

プログラミングD

# Java講義 第4回:バージョン管理ツール

肥後芳樹

# プログラミング中によくあること

- 高速化しようとしたがうまくいかなかった...
  - 元に戻したいがバックアップしていない...
  - バックアップはしていたけど、大量に有りすぎてどれを使えばいいかわからない...
- デバッグのため修正を繰り返していたら、今まで動いていた部分も動かなくなってしまった...



過去の動いていた状態に戻したい！

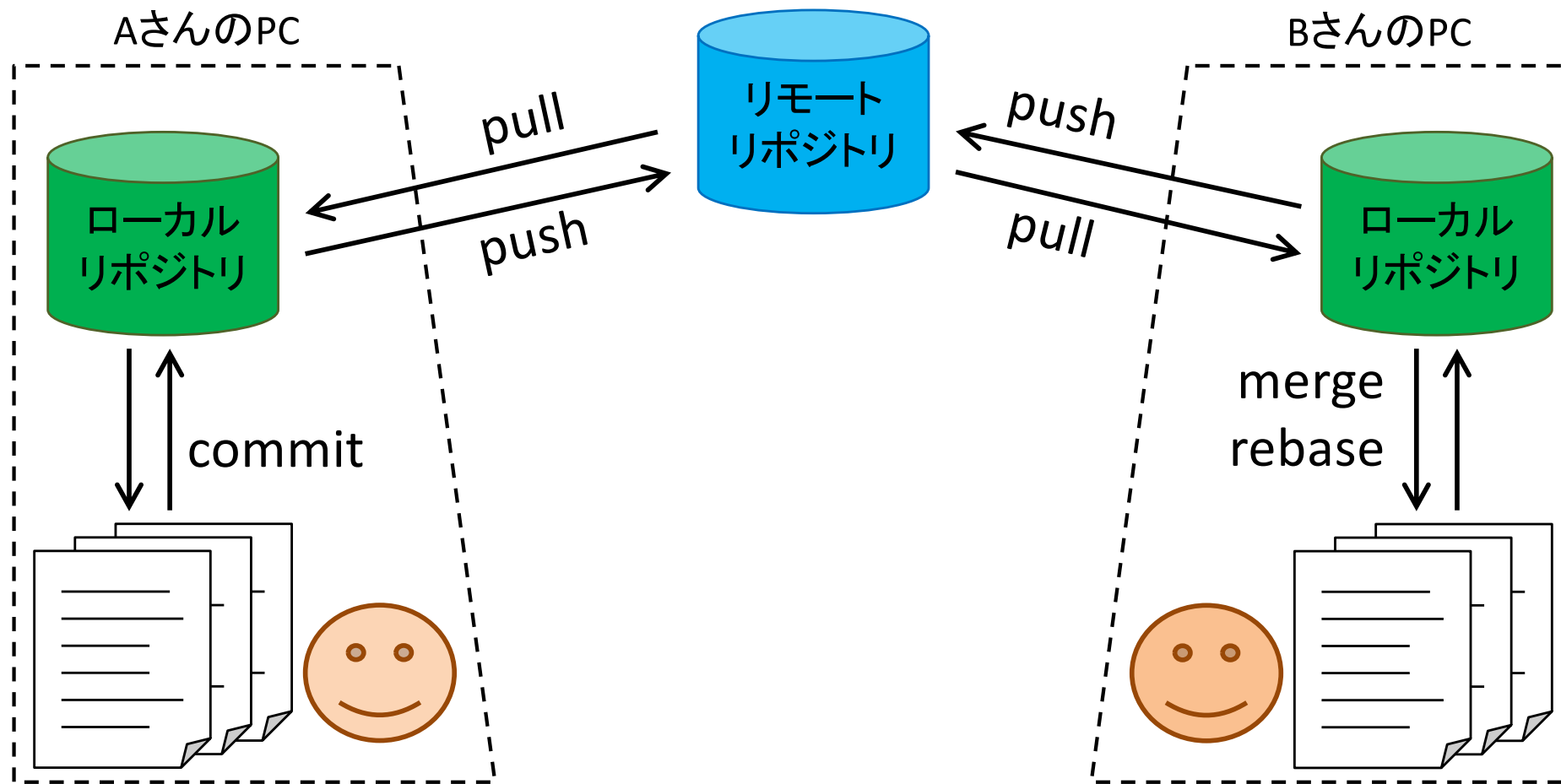
# バージョン管理ツール

- プロダクト(ソースコードやドキュメント)の変更履歴を管理するツール
- 過去に加えられた変更を記録している
- コメントを入力しておけば, 変更の意図や変更の概要も把握できる
- 過去のバージョンを取り出したり, 現状との差分を確認したりできる
- 複数箇所で作業をしてもソースコードを一元管理できる
- 複数人で開発する場合に, 手作業による集約作業(各人の変更点をまとめる)をしなくていい
  - 競合が発生した場合は手作業が必要



- 分散型バージョン管理ツール
- Linuxカーネルのソースコード管理のために,  
Linus Torvaldsによって開発された
- Gitクライアントの実装
  - Windows: TortoiseGit, cygwin上のgit
  - Eclipse: EGit

# gitのイメージ図



# ソースコードは3箇所に存在

- 前提
  - プログラムのルートディレクトリ: `/home/higo/prod/`
  - ソースファイル: `/home/higo/prod/src/HelloDate.java`
- ワーキングディレクトリ: `/home/higo/prod`
- ローカルリポジトリ: `/home/higo/prod/.git`
- リモートリポジトリ: 通常はGitHubやBitbucket
  - 授業では, GitBucketを使う
- 名前が長いので, 以下のように省略します
  - ワーキングディレクトリ: **WD**
  - ローカルリポジトリ: **LR**
  - リモートリポジトリ: **RR**

# RR → LR+WD

- 皆さんのリモートリポジトリを置く場所は以下のURL
  - <https://dev.ics.es.osaka-u.ac.jp/gitbucket/>
  - ログインしてみてください
  - ログイン後、画面左側に prod というリポジトリが見えるはず
  - prod をクリックしてみてください
- prod をcloneする(手元に持ってくる)
  - git clone `https://dev.ics.es.osaka-u.ac.jp/gitbucket/git/higo/prod.git`

httpではなく, https

皆さんの場合は学籍番号

# WD → LR, LR → RR

- まずはWDの状態を確認することが大切

- git status

- WDの内容をLRに反映

- emacs src/HelloDate.java

- git add ファイル名

- git status

git status したときに、**緑**で表示されたファイルは git commit でLRに反映される、**赤**で表示されたファイルは、LRに反映されない

- git commit -m “その変更を表すメッセージ”

- LRの内容をRRに反映

- git push

- RR をブラウザで確認 (pushした内容が反映されているか)

- <https://dev.ics.es.osaka-u.ac.jp/gitbucket/>

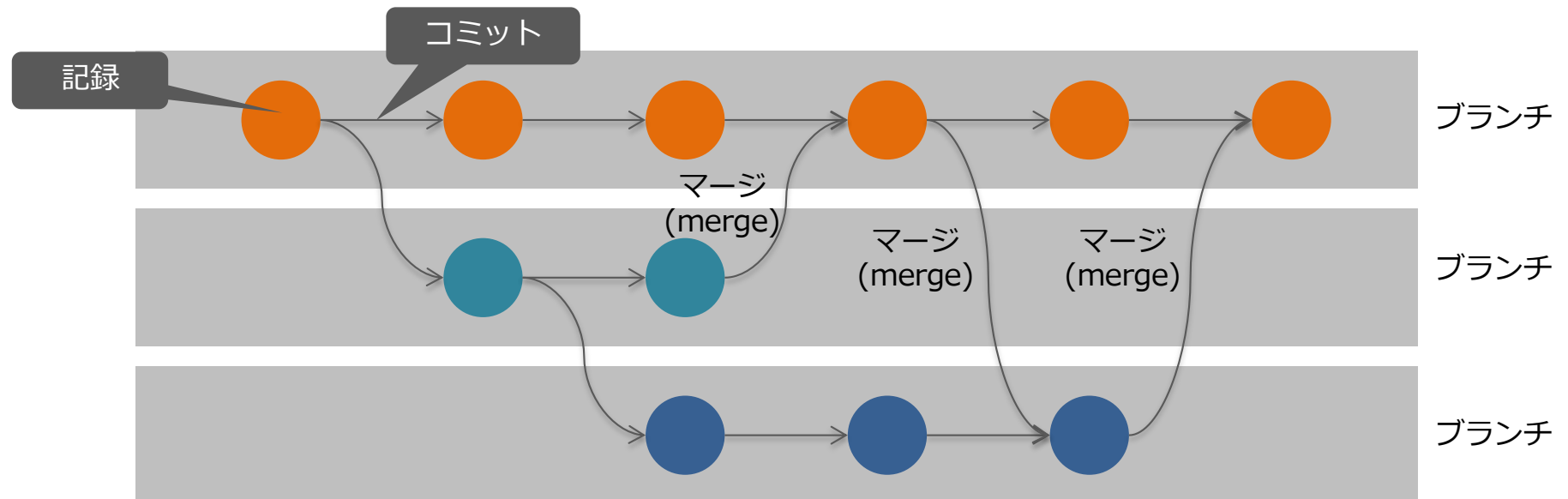


# git commit でエラーが出た人は

- ローカル環境のgitでメールアドレスとアカウントの情報を設定していないことが原因
- 以下のコマンドを実行してください
  - `git config --global user.email "09BXXXXX@localhost"`
  - `git config --global user.name "09BXXXXX"`
  - ただし, 09BXXXXX は自分のアカウント名に読み替えること

# 複数の歴史(ブランチ)

- コミットごとに記録が取られる
- 別のブランチに分岐できる
  - いくつでも分岐できる
  - すきなところから分岐できる
- 分岐したブランチを統合(merge)できる



# ブランチの利用方法: PR駆動型の開発

- PR: Pull Request
  - RR の或るブランチから RR の master に向けた「**自分の変更を取り入れてほしい**」というお願い
  - master: 最初から存在するブランチ
- 各機能の実装は個別ブランチで行う
- masterには、完成した機能のみを追加していく

# ブランチ作ってみましょう！

- まずは現在自分がどのブランチにいるか確認
  - `git branch`
- 新しいブランチ(branch1)を作り, そこに移動
  - `git checkout -b branch1`
  - `git branch`
- ファイルを変更し, add, commit, push する
  - `emacs src/HelloDate.java`
  - `git add src/HelloDate.java`
  - `git commit -m “change in branch1”`
  - `git push`

# もう1つブランチを作きましょう！

- いったん, masterにもどる

- `git checkout master`

これ大切！これがなかったら, branch2はbranch1から枝分かれしてしまう

- 新しいブランチを作り, そこに移動

- `git checkout -b branch2`

- `git branch`

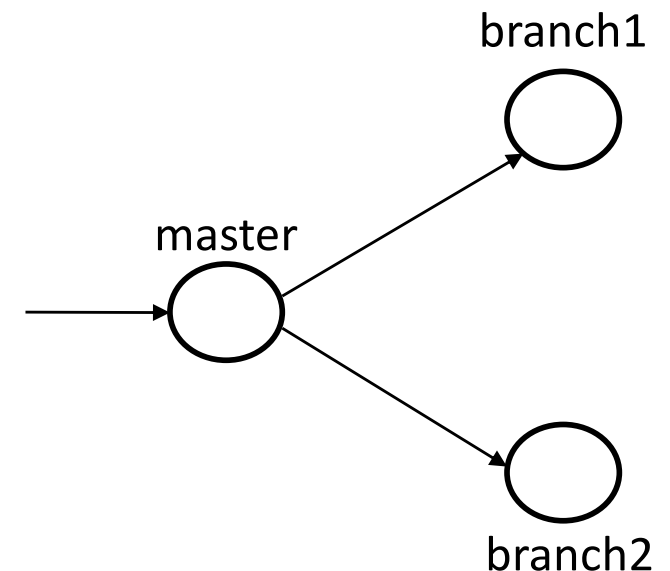
- ファイルを変更し, add, commit, push する

- `emacs src/HelloDate.java`

- `git add src/HelloDate.java`

- `git commit -m “change in branch2”`

- `git push`



# RR の Web で PR を作成し, マージ

- 今回は, 一人でやっているなので, PR の作成とマージはどちらも自分自身で行う
  - 多人数開発の場合は, 通常はPRの作成とマージは異なる開発者が行う
- PR駆動型開発は, PBL(ロボコード)でめっちゃ便利
  - Aさんは機能1を実装しPRを作成, Bさんは機能2を実装しPRを作成, ...
  - CさんはAさんのPRを確認し問題なさそうであればマージ, ...

# Conflict（競合）

- 複数の人が同じ場所（同じファイルの同じ行）を変更してしまった状態
- Conflict が起こった場合，Gitは自動でコードをマージできないので，手動で行う必要あり

# Conflict を(わざと)発生させてみる(1/3)

これ大切！これがなかったら, `conflict2`は  
`conflict1`から枝分かれしてしまう

- 現在どのブランチにいるか確認
  - `git branch`
  - `git checkout master`(masterでない場合)
- 新しいブランチ(`conflict1`)を作り, そこに移動
  - `git checkout -b conflict1`
  - `git branch`
- ファイルを変更し, add, commit, pushする
  - `emacs src/HelloDate.java`
  - `git add src/HelloDate.java`
  - `git commit -m "change in conflict1"`
  - `git push`

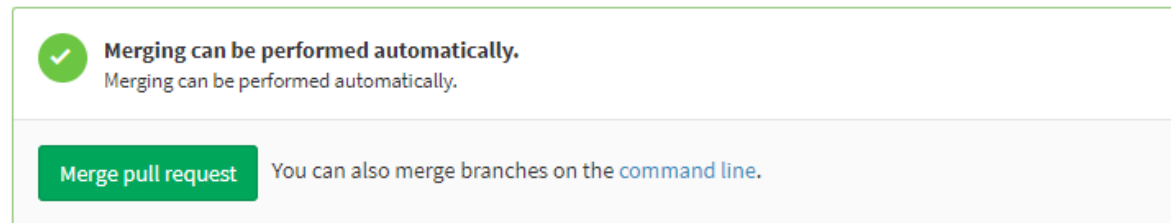
- いったん, masterにもどる
  - `git checkout master`
- 新しいブランチ(`conflict2`)を作り, そこに移動
  - `git checkout -b conflict2`
  - `git branch`
- ファイルを変更し, add, commit, pushする
  - `emacs src/HelloDate.java`
  - `git add src/HelloDate.java`
  - `git commit -m "change in conflict2"`
  - `git push`

同じ行を変更すること！

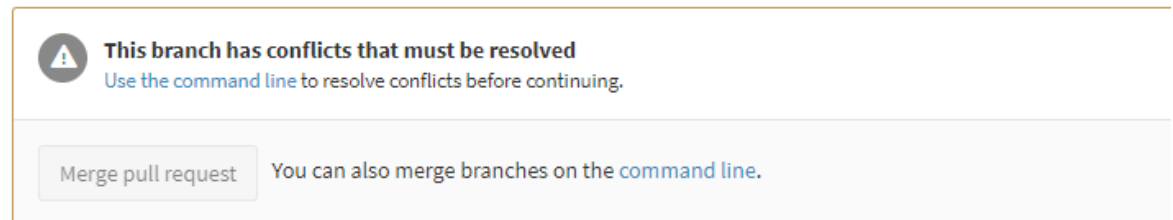


# Conflict を(わざと)発生させてみる(2/3)

- GitBucket のページを開く
- **conflict1**のPull Request を作成して, マージ
  - これはできるはず



- **conflict2**のPull Requestを作成して, マージ
  - これはできないはず
  - **conflict2**で書き換えようとした行は,  
**conflict1**でも書き換えられてしまっている



# Conflict を(わざと)発生させてみる(3/3)

マージできないよ、  
って警告がでてる

The screenshot shows a GitHub pull request for the repository 'higo/prod'. The pull request is titled 'change in conflict2'. A red circle highlights the warning message: 'Can't automatically merge. Don't worry, you can still create the pull request.' Below the warning, there is a 'Write' button and a 'Preview' button. A 'Create pull request' button is at the bottom right. The pull request shows 1 commit, 1 file changed, 0 commit comments, and 1 contributor. The file 'src/HelloDate.java' is shown with a diff. A red circle highlights the conflicting line in the diff: 'System.out.println("conflict2, it's: " + new Date());'. The line is highlighted in green, indicating it is the version from the pull request that is being merged.

base fork: higo/prod base: master head fork: higo/prod compare: conflict2

× Can't automatically merge. Don't worry, you can still create the pull request.

change in conflict2

Write Preview

Leave a comment

Attach images or documents by dragging & dropping, or selecting them.

Create pull request

1 commit 1 file changed 0 commit comments 1 contributor

Commits on 2018-10-12

YoshikiHigo change in conflict2 12d51bd

Showing 1 changed file

Unified Split

src/HelloDate.java

```
1 import java.util.*;
2 public class HelloDate {
3     public static void main(String[] args){
4         System.out.println("branch1");
5         System.out.println("Good morning, it's: " + new Date());
6         System.out.println("conflict2, it's: " + new Date());
7         System.out.println("branch2");
8     }
9 }
```

このプルリクエストで書き換  
えたい行はすでにconflict1で  
書き換えられている

# conflict の解消方法

- マージ先のブランチ(今回の場合はリモートリポジトリのmaster)の変更内容を, 手元(今回の場合はローカルリポジトリのconflict2)に取り込む
  - git branch
  - git pull origin master (**originはリポジトリ, masterはブランチを表す**)
- 手動でコードをマージして, add, commit, push
  - emacs src/HelloDate.java
  - git add src/HelloDate.java
  - git commit (**-mオプションつけない, viが起動したらShift+Zを二回押す**)
  - git push
- GitBucket に移動し, プルリクエストを作成し, マージ

pull したときにconflictが発生している  
ことを教えてくれている

```
$ git pull origin master
Username for 'https://loki.ics.es.osaka-u.ac.jp':
Password for 'https://higo@loki.ics.es.osaka-u.ac.jp':
remote: Counting objects: 1, done
remote: Finding sources: 100% (1/1)
remote: Total 1 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (1/1), done.
From https://loki.ics.es.osaka-u.ac.jp/gitbucket/git/higo/prod
 * branch          master      -> FETCH_HEAD
    665267b..520fd99  master    -> origin/master
Auto-merging src/HelloDate.java
CONFLICT (content): Merge conflict in src/HelloDate.java
Automatic merge failed; fix conflicts and then commit the result.
```

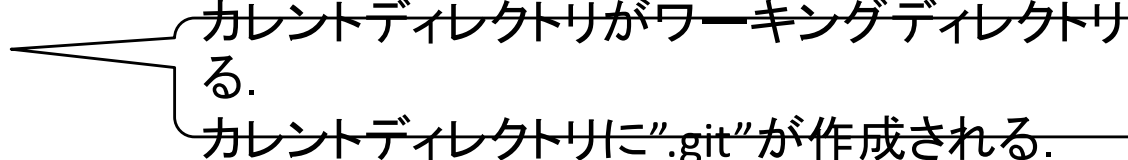
<<<<<<< と >>>>>>> で囲まれた範囲が  
conflictが発生している箇所

```
$ cat src/HelloDate.java
import java.util.*;
public class HelloDate {
    public static void main(String[] args){
        System.out.println("branch1");
<<<<<<< HEAD
        System.out.println("conflict2, it's: " + new Date());
=====
        System.out.println("conflict1, it's: " + new Date());
>>>>>>> 520fd996ba4d77fb3345e8a328e5f3659cd1cdhb
        System.out.println("branch2");
    }
}
```



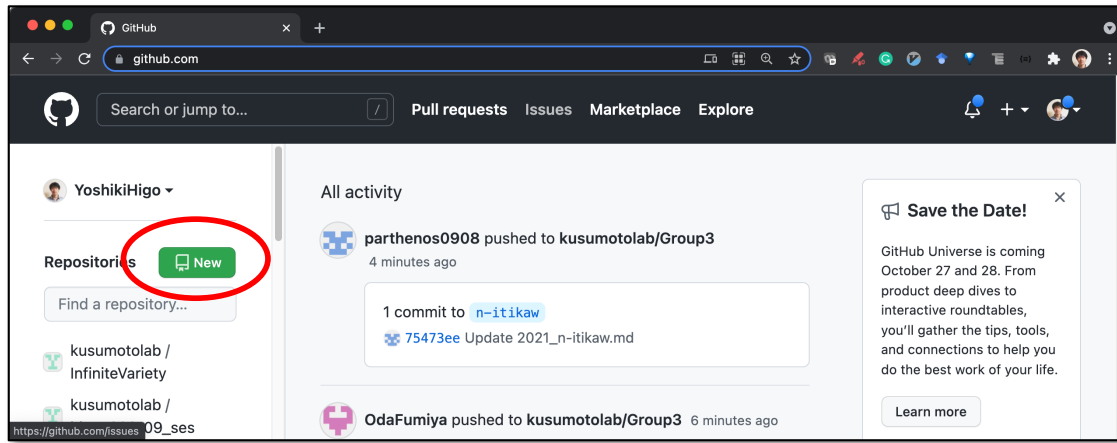
- Gitホスティングサービスの草分け的存在
- publicリポジトリとprivateリポジトリ
  - publicリポジトリ: リポジトリ内のコードを誰でも閲覧可能
  - privateリポジトリ: プロジェクトメンバーのみ閲覧可能

# ローカルリポジトリを作る

- 以下のコマンドを実行し、ディレクトリとファイルを作る
  - `mkdir newprogram`
  - `cd newprogram`
  - `touch new.java`
- リポジトリを作る
  - `git init` 
- 作成したファイルをリポジトリに反映
  - `git add .`
  - `git commit -m "first commit"`

A

# リモートリポジトリをつくる



B

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

**Owner \*** YoshiakiHigo

**Repository name \*** newprogram ✓

Great repository names are short and memorable. Need inspiration? How about [jubilant-happiness?](#)

**Description (optional)**

☐ **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☒ **Private**  
You choose who can see and commit to this repository.

**Initialize this repository with:**  
Skip this step if you're importing an existing repository.

☐ **Add a README file**  
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**  
Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**  
A license tells others what they can and can't do with your code. [Learn more.](#)

**Create repository**

C

**Quick setup — if you've done this kind of thing before**

[Set up in Desktop](#) or [HTTPS](#) [SSH](#) `git@github.com:YoshiakiHigo/newprogram.git`

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

**...or create a new repository on the command line**

```
echo "# newprogram" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:YoshiakiHigo/newprogram.git
git push -u origin main
```

**...or push an existing repository from the command line**

```
git remote add origin git@github.com:YoshiakiHigo/newprogram.git
git branch -M main
git push -u origin main
```

**...or import code from another repository**  
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

# .gitignore

- git で管理したくないファイルがある場合に利用する
- .gitignore も git で管理する

拡張子が.classや.logのファイルを  
無視するための設定

```
# Compiled class file
*.class

# Log file
*.log
```