

初心者のためのマテリアルズ・ インフォマティクス入門

産業技術総合研究所
安藤康伸

自己紹介

名前：安藤康伸（38）

出身：愛知県西尾市

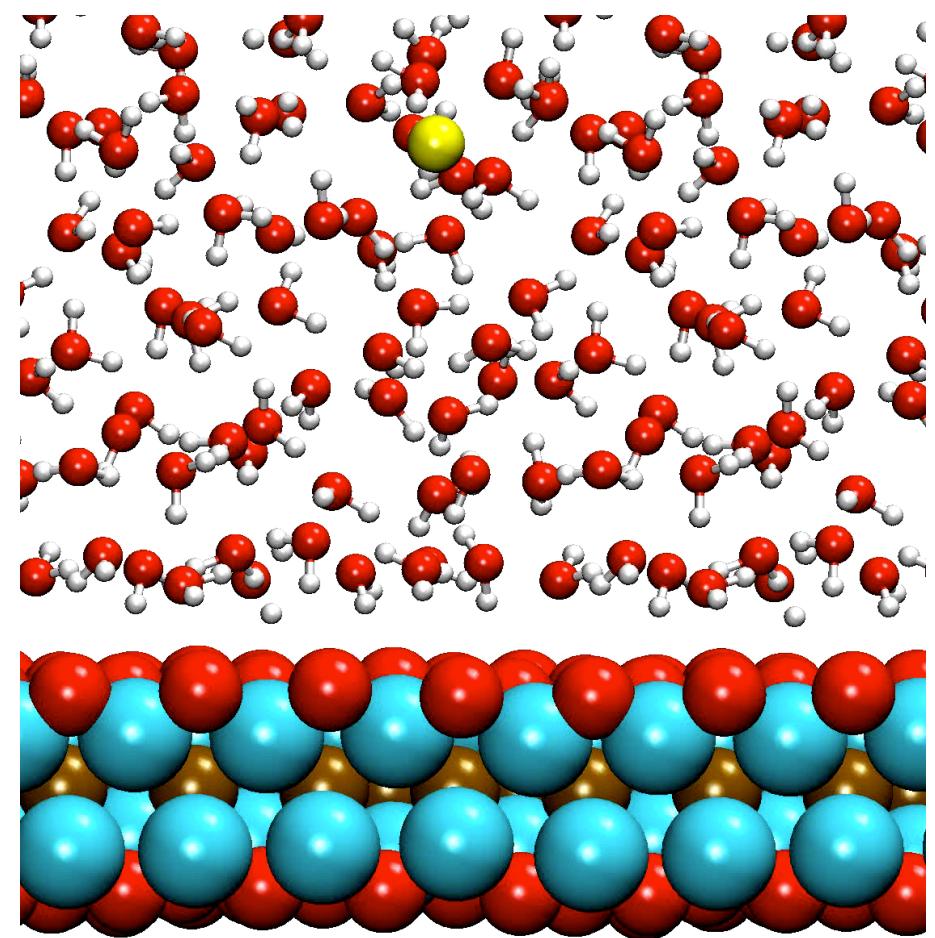
略歴

- ✓ 2012.3. 博士（理学）@東大院理
- ✓ 2012.4 ~ 2013.4 ポスドク@AIST
- ✓ 2013.5 ~ 2016.3 助教 @東大マテ工
- ✓ 2016.4 ~ 研究員 @
- ✓ 機能材料コンピューテーション
- ✓ デザイン研究センター (CD-FMat), AIST

Back ground

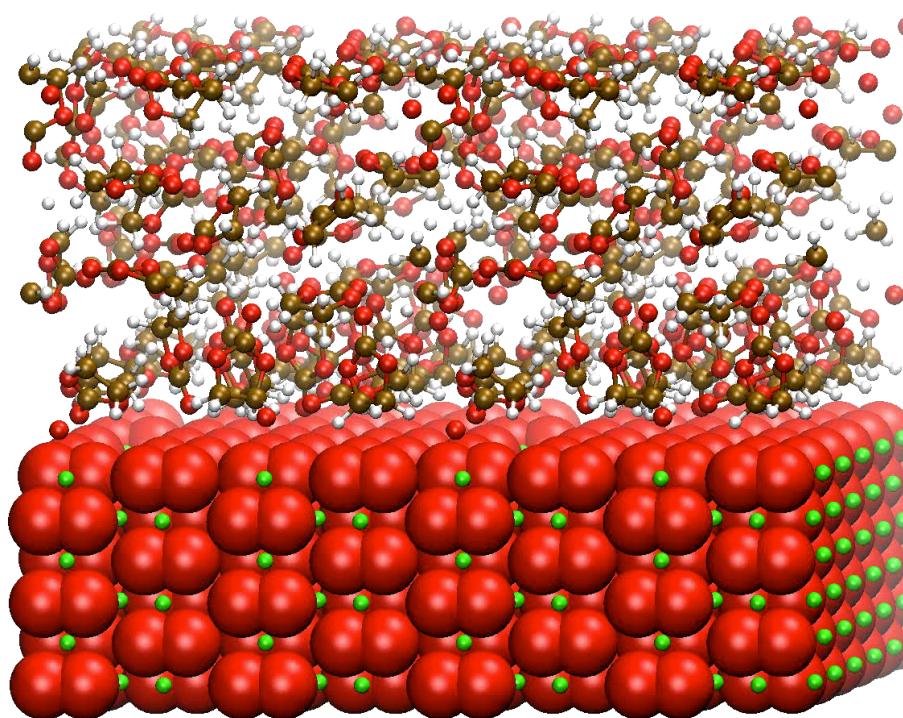
- ✓ 物性物理学
- ✓ 表面・界面科学
- ✓ 計算物質科学

情報科学 × 計算科学

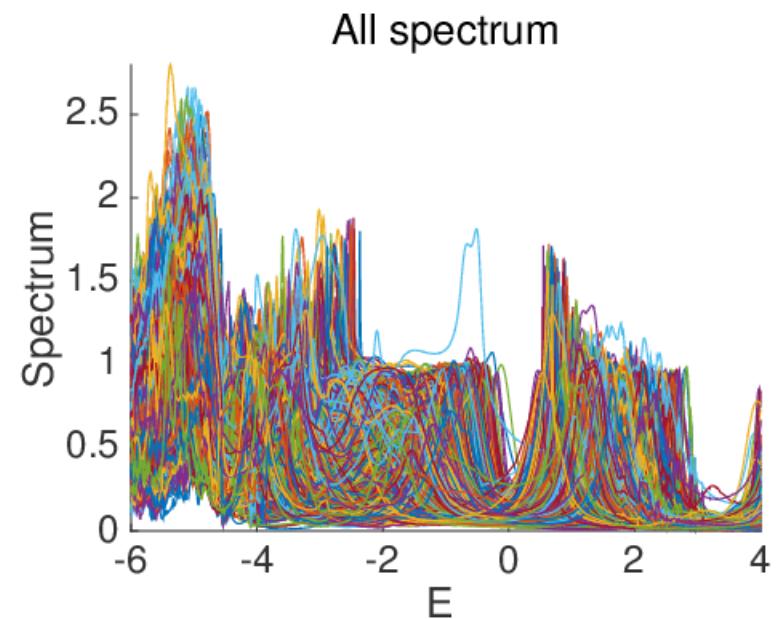


電極－電解液界面の第一原理分子動力学

情報科学に至った経緯



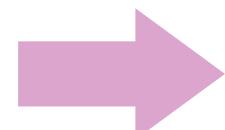
MDの結果をひたすら”観察”



600本... 一体どうすれば。。

その他 約10万本のスペクトル処理など

効率的な解析技術の習得が必須



機械学習の概要

機械学習の機能例

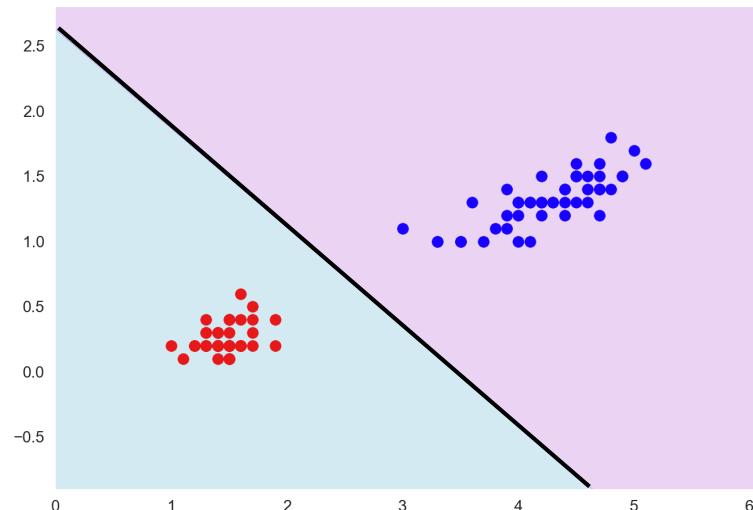
- ✓ 予測（補間）
- ✓ 特徴抽出（低次元化）
- ✓ 分類
- ✓ パターン認識

重回帰分析

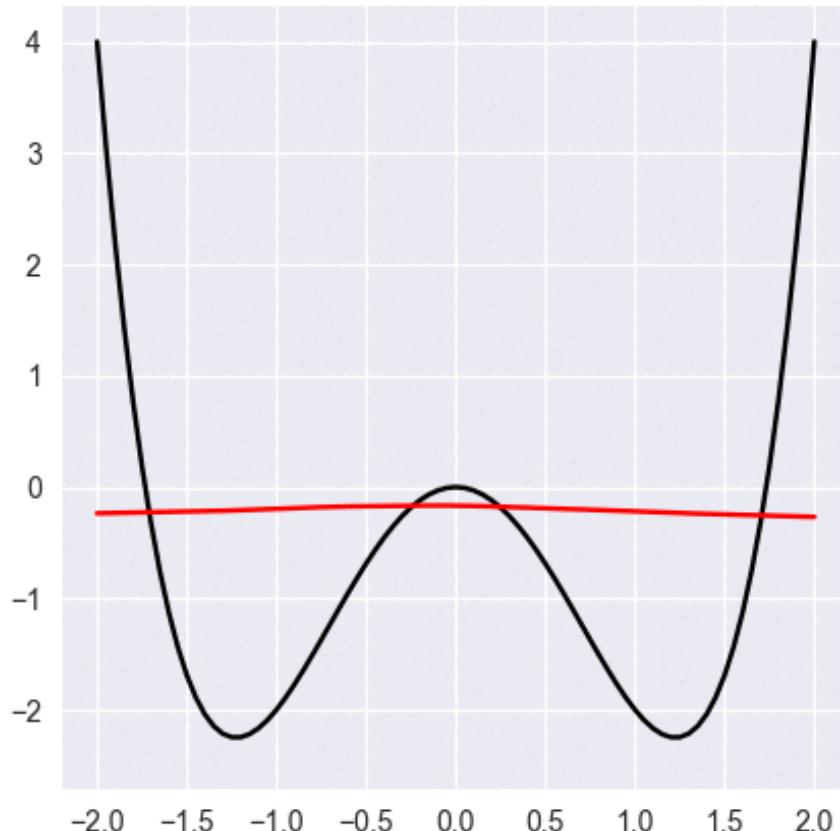
$$Y = \sum_i a_i X_i + b$$

サポートベクターマシン (SVM)

機械学習は
「点を打って線を引く」
ことから始まる



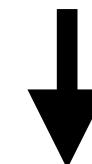
複雑な状況を表現する



Neural networkによるフィッティング

あくまで「線を引く」

直線がダメなら
曲がった線を引けば良い

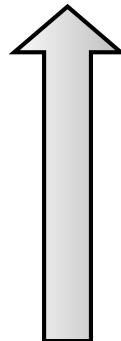


- ✓ 基底展開 (多項式, スプライン etc.)
- ✓ カーネル法, ガウス過程回帰
- ✓ ニューラルネットワーク

高次元空間でも同様に可能

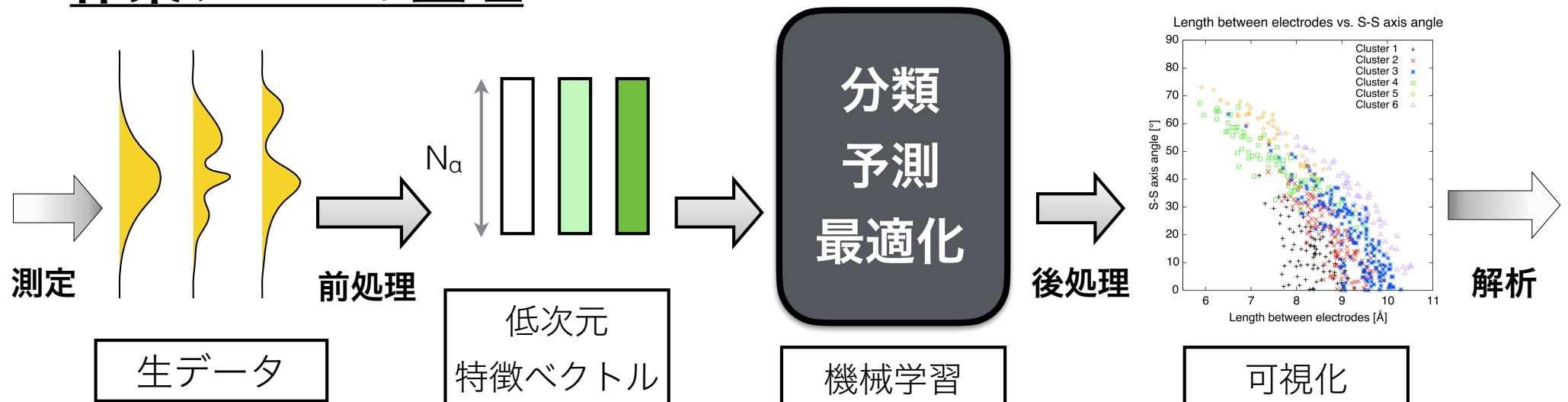
機械学習応用の基礎的な流れ

課題設定 ← 最重要ポイント



- ✓ 解析によって何が知りたいか？
- ✓ 比較したいデータは何か？（スペクトル・Heatmap etc.）
- ✓ 設定課題が十分ブレイクダウンできているかどうか
- ✓ 機械学習で見込める効果は何か？（高速化・客観化・因子発見など）

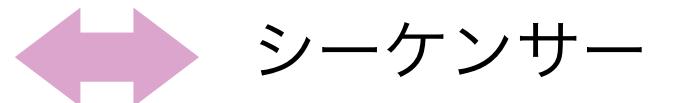
作業フローの整理



物質・材料データの特徴

ビッグデータというよりスモールデータ

- ✓ 典型的な実験：高々100, ようやく1,000サンプル出せることも
- ✓ データ取得コストが高い
- ✓ 他の実験室の結果とも直接比較が困難（環境・装置・人）
- ✓ 分光, 計算はビッグデータに近い



シーケンサー

汎用的な物質表現・処理方法の不在

- ✓ 対象：高分子・半導体・金属 etc.
- ✓ スケール：1 nm ~ 1 m (9桁！！)
- ✓ 構造やシステムの表現方法の多様性



ゲノム情報

物質科学の強みである「制御性（再現性）」「理論（事前知識）」を生かした中規模な情報処理スタイルの開発が必要

事例の蓄積が鍵

物質情報科学が抱える課題

「見つかった」は「わかった」わけではない

- ✓ 多くの物質科学者は「メカニズム」に興味がある
- ✓ 材料科学者・企業は「新材料」を合成したい
- ✓ 何を持ってわかったとするか物質科学観の転換点？
- ✓ 機械学習の結果は必ずしも「物理法則」を考慮していない

「高精度予測」と「モデル可読性」のトレードオフ

非線形

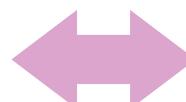
$$E[f(x)|Y] = k^T(K + \sigma^2 I_n)^{-1}Y$$

精度は高いがどのパラメータが
重要か読みづらい

線形

$$Y = \sum_i a_i X_i + b$$

精度は低いがどのパラメータが
重要か解析しやすい



情報科学VISAの獲得

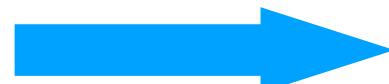
全員が機械学習の専門家になる必要はない

- ✓ 先端アルゴリズム開発などハードルが高すぎる
- ✓ まず初步的なことさえ理解できれば、先端研究をある程度フォローできる
- ✓ 必要なのは、専門家と「会話」ができること（VISAの発給要件）

課題設定こそ研究者の真骨頂

- ✓ 機械学習の専門家に「物性・材料」の理解を前提にするのは酷
- ✓ 自分たちの専門性を活かした「課題設定」こそ、物性科学者がやるべきこと
- ✓ 課題解決に必要な機械学習の技術に当たりがつけば最高

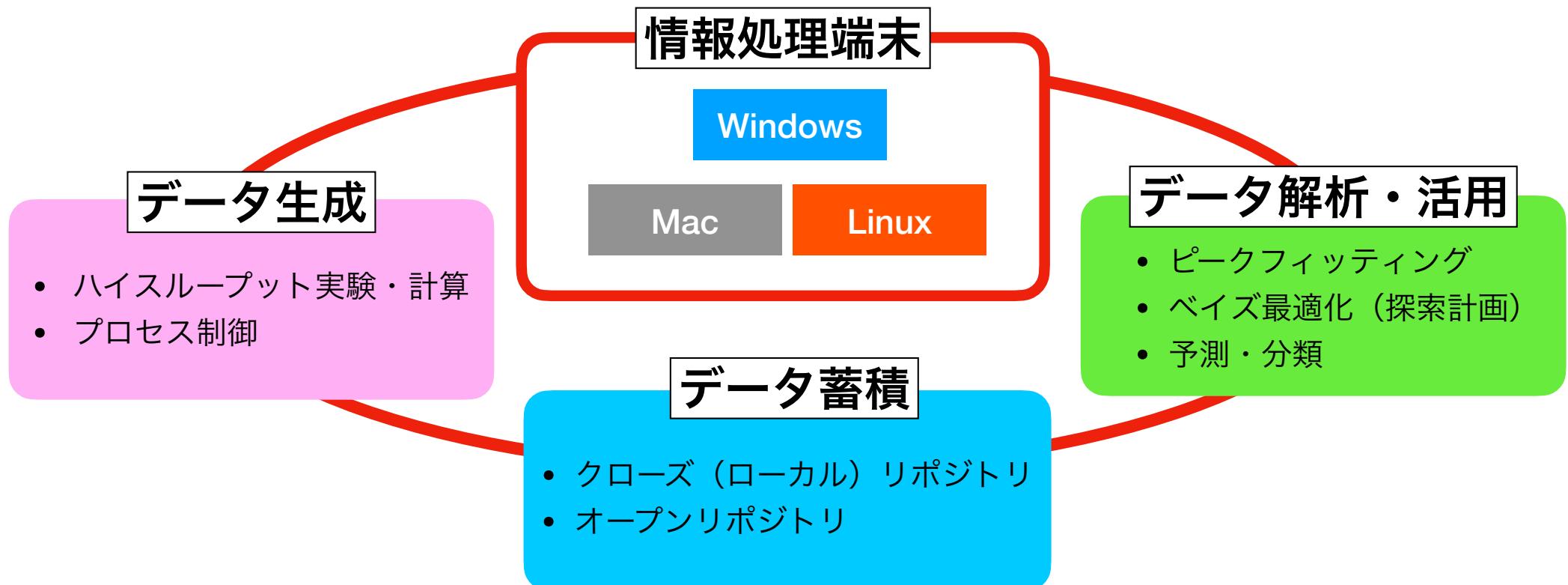
機械学習の流儀を嗜み（市民権の獲得）、専門家とコミュニケーションしながら自らの課題を解決することが理想



まずは初步と事例・環境整備

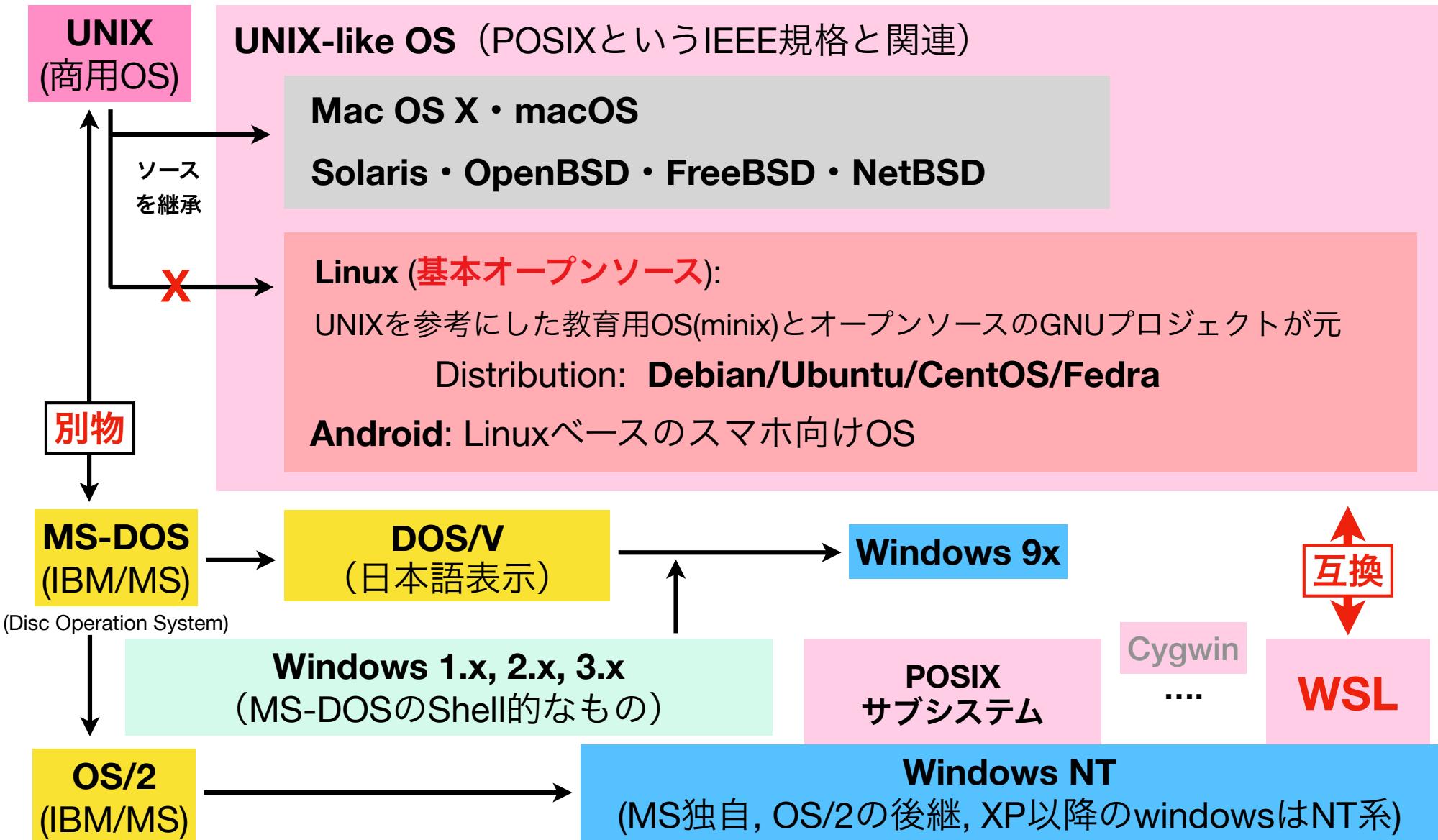
MIの研究環境

情報処理端末はMI研究環境の基盤



- 研究者が慣れている環境を利用すれば良いが、それぞれで整備方法が異なるので注意
- フリーソフトウェアの多くはオープンソースのLinux環境で整備されたものが多い

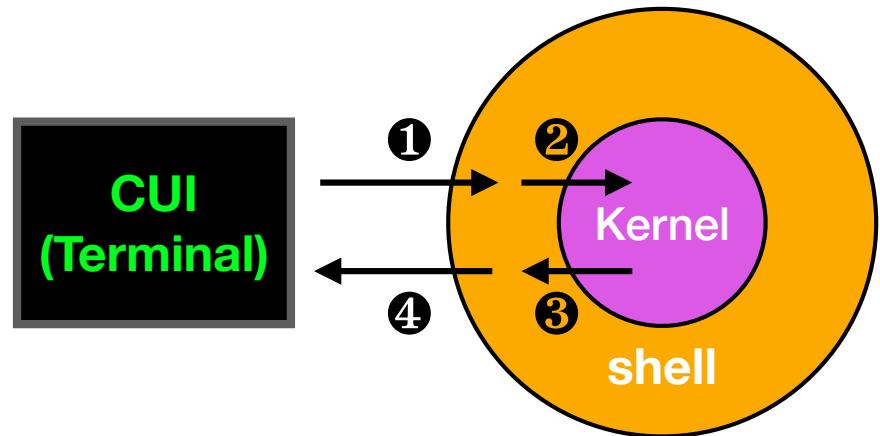
OSの分類



ユーザーインターフェース

コマンドユーザーインターフェース (CUI)

文字入力によりShell（シェル）と呼ばれるスクリプト言語を介してOSの中心部分であるkernel（カーネル）とやりとりするためのツール



Mac

CUI：ターミナル・端末

Linux

UNIX系Shell: sh, bash, zsh など

Windows

コマンドプロンプト：MS-DOSのためのシェルが利用可能（古い）

PowerShell：windows独自のシェルでAzureなどWin系サービスに利用可能
(エイリアスによりメジャーなbash commandも受け付ける)

Windows Terminal：上記とWSL(後述)を統合して使えるCUI

- ① コマンド入力 ② kernelがわかるように翻訳
③ kernelの処理結果を解釈 ④ 要求に応じて出力

プログラム開発・実行環境

マテリアルズ・インフォマティクス
の標準的な言語



しかし多くの人がトラブルに巻き込まれる。。。

未だにハードルが高い要因

- とりあえずanacondaを入れただけでよくわかっていない
- 気づいたらpythonが沢山あってどれが優先されてるのか謎
- パッケージ管理がうまくいかずに動かなくなる
- 実体がどこにあるのかわからない
- トラブルが生じた際に対処できなくなる
- そもそもpython, anaconda, jupyterとかよくわからない

Python環境の整理術を解説する

混ぜるな危険

Mac

Windows

Python標準搭載なし

Linux

distributionによる



Python環境の整備が必須

さまざまな導入目的

- 必要なパッケージをまとめて導入したい
- 使用したいプログラムの要件に合わせてパッケージを導入したい
- 外部計算環境に合わせた環境を準備したい
- データ解析プログラムを作成するための最小環境を導入したい

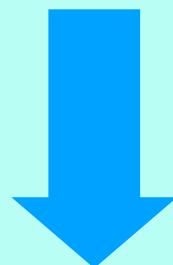
試行錯誤やさまざまな導入目的に対応した結果、トラブルが起きやすい
メンテナンスコストを下げる意味でも重要

如何にユーザーが管理可能な隔離環境を構築するかがポイント

隔離環境の構築

一般的な隔離環境の構築技術

OSレベル



軽量

- 仮想マシン (**Virtual machine**)
- コンテナ環境
- 仮想環境 (**Virtual Environment**)

Windows

Mac

Linux

例) **Anacondaを用いたpython仮想環境**

Option

Windows

UNIX-likeな実行環境の整備

1. Hyper-VによるLinux仮想マシン
2. **Windows subsystem for Linux (WSL)**
(Windows独自ソフトを使うのも選択肢)

Mac

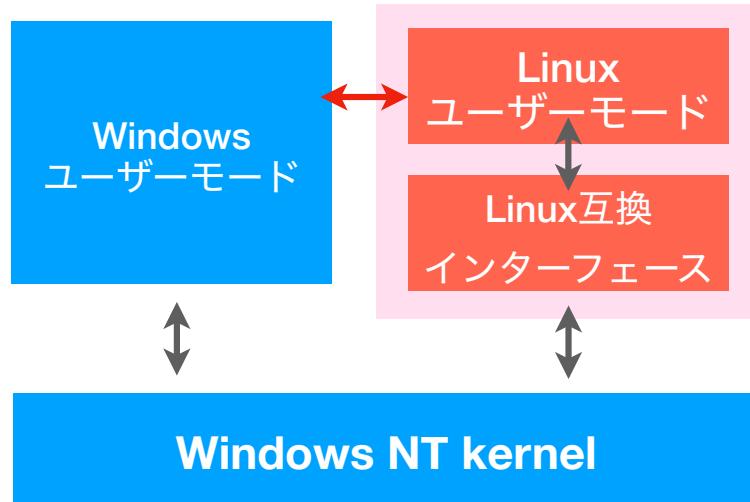
pythonのバージョン切替ソフト
pyenv 上にAnacondaを入れると便利
(Homebrewとの相性問題)

Windows subsystem for Linux (WSL)

Windows

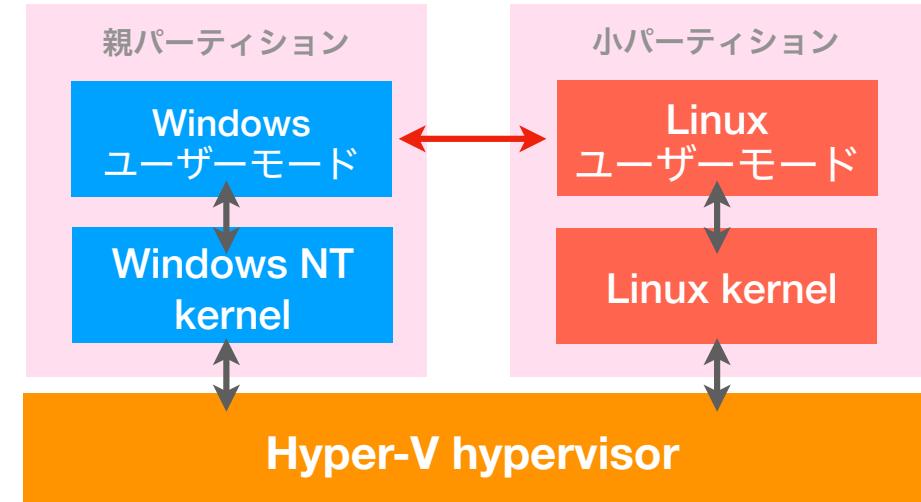
Windows NTは様々なOS (MS-DOS, OS/2, POSIX準拠OS) をサポートするよう設計されていた

➡ **Windows 10からLinux用のサブシステムが利用可能に！**



WSL1

- 軽量で使いやすい
- 正式なLinux kernelを採用していないので、利用できるサービスに制限
- 機能をオンにする手続きがやや煩雑



WSL2

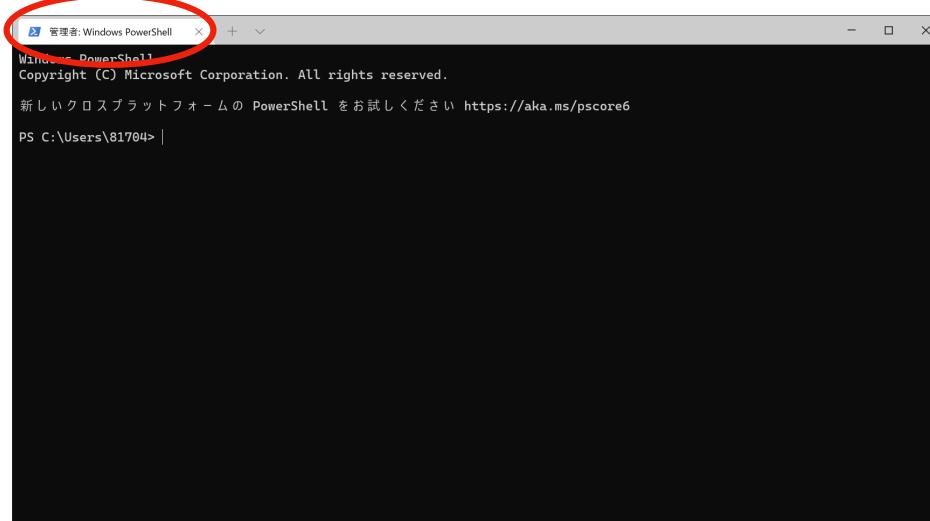
- Windows 10 May 2020 Update (version2004) で正式サポート
- 本物のLinuxカーネルを使用
- ホストとなるWindowsとは別のIPアドレスを使用

WSL2の整備

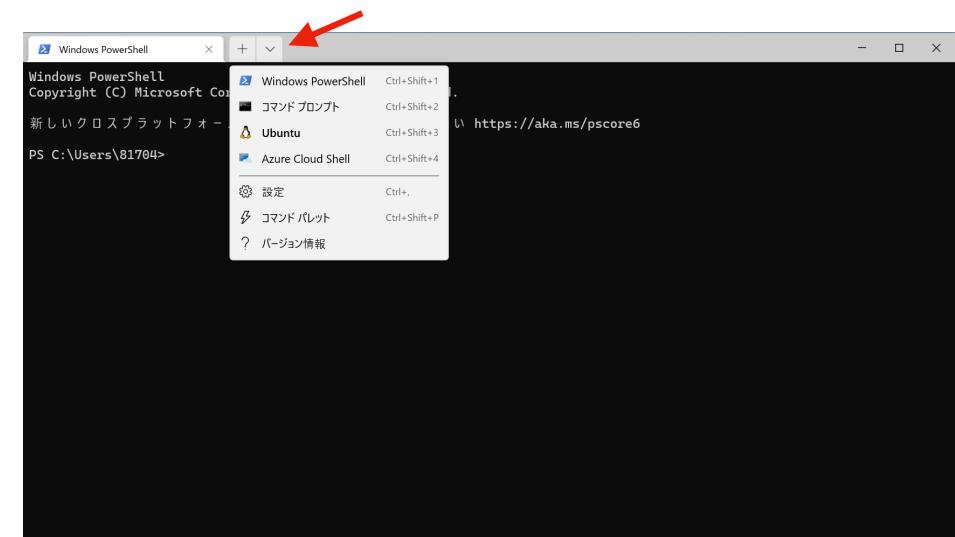
Windows

WSLのインストール

1. Windows terminal or Powershellを管理者として起動
2. Powershellを開き\$ wsl --install -d Ubuntuを実行
3. Ubuntuのターミナルが開いたらusernameとpasswordを入力



管理者になっていることを確認



再起動するとubuntuが選べるようになる

wslコマンドが通らないなどのトラブルがあった場合はwindowsのversionなどを確認

WSL2の整備

Windows

Windows 11からGUIにも対応（WSLg）

Windows Terminalの設定

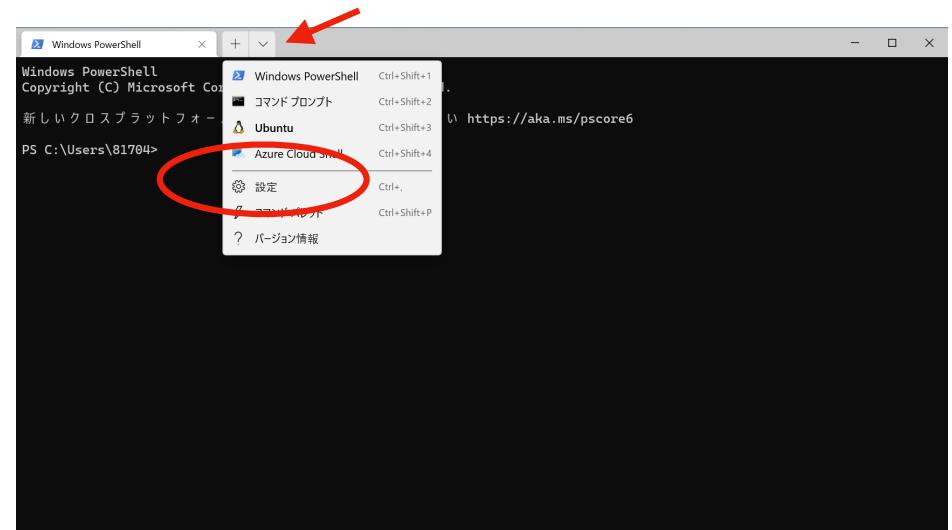
1. Windows terminalを起動
2. メニューを開いて「設定」を選択
3. サイドバーのスタートアップを選択して
「規定のプロファイル」を「Ubuntu」に指定
4. サイドバーのUbuntuを選択して
「開始ディレクトリ」をわかりやすいものに指定

Ubuntuのアップデート

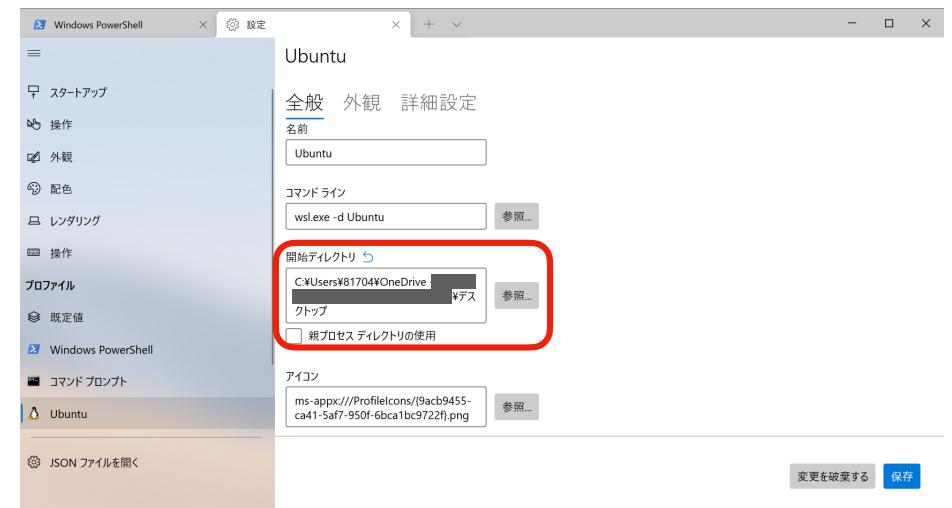
1. Windows terminalを起動
2. **\$ sudo update** を実行
3. **\$ sudo upgrade** を実行

UNIX command (cd, mkdir, rm -iあたり) を確認
emacs, lessなど入れておくと便利

\$ sudo apt install less emacs



メニューと設定の場所



開始ディレクトリの設定

Package Manager

Mac

Homebrewを使う場合、Anacondaとの相性がとても悪い（らしい）ので
pyenv (version management tools)で隔離しておくことを推奨する

Homebrew: rubyベースで書かれたMac (もしくはlinux)用のpackage manager.



Windows

Linux

apt (Advanced Package Tool):

Debian系(Ubuntu, KNOPPIX)のpackage manager.

yum (Yellowdog Updater Modified):

Redhat系(CentOS, Oracle, Fedora)のpackage manager.

dnf (Dandified Yum) がyumの後継でこちらを推奨



Python Package Manager

Windows

Mac

Linux

conda: Python, Rのパッケージ管理で使える

(極力)

pip: PyPIに登録されたパッケージを管理できる

混ぜるな危険



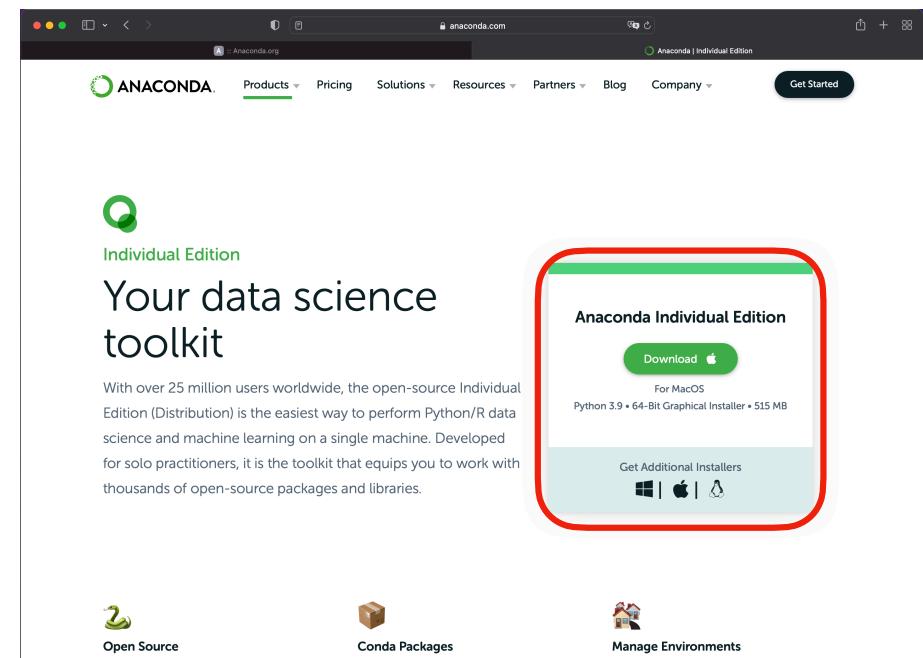
Anacondaのインストール

Anaconda:

- Python ディストリビューションのひとつ, Open data Science Platform
- R, Scalaもサポート
- “conda” commandによる依存関係を適切に処理したpackage管理
- 仮想環境の作成などをGUIベース (anaconda navigator) で実施可能

Windows Mac Linux

- Anaconda公式ページ
(<https://www.anaconda.com/products/individual>) からインストーラを取得可能
- Homebrewを用いる場合は次のページのスクリプトを実行すれば環境構築できる



Homebrewを用いた整備

setup anaconda.sh

```
$ brew install pyenv  
$ sh ~/.zshrc  
$ LATEST=$(pyenv install -l | grep anaconda3-20 | tail -n 1)  
$ pyenv install ${LATEST}  
$ pyenv global ${LATEST}  
$ .pyenv/shims/conda update -n base -c defaults conda  
$ .pyenv/shims/conda init zsh
```

macOSの標準Shellはzshに変更されたが、bashを用いる場合には、
zsh→bashとして実行

pyenvの導入場所

~/.pyenv以下に実態あり。隠しフォルダなので、\$ ls -la とすれば確認可能

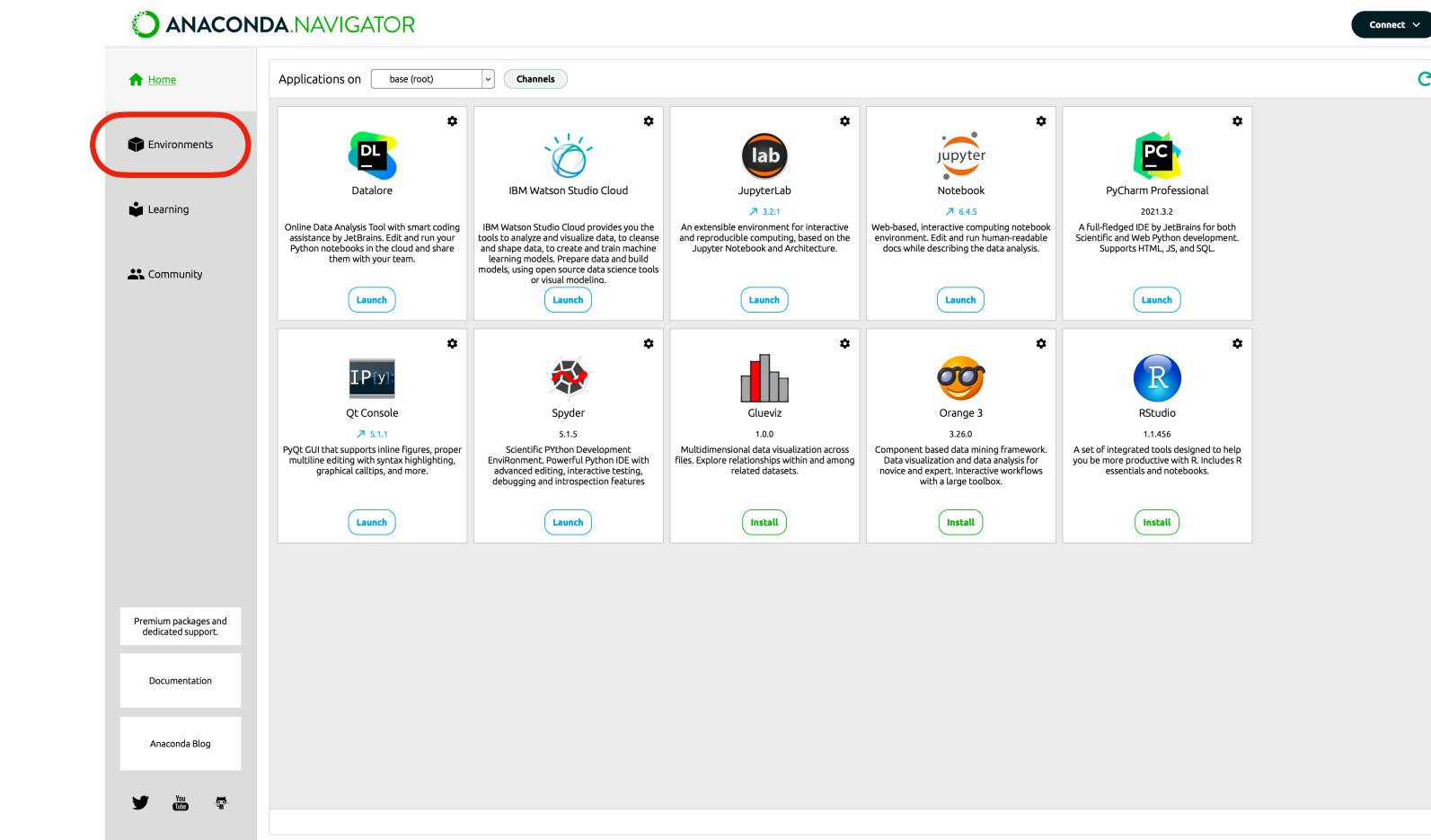
Python仮想環境

Windows

Mac

Linux

1. Anaconda Navigatorを実行
2. サイドバーのEnvironmentを選択



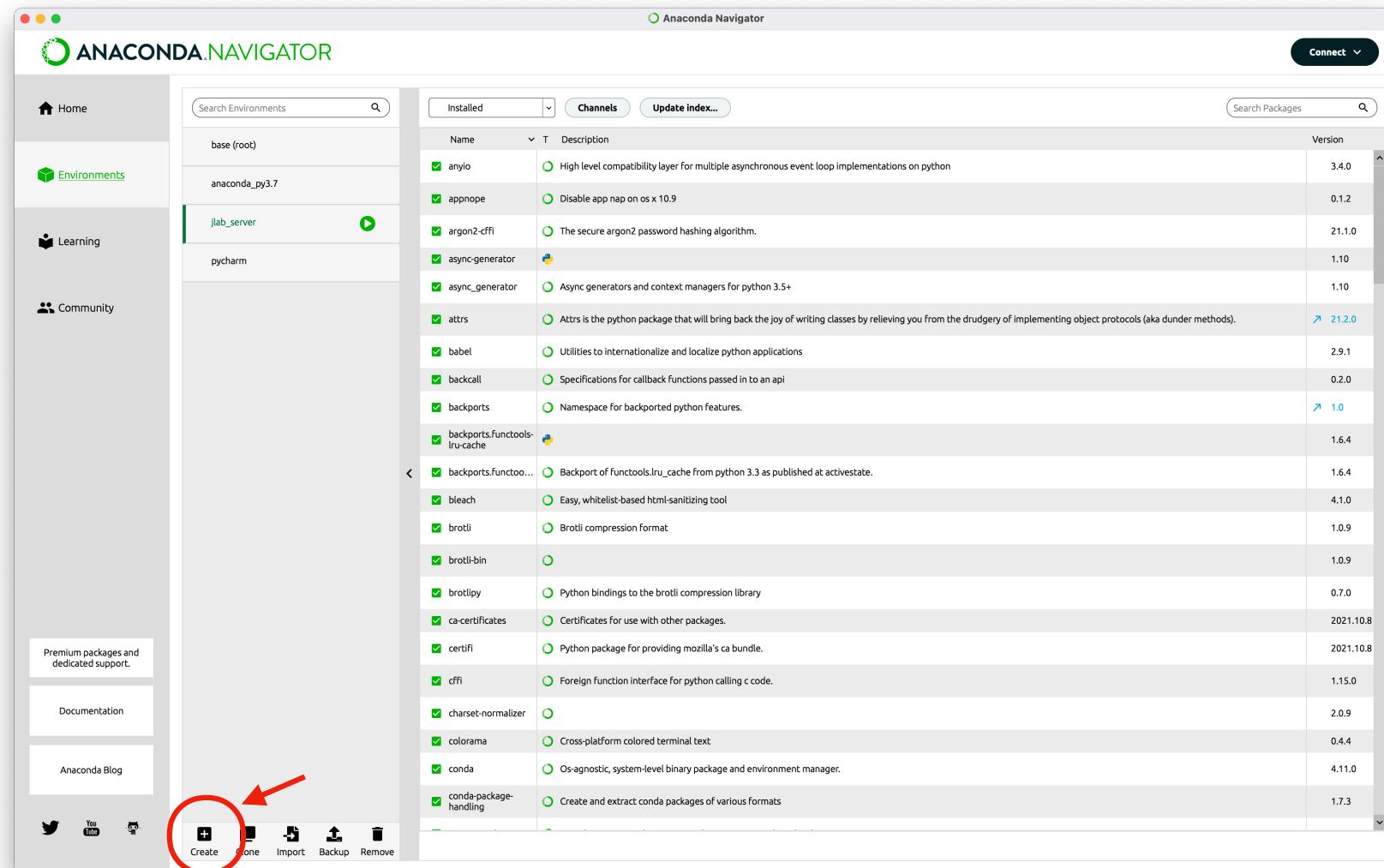
Python仮想環境

Windows

Mac

Linux

3. 新規追加, 4. 必須パッケージの追加



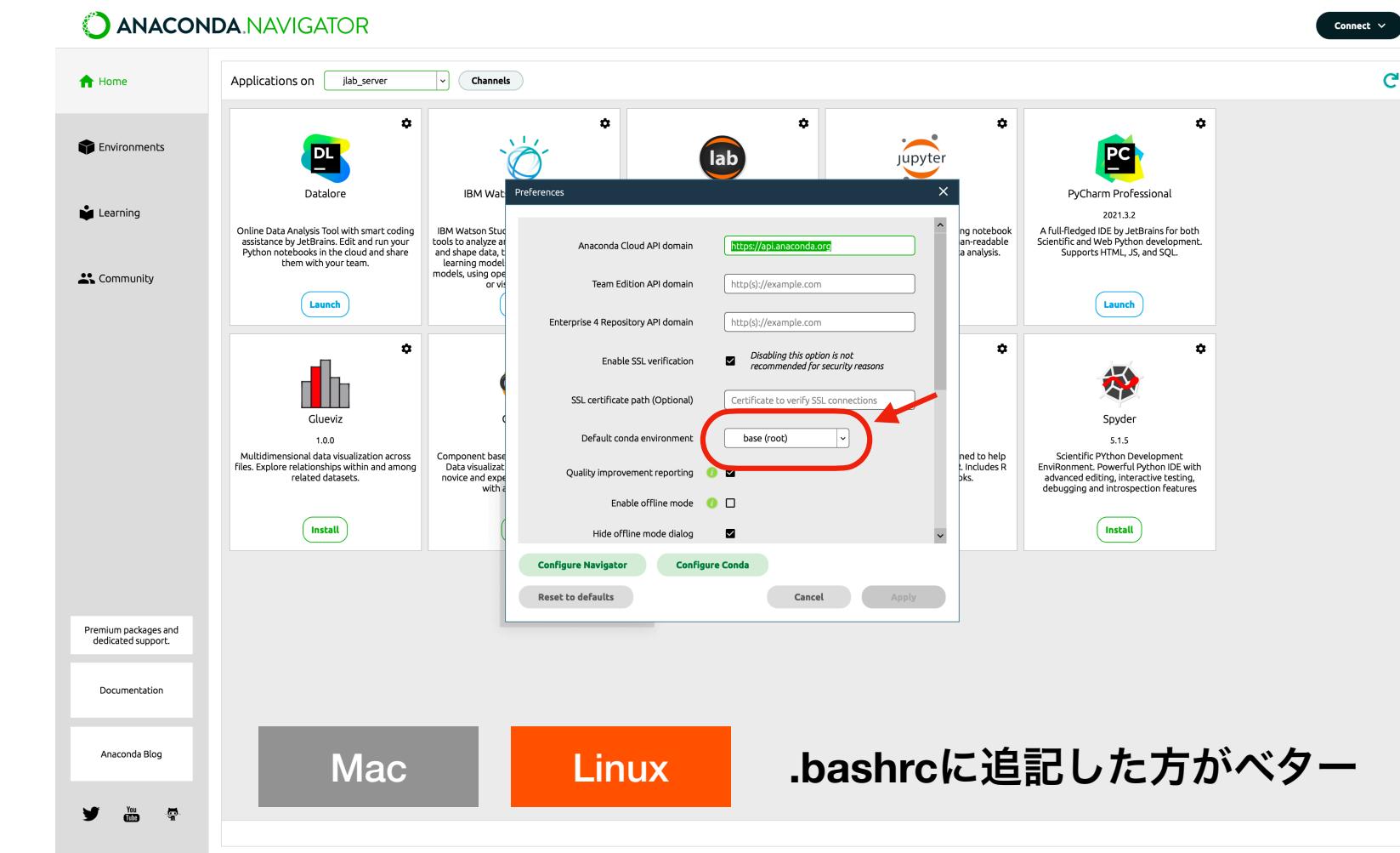
Python仮想環境

Windows

Mac

Linux

5. Preferenceでdefault env.を変更



Mac

Linux

.bashrcに追記した方がベター

Jupyter notebook・lab

- IPython (pythonの対話的シェル) から分離したProject Jupyterが主導して開発
- Jupyterは主としてサポートするJulia・Python・Rに由来
- Jupyter notebook(ipynb)形式はmarkdown/codeを同じレベルで記述・実行できるファイル
- プログラム開発環境ではなくあくまでも”スタディノート”という位置付け

Jupyter notebook

- Webアプリケーション
- Ipynbをサポート
- 対話型でコードを実行しながら作業記録も残せる



Jupyterlab

- Jupyter notebookの後継
- Ipynbをサポート
- 総合開発環境に近い使用感
- ターミナル, markdownなども個別に取り扱える



Pythonベース機械学習環境のスタンダード

Jupyterlabの起動

ANACONDA NAVIGATOR

Connect ▾

Home

Environments

Learning

Community

Premium packages and dedicated support.

Documentation

Anaconda Blog

Twitter YouTube GitHub

Applications on base (root) Channels

Datalore

IBM Watson Studio Cloud

JupyterLab 3.2.1

Notebook 6.4.5

PyCharm Professional 2021.3.2

Qt Console 5.1.1

Spyder 5.1.5

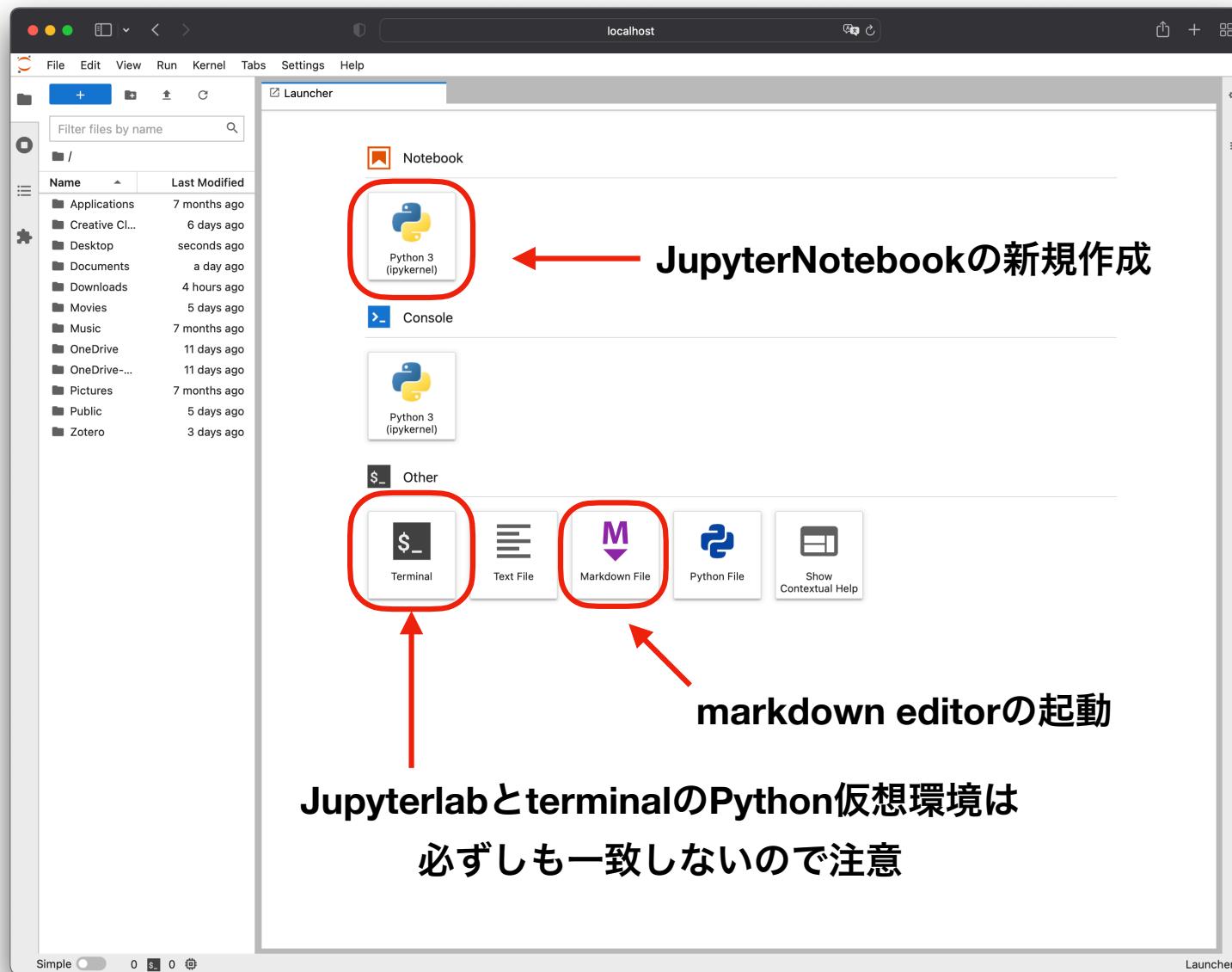
Glueviz 1.0.0

Orange 3 3.26.0

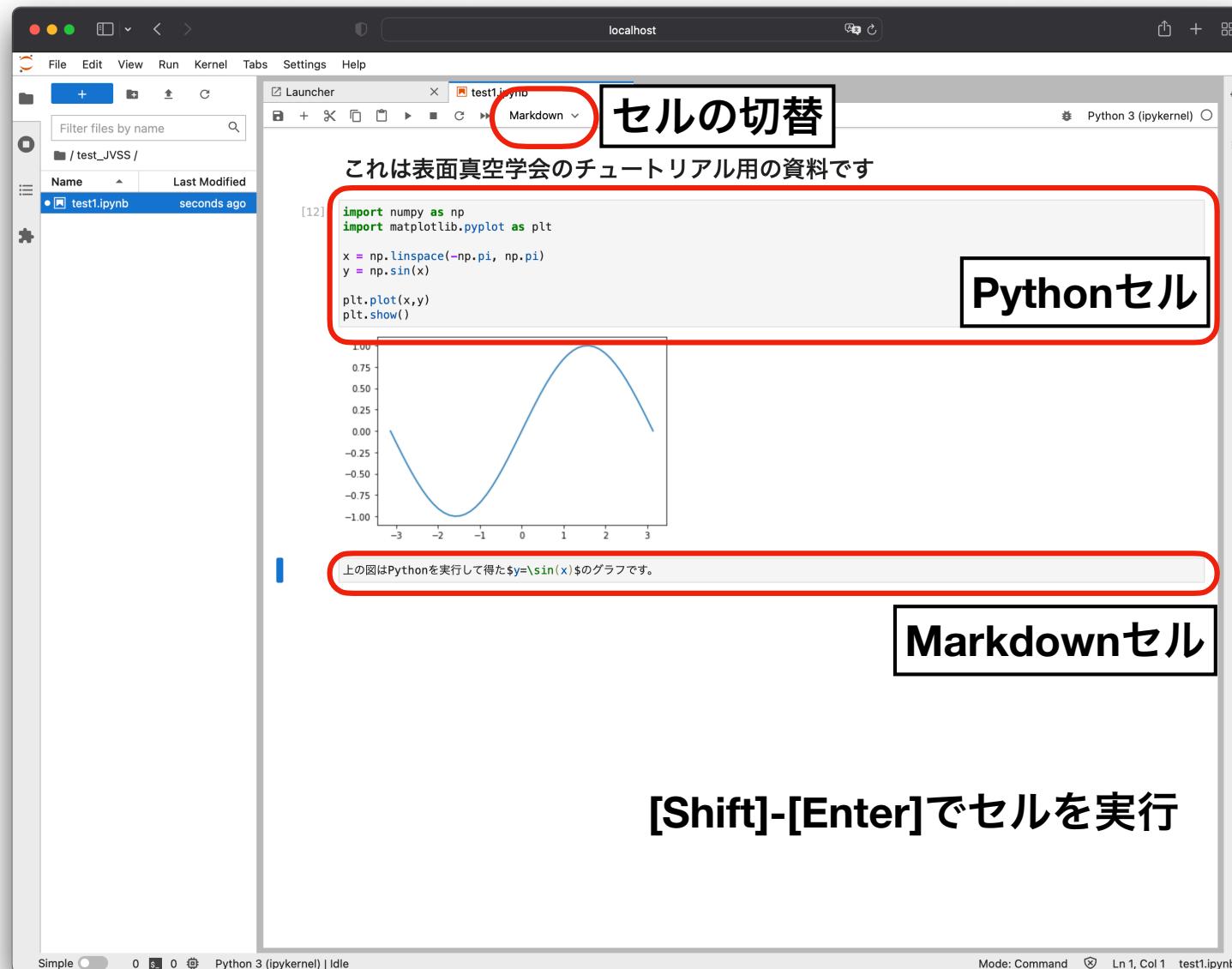
RStudio 1.1.456

The screenshot shows the Anaconda Navigator interface. On the left is a sidebar with links for Home, Environments, Learning, and Community. Below that are sections for Premium packages and documentation. At the bottom are social media links for Twitter, YouTube, and GitHub. The main area is titled 'Applications on base (root)' and shows several icons for different tools. The 'JupyterLab' icon, which has a version of '3.2.1' below it, is highlighted with a red rectangular box. Other applications shown include Datalore, IBM Watson Studio Cloud, Notebook, PyCharm Professional, Qt Console, Spyder, Glueviz, Orange 3, and RStudio. Each application card includes a 'Launch' or 'Install' button and a brief description.

JupyterLab Launcher

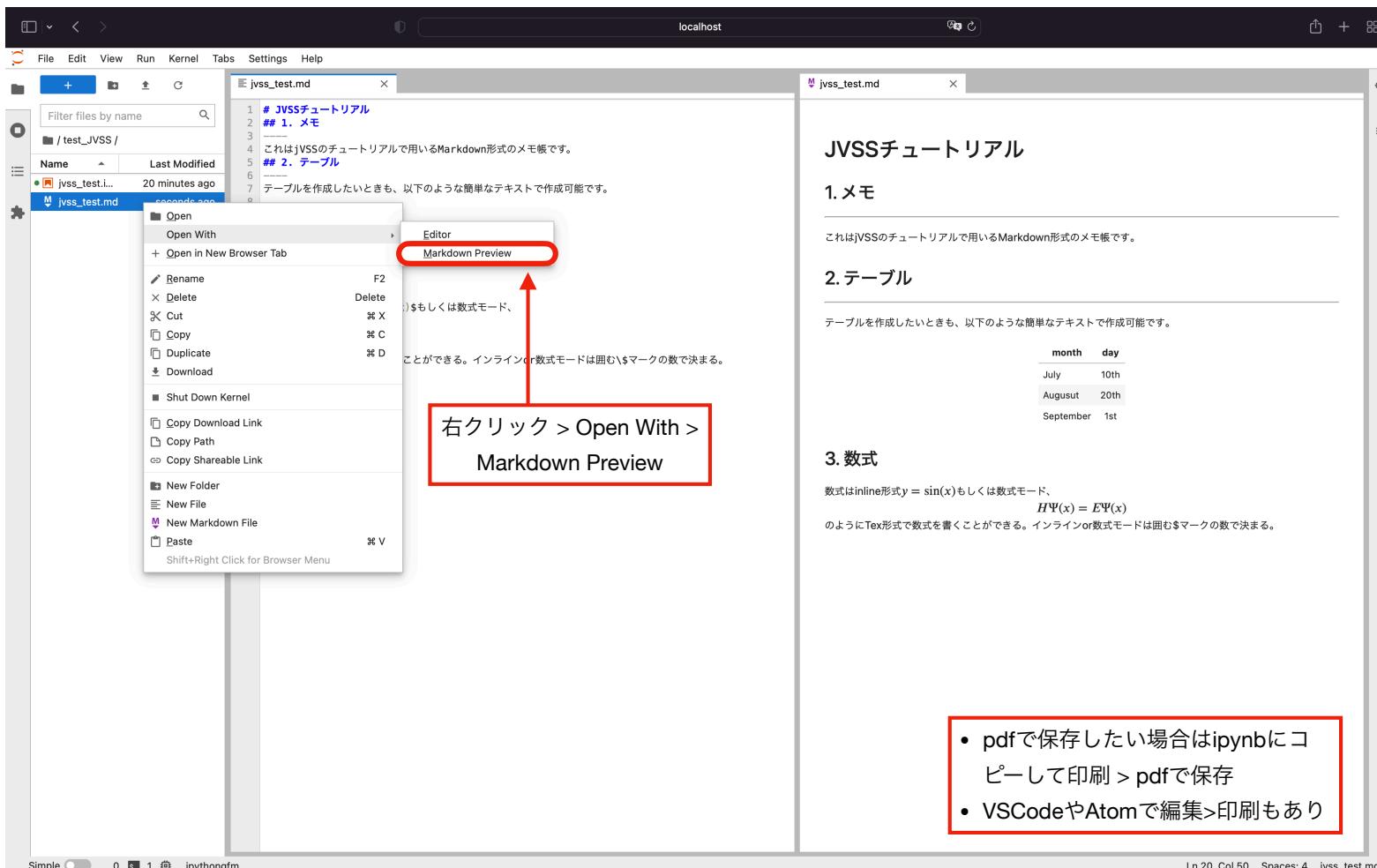


Jupyter notebook



markdown

- 数式・表などを簡単な表記でメモできる軽量マークアップ言語
- プレーンテキストからHTMLを手軽に作成する目的で開発



- pdfで保存したい場合はipynbにコピーして印刷 > pdfで保存
- VSCodeやAtomで編集>印刷もあり

JupyterLab Desktop App

実はelectron[*]によるデスクトップ版もリリースされている

Download

release v3.2.5-2

Before installing please read the [Python Environment Customization Guide](#) if you plan to customize the Python environment to add new packages. If you have an existing JupyterLab Desktop installation, please uninstall it first by following the [uninstall instructions](#).

- [Debian, Ubuntu Linux Installer](#)
- [Red Hat, Fedora, SUSE Linux Installer](#)
- [macOS Installer](#)
- [Windows Installer](#)

Launching JupyterLab Desktop



- カスタマイズ機能がまだ不完全
- まだまだ発展途上のため痒いところに手が届かない
- Version updateは `uninstall → reinstall` しかない
- 重い（1.3 GBくらいある）
- Anaconda navigatorで仮想環境は管理できる

***electron:** Web applicationをデスクトップアプリにするためのツール

Python Package管理

Windows

Mac

Linux

方法1. anaconda navigator

Environmentで仮想環境を選択しリストからパッケージを管理

(アップデートが個別選択なので面倒)

方法2. Jupyterlab

Pythonセル中でcondaコマンドを利用
(“!”を冒頭に付ける必要あり)

方法3. Terminal

ターミナルでcondaコマンドを利用
(仮想環境がjupyterと一致しているか要確認)

```
## anaconda環境の詳細情報
## "!"をつけるとshell command を受け付けることができる
!conda info -e
!conda info

# conda environments:
#
pycharm      /Users/yasunobu/.conda/envs/pycharm
base         * /Users/yasunobu/.pyenv/versions/anaconda3-2021.11

active environment : base
active env location : /Users/yasunobu/.pyenv/versions/anaconda3-2021.11/envs/anaconda_py3.7
shell level : 1
user config file : /Users/yasunobu/.condarc
populated config files : /Users/yasunobu/.condarc
conda version : 4.11.0
conda-build version : not installed
python version : 3.7.11.final.0
```

Jupyterlabでの実行の様子

macはここに出る

```
(anaconda_py3.7) yasunobu@Spiegel-2 ~ % conda info -e
# conda environments:
#
pycharm      /Users/yasunobu/.conda/envs/pycharm
base         * /Users/yasunobu/.pyenv/versions/anaconda3-2021.11
anaconda_py3.7          /Users/yasunobu/.pyenv/versions/anaconda3-2021.11/envs/anaconda_py3.7
```

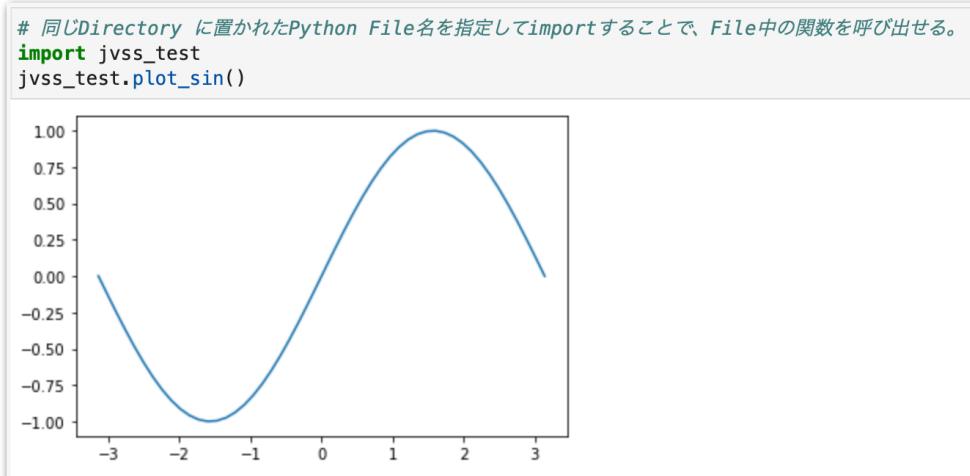
Anaconda リポジトリにないものはpip コマンドでインストール (方法2, 3)

自作関数の再利用

コードの保存・共有・共有は研究速度を高める上で極めて重要

Jupyter notebookで残されていても、そのコードは再利用できない

→ jupyter notebookでテストが済んだら関数化して外部ファイルにまとめる



JupyterLabでの実行結果

```
jvss_test.ipynb          jvss_test.py
 1 import numpy as np
 2 import matplotlib.pyplot as plt
 3
 4 def plot_sin():
 5     x = np.linspace(-np.pi, np.pi)
 6     y = np.sin(x)
 7
 8     plt.plot(x,y)
 9     plt.show()
10
11 return
```

外部ファイル(jvss_test.py)の中身

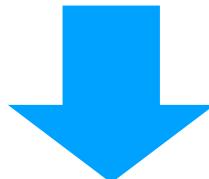
ノウハウの構造化・共有化

- .ipynb fileと同じフォルダに関数fileを設置
- 別フォルダに置く場合は__init__.pyが必要

Packageの公開

外部ファイル群を一般に公開したい場合

- Package化 (wheel, sdist, 構造化)
- マニュアル・ドキュメントの整備
- チュートリアルの整備
- インストーラー
- ユーザーサポート体制
- twineなどのuploadコマンドのチェック



リポジトリサイトの利用

- PyPI (Python Package Index)
- Github, Gitlab

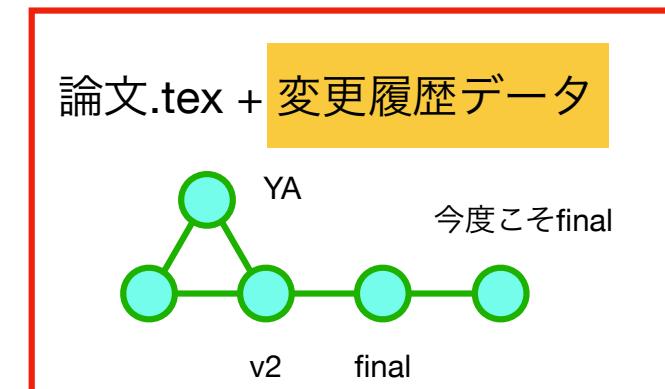


バージョン管理システム

Version Control System (VCS)

- IBMのソフトウェア更新ツールが前身
- コンテンツ履歴, メタデータ履歴, 並列開発の3要素を提供

220308_論文.tex
220308_論文修正_YA.tex
220309_論文修正v2.tex
220310_論文修正final.tex
220310_論文修正こんどこそfinal.tex



タイトルだけで管理は混乱の元

構造化された変更履歴データベース（リポジトリ）

VCSの例 CVS(Concurrent Versions System), subversion, mercurial, **git**

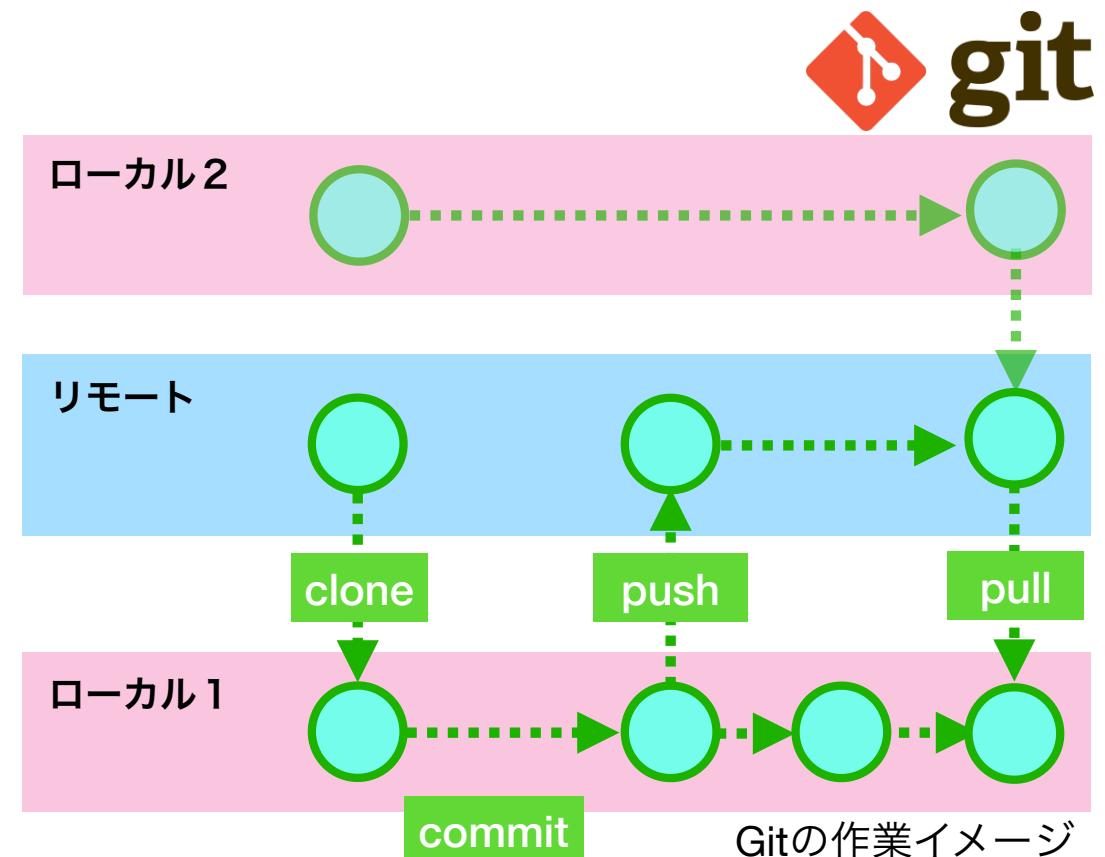
- ほとんどの情報系ではgitによるソースコード管理を行っている
- リモートリポジトリサイトの発展 (github, gitlab)

git

- Linux カーネルの管理目的で2005年に公開
- 分散型：リモートサーバにある中央リポジトリをローカルにコピーして作業を行う

Gitの基本作業

1. ローカルリポジトリの作成
(init, clone)
2. ローカルリポジトリの更新
(commit)
3. リモートリポジトリの更新
(push)
4. リモートの更新をローカルに反映
(pull = fetch+merge)



ホスティングサービス

Windows

Mac

Linux



GitHub

- 2018よりmicrosoft傘下の業界大手
- Pull requestなどの付加機能がある
- URL: <https://github.com/>



GitLab

- オープンソースのGitリポジトリ管理ソフトウェア
- 自前サーバに導入できる
- Gitlabを用いたホスティングサービスがgitlab.com
- URL: <https://about.gitlab.com>



Bitbucket

- Atlassian社のサービス（Trelloなども保有）
- MercurialとGitに対応
- GUIとしてsourcetreeを公開
- URL: <https://bitbucket.org>

GUIソフトウェア



GitHub Desktop

- **Price:** Free
- **License:** MIT

Windows

Mac

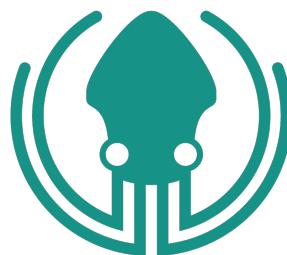


SourceTree

- **Price:** Free
- **License:** Proprietary

Windows

Mac



GitKraken

- **Price:** Free / \$29 / \$49
- **License:** Proprietary

Windows

Mac

Linux

Git のオフィシャルページにリストあり : <https://git-scm.com/downloads/guis>

git init, clone

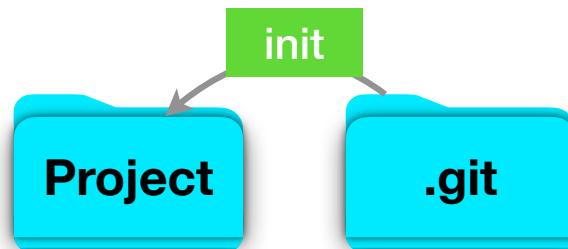
Gitによるプログラム・テキスト管理を始める

自分の作成したプロジェクト

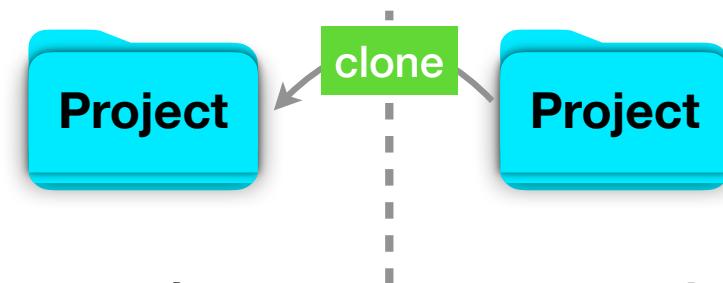
```
$ git init
```

他人の作成したプロジェクトを利用

```
$ git clone [repository]
```



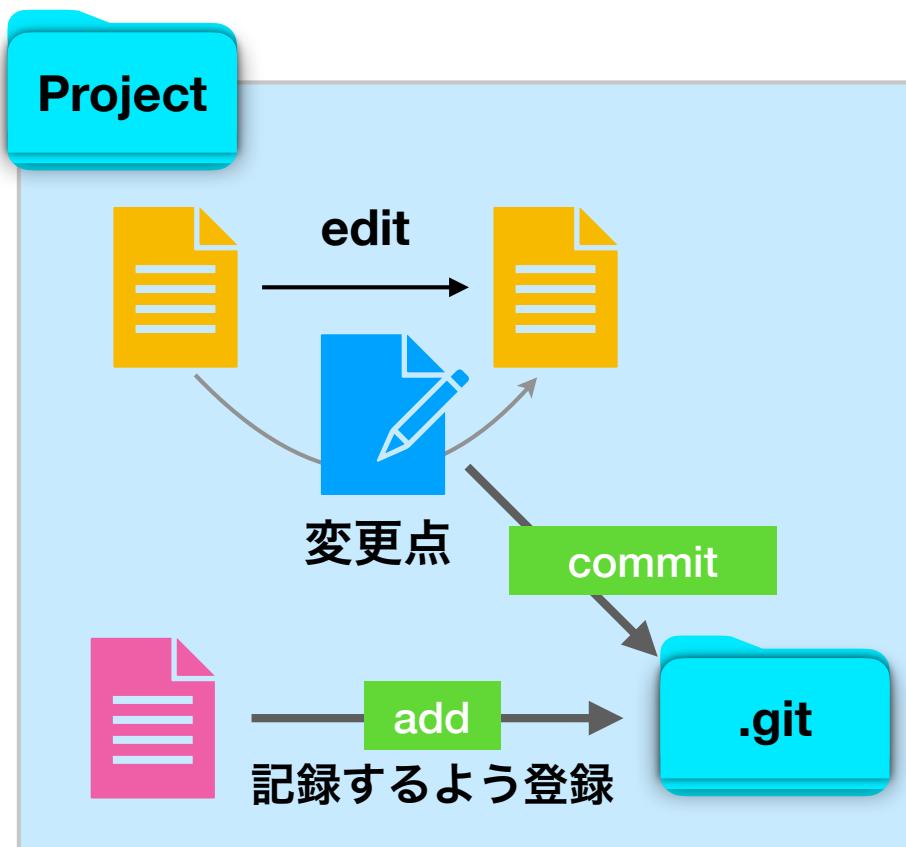
リポジトリの作成
(.gitがproject フォルダ内に生成)



ローカル
.gitを含んだプロジェクトごと
リモートから”複製”
リモート

git commit

プログラム・テキストの更新を記録する



\$ git commit -m “message”

変更点をリポジトリに記録する。変更点は、リポジトリに登録されているファイル全てについて記録される。

\$ git add [file name]

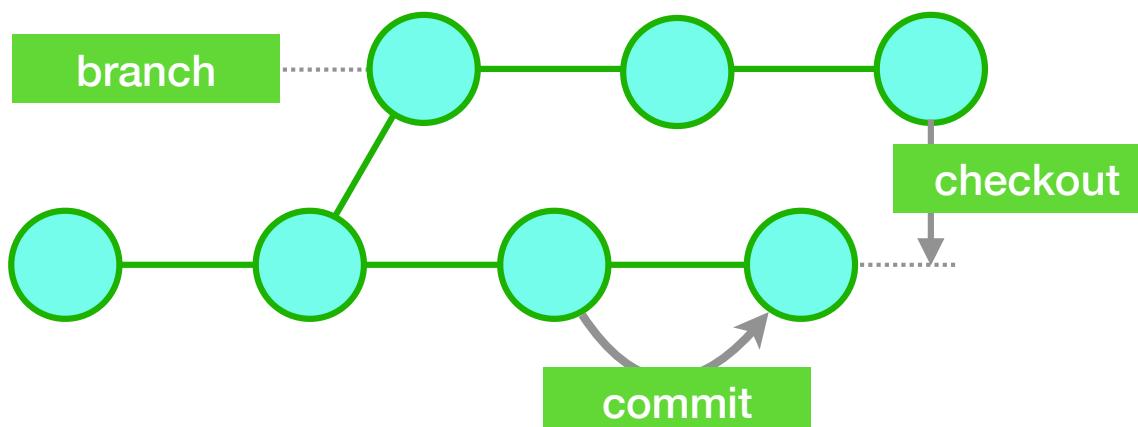
プロジェクトに新たに作成・追加されたファイルは自動では登録対象にならないのでaddコマンドで登録対象にする。

git branch

変更履歴を分岐させる

リポジトリ内の記録：ツリー構造

新しい世界線を生成



状態が生成され
その状態へ移動する

\$ git branch [name]

変更履歴の新しい分岐を作成する。これにより、既存バージョンを保持しながらの開発が可能になる。

\$ git checkout [to_branch]

分岐の間を移動する。

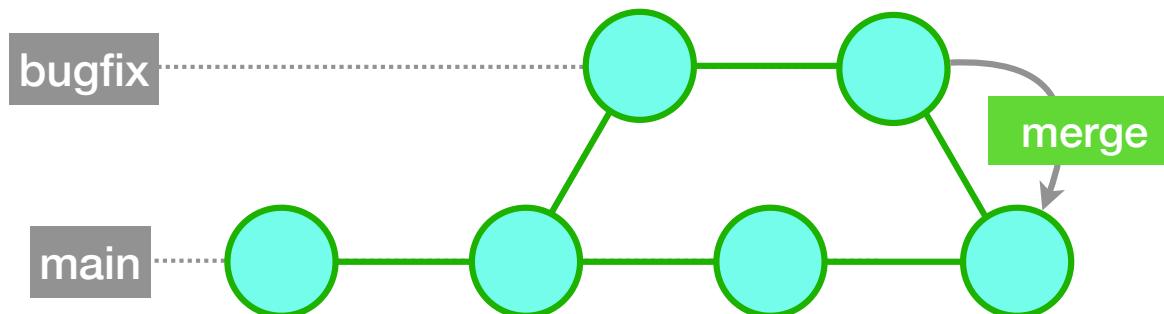
\$ git log --graph

リポジトリ記録のツリー構造を出力する。

git merge

依存関係のない状態を統合する

```
$ git merge [other_branch]
```



他のブランチにある状態を現在の状態と統合する。
もし競合 (conflict) が起きた場合は競合場所
を'<<<<' と '>>>>' で指示してくれる。

Mergeしても元のbranch
は残るので、将来的に利
用可能

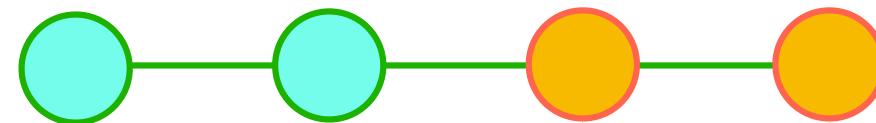
競合が起きた場合は、競
合箇所を確認して修正し
て競合を解消する。その
後mergeし直せば良い。

git push

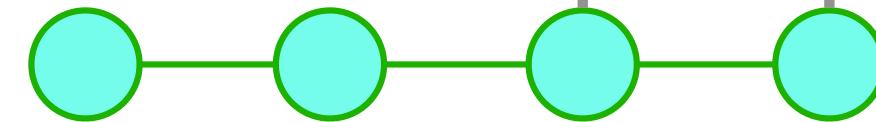
ローカルの状態をリモートに反映させる

```
$ git push origin main
```

リモート



ローカル



ローカルの変更をリモートに送信して状態を追加、
リモートの最新状態も移動させる。

origin

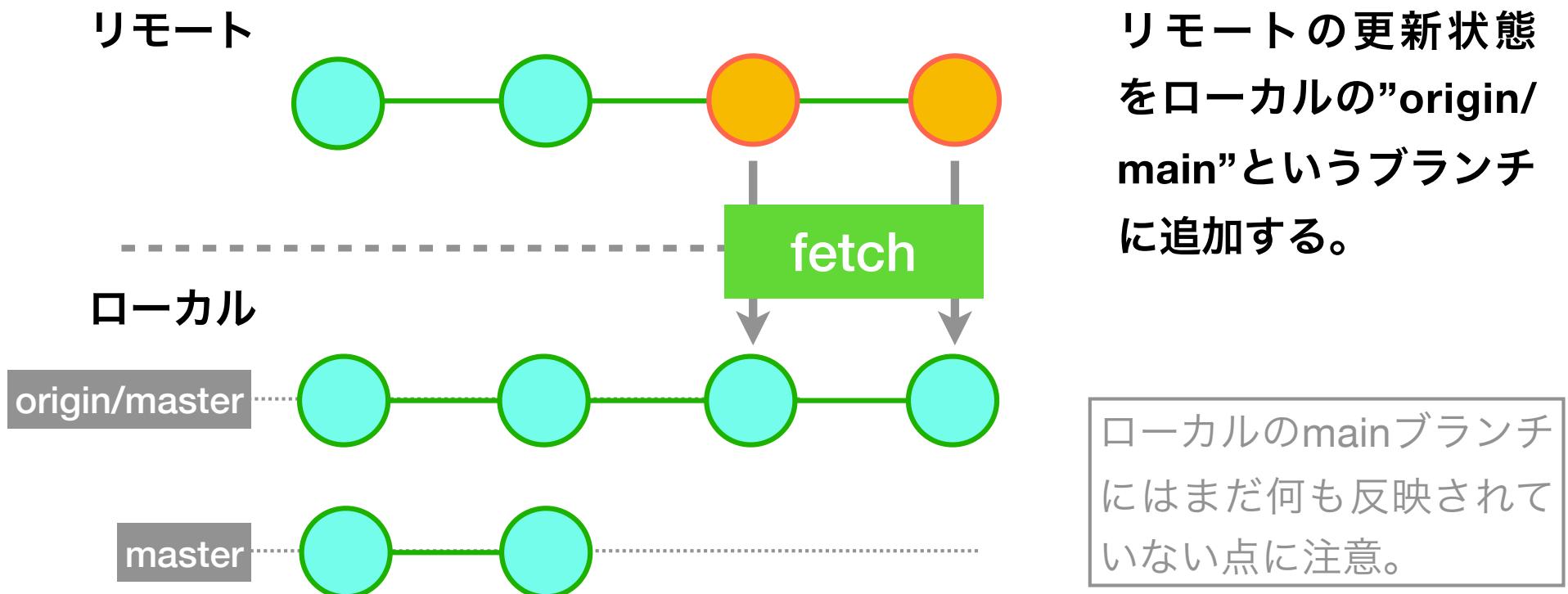
Gitはデフォルトではリモートサーバのことを"origin"と名づける

各々のブランチを指定したpushの仕方もある。
originの後はmain:mainの省略であり、[ローカルブランチ名]:[リモートブランチ名]

git fetch

リモートの状態をローカルにダウンロードする

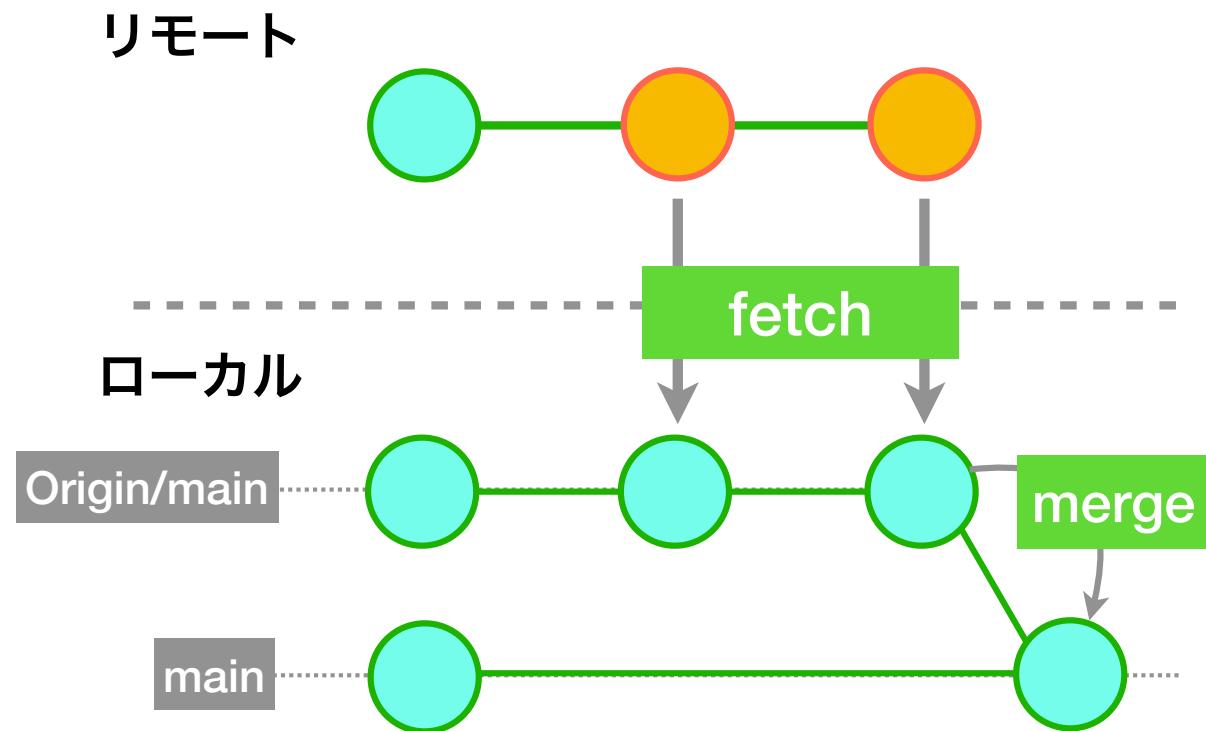
\$ git fetch



git pull

リモートの状態をローカルにダウンロードする

\$ git pull



Fetchしたリモートの更新履歴をそのままmainに自動でmergeする。

コンフリクトが生じた場合、リモートリポジトリをいじるのはリスクがあるので、fetchしてファイルを確認するのが安全。不特定な開発ではfetch+mergeが無難。

Git演習

1. Githubアカウントを作成しましょう

2. Github Desktopをインストールしましょう



[GitHub Desktop](#)

- **Price:** Free
- **License:** MIT

[Windows](#)

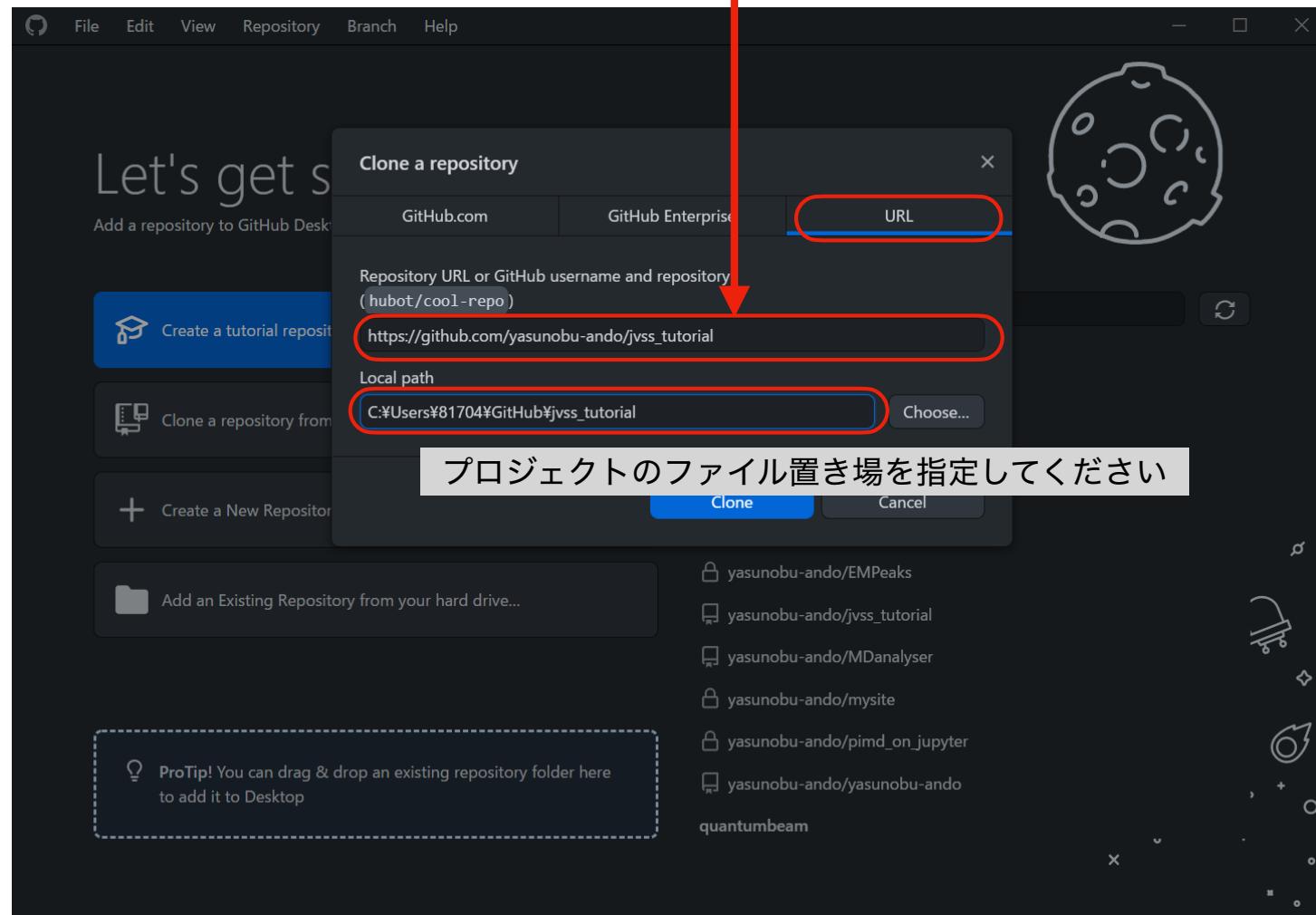
[Mac](#)

(Linux利用者はGitkraken, Atom, VSCode, CUIを活用してください)

Git演習

3. yasunobu-ando/jvss tutorialをcloneしましょう

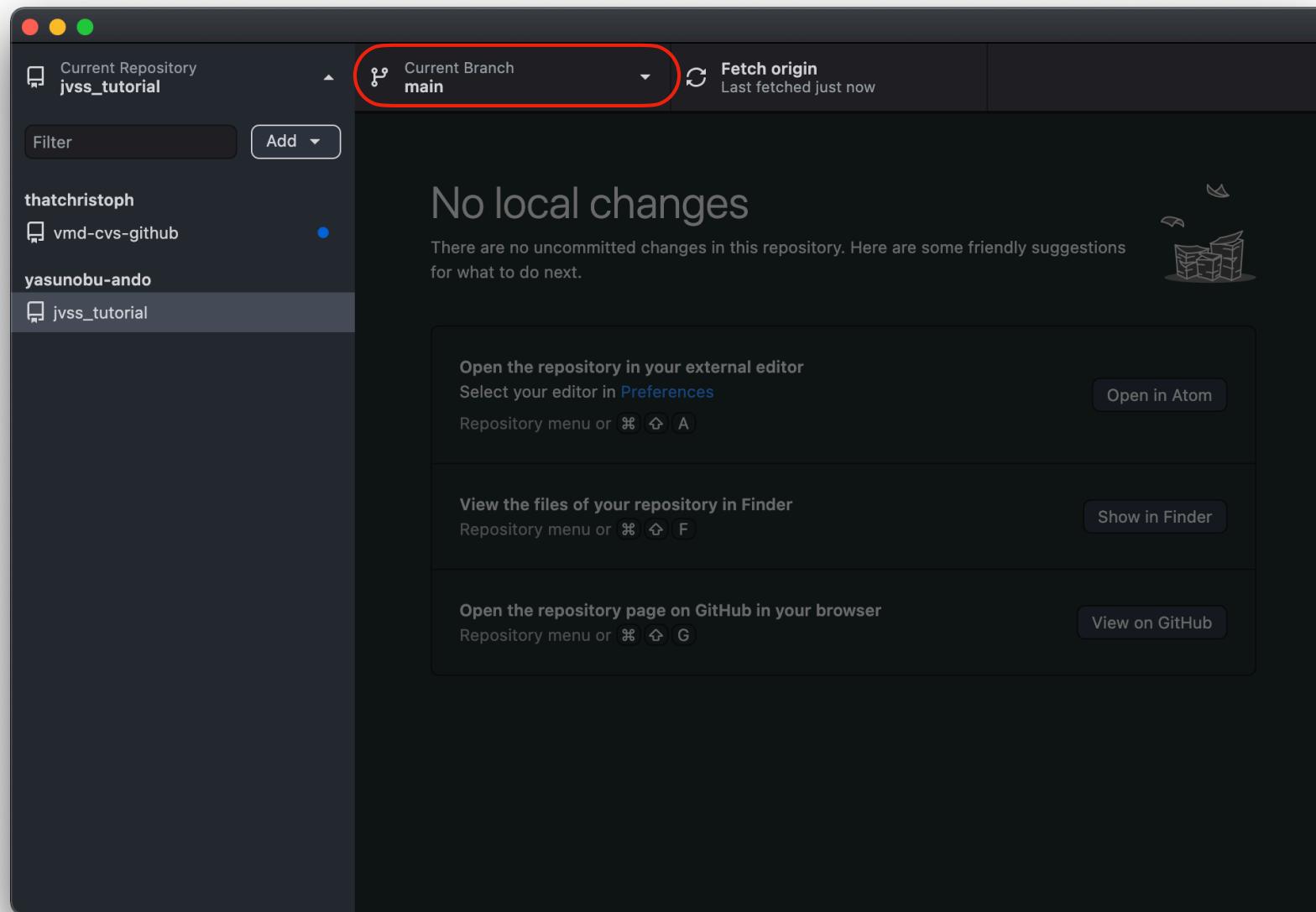
リポジトリサイトのURL: <https://github.com/yasunobu-ando/jvssTutorial>



Menu bar
> File
> Clone Repository
からもこのwindowを開く
ことができます

Git演習

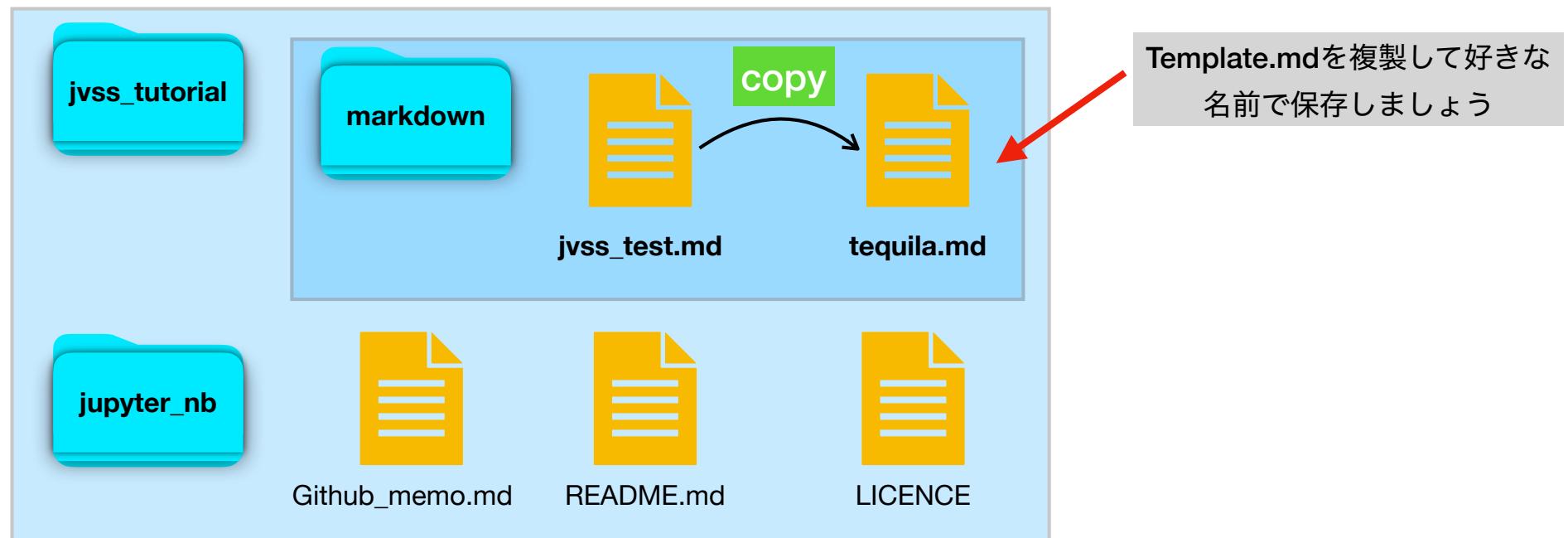
4. ブランチを"220310"に切り替えましょう



Git演習

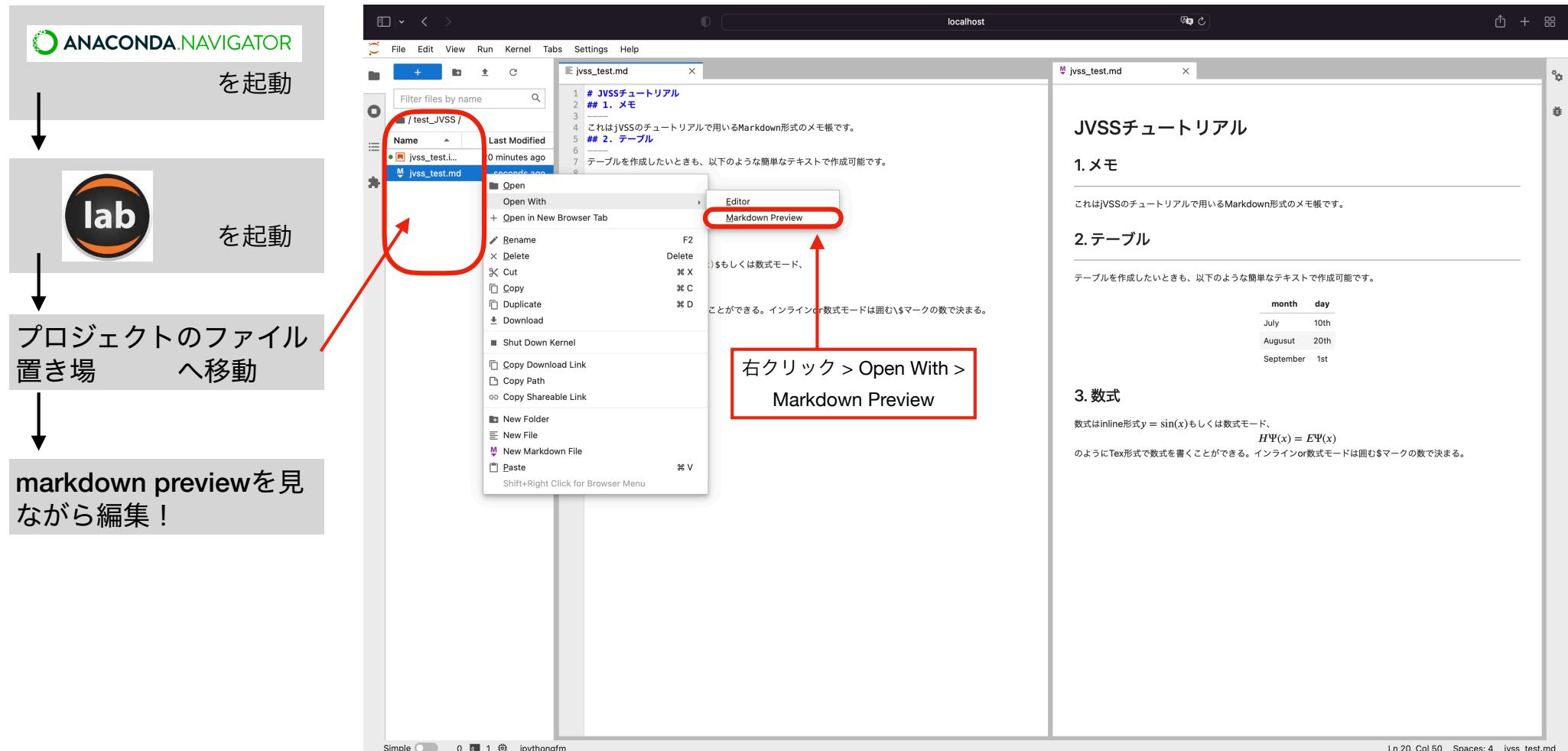
5. マークダウンをコピーして好きな名前で保存しましょう

Cloneした際に設定した **プロジェクトのファイル置き場** に移動します。



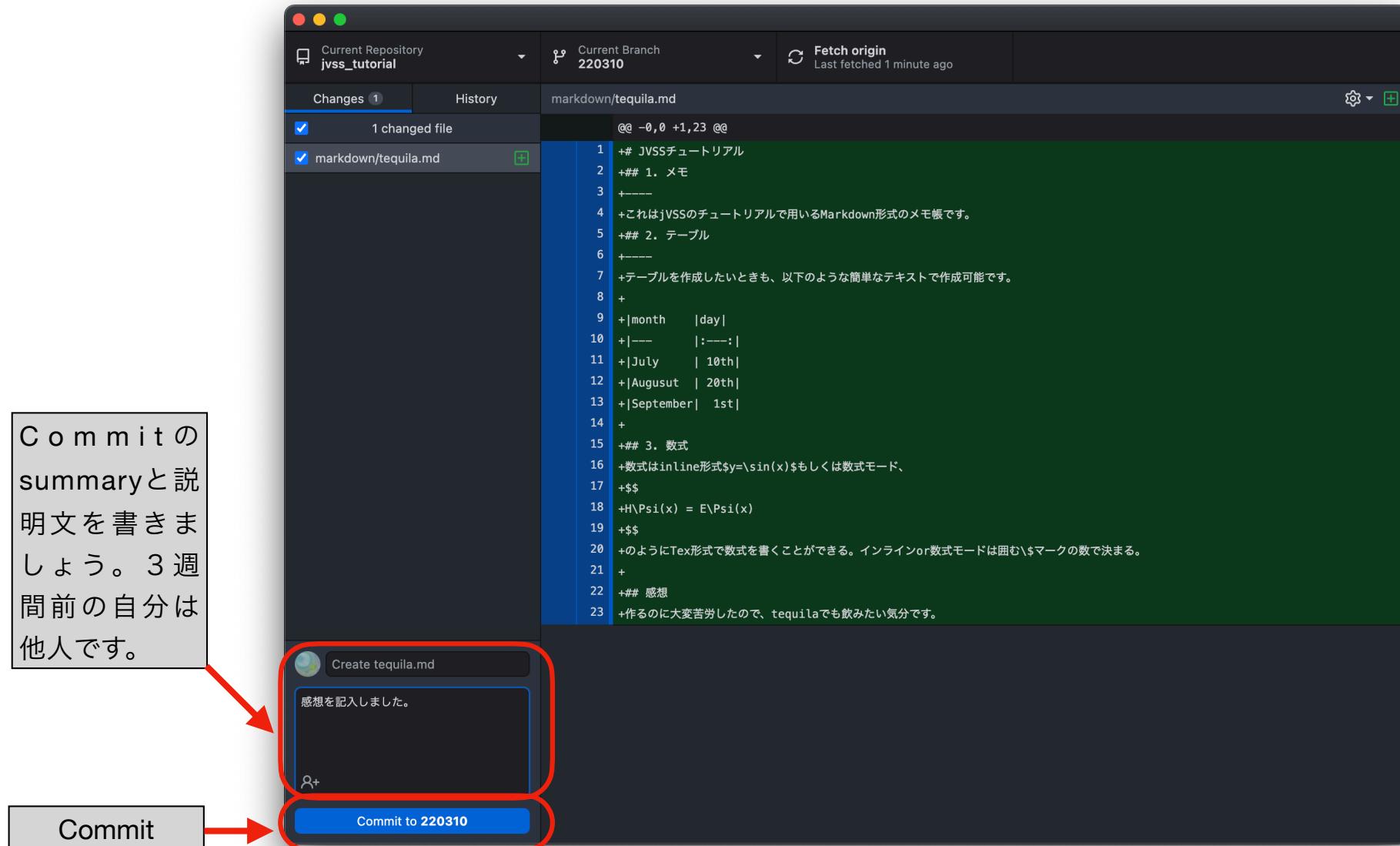
Git演習

6. マークダウンを編集して今日の感想や質問・まとめを記入しましょう



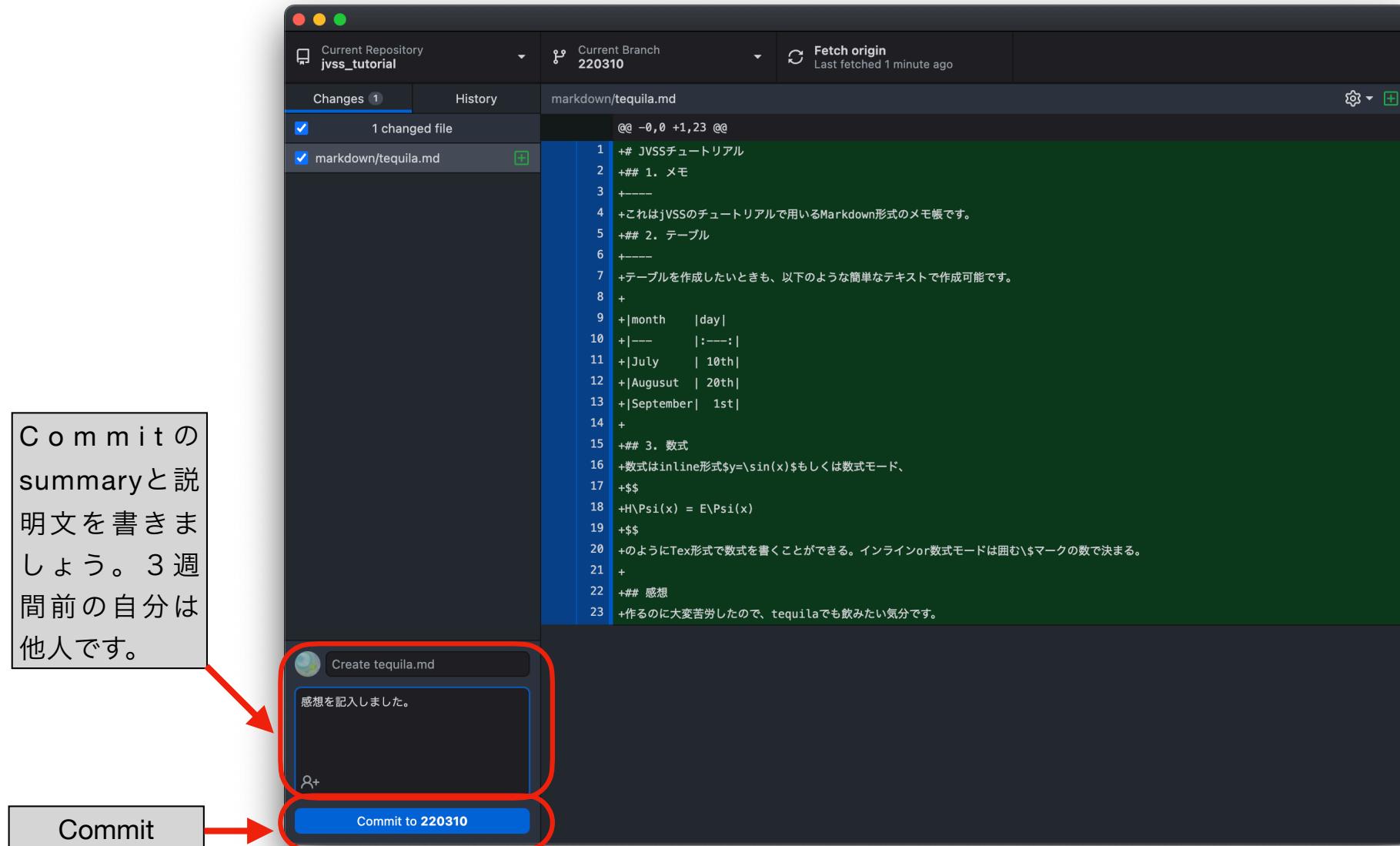
Git演習

7. 変更をcommitしましょう



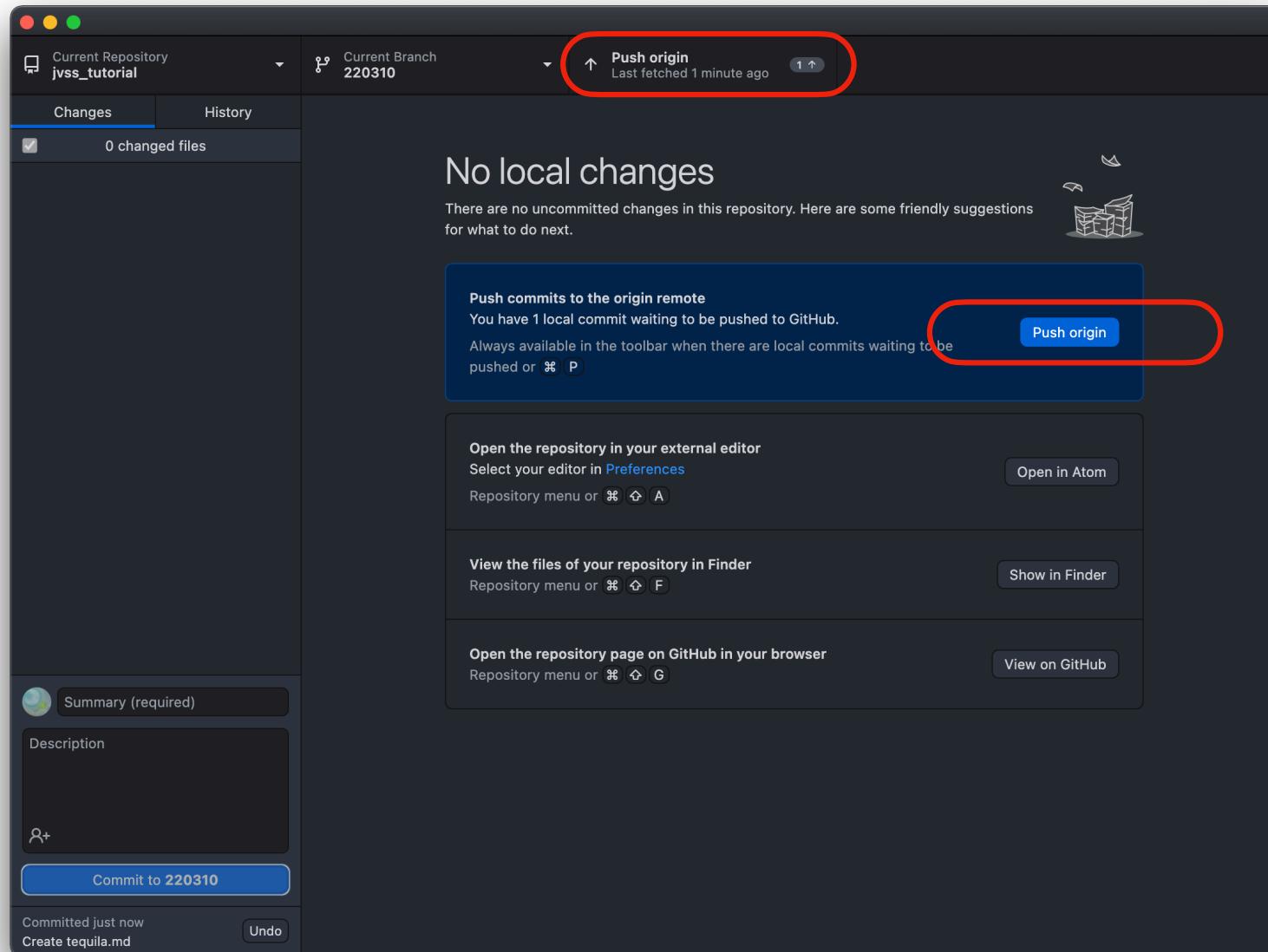
Git演習

7. 変更をcommitしましょう



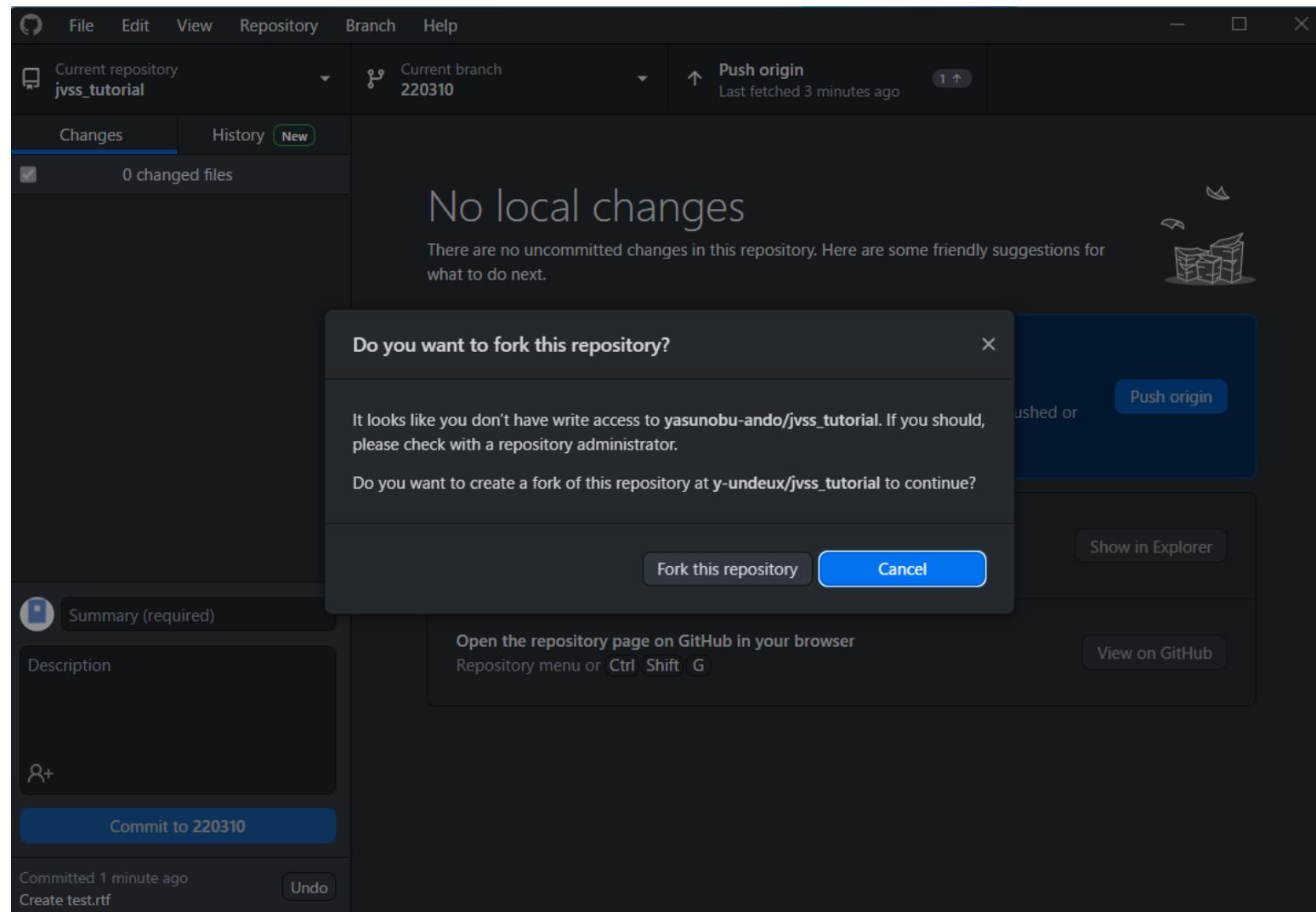
Git演習

8. 変更をpushしましょう



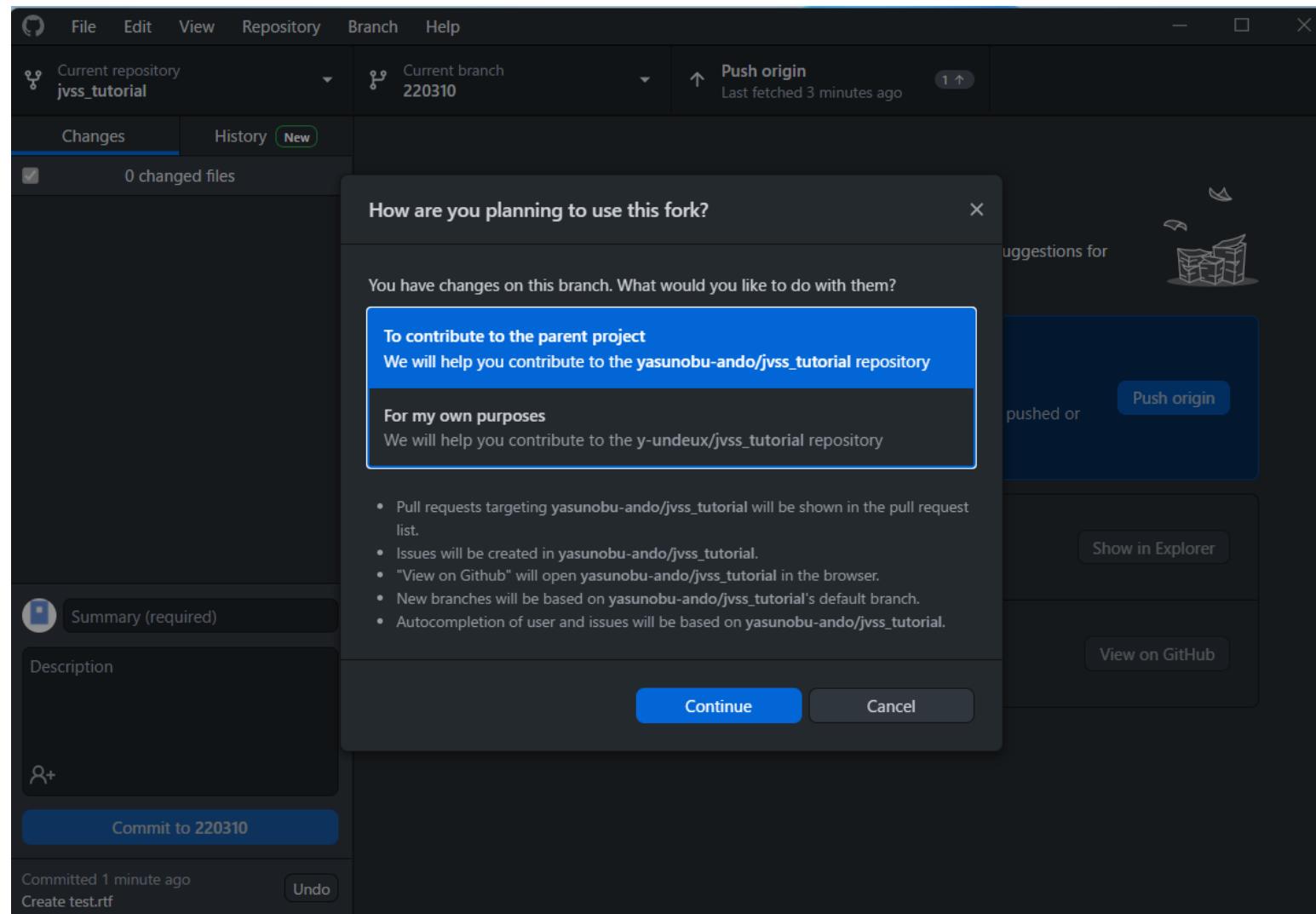
Git演習

9. リモート (origin) に親リポジトリをforkしましょう



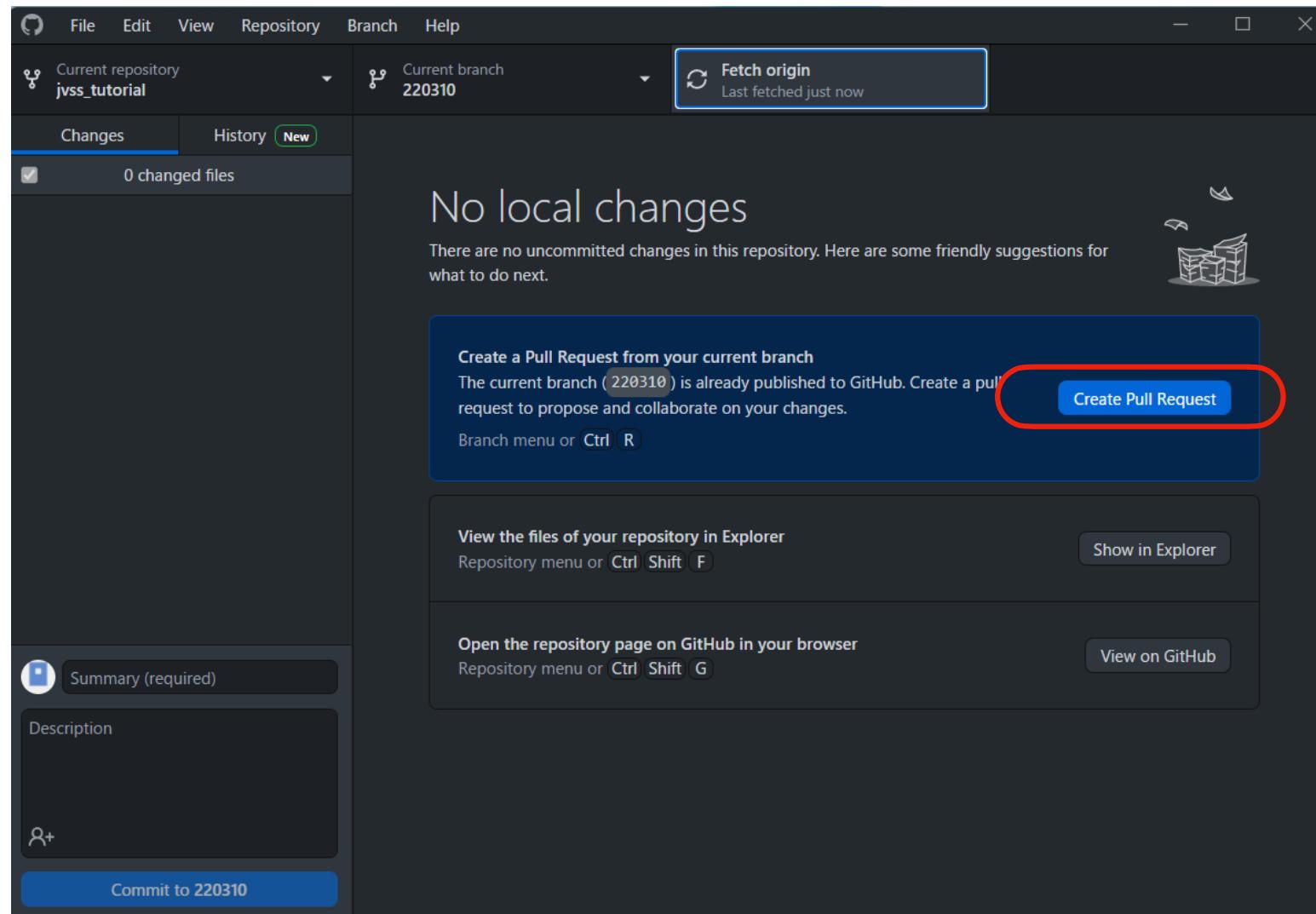
Git演習

10. 今回はコラボを選択して設定しましょう



Git演習

11. Pull Requestをしましょう



Git演習

12. 管理者のreviewを待ちましょう

Git演習

13. 親リポジトリに対してfetch upstreamしてみなさんのファイルを反映させましょう

①GitHub.comに移動

②branchを220310に変更

③Fetch Upstreamを選択

“commpare” or “Fetch and Merge”を選択

The screenshot shows a GitHub repository page for a user named 'y-undeux'. The repository has 2 branches and 0 tags. A red arrow points to the '220310' dropdown menu in the top left, indicating it's selected. Another red arrow points to the 'Code' dropdown menu in the top right, with 'Fetch upstream' highlighted. The main area displays a list of commits:

Author	Commit Message	Date
y-undeux undex	6234fd3 1 hour ago	9 commits
jupyter_nb	separating into folders	12 days ago
markdown	undex	1 hour ago
.DS_Store	ignoring reference	12 days ago
.gitignore	ignoring reference	12 days ago
Github_memo.md	新しいファイルを追加しました	12 days ago
LICENSE	Initial commit	12 days ago
README.md	Initial commit	12 days ago