GitHub & Pull Request

2020年10月30日（金）
システム開発プロジェクト応用第一

東京工業大学
特任助教　内田公太

本講義の概要

- 実際のシステム開発プロジェクトの現場で使われている現代的な開発ツールや手法を学ぶ
  - 正しいツールや手法の選択はソフトウェア開発を効率的に，そして楽しいものにする

到達目標：

- 現代的な開発ツールの基本的な使い方と適する用途が分かる

講義で扱うトピック

- 情報収集
- GDB
- Git
- バグトラッキング
- GitHub & Pull Request
- ユニットテスト
- 継続的インテグレーション
- デプロイと冪等性
- コミュニケーション

自己紹介

- 内田公太
- Twitter @uchan_nos
- 週3日：サイボウズ・ラボ株式会社
  週2日：東工大の特任助教

- osdev-jpコアメンバー
- 『30日でできる! OS自作入門』の校正担当
- 『自作エミュレータで学ぶ
  x86アーキテクチャ』の著者 →

講義の進め方

スタイル：

- 少し講義して演習，の繰り返し

成績評価：

- 現代の開発技術・手法の理解度を評価する
- 各トピックを受講者自身のソフトウェア開発プロジェクトに適用し，レポートおよびリポジトリを提出する
- レポートおよびリポジトリの充実度で成績を決定する

レポート，リポジトリの充実度

- 色々な要素がある
- トピックに対する回答
- ドキュメント
- コミットメッセージ
- プルリクのやり取り
- Etc.

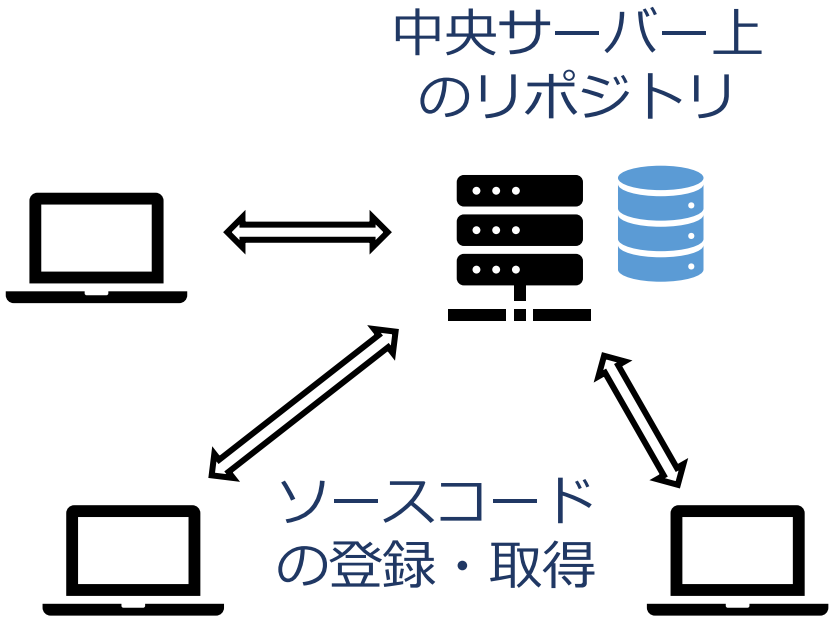
- 総合的に判断して評価します

課題提出の方法

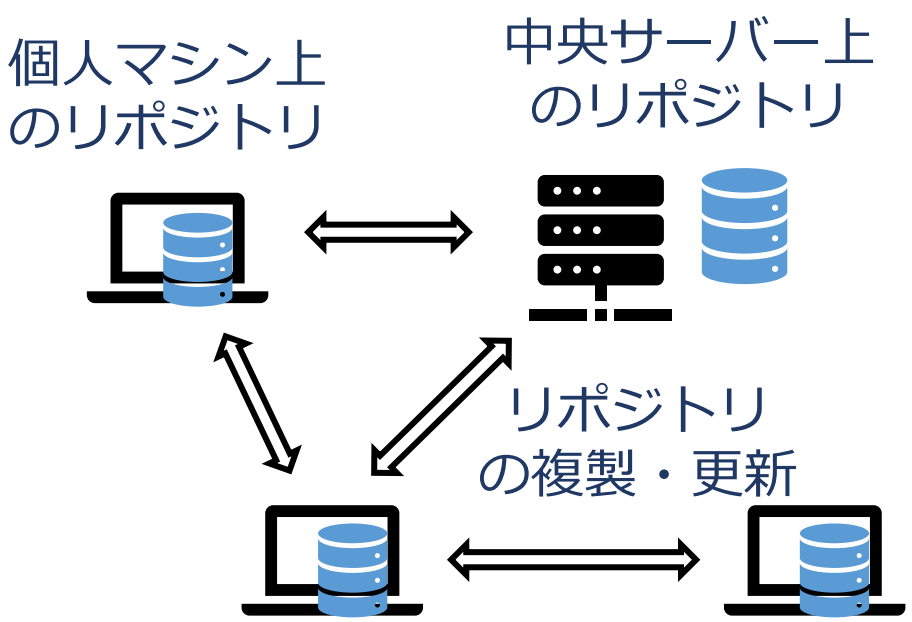
- 課題を含めたリポジトリとレポートを作成し，提出
- 初回（10/2）説明したので詳しい話はしないつもり
  - 改めて聞きたい方がいたらお知らせください

GitHub & Pull Request

Gitは分散型のバージョン管理



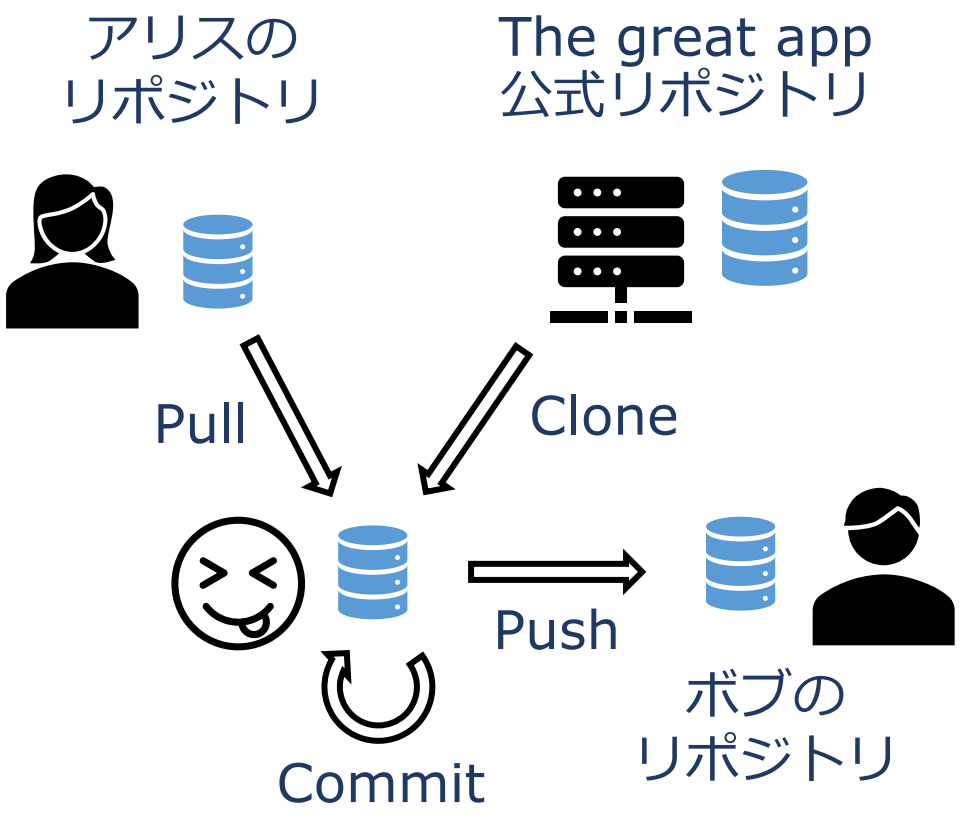
中央集中型
バージョン管理



分散型
バージョン管理

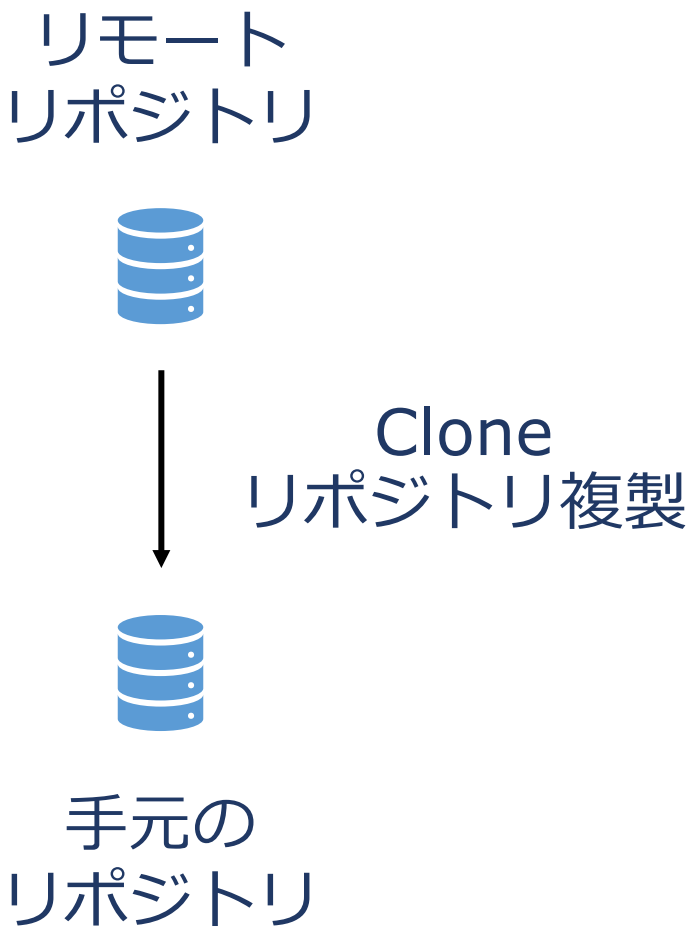
Gitを用いた開発の流れ

- 開発対象となる”The great app”のリポジトリを複製
- 開発し，変更をcommit
- アリスが便利なAPIを追加したらしい！使いたいので取り込もう！
- 素晴らしい機能が完成した！ボブが欲しがっているので送ってあげよう！



Clone

- リポジトリを複製
  - コミットオブジェクト，ブランチ，タグなどすべてが複製される
- リモートリポジトリとして**origin**が設定される
- リモートと言いつつ，ローカルマシン内での複製も可能



git remote

- 追跡されているリモートリポジトリを表示・管理する
- リモートリポジトリ=自分の外のリポジトリ
  - ローカルマシンにある別のリポジトリも含む
- git remote -v
  - 追跡されているリモートリポジトリを一覧する
- git remote add alice john@alice.com:great-app.git
  - alice.com上のgreat-appリポジトリをaliceという名前で追加
  - 使える通信プロトコルはLocal, HTTP, SSH, Git
  - https://git-scm.com/book/en/v2/Git-on-the-Server-The-Protocols

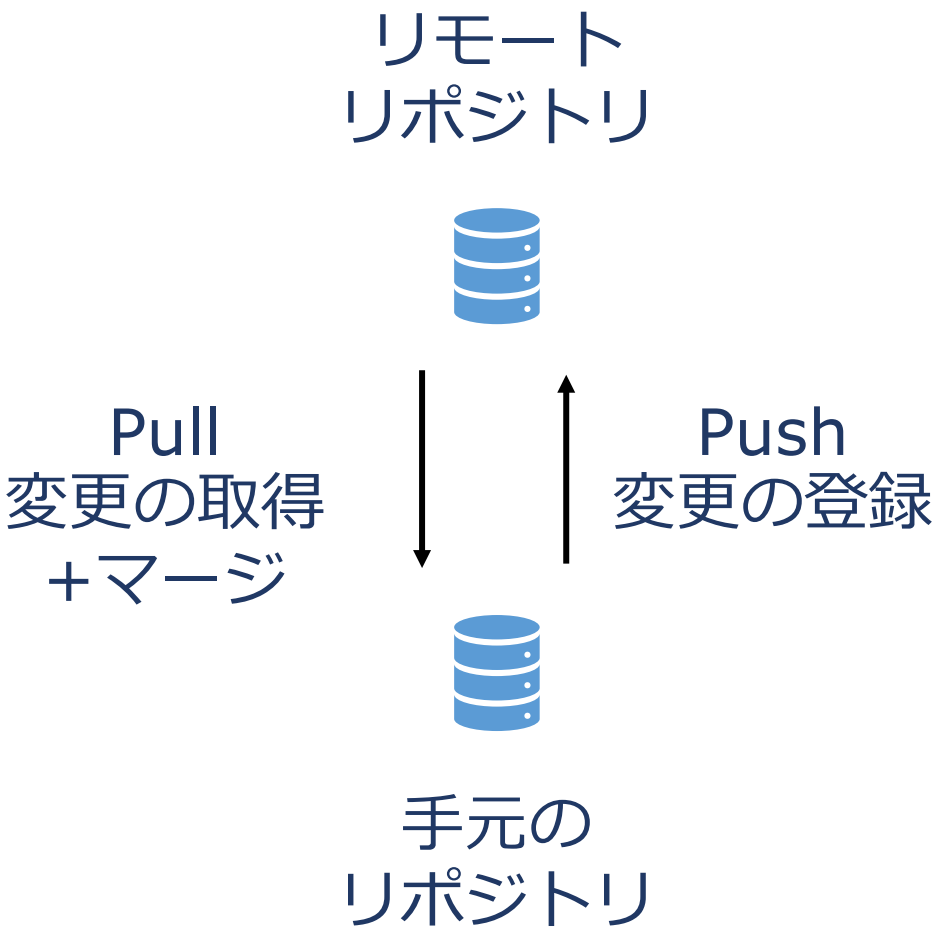
演習1

ローカルマシン内のGitリポジトリを
別のディレクトリに複製し
originを確認せよ

- cd /path/to/somewhere
- git clone /path/to/myproj cloned
- cd cloned
- git remote -v
- 制限時間10分

Push & Pull

- Pushはリモートに変更を送る
  - Pushを禁止しているリポジトリもある
- Fetchはリモートから変更を取ってくる
- Pull=fetch+merge
  - git fetch
  - git merge FETCH_HEAD



References

Tabを2回で補完

複製元
リポジトリ
(myproj)

```
$ git log refs/
refs/heads/br-a      refs/heads/br-b
refs/heads/master    refs/tags/tag-a
```

Clone

複製先
リポジトリ
(cloned)

```
$ git log refs/
refs/heads/master                refs/remotes/origin/br-b
refs/remotes/origin/HEAD         refs/remotes/origin/master
refs/remotes/origin/br-a         refs/tags/tag-a
```

- ブランチやタグの正式名称はrefs/…
- リモートブランチの略記：origin/br-a

Fetch

- git fetch <repo> <refspec>
- refs/remotes/…を更新 + FETCH_HEADを設定
- Pull=fetch+merge
  - git fetch
  - git merge FETCH_HEAD

```
$ cat .git/FETCH_HEAD
2c43fc8c947ab6b660eccdef7e4c44177fcaa48a        branch 'master' of
/home/uchan/workspace/./myproj
1925598567a8adef11b65c04d02249735f9629c3    not-for-merge   branch
'br-a' of /home/uchan/workspace/./myproj
E526eb9858aad877f1131f57659b0fdf11ace71b    not-for-merge   branch
'br-b' of /home/uchan/workspace/./myproj
```

Pullをもっと詳しく

https://git-scm.com/docs/git-pull#_description

Assume the following history exists and the current branch is "`master`":

```
          A---B---C master on origin
         /
    D---E---F---G master
        ^
        origin/master in your repository
```

Pull前はorigin/masterが古い状態になっている

Then "`git pull`" will fetch and replay the changes from the remote `master` branch since it diverged from the local `master` (i.e., `E`) until its current commit (`C`) on top of `master` and record the result in a new commit along with the names of the two parent commits and a log message from the user describing the changes.

Replay the changes

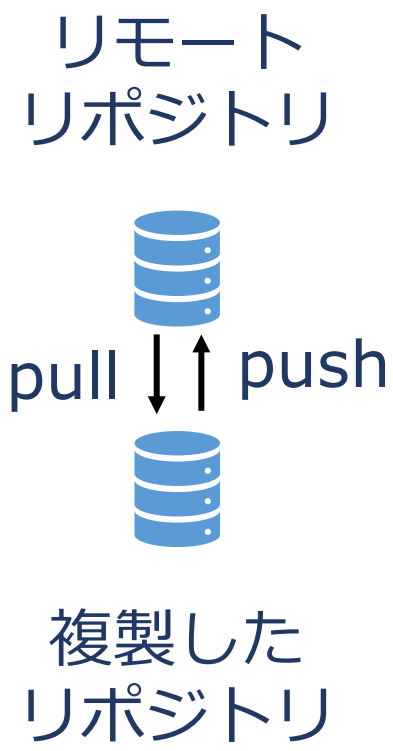
```
          A---B---C origin/master
         /         \
    D---E---F---G---H master
```

Record the result

演習2

ローカルマシン内で以下を実験せよ
リモートリポジトリにcommitしてpull
複製したリポジトリにcommitしてpush

- git pull alice main
  - aliceリポジトリのmainブランチをpull
- git push alice dev
  - devブランチをaliceリポジトリへpush
- 制限時間15分



PullとPushの非対称性

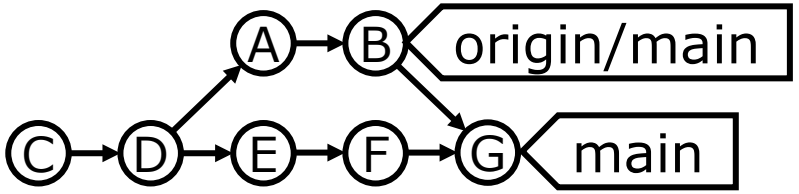
- Pull：リモートブランチを持ってきて**マージする**
  - リモートリポジトリに対するRead権限だけあればよい
  - 手元で変更が加えられていても良い
- Push：リモートへブランチを送信する
  - リモートリポジトリに対するWrite権限が必要
  - リモートで変更が加えられていると失敗する

    “! [rejected]　　dev -> dev (non-fast-forward)”

- ある変更を開発者間で伝達する：Pullが基本
  - Pull Requestは存在するが，Push Requestは無い

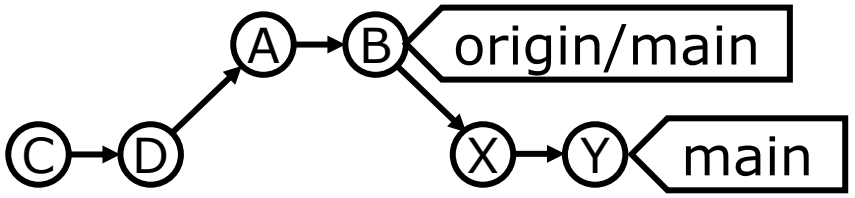
Pullの戦略は2つ

- デフォルトの戦略：マージ
- Fetchにより取得した更新を，現在のブランチにマージ



Gはマージコミット

- 2つ目の戦略：リベース
  - git pull --rebase
- Fetchにより取得した更新を，リベースにより適用
- 歴史が直線的になる



X=E′, Y=F′

演習3

複製したリポジトリ間で
Pullの2つの戦略を実験せよ

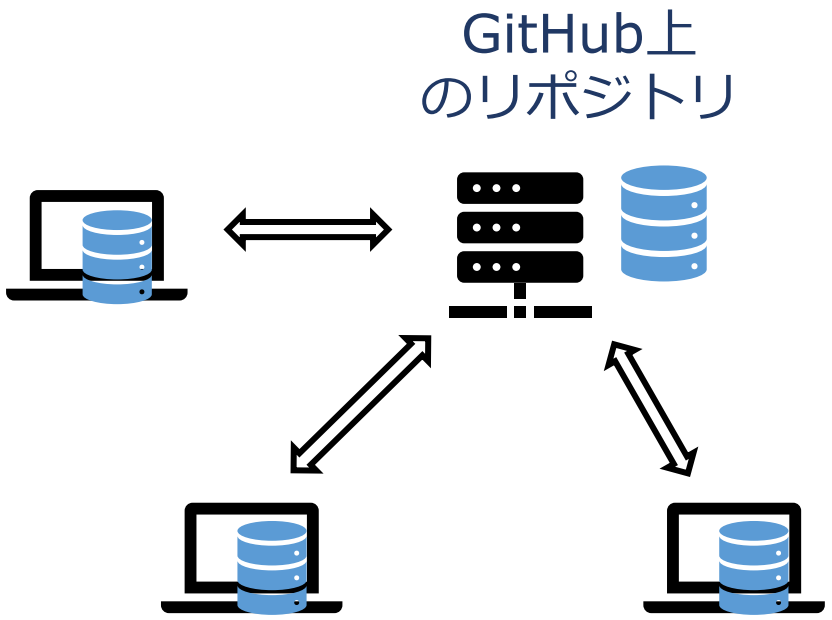
- 片方のリポジトリにだけ変更を加えた場合
- 両方のリポジトリを変更した場合
- 競合が発生しないように実験するのが楽
  - 別ファイルを編集すれば安全
- 制限時間15分

GitHub

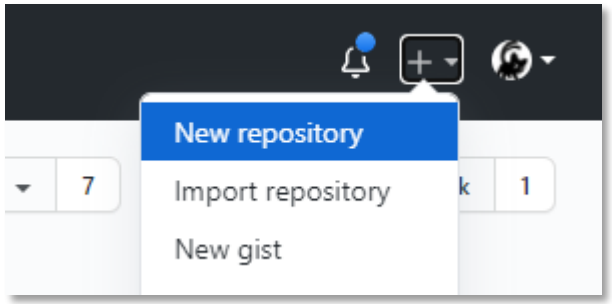
- 今日の主題
- Gitを核とした，開発プロジェクトのためのサイト
  - Gitリポジトリ
  - リポジトリのフォーク（clone+α）
  - Pull Request
  - バグトラッキングシステム（Issues）
  - Wiki
  - ビルド自動化支援
  - プロジェクト管理
- 2018年にMicrosoftがGitHub社を買収

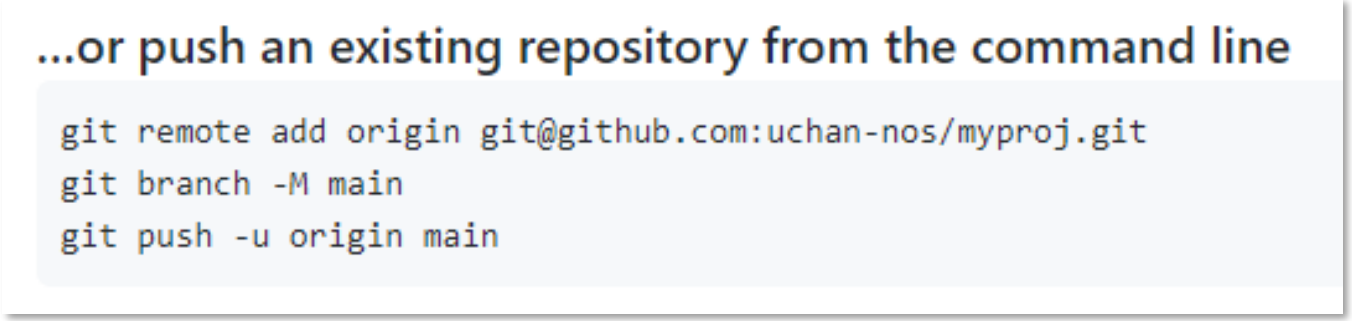
GitHubと中央集中型バージョン管理

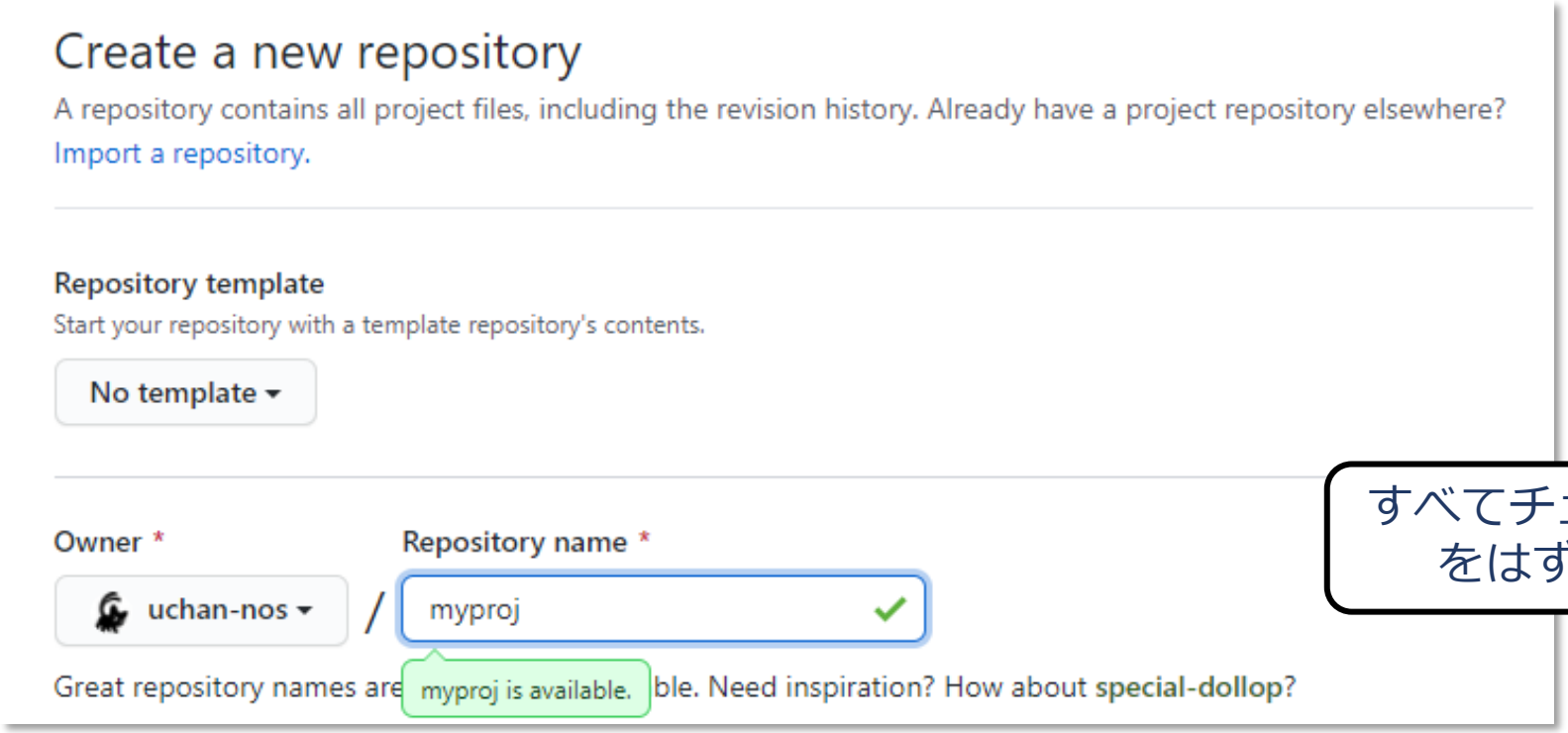
- GitHubにより中央集中型と同様の管理が可能
- 各個人は手元にGitHub上のリポジトリを複製
  - 更新の登録・取得は手元のリポジトリが対象
  - 手元のリポジトリをGitHub上のリポジトリへ反映
- 互いのマシン同士が疎通してなくても大丈夫

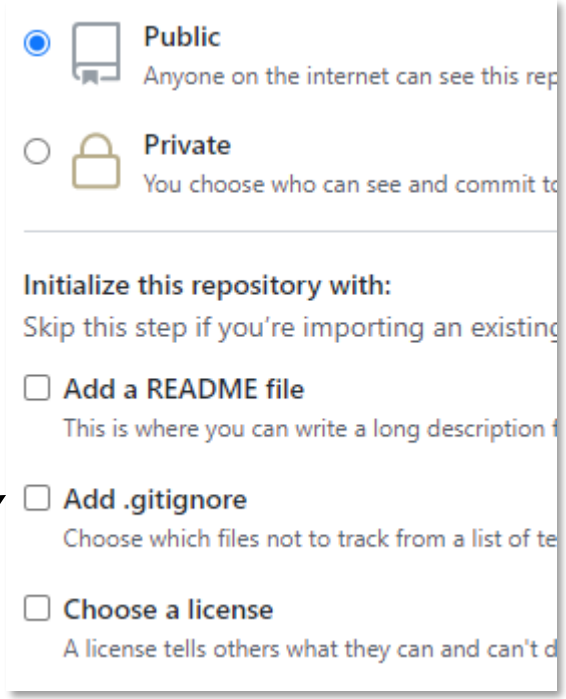


リポジトリをGitHubにアップロード









すべてチェックをはずす

演習4

GitHubで空のリポジトリを作り
手元で構築したリポジトリをpushせよ

- GitHubに表示される3行のコマンドの意味を理解しよう
  - git remote add origin git@...
  - git branch -M main
  - git push -u origin main
- 制限時間10分

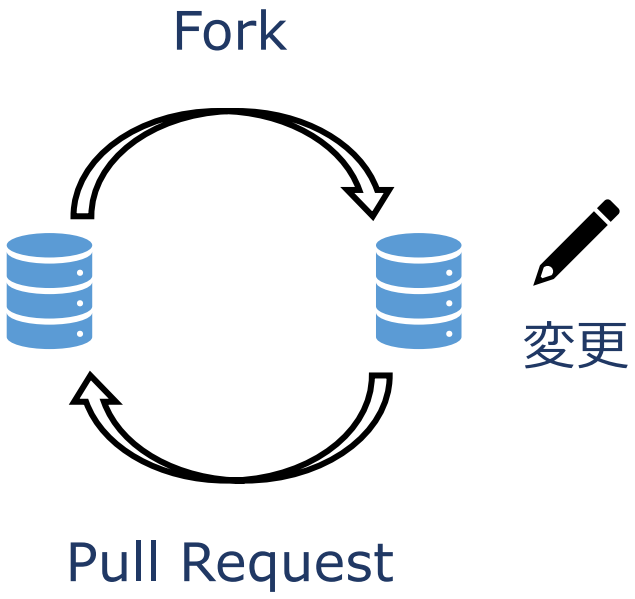
Fork（フォーク）

- 本流（メインストリーム）から派生する機能
- フォークする理由は様々
  - 本流のメンテが止まってしまった
  - 開発方針の相違により分家する
  - 本流をベースに差分開発をしたい
- Gitのcloneに似ている
  - 違い1：フォーク先のページにフォーク元が表示される
  - 違い2：フォーク元からフォーク先のリポジトリに飛べる
  - 違い3：フォーク先からフォーク元へのPRが簡単に作れる

Pull Request（プルリク）

- 変更を取り込んでもらう（pull）
- 要請（request）
- GitHub独自の仕組み

- Gitのブランチとマージの仕組みを利用して実現



GitHub Actions

- ビルド自動化支援の仕組み
- Pushなどのタイミングで，GitHub側で任意のコマンドを実行できる
- ビルドして，成果物をサイトに公開するまでを自動化
→継続的インテグレーション，継続的デリバリー

- 実際のリポジトリで説明
  - WebAssembly on Rust for kintone customize
    https://github.com/uchan-nos/rust-wasm-kintone

レポート提出のトピック名

- 10/30の前半はTOPIC=bts
- 10/30の後半はTOPIC=github

Tokyo Tech

GitHub & PullRequest
後編

11/6（金） 14:20-16:00

ブランチモデル

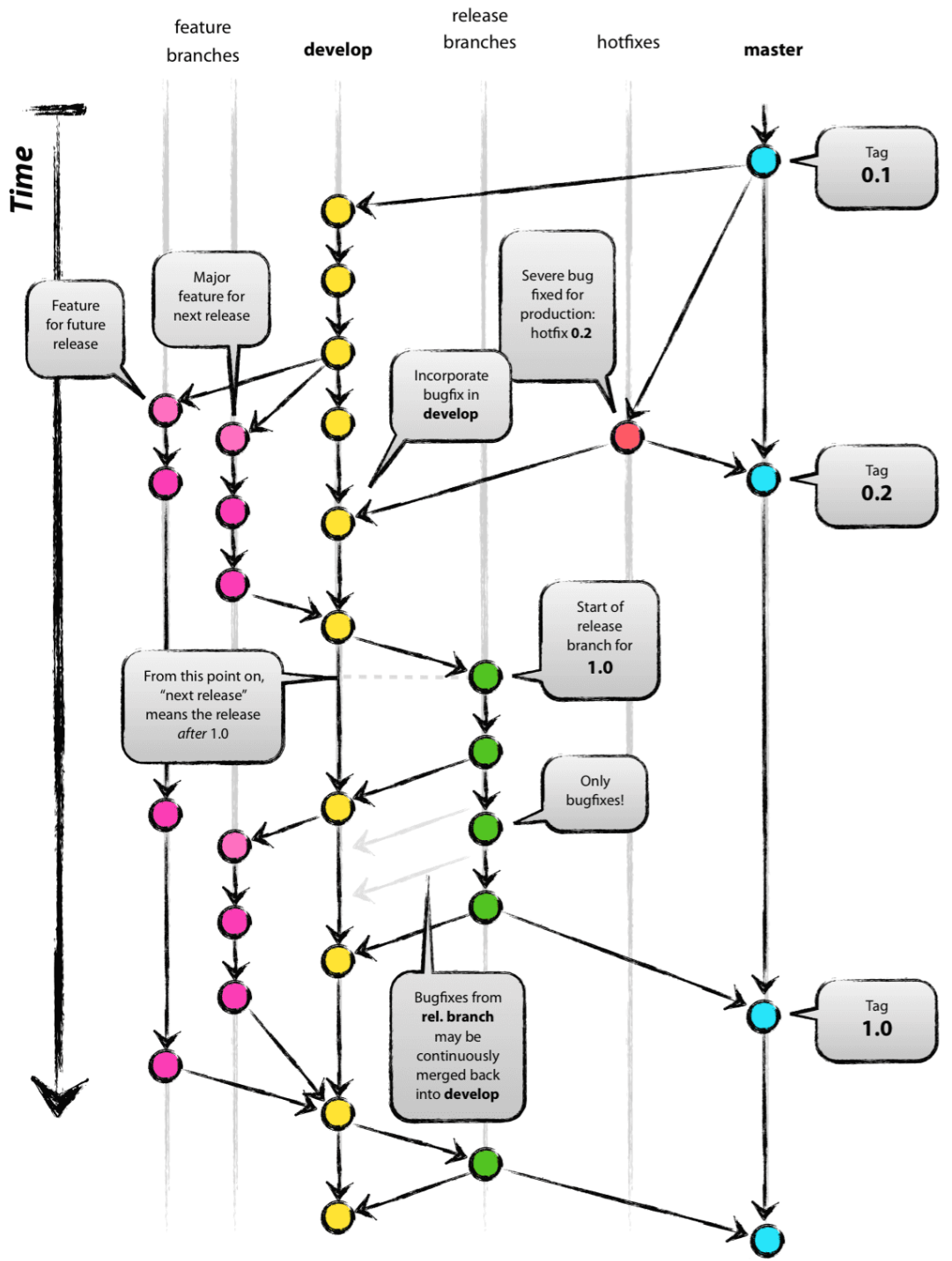
- ブランチの使い方を規定するモデル
- 複数の開発者が無秩序にブランチを作ると混乱する

代表的なモデル2つ

- Git-flow
  - A successful Git branching model
    https://nvie.com/posts/a-successful-git-branching-model/
- GitHub flow
  - Understanding the GitHub flow
    https://guides.github.com/introduction/flow/

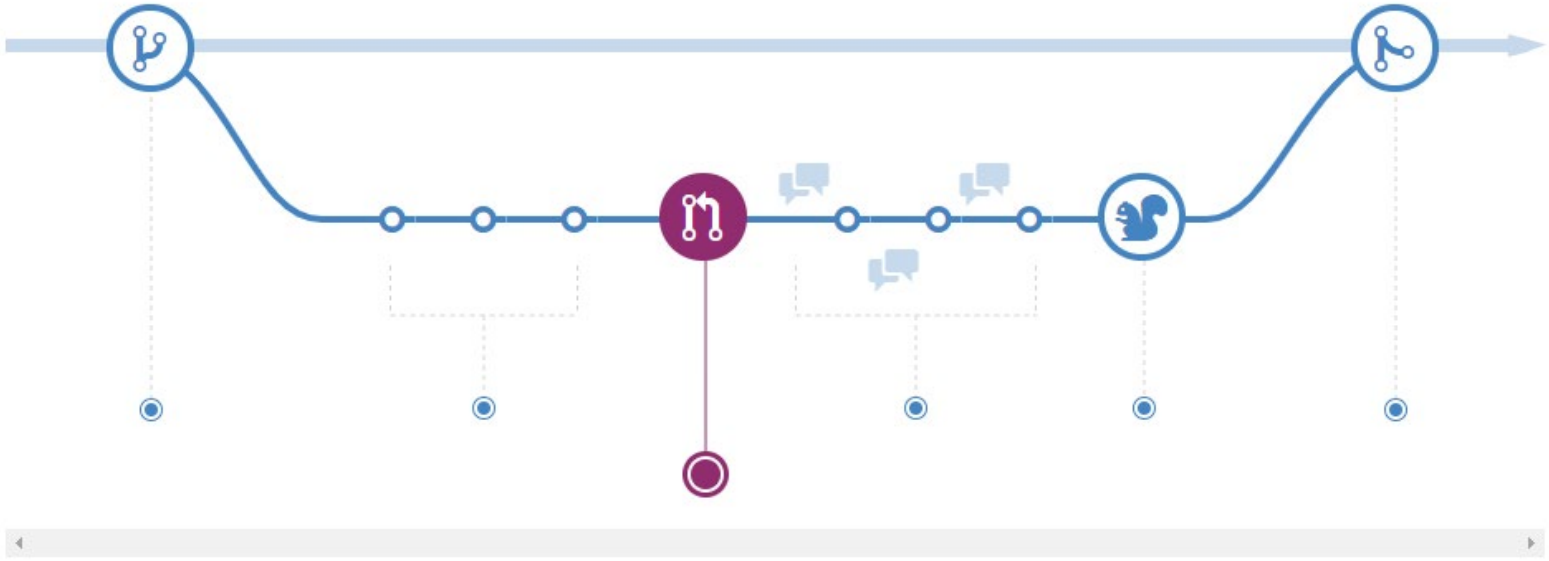
Git-flow

- 2010年に発表された手法
- ブランチモデルの元祖
- 重たいソフトウェア向け
  - 明確にバージョンが付き
  - 長期間のテスト期間があり
  - 複数バージョンを並行して保守するようなソフトウェア
- 日に何度もリリースするような開発には不適



GitHub flow

- Mainブランチ ＋ 機能ブランチ（複数）
  - Mainブランチを常にリリース可能状態に保つ
  - 機能毎に機能ブランチを作成し，そこで開発
- 頻繁にリリースするプロダクトに向く



Open a Pull Request

GitLab flow

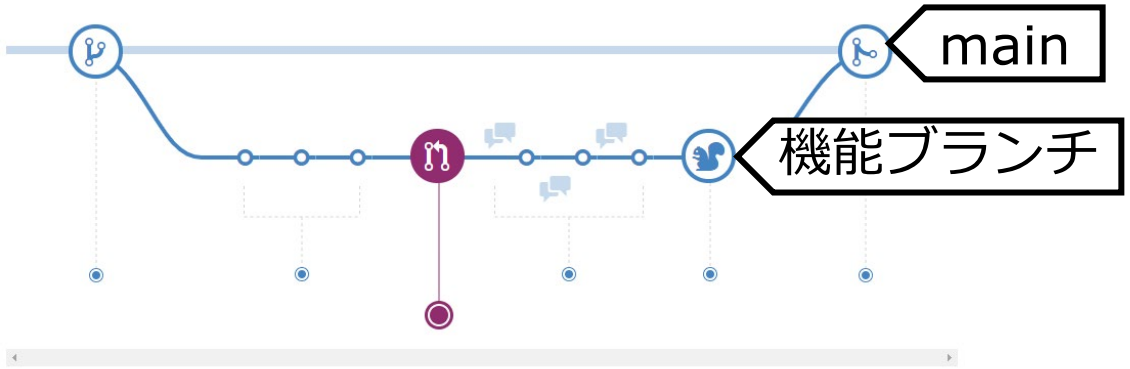
- GitHub flowの仮定：Mainブランチにマージしたものをすぐデプロイできる
- そうではないものも多い
  - iOSアプリは審査通過後にデプロイされる
  - デプロイウィンドウ（10時から16時まで，など）がある
- GitLab flowはGitHub flowに柔軟性を持たせた感じ

適切なブランチモデルの選択

- 複数バージョン vs 単一バージョン
  - SaaSのように1つのバージョンだけメンテすればいい？
- テスト期間
  - リリースのたびに長いテスト期間が必要？
- 開発チームの好み
  - 複雑なモデルできっちり派？シンプルなモデルが好き？
- などなど…
- 万能薬はないので柔軟にモデルを選ぶ/作る

リポジトリ内Pull Request

- リポジトリ内のブランチ間でのPull Request
- チームによる開発で良く使う
  - 開発チームは通常，全員がリポジトリへのアクセス権を持つ
- GitHub flowでは「機能ブランチ→Mainブランチ」のPull Requestを作成する

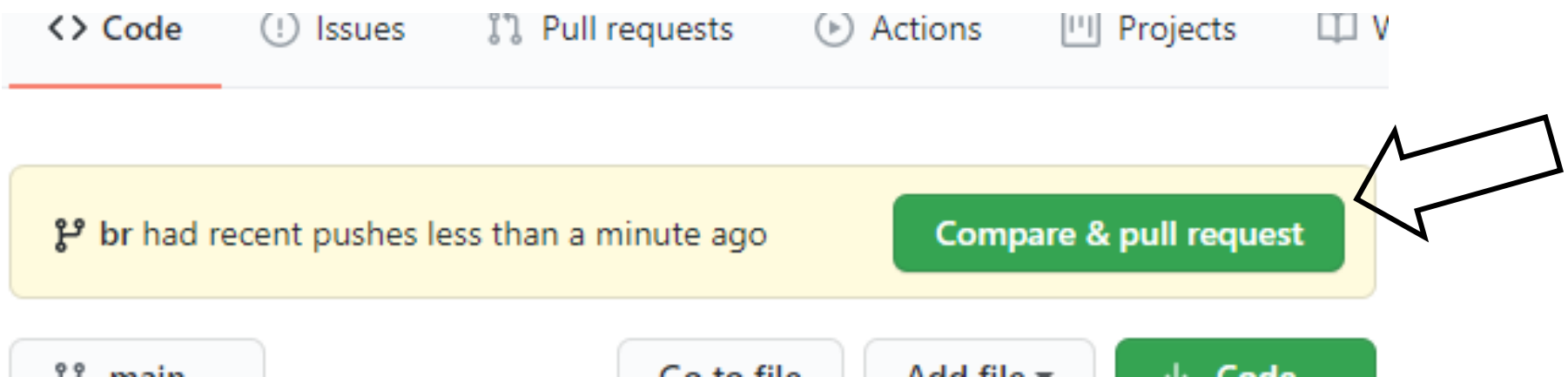


Open a Pull Request

リポジトリ内Pull Requestの出し方

- ローカルでブランチを切る
  - git checkout -b br
- そのブランチにコミットする
- ブランチをGitHubにPushする
  - git push -u origin br
- リポジトリページで Compare & pull request

-uはupstreamを設定するオプション。次回以降，引数無しでgit push可能になる



演習5

ブランチbr-devを切り，
いくつかコミットしPull Requestを出せ

- br-dev→mainのリポジトリ内Pull Request
- 制限時間10分

チーム開発とPull Request

- チーム開発では他人の力を借りられる
- レビューによりコード品質を高める

- Pull Request内でコードにコメントを付けられる
  - レビュー行為がしやすい
- LGTM : Looks Good To Me
  - コードに対して「私はいいと思います」という意思表明
- 一通り問題が解決したらPull Request上でマージ操作

レビュー

- ピアレビュー（Peer Review）
  - 同じ立場の同僚（peer）が行うレビュー
  - 論文なら査読
- 話題は多岐にわたる
  - コードスタイル（インデント），誤字脱字
  - 関数やモジュールへの分割
  - プログラムの論理構造，全体設計
- コードスタイルや誤字脱字をなるべく少なくしよう
  - それらが残っていると枝葉末節の議論をしがち
  - 綺麗なコードなら大きなところに目がいく

演習6

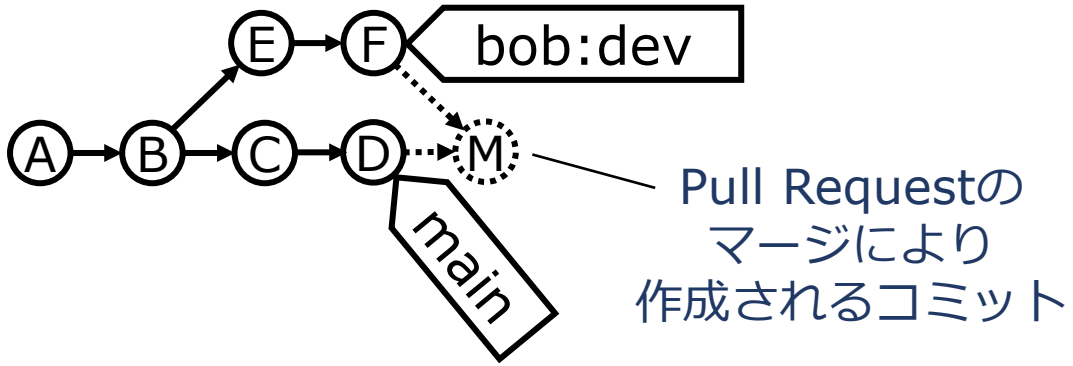
ブランチbr-devのPull Request上で
コードにコメントし，コミットを
いくつか追加し，マージせよ

- チーム開発なら他のメンバーにやってもらう行為
- 今回は自分で流れを体験しよう

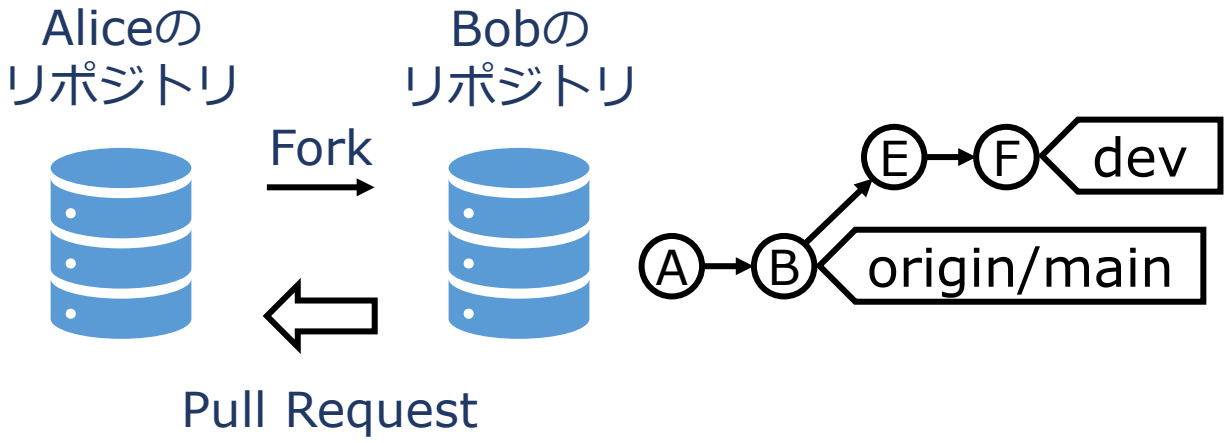
- 制限時間10分

リポジトリ間Pull Requst

- 異なるリポジトリ間のPull Request
- OSSの開発でよく使う
  - 一般の開発者は公式リポジトリへの書き込み権限を持たない

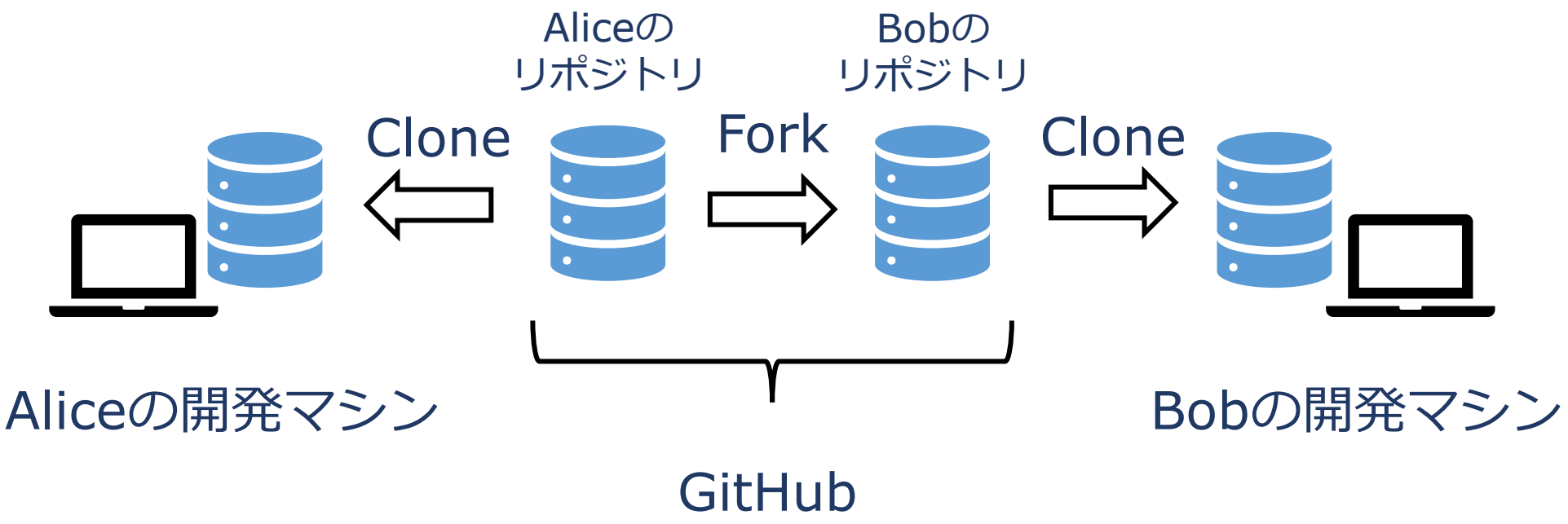


BobがE，Fをコミットする間に
AliceはC，Dをコミット



BobはE，Fを開発し，
Aliceに対しPullを要請

ForkとClone



- ForkはGitHubの独自機能
- CloneはGitの機能

演習7

受講生誰かのリポジトリをForkし,
手元にCloneしてremoteを確認せよ

- git clone …
- git remote -v

- 制限時間10分

OSS開発とリポジトリ間Pull Request

- OSS開発では不特定多数が開発に参加
- リポジトリのPush権限を信頼できる人に限定したい
- でも開発差分を受け取りたい

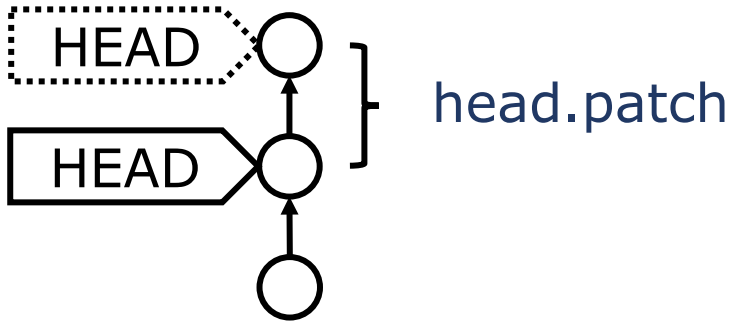
- →リポジトリ間Pull Request
  - ForkはPush権限が不要
  - Pull RequestもPush権限が不要

patchコマンド

- Diffファイルを受け取り，ワークツリーに適用
- 昔はメール等でdiffファイルを交換し，patchで適用していた
  - Apache: 有名なHTTPサーバー
  - 広く受け入れられている洒落
    A patchy server: パッチが多い（=バグが多い）サーバー

```
$ git diff HEAD^ > head.patch
$ git reset --hard HEAD^
$ patch < head.patch
```

Patchコマンドを使って
元の状態を復元するテスト



ML（メーリングリスト）

- GitHub等が登場する前はどうしてた？
  - 開発に関する議論や質問
  - バグ報告
  - Diffのやりとり
- →メールでやる

- ML：メールを複数人に配送する仕組み
- 歴史の長いソフトウェアは今もMLを使っている
  - LKML: Linux kernel mailing list
  - QEMU developers mailing list

演習8

Forkしたリポジトリにコミットし，
リポジトリ間Pull Requestを作成せよ。
受信したPull Request上で追加コミット
を促し，コミットされたらマージせよ

- 2人の共同作業
- 人気があるリポジトリの作者は忙しい
- 制限時間20分

演習9

GitやGitHubの操作を練習せよ

- 何を題材に練習してもよい
- 自分のOSSに関する開発作業をしても良い

- 制限時間25分

レポート提出のトピック名

- 11/6の前半はTOPIC=github
- 11/6の後半はTOPIC=ut

課題提出

- 課題を含めたご自身のリポジトリとレポートを提出
- 提出先は内田のGitHubリポジトリ
  - https://github.com/uchan-nos/titech-sysdev-2020
  - プライベートリポジトリのためアカウント登録必須
    皆さんのGitHubアカウントを教えてください
- このリポジトリに対し，レポートを送る
  - レポートには，トピックに対する回答を含める
- 提出期限は講義の1週間後の10:00（JST）

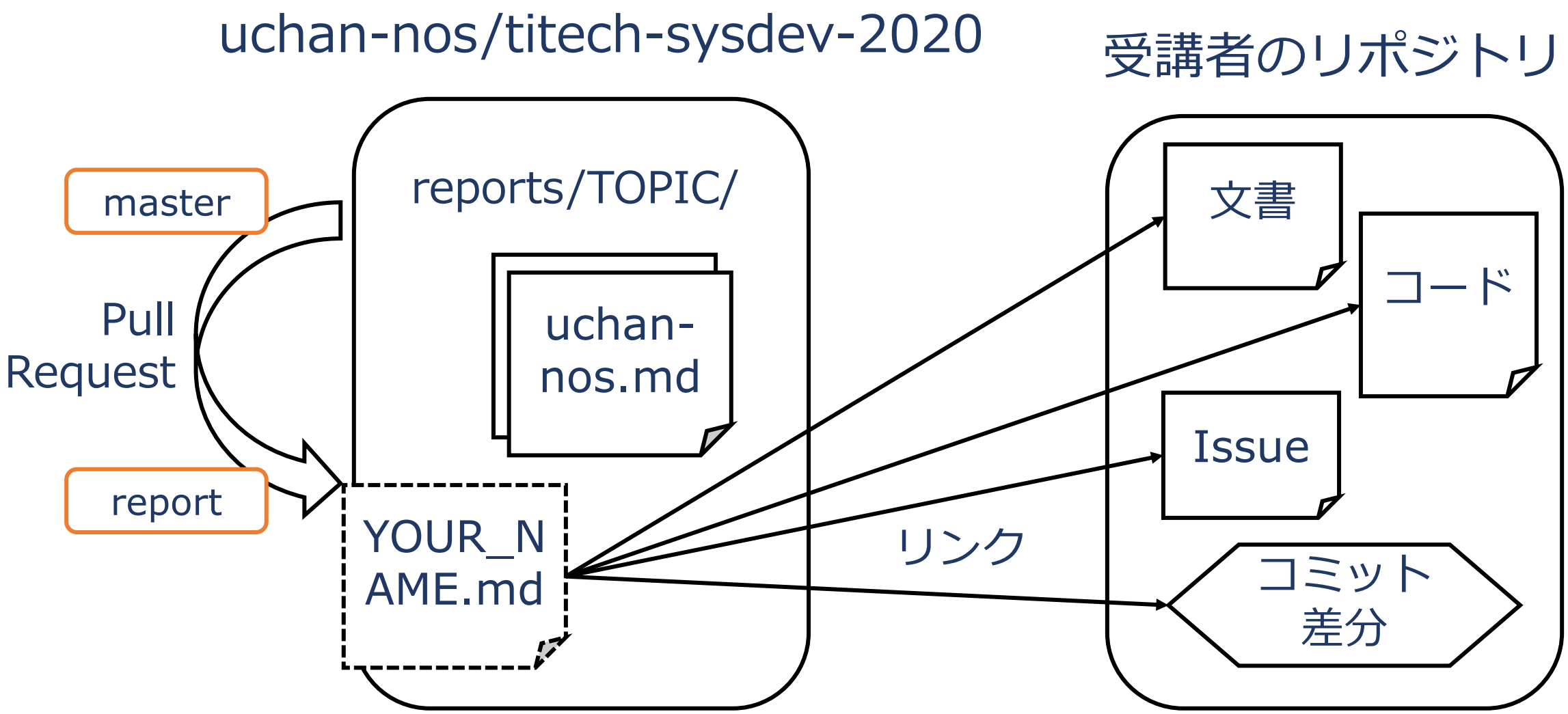
レポートの送り方

1. 独自のブランチを作る
   1. titech-sysdev-2020:master
      ↓branch
      titech-sysdev-2020:report-YOUR_NAME
2. 回答の概要をまとめたファイルを加える
   1. titech-sysdev-2020/reports/TOPIC/YOUR_NAME.md
   2. Commit & Push
3. プルリクを送る（リポジトリ内プルリク）
   1. titech-sysdev-2020:report-YOUR_NAME
      ↓pull request
      titech-sysdev-2020:master

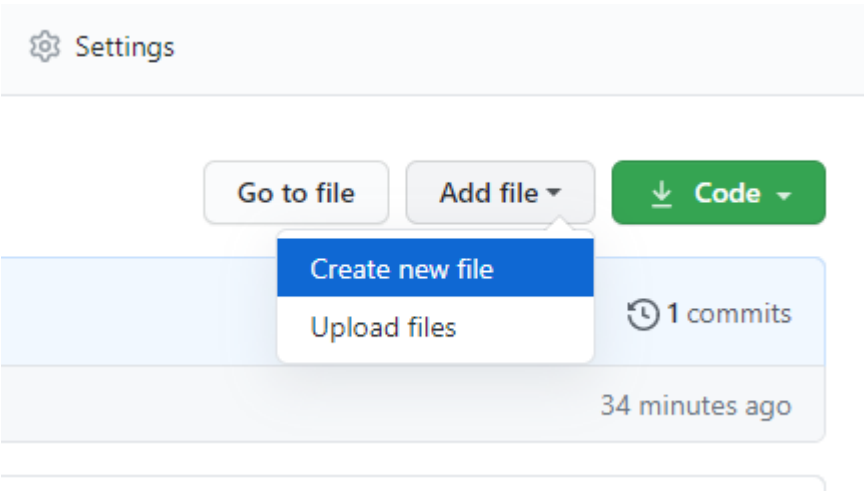
レポート

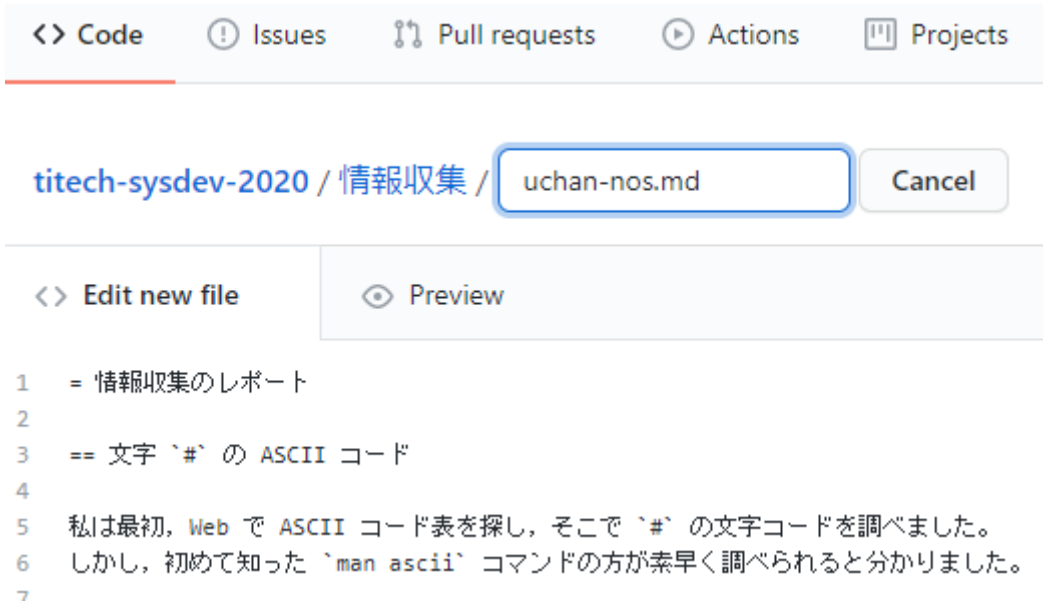
- reports/TOPIC/YOUR_NAME.md
- このファイルに課題への回答を記載する
- 必要なら以下のものを含める
  - Issueへのリンク
  - コミット差分へのリンク
    https://github.com/HOGE/REPO/compare/COMMIT1...COMMIT2
  - その他
- 要するに，成績評価に必要な情報をYOUR_NAME.md自体に記載するか，そこから辿れるようにする

レポートの構成


uchan-nos/titech-sysdev-2020
受講者のリポジトリ
master
Pull Request
report
reports/TOPIC/
uchan-nos.md
YOUR_NAME.md
リンク
文書
コード
Issue
コミット差分

レポートの送り方 1/2

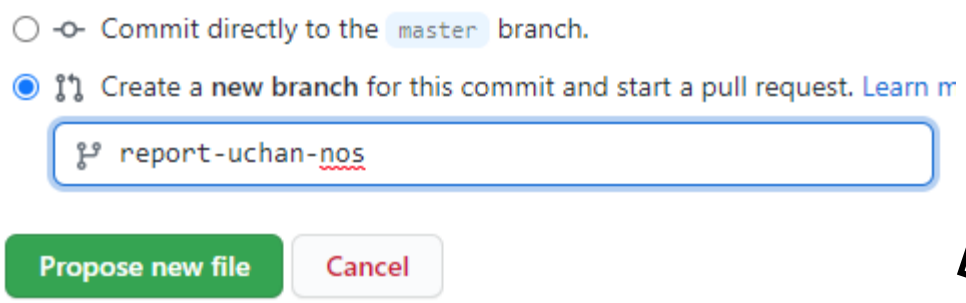




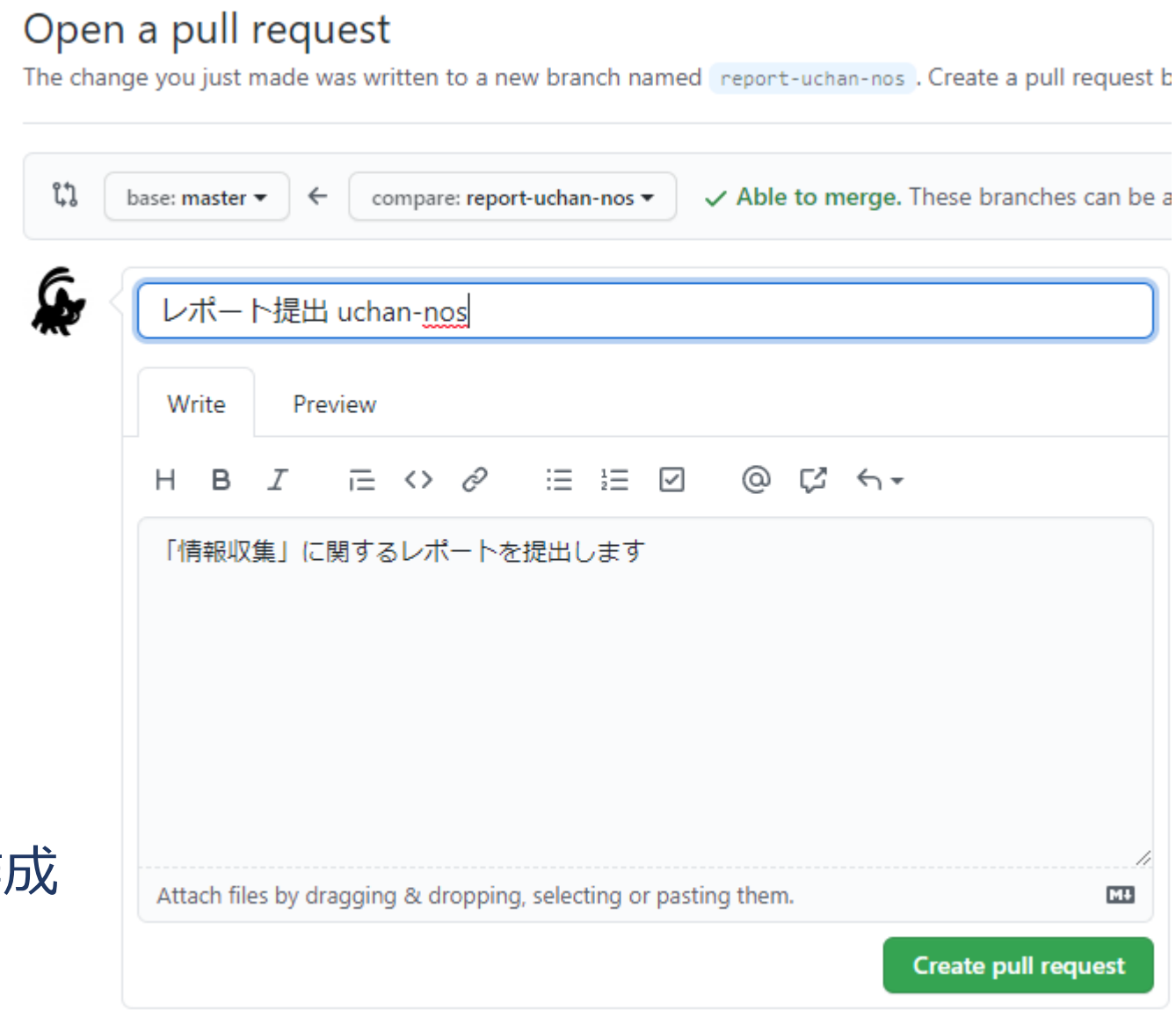
ファイルを新規作成

YOUR_NAME.mdの内容を記述

レポートの送り方 2/2

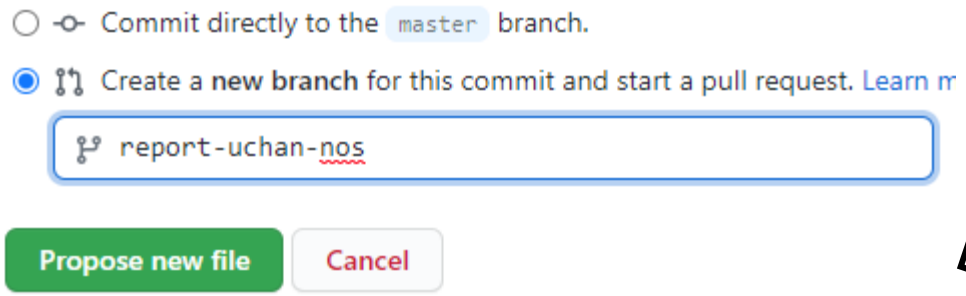


新規ブランチにコミット

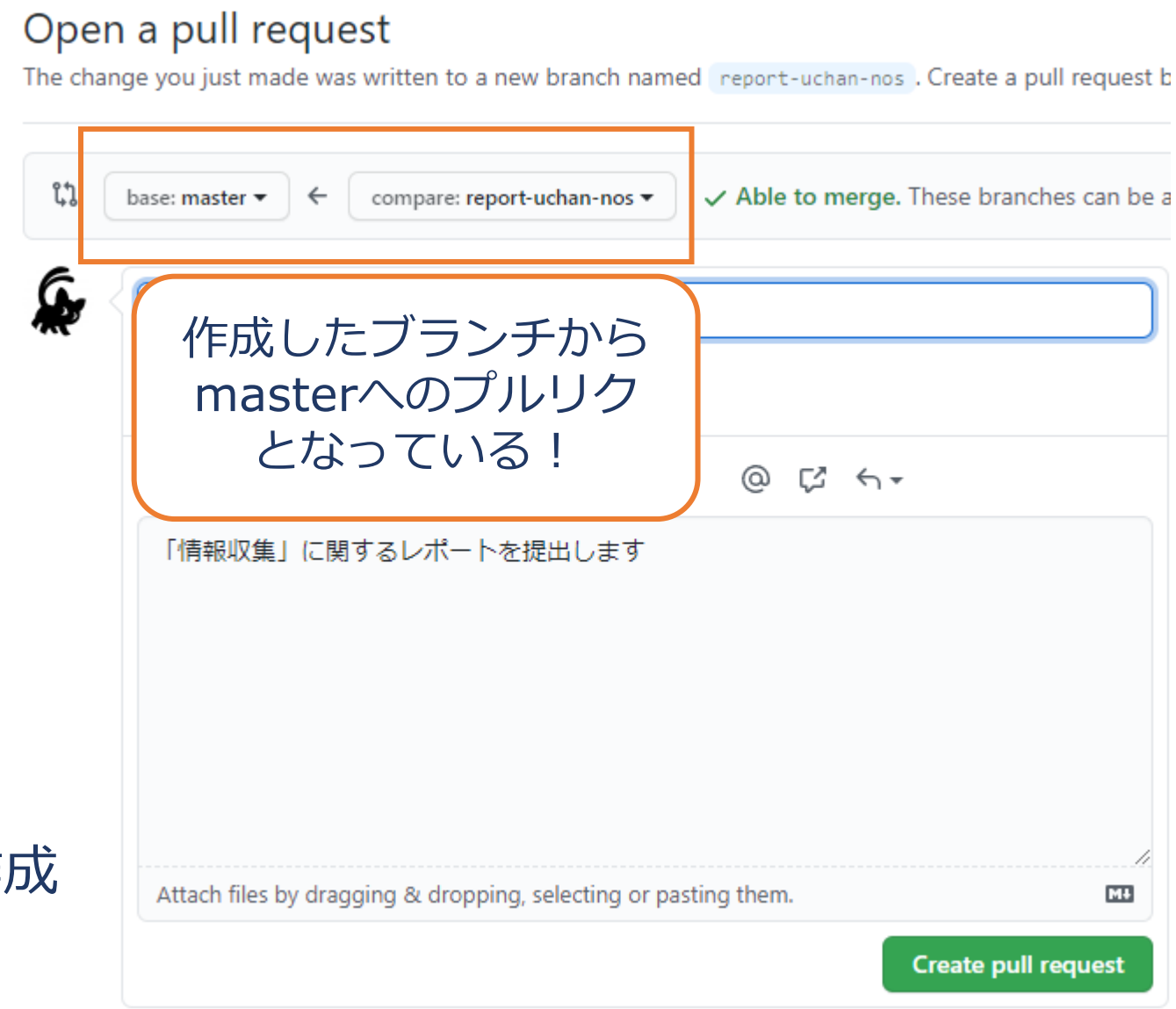


プルリクを作成

レポートの送り方 2/2



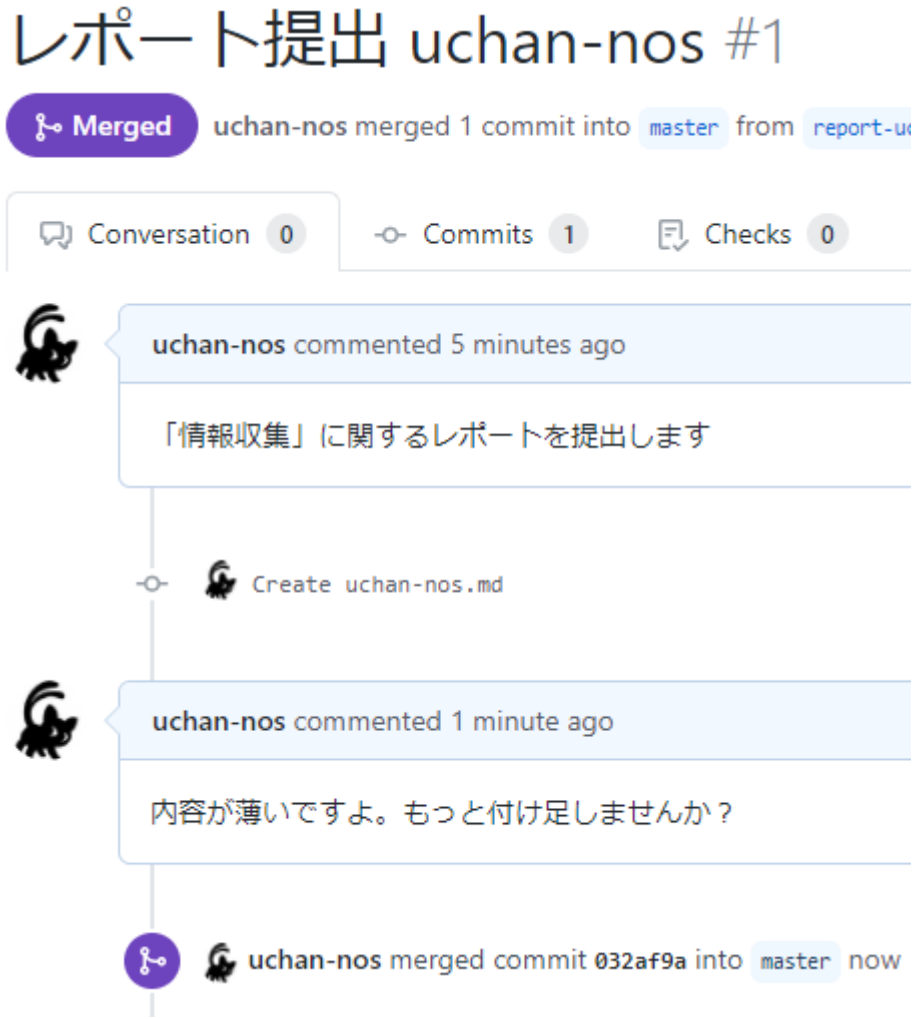
新規ブランチにコミット



作成したブランチから
masterへのプルリク
となっている！

プルリクを作成

レポートの受理



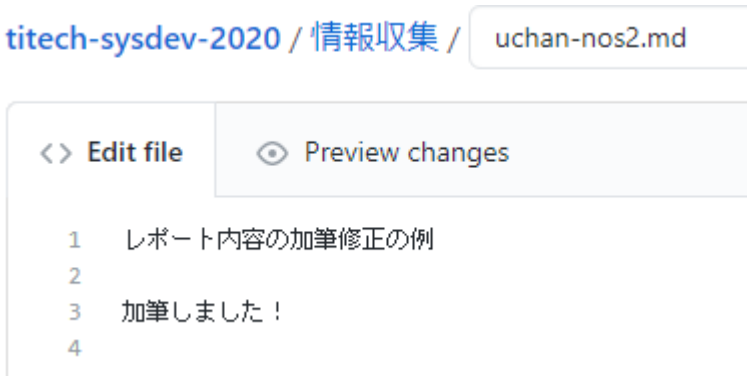
プルリクにコメントが付くことも

Merged：レポート受理済み

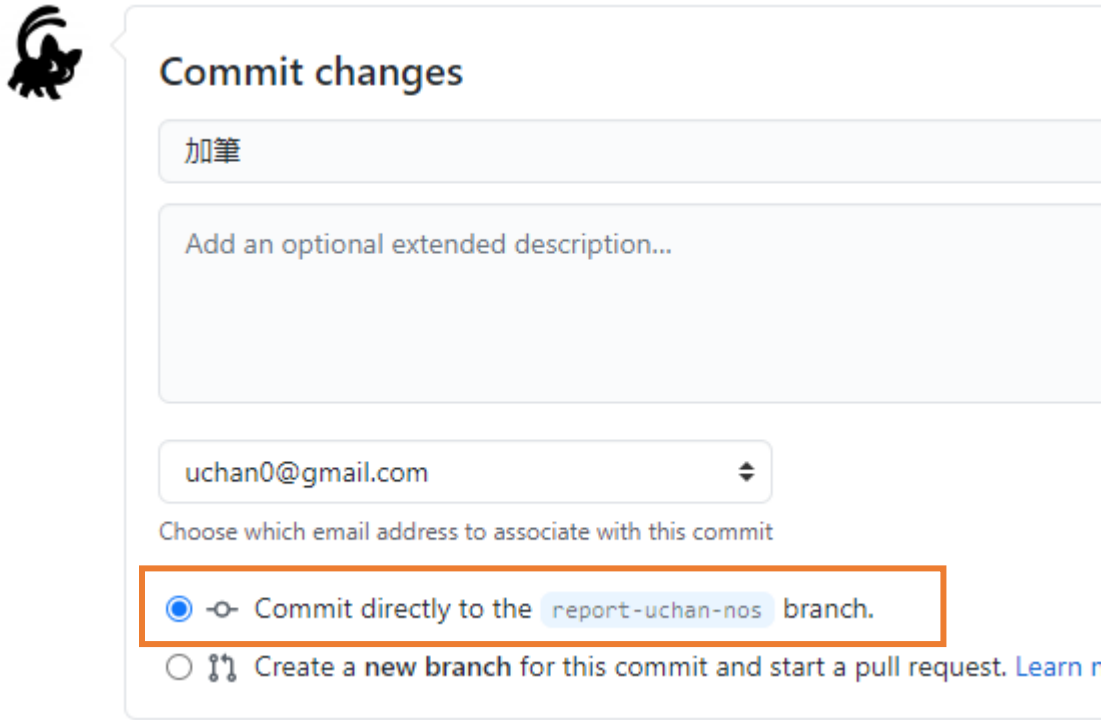
レポートの更新 1/2

レポートに不十分な個所があった！

まだマージされてないときの
更新方法を紹介



納得いくまで加筆修正



report-Xブランチにコミット

レポートの更新 2/2

一発目のコミット

Pull Requestに
コミットが追加されていく

レポートの送り方：その他の方法

- GitHubのWebインターフェースを使う必然性はない
- コマンドラインで作業してもよい
  - 具体的なコマンドラインは示しません
  - この講義は「情報収集」でしたね？
  - コマンドラインについて情報収集すれば，レポートをさらに充実させるネタになりますよ

次回予告

- 次回は11/06（金）14:20から
- 「GitHub & Pull Request」の続きと「ユニットテスト」をやります