Gitlab & Git

Xavier Crégut (Toulouse INP / ENSEEIHT)

2022-

Table des matières

Git est un outil de gestion de versions décentralisé. Il permet de faciliter le développement à plusieurs sur un même projet. Il est aussi utile pour un travail seul en permettant d'organiser son développement, de gérer plusieurs versions de son projet et de conserver les anciennes versions.

Gitlab est un service de forge open-source qui permet d'héberger des projets Git et qui offre des fonctionnalités supplémentaires pour faciliter le développement. D'autres services de forge existent comme Github.

Ce document présente l'instance de Gitlab, appelée Hudson disponible sur le réseau de Toulouse INP, la création d'un jeton et la configuration de Git pour l'utiliser comme moyen d'authentification, la copie d'un projet Git et enfin, les principales commandes de Git.

1 Gitlab à Toulouse INP

Toulouse INP fournit une instance de Gitlab qui est accessible à l'adresse https://hudson.inp-toulouse.fr et que nous appellerons Hudson. Il faut être connecté au réseau de Toulouse INP, physiquement ou via VPN. Dans ce document les termes Gitlab et Hudson seront utilisés indifféremment.

1.1 Accéder à Gitlab

Dans un navigateur, accéder à Hudson. Il faut s'identifier en utilisant votre identifiant et mot de passe Toulouse INP.

La page qui s'affiche présente les projets (*Projects*) auxquels vous participez. Il est aussi possible de sélectionner une vue par groupe (*Groups*) qui organise les projets par groupes. Un groupe est l'équivalent d'un dossier et un projet l'équivalent d'un fichier. Cette vue groupe est plus intéressante si le nombre de projets est important.

1.2 Ouvrir un projet

Ouvrir le projet `projects_group/2023-1SN-PIM/1SN-X/USER/tp` où « `X` » est votre numéro de TP (A, B, etc.) et « `user` » votre identifiant INP. La page d'accueil du projet s'affiche avec au centre les dossiers et fichiers qui le constituent.

1.3 Éléments affichés

La page du projet présente de nombreuses informations (voir la figure ci-après {@fig:projectMainPage}).

Page d'accueil d'un projet Gitlab

Sur la gauche, un menu donne accès aux fonctionnalités classées par catégories. Nous détaillons dans la suite la partie centrale.

Tout en haut, on a le chemin d'accès au projet avec d'abord les groupes puis le nom du projet. Cliquer sur un élément ouvre la page de cet élément.

En dessous sont présentés :

- des informations sur le projet avec quelques statistiques.
- une ligne qui indique avec des couleurs le pourcentage des langages utilisés.
- le dernier message de validation (*commit*) du projet.
- le message lié à la dernière validation (*commit*)
- un bouton qui indique la branche (`main` par défaut), un autre (+) qui donne accès à la création de différents éléments et d'autres boutons sur la droite, en particulier le dernier `clone` qui explique comment copier le projet.
- des boutons liés à quelques actions courantes.

— les dossiers et fichiers présents à la racine du projet avec le dernier message de validation les concernant. On peut cliquer sur un dossier ou sur un fichier pour afficher son contenu.

Par exemple, on peut cliquer sur `tp_git` puis sur `texte.txt` pour afficher le contenu de fichier. En cliquant sur `tp` sur la ligne de vie, on revient sur la page d'accueil du projet.

Dans un premier temps, nous allons utiliser Git en mode ligne de commande même si la plupart de ces commandes seront accessibles via votre IDE préféré.

2 Créer un jeton

À chaque fois que l'on veut accéder au serveur Gitlab et à l'un de ses dépôts, il faut s'authentifier. Il est fastidieux de devoir saisir à chaque fois son mot de passe. C'est également une mauvaise idée de l'enregistrer car ils sera certainement enregistré dans un fichier et pourrait être consulté par quelqu'un ayant accès à notre session/machine/système de gestion de fichiers (session non fermée, programme malveillant...).

Une alternative consiste à utiliser un jeton (*token*) auquel sont associés des droits sur Gitlab.

Les étapes à suivre sont les suivantes depuis Hudson :

1. Cliquer sur son avatar (en haut à droite du menu de gauche) puis choisir *Preferences*.
2. Choisir *Access token* (depuis le menu à gauche)
3. Faire *Add new token*
4. Donner un nom au jeton (par exemple, `gitRWToken`)
5. Modifier (ou supprimer avec la croix) la date de fin de validité du jeton.
6. Cocher les permissions (*scope*) : *write_repository*.
7. Cliquer sur *Create personal access token*.
8. Copier le jeton en lieu sûr. Il ne sera plus accessible ensuite. Si vous le perdez ou l'oubliez, il faudra le révoquer et en créer un nouveau...
9. Garder cette page ouverte tant que le jeton n'a pas été utilisé.

3 Configurer Git pour enregistrer l'accès à un serveur

La commande suivante demande à git de stocker dans un fichier les informations d'authentification (jeton, mot de passe, etc). Il faut la taper dans un terminal.

```
git config --global credential.helper 'store --file ~/.my-credentials'
```

Les informations seront enregistrées *en clair* dans le fichier mentionné (qui ne sera accessible en lecture/écriture que de l'utilisateur).

Attention, si on ne précise pas de fichier, les informations seront enregistrées dans le fichier `.git/config` de votre dossier Git qui par défaut est accessible de tous ! Il y a de plus le risque quelles soient transmises par exemple si vous faites une archive de votre dossier Git.

4 Cloner un projet git

Git est un gestionnaire de version décentralisé : il permet de récupérer tout l'historique d'un projet, c'est le *dépôt local*. Le projet hébergé sur Gitlab ou une autre forge est appelé *dépôt distant*.

Le dépôt local peut être manipulé via la ligne des commandes avec des commandes ressemblant à celles de l'interpréteur de commande mais préfixée par `git`. C'est la commande git qui permet les manipulations des fichiers qui sont en gestion de version via des commandes, en fait des sous-commandes, en précisant éventuellement des options et des arguments suivant la forme suivante : % `git commande [options] [arguments]`

La première chose à faire est donc de créer une copie locale du dépôt distant (clone). Ceci se fait grâce à la commande `git clone`. Nous allons donc récupérer une copie (clone) du dépôt localement. C'est généralement ce qui est fait. Le dépôt locale pourra être synchronisé avec le dépôt distant (celui sur Gitlab ou une autre forge.

4.1 Choisir le dossier dans lequel le dépôt sera cloné.

Les commandes suivantes [1] permettent de créer un *dépôt local* comme copie d'un *dépôt distant* via la commande `git clone`. Au préalable, on choisit le dossier [2] où mettre le dossier correspondant au dépôt local.

```
> cd          # aller dans votre dossier racine (home directory)
> mkdir pim   # créer le dossier pim
> cd pim      # se déplacer dans le dossier pim
```

4.2 Cloner le dépôt distant

Sur la page du projet sur Gitlab, ouvrir le menu « *Clone* » (à droite sur fond bleu) et copier le texte qui est sous « *Clone with HTTP* ». Le texte commence par `git clone https://...`. Il faut alors l'exécuter dans un terminal, donner son identifiant INP et pour mot de passe la valeur du jeton copiée tout à l'heure (commence par `glpat-`). Un nouveau dossier apparaîtra portant le nom du projet.

4.3 Vérifier le dépôt local

Constater que le nouveau dossier correspond au dépôt est apparu et qu'il contient bien les fichiers qui étaient dans le projet sur Gitlab.

En particulier, si on fait `ls -A` on constatera l'existence d'un dossier caché `.git`. Il ne faut surtout pas toucher à ce dossier ni à son contenu. Il est utilisé par Git pour gérer votre dépôt local.

À titre d'information, on peut regarder le contenu du fichier `config` en faisant `cat .git/config`.

1. Le caractère > symbolise l'invite de commande. Il ne faut donc pas le taper.
2. Choisir le dossier en fonction des habitudes ou préférences que vous avez pour organiser vos fichiers. Il est toutefois conseillé d'éviter les espaces, les lettres accentuées et les caractères spéciaux dans les noms de fichiers et dossiers.

5 Modifier un fichier

Aller dans le dossier `tp_git` et modifier le fichier `texte.txt` en supprimant la deuxième ligne et en ajoutant une nouvelle ligne à la fin (par exemple « Avec une nouvelle ligne en plus. »). Vous avez maintenant un fichier local qui est différent du fichier du dépôt local (et donc du dépôt distant). On peut le constater avec la commande `status`.

```
> git status .
...
    modifié :   texte.txt
...
```

Git est généralement bavard et vous indique ce que vous pouvez faire. En l'occurrence, il indique qu'il y a des modifications non validées sur le fichier `texte.txt`.

Le « `.` » est utile pour n'avoir que les modifications du dossier courant (et ses sous-dossiers). Par défaut, Git donne les informations pour le espace de travail complet, indépendamment du dossier courant. Ici, omettre « `.` » ne ferait pas de différences car il n'y a pas de modification dans les autres dossiers.

6 Visualiser les différences apportées sur l'espace de travail

On peut voir les différences entre le fichier de travail et les fichiers enregistrés dans le dépôt local grâce à la sous-commande `diff`. On peut faire :

```
> git diff texte.txt
```

pour voir les modifications du fichier mentionné, ici `texte.txt`. Notons que c'est le format de la commande Unix `diff` qui est utilisé. Les lignes ajoutées apparaissent en vert et les lignes supprimées en rouge. Une ligne modifiée apparaît en rouge (initiale) et vert (modifiée).

On peut aussi fournir un dossier. Dans ce cas, la différence de tous les fichiers du dossier sera affiché.

```
> git diff .     # le dossier courant (et les sous-dossiers)
> git diff       # tous l'espace de travail
```

Si rien ne s'affiche, c'est que la version de travail est identique à la dernière version marquée (celle de l'index, à défaut du dépôt local).

7 Ajouter un fichier à l'index

L'index enregistre tous les fichiers qui seront pris en compte lors de la prochaine validation. C'est la sous-commande `add` qui permet d'ajouter un fichier à l'index.

```
> git add texte.txt
```

Le fichier est alors ajouté à l'index. Le résultat de la commande `git status .` permet de le confirmer. Le fichier `texte.txt` apparaît en vert précédé de la mention `modifié :`. Notons que Git indique comment supprimer le fichier de l'index. Il suffit de faire `git restore --staged texte.txt`.

Notons également que `git diff texte.txt` n'affiche plus de différence. La dernière version du fichier `texte.txt`est bien celle que l'on vient d'ajouter à l'index.

On pourrait ajouter d'autres fichiers à l'index. Normalement, on ajouter à l'index toutes les modifications qui correspondent au travail que nous souhaitions faire.

8 Valider les modifications

Une fois l'index défini (et complet), on peut valider les modifications correspondantes via la sous-commande `commit`. Un message (option `-m`) permet de préciser les modifications apportées par cette validation.

La règle est de rédiger le message de manière à ce qu'il puisse est utilisé précédé de "Ces modifications permettent de…". Ici on peut utiliser le message "Modifier texte.txt d'après le tutoriel.", c'est-à-dire « Ces modifications permettent de modifier texte.txt d'après le tutoriel. ».

Si on ne précise pas l'option `-m`, git lancera l'éditeur par défaut (souvent `vi` [3]) pour permettre de saisir le message. La commande à taper ici est donc :

```
> git commit -m "Modifier texte.txt d'après le tutoriel."
```

Notons qu'il est possible de modifier le dernier message de validation via la commande `git commit --amend`.

9 Historique des modifications

On peut consulter l'historique des modifications via la sous-commande `log`. Par exemple, on peut faire :

```
> git log texte.txt
```

qui donne l'historique du fichier `texte.txt` ou, pour avoir l'historique du dépôt local, tout simplement :

```
> git log
```

Pour avoir un affichage plus concis, on peut faire :

```
> git log --oneline
```

10 Pousser les modifications sur le dépôt distant

Les modifications réalisées ont été enregistrées (validées) sur le dépôt local. Aucun modification n'a été apportée sur le dépôt distant. On peut le constater en allant sur la page du projet sur Gitlab : même en rechargeant la page, le contenu du fichier `texte.txt` n'est pas modifié.

Pour pousser les modifications sur le dépôt distant, on utilise la sous-commande `push`.

```
> git push
```

On peut constater que les modifications sont maintenant visibles sur Gitlab.

3. L'éditeur par défaut est souvent `vi` (ou maintenant `vim`, la version terminal de `Gvim`), un éditeur modal (mode commande et mode édition) qui est déroutant quand on a l'habitude d'un éditeur je-clique-je-tape. Si vous ne le connaissez pas, sachez qu'on peut le quitter en faisant `:q!`. Si vous êtes curieux, vous pouvez faire `:help`. Il faut éventuellement taper « Échap » avant `:q!`.

La sous-commande `push` pousse tous les modifications locales validées sur le dépôt distant. Il est à noter que la commande `push` ne pourra pas être réalisée si des modifications ont déjà été apportées sur les serveurs distants, par exemple par un autre membre du projet. Il faudra au préalable récupérer dans le dépôt locale les modifications réalisées sur le dépôt distant.

11 Récupérer les modifications distantes

La sous-commande `pull` permet de récupérer les modifications du serveur distant (`fetch`) et de les fusionner dans le dépôt local (`merge`).

```
> git pull
```

Ici, la commande n'a pas d'effet car aucune modifications n'a été faite sur le dépôt distant (à part celle que l'on a poussée mais et qui est déjà dans notre dépôt local).