

# 共同開発環境を構築しよう

檜内蒼太朗

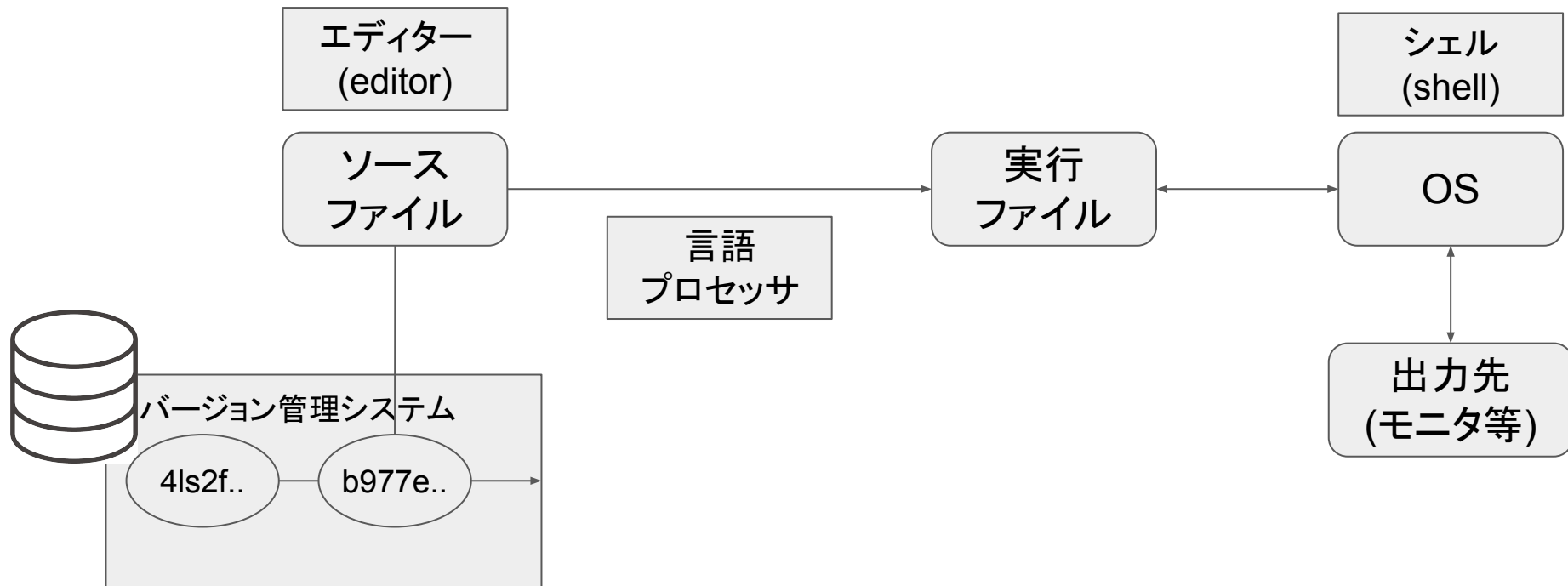
# 自己紹介

- 檜内蒼太郎
- 和歌山工業高等専門学校 電気情報工学科 2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

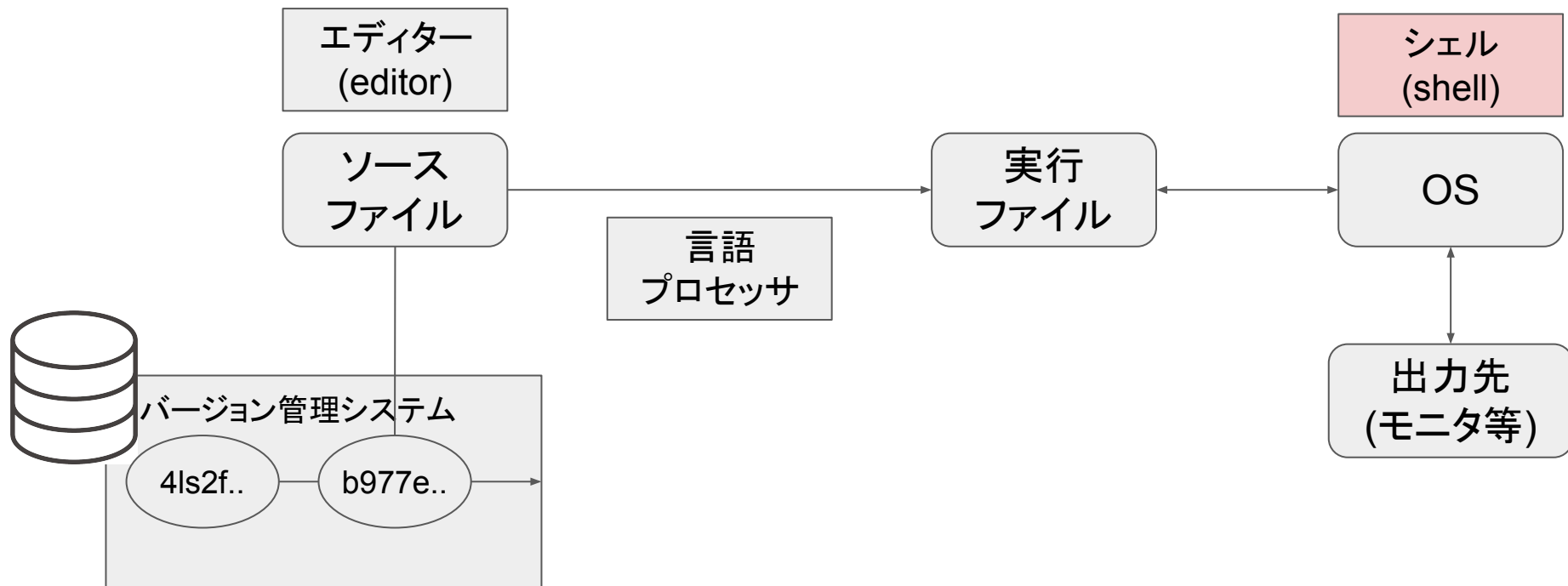
# 目的

- 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

- 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...

# シェル講座

## Windows

### 1. 「PowerShell」シェルを起動

コマンド名	意味
ls	ディレクトリ内のファイルを表示
cd	ディレクトリを移動
mkdir	フォルダの作成
touch	ファイルの作成 (Windowsの場合は「ni」)
cat	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1-1

- 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

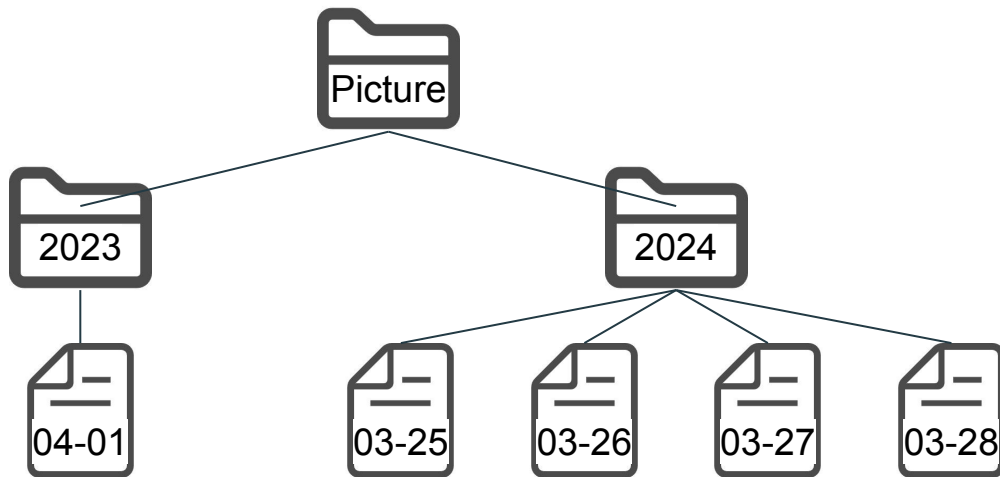
- パス
  - ファイルの場所を表すもの
- ディレクトリ
  - フォルダの別名と考えて問題ない
- カレントディレクトリ
  - 現在ユーザ自身がいるディレクトリ

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1-2

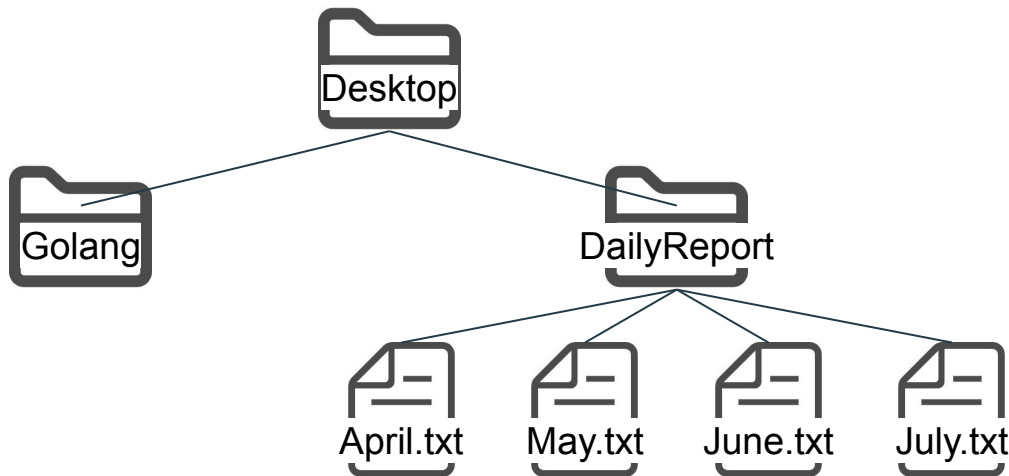
1) カレントディレクトリが「2023」の時以下のファイル又はディレクトを表すパスを答えなさい

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



## 課題1-2

- 1) 下のようなディレクトリとファイルを作りなさい
- 2) Desktop/DailyReport/April.txtに右下のような文を記述し保存しなさい

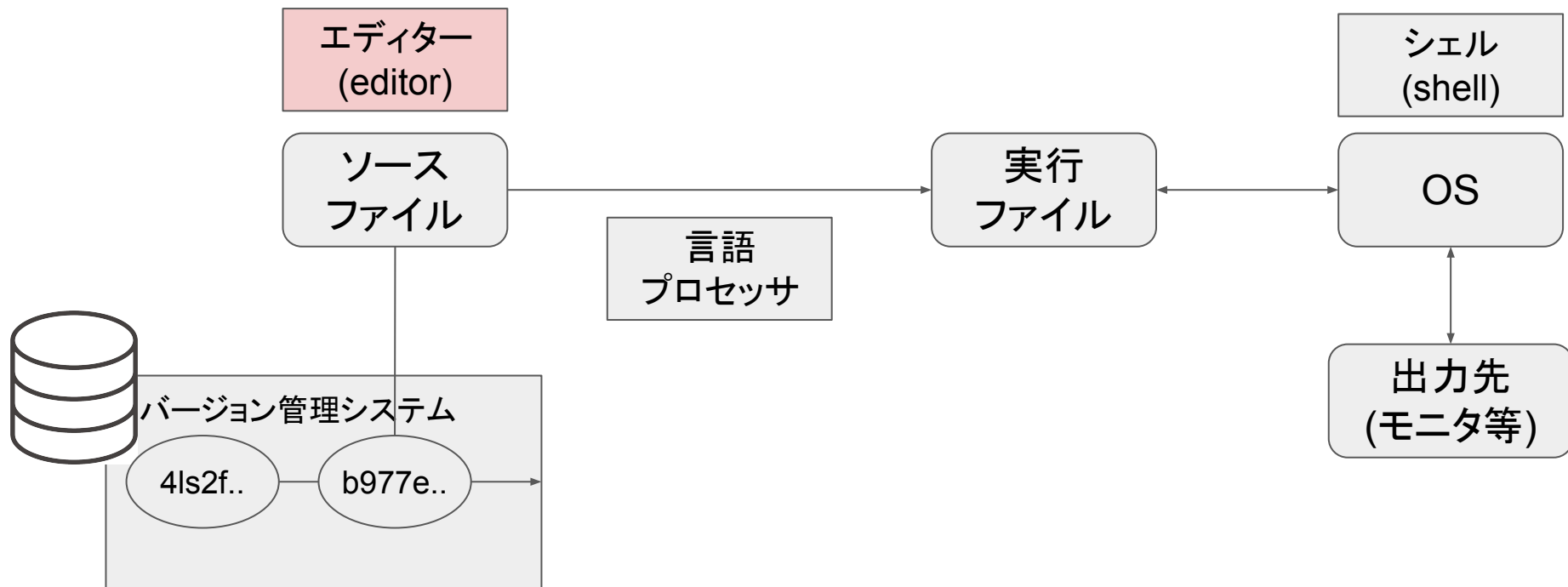


1日  
今日は昼寝をした

2日  
今日も昼寝をした

- 2) カレントディレクトリをDesktop/Clanguageにし、April.txtをシェルを用い表示させなさい

# 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」
- Goツールのインストールと更新
  - 「View」->「command Palette」をクリックし、「Go: Install/Update tools」を検索し、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(&amp;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2-1

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

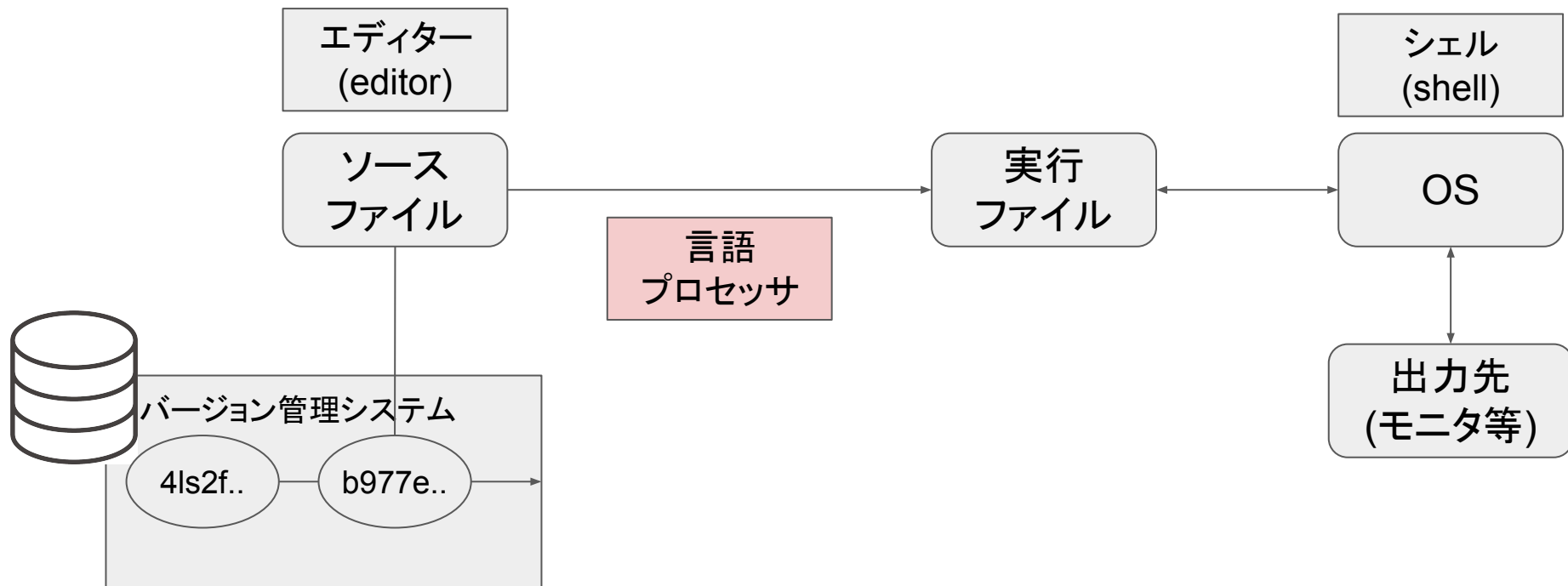
```
package main

import "fmt"

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

※このままでは実行できない！

# 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ユーザは事前課題でダウンロードした「Go」をインストールし、パスを通す

## 課題3-1

1) 「Hello World」を表示させなさい

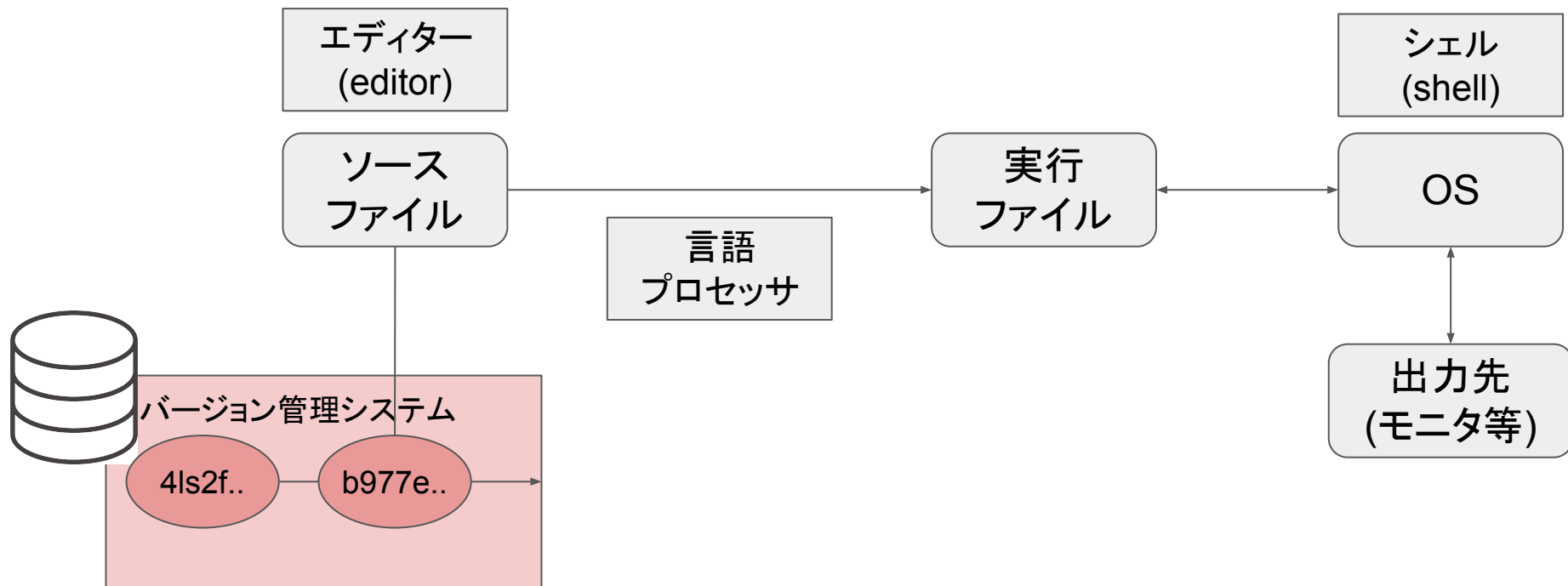
a) `> go run ファイル名`

## 課題3-2

- 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。



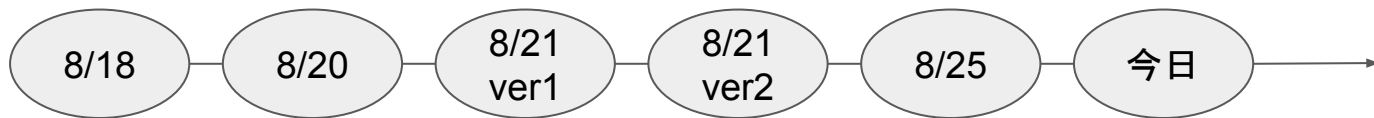
# 個人開発での一般的な開発環境



# 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

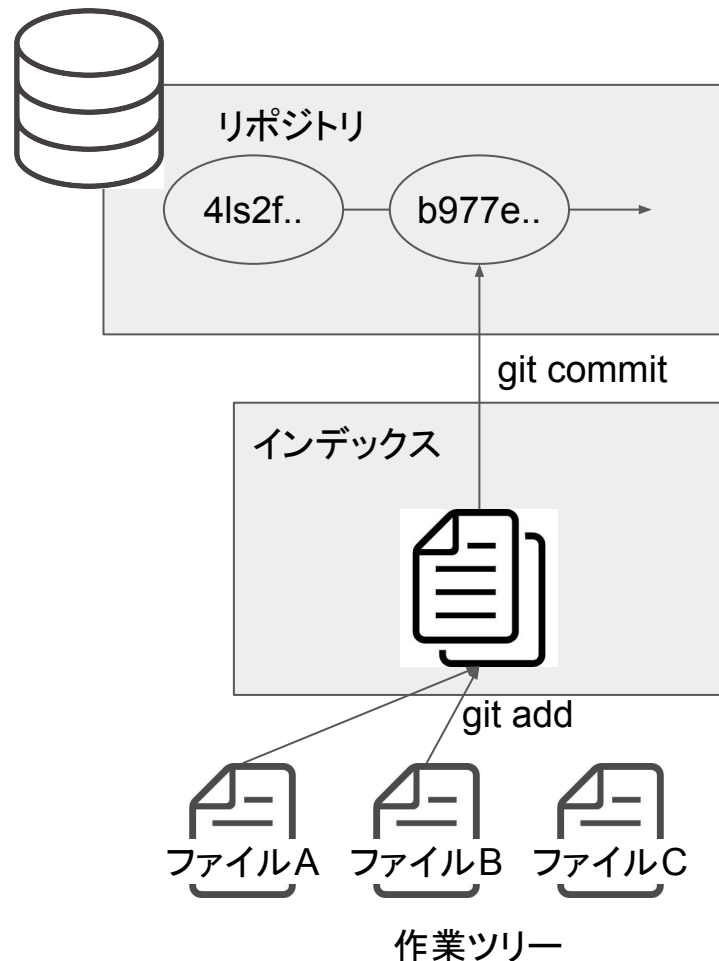
- a. `> git --version`
- b. 「git version x.xx.x」と表示される

## 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"`

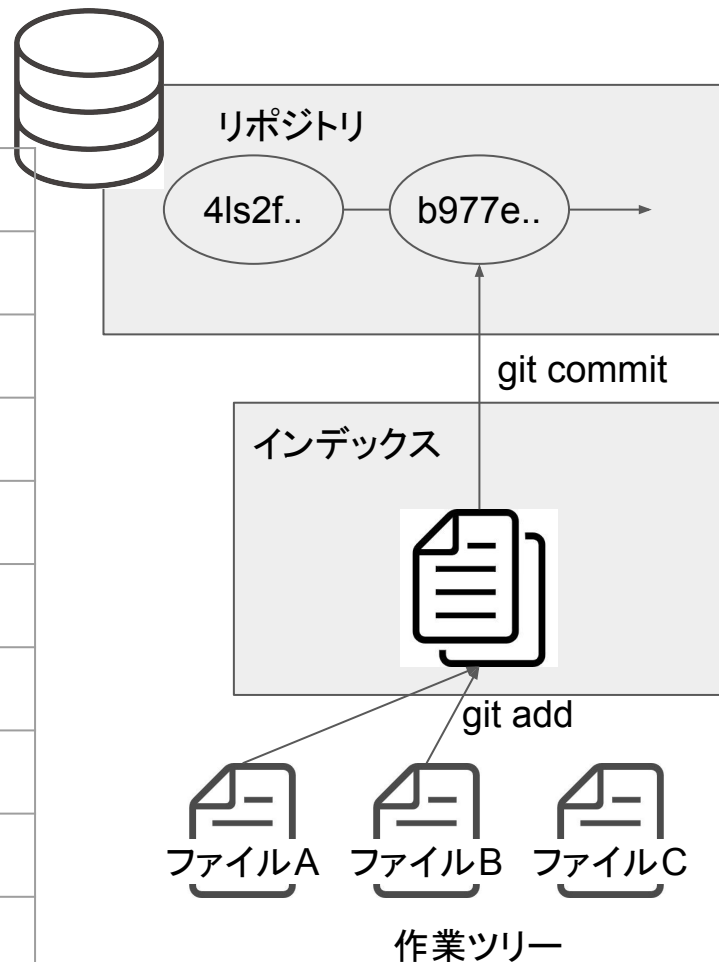
# バージョン管理システム講座

- 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

コマンド	意味
git init	Gitリポジトリの作成
git status	ファイルの状態を表示
git log	コミット履歴を表示
git diff	差分を表示
git add	インデックスに登録する
git restore --staged	インデックスの登録を削除する
git commit	リポジトリにコミットする
git reset	リポジトリのコミットを削除する
git revert	コミットの内容を打ち消すコミットをする



## 課題4-1

- 1) 「DailyReport」のリポジトリを作成しなさい
  - a) `> cd Desktop/DailyReport`
  - b) `> git init`
- 2) 作業ツリーの状態を表示しなさい
  - a) `> git status`

# バージョン管理システム講座

git status ファイルの状態を表示

表示	表示内容
Changes to be committed:	コミットされるファイル
Unmerged paths:	競合が解決されていないファイル
Changes not staged for commit:	コミットされないファイル
Untracked files:	Gitで管理されていないファイル



## 課題4-2

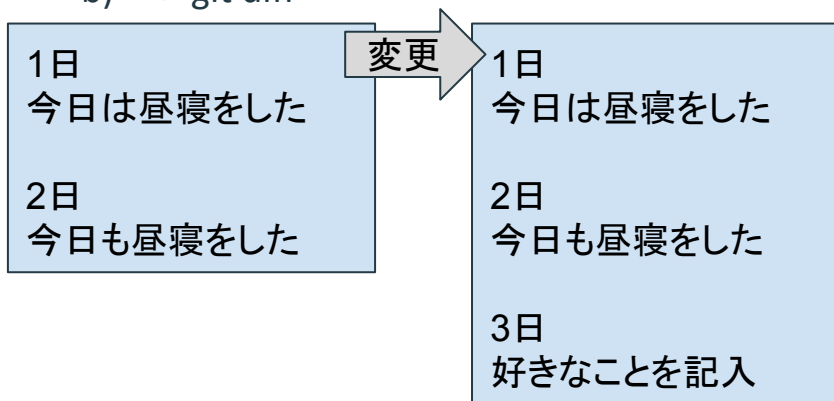
- 1) 「April.txt」をインデックスに追加し、コミットしなさい
  - a) > git add April.txt
  - b) > git status
  - c) > git commit -m "commit messages"
  - d) > git log
- 2) 「May.txt, June.txt, July.txt」をインデックスに追加し、コミットしなさい
  - a)
  - b)
  - c)
  - d)

## 課題4-2

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

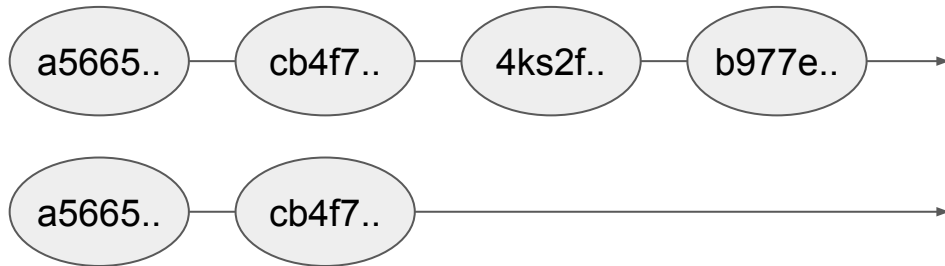
b) `> git diff`



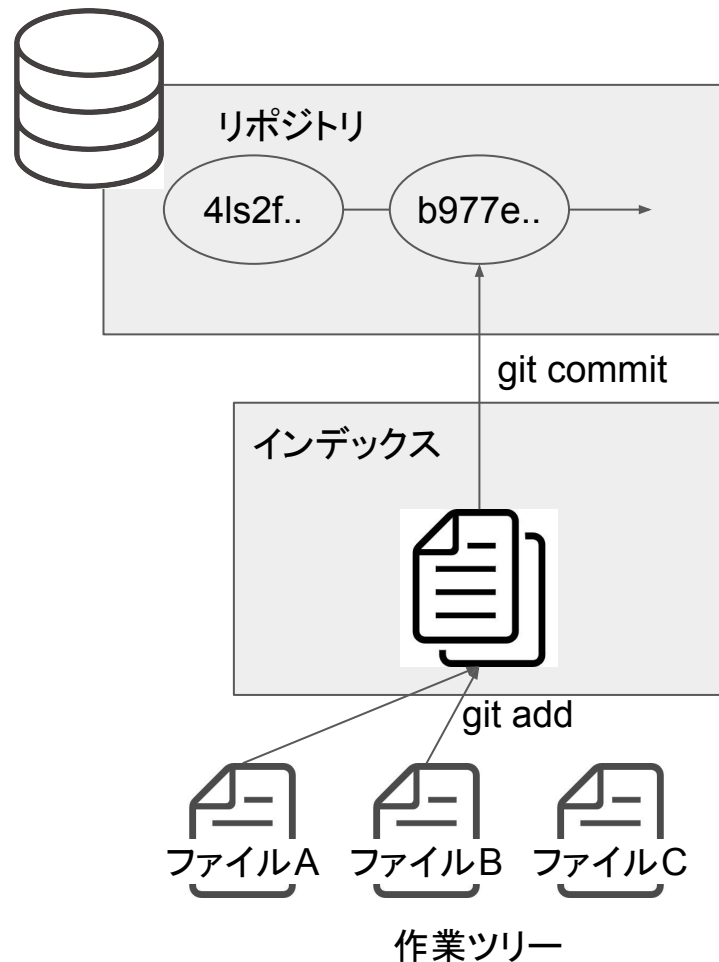
2) 変更した「April.txt」をコミットしなさい

# バージョン管理システム講座

git reset <mode> <commit> リポジトリのコミットを削除する

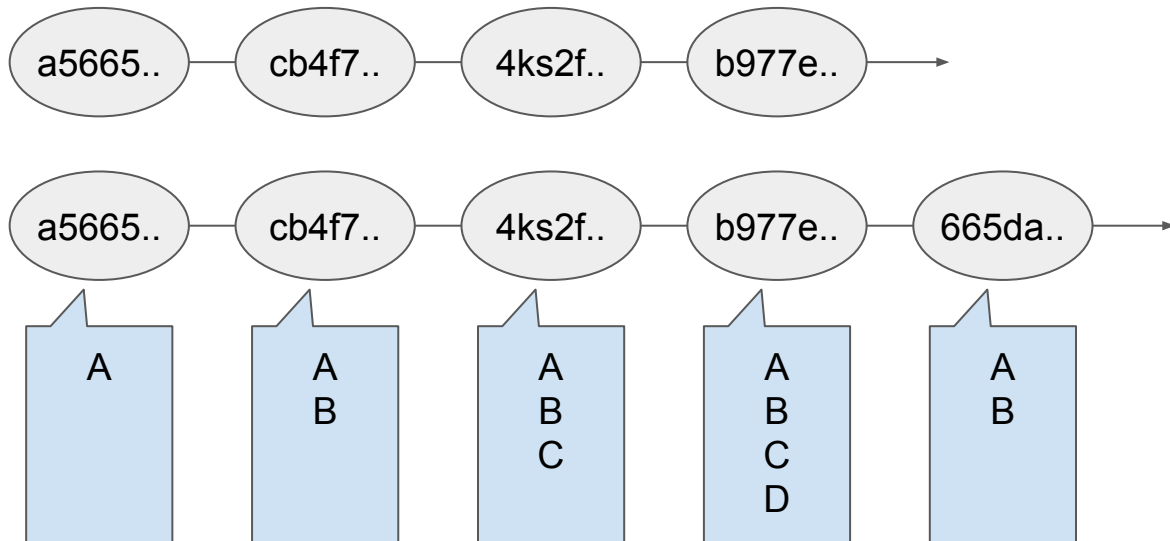


<mode>	コミット	インデックス	作業ツリー
--soft	O	X	X
--mixed	O	O	X
--hard	O	O	O



# バージョン管理システム講座

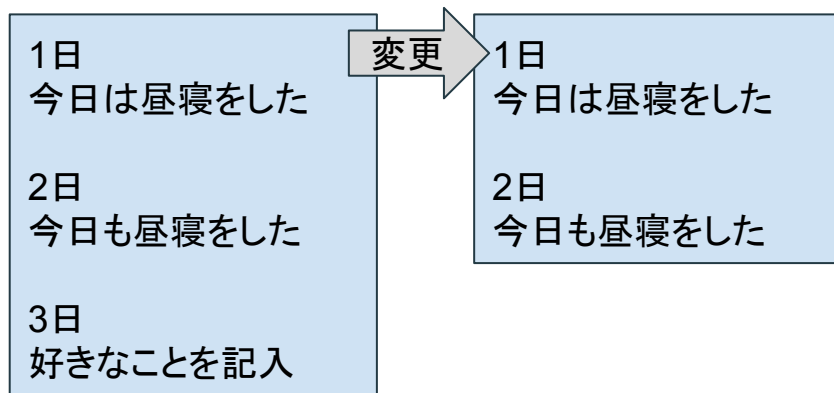
git revert <mode> <commit> コミットの内容を打ち消すコミットをする



## 課題4-3

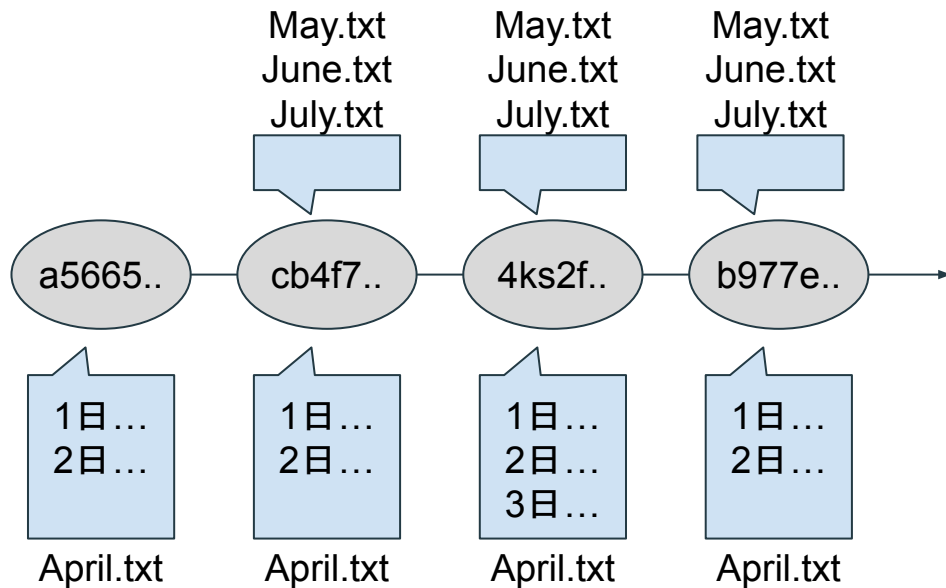
1) 「git revert」を用いて、右側のファイルの状態に戻しなさい

- a) > git log
- b) > git revert <commit ID> [--no-edit] 又は> git revert HEAD
- c) > git status
- d) > cat April.txt



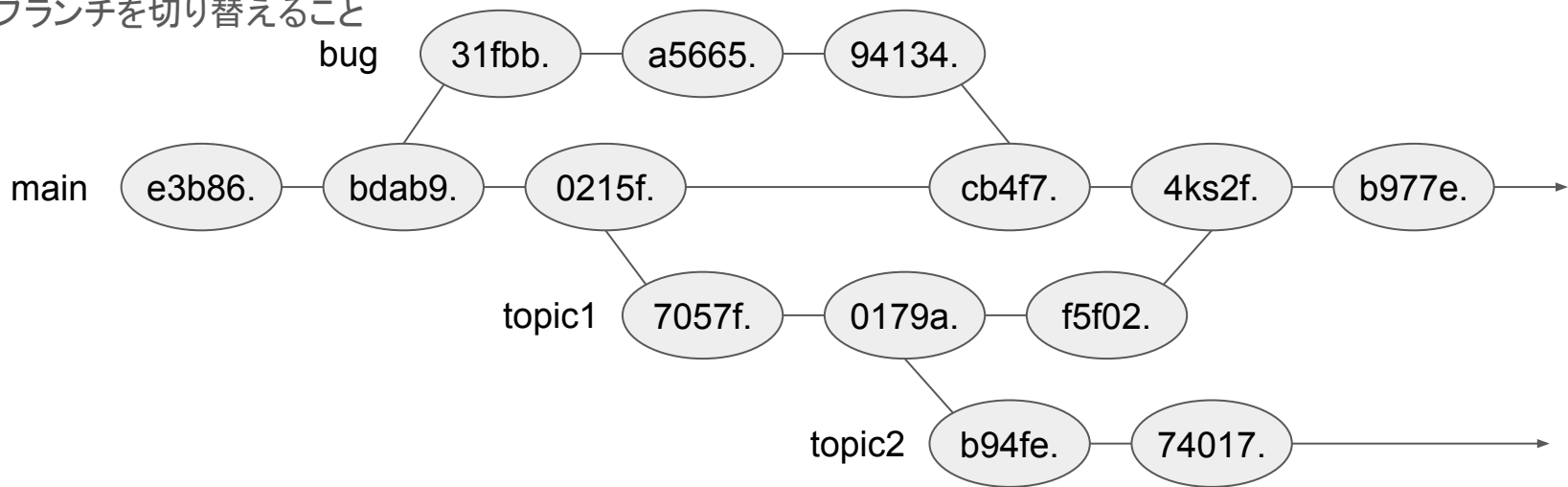
# 課題 解説

- a) > git log
- b) > git revert <commit ID>
- c) > git status
- d) > cat April.txt



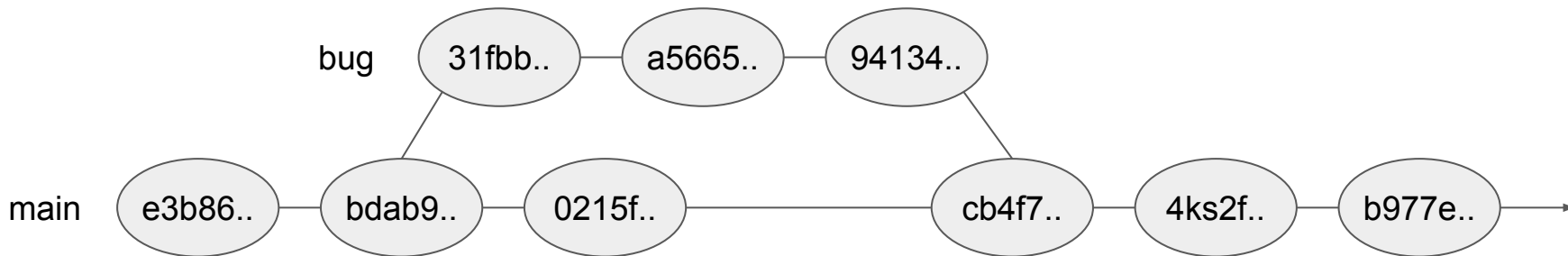
# バージョン管理システム講座

- ブランチ
  - 幹のように並行に2つのバージョンを管理するための機能
- マージ
  - 競合を解決すること
- チェックアウト
  - ブランチを切り替えること



# バージョン管理システム講座

コマンド	意味
<b>git branch</b>	ブランチの表示、作成、名前変更、削除を行う
<b>git checkout</b>	ブランチを切り替える
git stash	作業ツリーを一時的に保存
git show-branch	ブランチの状態を表示
<b>git merge</b>	マージする

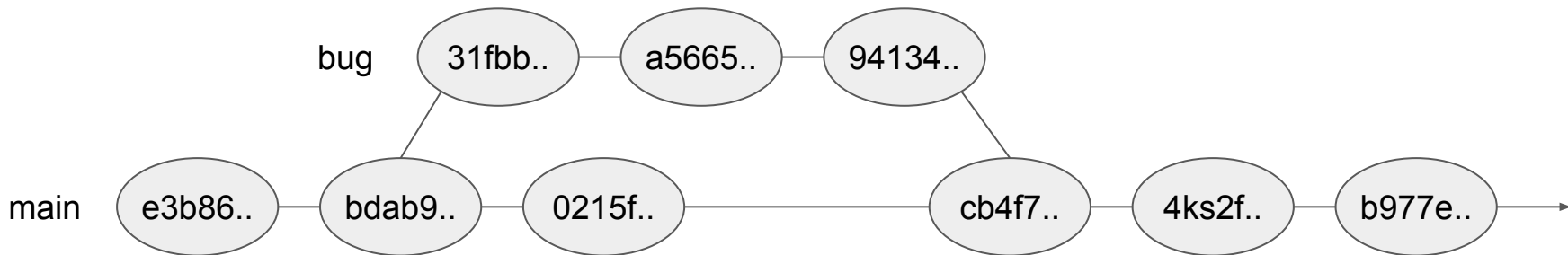




# バージョン管理システム講座

git branch <option> ブランチの表示、作成、名前変更、削除を行う

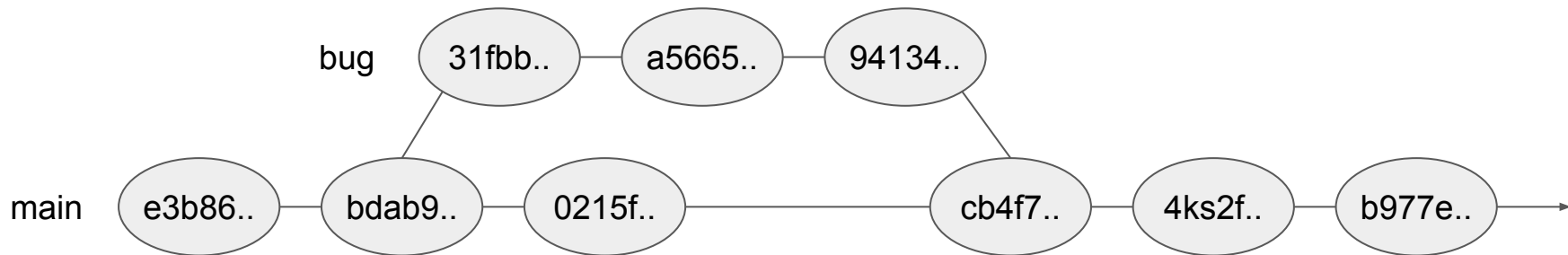
<option>	意味
指定なし	ブランチを一覧表示
<branchName>	ブランチを作成
-m <oldBranch> <newBranch>	ブランチの名前を変更
-d <branchName>	ブランチを削除

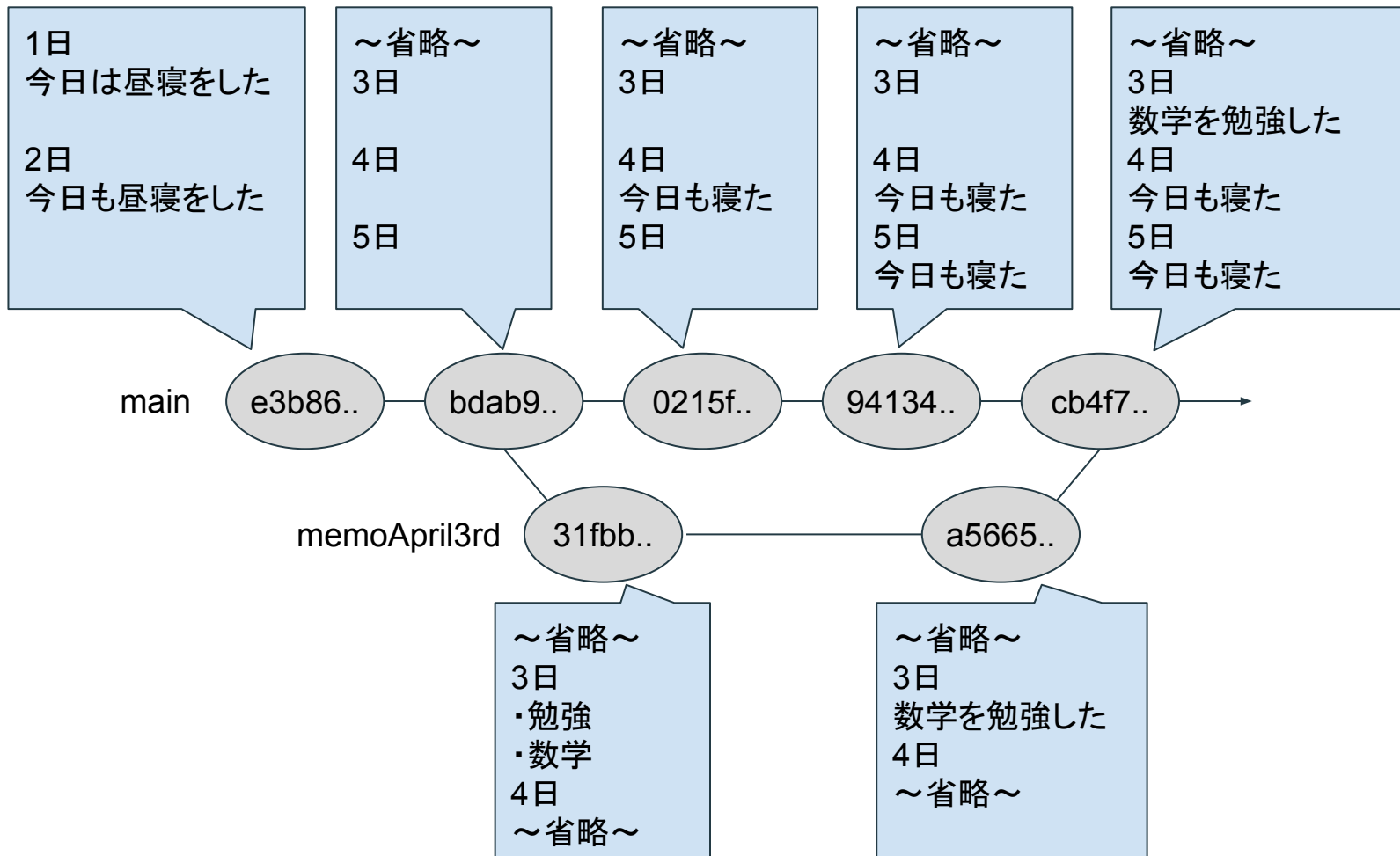


# バージョン管理システム講座

git stash <option>

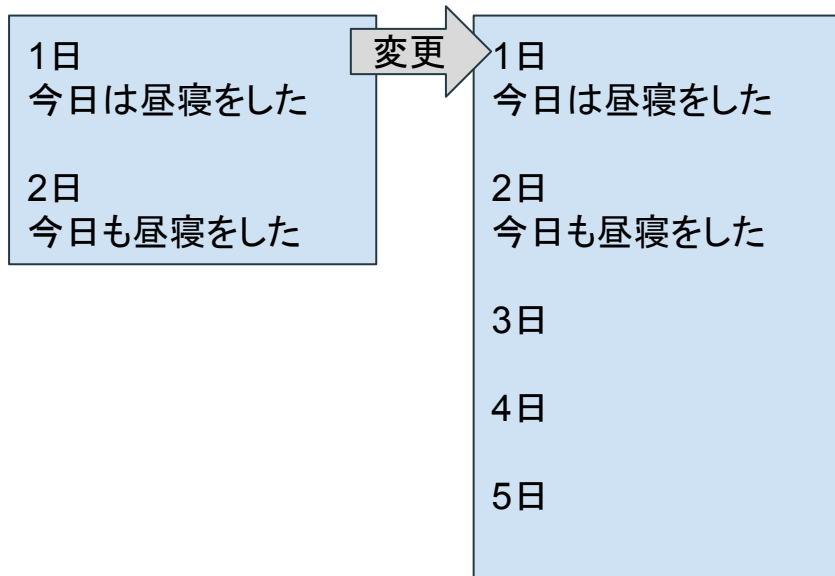
<option>	意味
指定なし	作業ツリーを一時的に保存
list	保存したstashを一覧表示
pop <stash ID>	stashを作業ツリーに適応
clear <stash ID>	stashを削除





## 課題4-4

1) 「April.txt」を右側のように変更し、コミットなさい



1日  
今日は昼寝をした

2日  
今日も昼寝をした

～省略～  
3日

4日

5日

～省略～  
3日

4日  
今日も寝た

5日

～省略～  
3日

4日  
今日も寝た

5日  
今日も寝た

～省略～  
3日  
数学を勉強した

4日  
今日も寝た

5日  
今日も寝た

main

e3b86..

bdab9..

0215f..

94134..

cb4f7..

memoApril3rd

31fbb..

a5665..

～省略～  
3日  
・勉強  
・数学

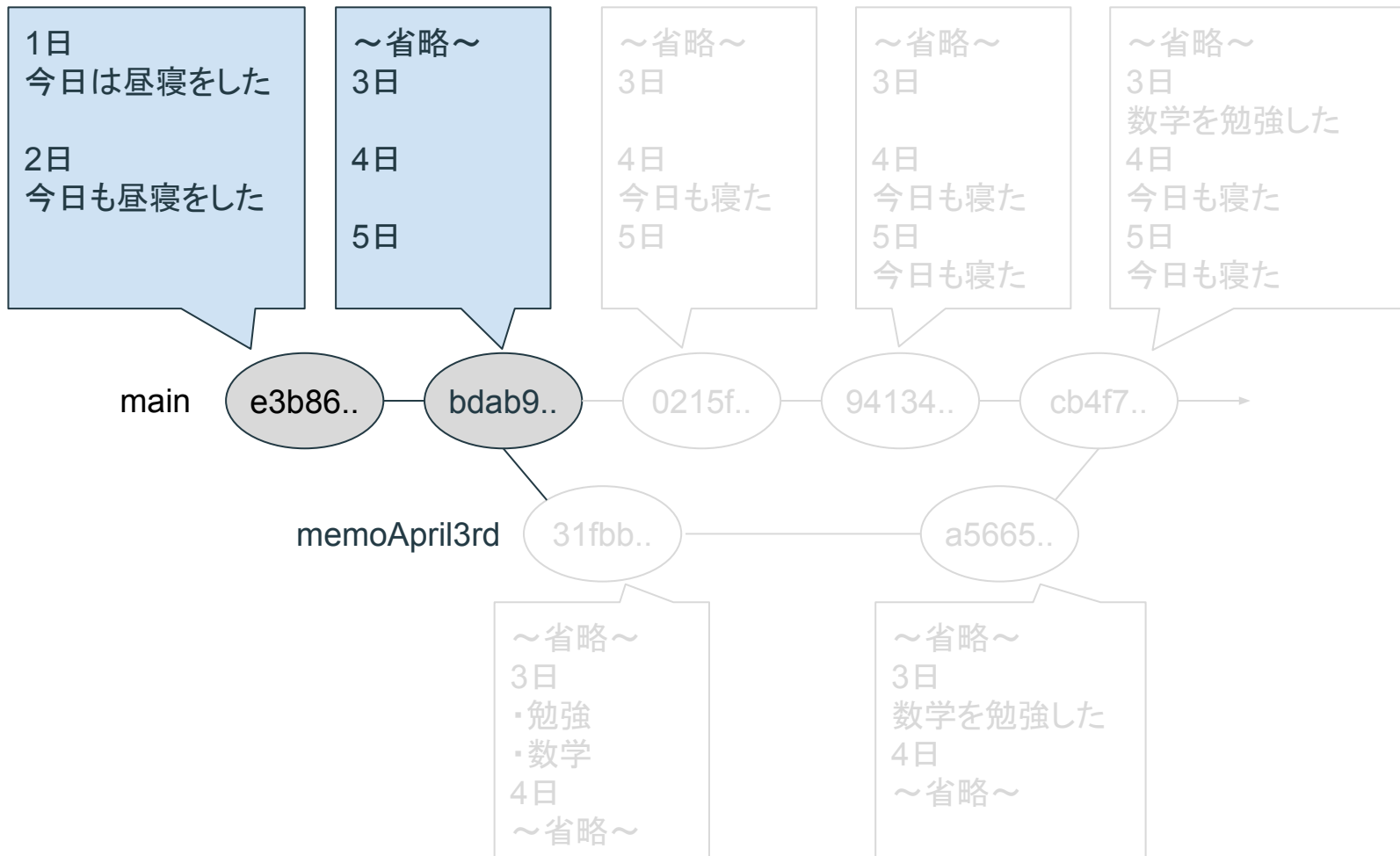
4日  
～省略～

～省略～  
3日  
数学を勉強した

4日  
～省略～

## 課題4-4

- 1) 現在のブランチを確認しなさい
  - a) `> git branch`
- 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「memoApril3rd」というブランチを作成しなさい
  - a) `> git branch memoApril3rd`
  - b) `> git branch`



## 課題4-4

1) 「memoApril3rd」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) > git checkout memoApril3rd
- b) > git branch
- c) ファイルを編集
- d) > git add .
- e) > git commit -m "commit messages"

～ 省略 ～

3日

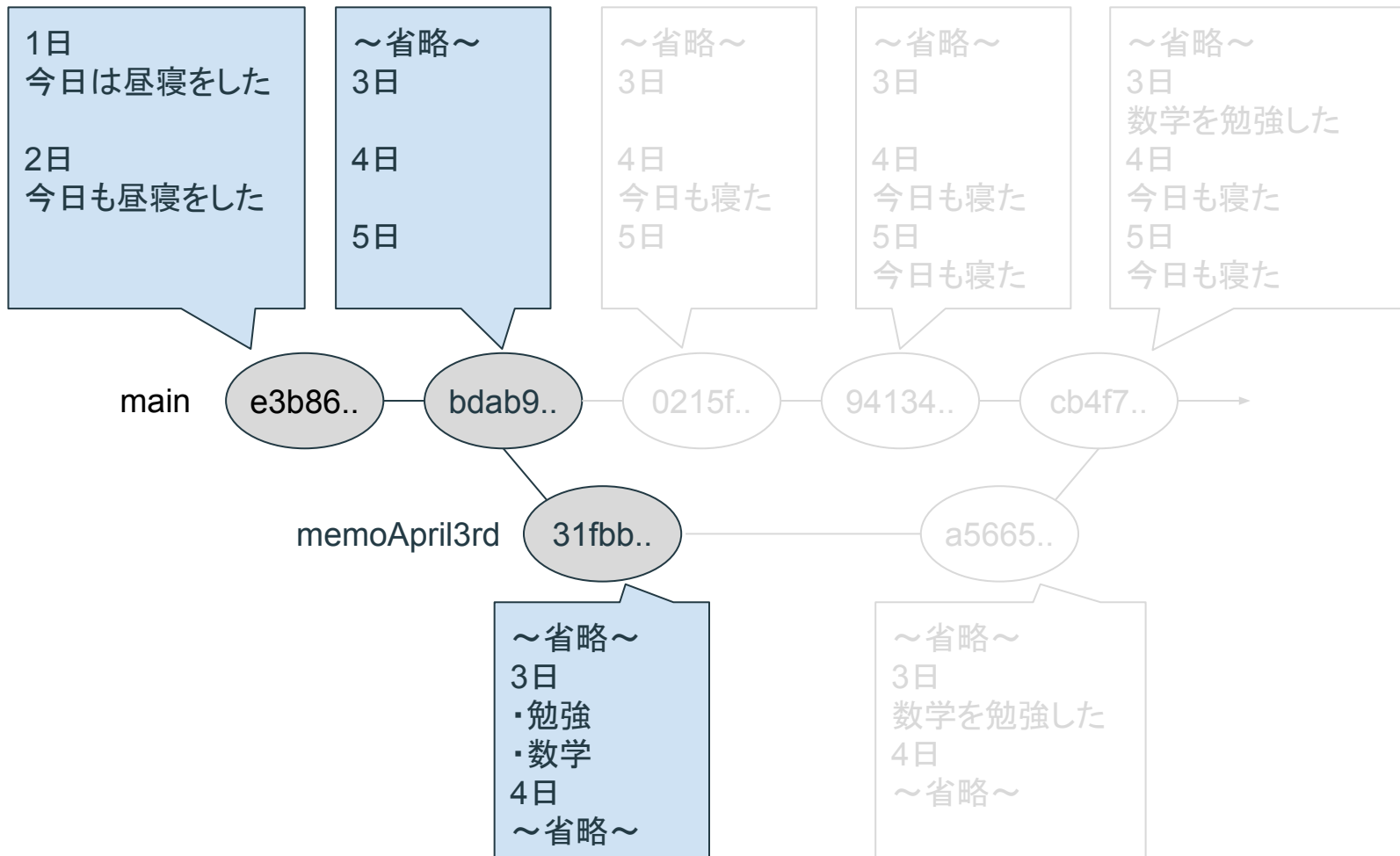
・勉強

・数学

4日

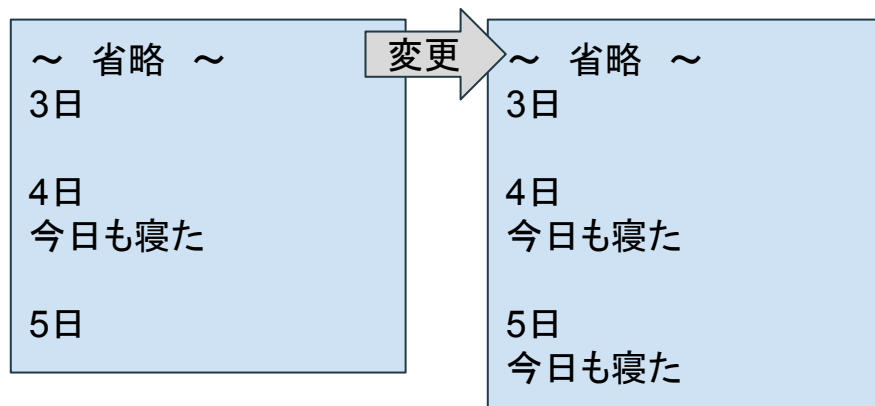
～ 省略 ～

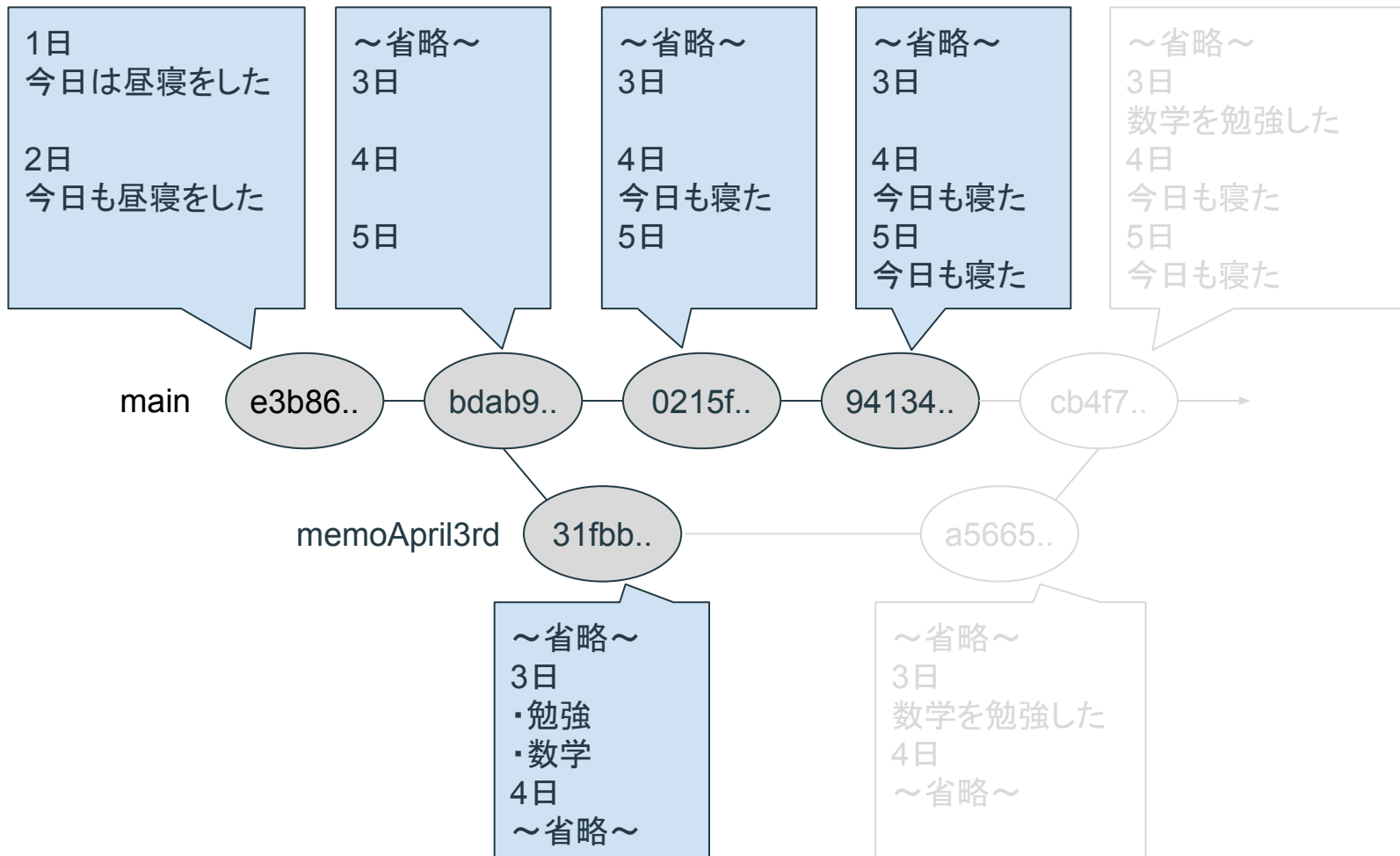




## 課題4-4

- 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) > git checkout main
  - b) ファイルを編集
  - c) > git add .
  - d) > git commit -m "commit messages"
- 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

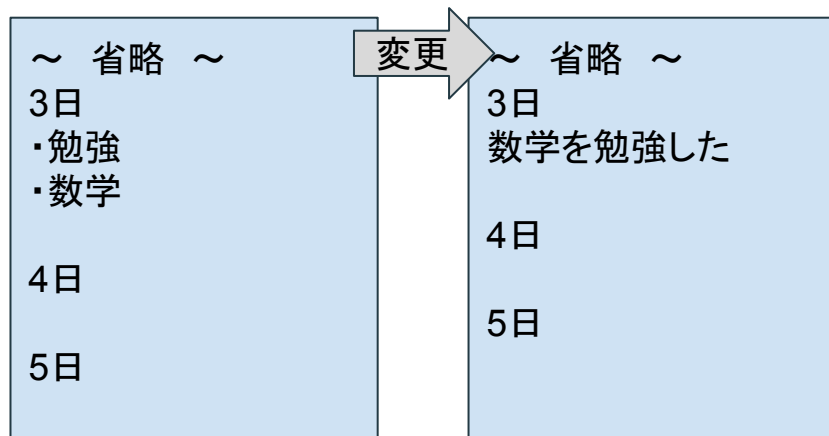


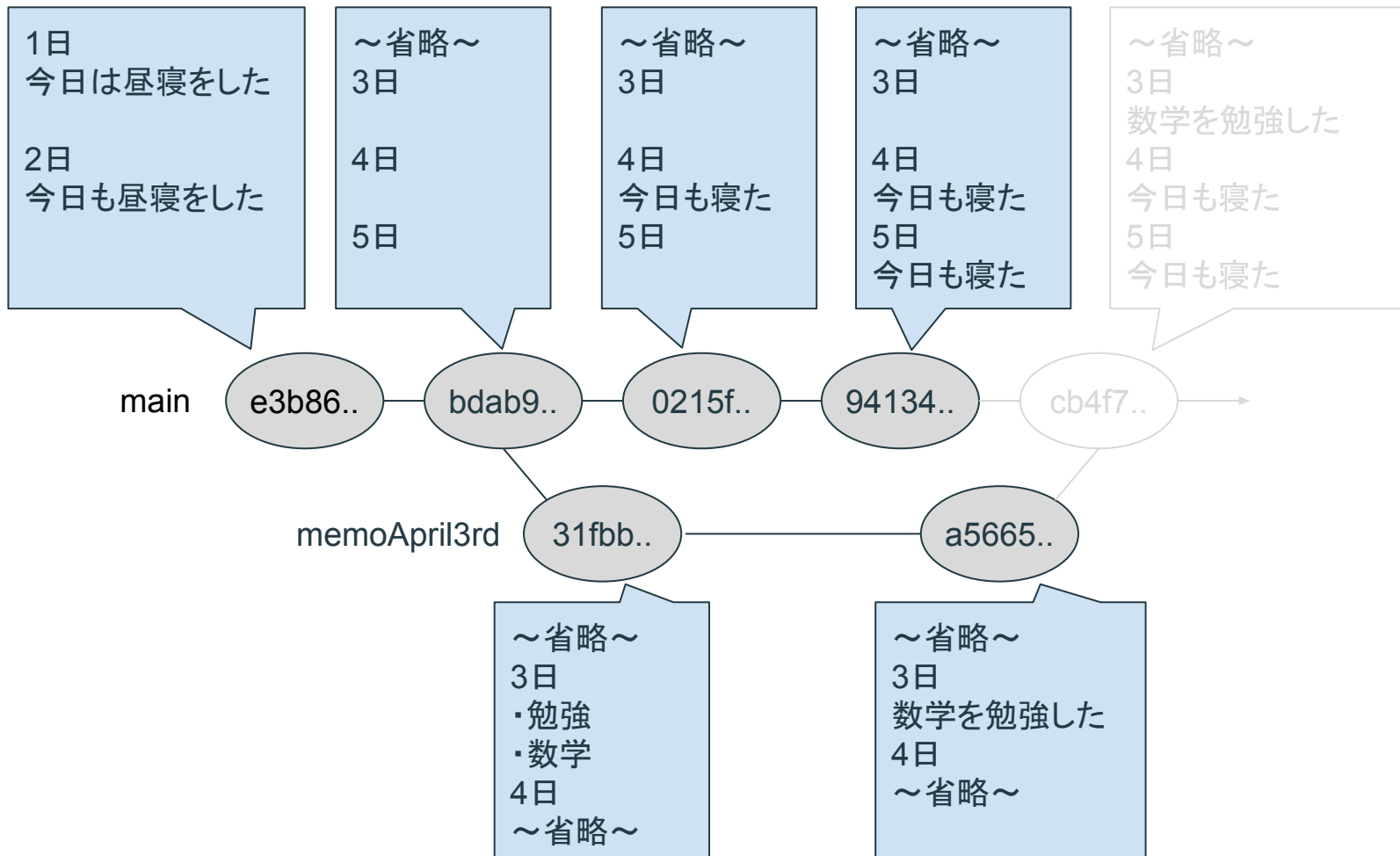


## 課題4-4

1) 「memoApril3rd」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)

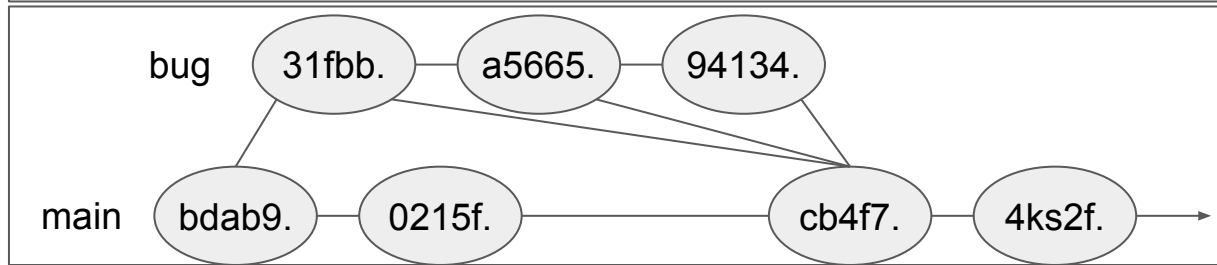
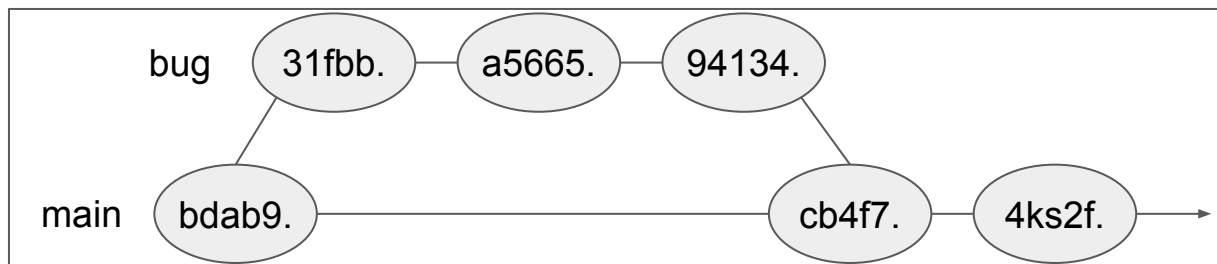




# バージョン管理システム講座

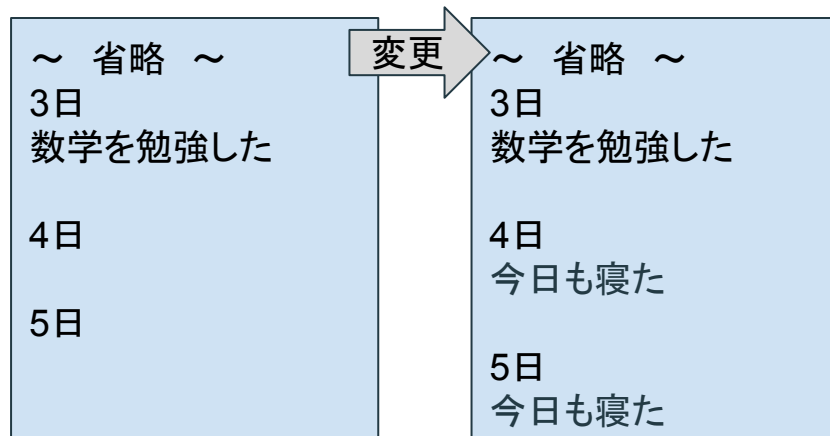
git merge <option> マージを行う ※マージされる側のブランチにチェックアウトする必要がある

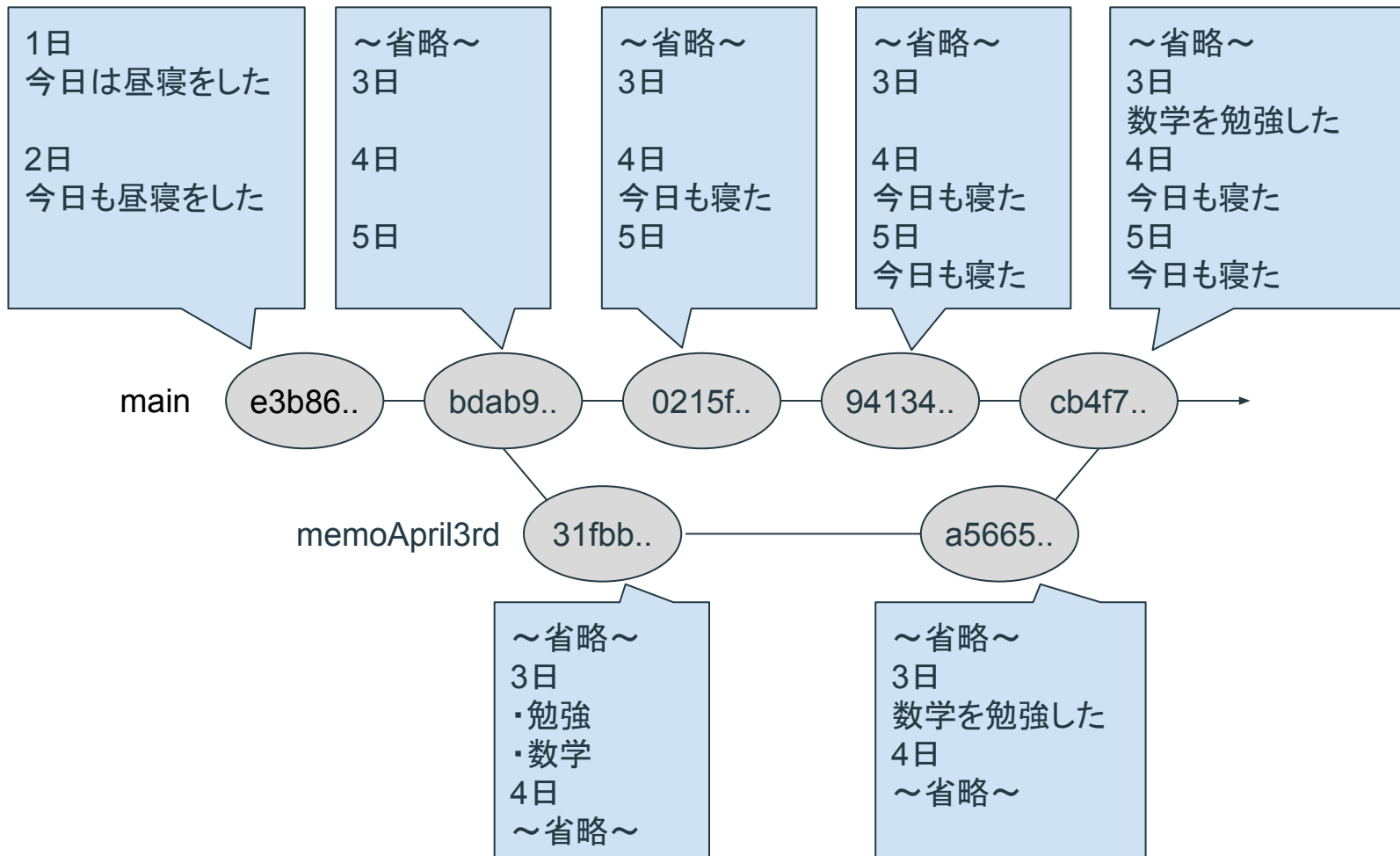
<option>	意味
--no-ff	fast-forwardマージを行わない
--squash	取り込むブランチのコミットをマージコミットのみにする



## 課題4-5

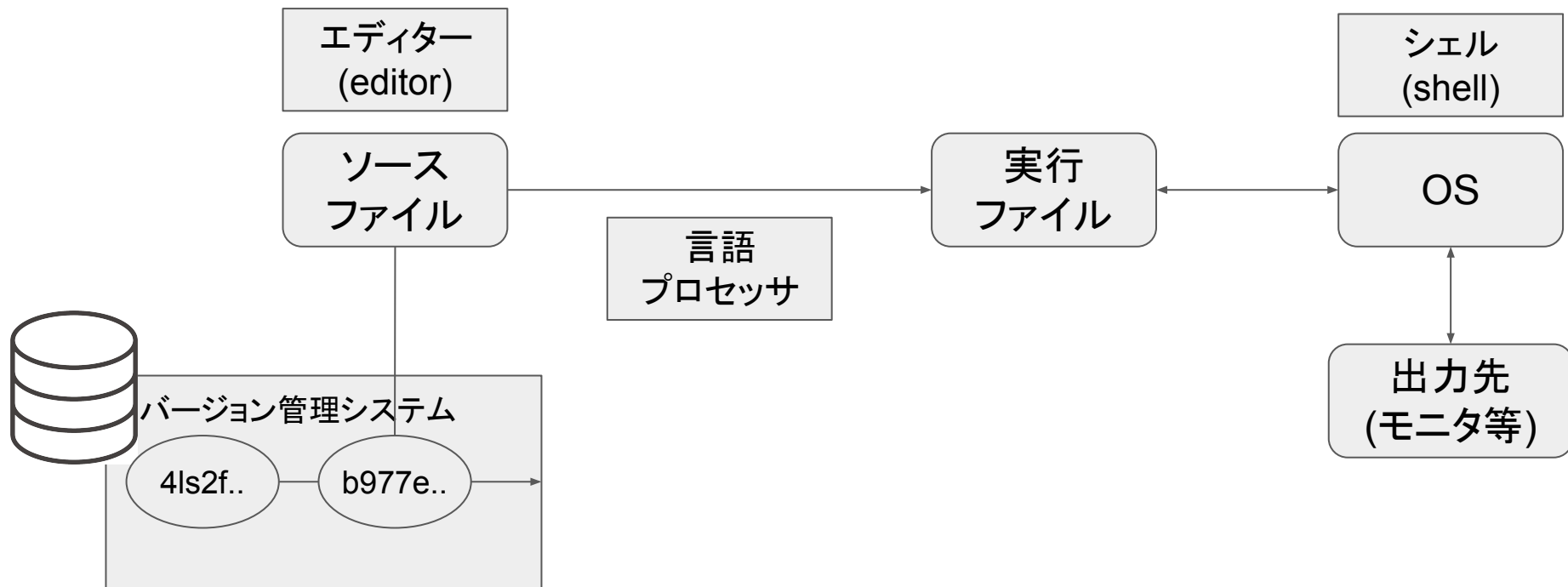
- 1) 「main」ブランチに「memoApril3rd」をマージしなさい
  - a) > git checkout main
  - b) > git branch
  - c) > git merge --no-ff memoApril3rd
- 2) マージした履歴を「git log」で確認しなさい
  - a) > git log







# 個人開発での一般的な開発環境



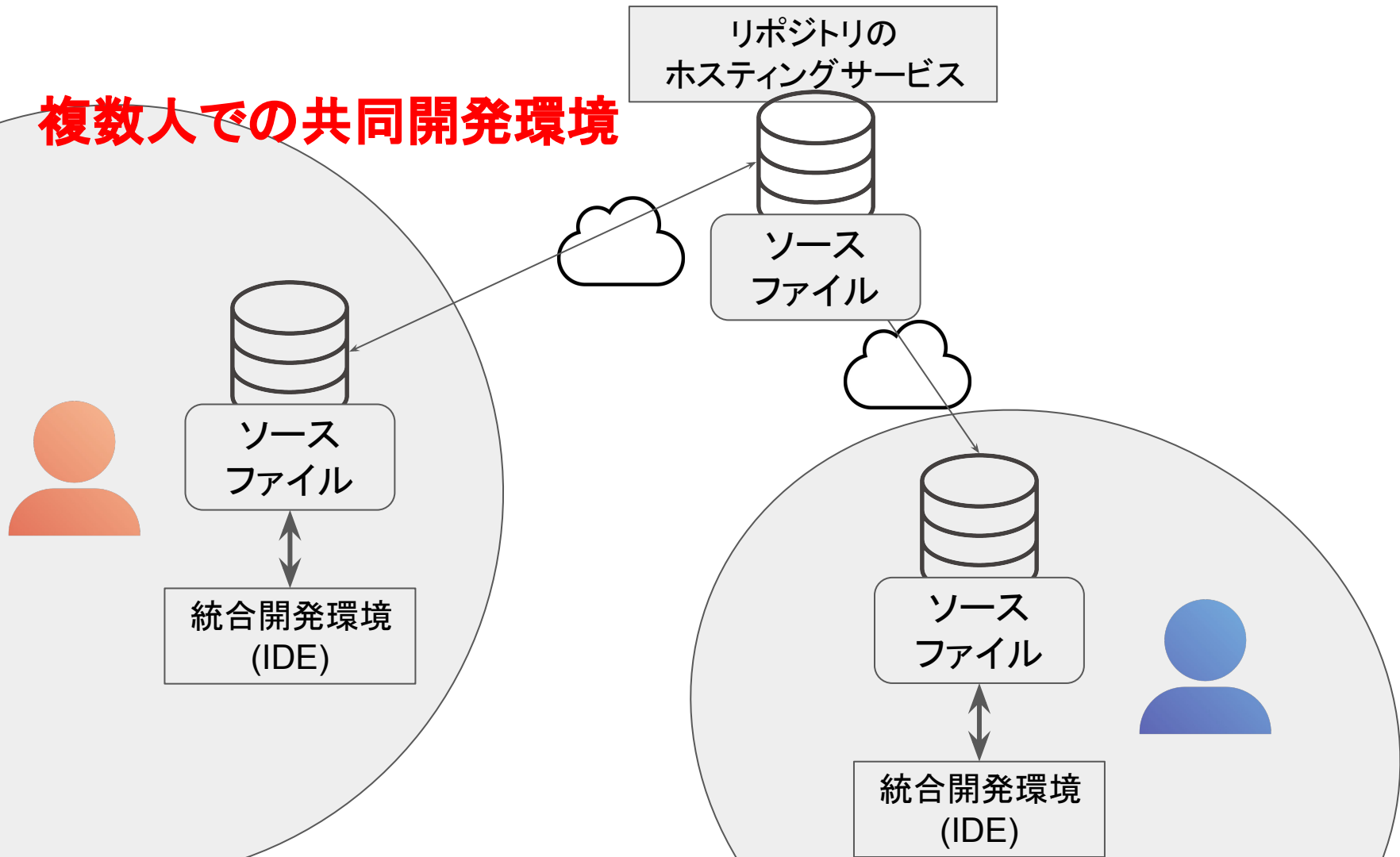
# 統合開発環境(IDE)

- 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

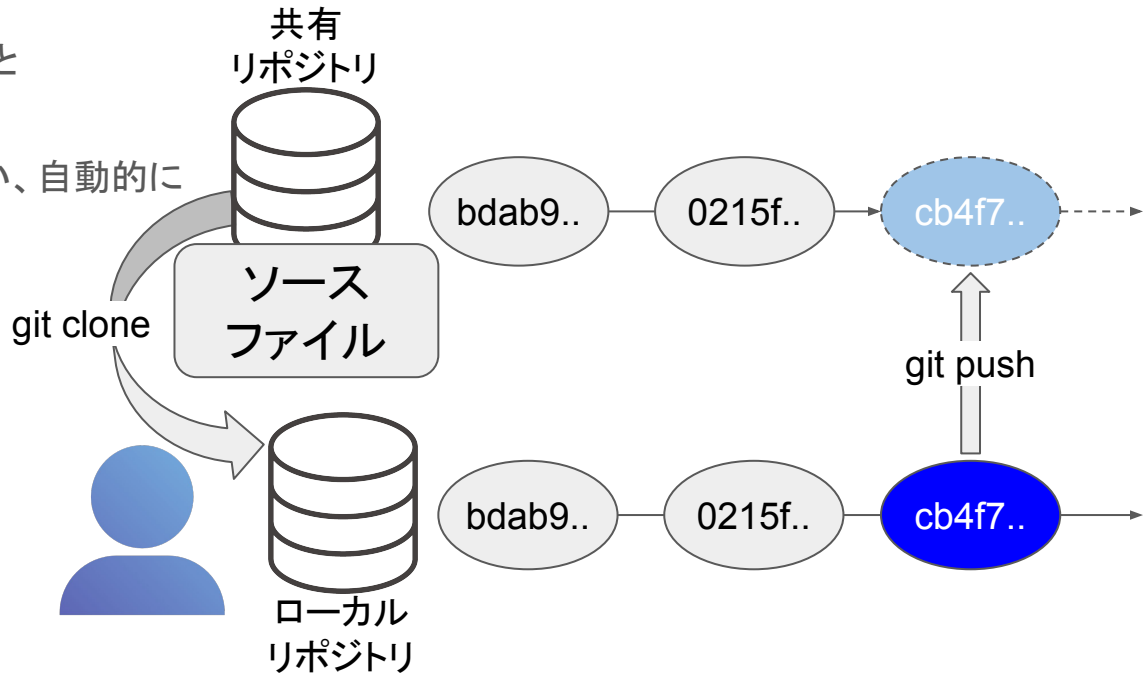


# リポジトリのホスティングサービス

- 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
  - Gitのリポジトリのホスティングサービスと言ったら、「GitHub」しか思い浮かないと言っても過言ではない
  - 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

- 共有リポジトリ(≡リモートリポジトリ)
  - 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
- 競合
  - 同じ場所を同時に編集してしまい、自動的に解決できない状態
- マージ(merge)
  - 競合を解決すること



# 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

- 空のリモ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) 共有リポジトリを以下の設定で作成しなさい
  - a) Repository Name : DailyReport
  - b) Kind of Repository : Private

# GitHub講座

コマンド	意味
git clone	リポジトリを複製する
git remote	リモートリポジトリを設定する
git pull	共有リポジトリから変更を取得し、自動でマージを行う 「fetch」「merge」を行うのと同じ
git fetch	共有リポジトリの変更を取得する
git push	共有リポジトリへローカルリポジトリを送信する
git merge	リモートリポジトリを設定する

# GitHub講座

git remote <option> リモートリポジトリを設定する

<option>	意味
-v	設定されているリモートリポジトリを表示
add <repositoryName> <url>	リモートリポジトリを設定する
set-url <repositoryName> <url>	リモートリポジトリを更新する

# GitHub講座

git push <option> リモートリポジトリを設定する

<option>	意味
<b>&lt;repositoryName&gt; &lt;localBranch&gt;</b>	<b>上流ブランチにpush</b>
<repositoryName> <localBranch>:<remoteBranch>	ローカルのブランチをリモートのブランチに push
-u <repositoryName> <localBranch>	上流ブランチを設定する
--tag <repositoryName>	タグをpush
指定なし	設定されている情報で push

# GitHub講座

git fetch<option> リモートリポジトリを設定する

<option>	意味
<b>省略</b>	<b>現在の上流ブランチから履歴を取得</b>
<repositoryName>	すべてのリモートブランチの履歴を取得
<repositoryName> <remoteBranch>	指定したリモートブランチの履歴を取得
<repositoryName> <remoteBranch>:<localBranch>	リモートブランチの履歴をローカルブランチに反映

# GitHub講座

git merge <option> リモートリポジトリを設定する

<option>	意味
<branchName>	指定したブランチを現在いるブランチにマージする
--continue	競合を解決し、git addの後マージの処理を続行させる (git commitと同じ処理)
--abort	マージ処理を中止する

# GitHub講座

git pull <option> リモートリポジトリを設定する

<option>	意味
----------	----

## 課題5-2

- 1) 「DailyReport」のリモートリポジトリとして作成した共有リポジトリを設定しなさい
  - a) `> git remote add origin https://.....`
  - b) `> git remote -v`
- 2) ローカルリポジトリを共有リポジトリにプッシュしなさい
  - a) `> git push origin main`
  - b) GitHubの共有リポジトリのページを見てみよう！



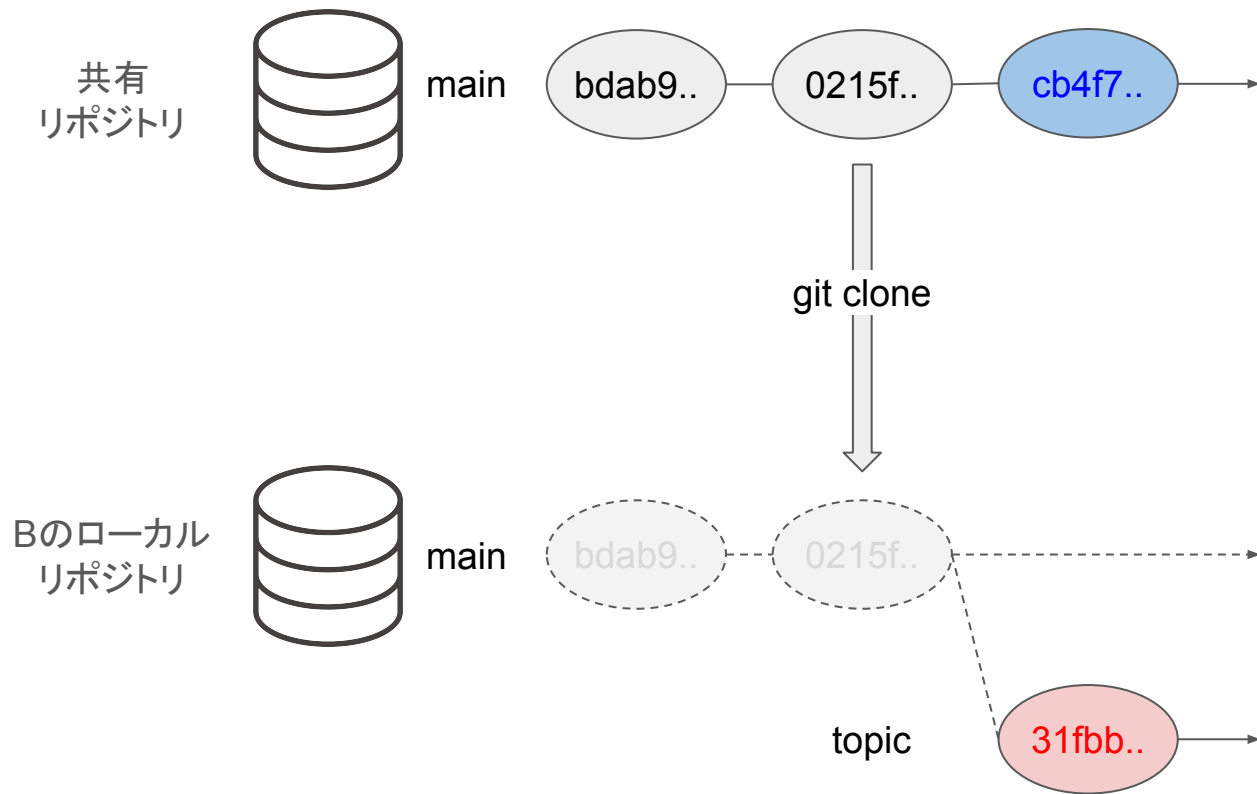
## 課題5-2

- 1) 「April.txt」の6日以降を自由に変更し、コミットし、共有リポジトリに反映させなさい
  - a) ファイルを編集
  - b) `> git add .`
  - c) `> git commit -m "commit message"`
  - d) `> git push origin main`
  - e) GitHubの共有リポジトリのページを見てみよう！

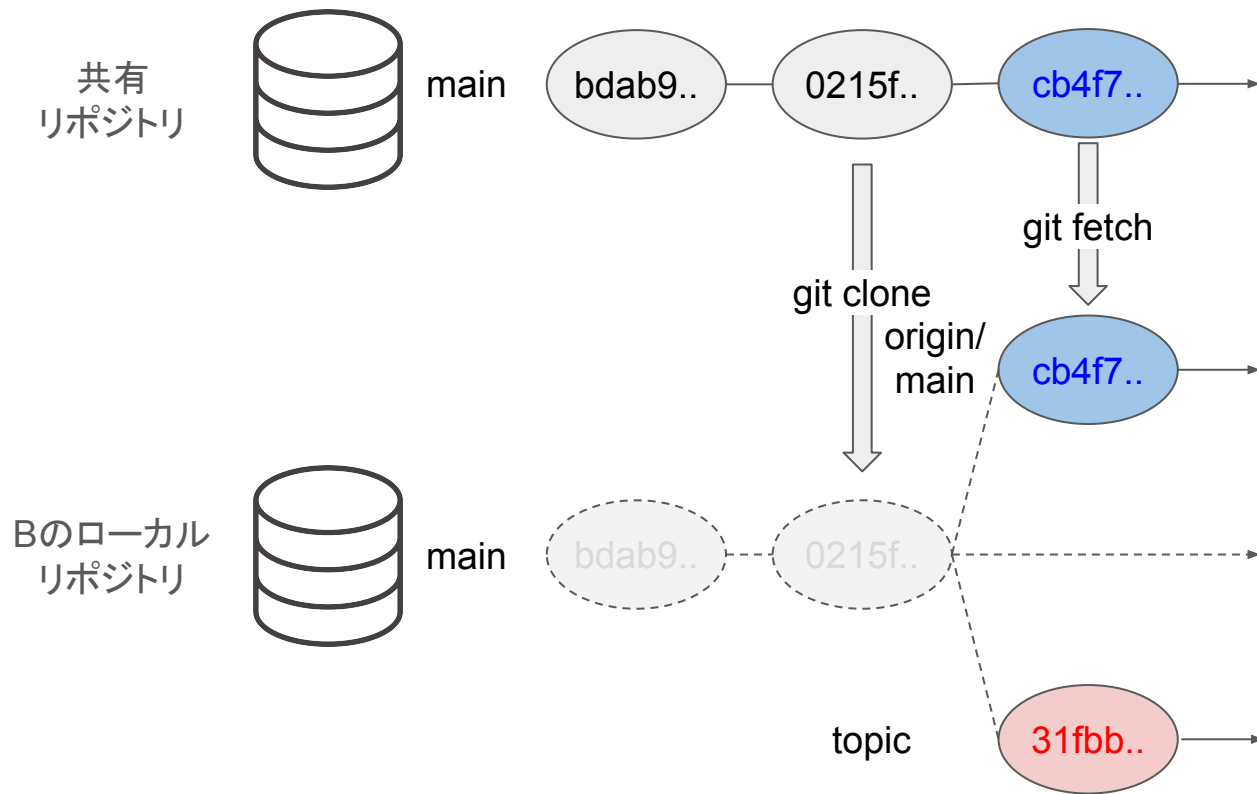
# GitHub講座

- 共同開発のルール(ルールは今回の講座に限る)
  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し、作業を行う
  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

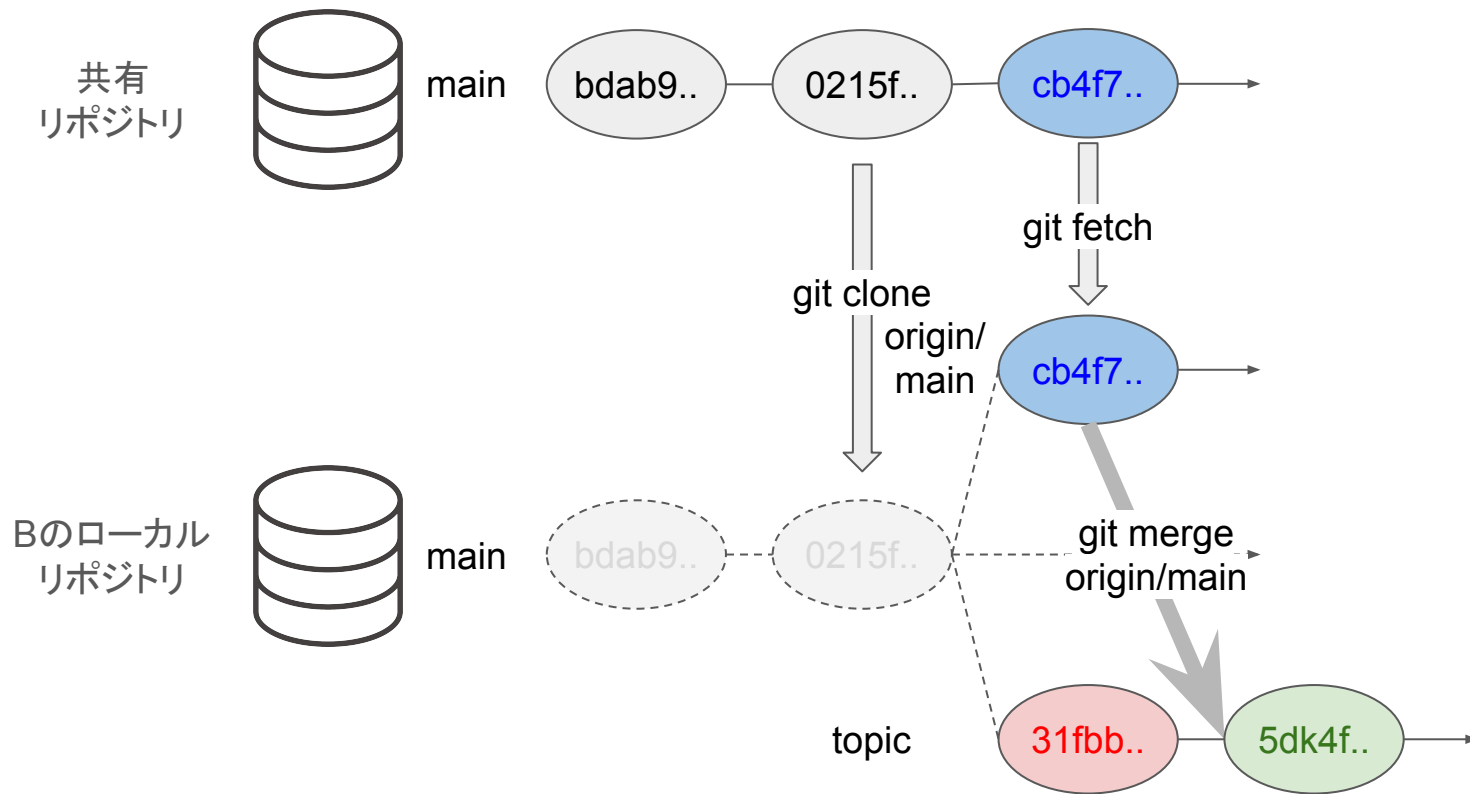
# GitHub講座



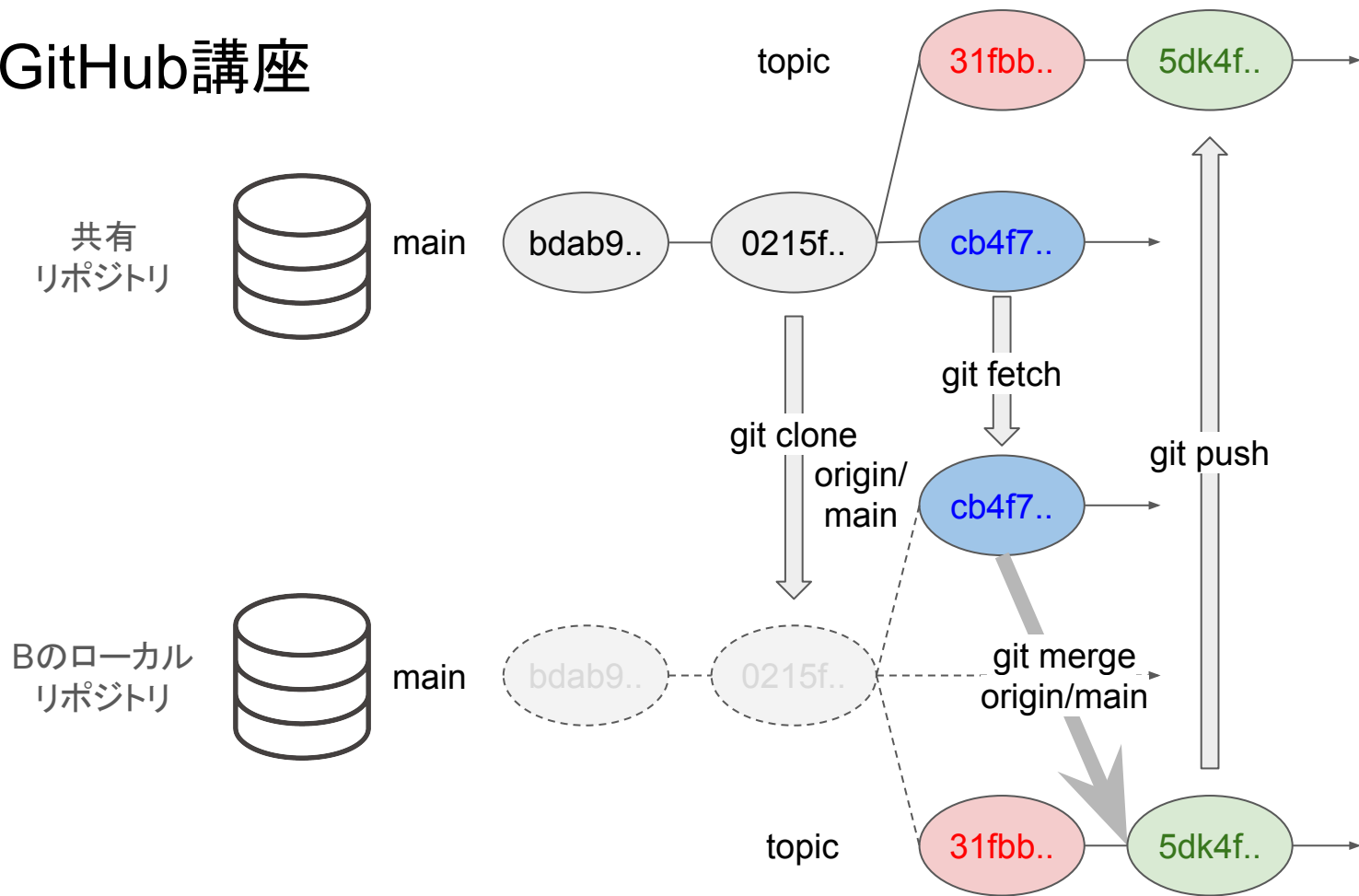
# GitHub講座



# GitHub講座



# GitHub講座

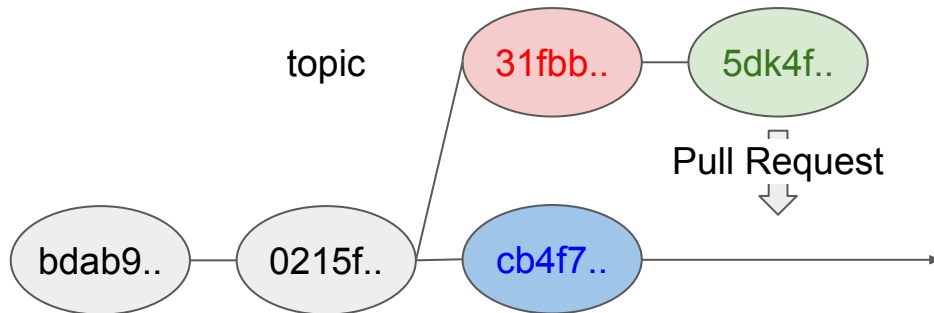


# GitHub講座

共有  
リポジトリ



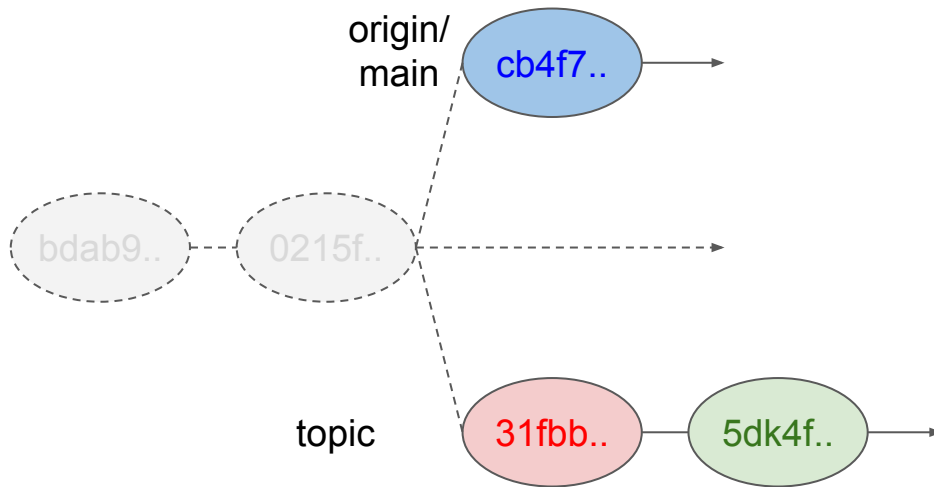
main



Bのローカル  
リポジトリ



main



# 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めたりしたくない
  - 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 )を相互に行う
  - マージを明示的に行う



# GitHub講座

- Pull Requestの送信方法

- GitHub上で、自分が作成したブランチに移動
- 「Compare & pull request」ボタン又は「Contribute」->「Open pull request」ボタンを押す
- 右側にある「Reviewers」にレビューアーを指定する
- 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
- レビューアーに「Approve」してもらい、「Merge pull request」ボタンを押す

- レビューアーの対応方法

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

# GitHub講座

- Pull Requestの送信方法

- GitHub上で、自分が作成したブランチに移動
- 「Compare & pull request」ボタン又は「Contribute」->「Open pull request」ボタンを押す
- 右側にある「Reviewers」にレビューアーを指定する
- 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
- レビューアーに「Approve」してもらい、「Merge pull request」ボタンを押す

- レビューアーの対応方法

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

# 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/JointDevelopmentEnvironmentLesson-BasedCodeForGolang.git>

## 課題5-3

1) 講師のGitHubのリモトリポジトリをcloneしなさい

a) > git clone

[https://github.com/sotarokashiuchi/JointDevelopmentEnviromentLesson-BasedCodeForGol  
ang.git](https://github.com/sotarokashiuchi/JointDevelopmentEnviromentLesson-BasedCodeForGol<br/>ang.git)

b) > git status

2) 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) > git branch Name

b) > git checkout Name

c) ファイル編集

d) > git add

e) > git push origin nicName

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

## 課題5-3

- 1) 作業用ブランチ(ニックネームブランチ)にリモートのmainブランチをマージし、プッシュしなさい
  - a) > git branch
  - b) > git fetch
  - c) > git merge origin/main
  - d) > git push origin nicName

## 課題5-3

### 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

# GitHub講座

- 競合の解決方法(リモートリポジトリのmainでの競合)
  - > git fetch
  - > git merge origin/main
  - 競合が発生 (git statusで競合しているファイルを確認できる )
  - ファイルを編集して競合を解決
  - > git add
  - > git commit
  - ※競合を解決後のコミットにはコミットメッセージは不要
  - > git push origin <branchName>

```
<<<<<<< HEAD
(ローカルリポジトリの内容)
=====
(共有リポジトリの内容)
>>>>>>> <commitID>
```

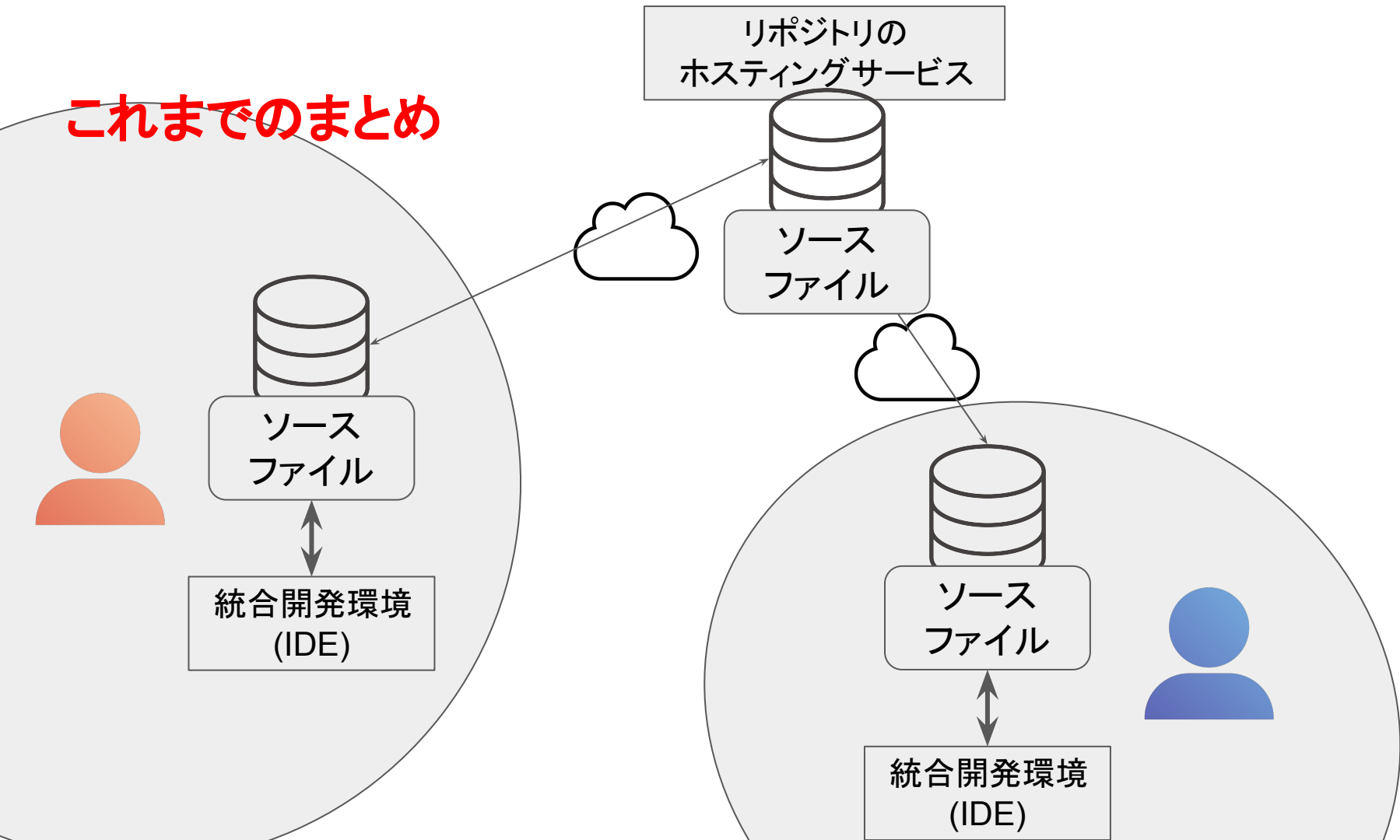
## 課題5-4

- 1) 「README.md」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
- 2) 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

完成！！



## これまでのまとめ



# ゆめくじら紹介

ご清聴ありがとうございました！！

# GitHub

ORIGINTOHA

nvim tab tabhokan

git pull の時競合が発生したときだけコミットが作成されるのか？