Git...

... et utilisation de GitLab et GitLab CI

D. Arrivault

Laboratoire d'Excellence Archimède
Aix Marseille Université

05 Novembre 2015 / LIFTech' 1ère session

Sommaire

Les systèmes de gestion de versions.

Problématiques.

Contexte.

- Une personne seule développe un logiciel (ou article...) :
  - conserver son historique.
- Une équipe développe un logiciel :
  - partager les modifications.
- Un projet collaboratif avec plusieurs personnes travaillant sur un logiciel distribué :
  - maintenir plusieurs versions ;
  - corriger les bugs et ajouter des fonctionnalités ;
  - gérer les conflits.

Problématiques.

Objectifs.

- Garder un historique :
  - pour retourner à une version précédente ;
  - pour voir d'où vient une ligne ;
  - pour trouver qui a cassé une fonctionnalité.
- Travailler en équipe :
  - pour gérer les travaux concurrents des différents collaborateurs.

Existant.

Beaucoup de solutions.

- Centralisées : CVS (1990), SubVersioN (2000)...
- Décentralisées : BitKeeper (2000), Darcs (2002), Mercurial (2005), Git (2005), Bazaar (2008)...

Existant.

Avantages du gestionnaire décentralisé.

- Accès à tout l'historique localement.
- Commits locaux intermédiaires. Favorise la pratique des petits commits indépendants.
- Robuste à la corruption des dépôts.
- Possibilité de créer pleins de branches locales.
- Maintient d'un dépôt public qui compile/marche.


git

Historique.

- 2002 : Linux passe à BitKeeper.
- 2005/04 : BitMover arrête la licence libre de BitKeeper
- Linus Torvalds débute le développement de GIT
- 2005/06 : Linux passe à GIT
- 2007/02 : GIT 1.5 releasé

Difficultés.

> *[Linus] is a guy who delights being cruel to people. His latest cruel act is to create a revision control system which is expressly designed to make you feel less intelligent than you thought you were. [...] So Linus is here today to explain to us why on earth he wrote a software tool which, eh, only he is smart enough to know how to use.*

Présentation de Linus Torvalds venu parler de GIT chez Google en 2007

Difficultés.

33 commandes dans SVN 1.6, plus de 100 dans GIT

- Des concepts très différents
- Une façon de développer très différente :
  - Sauver des commits pour y revenir plus tard
  - Créer une branche locale est trivial, il faut en abuser
  - Transfert de commits facile d'une branche à l'autre

Installation.

Debian

```
sudo apt-get install git gitk
```

MacOs X

http://sourceforge.net/projects/git-osx-installer/

Windows

https://git-for-windows.github.io/

Configuration.

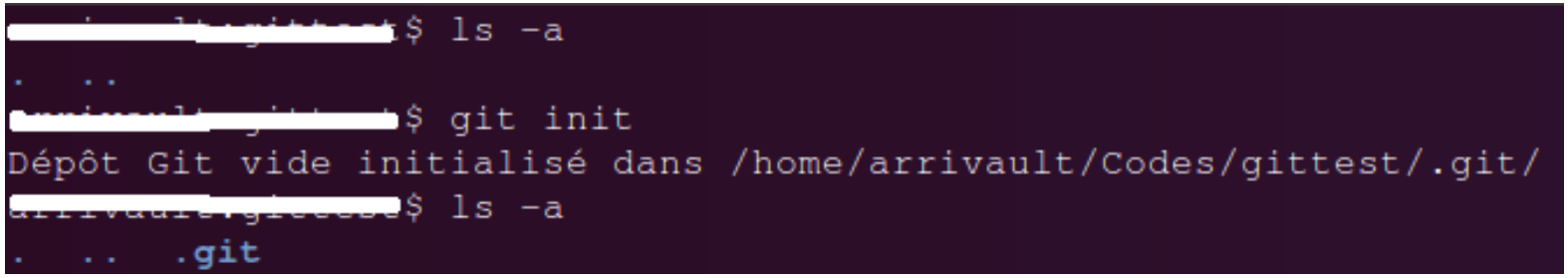
```
$ git config --global user.name 'Denis Arrivault'
$ git config --global user.email 'denis.arrivault@monmail.fr'
$ git config --list
user.name=Denis Arrivault
user.email=denis.arrivault@monmail.fr
```

- `--global` pour appliquer à tous les dépôts.
- Possibilité de modifier pour un dépôt donné.
- Effacement par `git var config --unset`

Dépôt local.

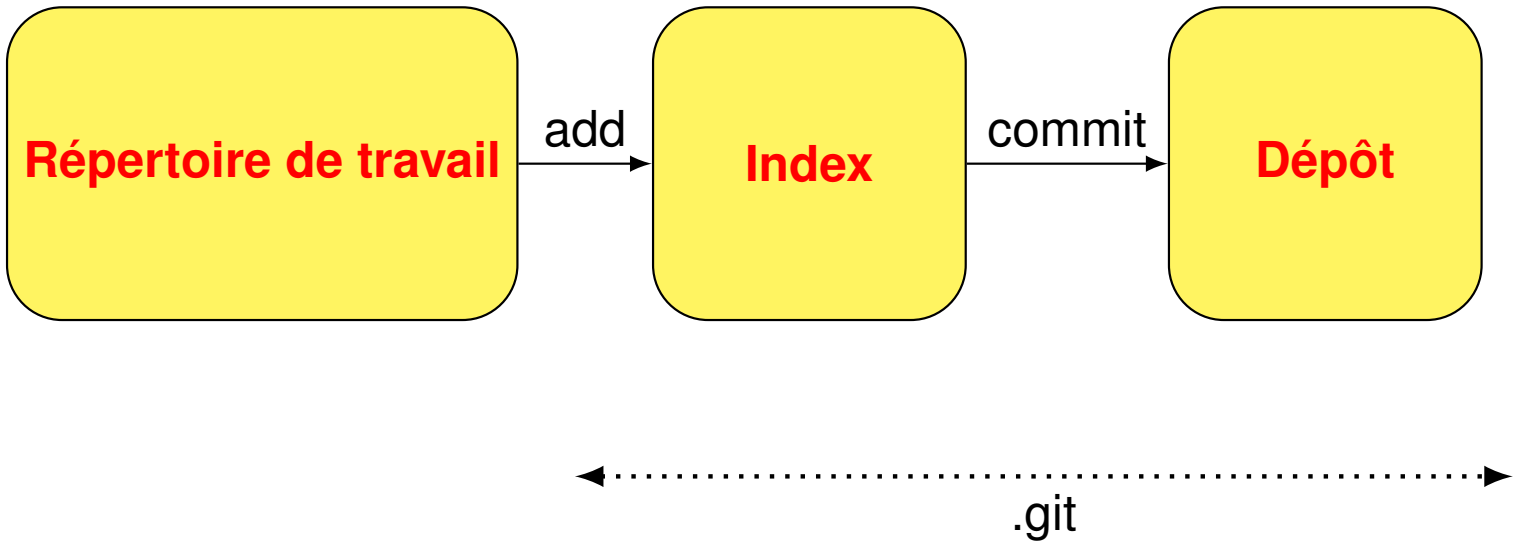
Création d'un dépôt

- Dans un répertoire de travail : `git init`
- Création d'un sous-repertoire `.git`



- Le répertoire de travail peut ne pas être vide.
- les valeurs de configuration globales sont utilisées mais peuvent être modifiées.

Dépôt local.



Structure du dépôt

- Les commandes git déplacent des informations entre ces trois espaces.

Dépôt local.

Quelques commandes :

Voir l'état du dépôt `git status`

Ajouter fichiers/répertoires à l'index `git add whatever` / `git add --all`

Enlever un fichier de l'index et du répertoire de travail `git rm myfile`

Enlever un fichier de l'index mais pas du répertoire de travail `git rm --cache myfile`

Renommer un fichier de l'index et du répertoire de travail `git mv oldname newname`

Mettre en dépôt `git commit -m "My message."`

Afficher les mises en dépôt `git log`

Empêcher le suivi d'une partie du dépôt Définir un fichier *.gitignore* à la racine du répertoire de travail contenant les répertoires et/ou fichiers à ne pas suivre.

Dépôt local.

Exemple :

```
[redacted] echo "Hello World" > english.txt
[redacted] ls -a
.  ..  english.txt  .git
[redacted] git status
Sur la branche master

Validation initiale

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        english.txt

aucune modification ajoutée à la validation mais des fichiers non suivis sont présents (utilisez "git add" pour les suivre)
[redacted] git add english.txt
[redacted] git status
Sur la branche master

Validation initiale

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)

        nouveau fichier: english.txt

[redacted] git commit -m 'Adding english hello message'
[master (commit racine) 06daf5b] Adding english hello message
 1 file changed, 1 insertion(+)
 create mode 100644 english.txt
arrivault:gittest$ git status
Sur la branche master
rien à valider, la copie de travail est propre
[redacted] git log
commit 06daf5bc66cbea14e6fad7a9233a2b98520454f8
Author: Denis Arrivault <denis.arrivault@lif.univ-mrs.fr>
Date:   Wed Oct 28 17:37:18 2015 +0100

    Adding english hello message
```

Dépôt local.

Les identifiants

```
commit 06daf5bc66cbea14e6fad7a9233a2b98520454f8
```

- Chaque commit possède un identifiant correspondant à une signature cryptographique (SHA1).
- Il dépend du contenu, du message et des prédécesseurs.
- Faible probabilité de collision.

Les branches.

- Chaque commit pointe vers son (ou ses) prédécesseur(s)
- Une branche est un pointeur sur un commit
- On créé une nouvelle branche avec `git branch nom_de_la_branche`
- La branche courante est appelée `HEAD`. Elle est repérée par un `*` dans la commande `git branch` qui affiche la liste des branches.
- Pour basculer `HEAD` vers une nouvelle branche on utilise `git checkout nom_de_la_branche`
- Le pointeur avance à chaque commit dans la branche courante.

```
[redacted] git branch
* master
[redacted] git branch mabranche
[redacted] git branch
  mabranche
* master
[redacted] git checkout mabranche
Basculement sur la branche 'mabranche'
[redacted] git branch
* mabranche
  master
[redacted] echo "Hallo Welt." > german.txt
[redacted] git add german.txt
[redacted] git commit -m 'Adding german hello message.'
[mabranche bd55360] Adding german hello message.
 1 file changed, 1 insertion(+)
 create mode 100644 german.txt
```

```
[redacted] git log --graph --decorate
* commit bd55360983c0cbcc650703f2f869ae72a355d6f9 (HEAD, mabranche)
| Author: Denis Arrivault <denis.arrivault@lif.univ-mrs.fr>
| Date:   Wed Oct 28 17:45:08 2015 +0100
|
|     Adding german hello message.
|
* commit 06daf5bc66cbea14e6fad7a9233a2b98520454f8 (master)
  Author: Denis Arrivault <denis.arrivault@lif.univ-mrs.fr>
  Date:   Wed Oct 28 17:37:18 2015 +0100

      Adding english hello message
```

Les branches.

Quelques commandes

Supprimer une branche `git branch -d nom`

Attention tout est perdu !

Renommer une branche `git branch -m oldname newname`

Afficher les mises en dépôts avec les branches `git log --graph --decorate`.

Fusionner la branche A avec la branche courante `git merge A`

Propagation des commits uniquement.

Conflits lors d'un merge

- Les fichiers en conflits sont retirés de l'index (visibles avec `git status`).
- Les lignes posant problème sont repérés dans les fichiers avec des chevrons (on peut utiliser `git diff`).
- Correction, ajout à l'index et mise en dépôt à nouveau.

Les branches.

Exemple de fusion de branches :

```
[redacted] echo 'Kaixo Mundua' > basque.txt
[redacted] git add basque.txt
[redacted] git checkout master
A       basque.txt
Basculement sur la branche 'master'
[redacted] git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

        nouveau fichier: basque.txt

[redacted] git commit -m 'Adding basque hello message.'
[master 88e719f] Adding basque hello message.
 1 file changed, 1 insertion(+)
 create mode 100644 basque.txt
[redacted] git log --graph --decorate
* commit 88e719f43ee6bfbf2ac0cfe9513444503e4484cc (HEAD, master)
| Author: Denis Arrivault <denis.arrivault@lif.univ-mrs.fr>
| Date:   Wed Oct 28 17:52:41 2015 +0100
|
|     Adding basque hello message.
|
* commit 06daf5bc66cbea14e6fad7a9233a2b98520454f8
  Author: Denis Arrivault <denis.arrivault@lif.univ-mrs.fr>
  Date:   Wed Oct 28 17:37:18 2015 +0100

      Adding english hello message
```

```
[redacted] git merge mabranche
Merge made by the 'recursive' strategy.
 german.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 german.txt
[redacted] git log --graph --decorate
*   commit e9049bb4f4b6c425db094cc5daa59f41ca100505 (HEAD, master)
|\  Merge: 88e719f bd55360
| | Author: Denis Arrivault <denis.arrivault@lif.univ-mrs.fr>
| | Date:   Wed Oct 28 17:53:09 2015 +0100
| |
| |     Merge branch 'mabranche'
| |
| * commit bd55360983c0cbcc650703f2f869ae72a355d6f9 (mabranche)
| | Author: Denis Arrivault <denis.arrivault@lif.univ-mrs.fr>
| | Date:   Wed Oct 28 17:45:08 2015 +0100
| |
| |     Adding german hello message.
| |
* | commit 88e719f43ee6bfbf2ac0cfe9513444503e4484cc
|/  Author: Denis Arrivault <denis.arrivault@lif.univ-mrs.fr>
|   Date:   Wed Oct 28 17:52:41 2015 +0100
|
|       Adding basque hello message.
|
* commit 06daf5bc66cbea14e6fad7a9233a2b98520454f8
  Author: Denis Arrivault <denis.arrivault@lif.univ-mrs.fr>
  Date:   Wed Oct 28 17:37:18 2015 +0100

      Adding english hello message
```

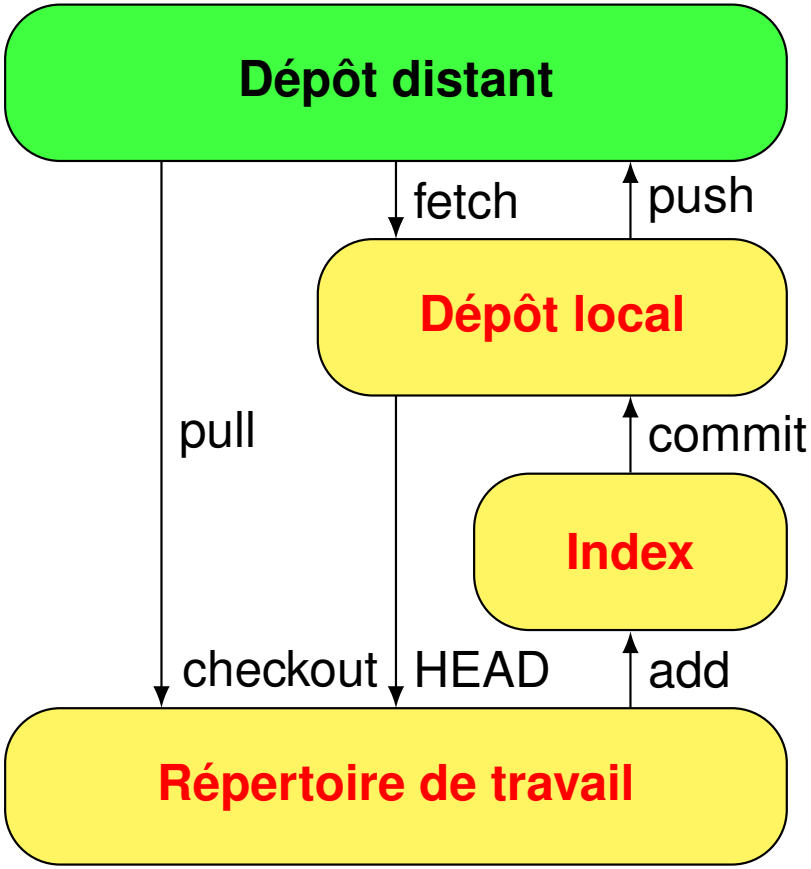
Dépôt distant.

Récupération d'un dépôt distant

- On fait une copie de travail locale avec `git clone url_depot_distant`
- La copie locale a la même structure qu'un dépôt local.
- le dépôt local contient par défaut tout l'historique de la branche master.

```
[redacted] git clone git@gitlab.lif.univ-mrs.fr:benoit.favre/macaon.git
Clonage dans 'macaon'...
remote: Counting objects: 9038, done.
remote: Compressing objects: 100% (4012/4012), done.
remote: Total 9038 (delta 6850), reused 6514 (delta 4976)
Réception d'objets: 100% (9038/9038), 371.48 MiB | 11.19 MiB/s, done.
Résolution des deltas: 100% (6850/6850), done.
Vérification de la connectivité... fait.
[redacted] ls
gittest  macaon
[redacted] cd macaon
[redacted] ls -a
.         build-cross-mingw.sh  clean.sh        COPYING          Doxyfile.in  .gitignore    python  tests
..        build.sh              CMakeLists.txt  COPYING.LESSER   external     INSTALL       README
AUTHORS   Changes               cmake-modules   data             .git         macaon.pc.in  src
[redacted] git branch
* master
[redacted]
```

Dépôt distant.



Dépôt distant.

Mises à jours

Mettre à jour son dépôt local sans changer le répertoire de travail `git fetch`

Mettre à jour son dépôt local et son répertoire de travail (`fetch + merge`)

```
git pull
```

Consultation des branches distantes

Lister les branches distantes `git branch -a`

Créer une branche locale pour suivre une branche distante existante `git branch localbranch --track origin/distantbranch`

Il est possible de pointer `HEAD` directement sur une branche distante mais on ne doit pas les modifier directement.

Dépôt distant.

Envoie des modifications

Pousser une branche locale suivie vers le dépôt distant `git push origin branchname`

Pousser une branche locale non suivie vers le dépôt distant `git push -u origin branchname`

Dans ce dernier cas la branche locale sera suivie par `remote/branchname` sur le dépôt distant.

Remarque

- Il est possible d'ajouter plusieurs dépôts avec : `git remote add name url`

Pour aller plus loin.

Contenu d'un commit `git show id`

Positionner `HEAD` sur un commit donné `git checkout id`

Amender le dernier commit `git commit -amend`

Générer une mise en dépôt qui défait ce qui a été fait depuis le commit xxx

`git revert xxx`

Effacer tous les commits réalisés depuis le commit xxx `git reset xxx`

Laisse les modifications dans le répertoire de travail. Utiliser l'option `--hard` pour une suppression définitive.

Enlever un fichier de l'index `git reset filename`

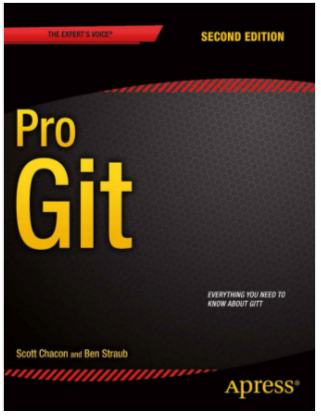
Obtenir de l'aide `man git`, `man git <command>`

Pour aller plus loin.

Pro Git

Scott Chacon

Apress, http://git-scm.com/book




GitLab

Quoi ?

- Plateforme de gestion de développement collaboratif de logiciel.
- Plus précisemment : gestionnaire web de dépôts Git avec un wiki et un traceur de bogues.
- Similaire à GitHub mais déployable sur un serveur privé.
- Développé par GitLab B.V. depuis 2011.
- Libre (licence MIT).
- Complètement gratuit jusqu'en 2013.
- un version gratuite (Community Edition) et une version payante (Enterprise Edition) avec support et fonctionnalités supplémentaires.

Pourquoi ?

- Un serveur privé a été installé au LIF : https://gitlab.lif.univ-mrs.fr
- Tous les membres du LIF peuvent créer un compte simplement en se connectant avec leur login et mdp usuels.
- Possibilité de créer des projets (dépôt git + wiki + traçeur de bogues) privés ou publics (interne au LIF ou extérieur)

Comment ?

Tour du propriétaire

- Connection : le tableau de bord.
- Profile Settings : commencer par la saisie des clefs ssh
- Création de projet : définition des droits.

GitLab CI

Intégration Continue ?

Augmenter la qualité du code

- Vérifier automatiquement à chaque commit que le code compile, s'installe et passe les tests.
- Faire cette vérification sur différentes architectures.
- Être prévenu quand la vérification échoue.

- GitLab CI est intégré à GitLab depuis la version 8.
- Deux clients disponibles sur la plateforme du LIF : Ubuntu 12.04 et Ubuntu 14.04
- Possibilité de rajouter des clients avec docker (ssh keys ?).
- Les instructions d'exécution sur les clients sont données dans un fichier *.gitlab-ci.yml* à la racine du projet.

Exemple : https ://gitlab.lif.univ-mrs.fr/ci/projects/3