

Coding Week

École Centrale
Casablanca
2026

Casablanca, Maroc

CODING WEEK

Gestion de version



L'historique comme graphe à A3, Coding Week 1A

Hermann Agossou

Doctorant, UTER MID@S, ECC / UH2C

¹École Centrale Casablanca

Le modèle

01

Retour sur la conception logicielle

Trois questions sur le module précédent

Le test de A2

Quel test décide qu'un héritage est légitime ?

La discipline

Pourquoi annule-t-on une étape rouge plutôt que de la corriger en avançant ?

Et l'historique

Vous avez rendu une suite de commits en A2. Que prouvait-elle, au juste ?

LA TROISIÈME QUESTION EST LE SUJET DU JOUR

Vous avez produit une preuve sans qu'on vous ait dit ce qui la rendait recevable. Ce module explique pourquoi un historique en est une.

Quand deux personnes modifient le même fichier

Le problème avant l'outil

SANS OUTIL

Ce que tout le monde a déjà fait.

- ▶ rapport_v2_final_VRAI.docx
- ▶ On s'envoie le fichier et on croise les doigts
- ▶ Le dernier qui enregistre gagne

CE QU'IL FAUDRAIT

Trois besoins, avant toute commande.

- ▶ Savoir qui a changé quoi, et quand
- ▶ Revenir en arrière sans rien perdre
- ▶ Fusionner deux travaux sans en écraser un

AUCUN OUTIL POUR L'INSTANT

Le problème est réel avant d'avoir un outil. Gardez les trois besoins de droite : tout le reste du module y répond.

Le vocabulaire : commit, blob, tree, ref

Nommer les objets avant de manipuler les commandes

blob

Le contenu d'un fichier. Sans nom, sans date.

tree

Un répertoire : des noms associés à des blobs et à d'autres trees.

commit

Un tree, un ou plusieurs parents, un auteur, un message.

ref

Un nom qui désigne un commit. Une branche est une ref qui bouge.

Quatre noms. Aucune commande sur cet écran, et ce n'est pas un oubli.

Ce que dit Pro Git sur le modèle objet

La référence, et pourquoi le chapitre 10 vient en premier

LE MODÈLE OBJET

Git enregistre chaque version comme un instantané complet adressé par son contenu, construit à partir d'objets blob et tree, et non comme une suite de différences entre versions.

Voir: [1] Chacon & Straub, ch. 10

POURQUOI ON COMMENCE PAR LA FIN DU LIVRE

Une fois le modèle objet acquis, les chapitres 1 à 9 se déduisent. L'inverse est faux : on peut réciter des commandes pendant des années sans savoir ce qu'est un dépôt.

Git stocke du contenu, pas des différences

Adressage par contenu et conséquences

EXPÉRIENCE : DEUX CHEMINS, UN CONTENU

'a.txt' et 'b.txt' contiennent tous deux `ré bonjour z.` 'git hash-object' leur donne le même identifiant d'objet.

01 Le nom est calculé à partir du contenu

Pas d'un compteur, pas d'une date. Le même contenu donne toujours le même nom.

02 Un contenu identique n'est stocké qu'une fois

Copier un fichier ne double pas la taille du dépôt.

03 Un hash désigne un état, exactement

C'est ce qui permettra plus tard de s'en servir comme preuve.

Dans Git, l'historique est un graphe orienté acyclique

Chaque commit pointe vers son parent

BRANCHE 'MAIN'

'C' pointe vers son parent 'B', puis 'B' pointe vers 'A'.

► L'étiquette 'main' pointe vers 'C'

BRANCHE 'CORRECTIF'

'E' pointe vers 'D', puis 'D' pointe vers le parent commun 'B'.

► L'étiquette 'correctif' pointe vers 'E'

LES FLÈCHES VONT VERS L'ARRIÈRE

Un commit connaît son parent ; un parent ignore ses enfants. On peut donc remonter les parents depuis une ref. Supprimer une branche ne supprime pas immédiatement les objets, mais leur récupération dépend des autres refs, des reflogs et de la rétention.

Le DAG est propre à Git. Les idées durables sont les instantanés nommés, les parents et l'accessibilité.

Lire un historique et le raconter

Exemple entièrement résolu

Commit	Étiquette	Changement	Parent ou parents
f3a1c4	main	fusion du correctif	4c8d10 et 9b2e77
9b2e77	correctif	corrige le calcul de TVA	1a0be5
4c8d10	aucune	ajoute l'export CSV	1a0be5
1a0be5	aucune	version initiale	aucun

VÉRIFICATION

Reconstruisez le même récit à partir de la sortie de 'git log –graph' et vérifiez qu'il correspond au dessin. Toute divergence signifie que la lecture était fausse.

Compléter le graphe manquant

La moitié du raisonnement est donnée

01 1. Commit 'base' sur 'main'

Identifiant donné : '1a0be5'.

02 2. Créer 'essai', puis committer 'tentative'

Identifiant donné : '7d4f92'.

03 3. Revenir sur 'main', puis committer 'correctif'

Identifiant et graphe à compléter.

À COMPLÉTER

Dessinez le graphe résultant, placez les deux refs, et dites combien de commits sont accessibles depuis 'main'.

Les opérations

02

Une branche est un simple pointeur

Pourquoi créer une branche ne coûte rien

‘GIT SHOW-REF’ RÉVÈLE UNE ASSOCIATION

Dans cet exemple SHA-1, ‘refs/heads/main’ est associé à l’identifiant ‘f3a1c4...c2d4’. Une ref empaquetée représente la même association autrement.

Coût d’une branche

une ref

Commits copiés

0

CONSÉQUENCE POUR LA SUITE

Supprimer une branche supprime une ref, pas immédiatement les objets. Une récupération dépend encore des autres refs, des reflogs, de leur rétention et de la collecte.

Merge et rebase comme opérations de graphe

Deux formes différentes, un même contenu

MERGE

Ajoute un commit à deux parents.

- ▶ Le graphe garde la trace des deux chemins
- ▶ Les commits existants gardent leur identité

REBASE

Rejoue les commits ailleurs.

- ▶ Le graphe devient linéaire
- ▶ De NOUVEAUX commits, donc de nouveaux hashes
- ▶ À éviter sur une branche déjà partagée

LE CONTENU FINAL PEUT ÊTRE IDENTIQUE

Ce qui diffère est la forme du graphe et l'identité des commits. Choisir entre les deux, c'est choisir ce que l'histoire doit montrer.

Trouver le commit fautif par bisection

La preuve, pas seulement la réponse

WORKFLOW 01

Marquer

Un commit connu bon, un commit connu mauvais.

WORKFLOW 02

Tester

Git place l'arbre au milieu. On exécute le test.

WORKFLOW 03

Répondre

bon ou mauvais. L'intervalle est divisé par deux.

WORKFLOW 04

Isoler

Il reste un seul commit. C'est la réponse.

Commits dans l'intervalle

1000

Tests nécessaires

10

VÉRIFICATION

Placez-vous sur le commit accusé et sur son parent, exécutez le test qui échoue sur chacun, et confirmez que la transition est exactement là. C'est cette trace, et non le hash, qui constitue la preuve.

Voir: [2] Chacon & Straub, ch. 7

Récupérer un état partagé cassé

Sans détruire le travail des autres

SITUATION

La ref 'shared-work' a été reculée de deux commits. Le tip déplacé n'apparaît plus sous ce nom, mais le reflog local et 'collaborator-snapshot' peuvent encore fournir des preuves distinctes.

- ▶ Auditez les refs et reflogs avant toute modification.
- ▶ Récupérez sous 'recovered-shared-work', sans déplacer une ref existante.
- ▶ Prouvez que les deux fichiers de travail sont atteignables.

CE QUI SERA ÉVALUÉ

Les étapes destructrices que vous avez choisi de ne PAS faire, et votre raison. La récupération est un jugement sous incertitude sur ce que détiennent les autres machines.

L'historique comme preuve et comme filet

Ce que devient le journal quand personne n'a écrit le code

CE QUE L'HISTORIQUE PROUVE

Contenu, filiation et métadonnées enregistrées.

- ▶ Quel changement a introduit quel comportement
- ▶ Quel auteur et quelle date sont enregistrés, sans preuve d'identité par défaut

CE QUE L'HISTORIQUE PERMET

Réversibilité.

- ▶ Annuler un changement précis, pas tout
- ▶ Isoler par bisection sans tout relire
- ▶ Vérifier par étapes réversibles

POURQUOI CELA COMPTE DAVANTAGE MAINTENANT

Quand produire devient plus rapide que relire, la trace et la réversibilité prennent plus de poids. On y revient en A5.

Ce qui reste quand les commandes changent

Trois idées à garder

Des objets immuables, adressés par contenu

Dans Git, de nouveaux objets étendent le graphe et les refs mutables nomment les états accessibles.

La réversibilité est une propriété

Elle se conçoit, elle ne s'improvise pas au moment de la panne.

Nommer un état pour pouvoir en parler

Refs, tags et branches ne sont qu'une instance de ce geste.

CE QUI N'ÉTAIT QUE DE L'OUTILLAGE

Les commandes Git et les forges sont de l'outillage. Les concepts transférables sont les instantanés, la filiation, les noms, l'accessibilité et la réversibilité sous conditions.

Références

03