Collaborate on github : A layman's guide

Subhasish Basak
subhasishbasak.github.io/
May 7, 2020

1 Disclaimer

This tutorial is designed to give the reader a basic working knowledge on how to work on a collaborative project using git. The article is based on the author's personal experience and thus it may contain some technical errors/misconceptions (the reader should never hesitate to bring such issues to the author, whenever found). One should not completely rely one this document only, the author highly recommends to go through other git tutorials available on the net, one of which are listed later.

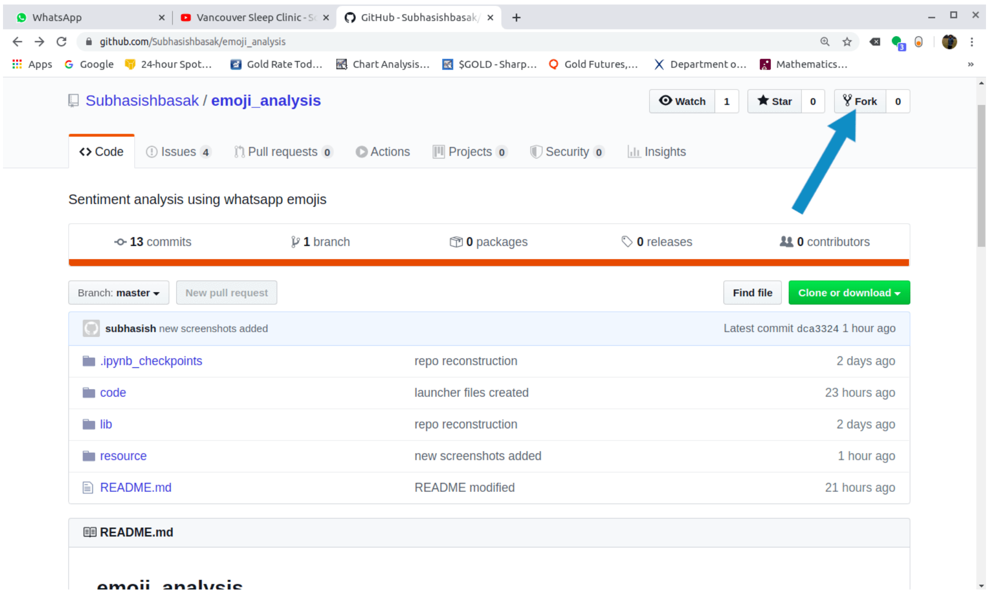
2 Basic setup

You need to have your own github account first. Now suppose you want to collaborate to an existing github repository, for e.g *Subhasishbasak/emoji_analysis*. There are two ways you can do that,

2.1 git fork

Git Fork means you just create a copy of the repository to your own GitHub profile. Here you can experiment whatever you like without affecting the main source of that project.

- In this approach you **WILL NOT** be listed as a direct collaborator to the project.
- Go to the repository using : `https://github.com/Subhasishbasak/emoji_analysis` and click on **fork**.



Sentiment analysis using whatsapp emojis

subhasish new screenshots added — Latest commit dca3324 1 hour ago

.ipynb_checkpoints	repo reconstruction	2 days ago
code	launcher files created	23 hours ago
lib	repo reconstruction	2 days ago
resource	new screenshots added	1 hour ago
README.md	README modified	21 hours ago

Figure 1: forking a repository

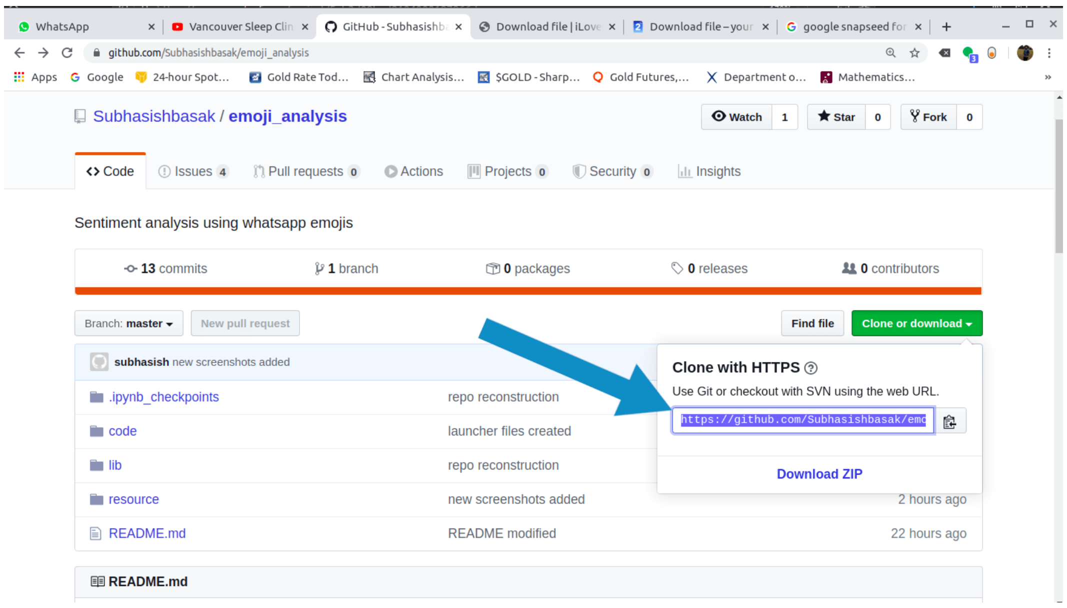
- This will create a copy of this repository in your own github profile.
- Now you can **clone** (described later, see section 3) the **FORKED** repo locally to your machine and start your experiments.

- Contributions made by this approach will be independent and **WILL NOT AFFECT** the main repository.
- You can share your work with the main repo by creating a **pull request**(described later).

2.2 Direct collaborator (Recommended)

To be a direct collaborator to the repository, first you are need to be **added as collaborator**, which can be done only by the owner the main repository. You will receive an email letting you know that you are added and will be listed as a collaborator.

- In this approach you will work as a direct collaborator (listed on the repo).
- Once **added** as a collaborator, go to the repository using : `https://github.com/Subhasishbasak/emoji_analysis`



Sentiment analysis using whatsapp emojis

13 commits · 1 branch · 0 packages · 0 releases · 0 contributors

subhasish new screenshots added

.ipynb_checkpoints	repo reconstruction	
code	launcher files created	
lib	repo reconstruction	
resource	new screenshots added	2 hours ago
README.md	README modified	22 hours ago

Clone with HTTPS

Use Git or checkout with SVN using the web URL.

`https://github.com/Subhasishbasak/emo`

Download ZIP

README.md

Figure 2: cloning a repository

- Click on **Clone or download** and copy the link for Clone with HTTPS. **DON'T FORK**.
- Now you can **clone** (described later, see section 3) the **MAIN** repo locally to your machine and start your experiments.
- Contributions made by this approach **WILL AFFECT** the main repository.

3 Clone

Now, the following steps demonstrate how to **clone** a repository. **Clone**-ing a repository makes a copy of the repository locally in your machine, so that you can make changes to it and **push** it back to the repository whenever required.

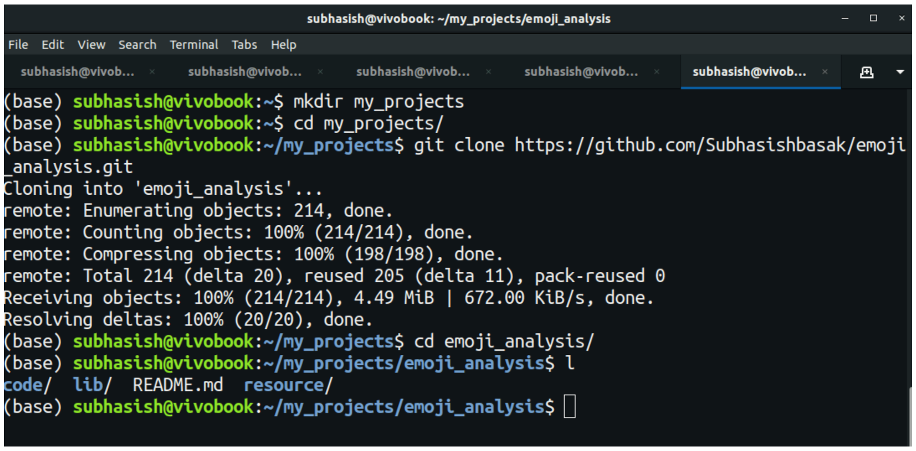
- **Prerequisites**:
  - **Installing git** : You can follow any standard tutorial available on net.
  - **OS preferences** : This tutorial is based on linux commands. Using a linux based OS is highly recommended (simply becoz I haven't used git on Windows).

3.1 Steps to follow

Once you are ready with your git installation and have the clone link copied from the repository, just follow the simple the steps.

- Select a working directory and **cd** into it
- run

```
$ git clone https://github.com/Subhasishbasak/emoji_analysis.git
$ cd emoji_analysis
```


subhasish@vivobook: ~/my_projects/emoji_analysis
File Edit View Search Terminal Tabs Help
subhasish@vivob...

```
(base) subhasish@vivobook:~$ mkdir my_projects
(base) subhasish@vivobook:~$ cd my_projects/
(base) subhasish@vivobook:~/my_projects$ git clone https://github.com/Subhasishbasak/emoji
_analysis.git
Cloning into 'emoji_analysis'...
remote: Enumerating objects: 214, done.
remote: Counting objects: 100% (214/214), done.
remote: Compressing objects: 100% (198/198), done.
remote: Total 214 (delta 20), reused 205 (delta 11), pack-reused 0
Receiving objects: 100% (214/214), 4.49 MiB | 672.00 KiB/s, done.
Resolving deltas: 100% (20/20), done.
(base) subhasish@vivobook:~/my_projects$ cd emoji_analysis/
(base) subhasish@vivobook:~/my_projects/emoji_analysis$ l
code/  lib/  README.md  resource/
(base) subhasish@vivobook:~/my_projects/emoji_analysis$
```

And now you're ready to collaborate!

4 Collaborating

Once you have the repository in your local machine, you can now start your work. For collaborating in github you need to know 5 basic commands,

- **git status** : to check the current status of your local repository and the remote main repository.
- **git pull** : suppose your partner has made some changes in the repo after you cloned it. This command will fetch all the current changes to your local repo. (to be used cautiously, described later)
- **git add** : suppose you have made a local change. This commang will "**add**" the change LOCALLY in your repo.
- **git commit** : after "**add**"-ing the change you need to **commit** the changes.
- **git push** : after "**commit**"-ing the changes you can **push** it to the remote MAIN repository.

The following demonstration shows how you can create a dummy file and push your changes to the main repository. Follow the **sequence carefully**.

☐ **Check status** : Always run status a check before & after you start doing anything. This is really helpful to avoid mistakes! just run ..

```
$ git status
```

For our case we haven't yet done any changes so it will say,

```
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

This means there is no difference between your local repository at the current time compared to when you last **pull**-ed / **clone**-d it. (things will be clear, just bear with me)

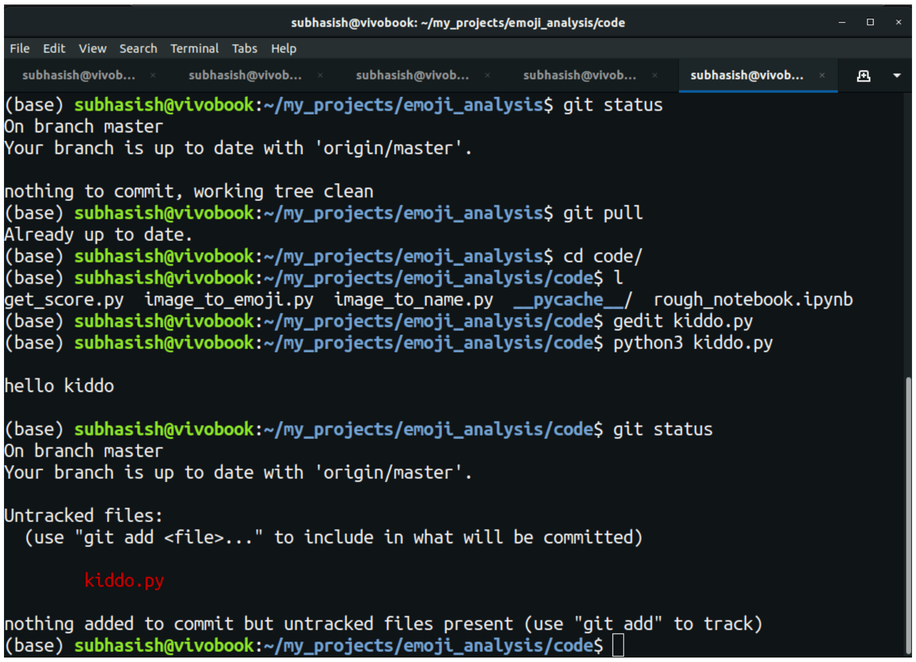
☐ **Pull** : Once you are assured that there is nothing to commit, ONLY THEN you do a **pull**. This will fetch all the changes that are done after the last time you **pull**-ed / **clone**-d th repo. For our case, let there are no changes done so we'll get something like this,

```
$ git pull

Already up to date.
```

This means your local repository is up to date with the remote main repository. It is always recommended to **pull** all the changes before making some new **commit**.

☐ **Make the changes** : Now we create a new python file named **kiddo.py**, which prints "Hello kiddo"; inside the *code* directory of the local repository. Once you create the file check status and you get the newly created file as **Untracked files** written in **RED**.



subhasish@vivobook: ~/my_projects/emoji_analysis/code
File Edit View Search Terminal Tabs Help
subhasish@vivob...

```
(base) subhasish@vivobook:~/my_projects/emoji_analysis$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
(base) subhasish@vivobook:~/my_projects/emoji_analysis$ git pull
Already up to date.
(base) subhasish@vivobook:~/my_projects/emoji_analysis$ cd code/
(base) subhasish@vivobook:~/my_projects/emoji_analysis/code$ l
get_score.py  image_to_emoji.py  image_to_name.py  __pycache__/  rough_notebook.ipynb
(base) subhasish@vivobook:~/my_projects/emoji_analysis/code$ gedit kiddo.py
(base) subhasish@vivobook:~/my_projects/emoji_analysis/code$ python3 kiddo.py

hello kiddo

(base) subhasish@vivobook:~/my_projects/emoji_analysis/code$ git status
On branch master
Your branch is up to date with 'origin/master'.

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

        kiddo.py

nothing added to commit but untracked files present (use "git add" to track)
(base) subhasish@vivobook:~/my_projects/emoji_analysis/code$ 
```

Figure 3: Creating a new file

Now we are just left to : **add**–>**commit**–>**push**

☐ **Add**: just run code,

```
$ git add kiddo.py
```

and check the status. Now it will show the file names just added in **GREEN** as *Changes to be committed.*

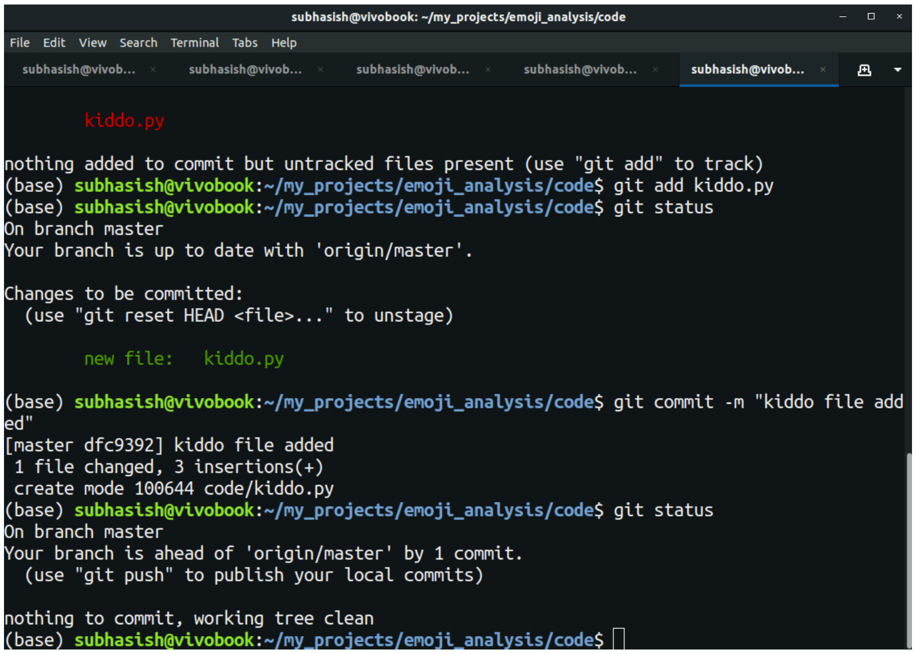
☐ **Commit**: just run the code with a suitable message.

```
$ git commit -m "kiddo file added"
```

and check status. It will say

```
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```



subhasish@vivobook: ~/my_projects/emoji_analysis/code
File Edit View Search Terminal Tabs Help
subhasish@vivob...

```
        kiddo.py

nothing added to commit but untracked files present (use "git add" to track)
(base) subhasish@vivobook:~/my_projects/emoji_analysis/code$ git add kiddo.py
(base) subhasish@vivobook:~/my_projects/emoji_analysis/code$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   kiddo.py

(base) subhasish@vivobook:~/my_projects/emoji_analysis/code$ git commit -m "kiddo file add
ed"
[master dfc9392] kiddo file added
 1 file changed, 3 insertions(+)
 create mode 100644 code/kiddo.py
(base) subhasish@vivobook:~/my_projects/emoji_analysis/code$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
(base) subhasish@vivobook:~/my_projects/emoji_analysis/code$
```

Figure 4: committing a file

- ☐ **Push** : The last step is to push your changes to the remote repository.

  ```
  $ git push
  ```

  And finally it is done. You can see the changes you just made at the on the remote repository.

If you have understood till this you are ready to collaborate, but wai(t)....wai(t).....wai(t)...

By this time you might be thinking , what if several collaborators modify the same file and push together? OR what if someone else already changed the file I am modifying ? Certainly, it will create a mess. Thus one should follow certain rules to before implement the chain **add**–>**commit**–>**push**, to avoid conflicts with the other collaborators. The solution is **branching**.

4.1 Branching

The idea is to make a branch independent of the main work-flow (also known as the **Master** branch) and merge them whenever your changes are done. The way to keep Master deployable is to create new branches for new features and merge them into Master when they're completed.

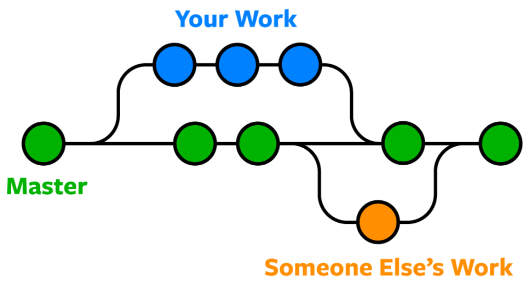


Figure 5: git branch

There is well written tutorial available on branching, here's the link click here