UCL Data Science Society - Internal Training 1: Introduction to *Git/Github*

Credit

Created by Zhaoxuan "Tony" Wu, Head of Science, *UCL Data Science Society*

See me on *Github*🙋‍♂️

Table of Contents

Prerequisition

- [ ] Setup a *Github* account
- [ ] For *MacOS/Linux*: Download *Git*
- [ ] For *Windows*: Download *Bash*

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. Also, remember to claim your *Github Student Package* 📦

What is *Git*, and Why?

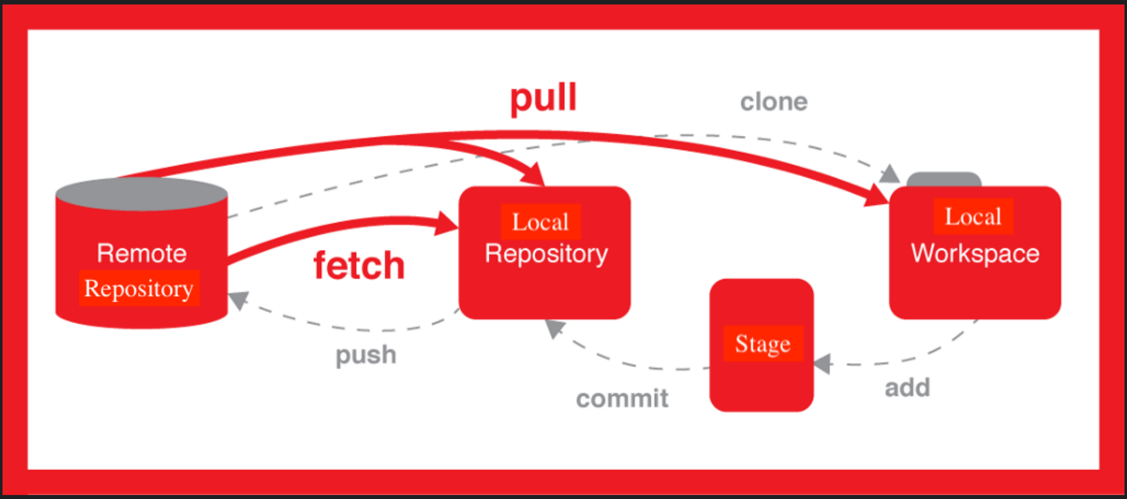
> *Git is a free and open source* ***distributed version control system*** *designed to handle everything from small to very large projects with speed and efficiency.*

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

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* (or better known as ***"folder"***) for your project:

```
mkdir my-novel
```

Go to that *directory*:

```
cd my-novel
```

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*.

Write something

Let's write a ***novel*** together! You can do it with your IDE or with command line. The method for doing that using command line is as the following:

Create `intro.md`:

```
touch intro.md
```

Edit the file using the built-in editor ***Vim***, or use ***IDE***. Copy and paste the following text:

```
It was the best of times, it was the worst of times; it was the age of wisdom, it was the age of foolishness; it was the epoch of belief, it was the epoch of incredulity; it was the season of light, it was the season of darkness; it was the spring of hope, it was the winter of despair; we had everything before us, we had nothing before us; we were all going direct to Heaven, we were all going direct the other way.
```

Stage the changes

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

```
git add intro.md
```

> ***Note that***: `git add` takes in a parameter, which is the filename. The "*wildcard*" mode is also available: it stages ***every*** file that has been changed. To do so, execute: `git add .` to pass `.` as a wildcard parameter

To check your *stage*:

```
git status
```

And you will see:

```
On branch master

No commits yet

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

        new file:   intro.md
```

> ***Note that***: here we can see some important information
>
> - Branch: `master` (stay tuned for what it means)
> - Commits: none (stay tuned for what it means)
> - Changes: `new file: intro.md` - `intro.md` is staged and is ready to be commited

Commit the changes

Now, commit your changes:

```
git commit -m "Initialise project with a intro"
```

> ***Note that***: `git commit` takes in commit comment using the `-m` flag, followed by your comment string. It can be anything. Here we use `"Initialise project with a intro"` 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***).

Branching: the magical bit of *Git*

When we develop a project, we tend to create a branch for the stuff we are working on here. For instance, in a collaborative working environment of a **webpage**, somebody will take care of the **frontend** while others will be working with the **backend**, or a team will be responsible for the ***styling***, and the rest will write **unit tests**. So in this specific project, we can ***branch*** our project into a few branches:

- `frontend`
- `backend`
- `unit-test`
- ...

and they can start working simultaneously. For instance: a frontend engineer can start writing JavaScripts while the backend scripting is not finished. Other good thing about branching is that it makes sure your commits do not "*contaminate*" your `master` branch (the default/major branch) before its ready.

Chapter 1

Create and switch to branch `chapter-1`:

```
git checkout -b chapter-1
```

> Note that:
>
> `-b` flag accepts a parameter `<branch_name>` and create such a branch which is not existed yet. Switching to an existing branch does not require `-b` flag

Create `ch-1.md` and write in the following:

```
In 1775, a man flags down the nightly mail-coach on its route from London to Dover. The man is Jerry Cruncher, an employee of Tellson's Bank in London; he carries a message for Jarvis Lorry, a passenger and one of the bank's managers. Lorry sends Jerry back to deliver a cryptic response to the bank: "Recalled to Life." The message refers to Alexandre Manette, a French physician who has been released from the Bastille after an 18-year imprisonment. Once Lorry arrives in Dover, he meets Dr. Manette's daughter Lucie and her governess, Miss Pross. Lucie has believed her father to be dead, and faints at the news that he is alive; Lorry takes her to France to reunite with her father.

In the Paris neighbourhood of the Faubourg Saint-Antoine, Dr. Manette has been given lodgings by his former servant Ernest Defarge and his wife Therese, owners of a wine shop. Lorry and Lucie find him in a small garret, where he spends much of his time making shoes – a skill he learned in prison – which he uses to distract himself from his thoughts and which has become an obsession for him. He does not recognise Lucie at first but does eventually see the resemblance to her mother through her blue eyes and long golden hair, a strand of which he found on his sleeve when he was imprisoned. Lorry and Lucie take him back to England.
```

Add and commit

Chapter 2

Create and switch to branch `chapter-2`

Create `ch-2.md` and write something:

```
> Side notes: I don't know what to write but there should be a chapter 2, any thought?
```

Add and commit

Write a bit more for *Intro*

Add the following to `intro.md`

```
> Side notes: Damn that's a good intro
```

Add and commit

Commit history

```
git log
```

> ***Note that***: here are some useful flags:
>
> - `--oneline` : show each *commit* in one line: `<hash> <commit_msg>`
> - `--graph` : visualisation of the commit history
> - `--all` : show the history across all branches
>
> To quit the log, press `<q>`

Put them together

Go to `master` branch

```
git checkout master
```

Merge with `chapter-1`

```
git merge chapter-1
```

Same for `chapter-2`

Security, security, security, and `.gitignore`

You might notice that it is quite dangerous to commit a copy of your sensitive information to a *repo* or to a *remote repo*.

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

My little secret

At `master`, create `secret.md` and write:

```
> Don't tell anyone:
I copied those chapters from the Wikipedia page of "A Tale of Two Cities". Don't tell anyone, otherwise my careers is ****ed. Hope they don't do turnitin here
```

`.gitignore`

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:

```
.DS_Store
secret.md
```

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

> *And here comes a real story about poor Leo and his crypto-currency tokens...*

Ah Damn... I want to go back a bit

Assume you did something wrong but you committed it, and you don't want to commit a debugged version of it for some reason (like it will look stupid in your commit history), you might want to ***remove*** some commits from the history.

For instance, going back to the last commit:

```
git reset --soft HEAD^
```

> ***Note that***:
>
> `git reset` has several ***modes***(`--soft`, `--hard` and `--mixed`) and there are ways to manipulate the pointer to point at a commit that is higher up in the tree, using `^` and `~`. Please refer to the doc

Enough for *Git*, how about some *GitHub*: Remote *repo*

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:

```
You.watch(Tony.live_demo("How to use GitHub"))
```

Collaborative Coding 101

If that's not your own project

Obtain the repo

- Fork an existing project
- Clone the *your remote repo* to local

Make changes

- Make changes to *local repo*
- Add *remote forked repo* to `remote`:

```
git remote -v
git remote add upstream <upstream_url>
```

Push to remote

- `push` to *your remote repo*
- Make a pull request

Sync with forked repo

EITHER

- Keep your local *repo* synced with the *remote forked repo* and push to *your remote repo*

```
git fetch upstream
git merge upstream/master
git push -u origin upstream
```

OR

- Sync *your remote repo* to *remote forked repo* on *Github*
- `pull` from *your remote repo* to keep *your local repo* synced

If that's your own project

- Manage pull request
- Review changes, make comments, reject or `merge` pull request

Practice: Signing an attendance sheet

Let's sign an attendance sheet collaboratively!

The repo to the attendance sheet

BONUS: Git Internals

If we've got time, execute the following code for learning ***Git internals***

```
import ucl.dss.science.Tony;

public class GitDemo {
     public static void main(String[] args) {
          Tony.present(Git.internals);
     }
}
```

To Wrap Up

Git/Github is massive, I haven't figure out all of it as well. This is a brief introduction to the tip of this iceberg.

Official Documentation 📑

One More Thing...

Stay tuned for everything about data science

Subscribe to our offical IG if you haven't do so ☑️

And our FB! 🍾

And follow me on GitHub