Guide étudiant : Mise en place de l'environnement informatique pour la formation en Intelligence Artificielle - Datalab

Sommaire

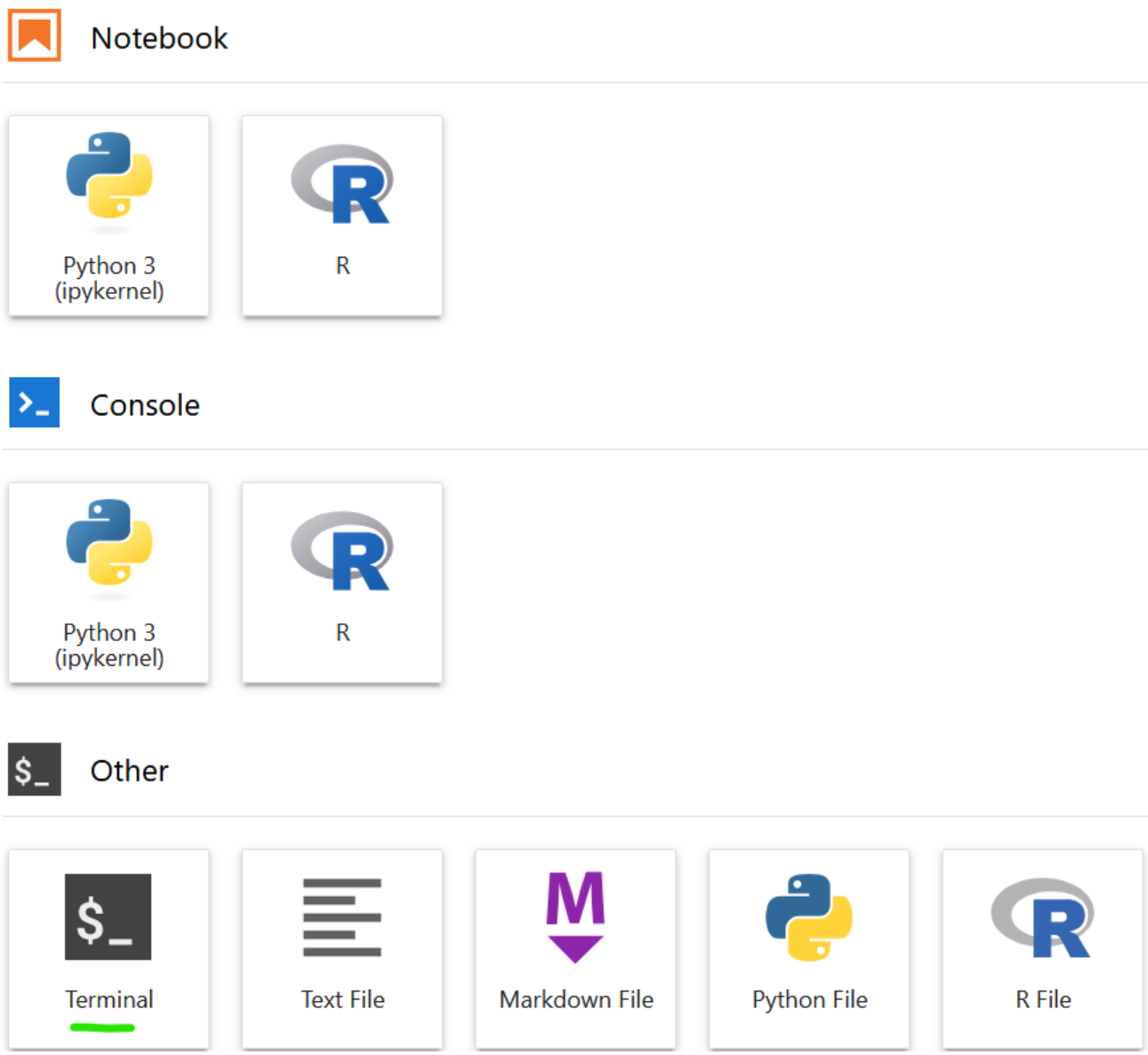
DataLab Partie 3 : GIT pour le travail collaboratif et RUDI pour les données

- 1. Débuter avec Git dans JupyterLab
  - GIT : Configuration de l'utilisateur
  - Clonage du TP depuis Gitlab vers JupyterLab
  - Sauvegarde de modifications du TP depuis JupyterLab vers Gitlab
- 2. RUDI pour les datasets de données (OpenData)
  - Présentation de RUDI
  - Récupération de données RUDI depuis le notebook
- 3. Bases de GIT en ligne de commande
  - Premier clonage du TP : git clone
  - Modification d'un fichier : git status
  - Sauvegarde dans le repo local : git add, git commit
  - Envoi de la modification sur le serveur: git push
  - Récupération d'une modification depuis le serveur: git pull
  - Gestion d'un conflit
- 4. Travail collaboratif via GIT
  - Création d'une branche de developpement : git checkout -b
  - Modification de fichier sur une branche
  - Changement de branche courante : git checkout, git stash
  - Merge d'une branche vers une autre : git merge

1. Débuter avec Git dans JupyterLab

GIT : Configuration de l'utilisateur

Commencer par ouvrir un terminal sous JupyterLab



JupyterLab ouvert

Puis il faut utiliser la commande "bash" pour avoir un terminal plus pratique

```
bash
```

> ***NOTE:*** les copié collé vers ce terminal doivent se faire uniquement avec les raccourcis clavier ctrl-c / ctrl-v

L'outil git permet de tracer les auteurs des changements.
Par défaut, lorsqu'un commit est effectué, le nom et l'adresse sont paramétrés par défaut : copié depuis les paramètres de la machine et peuvent très souvent se révéler inexacts.
Il est préférable de renseigner ces éléments en amont en utilisant la commande `git config`:

```
git config --global user.name "Your Name"
git config --global user.email "your.name@etudiant.univ-rennes.fr"
```

> ***NOTE:*** Le paramètre "--global" permet de modifier des paramètre pour votre compte utilisateur quelque soit le dépôt GIT local utilisé.

De plus, afin d'éviter d'avoir à entrer le token GIT (Personal Access Token) systématiquement, une autre commande GIT s'avère utile :

Sous Linux et MAC, on peut utiliser le "credential cache" qui stocke en mémoire vive de façon éphémère les tokens avec les deux commandes suivantes :

```
git config --global credential.credentialStore cache
git config --global credential.helper cache
```

On peut également définir la durée maximum de stockage éphémère (par défaut 900 secondes soit 15 minutes):

```
git config --global credential.cacheOptions "--timeout 5400"
git config --global credential.helper 'cache --timeout=5400'
```

Clonage du TP depuis Gitlab vers JupyterLab

Il est temps de cloner le dépôt GIT.

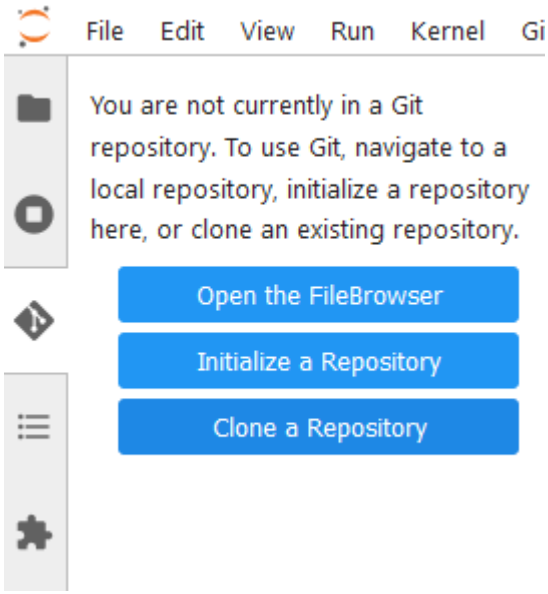
> ***NOTE:*** JupyterLab a déjà l'extension "jupyterlab-git" installée, cette extension permet de gérer les dépôts Git directement dans l'environnement.

Pour cloner le projet depuis Gitlab il faut tout d'abord cliquer sur l'icone Git dans la partie gauche de l'interface web de JupyterLab.

Icone Git dans JupyterLab

On accède alors au menu Git qui nous propose plusieurs possibilités.

Il faut choisir "Clone a Repository".

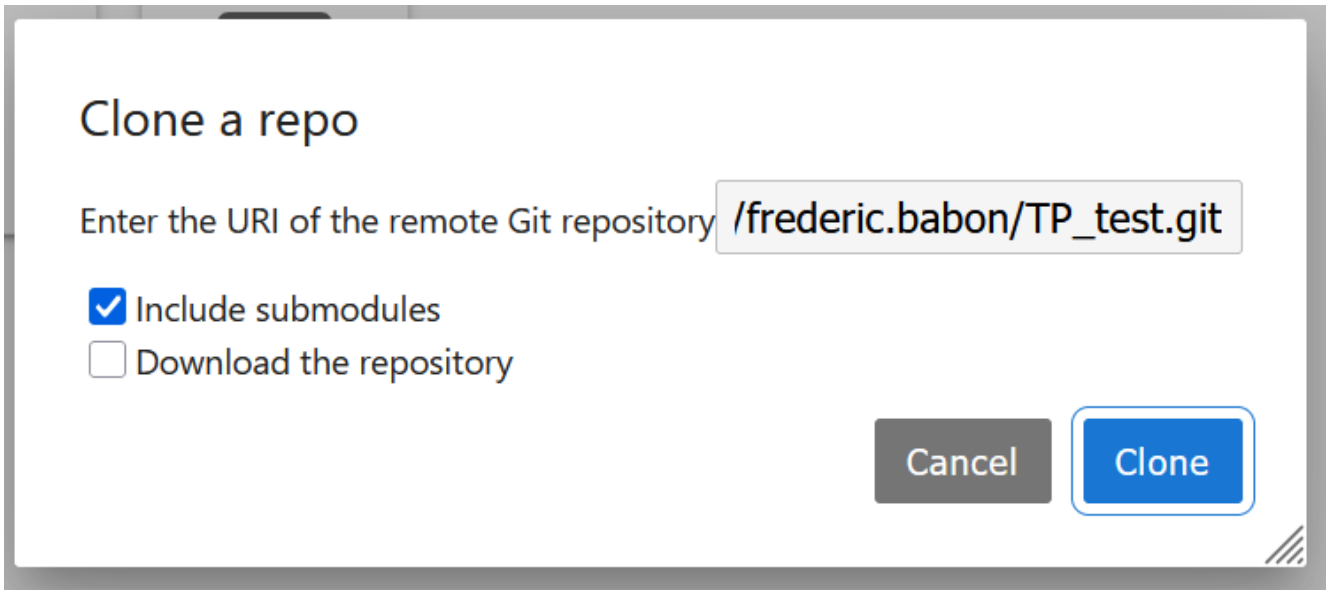


Menu Git dans JupyterLab

Un pop-up s'ouvre pour demander le lien https vers le dépôt Git. Il faut coller ici le lien récupéré via Gitlab, dans notre exemple "https://gitlab-ia.univ-rennes.fr/frederic.babon/TP_test.git"

> ***NOTE:*** Pour rappel ce lien peut être récupéré sur la page du projet dans GitLab. Voir détails en partie 1.

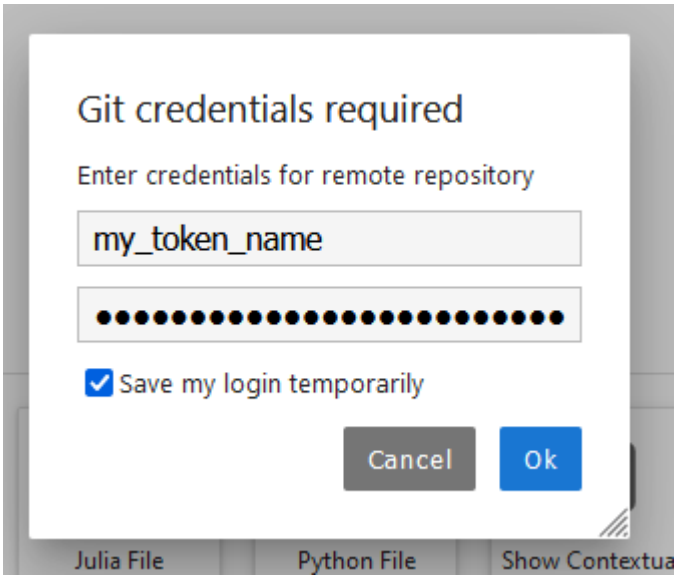
Puis cliquez sur le bouton bleu "Clone"



*Clonage via "jupyterlab-git" *

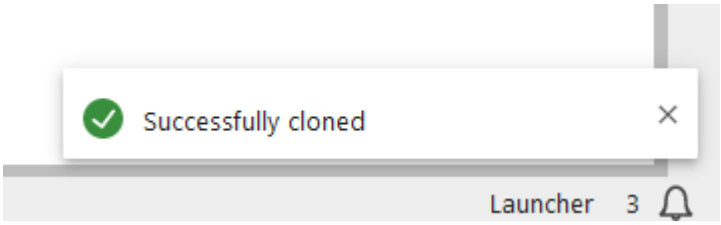
L'extension Git va alors demander le token et mot de passe nécessaires pour s'authentifier avec de réaliser l'opération de clonage. Il faut donc renseigner le nom du token Gitlab (dans notre exemple "my_token_name") et comme password le token lui-même.

> pour ne pas avoir à recopier à chaque opération Git le token, pensez à activer "Save my login temporarily"



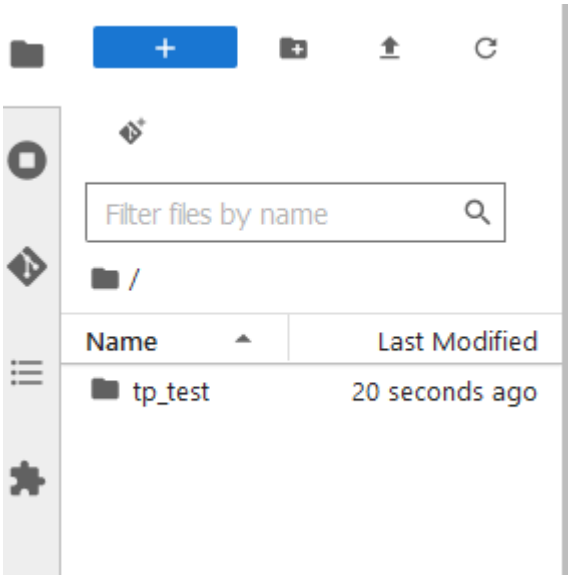
Token et mot de passe requis pour le clonage

Vous devriez alors voir apparaître en bas à droite un message qui confirme que le clonage s'est bien déroulé.



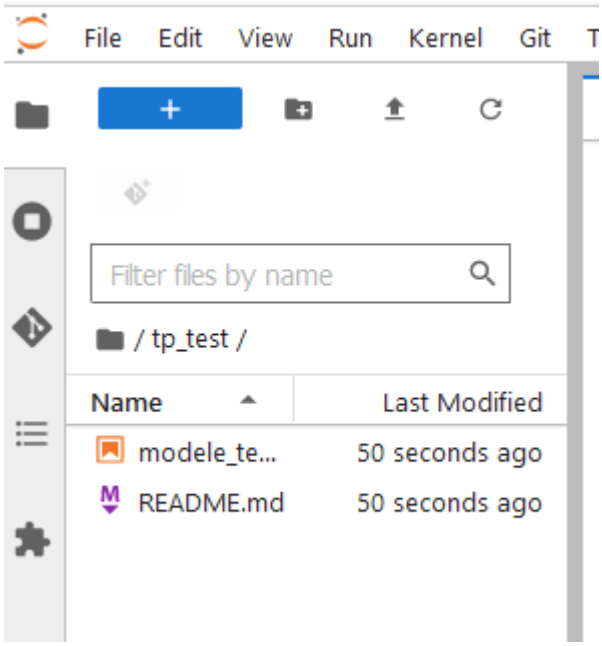
Clonage réussi : message

De même vous devriez voir, dans l'explorateur de fichier de JupyterLab, les fichiers récupérés par clonage.



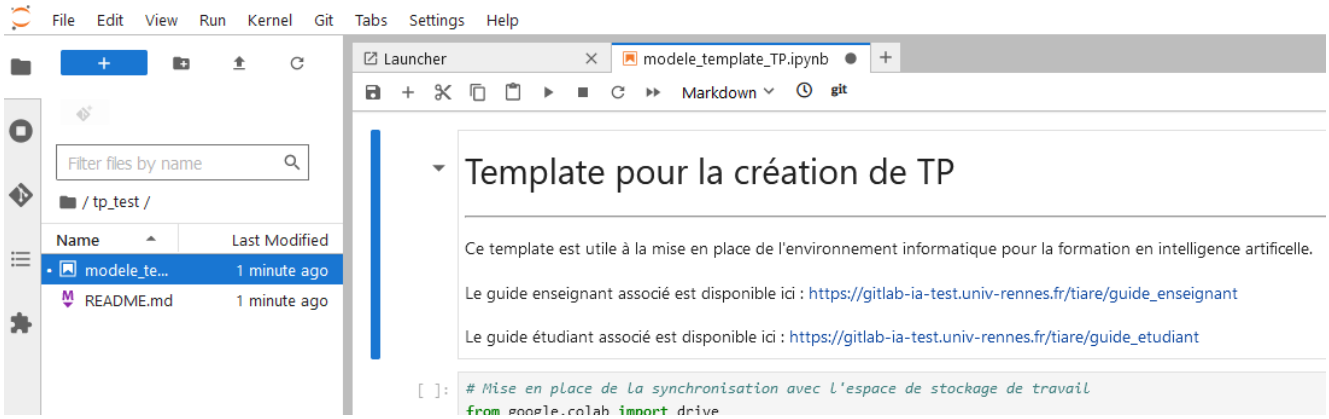
Clonage réussi : dossier

Double cliquez sur le dossier "tp_test" pour visualiser les fichiers.



Clonage réussi : fichiers

Et vous pouvez alors ouvrir le notebook "modele_template_TP.ipynb".



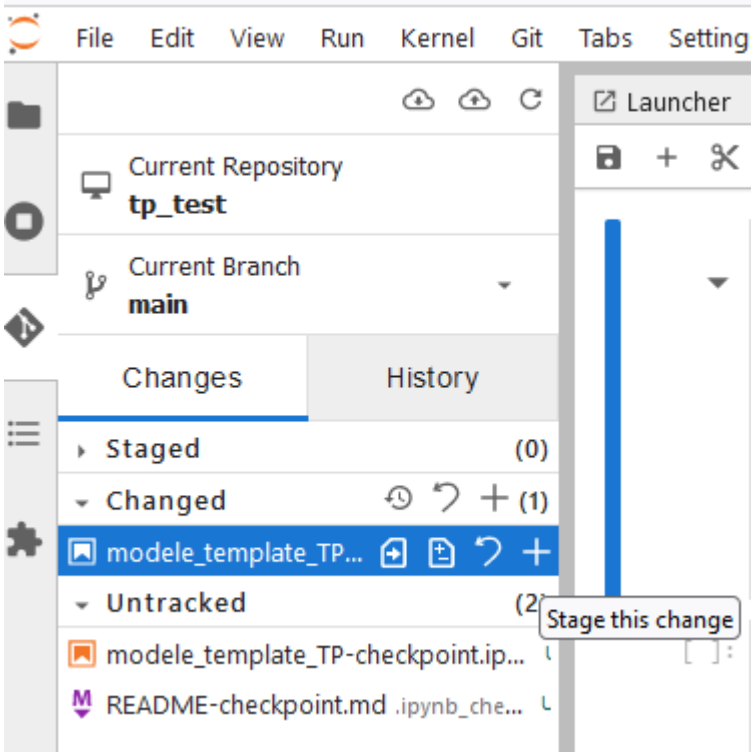
Ouverture du notebook

Sauvegarde de modifications du TP depuis JupyterLab vers Gitlab

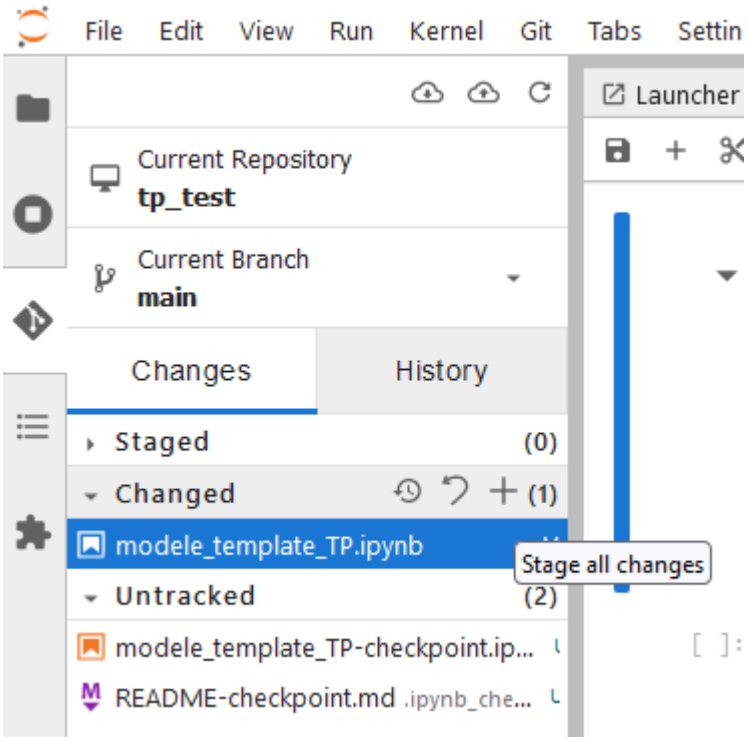
Lorsque vous aurez effectué des modifications sur un ou plusieurs fichiers, pensez à sauvegarder votre travail sur Git. Pour cela il faut ouvrir de nouveau le menu Git, et vous verrez alors la liste des fichiers modifiés "Changed".

Deux possibilités ici pour préparer le commit Git :

- ajouter un par un les fichiers modifiés
- ajouter tous les fichiers modifiés d'un coup

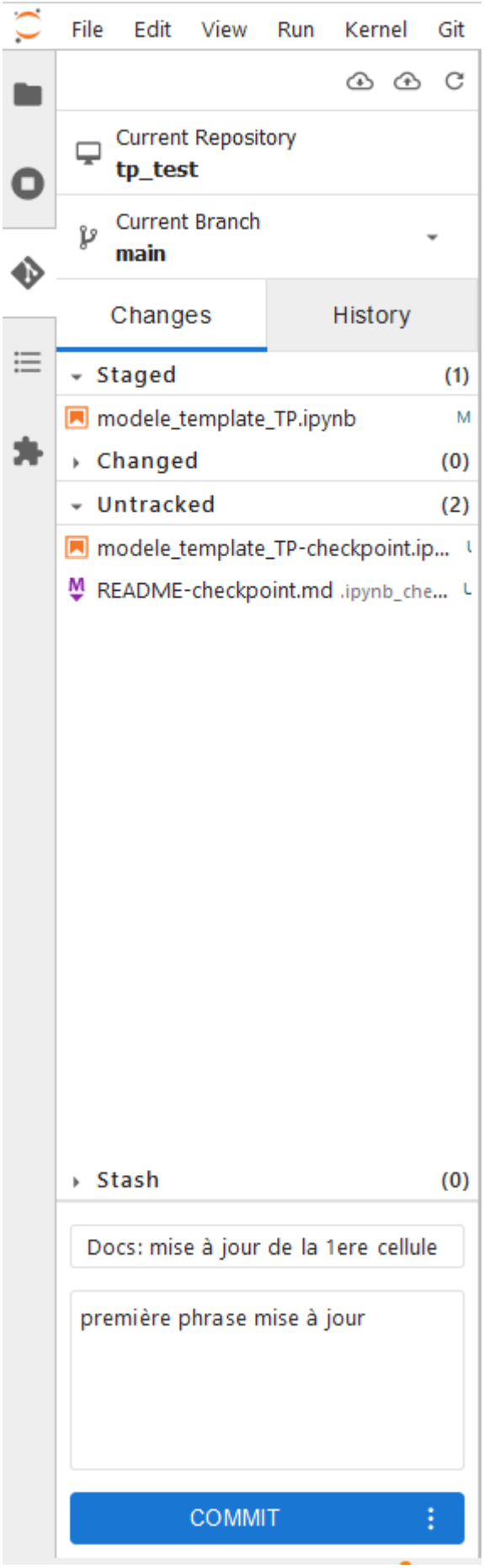


Ajout d'un fichier modifié "stage this change" pour le commit



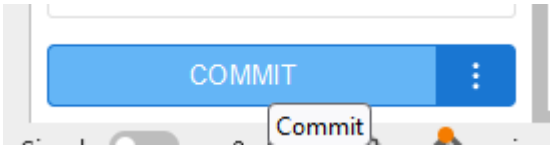
Ajout de tous les fichiers modifiés "stage all changes" pour le commit

Une fois les fichiers ajoutés, il faut définir le résumé "summary" et la description du commit Git.



Résumé "summary" et description du commit Git

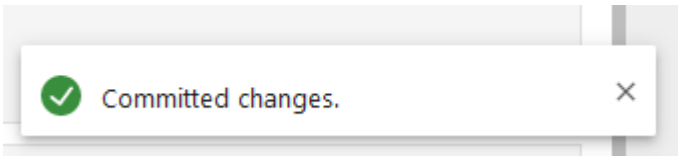
Cliquez ensuite sur le bouton bleu "COMMIT" en bas à gaucher pour sauvegarder (localement) le commit.



Bouton bleu "COMMIT"

> ***NOTE:*** pour rappel, lors d'un premier commit il peut être nécessaire de configurer nom du développeur et email. L'extension git peut afficher une invite graphique pour cette configuration. Dans ce cas, vous devrez saisir un nom d'utilisateur (Prenom Nom) et un email (prenom.nom@etudiant.univ-rennes.fr). Autrement, comme vu précédemment, il est possible d'ajouter manuellement une cellule au notebook courant et d'y mettre les commandes suivantes: '!git config user.name "Prenom Nom"' et '!git config user.email "prenom.nom@etudiant.univ-rennes.fr"'

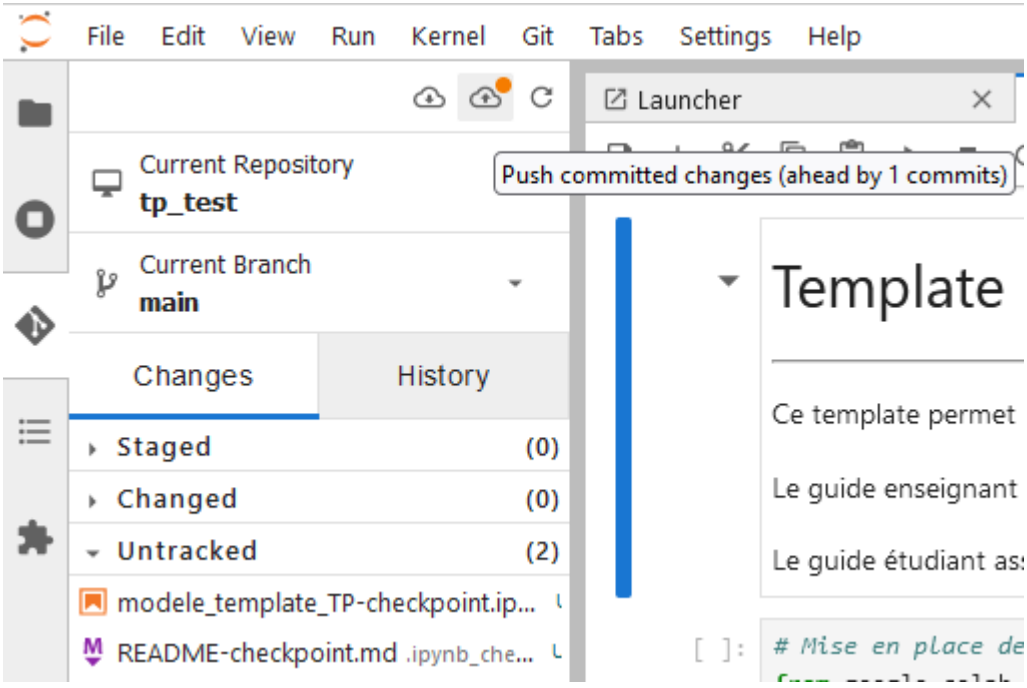
Un message s'affiche alors pour confirmer la prise en compte du commit (local).



Message de confirmation du commit (local)

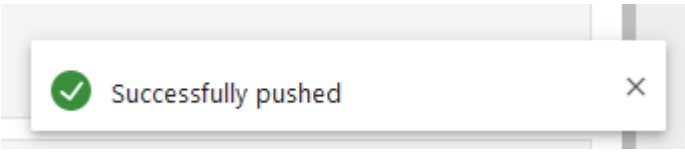
Pour pouvoir envoyer ce commit local vers le serveur Gitlab "gitlab-ia.univ-rennes.fr" il est nécessaire de cliquer sur l'icone de "push" en haut du menu Git.

> On peut voir qu'un commit est en attente d'envoi grâce à la petite pastille orange au dessus de l'icone de "push"



Envoi "push" du commit vers serveur Gitlab

Une fois le "push" réalisé, un message de confirmation s'affiche en bas à droite.



Message de confirmation de l'envoi du commit vers Gitlab "push"

Les modifications sont alors bien sauvegardées sur le serveur.

2. RUDI pour les datasets de données (OpenData)

Présentation de RUDI pour les datasets de données

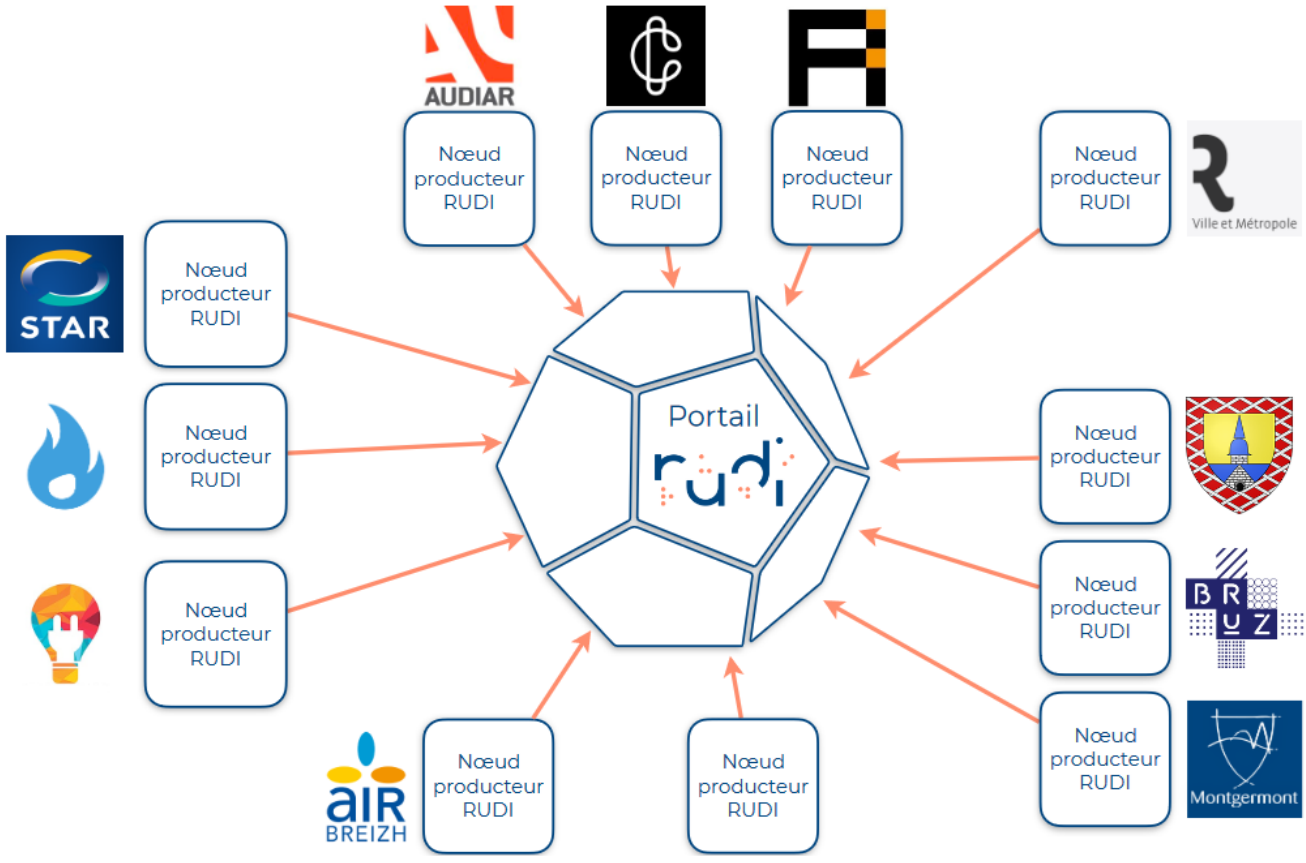
En 2010, Rennes devient la première collectivité Française à ouvrir l'accès à ses données.

RUDI, "Rennes Urban Data Interface", est un projet open source et souverain démarré en 2019 qui s'inscrit dans la démarche d'ouverture des données publiques.

Pour publier séparément données et métadonnées, chaque acteur public gère son propre "Noeud producteur RUDI".

Dans le cadre du projet TIARE, dont DataLab fait partie, un serveur rudi dédié a été créé pour gérer les données associées.

C'est sur ce serveur que les enseignants déposeront les données nécessaires pour les TP.

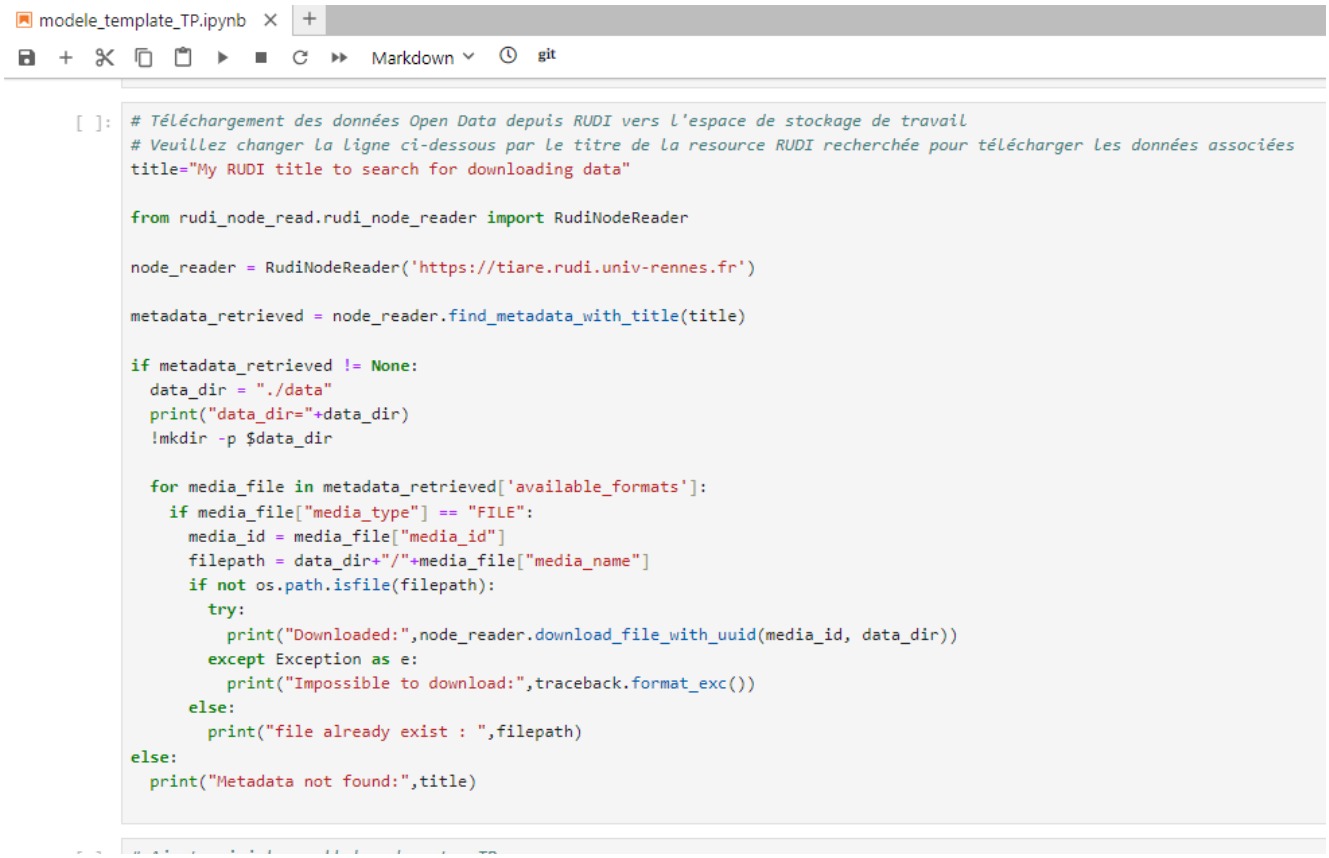


RUDI : fédération de noeuds en étoile

Récupération de données RUDI depuis le notebook

Pour rappel un notebook est composée de plusieurs cellules qui peuvent être du texte ou du code exécutable.

Dans le template modèle de TP, vous pourrez trouver une cellule dédiée au téléchargement de données Open Data depuis RUDI.



Cellule dédiée à la récupération de données depuis "RUDI"

Pour télécharger une ressource RUDI il faut disposer du titre de la ressource.

L'enseignant pourra au choix soit :

- vous fournir ce titre de ressource
- ou sinon il aura déjà préparé le modèle de TP avec le titre pré-rempli.

Pour choisir la ressource RUDI dont les données doivent être téléchargées, il faut modifier la ligne suivante dans le notebook du projet "https://gitlab-ia.univ-rennes.fr/frederic.babon/TP_test.git" :

```
title="My RUDI title to search for downloading data"
```

En remplaçant l'exemple de titre par le titre de la ressource RUDI recherchée.

Dans notre exemple la ligne de la cellule dans JupyterLab est déjà préparée :

```
title="JDD Enseignements IA TIARe"
```

Validez que l'exécution de cette cellule permet de télécharger les données du TP depuis Rudi.

> ***NOTE:*** Dans le cas d'un notebook R, pour utiliser les données du TP le working directory n'est parfois pas le bon dans ce cas on peut ajouter une cellule avec le contenu: setwd("/home/monTP")

3. Bases de GIT en ligne de commande

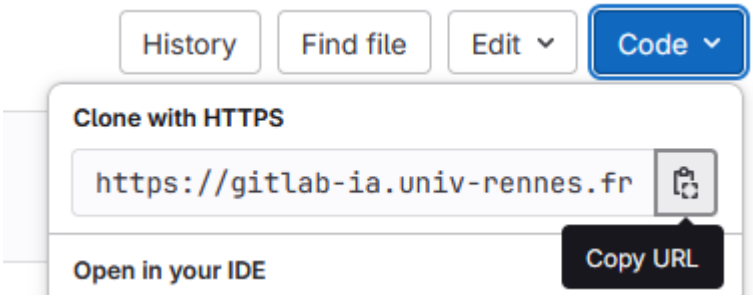
Dans cette partie nous allons découvrir comment utiliser les commandes GIT dans un terminal. Nous utiliserons aussi bien les termes "dépôt", "repository" ou "repo" pour désigner les versions locales ou distantes de notre projet.

Premier clonage du TP : git clone

Git clone permet de copier le contenu d'un repository distant.

```
git clone https://gitlab-ia.univ-rennes.fr/<your-name>/<repo-name>.git
```

L'adresse exacte peut être trouvée dans l'interface GitLab :



Adresse du repository dans GitLab

Un dossier nommé comme le repo est alors créé. Placez-vous dans ce dossier.

Modification d'un fichier : git status

La commande `git status` permet de connaître l'état de sauvegarde des fichiers du repo, qu'ils aient été modifiés, créés, supprimés ou même commités.

Sauvegarde dans le repo local : git add, git commit

Modifiez l'état du repo en effectuant une opération comme:

- créer un nouveau fichier
- ajouter votre nom dans le "README.md"
- ...

La commande `git status` doit révéler le changement effectué.

La commande `git add` permet d'ajouter le fichier dans l'ensemble de ceux à commiter.

```
git add <nouveau-fichier>
git add README.md
```

La commande `git commit` permet de commiter les changements.
Un message sera demandé afin de décrire les changements. Il peut aussi être inclut comme paramètre de la commande :

```
git commit -m "add name to readme.md"
```

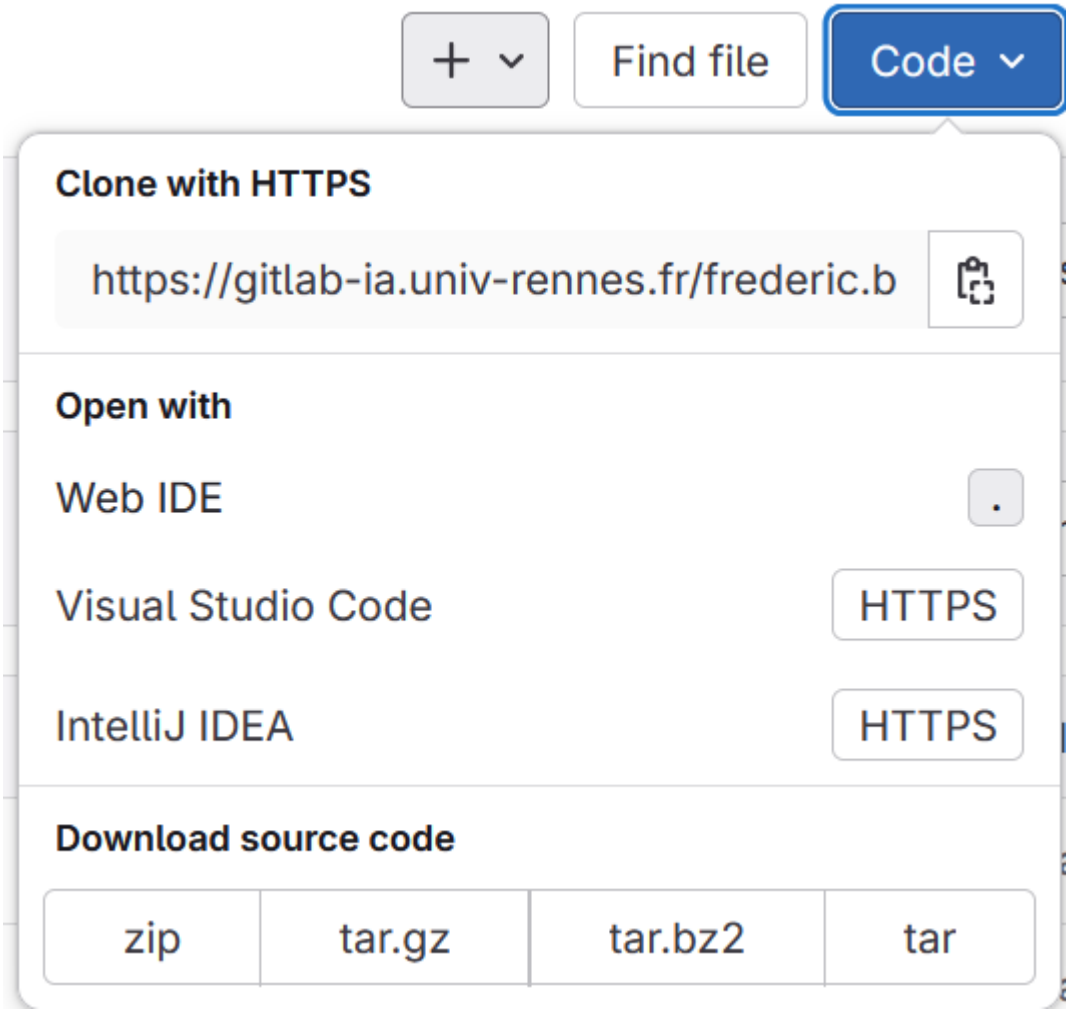
Envoi de la modification sur le serveur: git push

La command `git push` permet d'envoyer les changements sauvegardés dans les commits vers le dépôt distant "gitlab-ia".

```
git push
```

Récupération d'une modification depuis le serveur: git pull

Pour simuler une modification distante, modifiez un fichier directement via l'IDE Gitlab.



Bouton "Edit" dans GitLab pour accéder à l'IDE dans le navigateur

Ensuite, effectuez le commit de ces changements :
dans l'onglet "Source Control", cliquez sur "Commit and push".

Sur la console, utilisez alors la commande `git pull`

```
git pull
```

Les fichiers locaux seront automatiquement mis à jour pour refléter les changements effectués sur l'IDE Gitlab.

Gestion d'un conflit

Sur l'IDE Gitlab, faites une nouvelle modification, commitez et pushez.

Sur le repo local (la console), faites aussi une modification dans le même fichier, puis utiliser les commandes `git add` et `git commit`

```
git add .
git commit -m "another change"
```

> ***NOTE:*** "git add ." permet d'ajouter tous les fichiers du dossier courant ***NOTE:*** "git add -u" permet d'ajouter uniquement les modifications sur les fichiers préalablement commités

Encore sur la console, utilisez la commande `git pull`

```
git pull
```

La commande doit échouer, comme des changements ont été faits à la fois à distance et en local.
Dans le message d'erreur, il est indiqué qu'il faut choisir comment gérer ce problème.
Nous allons choisir comme option la stratégie de *merge* (commande "git config pull.rebase false") puis pull à nouveau.

```
git config pull.rebase false
git pull
```

Dans la plupart des cas, les changements vont être fusionnés et il ne vous restera qu'à commiter les fichiers concernés.
Il se peut cependant que des conflits ne soient pas gérés automatiquement, généralement car des modifications ont été effectuées sur les mêmes lignes.

La commande `git status` indiquera de tels conflits sous le label "both modified".

Les conflits prennent cette forme :

> <<<<<<< HEAD
> {code local}
> =======
> {code distant}
> >>>>>>> c1ec41ab3885c356edfa52e640a2d62a331cd7fc

Il vous faut alors éditer le fichier localement, puis commiter et pusher les changements.

4. Travail collaboratif via GIT

Création d'une branche de développement : git checkout -b

Si un des développeurs souhaite développer une nouvelle fonctionnalité sans perturber les autres pendant le développement, il peut utiliser le concept GIT de "branche" qui permet d'avoir un historique séparé le temps du développement de cette fonctionnalité.

> ***NOTE:*** Pour information la branche principale est nommée "master".

La commande `git checkout -b` permet de créer une nouvelle branche :

```
git checkout -b <nouvelle-branche>
```

Lister les branches : git branch

La commande `git branch` permet de connaître la liste des branches disponibles et de connaître celle qui est active. La commande `git status` indique aussi la branche active.

```
git branch
```

Modification de fichier sur une branche

Maintenant que vous êtes sur une nouvelle branche, vous pouvez modifier et commiter des fichiers sans modifier les autres branches et ainsi conserver, par exemple, une version stable. Les commandes restent les mêmes.

```
git add <fichier-modifié>
git commit -m "commit sur une autre branche"
git push
```

Si la branche n'existe pas sur le repo distant, il vous faudra préciser qu'il faut la créer au moment du push :

```
git push --set-upstream origin <nouvelle-branche>
```

Changement de branche courante : git checkout, git stash

Pour changer la branche courante, il faut utiliser la commande `git checkout` en précisant le nom de la branche cible :

```
git checkout main
```

Si des changements ont été effectués sans commit, vous pouvez sauvegarder les changements temporairement en utilisant `git stash` pour les retrouver grâce à `git stash pop`.

```

```

```
git stash
git checkout main
git checkout <nouvelle-branche>
git stash pop
```

Merge d'une branche vers une autre : git merge

La commande `git merge` permet de fusionner des branches, en rejouant les changements de la branche cible sur la branche active.

```
git checkout main
git merge <nouvelle-branche>
```

En cas de conflit, la méthode vue dans Gestion d'un conflit doit être utilisée.

Félicitations, la partie 3 du guide DataLab est terminée !