デプロイと冪等性
コミュニケーション

2020年11月20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

デプロイと冪等性

デプロイ

- 部隊・兵力などを配置する，展開する

- ソフトウェアのデプロイ：ソフトウェアを配布する
  - セットアップを含むことも
- ソフトウェアのデプロイの例
  - スマホアプリを作り，ストアにデプロイ
  - Webアプリを改造し，本番サーバーにデプロイ
  - MSがWindowsを改造し，みんなのPCにデプロイ

デプロイの主要な段階1：ファイルの配布

- ソフトウェアを構成するファイルを必要な場所に配布
  - 実行ファイル
  - 設定ファイル
  - データファイル
- 配布のやり方は様々
  - SSHで配る
  - メールの添付ファイルで配る
  - CD-ROMやUSBメモリで配る
  - ウェブページに載せ，ダウンロードしてもらう
  - 構成管理ツールで配る

デプロイの主要な段階2：セットアップ

- ファイルの必要な場所への配置
- データベースへの必要なデータの書き込み
- 設定の変更
  - 設定ファイルの書き換え
  - DBへの設定値の書き込み
- 変更の適用
  - サーバープロセスの再起動
  - ネットワークインターフェースの再起動
  - OSの再起動

デプロイを一言で

ソフトウェアを環境に導入し，
使用可能な状態にすること

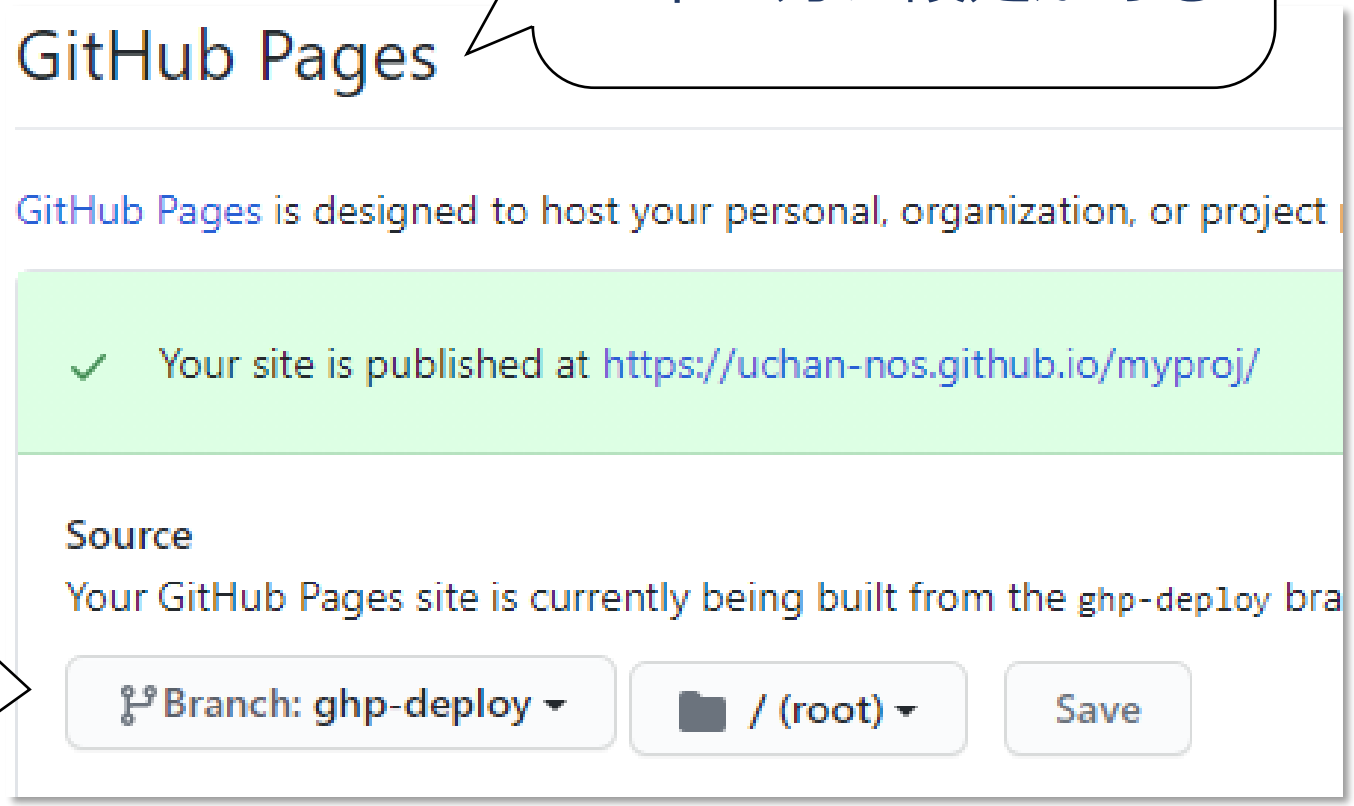
GitHub Pages

- リポジトリからHTMLやCSSを取得してウェブサイトとして公開する静的ホスティングサービス
- GitHubリポジトリに変更をPushするだけでウェブサイトを更新可能

- GitHub Pagesの例
  - Webサイト https://osdev-jp.github.io/
  - リポジトリ https://github.com/osdev-jp/osdev-jp.github.io

GitHub Pagesの設定例

リポジトリの
Settings > Options
の下の方に設定がある



ghp-deploy
ブランチの内容が
GitHub Pages
として公開される

演習1

授業で作ったリポジトリの中に
適当なブランチを作り,
GitHub Pagesとして公開してみよ

- 作成したブランチに foo.txt を追加してみると…
- https://uchan-nos.github.io/myproj/foo.txt として見えるようになるはず

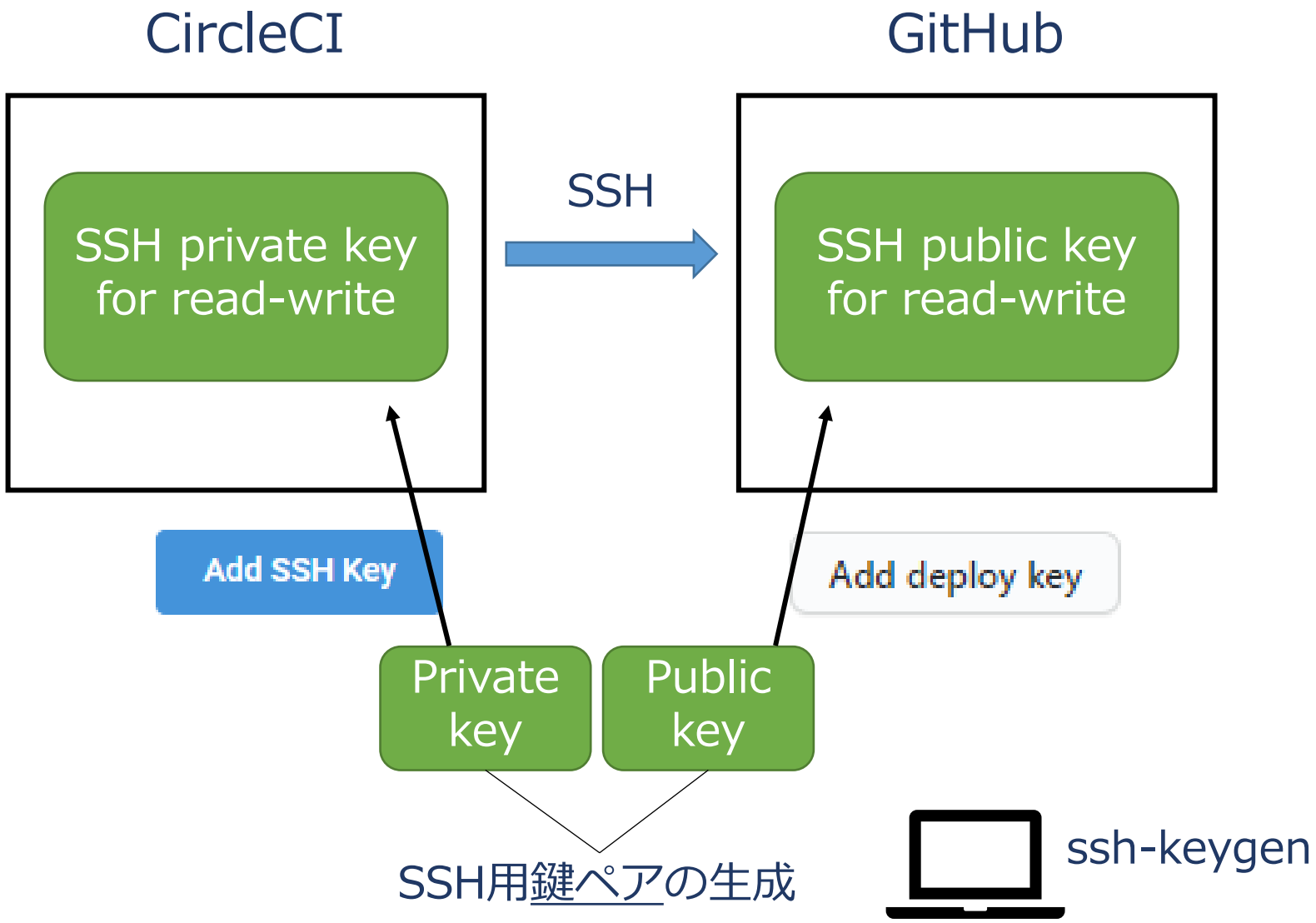
- 制限時間15分

CircleCIからGitHubへのデプロイ

- CircleCIからGitHubへ変更をデプロイしたい
  - CircleCIからGitHubへ，コミットをpushできるようにする必要がある
- キモはCircleCI→GitHubのSSH用鍵の設定

1. 鍵ペアを生成する
   - ssh-keygen
2. 鍵ペアを登録する
   - CircleCI側にPrivate keyを，GitHubにPublic keyを設定
   - GitHub側ではread-write可能な鍵として設定

CircleCIとGitHubへのデプロイ鍵の設定



ヒント

公開鍵暗号を使った
SSHの接続方向は
Private→Public

Privateが手元
Publicがリモート

Deploy keyとして使うための鍵ペアの生成

ローカルマシンで鍵ペアを生成する

```
$ ssh-keygen -m PEM -t rsa -C "uchida@c.titech.ac.jp"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/uchan/.ssh/id_rsa): ./id_rsa
Enter passphrase (empty for no passphrase):  ←― 空のまま
Enter same passphrase again:
Your identification has been saved in ./id_rsa.
Your public key has been saved in ./id_rsa.pub.
The key fingerprint is:
SHA256:oiPQ4RAqeN+FieCAZQJkpOUCcjfLZ9SBS9csIFNfJI0 uchida@c.titech.ac.jp
The key's randomart image is:
+---[RSA 2048]----+
|%=* =.++oBo      |
…
|                 |
+----[SHA256]-----+
```

ssh-keygenの紹介

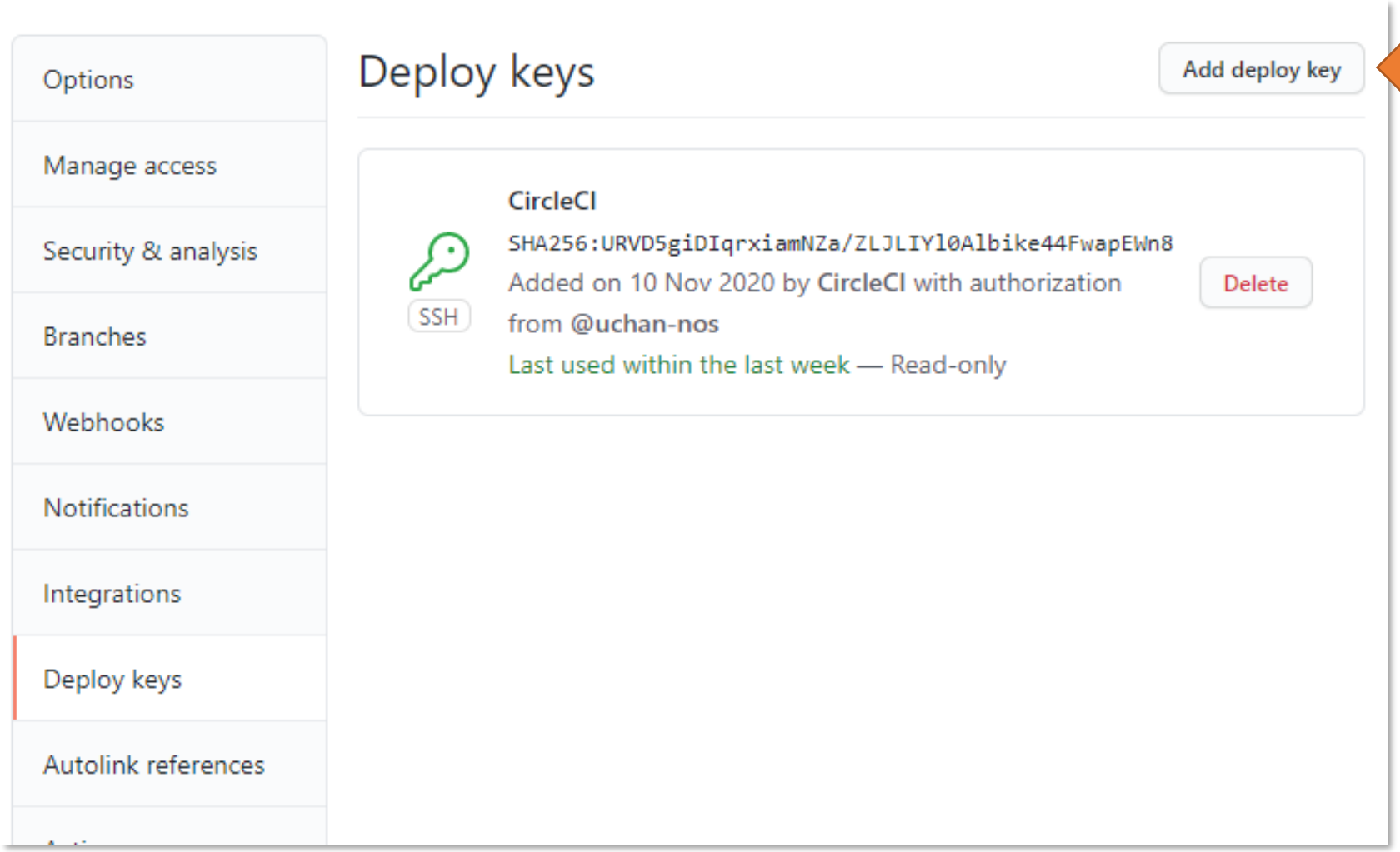
```
$ ssh-keygen -m PEM -t rsa -C "uchida@c.titech.ac.jp"
…
$ ls
a.c  a.out  bar  deploy.sh  foo  id_rsa  id_rsa.pub
```

Private key　Public key

- SSH用の鍵ペアを生成（gen）するコマンド

オプション	意味
-m PEM	PEM形式で保存（CircleCIによる要求）
-t rsa	RSAアルゴリズム（CircleCIによる要求）
-C "uchida@..."	鍵に付けるコメント

GitHubへDeploy keyを追加 1/2



Public key
を追加する

リポジトリの
Settings >
Deploy keys

GitHubへDeploy keyを追加 2/2

Tokyo Tech

Deploy keys / Add new

Title

Circle CI deploy key

任意の名前

id_rsa.pub
の内容をコピペ

Key

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC1ISzmz5CSH9e/c9DmZ8cIDkNiD8RFVEs
4G05DTOL7Z3GGAyuUgxXppsN1JWcSLtSS1OyyICu0spile+0UyDiFgv5I7DU+vIYgG7I
oer/FyygILK67BATTYNBe40ALcca0R4Umpij6+v+GYBEI+Os6Pw3CpS3ZCuSV1bwKeT
IJPkRYb7F5fJlu6usat8PAyDw0kR3WcdC2WoPmKhqJjUELfX2dQsG4t9iJPsB0+DKS0Yn
rpv9zfdkZVsJvxe4/KVaqTZ33ErxTOgMCZHhOCyVYgZjmYe2Tr3COY8+iBXIHo/m4Oz2
Yh8Okn7qu1VHfQY7fvPJnv/zFtuznY2ZmV5qP uchida@c.titech.ac.jp

Allow write access
Can this key be used to push to this repository? Deploy keys always have pull access.

Add key

Deploy keys

Add deploy key

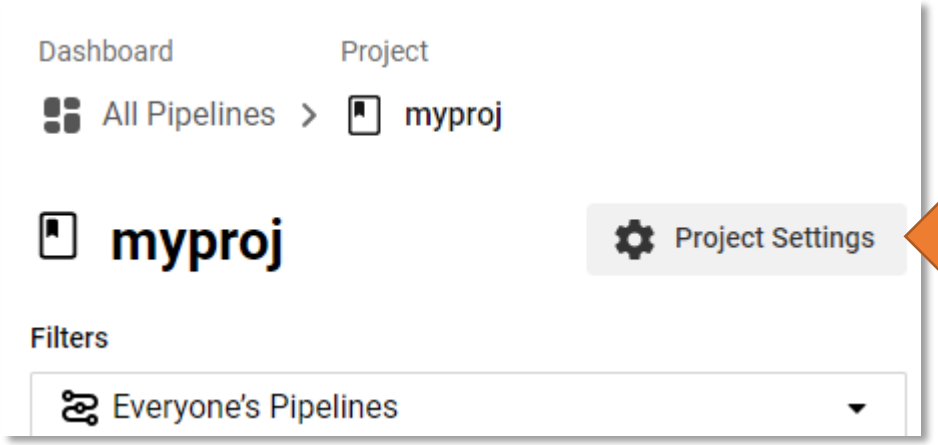
CircleCI
SHA256:URVD5giDIqrxiamNZa/ZLJLIYl0Albike44FwapEWn8
Added on 10 Nov 2020 by CircleCI with authorization from @uchan-nos
Last used within the last week — Read-only
SSH
Delete

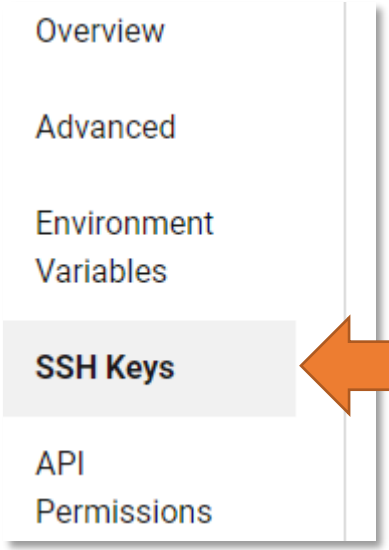
Circle CI deploy key
SHA256:oiPQ4RAqeN+FieCAZQJkpOUCcjfLZ9SBS9csIFNfJI0
Added on 13 Nov 2020 by @uchan-nos
Never used — Read/write
SSH
Delete

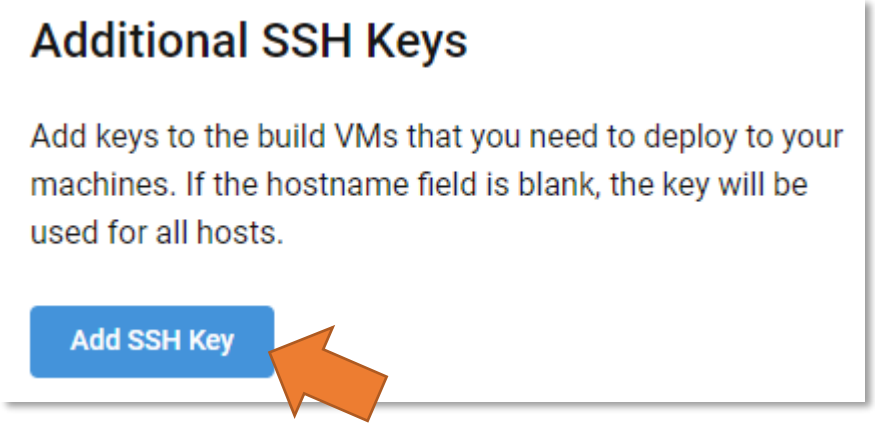
※Public keyの内容は公開されても大丈夫

CircleCIへDeploy keyを追加 1/2

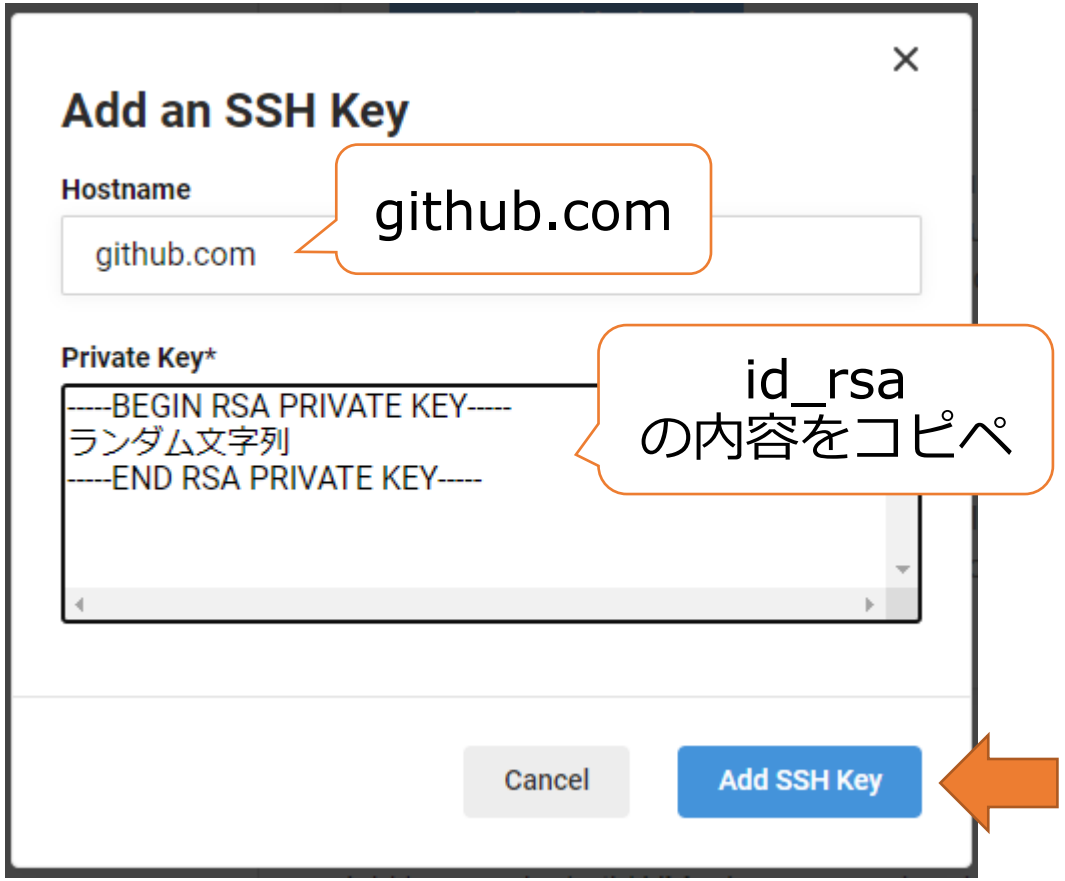
https://app.circleci.com/pipelines/github/uchan-nos/myproj

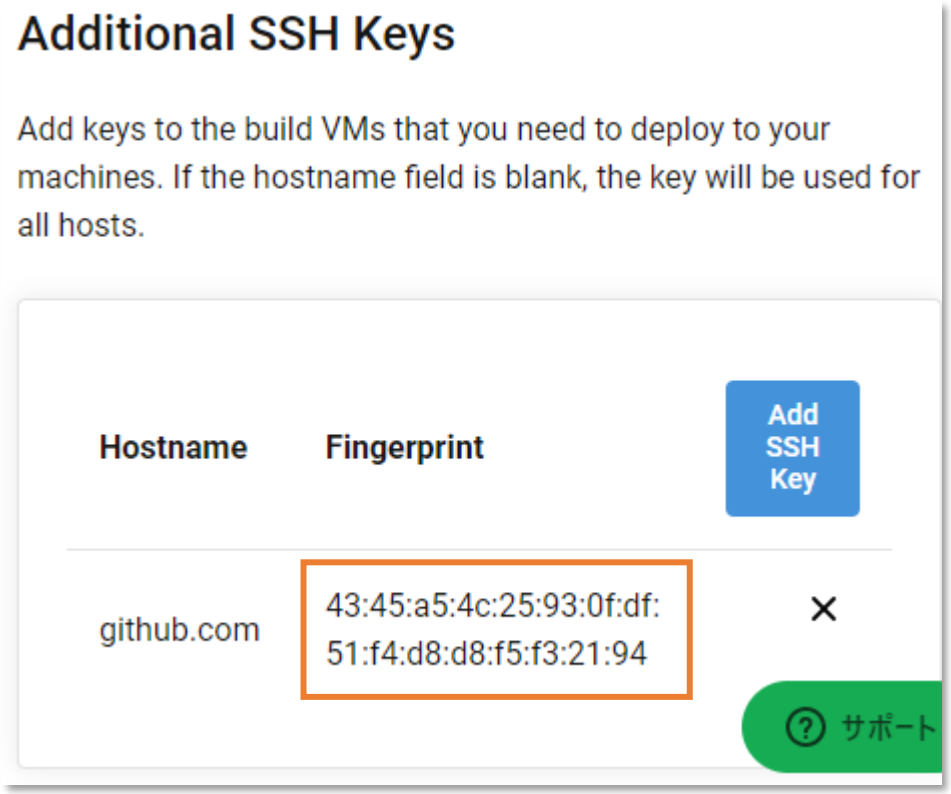






CircleCIへDeploy keyを追加 2/2





※Private keyの内容は公開してはいけない！！

Fingerprintをメモする

config.ymlに鍵を設定

config.ymlに，デプロイに用いる鍵を記入する必要がある

.circleci/config.yml

```
version: 2
jobs:
  build:
    docker:
      - image: quay.io/cybozu/ubuntu-dev:20.04
    steps:
      - checkout:
      - run: echo "Hello, world!"
      - add_ssh_keys:
          fingerprints:
            - "43:45:a5:4c:25:93:0f:df:51:f4:d8:d8:f5:f3:21:94"
      - run: sh ./deploy.sh
```

デプロイスクリプトの紹介

deploy.sh

```
#!/bin/sh -ex
target_branch="ghp-deploy"
git config --global user.name "CircleCI deployer"
git config --global user.email "<>"
git checkout $target_branch
git reset --hard origin/main

gcc -o a.out a.c
echo "output of a.out: $(./a.out)" > a.txt

git add a.out a.txt
git commit -m "[skip ci] updates GitHub Pages"
if [ $? -ne 0 ]; then
  echo "nothing to commit"
  exit 0
fi
git remote set-url origin "git@github.com:uchan-nos/myproj.git"
git push -f origin $target_branch
```

a.c and a.txt

a.c

```
#include <stdio.h>

int fib(int i) {
  if (i <= 1) {
    return i;
  }
  return fib(i - 2) + fib(i - 1);
}

int main() {
  printf("fib(10) = %d¥n", fib(10));
}
```

```
$ curl https://uchan-nos.github.io/myproj/a.txt
output of a.out: fib(10) = 55
```

演習2

授業で作ったリポジトリのCircleCIで
GitHub Pagesへデプロイせよ

- CircleCIの中で動的にファイルを生成し，GitHub Pagesへ追加してみよう
  - 「動的にファイルを生成」とは
    a.outを使ってa.txtを生成するような感じ
- 制限時間40分

継続的デリバリ

- CD: Continuous Delivery
- 自動的にビルドし，検証環境にデプロイし，テストすることで，本番環境にデプロイ可能な状態にすること
- 継続的インテグレーションをさらに一歩進めた考え方

- 「CI/CD」とよくまとめて語られる

継続的デプロイ

- CD: Continuous Deployment
- 継続的デリバリと違い，本番環境へ自動でデプロイする
- 開発者はGitHubに変更をpushし，待っていれば本番サービスが更新される
- →1日に何度もリリースするスタイルと相性が良い

- 「CI/CD」のCDをこっちの意味で捉える派閥もある

デプロイは失敗する

- 人間はミスをするから自動化すべし
- ただ，自動化しても失敗が完全になくなるわけではない
  - 自動化したプロセスが安定するには全てのコーナーケースへの対処が必要=ほぼ無理
- 様々な考慮漏れポイント
  - Disk full，ネットワーク故障
  - サーバ過負荷による処理のタイムアウト
  - OOM killerによる突然死
  - バージョンアップによるAPI挙動変化
  - なぜかサーバプロセスがグレースフルに終了せず，アップデートに失敗する

冪等性（Idempotence）

- ある操作を1回あるいは複数回実施しても同じ結果になる性質
- 冪等性が満たされたコマンドの例
  - $ mkdir -p abc
  - ディレクトリabcの有無に依らず，生成された状態になる
- ある操作が，アクションではなく最終状態を規定するものなら冪等性がある
  - 冪等性がない：DBにテーブルHOGEを追加し初期値を挿入
  - 冪等性がある：HOGEを削除＆追加し初期値を挿入
  - 冪等性がある？：HOGEがなければ追加し初期値を挿入

DBへのテーブル追加にみる冪等性

- テーブルHOGEを追加し初期値を挿入
  - 既にHOGEが存在する場合，エラーになる→**冪等ではない**
  - （HOGE.1のような名前で追加されるシステムもあるかも）
- テーブルHOGEを削除＆追加し初期値を挿入
  - もしHOGEが存在するなら削除し，改めてHOGEを追加
  - 初回デプロイ時のみHOGEが存在しない
  - 初回も，次回以降も，デプロイ後は同じ状態になる→**冪等**
- テーブルHOGEがなければ追加し初期値を挿入
  - HOGEの有無に関しては冪等
  - 挿入されている値は**冪等ではない**

演習3

冪等でない操作をいくつか挙げ，
それが冪等にならない場合を説明せよ

- 例: wgetによるダウンロード
  - 同名ファイルがある場合，hoge.1，hoge.2のような名前になってしまう

- 制限時間10分

構成管理ツール

- 冪等なデプロイを支援するツール
- Ansible, Chef, Puppetなど

- デプロイを冪等にするポイント=宣言的記述
  - 宣言的=目標状態を記述する
  - 手続き的=操作を記述する
- 構成管理ツールを使っても冪等になるとは限らない
  - 手続き的な設定をしてしまうと冪等にならない

Ansible

- リモートホストにSSHで接続し，様々な処理を行う
  - ローカル→リモートホストへのファイルコピー
  - リモートホストでの任意コマンド実行
  - など
- 組み込みの冪等な操作が豊富に提供されている
  - パッケージインストールの操作では「希望するバージョンがインストールされていること」のような指定が可能
  - インストールされていない→インストールする
  - インストールされているが古い→更新する
  - インストールされている→何もしない

AnsibleでMySQLにDBとユーザーを追加

```yaml
- name: Create Application Database
  mysql_db:
    name: "{{ dbname }}"
    state: present

- name: Create Application DB User
  mysql_user:
    name: "{{ dbuser }}"
    password: "{{ upassword }}"
    priv: "*.*:ALL"
    host: '%'
    state: present
```

目標状態を指定する
→冪等性を担保

Ansibleを使っても冪等にならない例

```
- name: add a configuration
  shell: echo "set colorcolumn=80" >> /home/user/.vimrc
```

- Ansibleで冪等ではないスクリプトも容易に書ける
- 任意コマンド実行は特に注意

コミュニケーション

コミュニケーション

- 情報を伝達すること
- 仕事の議論においては適切に情報を伝えることが大事
  - いわゆる「コミュ力」とは違う
- 適切に：
  - 正しい情報を
  - 相手を尊重しながら
  - 適する時期に
  - 適する方法で

オンライン・コミュニケーション・ツール

- 直接的な伝達相手が想定されているツール
  - メール，メーリングリスト
  - テキスト/ボイスチャット
  - テレビ会議
- 組織内に情報を伝達するツール
  - スケジューラ（個人やチームの予定共有）
  - 電子掲示板（組織内に広く広報）
  - ファイル共有
- 全世界に情報を伝達するツール
  - Webサイト，ブログ
  - SNS

ストックとフロー

- 情報の特徴を大別すると2つ
- ストック
  - 多少古くなっても重要性を失わず，蓄積したくなる情報
  - ソフトウェアの仕様書や設計書，社内規則集，辞書など
  - ディレクトリ（カテゴリ）やタグ等で整理されていると利用価値が高まる
  - Webページや電子掲示板，ファイル共有などとの相性が良い
- フロー
  - その場限りの情報や，会話や議論のために提供される情報
  - 時期が過ぎると急速に価値が低下
  - チャットやテレビ会議，SNSなどとの相性が良い

チーム開発とコミュニケーション

- チームで仕事をするにはコミュニケーションが必須
  - メンバー間で情報が伝達されない集団はチームか？
  - チーム：共通の目標に向かって協力する人の集まり
- コミュニケーションの難しさ
  - きちんと伝えるのはそもそも難しい
  - 言い方を間違えると意図に反して伝わる
  - 忙しいときに割り込みたくない
- 適切なコミュニケーション→HRT

HRT

- HRT：チームで働くときの「三本柱」
  - 『Team Geek』より
- 謙虚（Humility）
  - 世界の中心は君ではない。君は全知全能ではないし、絶対に正しいわけでもない。常に自分を改善していこう。
- 尊敬（Respect）
  - 一緒に働く人のことを心から思いやろう。相手を1人の人間として扱い、その能力や功績を高く評価しよう。
- 信頼（Trust）
  - 自分以外の人は有能であり、正しいことをすると信じよう。そうすれば、仕事を任せることができる。

議論のフレームワーク

- 効果的な議論をサポートするフレームワークがいくつもある
- ここでは，サイボウズ社内で使われる「問題解決フレームワーク」を紹介する

問題＝
理想と現実のギャップ

	解釈	事実
理想（目標）	部屋が暖かい	室温が23℃
現実	部屋が寒い	室温が15℃

本当の理想

- 「理想=部屋が暖かい」は本当か？
  - 部屋が寒いままでも，服をたくさん着れば？
  - 湯たんぽを使っても良さそう
- 本当の理想は「快適に過ごせること」かもしれない
  - 顧客が本当に欲しいのは速い馬ではなく目的地に早く着くこと
- 本当の理想を共有できるとより良い議論が可能

演習1

現状の大学運営（授業形態や研究室運営など）に関し問題を1つ挙げよ

- 問題=理想と現実のギャップ
  - 理想は何？現実はどうなっている？
- 4象限にまとめ，チームの代表者が発表
  - Jamboard
    https://jamboard.google.com/d/17kx9Ix1XSTPCAbk9NujKT9uQsp8W5FG6reGcgoJm6QQ/edit?usp=sharing
- 制限時間25分

課題=問題を解決するために行う行動

- 室温の例
  - 暖房を入れる
  - 上着を着る
  - 湯たんぽを使う
- 1つの問題に対し，通常は複数の課題を設定できる
  - 短期的課題，長期的課題
  - 達成が簡単な課題，難しい課題
- 課題の遂行まで含めて問題解決

振り返りのフレームワーク

- 振り返り
  - 次回以降の作業効率を上げる
  - 困りごとを解消する
- KPT法
  - Keep
  - Problem & Try
- YWT法
  - やったこと
  - 分かったこと
  - 次にやること

演習2

YWT法を用いて3Qを振り返る

- 皆でやる
- Jamboard
  - https://jamboard.google.com/d/17kx9Ix1XSTPCAbk9NujKT9uQsp8W5FG6reGcgoJm6QQ/edit?usp=sharing
- 制限時間40分

レポート提出のトピック名

- 11/20の前半はTOPIC=deploy
- 11/20の後半はTOPIC=comm

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）

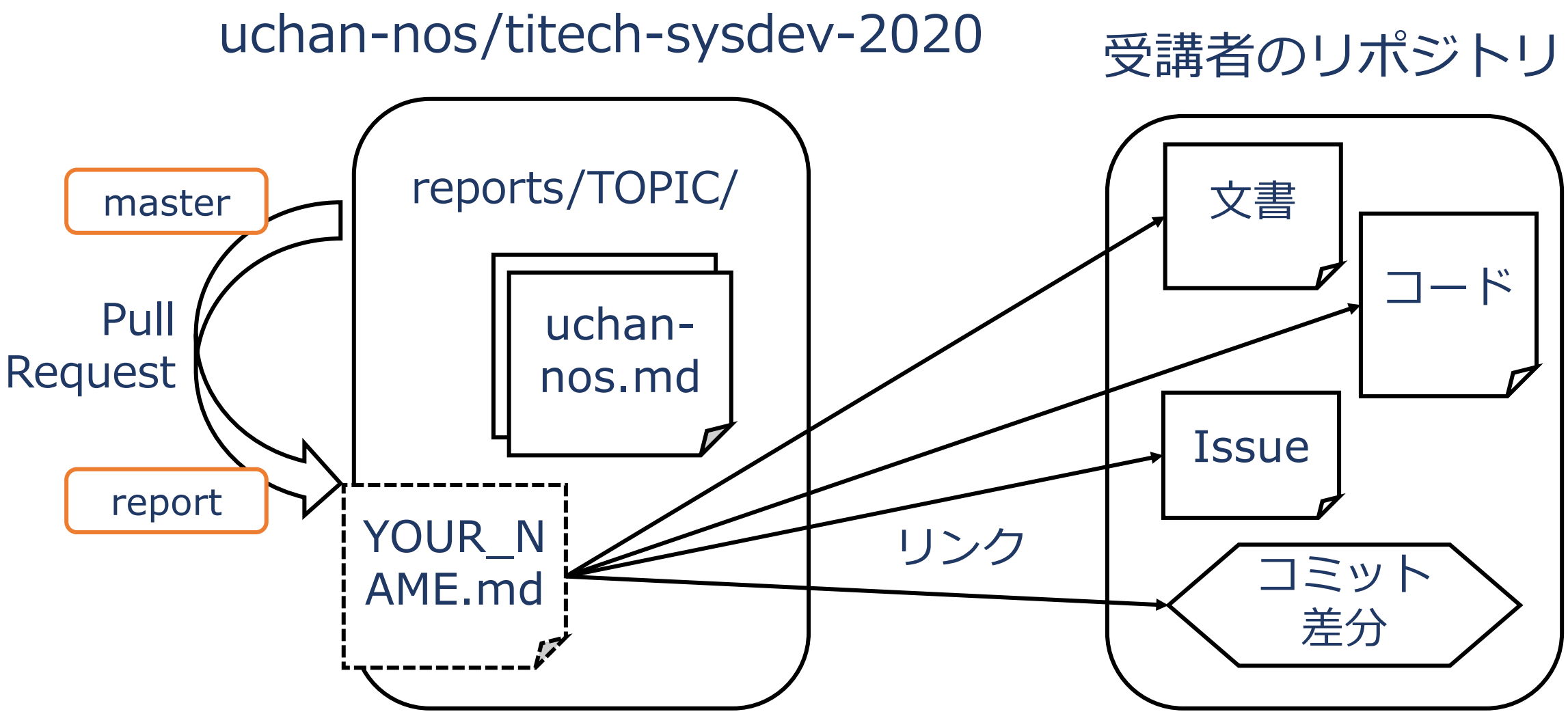
レポートの送り方

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

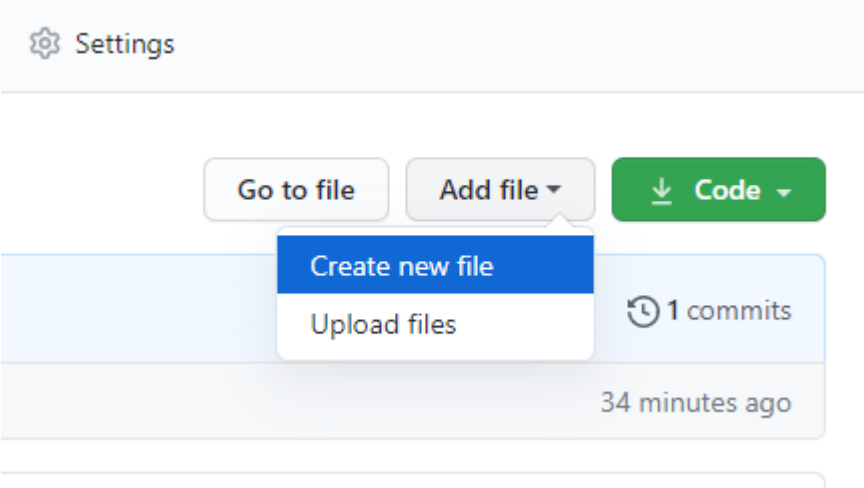
レポート

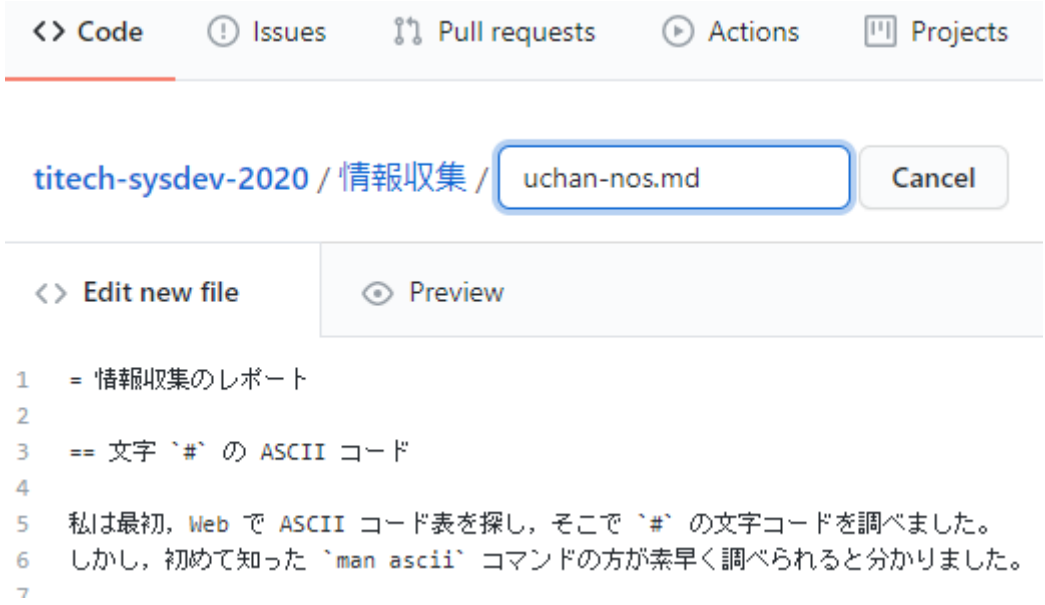
- reports/TOPIC/YOUR_NAME.md
- このファイルに課題への回答を記載する
- 必要なら以下のものを含める
  - Issueへのリンク
  - コミット差分へのリンク
    https://github.com/HOGE/REPO/compare/COMMIT1...COMMIT2
  - その他
- 要するに，成績評価に必要な情報をYOUR_NAME.md自体に記載するか，そこから辿れるようにする

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

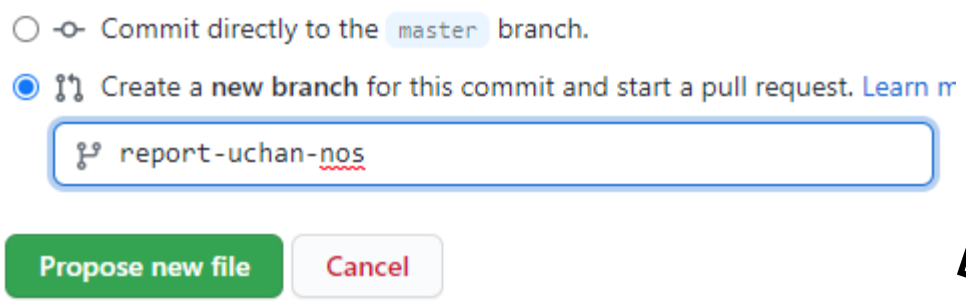




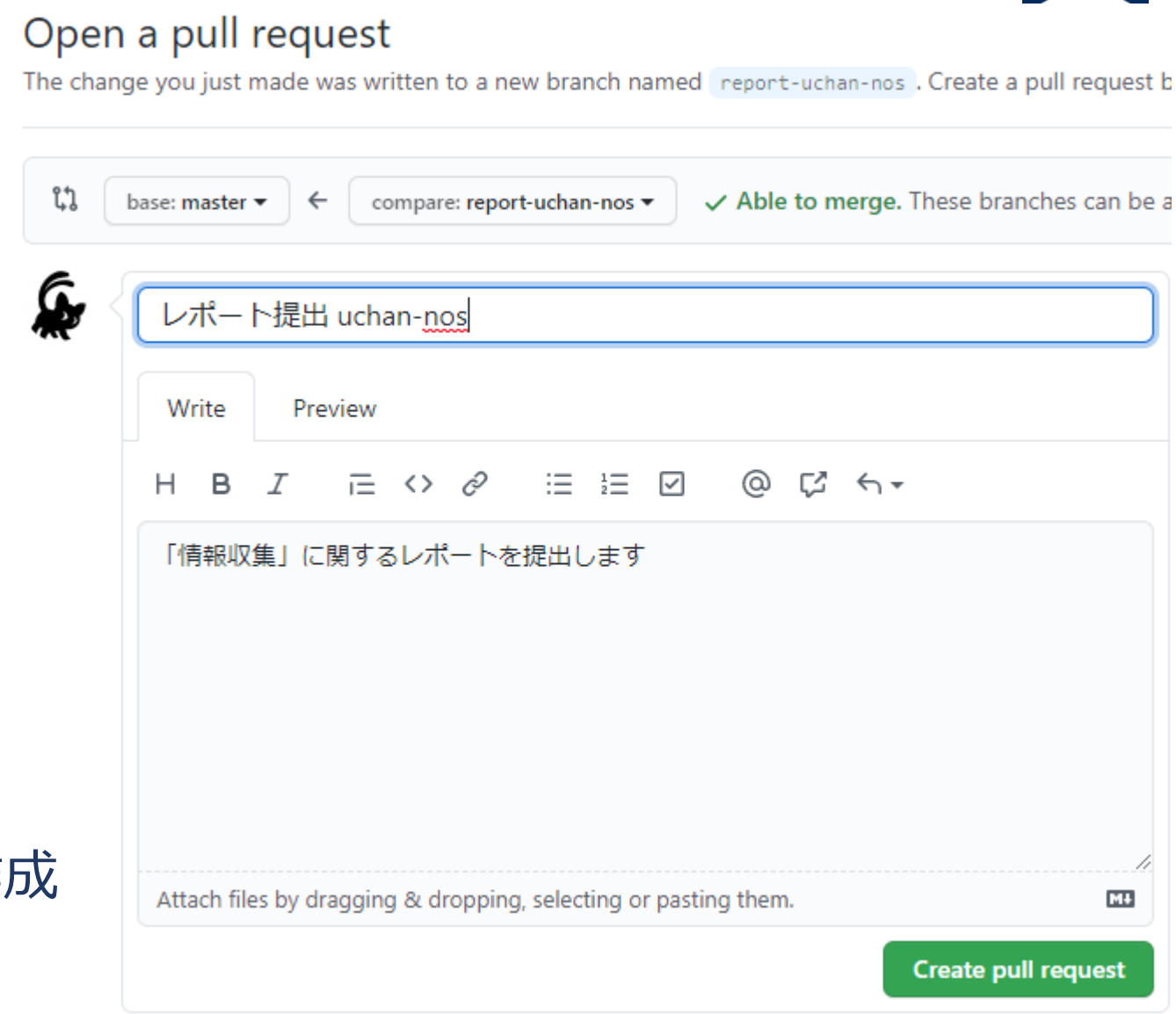
ファイルを新規作成

YOUR_NAME.mdの内容を記述

レポートの送り方 2/2

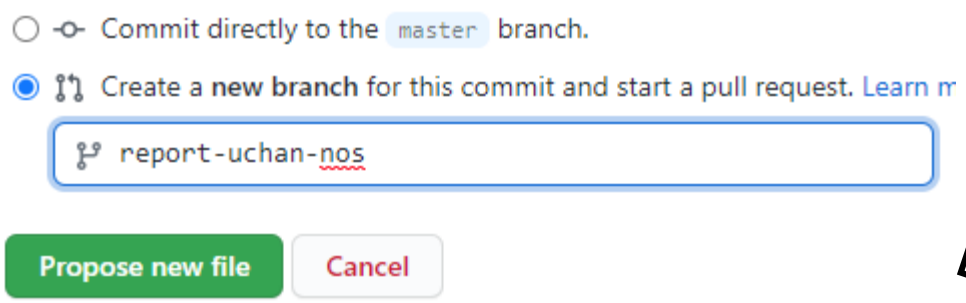


新規ブランチにコミット

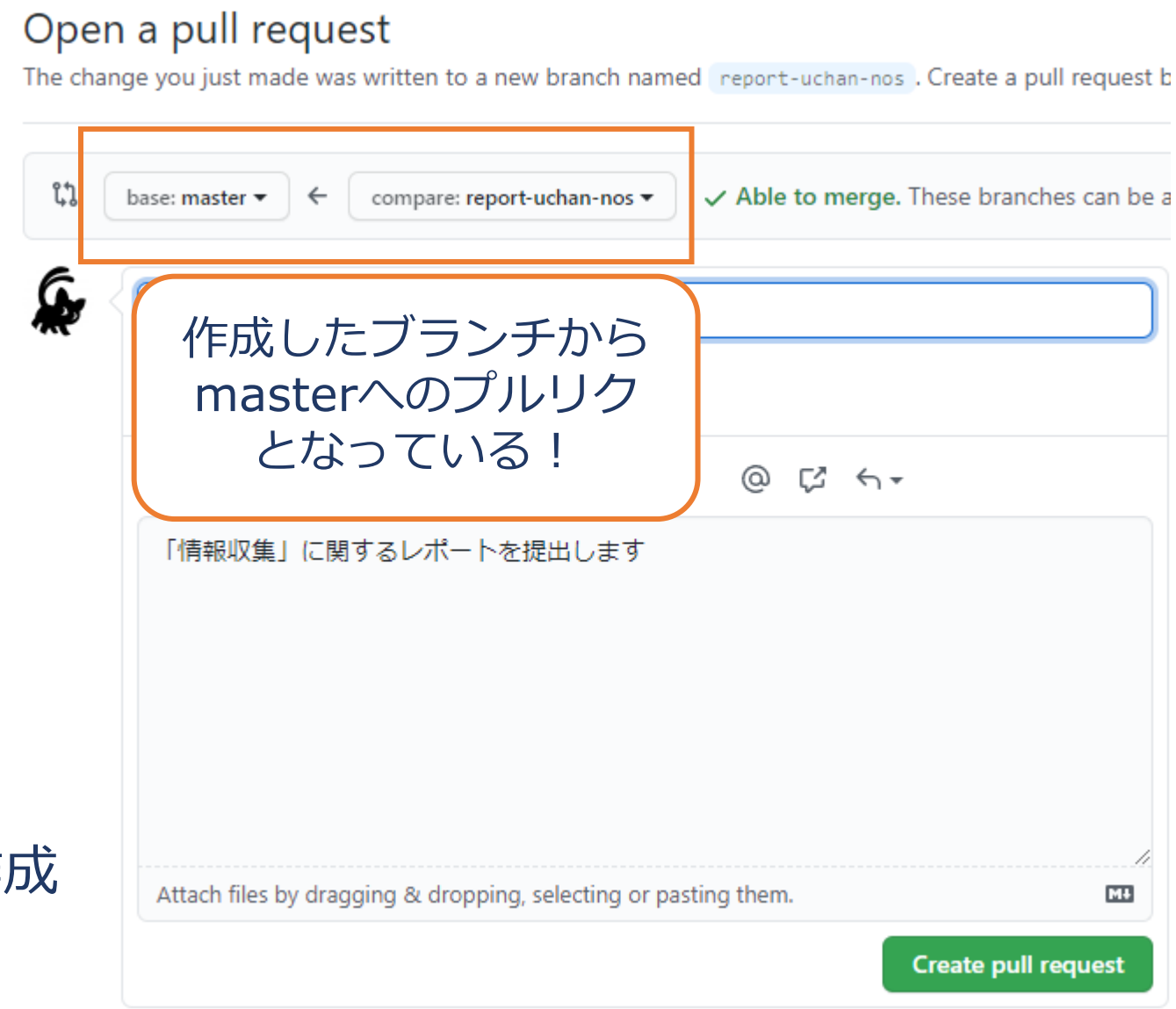


プルリクを作成

レポートの送り方 2/2



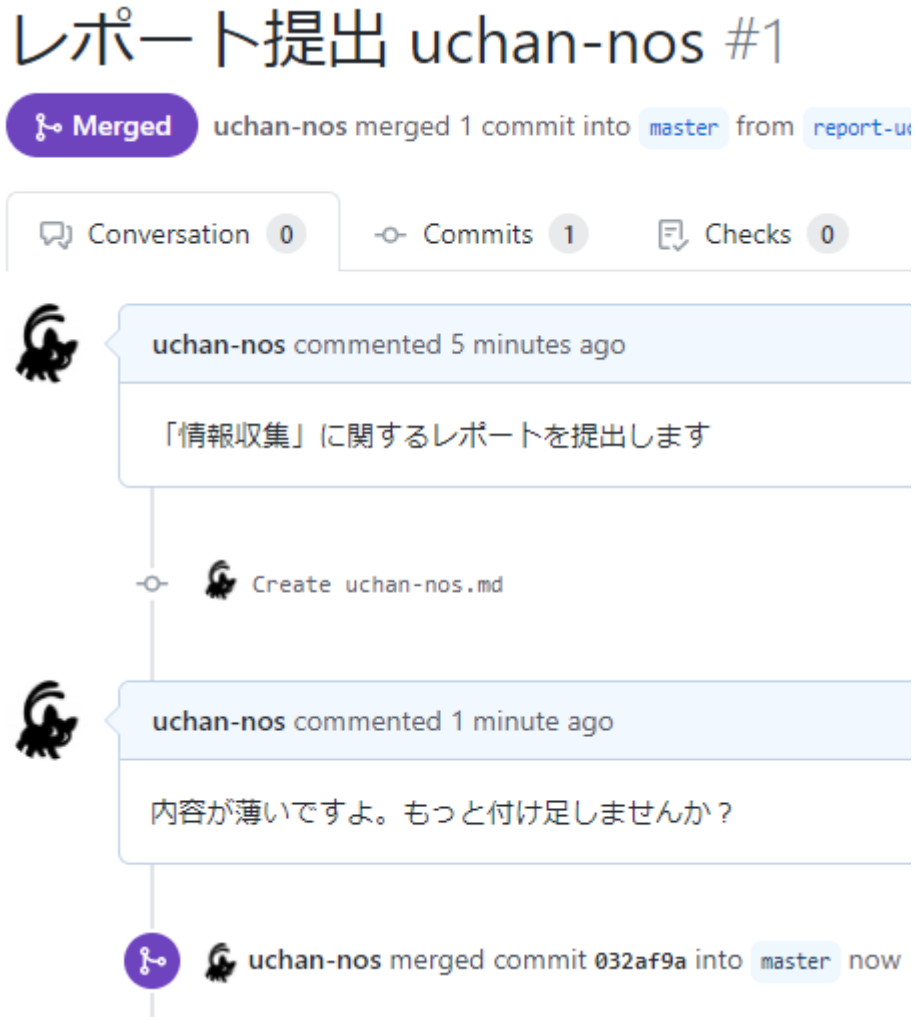
新規ブランチにコミット



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



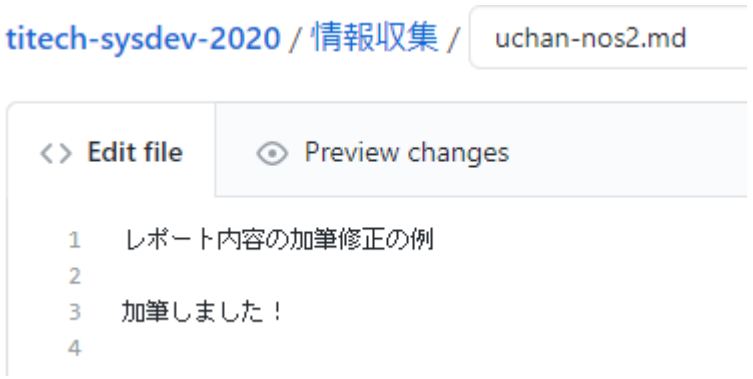
プルリクにコメントが付くことも

Merged：レポート受理済み

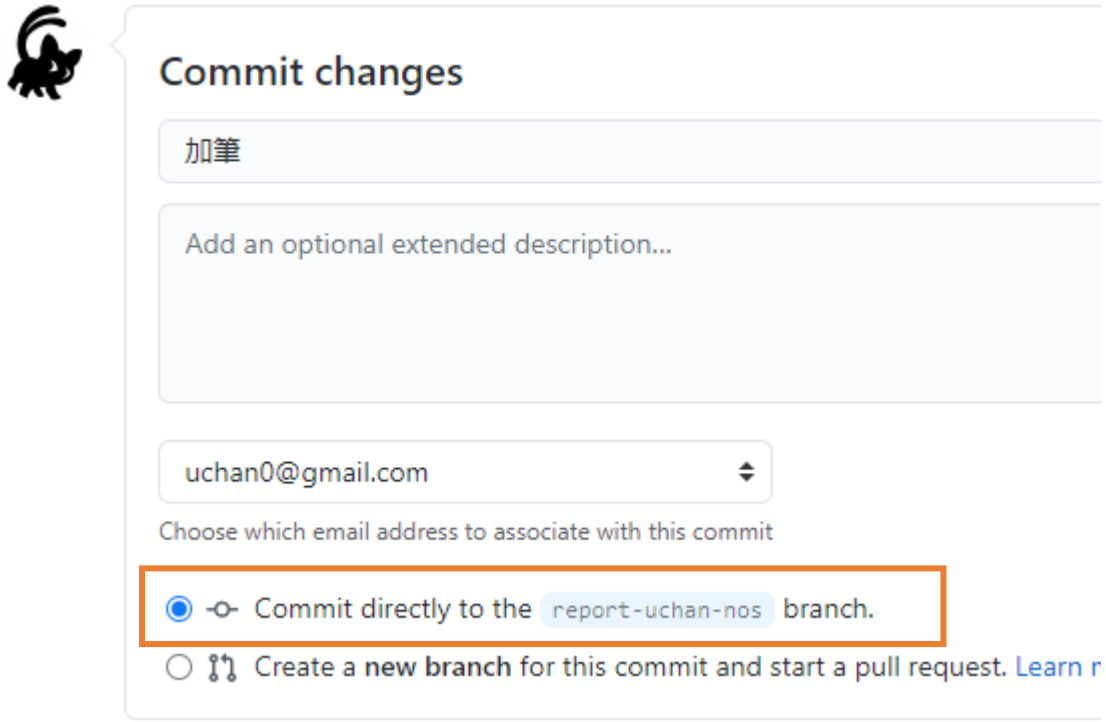
レポートの更新 1/2

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

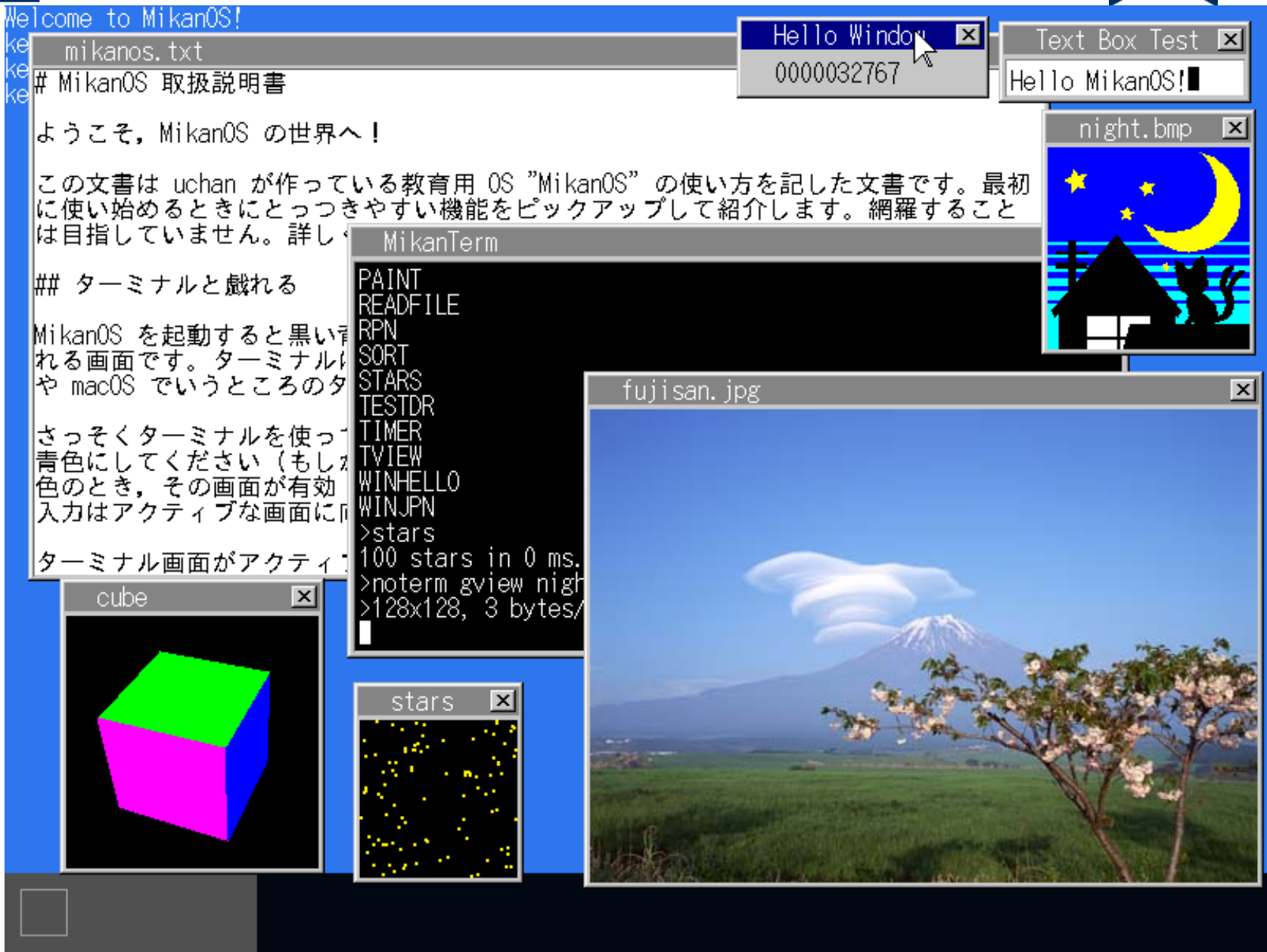
Pull Requestに
コミットが追加されていく

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

4Qの予告

- 初回は
  12/04（金）
  14:20

- 自作OSの講義です

作る予定の
OS完成画面 →