Le reflog

Temps de lecture : 3 minutes

Le `RefLog`

Le `RefLog` est l'historique des références qui ont été pointées par `HEAD` et par vos branches.

Autrement dit, il contient toutes les références des `commits` sur lesquels ont été placés `HEAD`, y compris de manière indirecte lorsque `HEAD` pointe sur une branche qui pointe sur un `commit`.

Et ce, également y compris pour les `commits` dits orphelins, à savoir les `commits` qui ne sont accessibles depuis aucune branche.

Le `ReFlog` est uniquement local, cela signifie que ce sont les déplacements que vous avez fait localement uniquement.

Si vous faites :

```
git reflog
```

Vous aurez par exemple :

```
5b0b87f (HEAD -> main) HEAD@{0}: rebase -i (finish): returning to ref
s/heads/main
5b0b87f (HEAD -> main) HEAD@{1}: rebase -i (pick): Un commit
8ca8356 HEAD@{2}: commit: Maj test pour corriger x
f6538e3 HEAD@{3}: reset: moving to HEAD^
12f0345 HEAD@{4}: rebase -i : avance rapide
f6538e3 HEAD@{5}: rebase -i (start): checkout HEAD~3
37b3e26 HEAD@{6}: rebase -i (finish): returning to refs/heads/main
37b3e26 HEAD@{7}: rebase -i (squash): Un commit
9c25c71 HEAD@{8}: rebase -i (squash): # Ceci est la combinaison de 2
commits.
6c9f6f4 HEAD@{9}: rebase -i (start): checkout HEAD~3
1a0f18f HEAD@{10}: rebase -i (finish): returning to refs/heads/main
1a0f18f HEAD@{11}: rebase -i (start): checkout HEAD~2
9b2e9c4 HEAD@{12}: rebase -i (finish): returning to refs/heads/main
9b2e9c4 HEAD@{13}: rebase -i (continue): Cinquième sur auth
1a0f18f HEAD@{14}: rebase -i (continue): Sixième
```

```
9abf3d6 HEAD@{15}: rebase -i (start): checkout HEAD~2
c010523 HEAD@{16}: rebase -i (finish): returning to refs/heads/main
c010523 HEAD@{17}: rebase -i (pick): Sixième
0854787 HEAD@{18}: rebase -i (pick): Cinquième sur auth
9abf3d6 HEAD@{19}: rebase -i (reword): Quatre sur auth
359ccca HEAD@{20}: rebase -i : avance rapide
```

Vous avez le raccourci du `hash` de chaque `commit` permettant de l'identifier.

Vous avez ensuite ce qu'on appelle le **raccourci** `Reflog`, nous y reviendrons ensuite.

Viens ensuite la commande effectuée.

Raccourcis `RefLog`

Les raccourcis `RefLog` sont tout simplement un moyen plus rapide de raisonner pour la navigation lors de récupération de données.

`HEAD@{n}` désigne le `commit` à `n` déplacement avant la position de `HEAD` actuelle (qui est `HEAD@{0}`) .

Par exemple :

```
git show HEAD@{2}
```

Cela affichera les détails du `commit` sur lequel était placé `HEAD` en avant-dernier.

La récupération de données avec `git reflog`

Comme nous l'avons vu, vous pouvez perdre la référence d'un ou plusieurs `commits` soit en faisant `git reset`, soit en supprimant une branche.

Ces références et les objets correspondants sont conservés par défaut pendant 90 jours localement.

Vous pouvez donc récupérer des `commits` perdus durant cette période.

Mais n'en abusez pas ! C'est complexe et cela ne doit servir qu'en cas de mauvaise manipulation.

Nous allons prendre un exemple, nous avons :

```
git log --oneline
```

Qui donne :

```
5b0b87f (HEAD -> main) Un commit
8ca8356 Maj test pour corriger x
f6538e3 trois
8cb6127 Second commit de test
bfc88f2 Premier commit
```

Mettons que nous supprimons de notre branche les deux dernières références par inadvertance en faisant :

```
git reset --hard HEAD~2
```

Nous avons donc maintenant :

```
f6538e3 (HEAD -> main) trois
8cb6127 Second commit de test
bfc88f2 Premier commit
```

Mais nous avons supprimé par inadvertance un `commit` de trop ! Nous voulions conserver le `commit 8ca8356` ! Comment le récupérer et le remettre sur notre branche ?

Commençons par voir la situation :

```
git reflog
```

Seules ces premières lignes nous intéresse :

```
f6538e3 (HEAD -> main) HEAD@{0}: reset: moving to HEAD~2
5b0b87f HEAD@{1}: rebase -i (finish): returning to refs/heads/main
5b0b87f HEAD@{2}: rebase -i (pick): Un commit
8ca8356 HEAD@{3}: commit: Maj test pour corriger x
```

Nous savons que `HEAD` pointe sur `main` qui pointe sur `f6538e3`.

Et nous savons qu'avant cette commande `HEAD` était sur `5b0b87f`.

Nous voyons également que le `commit` avec le message de validation qui nous intéresse a pour `hash 8ca8356`.

Nous pouvons donc créer une branche avec ce `commit` :

```
git branch recup 8ca8356
```

Voilà, le `commit` n'est plus orphelin car une branche pointe sur celui-ci, il est maintenant sauvé et ne sera pas détruit par le nettoyage automatique de `Git` au bout de quelques mois.

Nous pouvons faire une fusion en avance rapide sur `main` pour le récupérer sur la branche principale :

```
git merge recup
git branch -d recup
```

Nous aurions également pu ne pas passer par une branche et directement faire une fusion en avance rapide en utilisant le `hash` du `commit` à récupérer :

```
git merge 8ca8356
```

Dans les deux cas nous avons maintenant :

```
git log --oneline
```

Donnant :

```
8ca8356 (HEAD -> main) Maj test pour corriger x
f6538e3 trois
8cb6127 Second commit de test
bfc88f2 Premier commit
```

Notre `commit` est récupéré sur la branche `main` !

La récupération de données avec `git fsck`

Une autre option est de passer par la commande `git fsck` qui permet de vérifier la validité et l'intégrité des objets `Git` stockée dans la base de données locale de `Git`.

En faisant la commande :

```
git fsck --full
```

Vous obtiendrez la **liste de tous les objets `Git` qui ne sont pas référencés par d'autres objets et qui sont donc orphelins** :

```
Vérification des répertoires d'objet: 100% (256/256), fait.
dangling commit 419bd1b3112c583dc2fa59fd55699bf2f1c9cbe5
dangling blob 53dd7296d1e4dce32c962843fa0ff835dc3bc962
dangling commit a63a83939525aef5a46fdb1bd21615748bbefd56
dangling blob ac28f91b8c7314ce04f3f037948520dcc7a88ff7
dangling blob c05368e3ed4516b84b84980fc8e7927f32a90092
dangling commit c413b3309f58ddd7375f238969f858d069d9c0a7
```

Vous devez ensuite lire les `commits` pour retrouver celui que vous voulez en faisant :

```
git show 419bd1b3112c583dc2fa59fd55699bf2f1c9cbe5
```

Faites cette commande pour chaque objet `commit` qui sont orphelins jusqu'à retrouver le bon.

Ensuite, faites une fusion en avance rapide ou créez une branche sur le `commit` comme nous l'avons vu précédemment.