Utilisation de Git et GitLab

From devSpot-wikis

Contents

Références

- Pro Git Book (http://git-scm.com/book/fr)
- Git Community Book (https://alexgirard.com/git-book/index.html)
- Documentation officielle GitLab (http://doc.gitlab.com/ce/)

Mémo d'utilisation de Git

Travailler avec Git en local

- **Initialisation d'un dépôt** :

dans le répertoire existant du logiciel taper :

```
git init
```

- **Savoir où on en est dans les fichiers suivis et modification à ajouter au prochain commit** :

```
git status
```

- **Ajouter des fichiers pour suivre leur version** :

taper

```
git add nom-du-fichier
```

puis

```
git commit -m 'version initiale du projet'
```

*git add permet d'ajouter (indexer) les fichiers au projets (possible d'ajouter tout avec git add *).*
git commit permet d'enregistrer (valider) les modifications (l'option –amend permet d'annuler le commit).
ATTENTION : toujours inspecter les modifications pour savoir si tout est bien indexé en tapant la commande git status.

- **Se définir un patron de travail** :

renseigner un fichier .gitignore pour éviter d'indexer ou de valider ce qu'on ne veut pas, exemple :

```
# un commentaire, cette ligne est ignorée
# pas de fichier .a
*.a
# mais suivre lib.a malgré la règle précédente
!lib.a
# ignorer uniquement le fichier TODO à la racine du projet
/TODO
# ignorer tous les fichiers dans le répertoire build
build/
# ignorer doc/notes.txt, mais pas doc/server/arch.txt
doc/*.txt
# ignorer tous les fichiers .txt sous le répertoire doc/
doc/**/*.txt
# Don't ignore .gitignore
!.gitignore
```

Autres exemples sur [1] (https://github.com/github/gitignore) . Penser aux fichiers *~ et *.lock.

- **Supprimer ou déplacer un fichier** :

```
git rm nom-du-fichier
git mv nom-du-fichier
```

Les fichiers indexés qui feront parties de la prochaine validation :

```
git diff --staged
```

- **Obtenir des infos sur la version actuelle du projet** :

```
git log
```

(git log -p -2 permet de voir les différences avec les 2 commits précédents)

- **Lister et créer des étiquettes pour les versions stables** :

```
git tag pour lister les étiquettes.
git tag -a v1.4.0 -m 'Version 1.4.0' pour ajouter une étiquette au dernier commit.
```

Rappel – règles pour les étiquettes : vA.B.C avec A=version majeure, B=version mineure, C=correctif On commence la numérotation à 0. On réserve A=0 pour le protoype, A=1 est la première version stable.

- **Manipuler des branches**:

```
git branch nom-de-la-branche     pour créer une nouvelle branche
git checkout nom-de-la-branche   pour basculer sur cette nouvelle branche
git branch -b nom-de-la-branche  pour faire les deux d'un coup
git branch -d nom-de-la-branche  pour la supprimer
```

- **Différence entre des branches**:

```
git diff branche1..branche2      pour une différence dans les deux sens
git diff branche1...branche2     pour les modifications non commités dans la branche1
```

- **Fusionner la branche courante avec une autre branche** :

```
git merge <nom_branche>
git merge <nom_branche> --no-ff  pour gérer nous même les conflits et éviter le fast-forward
```

- **Pour abandonner la fusion si on s'est emmêlé les pinceaux** :

```
git reset --hard HEAD~1          pour supprimer le dernier merge (~3 si on veut effacer les 3 derniers, etc)
```

La bonne pratique pour travailler avec les branches

Git Flow! (http://danielkummer.github.io/git-flow-cheatsheet/)

Travailler avec un dépôt distant et GitLab

- **Création d'un dépôt** :

Se rendre sur l'interface Gitlab sur le serveur devspot et créer le dépôt à partir de l'interface graphique.

- **Récupérer un dépôt distant en local pour la première fois (clone)** :

(exemple avec le projet MiscBox)

```
git clone http@osur.univ-reunion.fr/outils/devspot/git:promise/miscbox.git MiscBox
```

Attention : ne pas oublier que le nom du dépôt distant se termine par .git. On peut récupérer l'URL exacte dans la page « Activity » du projet.

- **Enregistrer plusieurs dépôts**:

```
git remote add nom_depot adresse_depot
```

Par exemple si on veut ajouter le dépôts renater du projet en SSH:

```
git remote add sourcesup git+ssh://git@git.renater.fr:2222/projet.git
```

et pour pousser la branche develop on n'a plus qu'à faire: git push sourcesup develop

- **Récupération des branches distantes**:

Avec la commande précédente on vient de récupérer la branche principale du projet (master).
Pour voir l'ensemble des branches en local:

```
git branch
```

... ce qui dans ce cas va donner uniquement

```
*master
```

Pour récupérer la branche distante gestion_attributs:

```
git checkout -b gestion_attributs origin/gestion_attributs
```

On va récupérer et se positionner automatiquement sur cette nouvelle branche:

```
*gestion_attributs
 master
```

- **Pousser la branche master locale vers le serveur depuis lequel vous avez cloné le dépôt (origin)** :

```
git push origin master
```

ou sinon plus générique

```
git push [nom-distant] [nom-de-branche-locale]
```

Attention : si quelqu'un a poussé entre temps on aura un message de rejet, on doit d'abord tirer les modifications de la personne précédente, les fusionner avec les votres et pousser.

- **Pousser une branche vers plusieurs dépôts à la fois (GitLab OSU-R et Sourcesup)** :

```
git remote add latotale http://osur.univ-reunion.fr/outils/devspot/git/promise/atmdata.git
git remote set-url --add latotale git+ssh://git@git.renater.fr:2222/atmdata.git
```

- **Attention à la limitation en taille par défaut dans Git quand vous poussez vers le dépôt distant**:

Si vous avez l'erreur suivante:

```
fatal: The remote end hung up unexpectedly
```

c'est que la configuration Git de la taille du buffer d'envoi n'est pas adaptée à votre paquet, pour être tranquille on passe à 500MB:

```
git config http.postBuffer 524288000
```

- **Supprimer une branche sur le dépôt**:

```
git push [nom-distant] :[nom-de-branche]
```

- **Mettre à jour la branche locale à partir du dépôt distant**:

```
git pull origin master
```

ou sinon plus générique

```
git pull [nom-distant] [nom-de-branche-locale]
```

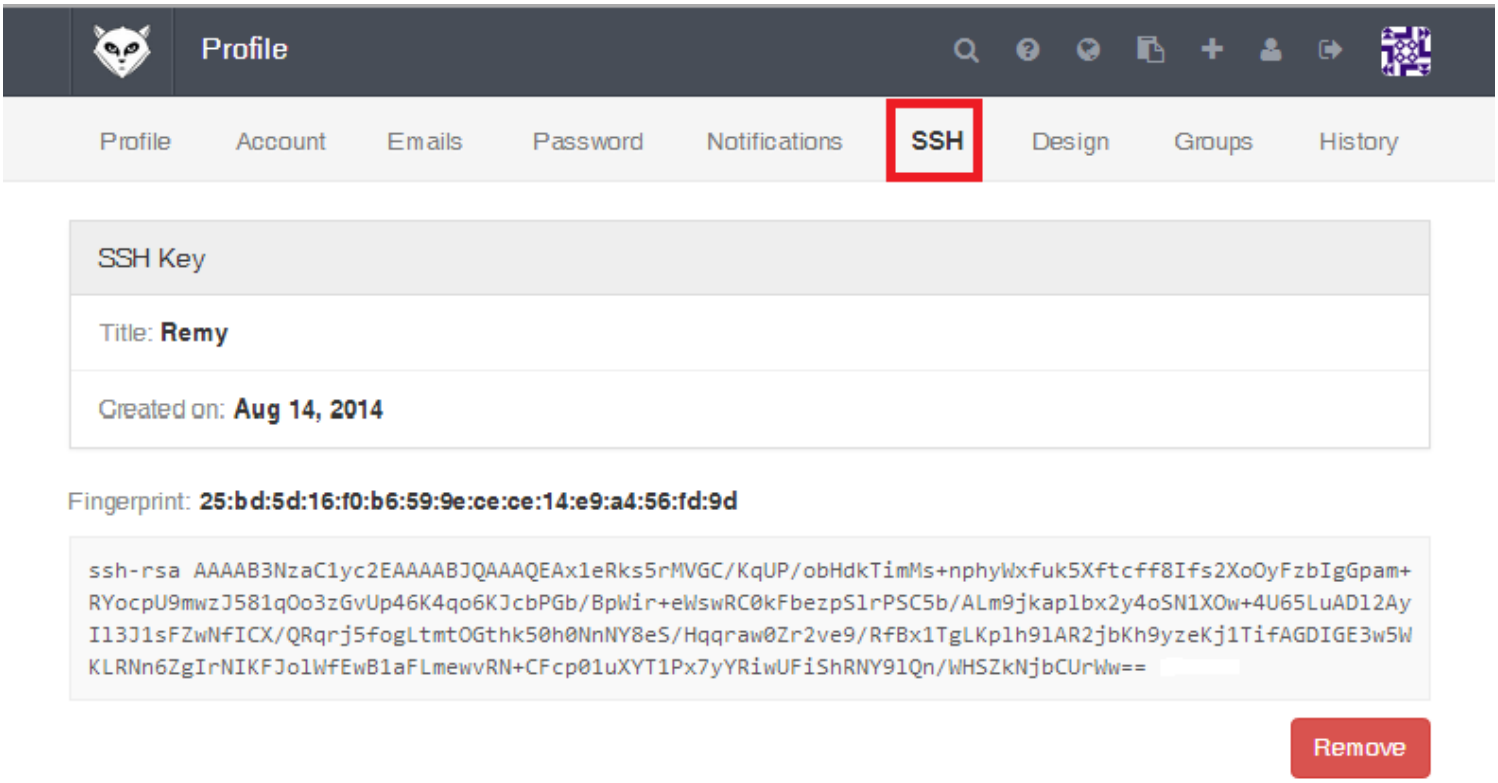
Ce qui est l'équivalent de:

```
git fetch origin master
git merge origin master
et on peut mettre du coup l'option --no-ff à merge
```

Pour les utilisateurs Windows

- **Ajouter sa clé SSH à GitLab**:

Utiliser l'outil puttyGen pour générer une paire de clés privées/publiques. Si vous avez déjà une paire de clés, l'ouvrir via puttyGen et récupérer la clé publique (car ATTENTION : copié/collé depuis le bloc-note ne fonctionne pas).

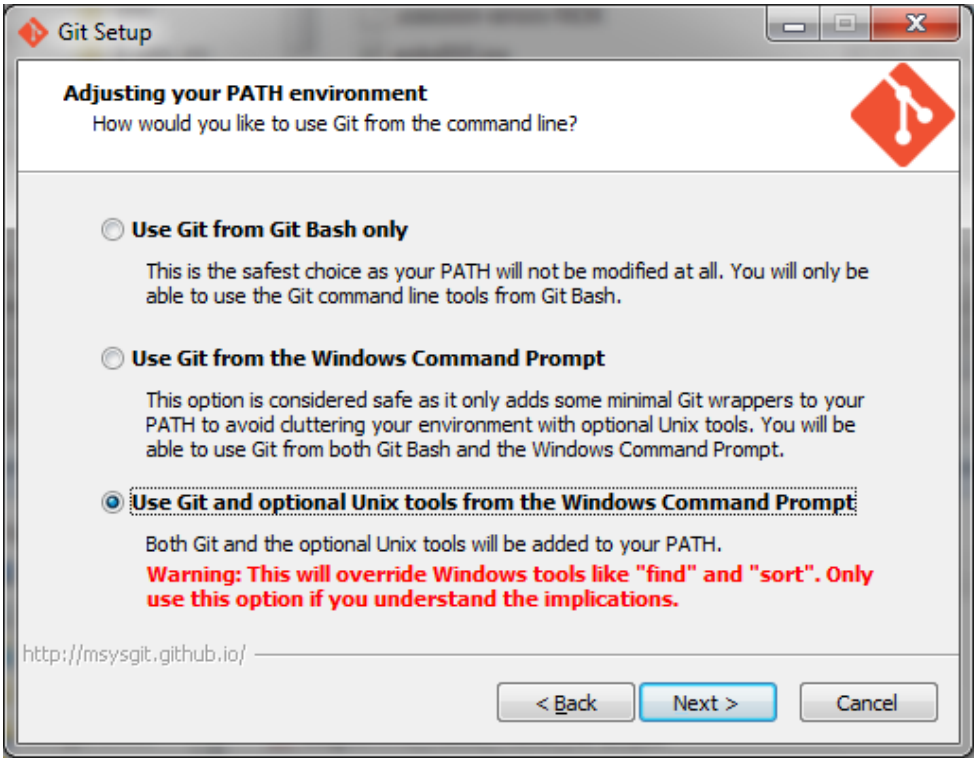


- **Installation de TortoiseGit**:

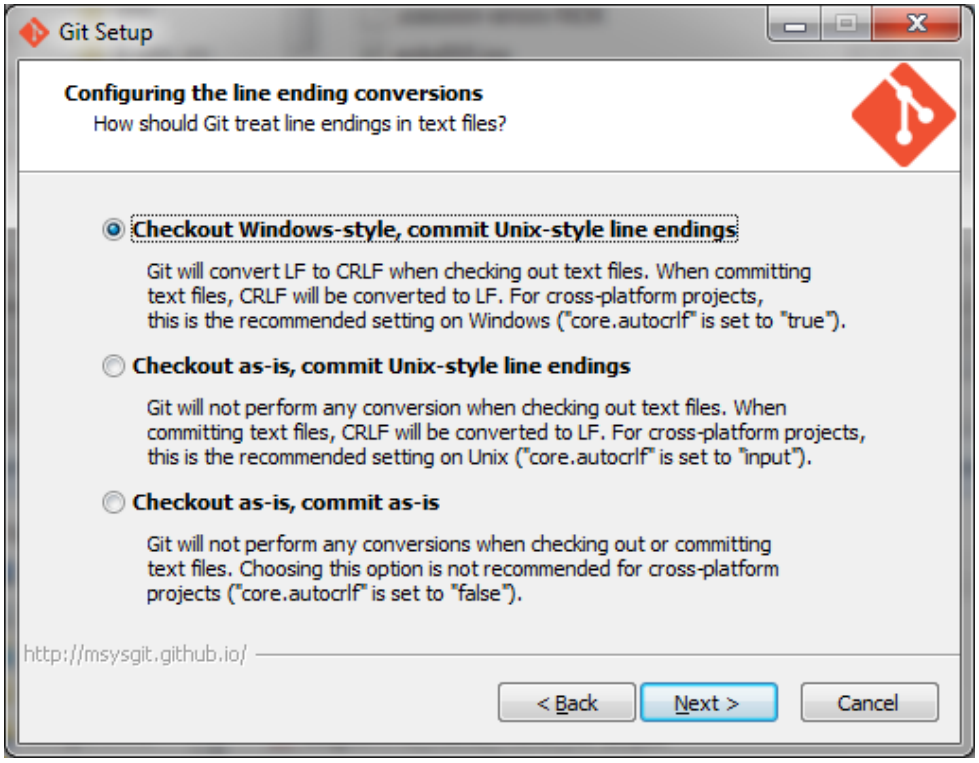
Ajoute les commandes GIT, de manière graphique, dans l'environnement de l'explorer de Windows. Suivre les indications d'installation du programme.

- **Installation de "Git for windows"**:

Lors de l'installation, 2 options peuvent être utiles. La première (ci-dessous) rend les commandes UNIX et Git utilisables depuis ms-dos.

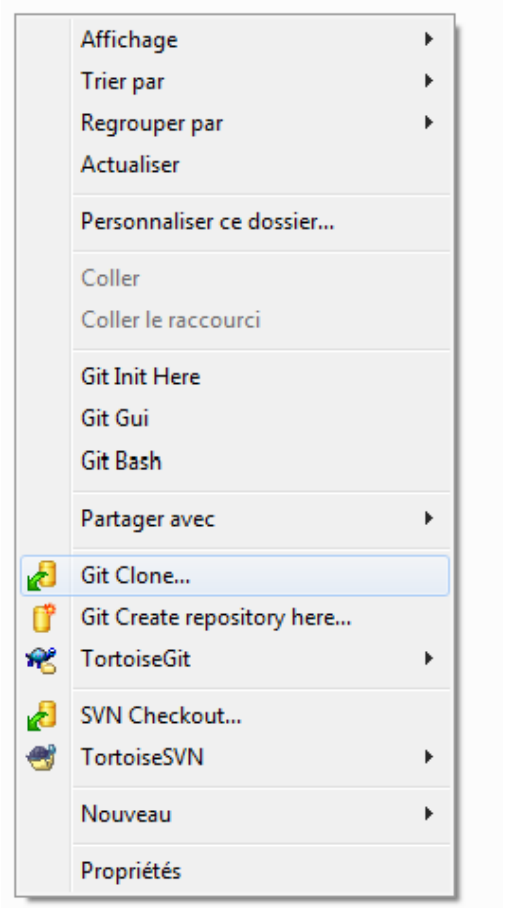


La deuxième option utile est de convertir les caractères « retour à la ligne » de windows en système UNIX :

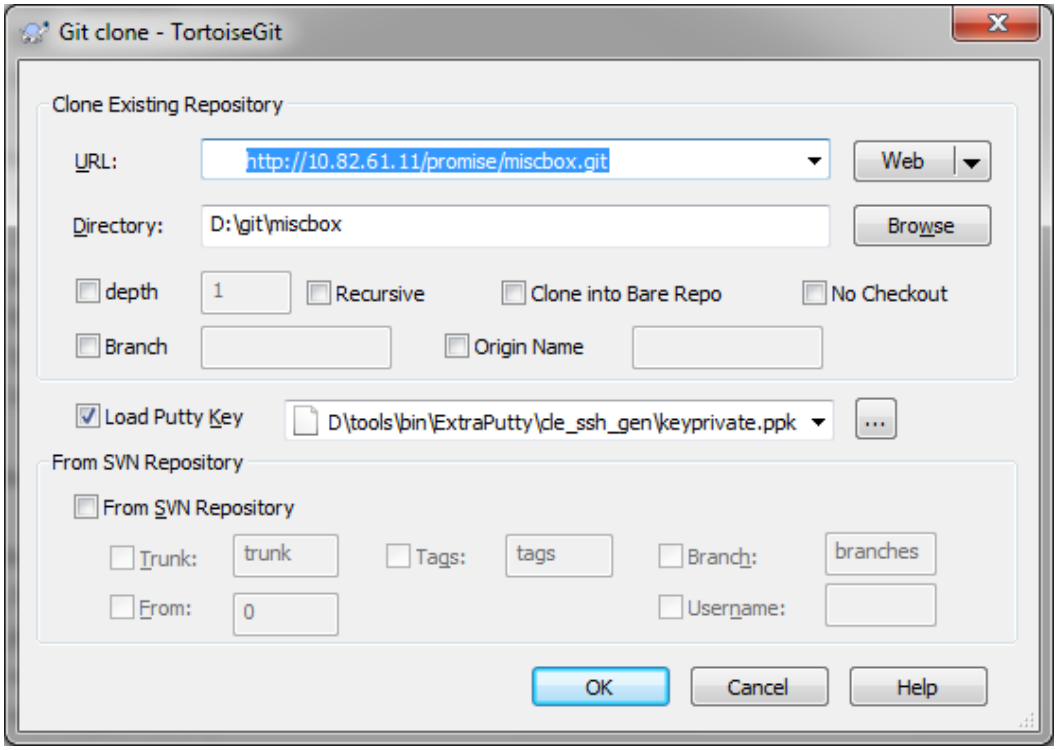


- **Utilisation de TortoiseGit**:
  - Cloner un repository:

Dans un explorer:



Récupérer l'URL du projet à cloner depuis le serveur web GIT :



- Ajouter/commit des fichiers:

Lors du premier commit, ouvrir une commande ms-dos. Se placer dans le répertoire contenant le dossir « .git » correspondant au projet. Taper la commande suivante : git push -u origin master Exemple :

```
D:\git\e-obs>git push -u origin master
Username for 'http://10.82.61.11': ****
Password for 'http://****@10.82.61.11':
Counting objects: 8, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 54.60 KiB | 0 bytes/s, done.
Total 8 (delta 0), reused 0 (delta 0)
To http://10.82.61.11/promise/e-obs.git
* [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

Dans l'explorateur windows, clique-droit sur le dossier contenant les fichiers à ajouter/commit :