zenika

Ré-implémentons GIT

Dans la joie, la sueur et la bonne humeur

01/07/2025

© ZENIKA 2025 All rights reserved - Proprietary & confidential

Damien Aranda
Consultant
Zenika Rennes
damien.aranda@zenika.com

Maxime Maigret
Consultant
Zenika Rennes
maxime.maigret@zenika.com

Objectifs de la session

On est curieux de savoir comment git fonctionne, donc on va essayer de le réécrire.

On va se pencher sur le fonctionnement interne de git.

Scope de la session : lecture des fichiers interne de git

Inspiré de

Lien du projet

Github : https://github.com/Zenika/nightclazz-git-rennes/tree/main

Pour initialiser le TP :

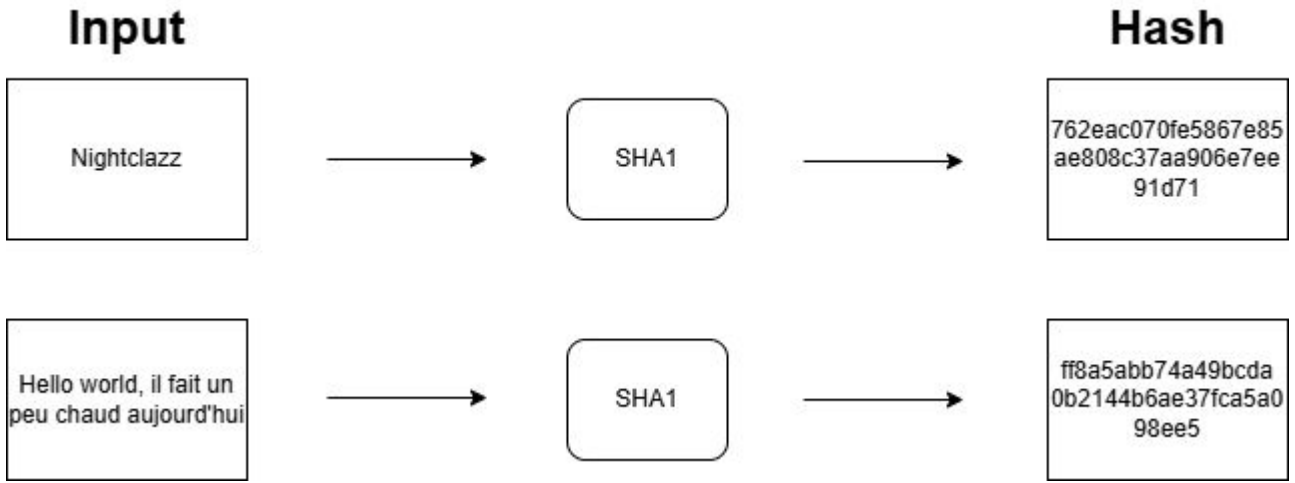
```
rm -r .git

git init

git add .

git commit -m “nightclazz”
```

Introduction

© ZENIKA 2025 All rights reserved - Proprietary & confidential

Sha1

Le sha1 est une fonction de hachage qui transforme une entrée en une chaîne de 40 caractère hexadécimal.



Organisation des fichiers

Deux dossiers :

- **objects** : l'ensemble de fichier qui servent à reconstruire le projet. Trié par sha1
- **refs** : traductions humaines des sha1, pour que ce soit plus pratique.

```
damien@why-damien:~/Prog/nightclazz-git-rennes$ tree .git/objects/
.git/objects/
├── 5b
│   └── adc4e5970cc9802578f45e1936602b26bbe609
├── 8a
│   └── 3e2535ec71c2e30e2a33e0d16ba95507fd9276
├── 8f
│   └── ce2f3567ce5013cba4a9542a84f25d32d9836c
├── 95
│   └── d09f2b10159347eece71399a7e2e907ea3df4f
├── 9c
│   └── d40397ba2c3b7429916b97930cc9584125b998
├── 9f
│   └── 11a4ecd446039411d5f3afc578a11bde3457fa
├── info
└── pack
    ├── pack-9a83c2b0e83f91a9444e29e4b0bc2d49b4d2c325.idx
    ├── pack-9a83c2b0e83f91a9444e29e4b0bc2d49b4d2c325.pack
    └── pack-9a83c2b0e83f91a9444e29e4b0bc2d49b4d2c325.rev
```

```
damien@why-damien:~/Prog/nightclazz-git-rennes$ tree .git/refs
.git/refs
├── heads
│   └── main
├── remotes
│   └── origin
│       ├── HEAD
│       └── main
└── tags
```

Contenu des fichiers

Tous les fichiers sont compressés au format deflate (zlib). Ils sont séparés en deux parties par le caractère “\x00”

- la première partie contient le type de fichier ainsi que la taille des données, le tout séparé par un espace
- la seconde partie contient les données sous format compressé

Exemple : 

Parlons des blobs

Les blobs (Binary Large OBject) sont la manière dont sont enregistrés les fichiers du projet. Ils contiennent aucune metadata sur le fichier, uniquement le contenu.

Pour lire le contenu d’un blob, il suffit d’ouvrir un fichier, vérifier que le type soit bien “blob” et que la taille des données correspondent à ce qui est indiqué dans le header

Exemple de projet

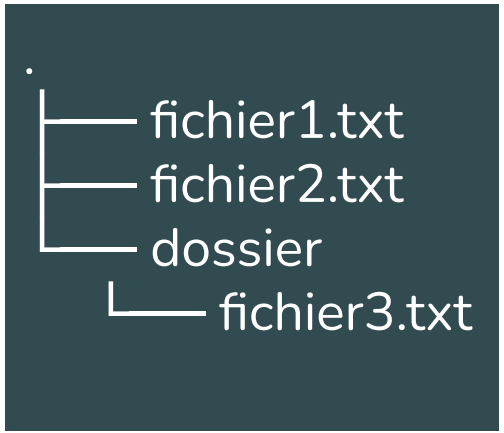
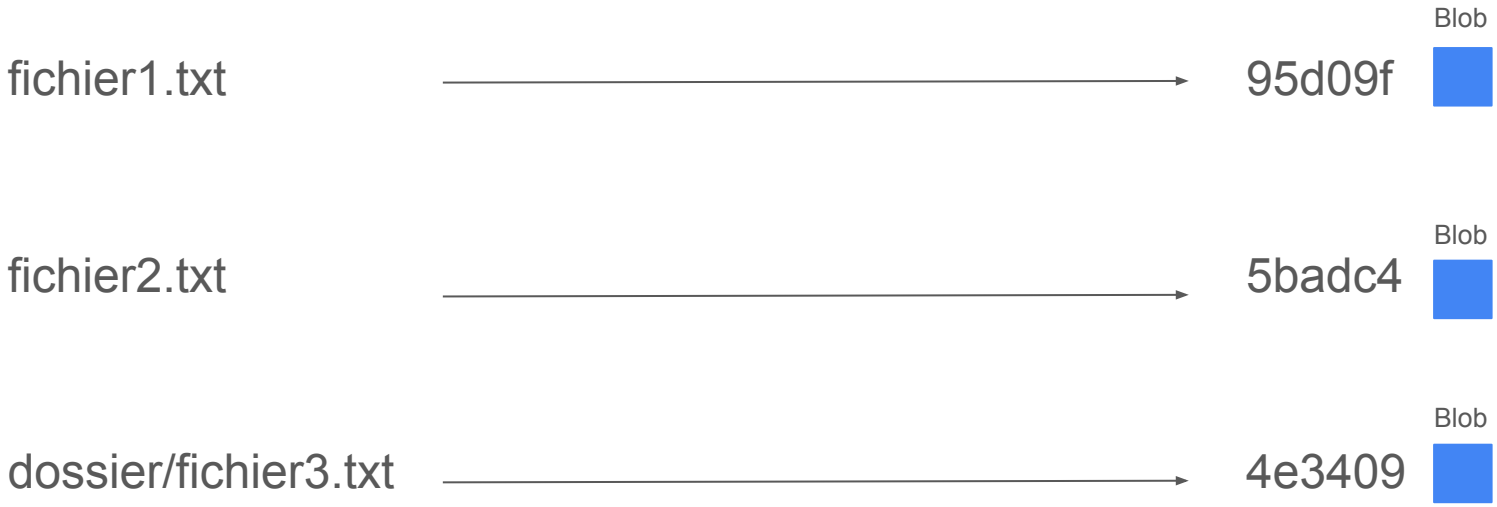


Illustration Blob



Exemple

Demandons à git de nous donner le sha1 d’un fichier du projet:

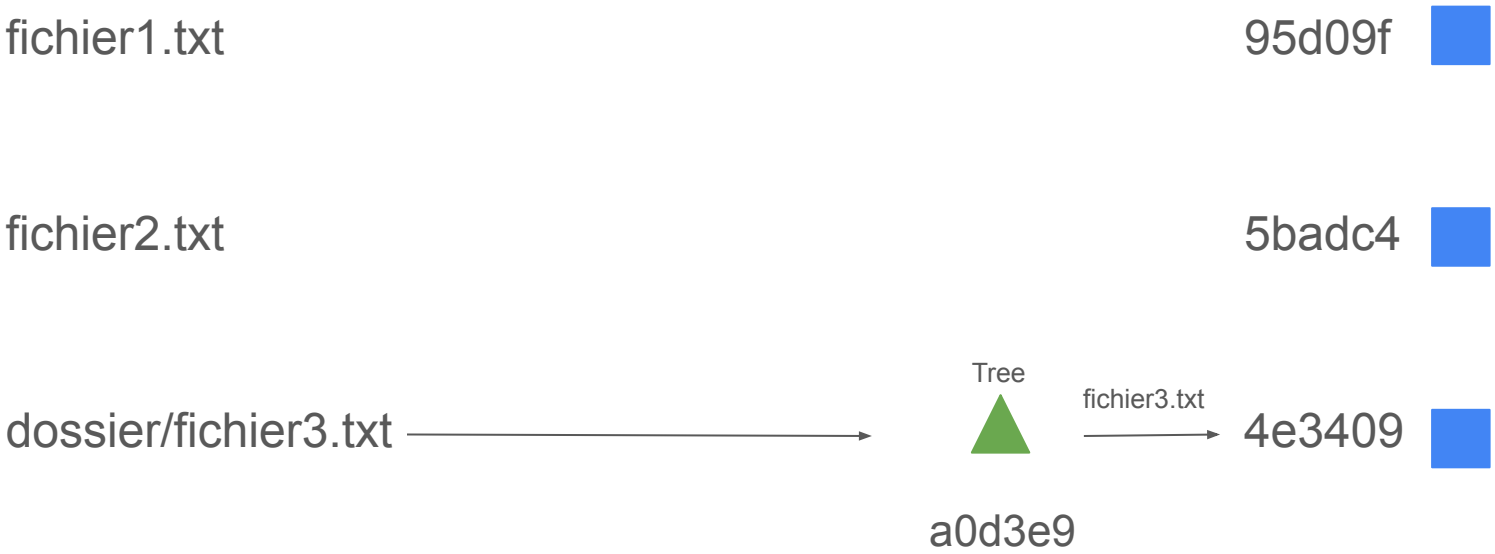
```
~/Prog/nightclazz-git-rennes$ git hash-object projet/fichier1.txt
95d09f2b10159347eece71399a7e2e907ea3df4f
```

Maintenant les trees

Les trees sont des fichiers qui contiennent l'arborescence des fichiers du projet. Ils contiennent aussi les metadatas sur les fichiers, comme le nom du fichier ou les permission qui lui sont associées.

```
~/Prog/nightclazz-git-rennes$ git rev-parse HEAD:projet/

9a09e4f267ebd5342e2b26c55919b6561e8ca9f4

~/Prog/nightclazz-git-rennes$ git ls-tree 8a3e2535ec71c2e30e2a33e0d16ba95507fd9276

040000 tree a0d3e9b3632b19e06340f5f8494bc94f66fca5f3    dossier

100644 blob 95d09f2b10159347eece71399a7e2e907ea3df4f    fichier1.txt

100644 blob 5badc4e5970cc9802578f45e1936602b26bbe609    fichier2.txt
```

Illustration tree



Données dans les trees

Les données sont enregistrées sous forme de liste de

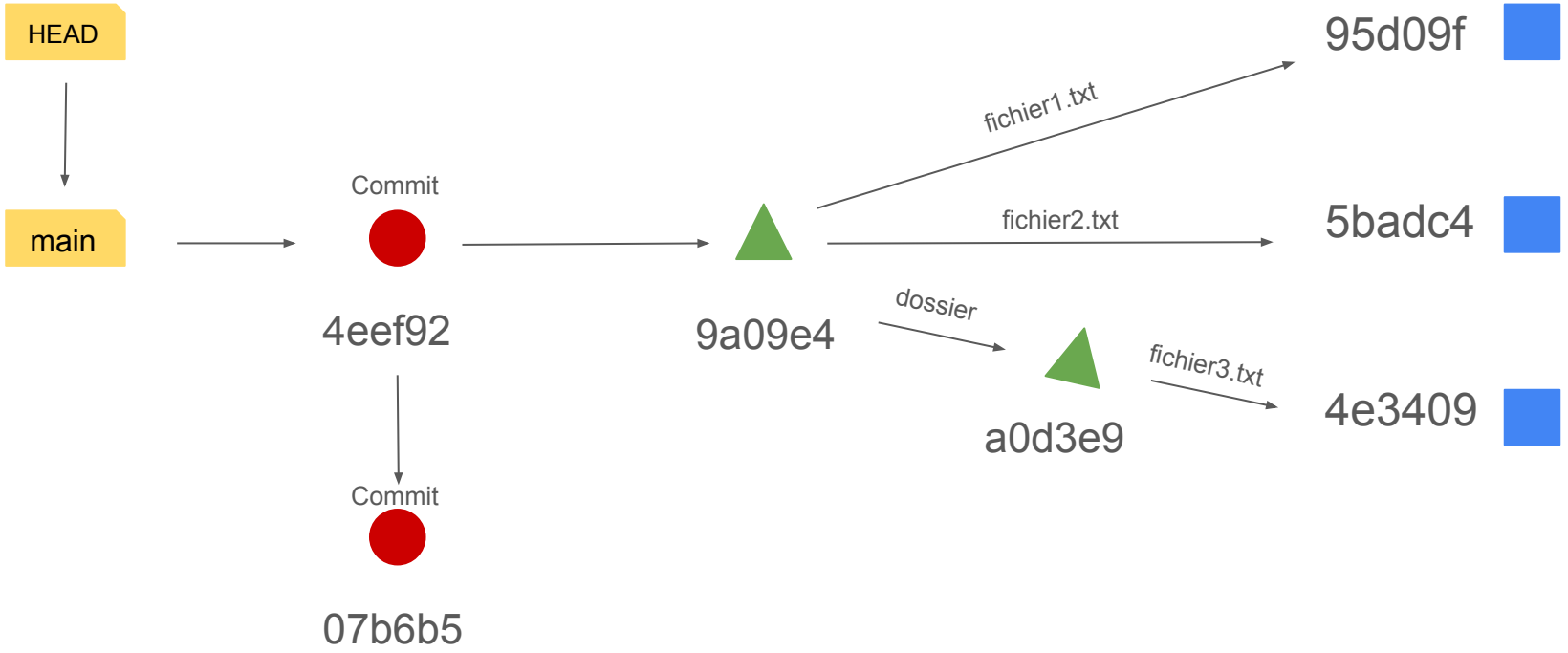
Il n’y a pas de séparateurs entre les éléments de la liste, et elle peut contenir des blob et d’autres trees

Bon et les commits alors

Les commits sont séparés en deux parties par deux retour à la ligne.

- La première partie est un ensemble de données comme l’auteur, le tree correspondant, le commit parent, etc. Ces données sont au format clé valeur, et les entrées sont séparées par des retours à la ligne.
- la seconde partie est le message entré par l’utilisateur lors du commit.

Exemple commit



Au boulot

© ZENIKA 2025 All rights reserved - Proprietary & confidential

TP : Récupération du path d’un fichier via son sha1

récupérez le path fichier vu précédemment :
⇒ 95d09f2b10159347eece71399a7e2e907ea3df4f

Implémentez la fonction : sha1→path

Spec :

- trouver la racine de répo git
- “aller dans le dossier .git/objects”
- retrouver le path à partir du sha1

points bonus :

- vérifier que le sha1 est valide

TP : récupération du contenu d’un fichier via son sha1

récupérez le contenu fichier : 95d09f2b10159347eece71399a7e2e907ea3df4f

- récupérer le path du fichier avec la méthode précédente
- ouvrir le fichier et en décompresser le contenu (zlib.decompress pour python)

TP : Récupération du type et des données d’un fichier git

récupérez le type et les données du fichier :
95d09f2b10159347eece71399a7e2e907ea3df4f

- Récupérer le contenu du fichier
- Séparer le header des données
- Séparer le type de fichier de la taille des données dans le header
- Renvoyer le tout

Point bonus :

- Vérifier que la taille des données est bien la bonne

TP Parsing d’un Tree via son sha1

Implémentez une fonction qui affiche le contenu d’un tree en connaissant son sha1

exemple de signature : sha1→liste(entrée)

spec :

- récupérer le path à partir du sha1
- convertir le contenu du fichier en une liste
- convertir les éléments de la liste en objet manipulable plus facilement

TP Parsing d’un commit via son sha1

Implémentez une fonction qui récupère les informations sur un commit à partir d’un sha1.

spec :

- récupérer le path à partir du sha1
- récupérer les headers du commit
- récupérer le message de commit

Ou en somme nous ?

On a développé un moyen de lire les commits de n’importe quel projet. On peut aussi lire les trees et les fichiers.

À partir de là il est possible de faire beaucoup de chose

- affichage de l’historique d’un projet
- checkout
- etc.

Conclusion

© ZENIKA 2025 All rights reserved - Proprietary & confidential

Merci