Tecnologías Multimedia - Study Guide - Milestone 3: Git, GitHub and the Fork-and-Branch Git Workflow

Vicente González Ruiz

September 24, 2020

1. Description

To work in the InterCom project [4] you must understand the basics of Git [3] and GitHub[1] [1], and how to use the The GitHub (Work)Flow and the Fork-and-Branch Git Workflow [2].

[1] There are other Git-based hosting services such as GitLab and Altassian/BitBucket, but GitHub is the most used one.

2. What you have to do?

1. Have a look to the Pro Git book [3], what is the main source of information about Git.

2. If you don't have an GitHub account, please, do the the Hello World guide at GitHub and create one. Be aware that to contribute to InterCom an GitHub account is required. Notice that in the Hello World guide, the `master` branch is named the `main` branch. Please, create a `README.md` file for the Hello World repo, as the guide suggests.

3. Now, we are going to to the same that we would have done using the GitHub web interface (except the Step 1: Create a Repo), but now using the terminal which will be the most used interface for dealing with Git. First, if Git is not installed in your host (try to run `git` in a terminal), install it with:

```
sudo apt install git
```

4. Clone (download) the Hello World repo. You need to click on the "Code" button (select "https", not "download a zip file"). Then run:

```
cd hello_world
```

Notice that a new directory named as the repo's name at GitHub has been created, and that inside you can find the `README.md` file written in Markdown.

5. Create (and switch to) a feature branch called `improving_readme`. In your terminal write:

```
git checkout -b improving_readme
```

6. Modify the file `README.md`. Append to it, for example, a link to the Hello World guide. Use an ASCII editor (`nano`, for example):

```
nano README
```

And write:

```
See the [Hello World](https://guides.github.com
```

7. Commit your modification(s):

```
git commit -am "Providing the Hello World link"
```

In your first commit you will be prompted with:

```
git config --global user.email "your_email@exam
```

Please, input such information.

After the `commit`, your *local* repo is *ahead* of your *origin* (copy at GitHub of the) remote repo. This means that your *local* has modifications that the *origin* doesn't have.

8. Synchronize your *local* and the *origin* using push:

```
git push
```

Notice that if you have not upoaded a public SSH key (or the corresponding private key is not properly installed in your computer), the GitHub server requests your username and password, and this is something that is going to happen with every `push`. To avoid this repetitive input of your GitHub login information, you need to login at GitHub using public-key criptography. For that, you must own a

pair of keys, one public and other private, and upload the public one to GitHub.

9. The first step is to check whether you already have a pair of keys (if your are using the just installed Xubuntu distribution, obviously you don't need to check anything and can go directly to the next step). Simply revise your `$HOME/.ssh` directory with:

```
ls -l ~/.ssh
```

   and if you find a pair of files with almost the same name, and one of they finising in `.pub`, you probably own a pair of SSH keys.

10. Let's create a pair of keys (if you don't have one or if you prefeer to create a new one). Open a terminal and write:

```
ssh-keygen -t rsa -b 4096 -C "your_email@examp
```

    using the email address you provided when you created your GitHub account. Then, when you are prompted with:

```
Enter a file in which to save the key (/home/yo
```

just press the Enter-key, to select such output prefix. Otherwise, write a different one, but don't change the path to the `.ssh` directory.

11. Now SSH should request you for a passphrase. If you write one, you will be asked for it each time you push your commits to GitHub. There are two options to avoid this:

    (a) Input no passphrase (just by pressing the Enter-key again in the previous step). This has the drawback that if somebody steals your keys, he could access to GitHub as he were you.

    (b) Input a passphrase and configure `ssh-agent` to send it to GitHub by you. This option is the preferable one becase you will be asked for the passphrase only when the `ssh-agent` is started (Xfce does that by you).

12. Now it's time to check whether the `ssh-agent` is already running in your computer. This can be done with:

```
ps aux | grep ssh-agent
```

and in the case of Xubuntu, you should get something similar to:

```
 989 ?        Ss     0:00 /usr/bin/ssh-agent /usr/bin/im-l
1433 pts/0    S+     0:00 grep --color=auto ssh-agent
```

This means that there are two processes in whose description there exists the string `ssh-agent`. The first entry is the agent process. The second one is the `grep` running at the same time that the `ps`.

13. If the `ssh-agent` were not running, it can be launched to run in the background with:

```
eval "$(ssh-agent -s)"
```

but you don't need to do that in your Xubuntu installation, because (remember) the `ssh-agent` the Xfce desktop environment lauches it.

14. With your keys, run:

```
ssh-add ~/.ssh/id_rsa
```

and the passphrase will be prompt.

15. Go now to GitHub -> Settings -> SSH and GPG keys -> New SSH key. Open a terminal and write:

    ```
    cat .ssh/id_rsa.pub
    ```

    and copy and paste the content of such file (which ends with your email address) inside of the space where you can read "`Begins with 'ssh-rsa', ...`". Don't forget to give a title (something such as "`tm`" (tecnonogías multimedia)) to the key pair.

16. When you use the key for the first time (clonning a repo or pushing a commit), the SSH client will warn you that the autenticity of `github.com` cannot be established. This is normal and should happen only once. Type `yes`. If this problem persists, then you could be suffering a man-in-the-middle attack.

17. Revise The Fork and Branch Git Workflow. Basically, this "protocol" explains that to contribute to an open-source repo hosted by GitHub without belonging to the develop team, you must do some `git`-steps (and some of them are below).

18. Make a fork of the InterCom project. We will call to this repo the *upstream*, whose URL is

    `git@github.com:Tecnologias-multimedia/intercom.git`

    This info can be found when you clone the InterCom. Notice however, that action of clonning the InterCom is a waste of time because you cannot contribute directly to it (remember, you must `clone` your own repo).

19. Add the remote[2] *upstream* with:

    ```
    git remote add upstream git@github.com:Tecnolo
    ```

    Check that everything has worked with:

    ```
    git remote -v
    ```

[2] Git is a decentralized control system for source code. Decentralization means that every developer has a copy of the *origin*, and that thus, the developers can synchronize their *locals* with any of the *remotes*.

where you should see two remotes: *origin* and *upstream*. Something similar to:

```
origin   git@github.com:Tecnologias-multimedia/intercom.gi
origin   git@github.com:Tecnologias-multimedia/intercom.gi
upstream git@github.com:you_at_GitHub/intercom.git (fetch)
upstream git@github.com:you_at_GitHub/intercom.git (push)
```

3. Timming

You should reach this milestone at most in one week.

4. Deliverables

None.

5. Resources

References

[1] GitHub.

[2] The Fork and Branch Git Workflow.

[3] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2020.

[4] The students of Tecnologías Multimedia at the UAL. The InterCom project.