Git

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 11 août 2016

LAMSADE
UMR CNRS 7243
laboratoire d'analyse et modélisation de systèmes pour l'aide à la décision

DAUPHINE
UNIVERSITÉ PARIS

PSL
RESEARCH
UNIVERSITY

Git

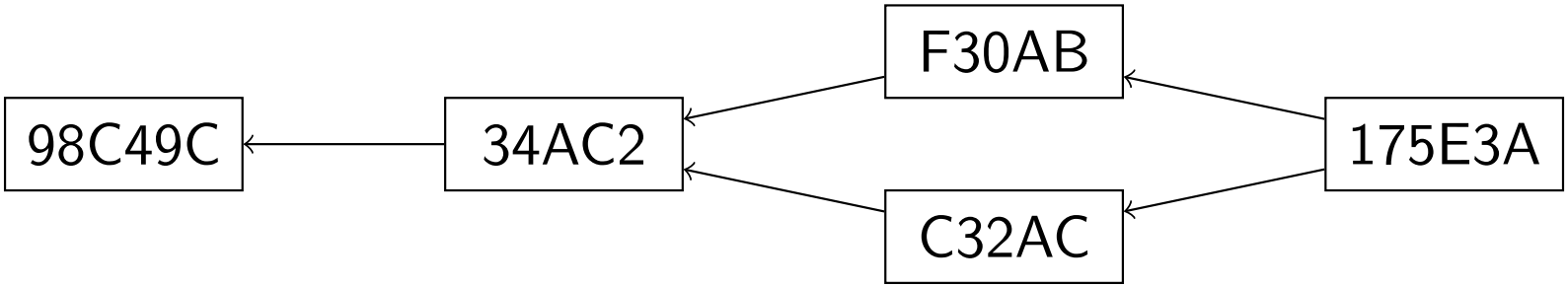
- Contrôle de version (VCS, SCM) : conserver l'historique
- Pour tous types de projet : code, images, présentations, article...
- VCS local, centralisé, distribué ?

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

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 »



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 répertoire ?

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 répertoire ? 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 répertoire ? 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 répertoire ? Index et HEAD vide
- Juste après un commit ? Index vide

Préparer un commit : définitions

Work dir	**Index**	**HEAD**
`/rép1`	`/rép1`	`/rép1`
`  /fich1`	`  /fich1'`	
`/fich2`	`/fich2`	`/fich2'`

Définitions (étant donné un fichier)

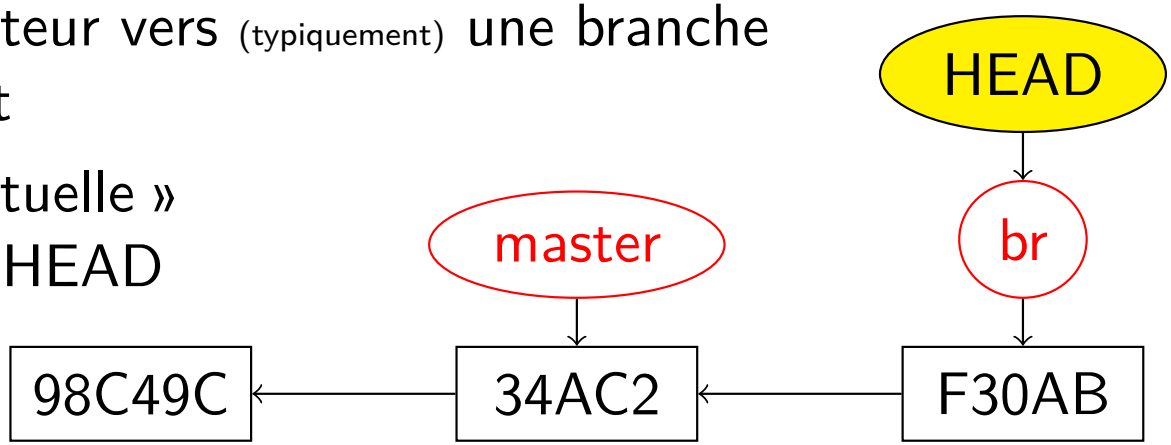
$\in$ index	blob ds index, peut être $\neq$ WD
$\in$ HEAD	blob ds HEAD, peut être $\neq$ WD
untracked	[$\notin$ index] et [$\notin$ HEAD]
tracked	[$\in$ index] ou [$\in$ HEAD] (ou les deux)
modified	[$\in$ index $\Rightarrow \neq$ index] ; [$\notin$ index $\Rightarrow \neq$ HEAD]
unmodified	[$\in$ index $\Rightarrow =$ index] ; [$\notin$ index $\Rightarrow =$ HEAD]

Préparer un commit : commandes

- `git add fichier` : blob mis dans index (« staged »)
- *clean* WD : tous (sauf fichiers dans gitignore) tracked et unmodified
- `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

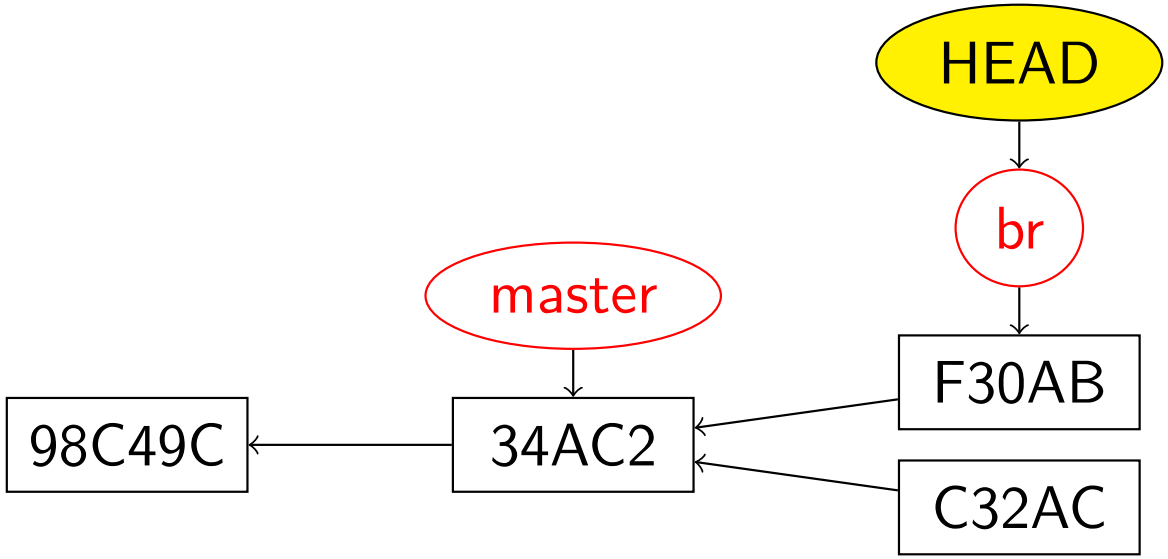
Branches et HEAD

- 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

- Réf. distante (« remote ref ») : pointeurs vers branches et tags sur dépots distants
- « Remote-tracking branch » : branche locale correspondant à une branche distante et qui connait l'état de la branche distante correspondante la dernière fois qu'on l'a vue
- Remote « origin » supposé configuré ici
- `git branch -vv` : voir branches et correspondants distants
- `git fetch` : récupère les commits distants ; met à jour (ou crée) les références distantes
- `git push origin mabranche` : sinon, nouvelles branches restent locales
- `git remote show origin` : voir les réf. distantes
- Suivre une branche distante : checkout `origin/branche` ; créer branche locale ; `git branch --set-upstream-to origin/branch`

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 ← 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
- Documentation
- GUI pour diff : `git difftool`
- GUI pour merge : `git mergetool`

Références

- Téléchargement officiel
- Livre Pro Git
- tryGit

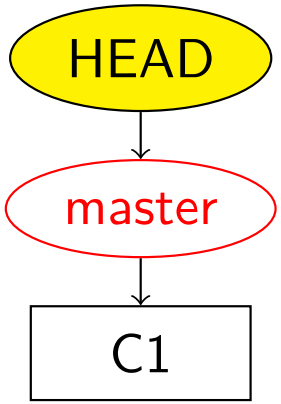
Exercices : Git I

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"`.

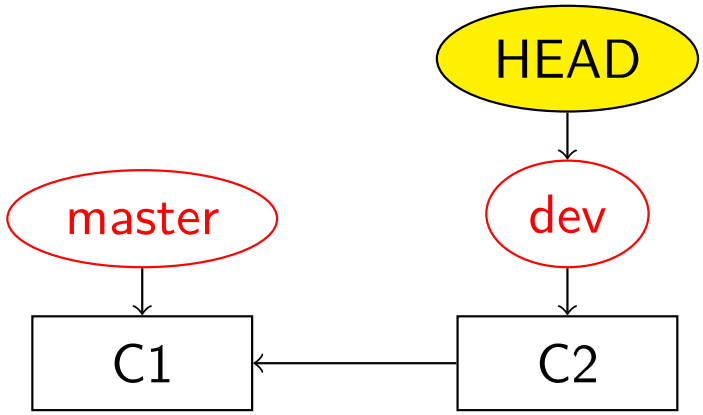
Exercices : Git II

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



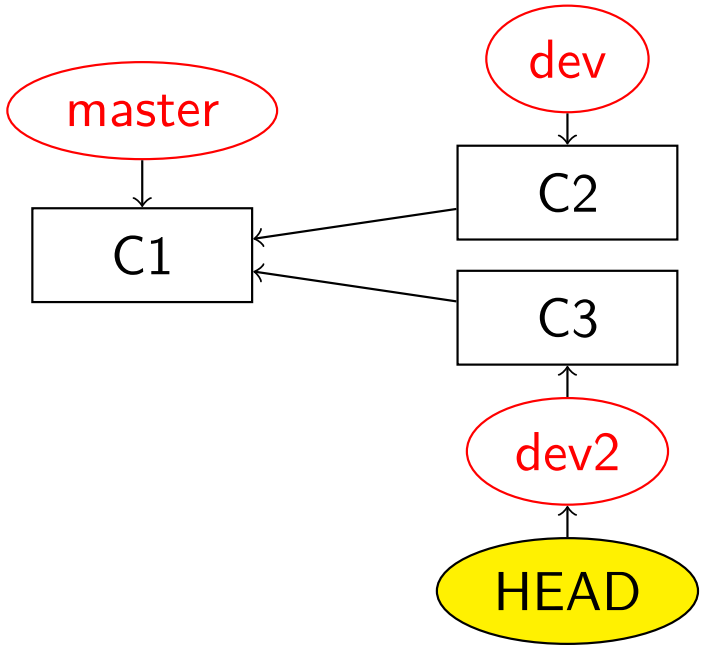
- Vous avez maintenant une idée audacieuse pour résoudre un problème dans votre projet. Comme vous n'êtes pas sûr de sa pertinence, vous désirez placer vos changements dans une nouvelle branche en attendant d'y réfléchir. Créer une branche `"dev"` ; y commettre un fichier `audacieux.txt` (en plus de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

Exercices : Git III



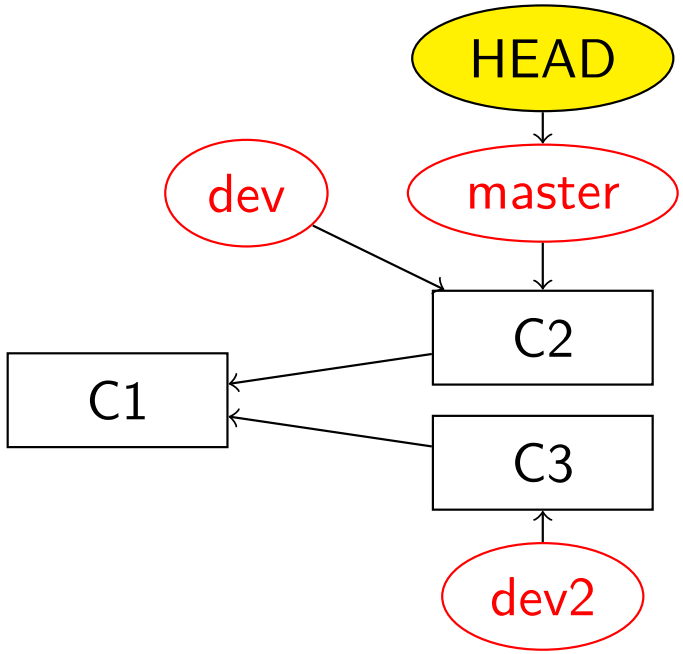
- À l'issue de ce travail harrassant, il vous vient une idée alternative. N'étant toujours pas sûr de la valeur de votre première idée (dans `dev`), vous repartirez de `master` pour l'implémenter. Depuis `master`, créer une branche `dev2`, et y commettre (en plus de `début.txt`, inchangé) un fichier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

Exercices : Git IV



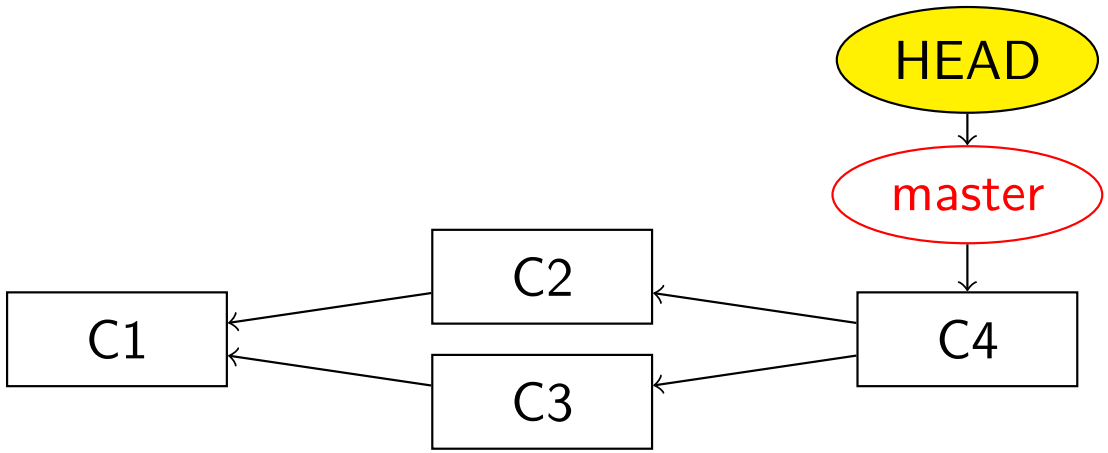
Exercices : Git V

- 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 VI

- Tout bien réfléchi vous aimez également votre deuxième idée. L'intégrer à son tour dans `master` et obtenir cet historique. Quel problème allez-vous rencontrer, ce faisant ?



- 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 central pour le projet de ce cours (ou votre dépôt local).
- 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`.
- Ajouter un fichier `"macontrib.txt"` contenant votre prénom à votre index local. Commettre dans votre dépôt git local. L'envoyer au dépôt distant. Vérifier (via l'interface web) qu'il s'y trouve et que le commit est associé à votre nom.
- Quand un collègue a fait de même, vous pouvez rapatrier sa modification en local. Après l'avoir fait, prédire où vont pointer `master` et `origin/master` et vérifier.

Exercices : Git VIII

- Si un collègue a publié sa modification avant la vôtre, quel va être votre problème ? Comment le résoudre ? Si vous êtes le premier à publier la modification, allez aider vos collègues à publier la leur !

Licence