Introduction à GIT

Objectifs de la séance

Avoir un premier aperçu (par la pratique) des commandes principales de git.
Cette mise en pratique sera fait sur le dépot du miniprojet du groupe.
Le but pour les étudiants sera de manipuler le dépot et d'ajouter divers fichiers/informations :

- Les slides des cours précédents,
- Des informations sur le projet.

Pour les séances suivantes (et le miniprojet) git sera utilisé afin de travailler en groupe.

Pourquoi utiliser Git ?

- **Gestion des versions et de l'historique d'un projet :**
  - Git permet de suivre chaque modification d’un projet.
  - Il aide à revenir à une version précédente en cas de besoin (introduction de bug, ...).
- **Collaborer efficacement :**
  - Git permet à plusieurs personnes de travailler en parallèle sur le même projet sans écraser les modifications des autres.

Pour ces raisons il est devenu **un outil incontournable** dans presque tous les projets de développement moderne.

Qu'est-ce que Git ?

- **Git est un système de gestion de versions.**
  - Il enregistre toutes les modifications d’un projet, fichier par fichier.
  - Chaque version d’un projet est conservée dans une "base de données" Git.
  - Git permet de collaborer sur un projet à plusieurs de manière structurée.

Git dans la pratique

> Imaginez que vous travaillez sur un projet durant quelques semaines puis faites un break. Deux mois plus tard, vous reprenez le projet ... Malheureusement, le code que vous avez ne fonctionne plus...

Sans Git :

- Comment retrouver une ancienne version du projet ? Sauvegarde (v1, v2, vXXXX) ?
- Comment vous y prendre pour avoir de nouveau une version fonctionnelle ?

Avec Git :

- **Revenir à une version précédente en quelques commandes.**
- **Sauvegarder des étapes importantes de votre travail (commits).**
- **Annoter des versions spécifiques du codes (tag)**

Git vs GitHub/GitLab/Bitbucket

Git : Le logiciel

- **Outil de gestion de versions décentralisé.**
- Fonctionne **localement** sur votre ordinateur.
- Vous permet de suivre l'historique de vos modifications, de gérer des branches, et de collaborer à plusieurs.

GitHub / GitLab / Bitbucket : Les plateformes

- **Services en ligne** qui hébergent des dépôts Git à distance.
- Facilitent le partage de code et la collaboration sur des projets via une interface.
- Proposent des fonctionnalités supplémentaires (**Issues**, **Merge Requests / Pull Requests**, **Intégration continue (CI/CD)**)

Partie pratique

Première étape : Configuration globale

Cette configuration sera valable pour tous vos projets git.

Elle pourra être changé par la suite

1. Identifier qui fait les modifications

```
git config --global user.name "Votre Nom"
git config --global user.email "votre.email@example.com"
```

2. Passer le proxy Météo-France (permet de communiquer avec le remote)

```
git config --global --unset https.proxy
git config --global --unset http.proxy
```

Création/Clonage d'un projet

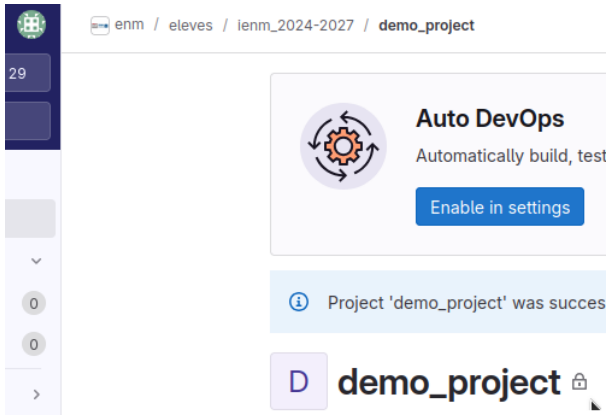
A. Connecter vous à git.meteo.fr (via vos identifiant LDAP)

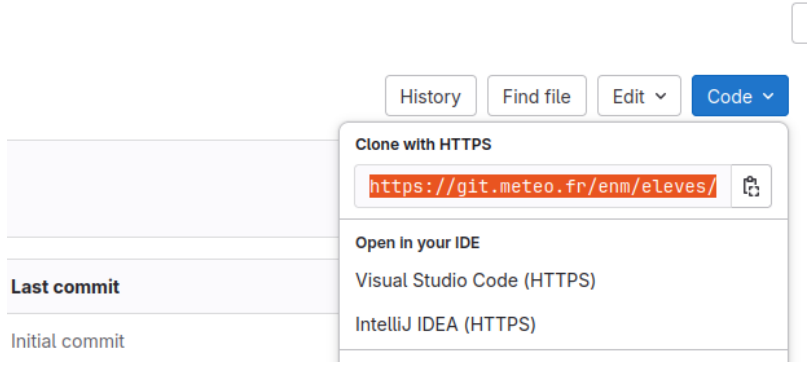
> login : chabotv
> password : NeJamaisDonnerSonPassword

Vous allez maintenant devoir vous mettre en groupe de 3 ou 4 étudiants (pour le tiers 1) ou de 4 étudiants (pour les tiers 2 et 3).

Les professeurs devront vous ajouter manuellement au bon projet git.

B. Aller sur le dépot du projet et cloner le dépot






```
git clone https://git.meteo.fr/enm/eleves/ienm_2024-2027/cours_cs/gpX-Y.git
```

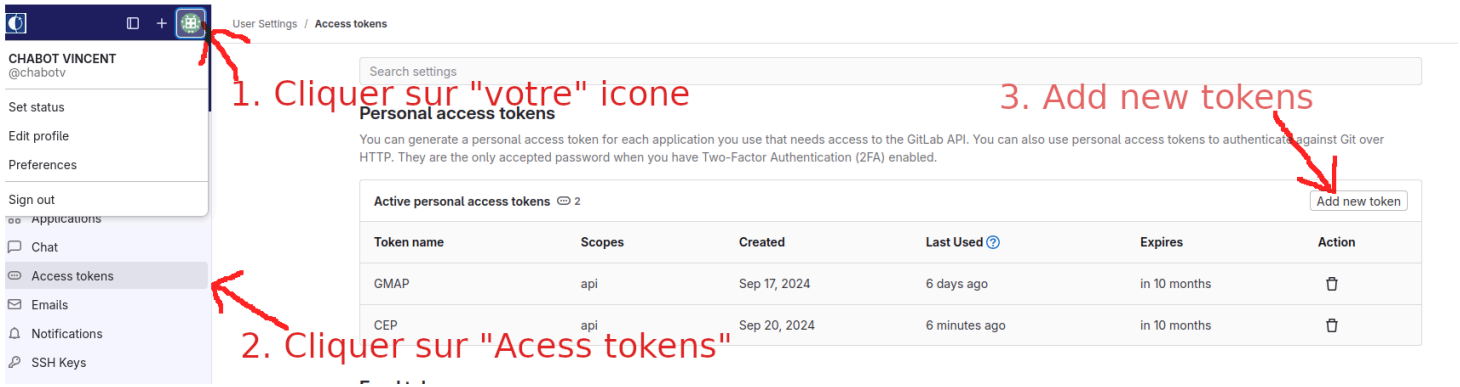
Ne pas avoir à rentrer son password à chaque communication

Il existe plusieurs moyens pour ne pas rentrer son password plusieurs fois :

- Garder en cache le password `git config credential.helper 'cache --timeout=3600'`

  Ici vous n'aurez besoin de rentrer votre password que toutes les heures.
- Générer un token et l'ajouter dans la configuration du dépot (slide suivante)

Génération de token



2.

Copier le token généré (il ne vous sera pas redonné).

Vous pouvez ensuite l'ajouter en modifiant le fichier `.git/config`

```
[remote "origin"]
        url = https://git.meteo.fr/MyPath/MyProject.git
```

doit etre changé en

```
[remote "origin"]
        url = https://git.meteo.fr:MyTOKEN@git.meteo.fr/MyPath/MyProject.git
```

B. Consulter l'historique du projet et première tâche

Commencer par regarder l'historique de votre projet.
Pour cela placer vous dans le répertoire de votre projet et faites :

```
git log
```

Maintenant vous allez tous modifier localement votre projet
Pour cela :

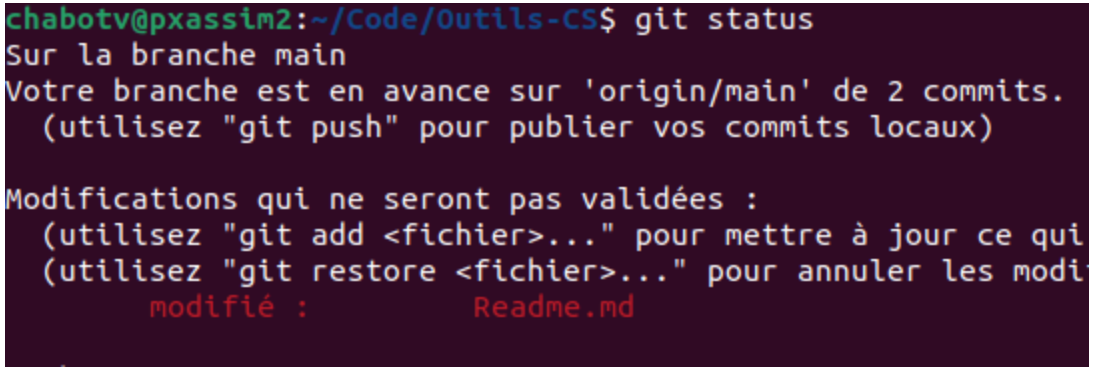
- Créer un dossiers `slides`
- Choisissez chacun (au sein du groupe) **un** pdf différent et mettez-le dans le dossier slides.

> Question : Est-ce que l'historique de votre projet a changé ?

C. Consulter l'état du dépot

Nous allons maintenant consulter l'état de "l'arbre de travail"

Pour cela faire `git status` .



Vous deviez avoir ce type de résultat

Ici cela me renseigne que le fichier Readme.md a été modifié. **Dans votre cas, vous devriez voir en rouge un dossier slides avec un pdf de slides.**

> Est-ce que le fichier que vous avez ajouté à votre dépot a été ajouté au projet sur `git.meteo.fr` ?
>
> Que comprenez-vous du message renvoyé par la commande ?

D. Ajouter les modificaitions

On va "ajouter" les modifications apporté au fichier et associer un message de "suivi".

```
git add slides/LeNomDeMonFichier.pdf
git commit -m "Mon message décrivant les changements"
```

> Votre fichier apparaît-il sur votre dépot sur `git.meteo.fr` ?
> Consulter l'historique de votre projet (via `git log`). De même consulter l'état du dépot local (`git status`). Que constatez-vous ?

E. Pousser les modifications vers la plateforme

Maintenant nous pouvons pousser les modifications sur la plateforme.

```
git push
```

Pour le premier qui **pousse** cela devrait bien se passer. Pour les autres vous devriez voir une erreur. Cette dernière est liée au fait que votre *branche* est **en retard** vis à vis de l'état du code sur la plateforme.

Pour vous remettre à jour faites `git pull`.

> A noter que vous aurez à faire plusieurs mise à jour (ou alors, il faut que vous fassiez les `push` et `pull` dans un certain ordre au sein du groupe).

Aller, au fur et à mesure de l'ajout des slides, vérifier sur la plateforme que tout s'est bien passé.

Travailler à plusieurs

A plusieurs, avant de pouvoir "pousser" ses modifications sur la plateforme il faut "tirer" l'état le plus récent de la `branche` sur laquelle on travaille.

Pour cela il faut faire

```
git pull
```

Attention : En cas de conflit (deux utilisateurs modifiant la même partie du code), il faudra les résoudre "à la main". Certains conflits peuvent cependant être géré automatiquement par git (à vos risques et perils).

La notion de branche dans Git

Qu'est-ce qu'une branche ?

- Une **branche** est une version parallèle de votre projet.
- Elle permet de travailler sur de nouvelles fonctionnalités ou corrections de bugs **sans affecter la branche principale** (nommée `main`). Ainsi, on isole les `bugs` du développement de la branche principale. Cette dernière doit rester fonctionnelle tout au long du projet.
- Chaque branche possède son propre historique de commits.

Pourquoi utiliser des branches ?

1. **Développement en parallèle**
    - Vous pouvez travailler sur plusieurs aspects du projet en même temps (nouvelle fonctionnalité, tests), sans impacter le reste du code.
2. **Sécurité des modifications**
    - Les changements dans une branche n'affectent pas le code principal.
    - Permet d'expérimenter ou de corriger des bugs sans casser la version stable.

Une fois le travail terminé, les branches peuvent être **fusionnées** (MergeRequest/PullRequest).

Votre première Branche

Cette fois-ci vous allez devoir chacun

- Créé une branche (slide suivante)
- Ajouter/Modifier un (répartissez-vous les tâches) des fichier suivant dans votre branche :
  - **Readme.md** : Changer le fichier Readme pour décrire ce que contient le dépot
  - **Commandes.md** : Vous pouvez ajouter queleques commandes utiles (par exemple comment activer votre environnement python, comment en créer un nouveau, comment créer un alias).
  - un fichier **.gitignore** (permet à git de ne pas suivre/montrer tous les fichiers) dans lequel vous ignorerez tous les dossiers `__pycache__`.
  - Le fichier du tp sur la programmation orientée objet (*guitar.py*).

Création d'une branche

Comment procéder

Vous pouvez chacun créer une branche dans le projet.

Pour cela faire

1. `git checkout -b MaBranche`
2. Ajouter le fichier et un message de commit.
3. Pousser vos modifications

```
git push --set-upstream origin MaBranche
```

Faite un `git status`. Que voyez-vous ?

Aller sur git.meteo.fr. Comment voir votre branche ?

Notion de Merge Request (MR)

Définition

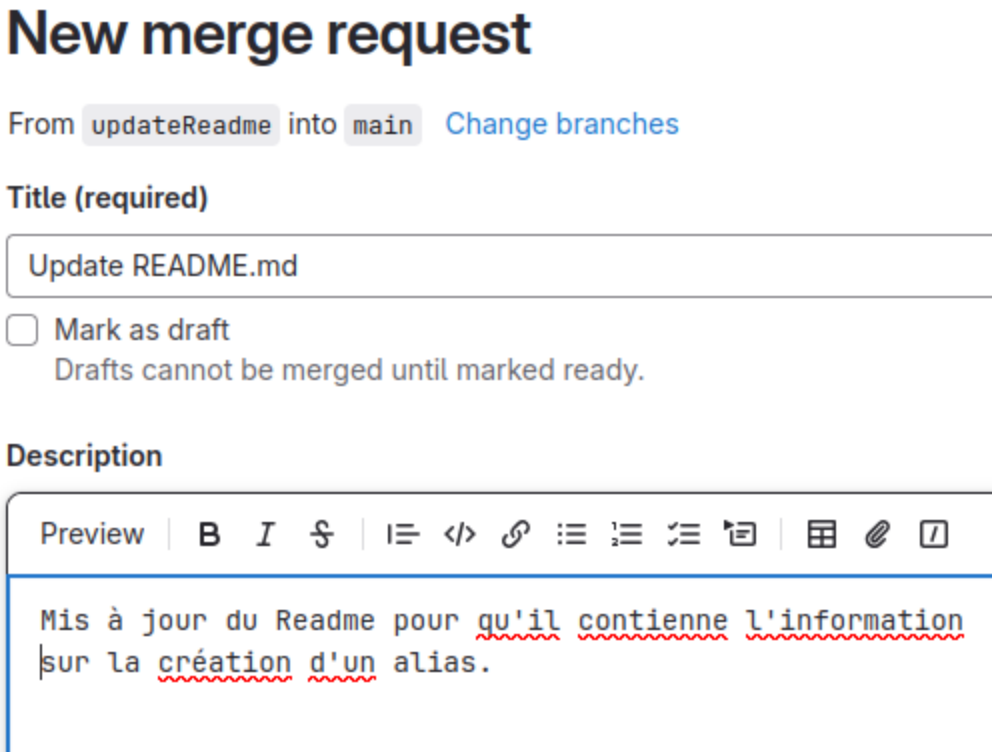
- Une **Merge Request** (ou **Pull Request** sur GitHub) est une demande d'intégration de code.
- Elle permet de proposer les modifications d'une branche vers une autre (souvent vers la branche `main` ou `dev`).

Pourquoi utiliser des MergeRequest ?

1. **Révision de code** :
    - Permet à d'autres de **relire et commenter** le code avant qu'il ne soit intégré.
    - Aide à repérer les erreurs, améliorer la qualité du code et partager des bonnes pratiques.
2. **Discussion et collaboration** :
    - Les MRs offrent un espace pour **discuter et ajuster les changements**.
    - Les équipes peuvent échanger sur des choix techniques, des suggestions, ou des améliorations.
3. **Validation des tests** :
    - Avant de fusionner, les MRs peuvent permettre de vérifier si le code passe des **tests** (si possible automatisés)

Votre première MergeRequest

Vous pouvez maintenant créer votre première Merge Request.



Vous pouvez ajouter un camarade responsable d'effectuer la fusion (`merge`).

Vous pouvez aussi ajouter des camarades responsables de relire et donner leur avis sur votre proposition d'amélioration (et poser des questions si quelque chose ne leur semble pas clair).

Ajout de commentaires de la part des relecteurs

Des commentaires peuvent être ajouter à une MergeRequest (de la part des relecteurs). Cela peut être fait pour (par exemple) :

- Demander des modifications
- Avoir des éclaircissement sur une partie du code
- ...

C'est un moment `d'interaction` autour de la proposition faite ou plusieurs personnes peuvent poser des questions ou indiquer des problèmes (fautes d'orthographes...).

> Faite chacun un commentaire sur une MergeRequest d'un autre.

Une fois tous les commentaires prise en compte, une personne (si possible pas celle qui a créer la MR) est responsable de la fusion de la branche.

Changer de branche

Une fois que toutes vos merges request on été effectuées, il vous faudra changer de branche (revenir dans la branche main) afin d'avoir l'ensemble des modifications faites par vous et vos camarades.

Pour cela faire :

```
git checkout main
```

> Est-ce que vous voyez les fichiers ajoutés ?

puis mettre à jour la branche

```
git pull
```

afin de tirer les modifications.

La notion de tag

Lorsque vous allez développer votre projet, vous allez toujours "vouloir" ajouter de nouvelles fonctionnalités, ...

Au bout d'un moment, le nombre de commit fait va être très important (plusieurs milliers) et il va être très difficile de savoir à quel moment vous aviez une version fonctionnelle (et avec quels fonctionnalités), quand vous avez supprimez (ou casser) une fonctionnalité.

Pour avoir un "historique" plus exploitable vous pouvez utiliser des tags. Cela constituera les versions du code vers lesquelles vous pourrez vous tourner pour avoir l'état du code comme il était `avec cette fonctionnalité`.

Vous pouvez ajouter des tags directement sur l'interface web.

> Créez un nouveau tag (par ex v_0.0.0).

Les issues

C'est l'interface d'échange pour rapporter des bugs, discuter/demander de nouvelles fonctionnalités aux développeurs d'un package.

Lorsque vous envisagez d'en créé une, commencer par regarder si une issue similaire n'existe pas déjà.

Comment bien décrire un bug

Informations à inclure :

- **Titre clair** : Soyez précis
- **Description détaillée** : Expliquez le problème rencontré de manière claire.
- **Étapes pour reproduire le bug** :
    i. Indiquez les étapes précises.
    ii. Fournissez le code, si applicable.
- **Résultat attendu** : Décrivez ce que vous pensiez que le programme ferait.
- **Résultat obtenu** : Ce qui s'est réellement passé, incluant tout message d'erreur.
- **Version et environnement** : Version du code , Version de python et des librairies, ...