Szoftvertechnológia

Verziókövető rendszerek

Cserép Máté
ELTE Informatikai Kar
2020.

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.**

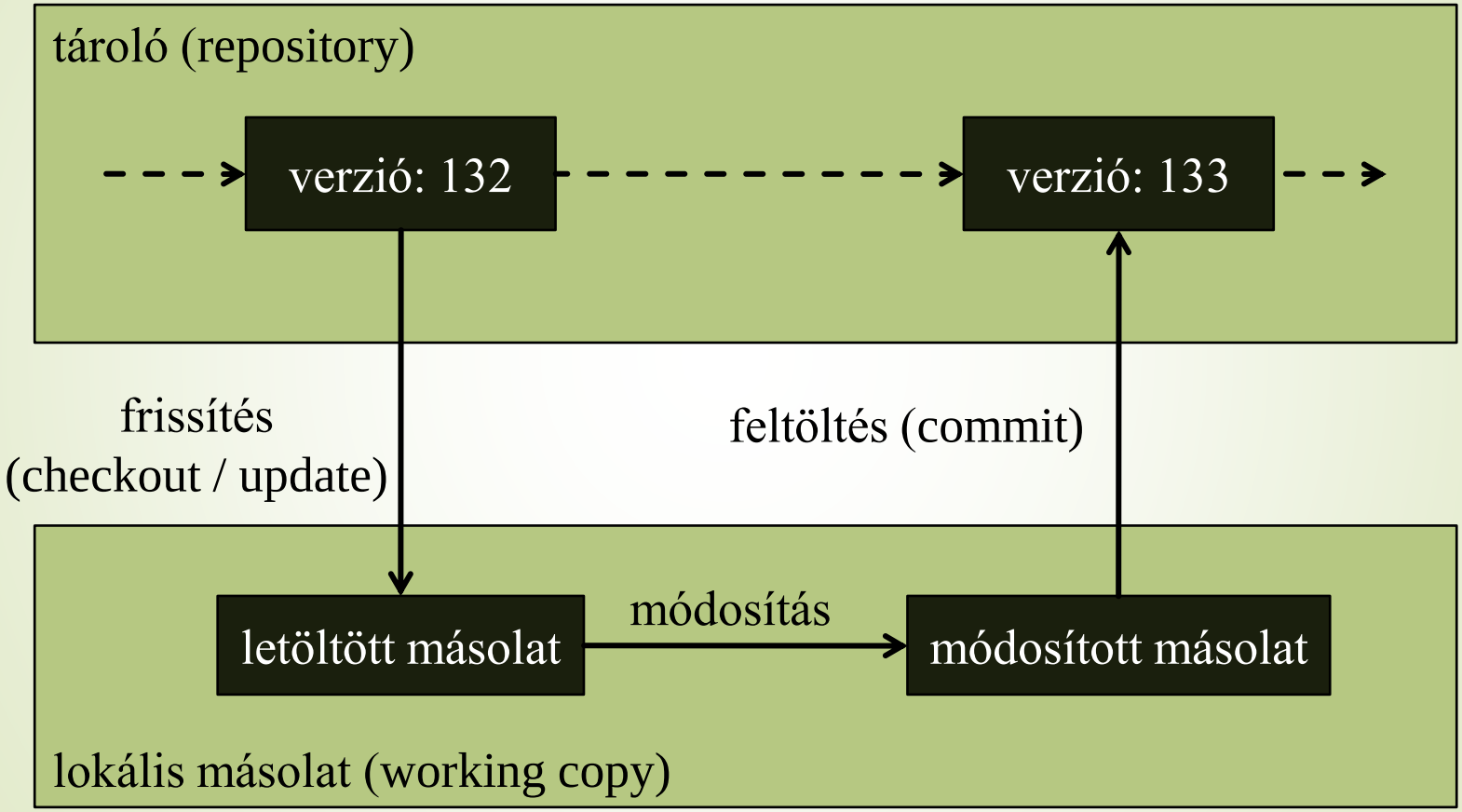
Verziókövető rendszerek

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

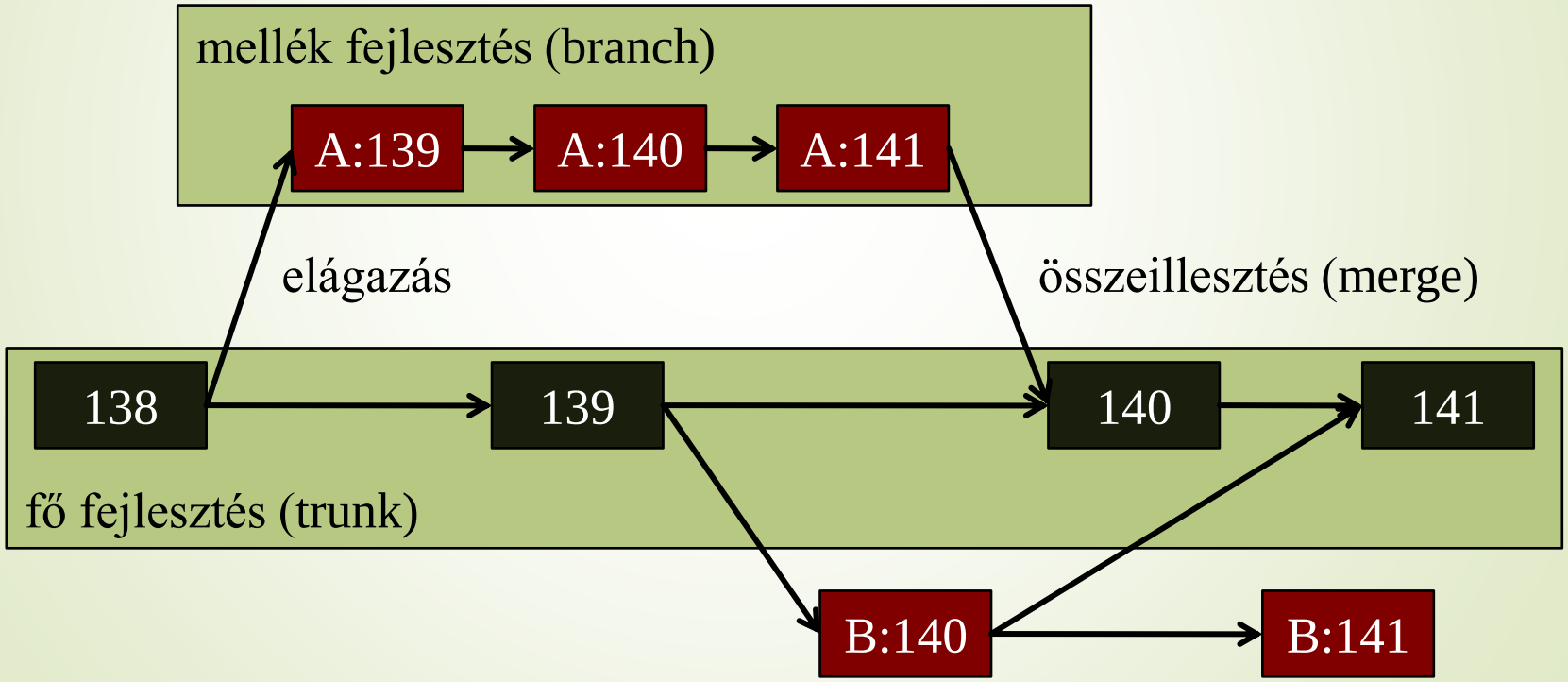
Funkcionalitás

- 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*

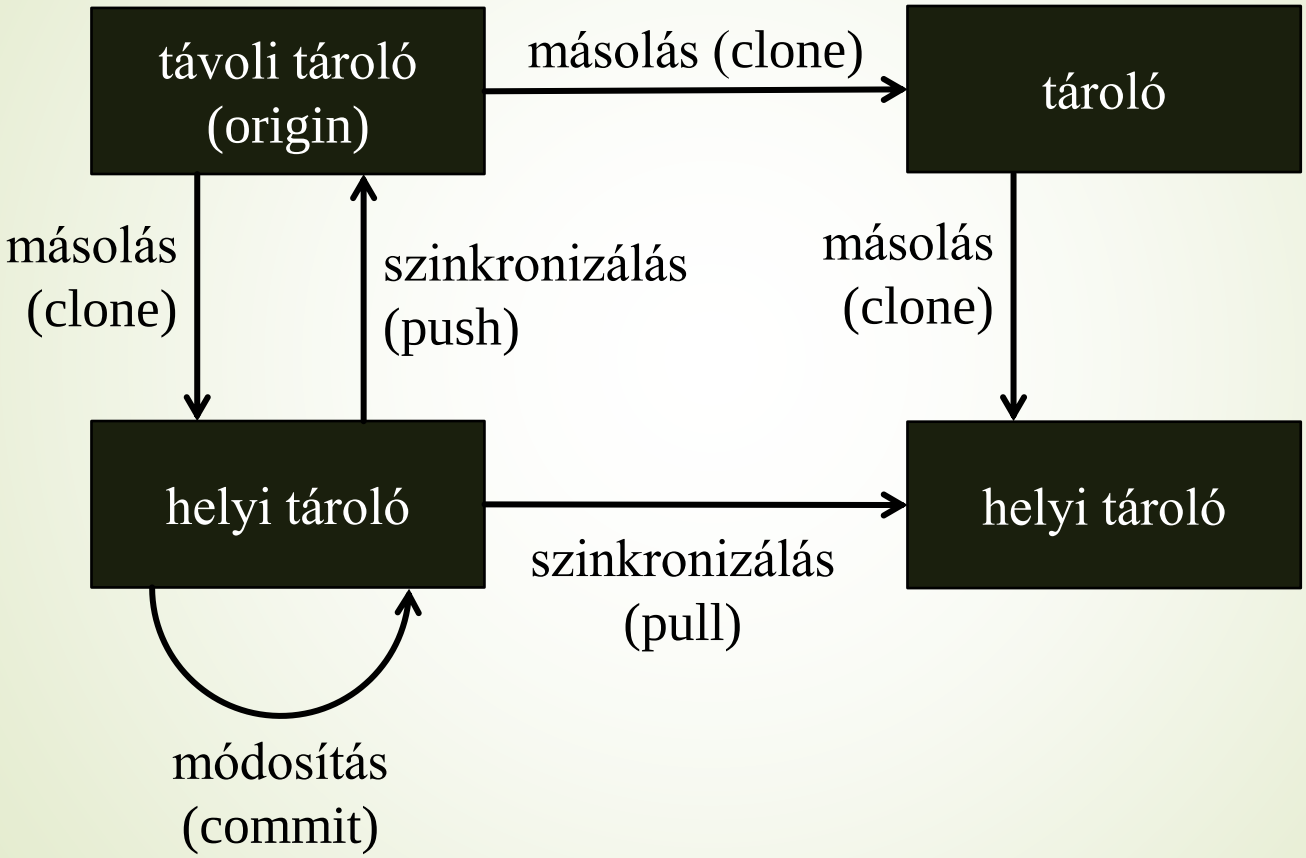
Verziókövető rendszerek

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.

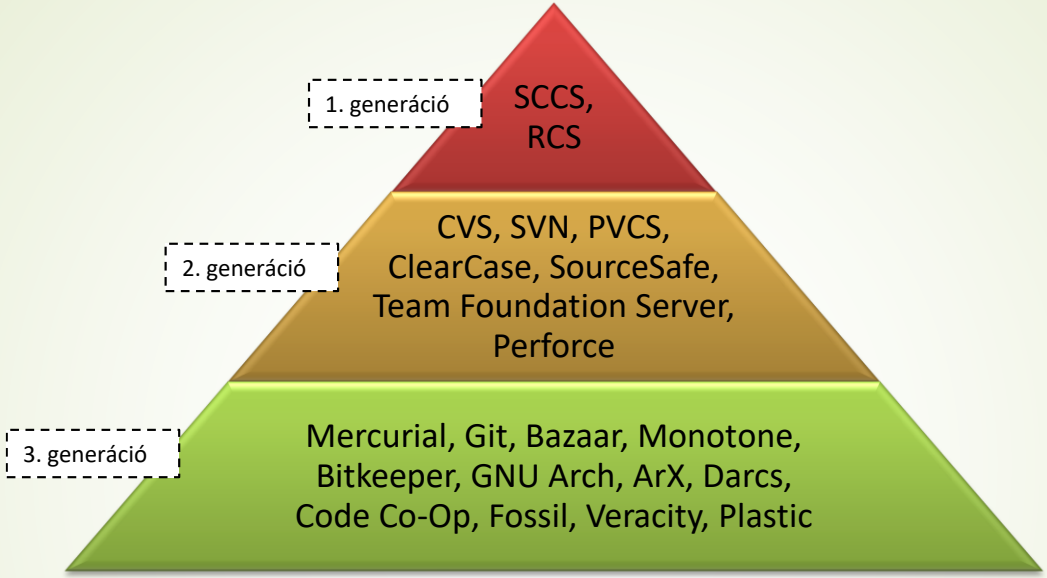
Verziókövető rendszerek

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)

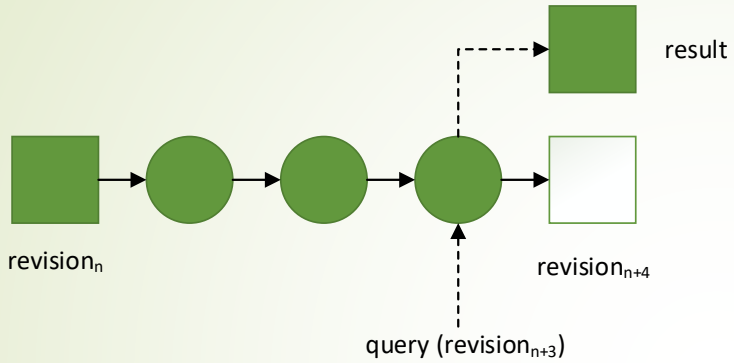
Verziókövető rendszerek

Változások reprezentációja

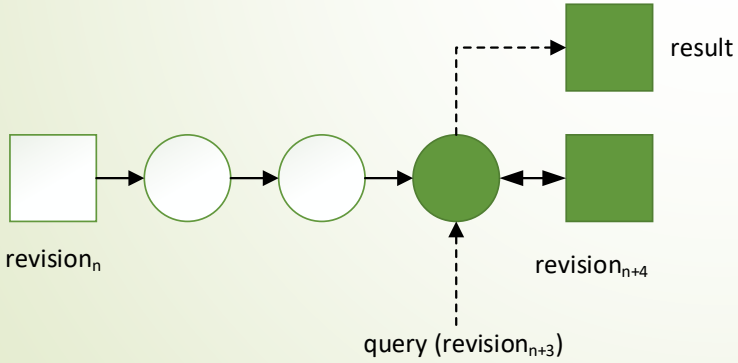
- A teljes revíziók tárolása nem lehetséges az adattárolás és adatkezelés jelentős költségei miatt
- A verziókezelő eszközök ezért csak két egymást követő verzió közötti különbséget, a *változáslistát* (*changeset*, *delta*) tárolják
  - egyes rendszerek (pl. Mercurial) időnként pillanatfelvételt (*snapshot*) 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

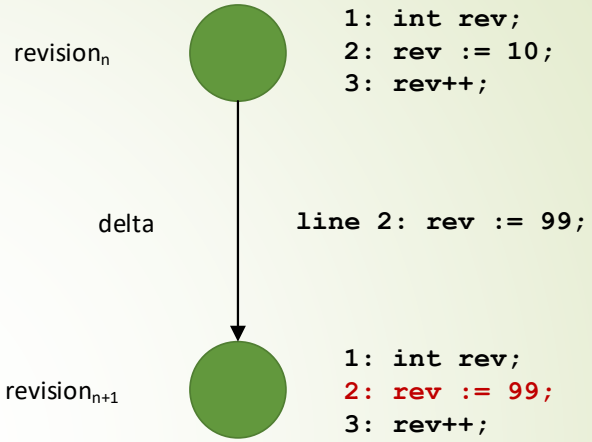


Reverse deltas

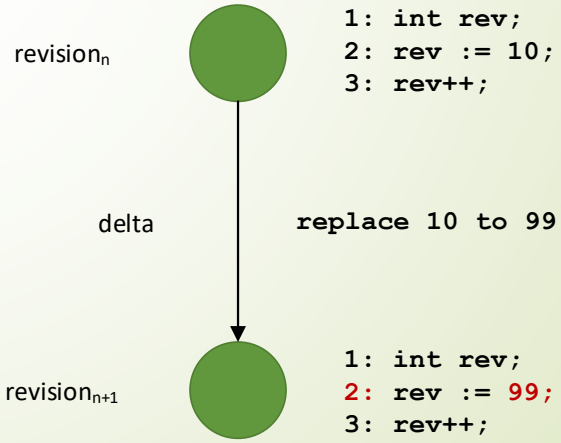
Verziókövető rendszerek

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

Git

- A félév folyamán a gyakorlati projektek forráskódját a Git verziókezelő rendszerben fogjuk követni, amelyet integráltan támogat a GitLab projektvezető szolgáltatás.
  - Kari GitLab szerver: https://szofttech.inf.elte.hu/
- A GitLab szerveren lévő távoli tároló (*remote repository*) tartalmát a GitLab webes felületén is böngészhetjük, megtekinthetjük, sőt egyszerűbb módosításokat is végrehajthatunk (ezekből ugyanúgy *commit* lesz).
- A verziókezelést azonban alapvetően egy lokális munkapéldányon (*local repository*) szokás végezni kliens programmal, majd szinkronizálni a távoli tárolóval.
  - Konzolos kliens utasítások
  - Asztali grafikus kliens alkalmazások

Verziókövető rendszerek

Git: telepítés

- A Git verziókezelő rendszer telepítése:
  - Windows, Mac telepítő: https://git-scm.com/downloads
  - Debian/Ubuntu: `apt-get install git`
  - Más UNIX rendszerek: https://git-scm.com/download/linux
- Telepítés után a konzolos Git parancsokkal egyből dolgozhatunk.
  - Windows telepítés esetén célszerű a Git hozzáadását választani a PATH környezeti változóhoz.
  - Grafikus kliens programok külön telepíthetőek.
- Minden Git commithoz hozzárendelésre kerül a szerzője neve és email címe, így ezeket szükséges globálisan beállítanunk, mielőtt először használjuk a Gitet.

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

Verziókövető rendszerek

Git: tároló létrehozása

- Egy új lokális tárolót létrehozhatunk üresen:

```
git init
```

- Vagy egy ismert távoli tároló lemásolásával:

```
git clone https://mysite.com/best-project.git
```

- Jellemzően távoli tárolók másolását alkalmazzuk, még akkor is, ha kezdetben üres a projekt.
  - Az így lemásolt távoli tárolóra *origin* néven hivatkozhatunk (alapértelmezetten) majd a későbbiekben, pl. a tárolók szinkronizálásakor.

Verziókövető rendszerek

Git: változások követése

- Új fájlokat valamint a fájlok módosításait a `git add` utasítással vonhatjuk verziókezelés alá, az ún. *staging area*-ba helyezve:

```
git add main.cpp
```

  - Konkrét fájl helyett mintát (pl. `*.cpp`) vagy könyvtárat is megadhatunk.

- A munkakönyvtár állapotát a `git status` utasítással ellenőrizhetjük bármikor.

```
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
```

Verziókövető rendszerek

Git: módosítások helyi tárolóba küldése

- Amennyiben a kívánt módosításokat a *staging area*-hoz adtunk, annak tartalmát egy új verzióként beküldhetjük a lokális tárolóba, a `git commit` utasítással:

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

Verziókövető rendszerek

Git: módosítások változáskövetése

- Új fájlokat is verziókezelés alá vonhatunk:

```
git add rectangle.h

git commit -m "Added Rectangle class."
```

- Egy új verzió új fájlokat és meglévő fájlok módosításait is tartalmazhatja:

```
git add circle.h

git add main.cpp

git commit -m "Added Circle class.
               Modified the main program."
```

Verziókövető rendszerek

Git: szinkronizálás távoli tárolóval

- A lokális tárolónkban létrehozott új verziókat szükséges szinkronizálni a távoli tárolóval, hogy a változtatásainkhoz mások is hozzáférhessenek, ezt a `git push` paranccsal tehetjük meg.

```
git push origin master
> 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
```

  - Szükséges megadni, hogy melyik távoli tárolóval szeretnénk szinkronizálni, és melyik fejlesztési ágat. Amiről klónoztunk alapértelmezetten az *origin* néven ismert, az ág itt a *master*.
  - Nyomkövető ágak (*tracking branches*) használatakor ezek elhagyhatóak, így az utasítás `git push`-ra egyszerűsödik.
- Fejlesztő társaink beküldött módosításait a saját lokális tárolónkkal és munkapéldányunkkal a `git pull` paranccsal szinkronizálhatjuk.

Verziókövető rendszerek

Git: fejlesztési ágak létrehozása

- Új fejlesztési ágat a `git branch` paranccsal hozhatunk létre.

```
git branch new-branch
```

  - Az új fejlesztési ág abból a verzióból fog elágazni, amelyiket aktuálisan betöltöttük a munkapéldányunkba.
- Fejlesztési ágak között a `git checkout` utasítással válthatunk.

```
git checkout new-branch
```

  - Az alapértelmezett fejlesztési ág neve jellemzően *master*.
- Új fejlesztési ág létrehozása és átváltás egyetlen lépésben:

```
git checkout -b new-branch
```

Verziókövető rendszerek

Git: fejlesztési ágak egyesítése

- A fejlesztési ágakon végrehajtott módosításokat az adott funkció elkészülte után szeretnénk a fő fejlesztési ágba visszacsatolni.
  - Az ágak egyesítésére a `git merge` utasítás szolgál.
  - Előbb betöltjük a fő fejlesztési ágat:

```
git checkout master
```

  - Majd a mellék fejlesztési ág módosításait egyesítjük vele:

```
git merge new-branch
```

- Amennyiben ugyanazon fájlok ugyanazon részét időközben mindkét fejlesztési ágon módosítottuk, az egyesítés jellemzően nem kivitelezhető automatikusan, ezt ütközésnek (*merge conflict*) nevezzük.

Verziókövető rendszerek

Git: ütközések feloldása

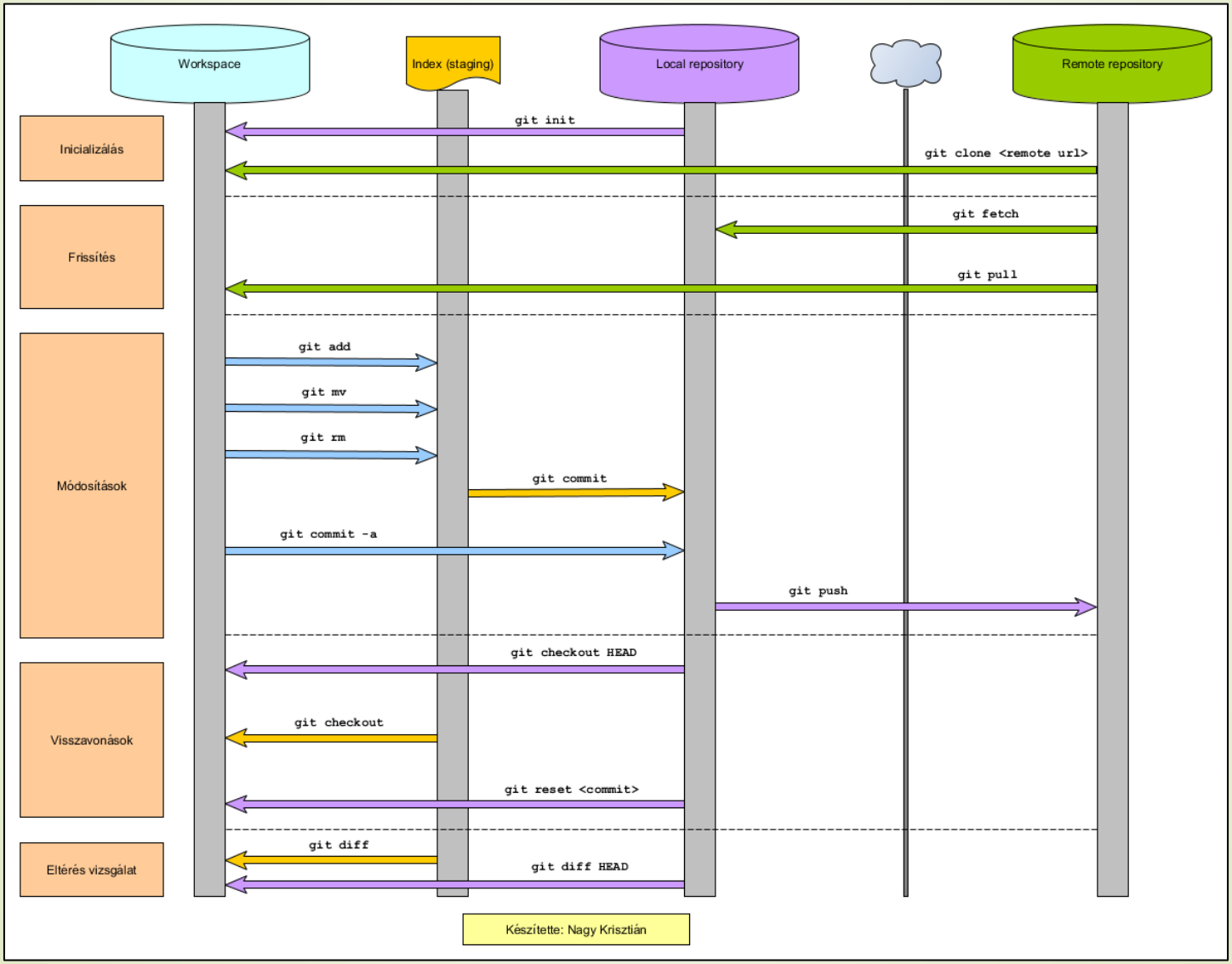
- Az ütközéseket manuálisan kell feloldanunk az érintett fájlok szerkesztésével.
- Az automatikusan nem egyesíthető részeket a Git forrásfájlokban speciális szintaxisba foglalja:

```
<<<<<<< HEAD
Az aktuális, jelen esetben master ágon lévő ütköző
tartalom
=======
Az egyesíteni kívánt, new-branch ágon lévő ütköző
tartalom
>>>>>>> new-branch
```

  - A fejlesztő feladata eldönteni, hogy a két lehetőség közül melyiket kívánja megtartani, esetleg a kettő vegyes megoldását szükséges alkalmazni.
  - A feloldott fájlokat a *staging area*-hoz kell adni (`git add`), majd a változásokat beküldeni (`git commit`).

Verziókövető rendszerek

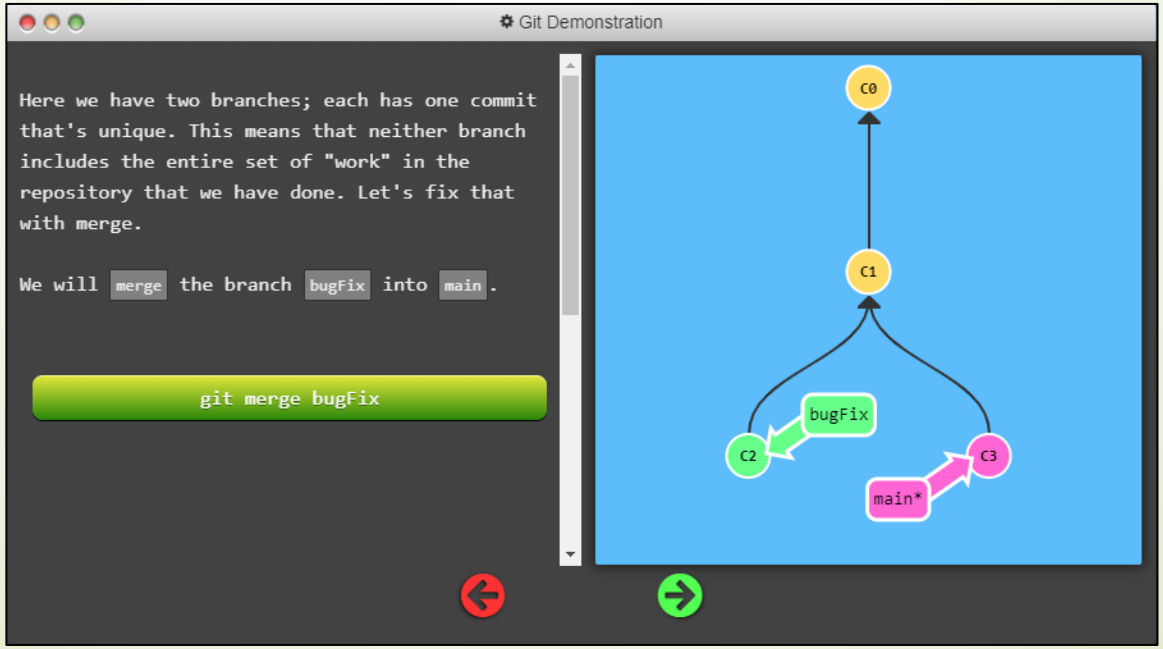
Git: alapvető konzolos utasítások áttekintése



Verziókövető rendszerek

Online eszközök a tanuláshoz

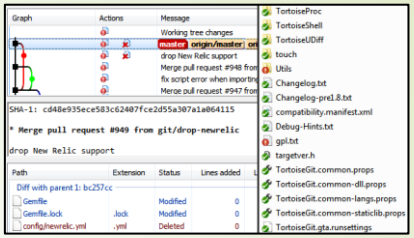
- Több online eszköz is elérhető, amelyekkel vizualizáció mellett, kitűzött feladatok teljesítésével vagy szabadon gyakorolva fejleszthetőek a Git használati ismeretek. Pl.:
  - https://git-school.github.io/visualizing-git/
  - https://learngitbranching.js.org/

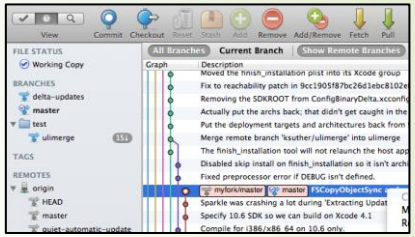


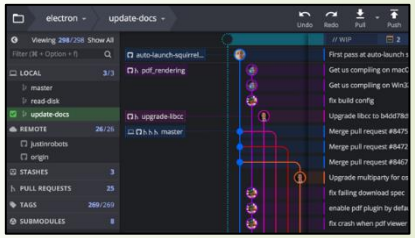
Verziókövető rendszerek

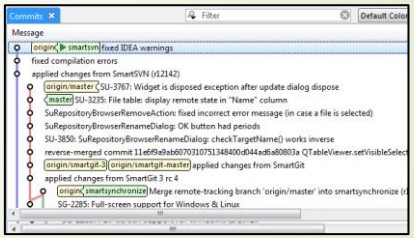
Git GUI kliensek

- TortoiseGit
  - Windows
  - *géptermi gépeken elérhető*
- SourceTree
  - Windows, Mac
- GitKraken
  - Linux, Windows, Mac
- SmartGit
  - Linux, Windows, Mac
- Továbbiak:
  - https://git-scm.com/downloads/guis





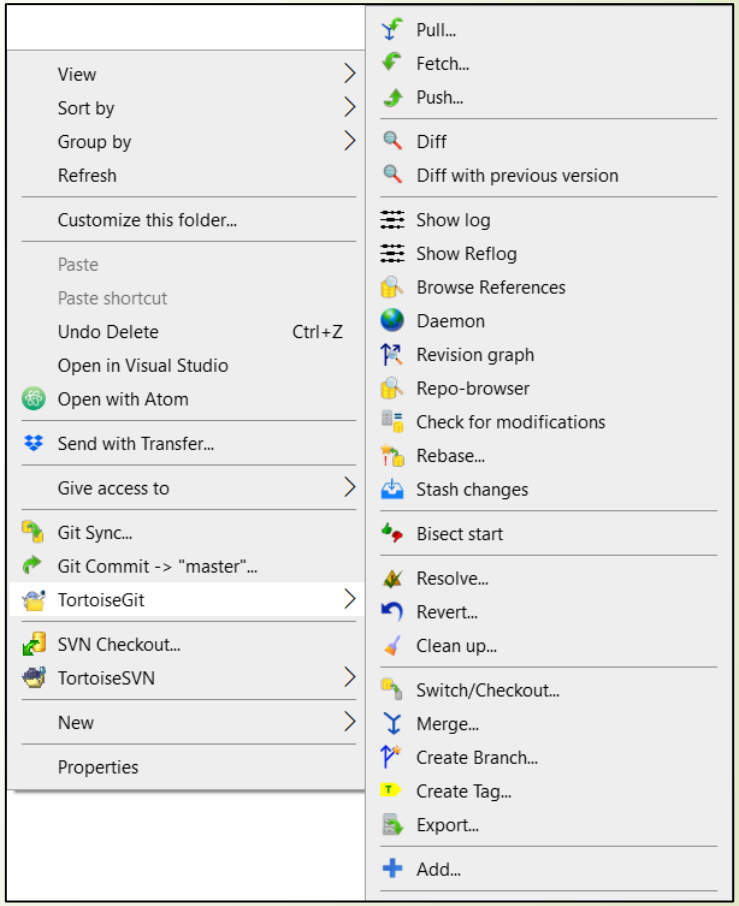




Verziókövető rendszerek

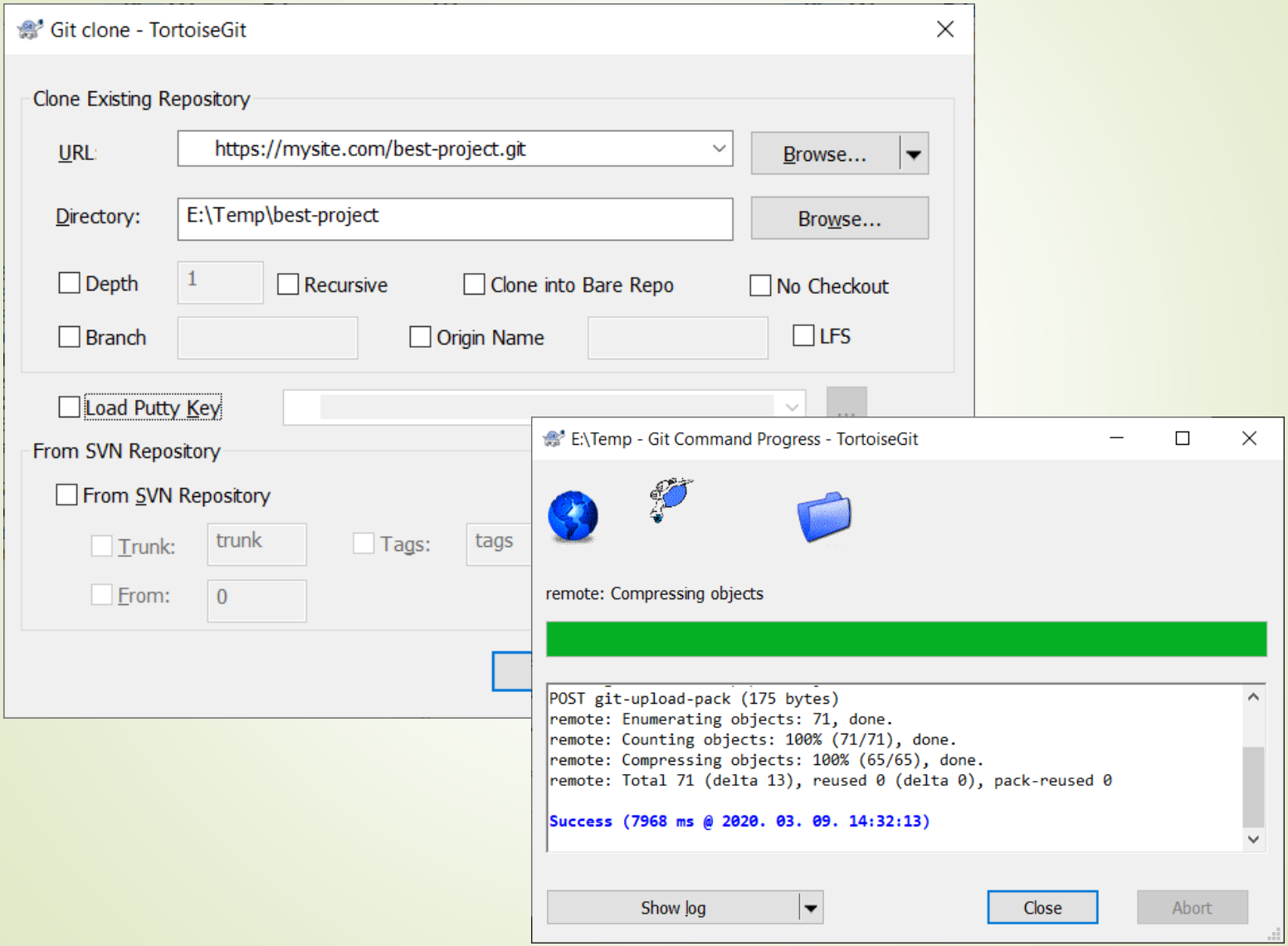
TortoiseGit használata

- A TortoiseGit egy Windows rendszekre fejlesztett, ingyenes, asztali grafikus Git kliens
  - Honlap, letöltés: https://tortoisegit.org/
  - Elérhető magyar nyelven is.
  - A Git-et nem tartalmazza, azt külön szükséges telepíteni.
- A TortoiseGit a fájlkezelő alkalmazások (pl. *File Explorer*) jobb egérkattintásra elérhető kontextus menüjébe épül be, innen hívhatóak meg a már megismert utasítások.



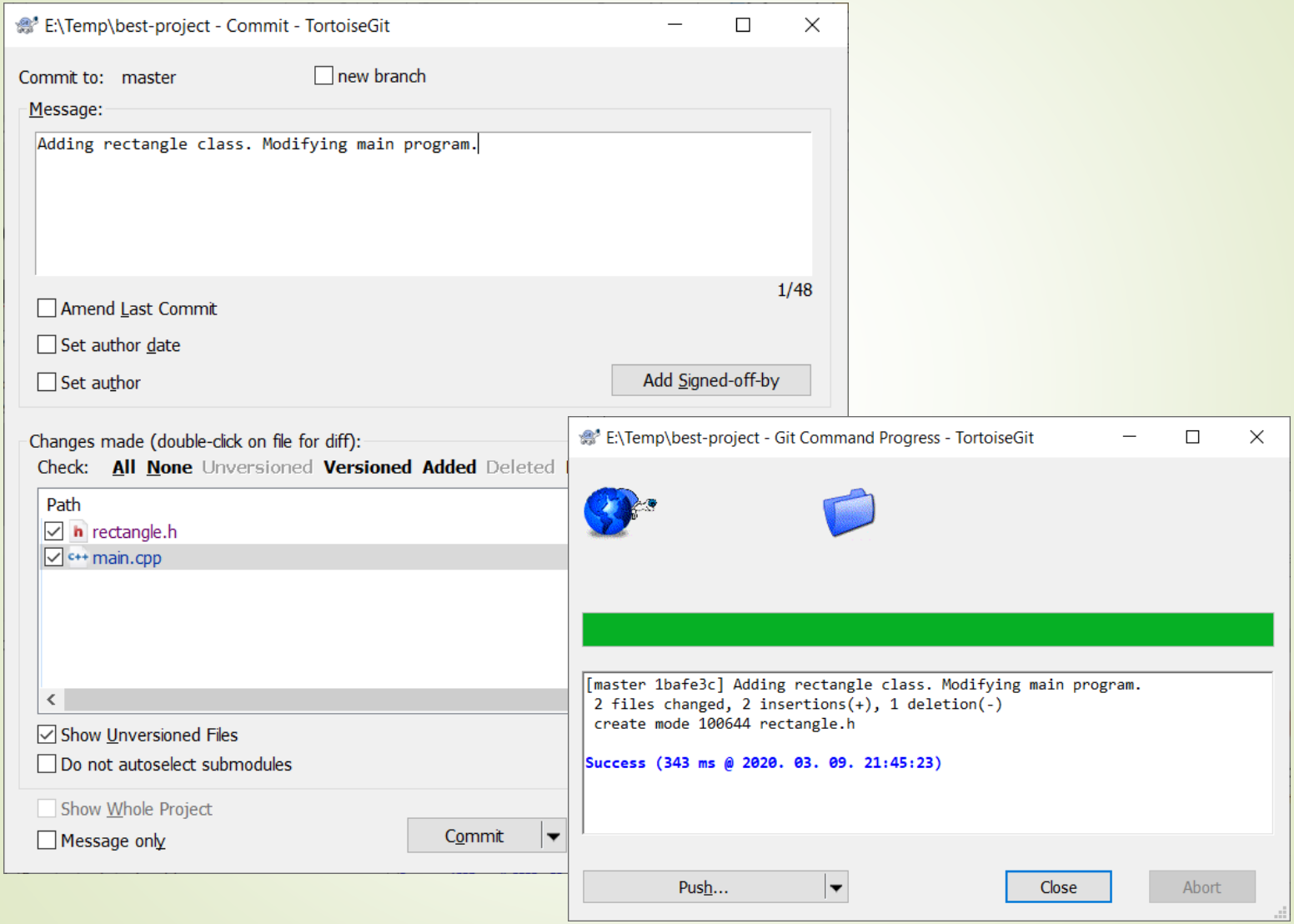
Verziókövető rendszerek

TortoiseGit: klónozás (*clone*)



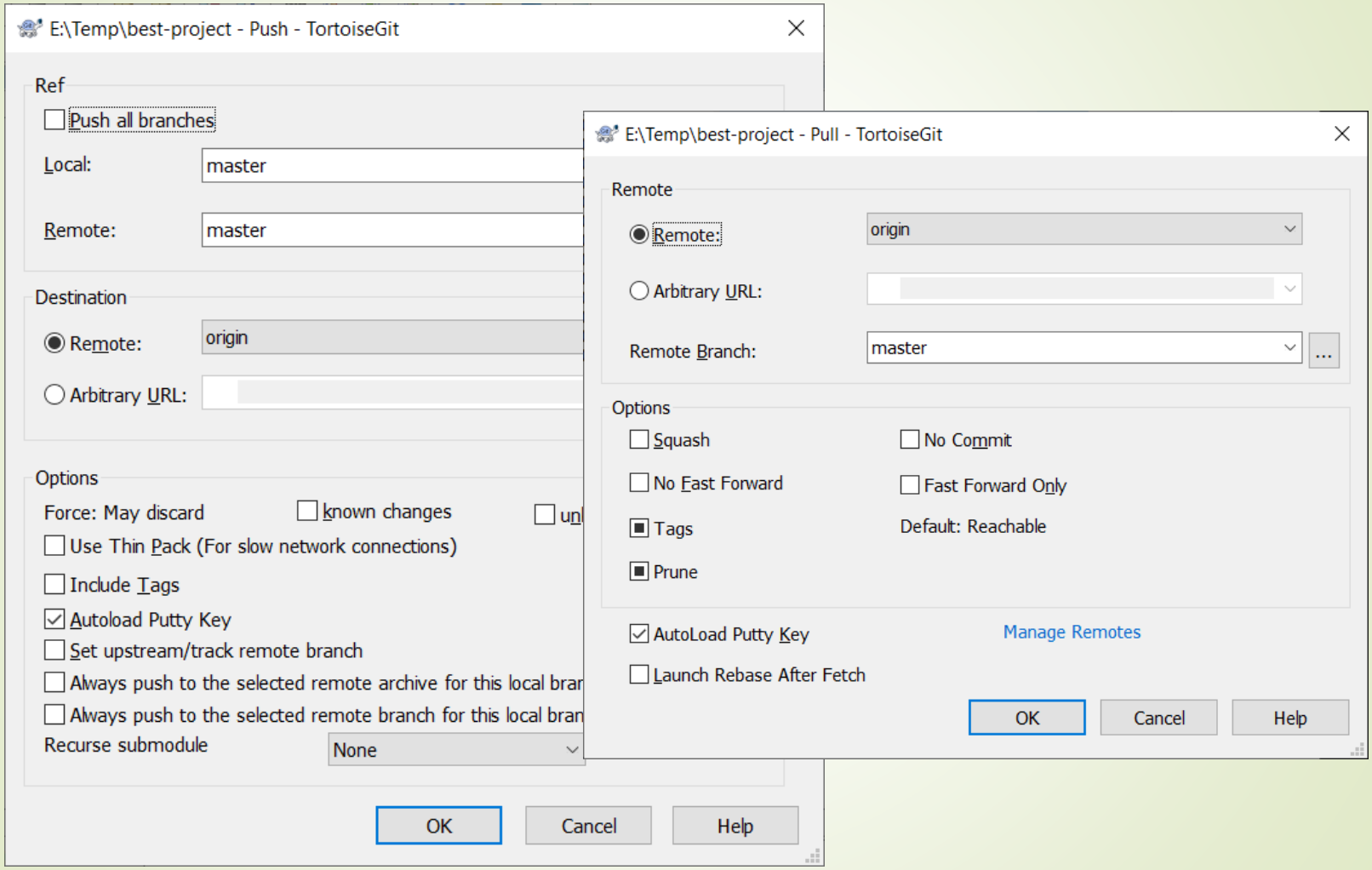
Verziókövető rendszerek

TortoiseGit: új verzió beküldése (*commit*)



Verziókövető rendszerek

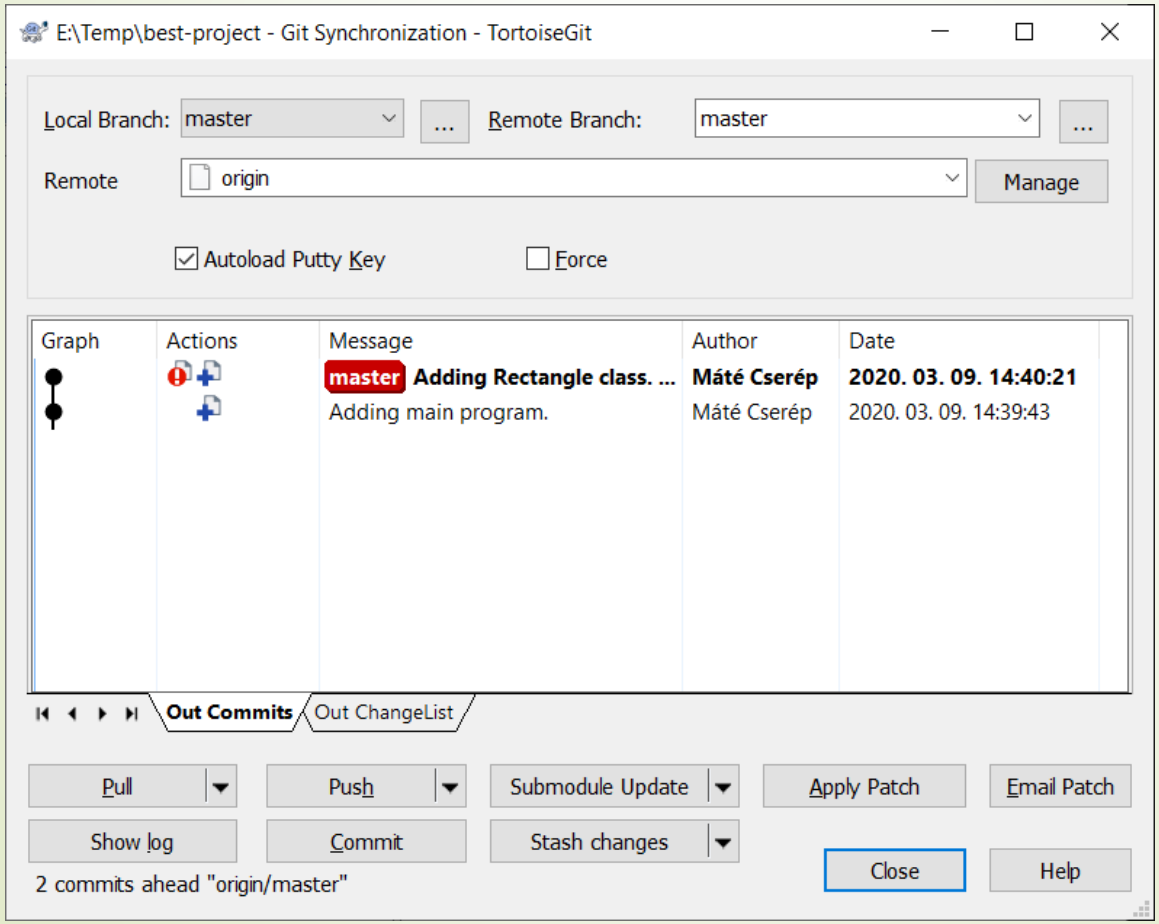
TortoiseGit: szinkronizálás (*push* & *pull*)



Verziókövető rendszerek

TortoiseGit: szinkronizálás (*sync*)

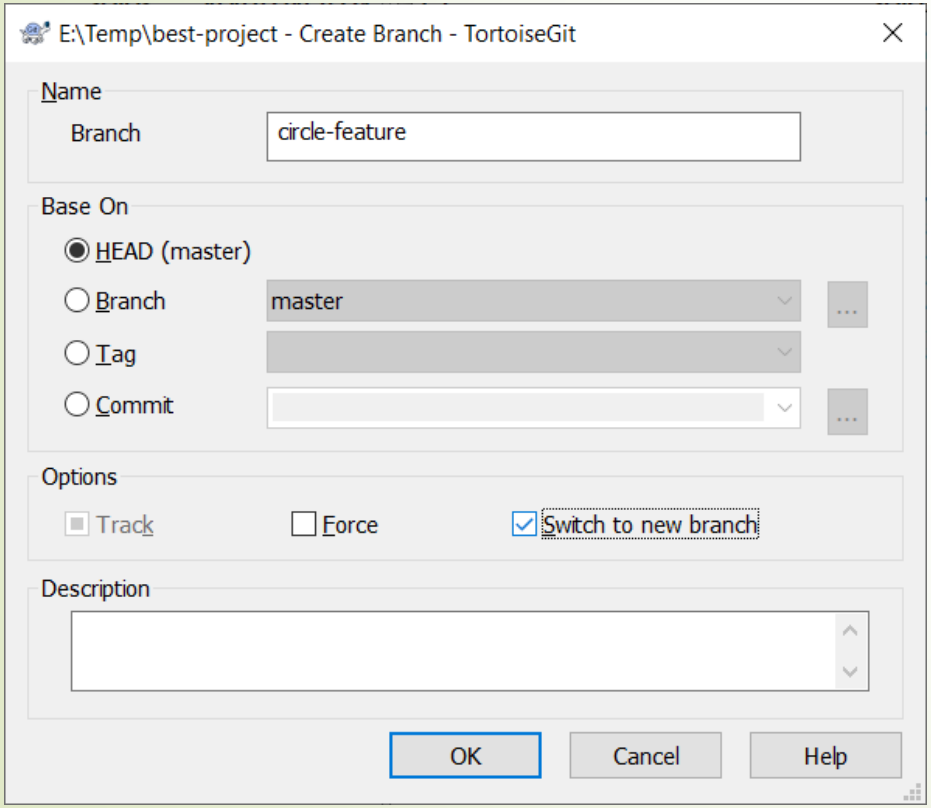
- A *Sync* funkcióval egyszerre érhetjük el a *pull* és a *push* opciókat is egy áttekintő felületen.

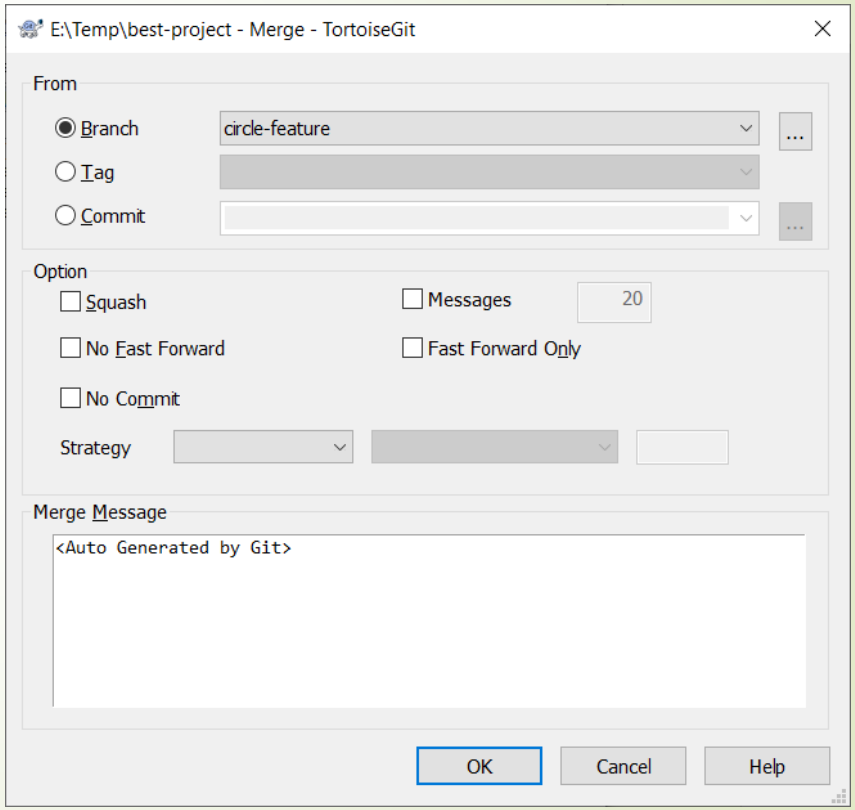


Verziókövető rendszerek

TortoiseGit: fejlesztési ágak (*branch*) kezelése

- Eleinte a grafikus felület jelentősen könnyebbé teheti a fejlesztési ágak létrehozását és összeillesztését.

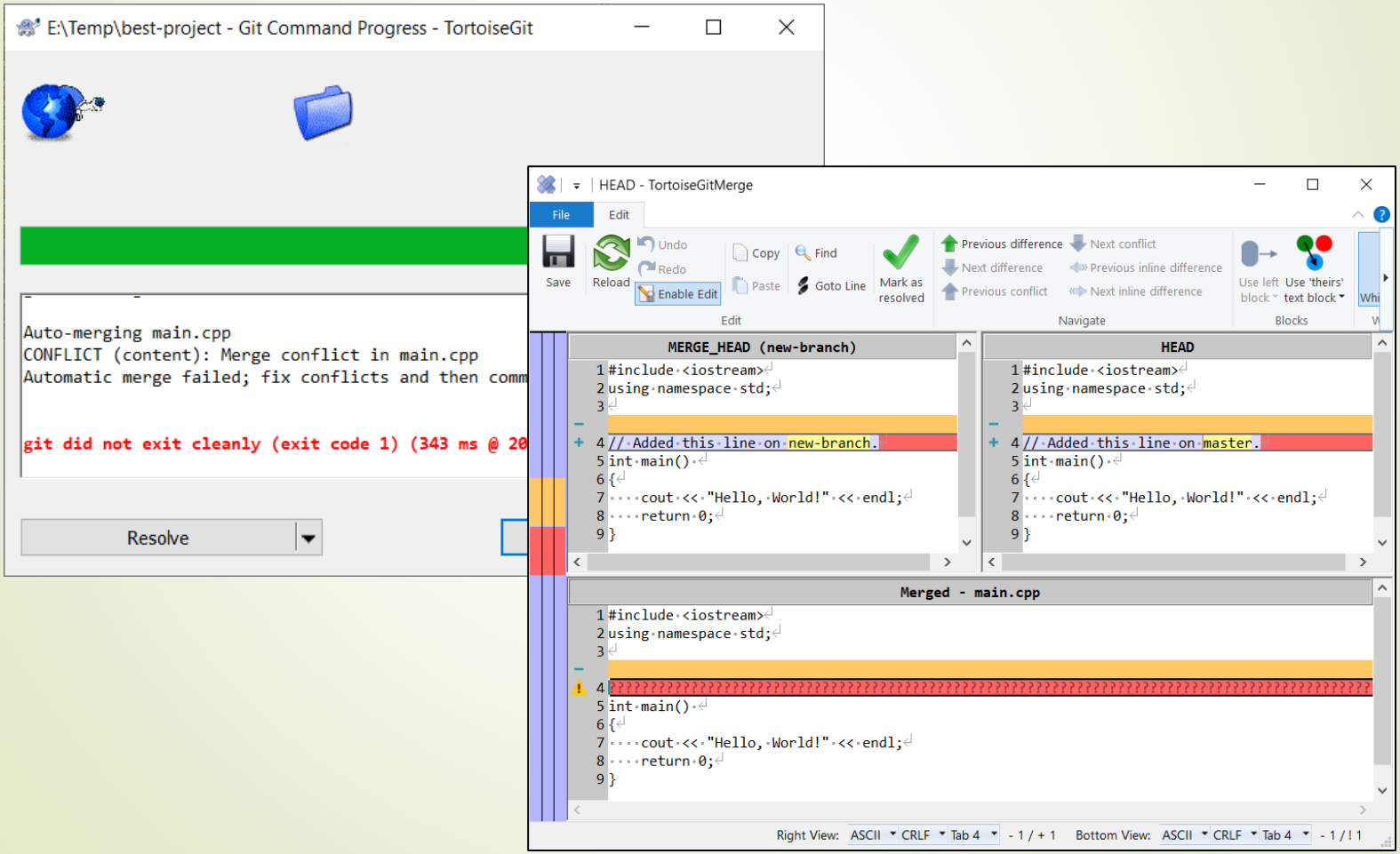




Verziókövető rendszerek

TortoiseGit: ütközések feloldása

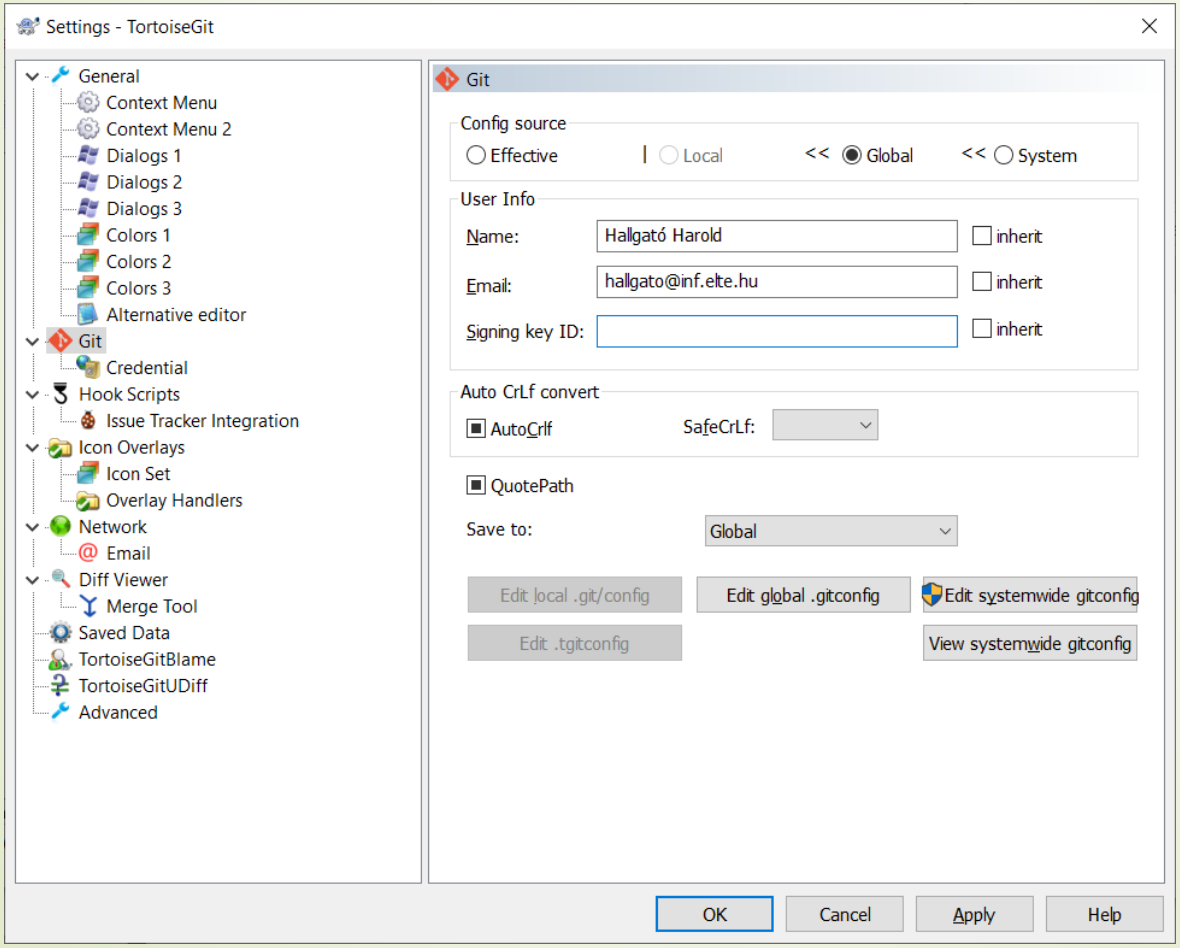
- Különösen igaz ez az egyesítéskor fellépő konfliktusok feloldására, ahol egy összevető nézet segíti a munkát.



Verziókövető rendszerek

TortoiseGit: beállítások

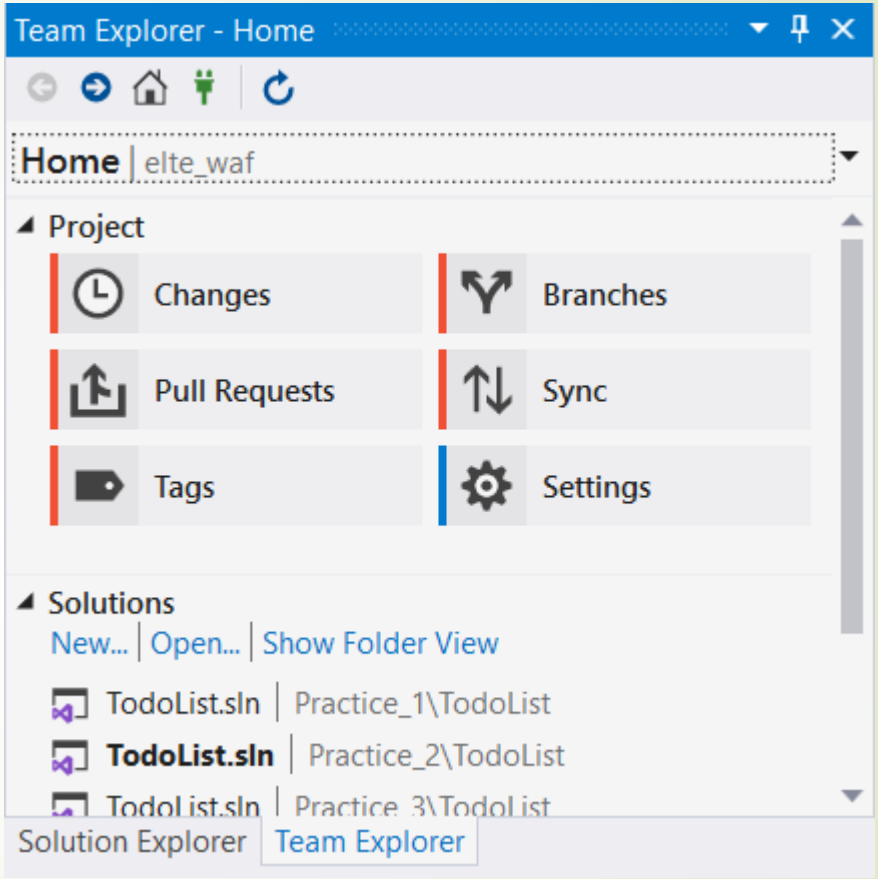
- *TortoiseGit -> Settings* menüpont alatt szerkeszthetjük a beállításokat is, pl. a felhasználó nevét és email címét.



Verziókövető rendszerek

Támogatás integrált fejlesztőkörnyezetekben

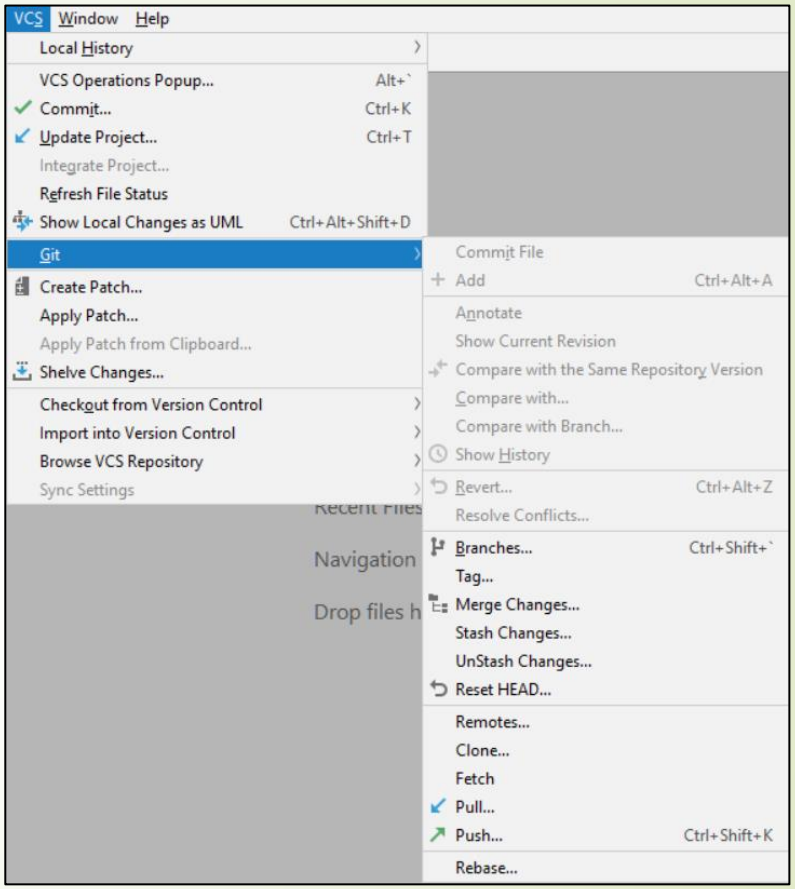
- A legtöbb modern integrált fejlesztőkörnyezet (IDE) beépített támogatást nyújt a verziókezelésre.
  - Részben éppen emiatt nevezzük integrált környezetnek őket.
  - Így nem szükséges különféle alkalmazások kontextusai között váltogatni.
- *Visual Studio* esetében a verziókezelő funkciókat a *Team Explorer* menüpont alatt találjuk.

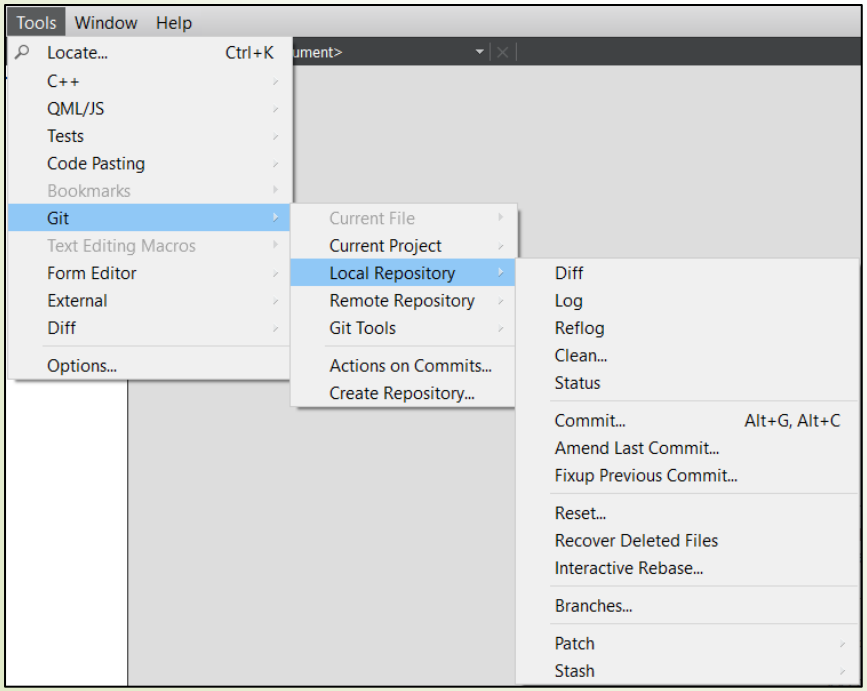


Verziókövető rendszerek

Támogatás integrált fejlesztőkörnyezetekben

- *JetBrains CLion* a funkciókat a VCS (*version control system*) menüpont alatt találjuk.
- QtCreator programban ugyanez a Tools -> Git menüpont alatt érhető el.





Verziókövető rendszerek

Mely fájlokat érdemes verziókezelés alá vonni?

- A Git verziókezelő rendszer a szöveges állományok, így tipikusan a forráskód fájlok változáskezelésében hatékony. Elsődlegesen a projekt forráskódját érdemes benne elhelyezni.
- Egy általános szoftver projekt esetén nem érdemes verziókezelés alá vonni:
  - fordítás során előálló köztes tárgykódot vagy a végső bináris állományokat, mert újból előállíthatóak, folyamatosan változnak és ütközéseket okoznak.
  - fejlesztő eszközök személyes beállításait (pl. Visual Studio esetén a *.vs/* vagy Netbeans esetén a *nbproject/private/* könyvtárak), amelyek felhasználónként eltérőek.
  - nagy méretű bináris állományokat (pl. videók, nagy méretű képek), amelyek kezelésében a Git nem hatékony. Bár a Git tárolók mérete jól skálázható, egy könnyen kezelhető *repository* mérete az 1-2 GB-os méretet nem haladja meg.

Verziókövető rendszerek

Fájlok kivonása a verziókezelés alól

- Verziókezelés alá kizárólag azok a fájlok kerülnek, amelyeket kifejezetten hozzáadunk (`git add`).
- Az esetleges véletlen hozzáadást elkerülendő megjelölhetjük azokat a fájlokat és könyvtárakat, amelyeket mellőzni szeretnénk.
  - A mellőzendő állományokat egy speciális `.gitignore` elnevezésű állományban adhatjuk meg
  - Ezt a fájlt érdemes verziókezelés alá is vonni, hogy a fejlesztők között egységes legyen a beállítás.
- A `.gitignore` minden sorában egy illeszkedési mintát adhatunk meg, hogy mely fájlokat akarjuk kizárni a verziókezelés alól.
  - A beállítás tranzitívan vonatkozik az alkönyvtárakra is, így gyakran elegendő lehet egyetlen `.gitignore` fájl létrehozása a projekt gyökér- könyvtárában.

Verziókövető rendszerek

Git: .gitignore minták

Minta	Illeszkedés	Leírás
program.exe	/program.exe /bin/program.exe	Minden **program.exe** fájlra illeszkedés.
/program.exe	/program.exe **de nem:** /bin/program.exe	Az adott könyvtárszinten lévő **program.exe** fájlra illeszkedés.
*.exe	/program.exe /bin/main.exe	Minden **exe** kiterjesztésű fájlra illeszkedés.
bin/	/bin/ /project/bin/ **de nem:** /logs/bin (fájl)	Minden **bin** könyvtárra illeszkedés (de **bin** nevű fájlokra nem!)
/bin/*.exe	/bin/program.exe /bin/main.exe **de nem:** /bin/program.dll /project/bin/program.exe	Az adott könyvtárban lévő **bin** könyvtárban lévő összes **exe** kiterjesztésű fájlra illeszkedés.

Verziókövető rendszerek

Git: .gitignore minták

Minta	Illeszkedés	Leírás
*.log !important.log	/application.log /logs/application.log **de nem:** /logs/important.log	Az összes **log** kiterjesztésű fájlra illeszkedés, kivéve, ha a fájl neve **important.log**.
logs/**/*.log	/logs/application.log /logs/runtime/main/01.log /project/logs/deploy.log **de nem:** /runtime.log	Minden olyan **log** kiterjesztésű fájlra illeszkedés, amely egy **logs** könyvtár alatt helyezkedik el (tetszőleges mélységben).

- Számos programozási nyelvhez és IDE-hez érhető el általános esetekre megfelelő `.gitignore` állomány a *GitHub*-on, ezekből vagy ezek kombinációjából jó ötlet kiindulni.
  - URL: https://github.com/github/gitignore

Verziókövető rendszerek

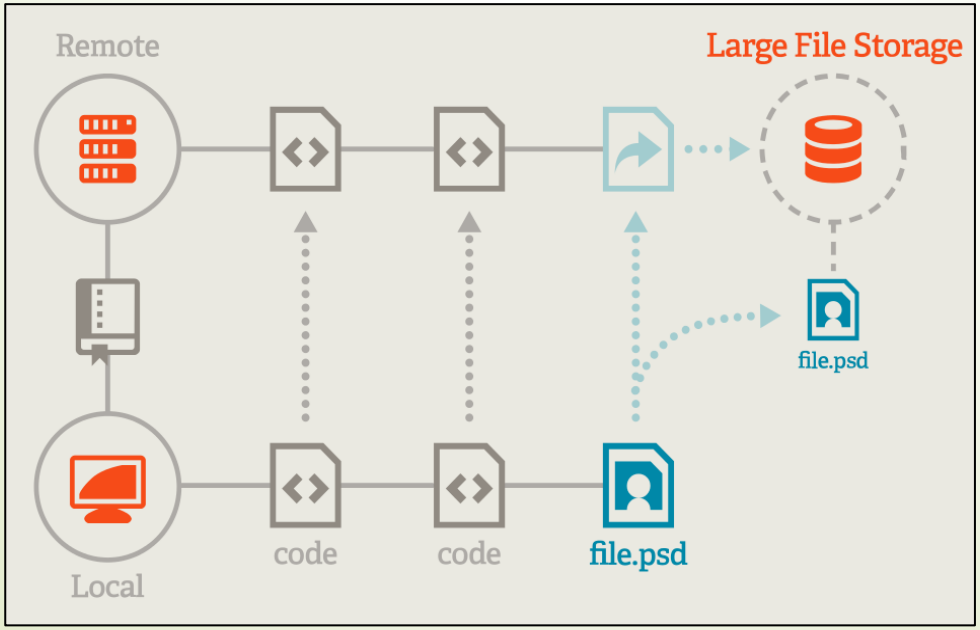
Nagy erőforrás állományok verziókezelése

- A nagy méretű videó, kép és hang erőforrás állományok hatékony kezelése körültekintést igényel.
  - A nagy méretű bináris állományok változásainak kezelésében a Git kevésbé hatékony.
  - Jelentősen megnöveli a tároló helyi másolatának lekéréséhez szükséges hálózati forgalmat (*git clone*).
  - Egy fejlesztőcsapatban a programozóknak nem feltétlenül van szükségük a fejlesztéshez a designerek által készített *assetekre*.
- Ezért a nagy méretű bináris erőforrás állományokat még akkor sem feltétlenül érdemes a Git tárolóban elhelyezni, ha amúgy ritkán változnak.

Verziókövető rendszerek

Git Large File Storage

- A nagy méretű bináris állományok kezelése a *Git Large File Storage* (Git LFS) segítéségével oldható meg.
  - A nagy méretű bináris állományokat egy hivatkozással helyettesíti és magukat a fájlokat egy másik (akár távoli) szerveren tárolja.
  - Így a Git tárolónk mérete kezelhető marad.



Forrás: git-lfs.github.com

Verziókövető rendszerek

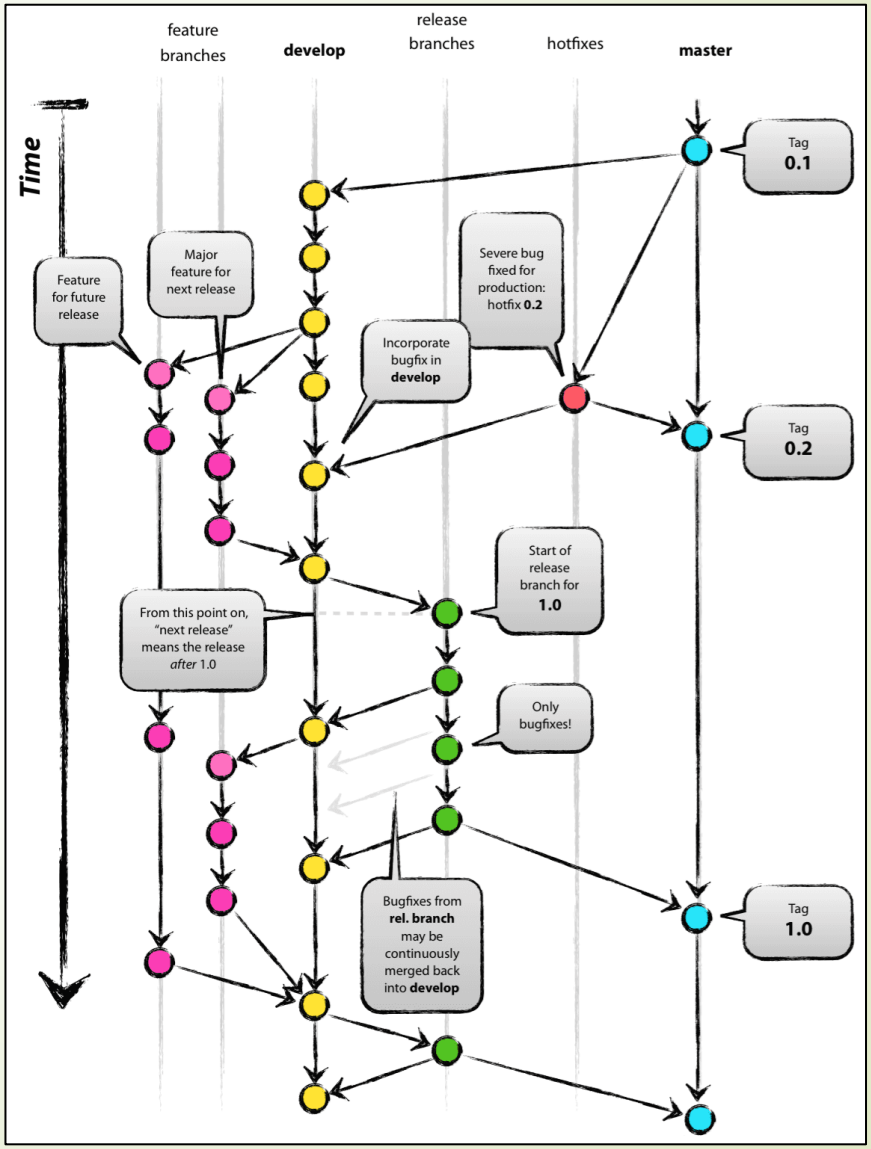
Git Large File Storage

- A szofttech.inf.elte.hu GitLab szerver támogatja a Git LFS-t.
- Használatához csak a kliens gépekre (saját gépetek) is szükséges a Git LFS telepítése.
  - Letöltés: https://git-lfs.github.com/
  - Használat: https://docs.gitlab.com/ee/administration/lfs/manage_large_binaries_with_git_lfs.html

Verziókövető rendszerek

Gitflow feature branching model

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



Forrás: nvie.com