

Git

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 3 décembre 2020

Git

- Contrôle de version (VCS, SCM) : conserver l'historique
- Pour tous types de projet : code, images, présentations, article. . .
- VCS Centralisé : seulement sur un serveur distant
- VCS Distribué : sur de multiples serveurs distants
- Git ?

- Créé par ?

Git

- Contrôle de version (VCS, SCM) : conserver l'historique
- Pour tous types de projet : code, images, présentations, article. . .
- VCS Centralisé : seulement sur un serveur distant
- VCS Distribué : sur de multiples serveurs distants
- Git ? Local (!), centralisé ou distribué \Rightarrow tout le monde a une copie complète de l'historique
- Créé par ?

Git

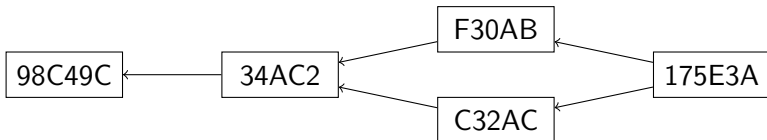
- Contrôle de version (VCS, SCM) : conserver l'historique
- Pour tous types de projet : code, images, présentations, article. . .
- VCS Centralisé : seulement sur un serveur distant
- VCS Distribué : sur de multiples serveurs distants
- Git ? Local (!), centralisé ou distribué \Rightarrow tout le monde a une copie complète de l'historique
- Créé par ? Linus [Torvalds](#) (?)

Git

- Contrôle de version (VCS, SCM) : conserver l'historique
- Pour tous types de projet : code, images, présentations, article. . .
- VCS Centralisé : seulement sur un serveur distant
- VCS Distribué : sur de multiples serveurs distants
- Git ? Local (!), centralisé ou distribué \Rightarrow tout le monde a une copie complète de l'historique
- Créé par ? Linus [Torvalds](#) (?) Créateur du noyau Linux

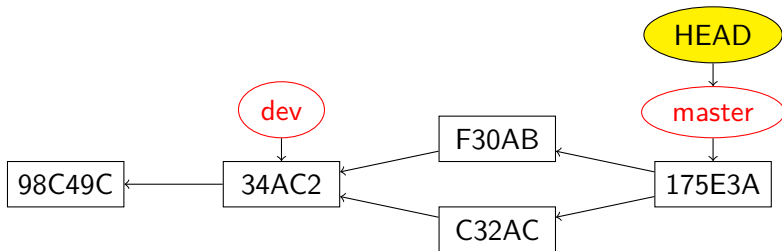
Commits et historique

- Blob : capture d'un fichier à un moment donné
- Commit : identifié par un hash SHA-1
 - Contient : structure de répertoires ; *blobs* ; auteur...
- Histoire : un DAG de « commits »
- Conservée dans un *dépôt* (repository)



Circuler dans l'historique

- Branches pointent vers des commits
- Pointeur HEAD vers la branche actuelle
- Indique le commit d'où est issu la version actuelle
- Circuler en utilisant la commande *checkout* (commit ou branche)



Work dir (WD)

- Histoire conservée *localement* dans `.git` à la racine du projet
- WD (« work dir ») : version du projet (fichiers et sous-répert.)
- Interaction avec sous-rép. `.git` : *uniquement* via outils git

```
/root
```

```
  /.git
```

```
  /rép1
```

```
    /fich1
```

```
  /fich2
```


Préparer un commit

Work dir	Index	HEAD
/rép1	/rép1	/rép1
/fich1	/fich1'	/fich1
/fich2	/fich2	/fich2'
/fich3		

- *Index* : changements à apporter au prochain commit
- *HEAD* : commit d'où le work dir actuel est issu
- Initialisation nouveau dépôt ?

Préparer un commit

Work dir	Index	HEAD
/rép1	/rép1	/rép1
/fich1	/fich1'	/fich1
/fich2	/fich2	/fich2'
/fich3		

- *Index* : changements à apporter au prochain commit
- *HEAD* : commit d'où le work dir actuel est issu
- Initialisation nouveau dépôt ? Index et HEAD vide

Préparer un commit

Work dir	Index	HEAD
/rép1	/rép1	/rép1
/fich1	/fich1'	/fich1
/fich2	/fich2	/fich2'
/fich3		

- *Index* : changements à apporter au prochain commit
- *HEAD* : commit d'où le work dir actuel est issu
- Initialisation nouveau dépôt ? Index et HEAD vide
- Juste après un commit ?

Préparer un commit

Work dir	Index	HEAD
/rép1	/rép1	/rép1
/fich1	/fich1'	/fich1
/fich2	/fich2	/fich2'
/fich3		

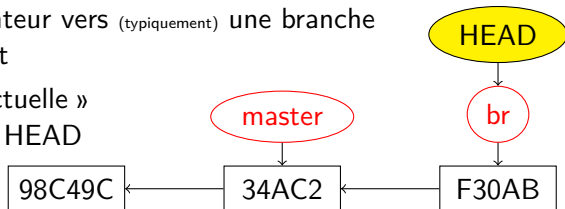
- *Index* : changements à apporter au prochain commit
- *HEAD* : commit d'où le work dir actuel est issu
- Initialisation nouveau dépôt ? Index et HEAD vide
- Juste après un commit ? Index vide

Préparer un commit : commandes

- `git add fichier` : blob mis dans index (« staged »)
- `git status` : liste untracked, tracked-modified, staged
- `git status --short` (sauf merge conflict) : idx VS HEAD ; WD VS idx.
- `git diff` : WD VS index
- `git diff --staged` : index VS HEAD
- `git commit` : commenter et expédier ! (Renvoie son id SHA-1)
- `git commit -v` : voir l'index en détail

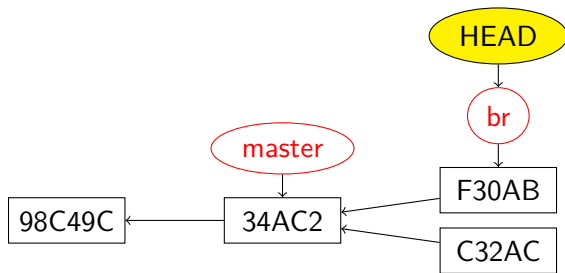
Branches et HEAD

- Branche : pointeur vers un commit
- HEAD : pointeur vers (typiquement) une branche et un commit
- Branche « actuelle » désignée par HEAD



- commit : avance HEAD et branche actuelle
- `git branch truc` : crée branche truc. HEAD inchangé !
- `git checkout truc` : change HEAD et met à jour WD
- Conseil : WD clean avant checkout !
- `git log --graph --decorate --oneline --all`

Fusion de branches



- `git merge autrebranche` : fusionne changements de autrebranche dans branche actuelle
- Si autrebranche est en avant de l'actuelle : « fast-forward »
- Sinon, « merge conflict » possible. Modifier les fichiers à la main et les ajouter à l'index puis `commit` pour créer un merge.
- `checkout d'un commit (ou tag) sans branche (detached head state)` : lecture !

Serveurs distants

- `git remote -v` : montrer les correspondants distants
- `git push` : envoyer historique au dépôt distant origin
- `git fetch` : récupère les commits distants (met à jour (ou crée) les références distantes)
- Réf. distante (« remote ref ») : branche `origin/branch` ou tag qui reflète branche sur dépôt distant
- « Remote-tracking branch » : branche locale qui connaît son correspondant distant
- `git branch -vv` : branches et leurs correspondants distants
- `git push origin mabranche` : sinon, nouvelles branches restent locales
- `git remote show origin` : voir les réf. distantes
- Suivre une branche distante `origin/branche` : `checkout branche`

Divers

- Utilisez gitignore ([modèles](#))
- Créez-vous une paire clé publique / privée
- Raccourcis : à éviter au début
- `git init` : dépôt vide dans rép. courant (rien n'est traqué)
- `git clone url` : cloner un dépôt (et non checkout!)
- `git stash` : WD \leftarrow HEAD
- `git tag -a montag` (tag annoté, recommandé) puis `git push origin montag`
- `git config --global` : écrit dans `~/.gitconfig`
- Indiquez propriété `user.name` (et `user.email`)
- Déterminer des [révisions](#) exemple : `HEAD^1` pour parent de HEAD
- [Alias](#)
- GUI pour diff : `git difftool`
- GUI pour merge : `git mergetool`

In case of fire



1. git commit



2. git push



3. leave building

Licence

Cette présentation, et le code LaTeX associé, sont sous [licence MIT](#). Vous êtes libres de réutiliser des éléments de cette présentation, sous réserve de citer l'auteur.

Le travail réutilisé est à attribuer à [Olivier Cailloux](#), Université Paris-Dauphine.

(Ceci ne couvre pas les images incluses dans ce document, puisque je n'en suis généralement pas l'auteur.)