# INTRODUCTION TO COMPUTATIONAL PHYSICS

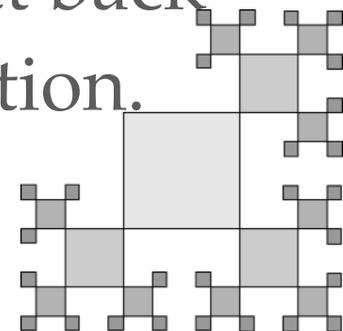
*Kai-Feng Chen*
*National Taiwan University*

Nowadays knowing how to use git is also one of the basic skills for all developers!

# THE NEEDS OF A VERSION CONTROL SYSTEM?

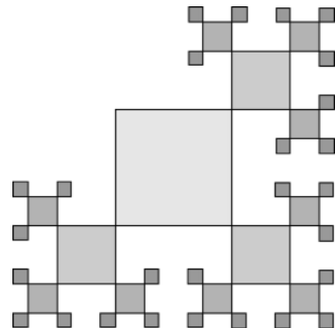
- It is very useful to introduce a **version control system** on your developing project:
  - Keep the tracks for (a collection of) files.
  - Preserve different versions of files in a repository.
  - Allow users to switch between versions!

- Example situation:
  - Would like to implement several experimental features to the program and do not want to destroy the integrity.
  - Using a version control system it is possible to switch between different configurations freely, or simply revert the project back to the original snapshot if want to cancel the implementation.



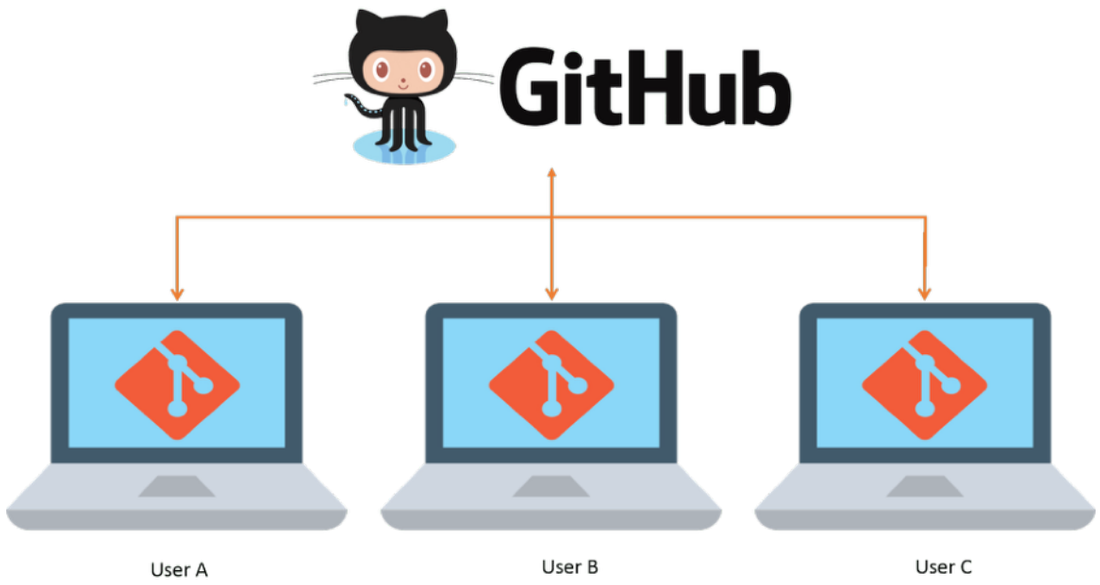
# VERSION CONTROL SYSTEMS SOLUTIONS

- **Localized solution:** simply keeps copies of the files locally, e.g. just make a manual copies of the relevant files. Difficult for a development with multiple users.
- **Centralized solution:** provides a server which keeps all the different versions; we can copy (usually called “checkout”) a version from the central sever. Okay for multiple users, usually but only a user can really work on a file at a time.
  - Typical software: **Subversion (svn)**, **CVS**.
- **Distributed solution:** developers can have a complete local copy (clones) at his individual systems, and every developer can deploy their own codes. Which file from which copies is considered to be the “central version” can be decided afterwards.
  - Typical software: **Git**.



# WHAT ARE GIT & GITHUB?

- **Git** was first founded in 2005 by Linus Torvalds, was mainly designed for the Linux kernel development; it is the most popular distributed version control system, and is used by many developers nowadays!
- **Github** is a service supporting git remote repository, and it hosts many open source projects! It is a very common/popular place for people to share their developing codes. There is another alternative remote service, too, such as Gitlab.

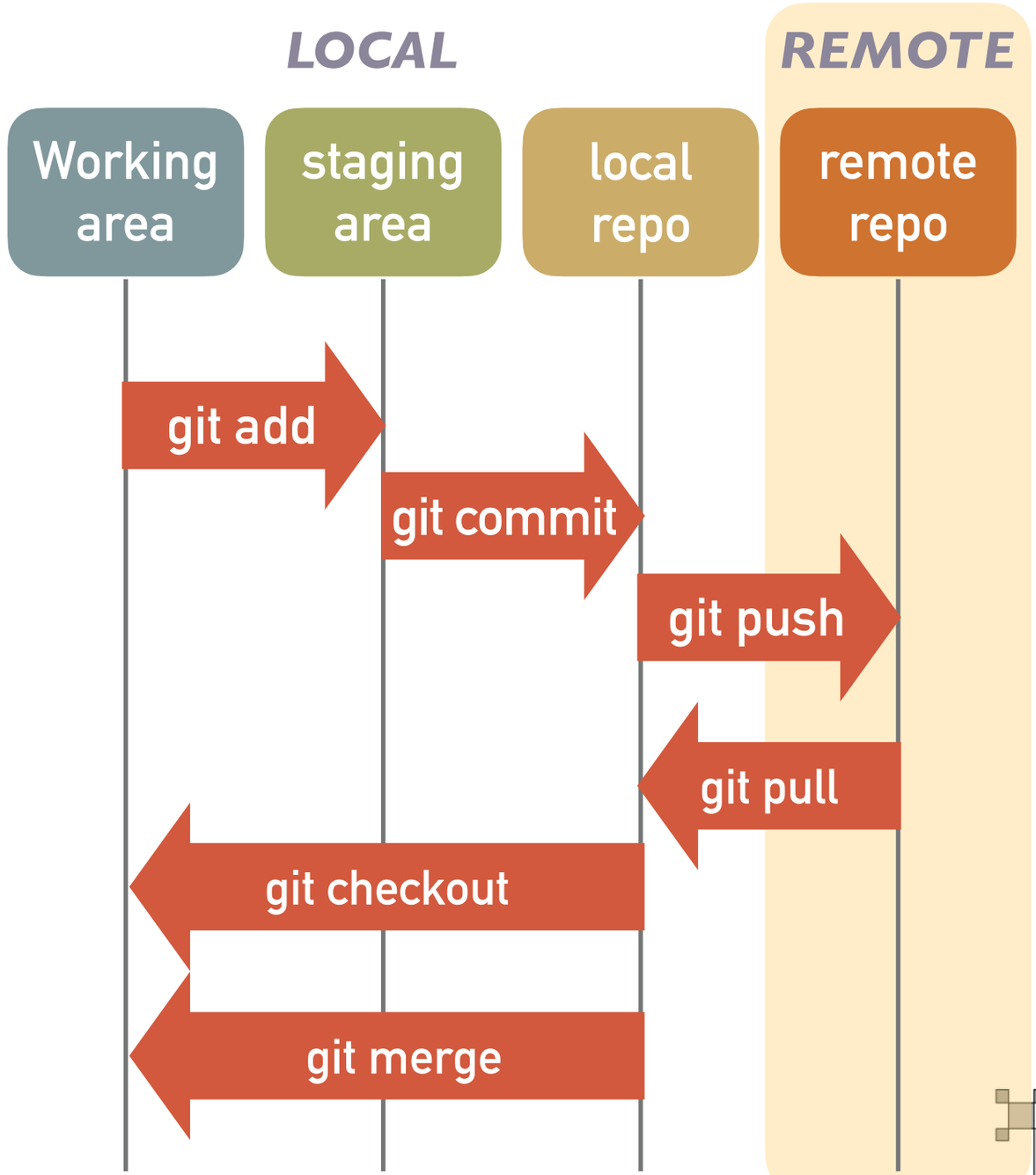



*GitHub = host service for managing Git repositories*

*Git = version control system*

# LOCAL & REMOTE REPO

- Git has two working environment, **local** and **remote**.
- Developers can work locally first, and commit the modified files to the local repositories.
- Perform push/pull to the remote repositories to integrate the works with other developers; if there is conflict with the deployment by other developers, it has to be resolved before merging the updates.



# GIT INSTALLATION

- Git is nothing more than a software! You can simply install it for your system accordingly:
  - For Linux (Debian/Ubuntu):

```
$ apt-get install git
```

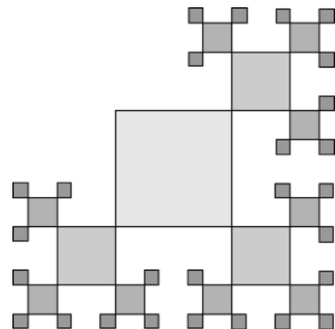
  - For Linux (Redhat/Centos):

```
$ yum install git
```

  - For Linux (Fedora 23+):

```
$ dnf install git
```

  - For Mac OS, if you have already install Xcode, you probably already have git installed. In newer system you can just type “git” in terminal to verify this.
  - For Windows, you can download from the official website: https://git-scm.com



# CONFIGURE YOUR GIT

- The first thing is to config your name and e-mail address:

```
$ git config --global user.name <name>
$ git config --global user.email <name@gmail.com>
```

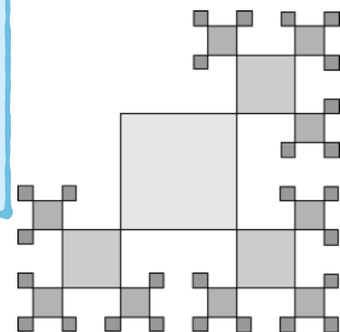
surely you have to replace <name> and <name@gmail.com> with your own setup!

- Now go to your working area and create a **local repository**:

```
$ cd /my/working/area
$ git init
```

and you should find a hidden directory under your working area, name **.git**:

```
$ ls -la
drwxr-xr-x  3 kfjack  staff   96 Sep 11 20:34 .
drwxr-xr-x  3 kfjack  staff   96 Sep 11 20:34 ..
drwxr-xr-x  9 kfjack  staff  288 Sep 11 20:34 .git
```



# LET'S COMMIT OUR VERY FIRST FILE

- Let's simply create a hello.py in your working area:

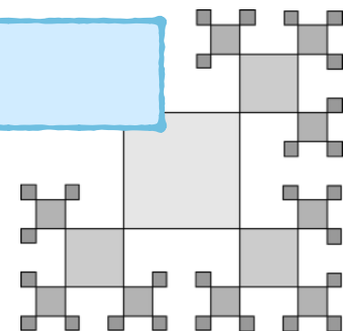
```
$ echo "print('Hello, World!')" > hello.py
```

- Then check the status of git by:

```
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  hello.py
```

- The new file has not yet been included in the tracking list, so let's execute the following to add it:

```
$ git add hello.py
```



# LET'S COMMIT OUR VERY FIRST FILE(II)

- Now you should find your file is ready to commit, e.g.

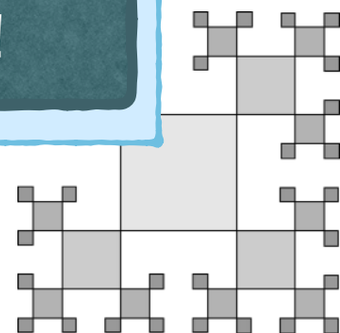
```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
  new file:   hello.py
```

you can use
git rm --cached <file>
to remove the file, too!

- Then let's commit it and check the status again:

```
$ git commit -m "very first commit"
[master (root-commit) 241232b] very first commit
 1 file changed, 1 insertion(+)
 create mode 100644 hello.py
$ git status
On branch master
nothing to commit, working tree clean
```

Now your hello.py is happily stay in the local repo!



# LET'S COMMIT OUR VERY FIRST FILE(III)

- Let's modify our hello.py and see what happens?

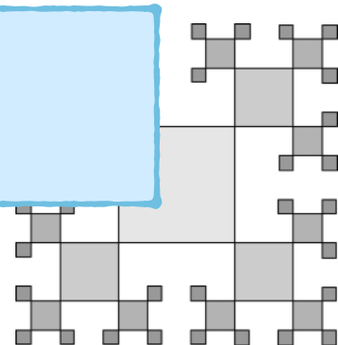
```
$ echo "print('HELLO, WORLD!')" > hello.py
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)

  modified:   hello.py
```

- If you want to revert the file back, you can do

```
$ git checkout -- hello.py
```

- If you want to commit the file to repo, here are the commands:

```
$ git add hello.py
$ git commit -m "second commit"
```

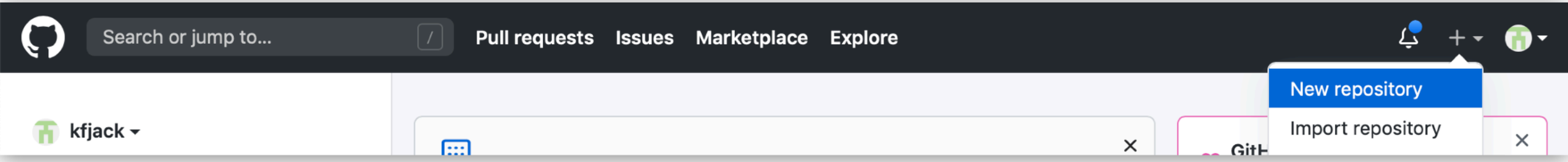


# GET TO REMOTE: GITHUB.COM

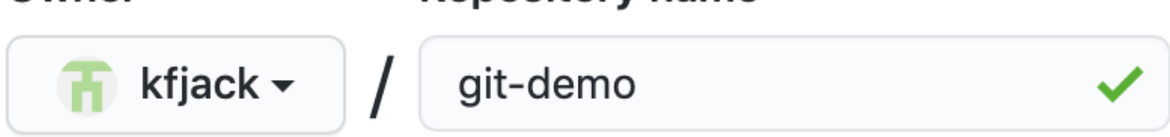
- Please create an account on github.com first, and set your local git user name accordingly:

```
$ git config --global user.username <github_account>
```

- Then create a new repository by selecting the "+" on the web:




- First is to decide the **repository name**:




- There are several options, such as public/private, adding README or not, adding ignoring list or not, adding license or not…

Press the "create" button, then get your repo ready!

# THIS IS WHAT YOU SHOULD SEE

## Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/kfjack/git-demo.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

## ...or create a new repository on the command line

```
echo "# git-demo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/kfjack/git-demo.git
git push -u origin main
```

Full list of commands if you want to start a new local repo.

## ...or push an existing repository from the command line

```
git remote add origin https://github.com/kfjack/git-demo.git
git branch -M main
git push -u origin main
```

If you want to connect to an existing local repo.

## ...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

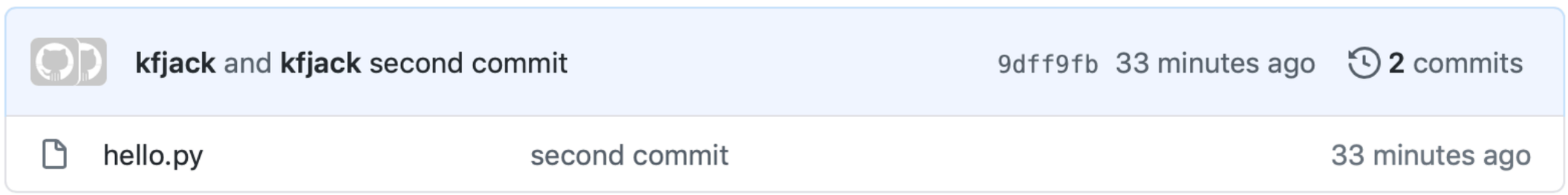
Import code

# LET'S PUSH OUR LOCAL REPO TO REMOTE

- Let's simply follow the instructions given by GitHub:

```
$ git remote add origin <your_repo_url_on_github>
$ git branch -M main
$ git push -u origin main
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
. . . . . .
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from
'origin'.
```

- Now you can find your file available on the GitHub site, bravo!



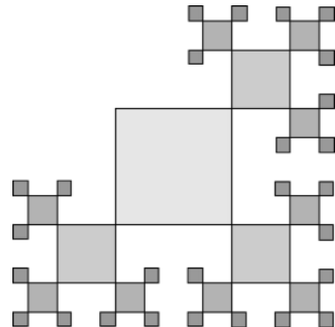

*Wait, it does not work for me..!?*

# IF YOU ARE RUNNING INTO THE TROUBLE...

- Very recently GitHub disabled the direct password authentication, hence you may see such a complaint when you do the git push:

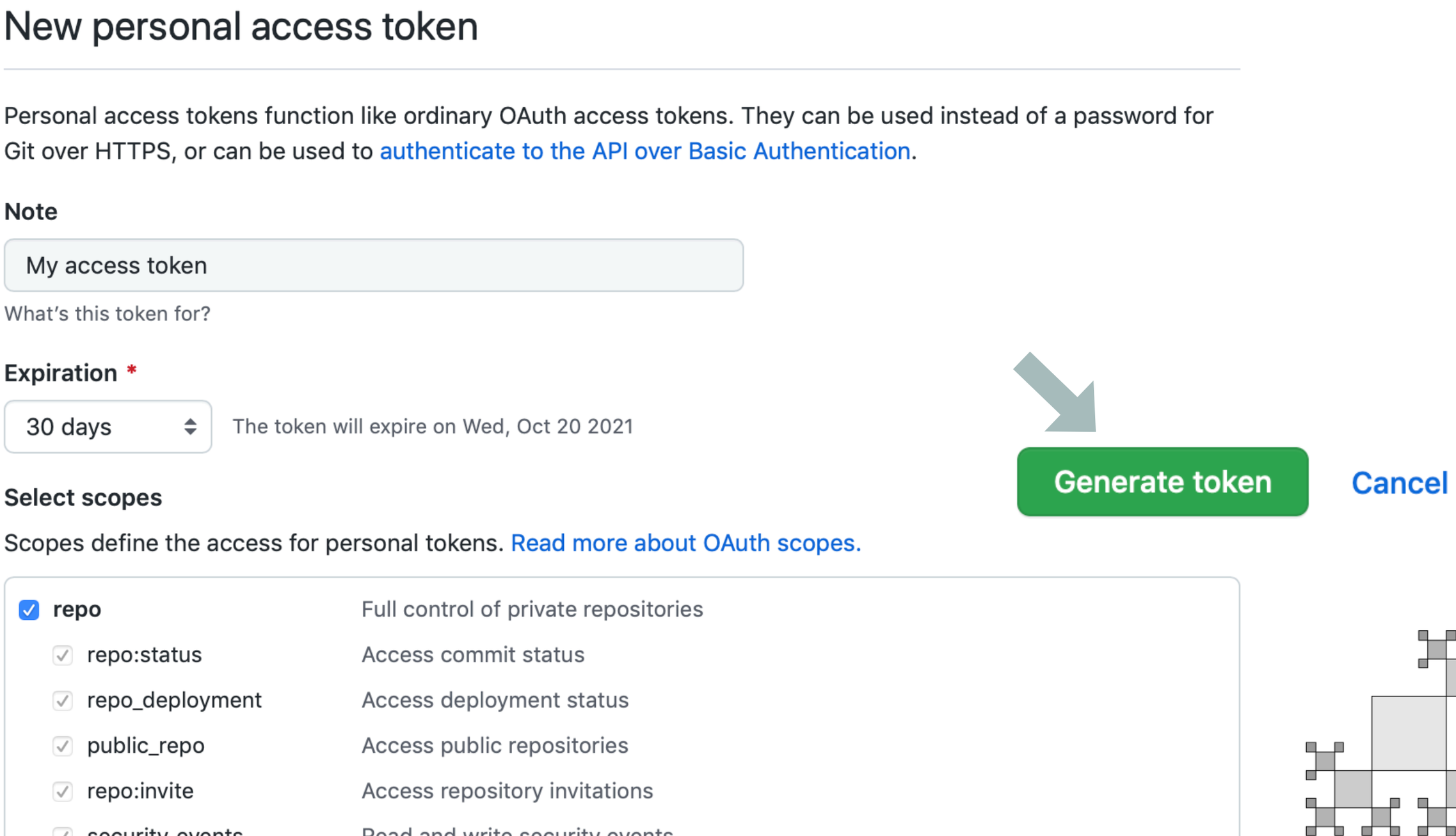
```
$ git push -u origin main
remote: Support for password authentication was removed on
August 13, 2021. Please use a personal access token instead.
remote: Please see https://github.blog/2020-12-15-token-
authentication-requirements-for-git-operations/ for more
information.
fatal: Authentication failed.
```

- This can be resolved by creating a personal token (there are other ways too, but this is recommended & easy).
- Please connect to the link below for creating a new token: https://github.com/settings/tokens/new



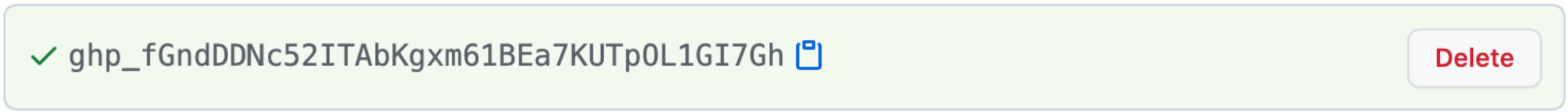
# PERSONAL ACCESS TOKEN CREATION

- Just insert a name, select **"repo"** access, then the **"Generate token"**.



# REMOTE ACCESS WITH PERSONAL ACCESS TOKEN

- After generate the token, copy the token:




and insert it with the following command:

```
git remote set-url origin https://your_account:your_token@your_repo
```

```
$ git remote set-url origin https://
kfjack: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@github.com/kfjack/
git-demo.git
$ git push -u origin main
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
. . . . . .
```

*It should work now! Surely you can try other ways like two-factor authentication instead.*

# JUST FOR ANOTHER COMMIT

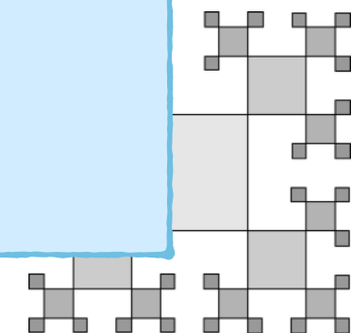
- Practice a full commit update to local again:

```
$ echo "print('HELLO, GIT!')" > hello.py
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
. . . . . .

  modified:   hello.py
$ git add hello.py
$ git commit -m "third commit"
```

- Now you can see the hint that your local repo is newer than the remote one! Let's push again to merge them!

```
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
. . . . . .
$ git push
. . . . . .
   9dff9fb..15fbb68  main -> main
```





# CREATE A NEW BRANCH

This can be done by your collaborator!
Find someone else to give it a try!

- It is possible to create a feature branch, e.g.

```
$ git checkout -b test
```

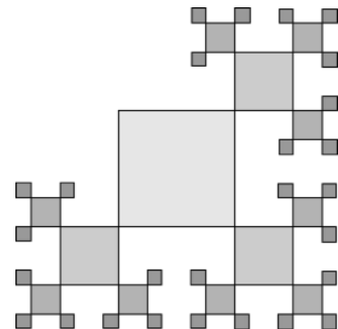
Change the **"test"** branch name to the one you want.

- Now we can modify the code and commit it through this branch:

```
$ echo "print('HELLO, BRANCH!')" > hello.py
$ git status
On branch test
Changes not staged for commit:
. . . . . .
  modified:   hello.py
$ git add hello.py
$ git commit -m "Commit to test branch"
```

In local repo now!

And we still need to push to the remote repo!

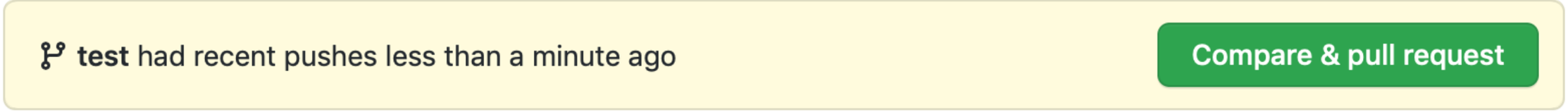


# MERGE THOUGH A PULL REQUEST

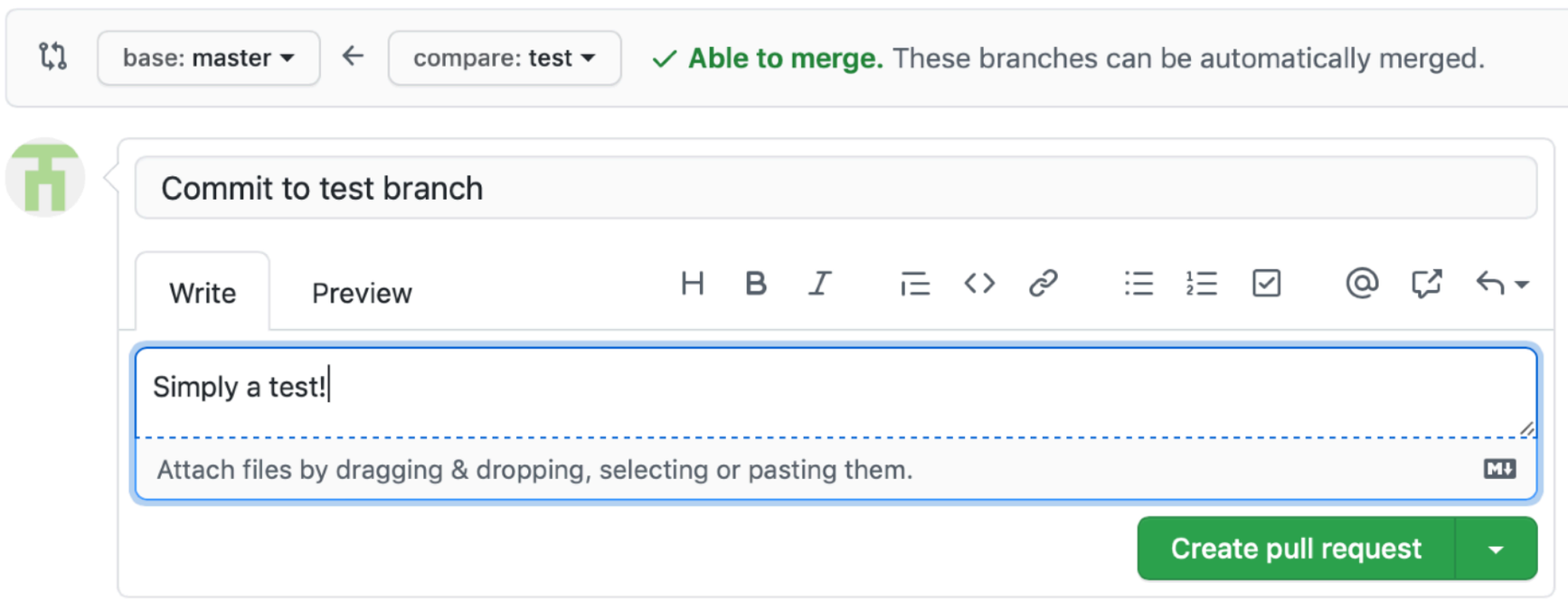
- Now we can push the “test” branch to remote repo, e.g.

```
$ git push origin test
. . . . . .
remote: Create a pull request for 'test' on GitHub by visiting:
remote:      https://github.com/kfjack/git-demo/pull/new/test
```

- Mean while you should see this on the GitHub web:

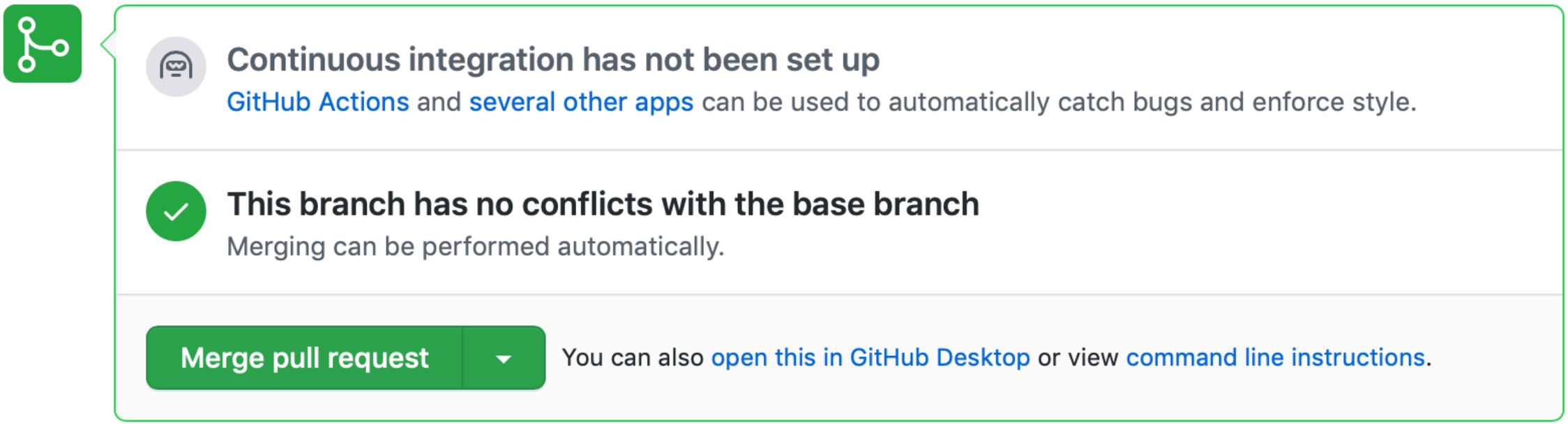



- Now you can try to create a **Pull Request** according:

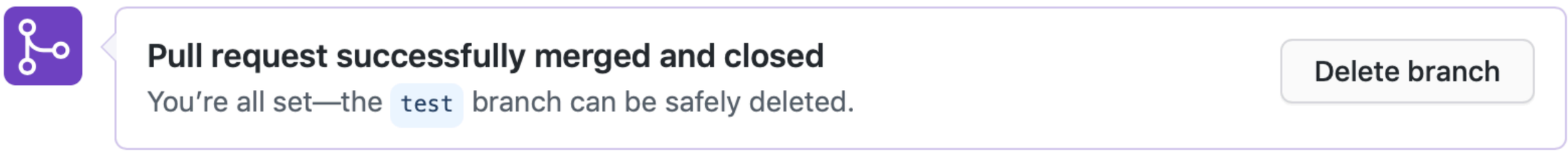


# MERGE THOUGH A PULL REQUEST (II)

- And you (as the project manager!) should be able to see this, if you go to the list of pull requests:




- If you agree to merge it, just click the bottom and proceed:




After this you should see your file is fully up-to-date!

# LET'S PRACTICE THIS AGAIN, ALL IN COMMAND LINES!

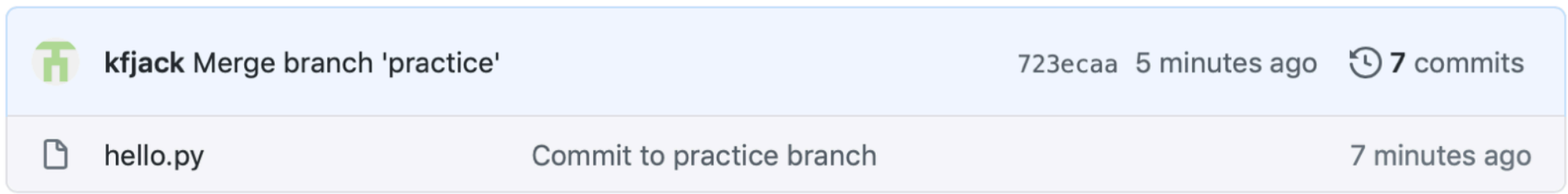
- Step 1: start from clone, create branch, update, and commit:

```
$ cd /some/other/working/area
$ git clone <your_repo_url_on_github>  ←clone to a local repo
$ cd git-demo
$ git checkout -b practice
$ echo "print('HELLO, BRANCH AGAIN!')" > hello.py
$ git add hello.py
$ git commit -m "Commit to practice branch"
```

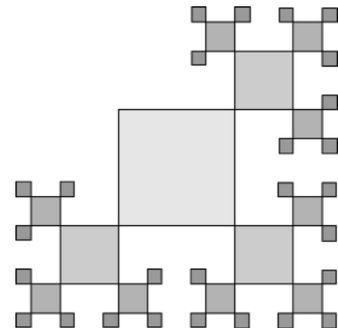
- Step 2: switch to main, merge, delete branch, push to remote:

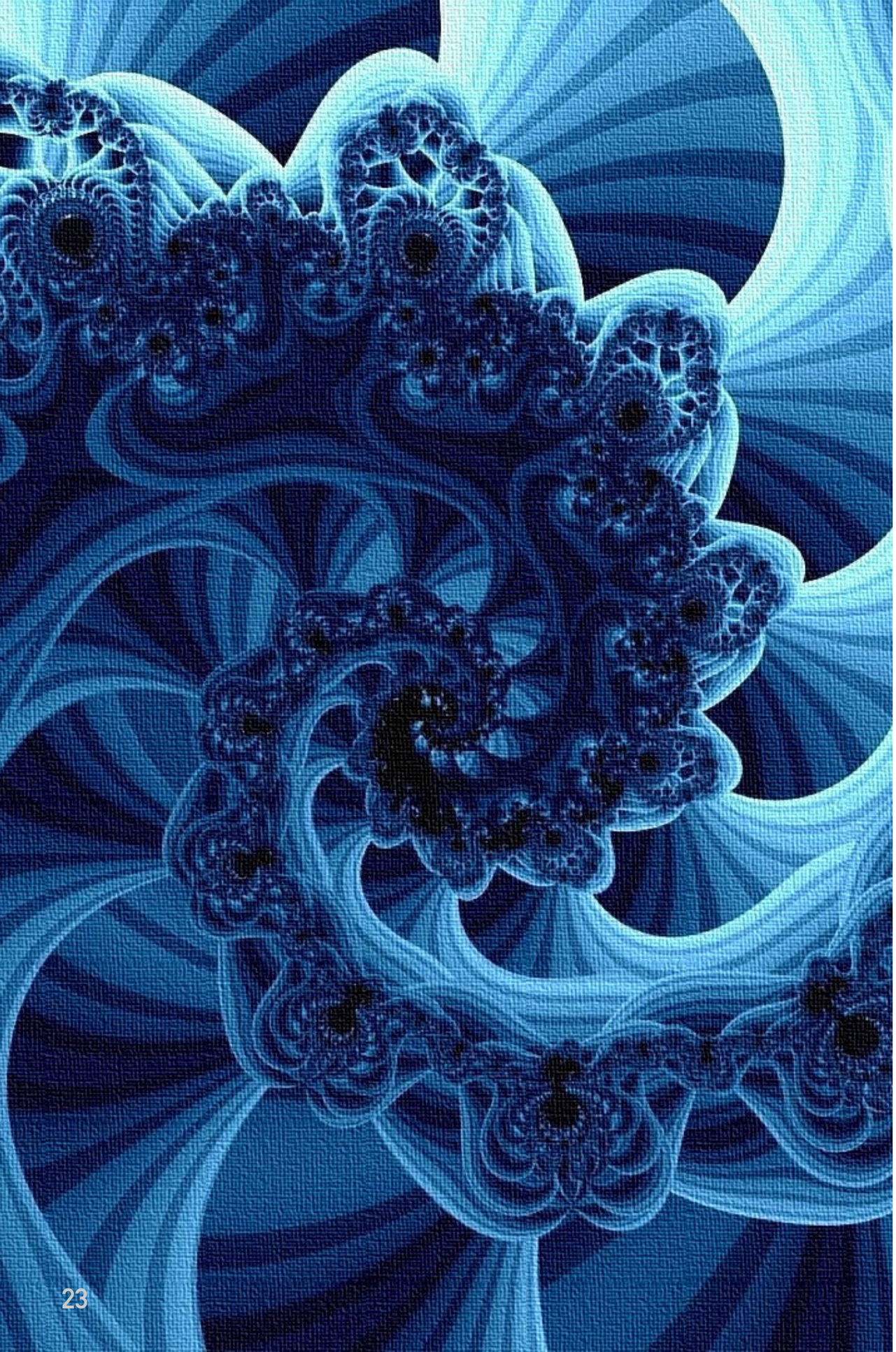
```
$ git checkout main          ←switch to master branch
$ git merge practice
$ git branch -d practice     ←merge "practice" branch and delete it
$ git push origin main       ←push to remote
```

- You should see the file is up-to-date again on the web:







# MODULE SUMMARY

- This module we have introduced the very useful tool Git, as it is the most popular version control system nowadays, as well as how to link your local repository to a remote repository on the GitHub.