Week 7 - Practical

Using Git Command Line

Outline

1. Install Git Command line
2. Create a new repository
3. Initialize a local repository on your machine
4. Check the status of your files.
5. Push files to the remote repository
6. Collaborate with another teammate
7. Deal with Conflicts

Step 1: Install Git Command Line

Installation on Windows

- Download Git from *Git for Windows* at http://msysgit.github.io/

- Install it.

Installation on Mac (1/2)

1. **Install *Homebrew***

(ONLY IF YOU HAVE NOT INSTALLED IT ALREADY)

Copy & paste the following into the terminal window and **hit *Return***

```
ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master
/install)"
```

```
brew doctor
```

You will be offered to install the *Command Line Developer Tools* from *Apple*.

Confirm by clicking Install. After the installation finished, continue installing Homebrew by **hitting Return again**.

Installation on Mac (2/2)

2. Install *Git*

Copy & paste the following into the terminal window and **hit *Return***

```
brew install git
```

You can use Git now.

Installation on Linux

- **Debian-based Linux systems**

  Copy & paste the following into the terminal window and **hit *Return***

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install git
```

- **Red Hat-based Linux systems**

  Copy & paste the following into the terminal window and **hit *Return***

```
sudo yum upgrade
sudo yum install git
```

Check that your installation was successful (on MacOS/Linux only)

Type the following command on the console

```
git --version
```

It should print the git version that you have currently installed. For example, you should visualize a message similar to the following one:

```
git version 2.20.1 (Apple Git-117)
```

Step 2: Create a New Repository

Create a Repository

Repository

The place where developers store all their work

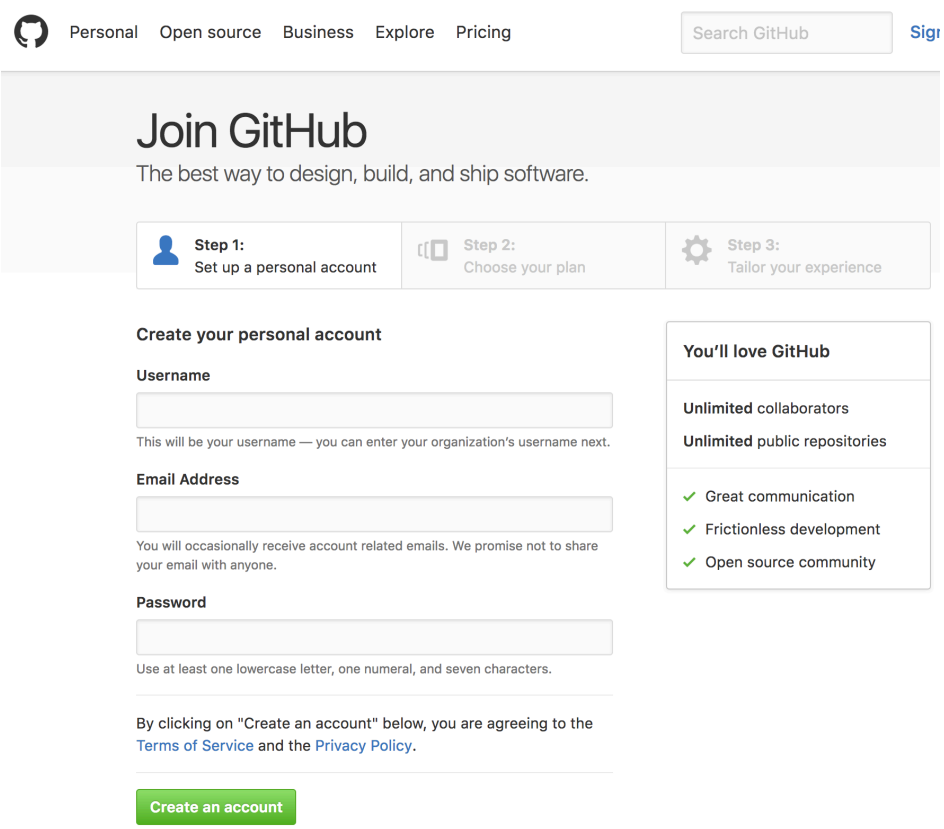
Not only stores files but also the history of changes

Accessed over the network

- To host a repository we will use **GitHub** which is a web-based hosting service for version control using Git.

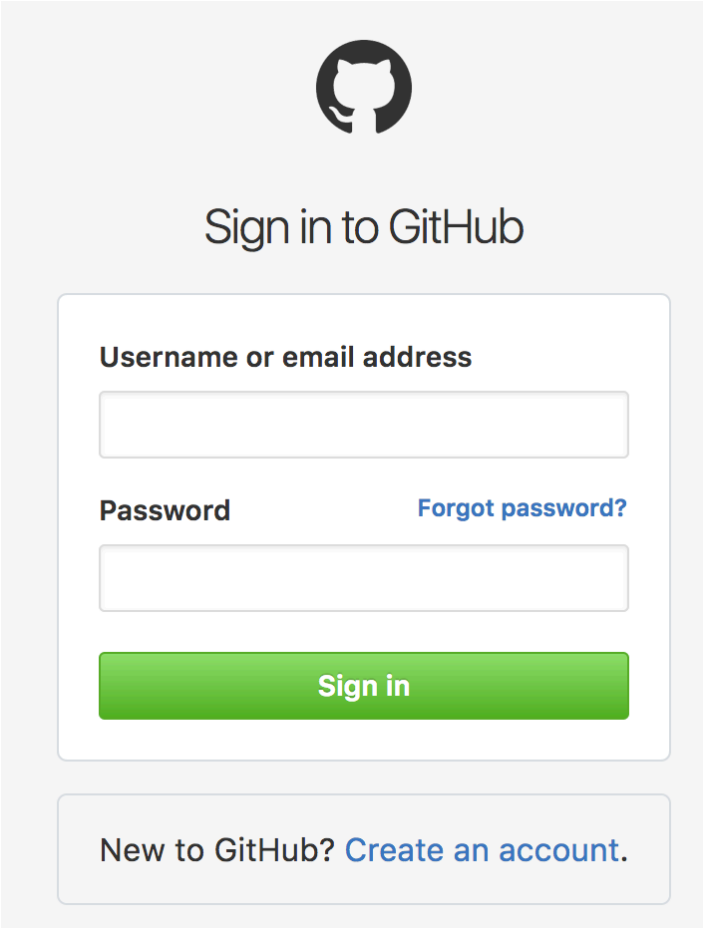
Create a GitHub account

- Open your favourite browser and go to https://github.com/join?source=login
- Fill the fields necessary to register and click on “Create an account”



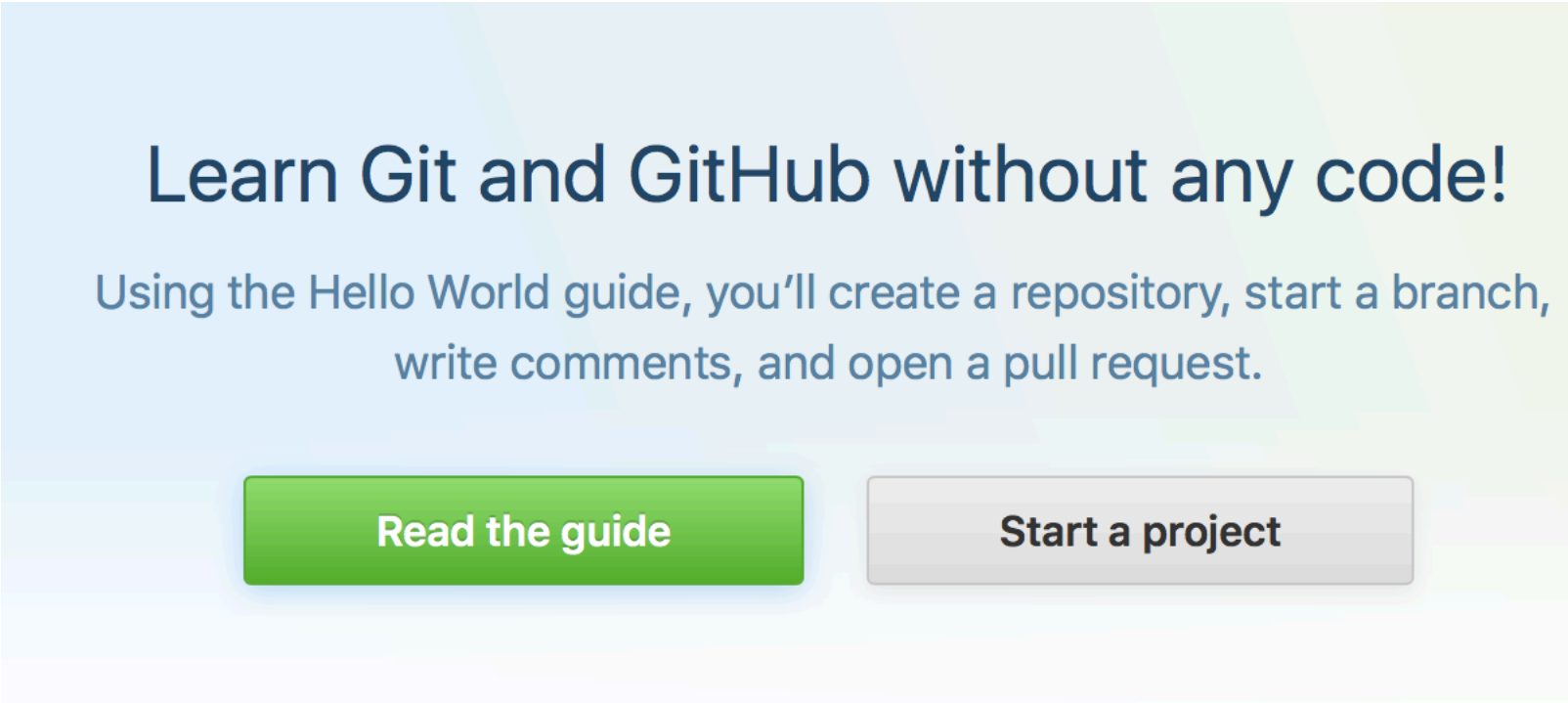
Login to Github

- Go to https://github.com/login
- Input the username and password you used to sign up on GitHub



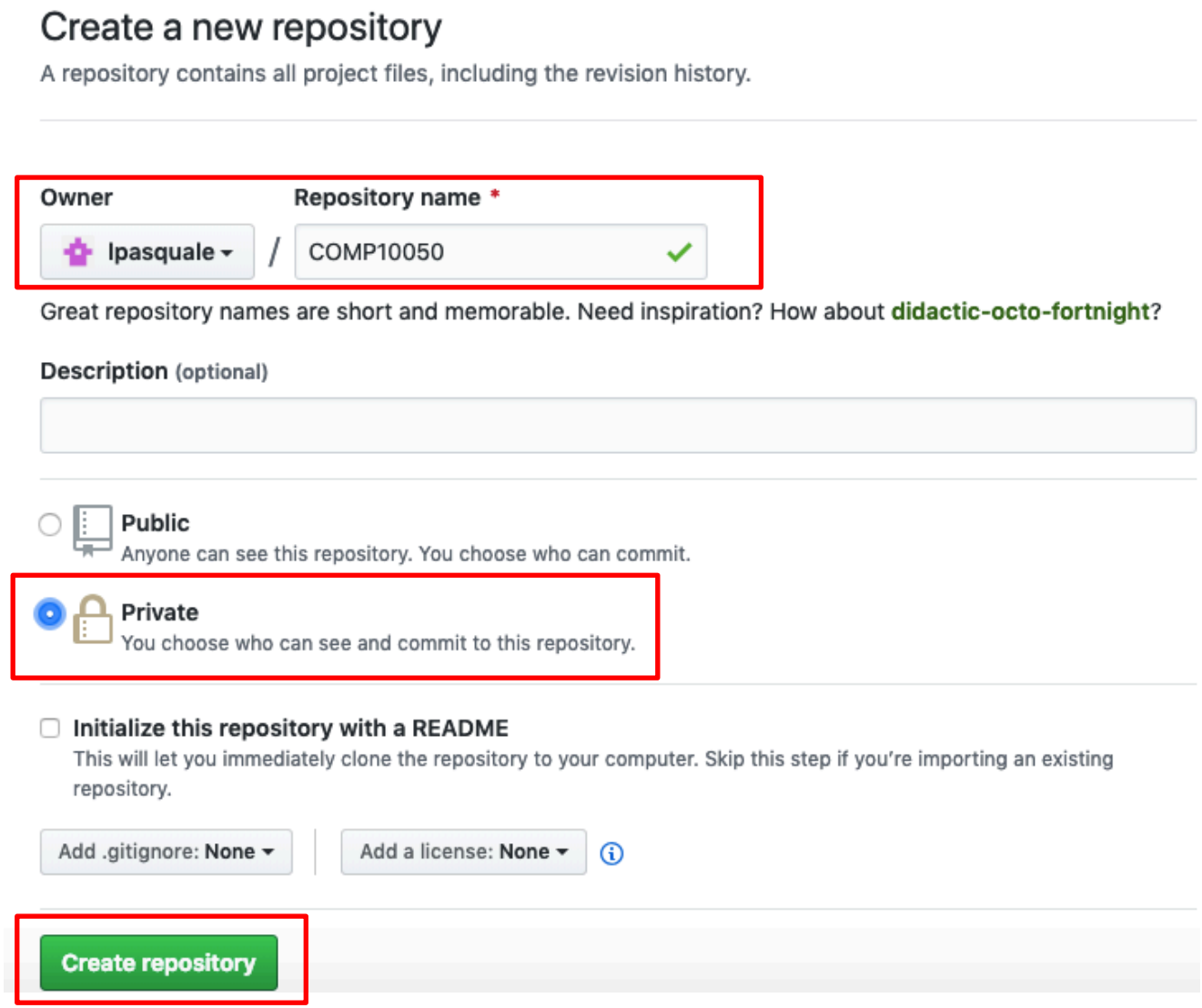
Create a New Repository

- Click to “Start a Project”



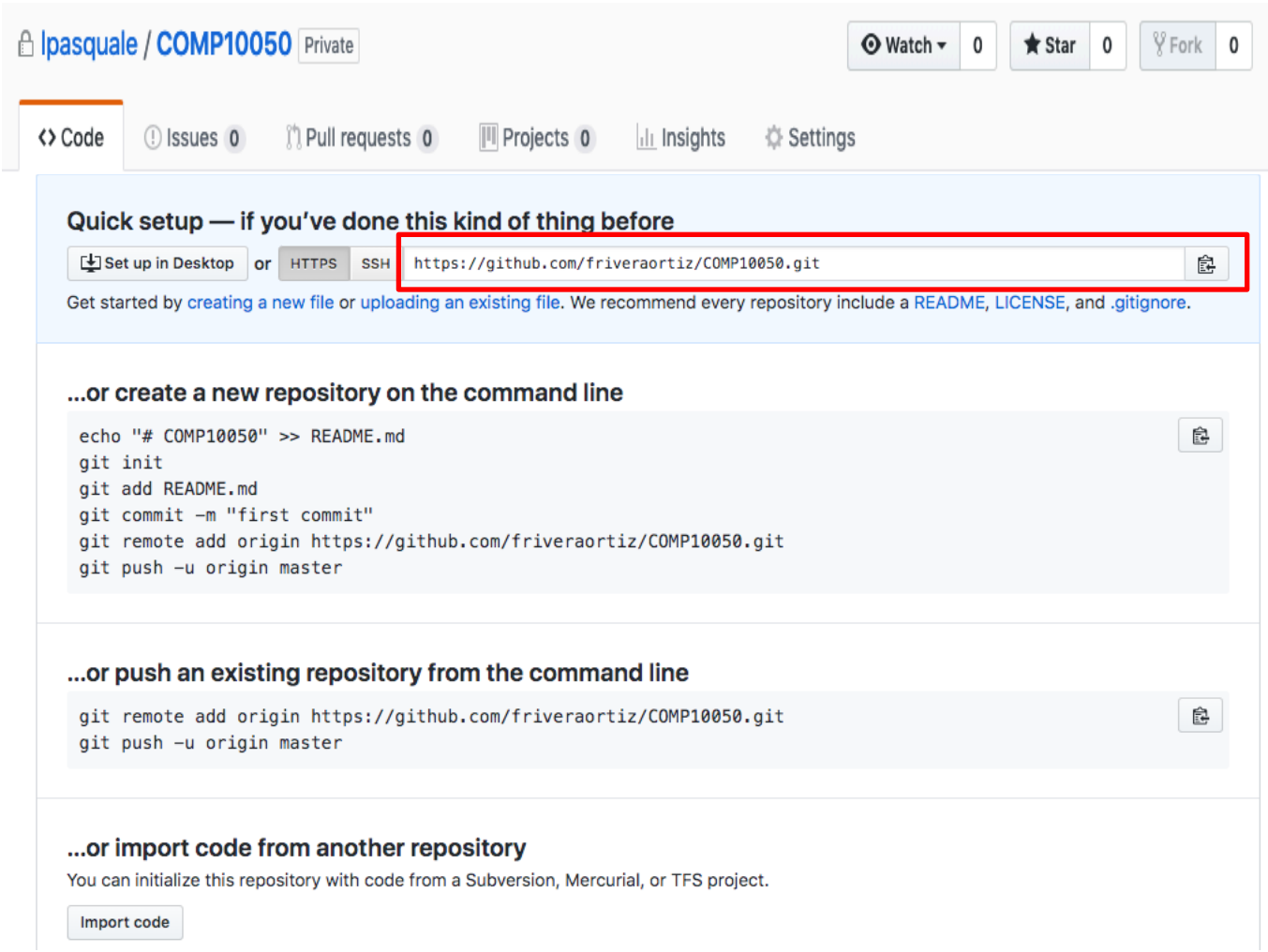
Create a New Repository

- Insert a repository name (e.g., COMP10050).
- Set the repository as private
- Click “Create Repository”



Create a New Repository

Here in the red square we can visualize the GitHub URL link of the repository, which will be our Remote Repository on the Internet.



Step 3: Initialize a Local Repository on Your Machine

Assumption From Now On

- **If you are on a MacOS or a Linux platform** from now on you will need to use the Terminal.

- **If you are on a Window platform** you will need to use **Git Bash** which you must have installed when you installed Git SCM in step 1.
  - **Git Bash** is a run emulation used to run Git from the command line.

Configure Git

- Configure your identity, i.e. username and email address.
  - Note that your username and email address should be the same as those used to sign up on GitHub.

For example:

```
git config --global user.name “lpasquale”
git config --global user.email liliana.pasquale@ucd.ie
```

Now you need to Initialize the Repository on Your Machine

- First of all, create a directory in your computer using either Finder in MacOS or File Explorer in Windows where you want to place your repository (e.g., COMP10050)

Initialize Your Repository

- From the Terminal/Git Bash go to the directory you created. For example,

```
cd /Users/liliana/COMP10050
```

Change the filepath above with the one you created in the previous slide.

- Then, initialize your repository typing the following command

```
git init
```

This command creates the .git subdirectory where all the versions of your project will be stored.

Add Files to Your Local Repository

- In the terminal create a new file named `README.md` containing the string “# COMP10050”. You can do so from the terminal typing the following:

```
echo "# COMP10050" >> README.md
```

- Add the file to the staging area (command *add*) and create a new version by committing the changes to the local repository (command *commit*)

```
git add README.md
git commit –m “first commit”
```

Note that when you do a commit you always need to provide a message describing the changes you have applied.

Synchronize the Local Repository with the Remote One on GitHub

- In the terminal, inside the directory containing your repository, type the following:

```
git remote add origin
https://user@github.com<remoteUrl>
git push –u origin master
```

- Note that `<remoteUrl>` should be replaced with the GitHub URL of your remote repository.

For example:

git remote add origin
https://lpasquale@github.com/Users/liliana/COMP10050.git

- Check slide 15 to see how to view the URL of your GitHub remote repository

Step 4: Check the Status of Your Files

Unmodified Files

- In your terminal go to the directory of your local repository and type:

```
git status
```

It should show that your branch is up-to-date:

- no tracked files (i.e. files staged for commit)
- no modified files (files modified but not yet staged for commit)
- no untracked files (newly added files not staged for commit)

For example:

```
dhcp-892b19cc:COMP10050 lpasqua$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Modified Files But Not Staged

- Now, add a new file to your local repository (e.g., a simple text file called Lab.txt) by typing the following

```
echo 'My project' > Lab.txt
```

- Now type the status command

```
git status
```

It indicates that Lab.txt has not been selected to be included in the next version of your project in the next commit).

```
dhcp-892b19cc:COMP10050 lpasqua$ 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)

        Lab.txt

nothing added to commit but untracked files present (use "git add" to track)
dhcp-892b19cc:COMP10050 lpasqua$
```

Staged Files

- Go to your local repository directory and type

```
git add Lab.txt
```

- Type the status command again to see that the Lab.txt is now tracked and staged to be committed

```
git status
```

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

        new file:   Lab.txt
```

- You can tell that it is staged because it is under “Changes to be committed” heading.

- The file will be placed in the next version of your project when you perform a commit.

Committing Your Changes

- You can type your commit message inline with the `commit` command by specifying it after a `-m` flag.

```
git commit –m “Lab.txt added”
```

```
[master 5ceb8e0] Lab.txt added
 1 file changed, 1 insertion(+)
 create mode 100644 Lab.txt
```

Remember that only the files that are staged (i.e., you have run **git add** on since you edited them) will go into this commit.

Step 5: Push files to the remote repository

Committing Your Changes Does Not Push them to the Remote Repository

Remember that performing commits on your local repository will not propagate the new version of your project on the remote repository hosted on GitHub.

- You can verify it logging in GitHub and opening your project.
- You won’t be able to see file “Lab.txt”

Pushing to a remote repository

Git push is essntially the transfer of your local information to the remote repository.

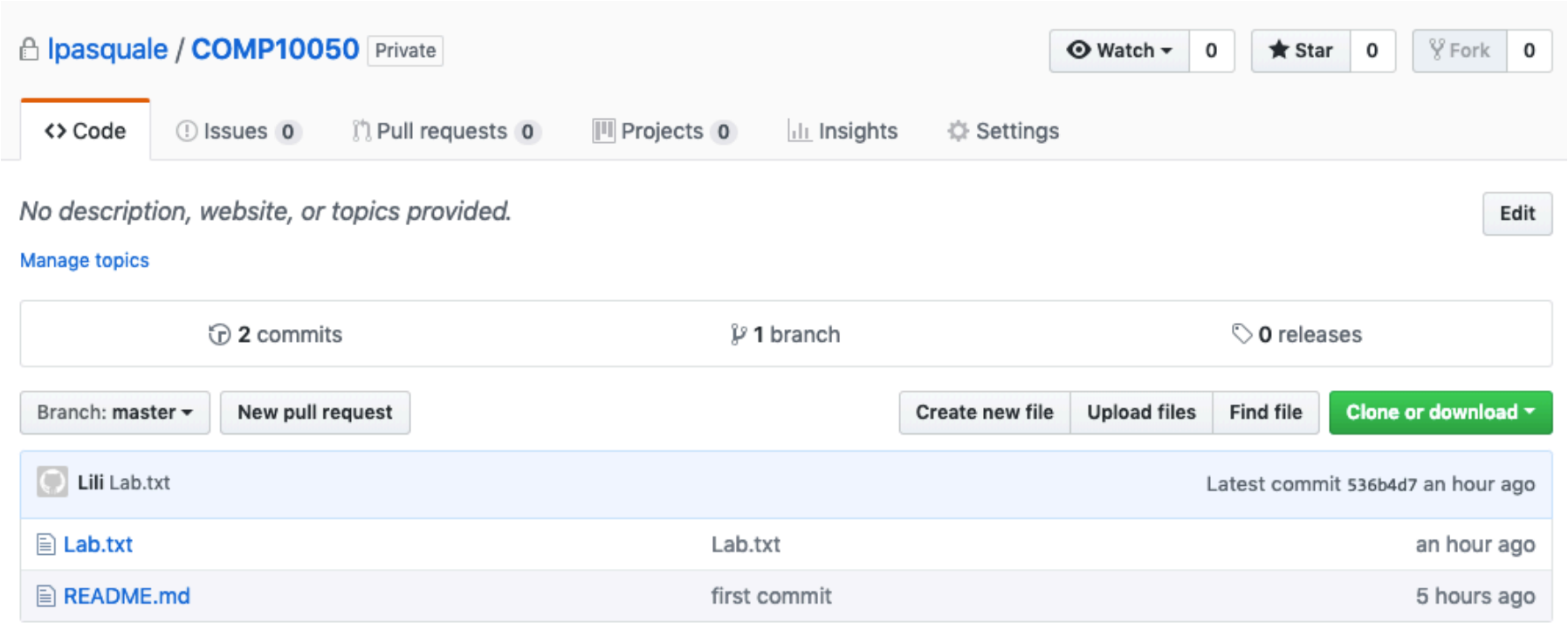
If you open the terminal and go inside the directory containing your project and type the following

```
git push
```

- This command will execute correctly only if you have committed your changes locally.
- git push will send all the local changes to the remote repository

Pushing to a remote repository

- You should now be able to see file Lab.txt in your remote repository on GitHub



Removing Files (1/2)

To remove a file from Git, you have to remove it from your tracked files (i.e. remove it from the staging area) and then commit.

- Remove file Lab.txt typing the following

```
rm Lab.txt
git status
```

If you simply remove the file using command `rm`, the file will show up under "Changed but not updated" (that is the unstaged area).

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

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    Lab.txt
```

Removing Files (2/2)

- The **`git rm`** command instead also removes the files from your working directory.
- Type the following to stage the file's removal

```
git rm Lab.txt
```

- Alternatively to remove the file from git, but still keeping it in your working directory. Type:

```
git rm --cached Lab.txt
```

- The file will be removed in the next commit and the changes will be propagated to the remote repository

```
git commit –m "Lab.txt removed"
git push
```

Step 6: Collaborate with Another Teammate

Work in a group of 2

- First, pair with a colleague who is sitting next to you.
- One of the persons in the pair can add the other as a collaborator to his/her project.

Assign roles:

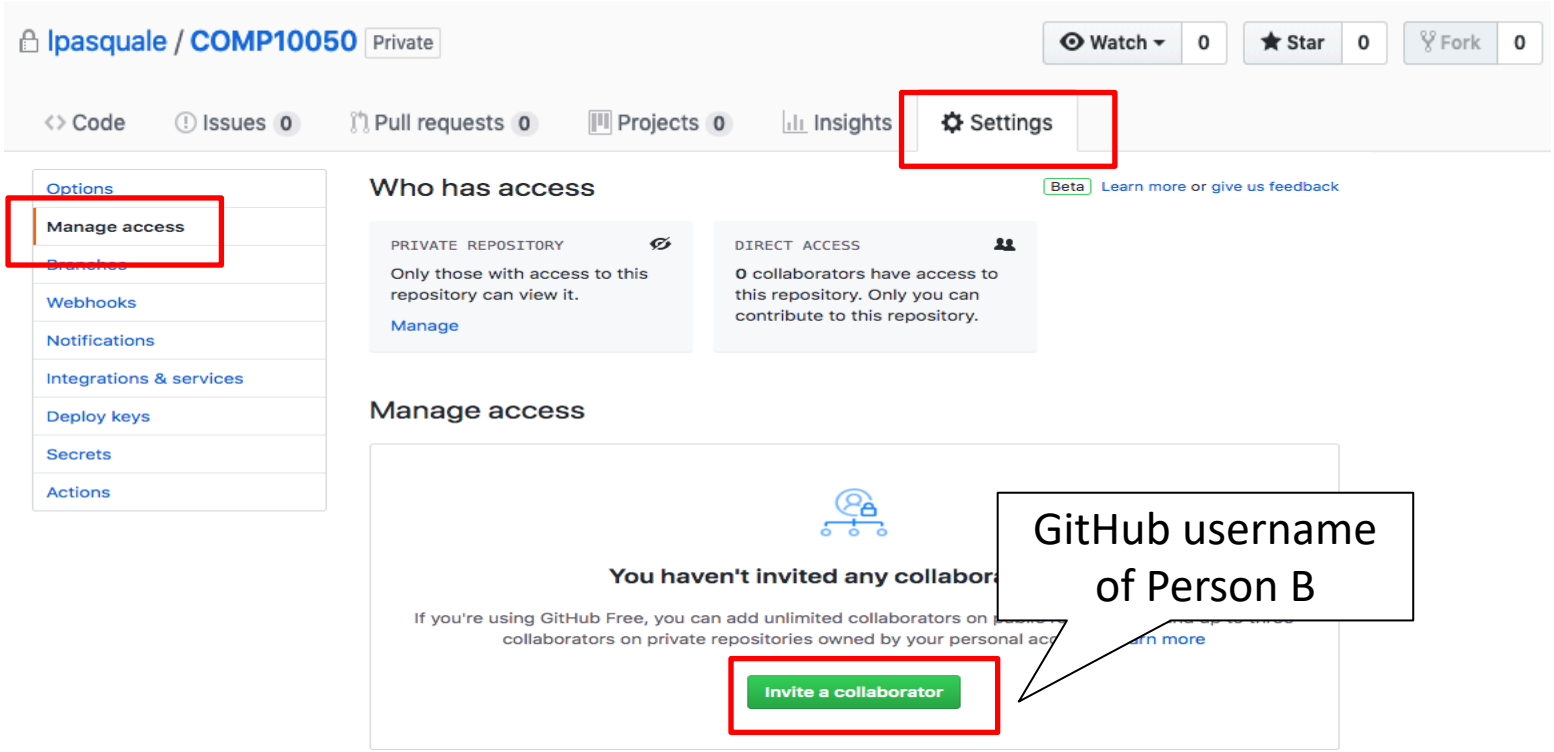
- One person in the pair will impersonate **Person A** (i.e. the one that adds a collaborator)

- The other person in the pair will impersonate **Person B** (i.e. the one that is added as a collaborator).

Cloning an Existing Repository (1/3)

Get a copy of an existing repository you would like to contribute to.

- Person A should invite Person B to collaborate to his/her project
- In the GitHub page of the project **Person A** should select Settings >> Manage Access and type the GitHub username of Person B in the tab below



Cloning an Existing Repository (2/3)

Now Person B should do the following

- Accept the invitation sent by Person A
- In the terminal, create a new directory where to place the new repository

```
cd ..
mkdir newRepository
cd newRepository
```

- Clone the new repository

```
git clone https://user@github.com <other_GitHub_url>
```

- Note that `<other_GitHub_url>` should be replaced with the GitHub url of the repository of Person A. For example:
  git clone https://lpasquale@github.com/Users/liliana/COMP10050.git

Cloning an Existing Repository (3/3)

- Person B should now be able to see inside directory `newRepository` the content of the repository created by person A (e.g., file README.md). To do so Person B should type the following

```
ls
```

Person A modifies the Repository

- Person A now opens file README.md and modifies it, for example, by adding new text.

- Person A subsequently commit changes in the local repository

```
git add README.md
git commit –m “README modified”
```

- Person A subsequently push the changes in the remote repository

```
git push
```

Person B retrieves Remote Changes

- Now if Person B opens the file README.md inside folder *newRepository,* s/he won't be able to view the changes that Person A has performed.

- To retrieve the changes Person A has performed it will be necessary to enter directory `newRepository` from the terminal and type

```
git pull
```

- Now if Person B opens file README.md inside folder newRepository, s/he will be able to view the changes that Person A has applied.

Step 7: Deal with conflicts

In this exercise you will merge 2 branches that contain conflicting files.

Clone a New Repository

Now create a new folder outside your Git local repository

```
mkdir git-conflicts
```

Clone a new repository:

```
git clone https://github.com/hcs/bootcamp-git.git
```

In your new repository go to the master branch

```
git checkout master
```

You should visualize the following message:

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

Edit Conflicting Files (dog.c)

1) Open file dog.c in your repository

2) Edit the file to make it look like the one below:

```
puppy
canine
wolf
bark
bow wow
corgi
shepherd
```

3) Add the file to the staging area and commit:

```
git add dog.c
git commit –m “edited dog”
```

Edit Conflicting Files (dog.c)

1) Now change branch (origin/merge-exercise)
   `git checkout origin/merge-exercise`
2) Edit the file dog.c to make it look like the one below:

```
puppy
canine
wolf
bark
bow wow
pikachu
charmander
squirtle
bulbasaur
charmelon
charizard
ivysaur
venasaur
wartortle
blastoise
```

3) Add the file to the staging area and commit.

Two conflicting files

dog.c in the master branch

```
puppy
canine
wolf
bark
bow wow
corgi
shepherd
```

dog.c in the origin/merge-exercise branch

```
puppy
canine
wolf
bark
bow wow
pikachu
charmander
squirtle
bulbasaur
charmelon
charizard
ivysaur
venasaur
wartortle
blastoise
```

Merging the master and the origin/merge-exercise branch will cause conflicts because there are 2 different versions of file dog.c

Merge master with origin/master

1) Return to the branch repository

```
git checkout master
```

2) Merge the master with origin/merge-exercise repository

```
git merge origin/merge-exercise
```

You should visualize the following message

```
Auto-merging dog.c
CONFLICT (content): Merge conflict in dog.c
Automatic merge failed; fix conflicts and then commit the result.
```

Visualize the Conflicts

1) Open file dog.c

```
puppy
canine
wolf
bark
bow wow
<<<<<<< HEAD
corgi
shepherd

=======
pikachu
charmander
squirtle
bulbasaur
charmelon
charizard
ivysaur
venasaur
wartortle
blastoise
>>>>>>> origin/merge-exercise
```

Different parts present in the dog.c file in the master branch

Different parts present in the dog.c file in the origin/merge-exercise branch

Resolve the conflict

A possible way to resolve the conflicts would be to edit the file dog.c as follows:

```
puppy
canine
wolf
bark
bow wow
pikachu
charmander
squirtle
bulbasaur
charmelon
charizard
ivysaur
venasaur
wartortle
blastoise
```

Commit the Changes

On the command line, execute the following

```
git add dog.c
git commit –m “merge my dogs”
```

Congratulations, you resolved the Conflicts and Completed the Git Lab!