1. Tuto Git et Gitlab

Sommaire

Introduction

Git est un **logiciel de gestion de versions décentralisée** (c'est même le plus utilisé au monde).

La **gestion de versions** consiste à maintenir l'ensemble des versions d'un ou plusieurs fichiers (du code par exemple). Cette activité étant fastidieuse et relativement complexe, un appui logiciel est presque indispensable (d'où l'utilité de Git).

La **gestion de versions décentralisée** consiste à voir l'outil de gestion de versions comme un outil permettant à chacun de travailler à son rythme, de façon désynchronisée des autres, puis d'offrir un moyen à ces développeurs de s'échanger leur travaux respectifs.

GitLab est un **service web d'hébergement** et de gestion de développement de logiciels (aussi appelé **forge**), basé sur Git. Ce type genre de service propose également un contrôle d'accès et des fonctionnalités destinées à la collaboration comme le suivi des bugs, les demandes de fonctionnalités, la gestion de tâches et un wiki pour chaque projet. Il en existe de nombreux, Github étant le plus utilisé. GitLab est simplement le logiciel de forge utilisé à Météo-France.

Souvent utilisé ensemble, il est assez facile de confondre Git et GitLab. Voici donc un petit tableau résumant les différences de chacun de ces outils :

Caractéristique	Git	GitLab
Quoi ?	Un logiciel	Un service
Où ?	Installé localement	Hébergé sur le Web (cloud)
Pour quoi ?	La gestion de versions	L'hébergement et le partage
Comment ?	En ligne de commande	Par un page web

Nous pouvons passer à la pratique ...

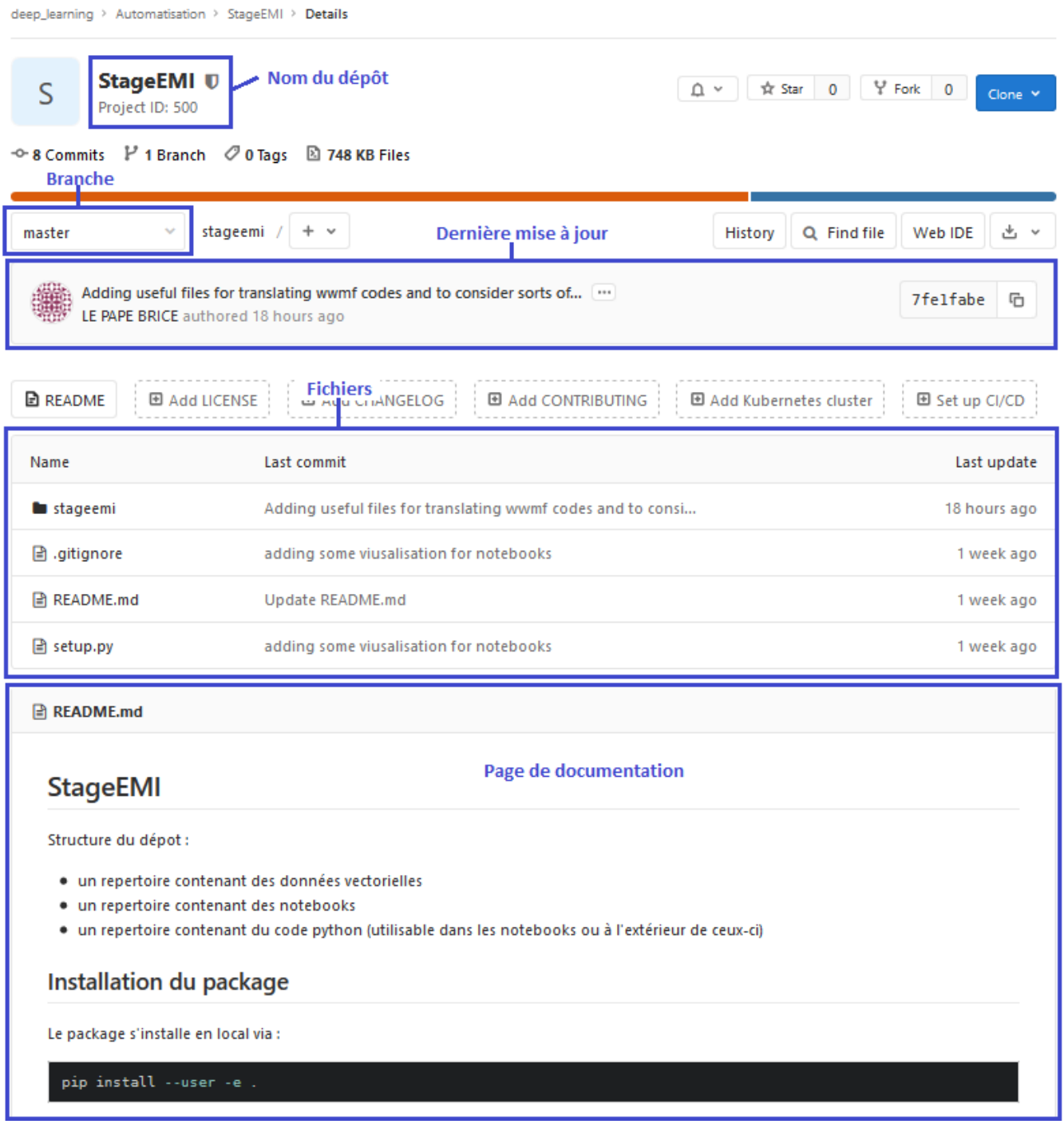
Étape 1 : Récupérer un dépôt

On appelle un dépôt (ou repository ou juste repo) un dossier dans lequel sont entreposés les fichiers d'un projet informatique (principalement des fichiers contenant du code, mais aussi des fichiers de documentation, de configuration, etc.). Il existe de façon de "créer" un dépôt : soit en le créant localement avec Git, soit en en récupérant un déjà existant depuis GitLab. Comme nous avons déjà créé un dépôt pour le stage, nous allons voir la deuxième option.

A. Se connecter à GitLab

Le dépôt est accessible à l'adresse suivante : http://gitlab.meteo.fr/deep_learning/automatisation/stageemi

Après avoir entré vos identifiants, vous arrivez sur la page de votre dépôt :



Il y a déjà différents fichiers et dossiers (que vous pouvez d'ailleurs ouvrir pour voir). Notre but est de copier (ou cloner) ce dépôt sur notre espace de travail local afin de pouvoir utiliser et modifier ces fichiers ou en ajouter des nouveaux.

B. Créer un token

Cette étape n'est pas obligatoire, mais elle simplifie grandement le reste du processus.

Un token est une chaine de caractère confidentielle associée à votre compte GitLab. Lorsque nous clonerons le dépôt de travail localement, nous passerons le token afin de nous identifier auprès de GitLab (une seule fois).

Pour créer un token, il vous faut aller dans *Settings* (dans l'icône déroulante en haut à droite) > *Access Tokens*.

Là il vous suffit de donner un nom à votre token (par exemple "emi_token"), de cocher la case *api* et de cliquer sur *Create personal access token*. Comme dans cet exemple.

Votre token apparait. **Copiez le et gardez le précieusement, il ne vous sera plus donné ultérieurement.** Dans mon exemple, GitLab m'a attribué le token suivant : DXXUuy-5ad4hdqJyp9fc

Vous pouvez ensuite retourner à la page du dépôt.

C. Cloner le dépôt

Sur la page du dépôt, cliquez sur le bouton ***Clone*** (en haut à droite) et copiez le lien *Clone with HTTP.* Dans l'exemple le lien est http://gitlab.meteo.fr/deep_learning/automatisation/stageemi.git

A ce lien, il faut rajouter entre le http:// et le gitlab.meteo.fr votre token de la façon suivante : gitlab.meteo.fr:*votre_token@* . Dans l'exemple cela donne : http://gitlab.meteo.fr:DXXUuy-5ad4hdqJyp9fc@gitlab.meteo.fr/deep_learning/automatisation/stageemi.git

Ouvrez un terminal localement (ou sur la machine sur laquelle je souhaite travailler) et placez vous dans votre dossier de travail.

Il vous suffit alors de taper la commande ***clone***

Exemple de la commande git clone

```
       $ cd code/
$ git clone http://gitlab.meteo.fr:DXXUuy-
5ad4hdqJyp9fc@gitlab.meteo.fr/deep_learning
/automatisation/stageemi.git
Cloning into 'stageemi'...
remote: Enumerating objects: 70, done.
remote: Counting objects: 100% (70/70), done.
remote: Compressing objects: 100% (66/66), done.
remote: Total 70 (delta 22), reused 0 (delta 0)
Unpacking objects: 100% (70/70), done.
$ cd stageemi/
$ ls
README.md  setup.py  stageem
```

Étape 2 : Utiliser le dépôt (ajout, modification et suppression de fichiers)

Vous disposez maintenant d'une copie du dépôt que vous pouvez modifier comme bon vous semble. Une modification de cette copie locale ne modifiera pas la version en ligne sur GitLab (à moins que vous ne l'ordonniez explicitement (voir Etape 3 : Envoyer des changements)).

D'ailleurs si vous modifier, ajouter ou supprimer un fichier, vous ne modifierez pas directement vôtre dépôt local. En effet, une modification du dépôt local est effective après un double processus d'ajout (**add**) et de validation (**commit**). Pour comprendre, intéressons nous rapidement au fonctionnement de Git.

A. Les arbres de Git

Schéma représentant les liens entre les différents arbres de Git

blocked URL

Votre dépôt local est composé de 3 "arbres" gérés par Git.

Le premier est votre **espace de travail** (ou working directory), qui contient réellement vos fichiers. C'est ce que vous voyez dans votre explorateur de fichier ou si vous faites la commande *ls.*

Le deuxième est un **Index** qui joue un rôle d'espace de transit pour vos fichiers.

Enfin, **HEAD** qui pointe vers la dernière validation que vous ayez faite.

Lorsque vous effectuer un changement sur votre dépôt local, vous affectez donc votre espace de travail. Pour acter ce changement, il faut donc l'ajouter à l'Index et la valider au HEAD.

B. Modifier l'espace de travail

Avant d'ajouter puis valider une modification, il faut d'abord apporter un changement à son espace de travail.

Dans l'exemple je vais donc créer un dossier *test/* et ajouter un fichier *my_file.txt* contenant le texte "Hello World".

Je peux constater l'état de mon dépôt et donc s'il y a eu des changements avec la commande ***status***

```
       git status
```

(le git status précise bien qu'il y a eu un changement avec l'ajout de fichier dans le dossier test/ mais qu'ils ne sont toujours pas "tracké" par l'index.)

Il faut ensuite ajouter et valider cette modification.

Exemple de modification de l'espace de travail

```
        $ cd stageemi
$ mkdir test
$ echo "Hello World" >> test/my_file.txt
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what
will be committed)
#
#       test/
$
```

C. Ajouter et valider un changement

Vous pouvez maintenant proposer un changement (l'ajouter à l'index) en exécutant la commande ***add***

```
        git add nom_du_fichier
```

C'est la première étape dans un workflow git basique. Pour valider ces changements, utilisez la commande ***commit***

```
        git commit -m "Message de validation"
```

Le message de validation est certes obligatoire, mais il est surtout important car c'est dans celui-ci que vous résumez succinctement les changements effectués.

Le fichier est donc ajouté au HEAD, mais pas encore dans votre dépôt originel distant (celui sur GitLab).

Exemple d'ajout et de validation

```
        $ git add test/my_file.txt
$ git commit -m "Adding new file my_file.txt"
[master 5dd7348] Adding new file my_file.txt
 1 file changed, 1 insertion(+)
 create mode 100644 stageemi/test/my_file.txt
$
```

Étape 3 : Envoyer des changements

Comme expliqué au début, les changements effectués sur votre dépôt local ne sont pas automatiquement répercutés au dépôt distant (ou remote) sur GitLab.

Vos changements sont pour l'instant dans le HEAD de la copie de votre dépôt local. Pour les envoyer à votre dépôt distant exécutez la commande

```
        git push origin master
```

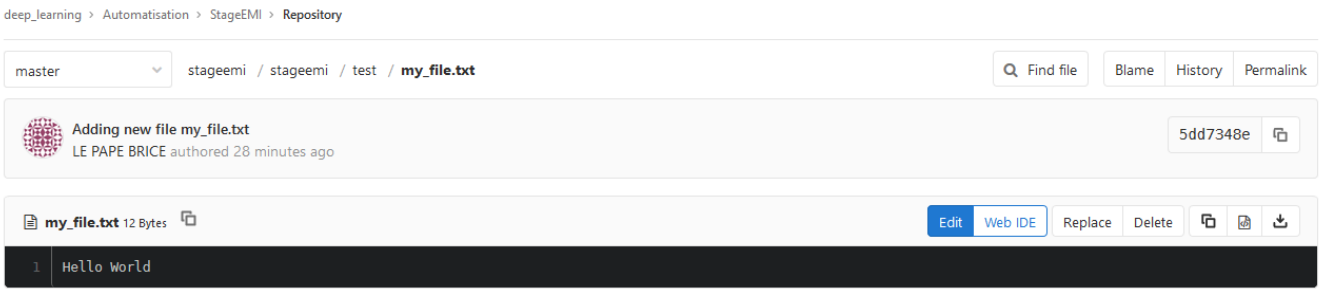
Dans cette commande, *origin* désigne le dépôt originel, donc celui sur GitLab (c'est le nom donné par défaut).

master désigne la branche dans laquelle vous souhaitez envoyer vos changements. La branche *master* est systématiquement la branche principale (branche maitresse) d'un dépôt. Nous allons nous intéresser aux branches par défaut dans la prochaine partie.

Maintenant, nous pouvons constater les changements effectués sur GitLab : le fichier *my_file.txt* dans son dossier *test/* est bien présent.

Exemple de commande push

```
        $ git push origin master
Counting objects: 7, done.
Delta compression using up to 36 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 391 bytes | 0 bytes
/s, done.
Total 5 (delta 2), reused 0 (delta 0)
To http://gitlab.meteo.fr:DXXUuy-
5ad4hdqJyp9fc@gitlab.meteo.fr/deep_learning
/automatisation/stageemi.git
   7fe1fab..5dd7348  master -> master
$
```



Étape 4 : Travailler avec des branches

Les branches sont utilisées pour développer des fonctionnalités isolées des autres. La branche master est la branche par défaut quand vous créer un dépôt.

Il faut utiliser d'autres branches pour le développement et fusionnez ensuite à la branche principale quand vous avez fini.

A. Créer une branche

Pour créer une branche, il suffit d'utiliser la commande ***checkout -b***

```
git checkout -b nom_de_ma_branche
```

Vous pouvez alors ajouter et valider des changements ainsi que les envoyer au dépôt *origin* comme expliqué dans les parties précédentes. Ces changements n'affecteront alors que la branche sur laquelle vous êtes.

Exemple de commande checkout -b

```
$ git checkout -b agregation
Switched to a new branch 'agregation'
```

Exemple de modification sur une branche

```
$ echo "1 2 1 2 ... this is a test" >> test
/my_file.txt
$ git add test/my_file.txt
$ git commit -m "Adding a new line to my_file.txt"
[agregation 539ad9a] Adding lines to my_file.txt
 1 file changed, 2 insertions(+), 1 deletion(-)
$ git push origin agregation
Counting objects: 9, done.
Delta compression using up to 36 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 440 bytes | 0 bytes
/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote:
remote: To create a merge request for agregation,
visit:
remote:   http://gitlab.meteo.fr/deep_learning
/automatisation/stageemi/merge_requests/new?
merge_request%5Bsource_branch%5D=agregation
remote:
To http://gitlab.meteo.fr:DXXUuy-
5ad4hdqJyp9fc@gitlab.meteo.fr/deep_learning
/automatisation/stageemi.git
 * [new branch]      agregation -> agregation
```

B. Changer de branche

Vous pouvez avoir besoin de changer de branche, pour travailler sur une autre fonctionnalité par exemple.

Pour cela il vous suffit d'utiliser la commande ***checkout***

```
git checkout nom_autre_branche
```

Exemple de commande checkout

```
$ git checkout master
Switched to branch 'master
```

C. Lister les branches existantes

Vous pouvez lister les branches existantes et ainsi savoir sur laquelle vous vous situez avec la commande ***branch***

```
git branch
```

Exemple de commande branch

```
$ git branch
  agregation
* master
```

Étape 5 : Mettre à jour et fusionner

Pour l'instant, nous n'avons pas eu besoin de mettre à jour notre dépôt local puisque c'était lui qui dictait les changements au dépôt sur GitLab.

A. Mettre à jour

Si vous êtes plusieurs à travailler en même temps sur le même dépôt vous devrez forcément récupérer les changements que vos collègues auront envoyé sur GitLab.

Pour cela il suffit d'utiliser la commande ***pull***

```
git pull # depuis la branche master
git pull origin nom_de_ma_branche # depuis une autre branche
```

Exemple de commande pull

```
$ git pull
Already up-to-date
```

B. Fusionner deux branches

Maintenant que vous avez fini de travailler sur une fonctionnalité, et que vous estimez qu'il est utile que cette fonctionnalité intègre en tant que telle ce que vous développez, il faut fusionner la branche de votre fonctionnalité dans la branche principale.

Pour cela il faut utiliser la commande ***merge*** depuis la branche principale.

```
git merge nom_de_ma_branche -m "Message de fusion"
```

Exemple de commande merge

```
$ git merge agregation -m "Merging and adding the agregation feature"
Updating 5dd7348..539ad9a
Fast-forward
 stageemi/test/my_file.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

Normalement, si vous avez fait les choses correctement (à savoir mis à jour, ajouter, valider et envoyer régulièrement vos changements), git tente d'auto-fusionner les changements. Malheursement, ça n'est pas toujours possible et résulte par des conflits. Vous devez alors régler ces conflits à la main en éditant les fichiers indiqués par Git.

C. Supprimer une branche

Enfin vous pouvez supprimer localement la branche qui ne vous est plus utile avec la commande ***branch -D***

```
git branch -D nom_de_ma_branche
```

Et vous pouvez supprimer la branche sur le dépôt GitLab (si elle y est) avec la commande ***push origin --delete***

```
git push origin --delete nom_de_ma_branche
```

Exemple de suppression de branche

```
$ git branch -D agregation
Deleted branch agregation (was 5dd7348).
$ git push origin --delete agregation
To http://gitlab.meteo.fr:DXXUuy-
5ad4hdqJyp9fc@gitlab.meteo.fr/deep_learning
/automatisation/stageemi.git
 - [deleted]         agregation
```

Vous savez désormais le nécessaire pour exceller dans la gestion de version avec Git et GitLab.

Git Cheat Sheet

Git cheat sheet

```
Common console commands:
 cd - change directory
 mkdir - make directory
 ls - view the files/folders in directory


-----------------------------------------
 BEGIN WORKFLOW
-----------------------------------------
Clone the Repo to local machine:
 $ git clone https://gitlab.meteo.fr:<token>@gitlab.meteo.fr/<group_name>/<repo_name>.git

Make sure the local master is up-to-date:
 $ git pull origin master

Move to branch:
$ git checkout branch_name

Create new branch:
 $ git checkout -b branch_name

Navigate file structure as needed:
 $ ls
 $ cd folder_name

Add the files to the branch:
 $ git add .

Verify file:
 $ git status

Commit the files:
 $ git commit -m "comment"

Add branch and files to the Remote Repo:
 $ git push origin branch_name

Merge your branch to the master locally
and push the new master to the remote repo:

Switch back to master branch:
 $ git checkout master

Merge the branch with the local master:
```

```
 $ git merge branch_name -m "comment"

Push the local master to the remote master:
 $ git push origin master

Delete local branch:
 $ git branch -d branch_name
 OR
 $ git branch -D branch_name

Or you don't want to do it locally, you can go to the GitLab
website to manage pull request and merge.

Switch back to local master so you can delete the local branch:
 $ git checkout master

Delete local branch:
 $ git branch -d branch_name
 OR
 $ git branch -D branch_name

-----------------------------------------
Managing your Local Repo
-----------------------------------------
NOTE: If you need to hard reset your local repo to match
the remote master use the following commands:
 $ git fetch origin
 $ git reset --hard origin/master

Undo the act of committing, leaving everything else intact:
 $ git reset --soft HEAD^:

Undo the act of committing and everything you'd staged,
but leave the work tree (your files intact):
 $ git reset HEAD^

Completely undo it, throwing away all uncommitted changes,
resetting everything to the previous commit:
 $ git reset --hard HEAD^
```

Autre page permettant de connaître plein de choses sur git :

Git pour FLEXPART à Météo-France

Références

- *git - petit guide* - Roger Dudler
- *A simple Git workflow for Github beginners and everyone else* - Eric Kleppen