Üzleti Intelligencia

1. Előadás: Verziókezelés és dokumentálás

Kuknyó Dániel
Budapesti Gazdasági Egyetem

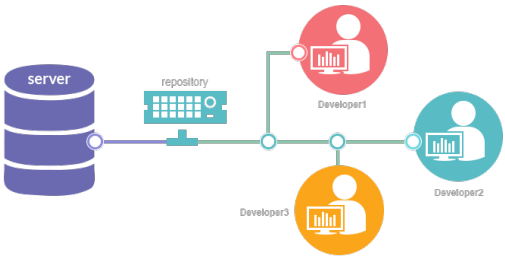
2023/24
1.félév

Tartalom

Verziókezelés alapjai

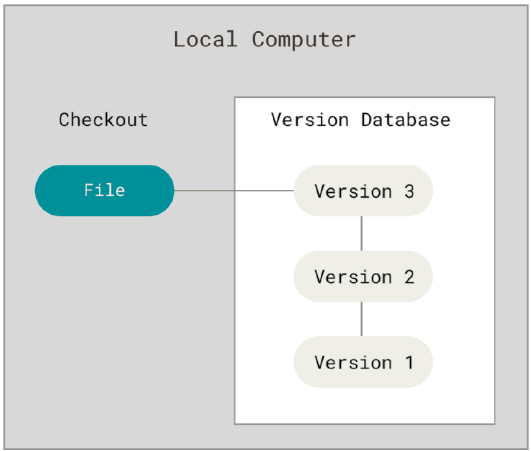
Miért van szükség verziókezelésre?

- A program változásainak követése
- A munka biztonságos elmentése
- Kollaboráció több fejlesztő között
- Programkód párhuzamos szerkesztése
- Feladatok szétosztása és követése



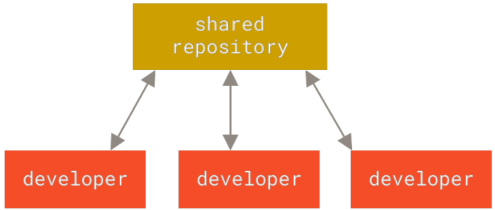
Lokális verziókezelők

A legegyszerűbb verziókezelés, ha a fejlesztő kézzel átmásol egy mappába fájlokat. Ezek lehetnek időbélyegzett mappák is, ha okos a fejlesztő. Ez a megoldás nagyon egyszerű, viszont fogékony a hibákra, mert sok a manuális munka. Ezenkívül sok benne a redundáns adat is, mert a nem változtatott adatot is el kell tárolni. Ezért hozták létre a lokális verziókezelőket, amik a teljes fájlok helyett csak a változtatásokat tárolták el egy külön erre kifejlesztett adatbázisban. Egy gyakori ilyen szoftver volt az RCS.



Centralizált verziókezelők

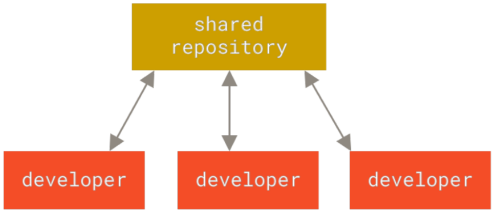
A következő jelentős probléma, amivel az emberek találkoznak, az az, hogy együtt kell működniük fejlesztőkkel más rendszereken. Ennek a problémának a kezelésére Központosított Verziókezelő Rendszerek (CVCS-ek) lettek kifejlesztve. Ezek a rendszerek (például CVS, Subversion és Perforce) egyetlen szerverrel rendelkeznek, amely tartalmazza az összes verziózott fájlt, és számos ügyfél, akik a fájlokat ebből a központi helyről töltik be.



Centralizált verziókezelők

Előnyei:

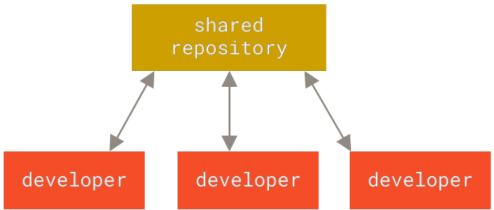
- Mindenki bizonyos mértékben tudja, hogy a projekt többi résztvevője mit csinál.
- Az adminisztrátorok részletes ellenőrzést gyakorolhatnak arról, hogy ki mit tehet meg, és sokkal könnyebb egy CVCS-t adminisztrálni, mint helyi adatbázisokkal foglalkozni minden kliens esetében.



Centralizált verziókezelők

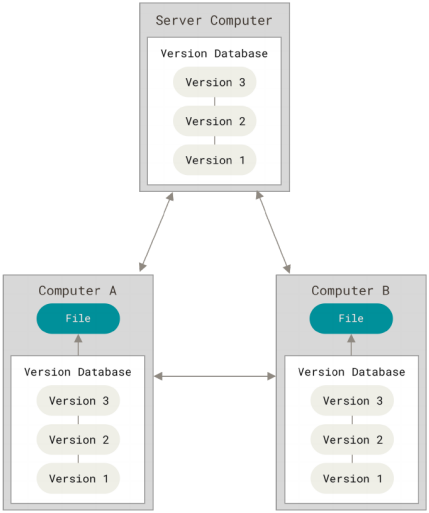
Hátrányai:

- Az egyetlen központi szerver hibapontot jelent, ahol akár egy órás leállás is lehetetlenné teszi a közös munkát és verziózási változtatások mentését.
- Adatvesztés veszélye, ha a központi adatbázis merevlemeze meghibásodik és nem rendelkezünk megfelelő biztonsági mentésekkel.



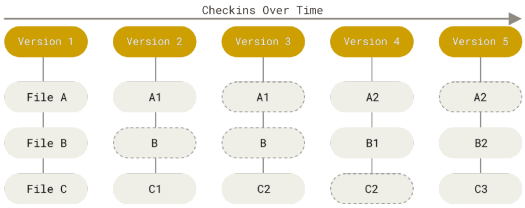
Elosztott verziókezelők

Ebben a helyzetben lépnek képbe az elosztott verziókezelő rendszerek (DVCS-ek). Egy DVCS-ben a kliensek nem csak a fájlok legfrissebb pillanatképét töltik le, hanem teljes egészében tükrözik a tárhelyet, beleértve annak teljes előzményeit is. Ezért minden klón valójában egy teljes adatbiztonsági másolat.
Sok ilyen rendszer nagyon jól kezeli a több távoli tárolóval való együttműködést, így lehetőség van különböző emberekkel egyidejűleg egyazon projekt keretein belül együttműködni.



A Git verziókezelő

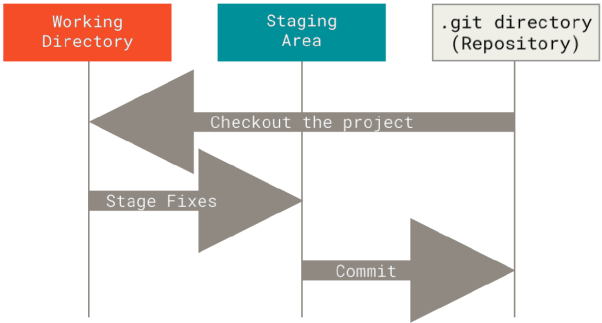
A Git úgy gondolkodik az adatokról, mint egy fájlrendszer pillanatképei.
Segítségével minden alkalommal, amikor commit történik, (azaz elmentődik a projekt) készít egy képet arról, hogy az összes fájl hogyan néz ki abban a pillanatban, és eltárol egy hivatkozást erre a pillanatfelvételre.
A hatékonyság érdekében, ha a fájlok nem változtak, a Git nem tárolja újra a fájlt, csak egy hivatkozást az előző, azonos fájlra, amit már tárolt.



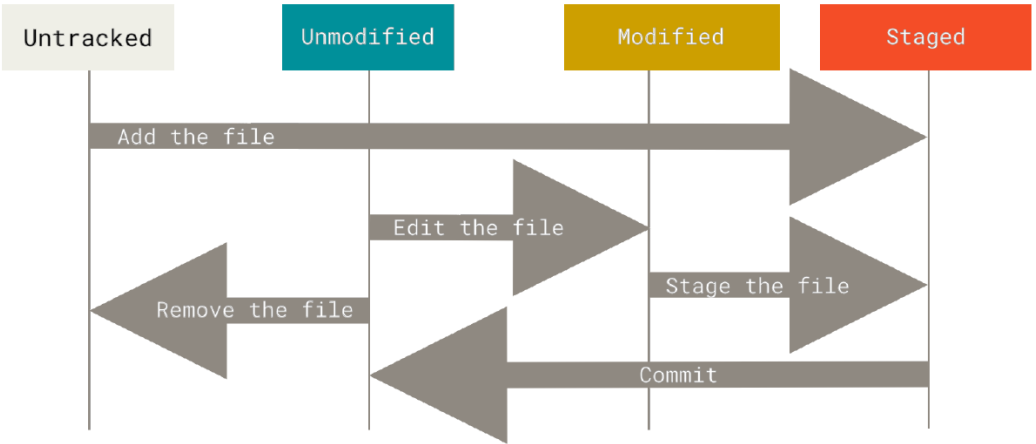
A három fájlállapot

A Git rendszerében három fő állapota van a fájloknak: módosított (modified), megjelölt (staged) és tárolt (committed):

- A módosított azt jelenti, hogy a fájl meg lett változtatva, de még nem lett tárolva, sem tárolásra megjelölve
- A megjelölt állapot azt jelenti, hogy a módosított fájl az aktuális verziójában meg lett jelölve, hogy a következő commit pillanatképbe kerüljön
- A tárolt azt jelenti, hogy az adat biztonságosan tárolva van a helyi adatbázisban

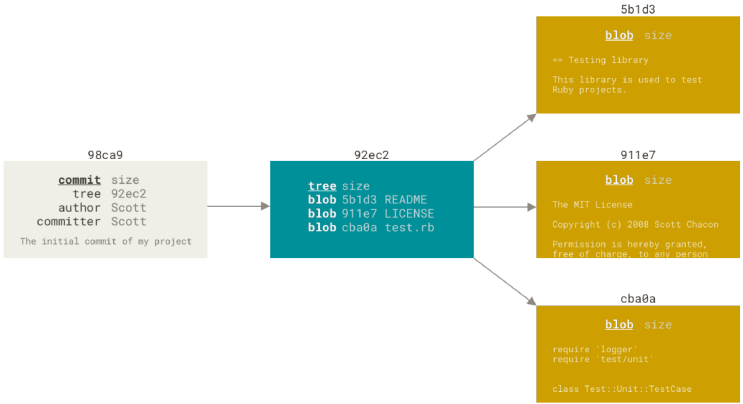


Státuszok változása



Commitok tárolása

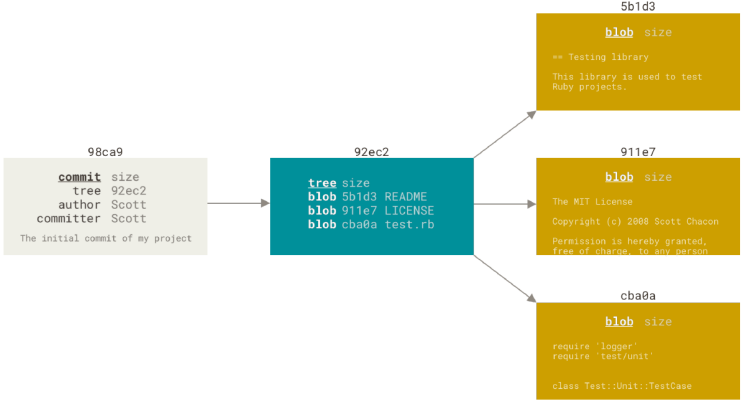
Amikor létrehozod a commitot a **git commit** futtatásával, a Git faobjektumként tárolja azokat az adattárházban. Ezután a Git létrehoz egy **commit objektumot**, amely a metaadatokat és egy mutatót tartalmaz a gyökérprojekt fához, így azt újra létre tudja hozni szükség esetén.



Commitok tárolása

A Git adattárház most öt objektumot tartalmaz:

- Három **blobot** (amelyek mindegyike egy-egy fájl tartalmát képviseli)
- Egy faobjektumot, amely felsorolja a könyvtár tartalmát és megadja, hogy mely fájlnevek tárolódnak mely blobokként
- Egy commitot a gyökérfa mutatójával és a metaadatokkal



Több commit tárolása

Ha néhány változtatás után ismét egy commit következik, a következő commit egy mutatót tárol arra a commitra, amely közvetlenül megelőzte.