Partie 1 : Introduction à `git`

Comment collaborer et versionner correctement son projet

Jérôme Buisine (PhD Student)
Team: IMAP (Images et Apprentissage)

3 septembre 2021

Sommaire

Gestion de version

Un problème commun ?

Trouver un nommage pour versionner un document n'est pas évident :

```
Mémoire_v1.doc
Mémoire_v2.doc
Mémoire_v3.doc
Mémoire_Final.doc
Mémoire_Final_v2.doc
Mémoire_Final_Final.doc
...
Mémoire_Final_pour_de_vrai.doc
```

Parlons convention

Une **convention** dans un premier temps ?

```
Mémoire_v0.0.1.doc
Mémoire_v0.0.2.doc
Mémoire_v0.1.0.doc
Mémoire_v0.2.0.doc
Mémoire_v1.0.0.doc
Mémoire_v1.0.1.doc
...
Mémoire_v2.0.0.doc
```

Problème de collaboration

Comment gérer le maintien du document avec plusieurs personnes ?

Problème de collaboration

Comment gérer le maintien du document avec plusieurs personnes ?

- Qui travaille actuellement et sur quelle version ?
- Quelle est réellement la dernière version ?

Problème de collaboration

Comment gérer le maintien du document avec plusieurs personnes ?

- Qui travaille actuellement et sur quelle version ?
- Quelle est réellement la dernière version ?

Figure 1 – Visage de la personne désignée volontaire pour la fusion du document

Problème de collaboration

Comment gérer le maintien du document avec plusieurs personnes ?

- Qui travaille actuellement et sur quelle version ?
- Quelle est réellement la dernière version ?

Figure 1 – Visage de la personne désignée volontaire pour la fusion du document

Édition de document en ligne

Outils **collaboratifs** permettant d'éditer en ligne et en parallèle le même fichier aisément

Vers des outils collaboratifs

Outils collaboratifs

Pour les développeurs :

(a) Fondation Apache en 2004

(b) Linus Thorvald en 2008

L'outil collaboratif `git`

Particularités :

- Système de contrôle de version décentralisé
- Chaque participant possède un clone de l'ensemble du référentiel en local
- Ajouter, contribuer et suivre les changements dans le code source

L'outil collaboratif `git`

Particularités :

- Système de contrôle de version décentralisé
- Chaque participant possède un clone de l'ensemble du référentiel en local
- Ajouter, contribuer et suivre les changements dans le code source

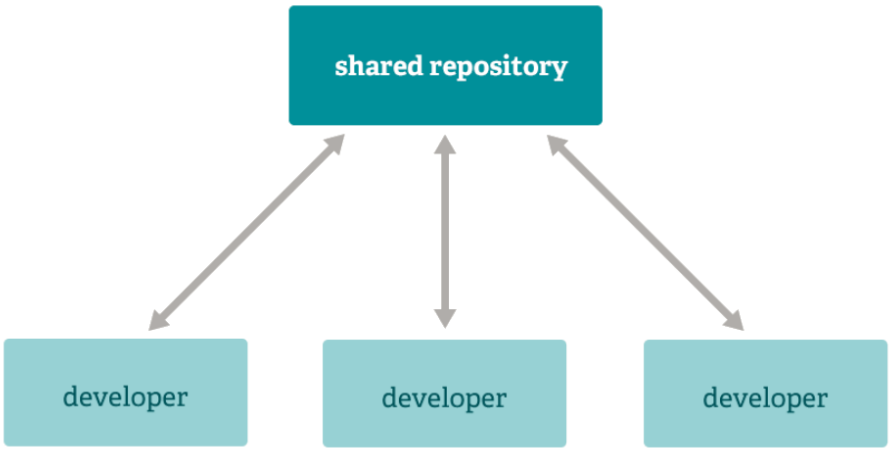


Figure 3 – Échanges bilatéraux de développeurs (source : git-scm).

Utilisation basique de `git`

Interface de commandes

Commandes git de base :

```
# Initialisation du projet
git init

# Ajout de fichier dans l'index
git add file.txt

# Vérification de l'index
git status

# Suppression d'un fichier dans l'index
git reset file.txt
```

Loading video...

Ajout de modifications

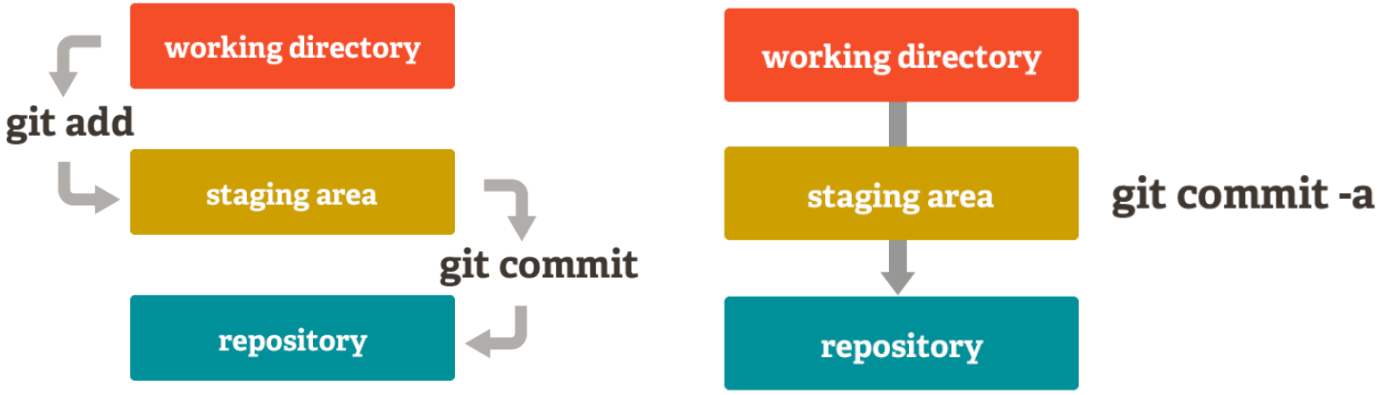


Figure 4 – Ajout et validation de contenu de fichiers (source : git-scm)

Gestion des modifications

Ajout de contenu :

```
# Validation de l'index (fichiers ajoutés)
git commit -m "some updates"

# Ajout des modifications courantes et validation
git commit -am "some updates"

# Aperçu des commits
git log

# Affichage condensé des commits
git log --oneline
```

Suppression de modifications

Suppressions locales :

```
# Supprimer le dernier commit local
git reset --soft HEAD^

# Supprimer les 2 derniers commits locaux
git reset --soft HEAD~2
```

Loading video...

Notions importantes

Version de projet

Numéros de version sous la forme `X.Y.Z`

- X – Majeur : suppression d'une fonctionnalité obsolète, modification d'interfaces, renommages...
- Y – Mineur : introduction de nouvelles fonctionnalités, fonctionnalité marquée comme obsolète...
- Z – Correctif : modification/correction d'un comportement interne, failles de sécurité...

Ajout de version

Réaliser une livraison avec version :

```
# Ajout d'un tag de version à l'état actuel du projet
git tag -a v1.0.0 -m "Releasing version v1.0.0"

# Liste l'ensemble des versions du projet
git tag -l

# Affiche les informations détaillées d'un tag
git show v1.0.0
```

Gestion des branches

Vers l'utilisation de branches ? ?

Gestion des branches

Vers l'utilisation de branches ? ?

Loading video...

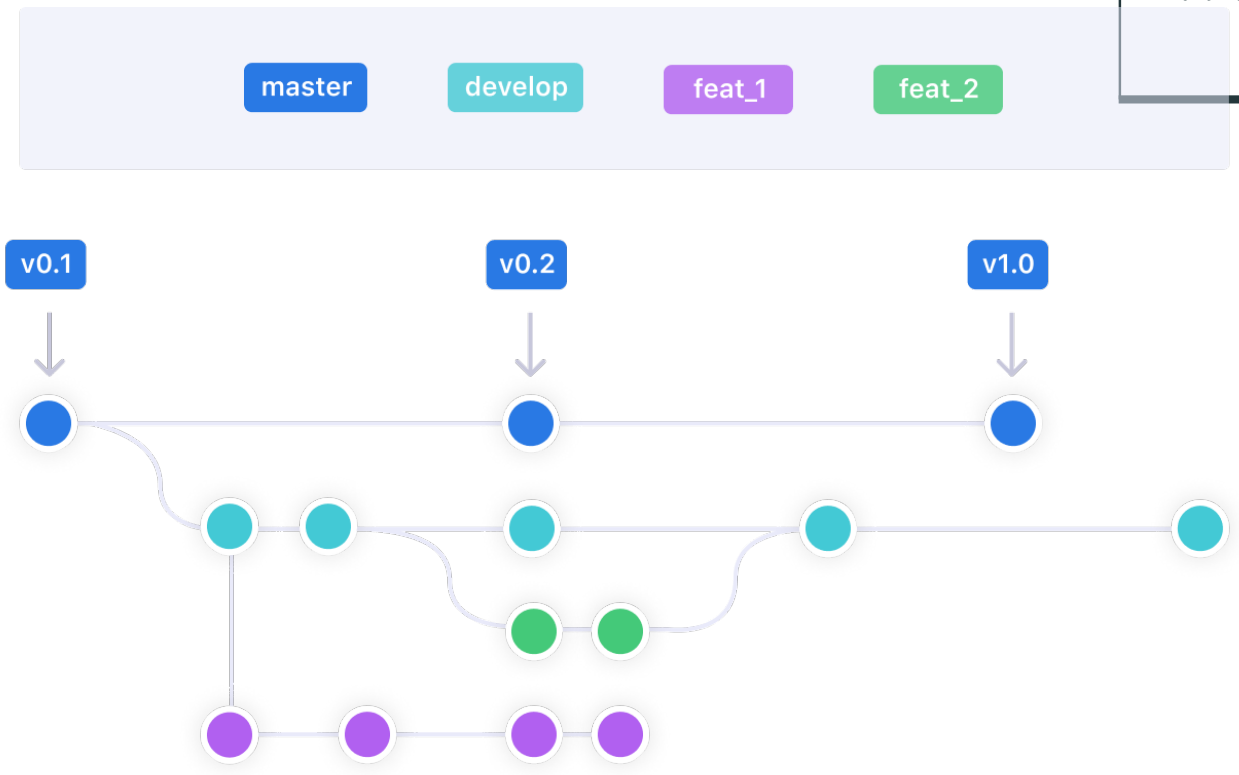


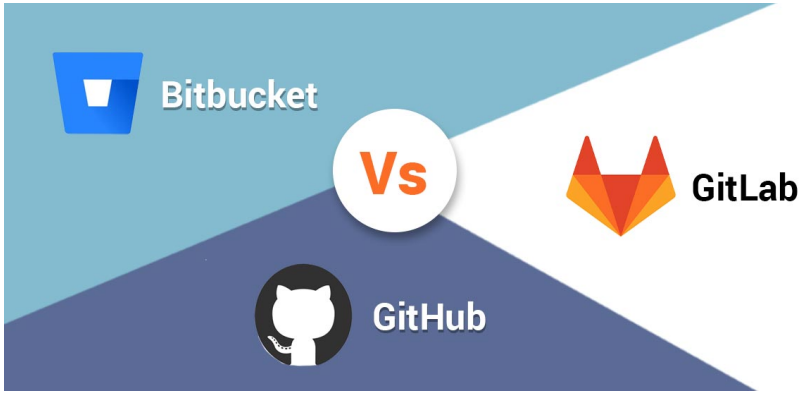
Figure 5 – Exemple de gestion des branches sous `git` (source : qovery.com).

Gestion des branches

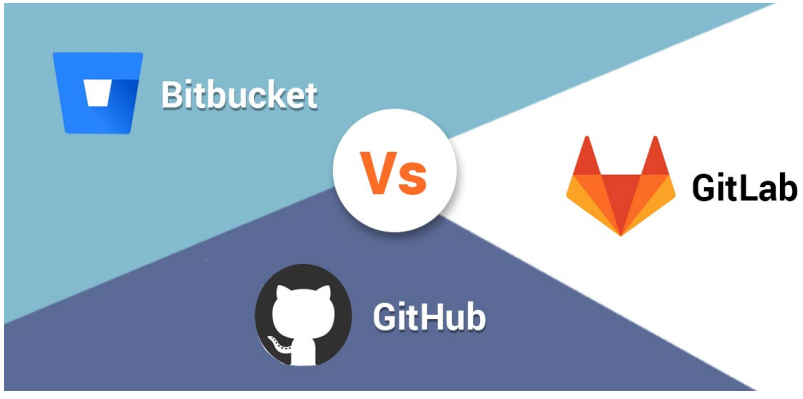
Commandes pour la gestion des branches :

```
# Liste toutes les branches disponibles
git branch

# Crée une nouvelle branche
git branch ma-branche

# Change de branche de développement
git checkout ma-branche

# Création et changement de branche courante
git checkout -b ma-branche

# Supprime une branche
git branch -d ma-branche

# Depuis la branche ‘develop‘, récupération des
    modifications de ‘ma-branche‘
# Plus évident à manipuler que rebase
git merge ma-branche
```

Serveur d'hébergement



Serveur d'hébergement



Interaction avec le serveur d'hébergement :

```
# Énumère les serveurs d'hébergements référencés
git remote -v

# Récupère et applique les modifications du serveur
    d'hébergement
git pull <remote-name> ma-branche

# Publie les modifications apportés par une branche sur un
    serveur
# !! toujours réaliser un pull avant de push !!
git push <remote-name> ma-branche
```

Autres commandes

Commandes à ne pas confondre :

- **git revert** : crée un nouveau commit qui annule les changements d'un commit précédent ;
- **git checkout** : extrait le contenu du référentiel et le place dans votre arbre de travail (elle permet aussi le déplacement vers une autre branche) ;
- **git reset** : il modifie l'index (la ≪ zone de transit ≫). Ou elle change le commit sur lequel une tête de branche pointe actuellement. Cette commande peut modifier l'historique existant.

Autres commandes

Cas d'utilisations possibles :

- **git revert** : si un commit a été fait quelque part dans l'historique du projet, et que vous décidez plus tard que ce commit est mauvais. L'utilisation de *git revert* annulera les changements introduits par le mauvais commit, en enregistrant le « undo » dans l'historique ;
- **git checkout** : restaure une révision historique d'un fichier donné (`git checkout <file-path> <commit-hash>`) ;
- **git reset** : si vous avez fait un commit, mais que vous ne l'avez pas partagé avec quelqu'un d'autre et que vous décidez que vous n'en voulez pas, alors vous pouvez utiliser git reset pour réécrire l'historique de sorte qu'il semble que vous n'ayez jamais fait ce commit.