

# Git

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 12 janvier 2022

THIS IS GIT. IT TRACKS COLLABORATIVE WORK  
ON PROJECTS THROUGH A BEAUTIFUL  
DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL  
COMMANDS AND TYPE THEM TO SYNC UP.  
IF YOU GET ERRORS, SAVE YOUR WORK  
ELSEWHERE, DELETE THE PROJECT,  
AND DOWNLOAD A FRESH COPY.



# Git

- Contrôle de version (VCS, SCM)
  - Conserver l'historique
  - Fusionner des changements parallèles
- Un ou plusieurs contributeurs
- Pour tous types de projet : code, images, présentations, article. . .
- VCS souvent centralisés : historique sur un serveur distant
- Git ?
  
- Créé par ?

# Git

- Contrôle de version (VCS, SCM)
  - Conserver l'historique
  - Fusionner des changements parallèles
- Un ou plusieurs contributeurs
- Pour tous types de projet : code, images, présentations, article...
- VCS souvent centralisés : historique sur un serveur distant
- Git ? Local (!) ; usage local, centralisé ou distribué  $\Rightarrow$  tout le monde a une copie complète de l'historique
- Créé par ?

# Git

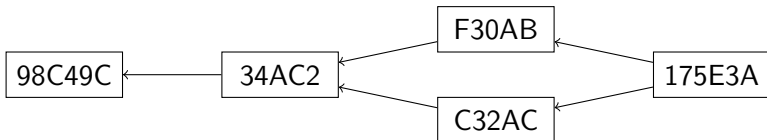
- Contrôle de version (VCS, SCM)
  - Conserver l'historique
  - Fusionner des changements parallèles
- Un ou plusieurs contributeurs
- Pour tous types de projet : code, images, présentations, article...
- VCS souvent centralisés : historique sur un serveur distant
- Git ? Local (!) ; usage local, centralisé 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
  - Fusionner des changements parallèles
- Un ou plusieurs contributeurs
- Pour tous types de projet : code, images, présentations, article...
- VCS souvent centralisés : historique sur un serveur distant
- Git ? Local (!) ; usage 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

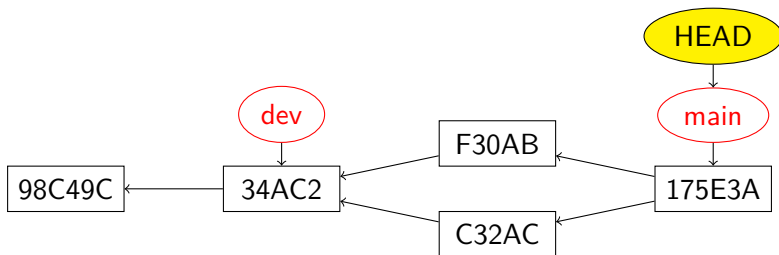
- Blob : capture d'un fichier à un moment donné
- Commit : identifié par un hash SHA-1
  - Contient : structure de répertoires ; *blobs* ; auteur...
- Histoire : un DAG de « commits »
- Conservée dans un *dépôt* (repository)



# Circuler dans l'historique

- Références git (git refs) : branches et références spéciales
- Références pointent vers des commits ou vers d'autres refs
- Branche : pointe un commit
- HEAD : pointe un commit, et généralement une branche ; appelés *actuels*
- Indique le commit d'où est issu la version actuelle
- Circuler en utilisant la commande `switch <branche>`

(anciennement `checkout`)





# 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` via commandes 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* : état capturé dans le commit référencé
- Initialisation nouveau dépôt ?
- 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* : état capturé dans le commit référencé
- Initialisation nouveau dépôt ? Index et HEAD vides
- Juste après un commit ?

# Préparer un commit

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

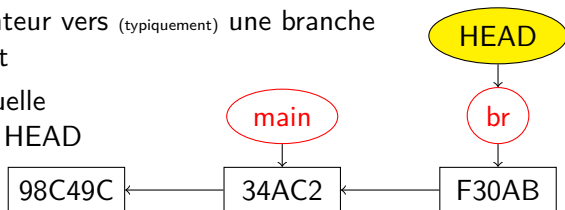
- *Index* : changements à apporter au prochain commit
- *HEAD* : état capturé dans le commit référencé
- Initialisation nouveau dépôt ? Index et HEAD vides
- Juste après un commit ? Index vide

## 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
- NB : commit bouge la branche actuelle

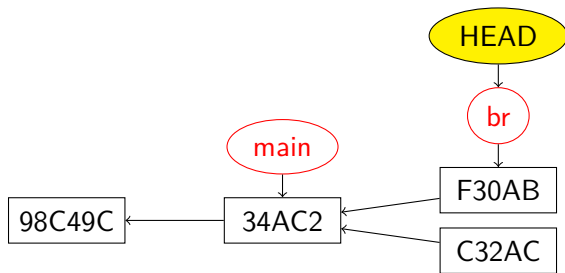
# Branches et HEAD

- 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

# 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/br` : `checkout br`

## Illustration



# 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.)