Szoftver technológia

Verziókövető rendszerek

Cserép Máté
ELTE Informatikai Kar
2019.

Verziókövető rendszerek

Történeti háttér

- A szoftverek méretének és komplexitásának növekedésével létrejött szoftverkrízis következményeként megnövekedett:
  - a programok forráskódjának mérete,
  - a szoftverprojektek megvalósításához szükséges idő,
  - és szükséges programozói erőforrás.
- A szoftveripar fejlődésével egyre több alkalmazás készült
  - a fejlesztések életciklusa gyakran nem ért véget a program első publikus verziójának kiadásával,
  - karbantartási és további fejlesztési fázisok követték.

- **A szoftverprojektek méretben, komplexitásban, időben és a résztvevő fejlesztők számában is növekedni kezdtek.**

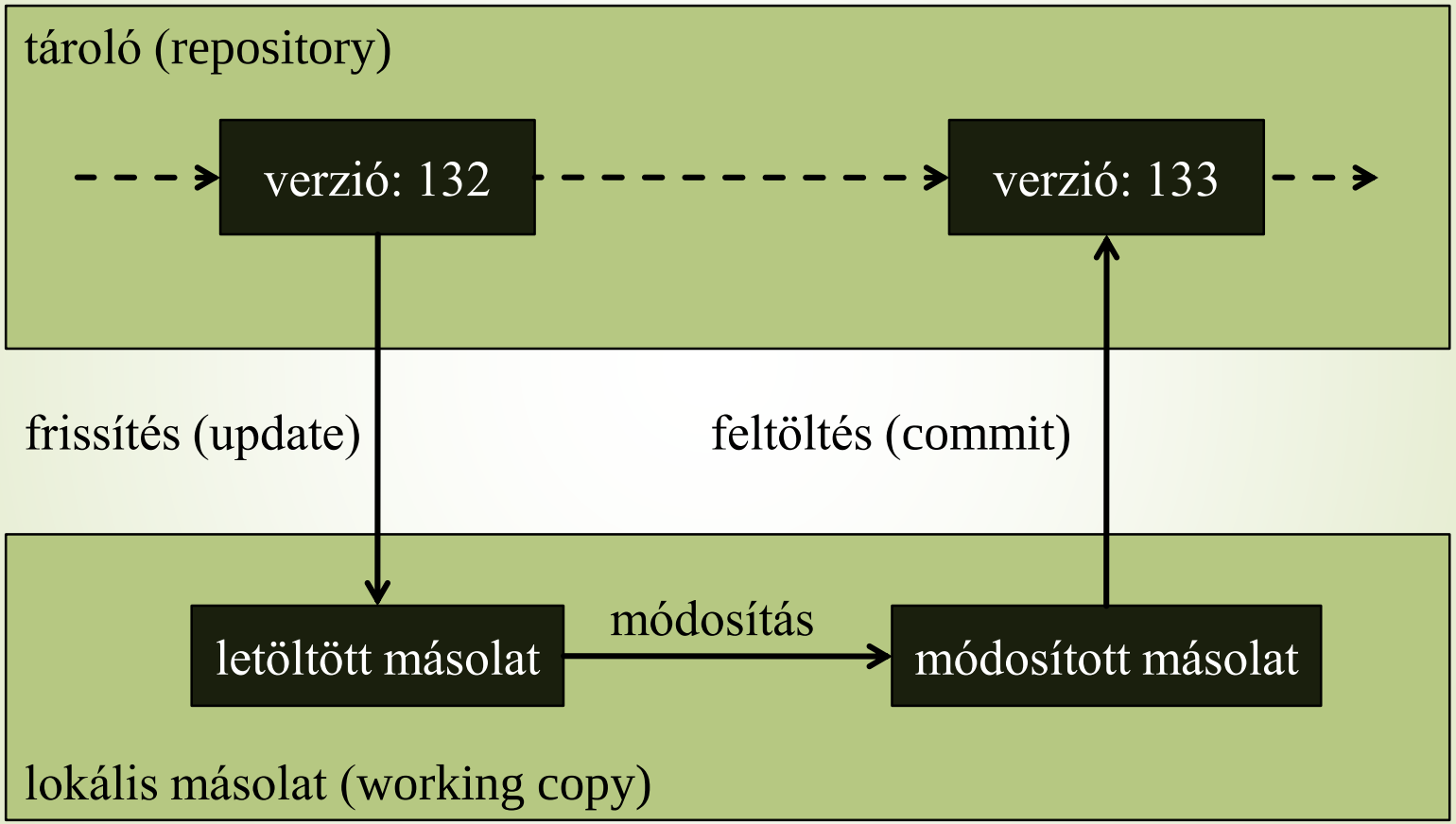
Verziókövető rendszerek

Funkcionalitás

- Mivel az implementáció tehát több lépésben, és sokszor párhuzamosan zajlik, szükséges, hogy az egyes programállapotok, jól követhetőek legyenek, ezt a feladatot a *verziókövető rendszer*ek (*revision control system*) látják el
  - pl. *CVS*, *Apache Subversion* (*SVN*), *Mercurial*, *Git*
  - egy közös tárolóban (*repository*) tartják kódokat
  - ezt a fejlesztők lemásolják egy helyi munkakönyvtárba, és amelyben dolgoznak (*working copy*)
  - a módosításokat visszatöltik a központi tárolóba (*commit*)
  - a munkakönyvtárakat az első létrehozás (*checkout*) után folyamatosan frissíteni kell (*update*)

Verziókövető rendszerek

Funkcionalitás



Verziókövető rendszerek

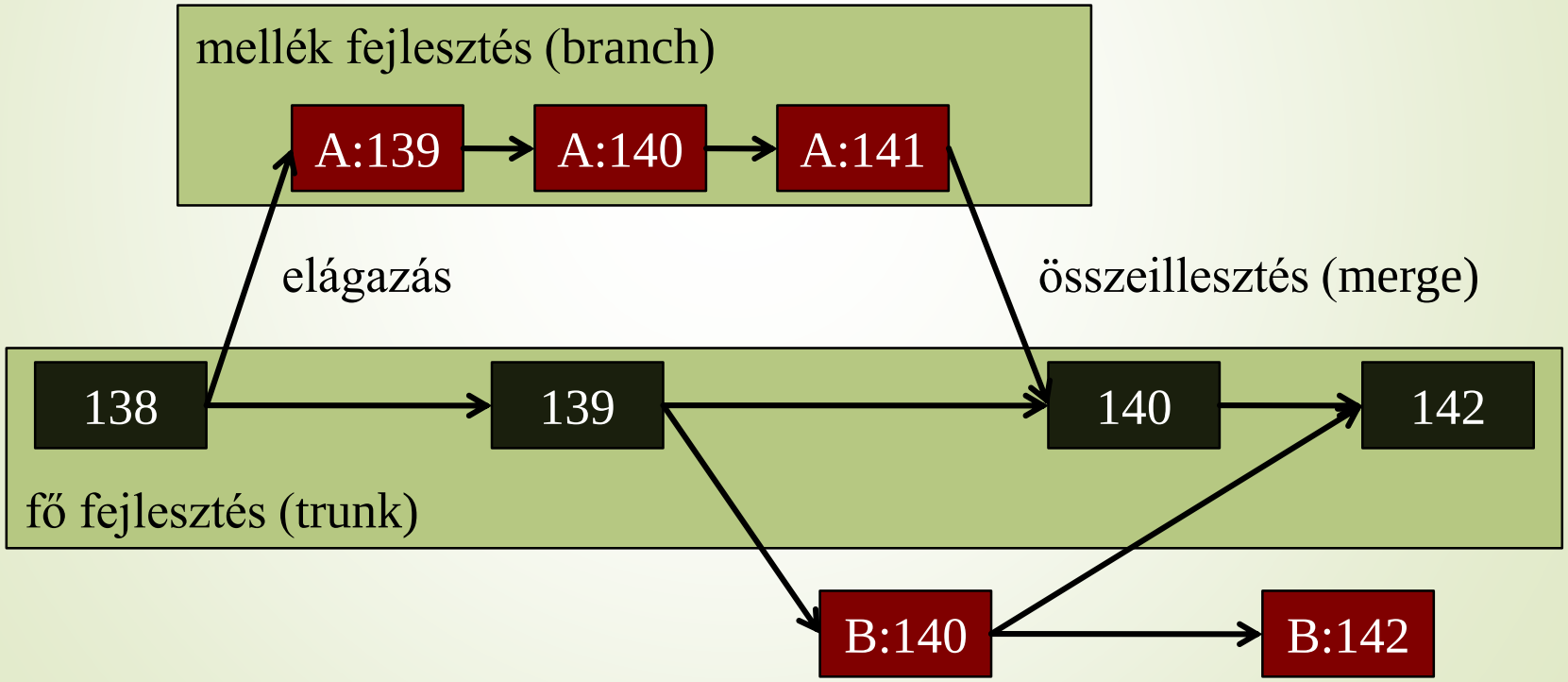
Funkcionalitás

- A verziókövető rendszerek lehetővé teszik:
  - az összes eddig változat (*revision*) eltárolását, illetve annak letöltési lehetőségét
  - a fő fejlesztési vonal (*baseline*, *master* vagy *trunk*) és a legfrissebb változat (*head*) elérését, új változat feltöltését annak dokumentálásával
  - az egyes változatok közötti különbségek nyilvántartását fájlonként és tartalmanként (akár karakterek szintjén)
  - változtatások visszavonását, korábbi változatra visszatérést
  - konfliktust okozó módosítások ellenőrzését, illetve megoldását (*resolve*)

Verziókövető rendszerek

Funkcionalitás

- a folyamat elágazását, és ezáltal újabb fejlesztési folyamatok létrehozását, amelyek a fő vonal mellett futnak (*branch*), valamint az ágak összeillesztését (*merge*)



Verziókövető rendszerek

Funkcionalitás

- az összeillesztés rendszerint utólagos manuális korrekciót igényel
- az összeillesztésnek rendszerint automatikusan illeszti a módosított tartalmakat kódelemzést használva, ez lehet 2 pontos (*two-way*), amikor csak a két módosítást vizsgálja, vagy 3 pontos, amikor az eredeti fájlt is

- programrészek zárolását (*lock*), hogy a konfliktusok kizárhatóak legyenek
- adott verzió, mint pillanatkép (*snapshot*) rögzítése (*tag*), amelyhez a hozzáférés publikus
- feltöltések atomi műveletként történő kezelését (pl. megszakadó feltöltés esetén visszavonás)

Verziókövető rendszerek

Lokális verziókövető rendszerek (1. generáció)

- Forráskód változásainak követése, a szoftver funkcióinak különböző kombinációjával készült kiadások kezelése
  - lokális tároló (de többen is elérhetik pl. *mainframe* esetén)
  - fájl alapú műveletvégzés (1 verzió 1 fájl változásai)
  - konkurenciakezelés kizárólagos zárak által

- Az 1970-es években lefektetésre kerültek az elméleti alapok
  - Source Code Control System (SCCS) – 1972
  - Revision Control System (RCS) - 1982

Verziókövető rendszerek

Centralizált verziókövető rendszerek (2. generáció)

- Több fejlesztő általi párhuzamos szoftverfejlesztés támogatásának előtérbe kerülésre
  - centralizált modellt megtartva, de kliens-szerver architektúra
  - fájlhalmaz alapú műveletek (1 verzió több fájl változásai)
  - konkurenciakezelés jellemzően beküldés előtti egyesítéssel (*merge before commit*)
- Az 1990-es évektől terjedtek el:
  - Concurrent Versions System (CVS)
  - Subversion (SVN)
  - SourceSafe, Perforce, Team Foundation Server, stb.
- *Hátrány: a szerver kitüntetett szerepe (pl. meghibásodás), továbbá a verziókezeléshez hálózati kapcsolat szükségeltetik*

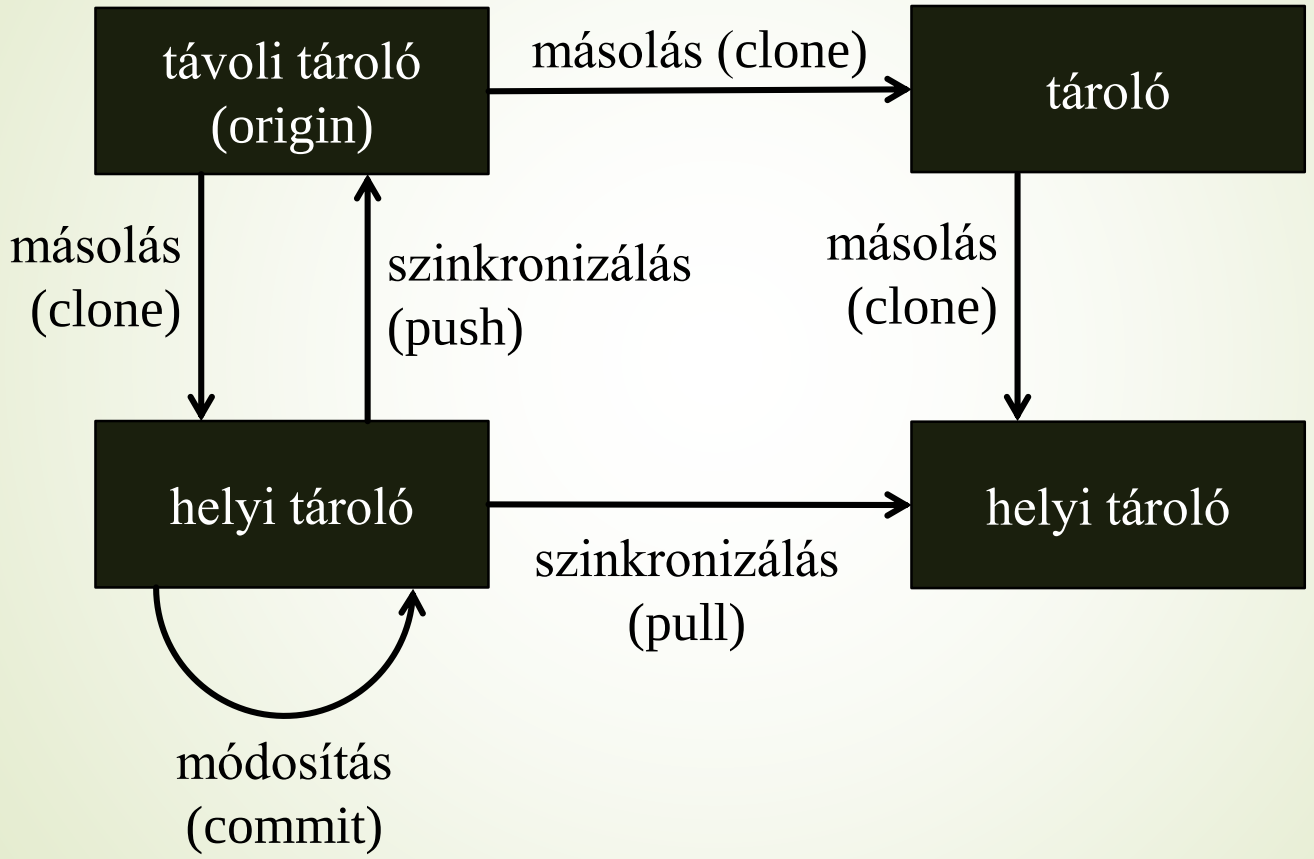
Verziókövető rendszerek

Elosztott verziókövető rendszerek (3. generáció)

- A klasszikus verziókezelő műveletekről leválasztásra kerül a hálózati kommunikáció, azok a felhasználó által kezdeményezhető önálló tevékenységekként jelennek meg
  - decentralizált, elosztott hálózati modell
  - minden kliens rendelkezik a teljes tárolóval és verziótörténettel
  - a revíziókezelő eszköz műveletei lokálisan, a kliens tárolóján történnek
  - a kommunikáció *peer-to-peer* elven történik, de kitüntetett (mindenki által ismert) szerverek felállítására van lehetőség
  - konkurenciakezelés jellemzően beküldés utáni egyesítéssel (*commit before merge*)
- A 2000-es évek első felében jelent meg:
  - Monotone, Darcs, Git, Mercurial, Bazaar, stb.

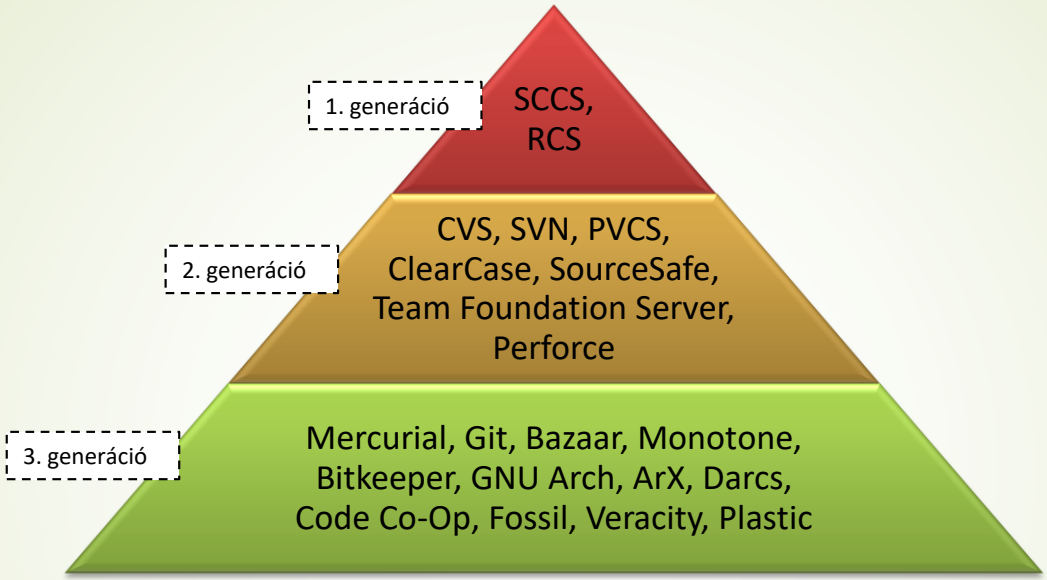
Verziókövető rendszerek

Elosztott verziókövető rendszerek (3. generáció)



Verziókövető rendszerek

Generációs modell



Generáció	Hálózati modell	Műveletvégzés	Konkurenciakezelés
Első	Lokális	Fájlonként (non-atomic commits)	Kizórálóagos zárak (exclusive locks)
Második	Központosított	Fájlhalmaz (atomic commits)	Egyesítés beküldés előtt (merge before commit)
Harmadik	Elosztott	Fájlhalmaz (atomic commits)	Beküldés egyesítés előtt (commit before merge)

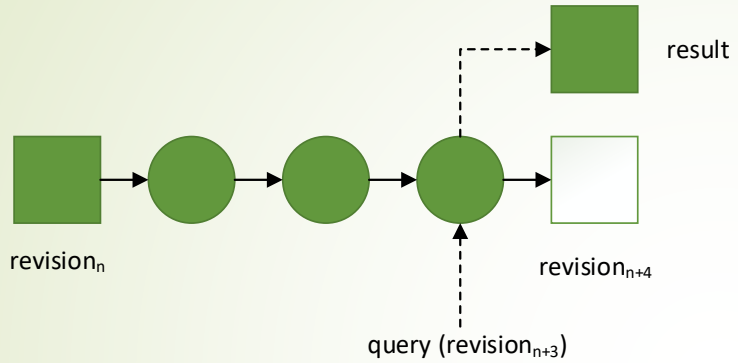
Verziókövető rendszerek

Változások reprezentációja

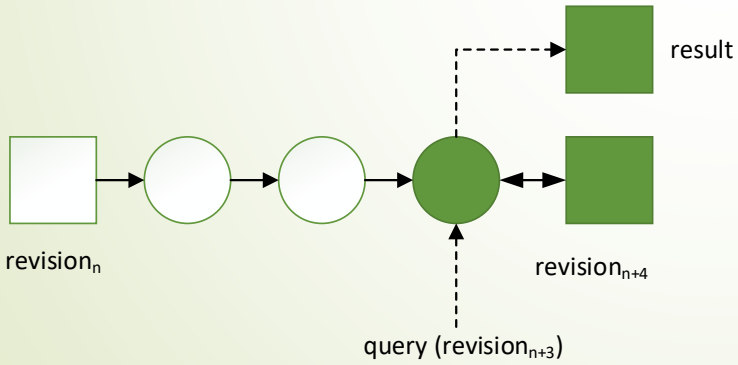
- A teljes revíziók tárolása nem lehetséges az adattárolás és adatkezelés jelentős költségei miatt
- A verziókezelő eszközök ezért csak két egymást követő verzió közötti különbséget, a *változáslistá*t (*changeset*, *delta*) tárolják
  - egyes rendszerek (pl. Mercurial) időnként pillanatfelvételt (*snapshot*) készítenek a teljes tartalomról
- Eleinte (SCCS) a delták a régi verzióból az újat tudták előállítani (*forward deltas*)
- Korán felmerült (RCS), hogy a fordított delták (*reverse deltas*) használata a legújabb verzió pillanatképének tárolásával jobb teljesítményt nyújthat, ugyanis leggyakrabban egy ág legfrissebb állapotát szokták lekérni
  - Kevert megoldás is lehetséges, pl. a fő ágon fordított irányú deltákat, a mellékágakon viszont előre mutató delták

Verziókövető rendszerek

Változások reprezentációja



Forward deltas

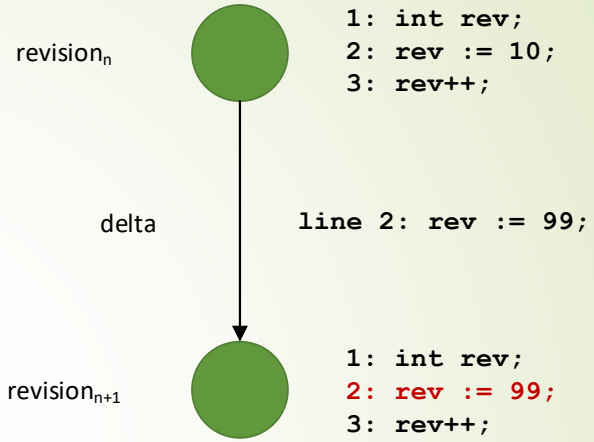


Reverse deltas

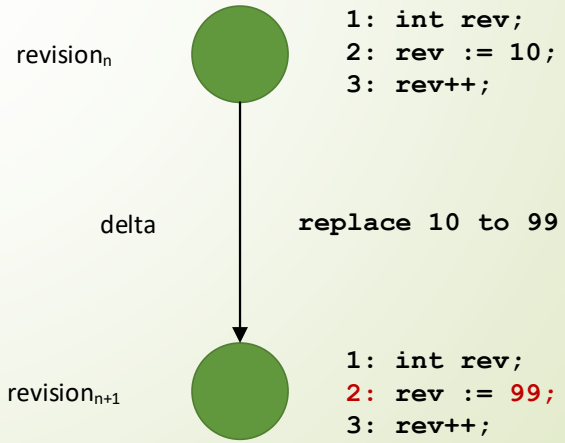
Verziókövető rendszerek

Változások reprezentációja

- Az eltérések meghatározása szöveges fájlok, így programnyelvi forráskódok esetében jellemzően állapot alapúan történik
  - a legtöbbször soronkénti összehasonlítással
    - Pl. GNU diff
  - struktúrált tartalom esetén az összehasonlítás egysége más is lehet (pl. XML, JSON, UML)
- Bináris adatok (pl. képek) esetén a *művelet alapú* megközelítés is alkalmazható.



Állapot alapú



Művelet alapú

Verziókövető rendszerek

SVN: tároló másolása

```
svn checkout https://mysite.com/best-project
> A    best-project\branches
> A    best-project\tags
> A    best-project\trunk
> Checked out revision 1.
```

Verziókövető rendszerek

SVN: módosítások beküldése és szinkronizálása

```
cd best-project/trunk
# main.cpp létrehozása
svn add main.cpp
svn commit –m "Added main program."
> Adding          main.cpp
> Transmitting file data .done
> Committing transaction...
> Committed revision 2.
```

Verziókövető rendszerek

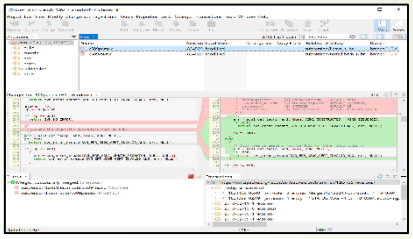
SVN: módosítások beküldése és szinkronizálása

```
# main.cpp módosítása
svn status
> M    main.cpp
svn commit –m "Added main program."

# más kliensek frissítése:
svn update
```

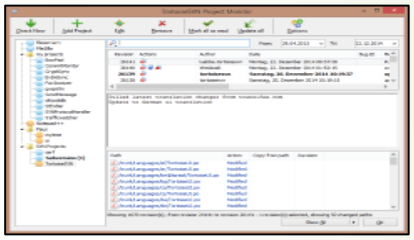
Verziókövető rendszerek

SVN GUI kliensek

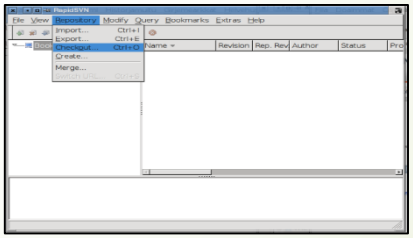
- TortoiseSVN
  - Windows



- SmartSVN
  - Linux, Windows, Mac



- RapidSVN
  - Linux, Windows, Mac



Verziókövető rendszerek

Git: tároló másolása

```
git config --global user.name "Hallgató Harold"
git config --global user.email hallgato@inf.elte.hu

git clone https://mysite.com/best-project.git
> Cloning into 'best-project'...
> remote: Enumerating objects: 3, done.
> remote: Counting objects: 100% (3/3), done.
> remote: Total 3 (delta 0), reused 0 (delta 0)
> Unpacking objects: 100% (3/3), done.
```

Verziókövető rendszerek

Git: módosítások helyi tárolóba küldése

```
cd best-project
# main.cpp létrehozása
git add main.cpp

git status
> On branch master
> Your branch is up to date with 'origin/master'.
> Changes to be committed:
>   (use "git reset HEAD <file>..." to unstage)
>         new file:   main.cpp

git commit -m "Added main program."
> [master d26c7a9] Added main program.
>  1 file changed, 1 insertion(+)
>  create mode 100644 main.cpp
```

Verziókövető rendszerek

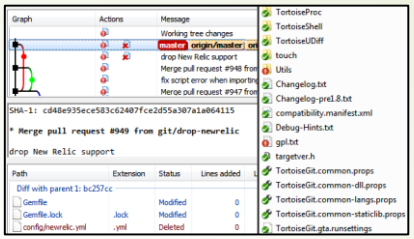
Git: távoli tárolóval szinkronizálás

```
git push origin master #or simply 'git push'
> Counting objects: 3, done.
> Writing objects: 100% (3/3), 247 bytes | 123.00 KiB/s, done.
> Total 3 (delta 0), reused 0 (delta 0)
> To /path/to/workspace/folder
   d45172c..80a39a2  master -> master

# main.cpp módosítása
git add main.cpp
git commit -m "Edited the main program."
git push

# más kliensek frissítése:
git pull
```

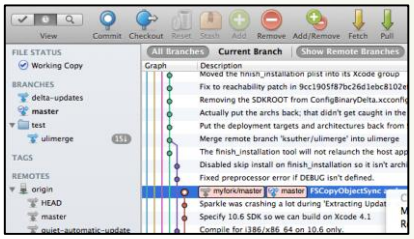
Verziókövető rendszerek

Git GUI kliensek

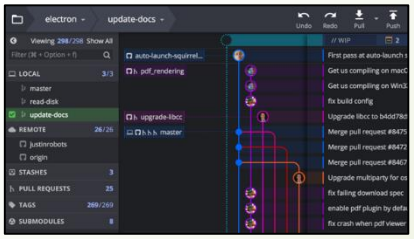
- TortoiseGit
  - Windows



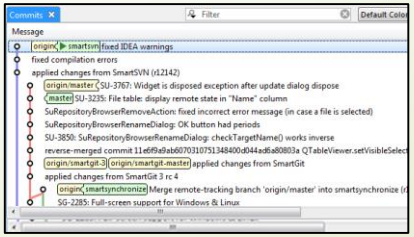
- SourceTree
  - Windows, Mac



- GitKraken
  - Linux, Windows, Mac



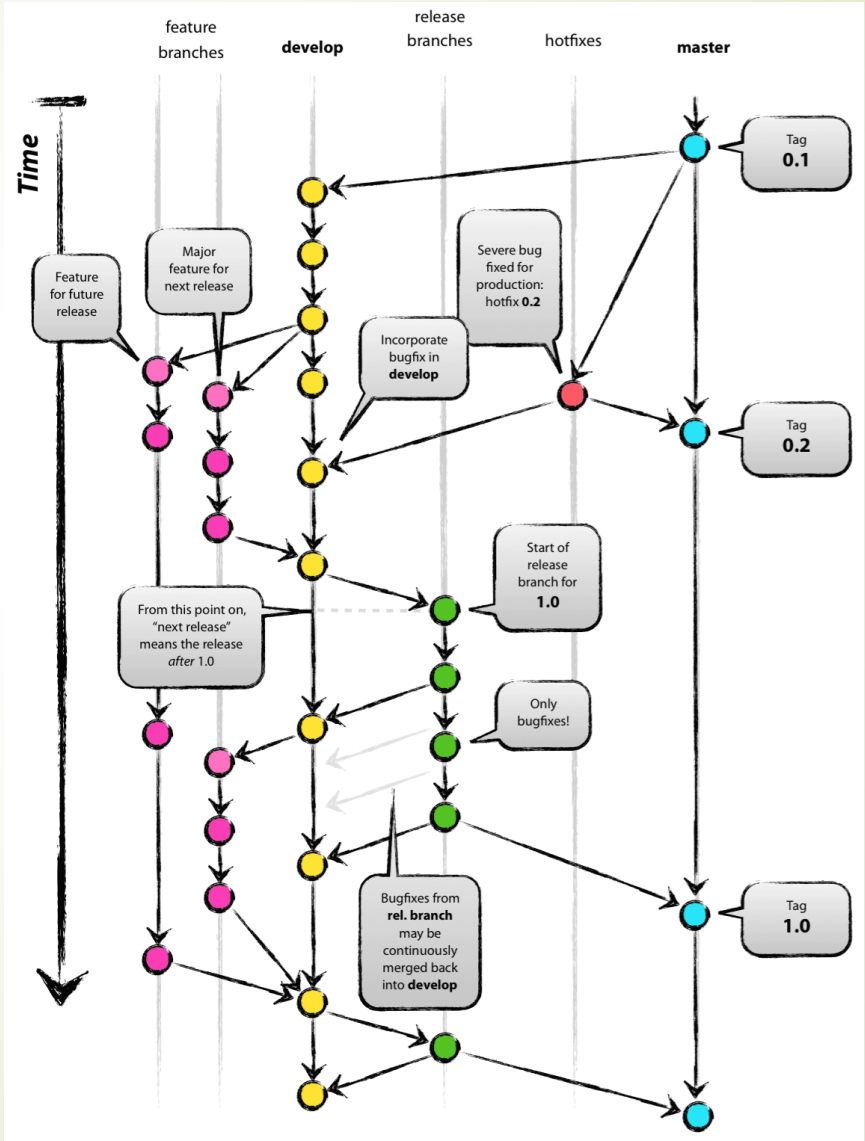
- SmartGit
  - Linux, Windows, Mac



Verziókövető rendszerek

Git Feature Branching Model

- Fő fejlesztési ágak:
  - master
  - develop
- Támogató ágak:
  - feature branches
  - release branches
  - hotfix branches



Forrás: nvie.com