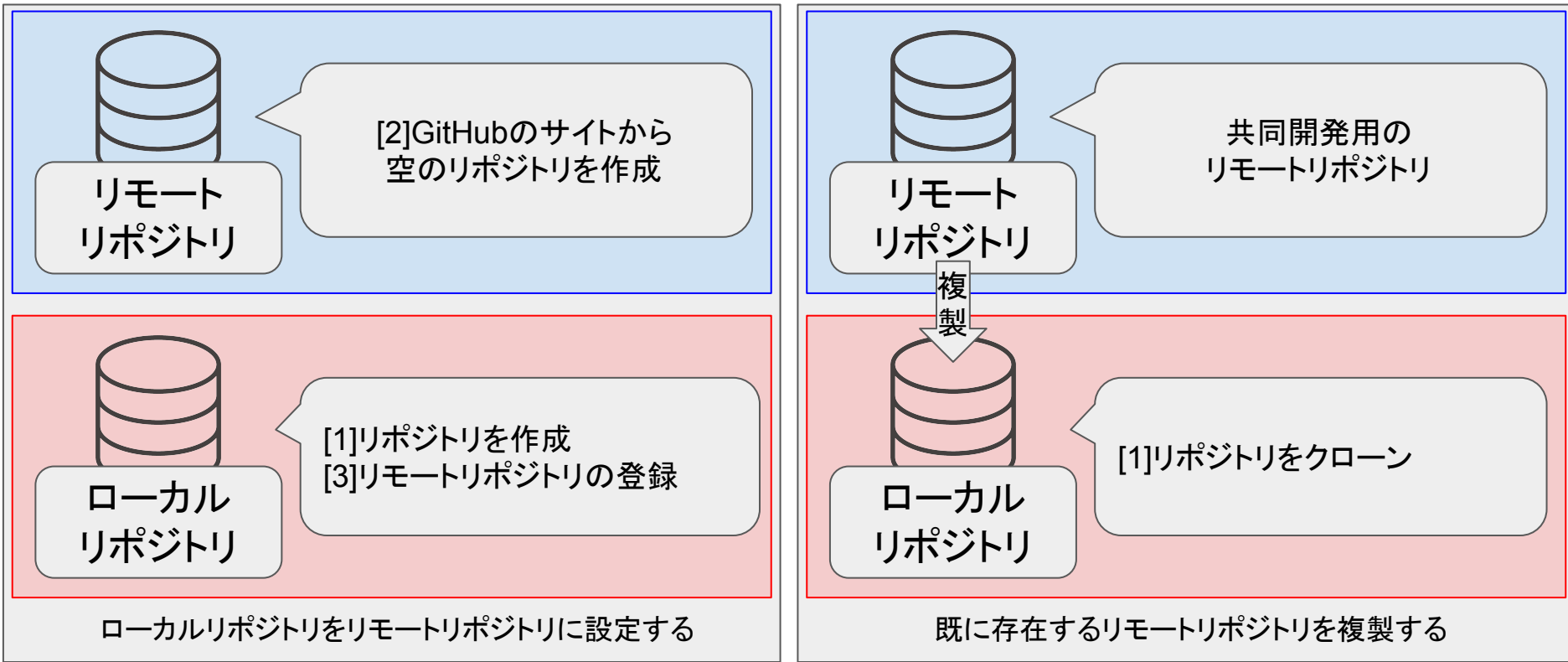
- 1) 数値解析ソフトを実行しなさい
 - a) `$ go run cmd/main.go`
- 2) 「仕様書」に書かれている各関数を作成しなさい。なおコミット、プッシュ、フェッチ、マージは各自適宜行うこと
- 3) 上記の課題ができて時間がある人は、他の機能を付け足したりしてみてください

※マージを少なくするには、プルリクを早く送るのと、修正を少なくすること

完成

Git, GitHubの流れ まとめ

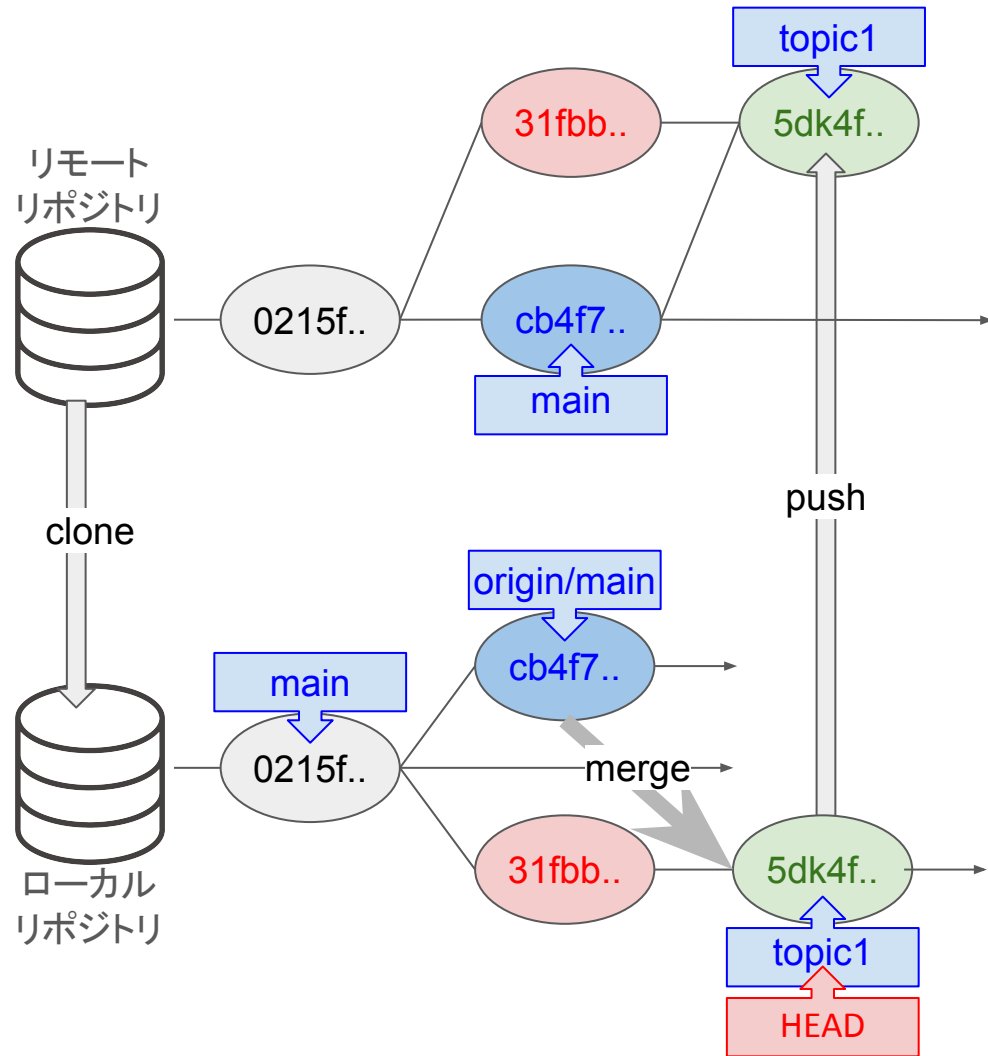
ローカルリポジトリ、リモートリポジトリの作成方法



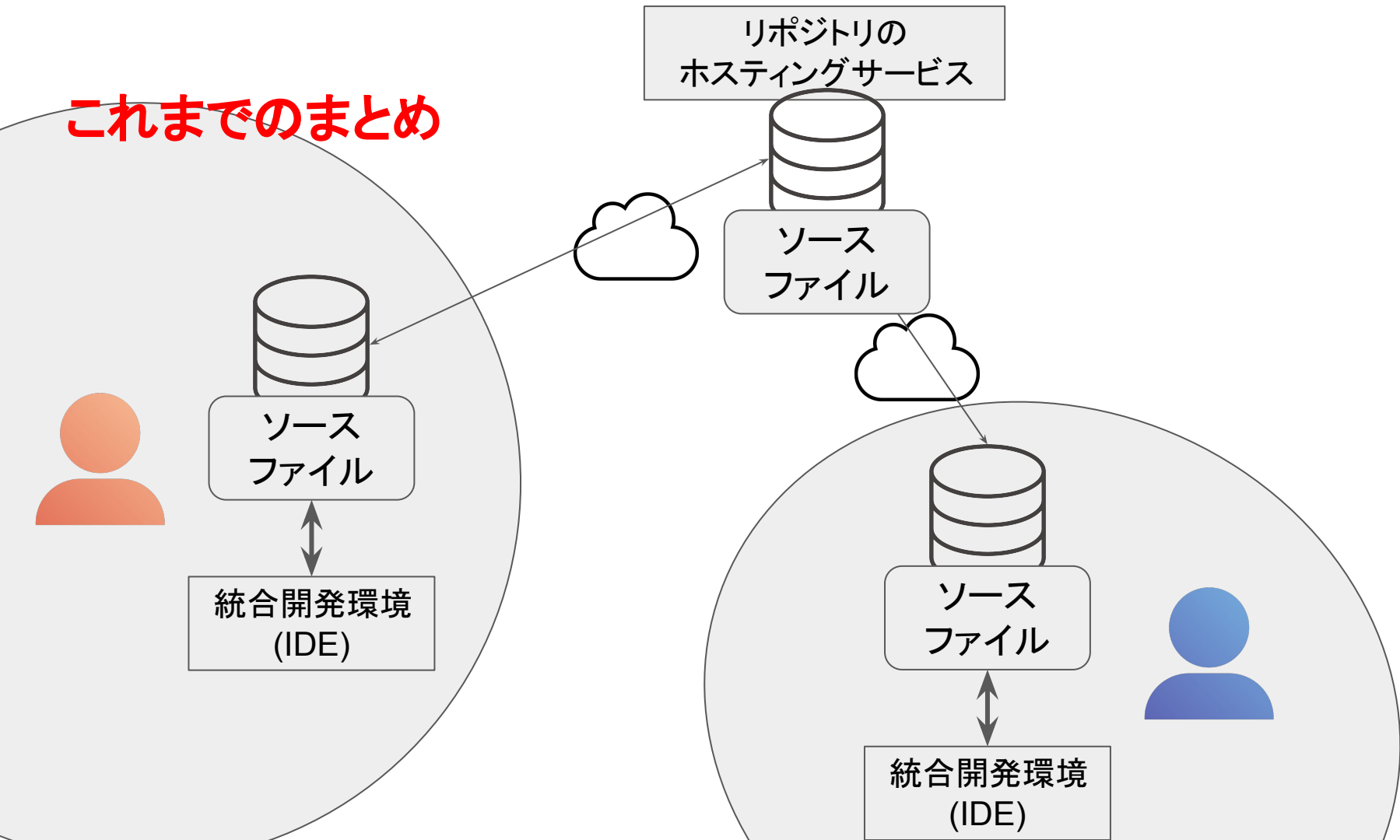
Git, GitHubの流れ まとめ

ローカルリポジトリでの作業の流れ

- 1) 作業用ブランチの作成
- 2) 作業用ブランチへチェックアウト
- 3) ファイルの編集
- 4) インデックスに追加
- 5) コミットの作成
- 6) フェッチの実施
- 7) 必要であればマージを実施
- 8) リモートリポジトリへプッシュ
- 9) プルリクエストの作成



これまでのまとめ



ご清聴ありがとうございました！！