Tecnologías Multimedia - Study Guide - Milestone 1: Git, GitHub and the Workflow

Vicente González Ruiz - Depto Informática - UAL

September 13, 2020

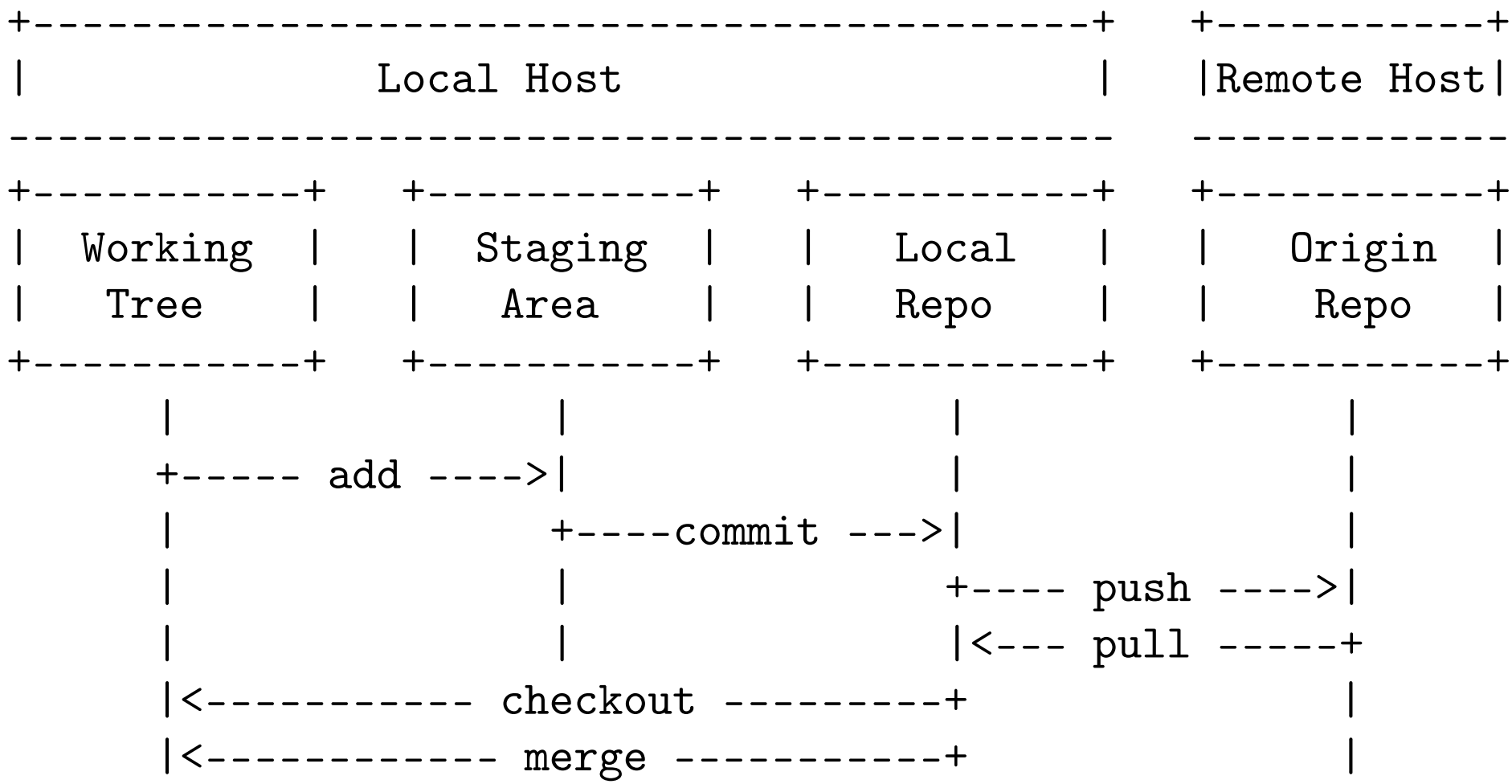

Figure 1: Structure-of and actions-on a Git repo.

1. Description

To work in the InterCom project [4] you will need to 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].

You should know that:

[2]

1. If you want to contribute to a repo (that we will call *the upstream repo* or simply *upstream*) [**?**] and you don't belong to the team of developers (such as it happens in he InterCom project), the first action that you must do is to fork the *upstream*. This will create a copy of the repo in your GitHub space, and the one thing that distinguish it from a repo directly created by youself is that there is a link to the *upstream*.

---

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

[2] Technically, a commit copy the modifications performed in your *working copy* (of your local repo) to the *staging area* (of your local repo).

2. You can do a lot of work in a repo hosted by GitHub using the Web interface, but it is much more convenient to clone the repo.[3]

3. When you copy the *origin* in your local disk, you create the data structure shown in the Figure 1. Such structure has 3 main parts: (1) The *working tree*, (2) the *staging area*, and (3) the *local repo*. The modifications are carried on the *working tree*, which can contain files and directories that *untracked* (probably temporal files that never should be included in the repo). The *staging area* (also called *index*) is form by the subset of tracked files of the *working tree*. The *local repo* is the data structure that holds all the history of the repo. When the *local repo* and the *origin* are synchonized, their contents are the same.

4. To update your *local repo* with the modifications carried on the *working tree*, you must perform a commit.

[3] Git is a decentralized control system for source code. Decentralization means that every developer has a copy of the *origin repo* (or simply *origin*, the first instance of the repo), and that thus, the developers can synchromize their local repos with the *origin*(al one) or any of the rest of developer's repos, by simply adding the right *Git remotes*.

5. At any moment, you can decide to synchronize your local repo with the *origin* or any other *remote*. This action is called a push.

6. Also, at any moment, you can decide to share your modifications (or even only your thoughts) with the *upstream* (developers). For that, you can do a pull-request.

You should also understand what is a Git branch:

1. *Main branches*, that have infinite lifetime. All repos have a *master* branch, which should be always deployable. Some have a *testing* branch, in which the code is tested before it is included in the master (stable) branch.

2. *Feature/debug branches*, which are aimed to develop a new functionality or to debug. Feature branches are also called topic branches. These type of branches have usually a short life.

2. What you have to do?

1. Have a look to the Pro Git book [3], what as the main source of information for 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 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 "the Git protocol", not "https" nor "download a zip file").

```
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` Markdown file.

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 "you@example.com"
```

   Plase, 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.

   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 goint to happen with every push. To avoid this repetitive input of your GitHub login information, you need to identify you at GitHub using public-key criptography. You need to own a pair of keys, one public and other private, and upload the public one to GitHub.

9. The first step to solve this drawback 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 finished with `.pub`, you have 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 the GitHub account. Then 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). This has the drawback that if somebody has 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.

12. Now it's time to check whether the `ssh-agent` is already running in your computer. This can be checked 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 /u
1433 pts/0    S+     0:00 grep --color=auto ssh-
```

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 the `ssh-agent` the XFCE 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 "tec-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.

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 clonning the InterCom is a waste of time because you cannot contribute directly to it.

19. Add the remote *upstream* with:

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

Check that everything has worked with:

```
git remote -v
```

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

3. Timming

You should reach this milestone at most in one week.

4. Deliverables

None.

5. Resources

[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.