Consultation et modification de l’historique git

ESGI - 3ème année IW 2020

Introduction

Avec git, nous avons vu que nous pouvions:

- Prendre des instantanés de notre travail pour y revenir si nécessaire (commit)
- Travailler sur des branches pour développer en parallèles sur différents aspects du projet
- Avoir un ou plusieurs dépôts de référence, locaux ou distants (remotes)
- Fusionner des branches, soit en fusionnant deux branches (merge) soit en réécrivant les commits à la suite des commits d'une autre branche (rebase)

Il existe d'autres outils pour travailler encore plus efficacement avec l'historique

git reflog

- Toute action entreprise sur le dépôt local est enregistré dans un historique consultable avec la commande: `git reflog`
- Les actions ont un sha1 de référence ainsi qu'un numéro selon leur ordre d'exécution
- Il est possible de revenir à l'état du dépôt à l'exécution d'une action avec:
  - `git reset --hard Id_du_commit`
  - `git reset --hard HEAD@{numéro}`
- Possibilité d'annuler une suppression de branche, un merge, un rebase etc en local
- Revenir sur une action est aussi une action en soit => possibilité de l'annuler aussi

git cherry-pick

Lorsqu'on travaille sur plusieurs branche, il arrive qu'on corrige un bug critique sur la branche de développement.

S'il existe une branche dédiée au déploiement, on peut vouloir récupérer ce hotfix sans le reste du code de développement pas suffisant prêt pour une mise en production.

Pour ça, on peut appliquer un ou plusieurs commits sur la branche actuelle avec:
`git cherry-pick ref_du_commit [other_ref, ...]`

git rebase interactive

- Permet de choisir un commit de référence et de modifier tous les commits entre le HEAD et le commit choisi, de HEAD à HEAD~(X-1) pour:
- modifier des messages de commits
- fusionner (squash) un commit avec le commit précédent
- modifier un commit (amend)
- réordonner les commits
- supprimer un commit

git stash

Lorsqu'on travaille et que l'on doit synchroniser sa branche ou effectuer une autre opération, une commande existe pour sauvegarder ses modifications sans devoir faire un commit:

- `git stash`
- `git stash save zone_de_remise`

Une fois la tâche terminée, pour réappliquer ces changements:

- `git stash pop`
- `git stash pop zone_de_remise`

On peut aussi lister les modifications remisées avec: `git stash list`

git stash

Un code “stashé” est encodé comme un commit.

`git/refs/stash` fait référence au stash le plus récent.

Un code “stashé” peut être réparti sur plusieurs commit, on peut consulter ceux du dernier stash avec: `git log stash` **ou** `git log stash@{0}`

git bisect

Permet de rechercher un commit défectueux en précisant un ancien commit correct et un commit contenant toujours le code défectueux.

Le principe est le même que de rechercher un nombre parmi un grand ensemble par dichotomie:

On commence par le milieu de l’ensemble, et si le nombre recherché est plus grand ou plus petit, on élimine la moitié des possibilités et on recommence.

git bisect

- On choisit un commit défectueux avec: `git bisect bad [ID]` (par défaut, HEAD sera choisi)
- On choisit ensuite un commit fonctionnel: `git bisect good ID`
- Git va alors effectuer un checkout entre les deux, on étudie alors le code, et on conclut soit que:
  a. le code est toujours défectueux: `git bisect bad`
  b. le code est fonctionnel: `git bisect good`
  c. le commit a été identifié, on termine l'opération: `git bisect reset`

git blame

Comme son nom l’indique, git blame permet de savoir à quel commit et par qui une ligne a été modifiée.