UCL Data Science Society - Term 2 Workshop 6: Introduction to Git/Github

Created by Zhaoxuan "Tony" Wu, First-Year Representative of Science Divison, *UCL Data Science Society*

See me on *Github*👨

Table of Contents

Prerequisition

Git

You should download *Git*. Feel free to check the *official documentation* if you are interested in. We will be using the *Terminal*, so it might be helpful to get familiar with the commands.

Recommended Terminal for MacOS: *iTerm2*

Recommended multi-lang IDE: *Atom*, *Visual Studio*, *Sublime Text*, or *Vim*, which is pre-installed on MacOS

Github

Get yourself a *Github* account, it's free! If you prefer graphical interface, *Github Desktop*. But we are going to learn *Git* and *Github* using command line today.

What is *Git*, and Why?

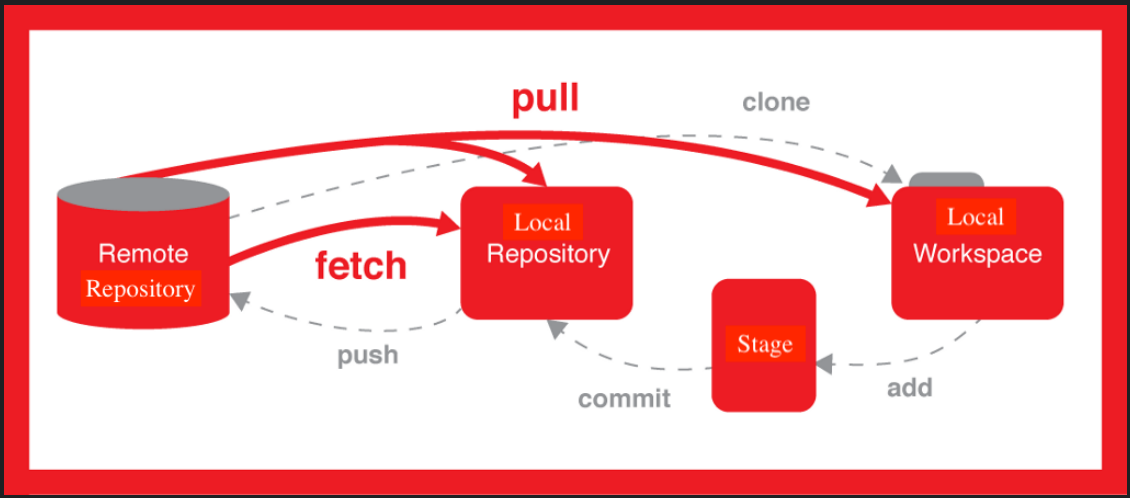
> *Git is a free and open source* ***distributed version control system*** *designed to handle everything from small to very large projects with speed and efficiency.*

- Industrial production standard
- Hackathon
- Writing a book
- Keeping lecture nots
- ...

How is it different from *Github*?

Git is a ***system*** that manages the version control of a project, while *Github* is a ***remote platform*** that hosts that project using *Git*. For instance: `git push` uploads your current local repository to *Github*

Relationship



> ***Repository*** ("Repo") : a receptacle or place where things are deposited or stored

Your first *Git* repo

Do the following on your own *Terminal*

Initialisation of project

Create a *directory* (folder) for your project:

```
mkdir myapp
```

Go to that directory:

```
cd myapp
```

Initialise the project with *Git* so that *Git* manages the version control of the project:

```
git init
```

Now you should have a `.git` folder in your directory, which is invisible currently. Also, your current project is called the *local workspace*.

Make changes

Now, create an `app.py` *Python* file in this folder and write a line of code that outputs `"Welcome to DSS Workshop 6"`

```
key = ""

def getKey():
     f = open("key.txt","r")
     key = f.read()
     f.close()

def login(account,password, ):
     if account ==
```

You can do it with your IDE or with command line. The method for doing that using command line is as the following:

Create `app.py` :

```
touch app.py
```

Edit the file using the built-in editor *Vim*, or use IDE:

```
vim app.py
```

```
key = ""

def getKey():
     f = open("key.txt","r")
     key = f.read()
     f.close()

def login(password):
     if account == key:
          print("successful")
     else:
          print("denied")
```

Save the file and quit:

```
:wq
```

Stage the changes

Before you commit your changes, you should *add* your changes to the stage, or "*stage*" it.

```
git add .
```

> ***Note that***: `git add` takes in a parameter, which is the filename. For example, you can do `git add app.py` to stage `app.py`, `git add .` is the wildcard mode that stages ***every*** file that has been changed

To check your *stage*:

```
git status
```

Commit the changes

Now, commit your changes:

```
git commit -m "first commit"
```

> ***Note that***: `git commit` takes in commit comment using the `-m` flag, followed by your comment string. It can be anything. Here we use `"first commit"` as example, which is usually what you do for your first commit literally

Congrats! 🥳 You just commited your first contribution to the project! Now this version of commit is officially in your *local repository* (*local repo*).

Create another file

Now create another file `key.txt` with whichever method you like and write the following line:

```
"ucldssistheBEST:)"
```

> Don't stage it yet

Security, security, security, and `.gitignore`

You might notice that it is quite dangerous to commit a copy of your `key.txt` to a *repo* or a *remote repo*.

- API keys
- Database password
- Credentials
- Biometrics data
- Data under NDA
- `.DS_Store`
- `/node_modules`
- ...

By using `.gitignore`, you can prevent certain files to be commited to a *repo*

```
touch .gitignore
```

Directly add the name of the files you want to hide to that `.gitignore` file:

```
vim .gitignore

.DS_Store
key.txt
```

Save and quit:

```
:wq
```

Now, *stage* and *commit* everything and see what happen.

Remote repo and *Github*

Now you might want to share your code with other developer, you can do this by putting your project on a remote repo. Do this by call the following function:

```
seeTonyForLiveDemo()
```

Branching, and uploading changes to *remote repo*

For instance, you want to create an *HTML* for your app. Create and switch to a new branch called `html`:

```
git checkout -b html
```

> ***Note that***: `-b` flag is for creating a new branch. If you want to see all available branches, use `git branch`, if you want to switch to an existing branch, use `git checkout <branch>` where `<branch>` is the branch name

Craete your HTML:

```
<html>
     <head>
          <title>UCL DSS</title>
     </head>
     <body>
          <h1> My heading</h1>
          <p> Your first programme/page for any languages should always be: Hello World! </p>
     </body>
 </html>
```

Stage it, check the *stage*, *commit* it. Now, upload it:

```
git push -u origin html
```

Go to your *Github* repo to see what happened.

Collaborative Coding: 101

> To be implemented

If that's not your own project:

- fork an existing project
- Upload to your know repo
- Pull request
- Keep your repo the same

```
git remote -v
git remote add upstream <your_upstream>
git fetch upstream
git merge upstream/master
git push -u origin upstream
```

If that's your own project:

- Manage pull request
- Merge

Practice: Signing an attendance sheet