クレジット：
UTokyo Online Education Education コンピュータシステム概論 2018 小林克志

コンピュータシステム概論
第2回

小林克志

- 事務連絡
- **演習 : GitHubClassroom へのアクセス**
- ソフトウェア構成管理
- バージョン管理
- **Git について**
- **演習 : URL 取得と git clone**
- 演習 : jupyter-notebook を使う
- **演習 Git : ステージング/コミット/Push**
- 課題 : markdown.ipynb

環境整備

先週出席していない受講生は以下をおこなうこと

- Google Classroom への登録(**)
- GitHub, GitHub Classroom への登録(**)
- Python 環境の整備
  Anaconda パッケージを推奨

(**) だけ説明します、残りは先週の資料を参考にすること

- ☑事務連絡
- **演習：GitHubClassroom へのアクセス**
- ソフトウェア構成管理
- バージョン管理
- Git について
- **演習：URL 取得と git clone**
- 演習：jupyter-notebook を使う
- **演習 Git：ステージング/コミット/Push**
- 課題：markdown.ipynb

GitHub Classroom への登録

- **GitHub の登録が終わったら…**

1.講義ページのお知らせページ
『第1回の課題でアクセスする GitHub Classroom の URL』
のリンクをクリック

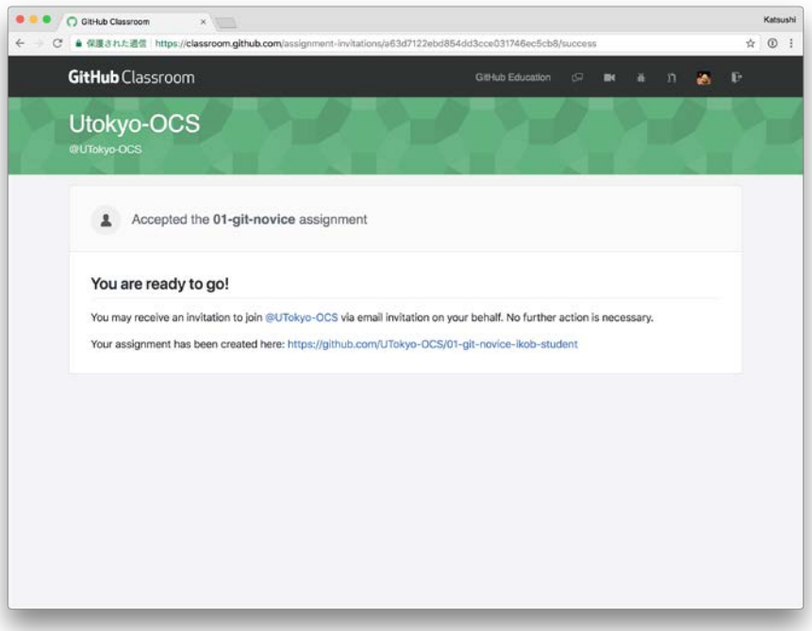
2.課題を Accept するかどうか聞かれるので、Accept する（右上）

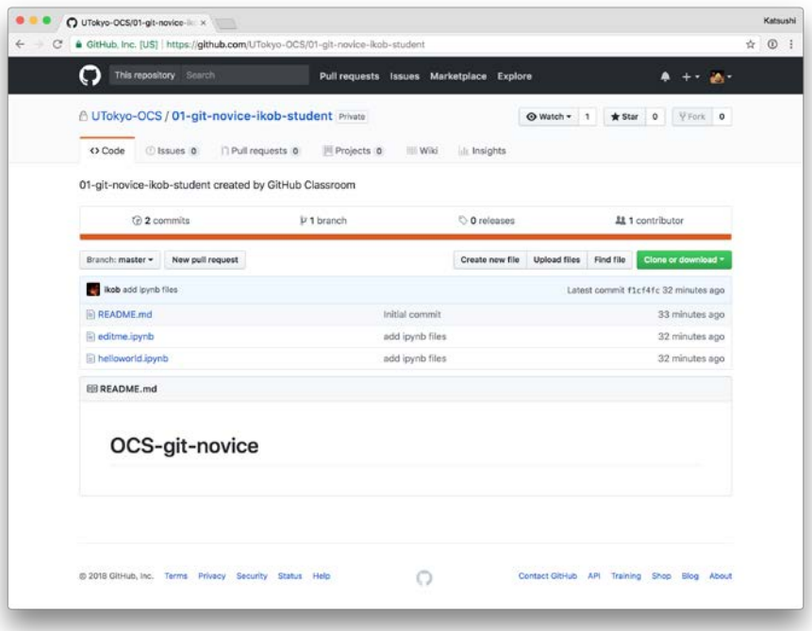
3.“Your assignment has been created here: ”以降にアクセスしてみる

4.自身の GitHub レポジトリにアクセスできる。（右下）

5.メールが飛ぶので Accept するようにしてください。

以降の操作は次回に説明します。





ソフトウェア構成管理：背景

著作権の都合により
ここに挿入されていた画像を削除しました

Hiro Yoshioka Twitter
https://twitter.com/hyoshiok/status/539588046107049984?lang=ja

ソフトウェア構成管理：背景（続き）

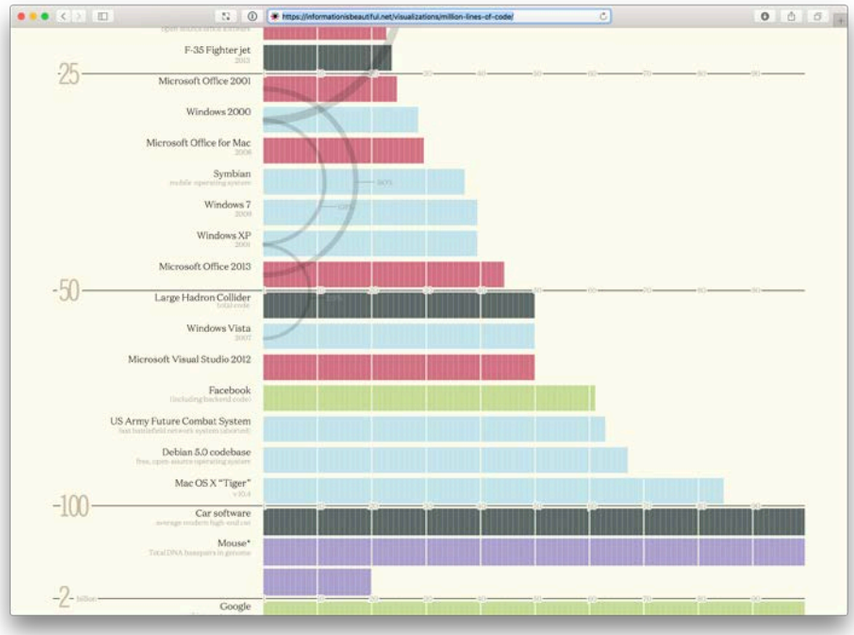
出典：Wikimedia Commons

- ソフトウェアの肥大化
  Linux OS カーネルは 8 年で 3 倍
  F35 は 20M 行、自動車 100M 行

- ソースコードは複数ファイルに分割して管理するのが一般的

- 複数の開発者が同時並行で変更
  変更箇所の衝突（コンフリクト）抑制・影響の軽減

- 更新に依存する部分だけ再ビルド
  ビルド時間の短縮



https://informationisbeautiful.net/visualizations/million-lines-of-code/

ソフトウェア構成管理:
ビルド支援ツールの例 make

- Unix/Linux コミュニティでは make (1976 –)が広く使われている

  “make ユーティリティの目的は、大きなプログラムの中の再コンパイルする必要がある部分を自動的に決定し、再コンパイルのためのコマンドを実行することである。…
  …まず makefile と呼ばれるファイルを書かなければならない。このファイルは、プログラムを 構成するファイル間の関係と各ファイルを更新するためのプログラムを記述し たものである。…”
  GNU make コマンドマニュアルより

- 実際には…
  - 開発者以外が Makefile を作成する機会はまずない
  - Makefile の生成も自動化されている、機種依存部分の設定など

ソフトウェア構成管理：
ビルド支援ツールの例 make(続き)

- Makefile の例、プログラム edit をビルド

```
edit : main.o kbd.o command.o display.o ¥
        insert.o search.o files.o utils.o
         cc –o edit main.o kbd.o command.o display.o ¥
                  insert.o search.o files.o utils.o

main.o : main.c defs.h
         cc –c main.c
kbd.o : kbd.c defs.h command.h
         cc –c kbd.c
command.o : command.c defs.h command.h
         cc –c command.c
display.o : display.c defs.h buffer.h
         cc –c display.c
insert.o : insert.c defs.h buffer.h
         cc –c insert.c
search.o : search.c defs.h buffer.h
         cc –c search.c
files.o : files.c defs.h buffer.h command.h
         cc –c files.c
utils.o : utils.c defs.h
         cc –c utils.c
clean :
         rm edit main.o kbd.o command.o display.o ¥
            insert.o search.o files.o utils.o
```

edit のオブジェクトファイル依存性を定義
edit の生成方法を定義

main.o オブジェクトファイルは main.c, defs.h に依存
main.c, defs.h が main.o より後に更新されていれば、
main.c から main.o を生成(コンパイル)する

オブジェクトファイルの片付け方法、以下を入力すると
実行される
$ make clean

書き換え,or 改竄,・文書管理 情報システムとも密接な関係

- “政府参考人「この電子決裁は特別の、….一元的な文書管理システム …. の中でこれはなされております。…書換えを行うと、その更新履歴だけではなくて、…書換え前のものも併せて保存をされており、更新履歴をたどるとそのことが確認できるという状況になってございます。
  なってございますと今御説明を申し上げておりますが、正直に申し上げると、今回の調査の過程でそのことを知って、そういうことが分かって御説明が申し上げられているわけですが…」”
  第196 回国会参議院・財政金融委員会 3月15日議事録より

- “「陸自、イラク日報を昨年３月確認」
  …当時の陸自研究本部…が昨年３月27日、南スーダン国連平和維持活動（PKO）の日報問題を巡る特別防衛監察の調査で発見した…”
  日経新聞 2018/4/5　朝刊より

財務省：Wikipedia より　CC BY-SA 3.0

Google 検索アプライアンス（販売終了）
https://enterprise.google.co.jp/intl/ja/search/products/gsa.html

著作権の都合により
ここに挿入されていた画像を削除しました

参議院インターネット審議中継
第196回国会
参議院予算委員会
2018年3月8日
審議時間約4時間45分
内、3時間44分23秒時点

http://www.webtv.sangiin.go.jp/webtv/index.php

VCS (Version Control System :バージョン管理システム) : 背景

- **複数バージョンソフトウェア**の開発・メンテナンスの要求:
  - 顧客毎に利用バージョンは異なる
    - 現象の再現には全てを用意しておく必要がある
  - 機能追加、修正、セキュリティ対策
    - 対応するのは全て or 特定バージョン？
- バージョン毎に別々の(膨大な)ソースコードの一貫した管理の必要性
- VCS の利用:
  変更履歴の保存、指定バージョン取得、差分表示、パッチ生成….

Windows 10 のバージョン履歴	提供日	サービスの終了
Windows 10 バージョン 1709	2017 年 10 月 17 日	2019 年 4 月 9 日**
Windows 10 バージョン 1703	2017 年 4 月 5 日	2018 年 10 月 9 日**
Windows 10 バージョン 1607	2016 年 8 月 2 日	2018 年 4 月 10 日**
Windows 10 バージョン 1511	2015 年 11 月 10 日	2017 年 10 月 10 日**
Windows 10 (2015 年 7 月にリリース、バージョン 1507)	2015 年 7 月 29 日	2017 年 5 月 9 日

Windows ライフサイクルのファクト シートより

VCS (Version Control System) : 歴史 (計算機環境の変化に依存)

- ローカル：共用計算機のファイルシステム上で管理
  - Revision Control System (RCS) 1982 –
    - ファイル単位のロックによる衝突抑制 ← 破綻
- クライアント・サーバ：レポジトリと作業ディレクトリの分離
  - Concurrent Version System (CVS) 1990 –
    - レポジトリ（変更履歴を含むデータ）をサーバ(S)に、作業ディレクトリをクライアント(C)に
    - 事前ロックから事後のマージ：衝突（コンフリクト）は自動あるいは開発者による修正
  - Apache Subversion (SVN) 2000 –
    - ファイル単位ではなく、ソフトウェア全体で一貫したバージョン管理
- 分散バージョン管理
  - Git 2005 –
    - レポジトリのコピーを全ての作業者が持つ
    - 差分ではなくファイルを持つ

VCS (Version Control System) : 応用

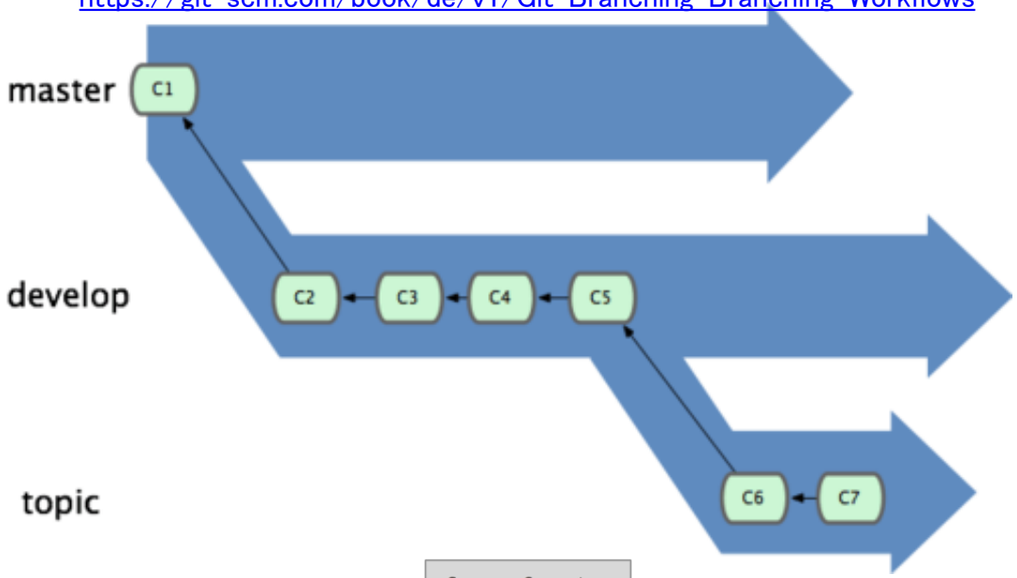
- ソフトウェア開発だけではなく通常の文書管理へも適用されている:
  - TourtoiseSVN, TourtoiseGit
    - GUI ベースのユーザインターフェース
    - MS Office suite の集中レポジトリによる管理と差分可視化
    - オンプレ(On-Premise)との相性の良さ

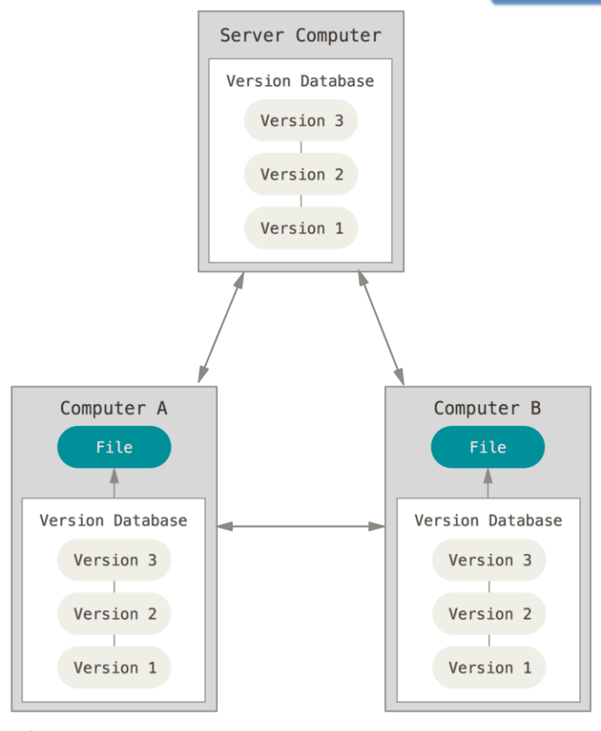
➡クラウドベースの Office suite サービスと競合

Git : a version control system(1)

- VCS の基本
  - **複数バージョンの同時開発**：ブランチ
  - **一貫したバージョン管理**：
    ハッシュを用いたコミット単位のリビジョン
  - 衝突解決：事後にマージ（ユーザ介入）、ロックなし
- Git の特徴
  - バージョンごとの完全スナップショット、
    前との差分ではない
  - 分散 VCS：
    - リモートレポジトリ(右図 Server)、作業者（Computer A, B）が履歴を含めた完全コピーをローカルに持つ
  - ステージング(add)とコミット(commit)の分離

『Pro Git』 1st Edition, 2009 CC BY-NC-SA 3.0
https://git-scm.com/book/de/v1/Git-Branching-Branching-Workflows





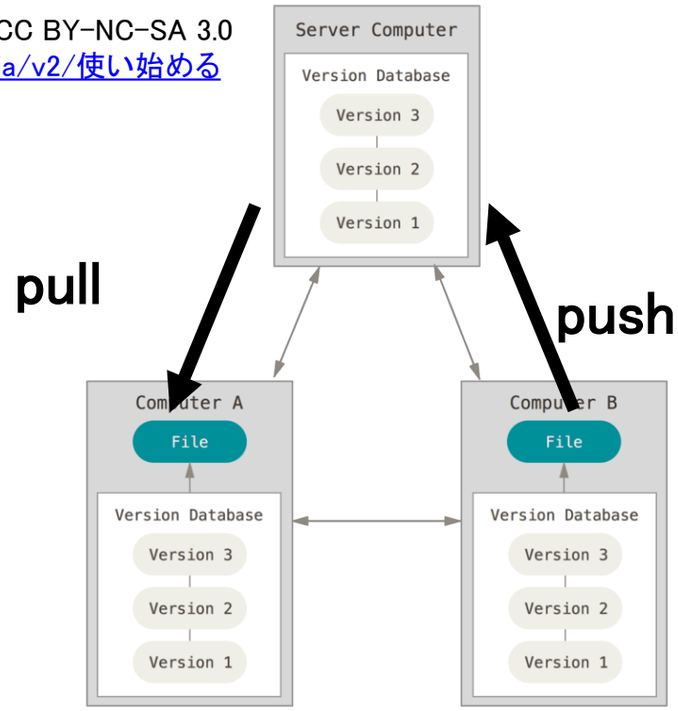
『Pro Git』 2nd Edition, 2014 CC BY-NC-SA 3.0
https://git-scm.com/book/ja/v2/使い始める-バージョン管理に関して

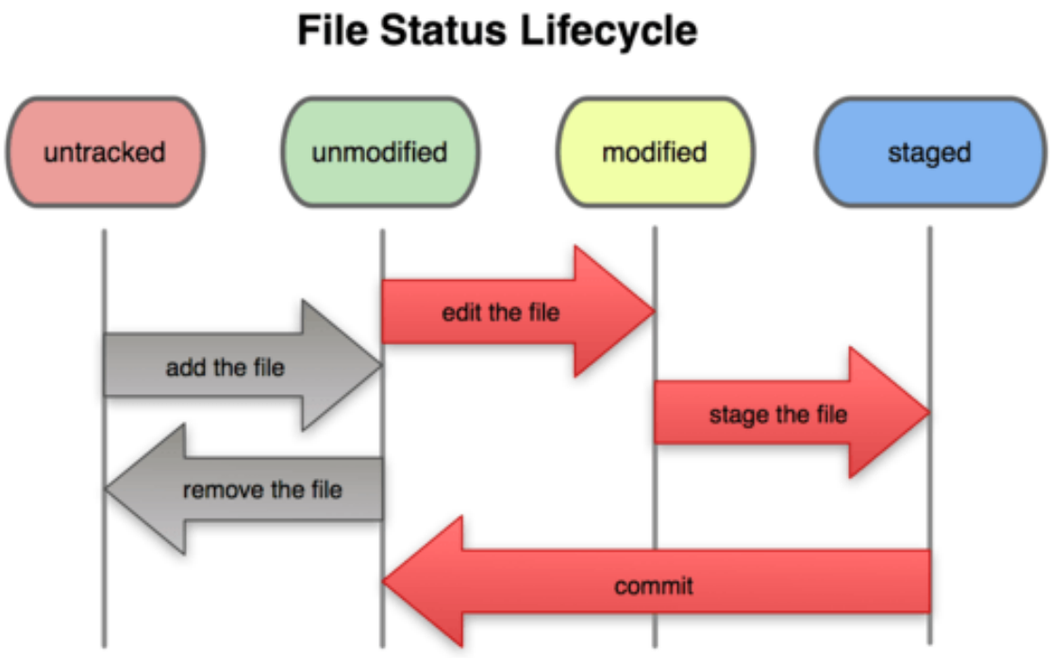
Git 参考書：https://git-scm.com/book/ja/v2/

Git : a version control system(2)

『Pro Git』 2nd Edition, 2014 CC BY-NC-SA 3.0
https://git-scm.com/book/ja/v2/使い始める-バージョン管理に関して

- Computer (ローカル)での4つの状態
  - 管理対象外
  - コミット済み(U) : pull, clone 後
  - 修正済み(E)
  - ステージ済み(S)
- 遷移トリガ
  - C -> E : checkout + 変更
  - E -> S : add
  - S -> C : commit
- Server (リモート)は1つの状態、コミット済み(C) のみ
  - データ移動 :
    - pull : Server -> Computer
    - push : Computer -> Server





『Pro Git』 1st Edition, 2009 CC BY-NC-SA 3.0
https://git-scm.com/book/ja/v1/Git-の基本-変更内容のリポジトリへの記録

git basic commands

1. Crete repository:

```
$ cd <specific directory>
$ git init
```

2. Clone remote repository:

```
$ git clone <URL>
```

3. Import server updates:

```
$ git pull
```

4. Stage files:

```
$ git add file1, file2, …
```

5. Commit changes:

```
$ git commit -m 'message'
```

6. Update server repository:

```
$ git push
```

7. Get current status:

```
$ git status
```

8. Get commit logs:

```
$ git log
```

9. Show local branches:

```
$ git branch
* alice-work
  master
```

10.Show all branches:

```
$ git branch -a
* alice-work
  master
  remotes/origin/HEAD -> origin/master
  remotes/origin/alice-work
  remotes/origin/ikob
  remotes/origin/master
$
```

11.Show tracked repositories:

```
$ git remote -v
origin<URL> (fetch)
origin<URL> (push)
$
```

If you want to:

1. Undo add:

$ git reset HEAD
— or —
$ git reset

2. Undo the last commit permanently:

$ git reset —-hard HEAD^

3. Cancel specific commit:

$ git reset —-hard <commit>

4. Back (specific file) to before last commits:

$ git checkout HEAD^ <filename>

5. Show diffs:
    1. Between current working and the last commit (HEAD):

$ git diff

    2. Contents of the last commit:

$ git diff HEAD^ HEAD

    3. With the 3rd latest commit:

$ git diff HEAD~2
— or —
$ git diff HEAD^^

    4. Specify revision with (short) SHA-1:

$ git diff
97e91e24429c26449d30893653e3f26eaed3540e
— or —
$ git diff 97e91e2

Others, consult Pro Git or cheat sheet by GitHub:

https://git-scm.com/book/en/v2
https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf

https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf

GitHub GIT CHEAT SHEET

Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

INSTALL GIT

GitHub provides desktop clients that include a graphical user interface for the most common repository actions and an automatically updating command line edition of Git for advanced scenarios.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Git for All Platforms
http://git-scm.com

CONFIGURE TOOLING

Configure user information for all local repositories

```
$ git config --global user.name "[name]"
```
Sets the name you want attached to your commit transactions

```
$ git config --global user.email "[email address]"
```
Sets the email you want attached to your commit transactions

```
$ git config --global color.ui auto
```
Enables helpful colorization of command line output

CREATE REPOSITORIES

Start a new repository or obtain one from an existing URL

```
$ git init [project-name]
```
Creates a new local repository with the specified name

```
$ git clone [url]
```
Downloads a project and its entire version history

MAKE CHANGES

Review edits and craft a commit transaction

```
$ git status
```
Lists all new or modified files to be committed

```
$ git diff
```
Shows file differences not yet staged

```
$ git add [file]
```
Snapshots the file in preparation for versioning

```
$ git diff --staged
```
Shows file differences between staging and the last file version

```
$ git reset [file]
```
Unstages the file, but preserve its contents

```
$ git commit -m "[descriptive message]"
```
Records file snapshots permanently in version history

GROUP CHANGES

Name a series of commits and combine completed efforts

```
$ git branch
```
Lists all local branches in the current repository

```
$ git branch [branch-name]
```
Creates a new branch

```
$ git checkout [branch-name]
```
Switches to the specified branch and updates the working directory

```
$ git merge [branch]
```
Combines the specified branch's history into the current branch

```
$ git branch -d [branch-name]
```
Deletes the specified branch

GIT CHEAT SHEET

REFACTOR FILENAMES

Relocate and remove versioned files

```
$ git rm [file]
```
Deletes the file from the working directory and stages the deletion

```
$ git rm --cached [file]
```
Removes the file from version control but preserves the file locally

```
$ git mv [file-original] [file-renamed]
```
Changes the file name and prepares it for commit

SUPPRESS TRACKING

Exclude temporary files and paths

```
*.log
build/
temp-*
```
A text file named `.gitignore` suppresses accidental versioning of files and paths matching the specified patterns

```
$ git ls-files --other --ignored --exclude-standard
```
Lists all ignored files in this project

SAVE FRAGMENTS

Shelve and restore incomplete changes

```
$ git stash
```
Temporarily stores all modified tracked files

```
$ git stash pop
```
Restores the most recently stashed files

```
$ git stash list
```
Lists all stashed changesets

```
$ git stash drop
```
Discards the most recently stashed changeset

REVIEW HISTORY

Browse and inspect the evolution of project files

```
$ git log
```
Lists version history for the current branch

```
$ git log --follow [file]
```
Lists version history for a file, including renames

```
$ git diff [first-branch]...[second-branch]
```
Shows content differences between two branches

```
$ git show [commit]
```
Outputs metadata and content changes of the specified commit

REDO COMMITS

Erase mistakes and craft replacement history

```
$ git reset [commit]
```
Undoes all commits after `[commit]`, preserving changes locally

```
$ git reset --hard [commit]
```
Discards all history and changes back to the specified commit

SYNCHRONIZE CHANGES

Register a repository bookmark and exchange version history

```
$ git fetch [bookmark]
```
Downloads all history from the repository bookmark

```
$ git merge [bookmark]/[branch]
```
Combines bookmark's branch into current local branch

```
$ git push [alias] [branch]
```
Uploads all local branch commits to GitHub

```
$ git pull
```
Downloads bookmark history and incorporates changes

GitHub Training

Learn more about using GitHub and Git. Email the Training Team or visit our web site for learning event schedules and private class availability.

training@github.com
training.github.com

演習：GitHub レポジトリの複製
ターミナルの起動

1.ターミナルを起動する：

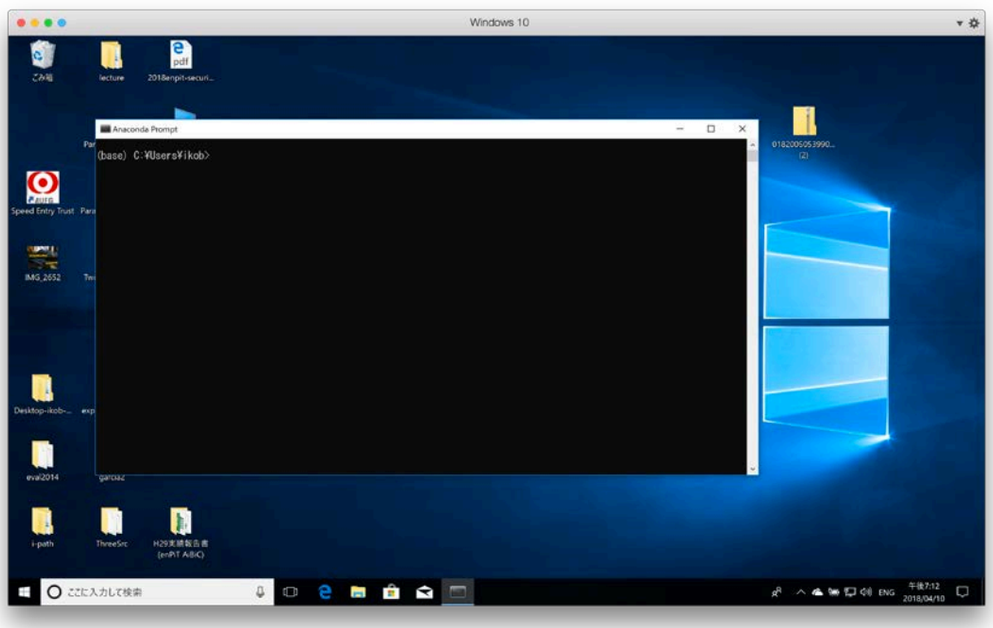
- Windows : Start -> Anaconda3 ->
  Anaconda Prompt
- Mac : /Application/Utilities/Terminal.app

2.講義用に作業ディレクトリを作成する。

```
$
$ mkdir OCS2018
$ cd OCS2018
$
```

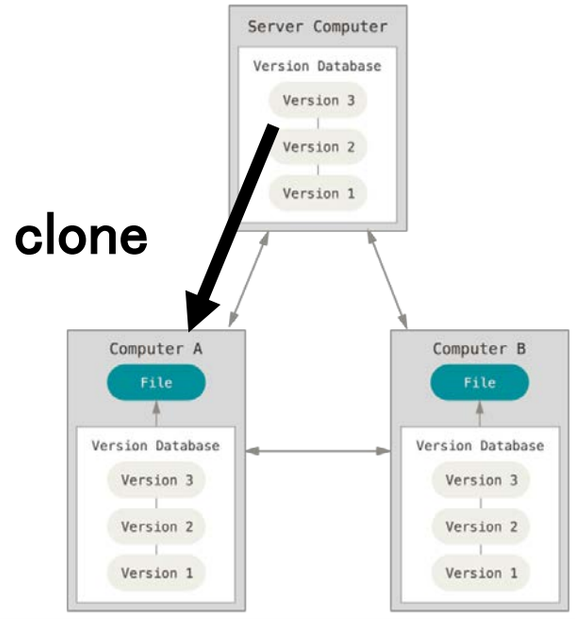
3.以下のコマンドで git が実行できることを確認する。

```
$
$ git ––version
git version 2.15.1 (Apple Git-101)
$
```

演習：GitHub レポジトリの複製
URL 取得と git clone

『Pro Git』 2nd Edition, 2014 CC BY-NC-SA 3.0
https://git-scm.com/book/ja/v2/使い始める-バージョン管理に関して



1.GitHub Classroom 登録で作られた GitHub レポジトリページに移動

2.“Clone or download” ボタンをクリック、URL を表示

3.URL をコピーする(右端のコピーボタンをクリック)

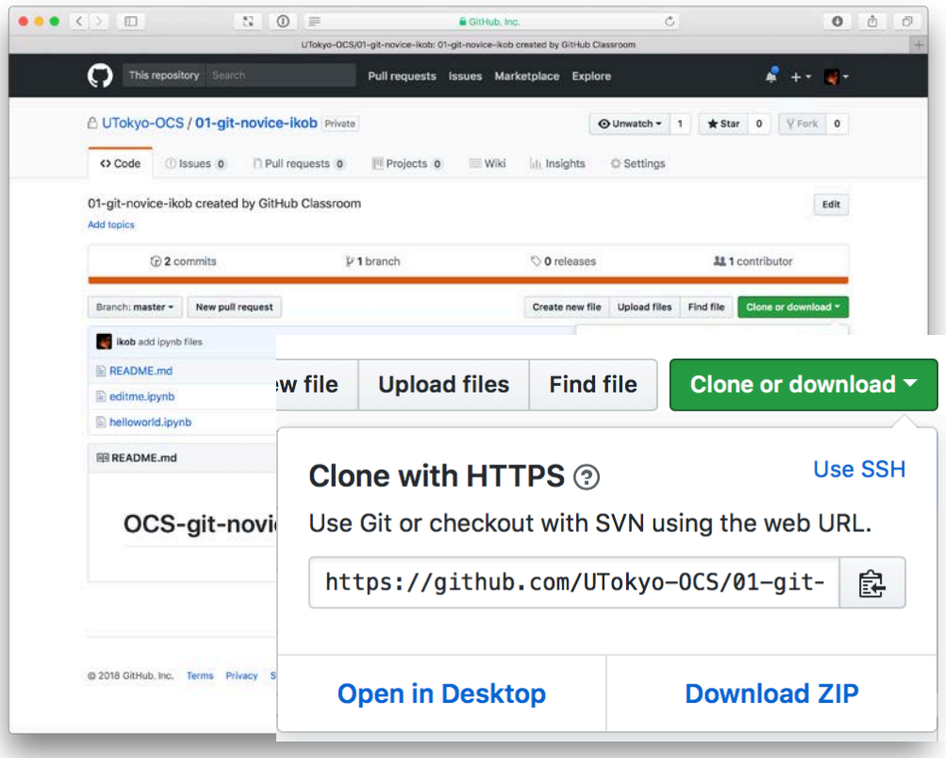
4.CLI で以下の git clone コマンドを実行する。途中で GitHub のユーザ名・パスワードを聞かれる:

```
$ git clone <コピーした URL をここにペーストする>
Cloning into ' 01-git-novice-ユーザ名' ...
git: 'credential-cache' is not a git command. See 'git --help'.
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 0), reused 7 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), done.
$
```

5.ディレクトリ”01-git-novice-ユーザ名”（以降作業ディレクトリ）が確認できれば複製は成功。

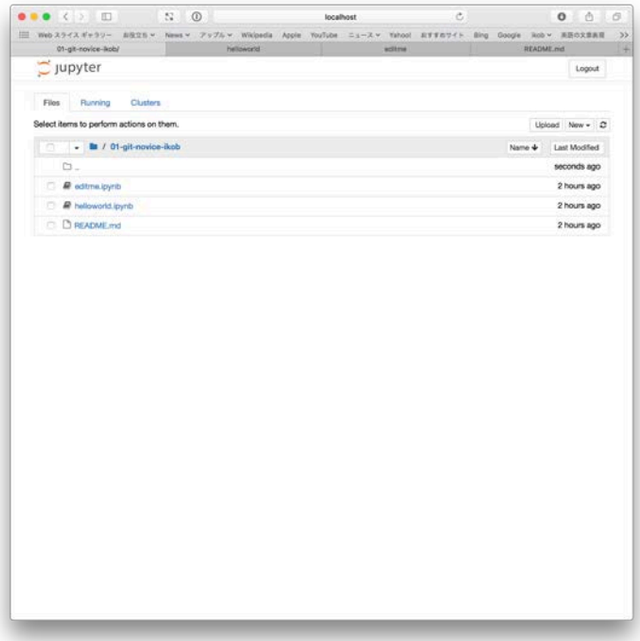
6.作業ディレクトリに移動、状態・ログを表示してみる

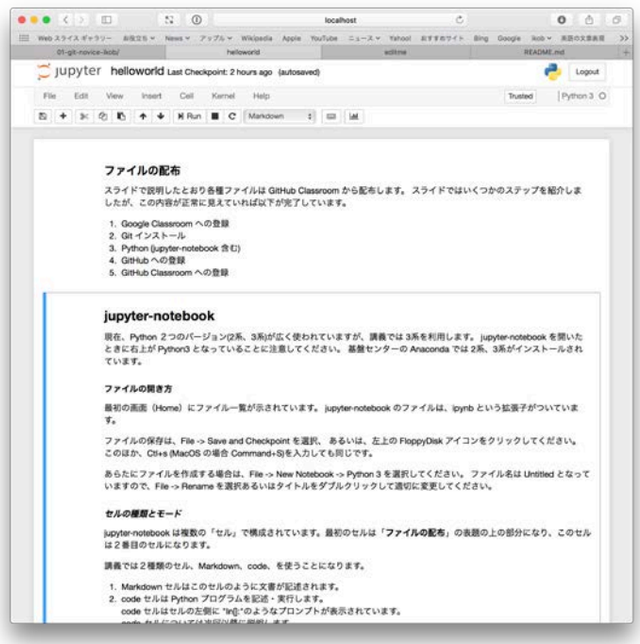
```
$ cd 01-git-novice-ユーザ名
$ git status
On branch master
Your branch is up to date with ‘origin/master'.
$ git log
コミットログの表示
```



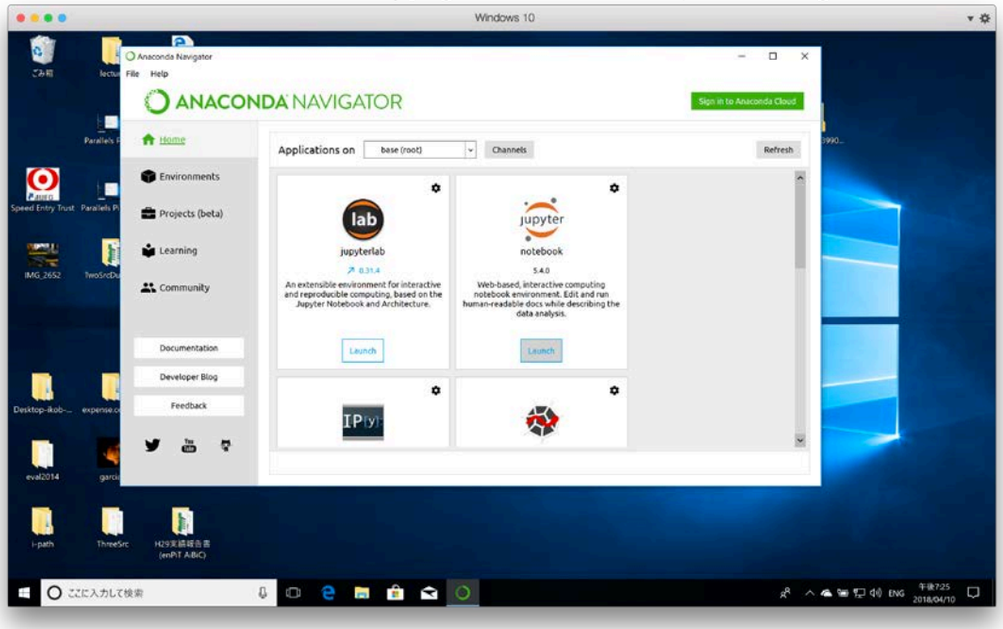
jupyter-notebook について

- 講義では jupyter-notebook を使う
- jupyter-notebook は Web ブラウザを利用した対話的なプログラミング環境
  - IPython Notebook から改名した
    対象が Python プログラミングに限定されないため
- プログラミング – プログラム実行がシームレスにおこなえる
- テキスト・画像出力に対応
- 残念ながら VCS との連携は弱い
  現在の IDE (Integrated Development Environment:統合開発環境) では VCS 連携は必須





演習：jupyter-notebok を使ってみる



1.Anaconda Navigator (3) を起動

2.Navigator から jupyter-notebook を起動

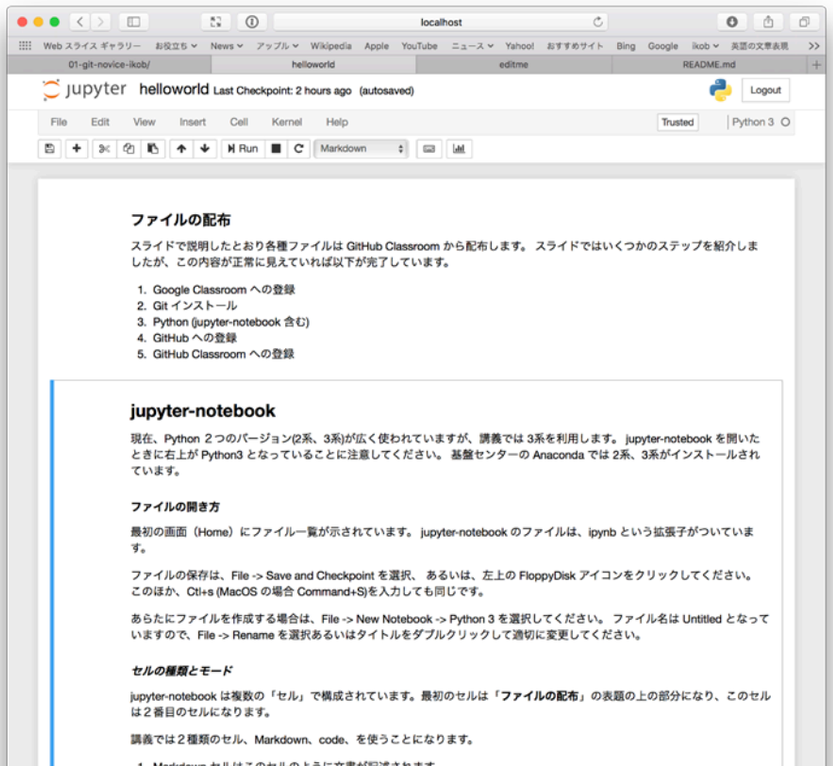
3.Web ブラウザが立ち上がり自身のホームディレクトリのファイル一覧が表示される

4.(作業ディレクトリに移動)

5.“01-git-novice-ユーザ名”に移動

6.“helloworld.ipynb” を開く(右下)

- しばらくこれを使って説明する



演習：GitHub レポジトリの複製
ステージング/コミット/Push

1.git 作業ディレクトリでファイルの差分を確認する

```
$
$ git diff
diff --git a/README.md b/README.md
index b2f5e52..2885af2 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,2 @@
-# OCS-git-novice
¥ No newline at end of file
+# OCS-git-novice
……
```

2.変更したファイルをステージングする。git status で ”Changed to be committed”に変更ファイルが含まれていれば良い

```
$
$ git add <変更したファイル、ここでは README.md>
$ git status
```

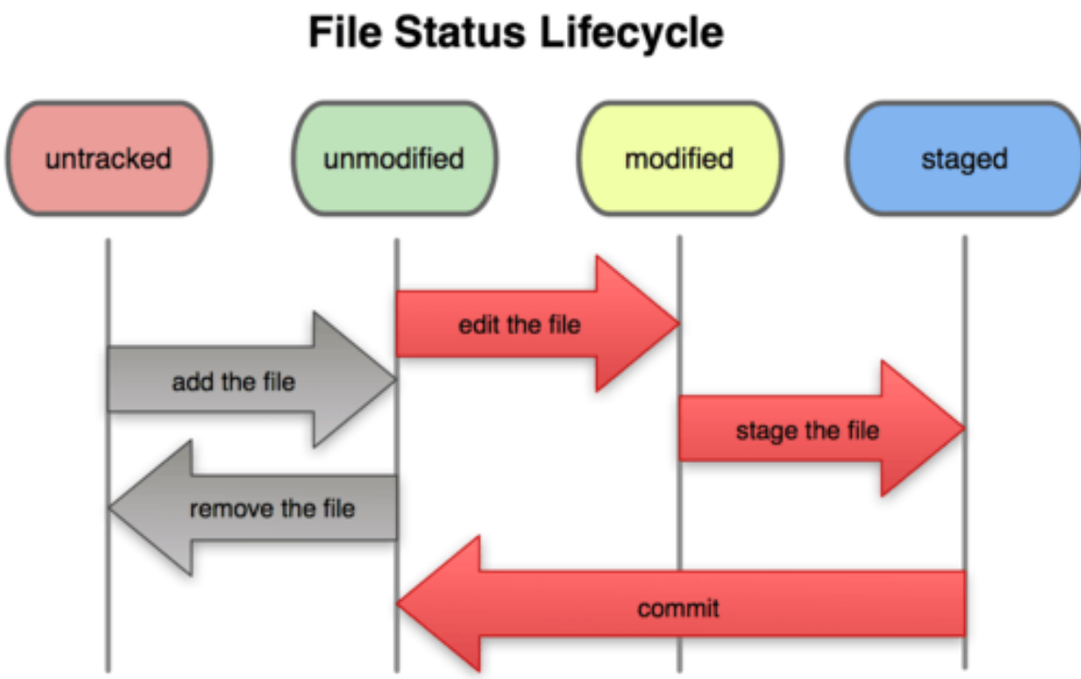
3.変更をコミットする、-m 以降のメッセージはなんでも良い。-m を忘れるとエディタが立ち上がる。ので注意するgit status で ”Changed to be committed”からステージングしたファイルがなくなっていれば良い

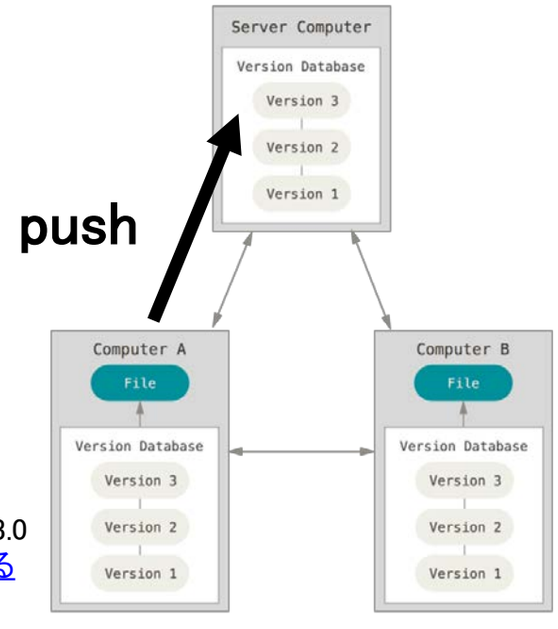
```
$
$ git commit -m "My first commit"
[master daf9da4] My first commit
 1 file changed, 2 insertions(+), 1 deletion(-)
$
```

4.Master に push する、途中で GitHub のユーザ名とパスワードを聞かれる

```
git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 425 bytes | 425.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/UTokyo-OCS/01-git-novice-ikob.git
   f1cf4fc..daf9da4  master -> master
```

『Pro Git』 1st Edition, 2009 CC BY-NC-SA 3.0
https://git-scm.com/book/ja/v1/Git-の基本-変更内容のリポジトリへの記録





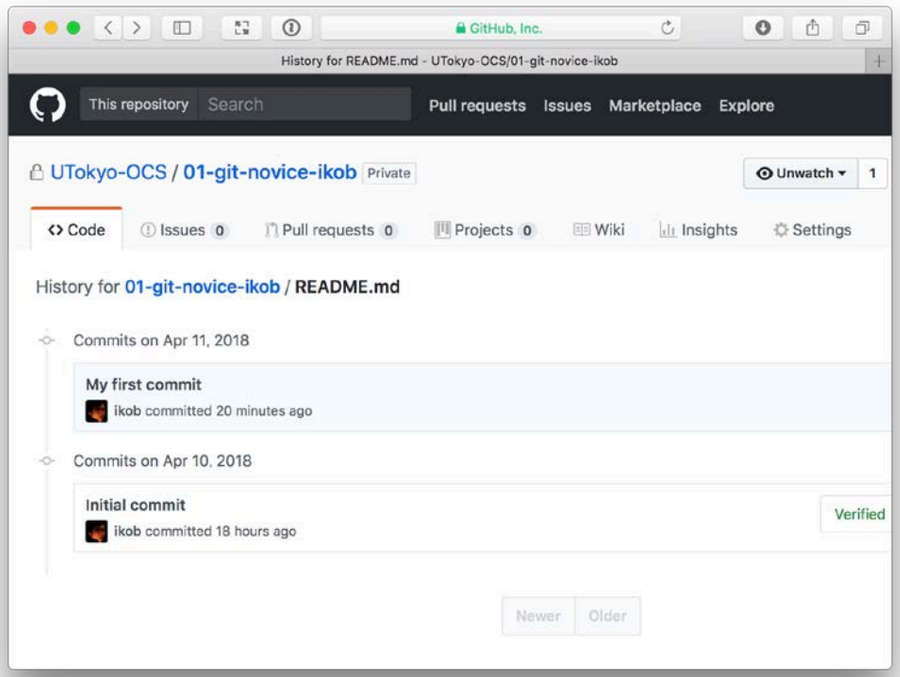
『Pro Git』 2nd Edition, 2014 CC BY-NC-SA 3.0
https://git-scm.com/book/ja/v2/使い始める-バージョン管理に関して

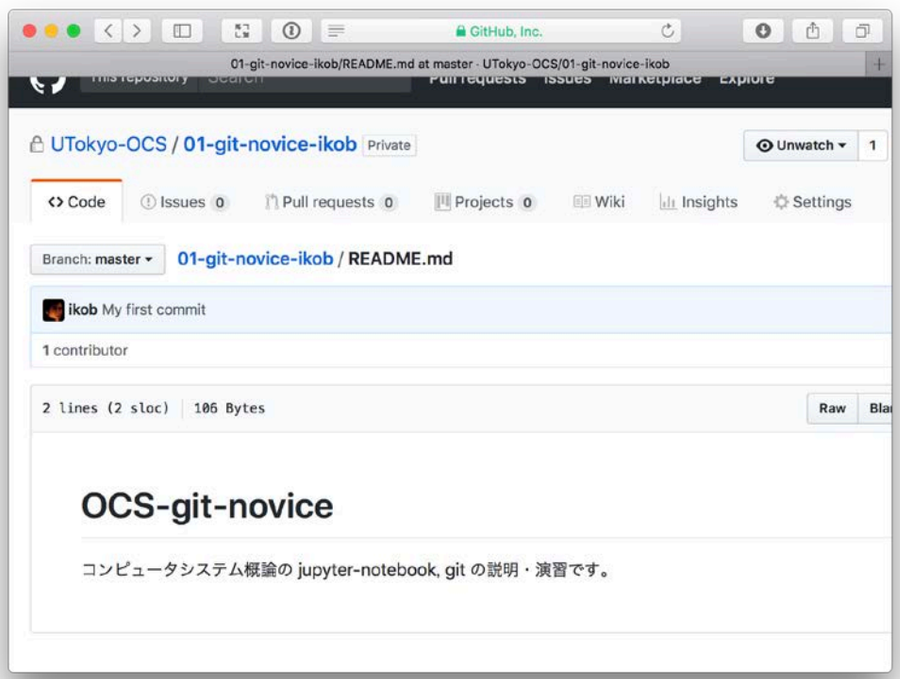
演習：GitHub レポジトリの複製
コミット/push 内容の確認

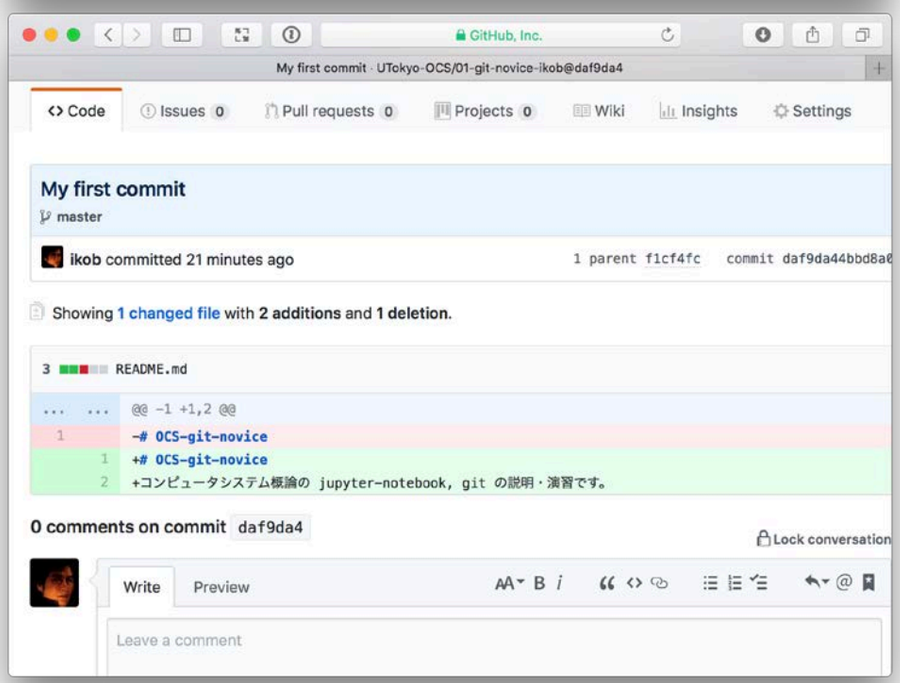
1.GitHub Classroom 登録で作られた GitHub レポジトリページを表示

2.README.md が更新されていることを確認する

3.README.md の内容を表示、History を確認する







本日の課題：markdown.ipynb
読んで指示にしたがってください