A szinkronizáció problémája a következő:

Tegyük fel van két fájl, A és B, amelyek két lassú kommunikációs kapcsolattal (slow communication link) összekötött gépen vannak. A-t úgy szeretnénk frissíteni, hogy tartalma megegyezzen B tartalmával. Ha A nagyméretű, akkor A másolása B-re lassú lesz. Gyorsabbá tétel érdekében A-t tömöríthetjük küldés előtt, de ez nem egy perfekt megoldás.

Tegyük fel, hogy A és B nagyon hasonló, mondjuk ugyanazon eredeti fájlból származnak. Felgyorsítás érdekében a hasonlóság valamilyen módon történő kihasználása lenne a logikus lépés. Gyakori módszer, hogy A és B közötti különbségeket küldik el egymásnak, majd ebből a listából a fájl rekonstruálásra kerül.

A probléma az ilyen módszerekkel, hogy a különbségek kialakításához olvashatónak kell lennie mindkét fájlnak, ami csak úgy érhető el, ha a kapcsolat egyik végén mindkét fájl rendelkezésre áll. Ha nem található meg egy eszközön mindkét fájl, akkor nem használhatók ezek a módszerek.

Mondhatjuk, hogy ez a probléma felvetés lokális adatbázis fájlok szinkronizálása esetén is fennáll.

A következő leírás célja olyan programok bemutatása, amelyek vagy kimondottan az ismertetett adatszinkronizálási probléma megoldására készültek, vagy valamilyen szinten közük van a témához. Ezen programok működése lesz feltüntetve az interneten található dokumentációk alapján.

rsync

Fájlszinkronizálási szoftver.

Úgynevezett delta-transzfer algoritmusáról híres, amely csökkenti a hálózaton keresztül küldött adatok mennyiségét azáltal, hogy csak a forrásfájlok és a célállomáson meglévő fájlok közötti különbségeket küldi el.

Az rsync az átvinni kívánt fájlokat alapértelmezett esetben egy "qucik check" algoritmus segítségével találja meg, amely a méretben vagy az utolsó módosítás idejében megváltozott fájlokat keresi.

A felvezetett szinkronizálási problémát próbálja az rsync megoldani a következő módon (hivatalos dokumentációból kiemelve és lefordítva):

Tegyük fel, hogy van két általános célú számítógépünk α és β. Az α számítógép hozzáfér az A fájlhoz, a β pedig a B fájlhoz, ahol A és B hasonló. Az α és β között lassú kommunikációs kapcsolat van.

Lépések: 1. β szétdarabolja a B fájlt nem átfedő, fix méretű blokkokra, melyek S byte méretűek (500 és 1000 byte közötti méret megfelelő legtöbb esetben).

2. Minden blokkhoz β kiszámít két ellenőrző összeget (checksum): egy gyenge 32 bites ’gördülő’ ellenőrző összeget (weak ’rolling’ 32bit checksum) és egy erős 128 bites MD4-es ellenőrző összeget.

3. β elküldi ezeket az összegeket α-nak.

4. α végignézi az A fájlt, hogy megtaláljon minden olyan S méretű blokkot, amelyeknek ugyanaz a gyenge és erős ellenőrző összege, mint B egyik blokkjának.

5. α elküld β-nak egy utasítássorozatot az A fájl másolatának előállításához. Minden utasítás vagy B fájl egy blokkjára való hivatkozás vagy szimpla adat. Szimpla adatot csak olyan A-beli részből küld, ami nem egyezett meg B egyik blokkjával sem.

Végeredményben β megkapja a kapcsolaton keresztül az A fájl másolatát, de csak A azon részeit, amelyek nem találhatók B-ben (illetve egy kevés adatot még, ami az ellenőrző összegeket és blokkok sorszámát tartalmazza). Az algoritmusnak csak egy körre van (one round trip) igénye a végrehajtáshoz, minimalizálva a kapcsolat késleltetésének a hatását (link latency).

A gyenge 'gördülő' ellenőrző összeg, erős MD4 ellenőrző összeg megtalálásának matematikai része nem kerül részletezésre.

OneDrive

A OneDrive szinkronizálási alkalmazás a Windows Push Notification Services (WNS) segítségével valós időben szinkronizálja a fájlokat. A WNS akkor értesíti a szinkronizáló alkalmazást, amikor a változás ténylegesen megtörténik, így elkerülhetők a felesleges lekérdezések és megtakarítható a felesleges számítási energia.

Működési lépések hivatalos dokumentáció alapján:

-Változás történik a Microsoft 365-ben.

-A WNS figyelmezteti a szinkronizáló alkalmazást a változásról. (képen 2. pont)

-A OneDrive hozzáadja azt az ún. Internal Server Changes Queue-hoz, azaz a belső változtatások sorához / listájához.

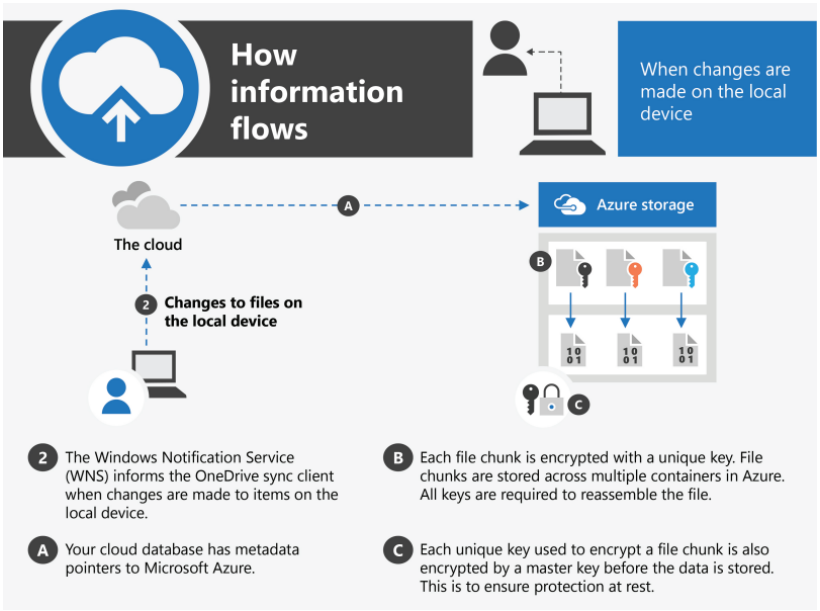
-Minden metaadatváltozás azonnal megtörténik, például fájlok törlése vagy átnevezése.

-Tartalom letöltése is elindít egy adott munkamenetet a klienssel.

-A Microsoft 365 metaadat pointer-ekkel irányítja a Microsoft Azure-on keresztül. (képen A. pont)

-A módosítások beérkezési sorrendben kerülnek feldolgozásra.

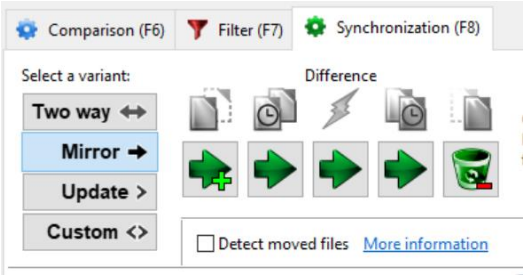
A korábbi 'OneDrive for Business' szinkronizálási alkalmazás egy lekérdező szolgáltatást használt a változások előre meghatározott ütemezés szerinti ellenőrzésére. A lekérdezés a rendszer késedelméhez és lassúságához vezethet, mivel nagy számítási teljesítményt igényel.



FreeFileSync

Az ún. RealTimeSync feladat, hogy minden alkalommal, amikor az ellenőrzött könyvtárak egyikében változásokat észlel, futtasson egy parancssort. Általában ez egy FreeFileSync batch munkát (? batch job) indít el.

A RealTimeSync közvetlenül az operációs rendszertől kapja a módosítási értesítéseket, hogy elkerülje a módosítások ismételt lekérdezésének teljesítményi költségét. Minden alkalommal, amikor egy fájl vagy mappa létrejön/frissül/törlődik a megfigyelt könyvtárakban vagy azok alkönyvtáraiban, a RealTimeSync megvárja, amíg eltelik a felhasználó által beállítható várakozás idő, ami alatt nem észlelt további változásokat, majd lefuttatja a parancssort. Ez biztosítja, hogy a szinkronizálás indításakor a felügyelt mappák ne legyenek használatban.



Az alkalmazás lehetővé tesz különböző szinkronizálási beállításokat.

Két alapvető szinkronizálási változat létezik attól függően, hogy egy mappapár hány mappáján dolgozik aktívan:

Ha mindkét mappa tartalmazza a munkafájlokat, és azt szeretné, hogy a változások (létrehozások, frissítések és törlések) mindkét irányba végbemenjenek, akkor a 'Two way' lehetőséget kell kiválasztani. Az irányok meghatározásához adatbázisfájlokra van szükség, amelyek az első szinkronizálás után automatikusan létrejönnek.

Ha csak az egyik mappa tartalmazza a munkafájlokat, a másik pedig a biztonsági másolatokat, akkor a 'Mirror' lehetőséget kell választani. A bal oldali mappa lesz a szinkronizálás forrása, a jobb oldali mappa pedig a cél.

A speciális szinkronizálási körülmények kezelése érdekében a mappák összehasonlítása után meghatározott kategóriák alapján egyéni – 'Custom' szabályok is beállíthatók.

Például az 'Update' opció pont egy ilyen beállításnak tekinthető: Olyan mint a 'Mirror' opció, de a fájlok törlésének elkerülésére van kialakítva.

Mélyebbre nyúló leírást nem tartalmazott a dokumentum, ez volt a legrelevánsabb információ amit találtam.

GoodSync

A hivatalos dokumentumban az alábbi információkat találtam a GoodSync szinkronizálási algoritmusával kapcsolatban:

„Ez egy általános vázlata az algoritmusnak (a tényleges algoritmus meglehetősen összetett, számos sajátos ötletet és üzleti titkot tartalmaz, amiket nem hozhatunk nyilvánosságra):

-Fájl aktuális állapota és bal oldalt tárolt fájl állapota közötti különbségek kiszámítása.

-Fájl aktuális állapota és jobb oldalt tárolt fájl állapota közötti különbségek kiszámítása.

-Bal oldalt észlelt változások propagálása a jobb oldalra.

-Jobb oldalt észlelt változások propagálása a bal oldalra.

-Ha a változás mindkét oldalon ugyanazt a fájlt érintette, deklarál egy Conflict műveletet.”

A bal és jobb oldal tulajdonképpen a két szinkronizálandó fájlt vagy mappát jelképezi.

Conflict művelet:

Ha a szinkronizált mappapár mindkét oldalán módosított fájlt, akkor a GoodSync nem tudja eldönteni, hogy melyik a preferált verzió (mivel nem tudja, hogy melyik változásokat kell megtartani és melyeket kell elhagyni), ilyenkor a GoodSync egy úgynevezett konfliktust deklarál.

Alapértelmezett esetben a Conflict nem egy másolási művelet. Ugyanakkor átállítható az állapota vagy LeftToRight vagy RighToLeft Copy-ra, így eldöntve, hogy mely változások a ’felsőbbrendűek’.

Kettő Conflict lehetőség például:

-Fájl mindkét oldalon törlésre vagy módosításra került. Megoldható konfliktus, ha a felhasználó kiválasztja a szinkronizálás irányát.

-A felhasználó mappát törölt a bal (jobb) oldalon és jobb (bal) oldalon pedig vagy törölte vagy módosította azt. Megoldható konfliktus, ha a felhasználó kiválasztja a szinkronizálás irányát.

Szinkronizációhoz közel álló fogalom a **verziókezelés.** A verziókezelés (angol megfelelői: version control, revision control, source control, source code management (leírás további részében előfordulhat ezen elnevezések használata)) szoftverfejlesztésben a forráskód módosításainak követésére és ellenőrzésére szolgáló folyamat. Néha a dokumentációs és konfigurációs fájlok karbantartására is használnak verziókezelő szoftvereket.

A változásokat általában egy számmal vagy betűkóddal azonosítják, amelyet revíziós / verzió számnak neveznek. Egy kezdeti fájlkészlet például az 1.verzió, első változtatáskor pedig 2.verzió lesz és így tovább. Minden egyes verzióhoz rendelhető egy időpont és egy személy / felhasználó, aki a módosítást elvégezte. A verziók összehasonlíthatók, visszaállíthatók, egyes fájltípusok esetén egyesíthetők is.

Tehát a verziókezelés egy adathalmaz időbeli változásait kezeli. Ezek a változások különböző felépítésűek lehetnek.

A hagyományos verziókezelő rendszerek központi modellt (centralized model) használnak, ahol az összes verziókezelő funkció együttesen megtalálható a megosztott szerveren. Ha két fejlesztő egyszerre szeretné ugyanazon fájlt módosítani, hozzáférés kezelése nélkül felülírnák egymás munkáját. Az ilyen rendszerek ezt a problémát két úgynevezett ’source management model’-lel, vagy

forráskezelési modellel próbálják megoldani: fájlzárolás (file locking) és verzió összevonás (version merging).

-File locking:

Legegyszerűbb megoldása egy fájl azonos idejű hozzáférésének a fájlok zárolása. Amikor egy fejlesztő 'bejelentkezik' egy fájlba, addig bárki olvashatja azt a fájlt, de nem módosíthatja, amíg az adott fejlesztő 'ki nem jelentkezik' abból a fájlból.

-Version merging:

Legtöbb verziókezelő rendszer több felhasználót is enged ugyanazon fájlok módosítására. Két fájl összevonása viszont bonyolult művelet is lehet, és itt lyukadunk ki a szinkronizáció kérdésénél, látható, hogy a verziókezelő rendszereknél is megjelenik ez a probléma.

Az elosztott verziókezelő rendszerek (distributed version control systems) a kliens-szerver megközelítéssel szemben az úgynevezett peer-to-peer megközelítést alkalmazzák. Egyetlen központi tároló helyett, amelyen a kliensek szinkronizálnak, minden fél a kódbázis egy másolatán dolgozik. Az elosztott revíziós kezelés a szinkronizációt a patch-ek (javítások / változtatások) peer-to-peer megosztásával / cseréjével végzi.

A programozás világában az egyik legelterjedtebb ilyen rendszer a **Git**. A Git egy ingyenes és nyílt forráskódú elosztott verziókezelő rendszer. Az alábbi sorok a Git verziókezelésének működését, logikáját fogom leírni hivatalos dokumentáció alapján.

A Git és bármely más VCS (verziókezelő rendszer) közötti fő különbség az, ahogyan a Git az adatokról gondolkodik. Legtöbb rendszer az adatokat fájlalapú változások listájaként tárolja, ezek a rendszerek ilyen adatokra úgy gondolnak, mint fájlok halamzára és fájlokban idővel végrehajtott változásokra.

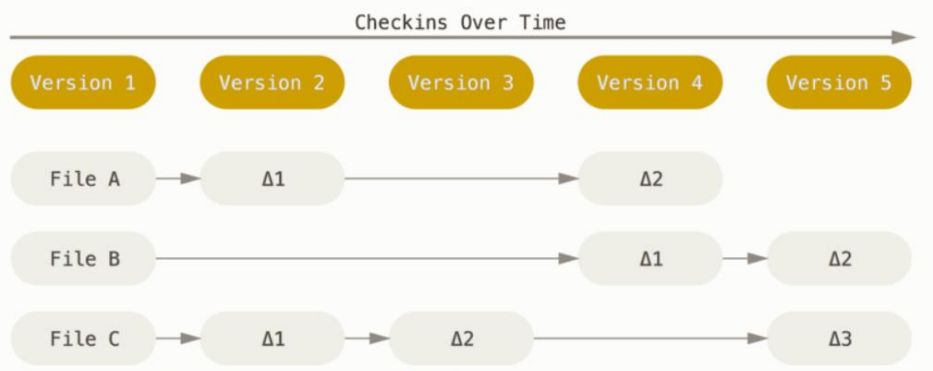


Figure 4. Storing data as changes to a base version of each file

A Git nem így tárolja az adatokat. A Git inkább az egyes verziókra úgy gondol, mint egy fájlrendszer 'pillanatképeire' (snapshot). Minden alkalommal, amikor commitolsz (snapshot készítés, nem tudok rá jó magyar szót), vagy elmented a projekted állapotát, a Git gyakorlatilag készít egy képet arról, ahogy a projekt összes fájlja az adott pillanatban kinéz és ennek a 'pillanatképnek' tárolja a referenciáját. Hatékonyság kedvéért, ha a fájlok nem változtak, a Git nem tárolja újra a fájlt, csak egy hivatkozást az előző megegyező fájlra, amit már egyszer eltárolt. Tehát a Git az adatokra inkább úgy tekint, mint pillanatképek sorozatára.

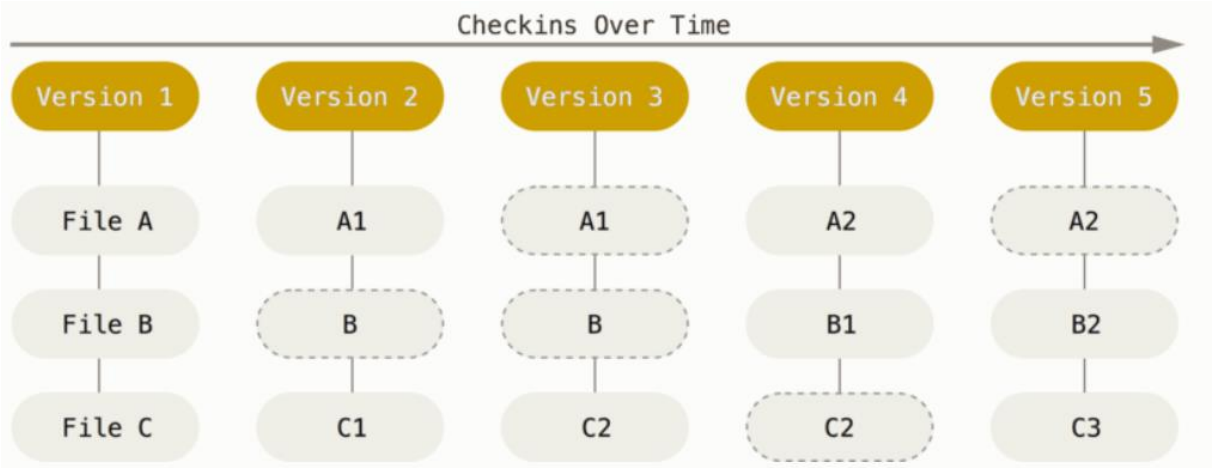


Figure 5. Storing data as snapshots of the project over time

Ez az gondolatmenet az adatok tárolásával kapcsolatban egy nagyon fontos megkülönböztetési szempont a Git és más verziókezelő rendszerek között.

A Git legtöbb műveletének csak helyi fájlokra és erőforrásokra van szüksége a működéshez - általában nincs szükség információra a hálózat egy másik számítógépéről. A projekt teljes előzménye a helyi lemezen van, a legtöbb művelet úgy tűnik, mintha azonnal elvégeződne. Ha egy fájl aktuális verziója és az egy hónappal ezelőtti verzió közötti változásokat szeretnénk látni, a Git képes megnézni a fájl egy hónappal ezelőtti verzióját, és elvégezni egy helyi különbségszámítást, ahelyett, hogy egy távoli szervert kellene megkérni erre, vagy a fájl egy régebbi verzióját kellene előhívni a távoli szerverről, hogy ezt helyben elvégezhessük. Ez azt jelenti, hogy nagyon kevés olyan dolog van, amit nem lehet offline megtenni.

A Git minden adatot ellenőrzőösszeggel lát el, mielőtt tárolja, és ezután az ellenőrzőösszeg (checksum) alapján hivatkozik rá, így lehetetlen megváltoztatni egy fájl anélkül, hogy a Git érzékelné a változást. A mechanizmus, amit a Git használ a checksum-hoz a SHA-1 hash. A Git az adatbázisába mindent a tartalom hash értékeként tárol, nem pedig a fájl neve alapján.

A fájlokat a Git három állapottal láthatja el: modified, staged és commited.

-Modified-nak nevezünk olyan fájlokat, amiket módosítottunk, de még nem commit-oltunk az adatbázisra.

-Staged azt jelenti, hogy a módosított fájl aktuális verzióját megjelölted, hogy bekerüljön a következő commit pillanatképbe.

-Commited azt jelenti, hogy az adat biztonságos tárolása a helyi adatbázison megtörtént.

Ebből adódóan egy Git projektnek három fő része van: working tree, staging area, Git directory.

A working tree a projekt egy verziójának ellenőrzésére szolgál. Ezek a fájlok a Git könyvtárban lévő tömörített adatbázisból kerülnek a lemezre, hogy használni vagy módosítani tudjuk őket.

A staging area egy olyan, általában a Git könyvtáradban található fájl, amely információkat tárol arról, hogy mi kerül a következő commitba.

A Git könyvtárban tárolja a Git a projektek metaadatait és objektum adatbázisát. Ez a Git legfontosabb része, és ez az, ami másoldóik, amikor egy másik eszközről egy repository-t / adattárolót klónoznak.

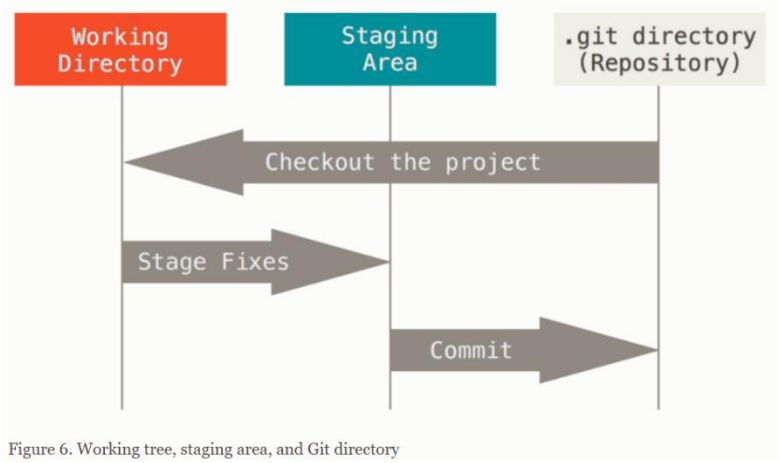


Figure 6. Working tree, staging area, and Git directory

A rész, amit szinkronizációnak tekinthetünk, az az úgynevezett 'merging', vagy összevonás / egyesítés. A Git ezt a következő módon végzi:

Egy meg nem említett fogalom: branch (ág) – nagyvonalakban, egy név, ami egy commit-ra mutat.

Amikor két branch szinkronizálására (merge) kérjük a Git-et akkor maga a Git generál egy commit-ot – a merge commit.

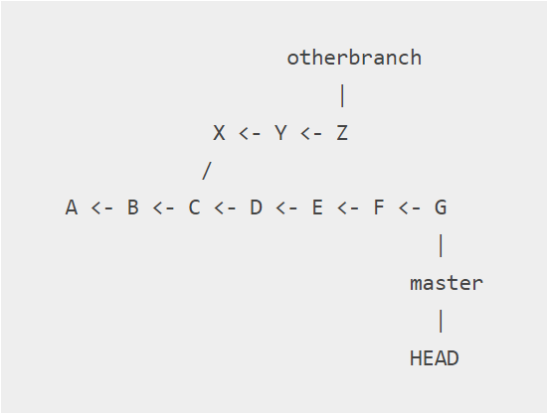
A merge commit generálása egy összetett folyamat de nagyvonalakban a folyamat a következő:

-Az egyesítés azzal kezdődik, hogy a Git megkeresi a közös commitot, ahol legutoljára tért el egymástól a két branch. Ezt nevezzük most egyesítési alapnak (merge base).

-A Git ezután két kiszámol két különbséget. A merge base-től az első ágig és a merge base-től a második ágig.

-Az egyesítési commit lérrehozásához a Git ezt a két különbséget alkalmazza a merge base-re.

Képszerűsítve:



A master-en vagyunk és a 'git merge otherbranch' paranccsal:

-Először a Git megállapítja, hogy a merge base a C.

-Ezután a Git kiszámítja a C és G (G – master) pont közötti különbséget, majd a C és Z (Z – otherbranch) közötti különbséget.

-Ezután a Git egyszerre alkalmazza ezeket a különbségeket C-re és az eredményt commit-olja a master-re. Ez az ún. merge commit, a folyamat pedig a logika, a Git szinkronizációjának működése mögött.