La chaine de blocs (La blockchain)

2022-03-09

# La chaine de blocs (La blockchain)

## Présentation

La structure de données de la chaîne de blocs est une liste ordonnée et rétro-liée aux blocs de transactions. La chaîne de blocs peut être stockée sous forme de fichier plat ou dans une simple base de données. Le client Bitcoin Core stocke les métadonnées de la chaîne de blocs à l'aide de la base de données LevelDB de Google. Les blocs sont "rétro-liés", chacun se référant au bloc précédent dans la chaîne. La chaîne de blocs est souvent visualisée comme une pile verticale, avec des blocs superposés les uns sur les autres et le premier bloc servant de base à la pile. La visualisation de blocs empilés les uns sur les autres entraîne l'utilisation de termes tels que "hauteur" pour désigner la distance par rapport au premier bloc, et "sommet" ou "pointe" pour désigner le bloc le plus récemment ajouté.

("bloc de genèse")Chaque bloc de la chaîne de blocs est identifié par un hachage, généré à l'aide de l'algorithme de hachage cryptographique SHA256 sur l'en-tête du bloc. Chaque bloc fait également référence à un bloc précédent, connu sous le nom de bloc *parent*, via le champ "hachage de bloc précédent" dans l'en-tête de bloc. En d'autres termes, chaque bloc contient le hachage de son parent dans sa propre en-tête. La séquence de hachages reliant chaque bloc à son parent crée une chaîne remontant jusqu'au premier bloc jamais créé, connu sous le nom de *bloc de genèse*.

Bien qu'un bloc n'ait qu'un seul parent, il peut temporairement avoir plusieurs enfants. Chacun des enfants fait référence au même bloc que son parent et contient le même hachage (parent) dans le champ "hachage du bloc précédent". Plusieurs enfants apparaissent lors d'une "fourche" de blockchain, une situation temporaire qui se produit lorsque différents blocs sont découverts presque simultanément par différents mineurs (voir [???](#forks)). Finalement, un seul bloc enfant devient une partie de la chaîne de blocs et la "fourche" est résolu. Même si un bloc peut avoir plusieurs enfants, chaque bloc ne peut avoir qu'un seul parent. En effet, un bloc a un seul champ "hachage de bloc précédent" faisant référence à son parent unique.

Le champ "hachage du bloc précédent" se trouve à l'intérieur de l'en-tête du bloc et affecte ainsi le hachage du bloc *courant*. La propre identité de l'enfant change si l'identité du parent change. Lorsque le parent est modifié de quelque manière que ce soit, le hachage du parent change. Le hachage modifié du parent nécessite une modification du pointeur "hachage de bloc précédent" de l'enfant. Cela entraîne à son tour la modification du hachage de l'enfant, ce qui nécessite une modification du pointeur du petit-enfant, qui à son tour modifie le petit-enfant, et ainsi de suite. Cet effet de cascade garantit qu'une fois qu'un bloc est suivi de plusieurs générations, il ne peut pas être modifié sans forcer un recalcul de tous les blocs suivants. Parce qu'un tel recalcul nécessiterait un calcul énorme (et donc une consommation d'énergie), l'existence d'une longue chaîne de blocs rend immuable l'historique profond de la chaîne de blocs, ce qui est une caractéristique clé de la sécurité du bitcoin.

Une façon de penser à la chaîne de blocs est comme des couches dans une formation géologique ou un échantillon de carotte de glacier. Les couches de surface peuvent changer avec les saisons, voire être soufflées avant d'avoir le temps de s'installer. Mais dès que l'on descend à quelques centimètres de profondeur, les couches géologiques deviennent de plus en plus stables. Au moment où vous regardez quelques centaines de mètres plus bas, vous regardez un instantané du passé qui est resté intact pendant des millions d'années. Dans la chaîne de blocs, les quelques blocs les plus récents peuvent être révisés s'il y a un recalcul en chaîne dû à une fourche. Les six premiers blocs sont comme quelques centimètres de terre arable. Mais une fois que vous allez plus profondément dans la chaîne de blocs, au-delà de six blocs, les blocs sont de moins en moins susceptibles de changer. Après 100 blocs de retour, il y a tellement de stabilité que la transaction coinbase - la transaction contenant le bitcoin nouvellement extrait - peut être dépensée. Quelques milliers de blocs en arrière (un mois) et la chaîne de blocs est une histoire établie, à toutes fins pratiques. Alors que le protocole permet toujours à une chaîne d'être annulée par une chaîne plus longue et alors que la possibilité qu'un bloc soit inversé existe toujours, la probabilité d'un tel événement diminue au fil du temps jusqu'à ce qu'elle devienne infinitésimale.

## Structure d'un bloc

Un bloc est une structure de données de conteneur qui agrège les transactions pour les inclure dans le grand livre public, la chaîne de blocs. Le bloc est composé d'un en-tête, contenant des métadonnées, suivi d'une longue liste de transactions qui constituent l'essentiel de sa taille. L'en-tête de bloc est de 80 octets, alors que la transaction moyenne est d'au moins 400 octets et que le bloc moyen contient plus de 1900 transactions. Un bloc complet, avec toutes les transactions, est donc 10 000 fois plus grand que l'en-tête du bloc. [table\_title](#block_structure1) décrit la structure d'un bloc.

La structure d'un bloc

| Taille | Champ | La description |
| --- | --- | --- |
| 4 octets | Taille de bloc | La taille du bloc, en octets, suivant ce champ |
| 80 octets | En-tête de bloc | Plusieurs champs forment l'en-tête du bloc |
| 1–9 octets (VarInt) | Compteur de transactions | Combien de transactions suivent |
| variables | Transactions | Les transactions enregistrées dans ce bloc |

## En-tête de bloc

L'en-tête de bloc se compose de trois ensembles de métadonnées de bloc. Tout d'abord, il y a une référence à un hachage de bloc précédent, qui connecte ce bloc au bloc précédent dans la chaîne de blocs. Le deuxième ensemble de métadonnées, à savoir *difficulty*, *timestamp* et *nonce*, concerne la concurrence minière, comme détaillé dans [???](#mining). La troisième métadonnée est la racine de l'arbre Merkle, une structure de données utilisée pour résumer efficacement toutes les transactions du bloc. [table\_title](#block_header_structure_ch09) décrit la structure d'un en-tête de bloc.

La structure de l'entête du bloc

| Taille | Champ | La description |
| --- | --- | --- |
| 4 octets | Version | Un numéro de version pour suivre les mises à niveau du logiciel/protocole |
| 32 octets | Hachage de bloc précédent | Une référence au hachage du bloc précédent (parent) dans la chaîne |
| 32 octets | Racine de Merkel | Un hachage de la racine de l'arbre Merkle des transactions de ce bloc |
| 4 octets | Horodatage | Le temps de création approximatif de ce bloc (en secondes écoulées depuis Unix Epoch) |
| 4 octets | Cible de difficulté | La cible de difficulté de l'algorithme Proof-of-Work (ou Preuve de travail) pour ce bloc |
| 4 octets | Nonce | Un compteur utilisé pour l'algorithme de preuve de travail |

Le nonce, la cible de difficulté et l'horodatage sont utilisés dans le processus d'extraction et seront discutés plus en détail dans [???](#mining).

## Identificateurs de bloc : hachage d'en-tête de bloc et hauteur de bloc

L'identifiant primaire d'un bloc est son hachage cryptographique, une empreinte digitale, réalisée en hachant deux fois l'en-tête du bloc via l'algorithme SHA256. Le hachage de 32 octets résultant est appelé *hachage de bloc* mais est plus précisément le *hachage d'en-tête de bloc*, car seul l'en-tête de bloc est utilisé pour le calculer. Par exemple, 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f est le hachage de bloc du premier bloc bitcoin jamais créé. Le hachage de bloc identifie un bloc de manière unique et sans ambiguïté et peut être dérivé indépendamment par n'importe quel nœud en double hachant simplement l'en-tête de bloc avec l'algorithme SHA256.

Notez que le hachage de bloc n'est pas réellement inclus dans la structure de données du bloc, ni lorsque le bloc est transmis sur le réseau, ni lorsqu'il est stocké sur le stockage de persistance d'un nœud dans le cadre de la chaîne de blocs. Au lieu de cela, le hachage du bloc est calculé par chaque nœud lorsque le bloc est reçu du réseau. Le hachage de bloc peut être stocké dans une table de base de données distincte dans le cadre des métadonnées du bloc, pour faciliter l'indexation et une récupération plus rapide des blocs à partir du disque.

Une deuxième façon d'identifier un bloc est par sa position dans la chaîne de blocs, appelée le hauteur de bloc . Le premier bloc jamais créé est à la hauteur de bloc 0 (zéro) et est le même bloc qui était précédemment référencé par le hachage de bloc suivant 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f. Un bloc peut ainsi être identifié de deux manières : en référençant le hachage du bloc ou en référençant la hauteur du bloc. Chaque bloc suivant ajouté "au-dessus" de ce premier bloc est une position "plus élevée" dans la chaîne de blocs, comme des boîtes empilées les unes sur les autres.

De plus, le terme *hauteur de bloc actuelle* indique la taille de la chaîne de blocs en blocs à un moment donné. Par exemple, la hauteur actuelle du bloc au 1er mars 2021 était d'environ 672 722, ce qui signifie qu'il y avait 672 722 blocs empilés au-dessus du premier bloc créé en janvier 2009.

Contrairement au hachage de bloc, la hauteur de bloc n'est pas un identifiant unique. Bien qu'un seul bloc ait toujours une hauteur de bloc spécifique et invariante, l'inverse n'est pas vrai : la hauteur de bloc n'identifie pas toujours un seul bloc. Deux ou plusieurs blocs peuvent avoir la même hauteur de bloc, en concurrence pour la même position dans la chaîne de blocs. Ce scénario est décrit en détail dans la section [???](#forks). La hauteur de bloc ne fait pas non plus partie de la structure de données du bloc ; il n'est pas stocké dans le bloc. Chaque nœud identifie dynamiquement la position (hauteur) d'un bloc dans la chaîne de blocs lorsqu'il est reçu du réseau Bitcoin. La hauteur de bloc peut également être stockée sous forme de métadonnées dans une table de base de données indexée pour une récupération plus rapide.

Le *hachage de bloc* d'un bloc identifie toujours un seul bloc de manière unique. Un bloc a également toujours une *hauteur de bloc* spécifique. Cependant, il n'est pas toujours vrai qu'une hauteur de bloc spécifique puisse identifier un seul bloc. Au contraire, deux blocs ou plus pourraient se disputer une seule position dans la chaîne de blocs.

## Le bloc d’origine (ou de genèse)

Le premier bloc de la chaîne de blocs est appelé le bloc d’origine (ou de genèse) et a été créé en 2009. Il est l'ancêtre commun de tous les blocs de la chaîne de blocs, ce qui signifie que si vous commencez à n'importe quel bloc et suivez la chaîne en arrière dans le temps, vous finirez par arriver au bloc de genèse.

Chaque nœud commence toujours par une chaîne de blocs d'au moins un bloc car le bloc de genèse est encodé de manière statique dans le logiciel client Bitcoin, de sorte qu'il ne peut pas être modifié. Chaque nœud "connaît" toujours le hachage et la structure du bloc de genèse, l'heure fixe à laquelle il a été créé et même la transaction unique qu'il contient. Ainsi, chaque nœud a le point de départ de la chaîne de blocs, une "racine" sécurisée à partir de laquelle construire une chaîne de blocs de confiance.

Voir le bloc de genèse encodé statiquement à l'intérieur du client Bitcoin Core, dans [*chainparams.cpp*](https://bit.ly/1x6rcwP).

Le hachage d'identifiant suivant appartient au bloc genesis :

000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

Vous pouvez rechercher ce hachage de bloc sur n'importe quel site web d'explorateur de blocs, tel que *Blockchain.com*, et vous trouverez une page décrivant le contenu de ce bloc, avec une URL contenant ce hachage :

<https://www.blockchain.com/btc/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f>

Utilisation du client de référence Bitcoin Core sur la ligne de commande :

$ bitcoin-cli getblock 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

{  
 "hash" : "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",  
 "confirmations" : 308321,  
 "size" : 285,  
 "height" : 0,  
 "version" : 1,  
 "merkleroot" : "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",  
 "tx" : [  
 "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"  
 ],  
 "time" : 1231006505,  
 "nonce" : 2083236893,  
 "bits" : "1d00ffff",  
 "difficulty" : 1.00000000,  
 "nextblockhash" : "00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048"  
}

Le bloc de genèse contient un message caché en son sein. L'entrée de transaction coinbase contient le texte "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks.". Ce message était destiné à offrir une preuve de la date la plus ancienne de création de ce bloc, en faisant référence au titre du journal britannique *The Times*. Il sert également de rappel ironique de l'importance d'un système monétaire indépendant, le lancement du bitcoin se produisant en même temps qu'une crise monétaire mondiale sans précédent. Le message a été intégré dans le premier bloc par Satoshi Nakamoto, le créateur de Bitcoin.

## Relier les blocs dans la chaîne de blocs

Les nœuds complets Bitcoin conservent une copie locale de la chaîne de blocs, à partir du bloc de genèse. La copie locale de la chaîne de blocs est constamment mise à jour au fur et à mesure que de nouveaux blocs sont trouvés et utilisés pour étendre la chaîne. Lorsqu'un nœud reçoit des blocs entrants du réseau, il validera ces blocs, puis les reliera à la blockchain existante. Pour établir un lien, un nœud examinera l'en-tête de bloc entrant et recherchera le "hachage de bloc précédent".

Supposons, par exemple, qu'un nœud ait 277 314 blocs dans la copie locale de la chaîne de blocs. Le dernier bloc dont le nœud a connaissance est le bloc 277 314, avec un hachage d'en-tête de bloc :

00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249

Le nœud Bitcoin reçoit alors un nouveau bloc du réseau, qu'il parse comme suit :

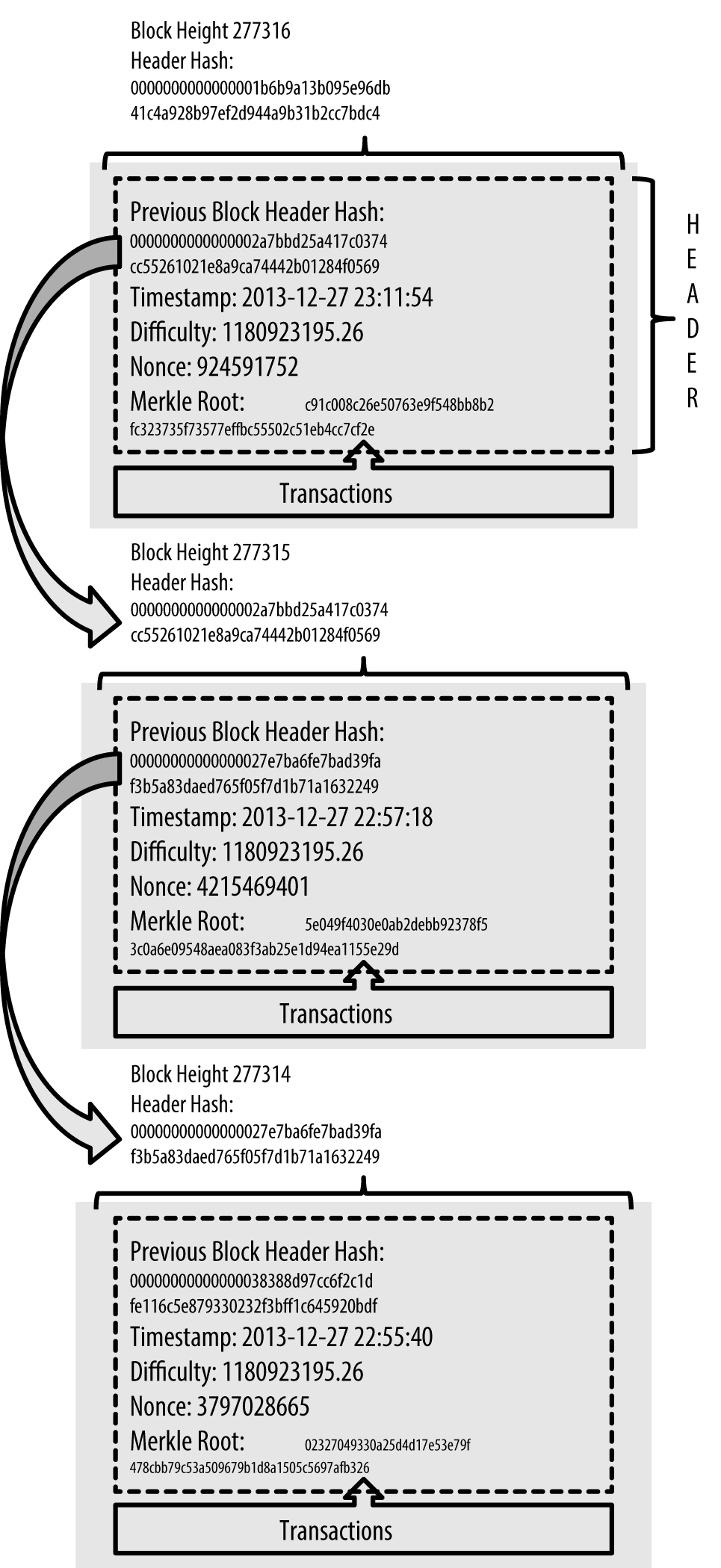
{  
 "size" : 43560,  
 "version" : 2,  
 "previousblockhash" :  
 "00000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",  
 "merkleroot" :  
 "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",  
 "time" : 1388185038,  
 "difficulty" : 1180923195.25802612,  
 "nonce" : 4215469401,  
 "tx" : [  
 "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",  
  
 #[... beaucoup plus de transactions omises ...]  
  
 "05cfd38f6ae6aa83674cc99e4d75a1458c165b7ab84725eda41d018a09176634"  
 ]  
}

En regardant ce nouveau bloc, le nœud trouve le champ previousblockhash, qui contient le hachage de son bloc parent. Il s'agit d'un hachage connu du nœud, celui du dernier bloc de la chaîne à la hauteur 277,314. Par conséquent, ce nouveau bloc est un enfant du dernier bloc de la chaîne et étend la blockchain existante. Le nœud ajoute ce nouveau bloc à la fin de la chaîne, rendant la blockchain plus longue avec une nouvelle hauteur de 277 315. [figure\_title](#chain_of_blocks) montre la chaîne de trois blocs, liés par des références dans le champ previousblockhash.

## Arbres de Merkle

Chaque bloc de la chaîne de blocs Bitcoin contient un résumé de tous les transactions dans le bloc à l'aide d'un *arbre de Merkle*.

Un *arbre de Merkle*, également appelé *arbre de hachage binaire*, est une structure de données utilisée pour résumer et vérifier efficacement l'intégrité de grands ensembles de données. Les arbres de Merkle sont des arbres binaires contenant des hachages cryptographiques. Le terme "arbre" est utilisé en informatique pour décrire une structure de données ramifiée, mais ces arbres sont généralement affichés à l'envers avec la "racine" en haut et les "feuilles" en bas d'un diagramme, comme vous le verrez dans les exemples qui suivent.



Blocs liés dans une chaîne par référence au hachage d'en-tête de bloc précédent

Les arbres de Merkle sont utilisés dans Bitcoin pour résumer toutes les transactions d'un bloc, produisant une empreinte numérique globale de l'ensemble des transactions, fournissant un processus très efficace pour vérifier si une transaction est incluse dans un bloc. Un arbre Merkle est construit en hachant récursivement des paires de nœuds jusqu'à ce qu'il n'y ait qu'un seul hachage, appelé *racine*, ou *racine de Merkle*. L'algorithme de hachage cryptographique utilisé dans les arbres de Merkle de Bitcoin est SHA256 appliqué deux fois, également connu sous le nom de double-SHA256.

Lorsque N éléments de données sont hachés et résumés dans un arbre de Merkle, vous pouvez vérifier si un élément de données est inclus dans l'arbre avec au plus 2\*log~2~(N) calculs, ce qui en fait une donnée très efficace structure.

L'arbre de Merkle est construit de bas en haut. Dans l'exemple suivant, nous commençons avec quatre transactions, A, B, C et D, qui forment les *feuilles* de l'arbre de Merkle, comme indiqué dans [figure\_title](#simple_merkle). Les transactions ne sont pas stockées dans l'arborescence de Merkle ; à la place, leurs données sont hachées et le hachage résultant est stocké dans chaque nœud feuille sous la forme HA, HB, HC et HD :

H

A

= SHA256(SHA256(Transaction A))

Des paires consécutives de nœuds feuilles sont ensuite résumées dans un nœud parent, en concaténant les deux hachages et en les hachant ensemble. Par exemple, pour construire le nœud parent HAB, les deux hachages de 32 octets des enfants sont concaténés pour créer une chaîne de 64 octets. Cette chaîne est ensuite doublement hachée pour produire le hachage du nœud parent :

H

AB

= SHA256(SHA256(H

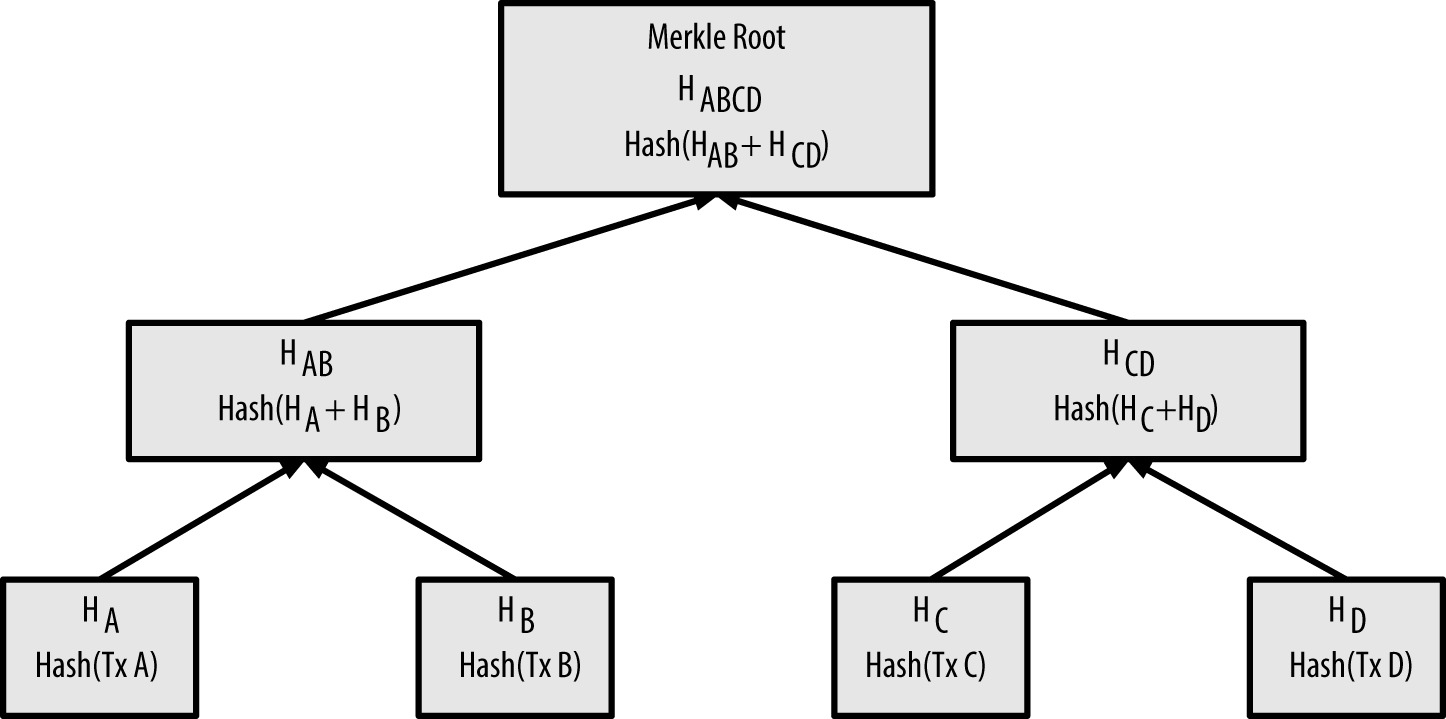
A

+ H

B

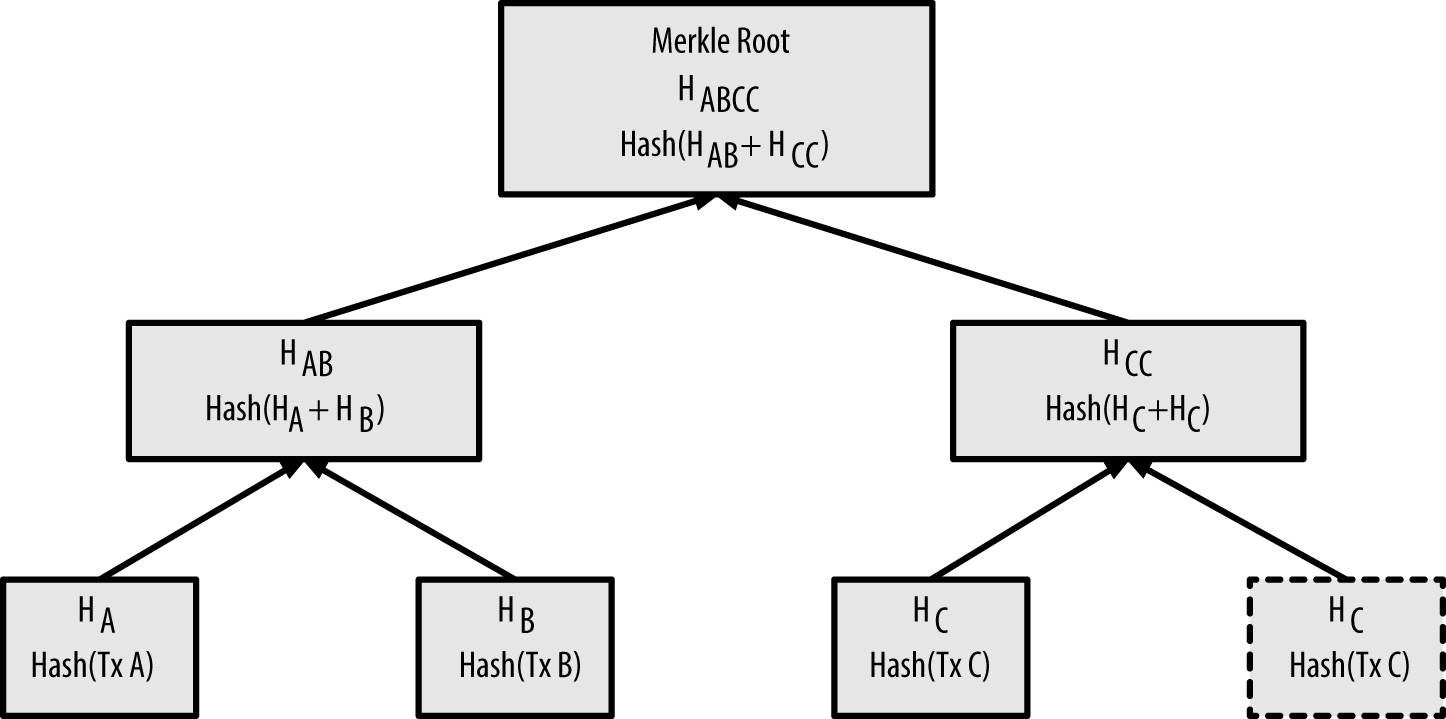
))

Le processus se poursuit jusqu'à ce qu'il n'y ait qu'un seul nœud au sommet, le nœud connu sous le nom de racine de Merkle. Ce hachage de 32 octets est stocké dans l'en-tête du bloc et résume toutes les données des quatre transactions. <<simple\_merkle> > montre comment la racine est calculée par hachages par paires des nœuds.



Calcul des nœuds dans un arbre de Merkle

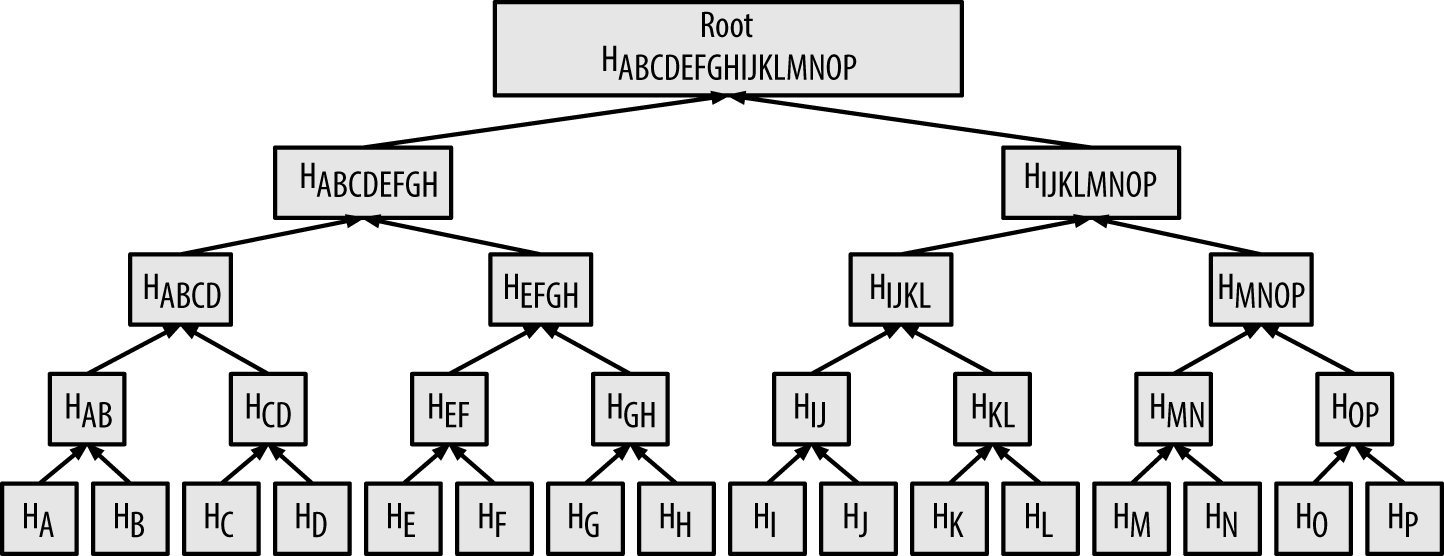
Parce que l'arbre de Merkle est un arbre binaire, il a besoin d'un nombre pair de nœuds feuilles. S'il y a un nombre impair de transactions à résumer, le dernier hachage de transaction sera dupliqué pour créer un nombre pair de nœuds feuilles, également appelés *arbre équilibré*. Ceci est montré dans [figure\_title](#merkle_tree_odd), où la transaction C est dupliquée.



La duplication d'un élément de données permet d'obtenir un nombre pair d'éléments de données

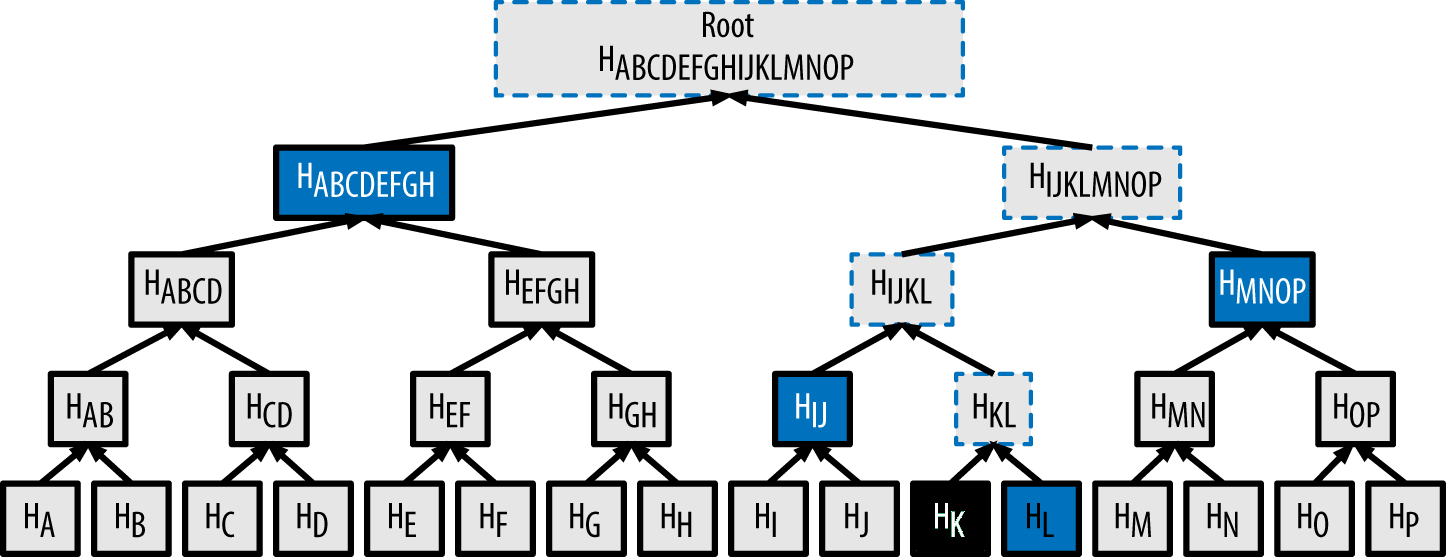
La même méthode de construction d'un arbre à partir de quatre transactions peut être généralisée pour construire des arbres de n'importe quelle taille. Dans le Bitcoin, il est courant d'avoir plusieurs centaines à plus d'un millier de transactions dans un seul bloc, qui sont résumées exactement de la même manière, produisant seulement 32 octets de données en tant que racine de Merkle unique. Dans [figure\_title](#merkle_tree_large), vous verrez un arbre construit à partir de 16 transactions. Notez que bien que la racine semble plus grande que les nœuds feuilles dans le diagramme, elle a exactement la même taille, seulement 32 octets. Qu'il y ait une transaction ou cent mille transactions dans le bloc, la racine de Merkle les résume toujours en 32 octets.

Pour prouver qu'une transaction spécifique est incluse dans un bloc, un nœud n'a besoin que de produire log~2~(N) hachages de 32 octets, constituant un *chemin d'authentification* ou *chemin de Merkle* se connectant la transaction spécifique à la racine de l'arborescence. Ceci est particulièrement important lorsque le nombre de transactions augmente, car le logarithme en base 2 du nombre de transactions augmente beaucoup plus lentement. Cela permet aux nœuds Bitcoin de produire efficacement des chemins de 10 ou 12 hachages (320 à 384 octets), qui peuvent fournir la preuve d'une seule transaction parmi plus de mille transactions dans un bloc de la taille d'un mégaoctet.



Un arbre Merkle résumant de nombreux éléments de données

Dans [figure\_title](#merkle_tree_path), un nœud peut prouver qu'une transaction K est incluse dans le bloc en produisant un chemin de Merkle qui ne fait que quatre hachages de 32 octets (128 octets au total). Le chemin se compose des quatre hachages (affichés avec un arrière-plan ombré dans [figure\_title](#merkle_tree_path)) HL, HIJ, HMNOP et HABCDEFGH. Avec ces quatre hachages fournis comme chemin d'authentification, n'importe quel nœud peut prouver que HK (avec un fond noir au bas du diagramme) est inclus dans la racine de Merkle en calculant quatre hachages supplémentaires par paires HKL, HIJKL, HIJKLMNOP et la racine de l'arbre de Merkle (indiquée par une ligne pointillée dans le diagramme).



Un chemin (ou trajet) de Merkle utilisé pour prouver l'inclusion d'un élément de données

Le code dans [example\_title](#merkle_example) illustre le processus de création d'un arbre de Merkle depuis les hachages de nœud feuille jusqu'à la racine, en utilisant la bibliothèque libbitcoin pour certaines fonctions d'assistance.

#include <bitcoin/bitcoin.hpp>  
  
bc::hash\_digest create\_merkle(bc::hash\_list& merkle)  
{  
 // Stop if hash list is empty.  
 if (merkle.empty())  
 return bc::null\_hash;  
 else if (merkle.size() == 1)  
 return merkle[0];  
  
 // While there is more than 1 hash in the list, keep looping...  
 while (merkle.size() > 1)  
 {  
 // If number of hashes is odd, duplicate last hash in the list.  
 if (merkle.size() % 2 != 0)  
 merkle.push\_back(merkle.back());  
 // List size is now even.  
 assert(merkle.size() % 2 == 0);  
  
 // New hash list.  
 bc::hash\_list new\_merkle;  
 // Loop through hashes 2 at a time.  
 for (auto it = merkle.begin(); it != merkle.end(); it += 2)  
 {  
 // Join both current hashes together (concatenate).  
 bc::data\_chunk concat\_data(bc::hash\_size \* 2);  
 auto concat = bc::serializer<  
 decltype(concat\_data.begin())>(concat\_data.begin());  
 concat.write\_hash(\*it);  
 concat.write\_hash(\*(it + 1));  
 // Hash both of the hashes.  
 bc::hash\_digest new\_root = bc::bitcoin\_hash(concat\_data);  
 // Add this to the new list.  
 new\_merkle.push\_back(new\_root);  
 }  
 // This is the new list.  
 merkle = new\_merkle;  
  
 // DEBUG output -------------------------------------  
 std::cout << "Current merkle hash list:" << std::endl;  
 for (const auto& hash: merkle)  
 std::cout << " " << bc::encode\_base16(hash) << std::endl;  
 std::cout << std::endl;  
 // --------------------------------------------------  
 }  
 // Finally we end up with a single item.  
 return merkle[0];  
}  
  
int main()  
{  
 // Replace these hashes with ones from a block to reproduce the same merkle root.  
 bc::hash\_list tx\_hashes{{  
 bc::hash\_literal("0000000000000000000000000000000000000000000000000000000000000000"),  
 bc::hash\_literal("0000000000000000000000000000000000000000000000000000000000000011"),  
 bc::hash\_literal("0000000000000000000000000000000000000000000000000000000000000022"),  
 }};  
 const bc::hash\_digest merkle\_root = create\_merkle(tx\_hashes);  
 std::cout << "Result: " << bc::encode\_base16(merkle\_root) << std::endl;  
 return 0;  
}

[example\_title](#merkle_example_run) montre le résultat de la compilation et de l'exécution du code de Merkle.

# Compiler le code merkle.cpp  
$ g++ -o merkle merkle.cpp $(pkg-config --cflags --libs libbitcoin)  
# Exécutez l&#39;exécutable merkle  
$ ./merkle  
Liste de hachage Merkle actuelle :  
 32650049a0418e4380db0af81788635d8b65424d397170b8499cdc28c4d27006  
 30861db96905c8dc8b99398ca1cd5bd5b84ac3264a4e1b3e65afa1bcee7540c4  
  
Liste de hachage Merkle actuelle :  
 d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3  
  
Result: d47780c084bad3830bcdaf6eace035e4c6cbf646d103795d22104fb105014ba3

L'efficacité des arbres de Merkle devient évidente à mesure que l'échelle augmente. [table\_title](#block_structure2) montre la quantité de données qui doit être échangée sous forme de chemin de Merkle pour prouver qu'une transaction fait partie d'un bloc.

Efficacité de l'arbre de Merkle

| Nombre de transactions | Environ. taille du bloc | Taille du chemin (hachages) | Taille du chemin (octets) |
| --- | --- | --- | --- |
| 16 opérations | 4 kilo-octets | 4 hachages | 128 octets |
| 512 opérations | 128 kilo-octets | 9 hachages | 288 octets |
| 2048 opérations | 512 kilo-octets | 11 hachages | 352 octets |
| 65 535 opérations | 16 mégaoctets | 16 hachages | 512 octets |

Comme vous pouvez le voir dans le tableau, alors que la taille de bloc augmente rapidement, de 4 Ko avec 16 transactions à une taille de bloc de 16 Mo pour 65 535 transactions, le chemin de Merkle requis pour prouver l'inclusion d'une transaction augmente beaucoup plus lentement, de 128 octets à seulement 512 octets. Avec les arbres de Merkle, un nœud peut télécharger uniquement les en-têtes de bloc (80 octets par bloc) et être toujours en mesure d'identifier l'inclusion d'une transaction dans un bloc en récupérant un petit chemin Merkle à partir d'un nœud complet, sans stocker ni transmettre la grande majorité des blockchain, qui peut avoir une taille de plusieurs gigaoctets. Les nœuds qui ne maintiennent pas une chaîne de blocs complète, appelés nœuds de vérification simplifiée des paiements (SPV), utilisent des chemins de Merkle pour vérifier les transactions sans télécharger des blocs complets.

## Arbres de Merkle et vérification simplifiée des paiements (SPV)

Les arbres Merkle sont largement utilisés par les nœuds SPV. Les nœuds SPV n'ont pas toutes les transactions et ne téléchargent pas les blocs complets, juste les en-têtes de bloc. Afin de vérifier qu'une transaction est incluse dans un bloc, sans avoir à télécharger toutes les transactions du bloc, ils utilisent un chemin d'authentification, ou chemin de Merkle.

Considérons, par exemple, un nœud SPV qui est intéressé par les paiements entrants à une adresse contenue dans son portefeuille. Le nœud SPV établira un filtre de floraison (voir [???](#bloom_filters)) sur ses connexions à des pairs pour limiter les transactions reçues uniquement à celles contenant des adresses d'intérêt. Lorsqu'un pair voit une transaction qui correspond au filtre de floraison, il enverra ce bloc à l'aide d'un message merkleblock. Le message merkleblock contient l'en-tête du bloc ainsi qu'un chemin de Merkle qui relie la transaction d'intérêt à la racine de Merkle dans le bloc. Le nœud SPV peut utiliser ce chemin de Merkle pour connecter la transaction au bloc et vérifier que la transaction est incluse dans le bloc. Le nœud SPV utilise également l'en-tête de bloc pour lier le bloc au reste de la chaîne de blocs. La combinaison de ces deux liens, entre la transaction et le bloc, et entre le bloc et la chaîne de blocs, prouve que la transaction est enregistrée dans la chaîne de blocs. Au total, le nœud SPV aura reçu moins d'un kilo-octet de données pour l'en-tête de bloc et le chemin de Merkle, une quantité de données plus de mille fois inférieure à un bloc complet (environ 1 mégaoctet actuellement).

## Chaînes de blocs de test de Bitcoin

Vous pourriez être surpris d'apprendre qu'il existe plus d'une chaîne de blocs Bitcoin. La chaîne de blocs Bitcoin "principale", celle créée par Satoshi Nakamoto le 3 janvier 2009, celle avec le bloc de genèse que nous avons étudié dans ce chapitre, s'appelle *mainnet*. Il existe d'autres chaînes de blocs Bitcoin qui sont utilisées à des fins de test : actuellement *testnet*, *segnet* et *regtest*. Regardons chacun à son tour.

### Testnet—Le terrain de test de Bitcoin

Testnet est le nom de la blockchain, du réseau et de la devise de test utilisés à des fins de test. Le testnet est un réseau P2P en direct complet, avec des portefeuilles, des bitcoins de test (pièces de testnet), l'exploitation minière et toutes les autres fonctionnalités du réseau principal. Il n'y a vraiment que deux différences : les pièces testnet sont censées être sans valeur et la difficulté d'extraction doit être suffisamment faible pour que n'importe qui puisse extraire des pièces testnet relativement facilement (en les gardant sans valeur).

Tout développement logiciel destiné à une utilisation en production sur le réseau principal de Bitcoin doit d'abord être testé sur testnet avec des pièces de test. Cela protège à la fois les développeurs des pertes financières dues aux bogues et le réseau des comportements involontaires dus aux bogues.

Garder les pièces sans valeur et l'exploitation minière facile, cependant, n'est pas facile. Malgré les appels des développeurs, certaines personnes utilisent des équipements de minage avancés (GPU et ASIC) pour miner sur testnet. Cela augmente la difficulté, rend impossible l'extraction avec un processeur et rend finalement assez difficile l'obtention de pièces de test pour que les gens commencent à les évaluer, de sorte qu'elles ne sont pas sans valeur. En conséquence, de temps en temps, le testnet doit être supprimé et redémarré à partir d'un nouveau bloc de genèse, réinitialisant la difficulté.

Le testnet actuel s'appelle *testnet3*, la troisième itération de testnet, redémarré en février 2011 pour réinitialiser la difficulté du testnet précédent.

Gardez à l'esprit que testnet3 est une grande blockchain, dépassant 25 Go en 2021. Il faudra environ un jour pour se synchroniser complètement et utiliser les ressources de votre ordinateur. Pas autant que le réseau principal, mais pas exactement "léger" non plus. Une bonne façon d'exécuter un nœud testnet est d'utiliser une image de machine virtuelle (par exemple, VirtualBox, Docker, Cloud Server, etc.) dédiée à cet effet.

#### Utiliser testnet

Bitcoin Core, comme presque tous les autres logiciels bitcoin, prend entièrement en charge le fonctionnement sur testnet au lieu du réseau principal. Toutes les fonctions de Bitcoin Core fonctionnent sur testnet, y compris le portefeuille, l'extraction de pièces testnet et la synchronisation d'un nœud testnet complet.

Pour démarrer Bitcoin Core sur testnet au lieu du réseau principal, vous utilisez le commutateur testnet :

$ bitcoind -testnet

Dans les journaux, vous devriez voir que bitcoind construit une nouvelle blockchain dans le sous-répertoire testnet3 du répertoire bitcoind par défaut :

bitcoind : Utilisation du répertoire de données /home/username/.bitcoin/testnet3

Pour vous connecter à bitcoind, vous utilisez l'outil de ligne de commande bitcoin-cli, mais vous devez également le passer en mode testnet :

$ bitcoin-cli -testnet getblockchaininfo  
{  
 "chain": "test",  
 "blocks": 1088,  
 "headers": 139999,  
 "bestblockhash": "0000000063d29909d475a1c4ba26da64b368e56cce5d925097bf3a2084370128",  
 "difficulty": 1,  
 "mediantime": 1337966158,  
 "verificationprogress": 0.001644065914099759,  
 "chainwork": "0000000000000000000000000000000000000000000000000000044104410441",  
 "pruned": false,  
 "softforks": [  
  
 [...]

Vous pouvez également exécuter sur testnet3 avec d'autres implémentations de nœud complet, telles que btcd (écrit en Go) et bcoin (écrit en JavaScript), pour expérimenter et apprendre dans d'autres langages et environnements de programmation.

En 2021, testnet3 prend en charge toutes les fonctionnalités du réseau principal, y compris le témoin séparé (voir [???](#segwit)). Par conséquent, testnet3 peut également être utilisé pour tester les fonctionnalités de témoin séparé.

### Segnet—Le réseau de test des témoins séparés

En 2016, un testnet à usage spécial a été lancé pour aider au développement et aux tests de Segregated Witness ou Témoin séparé (alias segwit; voir [???](#segwit)). Cette chaîne de blocs de test s'appelle segnet et peut être rejointe en exécutant une version spéciale (branche) de Bitcoin Core.

Depuis que segwit a été ajouté à testnet3, il n'est plus nécessaire d'utiliser segnet pour tester les fonctionnalités de segwit.

À l'avenir, il est probable que nous verrons d'autres chaînes de blocs testnet spécialement conçues pour tester une seule fonctionnalité ou un changement architectural majeur, comme segnet.

### Regtest—La chaîne de blocs locale

Regtest, qui signifie "Regression Testing", (ou tests de régressions) est une fonctionnalité de Bitcoin Core qui vous permet de créer une chaîne de blocs locale à des fins de test. Contrairement à testnet3, qui est une blockchain de test publique et partagée, les chaînes de blocs regtest sont destinées à être exécutées comme des systèmes fermés pour les tests locaux. Vous lancez une chaîne de blocs regtest à partir de zéro, en créant un bloc de genèse local. Vous pouvez ajouter d'autres nœuds au réseau ou l'exécuter avec un seul nœud uniquement pour tester le logiciel Bitcoin Core.

Pour démarrer Bitcoin Core en mode regtest, vous utilisez le drapeau regtest :

$ bitcoind -regtest

Tout comme avec testnet, Bitcoin Core initialisera une nouvelle blockchain sous le sous-répertoire *regtest* de votre répertoire par défaut bitcoind :

bitcoind: Using data directory /home/username/.bitcoin/regtest

Pour utiliser l'outil de ligne de commande, vous devez également spécifier l'indicateur regtest. Essayons la commande getblockchaininfo pour inspecter la blockchain regtest :

$ bitcoin-cli -regtest getblockchaininfo  
{  
 "chain": "regtest",  
 "blocks": 0,  
 "headers": 0,  
 "bestblockhash": "0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206",  
 "difficulty": 4.656542373906925e-10,  
 "mediantime": 1296688602,  
 "verificationprogress": 1,  
 "chainwork": "0000000000000000000000000000000000000000000000000000000000000002",  
 "pruned": false,  
 [...]

Comme vous pouvez le voir, il n'y a pas encore de blocs. Exploitons-en quelques-uns (500 blocs) et gagnons la récompense :

$ bitcoin-cli -regtest generate 500  
[  
 "7afed70259f22c2bf11e406cb12ed5c0657b6e16a6477a9f8b28e2046b5ba1ca",  
 "1aca2f154a80a9863a9aac4c72047a6d3f385c4eec5441a4aafa6acaa1dada14",  
 "4334ecf6fb022f30fbd764c3ee778fabbd53b4a4d1950eae8a91f1f5158ed2d1",  
 "5f951d34065efeaf64e54e91d00b260294fcdfc7f05dbb5599aec84b957a7766",  
 "43744b5e77c1dfece9d05ab5f0e6796ebe627303163547e69e27f55d0f2b9353",  
 [...]  
 "6c31585a48d4fc2b3fd25521f4515b18aefb59d0def82bd9c2185c4ecb754327"  
]

Cela ne prendra que quelques secondes pour miner tous ces blocs, ce qui facilite certainement les tests. Si vous vérifiez le solde de votre portefeuille, vous verrez que vous avez gagné une récompense pour les 400 premiers blocs (les récompenses coinbase doivent avoir une profondeur de 100 blocs avant que vous puissiez les dépenser) :

$ bitcoin-cli -regtest getbalance  
12462.50000000

## Utiliser des chaînes de blocs de test pour le développement

Les différentes chaînes de blocs de Bitcoin (regtest, segnet, testnet3, mainnet) offrent une gamme d'environnements de test pour le développement de bitcoin. Utilisez les chaînes de blocs de test, que vous développiez pour Bitcoin Core ou un autre client de consensus à nœud complet ; une application telle qu'un portefeuille, un échange, un site de commerce électronique ; ou même développer de nouveaux contrats intelligents et des scripts complexes.

Vous pouvez utiliser les chaînes de blocs de test pour établir un pipeline de développement. Testez votre code localement sur un regtest au fur et à mesure que vous le développez. Une fois que vous êtes prêt à l'essayer sur un réseau public, passez à testnet pour exposer votre code à un environnement plus dynamique avec plus de diversité de code et d'applications. Enfin, une fois que vous êtes sûr que votre code fonctionne comme prévu, passez à mainnet pour le déployer en production. Au fur et à mesure que vous apportez des modifications, des améliorations, des corrections de bogues, etc., redémarrez le pipeline, en déployant d'abord chaque modification sur regtest, puis sur testnet, et enfin en production.