VSCode & Git/GitHub 入門

スタートアップゼミ 第2回

D3 小川大智
交通・都市・国土学研究室　2026年4月13日

本日の内容（前半）

本日使うツール

課題	ツール
快適な編集環境	**VSCode**
変更履歴の管理	**Git**
チームでの共有	**GitHub**

座学

- **Part 1**: VSCode の基本操作
- **Part 2**: Git / GitHub の基礎

ハンズオン（テキストブックレポジトリを使用）

- fork → clone → branch → push

> 今日のハンズオンで，この3つを実際に動かすところまで体験します

Part 1: VSCode の基本操作

Visual Studio Code

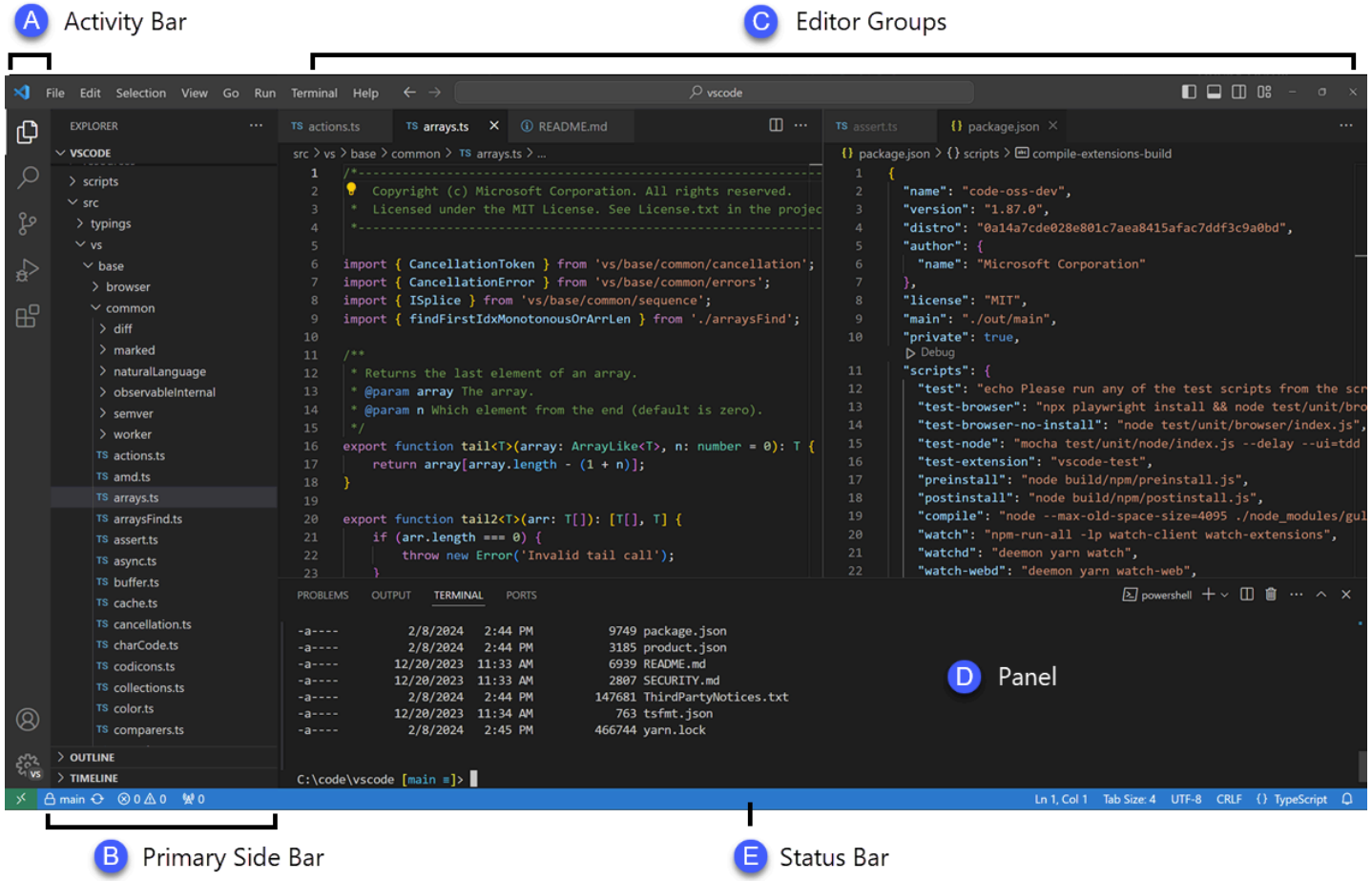
VSCode とは

特徴

- Microsoft 製の無料エディタ
- 軽量・高速・クロスプラットフォーム
- 豊富な拡張機能エコシステム
- Git との統合が標準装備

画面の5つのエリア

A. **アクティビティバー**（左端のアイコン列）
B. **サイドバー**（ファイルツリーなど）
C. **エディタ**（メインの編集領域）
D. **パネル**（ターミナル・問題タブ）
E. **ステータスバー**（下端の情報表示）
F. **メニュー**（Windows では上部の帯,



> https://code.visualstudio.com/docs/getstarted/userinterface

フォルダ・ファイルの開き方

フォルダを開く（基本）

- メニュー：`File > Open Folder`
- ターミナルから：`code .`（カレントディレクトリ「.」を開く）
- Finder / エクスプローラーからドラッグ&ドロップ

ファイルを開く

- サイドバーのエクスプローラーからクリック
- `Cmd+P`（`Ctrl+P`）で名前検索して開く
- メニュー：`File > Open File`

新規ファイル・フォルダの作成

- サイドバー上部のアイコンをクリック 
- またはメニュー：`File > New File`（`Cmd+N` / `Ctrl+N`）

最近使ったものを再度開く

- `File > Open Recent`
- コマンドパレット（`Cmd+Shift+P` / `Ctrl+Shift+P`）→ `Open Recent`

> **ファイル単体ではなくフォルダを開く**のが基本．サイドバーでプロジェクト全体を俯瞰できる．

ワークスペースと設定

ワークスペース（`.code-workspace`）

- 複数フォルダをまとめて管理・フォルダをまたいだ設定が可能 `File > Add Folder to Workspace...`
- ワークスペースファイルをダブルクリックまたは `File > Open Workspace from File` で開く
- フォルダを開いた状態で，コマンドパレットで `> Workspaces: Duplicate Workspace in New Window` で同じフォルダを複数のウィンドウで開くこともできる

```
// .code-workspace の構造
{
  "folders":    [{ "name": "handson", "path": "." }],
  "settings":   { /* ← ここにワークスペース固有の設定を書く */ },
}
```

設定の優先順位：ユーザー設定 ＜ **ワークスペース設定** ＜ フォルダ設定

> ワークスペース設定はリポジトリに含めてチームで共有できる．
> `Cmd+,`（Windows: `Ctrl+,`）の設定画面で「ワークスペース」タブから GUI でも編集可能．

コマンドパレット：最重要ショートカット

> **Cmd+Shift+P（Windows: Ctrl+Shift+P）　→　コマンドパレットを開く**

「名前でコマンドを検索して実行」する万能入口

- `>` を消すとファイル名で検索（Mac: Cmd+P / Windows: Ctrl+P と同じ）
- 例：`format document`, `git commit`, `open settings`

その他の基本ショートカット

機能	Mac	Windows
ファイルをすばやく開く	`Cmd+P`	`Ctrl+P`
エクスプローラーを開く	`Cmd+Shift+E`	`Ctrl+Shift+E`
エディタを左右に分割	`Cmd+\`	`Ctrl+\`
元に戻す / やり直す	`Cmd+Z` / `Cmd+Shift+Z`	`Ctrl+Z` / `Ctrl+Y`

機能	Mac	Windows
設定を開く	`Cmd+,`	`Ctrl+,`
保存	`Cmd+S`	`Ctrl+S`
現在のタブを閉じる	`Cmd+W`	`Ctrl+W`
サイドバーの表示/非表示	`Cmd+B`	`Ctrl+B`

編集を効率化するショートカット

カーソル操作

機能	Mac	Windows
マルチカーソル	`Opt+クリック`	`Alt+クリック`
同じ単語を選択	`Cmd+D`	`Ctrl+D`
行を上下に移動	`Opt+↑ / ↓`	`Alt+↑ / ↓`
行を下にコピー	`Shift+Opt+↓`	`Shift+Alt+↓`

検索・置換

機能	Mac	Windows
ファイル内検索	`Cmd+F`	`Ctrl+F`
ファイル内置換	`Opt+Cmd+F`	`Ctrl+H`
全体を検索	`Cmd+Shift+F`	`Ctrl+Shift+F`

その他

機能	Mac	Windows
行コメント切り替え	`Cmd+/`	`Ctrl+/`
行を削除	`Shift+Cmd+K`	`Ctrl+Shift+K`
Markdown プレビュー	`Cmd+Shift+V`	`Ctrl+Shift+V`
全て選択	`Cmd+A`	`Ctrl+A`
補完候補を呼び出す	`Ctrl+Space`	`Ctrl+Space`
補完候補を確定	`Tab`	`Tab`

ターミナルの活用

VSCode にはターミナルが内蔵されている．エディタと行き来せず作業できる．

- `Cmd+`` （Windows: `Ctrl+``）でトグル
- パネル右上の `+` ボタンで複数のターミナルを並列起動
- **シェルの種類**：zsh（Mac）/ PowerShell / bash など選択可能

ターミナルを使う際は **カレントディレクトリ** を確認する習慣を．`pwd` コマンドで現在地を表示できる．

拡張機能（Extensions）

主な拡張機能と用途

`Cmd+Shift+X`（`Ctrl+Shift+X`）で拡張機能パネルを開く

用途	拡張機能名（発行元）	主な機能
Python	**Python**（Microsoft）	IntelliSense・lint・デバッグ・仮想環境管理
Python	**Pylance**（Microsoft）	高速な型推論・補完（Python 拡張と同時導入）
Python	**Jupyter**（Microsoft）	Notebook の作成・実行・可視化
LaTeX	**LaTeX Workshop**（James Yu）	ビルド・PDF プレビュー・SyncTeX・補完
Markdown	**Markdown All in One**（Yu Zhang）	プレビュー・ショートカット・目次自動生成
SSH	**Remote - SSH**（Microsoft）	SSH 経由でリモートサーバーを直接編集
Copilot	**GitHub Copilot Chat**（GitHub）	チャットでコードの説明・修正・質問

> Copilot は GitHub Education で学生無料.

詳細は公式ドキュメント → https://code.visualstudio.com/docs

Part 2: Git / GitHub の基礎

バージョン管理とリモートリポジトリ

バージョン管理とは

いろいろな作業をしていると「昨日の状態に戻したい」「誰かの変更が自分の変更を上書き」が頻発する.
Git はこれを解決する.

Git なし（よくある状況）

```
thesis_draft.tex
thesis_draft_v2.tex
thesis_draft_v2_修正.tex
thesis_draft_v3_final.tex
thesis_draft_v3_final2.tex  ← ?
```

- どれが最新か分からない
- 複数人での作業が混沌とする

Git あり

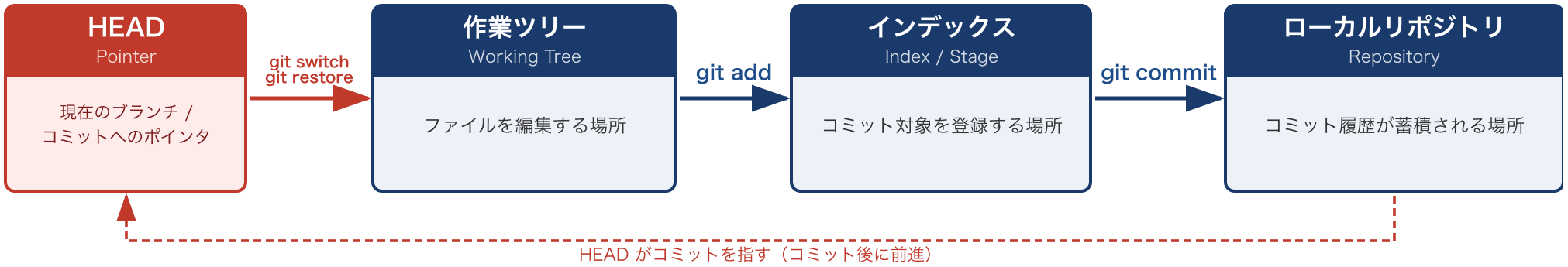
```
thesis_draft.tex  ← 常にこの1ファイル
```

履歴として管理：

```
2026-04-01  "初稿作成"
2026-04-05  "2章追記"
2026-04-08  "図を追加"
2026-04-09  "査読コメント反映"  ← 今ここ
```

- いつでどこを変更したかが明確
- 複数人で並行して作業しても衝突を管理できる

Git の4つのエリア



- **HEAD**：現在作業中のブランチ（またはコミット）を指す特殊なポインタ.
- **作業ツリー**：実際にファイルが存在する場所. エディタで編集するのは常にここ. 変更はまだ Git に記録されておらず「未管理の状態」.
- **インデックス**：次のコミットに含める変更を一時的に登録しておく領域. 複数ファイルを変更した場合も，選択した変更だけをコミットに含められるため，履歴を意味のある単位で管理できる.
- **ローカルリポジトリ**：コミット（変更のスナップショット）が時系列で蓄積される場所. 各コミットは固有のハッシュ値を持ち，過去の任意の状態に戻ることができる.

注: detached HEAD（ヘッド分離状態） — `git checkout <コミットハッシュ>` のようにブランチではなく特定コミットを直接チェックアウトすると発生. HEAD がブランチを経由せずコミットを直接指している状態のため，この状態でコミットしてもいずれのブランチにも属さず，別のブランチへ移動すると参照が失われる. `git switch -c <新ブランチ名>` で新ブランチを作成して通常状態に戻れる.

基本コマンド：一人で使う流れ

```
# 1. リポジトリを初期化（新しく始めるとき）
git init

# 2. 現在の状態を確認（一番よく使う）
git status

# 3. 変更をステージに追加
git add ファイル名          # 特定のファイルだけ
git add .                  # カレントディレクトリ以下すべて

# 4. コミット（履歴に刻む）
git commit -m "変更内容の説明"

# 5. 履歴を確認
git log --oneline          # 一行表示で見やすい

# 6. 差分を確認
git diff                   # 作業ツリーとインデックスの差分
git diff HEAD              # 最新コミットとの差分
```

Mac：ターミナル（Terminal.app）または VSCode 内蔵ターミナル（zsh） ／ **Windows**：PowerShell または VSCode 内蔵ターミナル（PowerShell / Git Bash）

コミットメッセージの書き方

良いコミットメッセージ ＝ 「なぜこの変更をしたか」が分かる説明

悪い例

```
fix
更新
変更した
いろいろ修正
```

良い例

```
Add demand estimation function with logit model
Fix typo in Section 3.2 reference list
Refactor: extract utility calculation to separate function
```

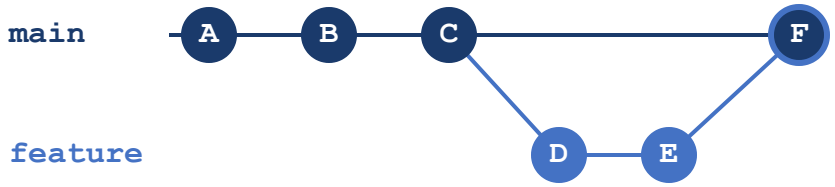
参考：Semantic Commit Messages

- `feat:` 新機能追加
- `fix:` バグ修正
- `refactor:` リファクタリング
- `docs:` ドキュメント変更

ブランチ：変更を安全に試す

ブランチとは

- `main`（主線）から分岐した作業ライン
- 実験的な変更を本線に影響させずに試せる
- 完成したら `main` に合流（マージ）



ブランチ操作

```
# ブランチ一覧を確認
git branch

# 新しいブランチを作って移動 (C -> D)
git switch -c feature/new-analysis

# ブランチを移動
git switch main

# main にマージ (E -> F)
git merge feature/new-analysis
```

ファイルの変更を取り消す

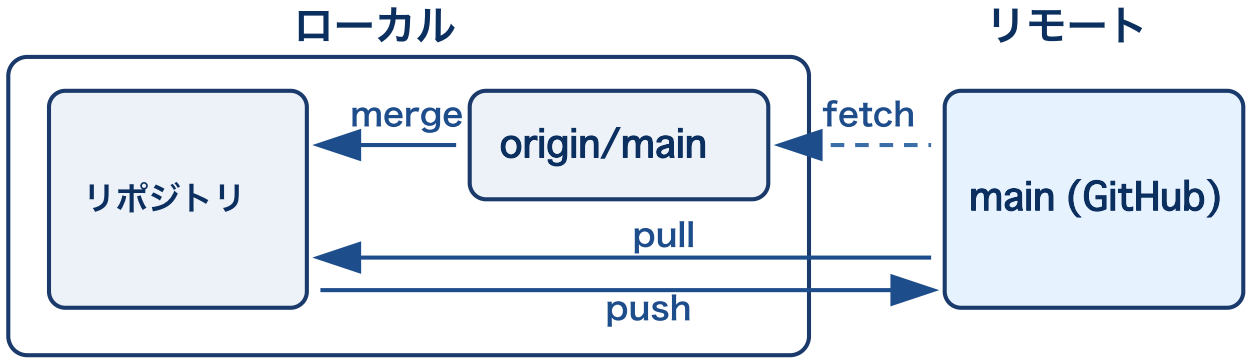
```
# 作業ツリーを最後のコミット状態に戻す
git restore ファイル名

# ステージ済みの変更をアンステージする
git restore --staged ファイル名
```

> `git switch` と `git restore` は，以前は **多機能コマンド** `git checkout` で兼用していた．混乱を避けるため Git 2.23（2019年）以降は用途別の専用コマンドが推奨されている．

GitHub：リモートリポジトリ

GitHub とは

- Git リポジトリをクラウドに置くサービス
- チームで共有・バックアップ
- Web UI でコードをレビューできる




```
# GitHub のリポジトリをローカルに複製
git clone https://github.com/ユーザー名/リポジトリ名.git

# リモートの変更を取り込む
git pull

# ローカルの変更をリモートに送る
git push

# リモートの情報だけ取得（マージしない）
git fetch
```

> **origin** は「このローカルリポジトリが既定で参照するリモート名」．通常は最初に clone した GitHub リポジトリが origin になり，**origin/main** は「origin 上の main ブランチ」を意味する．

.gitignore：追跡しないファイルを指定

追跡したくないファイルの例

- 中間ファイル（`.aux`, `.log`, `.pdf` など）
- 大容量データファイル
- 認証情報・APIキー（**絶対に Git に入れない**）
- OS が自動生成するファイル（`.DS_Store` など）

```
# .gitignore の例
*.aux
*.log
*.pdf
data/raw/
.DS_Store
.env            # 環境変数・APIキーは必ずここに書く
__pycache__/
.venv/
```

APIキーや認証情報を誤って push しないこと. 一度 push したら GitHub に永久に残る（履歴から完全削除は困難）.

VSCode の Source Control パネル

`Ctrl+Shift+G` で Source Control を開く 

GUI でできること

- 変更ファイルの一覧を確認
- ファイルをクリックして差分を表示
- `+` でステージ，`-` でアンステージ
- メッセージ入力後 ✓ でコミット
- `...` メニューから push / pull

インライン差分表示

- **GitLens**（`GitKraken.gitlens`）を導入すると
  各行に「誰がいつ変更したか」を表示できる

> コマンドに不慣れなうちは GUI を使い，慣れたらコマンドへ移行するとよい．

Pull Request（PR）：チームでのコードレビュー

PR の流れ（GitHub 上）

1. 作業ブランチに変更を push
2. GitHub 上で「Compare & pull request」をクリック
3. 変更内容・目的を説明するコメントを書く
4. レビュワー（指導教員・先輩など）がコメント
5. 修正・承認後に `main` へマージ

主な使い方

- 重要なコードや文書の変更前にレビューを依頼
- 「何を変えたか」の記録として残す
- Issues と連携して「#12 を修正」のように参照できる

ハンズオン準備：インストール確認

以下がインストール済みであることを確認してください

ツール	確認コマンド	未インストールの場合
Git	`git --version`	git-scm.com/downloads
VSCode	`code --version`	code.visualstudio.com

Git の初期設定（まだの人のみ）

```
git config --global user.name  "あなたのユーザー名"
git config --global user.email "your-email@example.com"
```

> GitHub アカウントがない方は github.com で作成してください

ハンズオン

Fork & ブランチ push

練習ガイド

対象リポジトリ

stone-ship

作業内容

1. stone-ship を自分のアカウントに **Fork**
2. Fork したリポジトリを `git clone`
3. 本家リポジトリを登録
4. `columns/自分の名前` ブランチを作成
5. ファイルを編集 → `git add` → `git commit`
6. `git push origin columns/自分の名前` で push

担当コラムを編集しよう

各自が担当するコラムファイルを開き，内容を編集してみても良いです．

#	担当者	ファイルパス
1	冨矢	`src/chapters/assignment/col5_miniresearch.tex`
2	鷲野	`src/chapters/assignment/col9_miniresearch.tex`
3	石河	`src/chapters/behavior_modeling/ch2_column.tex`
4	小西	`src/chapters/behavior_modeling/ch3_column.tex`
5	饗場	`src/chapters/behavior_modeling/ch4_column.tex`
6	桑本	`src/chapters/optimization/ch_2_column.tex`
7	名嘉山	`src/chapters/optimization/ch_5_column.tex`
8	永井	`src/chapters/optimization/ch_5_2_column.tex`
9	李	`src/chapters/machine_learning/machine_learning_column.tex`
10	長田	`src/chapters/machine_learning/reinforcement_learning_column.tex`
11	三木	`src/chapters/machine_learning/generative_model_column.tex`

> Pushまで行ったら，GitHub上でファイルが更新されていることを確認してみましょう．

【Windows】ターミナルで `conda` / `python` が見つからない

発生原因

`conda init powershell` はPATHの追加をPowerShell の**プロファイルファイル**（`$PROFILE`）に書き込む.

Windows のデフォルト実行ポリシー `Restricted` はスクリプトファイルの実行を禁止するため，プロファイル自体が読み込まれず `conda` が見つからない.

確認方法

```
# 実行ポリシーを確認（Restricted ならNG）
Get-ExecutionPolicy

# プロファイルの内容を確認
cat $PROFILE
```

解消方法

```
# ① 実行ポリシーを変更（一度だけ）
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

# ② conda init を再実行（プロファイルがない場合）
conda init powershell

# ③ VSCode を再起動して新しいターミナルを開く
```

外部の PowerShell では動くのに VSCode 内で動かない場合は同様の原因であることが多い.

本日のまとめ

VSCode

- コマンドパレット（`Cmd+Shift+P` / `Ctrl+Shift+P`）が万能入口
- ターミナル内蔵・Git 統合で作業が一か所に集約
- 拡張機能で Python・LaTeX・SSH・Copilot 等の環境を整備

Git

- `status` → `add` → `commit` が基本サイクル
- ブランチで実験・本線を分離
- `.gitignore` で不要ファイルを除外（APIキーは絶対 NG）

GitHub

- `clone` / `push` / `pull` でリモートと同期
- Pull Request でレビュー → マージの文化

> 最初は GUI（VSCode の Source Control）を使いながら，コマンドにも慣れていきましょう．

参考リソース

公式ドキュメント

- VSCode: code.visualstudio.com/docs
- Pro Git（無料書籍・日本語あり）: git-scm.com/book/ja
- GitHub Docs: docs.github.com/ja

チートシート

- VSCode Keyboard Shortcuts: コマンドパレットから `keyboard shortcuts reference` で開く
- Git Cheat Sheet: GitHub Education が公開している PDF

研究室での質問先

- エラーメッセージをそのまま検索 or LLM に質問
- それでもわからない場合は，先輩・教員への Slack / Issue 等で気軽に質問を

お疲れ様でした

次回：Python, LLM, 論文の書き方

質問・フィードバックは GitHub Issues または Slack で