Exercise 1 Gitlab

PART A

On the GITLAB server (remote repository):

- Create a project named **Project**
- Create the changelog.txt file in the main branch **Main**
- Edit the changelog.txt file so that it contains this 1st line:
  "Version 1.0: First version"
- Create 2 branches from the main main branch : **dev** and **test**
- Create 2 branches from the **main, de**: **dev1** and **dev2** branches
- Create and then delete a **Dev3 branch**

PART B

On your machine:

Open a terminal (or X console) and prepare the working directory:

```
$ mkdir ~/TP_GIT
$ cd TP_GIT
$ mkdir DEV1@paris
$ mkdir DEV2@nancy
$ mkdir DEV@metz
$ mkdir TEST@strasbourg
$ mkdir FONCTION1
```

Manually retrieve **the files sf1a.c**, **sf1b.c**, **sf1.c** from https://wiki.morere.eu/doku.php?id=brasil:courses:git_gitlab:2_git_gitlab_course#useful_files and store in the **FONCTION1** directory
Retrieve the profile.sh script from https://wiki.morere.eu/doku.php?id=brasil:courses:git_gitlab:2_git_gitlab_course#useful_files and store it in the **~/TP_GIT** directory. This script allows you to switch between profiles on the same machine by typing the dev1, **dev2,** or **dev** commands.

```
$ mv ~/DOSSIER_TELECHARGEMENT/profile.sh ~/TP_GIT
```

Run the profile.sh script

```
$ cd ~/TP_GIT
$ source profile.sh
```

Check with the **alias** command that the script worked

Setting up your GIT:

```
$ git config --global user.email "votre_email"
$ git config --global user.name "votre_pseudo"
$ more ~/.gitconfig
```

Type **CTRL-C** to exit the more command

Increase the ID caching time to 7200 (2 hours) seconds, which is set by default to 15min:

```
git config --global credential.helper "cache --timeout=7200"
```

Create the local repositories for each DEV1 and DEV2 environment:

```
$dev1
```

We are in **DEV1@paris**.

```
DEV1@paris$ git clone git@gitlab.com:USER/project.git
$DEV1@paris cd project
```

We check that the alias **origin** points to the project to simplify the writing.

```
git remote add origin git@gitlab.com:USER/project.git
```

A message indicates that the alias already exists.

```
$ dev2
```

We are in **DEV2@nancy**.

```
DEV2@nancy$ git clone git@gitlab.com:USER/project.git
```

We also create repositories for the DEV and TEST profiles, we type the same commands.

PART C

AS A DEV1 DEVELOPER, I PREPARE SUB-FUNCTIONALITY 1A, PUSH IT TO MY BRANCH LOCALLY AND THEN TO THE REMOTE REPOSITORY FOR DEV TO USE:

```
$dev1
```

In **DEV1@paris** :

```
$ cd project/
$ ls
$ git branch
$ git branch -a
$ git checkout dev1
$ cp ~/TP_GIT/FONCTION1/sf1a.c . # Pay attention to the point
$ gcc -o sf1a.o -c sf1a.c #on compiles the C source to an object file
$ gcc -o sf1a sf1a.o #on performs the linkage to generate the executable program
$ ./sf1a # we run the program from the current directory
$ git add sf1a*
$ git commit -m "adding sf1a"
$ git push -u
```

PART D

AS A DEV2 DEVELOPER, I PREPARE SUB-FUNCTIONALITY 1B, PUSH IT TO MY BRANCH LOCALLY AND THEN TO THE REMOTE REPOSITORY FOR DEV TO USE:

```
$ dev2
```

In **DEV2@nancy** :

```
$ cd project/
$ cp ~/TP_GIT/FONCTION1/sf1b.c . # Pay attention to the point
$ git branch
$ git checkout dev2
$ gcc -o sf1b.o -c sf1b.c #on compiles the C source into an object file
$ gcc -o sf1b sf1b.o #on performs the linkage to generate the executable program
$ ./sf1b # we run the program from the current directory
$ git add sf1b*
$ git commit -m "adding sf1b"
$ git push -u
```

PART E

AS A DEV DEVELOPER, I MERGE THE WORK OF DEV1 AND DEV2 AND PREPARE A NEW BUILD SCRIPT ADAPTED TO THE REAL PROCESSOR ARCHITECTURE:

```
$ dev
```

In **DEV@metz** :

```
$ cd project/
$ git branch
$ git branch -a
$ git checkout dev
$ git pull
$ ls
$ git merge origin/dev1
$ ls
$ git merge origin/dev2
$ ls
```

DEV builds a new **sf1.c** file by concatenating the 2 files **sf1a.c** and **sf1b.c** with an editor and then pushes it into its branch locally and into the remote repository so that TEST can use it (or I download the file already concatenated on arch).

NOTE: this sf1.c file has already been created to simplify the exercise, retrieve it and place it in the DEV project directory

```
$ gcc -o sf1-build.o -c sf1.c -march=native
$ gcc -o sf1-build sf1-build.o
$./sf1-build
$ git add sf1-build
$ git push -u
$ git commit -m "adding sf1-build"
$ git push -u
$ git pull
```

PART F

AS A TEST TESTER, I RUN THE **SF1-BUILD** PROGRAM PREPARED BY DEV TO VERIFY IT BEFORE GOING INTO PRODUCTION:

```
$ test
```

In **TEST@strasbourg** :

```
$ cd project/
$ git branch -a
$ git checkout test
$ ls
$ git pull
$ git branch -a
$ git merge origin/dev
$ ls
$./sf1-build
```

PART G

AS A DEV1 TESTER, I CHECK MY BRANCH'S COMMIT HISTORY USING THE FOLLOWING COMMANDS:

- **git status** (check which commit HEAD is pointing to)
- **git log --graph --oneline --decorate** (check commit history)
- **git show n°COMMIT** (know the details about a commit)
- **git checkout n°COMMIT** (move HEAD to a particular commit (detached HEAD))
- **git checkout dev1** (move HEAD to dev1)

```
$dev1
```

2 SOLUTIONS:

```
$ alias gl="git log --graph --oneline --decorate" #CREATING AN ALIAS FOR GIT LOG
```

OR

```
$ gitk --all  git log #EQUIVALENT with GUI
```

In **DEV1@paris** :

```
$ cd project
$ git status //check where HEAD points
```

```
$ gl or gitk --all GIT LOG ANALYSIS
```

```
$ touch module1.h CREATE A NEW FILE
$ git add module1.h
$ gl or gitk --all NOTE IF THERE IS A DIFFERENCE FROM THE PREVIOUS GIT LOG
$ git commit -m "adding module1"
$ gl or gitk -all NOTE IF THERE IS A DIFFERENCE FROM THE PREVIOUS GIT LOG
```

```
$ git push -u
$ gl or gitk --all NOTE IF THERE IS A DIFFERENCE FROM THE PREVIOUS GIT LOG
```

```
$ git show dc591f7 => NOTE HERE YOUR COMMIT NUMBER  SEE THE DETAILS OF A COMMIT
$ git checkout dev1 REPLACE HEAD TO ORIGINAL COMMIT
```

Create another module2.h and check that the contents of the directory change with ls -l

As an example, here is an interpretation of the result of the first git log :

1. cc140e8 (HEAD -> dev1, origin/dev1) added sf1a :
    - This is the last commit you made on the dev1 branch.
    - "HEAD -> dev1" indicates that your HEAD, i.e. your current pointer, is on the dev1 branch.
    - "origin/dev1" means that this commit is also the last commit on the dev1 branch of the remote repository (origin).
2. dc591f7 (origin/main, origin/test, origin/HEAD, origin/dev2, origin/dev, main, dev) added changelog :
    - This commit is the direct parent of the cc140e8 commit.
    - It is shared by multiple branches both in the remote repository (origin/main, origin/test, origin/dev2, origin/dev) and locally (main, dev).
    - This suggests that this commit is a recent commonality between all these branches.
3. 869a814 Initial commit :
    - This is the very first commit of the repository.
    - All branches start from this commit.

```
$ git tag -l //LIST TAGS
$ git tag -a v1.0 -m "my version 1.0" CREATE AN ANNOTATED TAG
$ gl //CHECK FOR TAG ADDITION
$ git checkout v1.0 VERIFY HEAD MOVE
```

When I perform the "git checkout DEV1" command, the system shows this message:

```
Branch 'dev1' set up to track remote branch 'dev1' from 'origin'Switched to a new branch 'dev1'
```

What does this mean?

When you run the git checkout dev1 or git switch dev1 command, two things happen:

1. . Remote branch tracking: The message "Branch 'dev1' set up to track remote branch 'dev1' from 'origin'" means that Git has configured your local branch dev1 to track remote branch dev1 that is on the remote origin repository. This lets Git know that this local branch corresponds to a branch on the server. Tracking makes it easier to do things like git pull or git push, because Git knows where to pull updates from or where to push changes.
2. Branch change: The message "Switched to a new branch 'dev1'" indicates that Git has changed your current working branch to dev1. This means that any commits you make from now on will be added to that branch until you switch branches with another git checkout command.

So, you changed branches to work on **dev1**, and that branch is configured to follow its counterpart on the remote origin repository.

Analysis of the HEAD in the case of a merge :

In **DEV@metz**

```
$ cd project/
$ gl
* fb86245 (HEAD -> dev, origin/DEVdev Merge branch 'dev' of git@gitlab.com:USER/projet into dev
|\
| * f376f57 Delete sf1b.o
| * 6583fe9 Delete sf1b.c
| * 0b4b52b Delete sf1b
| * 9d1db25 Delete sf1a.o
| * 199a54c Delete sf1a.c
| * 9c550fc Delete sf1a
* | 198CD28 Adding SF1-Build
|/
* 87f58ee Merge remote-tracking branch 'origin/dev2' into dev
|\
| * 25bf076 (origin/DEV2) added sf1b
* | CC140E8 (origin/DEV1) added SF1A
|/
* dc591f7 (origin/main, origin/test, origin/HEAD, main) added changelog
* 869a814 Initial commit
```

HELPFUL RESOURCES:

Internship: https://www.freecodecamp.org/news/undo-git-add-how-to-remove-added-files-in-git/#:~:text=There%20are%20two%20major%20commands,git%20rm%20%2D%2Dcached%20commands.
Log: https://www.ionos.fr/digitalguide/sites-internet/developpement-web/git-log/
Push: https://gdevops.gitlab.io/tuto_git/commands/push/push.html

Exercise 2 Gitlab

PART A

1. Create an empty git repository outside of the repository where you have worked so far, for example in a new subfolder of your home.
2. Configure the repository to ignore files with the ".o" extension.
3. Create a file and tell **git** that you want to version it.
4. Move this file to another location and notify **git**.
5. The output of **git status** should only talk about two files at the end, the one you just moved and the one that was created in question 2.

View the final status of the deposit. By ignoring the "On the main branch" part for the moment , do you understand the other messages displayed?

PART B

1. Add a few files of the content you want, and then save a **commit** containing those files.

2. Edit at least two of the files you've backed up, and then create a commit that includes only the changes to a single file.
3. View changes that haven't been saved yet.
4. Use a git shortcut to create a commit containing the changes to the other files without needing to name them in the commands you use.
5. View the difference between the first two commits you made within the repository.
6. Verify that the working directory is clean.
7. Switch to the grandparent of the current commit, and attach a tag to it, then go back to your original commit.

PART C

1. Create a copy of the repository used in Part A.
2. Visit the parent of its active commit, and create a branch that goes from it.
3. Switch to this branch, and create a commit adding a new file.
4. Visualize the main branch and the one you just created with a graphical tool, for example code and the git-graph extension that you will install as follows: In vscode, use the shortcut "CTRL-P" for the quick launch and type the ext install mhutchie.git-graph command . Once the extension is installed, you will have a graphical overview of the evolution of the branches. (cf. https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph)
5. Merge the changes of hands within the current branch.
6. Replace the contents of a file with two different texts within the main and the new branch, by making these changes.
7. Merge the changes from the new branch within main, and resolve the ensuing merge conflict.

PART D

1. Upload your deposit to Gitlab.