Git(Lab) : Pour bien commencer

GitLab

GitLab est un gestionnaire de projets de développements collaboratifs. Il intègre notamment une panoplie d'outils visant à faciliter les différents aspects lié au développement d'une application que sont la gestion des versions du code, la collaboration entre plusieurs contributeurs, la documentation et le partage du projet. Le FIL, département d'Informatique de l'Université de Lille, met à votre disposition un dépôt GitLab que vous pouvez utiliser librement pour réaliser vos TPs et projets en collaborant efficacement avec vos camarades.
Ce service est diponible à l'adresse suivante : `http://gitlab-etu.fil.univ-lille1.fr` et vous pouvez vous y connecter avec vos codes d'accès habituels. De l'extérieur du campus, l'accès au GitLab ou au serveur Git nécessite l'utilisation du VPN [1]. Attention, les dépôts que vous avez créés sont automatiquement supprimés *en fin d'année*.

Cette fiche est une rapide présentation de l'utilisation de GitLab pour vous aider à débuter. GitLab sera fortement utilisé dans votre cursus, cette année et les suivantes. Se familiariser dès maintenant avec son fonctionnement est donc utile et en fait indispensable. De plus Git est un outil très couramment utilisé en entreprise ou dans des projets de développement collaboratifs.
Pour des explications plus poussées sur le fonctionnement de GitLab ou des usages avancés, vous pouvez consulter la documentation en ligne de GitLab (`https://about.gitlab.com`). Vous pouvez aussi trouver des des ressources supplémentaires sur le portail de POO (onglets Documents).

Il est certainement intéressant de réaliser ces différentes opérations avec son binôme, même si ce n'est pas indispensable.

Étape n°1 : créer son dépôt distant

Connectez-vous sur l'interface en ligne de GitLab puis cliquez sur New project. Indiquez un Project name pour votre projet, par exemple *noms-des-binomes*-POO [2].
Indiquez ensuite une description du projet telle que « *les rendus des TP de POO des étudiants* xxx *et* yyy*, groupe* i *du S4 informatique* ». Cochez qu'il s'agit d'un projet Private, puis validez en cliquant sur Create project.
Vous pouvez ensuite donner des droits à votre binôme sur ce dépôt pour qu'il puisse accéder et contribuer au code que vous allez développer collaborativement. Il faut que votre binôme se soit connecté au moins une fois à GitLab, si ce n'est pas encore le cas, laissez-lui la main pour qu'il le fasse maintenant. Ensuite, cliquez sur le menu Members (en haut à droite, au niveau de la roue crantée), renseignez l'identifiant de votre binôme (son *login*), changez les droits d'accès pour Master, puis cliquez sur le bouton Add to project.
Renouvelez l'opération pour enregistrer votre enseignant de TP/TP en tant que Developer afin qu'il puisse accéder à votre travail.

Étape n°2 : créer son dépôt local

Votre projet est désormais créé sur le dépôt distant et vous allez pouvoir en créer la version locale dans votre espace de travail.

Identification SSH Avant cela, afin de faciliter l'authentification et vous éviter de saisir votre mot de passe lors de chaque opération, vous pouvez enregistrer votre clé publique SSH qui sera utilisée par GitLab pour vous authentifier. Cette étape est optionnelle [3] mais vous facilitera l'utilisation quotidienne de GitLab. Elle est donc fortement conseillée.

1. documentée à `http://fil.univ-lille1.fr/documentation-vpn`
2. Pour éviter des temps de réponse trop longs lors des opérations sur votre dépôt, il vous est **fortement déconseillé** de nommer votre dépôt *POO-quelque-chose*. Commencer par votre nom permet de garantir qu'il y aura peu de dépôts qui commenceront pas les mêmes caractères et donc des temps de réponse faibles lors des manipulations.
3. Attention, si vous passez cette étape la commande création du dépôt en local sera différente.

Pour enregistrer votre clef publique, choisissez Profile Settings dans le menu en haut à droite, puis sur l'onglet SSH Keys. Dans le champ Key, copiez votre clé publique SSH en faisant un copier/coller du contenu retourné par la commande :

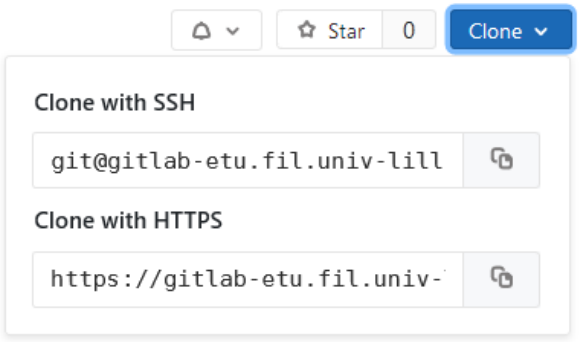
```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIwABAQEAklOUpkDHrfHY17SbrmTIpNLTGK9Tjom/BWDSU
GPl+nafzlHDTYW7hdI4yZ5ew18JH4JW9jbhUFrviQzM7xlELEVf4h9lFX5QVkbPppSwg0cda3
Pbv7kOdJ/MTyBlWXFCR+HAo3FXRitBqxiX1nKhXpHAZsMciLq8V6RjsNAQwdsdMFvSlVK/7XA
t3FaoJoAsncM1Q9x5+3V0Ww68/eIFmb1zuUFljQJKprrX88XypNDvjYNby6vw/Pb0rwert/En
mZ+AW4OZPnTPI89ZPmVMLuayrD2cE86Z/il8b+gw3r3+1nKatmIkjn2so1d01QraTlMqVSsbx
NrRFi9wrf+M7Q== timoleon@laptop.local
```

Si jamais, un message vous indique que le fichier n'existe pas, alors vous devez créer une paire de clés publique/privée en utilisant la commande :

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/timoleon/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/toto/.ssh/id_rsa.
Your public key has been saved in /Users/toto/.ssh/id_rsa.pub.
The key fingerprint is:
43:c5:5b:5f:b1:f1:50:43:ad:20:a6:92:6a:1f:9a:3a timoleon@laptop.local
```

Une fois votre paire de clés SSH créée, vous pouvez copier le contenu de la clé publique (le fichier `.pub`) dans GitLab pour qu'il vous reconnaisse ensuite. Attention, il est possible que GitLab prenne quelques secondes ou minutes pour reconnaître votre clé publique SSH.
Vous pouvez créer et enregistrer dans GitLab plusieurs clés publiques SSH pour vous identifier sur les différentes machines que vous utilisez à l'Université ou chez vous avec votre machine personnelle.

Première synchronisation du dépôt Vous pouvez maintenant synchroniser votre dépôt local avec le dépôt distant créé sur GitLab. Pour ce faire, dans votre espace de travail positionnez-vous dans le dossier dans lequel vous voulez placer vos travaux de POO.
Identifiez ensuite l'adresse de votre dépôt à partir du bouton Clone sur la droite de la fenêtre :



Le petit bouton à droite de cette adresse permet de la coller dans le presse-papier.

La première synchronisation se réalise grâce à la commande `git clone`. Il faut lui indiquer l'adresse du dépôt, si vous avez nommez celui-ci `NOM_DE_VOTRE_DEPOT`, cette adresse ressemblera à ce qui est indiqué :

```
$ git clone git@gitlab-etu.fil.univ-lille1.fr:[VOTRE_LOGIN]/NOM_DE_VOTRE_DEPOT.git
NOM_DU_DOSSIER_LOCAL
Clonage dans 'NOM_DU_DOSSIER_LOCAL'...
warning: Vous semblez avoir cloné un dépôt vide.
Vérification de la connectivité... fait.
```

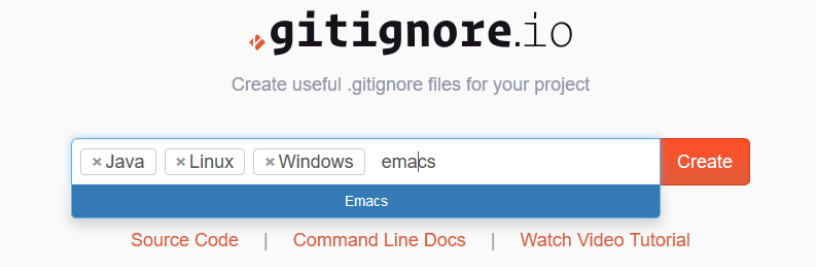
Cette commande permet de créer un dépôt Git local synchronisé avec votre dépôt distant et le place dans un dossier nommé `NOM_DU_DOSSIER_LOCAL` (le dernier argument de la ligne de commande).

Cette opération n'est à faire qu'une seule fois lors de la création du dépôt local (mais vous pouvez créer autant de dépôts locaux que vous le souhaitez). Nous allons maintenant voir comment travailler avec ce dépôt local et maintenir sa synchronisation avec le dépôt distant.

Attention Si vous n'avez pas saisi de clef SSH pour vous identifier sur GitLab vous devrez cloner voter projet en utilisant l'adresse `https` :
`$ git clone https://gitlab-etu.fil.univ-lille1.fr:[VOTRE_LOGIN]/NOM_DE_VOTRE_DEPOT.git NOM_DU_DOSSIER_LOCAL`
Il vous faudra alors fournir votre login et mot de passe lors des opérations vers le dépôt distant. Cela n'est donc pas conseillé car peu pratique à l'usage.

Étape n°3 : ajouter des fichiers

Un dépôt Git est destiné à héberger du code source et non pas des fichiers compilés. Pour éviter de stocker de tels fichiers par erreur, nous allons enregistrer un fichier `.gitignore` qui va indiquer à Git les fichiers qu'il doit ignorer lors des opérations d'ajout. La façon la plus simple de générer ce fichier est de vous rendre sur le site `https://www.gitignore.io`, d'indiquer que vous réalisez un projet `java`, de préciser les système, outils, etc. utilisés puis de cliquer sur le bouton `generate` :



Il suffit ensuite de copier le contenu qui vous est retourné dans un fichier `.gitignore` stocké à la racine du dossier correspondant à votre dépôt local. A nouveau cela est fait une fois pour toute.

La commande `git status` permet de connaître l'état du dépôt local par rapport au distant. Essayez la sur votre dépôt (vous devez vous trouvez dans le dossier du dépôt local, ou l'un de ses sous-dossiers quand il y en a).
Git détecte les fichiers (ici uniquement `.gitignore`) qui se trouvent dans le dépôt local et qui ne sont pas encore présents sur le dépôt distant :

```
$ git status
Sur la branche master
Validation initiale
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        .gitignore

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents
(utilisez "git add" pour les suivre)
```

Trois étapes sont nécessaires pour ajouter ces fichiers dans le dépôt distant. Ce sont les mêmes à appliquer pour mettre à jour un fichier modifié localement.

Ajouter les fichiers La première étape consiste à ajouter les fichiers à synchroniser (nouveaux ou modifiés) en utilisant la commande `git add`, comme suggéré dans la trace obtenue ci-dessus :

```
git add .gitignore
```

Plusieurs commandes `add` peuvent être enchainées si plusieurs fichiers sont à synchroniser. On peut aussi ajouter un dossier ce qui aura pour conséquence d'ajouter tout son contenu.
Exécutez la commande `git status` et constatez le changement du message par rapport au précédent.

Créez un fichier `readme.md` dans votre dépôt, placez-y un contenu tel que « *dépôt des TP de POO des étudiants* xxx *et* yyy. ».
Utilisez la commande `git status` pour vérifier que ce fichier est nouveau et non suivi puis ajouter le comme fichier à synchroniser.

Valider les changements La seconde étape consiste à valider (et à leur donner un numéro de version) l'ensemble des changements qui ont été accumulés par les différents `add`. Cela se fait grâce à la commande `git commit`. Chaque opération de *commit* doit être accompagnée d'un message :

```
$ git commit -m "initialiation du dépôt : .gitignore et readme.md"
[master (commit racine) 00fa862] initialiation du dépôt : .gitignore et readme.md
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 .gitignore
 create mode 100644 readme.md
```

Le message de *commit* passé en paramètre (option `-m`) n'est pas à négliger car il doit synthétiser la nature de la modification qui a été apportée sur le code du projet. Ces messages apparaîtront dans le journal du dépôt. Ils sont utilisés par les autres collaborateurs au projet pour prendre connaissance de ce qui a été fait. Dans le cas où l'on n'indique pas l'option `-m` un éditeur s'ouvre pour proposer de rédiger le message de *commit*. Certains considèrent que l'option `-m` n'est pas une bonne pratique car elle incite à faire des messages trop courts.

À cet instant, si vous vous connectez sur l'interface de GitLab vous constaterez que le dépôt est toujours vide : consultez l'onglet Repository de votre dépôt.
En effet, pour l'instant les fichiers sont simplement versionnés par Git dans le dépôt local mais ils n'ont toujours pas été transmis au dépôt distant. C'est le rôle de la troisième étape.

Envoyer les changements La troisième étape correspond donc à l'envoi effectifs des changements depuis le dépôt local vers le dépot distant. On utilise pour cela la commande `push` :

```
$ git push -u origin master
Décompte des objets: 4, fait.
Delta compression using up to 4 threads.
Compression des objets: 100% (3/3), fait.
Ecriture des objets: 100% (4/4), 572 bytes | 0 bytes/s, fait.
Total 4 (delta 0), reused 0 (delta 0)
To gitgitlab-etu.fil.univ-lille1.fr:[VOTRE_LOGIN]/NOM_DE_VOTRE_DEPOT.git
00fa862..fbc682f master -> master
La branche master est paramétrée pour suivre la branche distante master depuis origin.
```

Le message indique que tout s'est correctement déroulé et que les changements regroupés dans le *commit* ont été transmis au dépot distant.
L'exécution de la commande `git status` permet de vérifier que le dépôt local et le dépôt distant sont désormais synchronisés.
En consultant le Repository dans l'interface de votre dépôt GitLab vous pouvez en effet constatéer que les fichiers ont été ajoutés. Vous pouvez aussi remarquer que le contenu fichier `readme.md` est affiché dans cette page et la page d'accueil de votre projet [4].
La dernière phrase du message du *push* est une conséquence de l'utilisation de l'option `-u`. Elle signifie que les prochaines commandes `git push` pourront se faire sans paramètre : par défaut on effectuera ces opérations sur la branche *master* vers le dépôt distant désigné par `origin`. Les branches sont des notions très importantes dans le bon usage de Git mais dans cette première approche nous travaillerons sur la seule branche `master`.

Synthèse Ces trois étapes `git add`, `git commit` et `git push` sont nécessaires pour transmettre les changements que vous réalisez localement vers le dépôt distant.
Revoyons ces étapes à travers un petit exercice :

4. Vous pouvez utiliser la syntaxe *markdown* `https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet` pour mettre en forme le fichier `readme.md` – ce qui explique l'extension `.md` .

1. créez dans votre dépôt local un dossier `TP0` et placez-y un nouveau fichier `a.txt` (peu importe son contenu). Modifiez également votre fichier `readme.md` (ajoutez un `#` en début de la première de texte par exemple ;
2. consultez le résultat de la méthode `git status` ;
3. faites le nécessaire pour synchroniser les modifications du dépôt local vers le dépôt distant ;
4. vérifiez avec `git status` que tout est à jour.

Étape n°4 : Récupérer des fichiers

Nous avons vu comment envoyer des modifications vers le dépôt distant, reste à savoir comment réaliser l'opération inverse : rapatrier depuis le dépôt distant vers le dépôt local.

Commençons par créer un second dépôt local. Deux solutions s'offrent à vous : soit votre binôme se connecte et crée son propre dépôt local en clonant le projet comme décrit précédemment, soit vous créez un second dépôt local dans votre espace de fichiers. Une fois l'une ou l'autre des ces actions effectuée vous disposez de deux dépôts locaux synchronisés avec le dépôt distant.

Dans l'un de ces deux dépôts locaux faites un changement (modification d'un fichier existant ou création d'un nouveau fichier) puis envoyez la vers le dépôt distant.
Le second dépôt se retrouve alors nécessairement désynchronisé avec le dépôt distant. Il faut donc dans ce dépôt récupérer la nouvelle version du dépôt distant afin de pouvoir continuer à y travailler avec une version à jour. Pour cela il faut utiliser la commande `git pull` (dans le second dépôt !). Si vous aviez par exemple modifié le fichier `readme.md` dans le premier dépôt vous obtenez alors dans le second dépôt une trace de la forme :

```
$ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Dépaquetage des objets: 100% (3/3), fait.
Depuis gitlab-etu.fil.univ-lille1.fr:[LOGIN]/NOM_DE_VOTRE_DEPOT
f5d366c..f4d6258 master -> origin/master
Mise à jour f5d366c..f4d6258
Fast-forward
readme.md | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
```

Dans ce dépôt vous pouvez alors constater que vous avez bien la dernière version des fichiers modifiés et/ou ajoutés sur l'autre dépôt.

La commande `git log` vous permet de consulter le journal des *commit* et vous y voyez apparaitre les différents messages utilisés lors de ces opérations.

Pour résumer

Quand vous voudrez travailler dans votre dépôt local. Il faudra commencer par le mettre à jour et récupérer les modifications qui ont été enregistrées sur le dépôt distant, il vous suffira d'exécuter la commande :

```
$ git pull
```

C'est un réflexe à acquérir que de systématiquement commencer par exécuter cette commande avant tout travail dans le dépôt local [5].

Vous apportez ensuite localement vos modifications en les éditant localement.

Puis, quand vous souhaitez valider et partager ces modifications fichier avec votre binôme, il suffit de l'envoyer vers le dépôt local en exécutant la séquence de commandes :

5. Sinon vous ne tarderez pas à découvrir des messages de la part de Git qui vous invitera à exécuter la commande `git merge`. La plupart du temps faire ce qui vous est suggéré par Git en exécutant cette commande puis en la validant par *commit* puis *push* devrait suffire à vous remettre sur pied. Sinon cherchez des informations sur internet ou demandez à votre enseignant.

```
$ git add mon_fichier.ext mon_dossier/
$ git commit -m "Mon message de commit"
$ git push
```

Bien entendu, Git ne se limite pas à ces quelques commandes (voyez `git help -a`, par exemple `git rm` permet de supprimer un fichier du dépôt, etc). N'hésitez pas à consulter la documentation de Git pour découvrir son fonctionnement et les différentes commandes utilisables.

Rendre des TP en POO

Pour rendre vos TP en POO vous devrez donc :

1. créez dans votre dépôt local un dossier correspondant à ce TP ;
2. configurer le fichier `.gitignore` pour ce dépôt ;
3. rédiger dans ce dossier un fichier `readme.md` présentant le TP ;
4. placer dans ce dossier tous les fichiers que vous devez rendre pour ce TP ;
5. ajouter (`git add`), valider (`git commit`), transférer (`git push`) ces fichiers sur le dépôt distant avant la date échéance indiquée par votre enseignant.

Les bonnes pratiques de Git poussent à utiliser le triplet *add/commit/push* régulièrement sur de petites modifications qui devront être bien renseignées par le message de *commit*. A vous d'approfondir par vous-mêmes le bon usage de Git en vous informant grâce aux nombreuses ressources existantes.