Concepts Git

Git est un logiciel de gestion de versions (versionning), créé par Linus TORVALDS en 2005, initialement prévu pour le développement du noyau Linux.

Lancé en 2008, GitHub est un service web d'hébergement et de gestion de développement de logiciels, utilisant git.

Pour résumer : Diaporama Git

1. Repo/Repository

Un repository est un dépôt où un projet est hébergé, permettant de stocker les différentes versions du code.

2. Fork

Un “fork” est une copie d'un repository, permettant de travailler sur la copie qui devient un projet à part entière, au même titre que l'original. “Forker” un repository permet d'effectuer librement des changements sur 1 projet, sans affecter l'original.

La plupart du temps, les forks permettent d'effectuer et de proposer des changements à un projet original (dans ce cas il faudra faire une pull request pour solliciter une intégration des changements effectués), ou deviennent un nouveau point de départ pour un autre projet.

3. Clone

Le clone permet de créer une copie locale d'un projet forké ou créé, pour pouvoir ensuite travailler.

Réalisation d'un clone

```
git clone repositoryUrl
```

voir http://git-scm.com/docs/git-clone

4. Fetch

Le fetch permet de mettre à jour sa copie locale du projet à partir de la version hébergée sur le serveur et potentiellement modifiée par les autres membres de l'équipe. Le fetch est indispensable avant tout travail, pour éviter les futurs conflits. Il permet d'intégrer les commits réalisés par les autres dans sa copie locale.

Chacun des membres d'une équipe étant susceptible d’apporter des modifications au projet il est indispensable que tout le monde dispose en permanence du code le plus récent et demande fréquemment au dépôt principal s’il y a des mises à jours : c'est le rôle du fetch. Réalisation d'un fetch

```
git fetch repositoryName
```

voir http://git-scm.com/docs/git-fetch

5. Commit

Le commit permet de valider toutes les modifications effectuées sur un projet pour en créer une nouvelle version.

Le commit doit être accompagné (c'est indispensable) d'un message, composé d'un titre et d'une description, qui apporte des précisions sur les modifications opérées. Le message permettant aux autres développeurs de comprendre la nature des modifications, il est important de s'appliquer à le rédiger.

Comparez le message suivant :

```
some css
styling
ooops
misc fixes and cleanups
```

A celui-ci :

```
add subtle background pattern to body
make subheadings larger on archive pages
fix typo in site footer
cleanup code with htmltidy
```

L'un est clairement plus explicite et plus utile que l'autre...

Réalisation d'un commit

```
git commit -a -m "Message de validation du commit..."
```

voir http://git-scm.com/docs/git-commit

6. Branch

La branche par défaut d'un projet est la branche master, celle sur laquelle sont effectués les commits. Une branche correspond à un ensemble de commits consécutifs, permettant de faire avancer un projet.

Il est parfois indispensable de créer une nouvelle branche, par exemple pour ajouter une nouvelle fonctionnalité à un projet, ou pour résoudre un bug particulier.

Les branches permettent de séparer un projet, et de pouvoir travailler sur une nouvelle fonctionnalité (à risque) ou sur la résolution d'un bug, sans perturber et déstabiliser l'existant.

Une fois la fonctionnalité implémentée et testée ou le bug résolu, il sera alors possible de réunir la nouvelle branche avec la branche master par un merge (ou de précéder le merge par une Pull Request, pour faire une revue de code avant merge).

Création d'une branche

```
git branch branchName
```

voir http://git-scm.com/docs/git-branch

Positionnement sur une branche

```
git checkout branchName
```

voir http://git-scm.com/docs/git-checkout

Synchroniser une branche avec main (avant PR)

```
git checkout main
git pull
git checkout branchName
git merge main
```

Réalisation d'un merge de branchName avec master

```
git merge branchName
```

voir http://git-scm.com/docs/git-merge

7. Push

Le commit permet de créer une nouvelle version, qu'il faut ensuite envoyer vers le repository pour la rendre accessible aux autres développeurs, en faisant un push, vers une branche du projet (master par défaut).

Réalisation d'un push git push repositoryName master

8. Conflits

Identifier les conflits :

Identification de l'état

```
git status
```

voir http://git-scm.com/docs/git-status

- Les éventuels fichiers en conflit apparaissent comme unmerged
- Editez les fichiers en conflit, et effectuez les modifications nécessaires
- Valider ensuite les modifications :

Résolution de conflit

```
```

```
git add fileName
```

voir http://git-scm.com/docs/git-add

Effectuer le commit, puis le push.

9. .gitignore

Le fichier **.gitignore** permet de spécifier les fichiers, dossiers ou groupes de fichiers qui doivent être ignorés par git et non publiés (les fichiers de configuration, par exemple, qui peuvent contenir des infos sensibles, ou les fichiers spécifiques à la machine du développeur):

Exemple de fichier .gitignore

```
# Fichiers à ne pas synchroniser
 *.txt
 !readme.txt
 conf/
```

Signification

Exclusion de :

1. Tous les fichiers texte (*.txt)
2. Sauf le fichier readme.txt
3. et tout le dossier conf

Authentification

Elle se fait en mode console par user/email + token

Etape 1 Création token

- Générer un token sur votre compte Github dans les settings de votre compte : voir https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token
- Copier dans le presse papier le token généré
- Sur votre poste, configurer git :

Etape 2 Enregistrement côté client

```
git config --global user.name "myName"
git config --global user.email "myEmail"
git config --global credential.helper cache
```

Coller le token généré lors de la demande de mot de passe lors de la première opération git (push, pull...)
Dans putty, avec le bouton droit de la souris.

From:
http://slamwiki2.kobject.net/ - **SlamWiki 2.1**

Permanent link:
http://slamwiki2.kobject.net/cours/git

Last update: **2023/12/22 11:57**