Gitによるバージョン管理

2020年10月23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

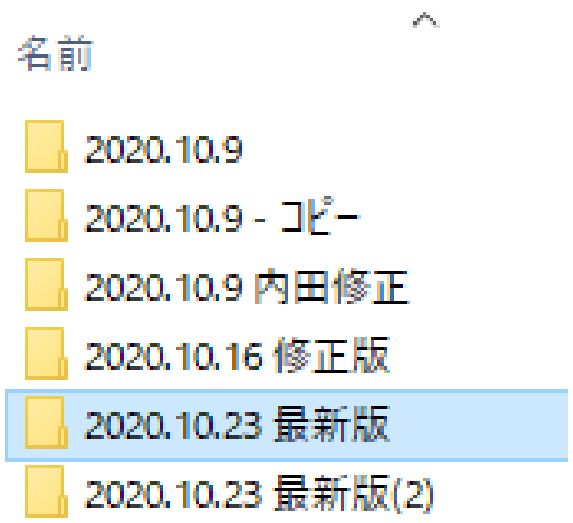
- 課題を含めたリポジトリとレポートを作成し，提出
- 初回（10/2）説明したので詳しい話はしないつもり
  - 改めて聞きたい方がいたらお知らせください

Gitによるバージョン管理

バージョンを管理するとは

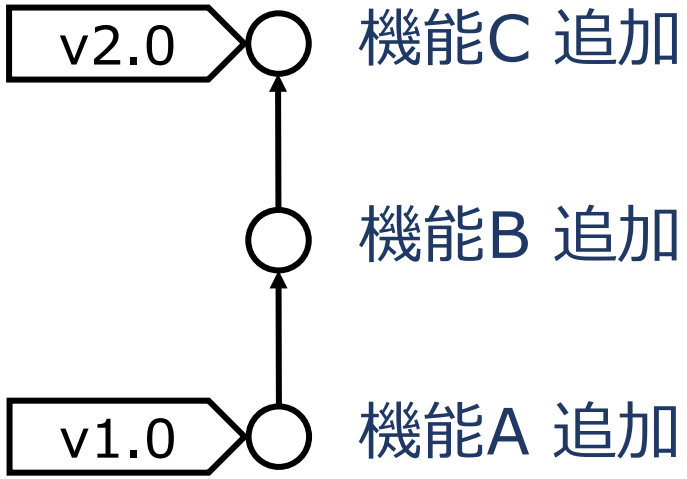
- プログラムはどんどん変更される
  - バグ修正，機能追加，変更

- バージョン間の差を知りたい
- 昔のバージョンに戻りたい
- 複数の並行するバージョンを扱いたい



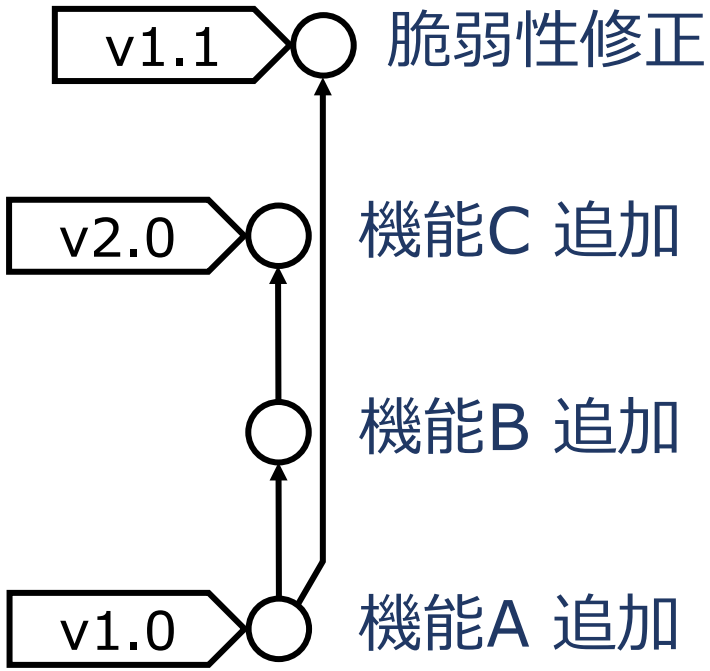
フォルダ名で管理しようとして破綻した例

複数の並行するバージョン 1/2



- とあるソフトウェアプロジェクト
- v1.0のサポート期間は1年間
  - その間にバグが見つかれば修正する
  - 機能追加・変更はしない

複数の並行するバージョン 2/2

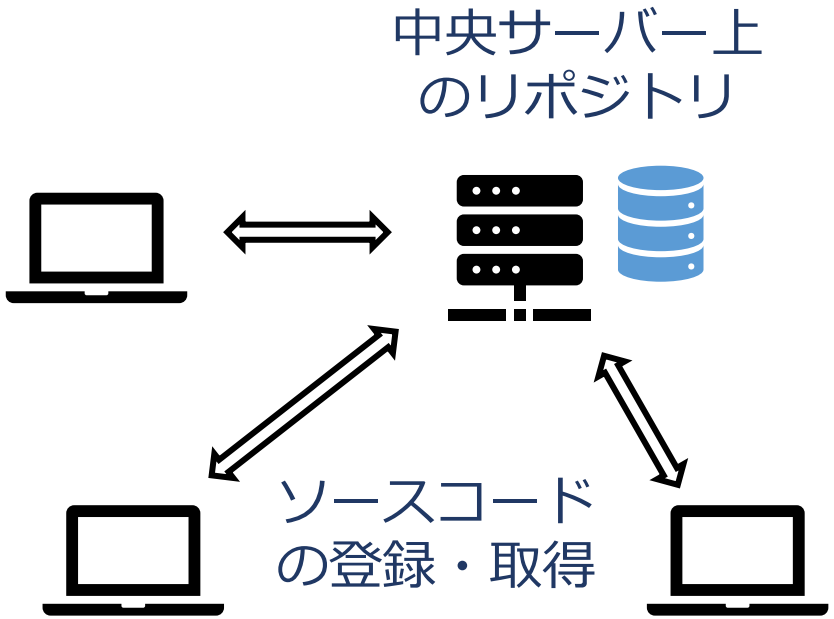


- v1.0に脆弱性が見つかった
- 緊急リリースしたい
- 機能B，Cは入れたくない
- →複数並行するバージョンの発生

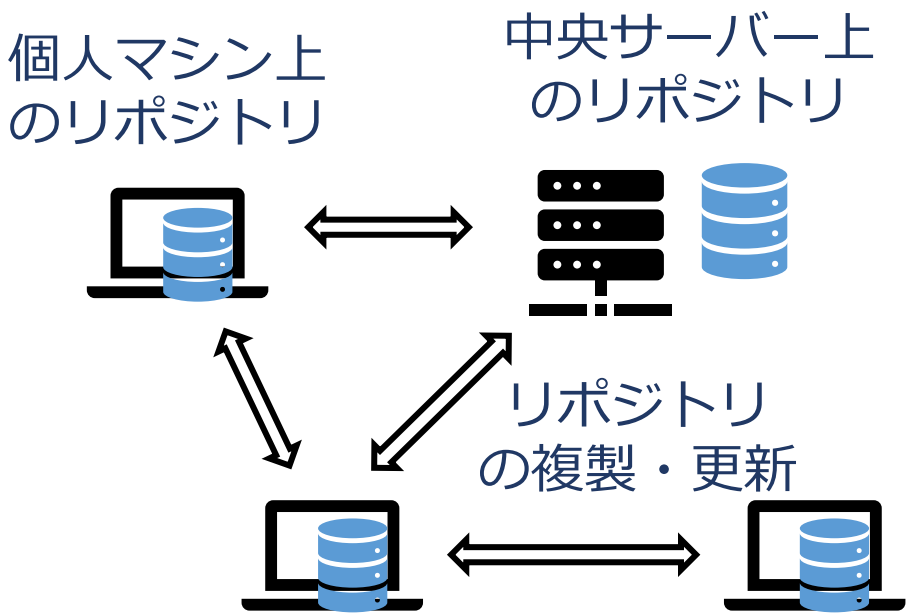
VCS: Version Control System

- ローカルのバージョン管理
  - ディレクトリのコピー
  - RCS (Revision Control System)
- 中央集中型のバージョン管理
  - CVS (Concurrent Versions System)
  - SVN (Subversion)
  - Azure DevOps Server（VSSの後継）
- 分散型のバージョン管理
  - **Git**
  - Mercurial

中央集中型と分散型 1/2



中央集中型
バージョン管理



分散型
バージョン管理

中央集中型と分散型 2/2

特徴	中央集中型	分散型
更新の登録・取得	オンライン	オフライン/オンライン
リポジトリ=歴史数	1つ	複数
リポジトリの複製	-	オフライン/オンライン
競合回避	マージ/ロック	マージ

Git

- 読み方：[gɪt]（[dʒɪt]ではない）
- Linuxカーネル開発のためにリーナスが開発した
  - Mercurialも同じ目的で開発されたが，Gitが採用された
- 分散型バージョン管理
- GitHubやGitLab，Bitbucket等のエコシステム
  - GitHubが最も普及
  - GitHub上に1.9億個のリポジトリ（2020/10/13時点）

最初に覚えるGitコマンド

コマンド	意味
git init	リポジトリを新規に作る
git add foo	ファイルfooをステージする
git commit -m "msg"	メッセージ “msg” を付けてコミット
git log	コミット履歴を見る
git status	現在の状況を表示する

演習1

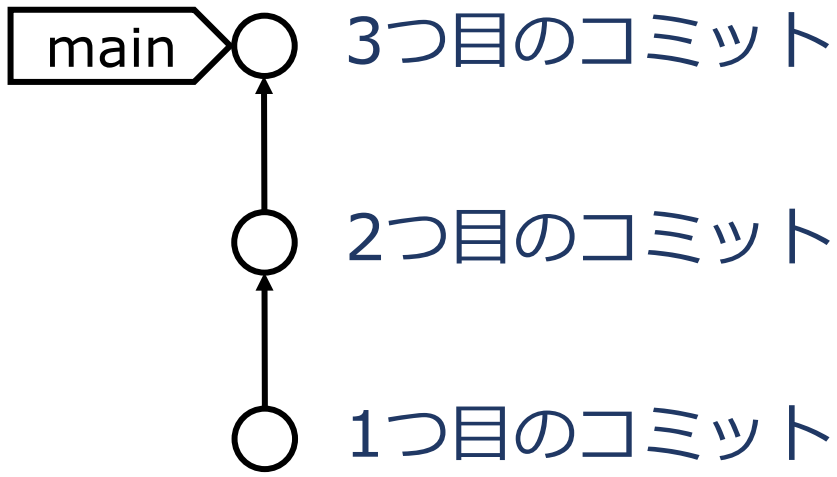
リポジトリを新規作成し，
変更を3回程度コミットせよ

- ```
  git init myproj
  cd myproj
  echo foobar > foo
  git add foo
  git commit -m "message"
  ```
- 制限時間8分

```
$ git config --global ¥
  user.email "you@example.com"
$ git config --global ¥
  user.name "Your Name"
$ git config --global ¥
  init.defaultBranch main
```

Gitの初期設定
```

演習1のゴール



- main（master）ブランチにいくつかコミットがある
  - ブランチ（枝）：コミット列の末尾についたラベル
- git logでコミット履歴を確認してみよう

講義資料の中でしばしばmasterブランチが登場する。
これはGitに何も設定しない場合のデフォルト値。
GitHubがつい最近master→mainに変えたので，今後多くの製品が追従するだろう。

```
uchan@workstation:~/workspace$ git init myproj
Initialized empty Git repository in /home/uchan/workspace/myproj/.git/
uchan@workstation:~/workspace$ cd myproj
uchan@workstation:~/workspace/myproj$ echo foobar > foo
uchan@workstation:~/workspace/myproj$ git add foo
uchan@workstation:~/workspace/myproj$ git commit -m "initial commit"
[master (root-commit) 5e7a74b] initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 foo
uchan@workstation:~/workspace/myproj$ echo "2nd line" >> foo
uchan@workstation:~/workspace/myproj$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   foo

no changes added to commit (use "git add" and/or "git commit -a")
uchan@workstation:~/workspace/myproj$ git add foo
uchan@workstation:~/workspace/myproj$ git commit -m "2nd commit"
[master 8747dd0] 2nd commit
 1 file changed, 1 insertion(+)
uchan@workstation:~/workspace/myproj$ git log
commit 8747dd099e54618294a364aeec0cf222ccaca5ca (HEAD -> master)
Author: Kota Uchida <uchida@c.titech.ac.jp>
Date:   Tue Oct 13 17:33:20 2020 +0900

    2nd commit
```

演習1のコマンド例

バージョン間の差を確認する

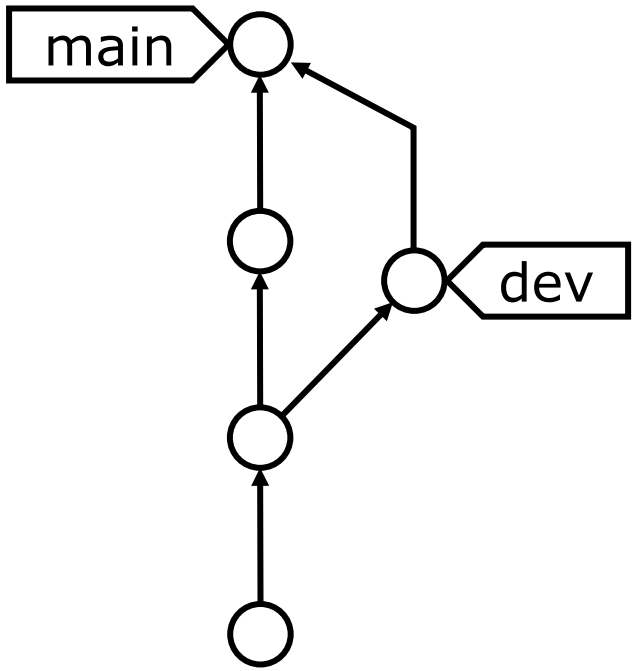
- git diff コミットA コミットB
- コミットA→コミットBの差分を見る
  - この差分をコミットAに加えるとコミットBになる

- git diff
- インデックス→ワークツリーの差分を見る
  - インデックスとワークツリーは後述
- コミット前に変更を確認するのに便利

コミット履歴を確認する

- git log
- コミット履歴
  =そのソフトウェアの歴史

- git log --graph --all
- --graph
  コミット履歴をグラフ構造で表示
- --all
  すべてのブランチを表示



コミット履歴の
グラフ構造

git log --graph --allの例

コミットID

ブランチ名

頂点

辺

```
uchan@workstation:~/workspace/myproj$ git log --graph --all
* commit 8747dd099e54618294a364aeec0cf222ccaca5ca (HEAD -> master)
| Author: Kota Uchida <uchida@c.titech.ac.jp>
| Date:   Tue Oct 13 17:33:20 2020 +0900
|
|     2nd commit
|
* commit 5e7a74b6613949346bae955d5b893701390d0546
  Author: Kota Uchida <uchida@c.titech.ac.jp>
  Date:   Tue Oct 13 17:32:34 2020 +0900

      initial commit
uchan@workstation:~/workspace/myproj$
```

演習2

各コミット間の差分を確認せよ

- コミット1とコミット2の差
  コミット2とコミット3の差
  コミット1とコミット3の差
- 制限時間3分

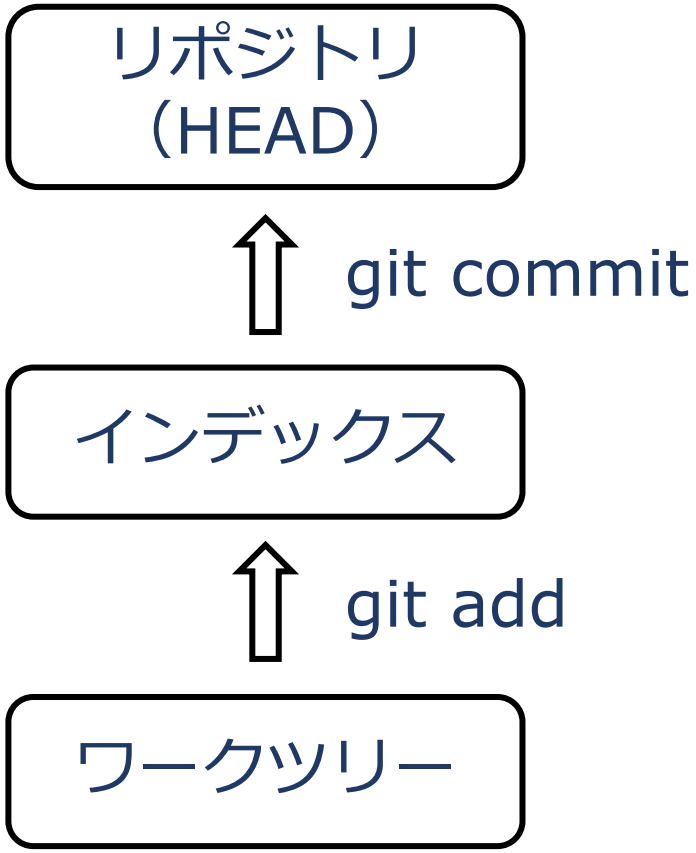
ワークツリーとインデックス

ワークツリー

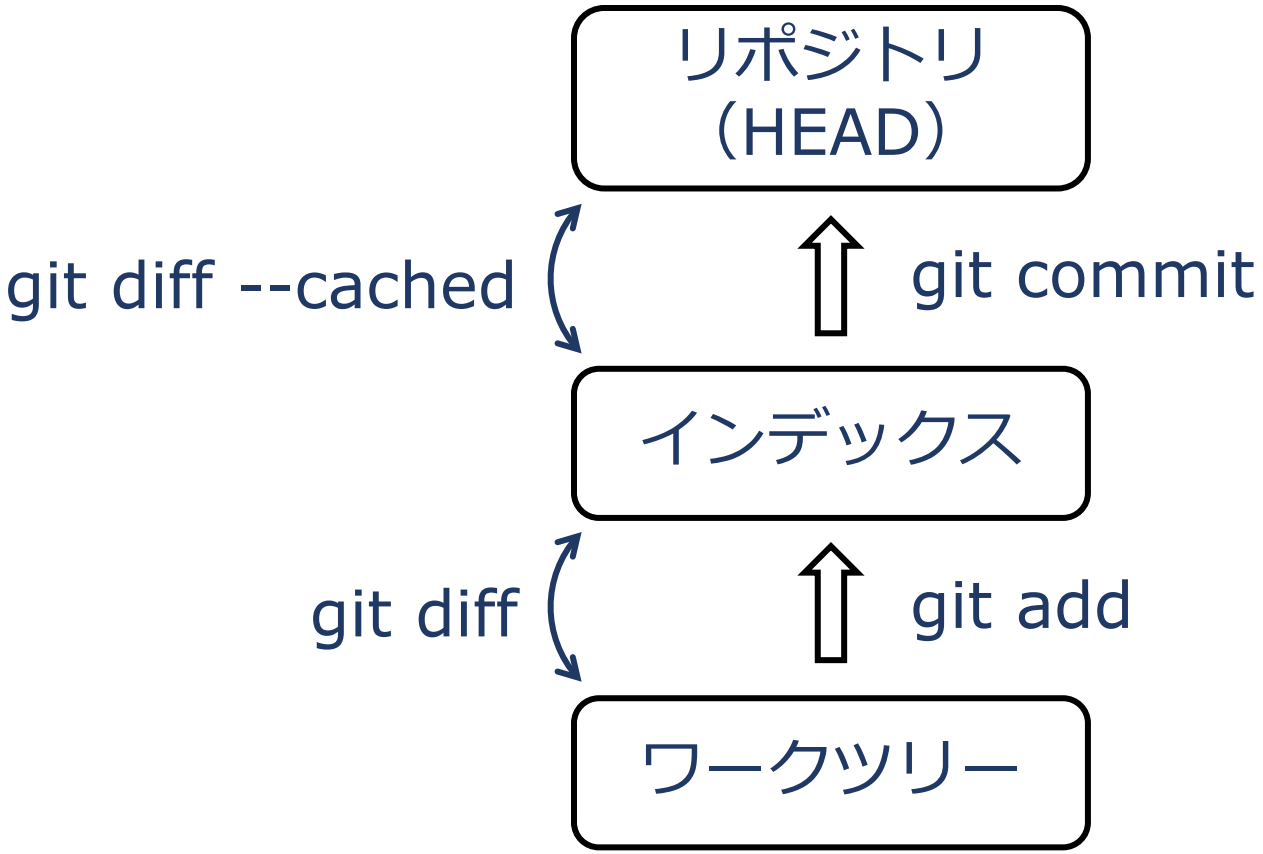
- 現在作業中のディレクトリ

インデックス

- コミットのステージング領域
- git commitのコミット対象
- git addは変更をインデックスに登録する



2種類の差分


リポジトリ
（HEAD）
git diff --cached
git commit
インデックス
git diff
git add
ワークツリー

選択的git add

- git add -p ファイル名
- ハンクを選択的にaddする

```
uchan@workstation:~/workspace/myproj$ git add -p a.c
diff --git a/a.c b/a.c
index a2d6c63..2c466d5 100644
--- a/a.c
+++ b/a.c
@@ -1,5 +1,9 @@
 #include <stdio.h>

+int f() {
+  return 42;
+}
+
 int main() {
   printf("hello, world!\n");

Stage this hunk [y,n,q,a,d,e,?]?
```

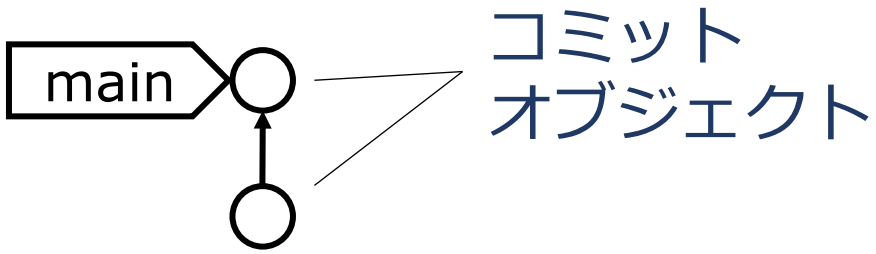
ハンク（hunk）

演習3

一部のハンクだけgit addし，
2種類の差分を確認せよ

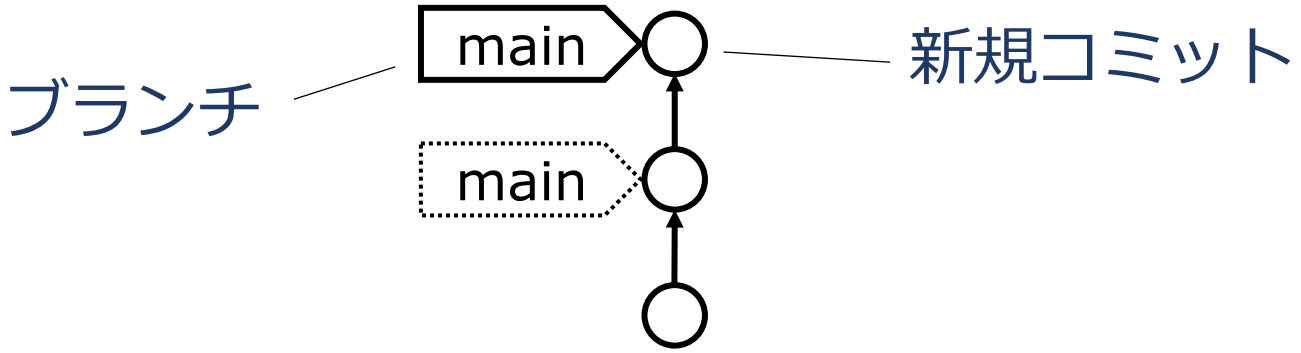
- 2種類の差分=git diff，git diff --cached
- 大量の行を含むファイルを作ると実験しやすい
- 演習が終わったら差分を全部commitしておこう
- 制限時間10分

コミットオブジェクト



- コミットオブジェクト=スナップショット
- 特定時点のファイル全体を保持する
  - 差分を持っているわけではない
- SHA1のIDで区別

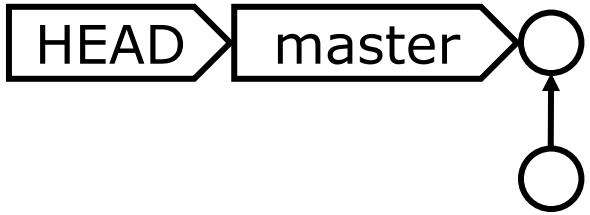
ブランチ



- ブランチ=コミットに付いたラベル（ポインタ）
- ブランチに対してコミット
  →最新コミットに追従

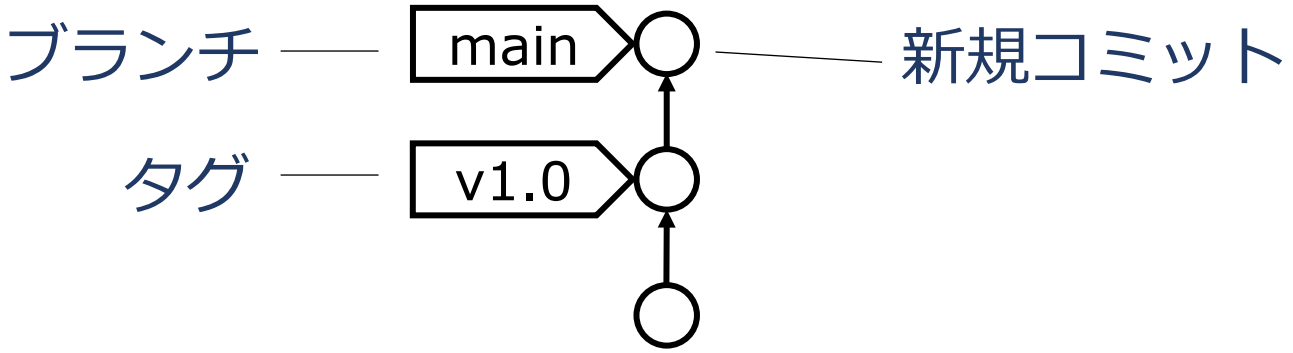
HEAD

```
--graph --all
5ca (HEAD -> master)
```



- 現在作業中のコミットを指す特別なラベル
- git diffなどの各種コマンドが暗黙に利用する
- 通常の状態：ブランチにくっついて一緒に動く
  - “HEAD -> master”: masterにくっついてる
- Detached HEAD：ブランチから外れて特定のコミットを指した状態

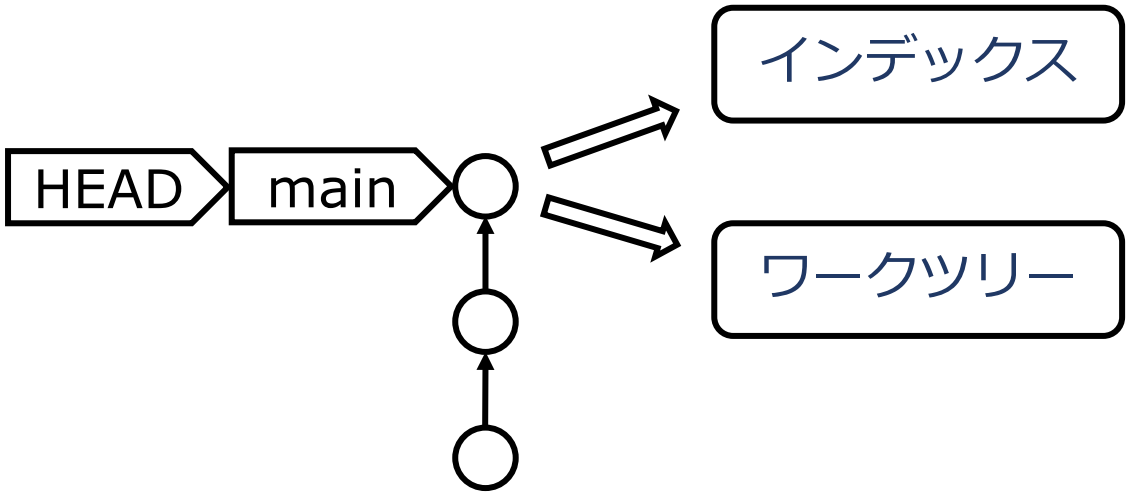
タグ



- タグ=コミットに付いたラベル
- ブランチと違い，最新コミットに追従しない
- git tag v1.0

git checkout

```
git checkout main
```



- HEADを特定のブランチやコミットに設定し
- ワークツリーとインデックスをHEADの内容に更新する
- コミットを指定するとDetached HEADとなる

演習4

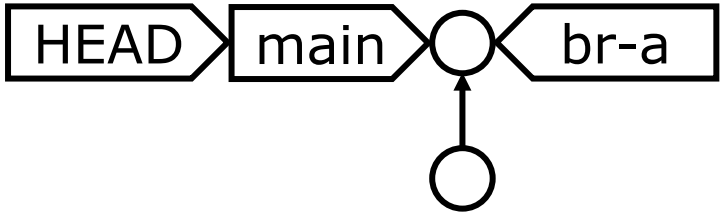
特定の時点のファイル内容を確認せよ

- 各コミットをcheckoutする
- その時点のファイルの内容を表示してみる

- 制限時間4分

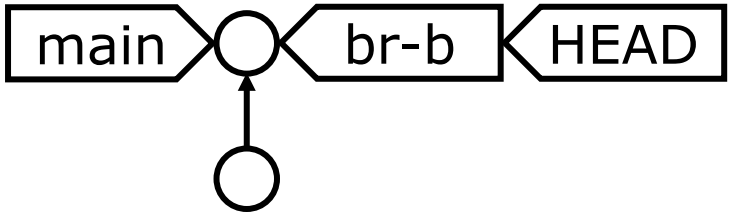
ブランチを作成する

- git branch br-a
- HEADを指すブランチを作る




```
git branch br-a
git log
df11ace71b (HEAD -> master, br-a)
```

- git checkout -b br-b
- HEADを指すブランチを作り，
  そのブランチをチェックアウトする



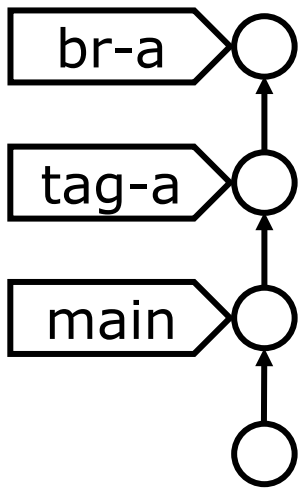

```
git checkout -b br-b

git log
df11ace71b (HEAD -> br-b, master, br-a)
```

演習5

新しく作ったブランチに
変更をいくつかコミットせよ
途中でタグを打て

- ブランチ名はbr-aとせよ
- タグ名はtag-aとせよ
- 履歴の確認
  git log --graph --all --decorate=full
- 制限時間10分

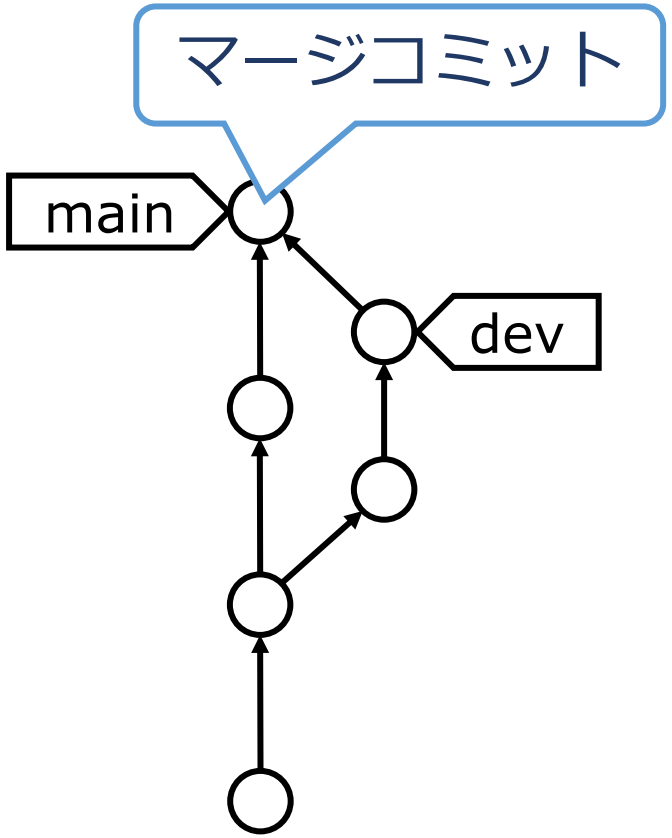


目標のコミット履歴

マージ

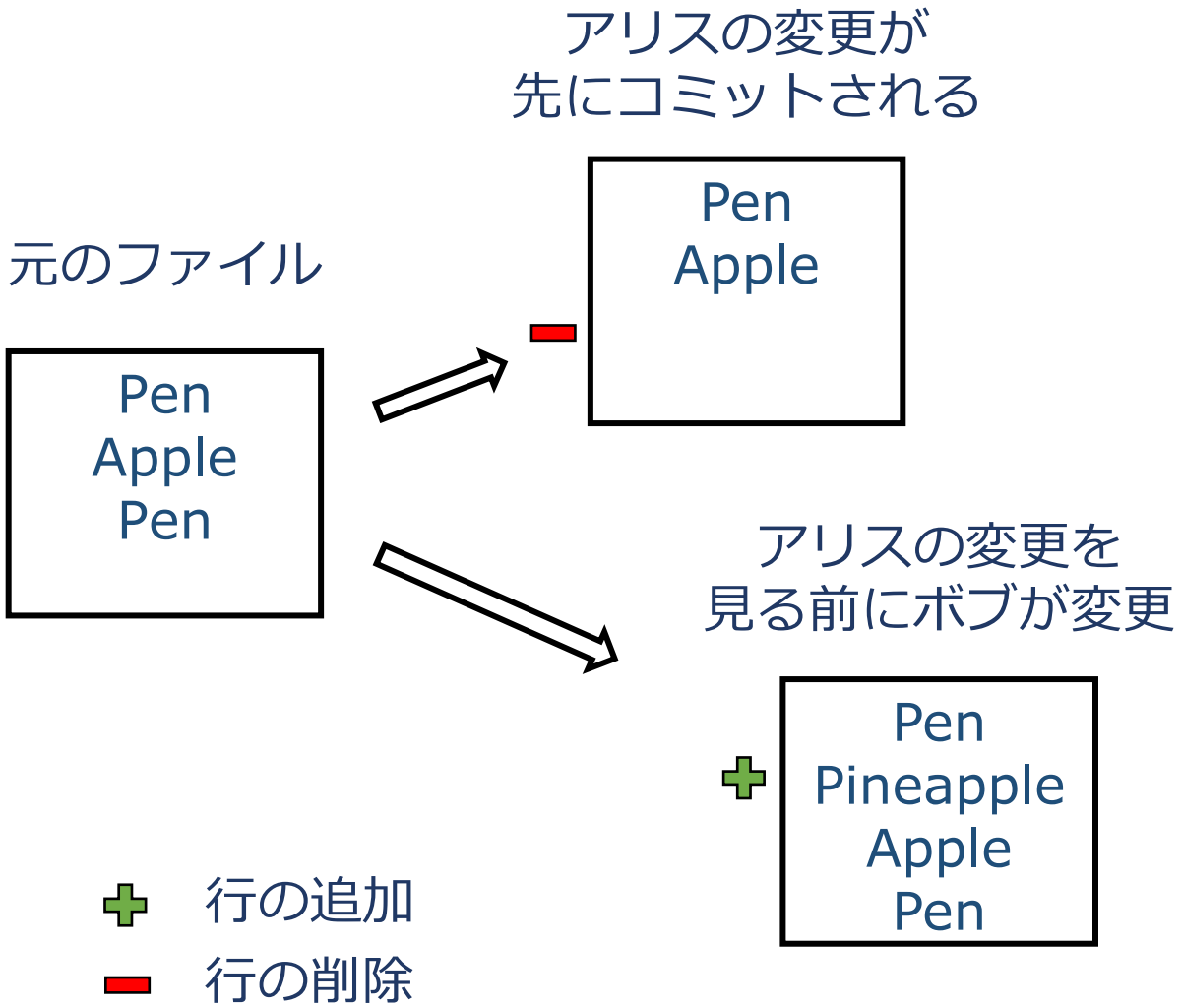
- 分岐を1つに統合すること
- 同時並行で変更を加え，それらを統合した新たなコミットを作成

- マージ方式
  - 2-way merge
  - 3-way merge
- Gitは3-way merge

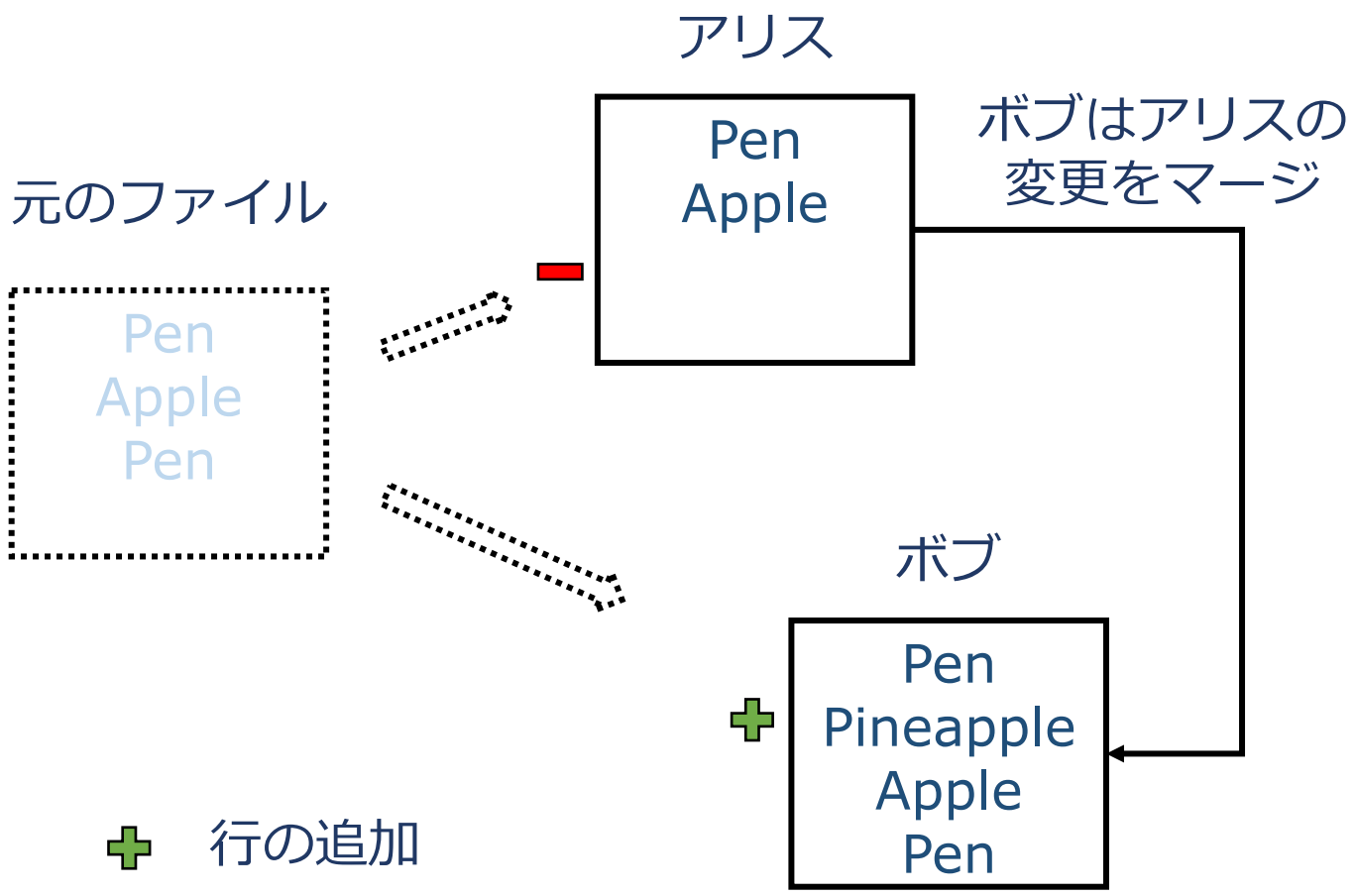


devをmainにマージ

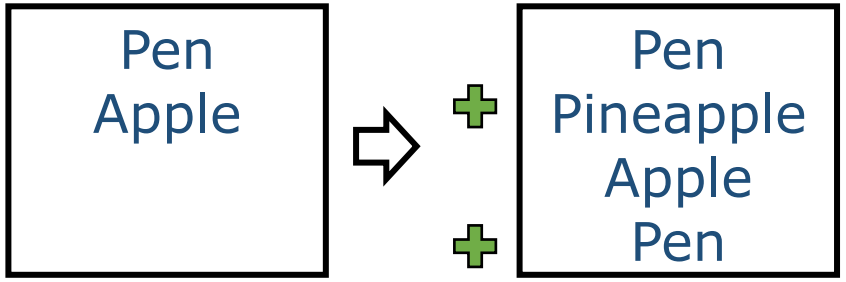
マージ方式：2-way merge (1/2)



マージ方式：2-way merge (2/2)

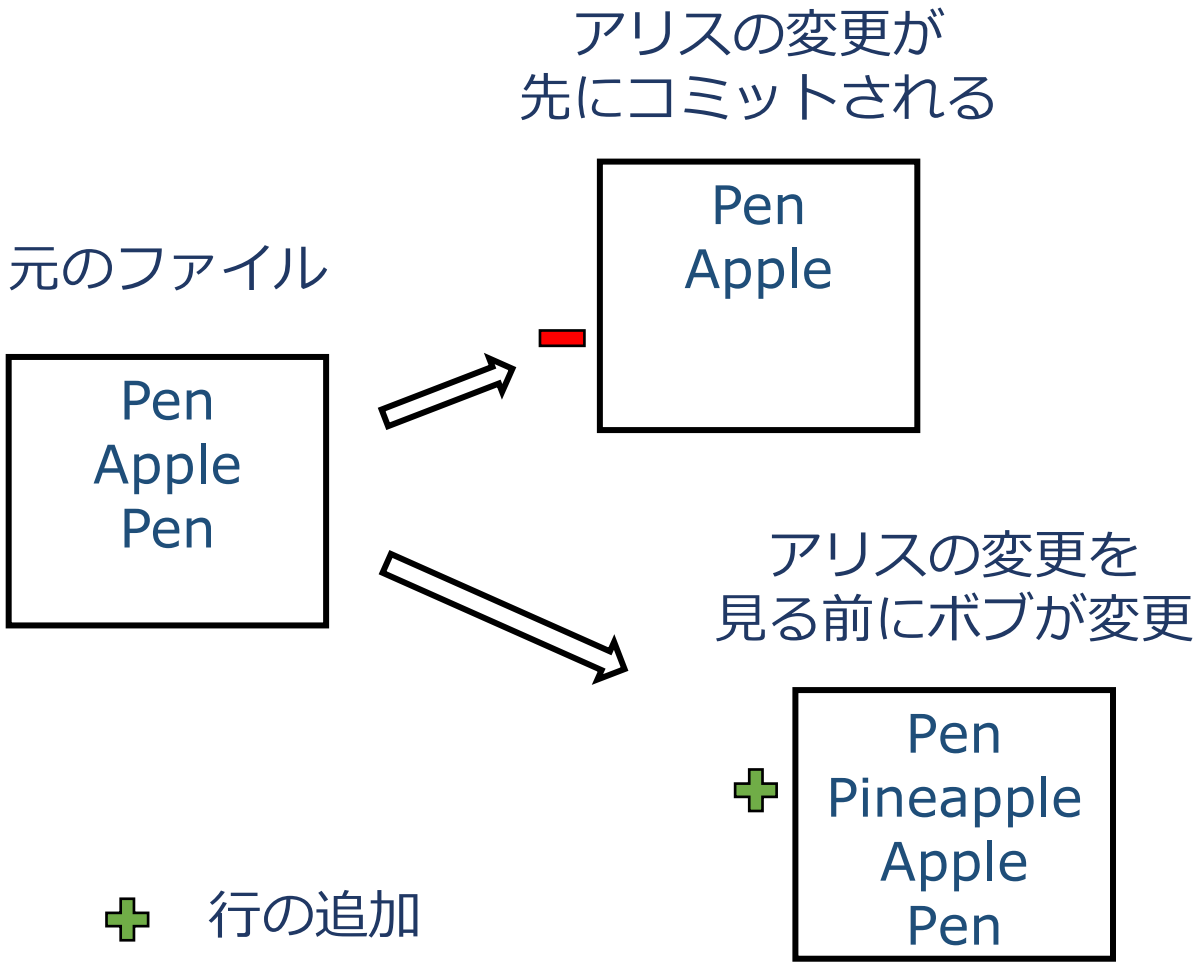


2-way merge：
2つのファイルだけを見て
差分を計算する



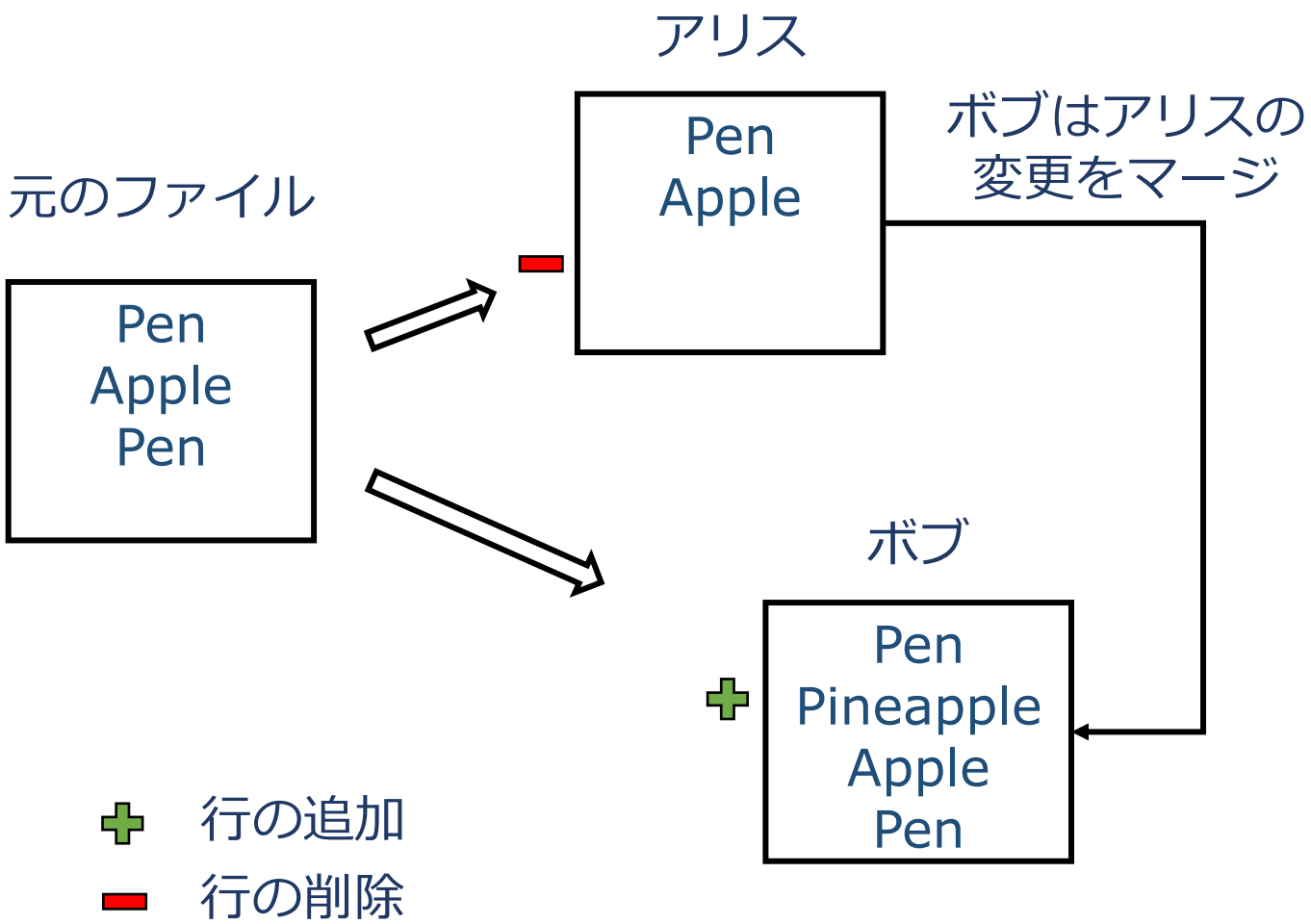
Pineapple，Pen
の2行が追加された？
→アリスの変更が無く
なってしまった…

マージ方式：3-way merge (1/2)



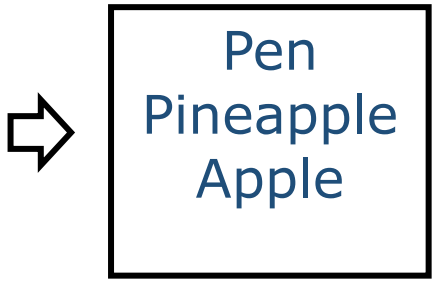
ここまでは
2-wayと3-way
で共通！

マージ方式：3-way merge (2/2)



3-way merge：
オリジナルのファイルも
考慮して差分を計算する

Penが削除され,
Pineappleが追加された
のだから…



マージ方式に関する注意

- ここまでに説明したのは大雑把な概念だけ
- 実際のマージアルゴリズムはもっと複雑

- 参考：
  「Gitの3-way mergeとは具体的にどのようなアルゴリズムですか？」
  https://ja.stackoverflow.com/q/52019

競合（Conflict）

- 同じ個所を同時並行で変更すること
- どちらの変更を採用すべきか自動決定できない

```
uchan@workstation:~/workspace/myproj$ git merge br-a
Auto-merging foo
CONFLICT (content): Merge conflict in foo
Automatic merge failed; fix conflicts and then commit the result.
uchan@workstation:~/workspace/myproj$
```

マージが
中断した様子

- 競合の解決
  - git statusで競合するファイルを確認し，
  - テキストエディタで競合を修正して保存し，
  - git add & git commit

競合時のファイル内容

```
git checkout main
git merge br-a
```

```
foobar
<<<<<<< HEAD
2nd line
3rd line
=======
this is a pen
>>>>>>> br-a
```

競合時の実際の
ファイル内容

```
<<<<<<< HEAD
```

HEADの内容

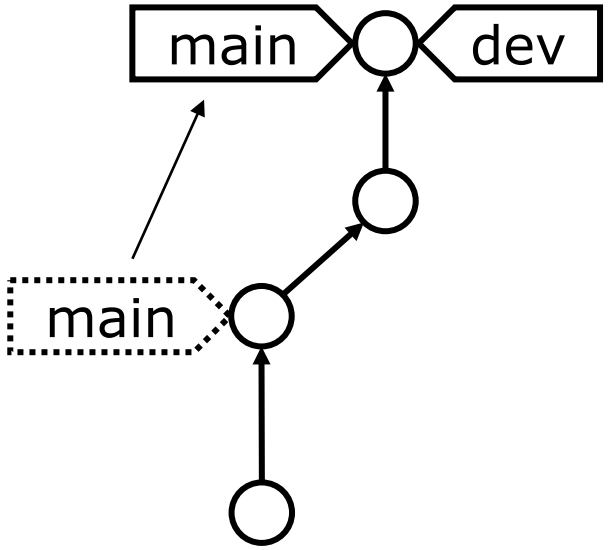
```
=======
```

マージしようとした内容

```
>>>>>>> br-a
```

Fast-forwardマージ

- Fast-forward：単にポインタを動かすだけで事足りる状況
- 敢えてマージコミットを作る：git merge **--no-ff**



Fast-forwardマージ

```
git checkout main
git merge --no-ff br-a
```

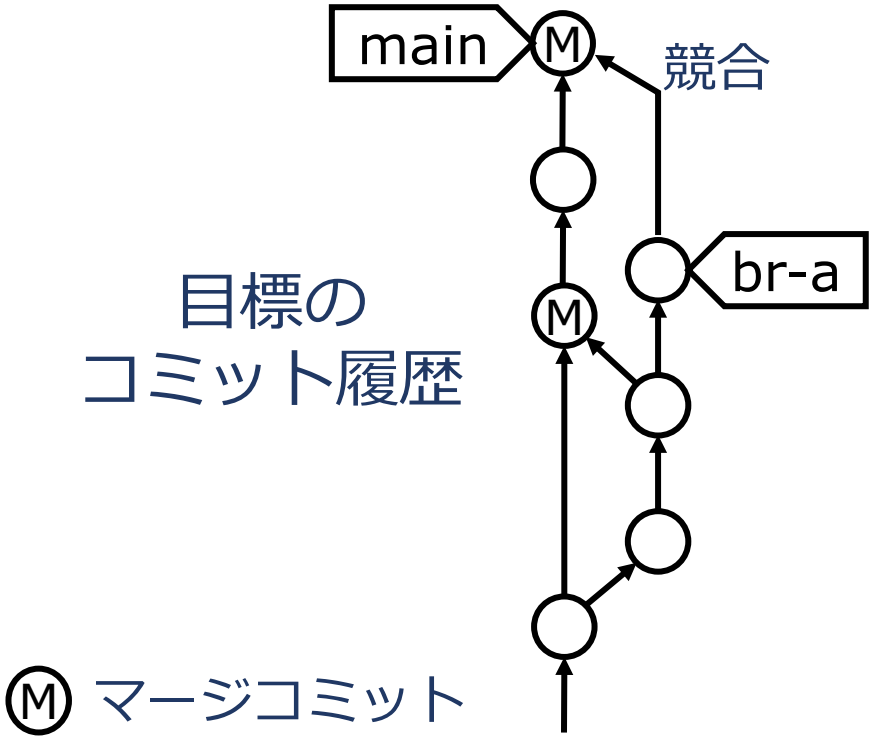
```
uchan@workstation:~/workspace/myproj$ git log --graph --all --oneline
*   8d38592 (HEAD -> master) Merge branch 'br-a'
|\
| * fea9950 (br-a) append lyrics
| * e40d0cc (tag: tag-a) add bar file
|/
* e526eb9 (br-b) add a.c
* 8747dd0 2nd commit
* 5e7a74b initial commit
uchan@workstation:~/workspace/myproj$
```

演習6

br-aをmainにマージせよ。
br-aとmainに競合するコミット
を加え，さらにマージせよ。

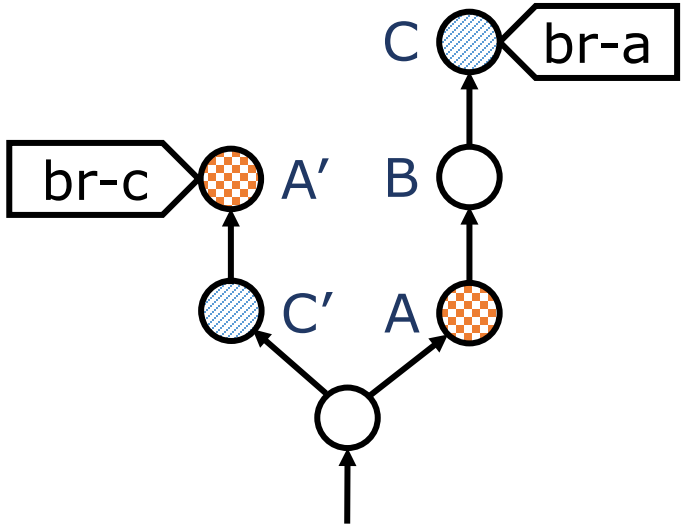
- マージは
  git checkout main
  git merge --no-ff br-a

- 制限時間10分



git cherry-pick

- 任意のコミットを取ってくる
- Cherry picking
  =サクランボの熟した果実だけを選別すること
- →いいとこどり，つまみぐい
- git cherry-pick COMMIT_ID

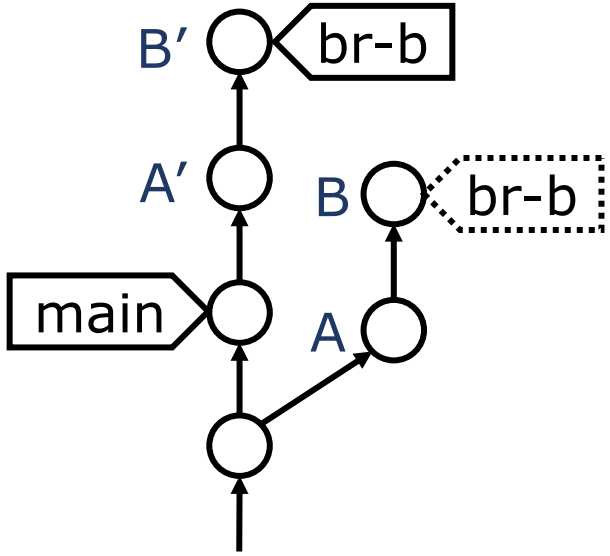


2つのコミットを
cherry-pick

リベース

- ベースを付け替える操作

- ブランチで作業中，他の人により mainが更新されてしまった！
- git checkout br-b
  git rebase main
- →あたかも最新のmainからブランチしたかのようになる



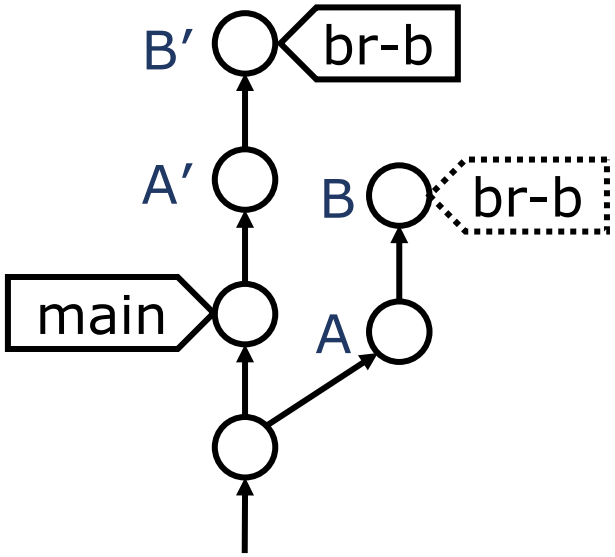
br-cをmainにリベース

演習7

br-bとmainにいくつかコミットし，
br-bをmainにリベースせよ

- git checkout br-b
  git rebase main

- 制限時間9分



目標の
コミット履歴

歴史を修正する

- git reflog
  ブランチが過去にどのコミットを指していたかを見る
- git rebase -i 起点となるコミット
  対話的リベース
  - コミット順を並び替えたり
  - コミットメッセージの変更を指示したり
- git commit --amend
  コミットの修正
  - 変更の追加・削除
  - コミットメッセージ編集

git rebase -i

```
git rebase -i tag-a
```

行を入れ替えるとコミット順が入れ替わる

```
pick fea9950 append lyrics
pick e1bff92 add 3rd line to foo
pick 1925598 edit 2nd line

# Rebase e40d0cc..65aa322 onto e40d0cc (3 commands)
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard
# x, exec = run command (the rest of th
# d, drop = remove commit
#
# These lines can be re-ordered; they a
```

pick
コミットを採用

edit
コミットを採用するが，修正のために中断する

drop
コミットを採用しない

エディタを変えたいときは
EDITOR環境変数を設定

```
$ export EDITOR=vim
```

git commit --amend

間違った内容でコミット

```
uchan@workstation:~/workspace/myproj$ git log --oneline
37b14dc (HEAD -> master) append 4th line
de3acd6 Merge branch 'br-a'
1925598 (br-a) edit 2nd line
```

git commit --amendは
- コミットメッセージ
- コミット内容

どちらも修正できる

（新しいコミットにまるっと置き換わる）

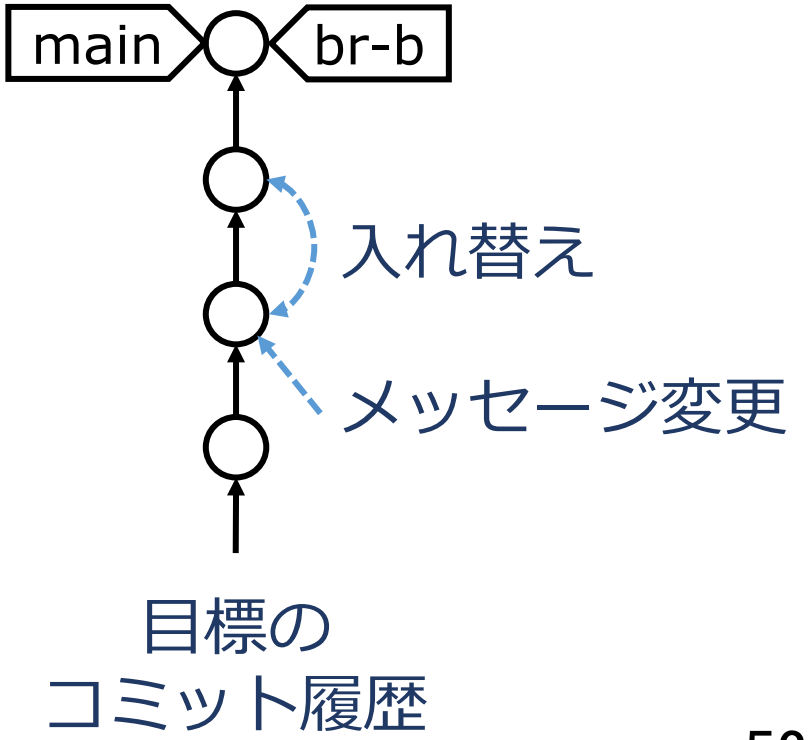
```
uchan@workstation:~/workspace/myproj$ git commit --amend
[master 037d196] append 5th line
 Date: Tue Oct 20 09:37:26 2020 +0900
 1 file changed, 2 insertions(+)
uchan@workstation:~/workspace/myproj$ git log --oneline
037d196 (HEAD -> master) append 5th line
de3acd6 Merge branch 'br-a'
1925598 (br-a) edit 2nd line
```

演習8

“git rebase -i”でコミット順と
コミットメッセージを修正してみよ

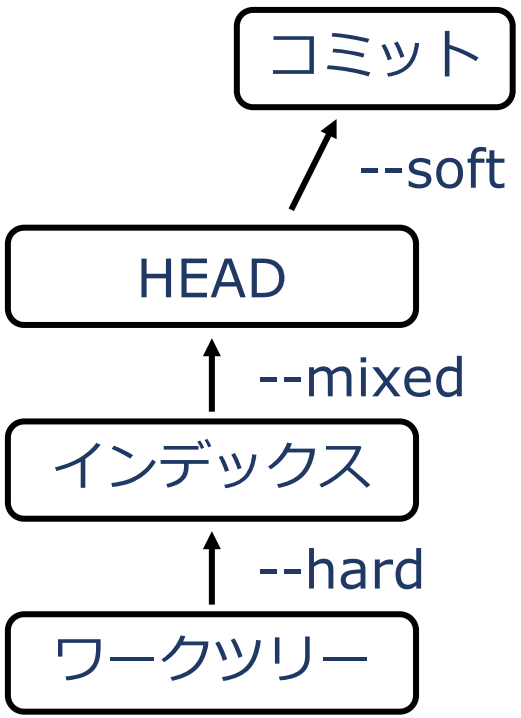
- git rebase -i …
  git commit --amend

- 制限時間9分



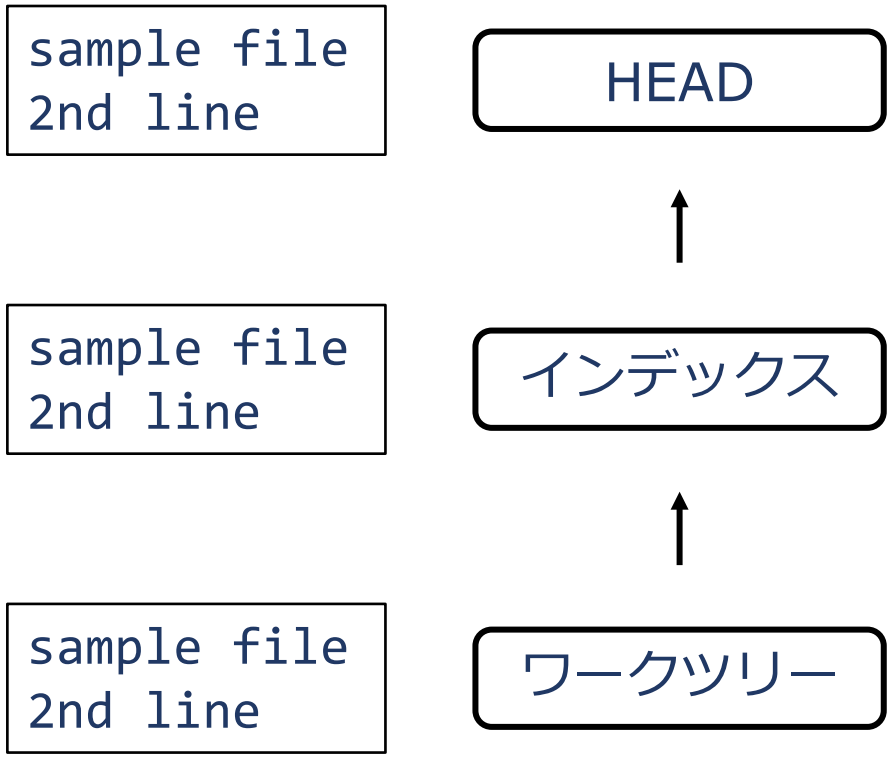
状態のリセット

- git reset
- --soft: HEADを指定したコミットにリセット
  - インデックスとワークツリーは無変化
- --mixed: インデックスをリセット
  - git resetのデフォルト動作
- --hard: インデックスとワークツリーをリセット
  - コミットしてない変更が消える



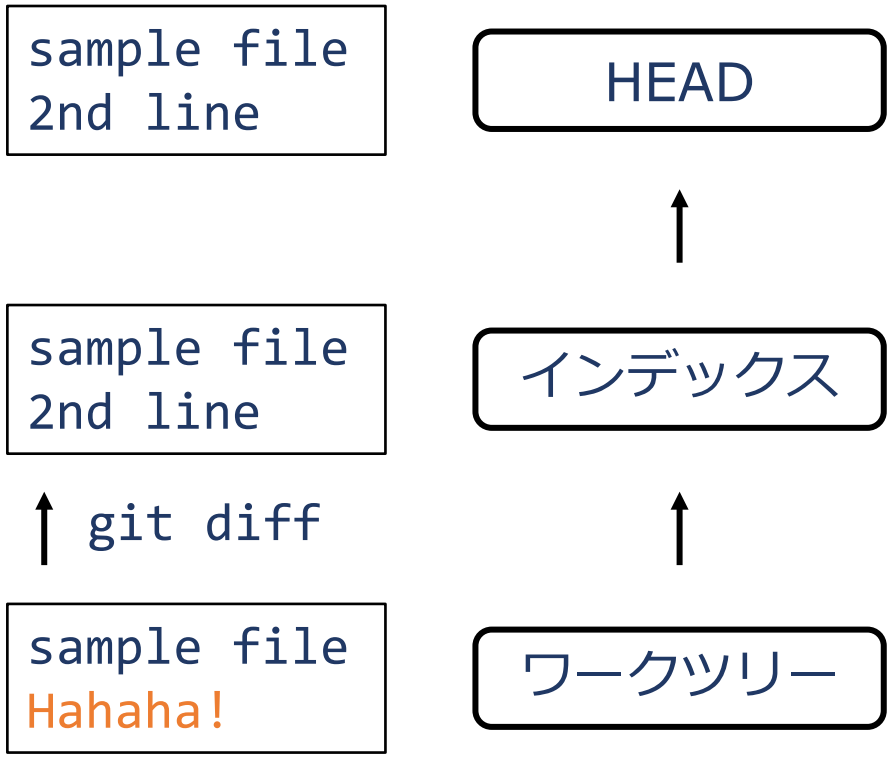
状態遷移ツアー 1/5

- 初期状態
- HEAD
  =インデックス
  =ワークツリー



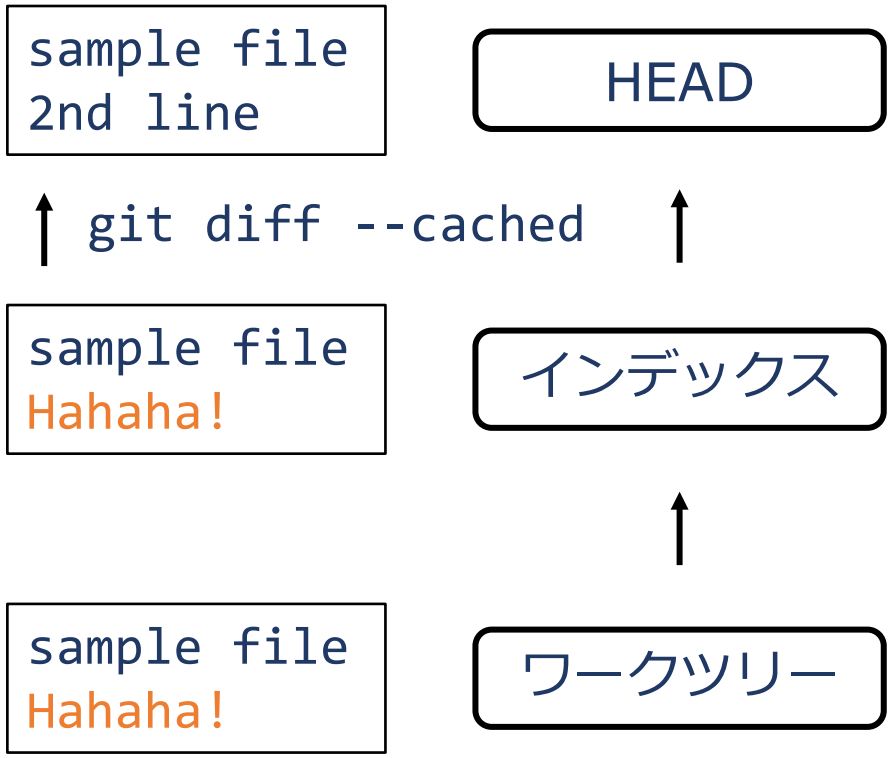
状態遷移ツアー 2/5

- 2行目を変更
- ワークツリーだけ変化
- Git diffは差分有
- Git diff --cached は差分無



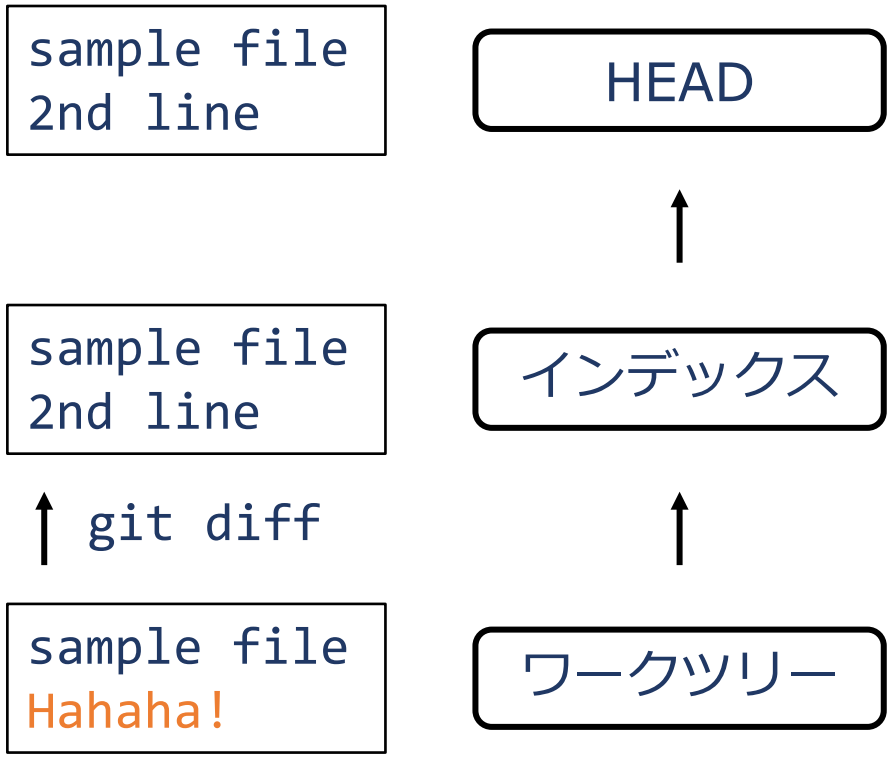
状態遷移ツアー 3/5

- git add sample-file
- インデックス
  =ワークツリー
- Git diffは差分無
- Git diff --cached
  は差分有



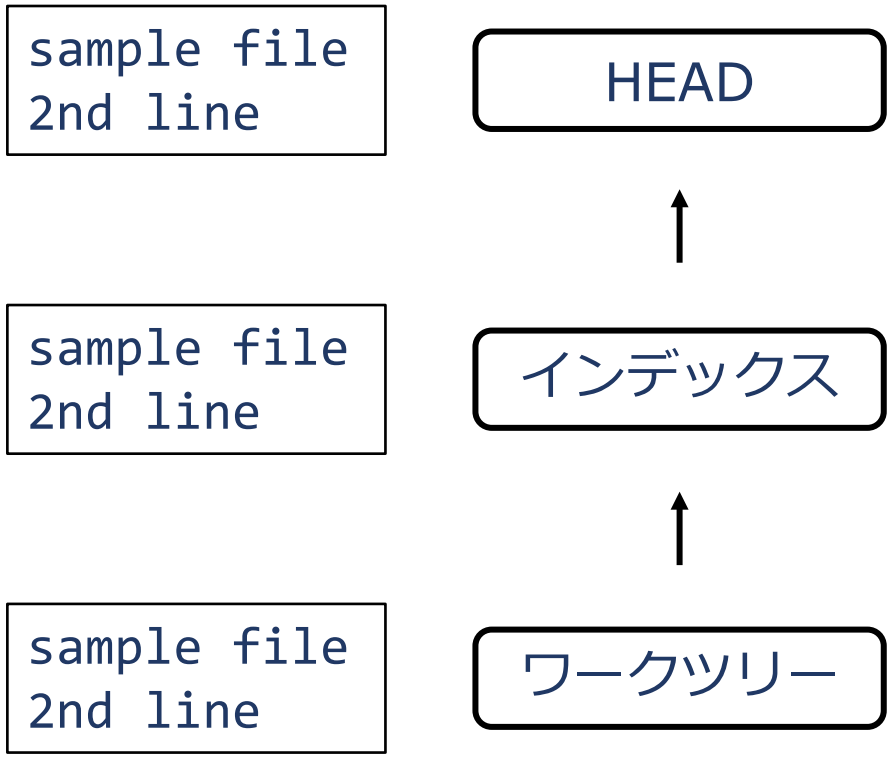
状態遷移ツアー 4/5

- git reset --mixed
- インデックスがリセットされる
- インデックス≠ワークツリー



状態遷移ツアー 5/5

- git reset --hard
- ワークツリーがリセットされる
- インデックス=ワークツリー



git stash

- コミットしたくないけど無くなると困る変更
- 一時的に変更を隠しておく
- Stash=隠し場所

- git stash save
- git stash pop
  - Stash領域はスタック構造
- git stash list

演習9

ファイルを変更してgit addし,
変更をなくさずにgit addを取り消せ

- git add foo
  git ???
- 制限時間4分

コミット粒度とメッセージ

- コミットをどんな単位でやるのが良いか？
- 単一の関心事項のみを含むようにする
  - 一言でそのコミットを説明できるようにする

- メッセージの良し悪し
- コミット内容を簡潔に包括する
- 現在形で書く
  - “This commit fixes a bug.”

Fix a bug about …

Add a feature to …

演習10

GitHubで良い/悪い
コミットメッセージの例を探せ

- 制限時間15分

演習11

Gitの操作を練習サイトで練習しよう

- 練習サイト
  https://learngitbranching.js.org/

- 制限時間：授業終了まで

レポート提出のトピック名

- 今回はTOPIC=git

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）

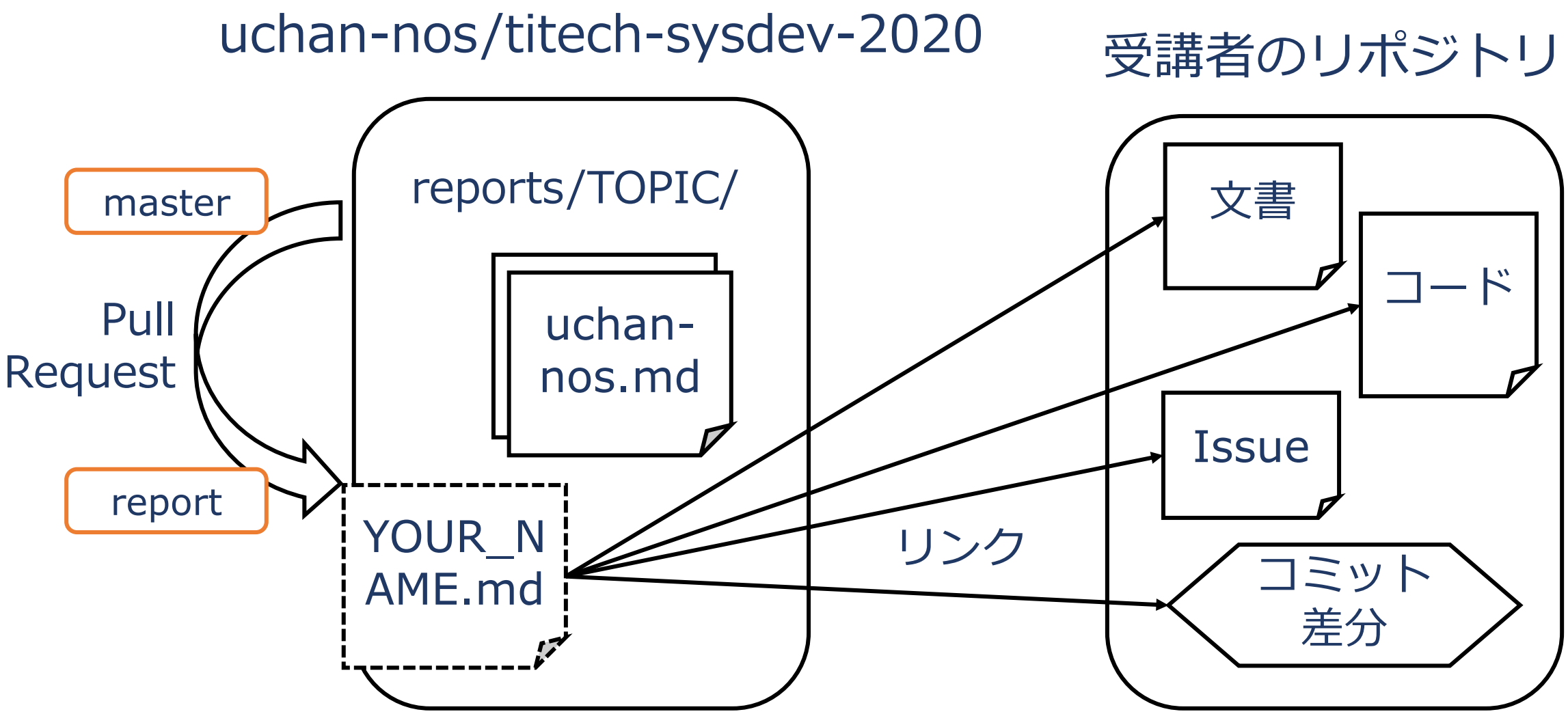
レポートの送り方

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

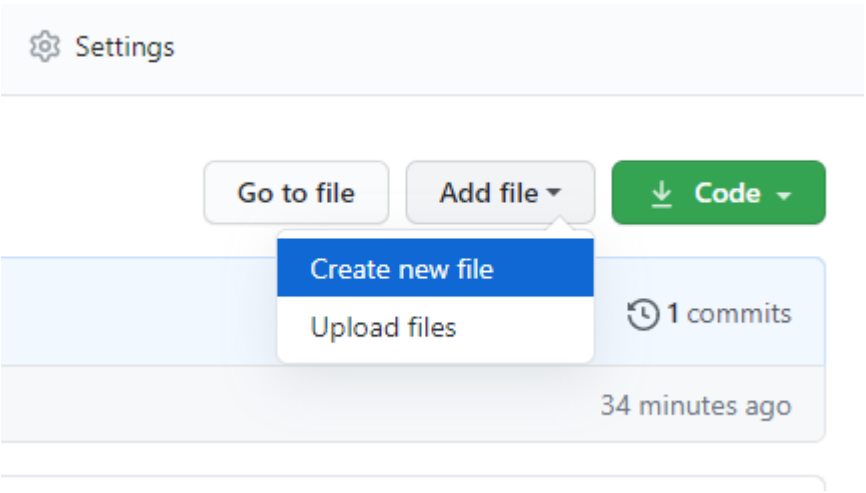
レポート

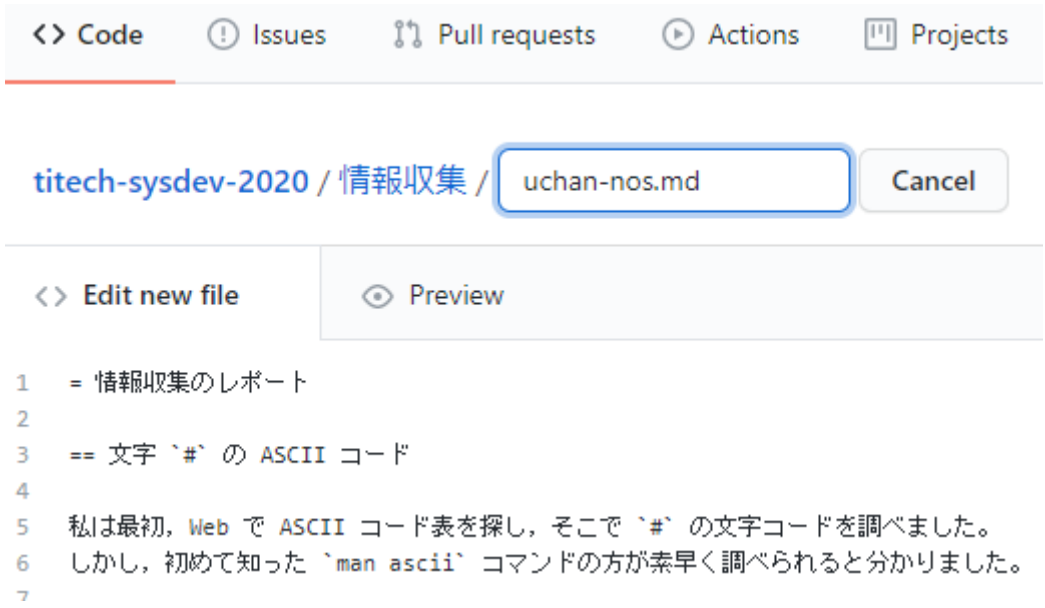
- reports/TOPIC/YOUR_NAME.md
- このファイルに課題への回答を記載する
- 必要なら以下のものを含める
  - Issueへのリンク
  - コミット差分へのリンク
    https://github.com/HOGE/REPO/compare/COMMIT1...COMMIT2
  - その他
- 要するに，成績評価に必要な情報をYOUR_NAME.md自体に記載するか，そこから辿れるようにする

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

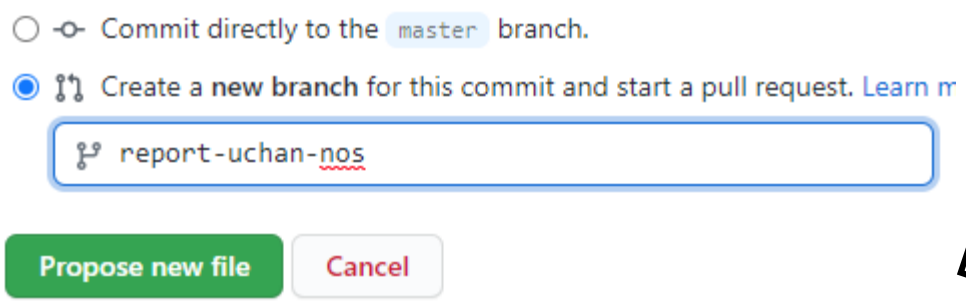




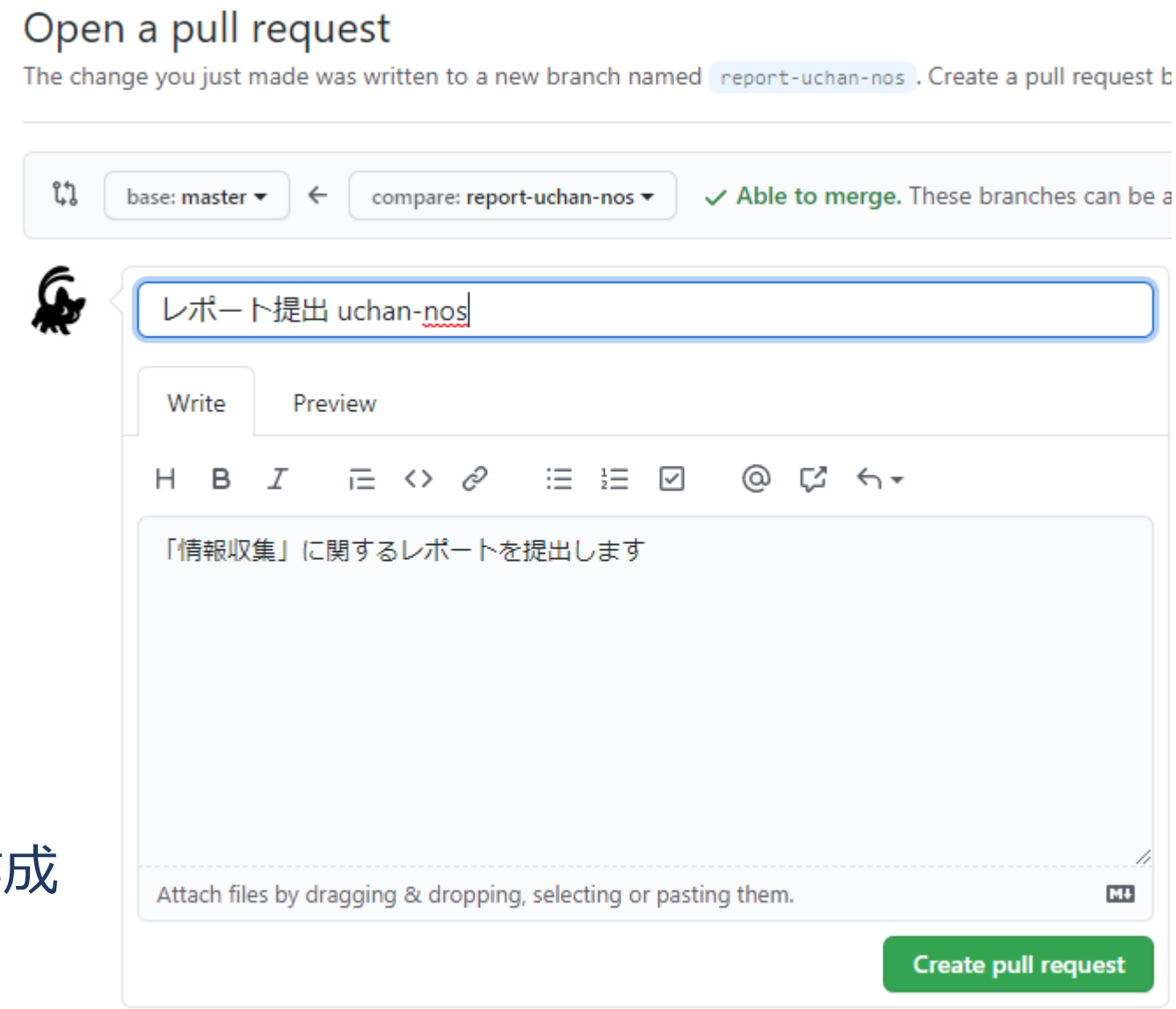
ファイルを新規作成

YOUR_NAME.mdの内容を記述

レポートの送り方 2/2

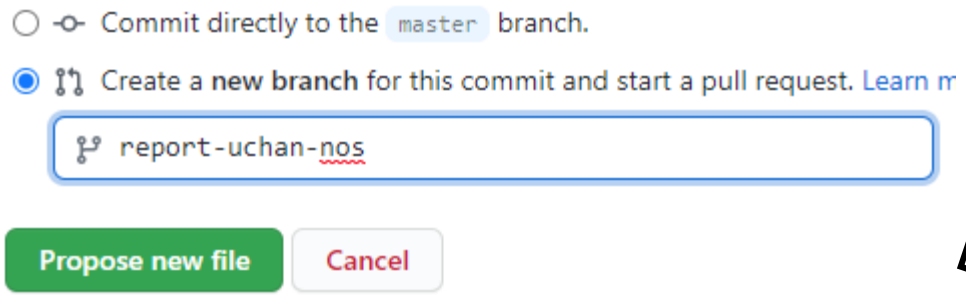


新規ブランチにコミット

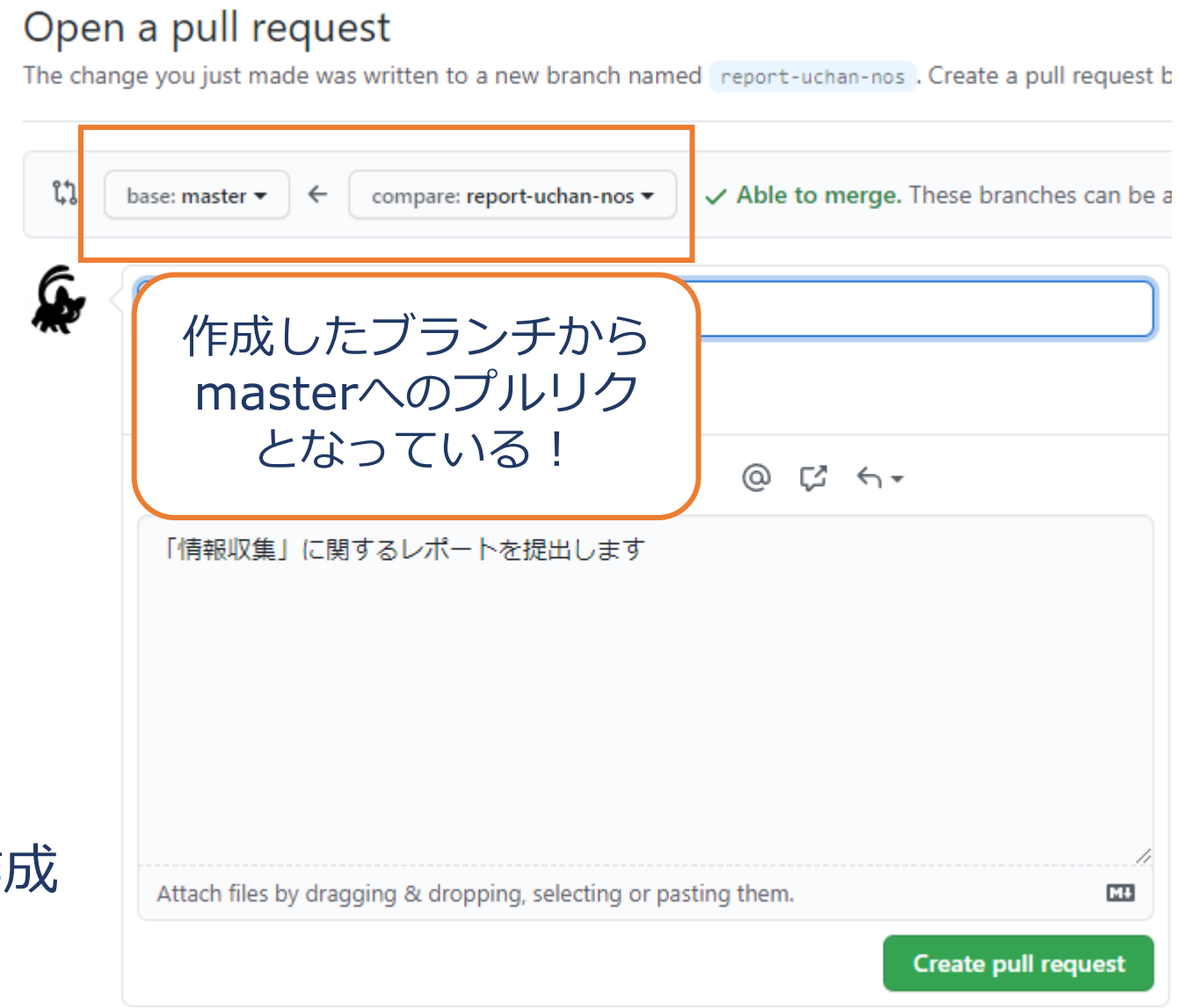


プルリクを作成

レポートの送り方 2/2



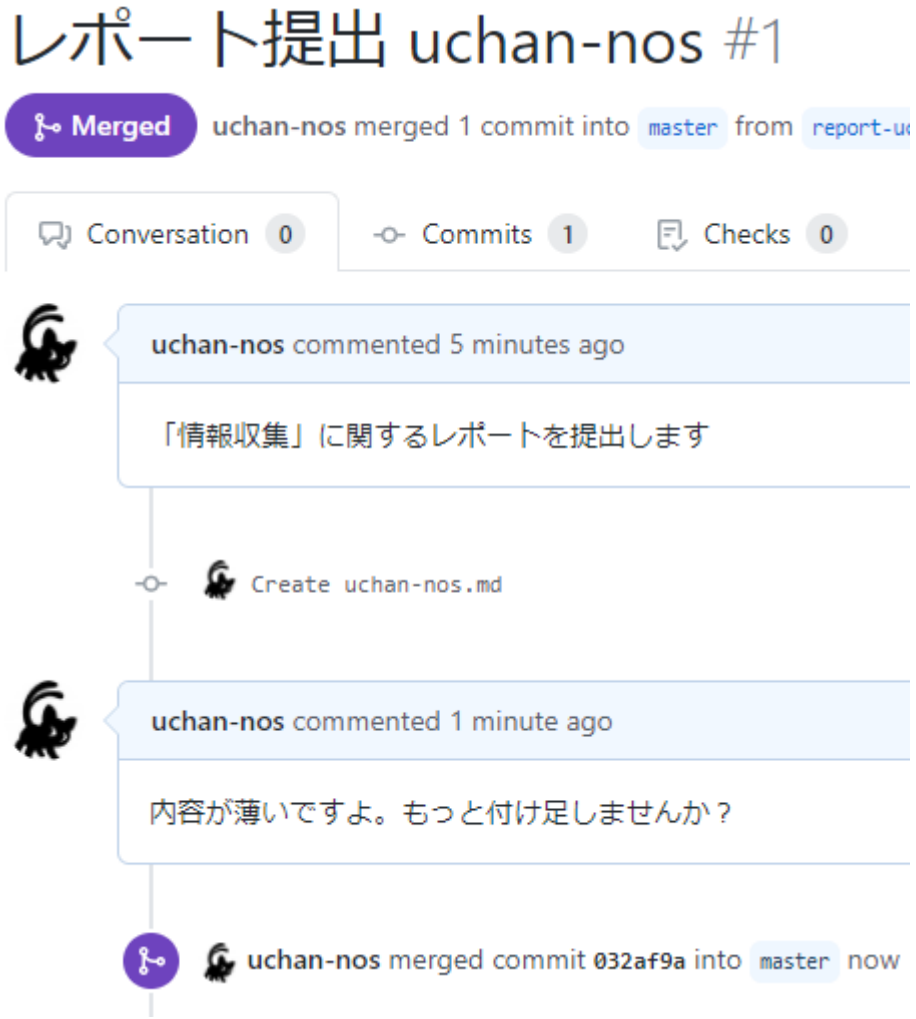
新規ブランチにコミット



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



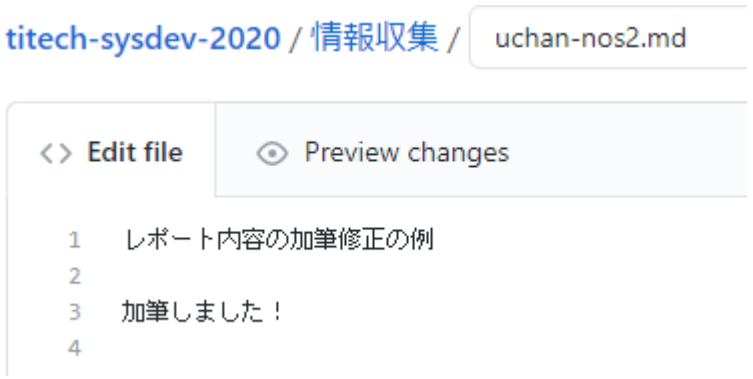
プルリクにコメントが付くことも

Merged：レポート受理済み

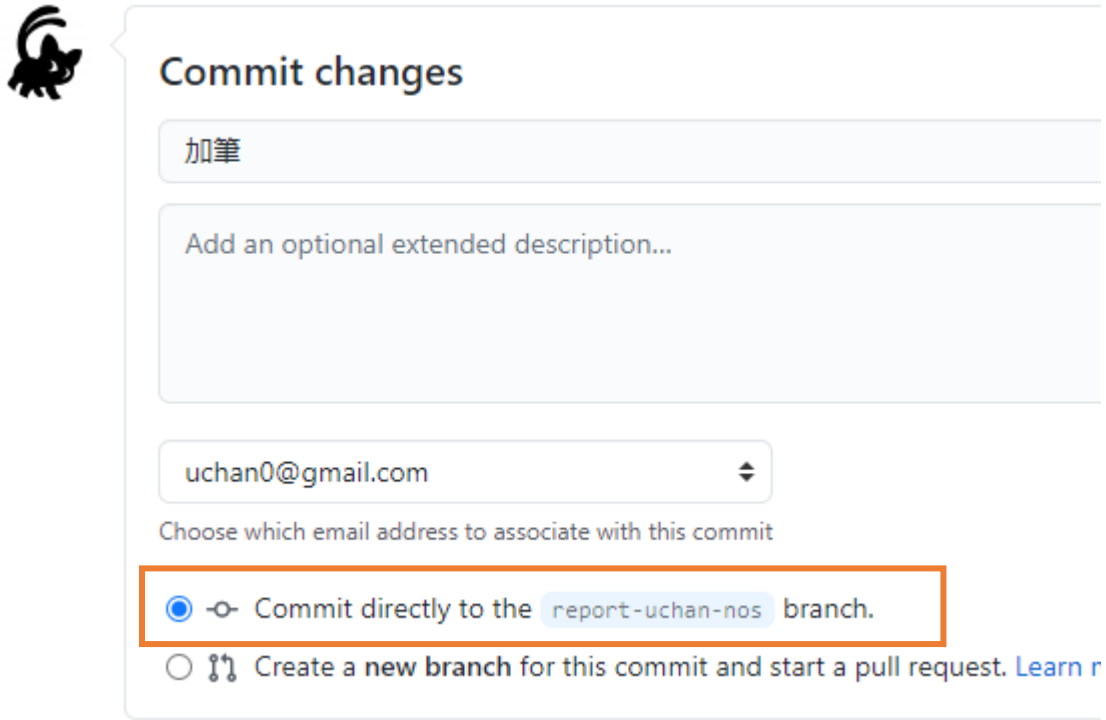
レポートの更新 1/2

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10/30（金）14:20から
- 「バグトラッキング」と
  「GitHub & Pull Request」をやります