

Git

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 21 février 2018

Git

- Contrôle de version (VCS, SCM) : conserver l'historique
- Pour tous types de projet : code, images, présentations, article. . .
- VCS local, centralisé, distribué ?

Git

- Contrôle de version (VCS, SCM) : conserver l'historique
- Pour tous types de projet : code, images, présentations, article. . .
- VCS local, centralisé, distribué ?
- Centralisé : seulement sur un serveur distant
- Distribué : copie locale et distante
- Git : distribué
- Créé par ?

Git

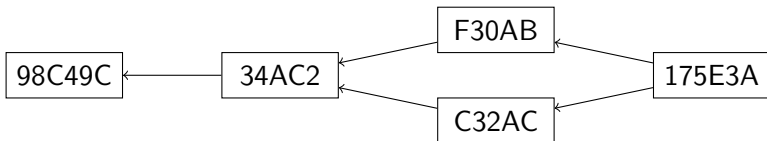
- Contrôle de version (VCS, SCM) : conserver l'historique
- Pour tous types de projet : code, images, présentations, article. . .
- VCS local, centralisé, distribué ?
- Centralisé : seulement sur un serveur distant
- Distribué : copie locale et distante
- Git : distribué
- Créé par ? Linus [Torvalds](#) (?)

Git

- Contrôle de version (VCS, SCM) : conserver l'historique
- Pour tous types de projet : code, images, présentations, article. . .
- VCS local, centralisé, distribué ?
- Centralisé : seulement sur un serveur distant
- Distribué : copie locale et distante
- Git : distribué
- Créé par ? Linus [Torvalds](#) (?) Créateur du noyau Linux

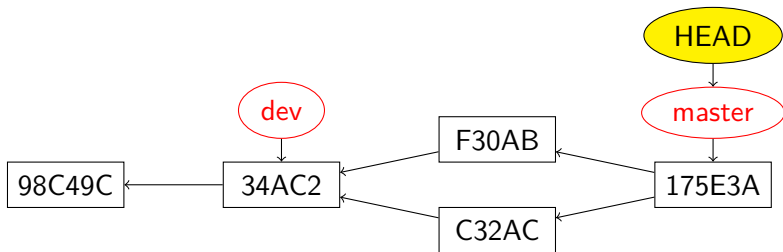
Commits et historique

- 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

/rép1
 /fich1
/fich2
/fich3

Index

/rép1
 /fich1'
/fich2

HEAD

/rép1
 /fich1
/fich2'

- *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		

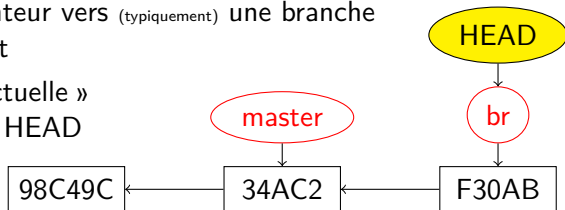
- *Index* : changements à apporter au prochain commit
- *HEAD* : commit d'où le work dir actuel est issu
- Initialisation nouveau dépôt ? Index et HEAD vide
- Juste après un commit ? Index vide

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

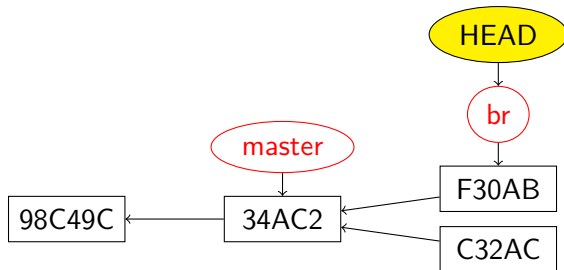
Branches et HEAD

- 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/branche` 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

Exercices : Git I

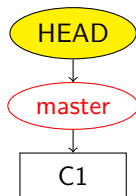
- Astuce : utilisez les Cheat sheets !

Git en local

- Définir globalement (au moins) `user.name`. Vérifier avec `git config --list`.
- Créer un répertoire projet et dedans un fichier `début.txt` contenant "coucou".
- Initialiser un dépôt git dans ce projet.
- Placer `début.txt` dans l'index. Modifier `début.txt` pour qu'il contienne "coucou2". Visualiser la différence sur ce fichier entre la version WD, index, et dépôt. Faire en sorte que le blob dans l'index contienne bien "coucou2".

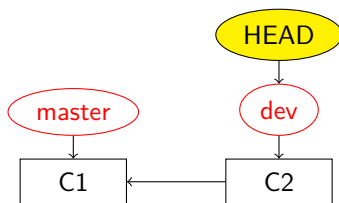
Exercices : Git II

- Effectuer un premier commit, qui contiendra uniquement `début.txt`. À l'issue de ce commit, vérifier que vous obtenez l'historique suivant.



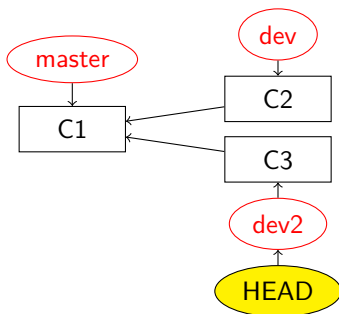
- Vous avez maintenant une idée audacieuse pour résoudre un problème dans votre projet. Comme vous n'êtes pas sûr de sa pertinence, vous désirez placer vos changements dans une nouvelle branche en attendant d'y réfléchir. Créer une branche "dev"; y commettre un fichier `audacieux.txt` (en plus de `début.txt`, inchangé) contenant "approche 1". Votre historique doit maintenant être celui-ci (vérifier!).

Exercices : Git III



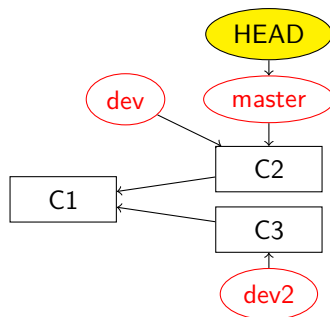
- À l'issue de ce travail harrassant, il vous vient une idée alternative. N'étant toujours pas sûr de la valeur de votre première idée (dans dev), vous repartirez de master pour l'implémenter. Depuis master, créer une branche dev2, et y commettre (en plus de début.txt, inchangé) un fichier audacieux.txt contenant "approche alternative". Vérifier ensuite votre historique.

Exercices : Git IV



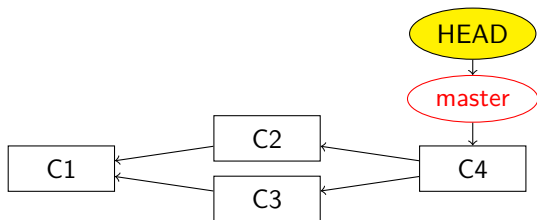
- À la réflexion, votre première idée est bonne. L'intégrer dans `master` pour obtenir l'historique suivant. Prédire si vous obtiendrez un fast-forward et vérifier.

Exercices : Git V



- Tout bien réfléchi vous aimez également votre deuxième idée. L'intégrer à son tour dans `master` et obtenir cet historique. Rencontrerez-vous un problème ?

Exercices : Git VI



- Imaginons qu'on aurait d'abord intégré dev2 à master (ceci aurait-il produit un fast-forward ?) puis dev au résultat. Quel aurait été le résultat final ?

Exercices : Git VII

Git distant

- Cloner votre dépôt local dans un autre répertoire sur votre même ordinateur, pour créer un deuxième dépôt. Nous les appellerons WD1, WD2, dépôt1 et dépôt2.
- Le clonage vous a créé un pointeur vers un serveur distant `origin`, et une « remote-tracking branch » `master`. Voir où pointent `origin`, `master` et `origin/master`. (Dans notre cas le terme serveur distant est impropre, puisque c'est votre même machine, mais généralement il s'agit effectivement d'une autre machine telle qu'un serveur GitHub.)

Exercices : Git VIII

- Ajouter un fichier "macontrib.txt" contenant votre prénom à WD1. Commettre dans dépôt1. Rapatrier les nouvelles informations dans dépôt2 (fetch). Vérifier que votre dernier commit dans dépôt1 existe bien dans l'historique de dépôt2 (et en profiter pour observer la différence entre `git log` et `git log -all`). Prédire où vont pointer `master` et `origin/master` et vérifier. Que contient le fichier `macontrib.txt` ? Pourquoi ?
- Fusionner dans dépôt2 vos dernières modifications venant de dépôt1.
- Créer un nouveau dépôt distant, dépôtGH, sur le site de GitHub. Connecter dépôt1 à dépôtGH (suivre et adapter les instructions fournies par GitHub). Envoyer vos informations locales à dépôtGH. Vérifier que vos informations locales sont bien parvenues (avec votre navigateur web).

Exercices : Git IX

- Cloner dépôtGH dans un nouveau répertoire, WDlocalclone et dépôtlocalclone. Modifier le fichier macontrib.txt (ajouter "coucou"), commettre dans dépôtlocalclone, envoyer à dépôtGH. Vérifier en ligne.
- Modifier macontrib.txt dans WD1 (ajouter "salut"), commettre dans dépôt1, envoyer à dépôtGH. Pourquoi l'envoi est-il refusé ? Que faire pour résoudre le problème et effectivement envoyer votre fichier modifié au dépôt en ligne ?

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.)