# TD02 - Git & Github

## Prerequisites

Even if it seems pretty usual to use Git in the development world, not every project is managed with this tool. The main goal here is to have you create and set up a Github account before using it for further purposes. Git will be required as well as it is a must have. You might want to start with

## Sign up to Github

First step is (if not already done) to sign up to Github with your CPE mail address: go to https://github.com/join and fill the required information. Although you are already reach IRC students, we recommend you to use an individual free plan for the next steps of this project. You can eventually fill the last page but it's not really important. Select "Complete setup".

There you are, your (probably not first) Github account is set up. Yay ! Now, let's move on to the next step !

## Project forking and publishing

For this part, we are going to fork the project that will be used for the rest of the lesson (I mean, till the end of the week). Go to and fork the project :

https://github.com/githubtraining/hellogitworld

Now you own the project under your Github workspace, you can basically do whatever you want on this project. However we recommend not to modify the whole java and maven content if you still want this project to compile.

First of all, make sure the git CLI is installed on your computer and then clone the project on your computer to be able to modify it locally.

## Securing Github access

There are actually two different ways of cloning and publishing a project. By default, Github will propose you to clone by HTTPS link. Copy to clipboard, then open a new terminal and enter :

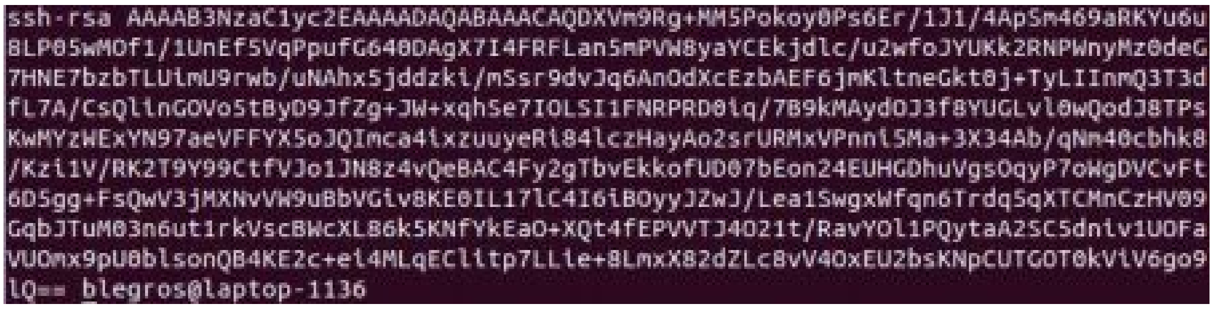
```
$ git clone <project_url_with_https>
```

Git will probably ask you to authenticate in order to be able to clone the repository. It will ask you the same thing every time you want to publish your work on a branch. This might be painful and you don't want to do this.

The second option is "use SSH" and the link starts with "git@github.com:…", but there is a prerequisite to use this solution, you'll need to create an SSH key and have it added to your account. Fine, then tape:

```
$ ssh-keygen -t rsa -b 4096 -f ~/.ssh/{theNameOfYourKeyPair}
```

It will ask you to enter and confirm a passphrase, this is for security purposes but we will let it empty for this course. Well done, you've generated a new RSA key pair of 4096 bits size. If you do "ls ~/.ssh" you'll see new files inside your folder, one is named theNameOfYourKeyPair and the other one theNameOfYourKeyPair.pub.The first one is your private key to NEVER communicate to anyone and the second one is you public key. Let's take a look to this last one, enter "cat ~/.ssh/theNameOfYourKeyPair.pub":



```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDXVm9Rg+MM5Pokoy0Ps6Er/1J1/4ApSm469aRKYu6u
8LP05wMOf1/1UnEf5VqPpufG640DAgX7I4FRFLan5mPVW8yaYCEkjdlc/u2wfoJYUKk2RNPWnyMz0deG
7HNE7bzbTLUimU9rwb/uNAhx5jddzki/mSsr9dvJq6AnOdXcEzbAEF6jmKltneGkt0j+TyLIInmQ3T3d
fL7A/CsQlinGOVo5tByD9JfZg+JW+xqhSe7IOLSI1FNRPRD0iq/7B9kMAydOJ3f8YUGLvl0wQodJ8TPs
KwMYzWExYN97aeVFFYX5oJQImca4ixzuuyeRi84lczHayAo2srURMxVPnni5Ma+3X34Ab/qNm40cbhk8
/Kzi1V/RK2T9Y99CtfVJo1JN8z4vQeBAC4Fy2gTbvEkkofUD07bEon24EUHGDhuVgsOqyP7oWgDVCvFt
6D5gg+FsQwV3jMXNvVW9uBbVGiv8KE0IL17lC4I6iBOyyJZwJ/Lea1SwgxWfqn6Trdq5qXTCMnCzHV09
GqbJTuM03n6ut1rkVscBWcXL86k5KNfYkEaO+XQt4fEPVVTJ4O21t/RavYOl1PQytaA2SC5dniv1UOFa
VUOmx9pU0blsonQB4KE2c+ei4MLqEClitp7LLie+8LmxX82dZLc8vV4OxEU2bsKNpCUTGOT0kViV6go9
lQ== blegros@laptop-1136
```

Something like this will appear on you terminal, this is the content of your public key that you will communicate to Github. Copy the hole content and past it to you Github account under "Settings" and "SSH and GPG keys". Click on New SSH key and paste the content
of your public key. Give it a name and validate the operation. Now try to clone the repository again with the git@ prefix. It will ask you to select a key pair to perform the action. take the one you've just indicated to Github and press enter. Now you are able to clone and publish work on your Github repository without entering a password every time,  I hope you enjoy this.

## Let's publish

Open the project inside your favorite IDE (I hope it's IntelliJ) and open the file README.md. Modify this file entering, for example "This project is now mine". Save it and check that Git has correctly seen you changes

```
$ git status
```

You'll see you file colored in red. This means that Git has seen you've made some modifications to this file, but it will not take them into account once you will publish them. Then ask git to add them to your work.

```
$ git add .
```

Actually, we did not ask him to add our file, but to add any modification made to any file inside our working directory. Now if you enter "git status" again you'll see that your file is colored in green. You work will be taken into account, hopefully. Let's commit this work:

```
$ git commit -m "The message of your commit"
```

Now if you try to "git status" again you'll see that your workspace is "clean". Git created a new reference with all the changes you've made. If you go on and enter:

```
$ git log
```

You'll see the message of you last commit on top of the references. However you cannot see the changes on the Github website because we did not publish yet our work. Let's do it !

```
$ git push origin master
```

This command literally means "I want to publish my work on the distant/remote branch master". And now you can see that your work is published online ! Big up guys !

## Configure your repository

Git is one of the most useful tool you'll find in your developer life. Almost everybody uses it and most of the time you'll have to work with other people on project using Github. However you'll find many people that use it wrongly, and many people that will create things you don't want to merge in you production branch. Let's secure a bit our labor to prevent any fool to throw it away.

Go back to your project on the Github webpage and click on settings. Go to Branches and you'll see that your default branch is master. Fine, it means that every time you connect on your repository, this branch will be displayed. Just under this indication, you’ll see a Branch protection rule. Try to add one.

You’ll see a bunch of options, most of them are very useful working in team (especially asking for pull request and review before merging inside master branch). You can also select options to block push force (when someone does push -f) because it doesn’t take care of Git warning messages that usually prevent you from pushing. As you are working alone on this project we will only add the name “master” to the naming pattern and let the rest as it is. It will only prevent you from doing bad things on you master branch.

Finally, be aware that all the work you do on Github is public by default. Therefore you should or you must NEVER publish any password on your repository. Thankfully you can turn your repository to private from the options and there are Environment Variables that you can set and secure (I mean encrypt) inside your Github repository under Secrets.

## Git basic commands

### Clone a project

```
$ git clone <url_of_the_project>
```

### Fetch distant modifications without merging them into your branch

```
$ git fetch -p
```

### Fetch distant modifications and merge them into you branch

```
$ git pull
```

### Add your changes to the workspace

```
$ git add .
```

### Commit your changes

```
$ git commit -m “Your message”
```

### Publish your changes

```
$ git push origin <name_of_the_remote_branch>
```

### Merge a branch into yours

```
$ git merge <name_if_the_branch>
```

### Rebase your branch on top of another

```
$ git rebase <name_of_the_branch>
```

## Git config

Your local git configuration is located in your user home, `~/.gitconfig`, you can also print the config with the command: `git config --list,` add the aliases that you find handy to your config.

```
[user]
 name = John Doe
 email = jdoe+git@example.com
[pull]
 rebase = true
[push]
 default = simple
[core]
 editor = vim
[alias]
 tree = log --graph --decorate --pretty=oneline --abbrev-commit --all -- full-history
 co = checkout
 st = status
```

There are 3 levels of config for git, those would be overloaded by the more specific one (ie: local overloads user, user overloads system):

- system, for all users of a system (on linux in /etc/gitconfig)

- global or user, for a specific user (~/.gitconfig)
- local, for a specific repository (path/to/repo/.git/config)

## Bonus

### Stash

Git provides a useful command: stash, this can be used to quickly save some changes before moving on the tree or try different solutions to a problem without committing or branching. Stashed modifications are stored in a FILO stack, give a try to a couple commands: https://git-scm.com/docs/git-stash.

### .git/ directory

Everything git has ever done and will ever be doing is simply stored in the .git/ directory, this is git hidden database. If you want to save the full history of a git repository you can just save this directory. Although I would not advise AT ANY COST editing any of the files in there, I encourage you to discover how git is working from the inside, connecting commit hash to object directories (.git/objects) etc. Indeed this software is very smart and quite simple at the same time, let’s not forget Linus Torvalds its father.

### Rewriting history

Once you have the history written in git it can be quite hard to change the past without jeopardizing the repository. But sometimes you may want to remove a bit of dust from your tree, just because you want everything picture perfect or just because the 55 commits with: fix, fix1, fix2, …, fix final, fix final12, fix “I am sick of this” just look awfully bad. The command git rebase -i will help you, give it a try.