Formation de formateurs
Simon Hay - Charles Tholliez
Novembre 2017

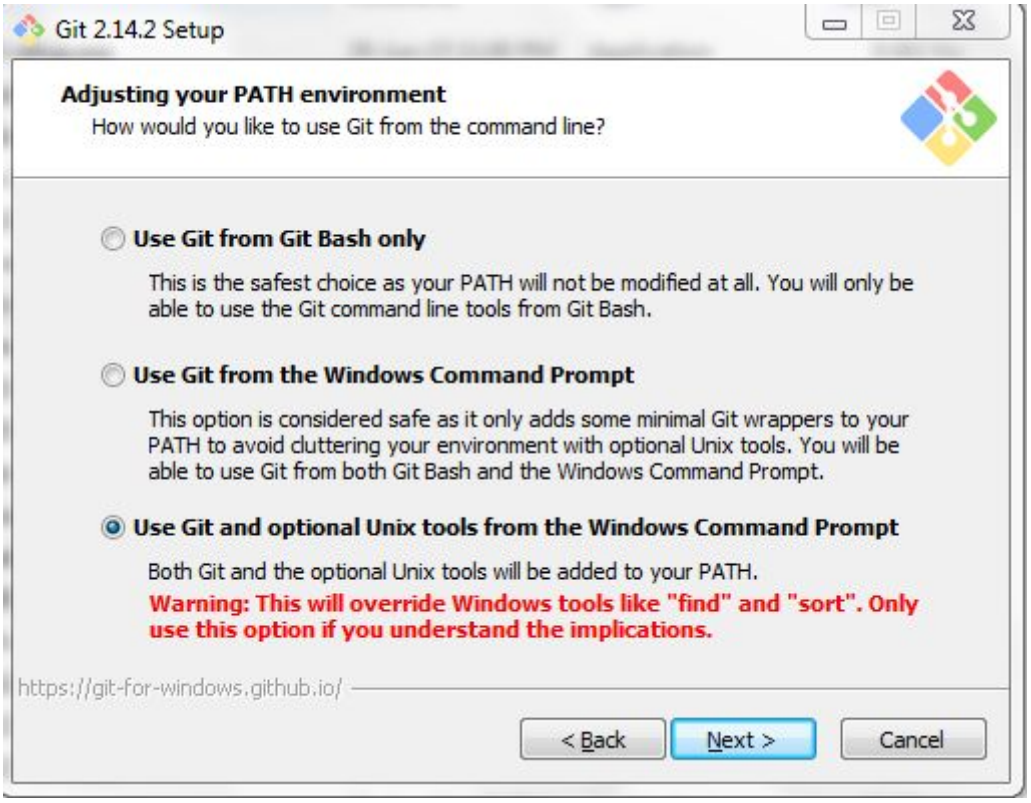
# Exercices Git

Présentation PowerPoint + fichier de commandes utiles au TP :
https://github.com/HellCarlito/Formation/tree/master/doc

# Pré-requis

## Installer Git

https://github.com/HellCarlito/Formation/blob/master/README.md
→ laisser options par défaut sauf pour :




Cela vous permettra d'avoir accès aux commandes Linux : start, ls, mkdir...

## Configurer Git

- Pour vérifier si Git est bien installé, ouvrez une interface de ligne de commande (GitBash que vous venez d'installer, par exemple) et faites la commande `git help` qui vous indiquera les commandes utiles pour manipuler Git.

- Pour une question d'ergonomie, entrez ces lignes qui activeront la coloration dans la console.

```
git config --global color.diff auto
git config --global color.status auto
git config --global color.branch auto
```

- Configurer nom/pseudo

```
git config --global user.name "votre_pseudo"
```

- Configurer email

```
git config --global user.email moi@email.com
```

- Pour voir la configuration global de votre git

```
git config --list
```

# Exercice 1 : les dépôts

## Création d'un dépôt en local

- Créez un nouveau répertoire sur votre machine et initialisez un dépôt Git à l'intérieur. Cela aura pour action de vous créer un sous-répertoire .git qui contient tous les fichiers nécessaires au dépôt (un squelette de dépôt Git).
- Créez un fichier README.txt dans votre dépôt.
- Faites un commit de ce nouveau fichier créé : vous avez fait votre première version de votre dépôt.
- Écrivez du texte à l'intérieur du fichier et faites un nouveau commit.
- Pour visualiser tout l'historique des modifications, vous pouvez utiliser la GUI de git. Ouvrez cette GUI depuis GitBash et allez voir l'historique de vos commits (Repository > Visualize master's History)

Remarque : Avant d'effectuer un commit, il faut indiquer à Git quels sont les fichiers à commettre. Pour cela, la commande `git add <nomFichier>` est utilisée.

## Création d'un dépôt sur GitHub

Pour pouvoir développer un projet informatique à plusieurs, il est possible d'utiliser une plateforme d'hébergement de code. GitHub est l'une d'entre elle.

- Créez un compte sur GitHub : https://github.com/
- Créez un nouveau répertoire sur GitHub et ajoutez-y le dépôt Git préalablement créé sur votre machine en suivant les instructions suivantes :
    - Ajoutez le dépôt GitHub au projet en local :

```
git remote add origin https://github.com/pseudoGitHub/nomDepotGitHub.git
```

    - Envoyer les élément du dépôt local sur le dépôt GitHub :

```
git push -u origin master
```

- Visualisez l'historique de vos commits depuis la plateforme GitHub.

## Clone d'un dépôt GitHub

Dans les étapes précédentes, vous avez dans un premier temps créé un dépôt git sur votre machine que vous avez ensuite envoyé sur un dépôt créé sur la plateforme GitHub. Vous pouvez également procéder de manière inverse, en créant d'abord un dépôt sur GitHub, que vous clonez en local sur votre machine.

- Clonez le dépôt GitHub préalablement créé sur votre machine.

# Exercice 2 : prise en main de Git

Dans cet exercice, vous allez devoir faire toutes les commandes nécessaires pour pouvoir créer correctement les versions de vos fichiers, en utilisant la plateforme GitHub.

1) Modifiez le fichier README.txt de votre dépôt et utilisez la commande `git diff`. Qu'observez-vous ?
2) Utilisez ensuite la commande `git status`. Qu'indique la commande ?
3) Indiquer à Git que vous souhaitez faire un commit sur le fichier README.txt.
4) Utilisez de nouveau la commande `git status`. Qu'indique-t-elle cette fois-ci ?
5) Enregistrez les modifications apportées à README.txt dans votre répertoire local.
6) Allez chercher les possibles modifications effectuées sur le dépôt présent sur GitHub. Expliquez le message indiqué par la console.
7) Mettez à jour le répertoire présent sur GitHub.
8) Ré-effectuez la commande `git status`. Qu'observez-vous ?

# Exercice 3 : travailler à plusieurs sur un projet

L'intérêt de travailler avec Git est de pouvoir développer un projet informatique à plusieurs de façon efficace.

1) Pour simuler cela, vous allez vous mettre en binôme et travailler sur le même répertoire GitHub. L'un des deux membres du binôme doit ajouter l'autre en tant que collaborateur sur son répertoire GitHub créé précédemment : sur la page d'accueil de votre répertoire, onglet `Settings` > `Collaborators`. Une fois cela fait, le membre du binôme venant d'être ajouté n'a plus qu'à cloner le projet sur sa machine.

*Membres du binôme : Etudiant1, Etudiant2*

2) **Chacun leur tour**, les deux membres du binôme doivent créer un fichier à leur nom `<Etudiant1/2>.txt` et faire les manipulations nécessaires pour envoyer leurs modifications sur le répertoire GitHub (cf. Uncle Sam). Que s'est-il passé au moment du "pull" effectué par l'Etudiant2 ?
3) L'Etudiant1 écrit sur la première ligne du fichier texte à son nom et enregistre ses modifications ("commit"). L'Etudiant2 écrit ensuite sur la même ligne de ce fichier, commit, pull et push ses modifications. L'Etudiant1 récupère les modifications effectuées sur le serveur (`git pull`). Qu'indique l'invite de commande ? Résolvez le problème et enregistrez vos modifications.

# Exercice 4 : commandes utiles

Le but de cet exercice est de vous faire découvrir quelques commandes utiles.

## Informations sur un commit

1) Voir tous les commits effectués via un `git log` (option `--oneline`)
2) Voir toutes les lignes modifiées via un `git log -p`
3) Voir un résumé des changements `git log --stat`

## Corriger les erreurs

1) Créer un nouveau fichier et ajoutez-y du texte. Faites votre commit avec un commentaire.
2) Ici vous allez modifier le texte du dernier commit via un `git commit --amend` car le message du dernier commit effectué n'était pas le bon. Corrigez donc votre dernier message.
3) Reproduisez l'étape 1 afin d'avoir de nombreux fichiers enregistrés en local.
4) Effectuez ensuite un `git reset HEAD`. Qu'observez-vous ? Pour les questions ci après pensez aux commandes utiles et celles vues précédemment.
5) Effectuez maintenant un `git reset HEAD~`. Qu'observez-vous ?
6) Effectuez maintenant un `git reset HEAD~~`. Qu'observez-vous ?
7) Effectuez maintenant un `git reset (suivi du numéro d'un commit)`. Que s'est-il passé ?
8) **BONUS : Effectuer un** `git reset --hard HEAD`**.** Que s'est-il passé ?
9) Faites des modifications sur votre projet et envoyez les sur le serveur GitHub.
10) Puis effectuez un `git revert HEAD`**.** Que s'est-il passé au sein de votre projet ? Attention avec cette commande !
11) **BONUS :** Faites de même avec `git revert HEAD~ et HEAD~~`

# Exercice 5 : travailler avec des branches

1) Faire un `git branch` pour voir de combien de branche est constitué votre projet.
2) Créer une branche : `git branch nom_branche`
3) Changer de branche : `git checkout nom_branche`
4) Faire de nouveau un `git branch`
5) Faire des modifications sur la nouvelle branche
6) Afficher les différences
7) Faire un commit
8) Revenir sur la branche master
9) Fusionner les changements via `git merge nom_branche_a_merger`
10) Vérifier que les ajouts/modifications apportées sur la branche ont bien été intégrés à la branche master.
11) Supprimer la branche `git branch nom_branche -d`

# Exercice 6 : mettre ses modifications de côté

1) Mettez vous dans le dossier racine de votre projet et modifiez un fichier. Créez une branche et placez vous à l'intérieur. Que se passe-t-il ?
2) Mettez ces modifications de côté comme indiqué dans le cours. Vérifiez que les modifications effectuées ont "disparu" du fichier modifié. Dans quel état se retrouve votre projet ?
3) Placez-vous à l'intérieur de la branche créée. Créez un fichier, modifiez des fichiers existants, commit, pull, push, etc.*
4) Revenez sur la branche master et restaurer les modifications mises de côté précédemment.

*Remarque : Pour "pull" et "push" à l'intérieur d'une branche, cf. la liste de commandes utiles, section "Branche".