Le travaille en équipe

Romain THERRAT

POCKOST

14 décembre 2020

Sommaire

1 Présentation

2 Tirer et pousser

3 Authentification

4 Services en ligne

5 Les conflits

6 Labs

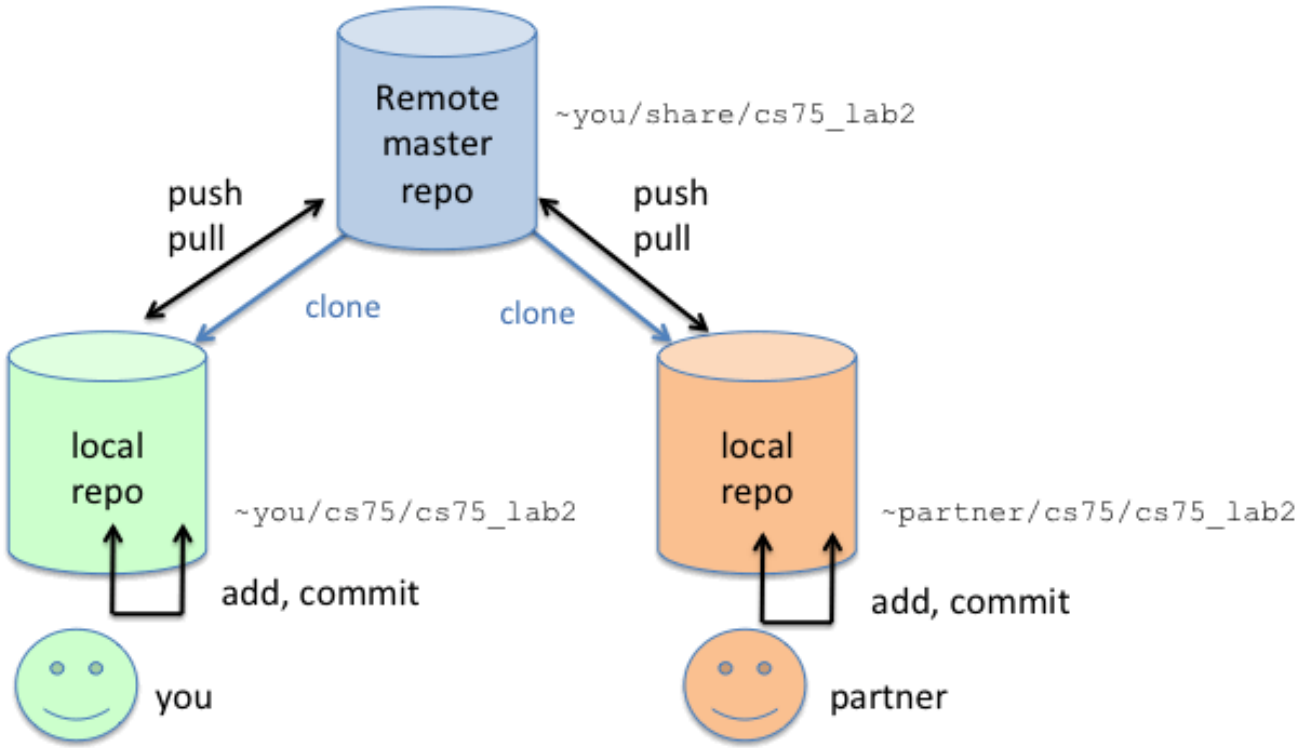
A quel problème on répond

Actuellement on ne travaille qu'en local

- Comment partager les données
- Serveur centrale ?
- Risque de conflit

Nous utiliserons donc les `remote`.

Les remote


Remote
master
repo
~you/share/cs75_lab2
push
pull
push
pull
clone
clone
local
repo
~you/cs75/cs75_lab2
local
repo
~partner/cs75/cs75_lab2
add, commit
add, commit
you
partner

git remote

```
$ git remote add origin <url>
$ git remote rm origin <url>
$ git remote show origin
$ git remote set-url origin git@github.com:myteam/myproj
```

Le nom du `remote` par défaut est `master` mais vous pouvez en avoir plusieurs

Sommaire

1 Présentation

2 Tirer et pousser

3 Authentification

4 Services en ligne

5 Les conflits

6 Labs

push

Un `push` permet de pousser ses modifications sur un serveur git distant.

- Envoyer ses modifications
- Sur une branche en particulier
- `git push <remote> <branch>`

Par exemple `git push origin master` Si vous avez fait des modifications du passé (fast foward) il faudra utiliser l'option `-f`

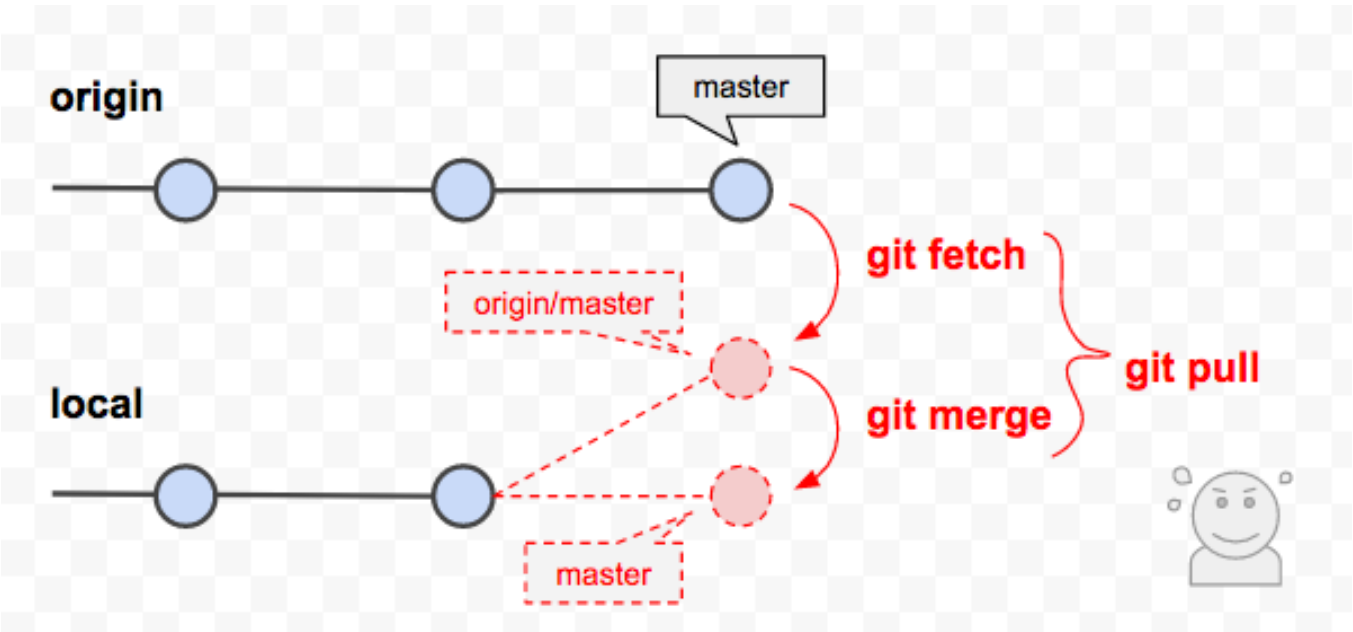
git pull

Contrairement au `push`, le `pull` permet de tirer (récupérer) une modification et les appliquer localement.

- `git pull <remote> <branch>`
- Récupérer les modifications et les appliquer
- Équivalent à
- `git fetch` et `git merge`
- `git fetch` = Récupérer les modifications distantes (Nommé `remote/banch`, exemple : `origin/master`)
- `git merge` = Appliquer les nouvelles modifications en local

Par exemple `git pull origin master`

pull


origin
master
git fetch
origin/master
git pull
local
git merge
master

Sommaire

1 Présentation

2 Tirer et pousser

3 Authentification

4 Services en ligne

5 Les conflits

6 Labs

Les modes de connexion

Plusieurs modes de connexion à un serveur git sont possibles

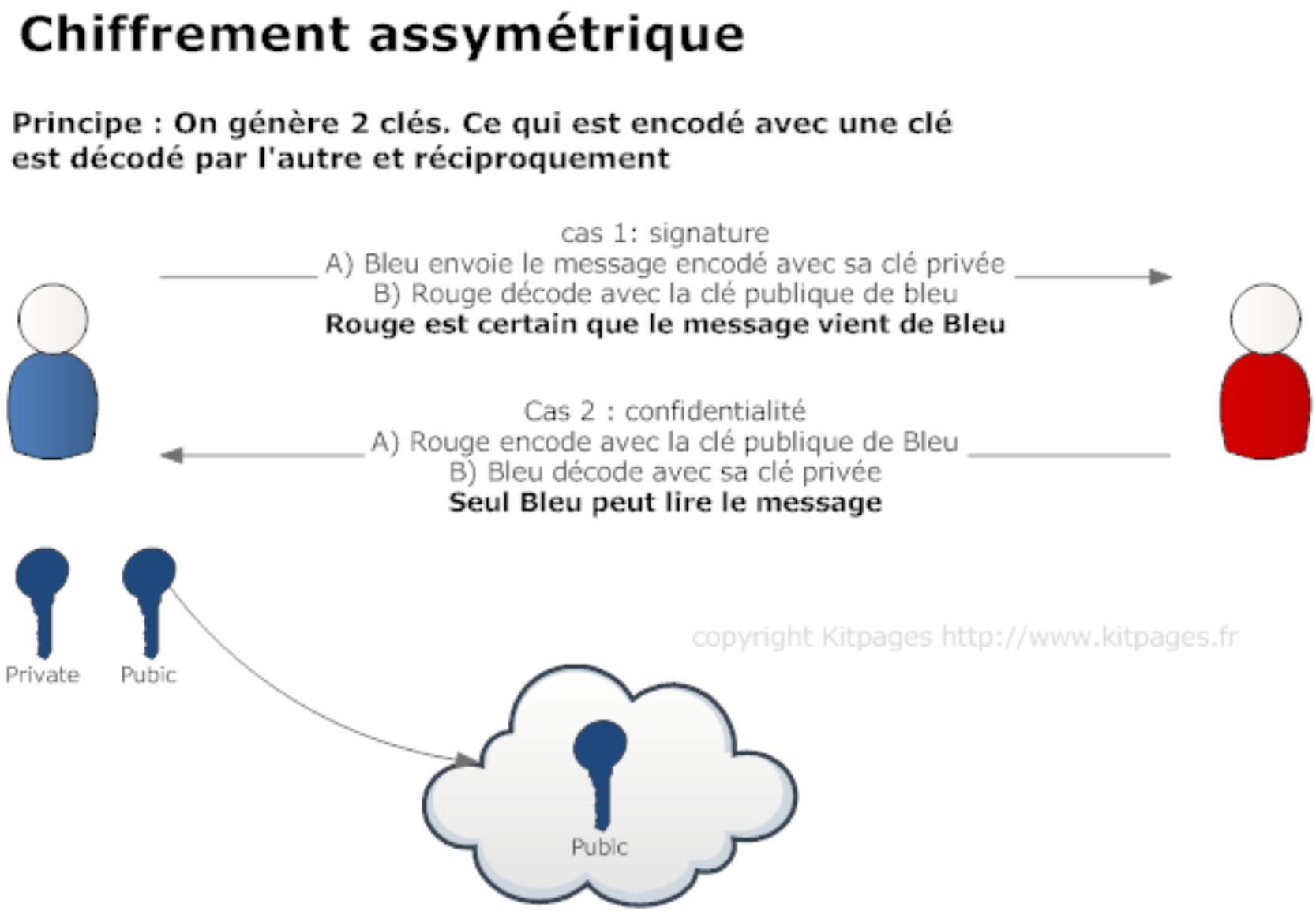
- HTTPS
  - Simple
  - Pas de configuration spécifique
  - Auth HTTP username/password
- SSH
  - Utilise les certificats SSH
  - Doivent être générés via `ssh-keygen`
  - Copier sa clé PUBLIQUE dans GitHub/GitLab
- Démonstration

Le chiffrement symétrique

Un message chiffré avec la clé est déchiffré avec la même clé.

Le problème : comment transmettre la clé de façon sécurisée ?

Le chiffrement asymétrique



Les certificats SSH

Les clés privées doivent . . . rester privées ! !

Sommaire

1 Présentation

2 Tirer et pousser

3 Authentification

4 Services en ligne

5 Les conflits

6 Labs

Service en lignes

- Nombreux acteurs
  - GitHub
  - GitLab
  - Bitbucket
  - ...

Historique-GitHub

- GitHub
- Créé en 2008
- Service en ligne (SaaS)
- Gratuit pour les dépôts publics
- Des options payantes
- Rachat par Microsoft 2018

Le succès

- Public first = Favorise l'OpenSource
- Les concurents à l'époque
  - Google source
  - SourceForge
- Une interface simple et lisible
- Le fork and merge multi projet (wait and see)
- Adopté par les principaux projets Opensource
  - Symfony
  - Twitter Bootstrap
  - JQuery
  - ...

Créer un dépôt

Utilisation de base

- Après avoir créé un compte
- Cliquer sur “New Repository”
- Choisir un nom et une description
- Équivalent à un git init sur un remote distant
- Démonstration

Pusher sur mon dépôt

- Si j'ai déjà un dépôt avec des commits
  - `$ git remote add origin <url>`
  - `$ git push origin master`
- Si je n'ai pas encore de source
  - `git clone <url>`

Les forks

- Copie de l'arbre git d'un dépôt sous un autre nom
- Cycle de vie indépendant
- Projet le plus forké ?
  `https://github.com/jtleek/datasharing`
- Vrais projets les plus forkés
  - TensorFlow
  - Twitter Bootstrap
- `https://bit.ly/2M3dw6g`

Les forks

Exemple d'un dépôt forké

`https://github.com/raphapr/ansible-statuscake/network`

Les pull requests

- Gestion des issues
- Proposition de solutions
- Peut être automatiquement intégrée
- Comme un `merge` inter-repository
- Favorise la collaboration

Les pull requests

Démonstration

Sommaire

1 Présentation

2 Tirer et pousser

3 Authentification

4 Services en ligne

5 Les conflits

6 Labs

Les conflits

Qu'est-ce qu'un conflit et pourquoi arrivent ils ?

- Travail à plusieurs sur le même fichier
- Merge de deux branches éditant le même fichier
- ...
- Que devons-nous garder ?

```
Les conflits sont courant et "normaux".
```

Les conflits

Le conflit peut se produire de deux méthodes

- J'ai des modifications locales non commitées
- J'ai des commits qui modifient le même fichiers

Conflit sur les modifications locales

- Le cas le plus simple
- Mettre les modifications en attente
- `git stash` et `git stash pop`
- Pas besoin de créer un commit de merge
- L'arbre reste propre
- Peut être automatiquement géré par git (modifications simples)

Conflit sur les modifications locales

```
$ git merge create-conflict
Updating 9ac6888..7c498aa
error: Your local changes to the following files would be ov
        fichier1.txt
Please commit your changes or stash them before you merge.
Aborting
$ git stash
Saved working directory and index state WIP on master: 9ac68
$ git merge create-conflict
Updating 9ac6888..7c498aa
$ git stash pop
Auto-merging fichier1.txt
CONFLICT (content): Merge conflict in fichier1.txt
```

Conflit sur les modifications locales

```
$ git status
On branch master
Unmerged paths:
  (use "git reset HEAD <file>..." to unstage)
  (use "git add <file>..." to mark resolution)
      both modified:   fichier1.txt
...
$ git diff fichier1.txt
...
++<<<<<<< Updated upstream
+Je cherche à créer un conflit
++=======
+ J'ai une modif ici aussi
++>>>>>>> Stashed changes
```

Conflit sur les modifications commitées

- Création d'un commit de merge
- Peut être automatiquement géré par git
- Faire les modifications nécessaires à la main
- Ajouter modifications : `git add`
- Commiter le résultat : `git commit`
- Possibilité d'annuler le merge : `git merge --abort`
- Pour se simplifier la vie il est possible d'utiliser l'option `--merge` à `git log`

Conflit sur les modifications commitées

```
$ git merge create-conflict
Auto-merging fichier1.txt
CONFLICT (content): Merge conflict in fichier1.txt
Automatic merge failed; fix conflicts and then commit the re
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   fichier1.txt
```

Sommaire

1 Présentation

2 Tirer et pousser

3 Authentification

4 Services en ligne

5 Les conflits

6 Labs

Labs

Dans ce Labs nous allons chercher à utiliser un serveur `git` central.

- Cloner un dépôt `https://github.com/nicholaskajoh/faceclone-html-template`
- Déplacer vous dans ce dossier
- Éditer le fichier `index.html` pour modifier le titre de la page en “MyFaceClone”
- Commiter la modification
- Réaliser un `git push origin master` et constater que ceci n’est pas possible

Labs

Nous allons maintenant travailler sur un serveur git où nous pourrons réaliser des `push`. Nous créerons aussi un couple de clés pour l'authentification git.

- Connecter vous à `git.pockost.com` avec les identifiants fournis par votre formateur
- Créer un dépôt vide sur l'interface web
- Dans `git bash` entrer la commande `ssh-keygen` pour générer une paire de clé
- Récupérer le contenu de la clé publique pour l'ajouter dans votre compte GitLab
- Initialiser un dépôt vide en local et commiter un fichier
- A l'aide de `git remote` ajouter l'URL de dépôt créé précédemment (git@git.pockost.com)
- Pousser vos modifications `git push`

Labs

Après s'être familiarisé avec la notion de `push` et `clone` nous allons traiter les cas des `pull` des conflits et `pull request`

- Créé un nouveau dépôt vide nommé `FaceClone` sous GitLab
- Ajouter un second `remote` dans le projet `FaceClone` qui a été cloné dans la première partie du Labs. Vous pouvez le nommer `gitlab` par exemple (`git remote add <name> <url>`).
- Pousser les sources de `FaceClone` dans ce `remote`
- Constater sur l'interface de `GitLab` que tous les commits de `GitHub` sont bien présents.
- Créer une branche nommée `new-feature` et commiter une modification sur le fichier `index.html`
- Pousser la modification sur GitLab
- Dans l'interface Web de GitLab créer une nouvelle Merge Request vers la branche `master`
- Valider la
- En local retournez sur la branche `master`, récupérer les modifications distantes et observez les commits (`git log`)

Labs

Sur la même base précédente nous allons volontairement généré un conflit

- En local créer une branche `feature42` basée sur `master`
- Modifier le fichier `index.html` pour modifier le titre en `FaceClone42`
- Commiter cette modification
- Créer une branche `feature43` basée sur `master`
- Modifier le fichier `index.html` pour modifier le titre en `FaceClone43`
- Commiter cette modification
- Merger la branche `feature42` dans la branche master
- Merger la branche `feature43` dans la branche master
- Constater le conflit et modifier le titre `FaceClone42 et 43`
- Commiter la correction

Labs

Si vous souhaitez aller plus loin vous pouvez tester les méthodes de partage de dépôts sur GitLab.

A vous les studios !