Git : concepts de base

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

Premier Commit

Deux étapes

- Indexation : ajouter le fichier au `Staging Area`
- Validation : valider seulement les fichiers modifiés et indexés

Avant indexation, tous les fichiers se trouvent dans le `staging area`

Premier Commit

Vérifions le contenu de notre dépôt

```
git status
```

Premier Commit

Vérifions le contenu de notre dépôt

```
git status
```

Commençons par créer un fichier `file.txt`

```
touch file.txt
```

Premier Commit

Vérifions le contenu de notre dépôt

```
git status
```

Commençons par créer un fichier `file.txt`

```
touch file.txt
```

Vérifions le contenu de notre dépôt

```
git status
```

Premier Commit

Vérifions le contenu de notre dépôt

```
git status
```

Commençons par créer un fichier `file.txt`

```
touch file.txt
```

Vérifions le contenu de notre dépôt

```
git status
```

Ajoutons une ligne dans `file.txt` **(utiliser** `subl file.txt`**)**

```
first
```

Premier Commit

Vérifions le contenu de notre dépôt

```
git status
```

Commençons par créer un fichier `file.txt`

```
touch file.txt
```

Vérifions le contenu de notre dépôt

```
git status
```

Ajoutons une ligne dans `file.txt` **(utiliser** `subl file.txt`**)**

```
first
```

Vérifions le contenu de notre dépôt

```
git status
```

Premier Commit

Indexons `file.txt`

```
git add file.txt
```

Premier Commit

Indexons `file.txt`

```
git add file.txt
```

On bien

```
git add .
```

Premier Commit

Indexons `file.txt`

```
git add file.txt
```

On bien

```
git add .
```

ou aussi

```
git add --all
```

Premier Commit

Indexons `file.txt`

```
git add file.txt
```

On bien

```
git add .
```

ou aussi

```
git add --all
```

Vérifions le contenu de notre dépôt

```
git status
```

Premier Commit

Faisons le commit

```
git commit -m "first commit"
```

Premier Commit

Faisons le commit

```
git commit -m "first commit"
```

Vérifions le contenu de notre dépôt

```
git status
```

Second (ou nième) Commit

Deux façons de faire

- Refaire les deux étapes de la section précédente
- Fusionner les deux étapes

Second (ou nième) Commit

Vérifions le contenu de notre dépôt

```
git status
```

Second (ou nième) Commit

Vérifions le contenu de notre dépôt

```
git status
```

Ajoutons une seconde ligne dans `file.txt` (son contenu devient)

```
first
second
```

Second (ou nième) Commit

Vérifions le contenu de notre dépôt

```
git status
```

Ajoutons une seconde ligne dans `file.txt` (son contenu devient)

```
first
second
```


Vérifions le contenu de notre dépôt

```
git status
```

Second (ou nième) Commit

Faisons le commit

```
git commit -a -m "second commit"
```

Second (ou nième) Commit

Faisons le commit

```
git commit -a -m "second commit"
```

Ou bien

```
git commit -am "second commit"
```

Second (ou nième) Commit

Faisons le commit

```
git commit -a -m "second commit"
```

Ou bien

```
git commit -am "second commit"
```


Vérifions le contenu

```
git status
```

Afficher la liste de Commit

Vérifions l'historique

```
git log
```

Afficher la liste de Commit

Vérifions l'historique

```
git log
```

Pour un affichage mono-ligne

```
git log --oneline
```

Afficher la liste de Commit

Vérifions l'historique

```
git log
```

Pour un affichage mono-ligne

```
git log --oneline
```


Pour un affichage mono-ligne mais avec un identifiant complet

```
git log --pretty=oneline
```

Afficher la liste de Commit

Pour un afficher seulement les deux derniers Commit

```
git log  -2
```

Afficher la liste de Commit

Pour un afficher seulement les deux derniers Commit

```
git log  -2
```

Pour afficher les points de différences avec le Commit précédent

```
git log -p -3
```

Afficher la liste de Commit

Pour un afficher seulement les deux derniers Commit

```
git log  -2
```

Pour afficher les points de différences avec le Commit précédent

```
git log -p -3
```

Pour un afficher les Commit sous forme d'un graphe

```
git log --oneline --graph
```

Afficher la liste de Commit

Les points de différence entre deux Commit

```
git diff idCommit1 idCommit2
```

Afficher la liste de Commit

Les points de différence entre deux Commit

```
git diff idCommit1 idCommit2
```

La différence d'un Commit avec le `staging area`

```
git diff idCommit1
```

Naviguer entre les Commit

Naviguer entre les Commit (ou voyager dans le temps)

Vérifier le contenu d'un fichier dans un commit précédent

Naviguer entre les Commit

Aller sur un autre Commit

```
git checkout idCommit
```

`idCommit` : identifiant du commit

Naviguer entre les Commit

Aller sur un autre Commit

```
git checkout idCommit
```

`idCommit` : identifiant du commit

On peut faire aussi

```
git checkout HEAD^^^
```

`HEAD^^^` : le troisième ancêtre du commit actuel

Naviguer entre les Commit

Aller sur un autre Commit

```
git checkout idCommit
```

`idCommit` : identifiant du commit

On peut faire aussi

```
git checkout HEAD^^^
```

`HEAD^^^` : le troisième ancêtre du commit actuel

Ou encore

```
git checkout HEAD~3
```

`HEAD~3` : le troisième ancêtre du commit actuel

Naviguer entre les Commit

On peut faire aussi

```
git checkout idCommit^^^
```

`idCommit^^^` : le troisième ancêtre du Commit précisé

Naviguer entre les Commit

On peut faire aussi

```
git checkout idCommit^^^
```

`idCommit^^^` : le troisième ancêtre du Commit précisé

Ou encore

```
git checkout idCommit~3
```

`idCommit~3` : le troisième ancêtre du Commit précisé

Naviguer entre les Commit

Pour pointer sur le dernier commit sans préciser son identifiant

```
git checkout master
```

Modifier le message du dernier Commit

En utilisant l'argument m

```
git commit --amend -m "second commit"
```

Modifier le message du dernier Commit

En utilisant l'argument m

```
git commit --amend -m "second commit"
```

Sans utiliser l'argument m

```
git commit --amend
```


Ensuite

- Saisir `i` pour modifier le message
- Pour terminer, cliquer sur `echap` puis saisir `:wq`
- Valider en cliquant sur `Entree`

Annuler un Commit

Avant cela

- Créer un deuxième fichier `file2.txt` et faire un troisième Commit avec le message `creating file2.txt`
- Ajouter une troisième ligne `third` dans `file.txt` et faire un cinquième Commit avec le message `third`
- Ajouter une quatrième ligne `fourth` dans `file.txt` et faire un cinquième Commit avec le message `fourth`

Annuler un Commit

Avant cela

- Créer un deuxième fichier `file2.txt` et faire un troisième Commit avec le message `creating file2.txt`
- Ajouter une troisième ligne `third` dans `file.txt` et faire un cinquième Commit avec le message `third`
- Ajouter une quatrième ligne `fourth` dans `file.txt` et faire un cinquième Commit avec le message `fourth`

Vérifier les nouvelles modifications

```
git log --oneline
```

Annuler un Commit

Comment annuler le commit ayant comme message `creating file2`

```
git revert idCommit
```

Ensuite, (modifier le message et) cliquer sur `echap` puis saisir `:wq` et cliquer sur `Entree` pour quitter

Annuler un Commit

Comment annuler le commit ayant comme message `creating file2`

```
git revert idCommit
```

Ensuite, (modifier le message et) cliquer sur `echap` puis saisir `:wq` et cliquer sur `Entree` pour quitter

Vérifier l'annulation avec

```
git log --oneline
```

Supprimer des modifications

Trois possibilités

- Annuler le commit et garder les modifications dans le `working directory` (mode **mixed** : par défaut)
- Annuler le commit et garder les modifications dans le `staging area` (mode **soft**)
- Annuler le commit et ne pas garder les modifications (mode **hard**)

Supprimer des modifications

Syntaxe

```
git reset --mode idCommit
```

Supprimer des modifications

Syntaxe

```
git reset --mode idCommit
```

Exemple

```
git reset --hard idCommit
```

Supprimer des modifications

Syntaxe

```
git reset --mode idCommit
```

Exemple

```
git reset --hard idCommit
```


Explication

- Tous les Commit réalisés après le commit ayant comme identifiant `idCommit` seront supprimés et impossible de les récupérer.
- En faisant `git status`, il n'y a rien à indexer ni à valider.

Supprimer des modifications

Exemple 2

```
git reset --soft idCommit
```

Supprimer des modifications

Exemple 2

```
git reset --soft idCommit
```


Explication

- Tous les Commit réalisés après le commit ayant comme identifiant `idCommit`.
- En faisant `git status`, les modifications sont dans le `staging area`.

Les tags

Problématique

- Pour accéder à un commit qui présente une version importante de notre projet
- Il faut chercher le commit en question en lisant les messages de tous les Commit, et ensuite faire `git checkout`
- Solution : utiliser les étiquettes (tags)

Les tags

Problématique

- Pour accéder à un commit qui présente une version importante de notre projet
- Il faut chercher le commit en question en lisant les messages de tous les Commit, et ensuite faire `git checkout`
- Solution : utiliser les étiquettes (tags)

Les tags ?

- une étiquette
- permet de marquer un Commit/une version de notre application
- référence vers un Commit

Les tags

Syntaxe de création d'un tag sur le Commit actuel

```
git tag -a nom-tag -m "message"
```

Les tags

Syntaxe de création d'un tag sur le Commit actuel

```
git tag -a nom-tag -m "message"
```

Exemple

```
git tag -a v0 -m "premiere version du projet"
```

Les tags

Syntaxe de création d'un tag sur le Commit actuel

```
git tag -a nom-tag -m "message"
```

Exemple

```
git tag -a v0 -m "premiere version du projet"
```

Syntaxe de création d'un tag sur un commit en utilisant son identifiant

```
git tag -a nom-tag idCommit -m "message"
```

Les tags

Syntaxe de création d'un tag sur le Commit actuel

```
git tag -a nom-tag -m "message"
```

Exemple

```
git tag -a v0 -m "premiere version du projet"
```


Syntaxe de création d'un tag sur un commit en utilisant son identifiant

```
git tag -a nom-tag idCommit -m "message"
```

Exemple

```
git tag -a v0 -m "premiere version du projet"
```

Les tags

On peut aussi se positionner sur un tag

```
git checkout nom-tag
```

Les tags

On peut aussi se positionner sur un tag

```
git checkout nom-tag
```

Et aussi ^ et ~

```
git checkout nom-tag^
```

Les tags

On peut aussi se positionner sur un tag

```
git checkout nom-tag
```

Et aussi ^ et ~

```
git checkout nom-tag^
```

Pour lister les tags

```
git tag --list
```

Les tags

On peut aussi se positionner sur un tag

```
git checkout nom-tag
```

Et aussi ^ et ~

```
git checkout nom-tag^
```


Pour lister les tags

```
git tag --list
```

Exemple

```
git tag nom-tag --delete
```

Les branches

Branches, oui on en connait déjà une : `master`

- branche principale
- contenant seulement des Commit représentant les différentes versions de notre application
- Comment faire alors ? $\Rightarrow$ Créer des branches et les utiliser

Les branches

Branches, oui on en connait déjà une : `master`

- branche principale
- contenant seulement des Commit représentant les différentes versions de notre application
- Comment faire alors ? $\Rightarrow$ Créer des branches et les utiliser

Une branche, c'est quoi ?

- déviation par rapport à la branche principale
- pointeur sur le dernier Commit
- permettant de développer une nouvelle fonctionnalité, préparer une correction

Les branches

Pour créer une branche

```
git branch nom-branche
```

Les branches

Pour créer une branche

```
git branch nom-branche
```

Changer de branche

```
git checkout nom-branche
```

Les branches

Pour créer une branche

```
git branch nom-branche
```

Changer de branche

```
git checkout nom-branche
```


Créer et changer de branche

```
git checkout -b nom-branche
```

Les branches

Remarque 1

- En créant une branche, cette dernière pointe sur le commit à partir duquel elle a été créée

Pour vérifier

```
git log --oneline
```

Les branches

Remarque 1

- En créant une branche, cette dernière pointe sur le commit à partir duquel elle a été créée

Pour vérifier

```
git log --oneline
```


Remarque 2

En faisant un Commit à partir de la branche créée, cette dernière dévie de la branche principale

Les branches

Pour lister les branches locales

```
git branch --list
```

Les branches

Pour lister les branches locales

```
git branch --list
```

Ou tout simplement

```
git branch
```

Les branches

Pour lister les branches locales

```
git branch --list
```

Ou tout simplement

```
git branch
```


Pour lister les branches (avec l'identifiant du dernier commit de chaque branche)

```
git branch -v
```

Les branches

Pour lister les branches distantes

```
git branch -r
```

Les branches

Pour lister les branches distantes

```
git branch -r
```

Ou aussi

```
git branch --all
```

Les branches

Pour supprimer une branche vide (ou fusionnée)

```
git branch -d nom-branche
```

Les branches

Pour supprimer une branche vide (ou fusionnée)

```
git branch -d nom-branche
```

Pour forcer la suppression d'une branche

```
git branch -D nom-branche
```

La fusion

Problématique

- Lors de l'élaboration d'un projet, plusieurs branches seront créées, chacune pour une tâche bien particulière
- Solution : fusionner les branches et rapatrier les modifications d'une branche dans une autre

La fusion

Fusion : deux cas possibles

- sans conflit
  - **fast forward** : sans commit de fusion
  - **non fast forward** (avec l'option `--no-ff`) : avec un commit de merge
- avec conflit : avec un commit de merge

La fusion

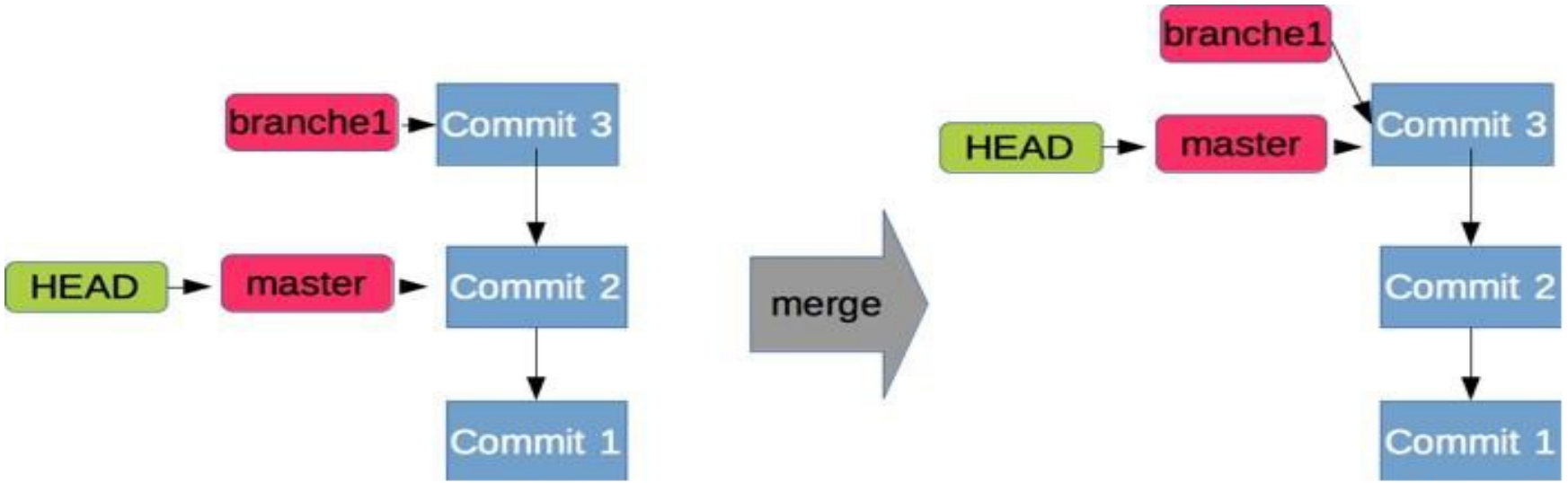
Fusion : deux cas possibles

- sans conflit
  - **fast forward** : sans commit de fusion
  - **non fast forward** (avec l'option `--no-ff`) : avec un commit de merge
- avec conflit : avec un commit de merge

Conflit ?

Sur deux branches différentes, sur une même ligne d'un même fichier, on a deux codes différents

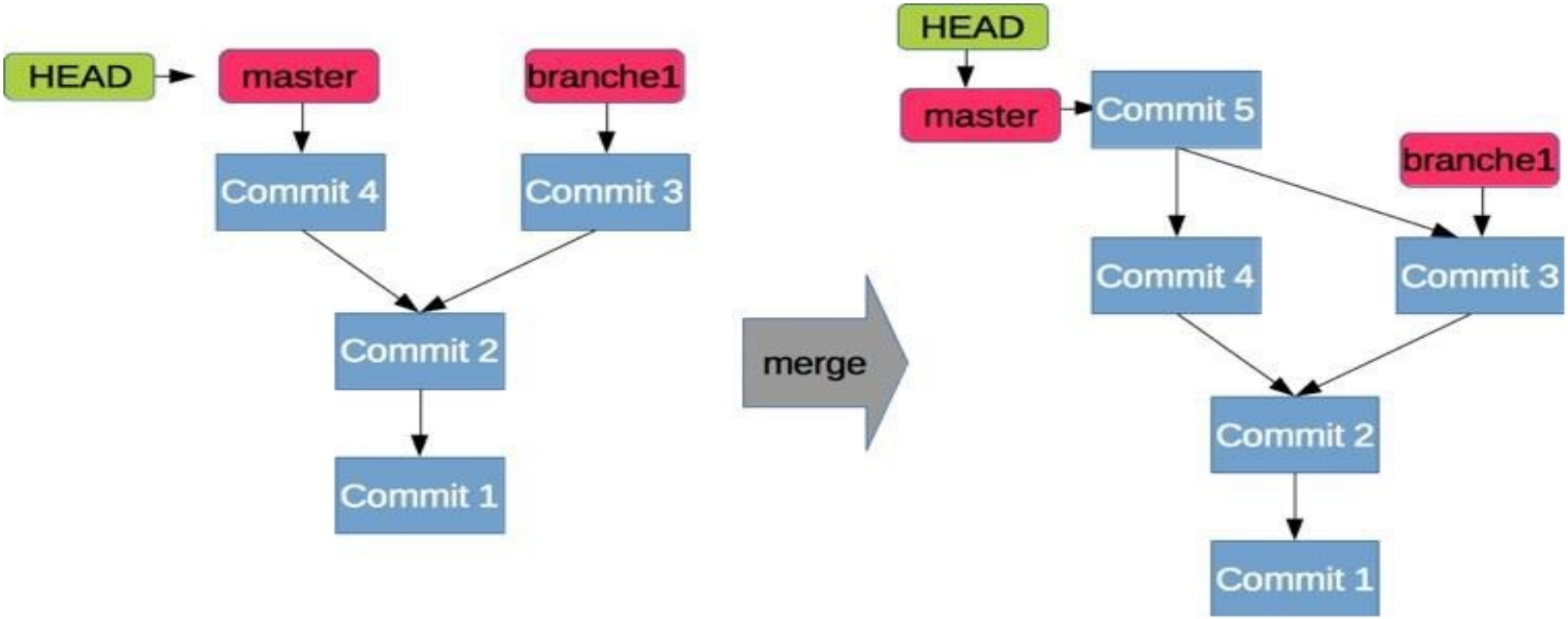
La fusion



À partir de la branche `master`

```
git merge nom-branche
```

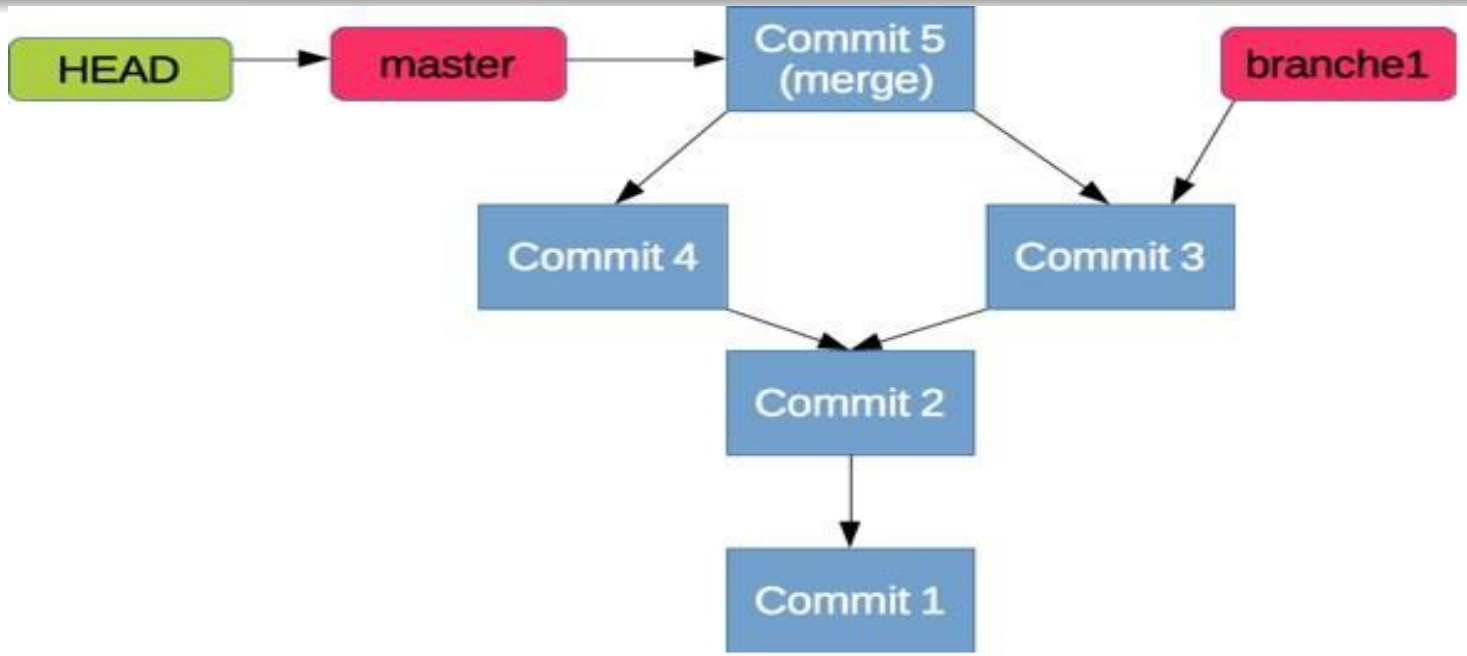
La fusion



Il faut ajouter l'argument `--no-ff`

```
git merge --no-ff nom-branche
```

La fusion



Étapes

- Exécuter la commande `git merge nom-branche`
- Résoudre le conflit en modifiant le(s) fichier(s) de conflit
- Faire un Commit de merge

La fusion

On peut annuler l'opération sans faire le commit de merge (après la détection un conflit)

- `git merge --abort`
- `git reset --merge`
- `git reset --hard HEAD`

La fusion

On peut toujours annuler le merge

```
git reset --hard HEAD^
```

La fusion

Exercice 1

- Créer un nouveau repository Git
- Ajouter un fichier et le commiter (C1)
- Créer une branche (B1) à partir de C1
- Faire un checkout sur B1
- Modifier le fichier et faire un Commit (C2)
- Merge B1 dans master de manière à avoir un Commit de merge dans master

La fusion

Exercice 2

- Créer un nouveau repository Git
- Ajouter un fichier et le commiter (C1)
- Modifier le fichier et le commiter (C2)
- Créer une branche (B1) à partir de C1
- Faire un checkout sur B1
- Modifier le fichier et faire un Commit (C3)
- Merge B1 dans master en résolvant le conflit

Le rebase

Problème de la fusion

- Des cycles, des fois, inutiles
- Des Commit de merge non nécessaires
- Solution : rebase

Le rebase

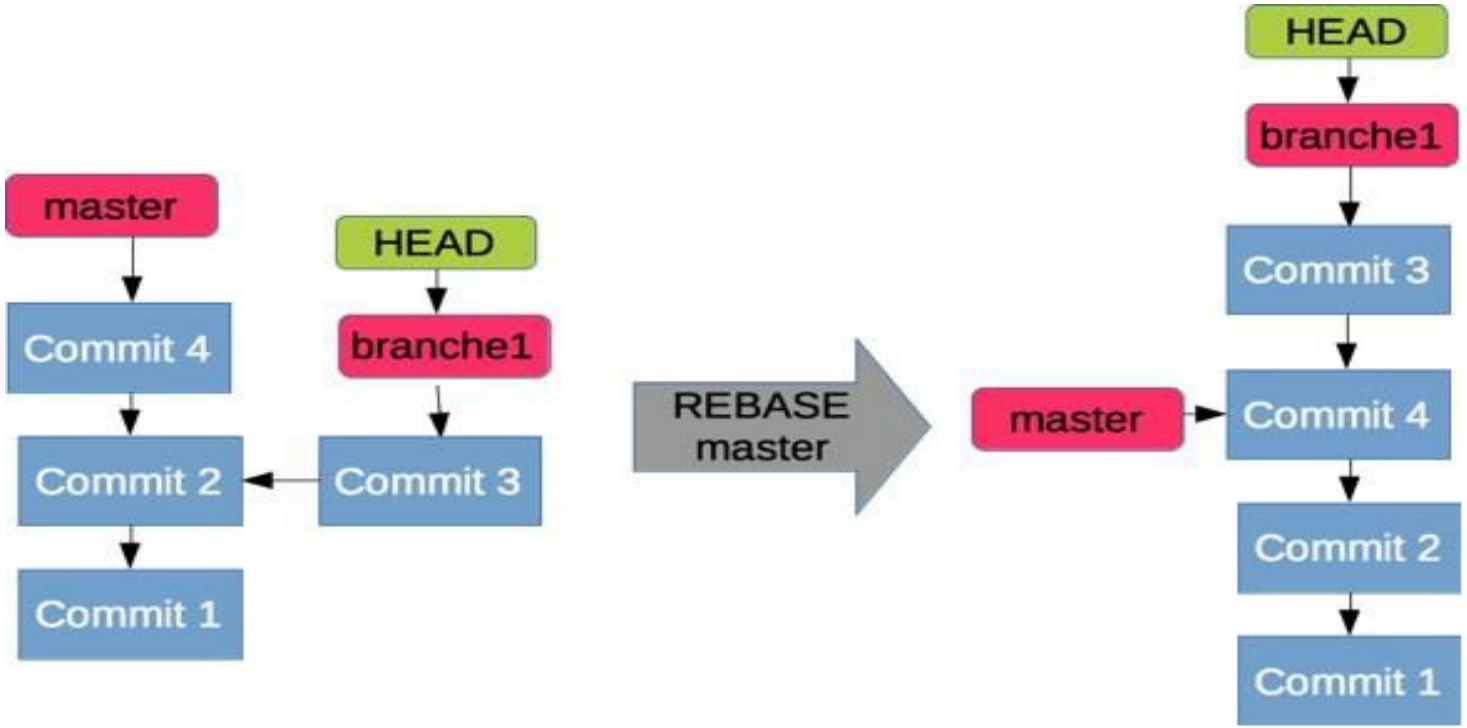
Problème de la fusion

- Des cycles, des fois, inutiles
- Des Commit de merge non nécessaires
- Solution : rebase

Le rebase, permet de

- manipuler l'historique en réécrivant le passé
- linéariser le graphe de commit en évitant les Commit de merge et en fusionnant les Commit d'une même branche dans un seul Commit

Le rebase



Depuis `branche1`, on exécute

```
git rebase master
```

Le rebase

Remarque

- `master`, branche principale, décalée par rapport à `branche1`

Solution : fast forward

```
git merge branche1
```

Le rebase

Exercice

- Créer un nouveau repository Git
- Ajouter un fichier et le commiter (C1)
- Modifier le fichier et le commiter (C2)
- Créer une branche (B1) à partir de C1
- Faire un checkout sur B1
- Créer un nouveau fichier et faire un Commit (C3)
- Merge B1 dans master de manière à avoir un historique linéaire

Le rebase

Le rebase interactif

- inverser l'ordre de deux ou plusieurs Commit
- modifier le message d'un Commit
- supprimer un Commit
- fusionner plusieurs Commit en un seul

Le rebase

Le rebase interactif

- inverser l'ordre de deux ou plusieurs Commit
- modifier le message d'un Commit
- supprimer un Commit
- fusionner plusieurs Commit en un seul

Comment ?

```
git rebase -i idCommit
```

Le rebase

Comment fusionner plusieurs Commit ? (dans cet exemple 3)

Commençons par

- créer un fichier `c.txt` et faire un Commit
- créer un fichier `d.txt` et faire un Commit
- créer un fichier `e.txt` et faire un Commit

En faisant `git log --oneline`

```
e8058fc (HEAD -> master) commit e
ff2c409 commit d
5960613 commit c
```

Pour fusionner les trois derniers Commit

```
git rebase -i HEAD~3
```

Les trois premières lignes affichées

```
pick 5960613 commit c
pick ff2c409 commit d
pick e8058fc commit e
```

Pour fusionner les trois derniers Commit

```
git rebase -i HEAD~3
```

Les trois premières lignes affichées

```
pick 5960613 commit c
pick ff2c409 commit d
pick e8058fc commit e
```

Dans **vim**

- cliquer sur `i` pour avoir le mode `INSERTION`
- remplacer `pick` de deux derniers Commit par `squash`
- cliquer sur `echap`, saisir `:wq` et cliquer sur `entree`

Pour fusionner les trois derniers Commit

```
git rebase -i HEAD~3
```

Les trois premières lignes affichées

```
pick 5960613 commit c
pick ff2c409 commit d
pick e8058fc commit e
```


Dans vim

- cliquer sur `i` pour avoir le mode `INSERTION`
- remplacer `pick` de deux derniers Commit par `squash`
- cliquer sur `echap`, saisir `:wq` et cliquer sur `entree`

Vérifier les changements

```
git log --oneline
```

Le rebase

Comment supprimer des Commit ?

Commençons par

- créer un fichier `f.txt` et faire un Commit
- créer un fichier `g.txt` et faire un Commit
- créer un fichier `h.txt` et faire un Commit

En faisant `git log --oneline`

```
e8058fc (HEAD -> master) commit h
ff2c409 commit g
5960613 commit f
```

Pour supprimer des Commit

```
git rebase -i HEAD~3
```

Les trois premières lignes affichées

```
pick 5960613 commit f
pick ff2c409 commit g
pick e8058fc commit h
```

Pour supprimer des Commit

```
git rebase -i HEAD~3
```

Les trois premières lignes affichées

```
pick 5960613 commit f
pick ff2c409 commit g
pick e8058fc commit h
```


Dans **vim**

- cliquer sur `i` pour avoir le mode `INSERTION`
- remplacer `pick` par `drop` pour les Commit à supprimer
- cliquer sur `echap`, saisir `:wq` et cliquer sur `entree`

Pour supprimer des Commit

```
git rebase -i HEAD~3
```

Les trois premières lignes affichées

```
pick 5960613 commit f
pick ff2c409 commit g
pick e8058fc commit h
```


Dans **vim**

- cliquer sur `i` pour avoir le mode `INSERTION`
- remplacer `pick` par `drop` pour les Commit à supprimer
- cliquer sur `echap`, saisir `:wq` et cliquer sur `entree`

Vérifier les changements

```
git log --oneline
```

Le rebase

Comment renommer un Commit ?

Commençons par créer un fichier `i.txt` et faire un Commit

En faisant `git log --oneline`

```
e8058fc (HEAD -> master) commit i
```

Pour renommer un Commit

```
git rebase -i HEAD~1
```

La première ligne affichée

```
pick e8058fc commit i
```

Pour renommer un Commit

```
git rebase -i HEAD~1
```

La première ligne affichée

```
pick e8058fc commit i
```


Dans vim

- cliquer sur `i` pour avoir le mode `INSERTION`
- remplacer `pick` par `reword` et modifier le message
- cliquer sur `echap`, saisir `:wq` et cliquer sur `entree`

Pour renommer un Commit

```
git rebase -i HEAD~1
```

La première ligne affichée

```
pick e8058fc commit i
```


Dans vim

- cliquer sur `i` pour avoir le mode `INSERTION`
- remplacer `pick` par `reword` et modifier le message
- cliquer sur `echap`, saisir `:wq` et cliquer sur `entree`

Vérifier les changements

```
git log --oneline
```

Le rebase

Comment inverser l'ordre des Commit ?

Commençons par

- créer un fichier `j.txt` et faire un Commit
- créer un fichier `k.txt` et faire un Commit
- créer un fichier `l.txt` et faire un Commit

En faisant `git log --oneline`

```
e8058fc (HEAD -> master) commit l
ff2c409 commit k
5960613 commit j
```

Pour inverser des Commit

```
git rebase -i HEAD~3
```

Les trois premières lignes affichées

```
pick 5960613 commit j
pick ff2c409 commit k
pick e8058fc commit l
```

Pour inverser des Commit

```
git rebase -i HEAD~3
```

Les trois premières lignes affichées

```
pick 5960613 commit j
pick ff2c409 commit k
pick e8058fc commit l
```


Dans **vim**

- placer le curseur au début du Commit à déplacer puis cliquer sur `dd` pour le couper
- cliquer sur `p` pour coller le Commit copié
- cliquer sur `echap`, saisir `:wq` et cliquer sur `entree`

Pour inverser des Commit

```
git rebase -i HEAD~3
```

Les trois premières lignes affichées

```
pick 5960613 commit j
pick ff2c409 commit k
pick e8058fc commit l
```


Dans **vim**

- placer le curseur au début du Commit à déplacer puis cliquer sur `dd` pour le couper
- cliquer sur `p` pour coller le Commit copié
- cliquer sur `echap`, saisir `:wq` et cliquer sur `entree`

Vérifier les changements

```
git log --oneline
```

La planque (stash)

Problématique

- Des travaux non-finis (qu'on ne peut valider)
- Nécessité de publier des travaux validés (d'une autre branche)
- Impossible de changer de branche sans supprimer les modifications courantes $\Rightarrow$ Solution : utiliser la planque pour mettre les travaux encours de coté

Exemple : sur la branche `master`

```
echo bonjour > fichier1.txt # crée et ajoute bonjour dans fichier.txt
echo bonsoir > fichier2.txt
git add . # indexer les deux fichiers
git commit -m "first commit"
git checkout -b ma-branche
```

Exemple : sur la branche `master`

```
echo bonjour > fichier1.txt # crée et ajoute bonjour dans fichier.txt
echo bonsoir > fichier2.txt
git add . # indexer les deux fichiers
git commit -m "first commit"
git checkout -b ma-branche
```

Exemple : sur la branche `ma-branche`

```
echo hello >> fichier1.txt
git commit -am "second commit for fichier1"
echo ciao >> fichier1.txt #
```

Exemple : sur la branche `master`

```
echo bonjour > fichier1.txt # crée et ajoute bonjour dans fichier.txt
echo bonsoir > fichier2.txt
git add . # indexer les deux fichiers
git commit -m "first commit"
git checkout -b ma-branche
```

Exemple : sur la branche `ma-branche`

```
echo hello >> fichier1.txt
git commit -am "second commit for fichier1"
echo ciao >> fichier1.txt #
```


Impossible de changer de branche car on n'a pas validé les dernières modifications

```
$ git checkout master
error: Your local changes to the following files would be overwritten
   by checkout:
         file1.txt
Please commit your changes or stash them before you switch branches.
Aborting
```

La planque (stash)

Pour ajouter un (ou plusieurs) fichier(s) à la planque

```
git stash save
```

La planque (stash)

Pour ajouter un (ou plusieurs) fichier(s) à la planque

```
git stash save
```

Si le fichier modifié se trouve dans le `staging area`

```
git stash push
```

La planque (stash)

Pour ajouter un (ou plusieurs) fichier(s) à la planque

```
git stash save
```

Si le fichier modifié se trouve dans le `staging area`

```
git stash push
```


Pour afficher le contenu de la planque

```
git stash list
```

La planque (stash)

Pour récupérer un fichier de la planque

```
git stash apply
```

La planque (stash)

Pour récupérer un fichier de la planque

```
git stash apply
```

Mais une copie de ce fichier est toujours dans la planque, vérifier

```
git stash list
```

La planque (stash)

Pour récupérer un fichier de la planque

```
git stash apply
```

Mais une copie de ce fichier est toujours dans la planque, vérifier

```
git stash list
```


Pour vider la planque

```
git stash drop
```

La planque (stash)

Pour récupérer un fichier de la planque sans qu'une copie y reste

```
git stash pop
```

La planque (stash)

Pour récupérer un fichier de la planque sans qu'une copie y reste

```
git stash pop
```

On peut préciser le nom du fichier à ajouter et donner un nom au stash au moment de l'ajout, ce qui nous permettra de traiter les fichiers un par un

La recherche

Pour chercher un mot dans le dépôt

```
git grep "mot"
```

La recherche

Pour chercher un mot dans le dépôt

```
git grep "mot"
```

Pour afficher le numéro de la ligne dans le fichier où le mot se trouve

```
git grep -n "mot"
```

Le fichier .gitignore

Idée

- Si on a un (ou plusieurs) fichier(s) (de configuration par exemple) qu'on ne voit aucun intérêt de les valider
- On peut les citer dans un fichier de configuration appelé `.gitignore`
- Un nom par ligne
- Ce fichier peut être indexé et validé

Le fichier .gitignore

Exemple

```
echo informatique.txt >> .gitignore
echo *.html  >> .gitignore
echo view/*  >> .gitignore
echo  java  >> informatique.txt
```

Explication

- En faisant `git status`, aucun fichier à indexer à l'exception de `.gitignore`
- Tous les fichiers avec l'extension `html` sont ignorés
- Aussi, tous les fichiers du répertoire `view`

L'historique du pointeur HEAD

Pour connaître le journal du pointeur HEAD

```
git reflog
```

L'historique du pointeur HEAD

Pour connaître le journal du pointeur HEAD

```
git reflog
```

Pour avoir un peu plus de détails

```
git log -g
```