Szakdolgozat

Csernai Csaba

Debrecen

2008

Debreceni Egyetem

Informatikai Kar

Verziókövető rendszerek

Témavezető:

Dr. Tornai Róbert

egyetemi adjunktus

Készítette:

Csernai Csaba

programozó matematikus

Debrecen

2008

Tartalom

1. Bevezetés

Szakdolgozatom célja, hogy bemutassa a verziókövető rendszerek általános jellemzőit, használatát. Ugyanakkor ismertetem néhány program hátterét, előnyét, hátrányát és végül összehasonlítom a legfontosabb szempontok szerint, ami egy fejlesztő csapatnál szóba jöhet.

A dolgozat emellett egy iránymutató szerepet szeretne betölteni a verziókövető rendszerek népes táborában, és ismerteti azon alapfogalmakat, kezdeti lépéseket, amelyek ahhoz szükségesek, hogy egy kezdő felhasználó elboldoguljon bármely rendszerrel.

A témaválasztásnál sokat számított, hogy már volt ilyen rendszerrel tapasztalatom, és jelenleg is használom az egyik általam bemutatott egyedet.

Tapasztalataim szerint akár gyakorlott programozók között is előfordulhatnak emberek, akik még nem találkoztak ilyen programmal, vagy csak az egyik típusával.

Ezért is tartottam fontosnak egy átfogó leírást amely lefekteti az alapokat és segítséget nyújt a programok közötti eligazodásban.

2. Verziókövető rendszerek (Version Control Systems)

Verziókezelés alatt több verzióval rendelkező adatok kezelését értjük. Leggyakrabban a mérnöki tudományokban és a szoftverfejlesztésben használnak verziókezelő rendszereket fejlesztés alatt álló dokumentumok, tervek, forráskódok és egyéb olyan adatok verzióinak kezelésére, amelyeken több ember dolgozik egyidejűleg. Az egyes változtatásokat verziószámokkal vagy verzióbetűkkel követik nyomon.

2.1 Általános fogalmak

A terminológia rendszerenként változik, de vannak általánosan használt szakkifejezések.

Baseline

Egy dokumentum vagy fájl jóváhagyott verziója, melyhez az azt követő változtatásokat viszonyítják.

Branch (ág)

A verziókezelt fájlok egy részhalmaza elágazhat, így azoknak több aktuális változatuk lesz egyidejűleg, melyeket akár különböző sebességgel és különböző irányokba is fejleszthetnek.

Check-out

Lokális másolat készítése valamely verziókezelt fájlról. Alapértelmezésben ilyenkor a legfrissebb verziót kapja a felhasználó, de általában van lehetőség konkrét verzió kikérésére is verziószám alapján.

Check-in vagy Commit

Az a művelet, amikor a lokális példány változtatásai beíródnak (vagy egyszerű másolás vagy összefésülés eredményeként) a szerveren tárolt változatba.

Conflict

Konfliktusról akkor beszélünk, ha ketten akarnak megváltoztatni egy dokumentumot vagy fájlt és a rendszer nem képes összeépíteni a változásokat. A felhasználónak ekkor fel kell oldania a konfliktust, amit vagy úgy tehet meg, hogy a változtatásokat összekombinálja vagy úgy, hogy kiválasztja az egyik változtatást és csak azt juttatja érvényre.

Change

Egy változtatás (*change*, *diff* vagy *delta*) mindig egy verziókezelt dokumentumon vagy fájlon tett változtatást jelenti. Rendszerfüggő, hogy milyen mértékű módosítások számítanak *change*-nek.

Change list

Egy *change list* vagy *change set* egy check-in művelet során bevitt változtatások listája, olyan rendszereken, melyek támogatják atomi műveletként több változás egyidejű becsekkelését.

Dynamic stream

Egy olyan adatszerkezet, amely egy adott tárolón lévő elemek konfigurációját reprezentálja, és időben változik.

Export

Az export a checkout-hoz hasonlít azzal a különbséggel, hogy tiszta könyvtárat csinál a verziókezeléshez szükséges meta adatok nélkül. Ezt a műveletet általában közvetlenül a tartalom publikálása előtt szokták használni.

Head

A legutóbbi checkin.

Import

Az import művelettel lehet egy lokálisan tárolt adathalmazt, amely még nem munkamásolat, felmásolni a tárolóra és verziókontroll alá helyezni.

Mainline

Hasonlít a *trunk*-hoz, de minden ágnak lehet saját *mainline*-ja.

Merge

A *merge* művelettel két változtatáslistát lehet összefésülni, s ezáltal egy közös verziót létrehozni. Erre a következő esetekben lehet szükség:

- Ha egy felhasználó módosítja a saját munkamásolatát, majd letölt a szerverről egy másik módosított változatot. Ekkor a szerveren lévő változásokat össze kell fésülni a lokális munkapéldány változásaival a kliensen.
- Ha a fejlesztésben elágazás történt, majd egy hibát kijavítottak valamely ágban, s a javítást alkalmazni kell a másik ágra is.
- Ha a fejlesztésben elágazás történt, majd az ágakat különböző irányba fejlesztettek tovább, s a különböző fejlesztéseket össze kell vonni egy közös változatba (*trunk*-ba).

Repository

A *repository*, *depot* vagy *tároló* az a hely (tipikusan egy szerver), ahol az aktuális és a korábbi verziók tárolódnak.

Reverse integration

Az egyes ágak összedolgozása és bedolgozása a verziókezelő fő *trunk*-jába.

Revision

A *revision* szó ugyanazt jelenti, mint a *version*. Egy verzió.

Tag

A *tag*, *label* vagy *címke* egy fontos időpillanatot jelöl. Egy adott fájlcsoporthoz hozzárendelhető egy címke, amely beszédes, felhasználóbarát nevet vagy verziószámot adhat a csoportnak.

Trunk

A fejlesztés egyik olyan vonala, amely nem *branch*.

Resolve

Változási konfliktusok feloldására irányuló felhasználói tevékenység.

Update

Az *update* vagy *sync* a *repository*-ban lévő változtatásokat dolgozza bele a felhasználó munkamásolatába.

Working copy

Magyarul munkamásolat. A *repository* fájljainak másolata a felhasználó lokális gépén. Minden olyan munka, ami bekerül a *repository*-ba, először mindig egy munkamásolatban történik meg, innen a neve. Fogalmilag a munkamásolat egy homokozó.

2.1.1 Atomi szintű commit

A részben végrehajtott *commit*-ok korruptálhatják az adatbázist, ezért bevezették az atomi szintű *commit* fogalmát, amely azt jelenti, hogy egy *commit* által küldött változtatások csak akkor érvényesülnek, ha minden rendben zajlott az átvitel folyamán. Nem minden rendszer ismeri, de az újabbakra már jellemző a technika megléte.

2.1.2 Cherry-Picking

Algoritmus mely teljesen különböző commitok (patchek vagy changesetek) esetén kiválaszt egyet, és azt alkalmazza a *branch*re.

2.1.3 Changeset

Fájlok különböző verzióinak tárolása helyett egyes rendszerek ún. *changeset* –t használnak. A *changeset* csak a változásokat tárolja le két *tree* között, ezáltal elő lehet állítani egy verzióból a rákövetkező verziót.

2.1.4 Snapshot

Fájlok és könyvtárak egy csoportja, amely a jelenlegi verziót megelőző állapotot írja le.

2.2 Elosztott verziókezelő rendszerek

Az elosztott verziókövető rendszerek a peer-to-peer szemléletet követik, szemben a kliens-szerver centralizált modelljével. Itt egy központi tároló (angolul *repository*) helyett minden felhasználó gépe egy-egy külön tárolóként jelenik meg. A szinkronizáció az egyes gépek között küldött patch-ek (módosításcsomagok) által valósul meg. Ez a megközelítés jelentős változásokat okoz:

- Nincs nagy központi adatbázis, csak munkamásolatok vannak.
- A gyakori műveletek, mint a becsekkelés, verziótörténet böngészés és a változtatások visszaállítása gyorsak, mert nem kell központi szerverrel kommunikálni.
- Minden munkamásolat egy-egy backup, ami természetes védelmet ad az adatvesztés ellen.

Két fajta elosztott verziókezelő létezik, a nyitott és a zárt. A nyitott rendszereket inkább nyílt forráskódú termékeknél használnak, zártakat inkább a nem nyilvános forráskódú termékeknél.

2.2.1 Nyílt rendszerek

A nyitott, elosztott verziókezelők támogatják különböző ágak létezését, és erősen függenek a fent tárgyalt összefésülés (*merge*) művelettől. Általános jellemzőik a következők:

- Minden munkamásolat gyakorlatilag egy ág.
- Minden ág egy-egy munkamásolatként implementálódik. Az ágak összefésülés patch-ek küldözgetésével történik.
- Lehet válogatni az egyes változtatások között, nem kell feltétlenül minden változtatást letölteni.
- Új tagok bármikor csatlakozhatnak a rendszerhez, nincs szükség szerveroldali regisztrációra.
- Ha szükség van rá,a kód elágaztatása könnyen kivitelezhető,mivel minden munkamásolat egy lehetséges elágazás.

Az egyik első nyitott rendszer a BitKeeper volt, mely azért is nevezetes, mert a Linux-rendszermag fejlesztéséhez is használták.

2.2.2 Zárt rendszerek

A zárt, elosztott verziókezelők adatbázis replikáción alapulnak. Csak egy *baseline* van, minden becsekkelt változás ebbe kerül bele

2.2.3 Lépések

Ebben a részben leírom azokat a lépéseket, amelyek szinte minden elosztott rendszernél megegyezik. Egyes esetekben lehetnek eltérések. Kivételt képez a GNU Arch a felépítése, szintaxisa és bonyolultsága miatt, ezért erre nem térnék ki.

- Első lépésként létre kell hoznunk a lokális adatbázisunkat. Mi ezen fogunk dolgozni, és ezzel szinkronizáljuk majd a távoli *repository*kat. Általánosan ezt az *init* paranccsal szokták megtenni.
- A második kétféle lehet, attól függően, hogy egy már meglévő adatbázissal akarunk *sync*elni vagy pedig egy újat akarunk létrehozni.
    - Újat létrehozni fájlok hozzáadásával tudunk. Jellemzően az *add* kulcsszót használják erre.
    - Létező adatbázisok lemásolásánál már eltérő a helyzet. Általában a *sync*, *update*, *pull*, *clone* parancsokat használják.
- Ha módosítottunk valamit és a változtatásokat érvényesíteni akarjuk, akkor egységesen a *commit* paranccsal tehetjük ezt meg.
- Ezeket a változásokat más *repository*nak általában a 2. pontban leírtak szerint küldjük el. De gyakori parancs a *pull* is.

2.3 Központosított rendszerek

Ebben a modellben minden fejlesztő egy közös *repository*t használ. Az adatbázis lehet egy külön gépen vagy akár ugyanazon is. A munkamenet gyakorlatilag egy módosít – *commit* - *update* folyamatból áll. Minden *commit* után ajánlatos frissíteni a *working directory*t, hogy a mások által írt változtatások megjelenjenek.

E kategóriában túlsúlyban vannak a vállalati szoftverek. Többségüket nem parancssoros módban kell beállítani, hanem egy grafikus felhasználói felületen keresztül.
Így az itt leírt lépések leginkább a nyílt forráskódú programokra vonatkozik.

Ez esetben egy központi szerver van, a *repository* pedig gyakran egy könyvtár vagy könyvtárrendszer.

- Első lépésként ezt a könyvtárat kell megadni.
- Második lépésként (amennyiben a kliens szerepét töltjük be) szükségünk lesz egy munkapéldányra, amit a checkout tudunk végrehajtani.
- Harmadikként pedig már dolgozhatunk a fájlokon. (központosított rendszereknél nincs szükség távoli adatbázisok lemásolására,mivel a szervert közvetlen elérjük vagy épp most hozzuk létre).
- A módosításokat ugyanúgy a *commit* (más néven checkin) paranccsal vihetjük fel a *repository*ba.

2.4 Forráskezelési technikák

A hagyományos verziókezelők központosított modellel dolgoznak, ahol minden verziókezelési művelet egy közösen használt szerveren történik. Ha két fejlesztő egyidejűleg próbálja meg módosítani valamelyik fájlt, akkor valami módon el kell kerülni azt, hogy a két személy felülírja egymás munkáját. Az ilyen (centralizált) rendszerek kétféleképpen oldják meg ezt a problémát: zárolással és/vagy összefésüléssel.

2.4.1 Zárolás

A konkurens hozzáférés kezelésének legegyszerűbb módja, ha megtiltjuk a konkurens hozzáférést, azaz ha egy valaki már elkezd módosítani egy fájlt, akkor azt már más felhasználó nem nyithatja meg írásra. Ezt hívják elterjedt kifejezéssel *lock*-olásnak, a magyarosabb, de kevésbé elterjedt zárolás szó helyett. Ha egy felhasználó kivesz (*kicsekkel*) egy fájlt, akkor a többi felhasználó már csak olvasásra nyithatja meg azt egészen addig, amíg a kicsekkelő felhasználó visszateszi (becsekkeli) a módosított változatot (vagy elveti a módosítást).

Ennek a módszernek előnyei és hátrányai is vannak. A nagyobb vagy sok fájlt érintő változtatásoknál célszerű ezt választani, mert bonyolult összefésülési műveleteket lehet megtakarítani vele. Ha azonban egy fájl túl sokáig zárolt állapotban marad, akkor a többi fejlesztő esetleg arra vetemedhet, hogy a verziókezelést megkerülve a fájl lokális másolatát módosítsák, ami nagyobb bonyodalmakhoz vezethet.

2.4.2 Összefésülés

Itt is az angol szóhasználat az elterjedtebb a magyarosabb *összefésülés* helyett. A legtöbb verziókezelő, például a CVS is, lehetővé teszi, hogy több felhasználó dolgozzon egyidejűleg ugyanazon a fájlon. Ekkor a saját változtatását elsőként becsekkelő felhasználó mindenképpen sikerrel fog járni. A rendszer a többi felhasználónak összefésülési lehetőséget ad, mellyel a különböző módosítások összeolvaszthatóak, így a felhasználók nem írják felül egymás munkáját. Az összefésülés lehet automatikus vagy kézi.

Általában az összefésülésre képes verziókezelők is adnak lehetőséget fájlok egy felhasználós, kizárólagos szerkesztésére *reserved edit* néven.

3. Központosított rendszerek

Az első rendszereket a 1970-es években készítették, ám némelyiket még a mai napig karban tartják (a 80-as években fejleszteni kezdett RCS honlapját például legutóbb 2007-ben frissítették).

3.1 Nyílt rendszerek

Nyíltnak nevezik őket, mert a GNU licenc alatt teszik közzé és így ingyenesen felhasználhatóak. Néha, nem csak a program, de a forráskódja is hozzáférhető.

3.1.1 Source Code Control System (SCCS)

1972-ben a Bell Labs-nál, egy OS/MVT operációs rendszert futtató IBM System/370 számítógépre fejlesztették ki. Ezt később átírták, hogy támogassa a UNIX rendszert is.

Kezdetben bekerült néhány disztribúcióba,majd végül a része lett a Unix rendszerek specifikációjának.

Ez volt az uralkodó revíziókezelő szoftver egészen az RCS megjelenéséig. Manapság már csak a fájlformátumát használják.[19]

3.1.2 Revision Control System (RCS)

Az 1980-as években a SCCS alternatívájaként készült el. Ennek egy továbbfejlesztett változata lett a CVS. Ezzel megoldható volt a revíziók tárolásának, naplózásának, összefésülésének és azonosításának az automatizálása.[18]

3.1.3 Concurrent Versions System (CVS)

Kliens-szerver modell alapú.[20] Egyszerre többen is dolgozhatnak ugyanazon a projekten, ilyenkor viszont a rendszer mindig csak a fájl legutóbbi verzióján végzett módosításokat fogadja el. Azaz a fejlesztőknek gyakran kell frissíteniük. Ezt viszont automatikusan megteszi helyettük a rendszer. Emberi beavatkozásra csak konfliktusnál van szükség.

Sikeres *commit* esetén a CVS növeli a fájlok verziószámát és módosítja log fájljait, majd – ha van ilyen – akkor végrehajtja a felhasználó által megadott utasításokat is.

Hiányosságai:

- Nem kezeli fájlok vagy könyvtárak mozgatását, átnevezését. Programhiba helyett ez a tudatos tervezés része volt, mivel régebben a refactoring nem tartozott a fejlesztés folyamatához.
- Biztonsági okból nem kezeli a szimbolikus linkeket.
- Nem támogatja a teljes Unicode-ot, csak az UTF-8-at, mivel eredetileg Unix-ra tervezték, ahol ez a natív formátum.
- Nincs atomi szintű *commit*. A használt szervernek és hálózatnak elég rugalmasnak kell lennie, hogy a *commit* során ne szakadjon meg a kapcsolat.
- Képes tárolni bináris fájlokat is, de elsődlegesen szövegfájlok tárolását preferálja.

3.1.4 Subversion

A Subversion a CVS továbbfejlesztett változata.[2] A fő cél a CVS hibáinak a kiküszöbölése volt. A készítők nem terveztek nagyobb átalakítást, hiszen a CVS sikeresen helyt állt a verziókövető rendszerek között. De az változó gondolkodásmód megkövetelte a CVS „újraírását" [21]. Azaz a CVS –nél említett tervezési hiányosságokat pótolták, konkrétan:

- Fájlok, könyvtárak átnevezésének, mozgatásának valamint meta-adatok verzió számozása.
- Atomi szintű *commit.*
- A *branch* műveletek konstans időben futnak.
- A küldendő adat mérete a változás méretétől függ és nem az adat méretétől.
- Kétirányú kommunikáció (szerver-kliens és kliens-szerver) .
- A szimbolikus linkek verzió számozása, de csak Unix alatt.
- Hatékonyan kezeli a binárisokat.

3.2 Zárt rendszerek

Zárt, azaz a vállalati szoftverek, melyek használata esetenként súlyos pénzekbe kerül. Forráskódba nem vagy csak nagyon kivételes esetben engednek betekintést.

3.2.1 Visual SourceSafe (VSS)

A VSS elődjét a One Tree Software cég készítette, melyet a Microsoft felvásárolt, majd egy évre rá kiadta a saját rendszerét. Ami nem volt más, mint a SourceSafe portolása 32 bitre.

Ez egy lokális rendszer volt. A többi SCM rendszerrel szemben számos hiányossággal küszködött. Ilyen például az atomi szintű *commit*ok hiánya.
2005-ben kiadott verzió már tartalmazta a kliens-szerver mód lehetőségét, ami egyben lehetőséget nyújtott korábbi hibák elkerülésére.

Elméletileg bármilyen fájltípust tud kezelni, de nem szövegfájlok esetén instabillá válnak. Előnye, hogy viszonylag könnyű használni és integrálva van a Visual Studio-ba. Legfeljebb pár emberből álló fejlesztő csapatnak érdemes használni.

3.2.2 Team Foundation Server

A Visual SourceSafe továbbfejlesztett változata, amelyet már nagyobb fejlesztőcsapatoknak is ajánlanak.

Három rétegre osztották fel a rendszert, ezek kliens réteg, alkalmazás réteg és adat réteg.
A kliens réteg egy programozható interfész, amely lehetőséget biztosít, hogy hozzáférjünk az adatbázisban tárolt projektek fájljaihoz. A felhasználó felé alapból egy web felületet biztosít, ezt viszont az alkalmazás réteg tartalmazza. Az adat réteg tulajdonképpen egy SQL adatbázis, amely csak az alkalmazás réteg láthat, a kliens réteg nem.

Forráskezelése támogatja a jelentősebb funkciókat, mint például több *branch* összevonása, fejlett konfliktuskezelő, összefésülő algoritmus.

3.2.3 IBM Rational ClearCase

A ClearCase nem csak egy verziókövető rendszer, hanem egy build eszköz is egyben (bár ebben az aspektusban hagy némi kívánni valót maga után, lásd később).[22]
Más rendszerektől eltérő pozitív tulajdonságai:

- VOB (Versioned Object Base): a program ilyen vob-okban tárolja a fájl, könyvtár verziószámokat és meta adatokat.
- Snapshot-kat is támogatja, lehetővé téve a hálózat nélküli munkát. A VOB-ról egy lokális másolatot tárol és frissíti amint újra tud csatlakozni a szerverhez.

- *nix/Windows kompatibilitás áthidalása VOB fájlokkal. Egy Windows kliens hozzáférhet egy Unix alapokon működő rendszerhez a snapshot-on keresztül.

Negatívumai:

- Az átvitel nem atomi szintű.
- Öregedés. Maga a program ’92-ben jelent meg, és még mindig tartalmaz olyan kódot, mely korábbi rendszerek támogatottságát hivatott megoldani. Ez pedig jelentős lassulást eredményez.
- A rendszer sebessége nagyban függ a hálózat sebességétől, kapacitásától, valamint a csatornán jelentkező problémáktól.
- Technikai okok miatt (az indokoltnál ~2-szer annyi csomag küldése a hálózaton) Windows rendszereken extrém lassú.[1]

3.2.4 Perforce

Annak ellenére, hogy zárt rendszerről van szó, lehetőséget ad az ingyenes használatra nyílt forráskódú szoftverek tervezése esetén. Alaphelyzetben van egy központi adatbázis és egy master *repository*. A program az adatbázisban tárolja a meta-adatokat, és a fájlok tartalmáról készített MD5 hash kulcsokat. Az adatbázis védelmét ellenőrzőpontokkal és naplózással biztosíthatjuk. A program különlegessége (nem egyedi, de ritka), hogy képes elosztott rendszerként is működni.[23]

Előnyei:

- 3 utas *merge* és összefűzések verziószámozása.
- GUI támogatás diff-hez, előzmények megtekintéséhez és adminisztrációhoz.
- Centralizált és decentralizált rendszermód.
- Változások csoportos nyomon követése.
- Szöveges, bináris fájlok és szimbolikus linkek támogatása.
- Programozható kliens és API.

3.2.5 PlasticSCM

Mint a legtöbb zárt rendszer, a PlasticSCM is rendelkezik grafikus felülettel. Előnye, hogy jól kinéző, átlátható a grafikus felülete.[24]

A GUI-n keresztül elvégezhetjük az új *branch*ek létrehozását, kezelését és tulajdonképpen minden támogatott műveletet, és a forrásfájlokat akár azon keresztül is megnyithatjuk.

Tulajdonságai:

- Diff és *merge* eszközök forrás, kép, bináris és könyvtárak kezelésére.
- Csoport alapú jogrendszer.
- Könyvtárak verziószámozása.
- Összefűzések követése.

3.2.6 SourceHaven

A fájlokat, és meta-adatokat egy beépített Oracle adatbázisban tárolja. Alapjaiban véve a Subversion egy továbbfejlesztett változata. Annyira, hogy az svn alapú kliensek képesek együttműködni a SourceHaven szerverrel, mintha az egy Subversion szerver lenne. [25]

3.2.7 StarTeam

A StarTeam a Borland cég terméke [6], ennél fogva leginkább alkalmazkodó képes a vállalat saját termékeihez. Windows támogatás alapból él benne, más integrációs lehetőségekkel együtt.

A háttérben egy IBM DB2, Microsoft SQL vagy Oracle adatbázisszerver lapul.[26]

Előnyei:

- Fejlett felhasználói jogok kiosztása.
- RSA titkosítás hálózati azonosításhoz.
- Windows, Web, Java és konzolos kliensek valamint a népszerűbb IDE –k támogatása.
- Atomi szintű *commit* (azaz checkin).

4. Elosztott rendszerek

E rendszerek P2P alapokon nyugszanak, azaz nincs kitüntetett szerver. Legtöbbször egy már létező protokollt használnak, de nem ritka a saját termés sem.

4.1 Nyílt rendszerek

Ugyanúgy, mint a központosított esetben, itt is ugyanazt jelenti a nyílt fogalom. A különbség az, hogy az elosztott rendszerekre a nyílt típus a jellemző.

4.1.1 SVK

Ez a Subversion fájlrendszere alapján kifejlesztett elosztott verziókövető rendszer, de tervezése jobban hasonlít a BitKeeper és a GNU Arch programokra.
Bevezet néhány új fogalmat is (legalábbis saját használatra):
Depot: rövid név, amely a valódi *repository*-ra mutat. Az üres string a deafult depot-ot jelenti.
Resource: fájl, könyvtár vagy speciális elem (mint például szimbolikus link) amelyet a SVK számoz.
Depotpath: egy resource a megadott depot-on belül, plusz az útvonal.
Mirror: egy külső *repository*-hoz linkelt és esetleg azzal szinkronizált depotpath.
A külső *repository* lehet egy teljesen más rendszer *repository*-ja, amelyet teljesen lemásol, így offline munka esetén is elérhető marad.
Revision: egy adott időben a teljes tree állapota a *repository*-ban.
Change: két különböző verziójú depot tartalma közötti különbség.
Tree Delta: két fa közötti különbség.
XD: ugyanaz, mint a Working Copy

4.1.2 GNU Arch

Az Arch egy jól konfigurálható, ám eléggé bonyolult rendszer, különösen az új felhasználók számára. Az egyébként jó képességekkel rendelkező programnak ezen felül a másik nagy hibája a szokatlan generált fájlnevek. Ez nehezebbé teszi más (nem Unix) rendszereken való használatát. Elméletileg a 2.0 verzióban a fájlneveket egyszerűsítik, és a parancsok számát

is igyekeznek csökkenteni, ezáltal egyszerűbbé tenni az Arch-ot (ám ezen dolgozat készültekor a legfrissebb verzió még csak 1.3.5).

Előnyei:

- Könnyű *branch* kezelés.
- Fejlett *merge* algoritmus.
- Meta-adatok kezelése.

4.1.3 Monotone

A Monotone tervezésénél nagy hangsúlyt fektettek az elosztott rendszerek közötti műveletekre és a titkosításra. SHA-1 hash kulcsokat használ a revíziók számozására és RSA titkosítást a felhasználók azonosítására.[10]

Minden felhasználó saját adatbázissal rendelkezik. A műveleteket is ezen hajtja végre. A műveletek alacsony költségűek és gyorsak, lévén helyi adatbázisról van szó. Viszont új felhasználó létrehozásakor az adatbázis syncelése (klónozása) sok időt vehet igénybe az integritási ellenőrzés és azonosítás miatt.

- Jól dokumentált.
- Kevésbé elterjedt.
- Kevés és kezdeti stádiumban lévő GUI.
- Sebesség problémák a kezdeti adatbázis létrehozásakor.

4.1.4 Bazaar

Eredetileg a GNU Arch egy forkjaként jött létre [9], de nem ez volt a végleges formája. Újragondolták, majd újraírták Python nyelven és ezzel megszületett a *Bazaar-NG* melyet később szimplán csak Bazaar-nak neveztek el [3]. A tervezésnél a könnyű kezelhetőséget, egyszerű parancsokat a CVS és a Subversion mintájára alakították ki.

- Központosított és elosztott modell támogatása.
- Képes együttműködni más SCM programokkal (mint például Subversion, Git, Mercurial).
- Sebesség szempontjából csak a hálózattól függ (bár lassabb mint a Git).

4.1.5 darcs

Egy funkcionális nyelven íródott program [17], amely – mint korábban említett rendszerek is – a CVS és Subversion rendszereket próbálta helyettesíteni.

Eltérően az említett programoktól, ez teljes mértékben elosztott rendszer. A központi *repository* egy másolatával dolgozunk és az ebben történt változásokat küldjük el, vagy fogadjuk más *repository*-tól. A szerző kidolgozott egy elméletet a patch-ek kezelésére, amit ő „theory of patches" [8] néven illet, melynek a lényege, hogy a teljes fa leírható patchek egy sorozatával.

Támogatja az ssh, http protokollokat, de szinkronizálhatunk akár email-en keresztül is. Ám a távoli *repository* lehet akár egyazon gépen is.[8]

A program lassúsága, *branch*ek összefésülése nem vetnek kedvező fényt a darcs-ra. Emellett a Windows alatti felhasználók, pedig további hibákra is számíthatnak.

4.1.6 Git

A Git-et Linus Torvalds készítette miután a BitKeeper megvonta a linux kernelfejlesztőktől az ingyenes licencet. Kezdetben egy alacsony-szintű felületnek indult, aminek segítségével mások kész rendszereket írhatnak. Végül egy teljes, használható program lett belőle.

- Több fajta hálózati protokoll támogatása.
- Közvetlen SVN és SVK támogatás, CVS szerver emuláció.
- Sebességre és kifejezetten nagy projektek kezelésére kifejlesztett rendszer.
- Komponens alapú, egyes műveleteket több komponens összekapcsolásával végezhetünk el.
- Helyigényes, időnként szükség van a felesleges adatok eltakarítására, ami lassú lehet.

4.1.7 Mercurial

A Mercurial-t (mint például a Git-et is) a Monotone ihlette [16]. Maga a program Python nyelven íródott, kivéve a diff algoritmusát, melyet C-ben írták.

Előnyei:

- Saját fejlesztésű GUI kiterjesztés, mely irányított aciklikus gráfokkal jeleníti meg a revíziókat.
- Subversion *repository*k importálása. [5]

4.2 Zárt rendszerek

Elosztott zárt rendszerből igazán nem sokat lehet találni, ezért csak egyet mutatok be, amely egyedi (email használata a csomagküldésre) módszerével tűnik ki a többi közül.

4.2.1 Code Co-op

1996-ban a rendszert fejlesztő cégnek elsődleges egy olyan rendszerre volt szüksége, amellyel meg lehetett oldani a kommunikációt. Akkoriban az egyetlen anyagi szempontból is előnyös megoldás az email volt. Így született meg a Code Co-op-ban az emailen keresztüli szinkronizáció [11]. Persze egy ilyen lassú módszerrel a központosított modell nem reális elképzelés, ezért döntöttek az elosztott modellű rendszer mellett. Későbbiekben elkészült hozzá a LAN támogatás is.

Az elosztott rendszereknél problémát jelentő összefűzésen azzal az illúzióval könnyít, mintha csak egy *trunk* lenne, és ebben végzi el a *commit*-ot.

5. Összehasonlító táblázat

5.1 Nyílt rendszerek

	Adatbázis modell	Forráskezelési technika	Támogatott platformok	Program nyelv	Revízióazonosító
Bazaar	Elosztott és Kliens-szerver	Összefűzés	Unix-like, Windows, Mac OS X	Python	Pszeudorandom
CVS	Kliens-szerver	Összefűzés	Unix-like, Windows, Mac OS X	C	Névtér
CVSNT	Kliens-szerver	Összefűzés vagy zárolás	Unix-like, Windows, Mac OS X , OS/400	C++	Névtér
darcs	Elosztott	Összefűzés	Unix-like, Windows, Mac OS X	Haskell	Névtér
Git	Elosztott	Összefűzés	POSIX, Windows, Mac OS X	C, Shell scriptek	SHA-1 hashek
GNU arch	Elosztott	Összefűzés		C, Shell scriptek	Névtér
Mercurial	Elosztott	Összefűzés	Unix-like, Windows, Mac OS X	Python	Számok és SHA-1 hashek
Monotone	Elosztott	Összefűzés	Unix-like, Windows, Mac OS X	C++	SHA-1 hashek
Subversion(SVN)	Kliens-szerver	Összefűzés vagy zárolás	Unix-like, Windows, Mac OS X	C	Névtér
SVK	Elosztott	Összefűzés	Unix-like, Windows, Mac OS X	Perl	

	Támogatott hálózati protokollok	Atomi szintű commit	Fájlok átnevezése	Átnevezett fájlok mergelése	Külön jogok beállítása a repository egyes részeihez
Bazaar	HTTP, SFTP, FTP, saját protokoll ssh támogatással, email bundles	Van	Van	Van	Alapszintű hozzáférés kezelés
CVS	pserver, ssh	Nincs	Nincs	Nincs	Korlátozott, scriptek segítségével
CVSNT	sspi, sserver, gserver, pserver, saját protokoll ssh-val	Van	Van		Van
darcs	HTTP, saját protokoll ssh-val,email	Van	Van	Van	Nincs
Git	saját protokoll ssh-val, rsync, HTTP, email, csomagok	Van	Van	Van	N/A
GNU arch	WebDAV, HTTP	Van	Van		Van
Mercurial	HTTP, saját protokoll ssh-val, email	Van	Van	Van	Van
Monotone	saját protokoll ssh-val (netsync), fájlrendszer	Van	Van	Van	Van
Subversion(SVN)	saját protokoll ssh-val(svnserve), HTTP és SSL(WebDAV-on keresztül)	Van	Van	Nincs	Van, WebDav-on keresztül
SVK		Van	Van	Van	Mint Subversion

	Szimbolikus linkek	Aláírt revízió	Összefűzések követése	Sorvégjel kezelése	Integrálhatóság
Bazaar	Van			Nincs	Eclipse, Visual Studio
CVS	Nincs	Nincs		Van	Eclipse, Kdevelop, IDEA
CVSNT	Van	Nincs	Van	Van	
darcs	Nincs	Van			
Git	Van	Van	Van	Van	Eclipse
GNU arch	Van	Van			
Mercurial	Van	Van	Van	Van	Eclipse, NetBeans
Monotone	Nincs	Van	Van	Nincs	
Subversion(SVN)	Van	Nincs		Van	
SVK	Van	Van	Van	Van	

5.2 Zárt rendszerek

	Adatbázis modell	Forráskezelési technika	Támogatott platformok	Program nyelv
ClearCase	Elosztott és Kliens-szerver	Összefűzés vagy zárolás		
Code Co-Op	Elosztott	Összefűzés		C++
Perforce	Kliens-szerver	Összefűzés vagy zárolás	Unix-like, Windows, Mac OS X	C/C++
PlasticSCM	Kliens-szerver	Összefűzés	Unix-like, Windows, Mac OS X	C#
PureCM	Kliens-szerver	Összefűzés vagy zárolás	Unix-like, Windows, Mac OS X	C++, C#, Java
Razor	Kliens-szerver	Összefűzés vagy zárolás	Unix-like, Windows, Mac OS X	C/C++
SourceHaven	Kliens-szerver	Összefűzés vagy zárolás	Unix-like, Windows, Mac OS X	C, Java
StarTeam	Kliens-szerver	Összefűzés vagy zárolás		C, Java
Surround SCM	Kliens-szerver	Összefűzés vagy zárolás	Unix-like, Windows, Mac OS X	C++
Team Foundation Server	Kliens-szerver	Összefűzés vagy zárolás	Windows Server 2003, Windows	C++ és C#
Vault	Kliens-szerver	Összefűzés vagy zárolás		C#
Visual Source Safe	Kliens-szerver	Összefűzés vagy zárolás	Windows	C

	Revízió azonosító	Hálózati protokollok	Atomi szintű commit	Fájlok átnevezése	Átnevezett fájlok mergelése
ClearCase	Névtér	HTTP, saját protokoll (CCFS és MVFS fájl rendszermeghajtó)	Nincs	Van	Van
Code Co-Op	Felhasználó azonosító és sorszám	e-mail (MAPI, SMTP/POP3, Gmail), LAN	Van	Van	Van
Perforce	Névtér	saját protokoll	Van	Van	Van
PlasticSCM	Névtér	saját protokoll	Van	Van	Van
PureCM	Névtér	TCP/IP, SSL	Van	Van	Van
Razor	Sorozatszám	TCP/IP	Van	Van	Van
SourceHaven	Névtér	WebDAV, saját protokoll	Van	Van	
StarTeam	MD5 hashek	TCP/IP, saját protokoll	Van	Van	
Surround SCM	Névtér	saját protokoll	Van	Van	Van
Team Foundation Server	Névtér		Van	Van	Van
Vault		HTTP, HTTPS	Van	Van	Van
Visual Source Safe	Névtér	Nincs,legfeljebb megosztott könyvtárakon keresztül	Nincs	Van	

	Szimbolikus linkek	Aláírt revízió	Összefűzés követése	Sorvégjel kezelése	Integrálhatóság
ClearCase	Van	Van	Van	Van	Eclipse, IDEA, Visual Studio, Kdevelop
Code Co-Op	Nincs	Nincs		Nincs	
Perforce	Van	Van	Van	Van	Eclipse, IDEA, Visual Studio, Kdevelop
PlasticSCM	Van	Van	Van		
PureCM	Van	Nincs	Van	Van	
Razor	Nincs	Van	Van	Van	
SourceHaven	Van	Nincs			
StarTeam	Van	Nincs	Van	Van	
Surround SCM	Nincs	Nincs	Van	Van	
Team Foundation Server			Van		
Vault	Nincs	Nincs		Van	
Visual Source Safe		Nincs			

6. Összefoglalás

Megfigyelhető, hogy az elosztott rendszerek között leginkább nyílt forrású programok találhatók, míg a központosított rendszerekre a zárt a jellemző. Ennek egyik oka a fejlesztés helyében rejlik.

Elosztott rendszert használók sokszor távol vannak egymástól, pl.: egy GNU projekt önkéntes résztvevői lehetnek a világ bármely tájáról, ami nehezebbé tenné a folyamatos kapcsolatot a központi szerverrel.

Centralizált programok felhasználói többnyire vállalatok, melyeknek fejlesztői részlege egy helyen található, így biztosított a szerverrel való kapcsolat.

Összességében elmondható, hogy a legtöbb centralizált rendszer helyettesíthető decentralizáltakkal. A táblázatból is látható, hogy biztonság szempontjából az újabbak megfelelnek az elvárásoknak (egy SHA-1 hasht használó rendszer már elég biztonságos).

Elosztott rendszerek előnye a központosítottakkal szemben, hogy

- Több felhasználó dolgozhat egyszerre ugyanazon a forráson.
- A fejlesztés egyszerre több helyen folyhat.
- A legtöbb ilyen program ingyenes, ezzel csökkentve a projekt költségeit.
- Nem igényel folyamatos kapcsolatot a szerverrel. Azaz például egy szerverleállás esetén sem kell abbahagyni a munkát, mivel az adatbázis másolatát lokálisan elérjük.
- Redundáns tárolás miatt védettebbek az adatbázis problémákkal szemben.

Ellenben egy központosított szerverrel nagyobb biztonságot érhetünk el. Leginkább a külső támadásokkal szemben (a felhasználó azonosítás az elosztott rendszereknél is megvan), mivel az adatbázis csak a belső hálózatról érhető el.

Ezzel el is érkeztem a dolgozat végére, mely alapján a nyílt forráskódú elosztott rendszereket tudom ajánlani a kedves olvasónak, amennyiben nem riad vissza az apróbb kellemetlenségek láttán.

Úgy vélem, hogy dolgozatom elérte célját. Először az alapokat ismertette az olvasóval, majd bemutatta a népszerűbb verziókövető rendszereket. Az egyes rendszerekre külön nem térhettem ki, hiszen az egyes rendszerek dokumentációja a legegyszerűbb pár oldaltól akár

ezerötszáz oldalig is terjedhet. Amennyiben sikerült kiválasztani a kívánt rendszert, további ismeretek elsajátítására lesz szükség e rövid bemutató után.

7. Irodalomjegyzék

[1] Patrick Dugal: ClearCase Build Performance Degradation: Technical Report, 2004

[2] Mike Mason: Pragmatic Version Control using Subversion, 2nd Edition, Pragmatic Bookshelf kiadó, 2006

[3] Farkas Szilveszter: Elosztott verziókövető rendszerek (Bazaar)

[4] Ian Clatworthy, Canonical: Distributed Version Control Systems - Why and How

[5] Bryan O'Sullivan: Distributed revision control with Mercurial, 2007

[6] Borland Corporation: Borland® StarTeam® 2008 - Administering and Using StarTeam, 2008

[7] http://darcs.net/manual/node8.html#Patch

[8] http://lists.osuosl.org/pipermail/darcs-users/2007-March/010877.html

[9] http://en.wikipedia.org/wiki/Bazaar_software

[10] http://en.wikipedia.org/wiki/Monotone_%28software%29

[11] http://en.wikipedia.org/wiki/Code_Co-op

[12] http://en.wikipedia.org/wiki/Comparison_of_revision_control_software

[13] http://better-scm.berlios.de/comparison/comparison.html

[14] http://en.wikipedia.org/wiki/Revision_control

[15] http://www.dwheeler.com/essays/scm.html

[16] http://en.wikipedia.org/wiki/Mercurial_%28software%29

[17] http://en.wikipedia.org/wiki/Darcs

[18] http://en.wikipedia.org/wiki/Revision_Control_System

[19] http://en.wikipedia.org/wiki/Source_Code_Control_System

[20] http://en.wikipedia.org/wiki/Concurrent_Versions_System

[21] http://en.wikipedia.org/wiki/Subversion_%28software%29

[22] http://en.wikipedia.org/wiki/IBM_Rational_ClearCase

[23] http://en.wikipedia.org/wiki/Perforce

[24] http://en.wikipedia.org/wiki/Plastic_SCM

[25] http://en.wikipedia.org/wiki/SourceHaven

[26] http://en.wikipedia.org/wiki/StarTeam