Concepts Git

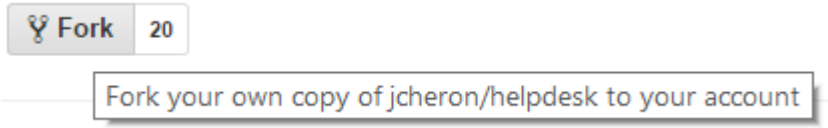
Git est un logiciel de gestion de versions (versionning), créé par Linus TORVALDS en 2005, initialement prévu pour le développement du noyau Linux.

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

-- Repo/Repository

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

-- 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.

-- Clone

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

```
git clone repositoryUrl
```

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

-- 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 succeptible d’apporter des modifications au projet il est indispensable que tout le monde dispose en permanence du code le plus récent et que tout le monde demande fréquemment au dépôt principal s’il y a des mises à jours : c'est le rôle du fetch

```
git fetch repositoryName
```

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

-- 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...

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

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

-- 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**.

```

```

```
git branch branchName
```

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

```
git checkout branchName
```

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

```
git merge branchName
```

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

-- 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).

```
git push repositoryName master
```

-- Conflits

Identifier les conflits :

```
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 :

```
git add fileName
```

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

-- .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 exmple, 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 :

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

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

Permanent link:
http://slamwiki2.kobject.net/slam5/collaboration/git?rev=1442448389

Last update: **2019/08/31 14:37**