Atelier de Développement: Contrôle de Version avec Git

Nicolas HERBAUT

Année 2018-2019

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Qu'est-ce que le contrôle de version?

Conservation du Code source

Le contrôle de versi cons permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications. Il permet notamment de retrouver les différentes versions du code source utilisées pour construire un projet

Collaboration

Le contrôle de version permet également à plusieurs développeurs de travailler simultanément sur le même code source et de synchroniser leurs efforts. Les logiciels permettent de régler les éventuels conflits liés à des modification concurrentes du code.

Gestion des fonctionnalités

A l'aide du système de branches, les logiciels de contrôles de version permettent de séparer les différentes fonctionnalités du logiciel lors de leur implémentation. Une fois l'implémentation achevée, les branches peuvent être consolidées pour créer une version contenant toutes les fonctionnalités.

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Contrôle de Version avec Git

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Caractéristiques de Git

- Crée par Linus Torvald pour gérer les versions du noyaux Linux
- Propose un modèle de version décentralisées
- chaque développeur poscède l'intégralité des versions sur son poste
- il synchronise son **dépot local** régulièrement avec ces autres collègues
- Github permet de partager plus facilement les dépôts entre collègues et fourni une interface graphique permettant

Commandes de configuration initiales

Vérification de la version

```
git --version
```

Identification du développeur

```
git config --global user.name "Nicolas Herbaut"
git config --global user.email "nicolas.herbaut@univ-paris1.fr"
```

Permet de tracer les modifications du code par le nom du développeur

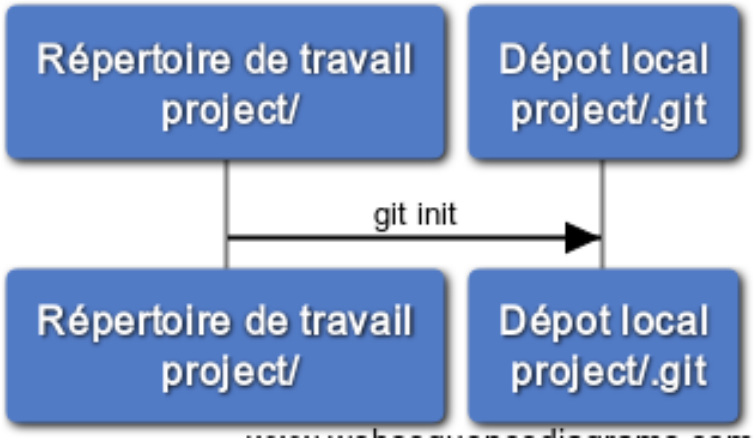
UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Obtenir de l'aide

```
git help config
```

Toutes les commandes de git sont documentées à l'aide de la ligne de commande. Par exemple pour config:

Initialisation d'un dépot local



- Ouvrez une invite de commande (Git Bash sous windows, un terminal sous Linux ou OSX)

création du dépôt

```
$ mkdir project
$ cd project
$ git init
Initialized empty Git repository in /home/user/project/.git/
```

Initialisation d'un dépôt local

git va créer un répertoire caché `.git` qui va stoker toutes les révisions du code et la configuration (pas besoin de rentrer dedans)

Que contient le répertoire git?

```
+-- .git
  +-- branches  # liste des branches locales
  +-- config    # configuration local du dépo
  +-- HEAD      # référence vers la branche actuelle
  +-- hooks     # scripts appelés lors de l'utilisation des commandes git
  +-- index     # reférence vers les fichiers ajoutés dans l'index
  +-- info      # diverses informations sur le repo
  +-- objects   # toutes les données
  +-- refs      # références vers les objets
```

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Premier Fichier

créons ensuite notre premier fichier et affichons le statuts de notre repo

Inspection de notre premier dépôt

```
$ echo "hello git!" > file.txt
$ git status
$ git status
On branch master
No commits yet
nothing added to commit but untracked files
present (use "git add" to track)
```

astuce windows

Pour ouvrir l'explorateur de fichier sous Windows dans la fenêtre de GitBash, tapez

```
explorer .
```

Ignorer certains fichiers

- Le fichier caché .gitignore, positionné à la racine du dépôt, permet à git d'ignorer certains types de fichiers ou de répertoires. Vous pouvez
- l'utiliser pour exclure automatiquement les fichiers compilés (.class .jar .o) de votre dépôt.

ajout de règle dans .gitignore

```
$ touch toto.class
$ echo ".class" >> .gitignore
$ echo ".jar" >> .gitignore
```

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Workflow git

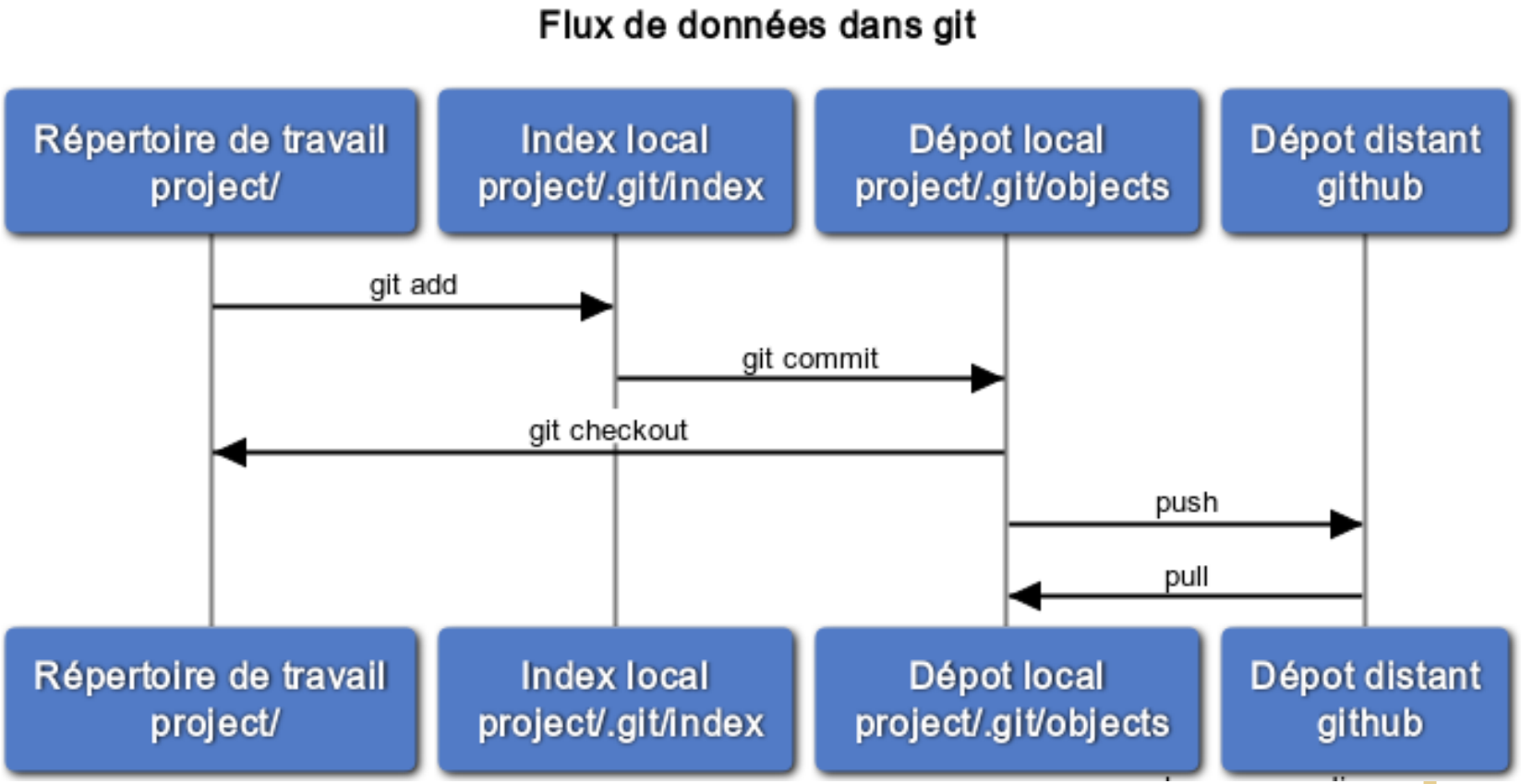
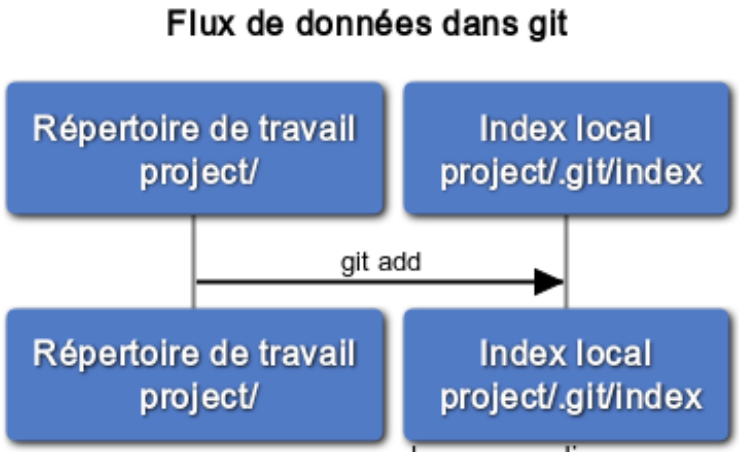


FIGURE 1 – Worflow dans Git

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Ajout de fichier dans l'index

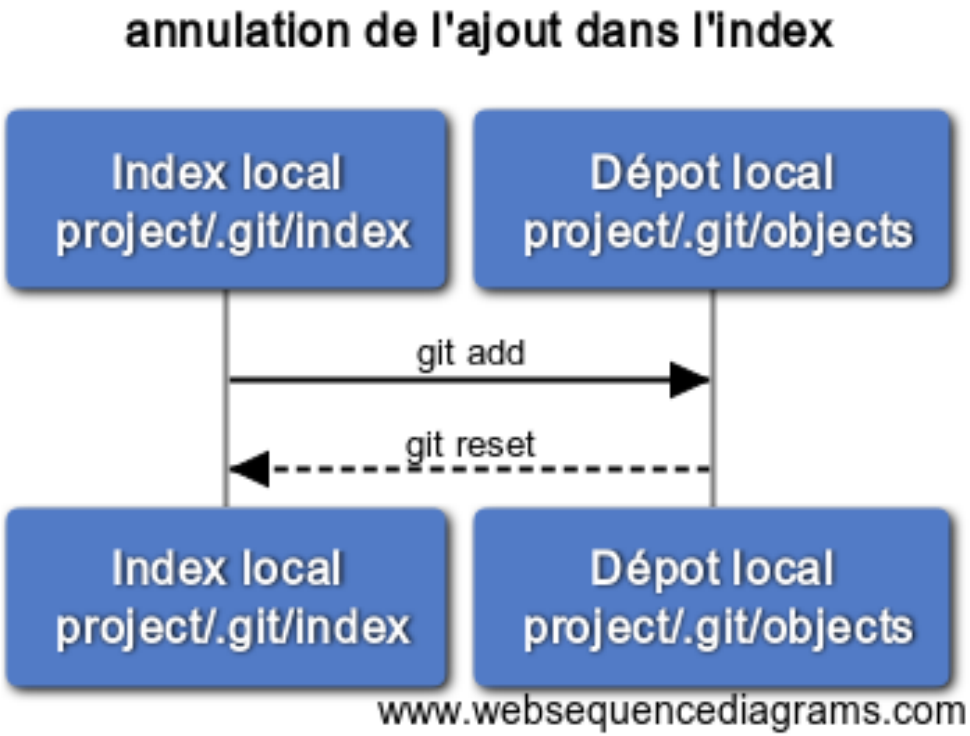


Ajout de fichier dans l'index

```
$ git add -A
$ git status
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   .gitignore
    new file:   file.txt
```

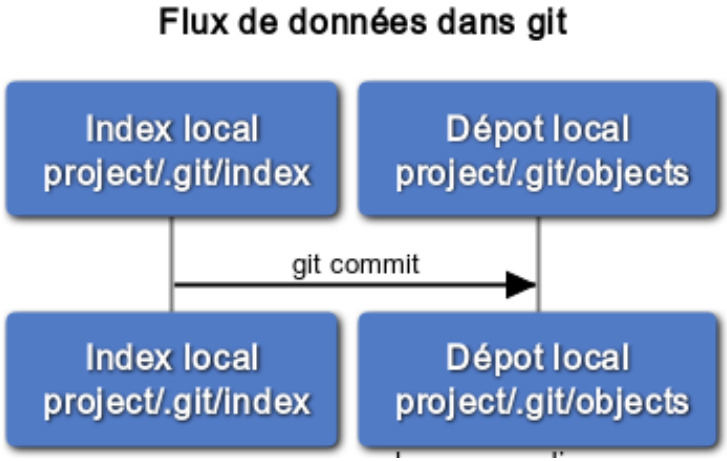
Annuler l'ajout dans l'index



annuler un git added

```
$ git reset
```

ajouter des fichiers dans le dépo (commit)

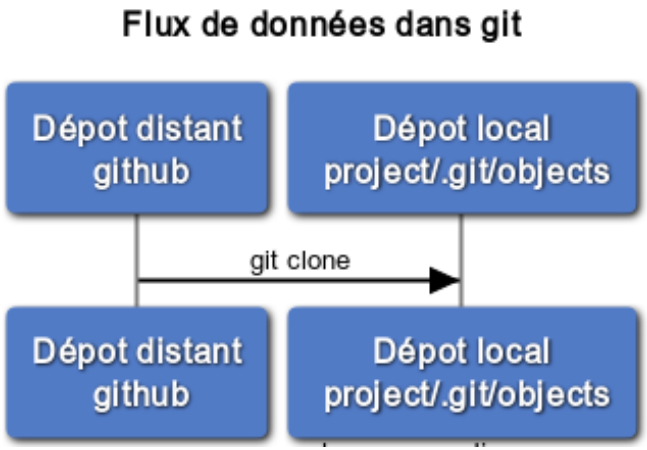


Ajouter un fichier dans le dépôt local

```
$ git commit -m "commit initial"
$ git log
commit a314b3a0fd78ca06a01dce11f179c426aaa795fa (HEAD -> master)
Author: Nicolas Herbaut <nicolas.herbaut@univ-paris1.fr>
Date:   Sat Nov 24 02:20:53 2018 +0100

    commit initial
```

Récupération d'un dépôt distant et synchronisation



- Les dépôts sont définis par des adresses (e.g. git@github.com:UFR27/miage-diploma-students.git)
- récupérer toutes les données d'un dépôt est appelé "cloner le dépôt"
- ici, un nouveau répertoire diploma-app est crée permettant de récupérer tous les commits existants

clone d'un dépôt distant

```
$ cd
$ git clone git@github.com:UFR27/miage-diploma-students.git
```

modifier un dépôt local

- Si on modifie un fichier, la modification n'impacte que le dépôt local

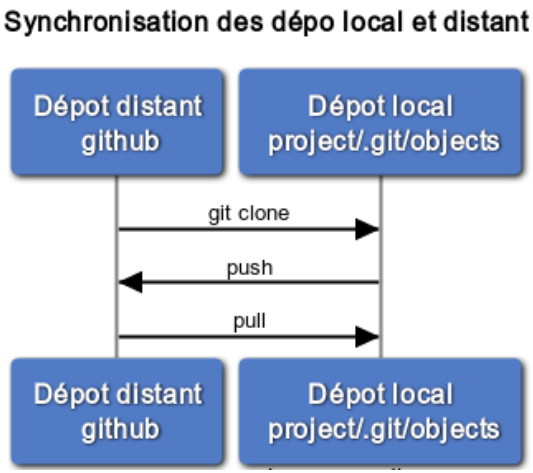
Modification locale

```
$ echo "salut" > README.md
$ git add README.md
$ git commit -m "mise à jout de readme"
```

Différent du répo distant

```
$ git fetch # récupère les données du dépôt distant
$ git diff master origin/master --name-status
M README.md
```

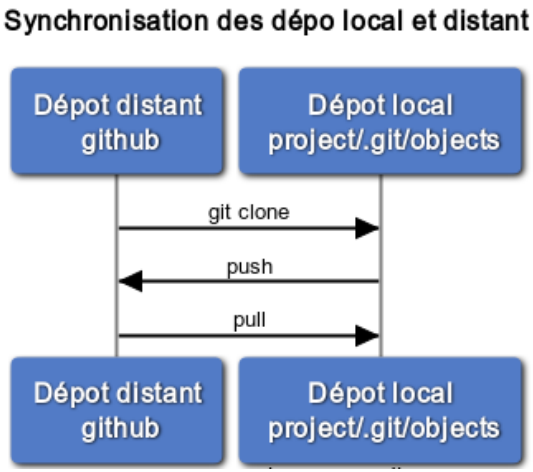
Synchronisation dépôt local/distant



- Pour publier notre code, nous devons synchroniser le dépôt local avec le dépôt distant
- la synchronisation se fait sur la branche par défaut **master**
- on peut se synchroniser avec le dépôt distant à l'aide des commandes **pull** et **push**

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Synchronisation dépôt local/distant



synchronisation

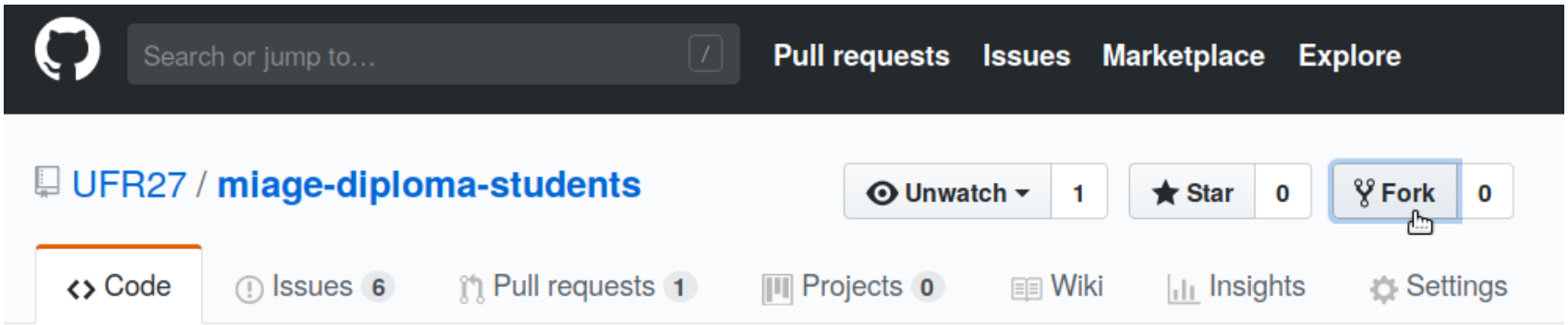
```
$ git pull origin master
$ git push origin master
Counting objects: 2, done.
Writing objects: 100% (2/2), 270 bytes | 270.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
To github.com:UFR27/diploma-app.git
   d4aceb9..5fbeebe  master -> master
```

Droits d'écriture

- Pour pouvoir écrire (push) dans un dépot distant, il faut avoir le droit d'écrire.
- Or nous n'avons pas le droit d'écriture sur le depo git@github.com:UFR27/miage-diploma-students.git.
- Nous devons donc créer un dépôt où nous avons les droits à partir de notre compte github.
- Nous allons créer une copie du dépôt original appelée un **fork**
- Le fork permettra de re-synchroniser nos modification avec le dépôt original (**contribution**)

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

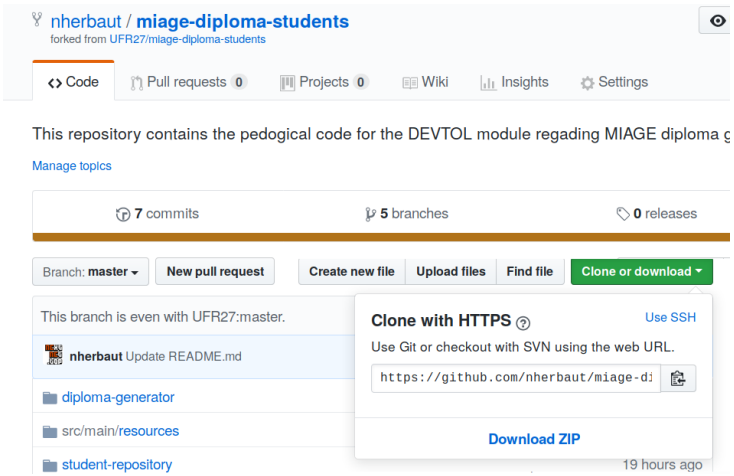
Création d'un fork dans Github



UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Clone de notre fork

- il faut récupérer l'adresse de notre fork



UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Les branches dans git

- Les branches permettent d'organiser nos différents Développements
- un commit est fait sur un branche
- e.g. je souhaite développer une fonctionnalités sans impacter mes camarades.
- en pratique: 1 fonctionnalité = 1 branche
- en pratique: 1 version du projet = 1 branch release avec une autre branche fusionnée.

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Git tag

Il est possible de “tagger” (marquer) une version pour faciliter l'Identification

```
$ git tag v1
```

Référence sur les branches

git branch & git checkout

```
$ git branch my_feature # crée la branche my_feature
$ git checkout my_feature  # se rendre sur la branch my_feature

$ git branch checkout -b my_feature # crée la branche + change de branche
```

git merge

```
$ git checkout master
$ git merge my_feature
```

Les merges peuvent entraîner des conflits en cas de modifications simultanées. Pour résoudre un conflit, il suffit d'éditer le fichier à la main + commits

Manipulation pratique

Suivre les 8 premières leçons de ce tutorial: git branches

Objectif à atteindre