

# 計算物理学II

## 第4回

### GitとGitHubについて + これまでの復習

秋山 進一郎

2025年10月24日

# 授業日の確認

- ・ 全10回

- ・ 第1回：10月3日（金）

- ・ 第2回：10月10日（金）

- ・ 第3回：10月17日（金）

- ・ 第4回：10月24日（金）★

- ・ 第5回：10月31日（金）

- ・ 第6回：11月14日（金）★

- ・ 第7回：11月21日（金）

- ・ 第8回：12月5日（金）★

- ・ 第9回：12月12日（金）

- ・ 第10回：12月19日（金）★

- ・ ★の付いた授業にてレポート課題を配布予定

- ・ 本日（10/24）は自習とします
- ・ 各自でレポート課題に取り組み、11/7までに提出して下さい

# 今日の授業の目標

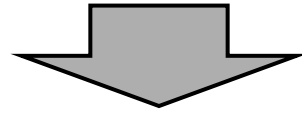
- **これまでの授業の復習**
  - **レポートに取り組める環境は整っているか？**
  - **VS code, Overleafの利用**
- **GitとGitHubについて（やや発展的な話題）**

# 第1回レポート課題について

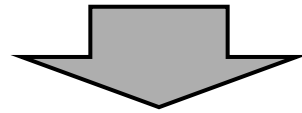
- **lecture4の中に格納されています**
  - **report1.pdf（レポート課題）**
  - **report1ディレクトリ**
    - **report1.pdfのソースファイル**
    - **課題で使う.bibファイル**

# 毎回の授業の流れ

**“compphy2”に移動する**



**“git fetch”と“git merge”で演習資料を入手**



**“lecture\*”に移動し、  
“lecture\_material\_\*.pdf”を開いて演習開始**

# 本日の演習内容

- これまでの復習
  - これまでの演習で終わっていないものを消化する
  - 特に, 第2, 3回の演習課題
    - これらが終わっていると, レポート課題を進めやすいはず
- レポート課題に取り組む
- GitとGitHubについて
  - やや発展的な話題なので, これまでの復習を優先してください
  - もし, コード開発などに興味がある人は, 早い段階でGit/GitHubに習熟されることをお勧めします

# LaTeXが使える環境は整っていますか？

- まだの人は、まずは第3回の演習資料に沿って設定を進めましょう
- Overleafも使えるようにしておくとおもしろでしょう
- ここまでできている人は、レポート課題の内容を確認しましょう
  - report1.pdfを開き、内容を確認する

# レポート課題の進め方

- 第3回の講義で配布したサンプルを使ってレポートを書き始めるのが楽でしょう
  - 例えば, lecture4の中にディレクトリを作り, lecture3/src/main.texをcpする.  
そのディレクトリをVS codeで開くと良い (lecture\_material\_3.pdfのp.22参照)
  - Overleafを使う場合, lecture3/src/template\_overleaf.zipをアップロードする
  - 「LaTeX入門」の1節を参照のこと
- 慣れないうちは難しく感じると思います
  - 授業中, 授業外の時間に, 積極的に周りの人と相談する and/or スタッフやTAを頼る



# 以降の演習

- ・ この時間を使ってレポート課題に取り組んでもらってOKです
- ・ 以降のスライドには、GitとGitHubに関する簡単な演習資料が載せてあります
  - ・ やや発展的な内容なので、興味のある人向け
  - ・ 仮に以降のスライドを読んでいなくても、今後の授業で支障は生じません

# Gitとはバージョン管理システムの一つ

- バージョン管理システムとは？
  - ファイルのバージョン（更新履歴）を管理するためのツール
  - いつ、誰が、何を修正したのか、その履歴を管理してくれる
  - 必要となれば古いバージョンを参照できる
- 以下のような場面で役にたつ、あるいは不可欠なツール
  - 複数人でのドキュメント作成、プログラム開発
  - 一人でも複数の場所（パソコン）でドキュメント作成、プログラム開発をする場合
    - USBメモリでデータを持ち歩く必要がない
  - バグの発見と修正

# バージョン管理システムを使わない場合 ← 非推奨です

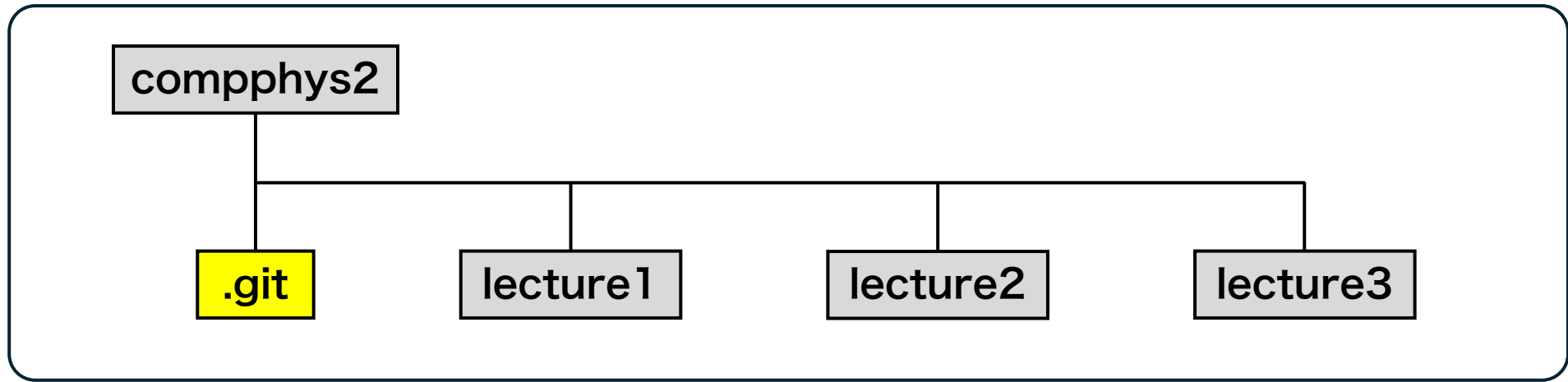
- ・ファイル名でバージョンを管理する（非推奨）
  - ・ファイルを修正することに名前を変えないといけない
  - ・いつ，誰が，何を修正したのか分からなくなる
  - ・どれが最新版なのか分からなくなる

# リポジトリ 1/2

- Gitでは、ファイルの更新履歴をリポジトリ単位で管理する
- リポジトリの実体は、.gitという名前のディレクトリ

例) 授業で使っているcompphys2

- compphys2に入っている全てのファイルがGitの管理下にある



# リポジトリ 2/2

- リポジトリには二種類ある
  - ローカルリポジトリ
    - 自分の手元の計算機，パソコンに存在する
  - リモートリポジトリ
    - リモートサーバ上に存在する
    - よく用いられているリモートリポジトリの一つがGitHub
- Gitでは，ローカルリポジトリとリモートリポジトリを連携して使う

# GitHubを使うための二つの準備

- ① ブラウザからGitHubにアクセスし, Sign upする
  - ・アカウント名とパスワードの設定をする
- ② 手元のTerminalからGitHubへアクセスできるようにする
  - ・SSH公開鍵認証を設定する
    - ・ローカルリポジトリでの修正内容をGitHubへ反映させる場合, その修正を送ってきた人物がリモートリポジトリにアクセスする権利を持っているかを確認するため
    - ・知らない人が勝手に自分のリモートリポジトリに修正を送ってくることを防ぐ

以下, ①と②をやっていきます

# まずはGitHubに登録 1/3

- ・ブラウザから[GitHub](#)(クリック)にアクセスし、アカウントを作ります
  - ・「GitHubに登録する」または「サインアップ」をクリック



# まずはGitHubに登録 2/3

- 右のような画面が現れるので必要事項を入力
  - 入力できたら「Create account」を押す
- しばらくすると、登録したメールアドレスにパスコードが届くので、以下の画面から入力する

## Confirm your email address

We have sent a code to [akiyama.shinichir.ga@u.tsukuba.ac.jp](mailto:akiyama.shinichir.ga@u.tsukuba.ac.jp)

Enter code

Continue

Didn't get your email? [Resend the code](#) or [update your email address](#).

Already have an account? [Sign in](#) →

## Sign up for GitHub

 Continue with Google

 Continue with Apple

or

Email\*

Password\*

Password should be at least 15 characters OR at least 8 characters including a number and a lowercase letter.

Username\*

Username may only contain alphanumeric characters or single hyphens, and cannot begin or end with a hyphen.

Your Country/Region\*

Japan

For compliance reasons, we're required to collect country information to send you occasional updates and announcements.

Email preferences

☐ Receive occasional product updates and announcements

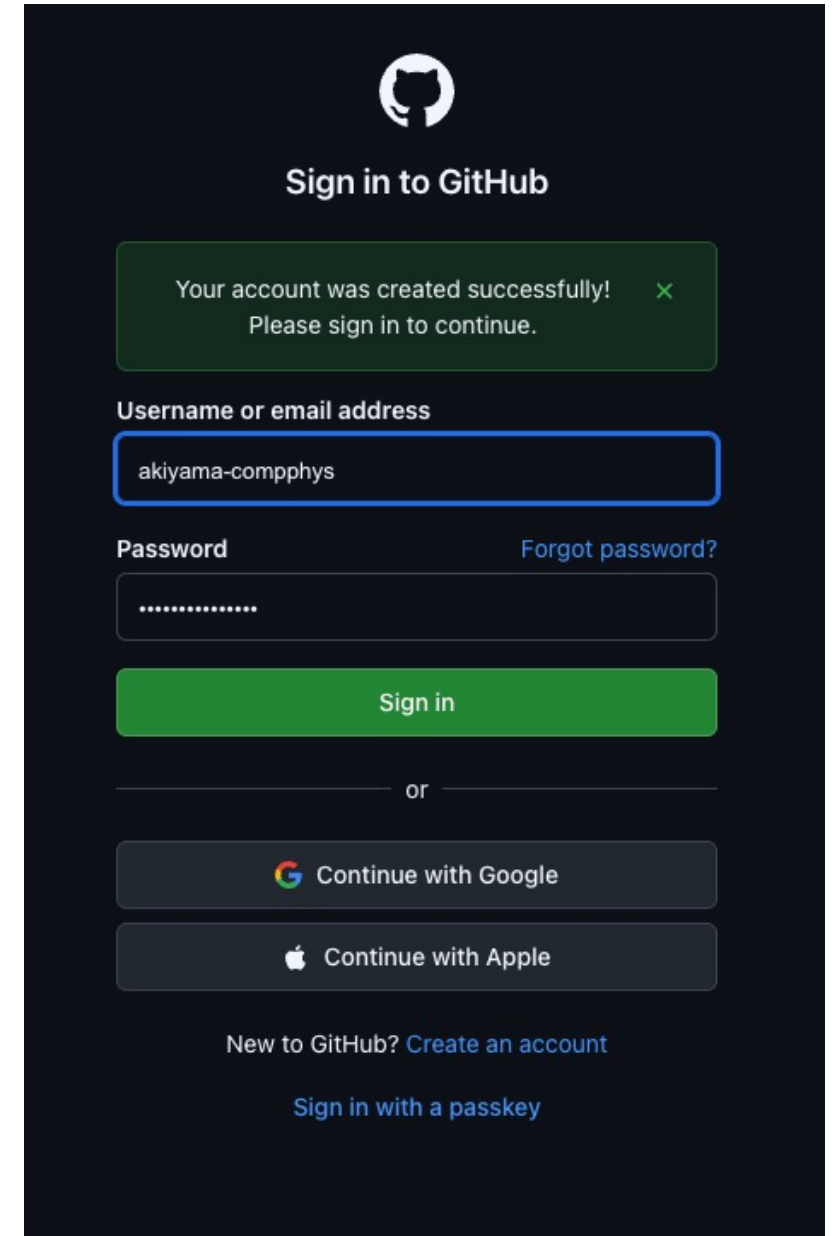
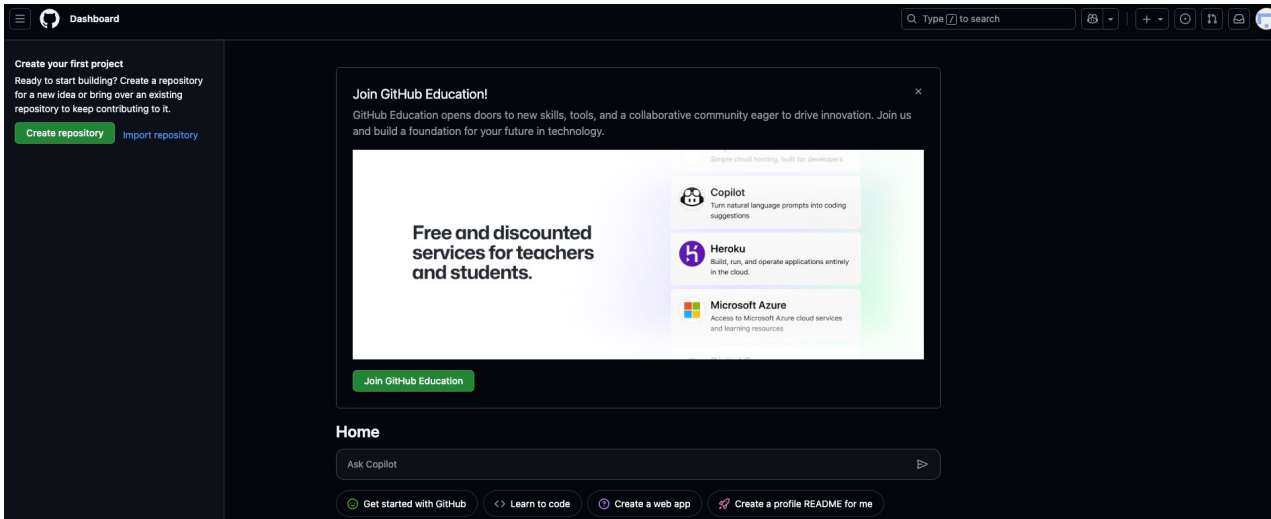
Create account >

By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails.



# まずはGitHubに登録 3/3

- ・ パスコードが認証されると右の画面に遷移する
  - ・ 「Sign in」 をクリック
- ・ 下のよう画面になれば, 登録完了



# SSH公開鍵認証

- 公開鍵と秘密鍵のペアを作る
  - 公開鍵をリモート（今のケースではGitHub）に登録する
  - 秘密鍵はローカルに置いておく
  - 秘密鍵を持っている人だけが、対応する公開鍵の置かれた場所にアクセスできる
  - 秘密鍵は絶対に外部へ公開しないこと

# 公開鍵と秘密鍵の作成 1/4

- 以下のコマンドを入力し, -tまで打ったら, スペースを開けてTab補完を試みよう

```
akiyama@:~/compphys2$ ssh-keygen -t  
dsa          ecdsa          ecdsa-sk      ed25519      ed25519-sk   rsa
```

- 公開鍵と秘密鍵の保存場所を尋ねられているが, ここではこのままホームディレクトリの.sshというディレクトリに鍵のペアを作成する
  - 上の画面の状態でEnterを押す

## 公開鍵と秘密鍵の作成 2/4

- ここでは, ed25519方式を使ってみます
  - ここでもTab補完が効きます
  - 以下のコマンドを実行すると, 次のような二行がterminalに表示されます

```
akiyama@:~/compphys2$ ssh-keygen -t ed25519
```

```
Generating public/private ed25519 key pair.
```

```
Enter file in which to save the key (/home/akiyama/.ssh/id_ed25519):
```

.sshは公開鍵&秘密鍵の保管場所

- 最後の行では, 生成する鍵ペアを保存する場所の確認が要求されています
  - ホーム以下の.sshディレクトリが指定されていると思います
  - .sshディレクトリが指定されていることを確認したらEnterを押しましょう

## 公開鍵と秘密鍵の作成 3/4

- Enterを押すと以下のようにパスワードの設定が求められます
  - パスワードの設定は任意ですが、鍵ペアには必ずパスワードを設定しておきましょう

```
akiyama@:~/compphys2$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/akiyama/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
```

- パスワードを入力したら, Enterを押します

```
akiyama@:~/compphys2$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/akiyama/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

- もう一度同じパスワードの入力が求められます
  - 入力したらEnterを押します

# 公開鍵と秘密鍵の作成 4/4

- 以下のような画面になれば、鍵ペアの生成に成功です

```
Your identification has been saved in /home/akiyama/.ssh/id_ed25519
Your public key has been saved in /home/akiyama/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:9GI6+pm3xBfbKNyqDxm1IrOQ1L+qfLEa5S2zUbeDMVM akiyama@akiyama-Precision-5860-Tower
The key's randomart image is:
+--[ED25519 256]--+
|
| .
| . . E
| . . . + o
| o + 0 S o
| + . * / + =
| . * o 0 B = .
| . . o B * . =
| + + + . = + + .
+-----[SHA256]-----+
```

こちらが秘密鍵の名前

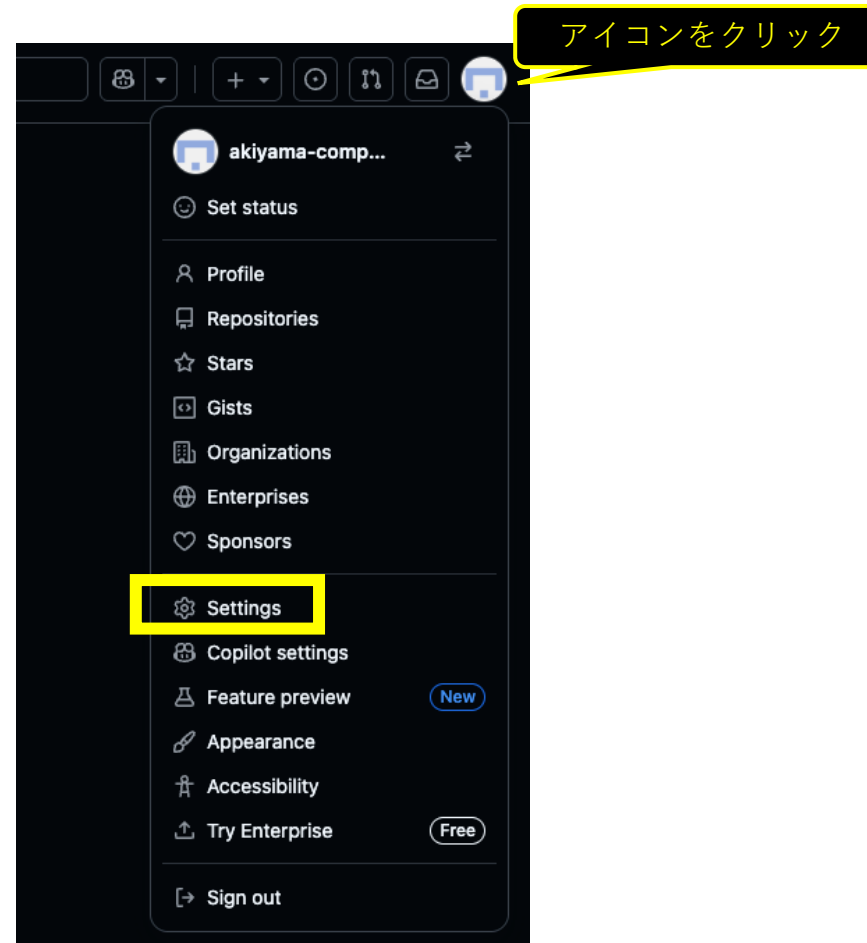
こちらが公開鍵の名前(.pub)

- 次に、作成した公開鍵をGitHubに登録していきます
  - 間違えても秘密鍵は登録してはいけません！

# 公開鍵をGitHubに登録する 1/6

- ・続いて, 以下の手順で公開鍵(.pub)を登録する

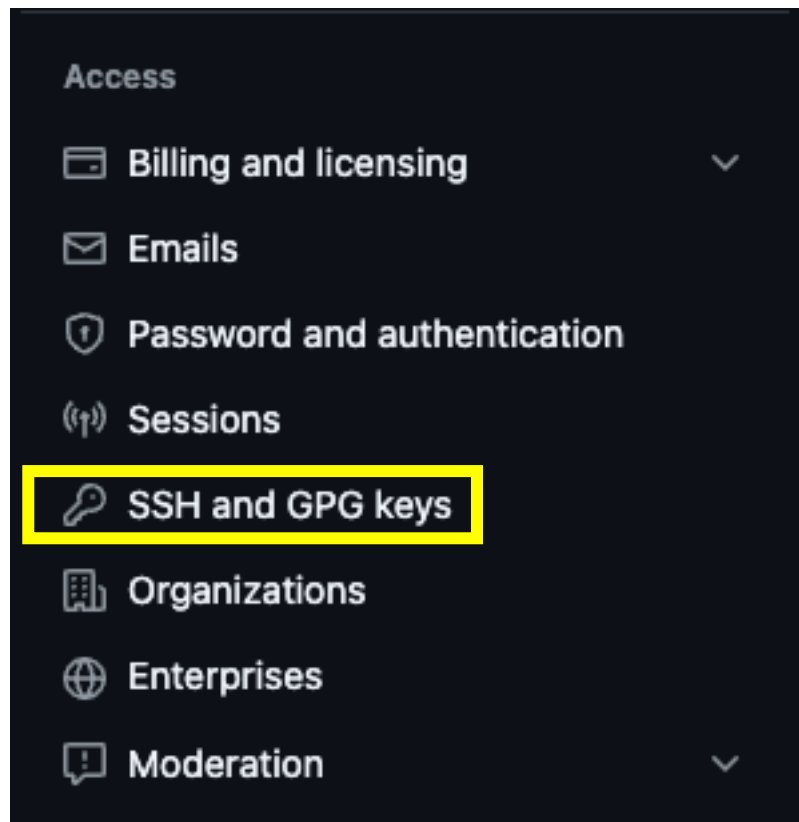
① 右上のアイコンをクリックし, 下の方にある「Settings」をクリック



## 公開鍵をGitHubに登録する 2/6

- ・続いて, 以下の手順で公開鍵(.pub)を登録する

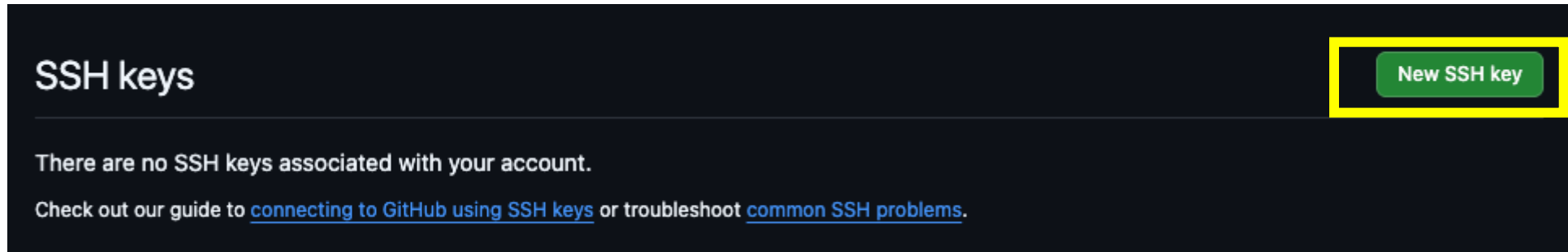
② 左側の一覧から「SSH and GPG keys」をクリック





# 公開鍵をGitHubに登録する 3/6

- ・ 続いて, 以下の手順で公開鍵(.pub)に登録する
- ③ 右上緑色の「New SSH key」をクリック

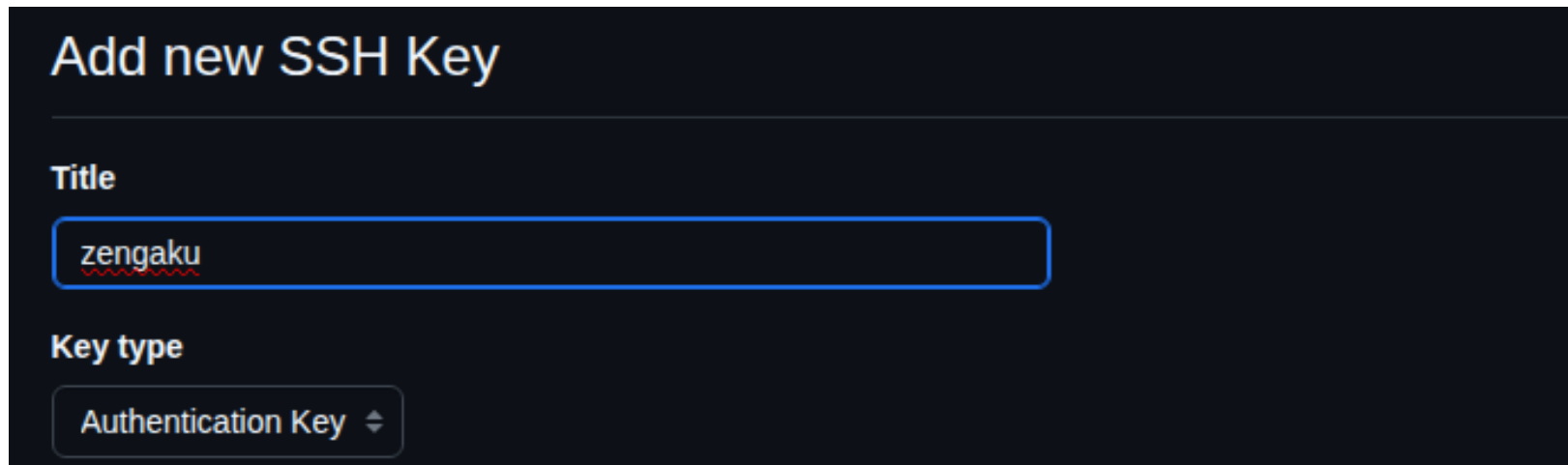


# 公開鍵をGitHubに登録する 4/6

- ・ 続いて, 以下の手順で公開鍵(.pub)を登録する

## ④ Titleを入力する

- ・ Titleはなんでもよいが, どのローカル（計算機）で生成された鍵ペアなのか分かるようにしておくと後々便利だろう
- ・ ここでは「zengaku」として試みました



The screenshot shows the 'Add new SSH Key' interface. The 'Title' field is filled with 'zengaku'. The 'Key type' dropdown menu is open, showing 'Authentication Key' as the selected option.

# 公開鍵をGitHubに登録する 5/6

- ・ 続いて, 以下の手順で公開鍵(.pub)を登録する

## ⑤ Keyを入力する

- ・ Keyには公開鍵の内容を貼り付ける
- ・ 公開鍵の内容を見るには, 自身のTerminalで`cat ~/.ssh/id_ed25519.pub`
- ・ `cat`された公開鍵をCtrl+Shift+Cでコピーし, 以下のKeyにCtrl+Vで貼り付け

.pubの方です!

Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'

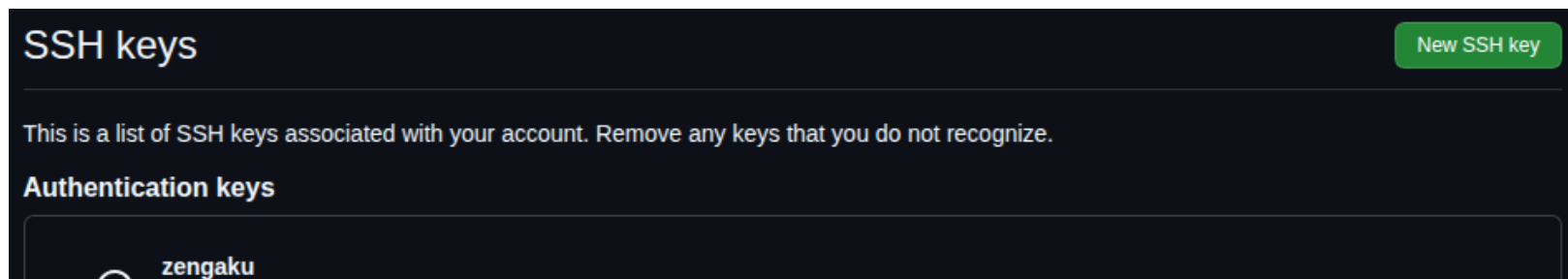
Add SSH key

# 公開鍵をGitHubに登録する 6/6

- ・ 続いて, 以下の手順で公開鍵(.pub)を登録する

⑥ TitleとKeyの入力が終わったら, 緑色の「Add SSH key」を押して登録完了

- ・ 以下のような画面が出れば成功
- ・ Titleに入力した「zengaku」として登録した公開鍵情報が表示されます



# 公開鍵が正しく登録されたことを確認する

- Terminalで以下のコマンドを実行する

```
akiyama@:~/compphys2$ ssh -T git@github.com
```

- もし、以下のようなメッセージが出たらyesと入力してEnterを押す

```
[Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

- もし、Enter passphrase for key ... と出たら公開鍵と秘密鍵の作成時に設定したパスフレーズを入力してEnterを押す
- 以下のメッセージが出ればOK（自分のGitHubのアカウント名が出る）

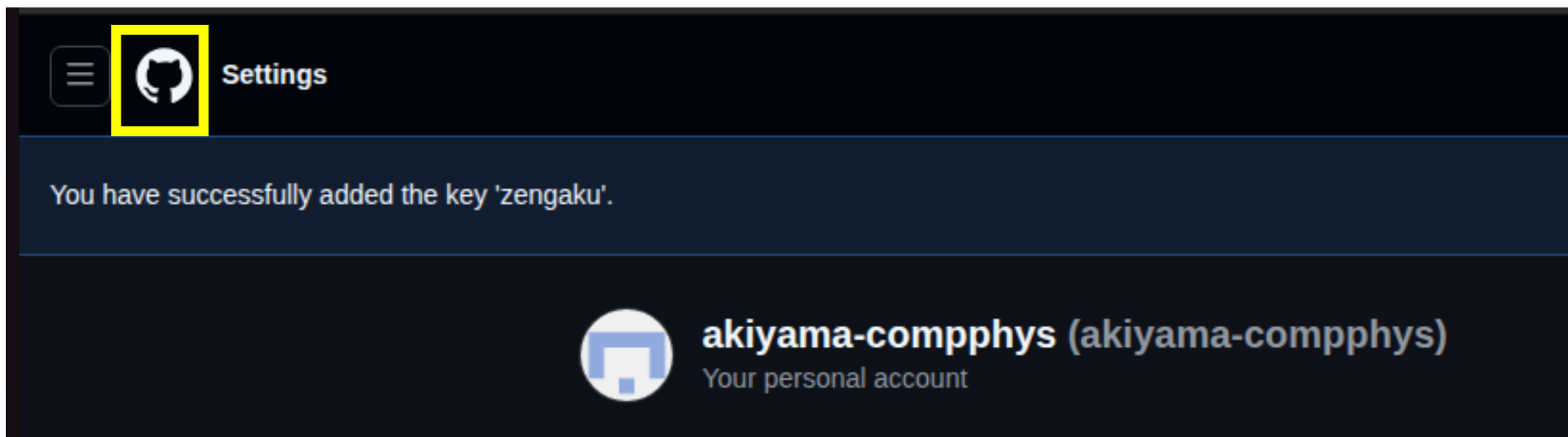
```
Hi akiyama-compphys! You've successfully authenticated, but GitHub does not provide shell access.
```

# リポジトリの作成 1/4

- GitHubでリポジトリを作ります

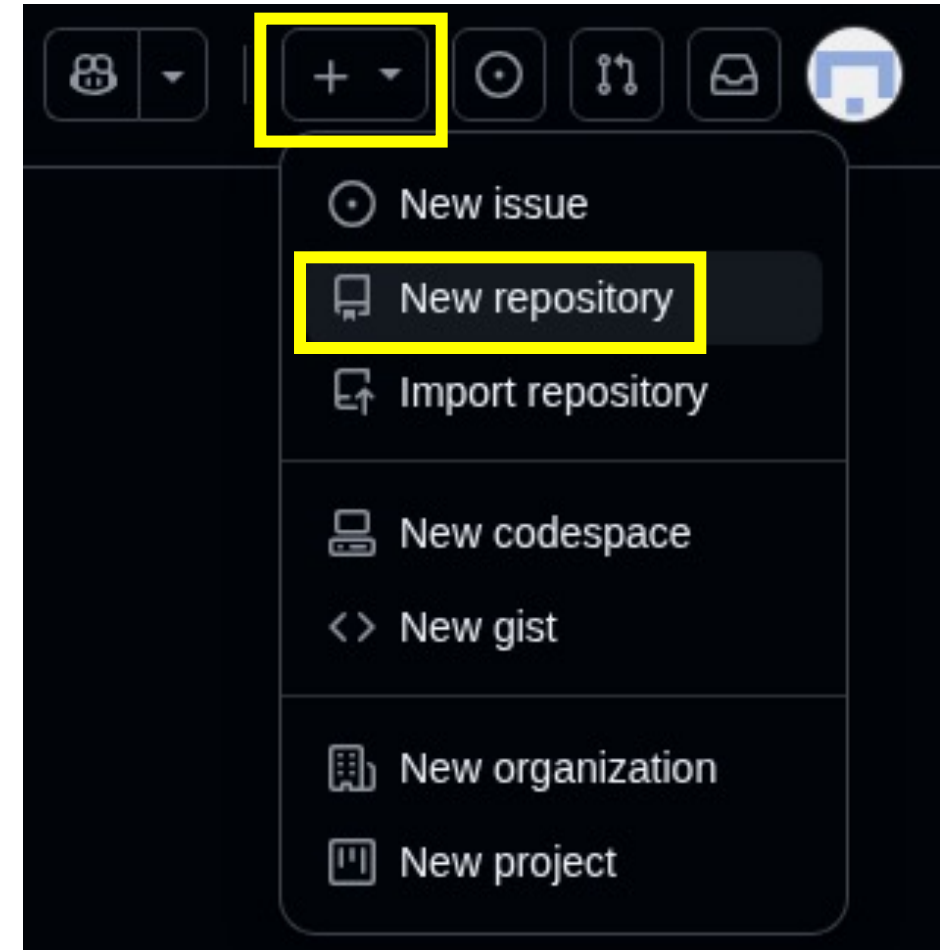
① まずブラウザからGitHubのホーム画面に移動する

- 左上のアイコンをクリック



## リポジトリの作成 2/4

- GitHubでリポジトリを作ります
  - ② 画面の右上の「+」から「New repository」をクリック



# リポジトリの作成 3/4

- GitHubでリポジトリを作ります
- ③ リポジトリの名前(必須)と説明(任意)を入力
  - リポジトリの名前は英数字のみ

## Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository.](#)  
Required fields are marked with an asterisk (\*).

1

General

Owner \*

akiyama-compphys

Repository name \*

test\_GitHub

✓ test\_GitHub is available.

Great repository names are short and memorable. How about [potential-parakeet?](#)

Description

Exercise in compphys2

21 / 350 characters

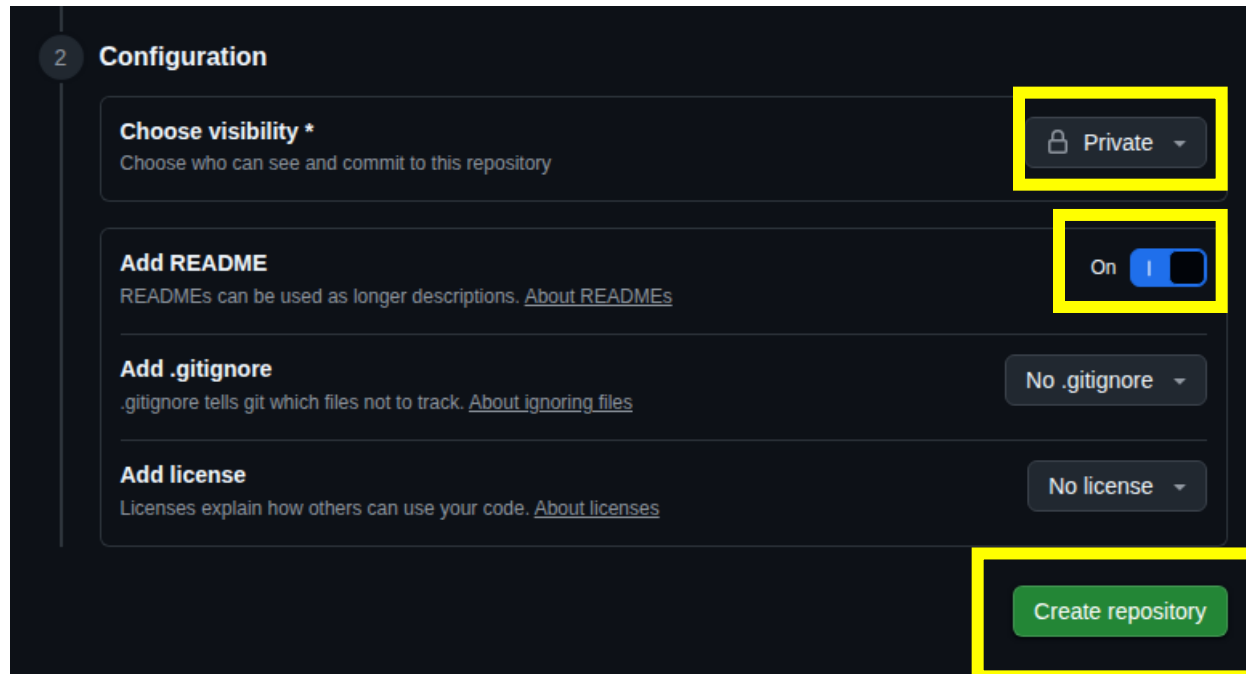


# リポジトリの作成 4/4

- GitHubでリポジトリを作ります

- ④ リポジトリの公開設定, READMEの追加をして, 「Create repository」を押す

- Publicにすると公開(ブラウザから誰でも見れる)
    - Privateにすると非公開(自分だけが見れる)



The screenshot shows the 'Configuration' step (labeled '2') of creating a new repository on GitHub. The interface is dark-themed. Several elements are highlighted with yellow boxes: the 'Private' visibility dropdown, the 'On' toggle for 'Add README', the 'No .gitignore' dropdown, the 'No license' dropdown, and the 'Create repository' button at the bottom right.

**2 Configuration**

**Choose visibility \***  
Choose who can see and commit to this repository

**Add README**  
READMEs can be used as longer descriptions. [About READMEs](#)

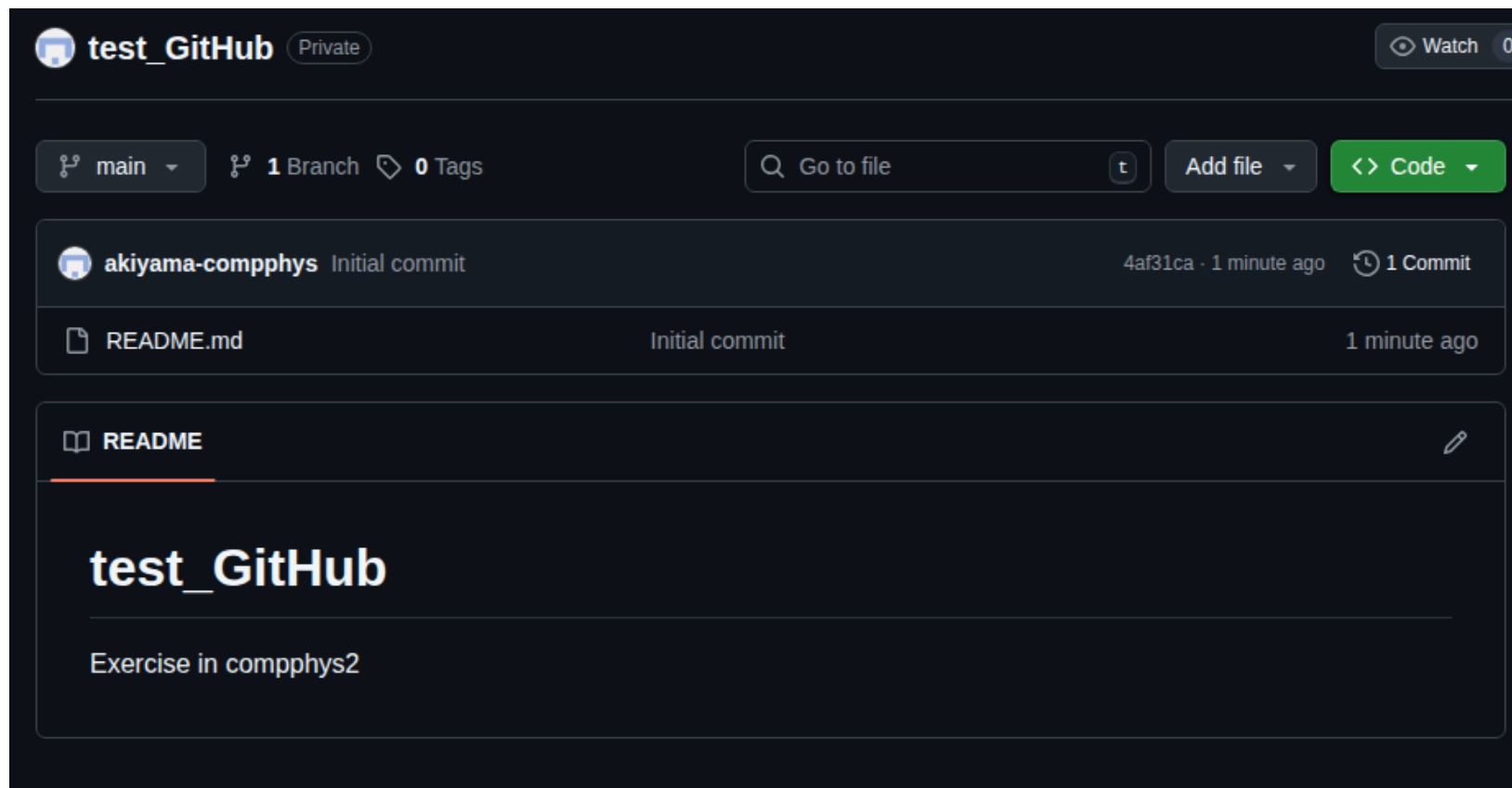
**Add .gitignore**  
.gitignore tells git which files not to track. [About ignoring files](#)

**Add license**  
Licenses explain how others can use your code. [About licenses](#)

**Create repository**

# 作成したリポジトリをローカルにクローンする 1/3

- 先ほどGitHub作成したリポジトリをローカルと連携させます
  - 「Create repository」を押した後, 以下のような画面に移動しているだろう



## 作成したリポジトリをローカルにクローンする 2/3

・先ほどGitHub作成したリポジトリをローカルと連携させます

- ① 「Code」 をクリック
- ② 「SSH」 を選択する
- ③ 右のアイコンからurlをコピーする



## 作成したリポジトリをローカルにクローンする 3/3

- 以下のように、**ホームディレクトリに**リポジトリをクローンしてみよう
- **授業で使っているcompphys2ディレクトリの中ではgit cloneしないように**

```
akiyama@:~$ git clone git@github.com:akiyama-es/test_GitHub.git
Cloning into 'test_GitHub'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
akiyama@:~$
```

- lsコマンドから、GitHubで作成したレポジトリがホームディレクトリ内に存在することを確認しておきましょう
- 以上で、GitHubを使ったリモートとローカルの連携が完了です

# GitHubを使ったコード開発

- コード開発には二種類のスタイルがある
  - チーム開発 ⇒ 複数人でコードを開発する
  - 個人開発 ⇒ 一人でコードを開発する
- いずれの場合でも, GitおよびGitHubを使ったバージョン管理は極めて有効(不可欠)
- チーム開発の場合, 特に重要になるのが「ブランチ」と呼ばれる操作手段である
  - 以下では, 最も素朴な個人開発を前提とし, 「ブランチ」についてはほぼ説明しない
  - 興味のある人は, 以下の文献などを参照してください
    - 渡辺宙志「ゼロから学ぶGit/GitHub 現代的なソフトウェア開発のために」

# ローカル側での初期設定

- Gitに名前とメールアドレスを登録しておく
  - これをしておかないと, ローカルで行なった修正をGitで管理できない
- 以下の二つを実行し, 名前とメールアドレスを登録

```
akiyama@:~$ git config --global user.name "sakiyama"
```

```
akiyama@:~$ git config --global user.email akiyama@ccs.tsukuba.ac.jp
```

- もし, 将来GitHubに公開リポジトリを作ると, ここで登録したメールアドレスも公開されるため注意しましょう
- 今登録した情報は, ~/.gitconfigに保存されています
  - 以下のコマンドから登録内容を確認しておきましょう

```
akiyama@:~$ cat ~/.gitconfig
```

# GitHubによるコード開発の流れ

- GitおよびGitHubを使ったコード開発では、毎回以下の手続きを行う
  - ① 作業を始める前に、ローカルリポジトリを最新の状態にアップデートする
    - git fetch, git merge
  - ② 実際の作業を行う：ファイルの追加, 削除, 編集, ...
  - ③ 作業が終わったら、編集内容をリモートリポジトリへ反映させる
    - git add, git commit, git push
- 以下、このワークフローを体験してみましょう
  - 慣れるまでは面倒に感じると思いますが、慣れるまで使いましょう
  - 実際の個人開発は、以下のp.40~p.49を繰り返すことによって遂行できます

# ローカルを最新の状態にアップデートする 1/2

- 先ほどクローンしたリポジトリに戻しましょう
  - このリポジトリはGitの管理下にあるので、今の状況を以下のコマンドで確認できます

```
akiyama@:~/test_GitHub$ git status
On branch main
Your branch is up-to-date with 'origin/main'.

nothing to commit, working tree clean
```

- 今自分のいる場所には「main」という名前が付いています
- 次頁の手順でリポジトリを最新の状態にします
  - git cloneしてきた直後なので、今の場合に限っては次頁の操作は何の影響も与えません



## ローカルを最新の状態にアップデートする 2/2

- git fetchする

```
akiyama@:~/test_GitHub$ git fetch
```

- 続いてgit mergeする
  - 下のような出力があれば, OKです

```
akiyama@:~/test_GitHub$ git merge  
Already up-to-date.
```

- もし, ローカルがリモートよりも遅れていれば, ローカルが更新される
  - 毎回授業の冒頭で行なっているように, リモートのデータがローカルに反映されます

# 作業用のブランチを切る 1/2

- まず, git branchを実行し, 今いるブランチを確認します

```
akiyama@:~/test_GitHub$ git branch
* main
```

- 続いて, 作業用ブランチを作ります
  - ここでは, 「feature」という名前で新しくブランチを作ります

```
akiyama@:~/test_GitHub$ git branch feature
```

- git branchで再度ブランチを確認しましょう
  - 今切ったばかりのfeatureが現れています

```
akiyama@:~/test_GitHub$ git branch
feature
* main
```

## 作業用のブランチを切る 2/2

- git switchでfeatureブランチに切り替えます

```
akiyama@:~/test_GitHub$ git switch feature  
Switched to branch 'feature'
```

- git branchで再度ブランチを確認しましょう
  - featureブランチに移動できました

```
akiyama@:~/test_GitHub$ git branch  
* feature  
main
```

- 原則として, mainブランチでは作業は行わず, 別のブランチを用意して作業をします
  - 作業後に, 変更内容をmainブランチに取り込みます(後述)

# 作業を行う

- ここでは, 具体的な作業内容として, 以下の複数のコマンドを実行し, 新しいディレクトリを作って, その中にcompphys2の中にあるファイルをいくつかcpしてきましょう

```
akiyama@:~/test_GitHub$ mkdir report1
akiyama@:~/test_GitHub$ cd report1/
akiyama@:~/test_GitHub/report1$ cp ~/compphys2/lecture2/src/sample_plot_functions.py ./
akiyama@:~/test_GitHub/report1$ cp ~/compphys2/lecture3/src/main.tex ./
akiyama@:~/test_GitHub/report1$ ls
main.tex  sample_plot_functions.py
akiyama@:~/test_GitHub/report1$
```

# 作業内容をリモートに反映させる 1/5

- ここまでできたら, 一度git statusを実行しましょう
- git statusを実行することで, ローカルにあるファイルがGitの管理下にあるかどうか調べることができます

```
akiyama@:~/test_GitHub/report1$ git status
On branch feature
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ./

nothing added to commit but untracked files present (use "git add" to track)
```

- このメッセージは, ./(カレントディレクトリ=report1)がGitの管理下に置かれていないと言っています
- 以下で編集内容をGitの管理下に置いていきます

## 作業内容をリモートに反映させる 2/5

- git addとgit commitでカレントディレクトリをGitの管理下に置きます
  - 「git add ./」 で「カレントディレクトリ(./)をステージングする」
  - 「git commit -m “commit メッセージ”」 で「ステージングされたファイルをコミットして, Gitで管理される編集の歴史に加える」
- commit メッセージとして, どのような編集を行なったかなどを記録できる

```
akiyama@:~/test_GitHub/report1$ git add ./
akiyama@:~/test_GitHub/report1$ git commit -m "report1 is prepared"
[feature 5774a39] report1 is prepared
2 files changed, 65 insertions(+)
create mode 100644 report1/main.tex
create mode 100644 report1/sample_plot_functions.py
```

- ここまでできたら, 一度report1を出て, git cloneしてきたリポジトリ直下に移動します

```
akiyama@:~/test_GitHub/report1$ cd ../
```

## 作業内容をリモートに反映させる 3/5

- さて, lsでリポジトリ内を確認します
  - 作業して作ったreport1が見えていますね

```
akiyama@:~/test_GitHub$ ls
README.md  report1
```

- 作業が終わったので, git switchでmainブランチに戻ります

```
akiyama@:~/test_GitHub$ git switch main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)
```

- 「いま作業したローカルのブランチは, リモートよりも一つコミットが進んでいる」と言っています
  - ⇒ つまり, ローカルの編集内容がリモートに未反映であることを意味しています

## 作業内容をリモートに反映させる 4/5

- さて, もう一度lsでリポジトリ内を確認します
  - report1が見えなくなっていますね

```
akiyama@:~/test_GitHub$ ls
README.md
```

- これは, 編集内容がfeatureにのみ存在し, mainには未反映であるためです
- git merge featureで「featureブランチをmainブランチにマージ」します  
(このコマンドを実行するには, 自分がmainブランチにいないとダメ)

```
akiyama@:~/test_GitHub$ git merge feature
Updating b9914db..22e0c02
Fast-forward
 report1/main.tex | 38 +++++++++++++++++++++++++++++++++++++
 report1/sample_plot_functions.py | 27 +++++++++++++++++
 2 files changed, 65 insertions(+)
 create mode 100644 report1/main.tex
 create mode 100644 report1/sample_plot_functions.py
```



# 作業内容をリモートに反映させる 5/5

- さて, もう一度lsでリポジトリ内を確認します
  - report1がmainブランチからも見えるようになりましたね

```
akiyama@~/test_GitHub$ ls
README.md  report1
```

- これで, 作業用ブランチの内容がmainブランチに反映されました
- この状態で, git push origin mainからローカルのmainブランチをリモートリポジトリにpushし, ローカルの編集内容がリモートに送られます

```
akiyama@~/test_GitHub$ git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 52 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 600 bytes | 600.00 KiB/s, done.
Total 6 (delta 1), reused 1 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:akiyama-es/test_GitHub.git
5774a39..22e0c02  main -> main
```

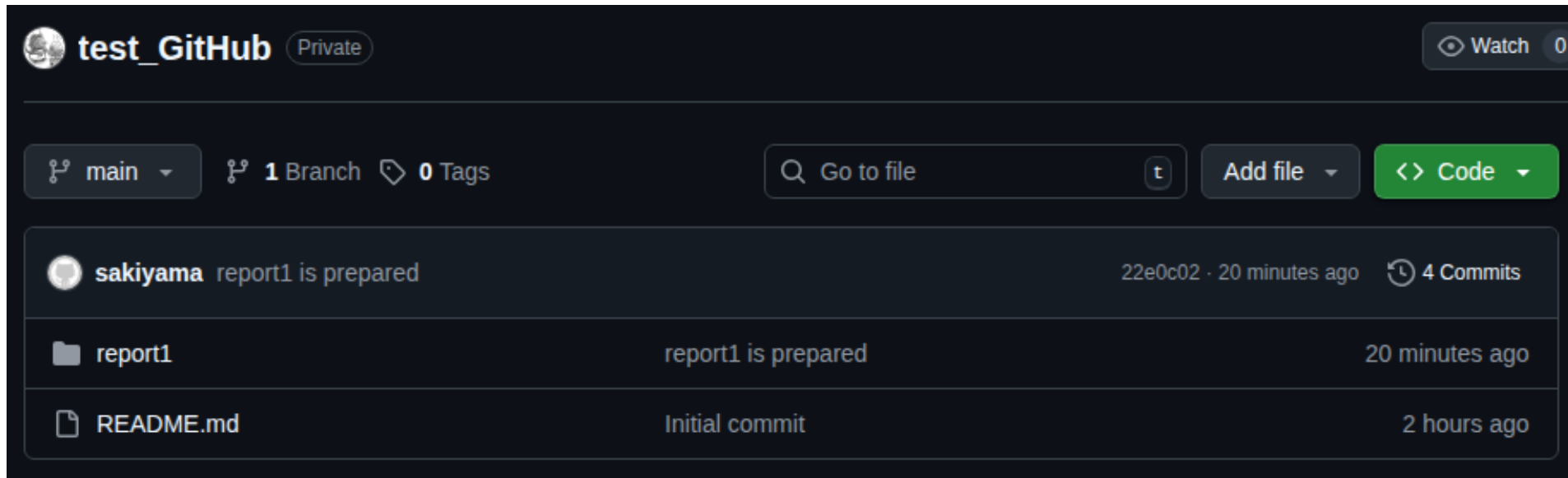
# 作業用に切ったブランチを片付けます

- 作業が済んだので、作業用ブランチを削除します
  - 削除方法は, `git branch -d feature`
  - 削除後に, `git branch`からブランチを確認します
    - `feature`ブランチが消えて, `main`だけになりました

```
akiyama@:~/test_GitHub$ git branch -d feature
Deleted branch feature (was 22e0c02).
akiyama@:~/test_GitHub$ git branch
* main
```

# 編集内容がリモートに反映されたことを確認してみる

- ブラウザからGitHubにアクセスし, repositoryのページからtest\_GitHubを見てみましょう



- 先ほどpushしたreport1が確かに居ますね
  - 見えない場合はブラウザの更新を行いましょう

## 他のロカールからアクセスしたい場合

- すでに存在するリモートリポジトリに対して, 他の計算機(自分のlaptop, 自宅のパソコン, …)からもアクセスしたい場合
  - そのローカルでも秘密鍵・公開鍵を作り, GitHubに登録し, リモートリポジトリをクローンすればよい(p.18~p.29の作業を行う)
    - git cloneは一度だけ

# GitとGitHubを使った個人開発のメリット

- リモートリポジトリでデータが管理されているので, どのローカルからでも作業できる
  - いちいちメモリディスクなどでデータを持ち歩かなくて良い
- バージョン管理はGitがやってくれている
  - 大胆な編集などを恐れずに遂行できる
- “report\_v1.tex”, “report\_v2.tex”, “report\_final.tex”, “report\_修正版.tex”  
のようなバージョン管理からの脱却 ← しばらく時間が経ってから見返すと, どれが  
最新版なのか不明になってしまう. Git/GitHubの利用が賢明(必須)

# GitとGitHubを使った個人開発の注意点

- リポジトリ内でいきなり作業を開始してはならない
  - まずは, リモートに置いてある最新の状態にローカルも揃える
- mainブランチで作業を開始してはならない
  - 個人開発の場合, 作業用ブランチでの作業中にmainブランチが変更されることはほぼないはずなので, ブランチを切るご利益をあまり体感できないかもしれない
  - チーム開発の場合, 作業用ブランチでの作業中にmainブランチが変更されることが普通なので, ブランチを切る習慣を個人開発でも養っておくと良いだろう
- 作業が終わったら, 編集内容を必ずリモートへ反映させないといけない
- p.40~p.49の手順が癖になるまで, 繰り返しながら体得する姿勢が重要
  - 仮にどこかの手順が抜けてしまっても, 修復は効くので慌てないこと

# 個人開発のワークフローを続けてみましょう 1/4

- p.40~p.49をもう一度繰り返しましょう
  - ただし, p.44の具体的な作業として, 第1回レポートの課題1を行いましょう
    - report1ディレクトリをVS codeを開きましょう
    - report1ディレクトリにあるmain.texを編集し, 課題1に取り組みましょう
    - 適当なところで区切りをつけ, main.texの保存とコンパイルまで終わらしましょう
  - p.45からの手順を確認し, git statusを実行することで, git addすべきファイルを全てステージングしましょう
  - 適当なコミットメッセージをつけましょう
- 終わったら, ブラウザから編集内容が反映されていることを確認しましょう

## 個人開発のワークフローを続けてみましょう 2/4

- p.40~p.49をもう一度繰り返しましょう
  - ただし, p.44の具体的な作業として, 第1回レポートの課題2を行いましょう
    - report1ディレクトリにある.pyファイルを編集し, 課題2に取り組みましょう
    - 出力された.pdfファイルをmain.texに挿入しましょう
    - 適当なところで区切りをつけ, main.texの保存とコンパイルまで終わらしましょう
  - p.45からの手順を確認し, git statusを実行することで, git addすべきファイルを全てステージングしましょう
  - 適当なコミットメッセージをつけましょう
- 終わったら, ブラウザから編集内容が反映されていることを確認しましょう



## 個人開発のワークフローを続けてみましょう 3/4

- p.40~p.49をもう一度繰り返しましょう
  - ただし, p.44の具体的な作業として, 第1回レポートの課題3を行きましょう
    - lecture2の/data/population.datと/src/sample.gplをreport1へcpしましょう
    - lecture\_material\_2.pdfのp.46~を参照し, 課題3に取り組みましょう
    - 適当なところで区切りをつけ, main.texの保存とコンパイルまで終わらしましょう
  - p.45からの手順を確認し, git statusを実行することで, git addすべきファイルを全てステージングしましょう
  - 適当なコミットメッセージをつけましょう
- 終わったら, ブラウザから編集内容が反映されていることを確認しましょう

## 個人開発のワークフローを続けてみましょう 4/4

- p.40~p.49をもう一度繰り返しましょう
  - ただし, p.44の具体的な作業として, 第1回レポートの課題4を行きましょう
    - 適当なところで区切りをつけ, main.texの保存とコンパイルまで終わらしましょう
- p.45からの手順を確認し, `git status`を実行することで, `git add`すべきファイルを全てステージングしましょう
- 適当なコミットメッセージをつけましょう
- 終わったら, ブラウザから編集内容が反映されていることを確認しましょう