GIT VERZIÓKEZELÉS

GIT VERZIÓKEZELÉS
I.
verziókezelő rendszerek típusai, bevezetés a Git-be

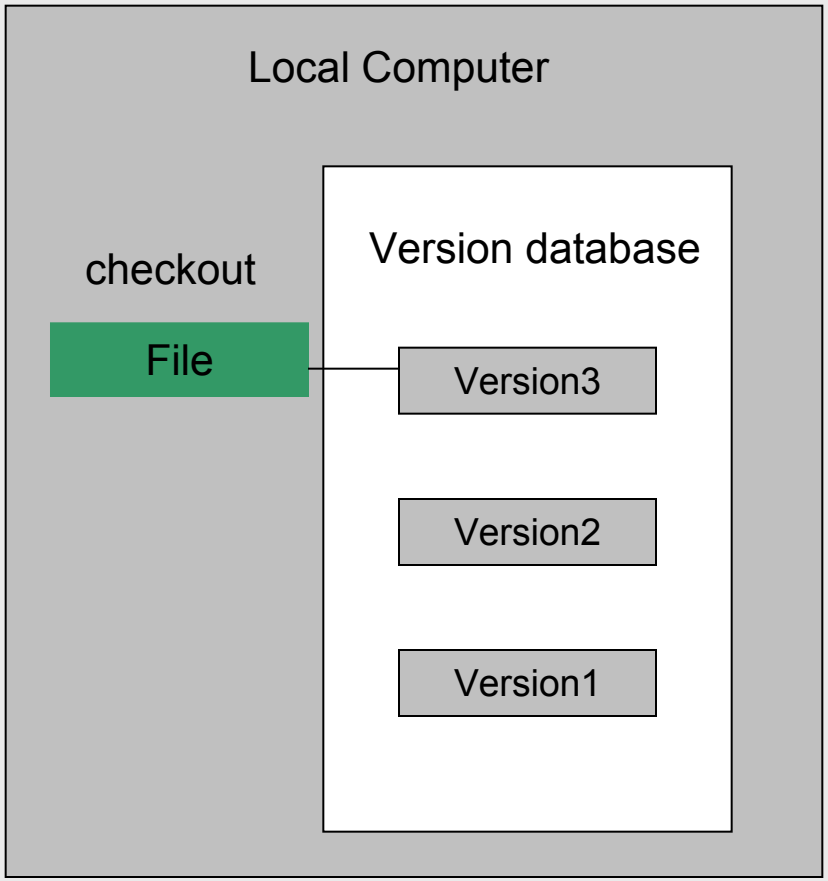
A verziókezelőkről

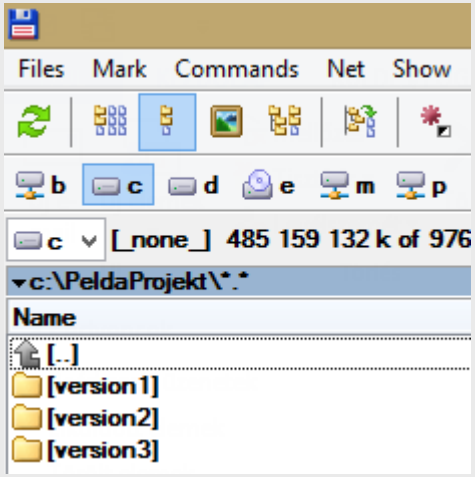
- **A verziókezelő rendszerek nagyobb, sok munkaerőt kívánó projektek fejlesztésében teszik hatékonyabbá, gyorsabbá a szoftverfejlesztők, mérnökök közötti együttműködést, lehetővé téve a projekt rendszerezett, strukturált fejlődését és a verziótörténetek visszanézését, „history” - zását. Ily módon az alkalmazás akármelyik verziójára, „időpillanatára”, bármikor vissza tudunk állni.**
- Vannak köztük fizetősek és ingyenesek is.
- Ilyen verziókezelő rendszerek:
  - CVS (Central Verification System)
  - SVN (Subversion)
  - ClearCase,
  - Mercurial, Git, TFS (Team Foundation Server) stb.

  Jelölések:
  - A diasoron használt **fogalmak, definíciók megnevezése vastagbetűvel és aláhozottan szerepelnek, a definíciós leírásuk pedig vastag betűvel.**
  - **Azokon a diákon * jel van a diacím mögött, amelyek olyan műveleteket mutatnak be, amelyek nem elengedhetetlenek a mindennapi használathoz, de alapvető műveleteknek számítanak.**
  - **Azokon a diákon ** jel van a diacím mögött, amelyek olyan műveleteket mutatnak be, melyek nem létfontosságúak a git mindennapi használatához**

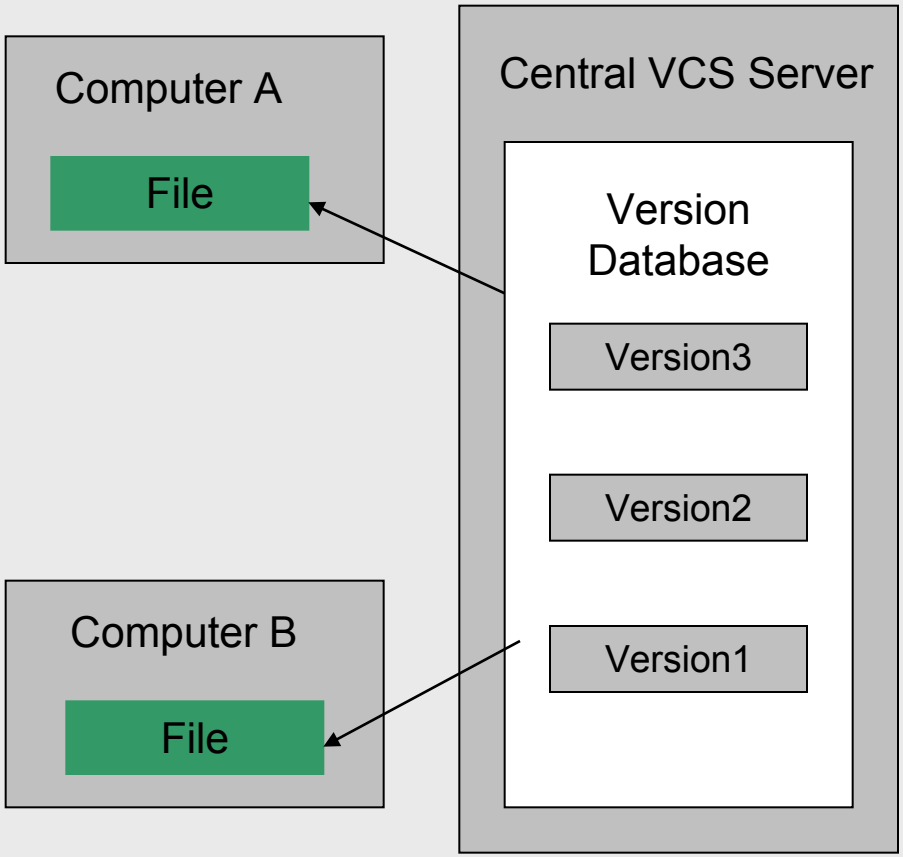
Verziókezelők típusai
Lokális verziókezelés

A programozás hőskorában még csak a lokális verziókezelés létezett a helyi gépeken





Verziókezelők típusai – Központi (Central) verziókezelő rendszerek



Központi verziókezelő rendszerek:

A verziók egy központi szerveren vannak, mindenki ide menti, vagy innen szedi le a módosításait. Hátránya, hogy hálózati kimaradás esetén senki nem tud dolgozni, a szerver meghibásodása esetén pedig elvesznek a verziók.

Az első központi verziókezelő a CVN, amelyet amerikai egyetemeken kísérleteztek ki, majd ennek utódja az SVN, amely **1986-**ban jelent meg. Nyílt forráskódú, és a központi verziókezelők közül azóta is a legnépszerűbb.

Verziókezelők típusai – Elosztott (Distributed) verziókezelő rendszerek

Server Computer

Version database

version3

version2

version1

Computer A

File

Version database

version3

version2

version1

Computer B

File

Version database

version3

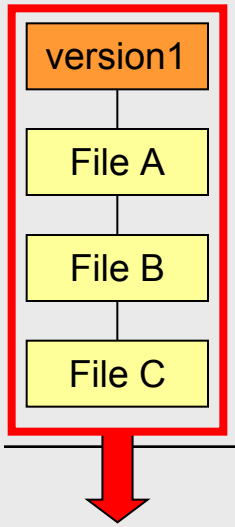
version2

version1

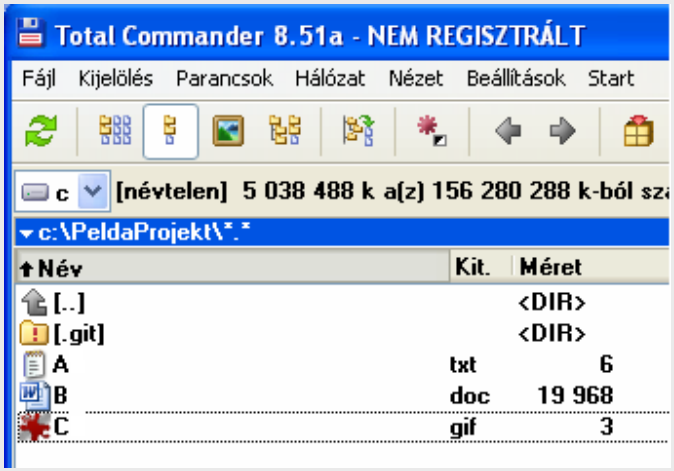
Miért pont a Git?

- Sokféle verziókezelő rendszer létezik (pl.: ClearCase, ami csak fájlokat verzióz), és ezek között sok fizetős.
- **Az elosztott verziókezelő rendszerek decentralizált verziókezelő rendszerek, ahol minden gép a központi adatbázis egy külön adatbázisa, munkamásolata.**
- A Git az egyik elsőként megjelent elosztott verziókezelő rendszer, a 2000-es évek elején kísérletezték ki Linux programozók a Linux kernel programozása során. Egész biztos, hogy fogják még fejleszteni.
- A legújabb technikájú, elosztott verziókezelő
- Ingyenes és nyílt forráskódú (más hasonló elosztott verziókezelő például a Mercurial).
- **2005-ös kiadása óta** töretlen népszerűségnek örvend. SVN-t , vagy más verziókezelőket is csak azért nem mindig váltják le Git-tel , mert utólag nehéz egy nagy projektet átmigrálni más verziókezelők alá, de a jövő mindenképpen a Git.

Git filozófiája



Idővonal

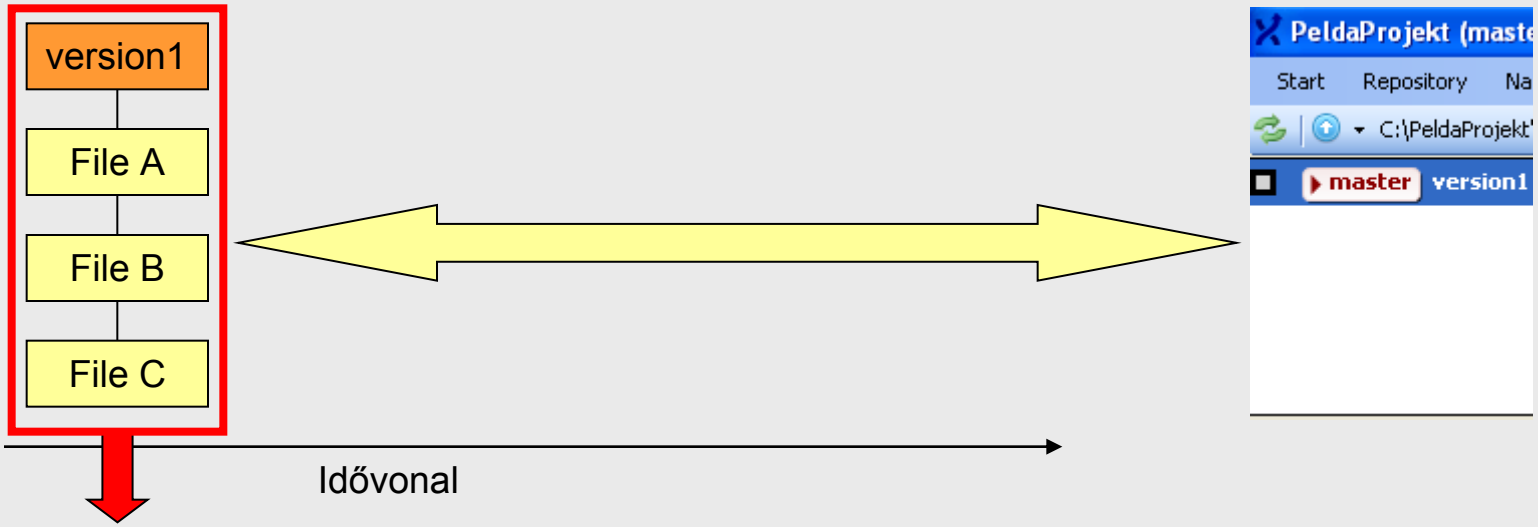


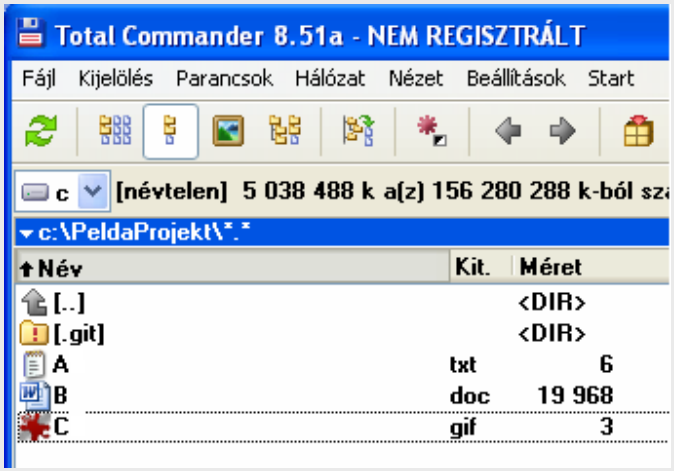
Miután feltelepítettük a Git-et és létrehoztuk az új projektkönyvtárunkat, létre tudjuk benne hozni a kezdetben üres lokális adatbázisunkat (egy .git nevű rejtett könyvtár, amibe kézzel később se szoktunk belenyúlni!). Ez azt jelenti, hogy ezután a Git-tel „monitorozzuk”, hogy mit csinálunk a projektkönyvtárunkban, mégpedig a következő módon:

elkezdünk a projektkönyvtárunkban dolgozni, majd ha úgy érezzük, hogy kész vagyunk egy logikai résszel, akkor készítünk egy pillanatfelvételt, vagy snapshot-ot. A snapshot-okból lesznek a verziók, vagy revision-ök , amelyek a .git rejtett könyvtárunk-ban lesznek időrendben egy faszerkezetben letárolva.

Fontos, hogy a Git-tel kapcsolatos kifejezéseket és a Git műveleteit (repository , commit , pull , push , stb.) sose szoktuk magyarra fordítani, mindig az angol megnevezéseivel használjuk!

Git filozófiája

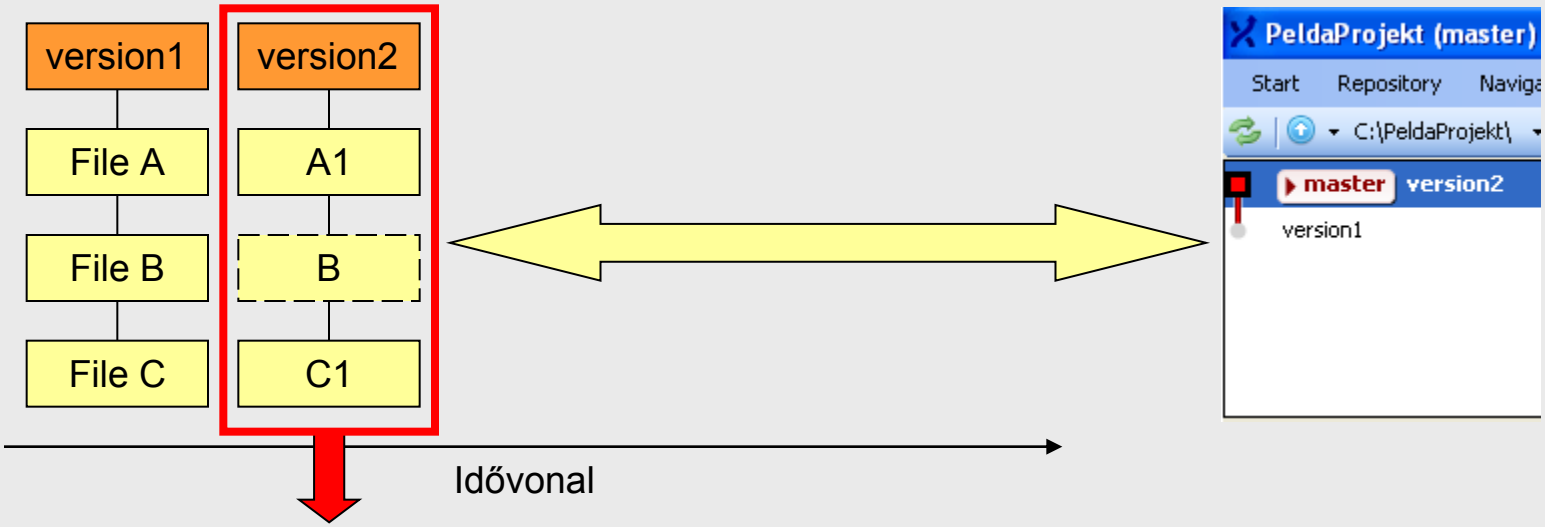


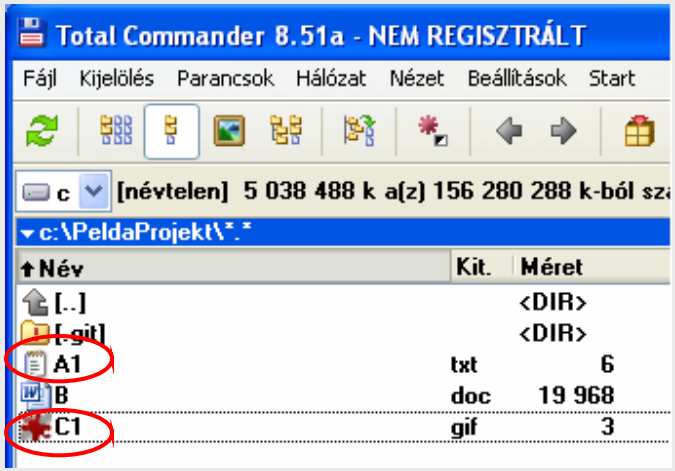


<u>Verziók vagy más néven revision-ök</u>

A projekt-munkakönyvtár (vagy más néven working directory) egy adott időpillanatáról készített, majd a commit műveletével (lásd később) lementett felvételek, más néven <u>„snapshot”-</u>ok.

Git filozófiája

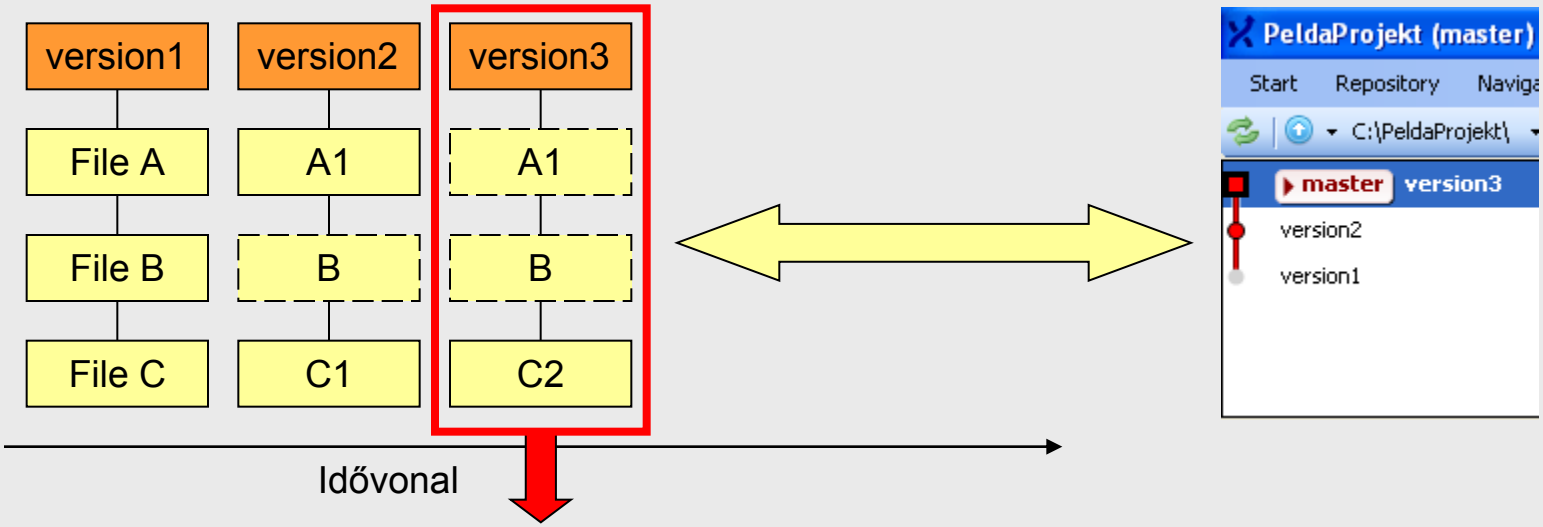


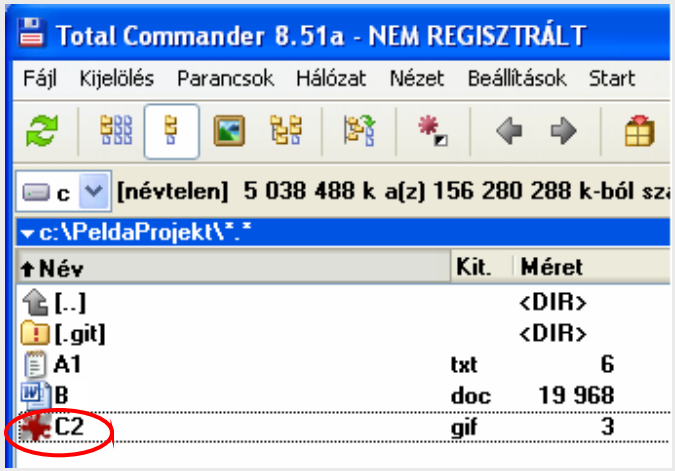


Verziók vagy más néven revision-ök

A projekt-munkakönyvtár (vagy más néven working directory) egy adott időpillanatáról készített, majd a commit műveletével (lásd később) lementett felvételek, más néven „snapshot”-ok.

Git filozófiája

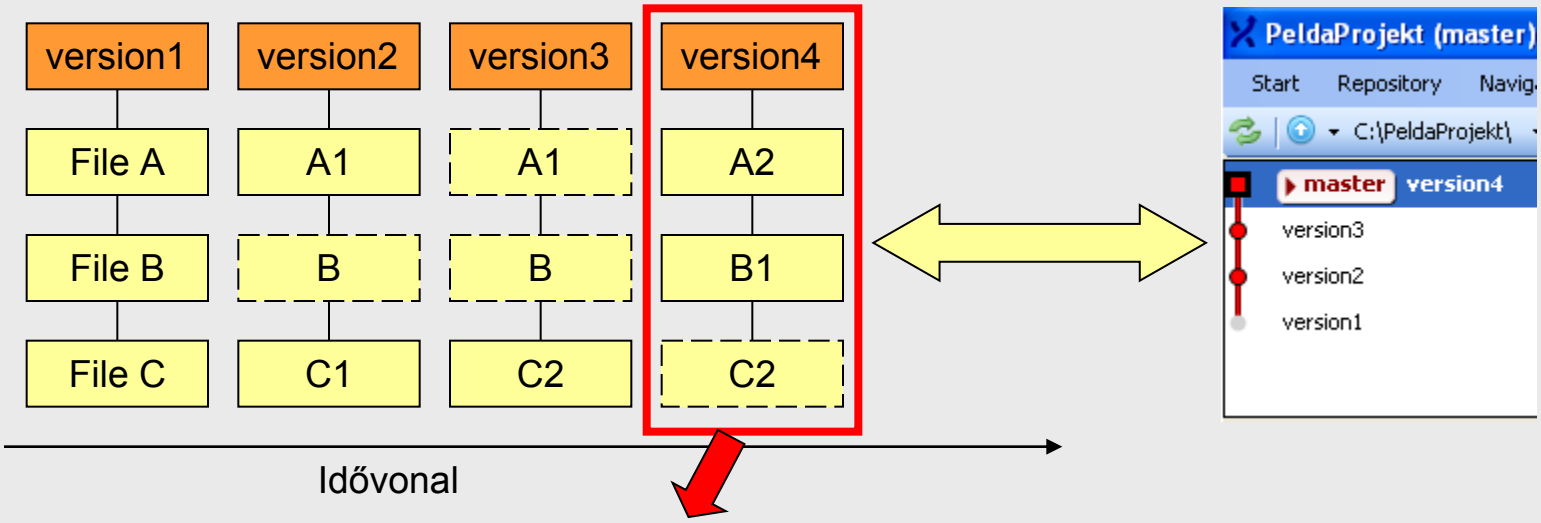


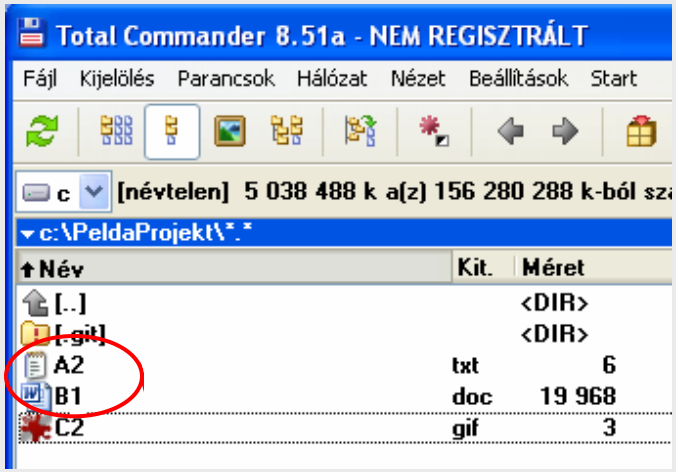


Verziók vagy más néven revision-ök

A projekt-munkakönyvtár (vagy más néven working directory) egy adott időpillanatáról készített, majd a commit műveletével (lásd később) lementett felvételek, más néven „snapshot”-ok.

Git filozófiája

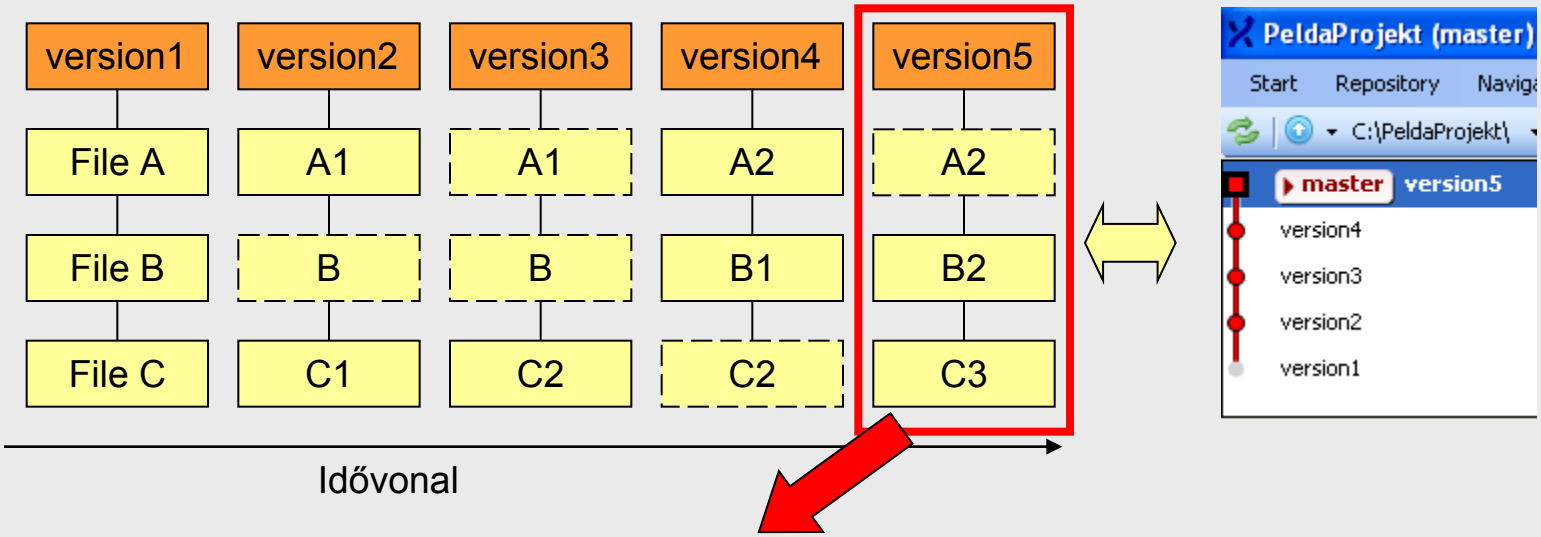


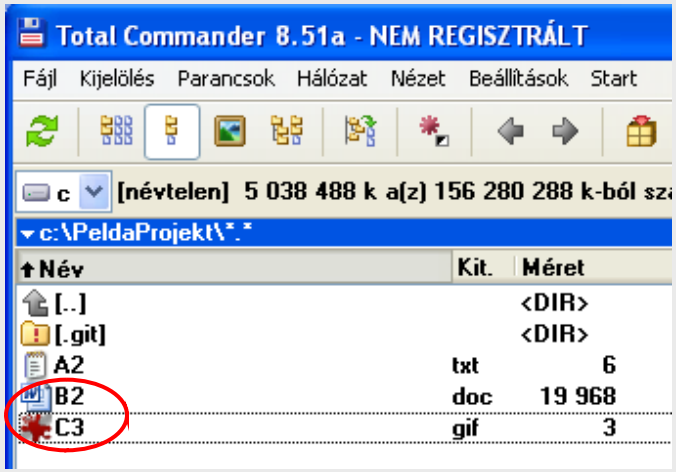


Verziók vagy más néven revision-ök

A projekt-munkakönyvtár (vagy más néven working directory) egy adott időpillanatáról készített, majd a commit műveletével (lásd később) lementett felvételek, más néven „snapshot”-ok.

Git filozófiája

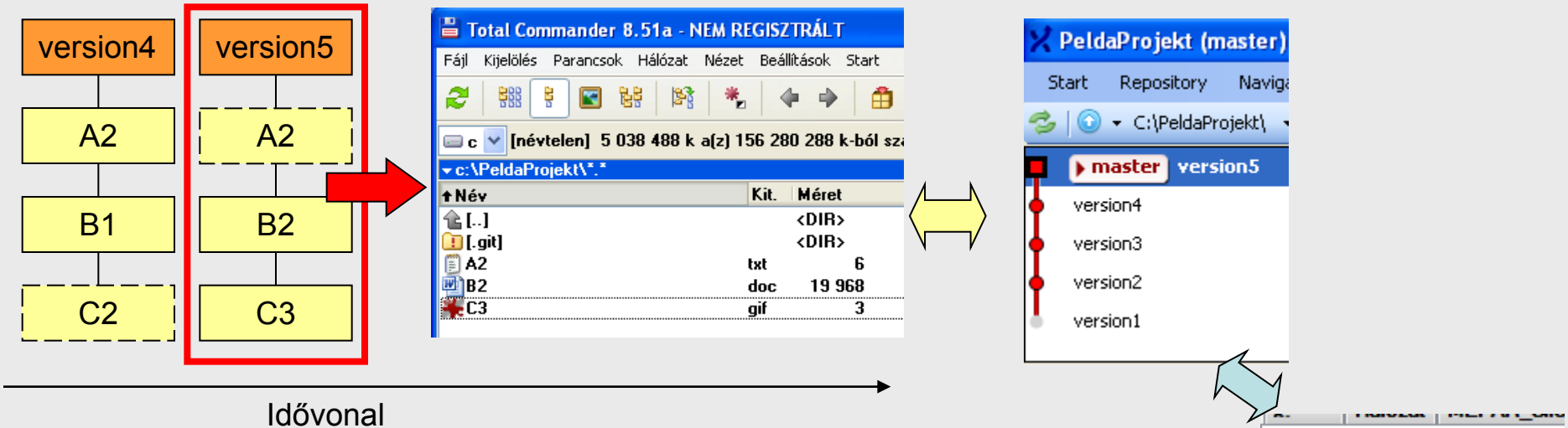


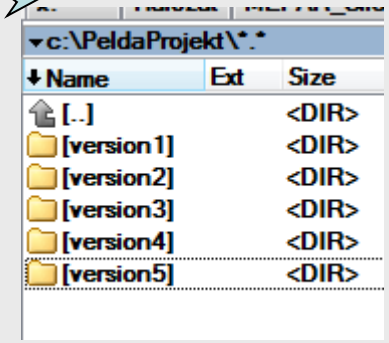


Verziók vagy más néven revision-ök

A projekt-munkakönyvtár (vagy más néven working directory) egy adott időpillanatáról készített, majd a commit műveletével (lásd később) lementett felvételek, más néven „snapshot”-ok.

Git - lokális verziókezelésre is

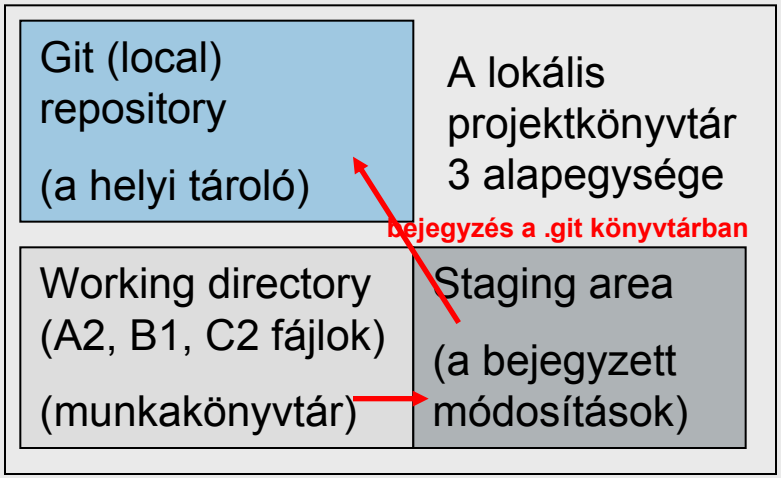




A Git ezzel a filozófiájával lokális verziókezelésre is tökéletesen alkalmas, mivel nem kell annyi könyvtárat létrehoznunk, ahány verziója van. Az új verzióban csak azok a fájlsorok vannak elmentve, amelyek módosultak, a többi fájlról pedig csak egy hivatkozás van az előző verzióra (az ábrán ily módon csak egy szaggatott vonal van például a version5-nél az A2 fájlnál, mivel ott nem történt változás. A PeldaProjekt könyvtár (és benne az 5 lementett verzió) mérete így a Git verziókezelővel: A2+B2+C3+. git könyvtár méretét jelenti, ahol a . git könyvtár egy elhanyagolhatóan kicsi szám, míg a verziókezelő nélkül (jobb oldali ábra) az 5 verzió területe = version1+version2+…+version5 könyvtár mérete, ami kb. 4-5-ször akkora területet jelent.

(Bináris fájloknál (pl.: képfájlok) egy kicsit más a helyzet, de egy garbage collector, vagy egy tömörítés kiadása után (git gc parancs) ezek is gyakorlatilag elhanyagolható méretűek lesznek az újabb verziókban.
Másrészről viszont a tárolási mód miatt a .git könyvtár tele lesz referenciafájlokkal, így a PeldaProjekt másolása kis mérete ellenére is sokkal lassabb lehet, ha a lokális adatbázisunkat (.git könyvtár) is másolni szeretnénk!

Git lokális projektkönyvtár szerkezete (local) repository



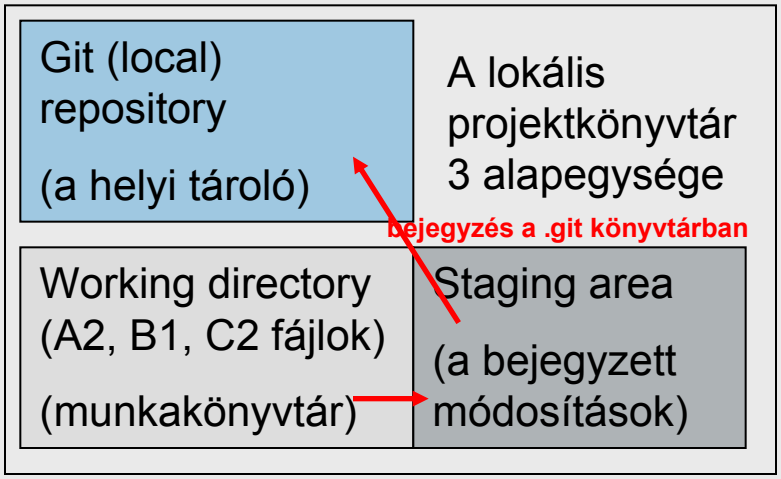
Repository fogalma:

az adatbázis, amelyben a verziók vannak.

A lokális repository a helyi gépünkön, a projektkönyvtárunkban található lokális adatbázis (.git rejtett könyvtár tartalma)

Megjegyzés: A .git egy rejtett fájl, láthatóvá tétele TotalCMD-ben a következő módon:
Beállítások-> Képernyő-> Rejtett fájlok megjelenítése

Git lokális projektkönyvtár szerkezete working directory

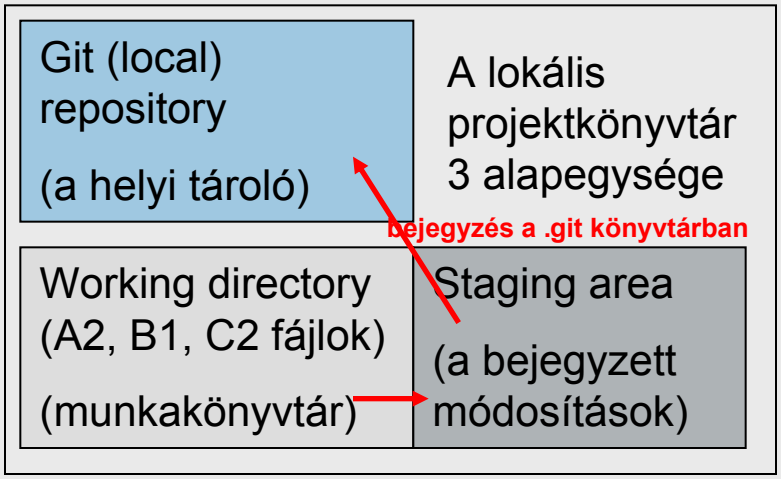


Working directory

(más néven: working tree / working copy):

Azok a projektkönyvtárban és alkönyvtáraiban lévő aktuális fájlok, amelyekkel éppen dolgozunk

Git lokális projektkönyvtár szerkezete staging area



Staging area:

Azok a módosított fájlok, amelyekről snapshot-ot készítünk, azaz bejegyezzük az adott időpillanatokban lévő fájlok aktuális állapotát a repository-ba.

Ez a snapshot majd a commit műveleténél (lásd később) fog egy új verzióként/revision-ként elmentődni a lokális adatbázisunk git hash-fájába.

Megjegyzés:

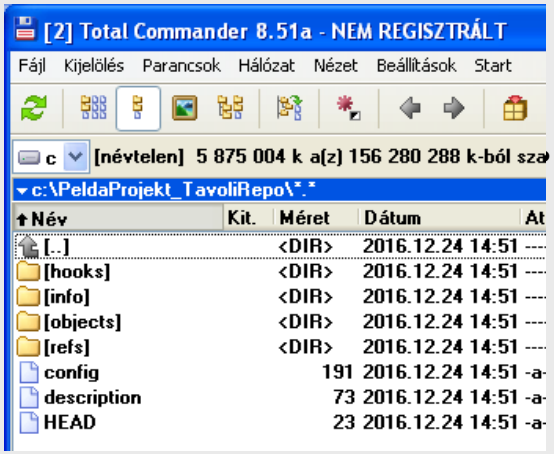
Staging Area-ra leginkább azért van szükségünk, mert ha nem vagyunk kész egy aktuális résszel, de gyorsan ki kell javítanunk valamit egy másik verzióban, akkor, ha a staging area-ba teszem az aktuális állapotomat, átállok a másik verzióra, ott megcsinálom a módosításaimat, am eyeket rögtön commit-álom is. Ezután visszatérve az előző verzióra, ott tudom folytatni a munkámat, ahol abbahagytam.

Az Amend Commit művelettel azonban lényegében kiváltható ez a rész és sokkal egyszerűbb is a használata. Lásd később!

Git központi tárhely szerkezete remote repository

Központi tárhely

git central/remote repository



Az alábbi ábrán egy fájlszerver típusú központi tárhely-et látunk, benne a remote repository tartalmával. Gyakorlatilag ugyanúgy épül fel, mit a lokális projektkönyvtárunkban található .git rejtett könyvtár, csak itt nem rejtett a könyvtár.

Repository fogalma:

Az adatbázis amelyben a verziók vannak

A központi tárhely (remote repository) lehet:

- Egy **fájlszerver** (lásd jobbra) . Ilyenkor a lokális és központi repository között az online műveletek egy teljesen nyílt csatornán mennek.

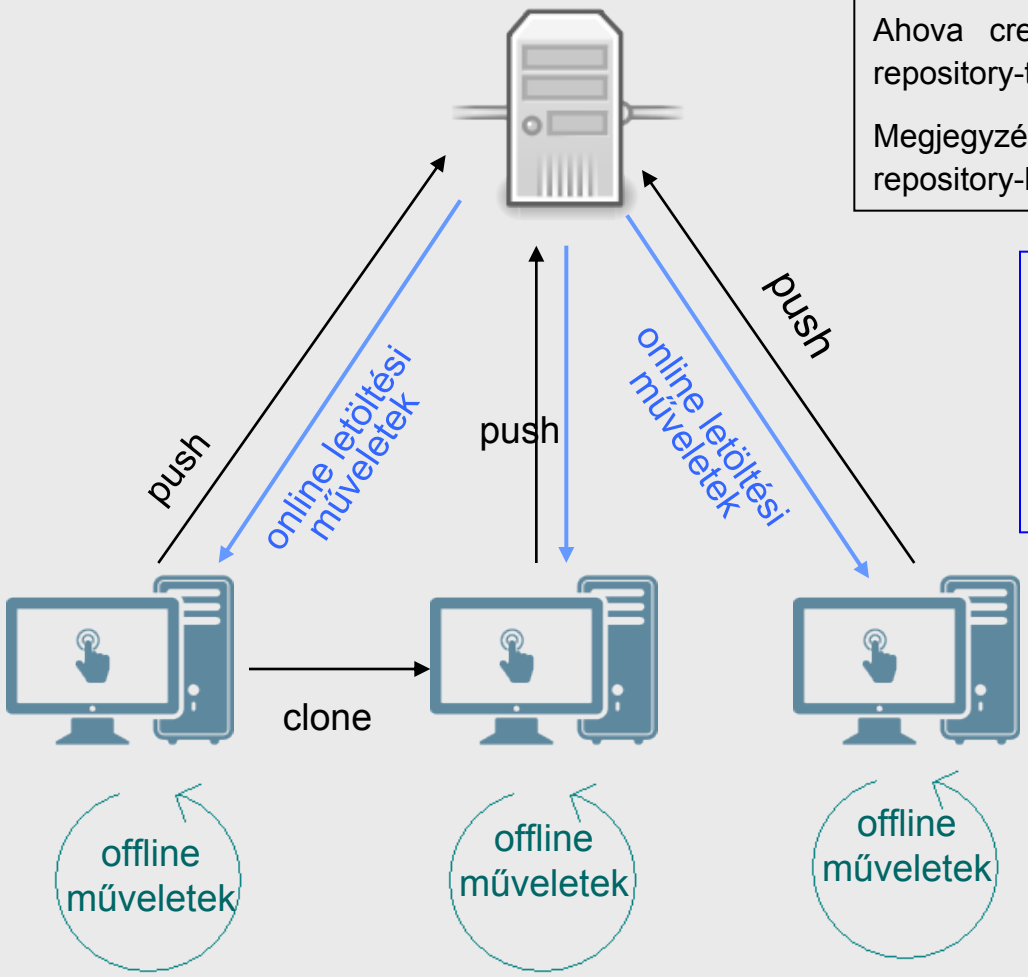
Webszerver: Ez utóbbi esetben az online műveleteket vagy http kapcsolaton keresztül, vagy egy ssh kapcsolaton keresztül, titkosított csatornán keresztül végezhetjük. Létrehozhatjuk például a GitHUB-on a projekt központi tárhelyét akár publikus , akár privát projektként (ez utóbbi esetben fizetni kell érte), vagy valamelyik gépünkön létrehozhatunk egy saját GitLab szervert, amire feltehetjük.

Git műveletek

Git szerver:

Ahova create central repository-val a central / remote repository-t létrehozzuk.

Megjegyzés: nagy projekteknél több és egyúttal többféle távoli repository-ba is push-olhatunk



Online szinkronizálási műveletek:

- **Letöltési művelek:**
  - Pull (fetch, fetch all, fetch+merge fetch+rebase (utolsó kéttőből a fetch rész)
  - Clone
- **Feltöltési műveletek: push**

Offline, lokális műveletek:

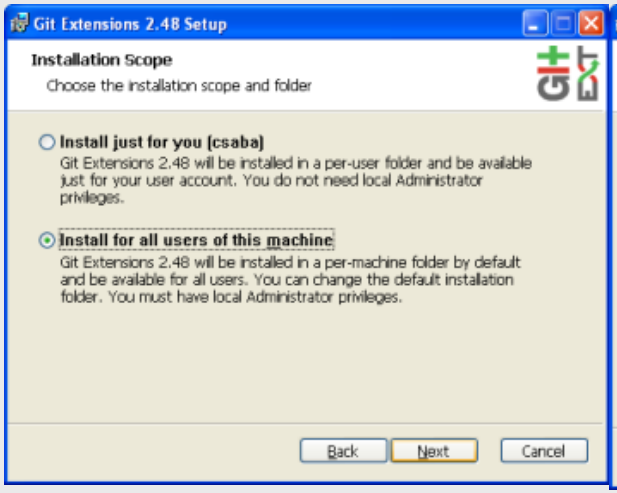
- commit, amend commit, checkout revision, stb.
- create, checkout, create new, delete, rename
- merge, revase, fast forward, reset, stb.

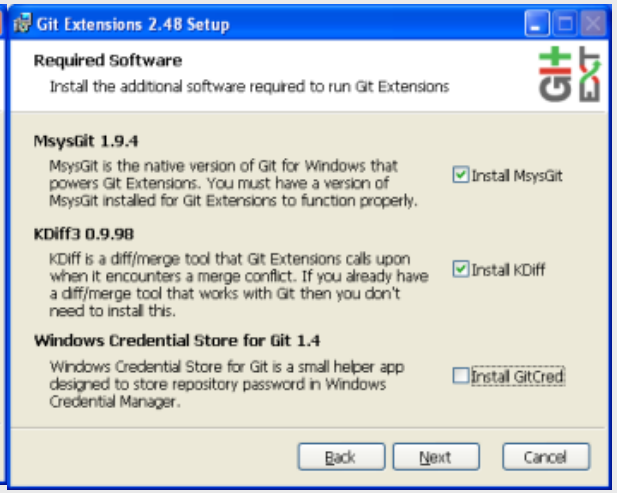
Telepítés

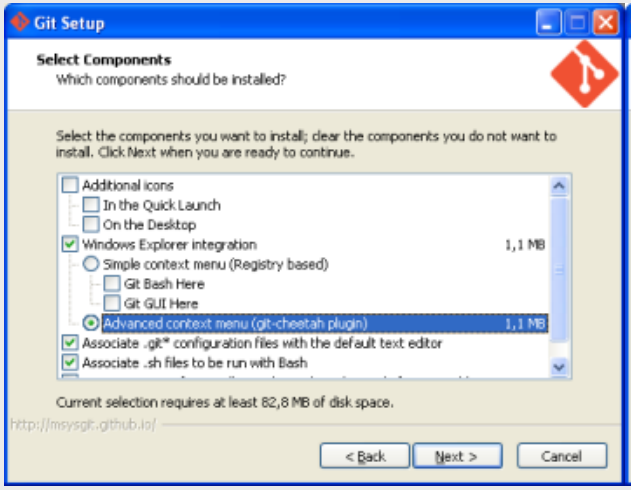
Windowsra:

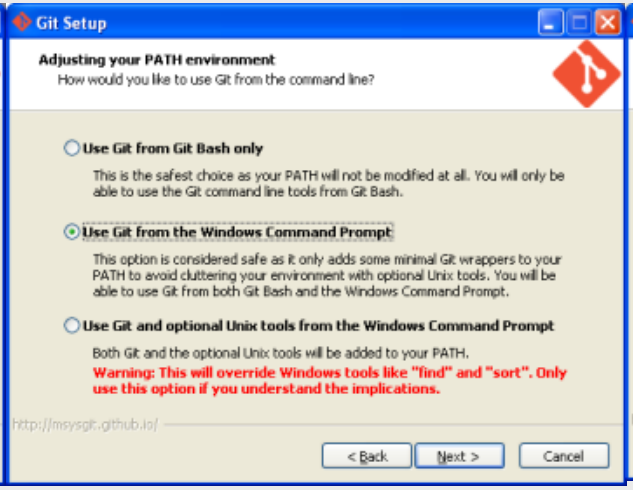
- MySysgit (Git for Windows): **Git-1.9.4-preview20140929.exe**
- KDiff3 (vagy más merge tool).
- GitExtension legutolsó verziója: (ez egy GUI) – opcionálisan feltelepíti a MySysgit-et és KDiff3-at is (2. ábra) ***GitExtensions-2.48-SetupComplete.msi***

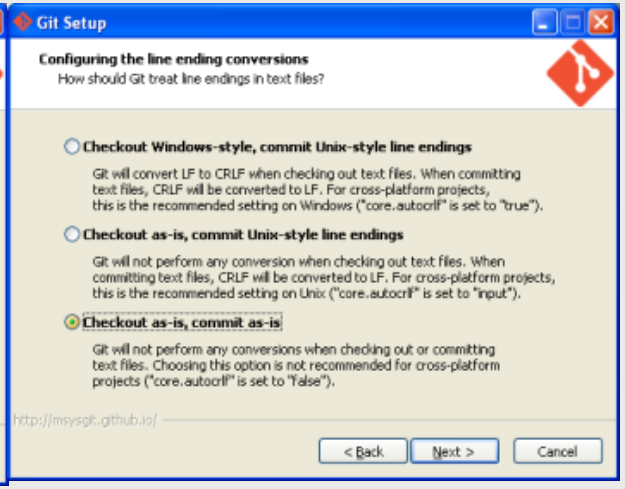
Figyeljünk a verziók és a Windows verziók közötti kompatibilitásokra.







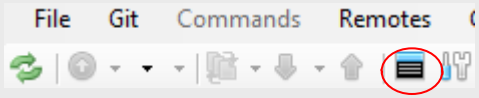




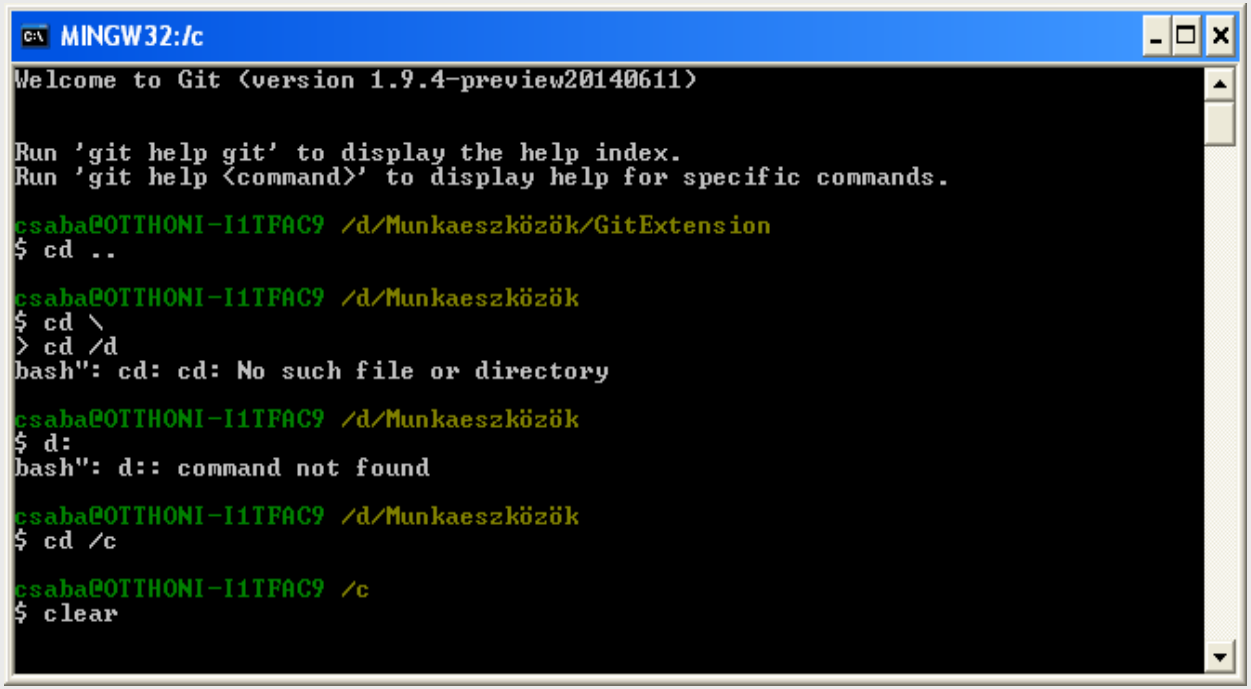
Telepítés pontjai lépésről lépésre (többségüket utólag is be lehet állítani): a 4. ábrán azt állíthatjuk be, hogy a Windows Command Promptjából (cmd) futtathatjuk a Git parancsokat, vagy magából a GitExtension-ből, vagy máshonnan

Git parancssorosan

Git Extension Browser-t elindítva:



Vagy, ha Windows Prompt-ot állítottunk be, akkor Start->Futtatás, majd írjuk be, hogy cmd



Nevet és e-mail címet a következőképp tudunk megadni:
(paramétereket érdemes idézőjelbe tenni)

git config --global user.name „Csaba”
git config --global user.email „sz.csabessz@gmail.com”

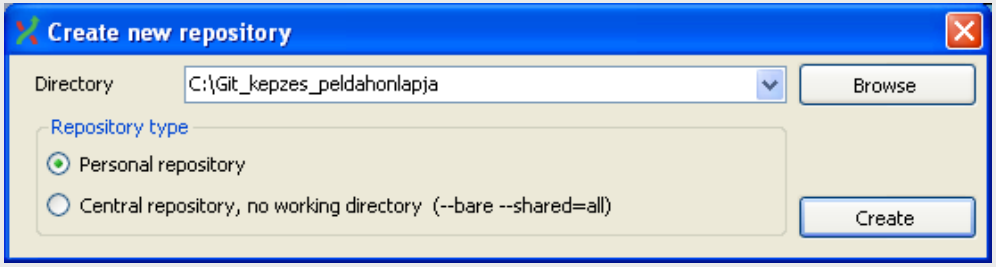
Egy commit hash-nek elég az első 5 karakterét megadni
(vagy ami egyértelmű)

git checkout 4de23398c1

GIT VERZIÓKEZELÉS
II.
Lokális műveletek

Create new repository

- **<u>Új repository bejegyzése:</u> Egy könyvtár „bejelentése” a Git számára, innentől kezdve tudjuk a könyvtár módosításait „history” - zni**
- Előfeltétel:
  - A könyvtár ne legyen repo-zva
- Ablakmegnyitás (kétféleképp)
  - Indítsuk el a GitExtension-t majd Create New Repository
- Az ablak tartalma:
  - **Directory** : melyik legyen a repositorykönyvtár
  - **Repository type**
    - Central
    - Personal (.git rejtett könyvtár)



Personal repository létrehozása egy egyszerű honlap szerkesztéséhez

<u>Parancssorosan</u>:

```
git init
c:/Git_kepzes_peldahonlapja
```

vagy:

```
mkdir Git_kepzes_peldahonlapja
cd Git_kepzes_peldahonlapja
git init
```

Commit

A Commit művelettel mentjük el a staging area aktuális tartalmát/snapshot-ját, ezzel a művelettel készítünk egy új revision-t.

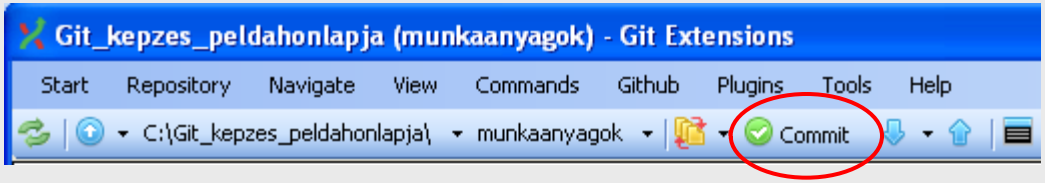
Egy revision-höz a következő paraméterek tartoznak:

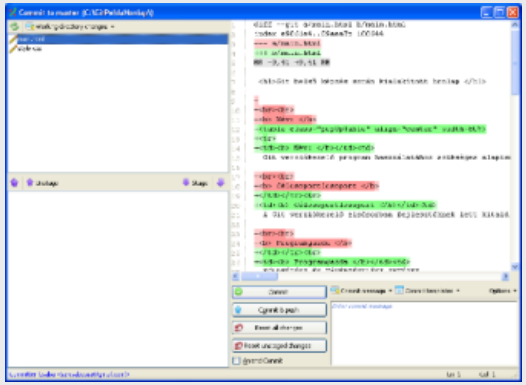
- **Egy 40 karakter hosszúságú hexadecimális karakterekből álló SHA-1 algoritmusú (Secure Hash Algorithm) hash-kód, ami teljesen egyedi, nemcsak az adott hash-fában, hanem a világegyetem összes git hash-fájában!**
- **Egy általunk beírt szöveg, azaz Commit message.**
- **A commit létrejöttének időpontja:**
- **Az új revision Parent(s), azaz szülőrevisionja(i), amely(ek)-ből létrejött (lásd példasor)**
- **Az új revision Children, azaz gyerekrevisionja(i), amelyek ebből a commit-ból fognak létrejönni. A commit során létrejött revision-nek még nincsenek gyerekrevision-jei, ezeket majd utólag fogjuk létrehozni (lásd példasor).**

A commit folyományaként létrejött revision-ökből áll össze git repository hash-gráfja, vagy hash-fája, amelynek minden csomópontja egy-egy revision-t jelöl. Ha később a revision felkerül a central repository-ba, akkor ott is ugyanez lesz minden paramétere!

Megjegyzés:

Az informatikában az MD5 kódolás mellett az SHA-1 kódolást szokták a leggyakrabban használni

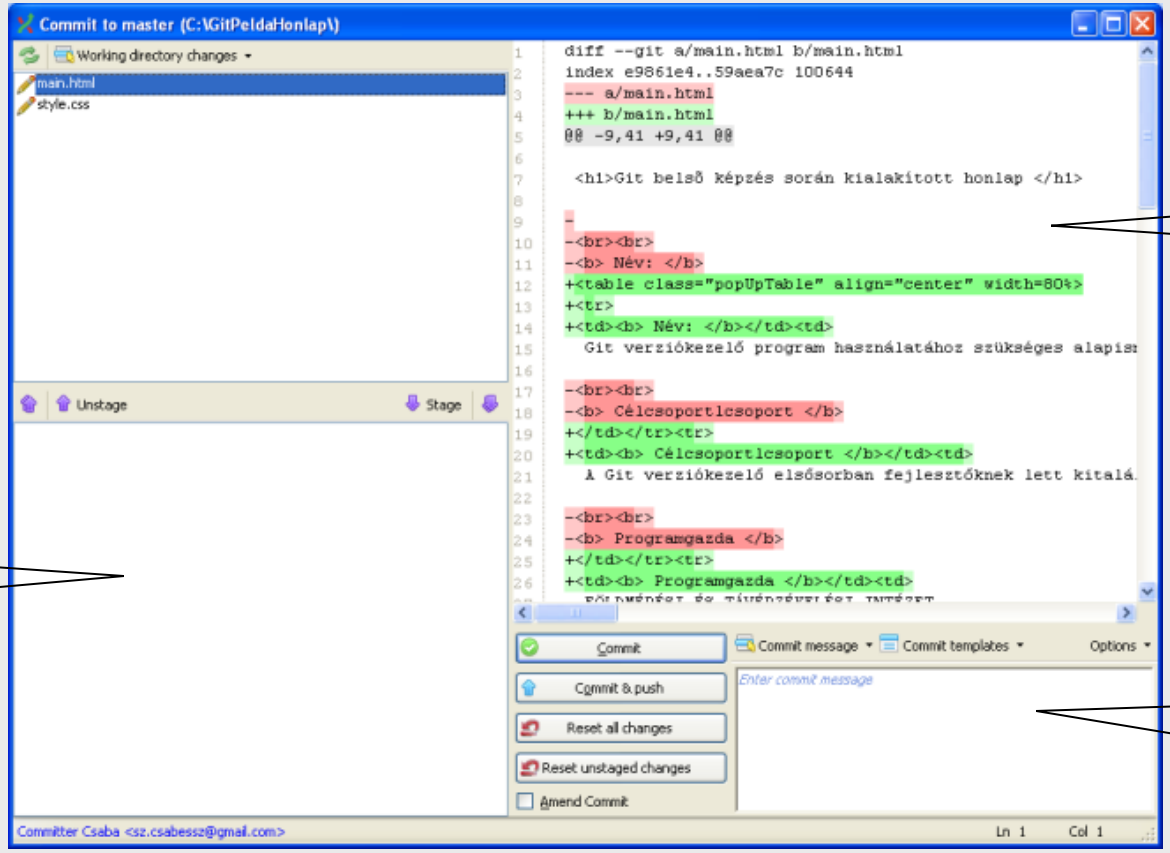




Commit

Working Directory Changes

Staging Area (Index/Cache)



Diffarea

Commit message

Commit

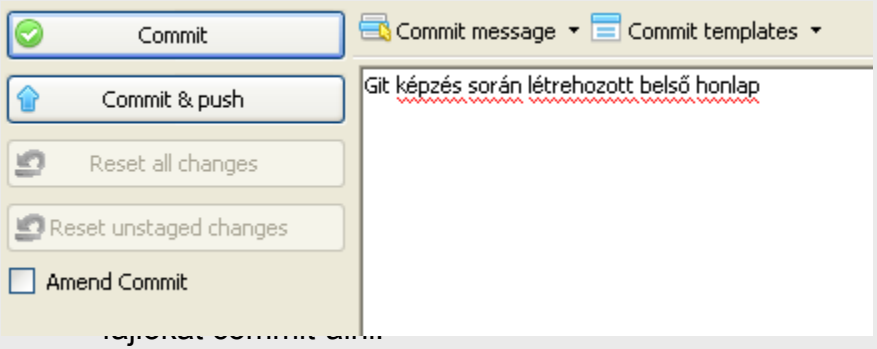
Working Directory Changes és Staging Area-ban lévő fájlok előtti szimbólumok jelentése:

- +: Az utolsó commit óta egy újonnan létrehozott fájl
- —: Az utolsó commit óta törölt fájl
- : Az utolsó commit óta módosított fájl

A bal oldalon látható <u>stage</u> / <u>unstage</u> <u>műveletekkel</u> tudjuk a fájlokat a stash area-ba, azaz <u>stage</u> <u>állapotba</u> tenni és fordítva

Módosítások, különbségek vizsgálata a diffarea-n:

- Ezekkel a nyilakkal tudunk a módosított sorokon végigugrálni,
- módosított sor környező sorainak láthatóságának beállításai
- ugyanaz a funkciója, mint a wordben
- white space-különbségek megjelenítésének beállításai
- UTF8 Különböző karakterkódolással tudjuk megnézni a fájl tartalmát



Commit-áláskor a következőket kell megadni:

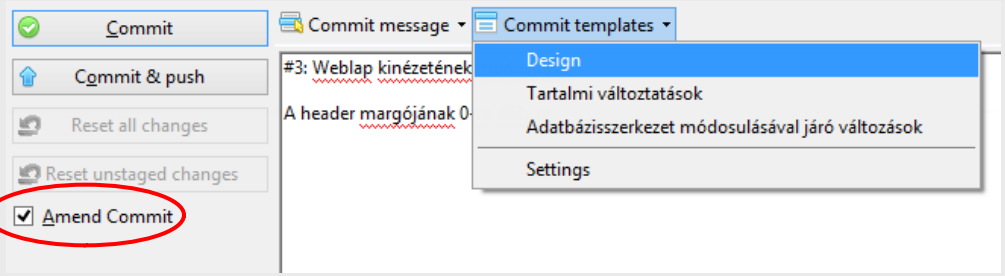
Commit message (mi is megadhatjuk, vagy template-ekkel)

<u>Reset all/unstaged changes:</u> A working directory changes állapotban lévő, mind a nem bejegyzett, mind a staging area-ba bejegyzett fájlmódosítások törlése (u.az , mint a fájlra kattintva jobb gomb, majd reset file changes)

(Kijelölöm a fájl(oka)t, majd jobb gomb, majd Reset all/unstaged changes)

Amend Commit *

Az Amend Commit egy olyan commit művelet, amely során az éppen aktuális revision tartalmához (amire utoljára ráálltunk, azaz amire checkout-oltunk (lásd checkout művelete)) hozzátesszük a még eddig nem commitált bejegyzéseket, vagy egyszerűen csak módosítjuk a commit message szövegét, azaz az aktuális revision tartalmát, vagy commit message-ét frissíteni.



A háttérben ilyenkor valójában a régi revision-ünk helyett egy újat készít, egy új hash-kóddal., ezért egy olyan revision-ra ne alkalmazzuk, amelyet már push-oltunk (lásd később a push műveleténél) a központi repository-ba, mert csak bonyodalmakat okoz.

Viszont, ha nem push-oltuk a revision-t, akkor, ha a stage és commit műveletét gyakorlatilag csak egymás után, egyszerre használom, akkor olyan esetekben, amikor át kell állnom egy másik verzióra, úgy, hogy itt még nem vagyok teljesen kész, akkor az amend commit-tal kiválthatom a staging area fogalmát, mégpedig a következő műveletsorral:

1. commit,
2. átállok, azaz checkout-olok a másik verzióra (lásd checkout résznél)
3. megcsinálom benne a szükséges részt, majd commit
4. Visszaállok, azaz checkout-olok a szóban forgó verzióra (lásd checkout résznél)
5. befejezem az itteni módosításaimat, majd commit helyett amend commit-ot nyomok.

Edit GitIgnore *

Untracked/nem nyomon követett fájlok:

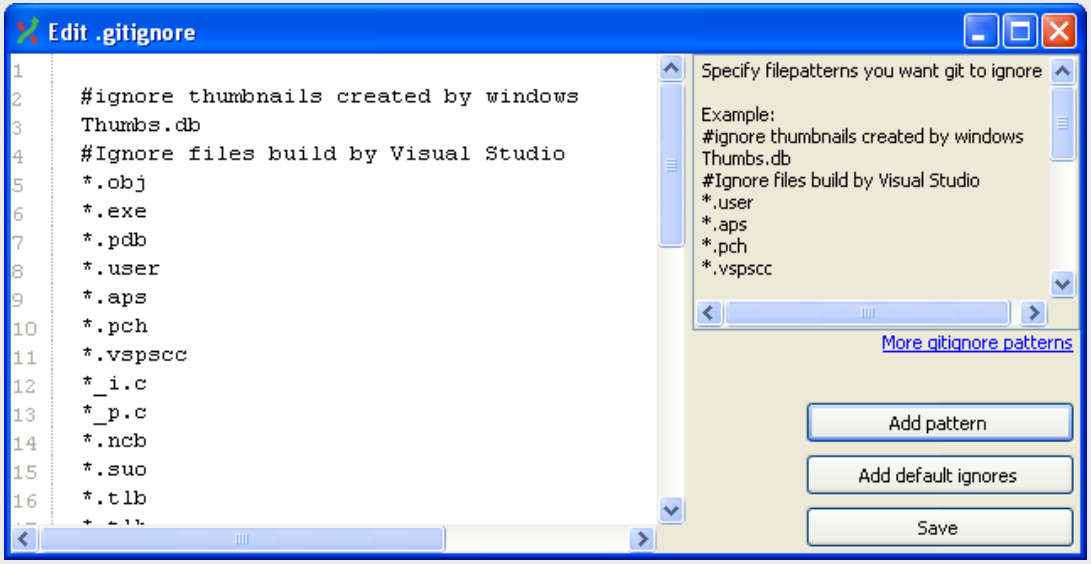
Helyi jelentőségű fájlok, amelyeket változtatásait nem akarom, hogy bekerüljenek a lokális adatbázisba (és így később a központi repo-ba)

(pl.: a Release / Debug-ba készült fordítást)

Repository-> Edit .gitIgnore

Ablak szerkezete:

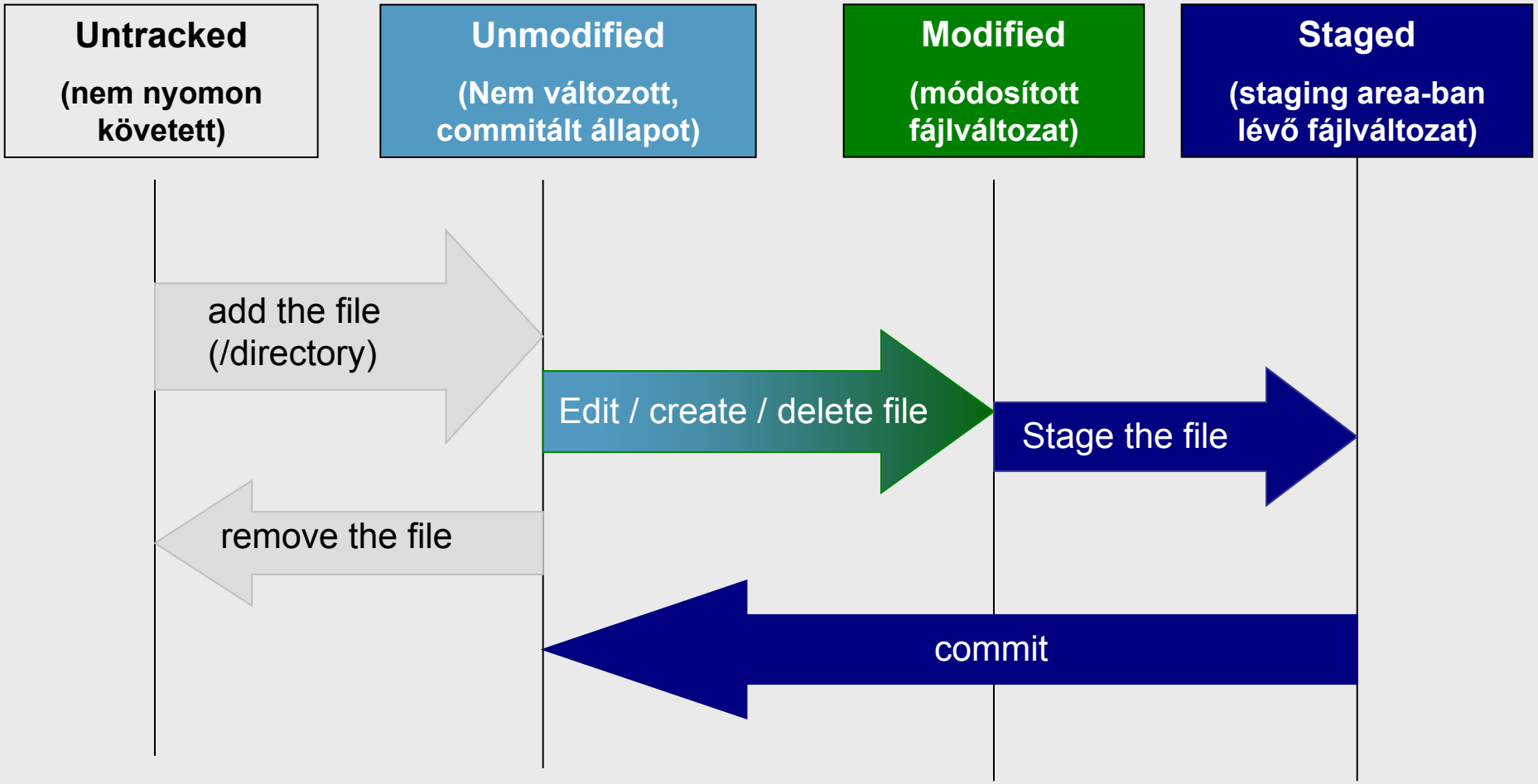
- Jobb oldalt a kihagyandó fájlok (kezdetben üres, az Add default ignores-ra nyomva automatikusan feltölti, de kézzel is állíthatjuk, sőt egész bonyolult reguláris kifejezéseket ishasználhatunk.
- Save-vel mentünk.



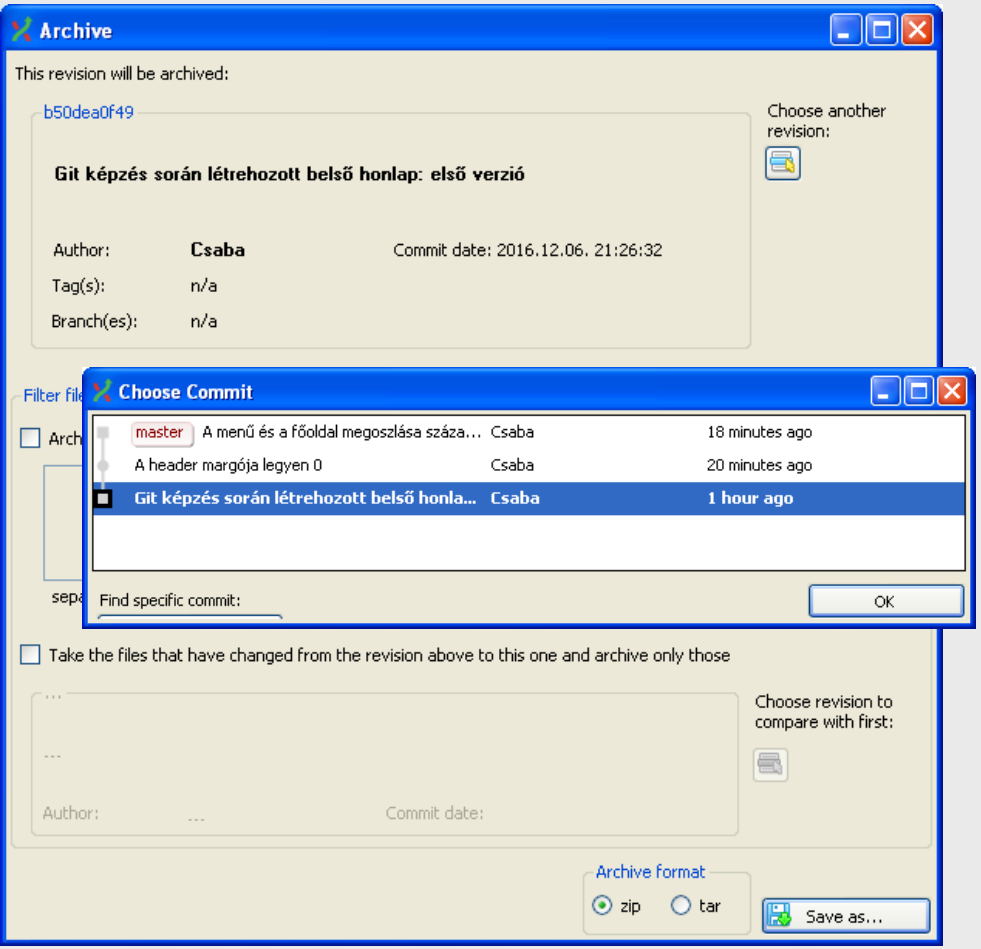
A bal oldali szövegdobozban néhány sor jelentése:

- Egy sorban, ha van # (nem feltétlenül az elején), az utána lévő szöveget nem veszi figyelembe (comment)
- *.suo: a könyvtárban a sou kiterjesztésű fájl (ez a solution fájl .NET-ben)
- *_i.c: _i-re végződő fájl, aminek a kiterjesztése .c
- obj/: a repository könyvtárában, ha van egy obj könyvtár, akkor annak a tartalma
- [Bb]in: a bin vagy Bin könyvtárakban lévő fájlok módosításai

Working directory-ban lévő fájlok állapotuk szerint



Archive revision *

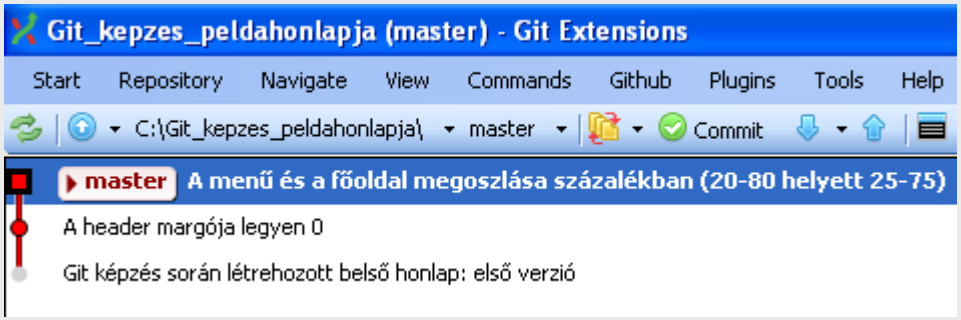


- **Egy repository egy tetszőleges revision-tartalmának tartalmának lementése**
- Előfeltétel: nincs
- Ráállva a gráfpontra, jobb gomb, majd archive revision
- Ablak szerkezete:
  - This revision will be archived résznél a commit hash-e és tulajdonságai
  - Archive format: zip-pel, vagy tar tömörítéssel

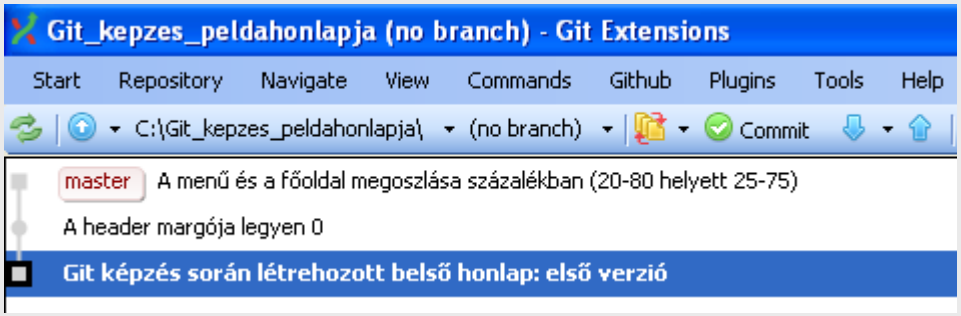
Parancssorosan:

```
git archive --format=zip 0f4b3de506
–output „D:\archiveVersion.zip”
```

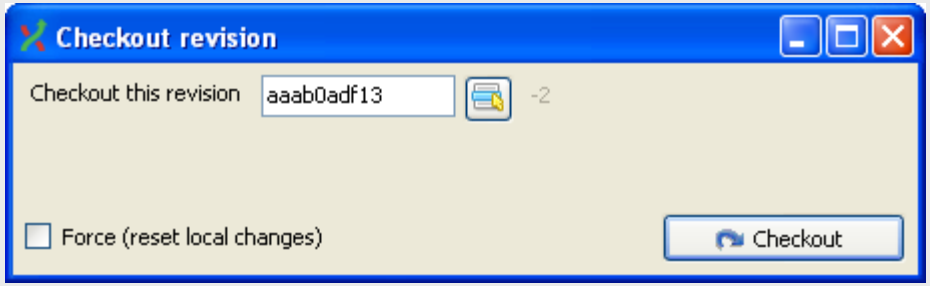
Checkout revision



Checkout revision előtt



Checkout revision után

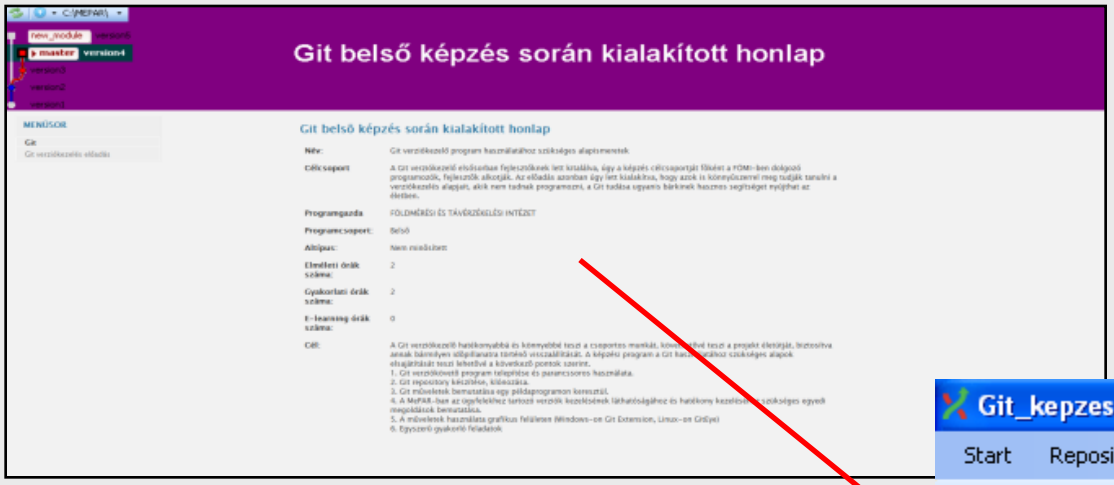


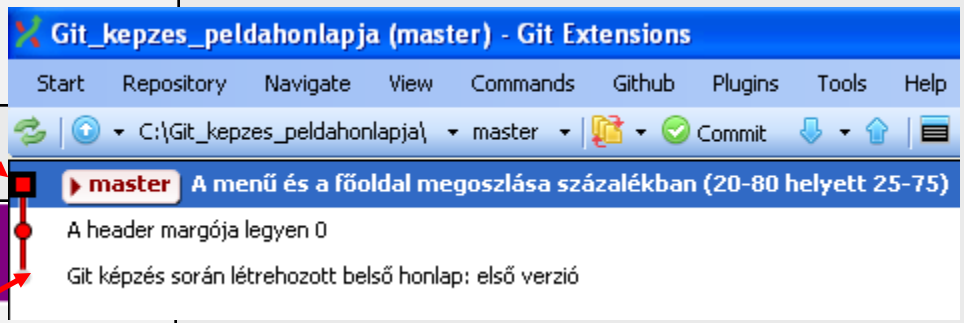
- **Checkout revision:**
- **átállás egy másik revision-ra (lásd később a példasort és a HEAD fogalmát)**
- A megadott revision-re állva jobb gomb, majd checkout revision
- Az ablakban még meg tudjuk változtatni azt a revision-t amelyikre állni szeretnénk.

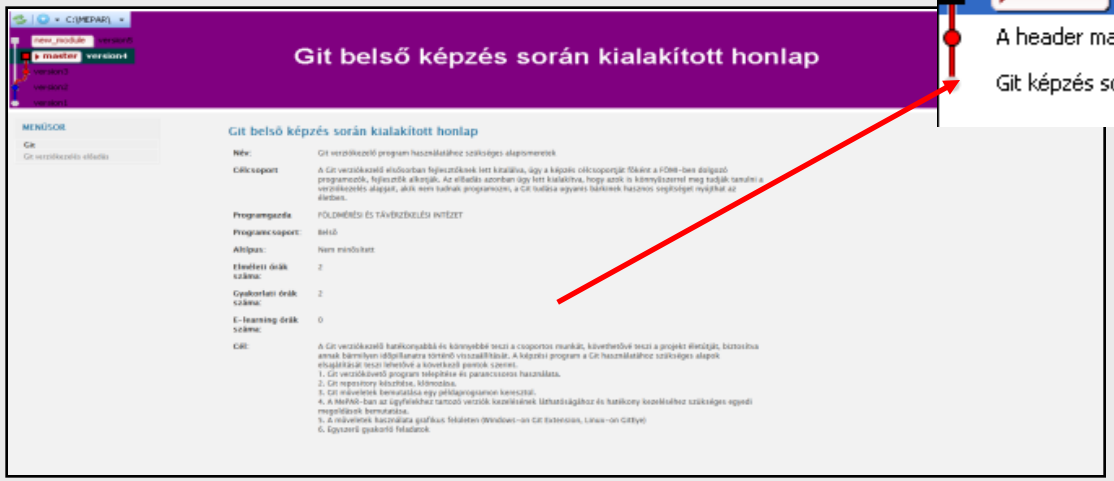
Parancssorosan:

```
git checkout aaab0adf13
```

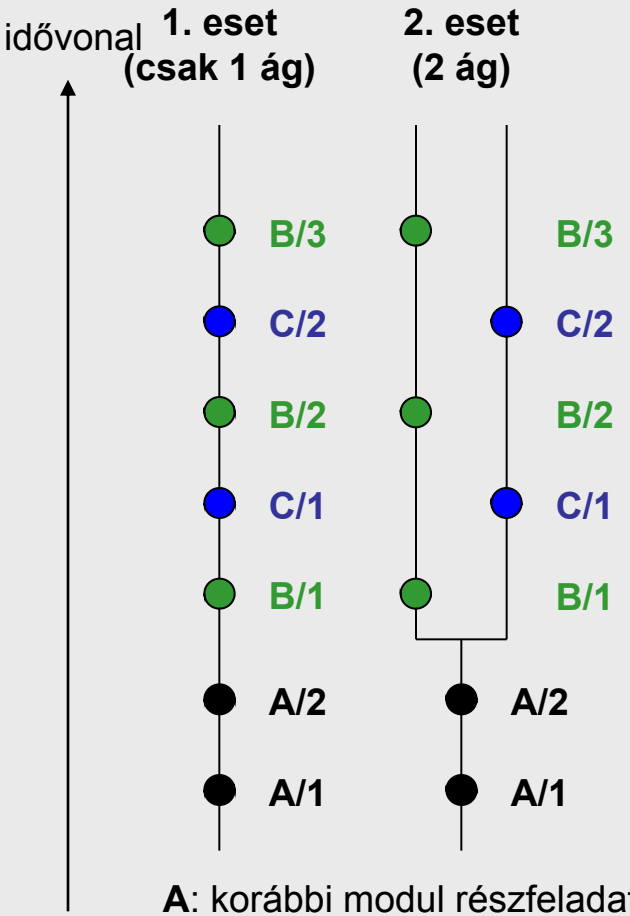
Checkout revision







A branch-ek szerepe a Git-ben



A: korábbi modul részfeladatai

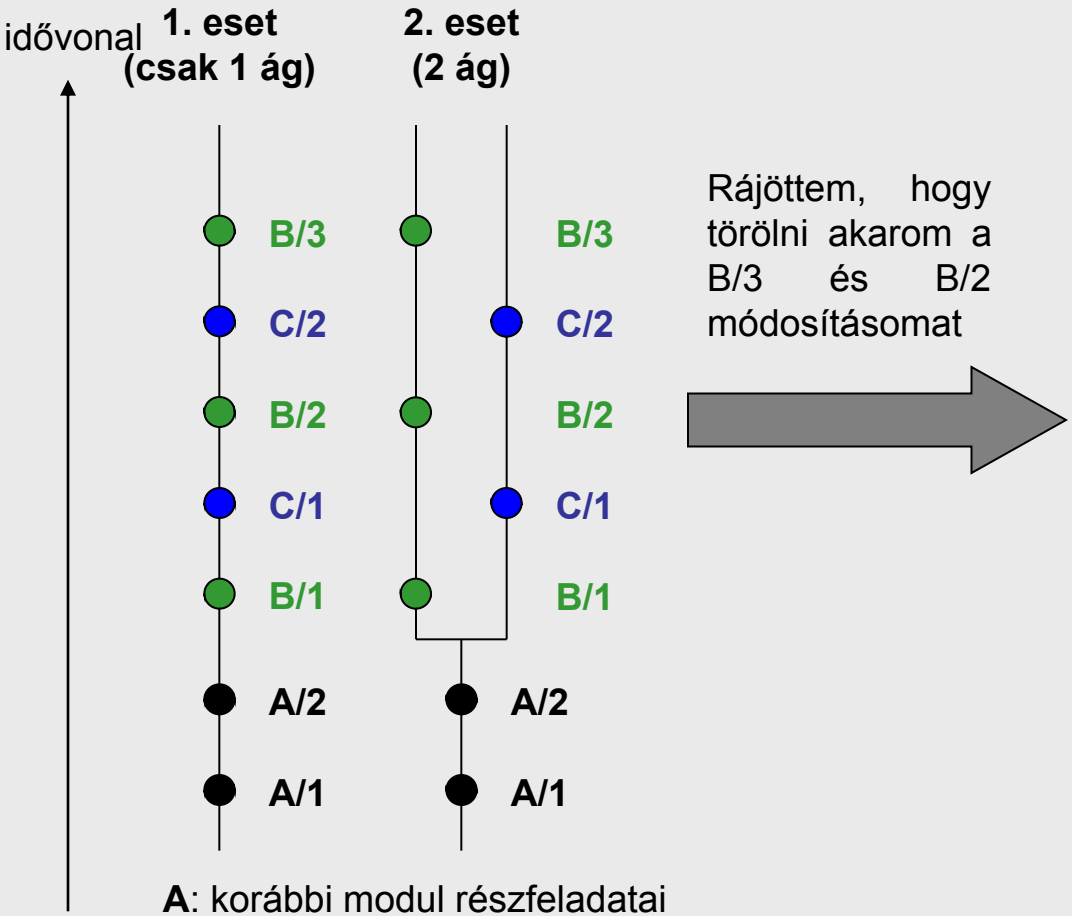
B,C: két újabb párhuzamosan készített modul részfeladatai

A <u>branch</u> a fejlesztés egy ága, a repository-ban lévő hash-fa egy részfája. Ha minden fejlesztést elszeparálva, külön ágba tudok tenni, az nagyban megkönnyíti a munkát.

A default ág a <u>master</u>, ami minden projekt létrehozásakor automatikusan létrejön!

Tipikusan jó példa a branch-ek szükségességére, amikor az ArcMap 9.3-ról folyamatosan 1-2 hónap alatt kellett átállnunk az ArcMap 10.0-ra. Mivel a 10.0-hoz egy teljesen új osztálygyűjtemény tartozott, ezért teljesen át kellett írnunk a kódot, viszont a sok nem várt hiba miatt csak néhány ügyintézőhöz mertük feltenni az új, ArcMap 10.0-ra átírt programverziónkat. Ebből fakadóan viszont párhuzamosan mindkét programverzió futott élesben.

A branch-ek szerepe a Git-ben

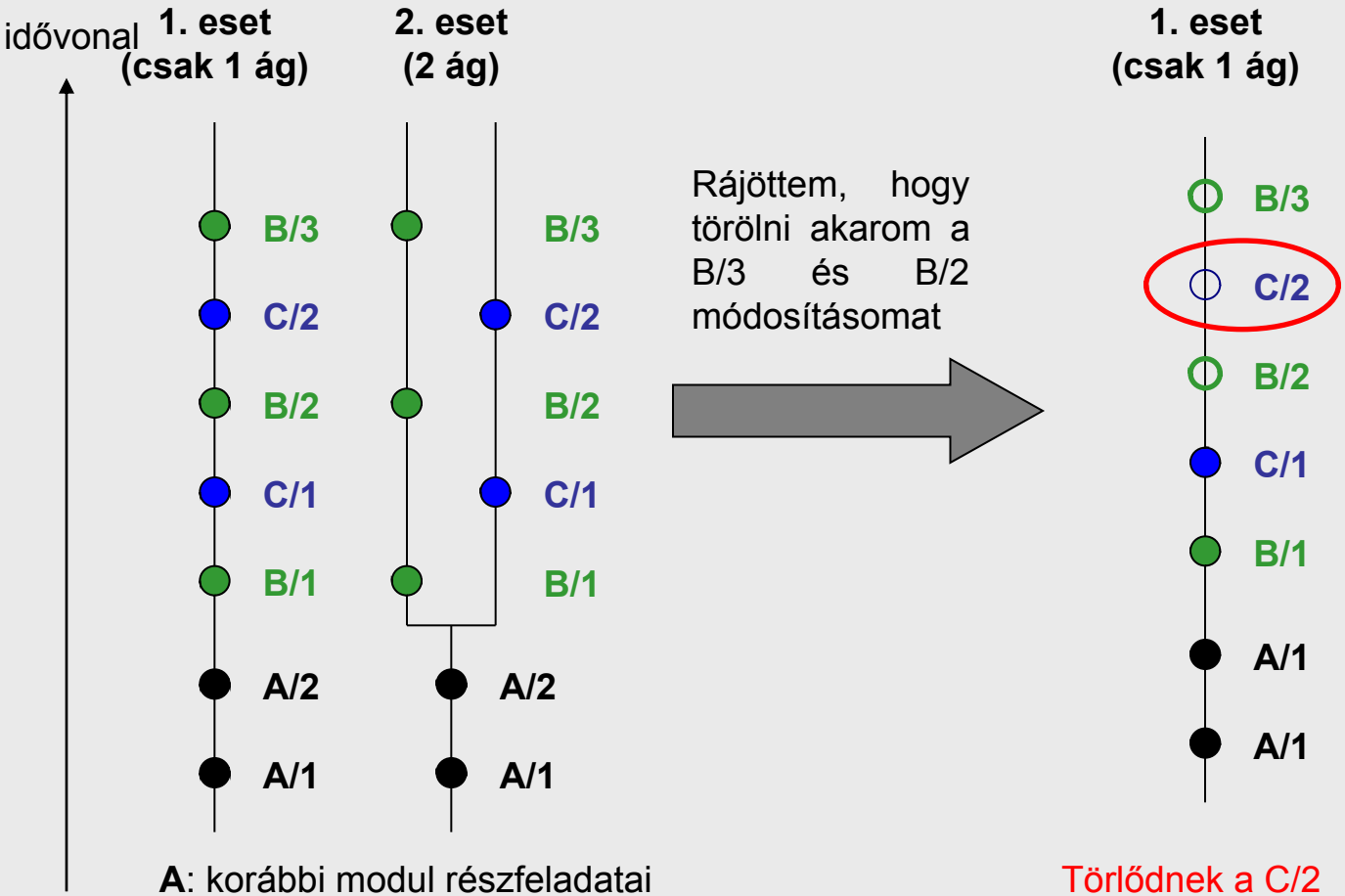


Rájöttem, hogy törölni akarom a B/3 és B/2 módosításomat

A: korábbi modul részfeladatai

B,C: két újabb párhuzamosan készített modul részfeladatai

A branch-ek szerepe a Git-ben



A: korábbi modul részfeladatai

B,C: két újabb párhuzamosan készített modul részfeladatai

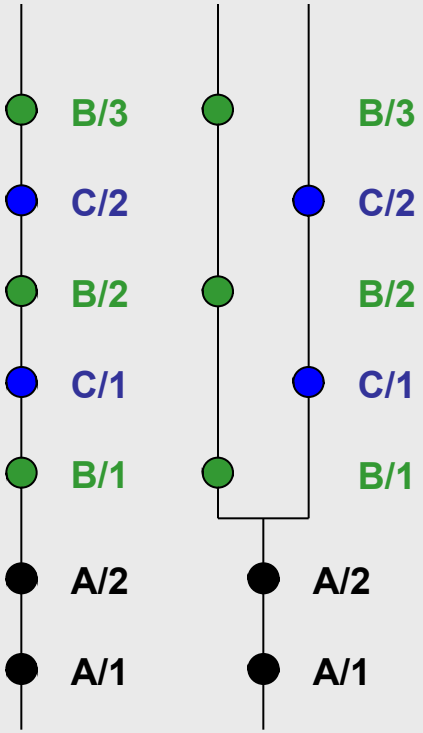
Törlődnek a C/2 módosításai is!!!!

A branch-ek szerepe a Git-ben

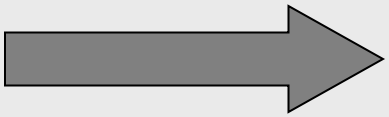
idővonal

1. eset (csak 1 ág)

2. eset (2 ág)

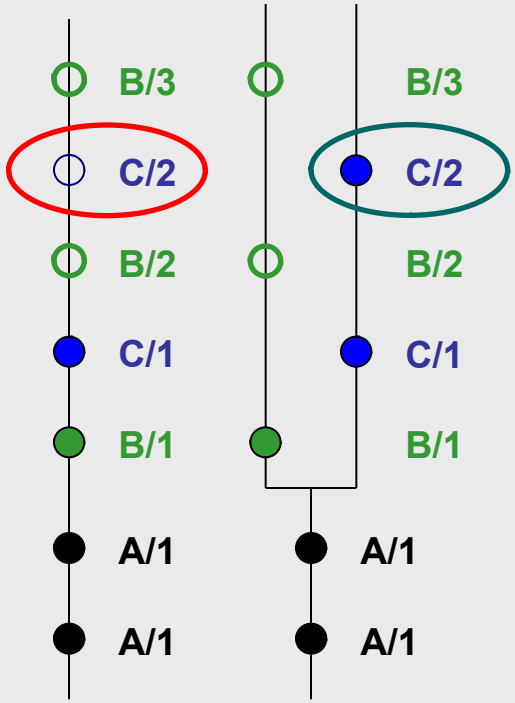


Rájöttem, hogy törölni akarom a B/3 és B/2 módosításomat



1. eset (csak 1 ág)

2. eset (2 ág)



A: korábbi modul részfeladatai

B, C: két újabb párhuzamosan készített modul részfeladatai

Törlődnek a C/2 módosításai is!!!!

A C/2 állapota megmaradnak

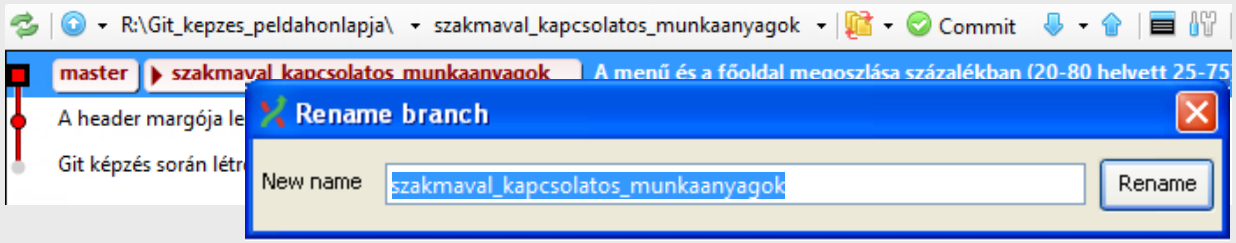
Rename branch

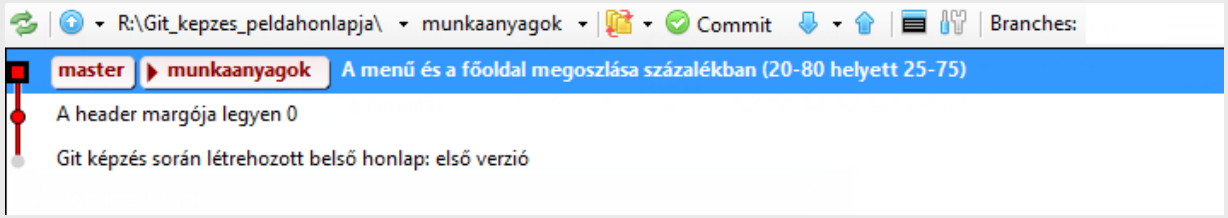
Ezzel változtatjuk meg egy ág nevét.

Kijelölt revision-re nyomva jobb gomb, majd a kiválasztott ágon rename branch.

Parancssorosan:

```
git branch –m szakmaval_kapcsolatos_munkaanyagok munkaanyagok
```

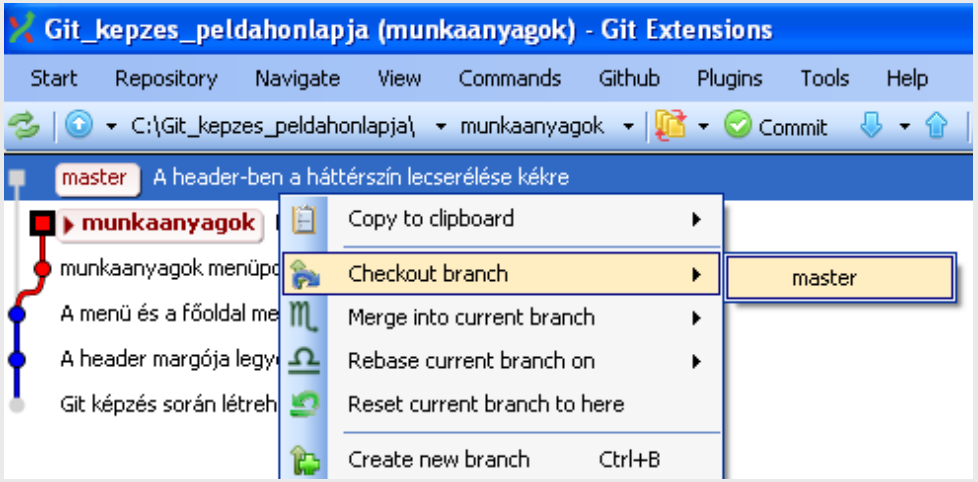


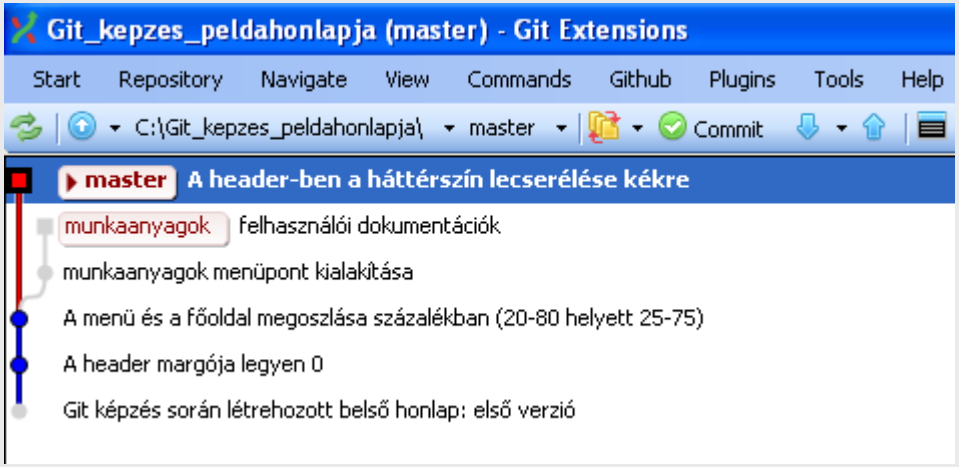


Rename branch előtt és után

Checkout branch

- **Váltás egyik ágról a másikra (lásd később a lokális példasort és a HEAD fogalmát)**
- Megfelelő ág egy revision-jére kattintva jobb gomb majd checkout branch , ezután, ha több ág is van a megadott revision-ön , a megfelelő kiválasztása
- Itt nem jön be külön ablak, egyből ráugrik



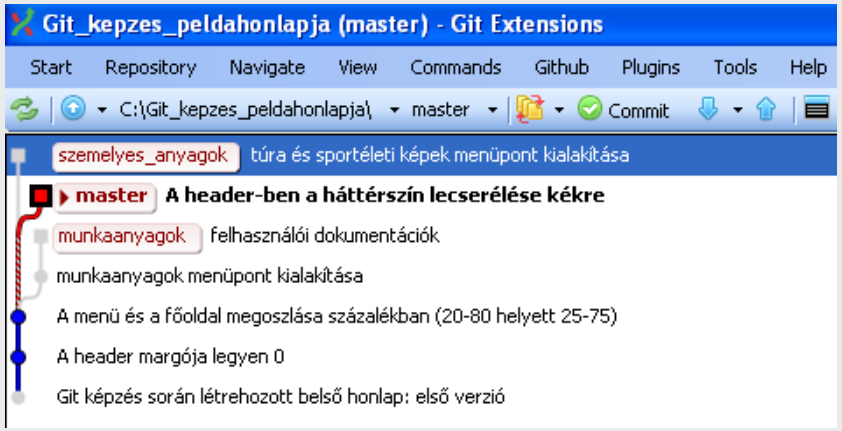


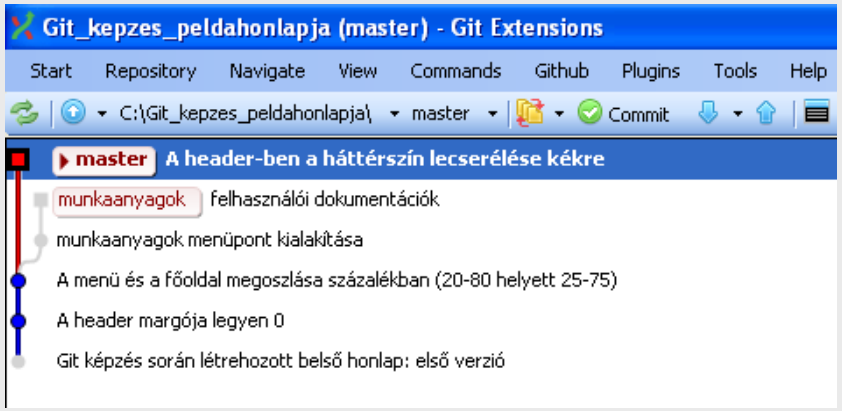
Parancssorosan:

```
git checkout –b master
```

Git checkout előtt és után: munkaanyagok ágról váltottunk a master ágra

Delete branch



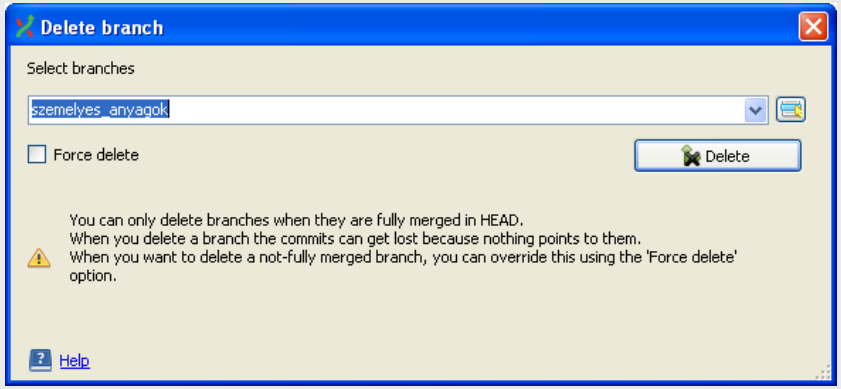


A FOMI_szemelyes_anyagok ág törlése előtt és után

- Előfeltétel: ne a törlendő ág legyen checkoutolva , különben inaktív lesz a funkció
- A megadott ág tetszőleges revisionjére állva jobb gomb majd delete branch vagy Commands->Delete branch
- Ha Commands->Delete branch-et választjuk, akkor az ablakban kiválasztjuk a törlendő ágat (ha többet is akarunk, a Select multiple branch ablakkal tudjuk), majd nyomjuk be a force delete funkciót, mert különben nem minden esetben töröl (lásd később a példáknál)

Parancssorosan:

```
Git branch –D szemelyes_anyagok
```



Jelmagyarázat a lokális műveletek diapéldasorához

V1: 98ca9

master

HEAD

→

V2: 34ac2

V2: 34ac2

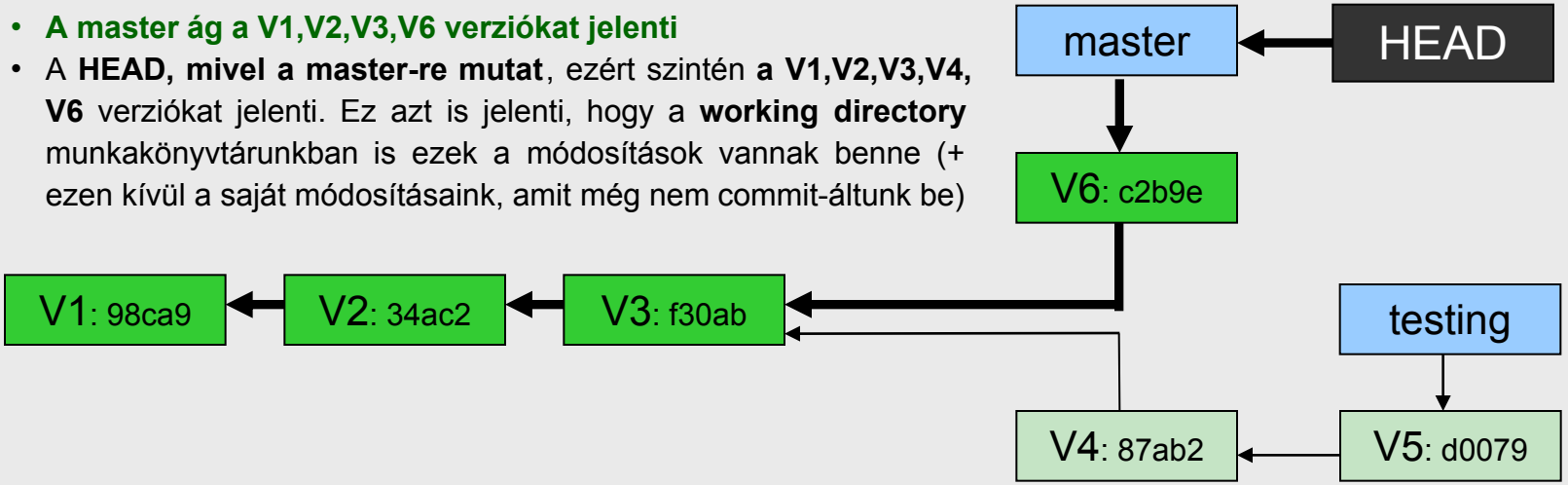
↑

V3: f30ab

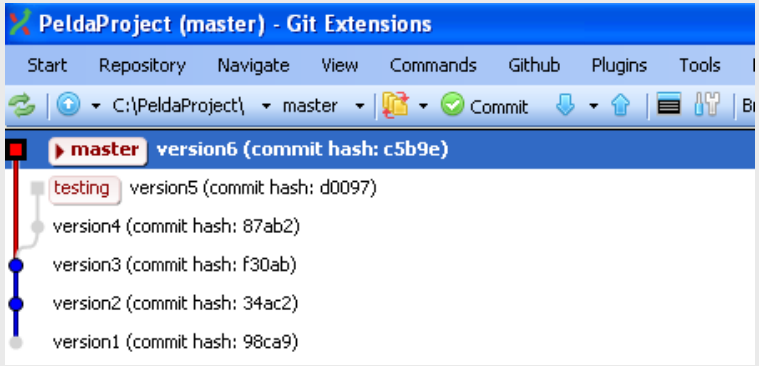
- Zöld téglalap-ban a revision-ök commit message-ei rövidítve (pl.: V1 a version1-et akarja jelenteni), mögöttük pedig a commit hash kódjának első néhány karaktere (amelyek az ábrákon a commit message-ben is benne vannak)
- Kék téglalapban a branch-ekhez tartozó label-ek. **A Branch a fejlesztés egy ága. A Git verziókezelőben egy mutató egy revision-ra (lásd részletesebben a következő diát)**
- **Fekete téglalap-ban a HEAD tartalma, ami a legutolsó állapot, vagy amihez képest módosult a working directory tartalma. A HEAD egy speciális mutató, ami az éppen aktuális branch-re, vagy aktuális revision-re mutat, amire utoljára checkout-oltunk.**
- A vastagabb nyilak azok felé a revision-ök felé mutatnak, amelyeket a HEAD állapotán keresztül közvetlenül, vagy közvetetten elérhetőek az időben visszafelé haladva. Azok a revision-ök , amelyek ezekkel a vastag nyilakkal elérhetőek, közvetlenül, vagy közvetetten benne vannak a working tree-ben/working dir.-ban időben visszafelé haladva (version1 benne a a working tree-ben, erre épülve a version 2 is, stb.)
- Halványzöld téglalapban lévő revision-ök nincsenek benne a HEAD fájában (nem érhetőek el a HEAD-ből a nyilak segítségével sem közvetlenül, sem közvetetten), és így az ezekben található módosítások tartalmai nincsenek benne a working tree-ben sem.
- **Ha egy V2 revision-re közvetlenül mutat V3 revision-ből egy nyíl (akár vastag, akár vékony), akkor azt mondjuk, hogy V2 a szülő vagy parentvision-ja V3-nak , V3 pedig a gyerek vagy childrevision-ja V2-nek. Egy revision-nek akárhány parentrevision-je (akár 0 is) és akárhány gyerekrevision-je (akár 0 is) lehet.**

A branch fogalma

- **A master ág a V1,V2,V3,V6 verziókat jelenti**
- A **HEAD, mivel a master-re mutat**, ezért szintén **a V1,V2,V3,V4, V6** verziókat jelenti. Ez azt is jelenti, hogy a **working directory** munkakönyvtárunkban is ezek a módosítások vannak benne (+ ezen kívül a saját módosításaink, amit még nem commit-áltunk be)

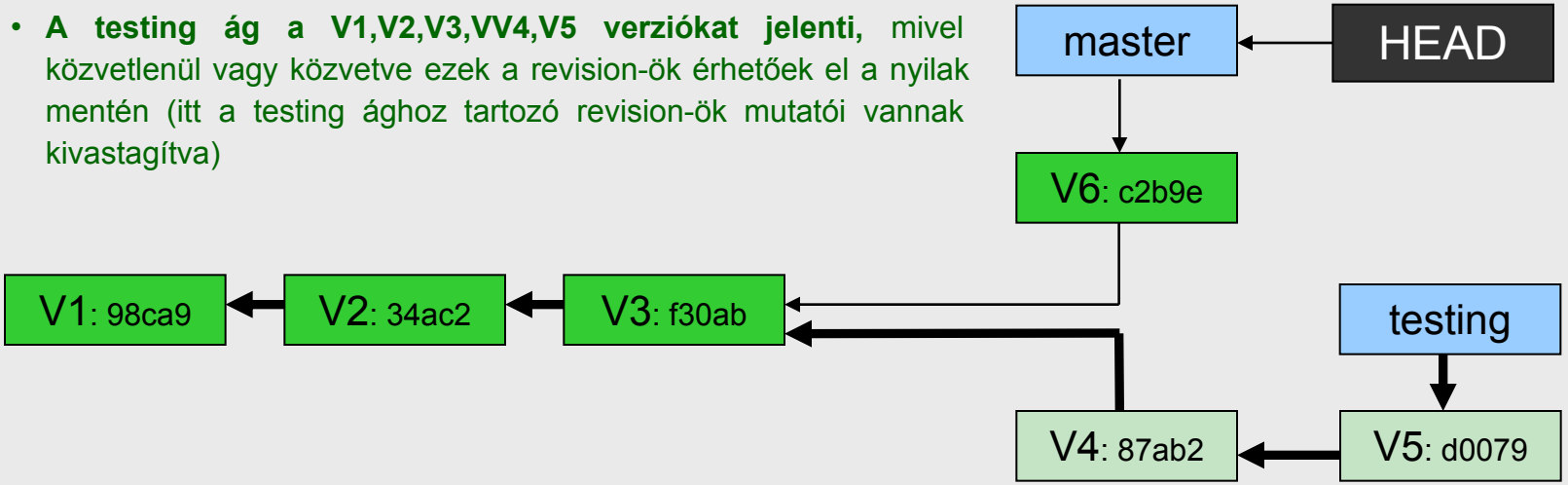


Egy branch , vagy ág tulajdonképpen mindig az ágcímke mutatójával mutatott revision és annak a szülő/nagyszülő/… revision-jeit tartalmazza mindenféle oldalági „rokon” nélkül (vagyis csak a közvetlen ősöket). Ily módon egy ág tulajdonképpen a repository fájának egy részfáját jelenti!
A GitExtension-ben a színes revision-ök azt jelentik, hogy benne vannak a HEAD-címkével mutatott hash-fában (jelenleg a master ág) és a working directory-ban , a kiszürkített revision-ök pedig, hogy nincsenek benne. (a színesen belül a piros/kék, stb. színeknek nincs jelentőségük)

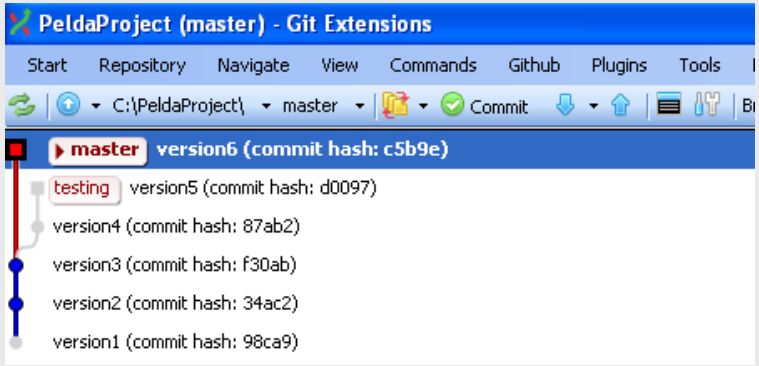


A branch fogalma

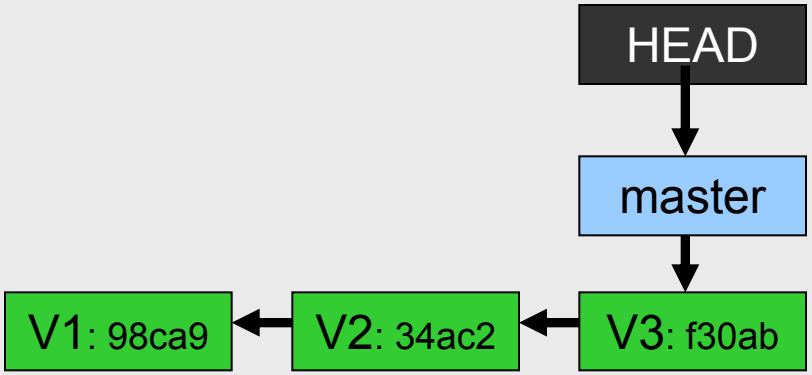
- **A testing ág a V1,V2,V3,VV4,V5 verziókat jelenti,** mivel közvetlenül vagy közvetve ezek a revision-ök érhetőek el a nyilak mentén (itt a testingághoz tartozó revision-ök mutatói vannak kivastagítva)

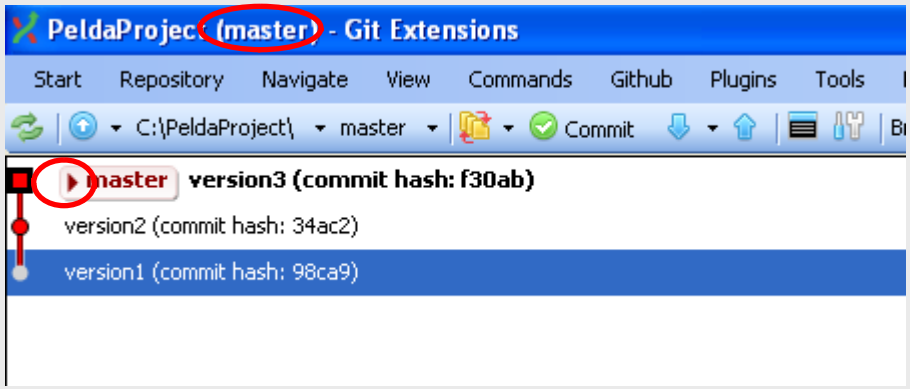


Egy branch , vagy ág tulajdonképpen mindig az ágcímke mutatójával mutatott revision és annak a szülő/nagyszülő/… revision-jeit tartalmazza mindenféle oldalági „rokon” nélkül (vagyis csak a közvetlen ősöket). Ily módon egy ág tulajdonképpen a repository fájának egy részfáját jelenti!
A GitExtension-ben a színes revision-ök azt jelentik, hogy benne vannak a HEAD-címkével mutatott hash-fában (jelenleg a master ág) és a working directory-ban , a kiszürkített revision-ök pedig, hogy nincsenek benne. (a színesen belül a piros/kék, stb. színeknek nincs jelentőségük)



Lokális műveletek – példasor
Alapállapot



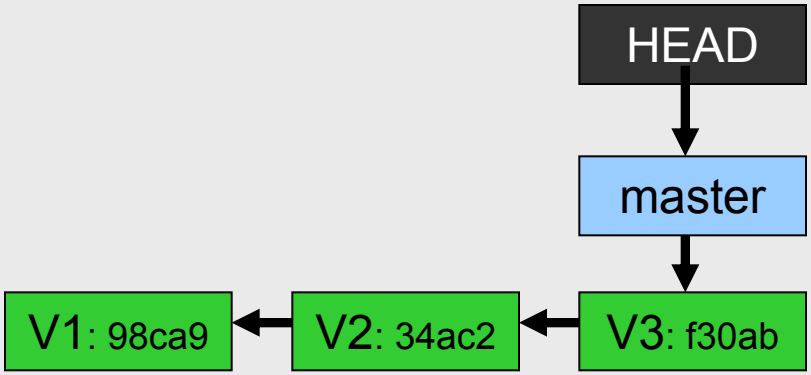


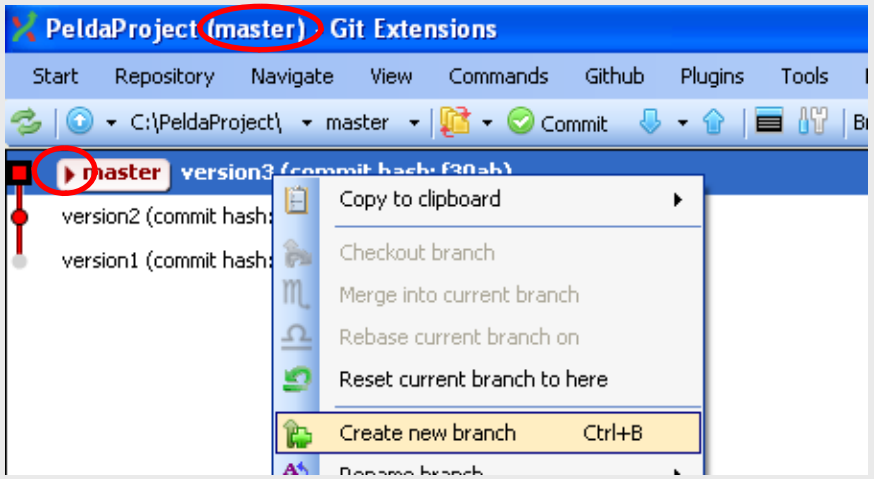
Megjegyzések:

- Checkout előtt, ha lehet, mindig commitáljuk a módosításainkat!!!

- Ha a GitExtension-ben egy piros telített háromszög van az ágcímkén, az azt jelenti, hogy a HEAD erre az ágcímkére mutat (a fejlécben is ki van írva)

Lokális műveletek – példasor
Create new branch (1)



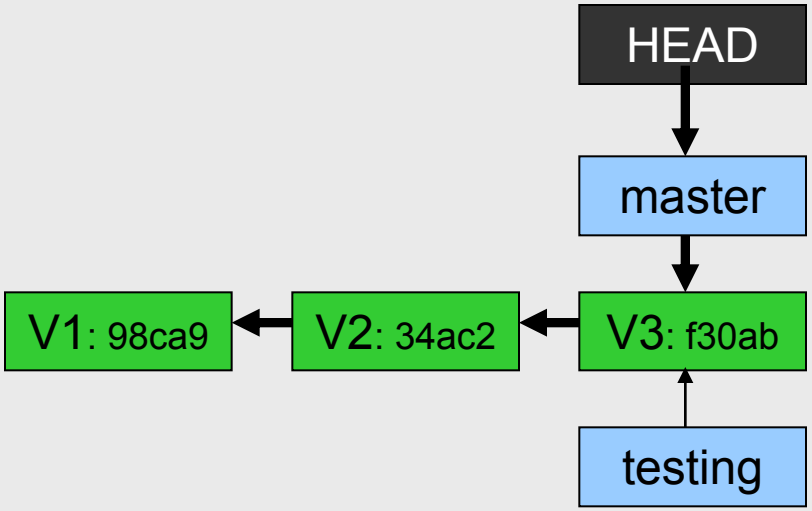


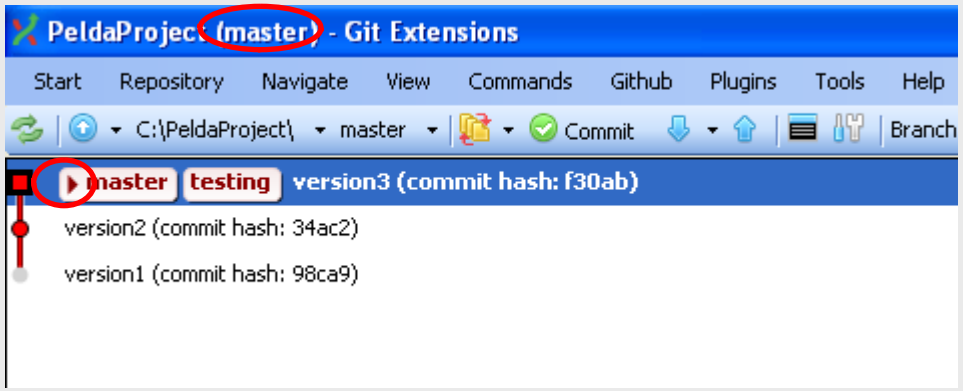
Megjegyzések:

- Checkout előtt, ha lehet, mindig commitáljuk a módosításainkat!!!

- Ha a GitExtension-ben egy piros telített háromszög van az ágcímkén, az azt jelenti, hogy a HEAD erre az ágcímkére mutat (a fejlécben is ki van írva)

Lokális műveletek – példasor
Checkout revision (2)



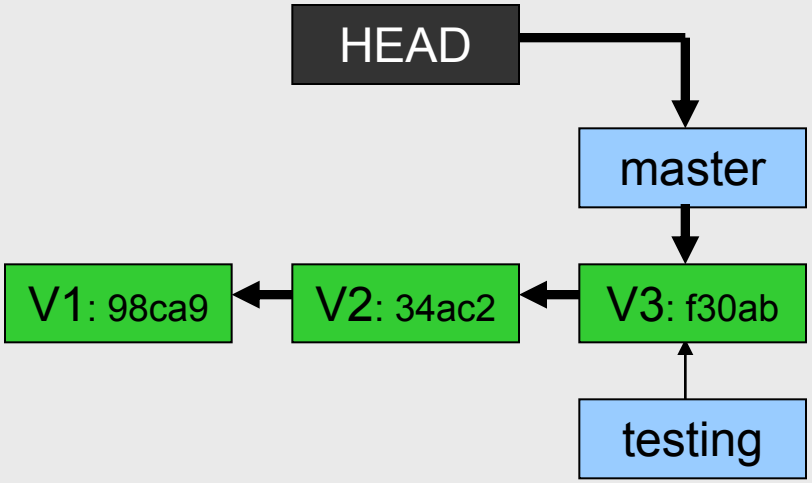


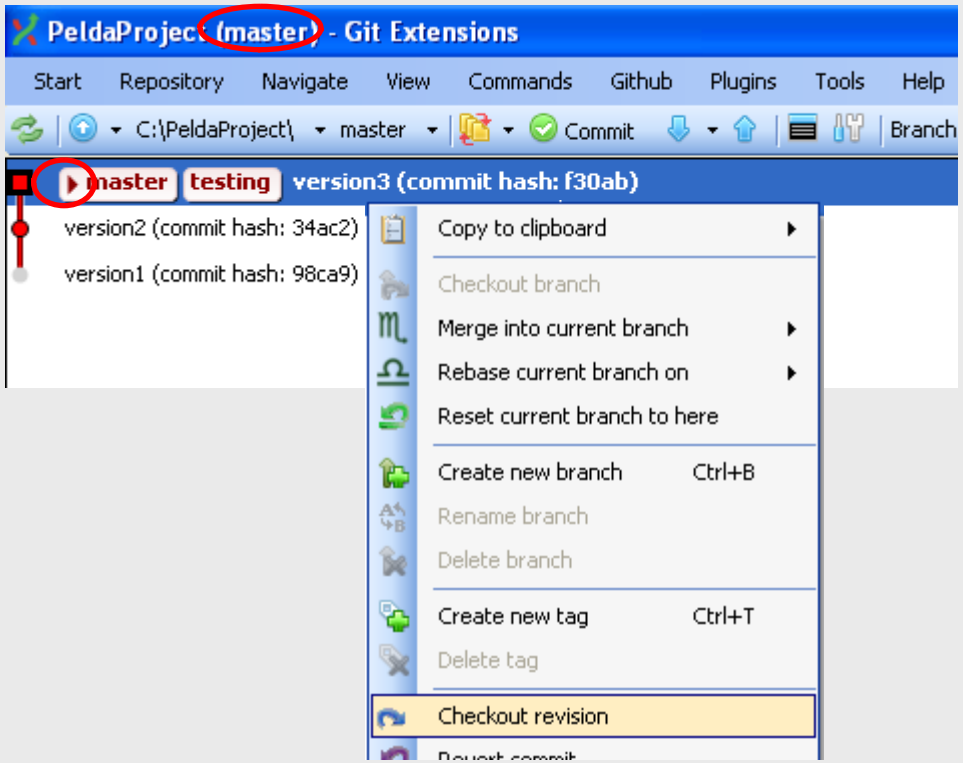
Megjegyzések:

- Checkout előtt, ha lehet, mindig commitáljuk a módosításainkat!!!

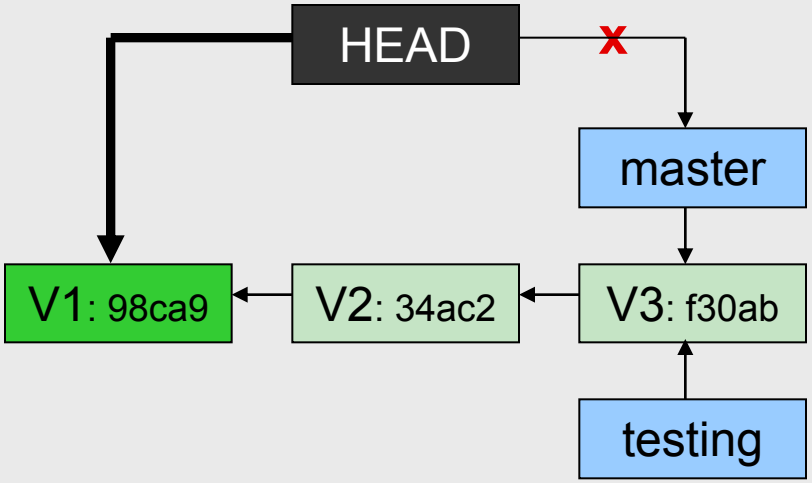
- Ha a GitExtension-ben egy piros telített háromszög van az ágcímkén, az azt jelenti, hogy a HEAD erre az ágcímkére mutat (a fejlécben is ki van írva)

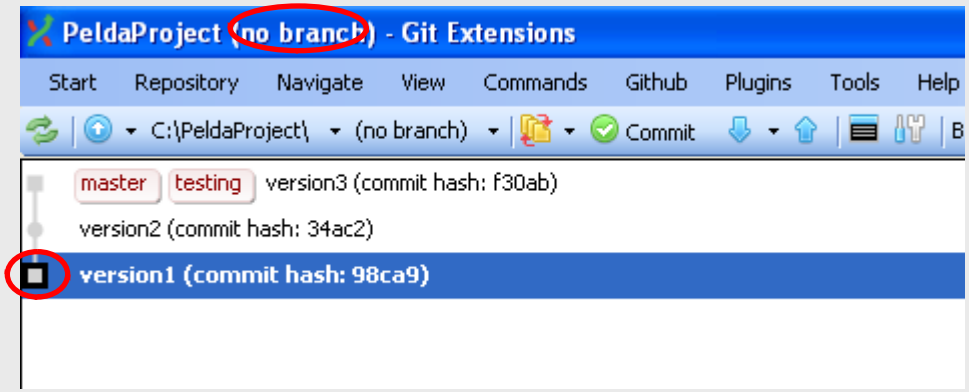
Lokális műveletek – példasor
Checkout revision (2)



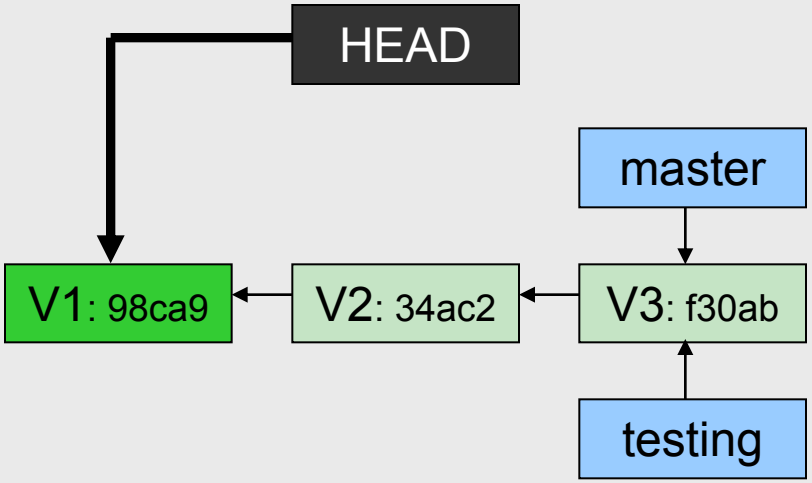


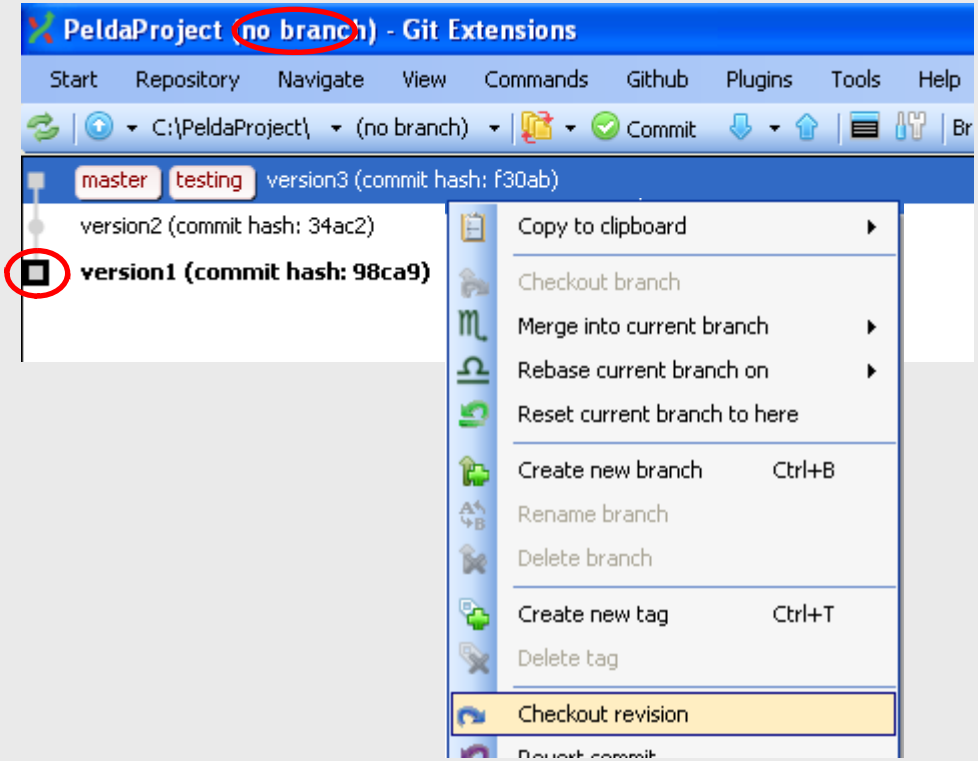
Lokális műveletek – példasor
Checkout revision (2)



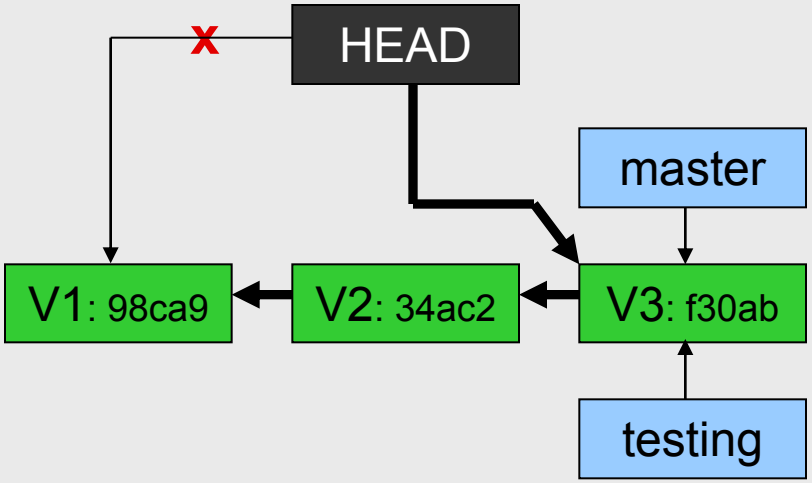


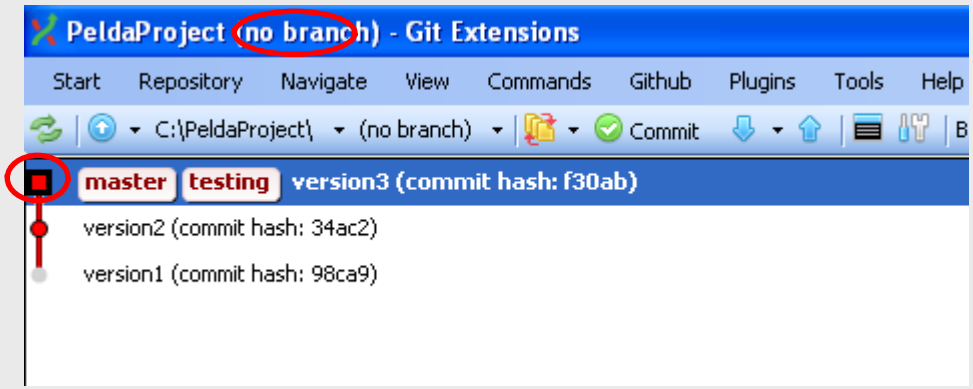
Lokális műveletek – példasor
Checkout revision (3)



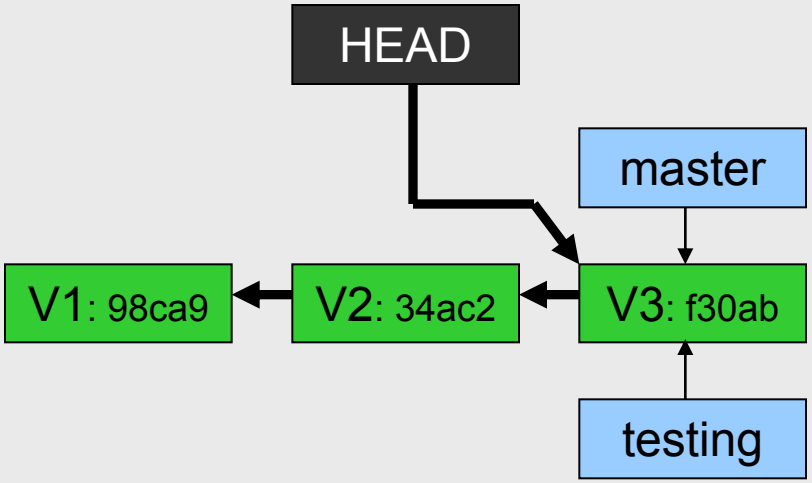


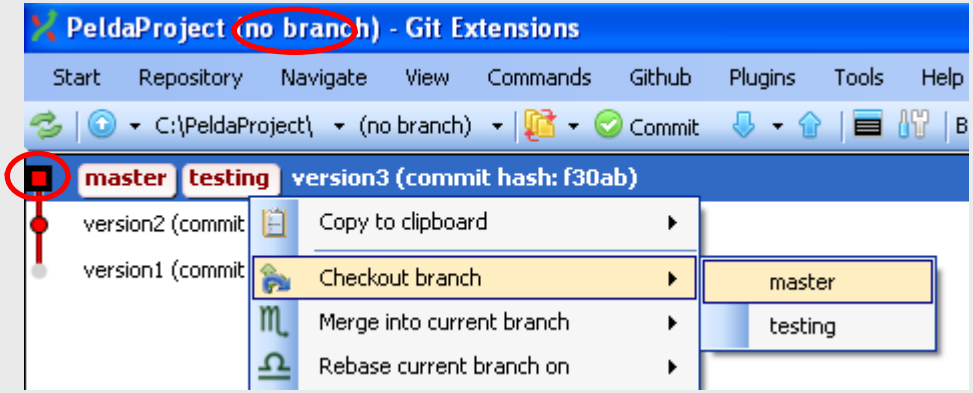
Lokális műveletek – példasor
Checkout revision (3)



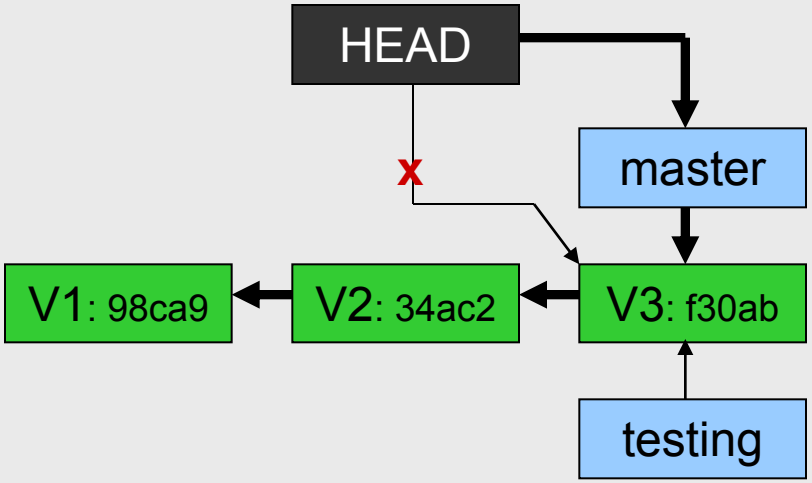


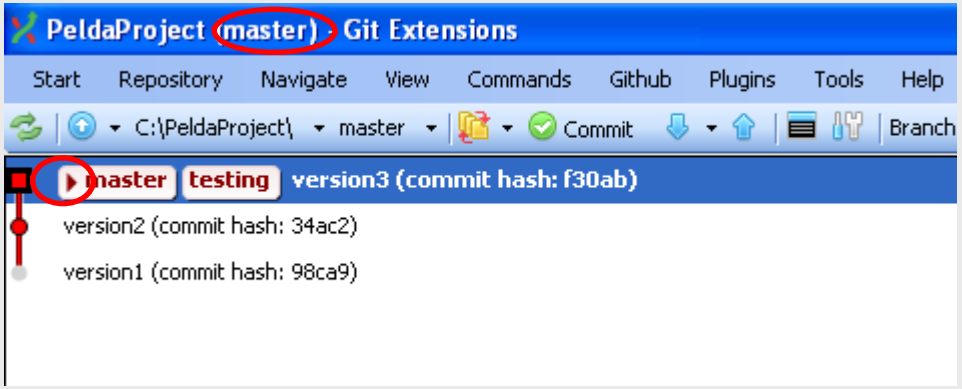
Lokális műveletek – példasor
Checkout revision (3)



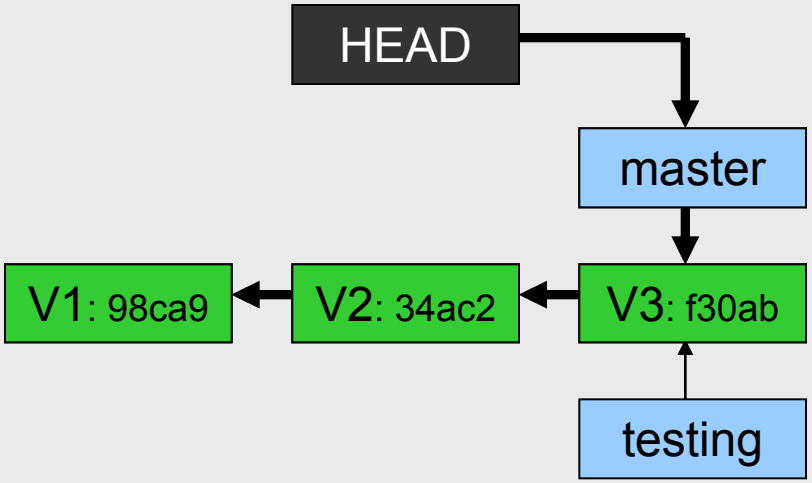


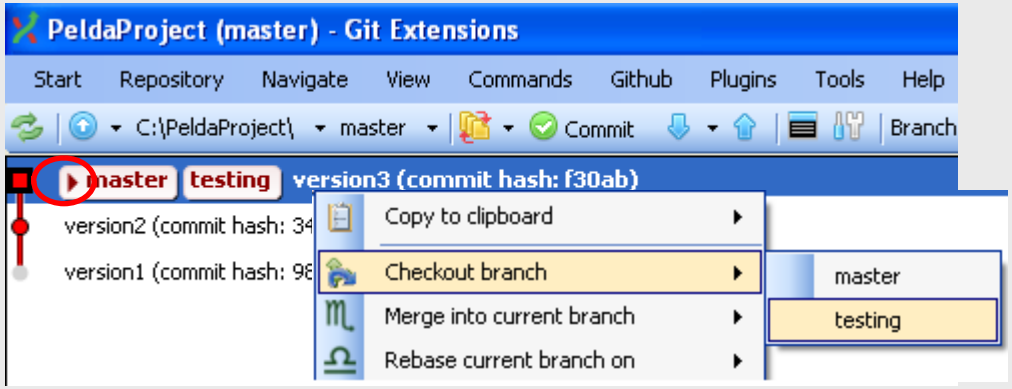
Lokális műveletek – példasor
Checkout branch (4)



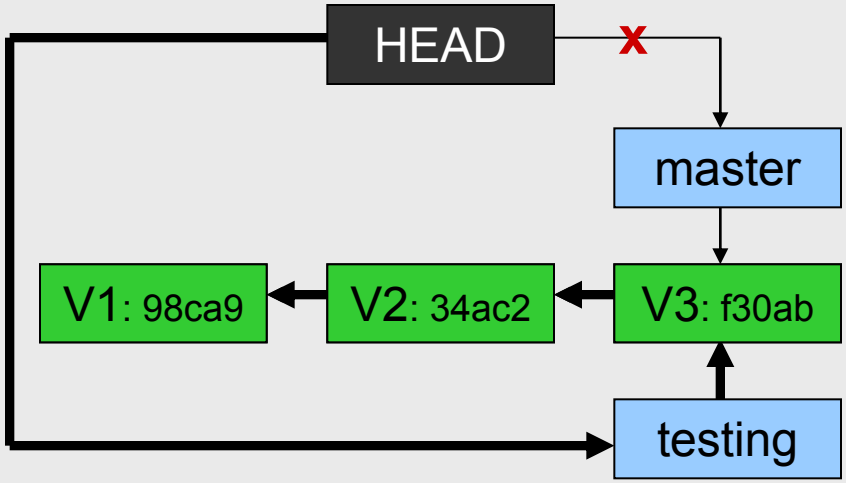


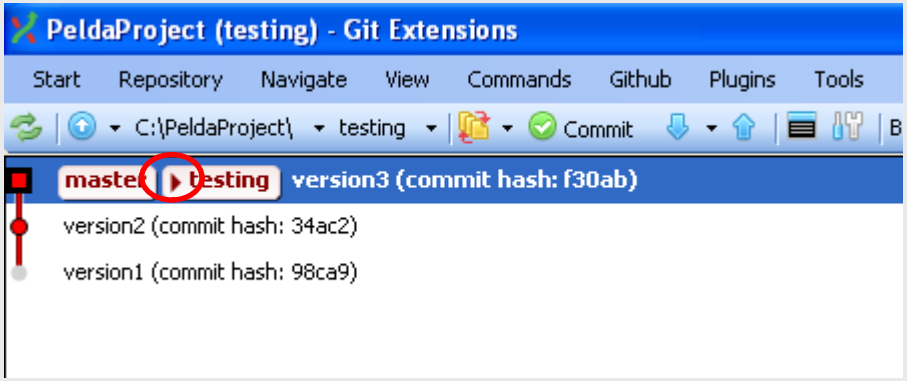
Lokális műveletek – példasor
Checkout branch (5)



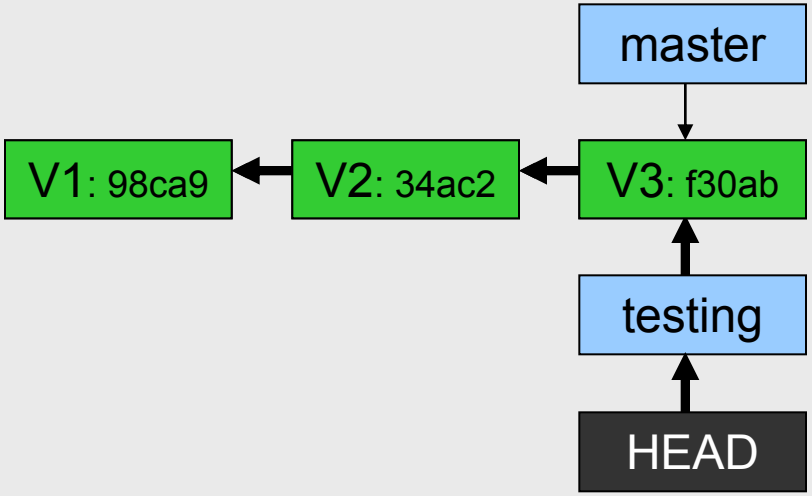


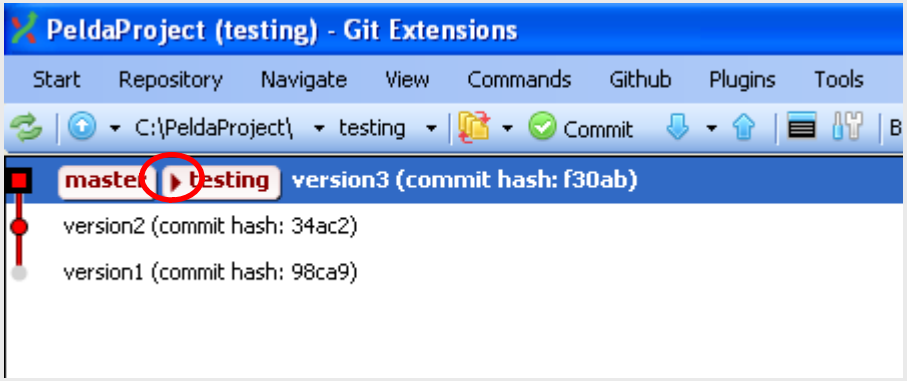
Lokális műveletek – példasor
Checkout branch (5)



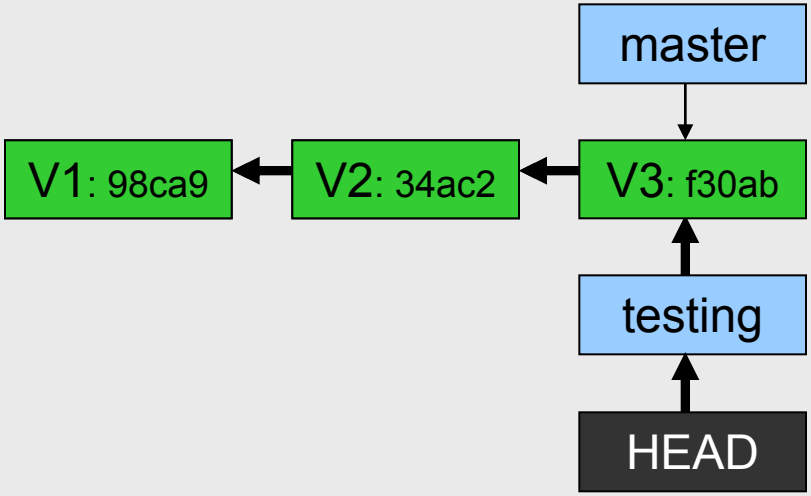


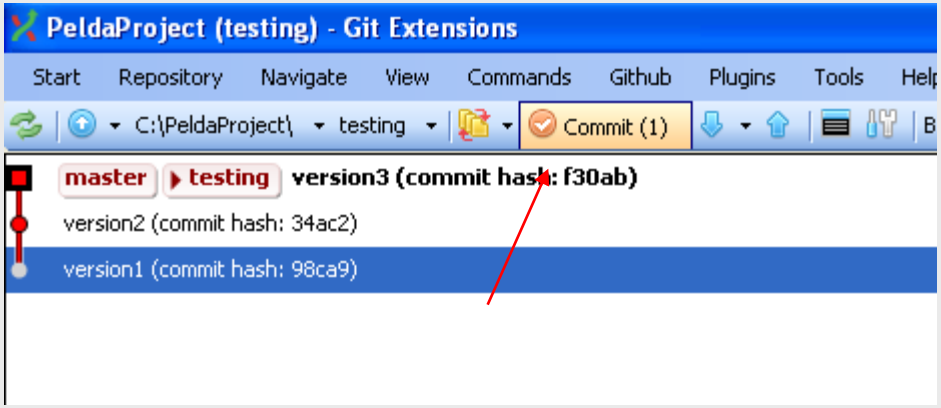
Lokális műveletek – példasor
Checkout branch (5)



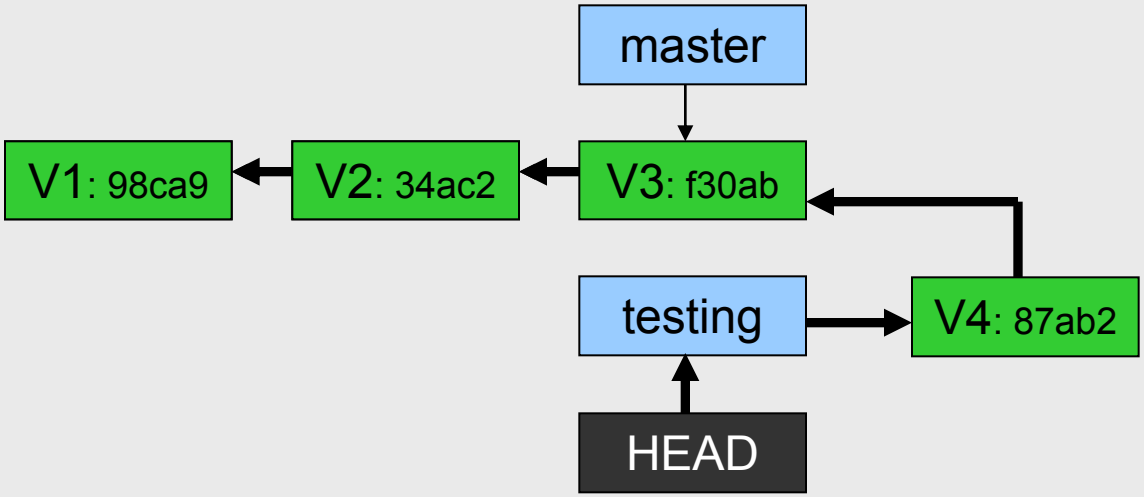


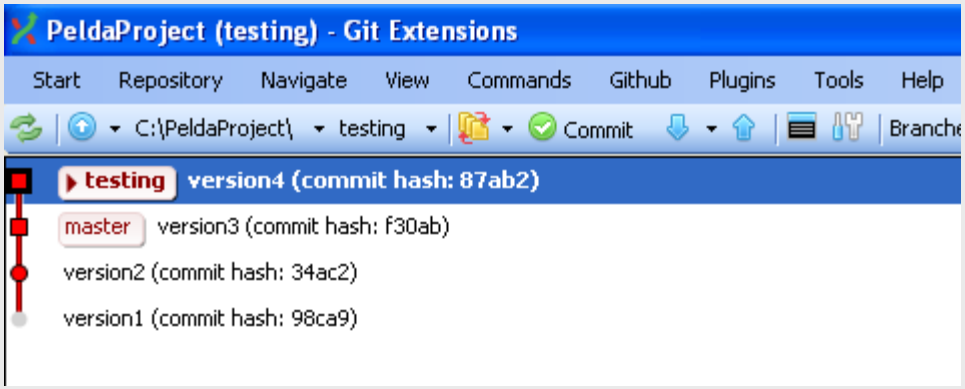
Lokális műveletek – példasor Commit (6)



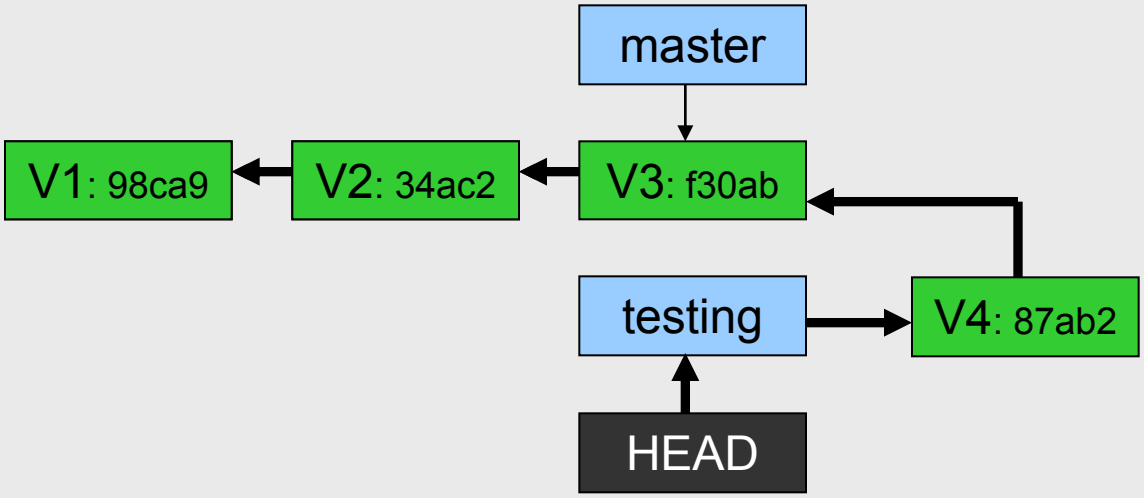


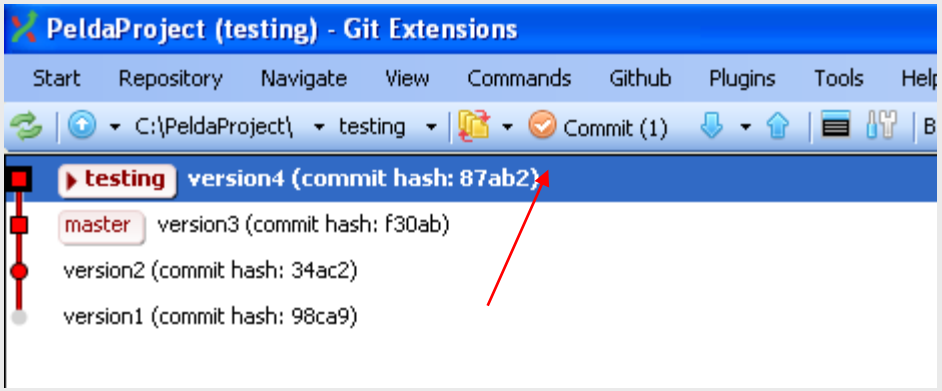
Lokális műveletek – példasor Commit (6)



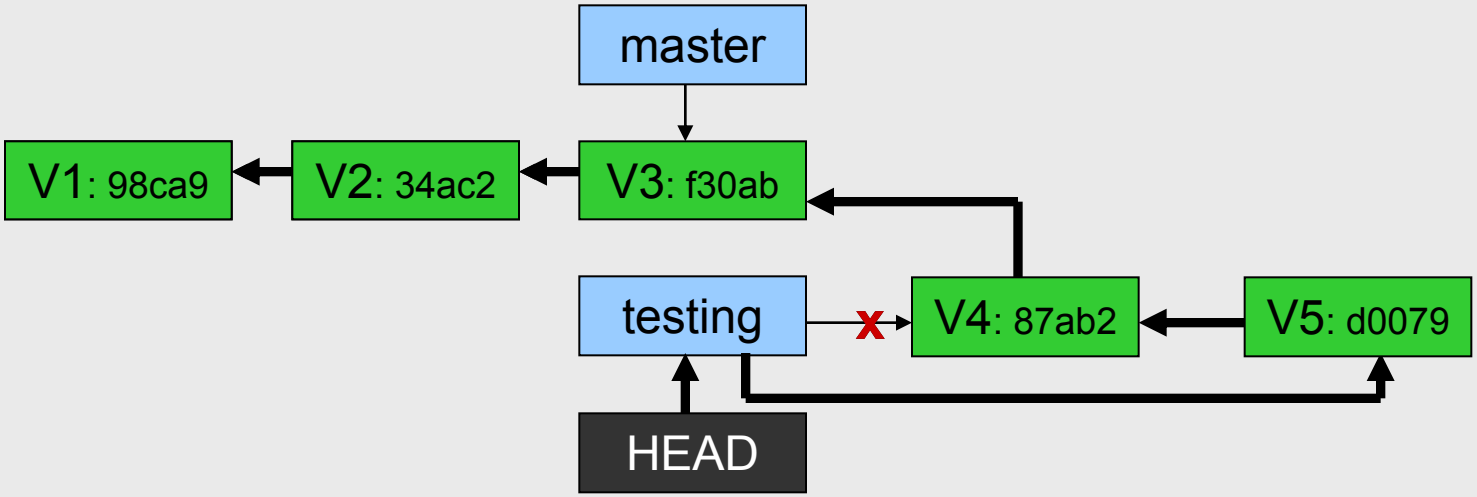


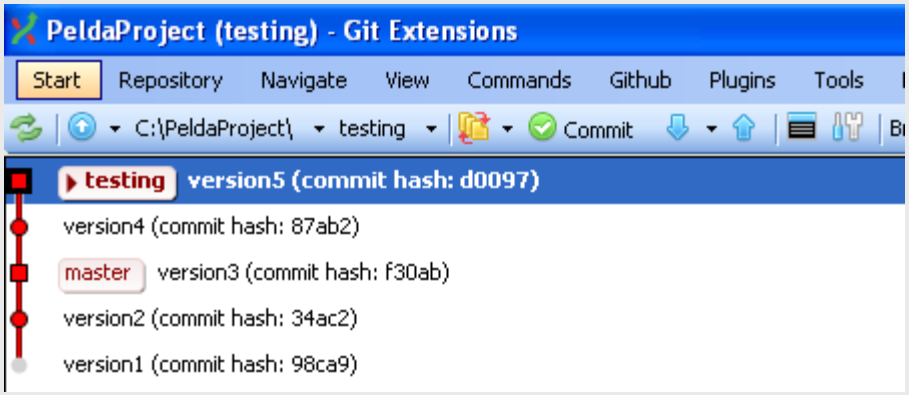
Lokális műveletek – példasor Commit (7)



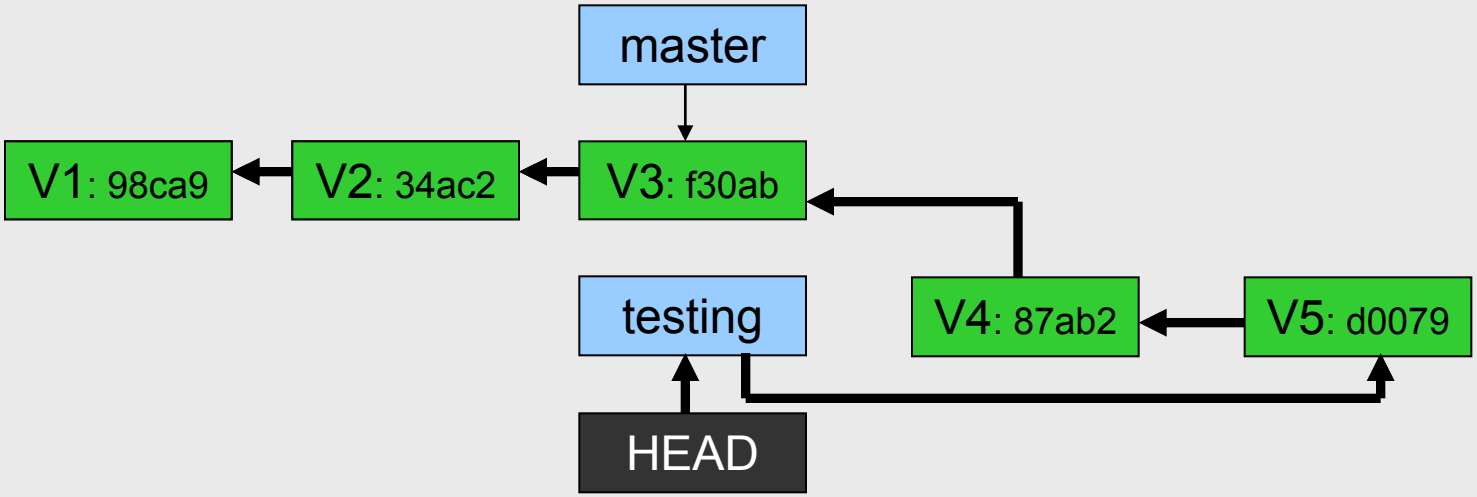


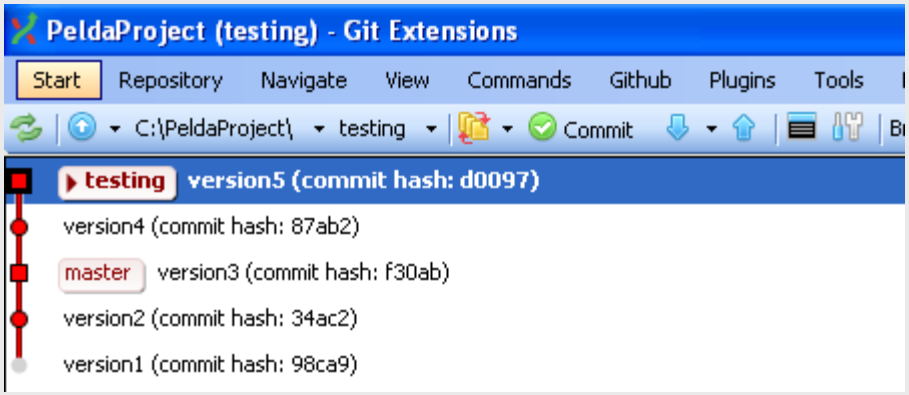
Lokális műveletek – példasor Commit (7)



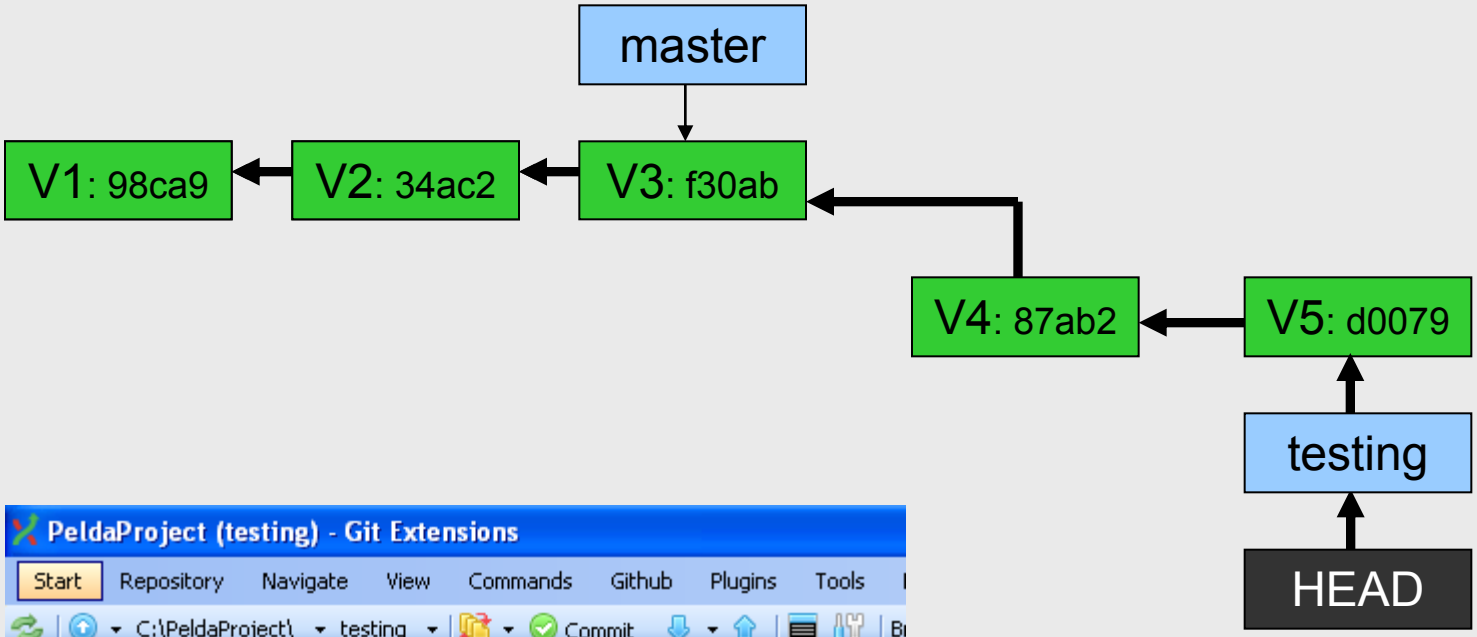


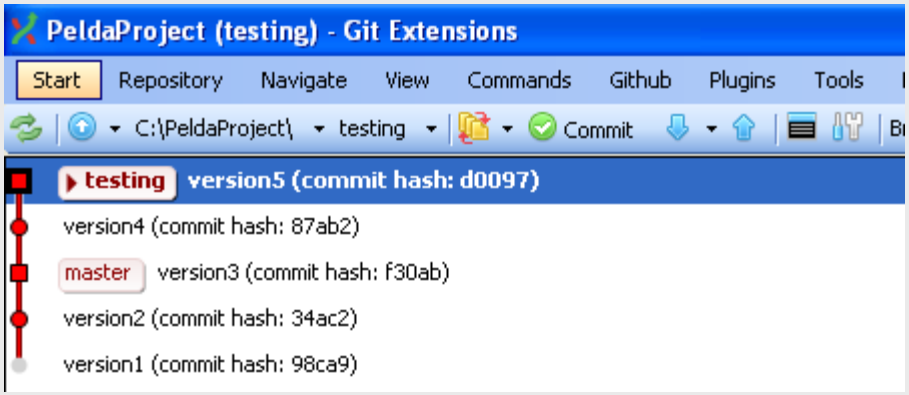
Lokális műveletek – példasor Commit (7)



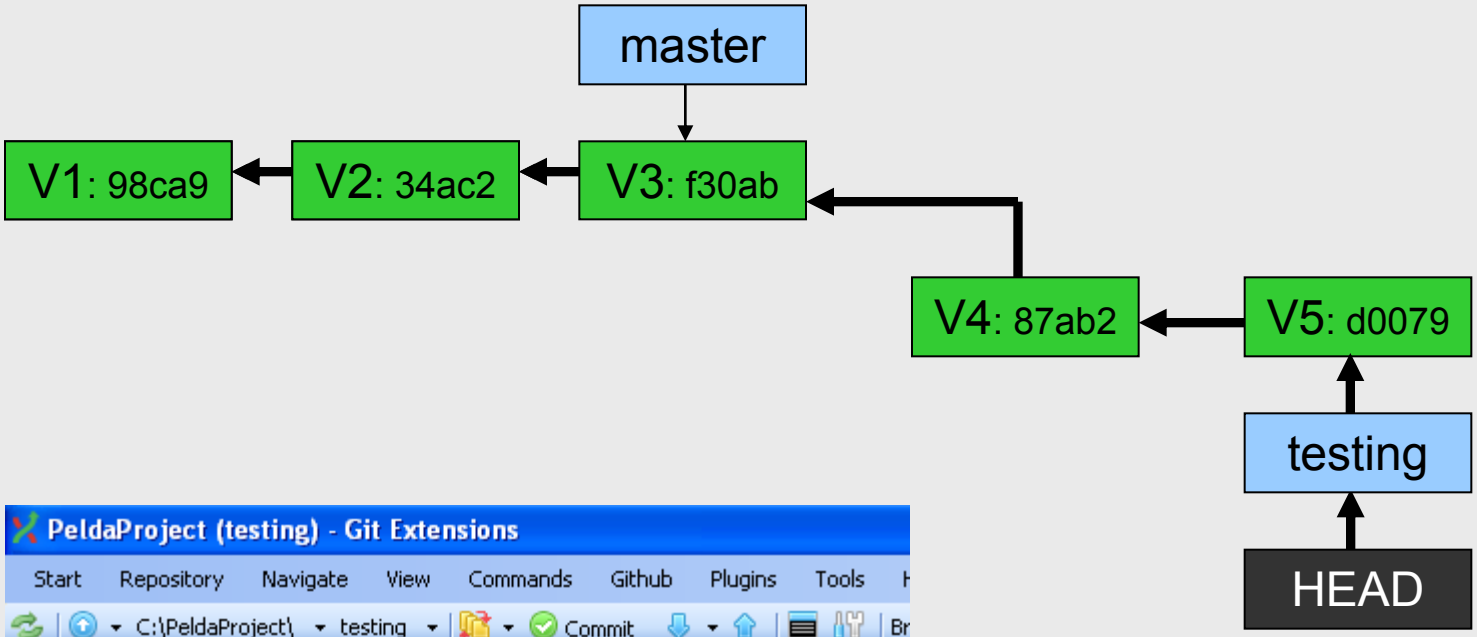


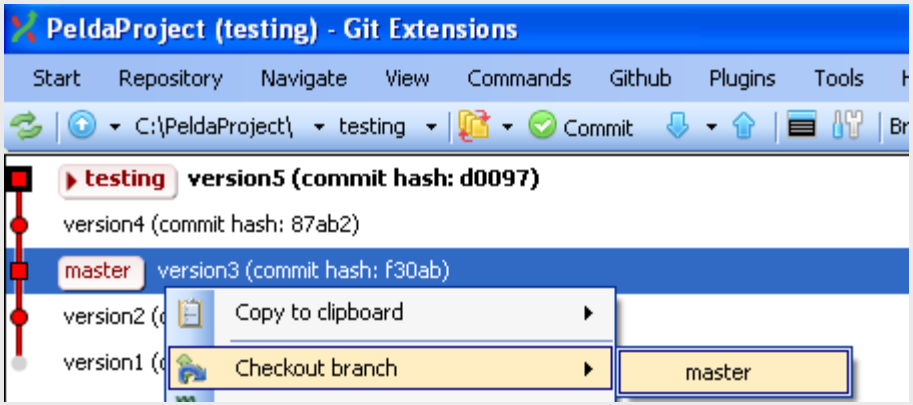
Lokális műveletek – példasor
Commit (7) - rendezés



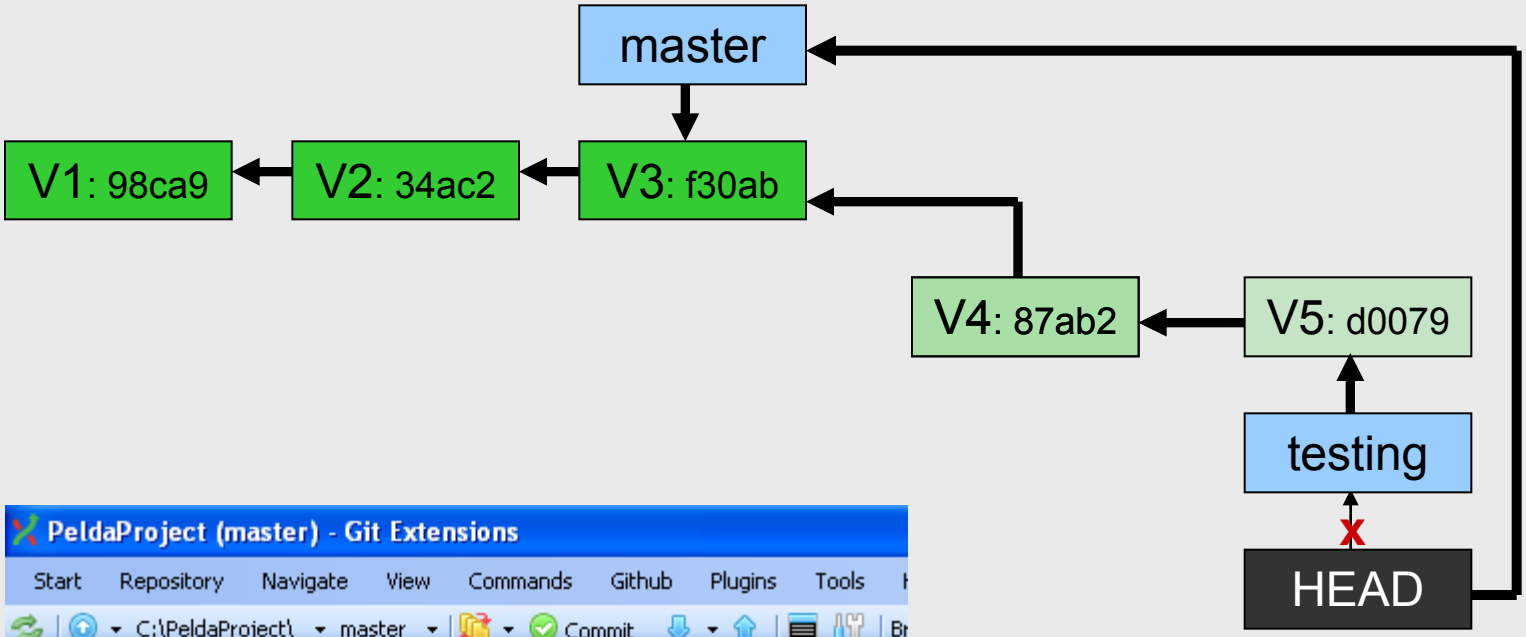


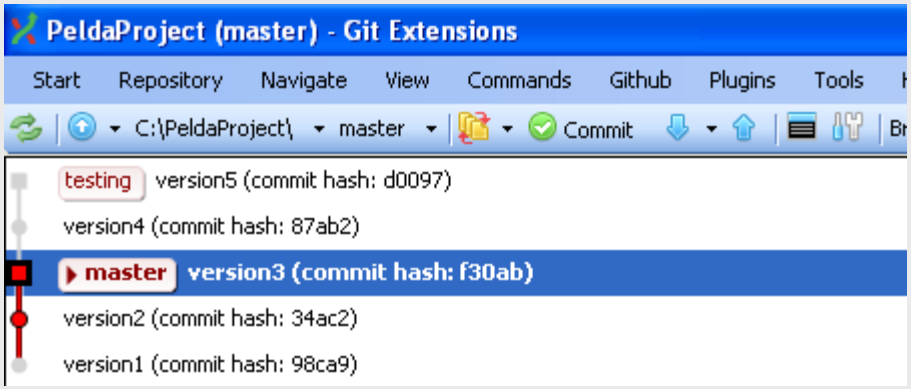
Lokális műveletek – példasor
Checkout branch (8)



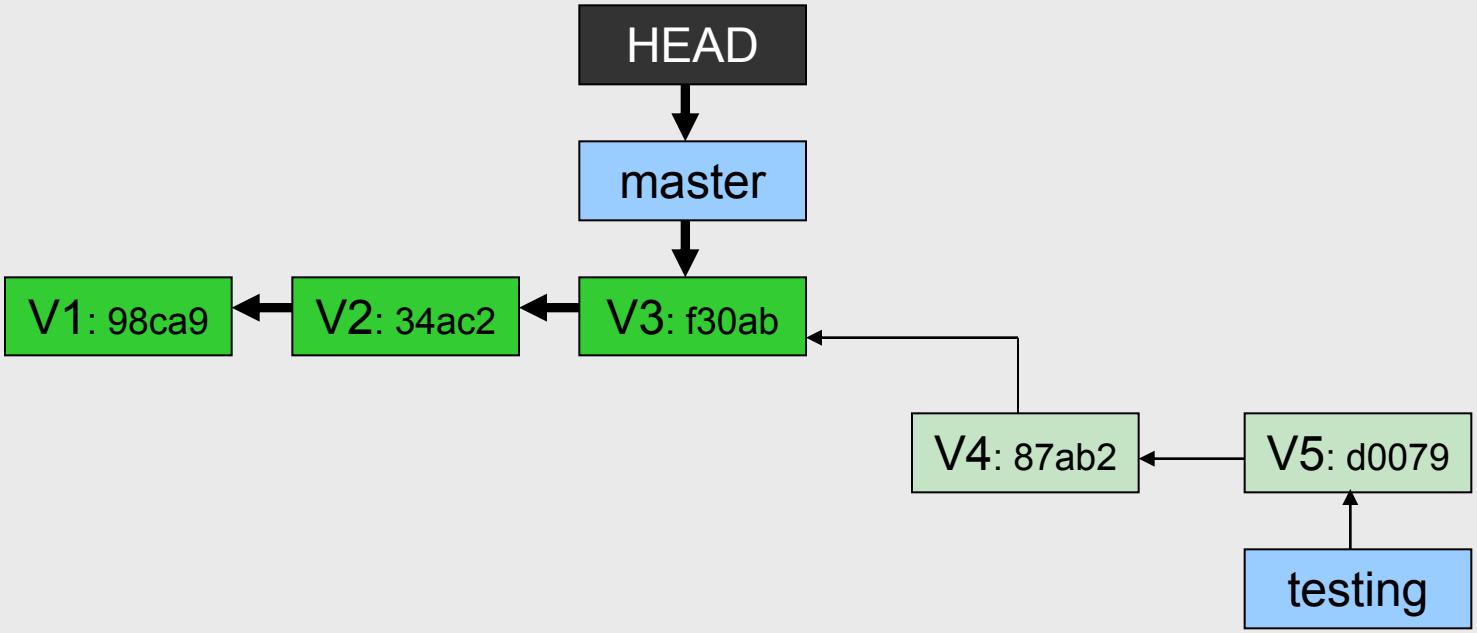


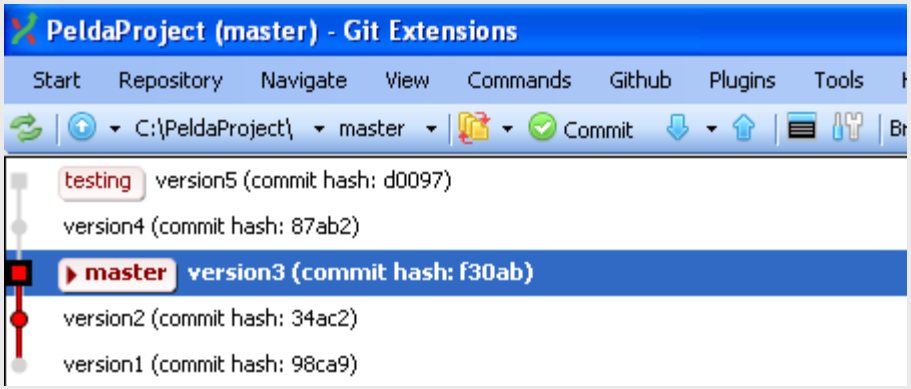
Lokális műveletek – példasor
Checkout branch (8)



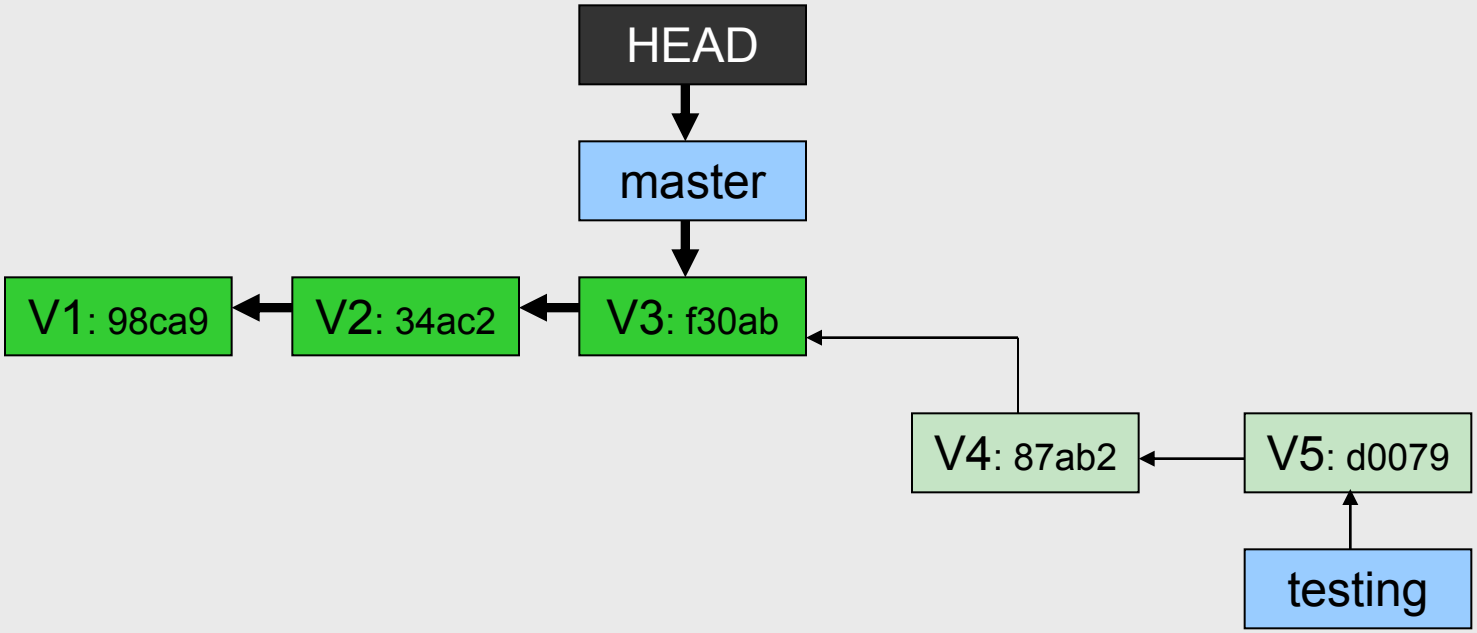


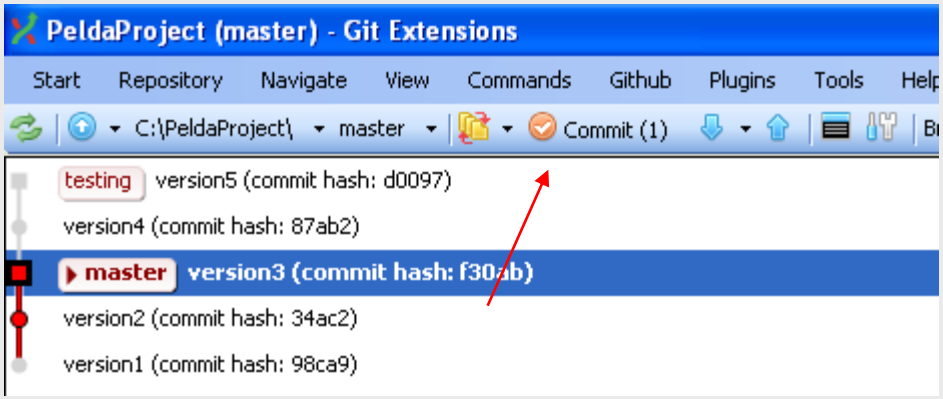
Lokális műveletek – példasor
Checkout branch (8) - rendezés



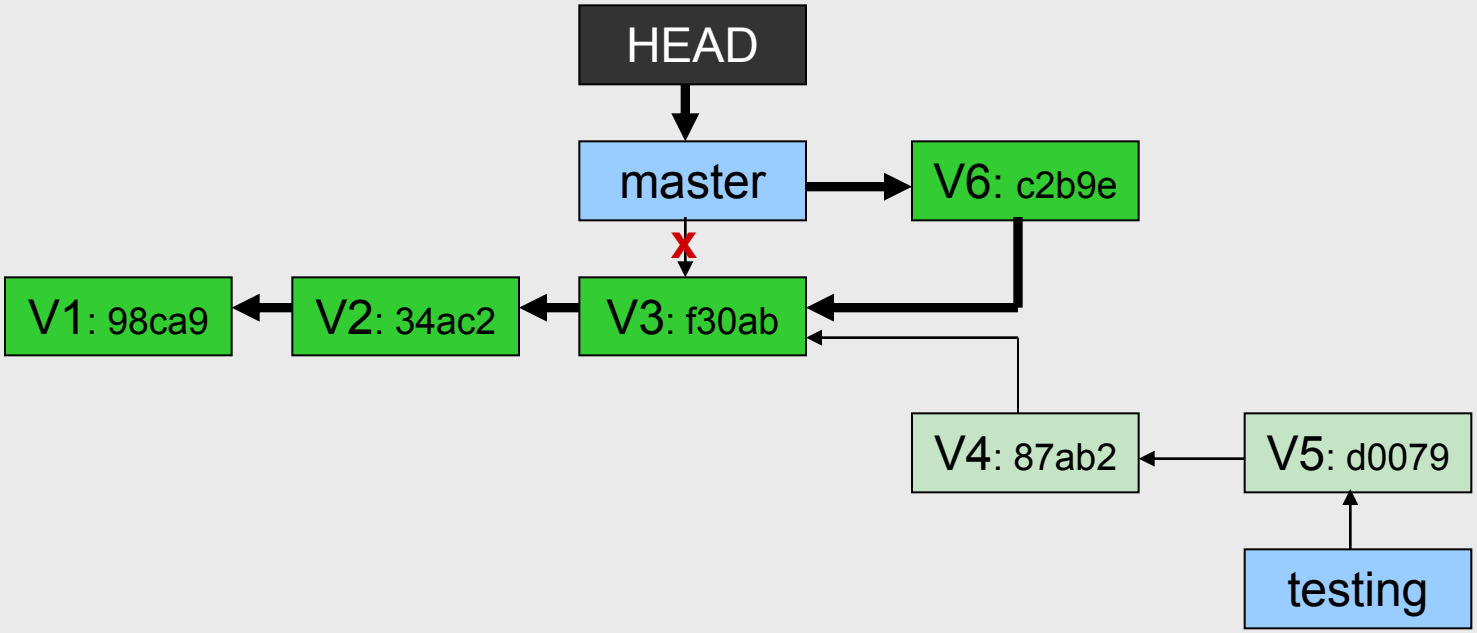


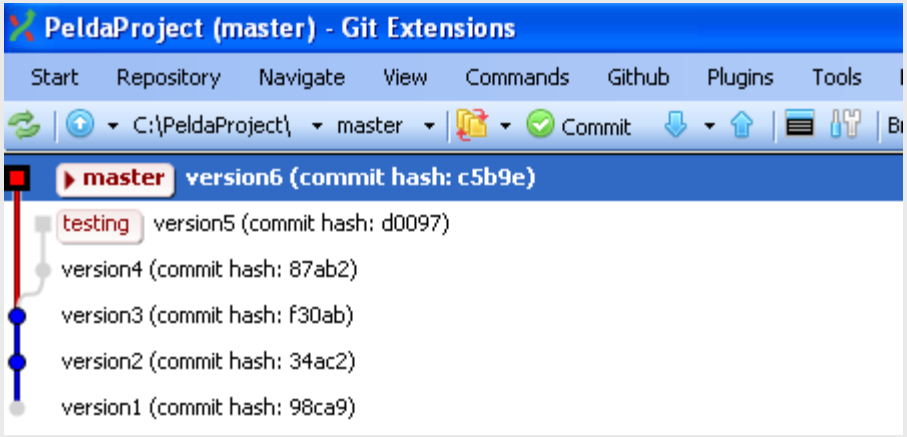
Lokális műveletek – példasor Commit (9)



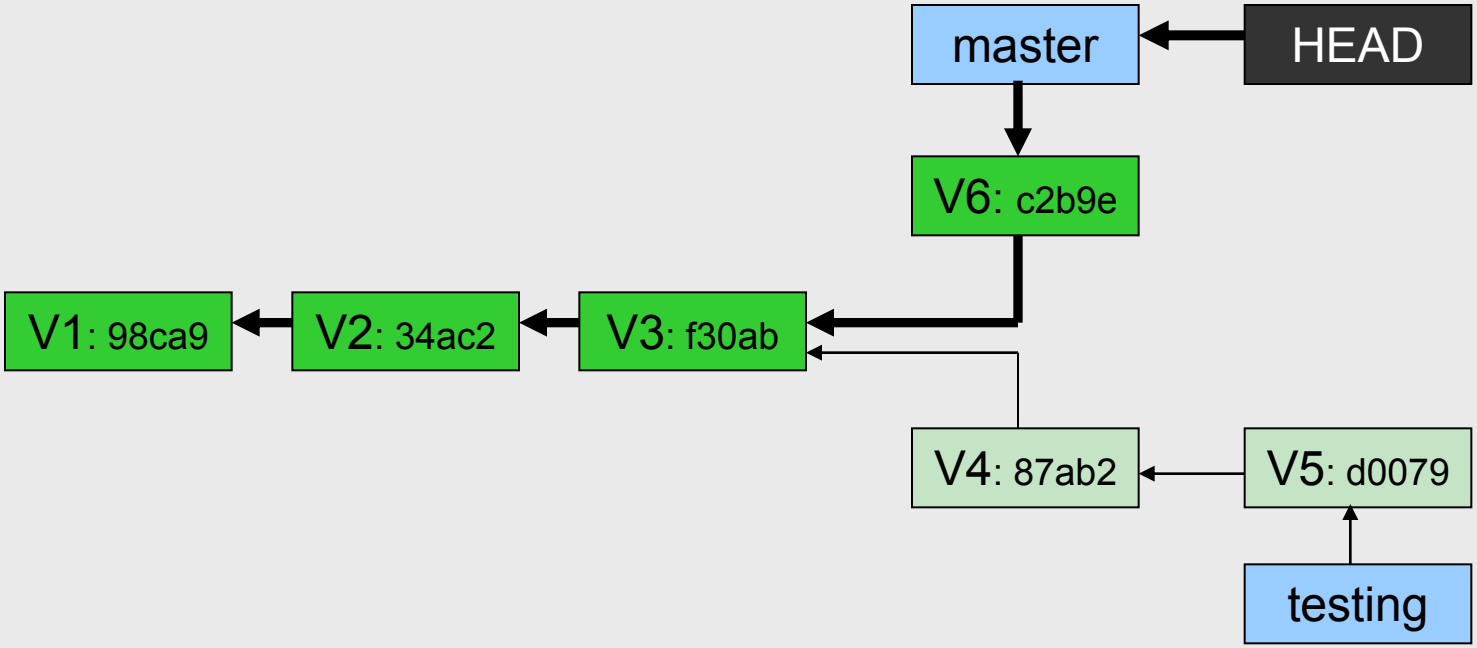


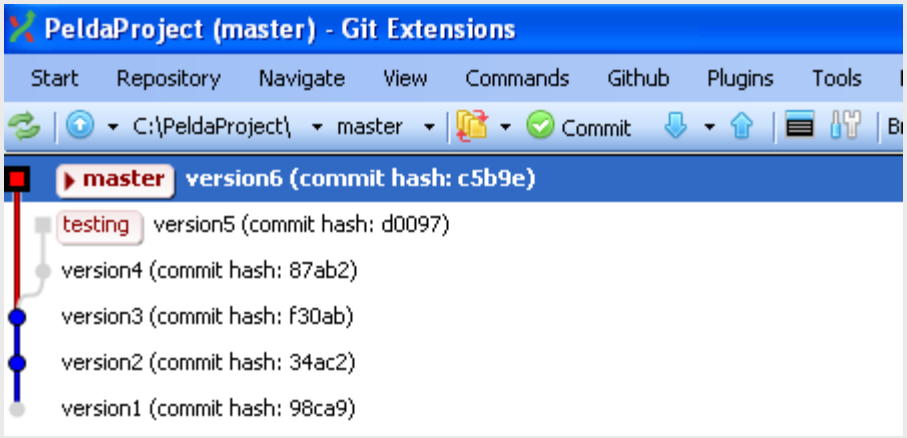
Lokális műveletek – példasor Commit (9)



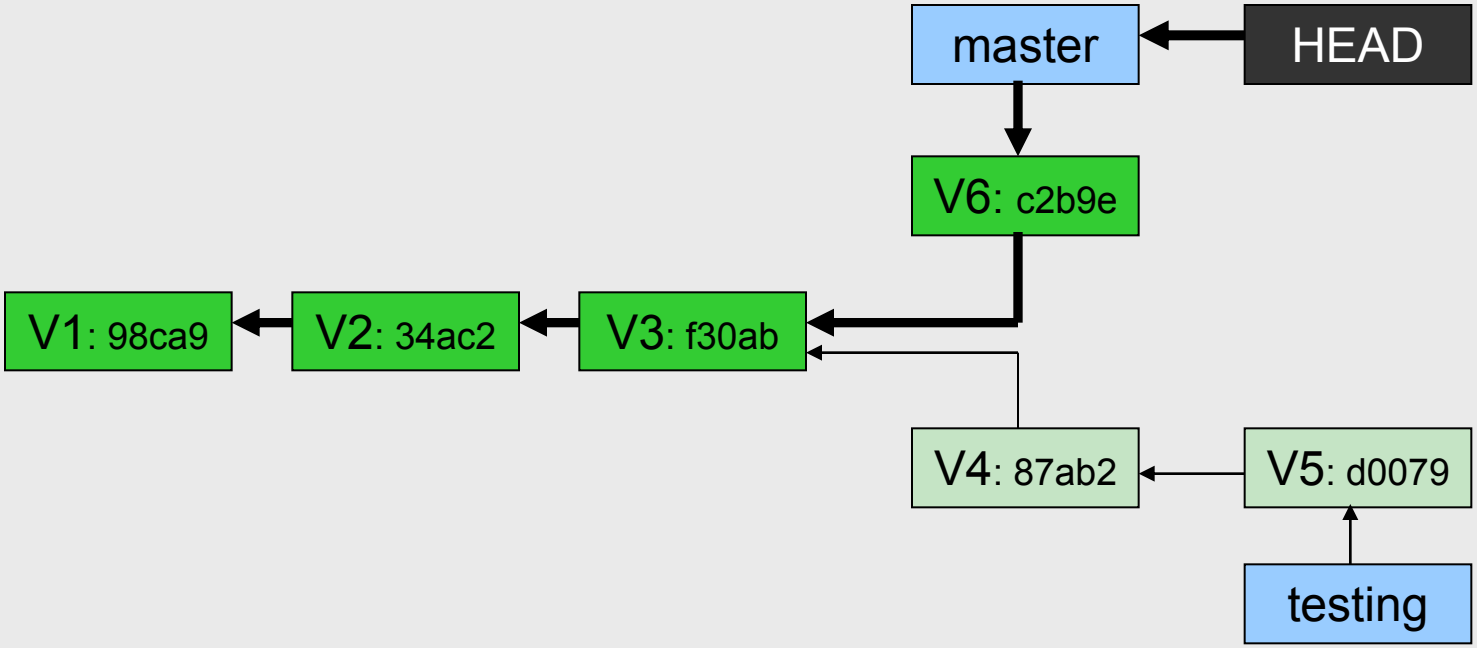


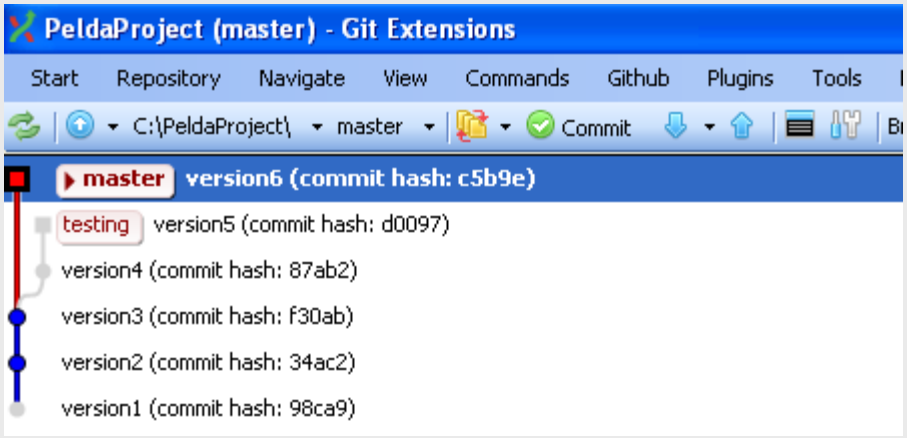
Lokális műveletek – példasor
Commit (9) - rendezés





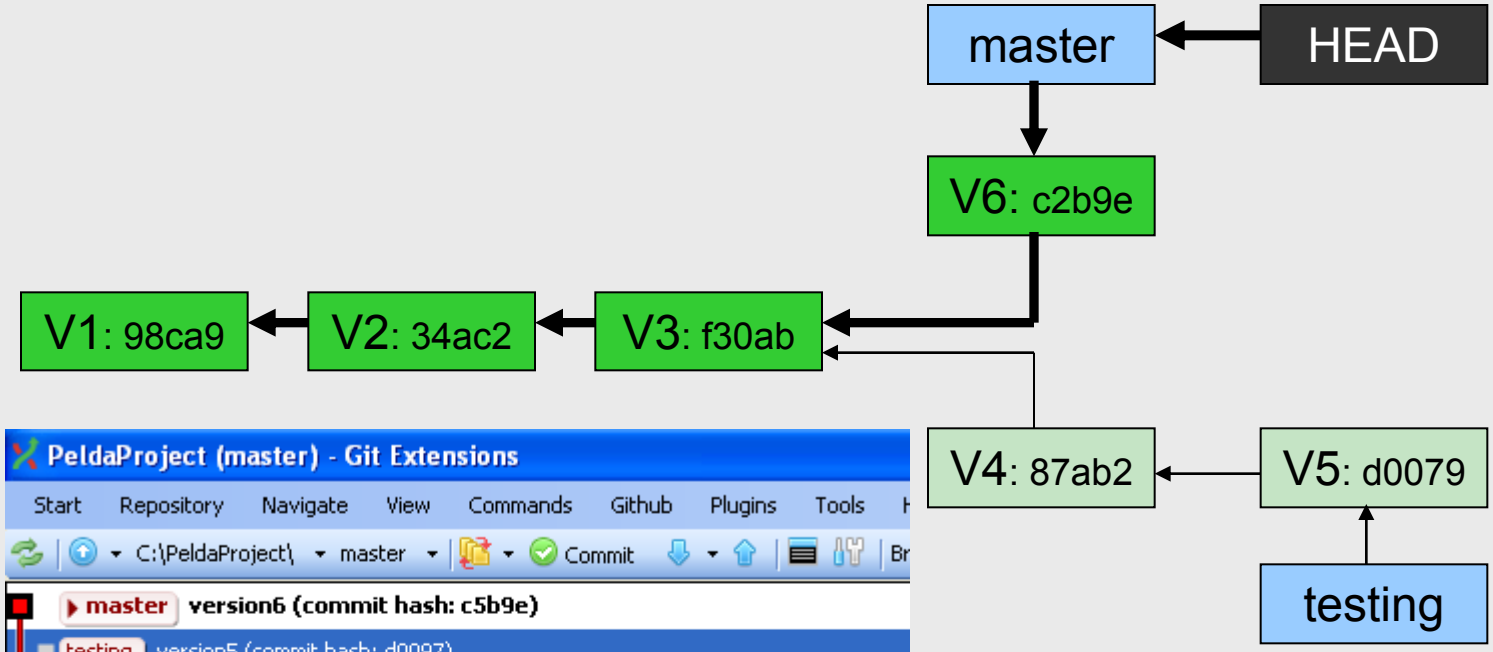
Lokális műveletek – példasor
Delete branch (10)

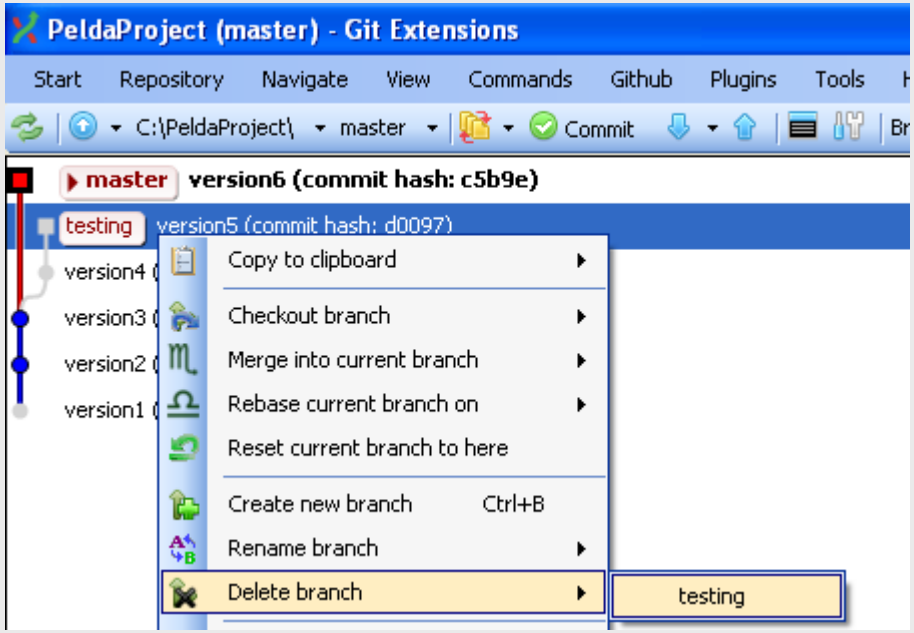




A master ágat nem tudjuk törölni, mert erre mutat a HEAD. Ha ezt szeretnénk törölni, előbb át kell checkout-olnunk a testing ágra, vagy egy revision-re.

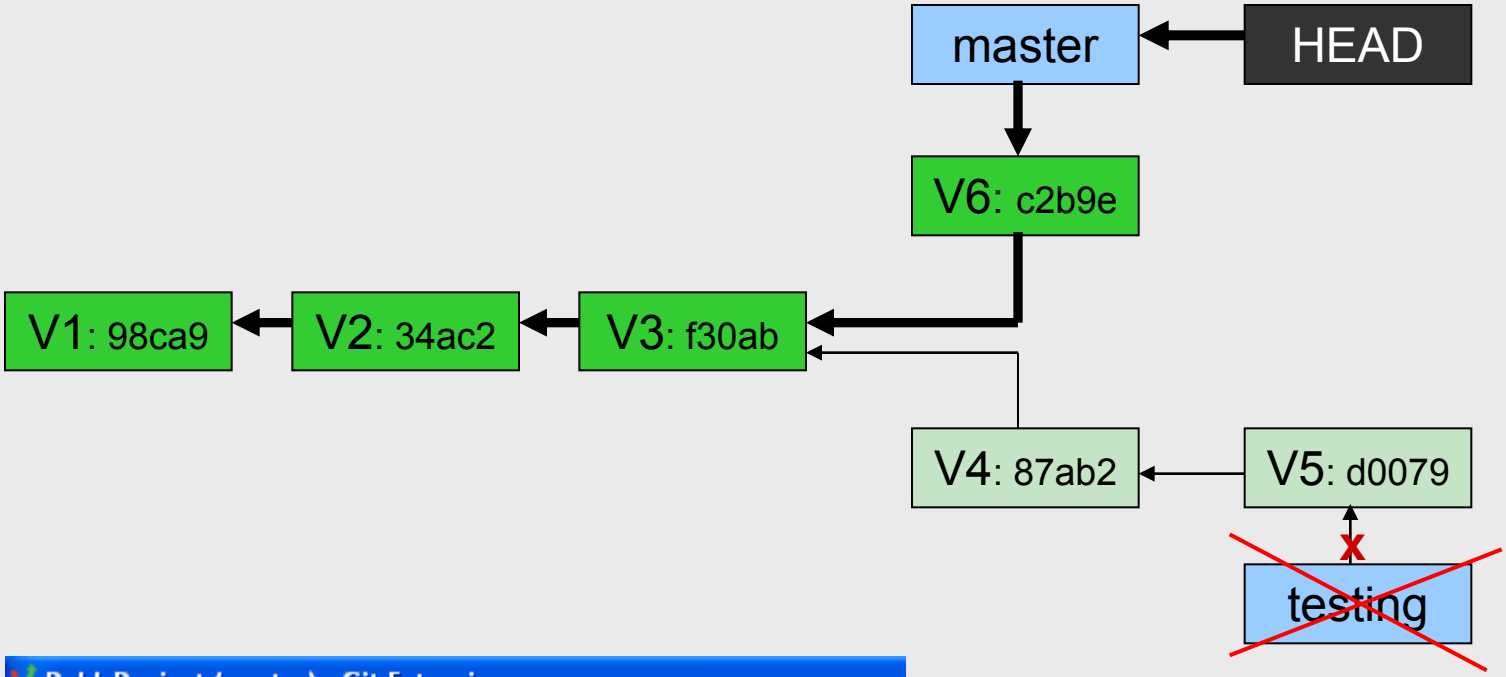
Lokális műveletek – példasor
Delete branch (10)

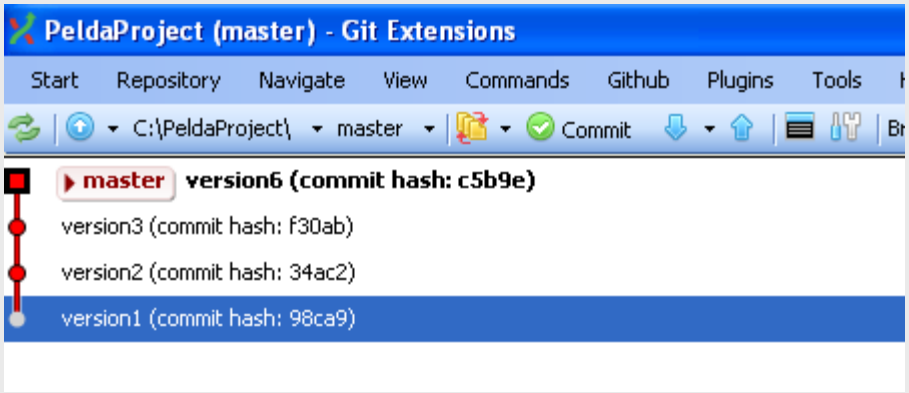




Töröljük a testing ágat!

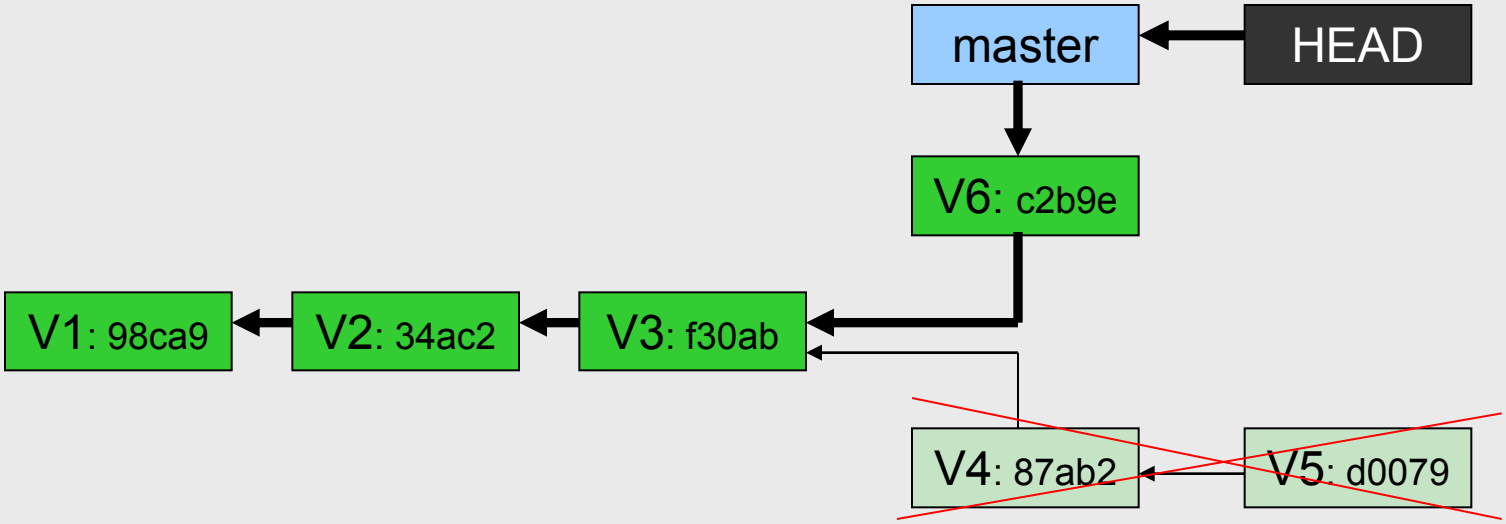
Lokális műveletek – példasor
Delete branch (10)

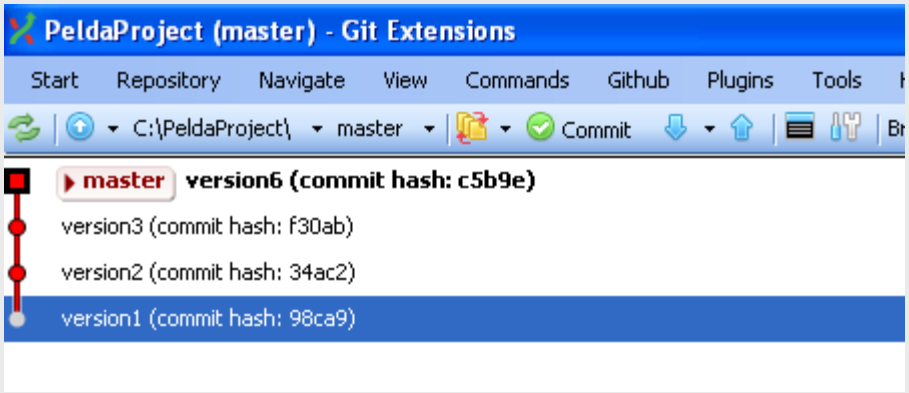




Töröljük a testing ágat!

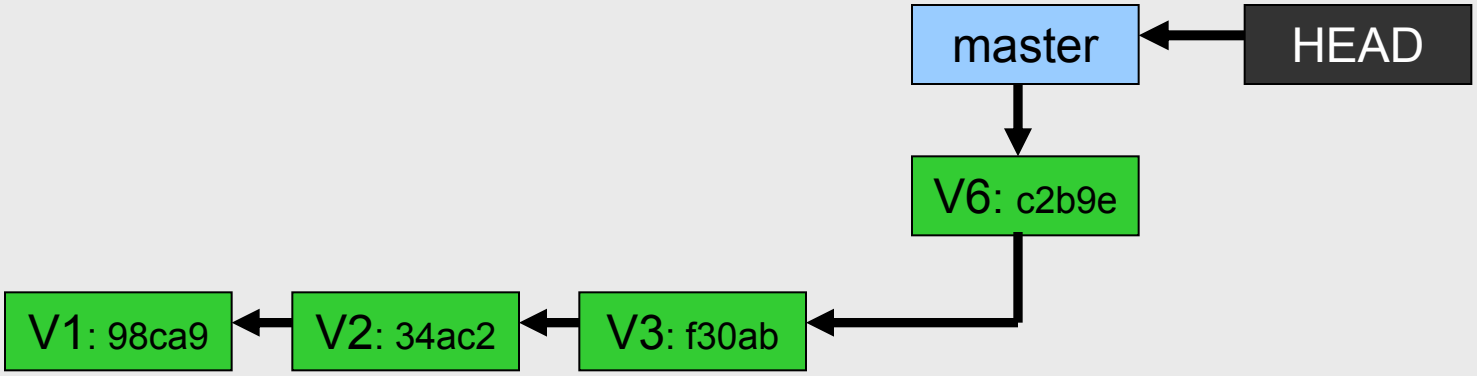
Lokális műveletek – példasor
Delete branch (10)

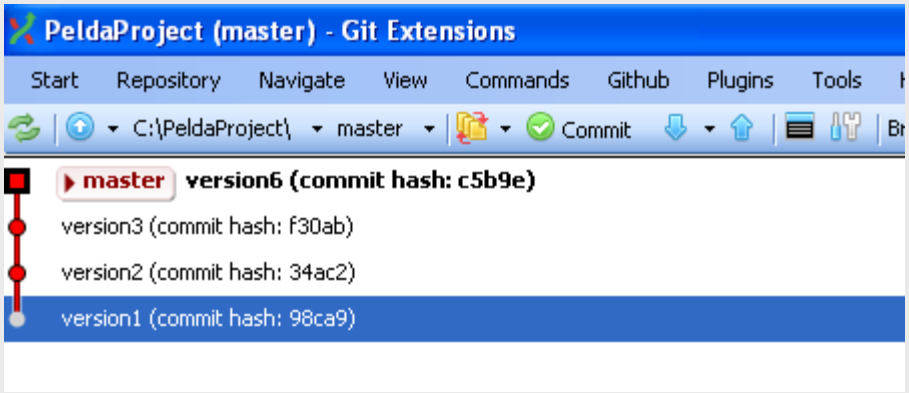




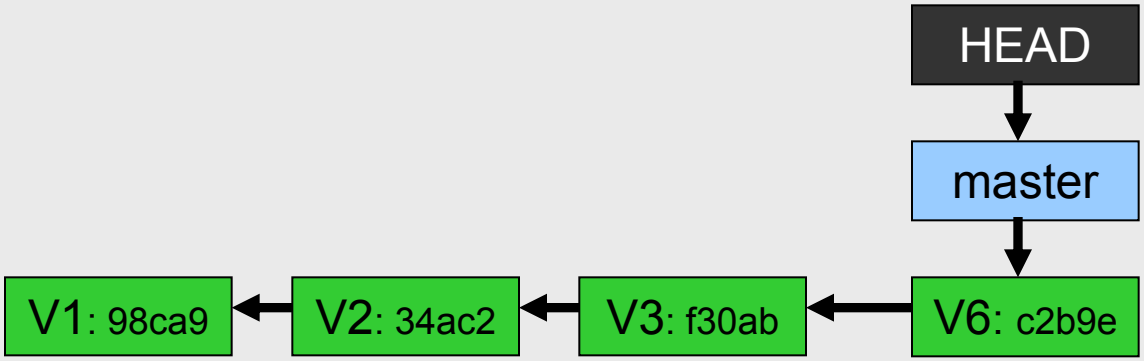
Mivel többé nem mutat semmilyen címke a V4 és V5 revision-ökre (sem helyi címke, sem távoli ágcímke (lásd push és pull művelete után), sem tag (lásd tag-eknél), így, ha törléskor a force opciót is bepipáljuk, akkor ezek a revision-ök is törölve lesznek. Ha nincs bejelölve, akkor pedig nem enged törölni! (lásd a végén a force diát)

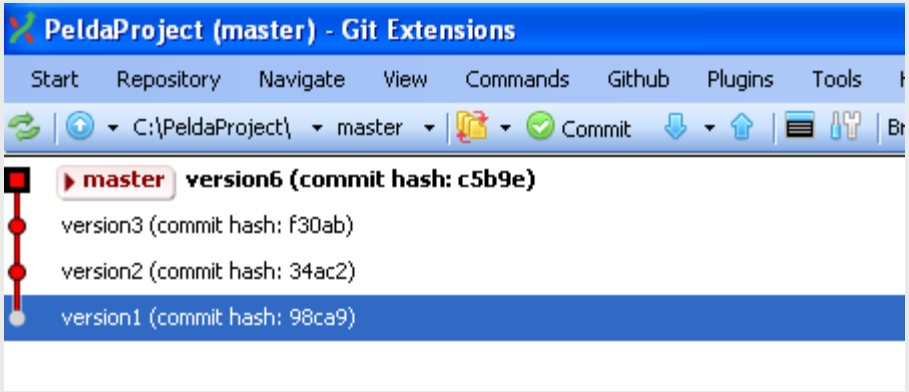
Lokális műveletek – példasor
Delete branch (10)





Lokális műveletek – példasor
Delete branch (10) - rendezés





Elemi lokális tevékenységek (offline műveletek)

Working directory (munkakönyvtár)

Staging area (index/cache)

.git directory (git könyvtár helyi tároló)

Checkout head (checkout revision, checkout branch stb.

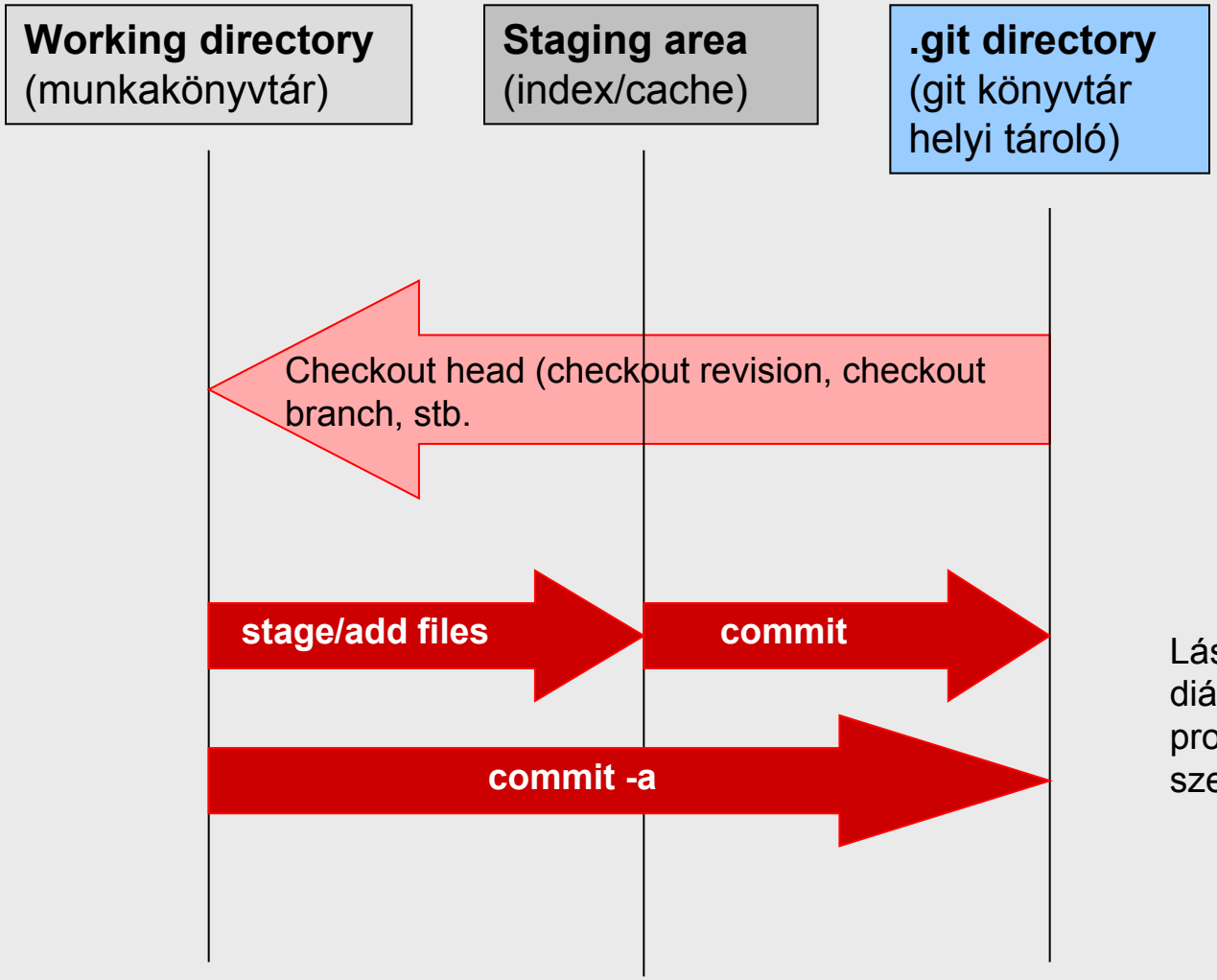
stage/add files

commit

commit -a

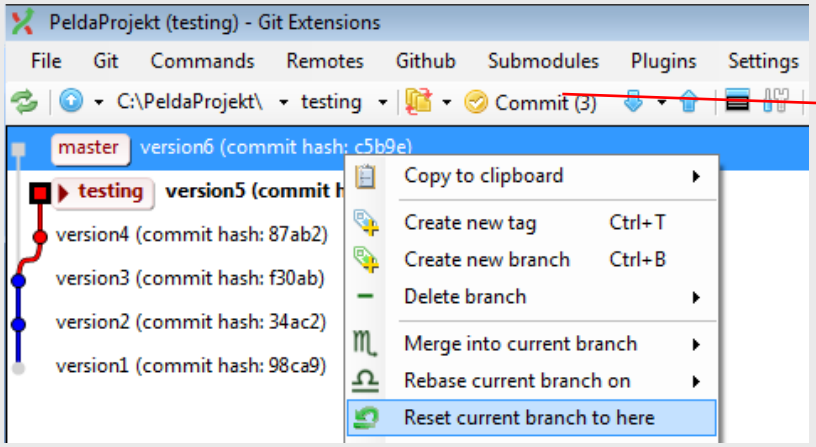
Lásd a checkout-os diákat és a Git projektkönyvtár szerkezete c. diát

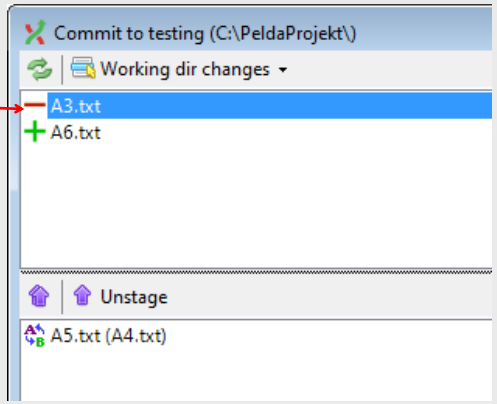
Elemi lokális tevékenységek (offline műveletek)



Lásd a commit-os diákat és a Git projektkönyvtár szerkezete c. diát

Reset branch



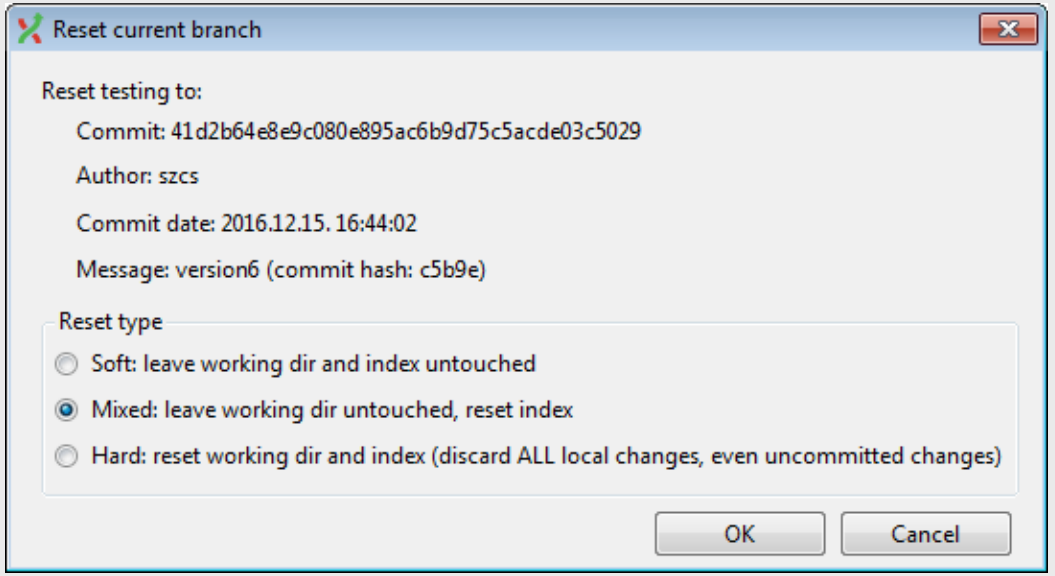


<u>Reset current branch</u> esetén az utjára checkout-olt aktuális branch címkéjének mutatóját átállítom egy általam megadott revision-re.

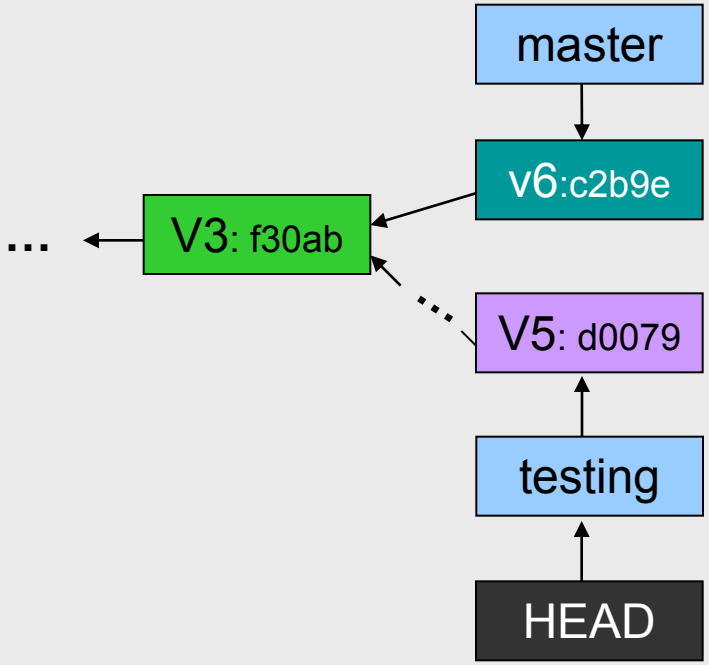
A példában a Working directory tartalma 3 fájlban különbözik a testing (azaz jelen esetben a HEAD) tartalmától.

Ha nem lenne különbség, azaz a reset branch művelet előtt commitálunk , akkor mindegy, hogy hard , soft , vagy reset-re nyomunk!

A következő példában a tesing branch-et akarom a V6 revision-re átállítani



Reset branch - reset előtti állapot (előző példa alapján)



Megjegyzés:

Az állapotokat úgy érdemes a megérthetőség szempontjából vizsgálni, mintha halmazműveletekről beszélnénk. A halmaz egy eleme a fájlnak egy sorát jelenti!

Working dir. changes = modified files + staged files

Working dir. changes: 3 fájlmódosítás

Working Directory Changes:

X állapot és V5 állapot különbséghalmaza. Két része van:

- **Nem commitált bejegyzések (2 fájlmódosítás)**
- **Staging area (1 fájlmódosítás)**

Master:
c2b9e állapot fájljai (version5)

HEAD \ Testing:
d0079 állapot fájljai (version4)

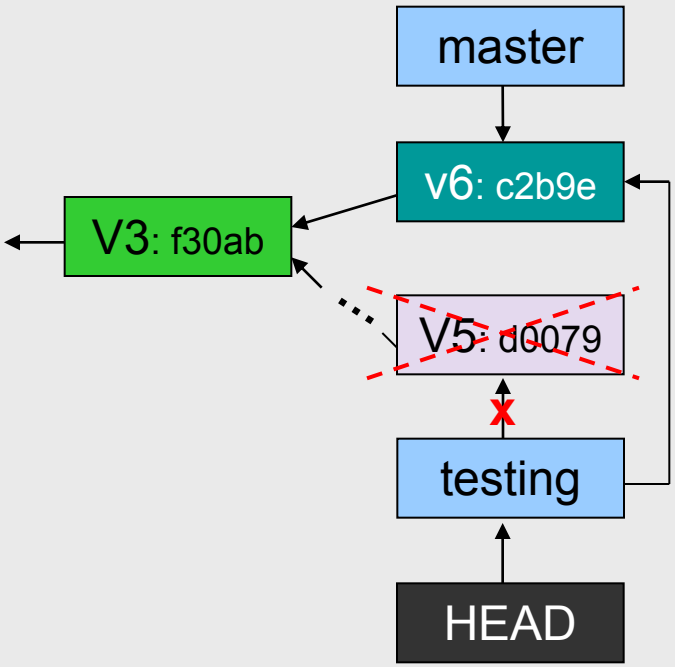
Working directory: X állapot

d0079 állapot fájl-jai (mivel a HEAD erre a revision-re mutató testing ágra mutat)

+ az azóta eltelt módosítások (3 fájlmódosítás)

Reset branch - hard reset

reset working dir. and index



Working dir. changes:
ÜRES

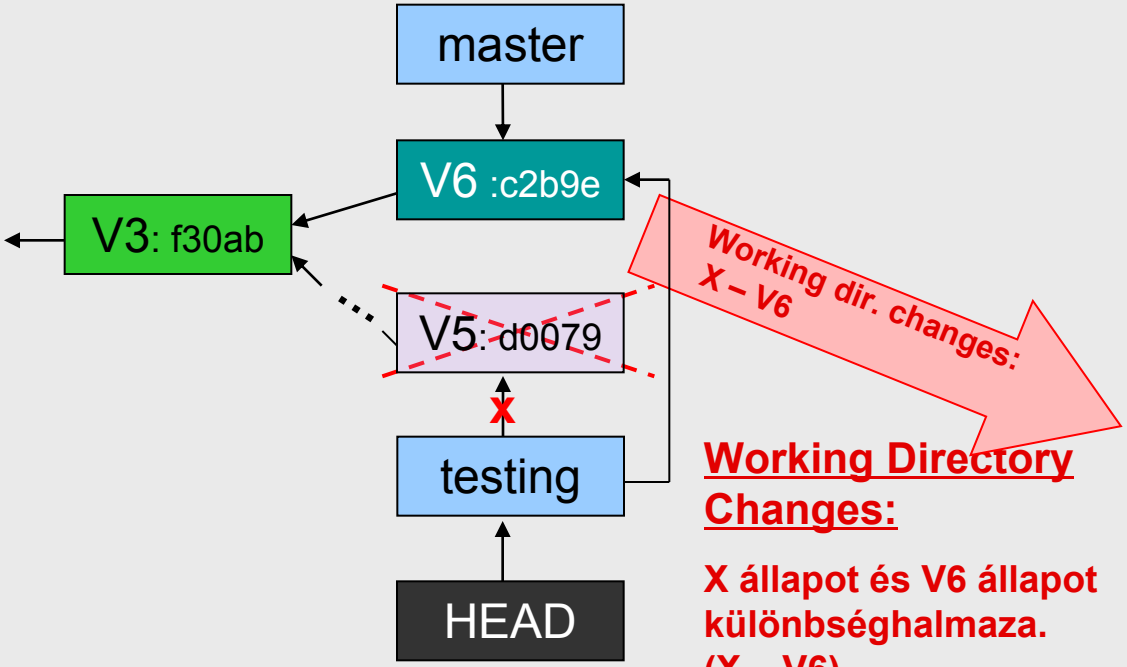
Master:
c2b9e állapot fájljai (version5)

HEAD \ Testing:
c2b9e állapot fájljai (version5)

Working directory: ÜRES!

Reset branch - mixed reset

leave working dir. untouched, reset index



Working Directory Changes:

X állapot és V6 állapot különbséghalmaza. (X – V6)
Két része van:

- **Nem commitált bejegyzések**
- **Staging area: ÜRES LESZ!**

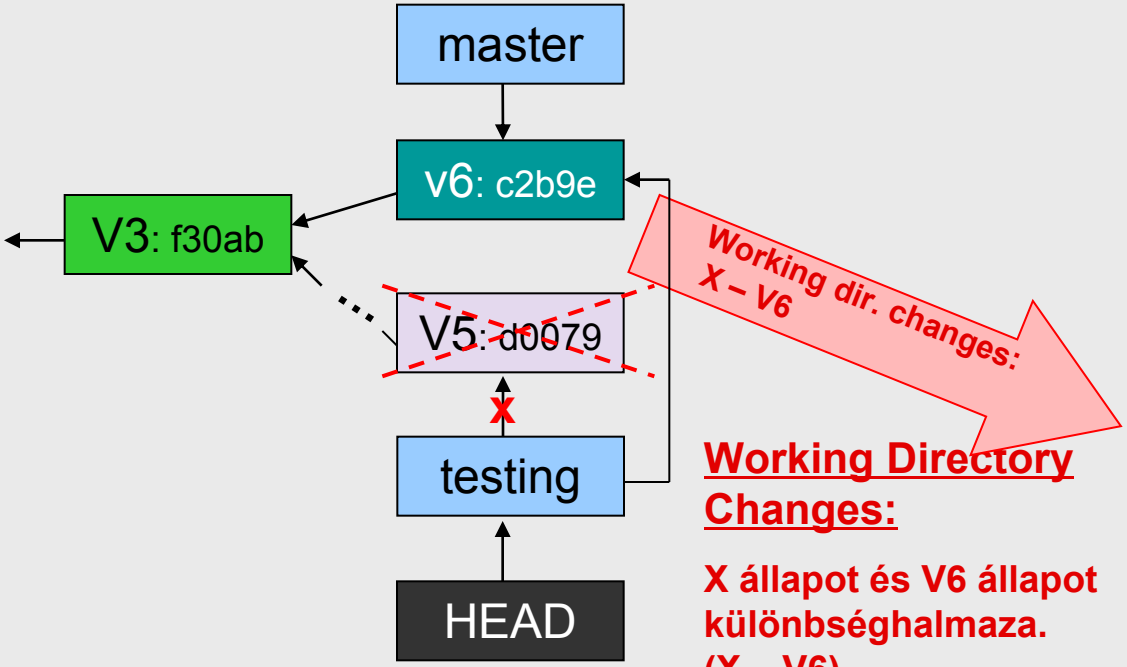
Master:
c2b9e állapot fájljai (version5)

HEAD \ Testing:
c2b9e állapot fájljai (version5)

Working directory: X állapot, azaz NEM MÓDOSUL A TARTALMA A RESET ELŐTTI ÁLLAPOTHOZ KÉPEST!

Reset branch - mixed reset

leave working dir. untouched, reset index



Working Directory Changes:

X állapot és V6 állapot különbséghalmaza. (X – V6)
Két része van:

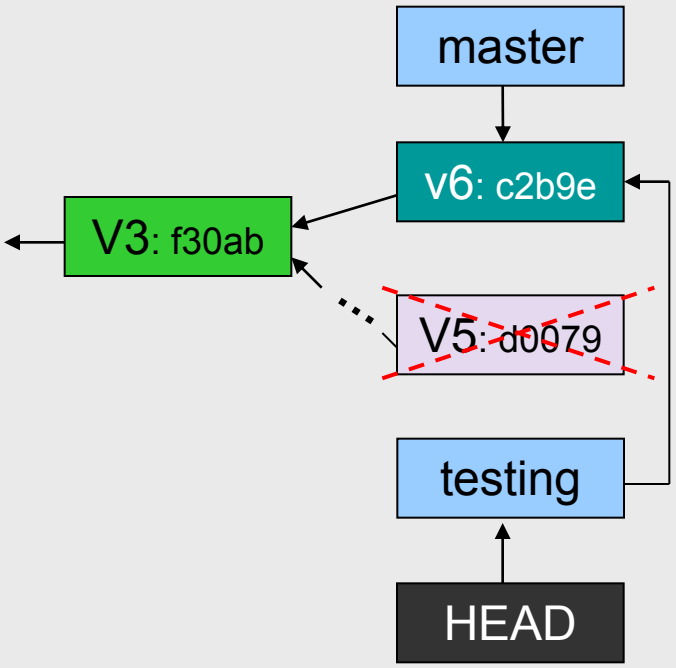
- **Nem commitált bejegyzések**
- **Staging area:**
  reset előtti állapotban lévő staging area-ban lévő fájlváltozatok közül azok, amelyek nincsenek benne a v6-ban.

Master:
c2b9e állapot fájljai (version5)

HEAD \ Testing:
c2b9e állapot fájljai (version5)

Working directory: X állapot, azaz NEM MÓDOSUL A TARTALMA A RESET ELŐTTI ÁLLAPOTHOZ KÉPEST!

Reset branch hatása



Megjegyzés:

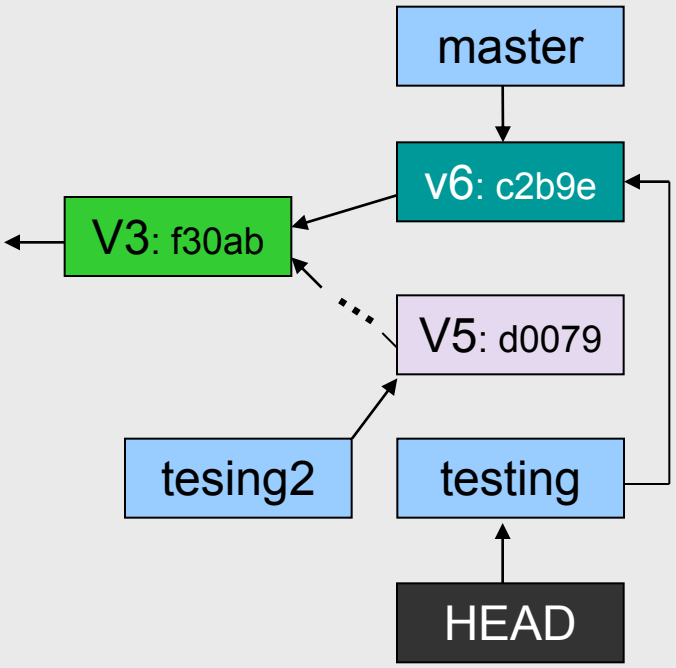
Ahogy ez a korábbi példáknál is említve volt, a Reset, Checkout, vagy bármilyen Git művelet után az(ok)ra a revision(ök)re, amely(ek)re nem mutat közvetlenül, vagy közvetetten:

- lokális branchcímke
- remote branchcímke (lásd push és pull művelet után távoli ágcímkék-nél)
- tag (lásd később)

vagyis nem lehet egy címkétől közvetlenül, vagy közvetetten nyilak mentén eljutni hozzá (jelen esetben a d0079 revision), menthetetlenül törlődni fognak!

Természetesen, ha a resetnél a soft, vagy mixed opciót választjuk, akkor a V5 állapot V6-ben nem lévő része mind bekerül a nem commitált fájlok közé

Reset branch hatása



Megjegyzés:

Ahogy ez a korábbi példáknál is említve volt, a Reset, Checkout, vagy bármilyen Git művelet után az(ok)ra a revision(ök)re, amely(ek)re nem mutat közvetlenül, vagy közvetetten:

- lokális branchcímke
- remote branchcímke (lásd push és pull művelet utáni origin ágcímkék-nél)
- tag (lásd később)

vagyis nem lehet egy címkétől közvetlenül, vagy közvetetten nyilak mentén eljutni hozzá (jelen esetben a 87ab2 revision), menthetetlenül törlődni fognak!

Természetesen, ha a resetnél a soft, vagy mixed opciót választjuk, akkor a V5 állapot V6-ben nem lévő része mind bekerül a nem commitált fájlok közé

Nyilván, ha a V5 állapotra más ág is mutat, akkor nem fog a revision elveszni, viszont azért maradt az ábrán a színe áttetsző, mert a HEAD-en keresztül nem elérhető!

Reset branch / Revert commit **

- Előfordulhat olyan eset is, hogy nem akarjuk az egész branch-et törölni, csupán az utolsó néhány revision-t szeretnénk törölni, egybevonni, vagy pedig a megadott branch aktuális állapotának egy másik branch bizonyos revision-jét szeretném megadni. Ilyenkor is nagyon hasznos funkció a reset branch. A megadott revision-re állva jobb gomb, majd nyomjunk a reset branch pontra.
- Az első pontot addig lehet használni, amíg nem push-oltunk. Utána inkább kézzel, egy új commit-ban állítsuk vissza az eredeti commit-ot. Ilyenkor használhatjuk inkább a revert commit-ot (a revertálandó revision-re ráállva jobb gomb, majd revert commit, ezzel a revision-módosítás diff-jének negligáltját commitáljuk, de nem tanácsos a használata. Főleg akkor nem, ha merge commit-ot akarunk revert-álni!!)

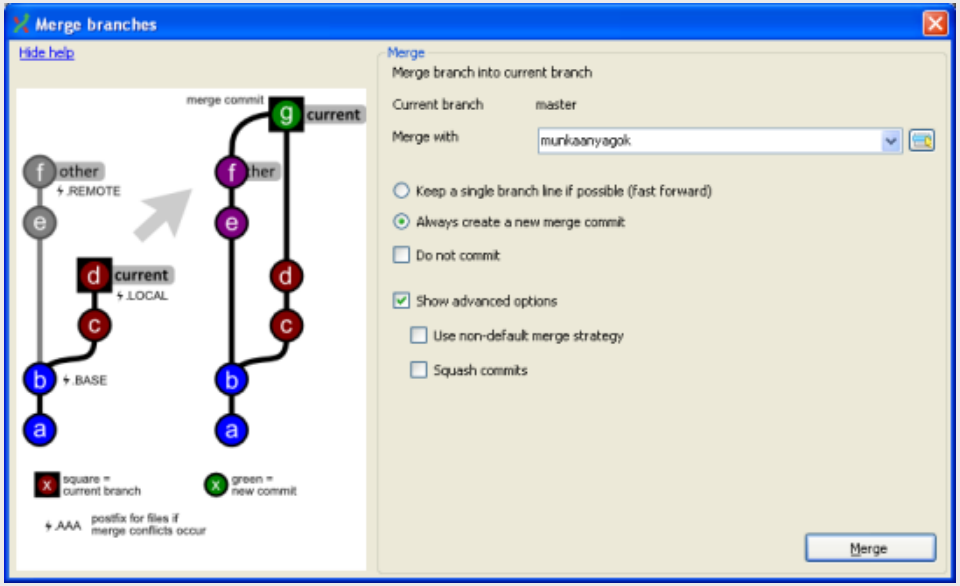
Merge 1

Amikor egy modullal készen vagyunk és vissza akarjuk vezeti a főágba, akkor azt mondjuk, hogy merge-eljük az adott modul ágát a főágba. Mindig egy ágba tudunk csak merge-ölni (amire a HEAD mutat), de egyszerre több ágat is merge-ölhetek

Ha vannak nem kommitált fájljaink, akkor azokat előbb kommitoljuk egy új revision-be, majd álljunk arra az ágra, amelyikbe merge-ölni szeretnénk. Eztán Commands->Merge, majd a jobb oldalt lévő ablak jön be.

Az ablak szerkezete:

- **Current branch:** az az ág, ahol épp állunk (ha no branch, nem fog merge-ölni)
- **Merge with** : a legördülő menüből kiválaszthatjuk, hogy mit akarunk merge-ölni . (ha több ágat is merge-ölni szeretnénk, a gombbal Select multiple branch ablak behozása)



Ha a **Don't commit-ot** bepipáljuk, akkor nem jön létre automatikusan a g commit , hanem a két ágban módosult file-ok a commit-nál a *working directory changes*-ben jelennek meg modified files-ként, azaz nekünk kell kézzel kommitálni

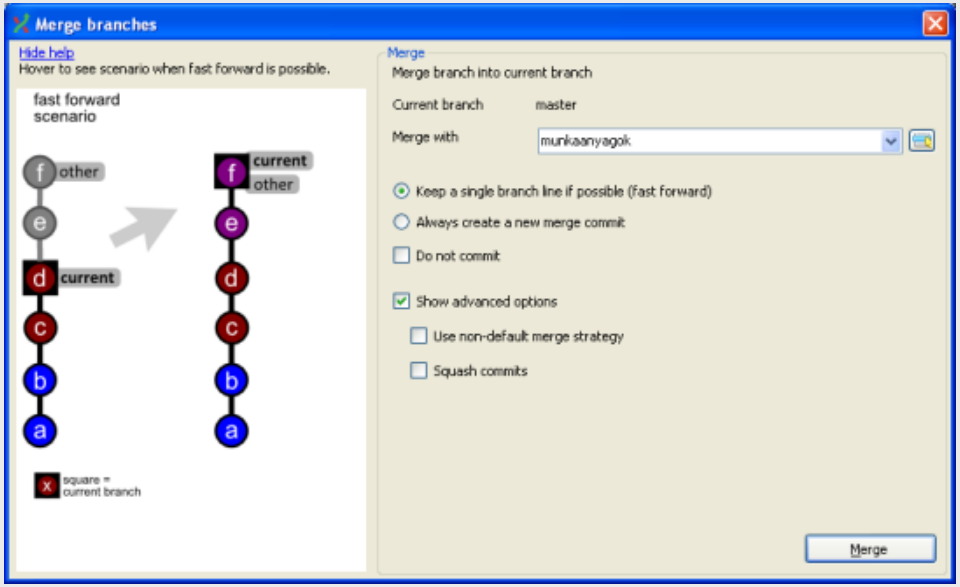
Parancssorosan: álljunk rá a master legutolsó revision-jére, majd:

git merge munkaanyagok vagy **git merge –no-commit munkaanyagok**

Merge 2

Merge branches ablak

A **g** a merge által létrehozott új revision lesz, amely az **f és az e** –ben történt változások uniója. A hash-fa szerkezete szempontjából mindegy, hogy mit mibe merge-ölünk , de ha a *FOMI_munkaanyagok* **távoli** ágat a jelenlegi *master* -be merge-eljük , a master ágcímke a g commit-ra fog mutatni, azaz benne lesz a FOMI_munkaanyagok ág **f,e** commit-ja is, de a FOMI_munkaanyagok ág mutatója továbbra is az f revision-re fog mutatni, azaz nem lesz benne a master ágban elkészített **c és d** revision-módosítások . Merge nyilas példát lásd a Pull : fetch+merge diáknál a merge résznél



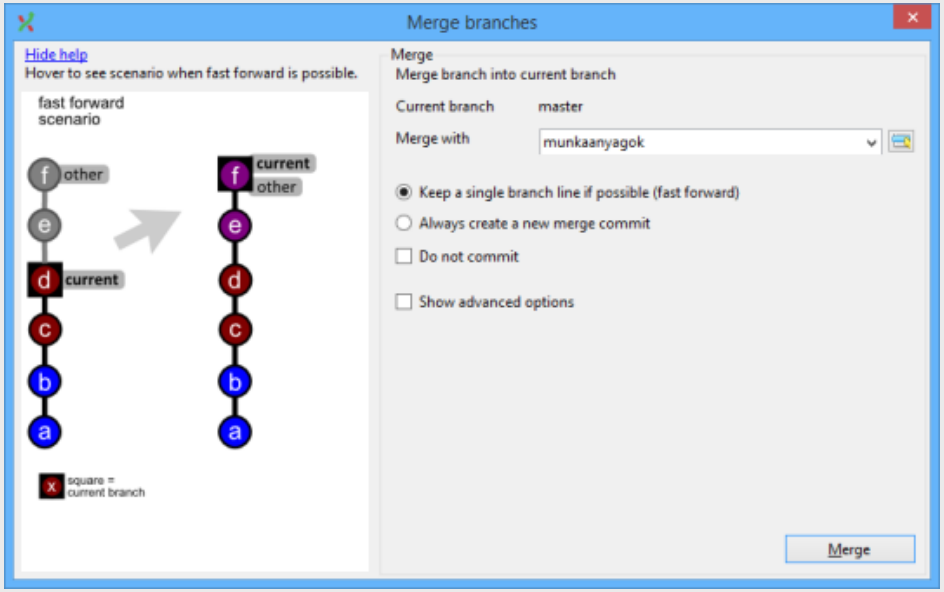
Parancssorosan: álljunk rá a master legutolsó revision-jére, majd:

git merge munkaanyagok vagy **git merge –no-commit munkaanyagok**

Fast forward *

Fast Forward opció:

Ha a master-ben nem történt módosítás a FOMI_munkaanyagok kiágaztatása óta, és ha ez az opció van bejelölve, akkor nem jön létre merge commit (g revision), hanem csak az ág címkéjének mutatója állítódik át (lásd a példát két diával hátrébb). Ha mindkét ágban történt módosítás, akkor viszont mindegy, hogy melyik pontot választjuk ki.

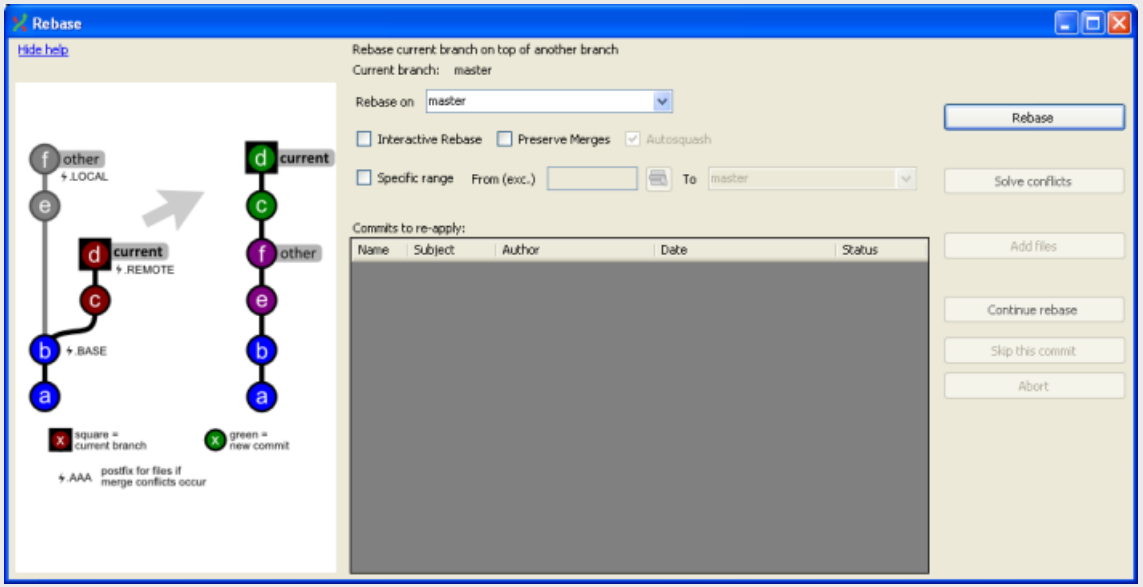


Parancssorosan: álljunk rá a master legutolsó revision-jére, majd:

git merge munkaanyagok vagy **git merge –no-commit munkaanyagok**

Rebase *

- **A <u>rebase</u> ugyanaz, mint a Merge, annyi különbséggel, hogy a két ág gráfpontjait összefésüljük**
- Commands- > Rebase , majd bejön az ablak
- Itt, ha a Show options ablakot benyomjuk, akkor több mindent is be tudunk állítani benne (ugyanúgy, ahogy a merge-nél), de ezt nem szoktuk használni

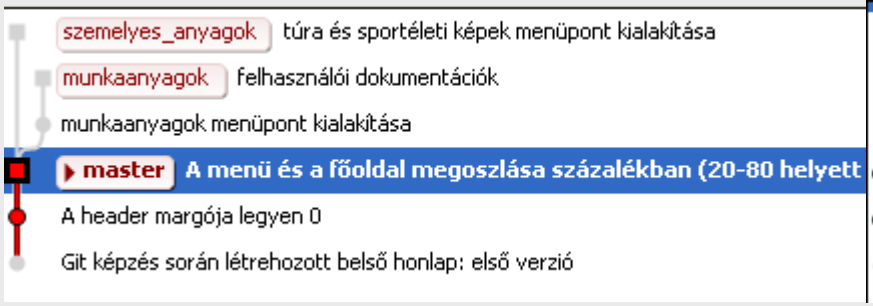


<u>Megjegyzés:</u>

- merge és rebase esetén mindig csak az aktuális branch-be végezzük a módosítást (ha a HEAD-del csak revision-ön állunk, akkor az aktuális revision-be), vagyis a merge-elendő/rebase-elendő ág címkéje nem változik!
- Ha valamit önmagába merge-ölök / rebase-elek , akkor logikusan nem történik semmi!
- Az ábrán a **c** éd **d** revision tartalma rebase után lényegében megmarad, de van, hogy új commit hash-kód jön létre (többnyire, ha konfliktus jön létre)
- Az ablak szerkezeténél current branch rebase on sorrendben ugyanazokat jelenti, mint a merge-nél a current branch, merge with

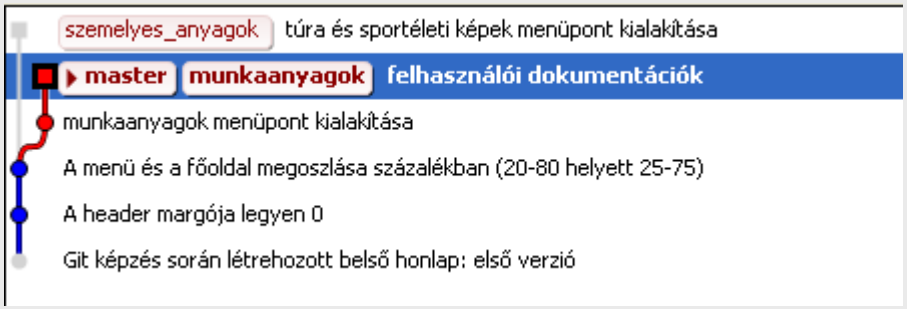
Merge / Rebase / Fast forward

munkaanyagok ág merge-ölése a masterbe



Kiinduló állapot

Merge utáni állapot



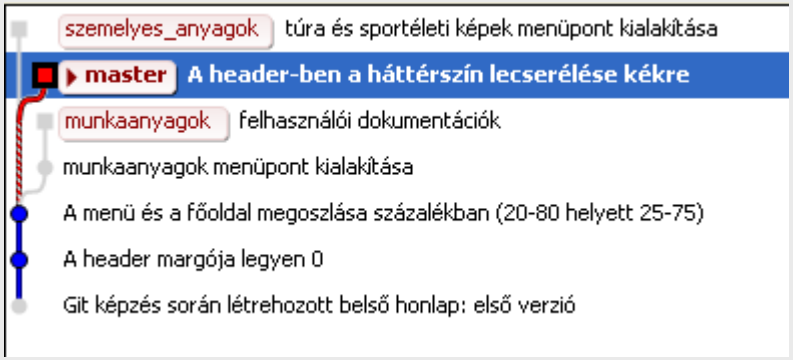
Fast-forward utáni állapot:

Git Extension-ben ebben az esetben az alábbi 3 opció ugyanazt jelenti:

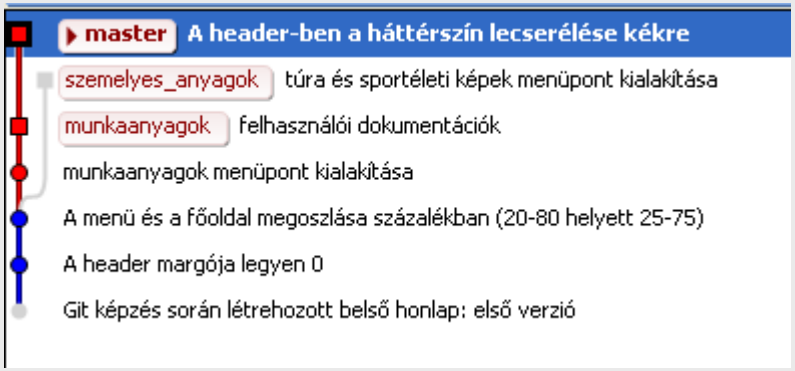
- merge utáni állapot fast-forward opcióval
- rebase utáni állapot
- A master ágcímke mutatóját egy előre pörgetett reset current branch művelettel a „MEPAR: raszterkatalógus dokumentumai” commit message-el ellátott revision-re állítjuk (vagyis oda, ahol a FOMI_munkaanyagok ágcímke is mutat)

Merge / Rebase

munkaanyagok ág merge-ölése a masterbe, ha mindkét ágban történt módosítás, azután, hogy a munkaanyagok ág kiágazott a masterből



Merge/Rebase előtti állapot



Merge utáni állapot (itt mindegy, hogy be van-e nyomva a FastForward állapot)

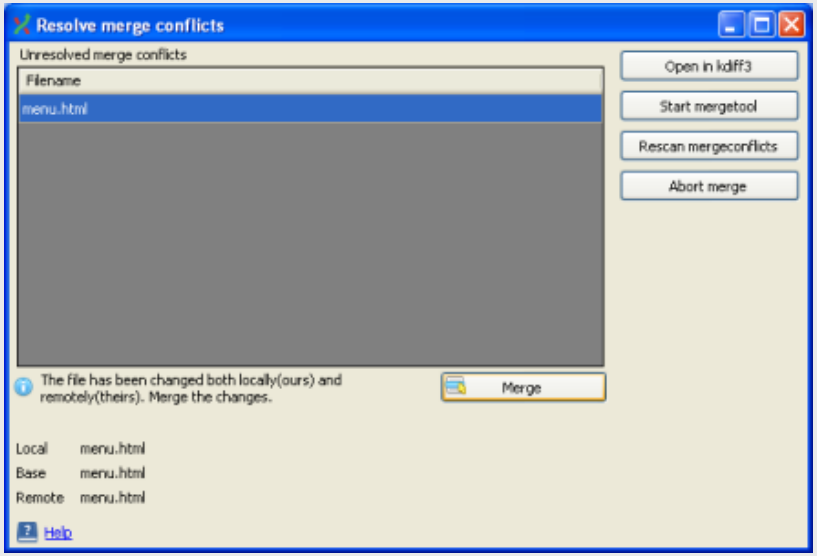
Megjegyzés: Ha merge-elés után vissza akarjuk állítani az előző állapotot, akkor itt a master-t kell „A headerben a háttérszín lecserélése kékre” commit message-el ellátott commitra resetelnünk.

Rebase utáni állapot

Merge conflicts 1

- **Merge konfliktus akkor lép fel, ha**
  - **Két (több) különböző ágban ugyanannak a fájlnak ugyanazon sorában módosítottunk, majd ezt a két (/több) ágat merge-elni szeretnénk.**
  - **Ha az egyik ágban töröltük a fájlt, a másikban pedig módosítottuk, majd ezt a két (több) ágat merge-elni szeretnénk.**

  Ilyen esetekben a merge-ölés során a jobb oldalon látható ablak jön fel.



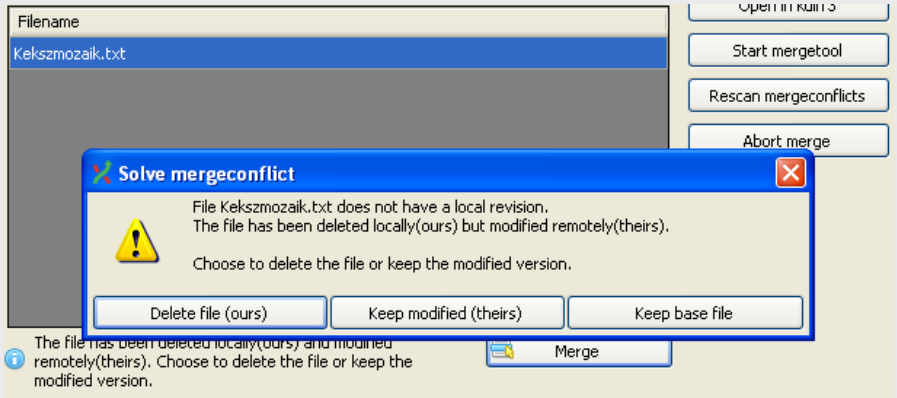
Ablak szerkezete

- **Filename:** itt sorolja fel azokat a fájlokat, amelyekben konfliktus lépett fel.
- **Local/Base/Remote** label-ek: lásd következő ábra
- **Abort merge:** visszatérhetünk a fájl eredeti, lokális águnkban lévő (local) állapotára, mintha a műveletet nem is hajtottuk volna végre
- **Merge / Open KDiff** : a kijelölt file típusának megfelelően a kijelölt file-t merge-öli , ha feltelepítettük a kdiff3-at, akkor leginkább azzal, vagy a Start mergetool gomb-ra nyomva a megfelelő feltelepített merge-tool-al, ami tudja a megadott fájlformátumot kezelni

Merge conflicts 2

Ha a merge-ölési konfliktus-típusnál a fenti 2. eset jön elő (az egyik ágban töröltük, a másikban pedig módosítottuk a sort), akkor a Merge vagy az Open in KDiff3 gombra való kattintás után megkérdi, hogy melyik ágverzió maradjon

- ours: ahova merge-ölünk,
- theirs: amit mergeölünk,
- base: utolsó állapot, amikor még mind2-ben u.az volt

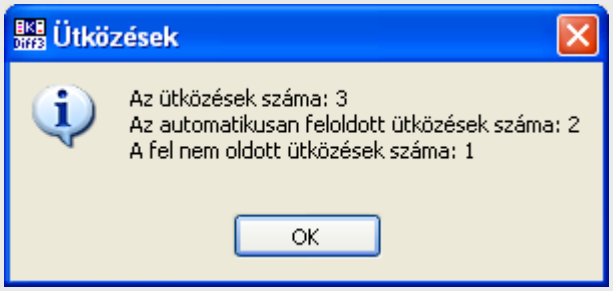


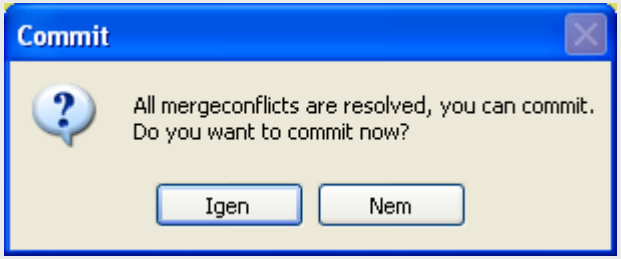
(itt a delete azt jelenti, hogy úgy módosítottuk a fájlt, hogy letöröltük, azaz itt azt szeretnék, hogy a törlés módosítása maradjon meg, vagyis töröljük ki a fájlt.

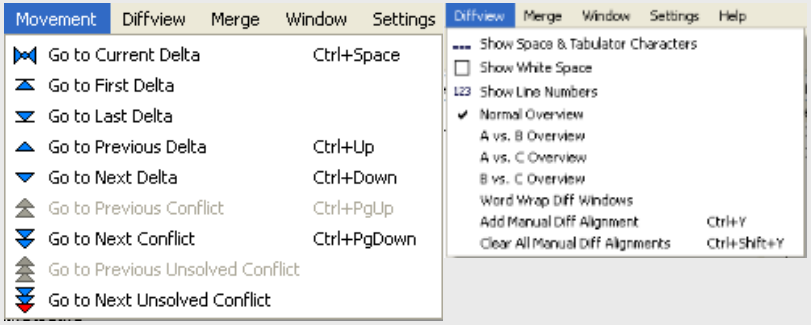
Megjegyzés:

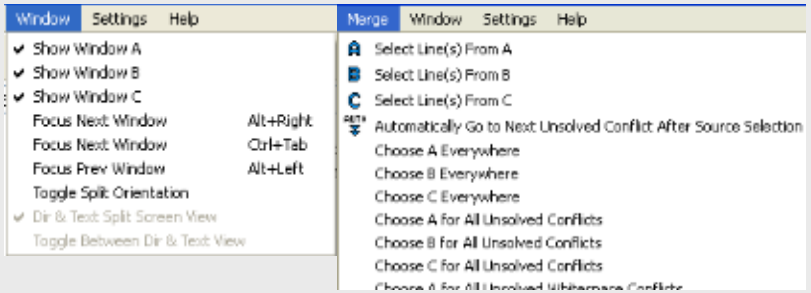
- **Dokumentum, és bináris fájlok esetén nem lesz diff-ünk, a KDiff3-al nem fogjuk látni a különbségeket, ilyenkor a fájlformátumnak megfelelő merge-tool-okat is fel kell telepítenünk ahhoz, hogy ezekben a fájlokban is meg tudjuk vizsgálni a különbségeket!**
- Ha Notepad++-al dolgozunk, akkor érdemes az UTF8 without Boom kódolást beállítani, mielőtt szerkesztjük a szöveget!

Merge conflicts 3









Ha mindkét helyen módosították a fájlt, akkor a Git elindítja a KDiff-et. A példában ha ráállunk a master ágra és próbáljuk merge-ölni a FOMI_munkaanyagok ággal, akkor Merge Conflict-ot kapunk a menu.html fájlban. Erre rányomva bejön egy ablak, ami jelzi, hogy 3 konfliktus volt a fájl-ban, 2-t automatikusan megoldott a Git (mert ezekben a sorokban vagy csak az egyik fájlban történt módosítás, vagy mindkettőben ugyanaz lett a módosítás végeredménye), 1-el pedig nem tudta, hogy mit csináljon, mivel abban a sorban mindkét fájlban módosítás történt, így ezt nekünk kell kézzel megoldanunk.
Az ablakot leokézva a következő dián lévő KDiff3-as ablak jön be:

Merge conflicts 4

menu.html.BASE <-> menu.html.LOCAL <-> menu.html.REMOTE - KDiff3

Fájl Szerkesztés Könyvtár Mozgatás KDiff3 Összeolvasztás Ablak Beállítások Súgó

A (Base): ...it_kepzes_peldahonlapja\menu.html.BASE

B: C:\Git_kepzes_peldahonlapja\menu.html.LOCAL

C: C:\Git_kepzes_peldahonlapja\menu.html.REMOTE

Output: menu.html

Encoding for saving: Codec from C: UTF-8-BOM Line end style: DOS (A, B, C)

```
        <div class="sidenav" width = 15% align = "left">

            <h1>MENÜSOR</h1>
            <ul>
        <li><div class="sidenavActive">Git</div></li>
                <li><a href="main.html" target="mainframe">Git verziókezelés előadás </a></li>
            </ul>
            <ul>
<Összeolvasztási ütközés>
                <li><a href="FOMI_munkaanyagok\rastercatalog.html" target="mainframe">Raszterkatalógus felhasználói
            </ul>
    </div>
</BODY>
</HTML>
```

A fel nem oldott ütközések száma:

Merge Conflict – leírás 1

Az ablakban összesen 3 részablakot látunk, felül 3-at, lent 1-et.
Az ablakok jelentése a következő

- A – **Base** (bal felső): **A fájlnak az az utolsó commit-ált állapota, amelyik állapotban a personal repository-nkban és a central repository-ban / lokális merge-nél a másik personal repositorys águnkban még ugyanaz volt a fájl.**
- B – **Local** (bal középső): **A fájlnak personal repositorynkban jelen lévő, abban az ágban lévő változata, ahova merge-öltünk.**
- C – **Remote** (jobb fölső): **A fájlnak a central repository-ban / lokális merge-nél a másik ágban jelen lévő változata.**
- D – **Output** (alsó ablak): **Az éppen szerkesztendő új fájl, ami a mostani commit gráfpontunkhoz készül.**

Néha lehet olyan, hogy felül a BASE ablak hiányzik, például amikor a fájlt mindkét ágban az új ág kiágaztatása után hoztuk létre, csak más-más tartalommal (vagy két egymástól eddig független orphan branch-et merge-ölünk)
Azokban a sorokban, ahol csak az egyik ágban módosult a fájl, automatikusan az új változtatott sorokat teszi bele, sosem a base-t.

Merge Conflict – leírás 1

Amelyik sor mindkét ágban különbözik a BASE-től (és egymástól is), mindig ott vannak merge-ölési konfliktusok (a D ablakban <Merge conflict> -al jelezve a kérdéses sorokat). Ilyenkor szükséges merge-ölnünk, amelyhez szükségünk lesz **a KDiff eszköztár legalapvetőbb funkcióira (lásd 2 ábrával ezelőtt):**

Ez az ablak mindig akkor jön fel, ha vannak a fájlban merge conflictus-ok. A merge conflictus-ok mindig egymás után a megadott sorokban vannak. A KDiff-ben mindig egy megadott soron állva tudjuk kiválasztani az A,B,C gombokkal, hogy a sor melyik verzióját is tegyük bele az output fájlba (D ablak). Ha többet is be akarunk tenni, akkor többet is lenyomhatunk. Ekkor a lenyomás sorrendjében teszi be a sorokat az output fájlba. De természetesen kézzel is írhatunk bele.
Megjegyzés: A Git meglepően okosan merge-öl, például, ha az egyik fájlban ütöttünk egy Entert, akkor ezután a fájlsorok teljesen el fognak csúszni egymástól a két féjlváltozatban, hiába nincs utána módosítás. A Git mégis felismeri, hogy csak egy Entert ütöttünk le és elcsúsztatva vizsgálja tovább a fájlokat!

Összetett lokális műveletek (offline műveletek)

Working directory
(munkakönyvtár)

.git directory
(git könyvtár helyi tároló)

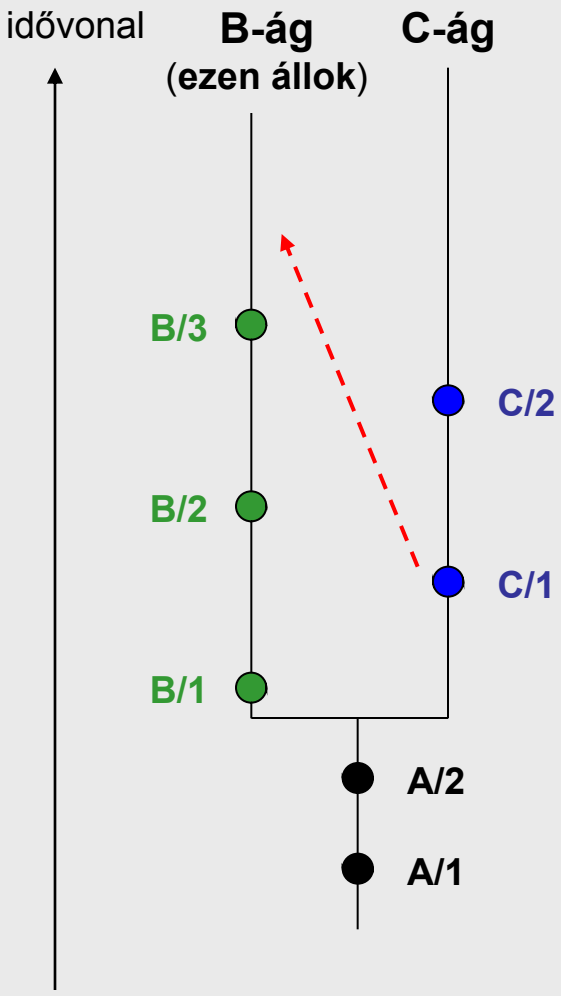
merge/rebase

merge commit miatt a .git könyvtárban a hash-ágszerkezet is változik + annak az ágcímkéje, ahova merge-ölök

Fast forward esetén a .git könyvtárban csak az ágcímke mutatója változik, ahova merge-ölök

Rebase esetén a hash-ágszerkezet is változik és így annak az ágcímkéje is, ahova merge-ölök

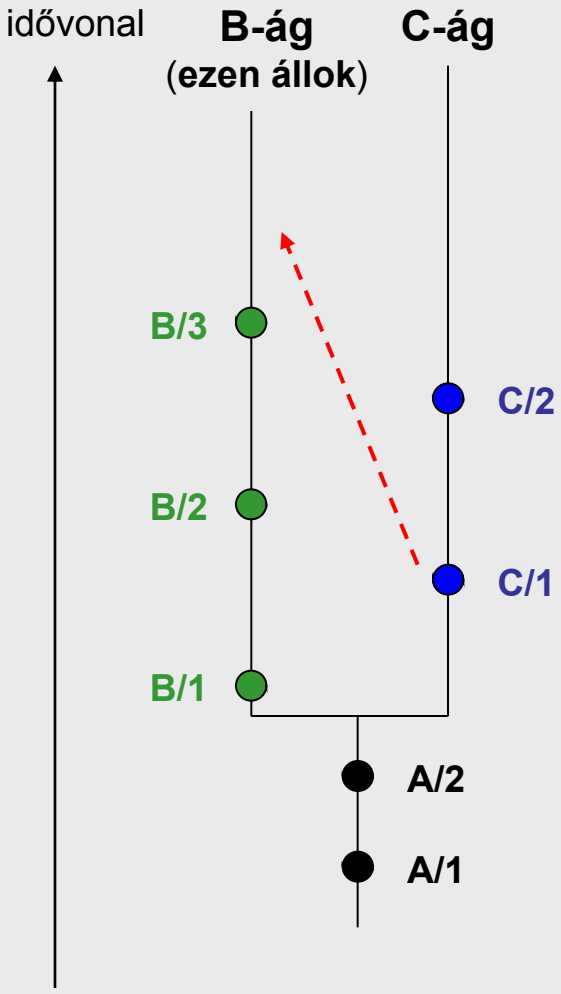
Cherry Pick Commit **



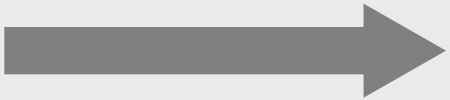
A cherry pick commit egy olyan, speciális commit művelet, amely során létrejött új revision tartalma (és opcionálisan a commit message is) a művelet elején egy általunk kiválasztott revision tartalmával és opcionálisan commit message-ével lesz megegyező (a commit hash és a parents és childern revision-ök viszont nyilván mások lesznek). Lényegében egy revision tartalmának másolata (klónja), a hash-fa egy másik pontjára.

A branch-ek szerepe diasorozatban leírt ArcMap 9.3 és ArcMap 10.0 verziók közti különbségekből fakadó bonyodalmak a Cherry Pick műveletére szintén jó példát adnak. Mivel az ArcMap 9.3 és 10.0 nem voltak egymással kompatibilisek és az ArcMap 10.0-hoz egy teljesen új osztálygyűjteményt adtak ki, így nekünk is teljesen át kellett írni a kódot. A sok hibalehetőség miatt viszont kezdetben csak néhány ügyintézőhöz mertük feltenni az új verziót. Mivel ily módon pár hónapig párhuzamosan működött élesben mindkét verzió, ezért néhány kisebb modult mindkét ágban le kellett volna fejlesztenünk. Ezekben az esetekben a cherry pick commit művelete nagyon hasznosnak bizonyult.

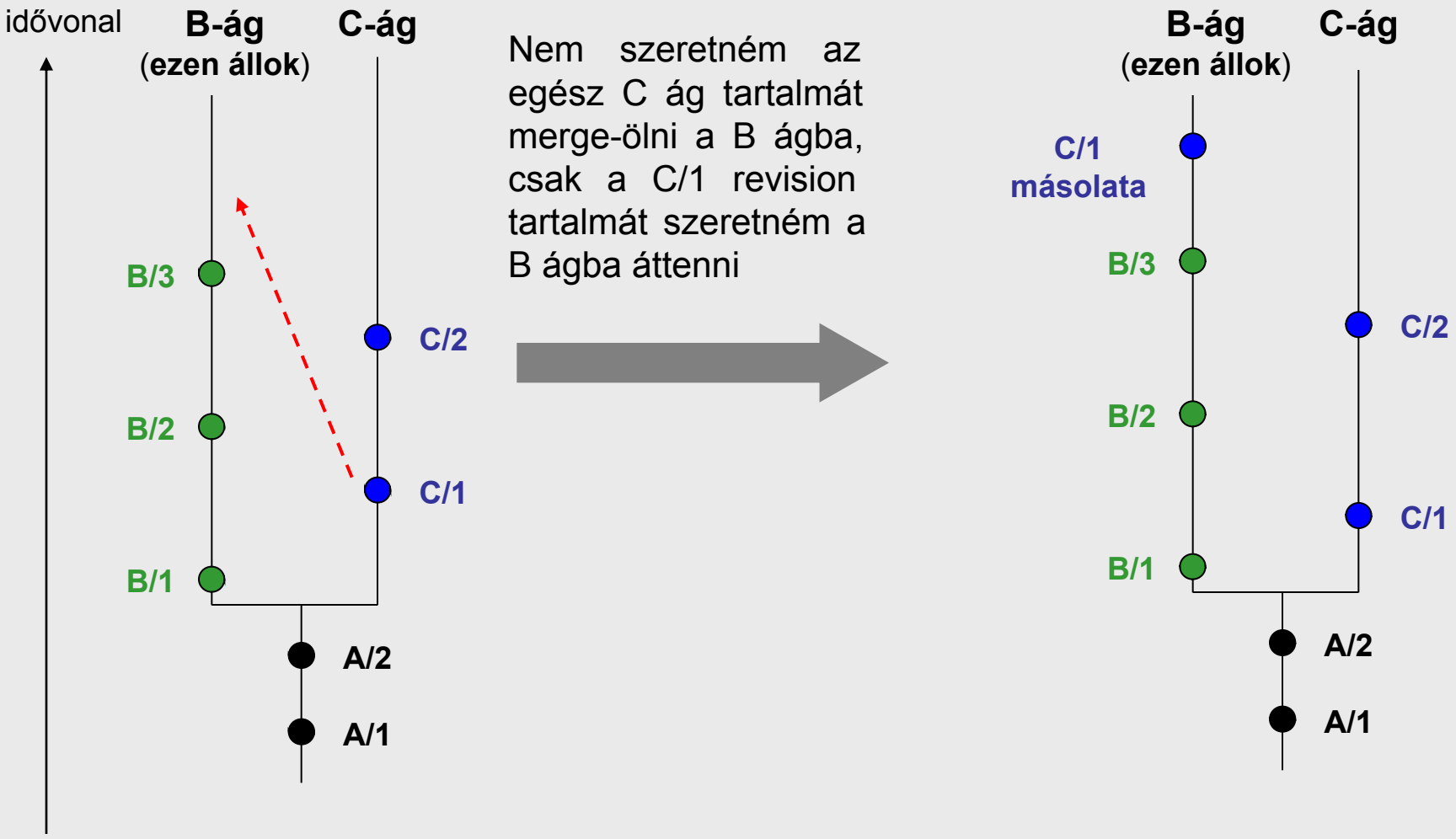
Cherry Pick Commit **



Nem szeretném az egész C ág tartalmát merge-ölni a B ágba, csak a C/1 revision tartalmát szeretném a B ágba áttenni

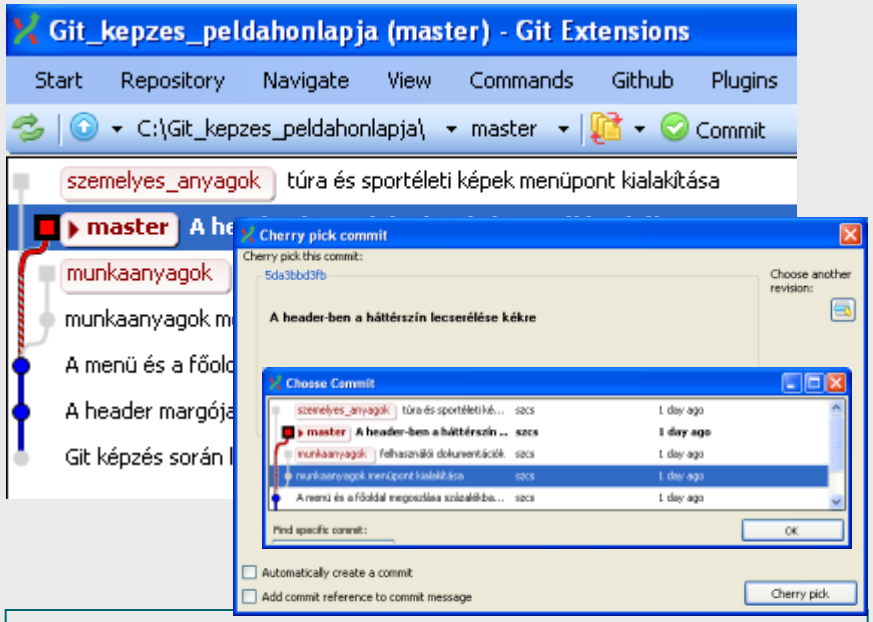


Cherry Pick Commit **



Cherry Pick Commit **

- Revision átmásolása egy másik ágba
- Álljunk rá arra a revision-re (gráfpontra), amelyre egy új gyerek-revision-ként át szeretnénk vezetni egy commit-ot, majd jobb gomb -> cherry pick / Commands->Cherry Pick
- A megjelenő ablak szerkezete:
  - **Choose Another Revision** : a cherry pick-elendő revision kiválasztása az újonnan bejövő ablakban
  - **Cherry Pick this commit**-nál látjuk annak a commit-nak (gráfpontnak) a sorozatszámát, amelyiket cherry pick-elni akarunk.
  - **Automatically create commit**: a commitálást is elvégzi helyettünk (létre is hozza a klónozott gráfpontot)
  - **Add Commit Reference to commit message**: a commit üzenetbe automatikusan beteszi, hogy cherry pick történt, és hogy ez melyik ágból következett be.
  - Megjegyzés: merge gráfpontot nem lehet cherry pick-elni



Parancssorosan: álljunk rá arra az ágra, ahova a revision-t be akarjuk szúrni, majd:

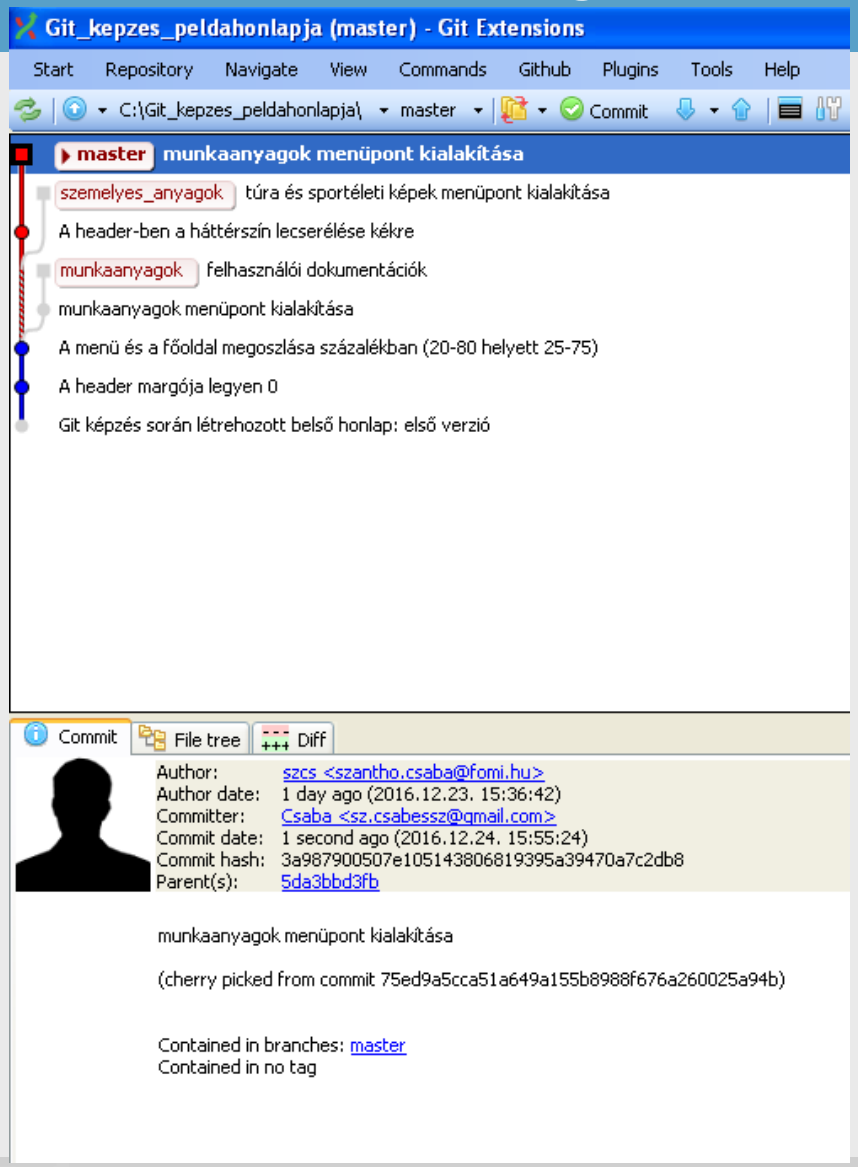
cherry-pick 75ed9…

vagy

cherry-pick –no-commit –x 75ed9…

(az x a commit reference benyomását jelenti

Cherry Pick Commit **



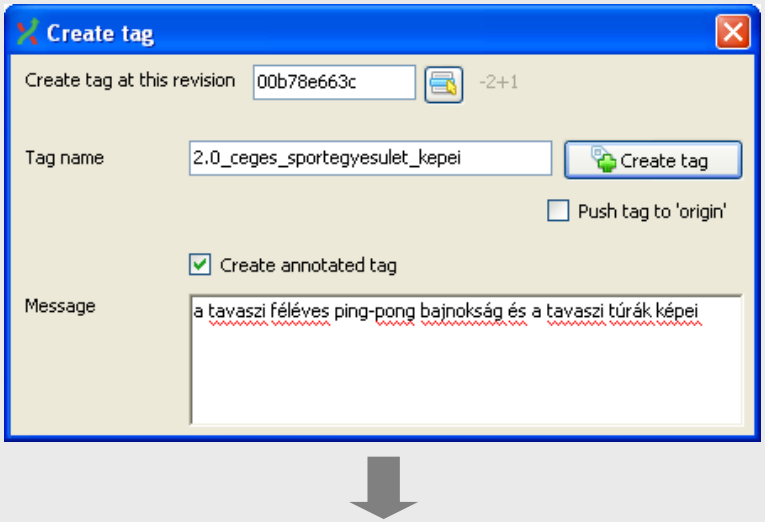
Cherry pick után (miután a „ munkaanyagok menüpont kialakítása” commi t message-el ellátott revision -t átmásol tuk a munkaanyagok ágból a szemelyes_anyagok ágba:

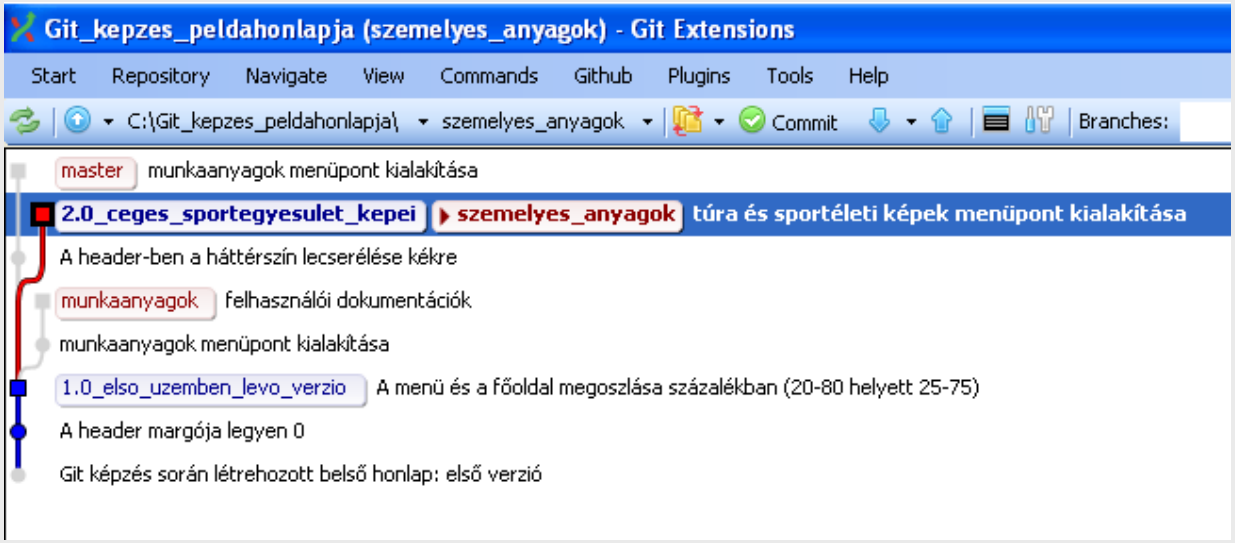
(alul látható, hogy mivel benyomtuk a commit reference message-et, hozzáírta a commit szövegéhez, hogy: cherry pick from commit…)

Cherry pick commit esetén az **author**, annak az eredeti revision-nek az author-a , amelyikkel a cherry pick commit műveletét végeztük, a **committer** személye pedig mi vagyunk.

Create/Delete Tag **

- **<u>Tag-ek</u> <u>(információs</u> <u>címkék)</u> : Ezekkel tudok egyéb információt megjeleníteni bizonyos revision-ökre.**
- Ráállok egy tetszőleges revision-re , jobb gomb és create/delete tag
- Ezeket a címkéket is tudom pusholni (push tag to origin opció, ha szeretném)/ pullolni , módosítani.





Jelmagyarázat a következő tag-es diapéldasorhoz **

2.0: kiadott verzio

→

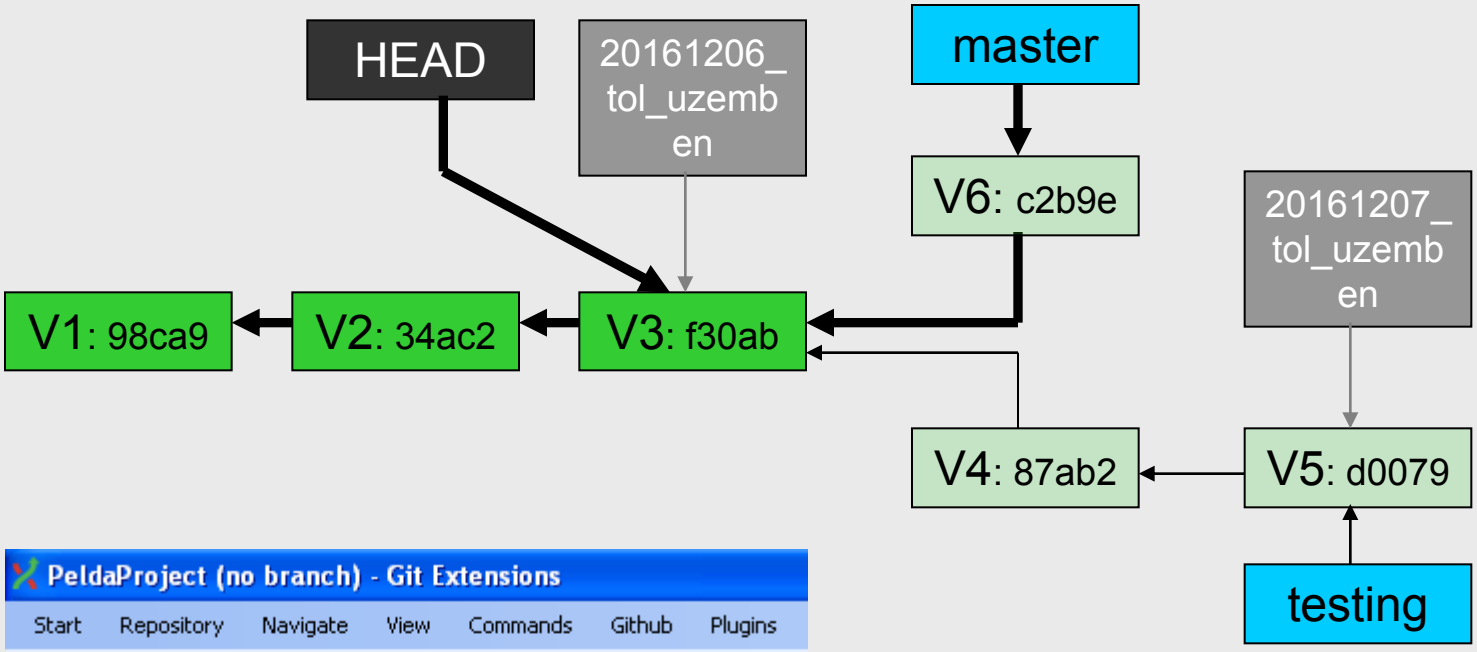
Tag-ek (információs címkék):

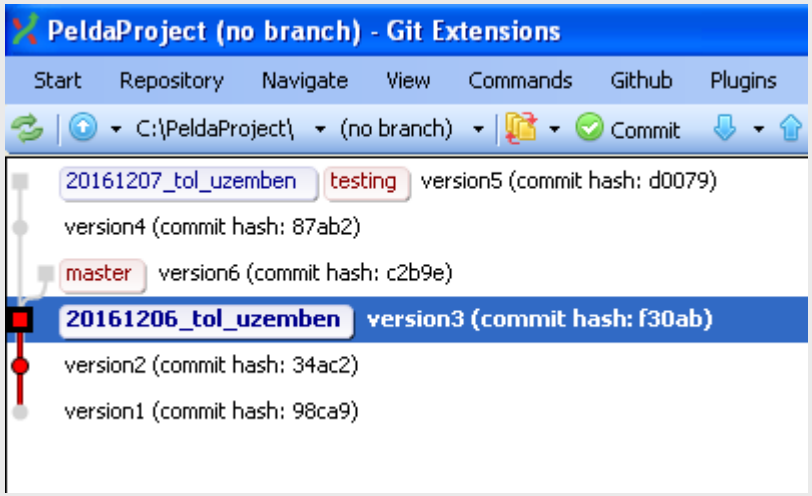
Ezek a szürke téglalapban lévő olyan címkék, amelyek számunkra szolgáltatnak plusz információt az adott revision-ről (például, hogy egy bizonyos dátumtól kezdve ki lett adva és élesben használják a felhasználók).

Egy tag létrejöttekor meg kell adni, hogy melyik revision-höz tapasztjuk. Akár úgy van beállítva, hogy push-oláskor a tag is menjen föl, akár úgy, hogy ne, a tag létrejöttétől kezdve a revision hash-kódjához van ragasztva, egészen addig, amíg vagy a revision, vagy a delete tag-gel csak maga a tag meg nem szűnik. **A mutatója soha nem állítódhat át**, ezért is van szürke nyilakkal jelölve. **A HEAD címke mutatóját pedig soha nem tudjuk ráállítani az ilyen címkékre**

A többi jelölést lásd a lokális műveletek diapéldasor jelmagyarázatánál!

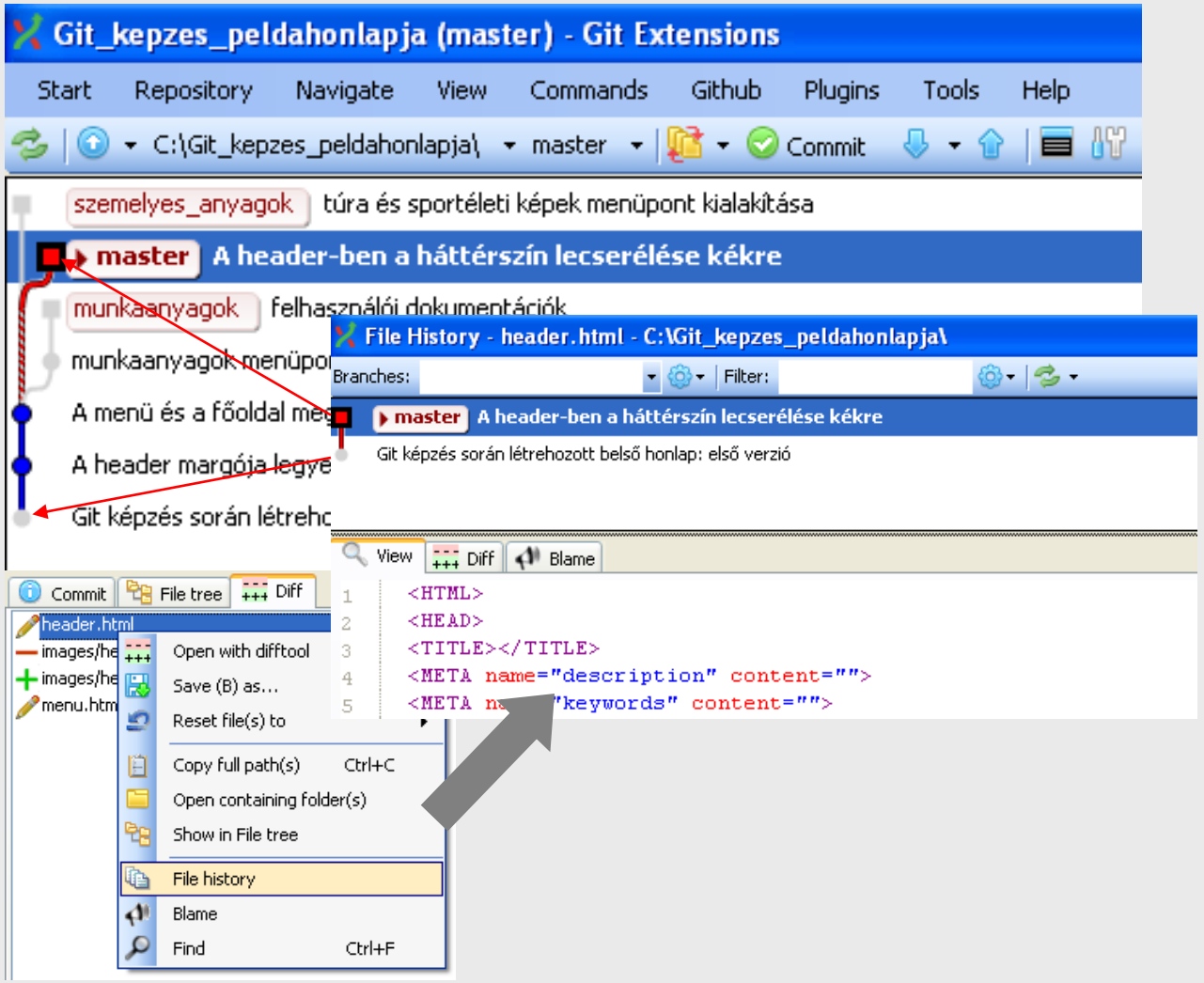
Lokális műveletek – példasor
Create tag **





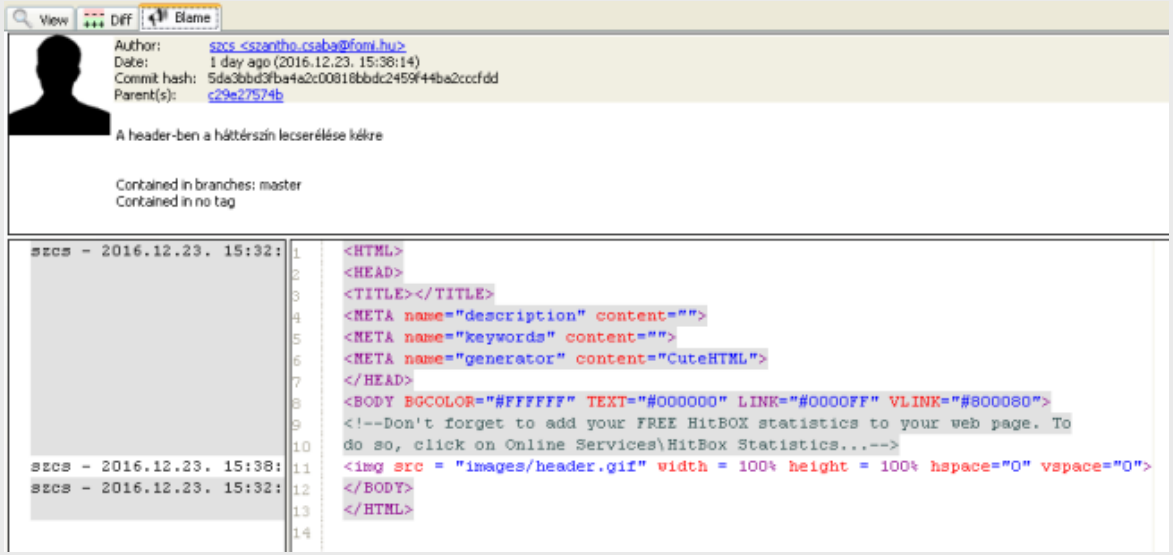
Egyéb - File history **

- Ha csak egy fájl „életútja érdekel, a File Tree-ben nyomjuk meg a jobb gombot, majd a file history pontot. Ekkor külön ablakban megmutatja a file életútját – lásd az ábrákat
- A Find-dal file-okat tudunk a file tree-ben megkeresni stb. (érdemes végigpróbálgatni a lehetőségeket)
- Ha egy revision állapota érdekel: working tree fül
- Ezen kívül van lehetőségünk szűrni különböző dolgokra: ágnévre, commit szerzőjére, commit message-re (csak részletet is be tudok írni), remote/local ágakra, stb.

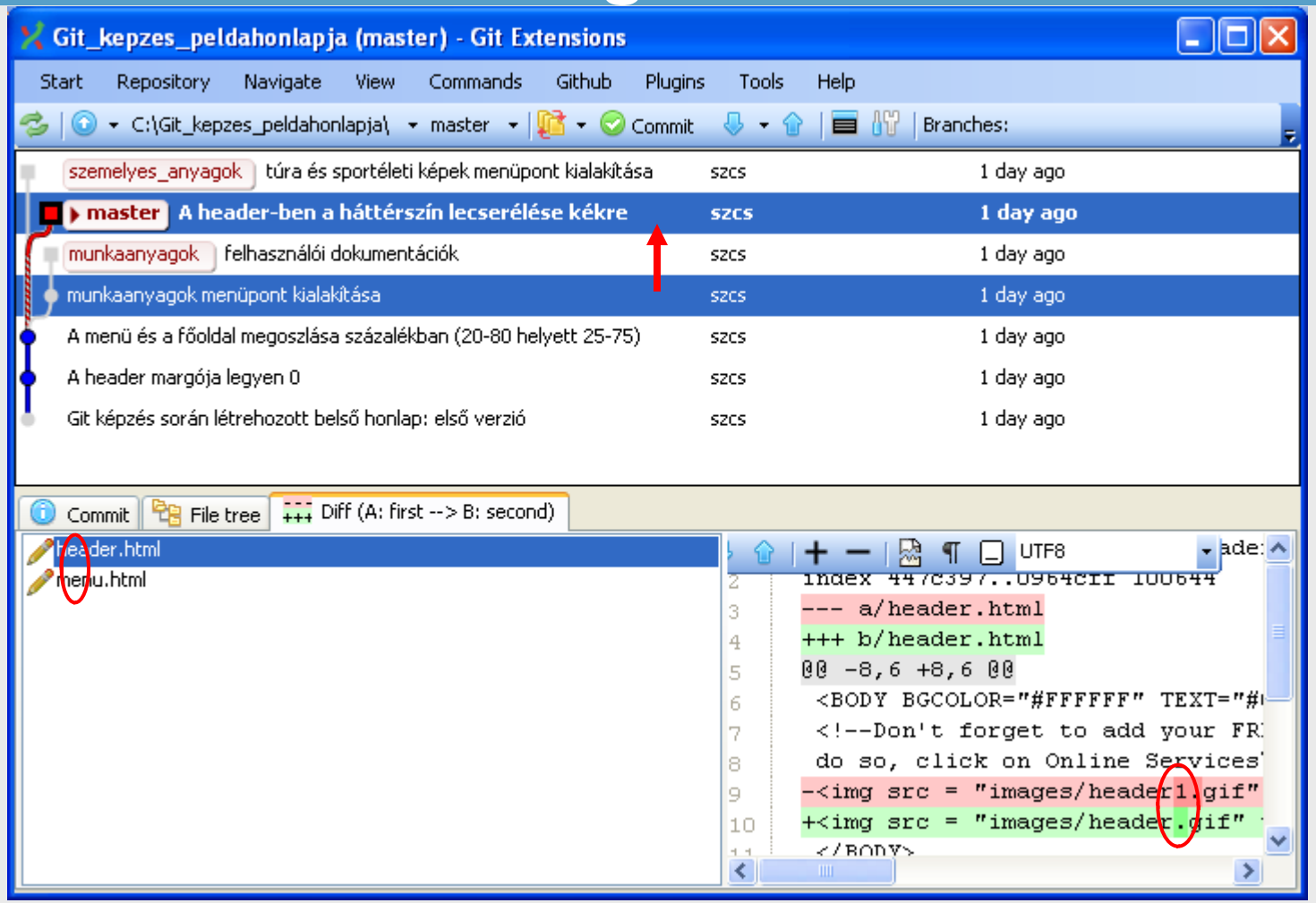


Egyéb – File history: Blame **

- A blame opcióval pedig egyben meg tudjuk nézni, hogy a fájl melyik sorát mikor és ki módosította.

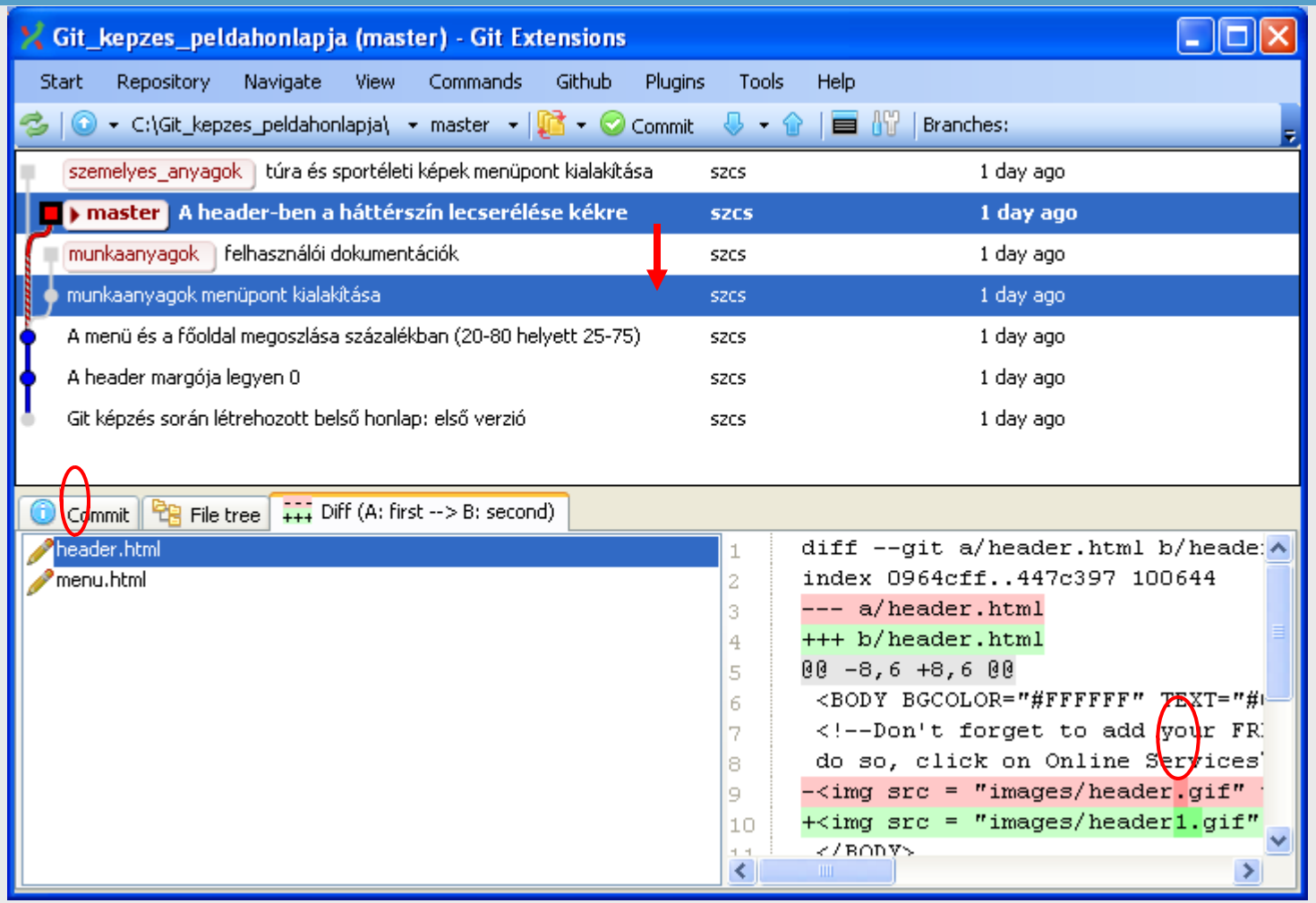


Egyéb – különbségek vizsgálata **



Ha két különböző helyen lévő branch-et kijelölök, akkor azoknak a különbségeit mutatja (nem mindegy a kijelölés sorrendje)

Egyéb – különbségek vizsgálata **

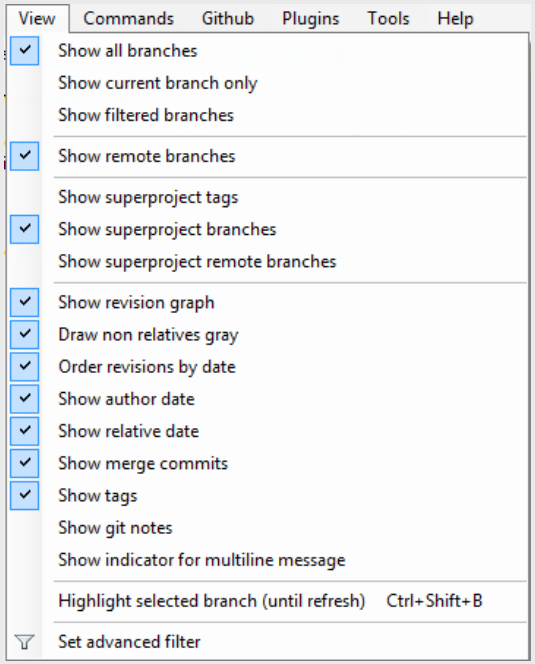


Ha két különböző helyen lévő branch-et kijelölök, akkor azoknak a különbségeit mutatja (nem mindegy a kijelölés sorrendje)

Egyéb
View menüpont + statisztika **

View menüpont:

Ebben a menüpontban beállíthatjuk a repository-hoz tartózó címkék, részfák, stb. láthatóságát (összes branch-et lássa, csak egy aktuálisat, az ágakat ne mutassa, csak a revision-öket, a remote ágakat ne mutassa, minden ágat színesen mutasson, stb.)



Statisztikák:

Érdemes még a Tools-ban megnézni a Commits/user-t, vagy a commit log-okat

(nem feltétlenül az dolgozik a legtöbbet, aki a legtöbbet kommitál, néha egy-egy commit egy kisebb hiba módosítását jelenti, máskor pedig több napi munkát is jelenthet. Érdemes minél többször commit-álni , és pull-olni , mert kisebb valószínűséggel fogunk merge konfliktus-ba ütközni!

GIT VERZIÓKEZELÉS
III.
Műveletek a lokális és a távoli adatbázis között (csoportos használatra)

Clone repository

Klónozás : Egy remote repository teljes tartalmának, vagy egy ágának/ branch-ének (lásd később) másolása.

Előfeltétel:

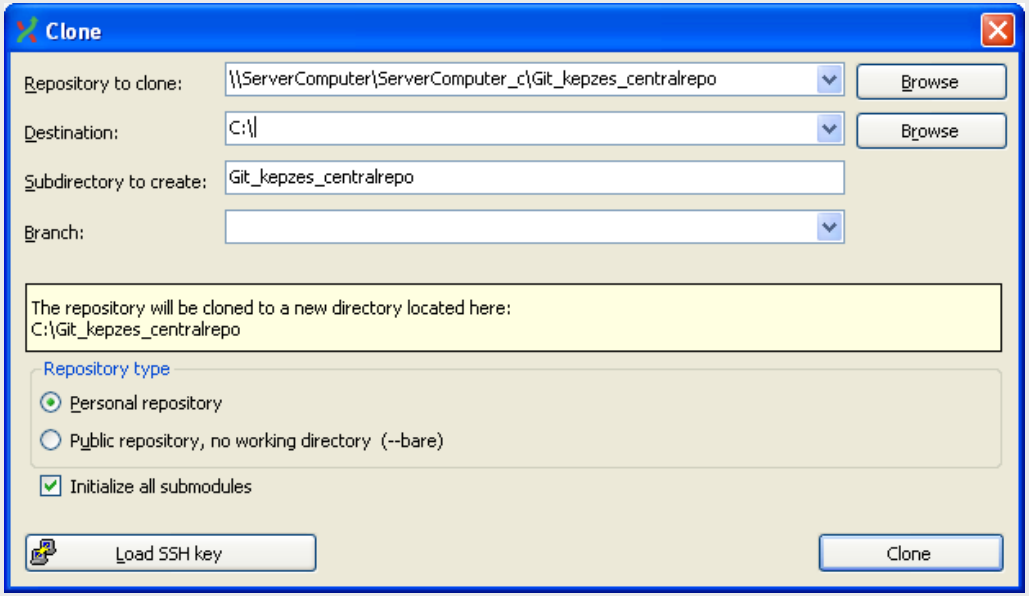
- A klónozandó könyvtár ne legyen repozva

Ablakmegnyitás:

- Indítsuk el a GitExtension-t, majd GitEx Clone…

Az ablakban a főbb funkciók jelentése:

- **Repository to clone** : melyik repository-t akarjuk klónozni.
- **Destination:** hova akarunk klónozni
- **Subdirectory to create:** az új könyvtár neve
- **branch** -nél tudjuk kiválasztani, hogy melyik ágat akarjuk klónozni (ilyenkor csak az ahhoz azághoz tartozó adatok fognak klónozódni)
- **A repository type** : új repository központi, vagy personal , azaz lokális legyen-e, ahova dolgozni szeretnénk.



Parancssorosan:

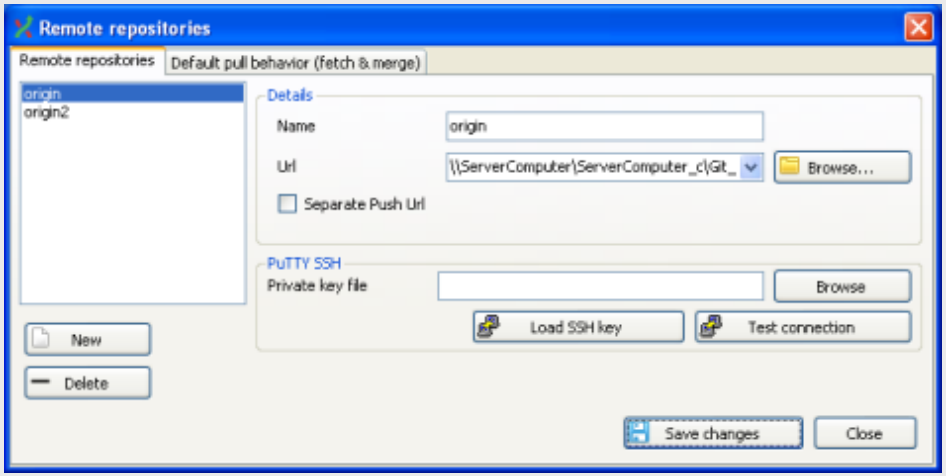
```
git clone /c/RemoteGitPeldaHonlap
/c/GitPeldaHonlap –bare –shared=all
```

Manage Repositories

Menüsor: Repositories->Remote Repositories

A central repository-knak itt tudunk címkéket beállítani, hogyha push-olni / pull-olni szeretnénk, ott elég legyen csak a Name-ben megadott névre hivatkozni az elérési út helyett. Olyan, mint például a mount-olás.

Itt tudjuk megadni továbbá, hogyha nem adunk meg ágakat a pullnál / merge-nél / fetch-nél , akkor alapesetben honnan hova tudjunk pullolni/merge-ölni/fetch-elni. (Default pull behavior fül)



Ablak szerkezete:

Remote repositories fül (Central repository-k megadása és címkézése)

- Bal oldali ablak: eddigi central repository-beállítások címkenevei. Amelyikre rákattintunk, annak az Url-jét automatikusan behozza jobb oldalt, amit meg is tudunk változtatni.
- Name: a címke neve
- Url: címke elérési útja
- New: ha új repository-t szeretnénk (ekkor fent új <new> oszlop és rá is áll).
- Delete : az ablakban éppen kijelölt repo törlése (rákérdezés után), Eztán bezárva az ablakot a fában is eltűnik a törölt central repository összes ága.
- Save Changes-re az új (módosított) címke elmentése (az elérési út validitását itt nem figyeli!!!)

Default pull behaviour fül:

- Ha push / pull műveleténél távoli central repo-nak nincs kiválasztva semmi, itt adjuk meg, mi legyen a default (általában vagy a neki megfelelő párja, vagy a master szokott lenni)

Pull

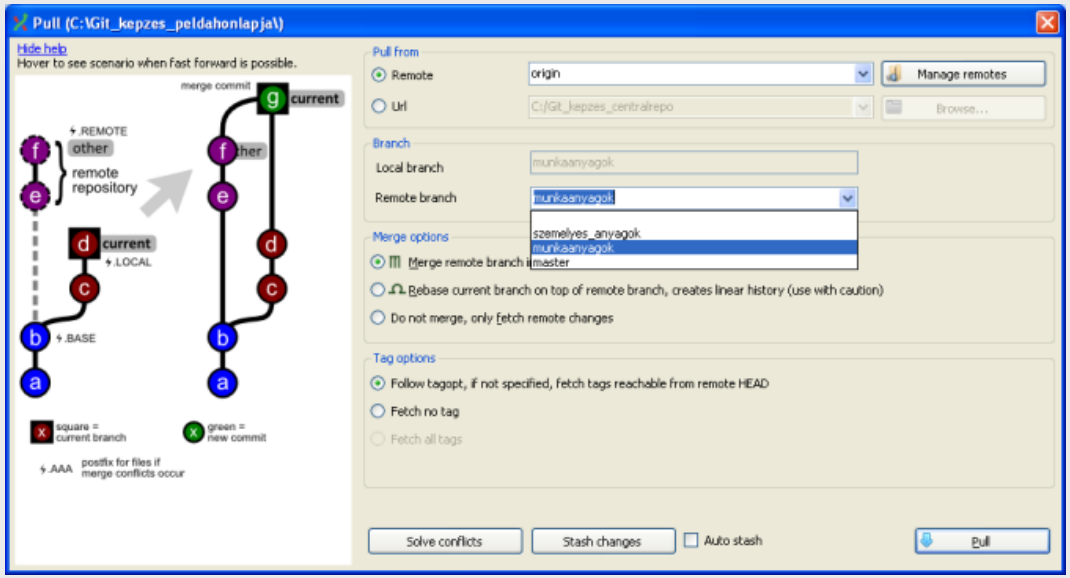
A Pull művelete: a megfelelő central repository-ból a nekünk szükséges ágak mások által írt módosításainak lehúzása a saját reponkba és adott esetben merge-ölése/rebase-elése a saját águnk hash-fájába

Jobb megértéshez lásd a fetch és push hatása a faszerkezetre diát.

Commands->Pull vs eszköztáron a megfelelő pull művelet

Az ablak szerkezete:

- **Pull from:**
  annak a central repository címkéjének kiválasztása, hogy melyik központi repository-ból akarjuk leszedni a változtatásokat. (lásd Manage Repositories)
- **Local branch**:
  az a saját personal repo-s ág, amelyiket frissíteni akarjuk
- **Remote Branch**:
  az central repository ág, amelyikből szeretnénk az összes változást lehúzni a local branch-ünkbe.



Parancssorosan:

git pull –rebase –progress munkaanyagok munkaanyagok

vagy

git pull –merge –progress munkaanyagok munkaanyagok

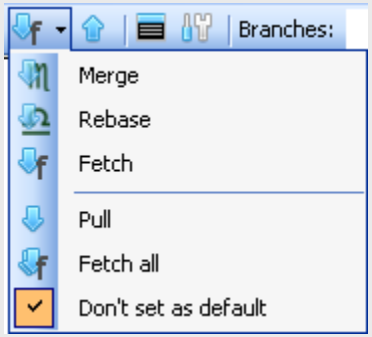
Pull – Pullolás típusai 1

Pull: Fetch:

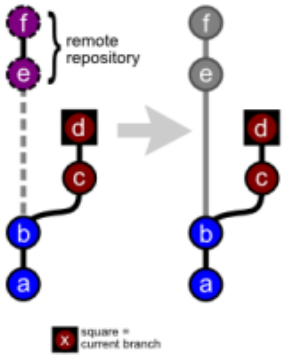
Csak leszedi a módosításokat (ábrán szürke revision-ök), de nem teszi bele a mi módosításainkba, azt nekünk utólag kell beletennünk.

Pull: Fetch all

nem csak az akuális , hanem az összes remote ághoz tartozó módosításokat lefetch-eli (és átcímkézi rájuk a remote (zöld színű, lásd később távoli ágcímkék diát) ágakat.



Pull típusai menüből



fetch

Pull – Pullolás típusai 2

Pull: Fetch + Merge:

Leszedi a módosításokat központi repo-ról , majd automatikusan összemerge-öli a mi általunk készített módosításokkal (pontosan úgy, ahogy azt a merge műveleténél láttuk, annyi megszorítással, hogy mindig egy távoli ágat merge-öl egy kijelölt lokális ágba). Tulajdonképpen ez a funkció jelenti a hagyományos pull-t.

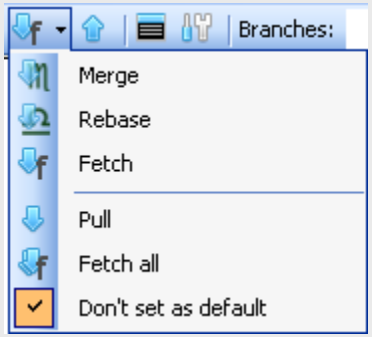
Pull: Fetch + Rebase:

ugyanaz, mint a fetch + merge , de a végeredményben szépen összefésüli az általunk készített és mások által tett módosítások commit-jait időrendi sorrendben egyetlen ágba (lásd rebase-nél). (és így megspórolunk vele egy merge commit-ot)

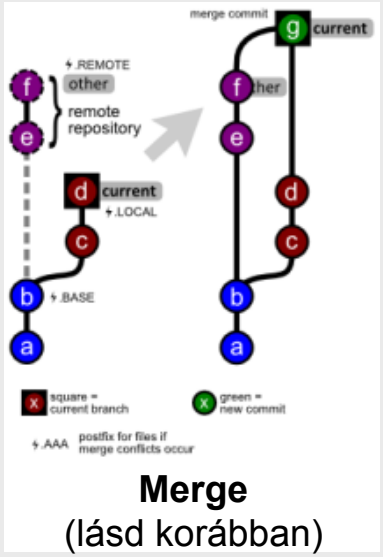
Megjegyzések:

Fetch + merge, vagy Fetch + Rebase opció mindig csak egyetlen ágból egyetlen ágba, vagy revision-ba (ahova a HEAD mutat) történhet. Fetch-elni viszont a central repo összes revision-jét tudjuk a Fetch All művelettel.

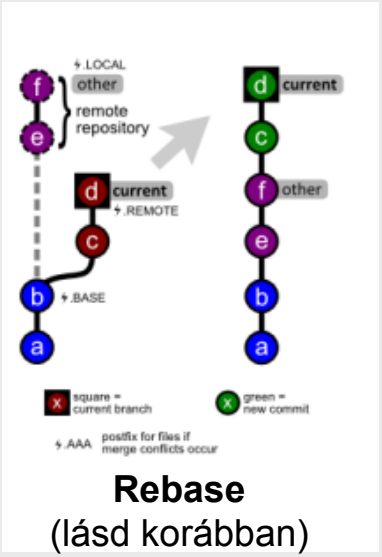
A Merge és Rebase itt ugyanazt jelentik, mintha két lokális ágat merge-ölnék , azzal a megszorítással, hogy itt a merge-ölendő / rebase-elendő ág csak a távoli ágcímke lehet (pl.: origin \ master , stb. lásd később az online műveletek példasornál)



Pull típusai menüből



Merge
(lásd korábban)



Rebase
(lásd korábban)

Push

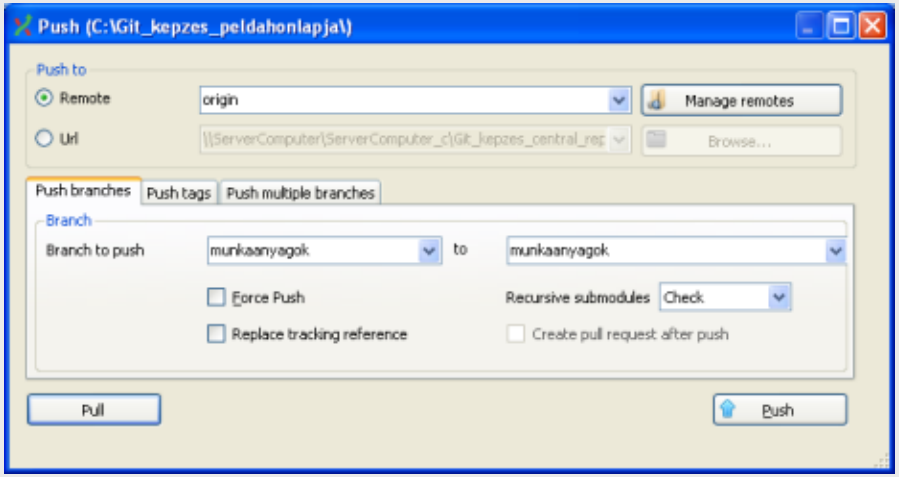
Commands->Push (vagy eszköztár)

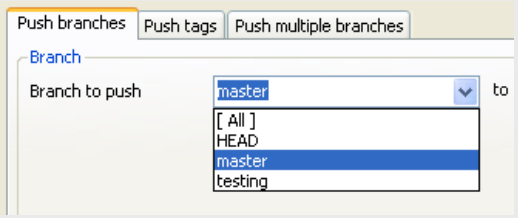
Itt mi tesszük fel a módosításainkat a central repository-ba. Jobb megértéshez lásd a fetch és push hatása a faszerkezetre diát.

Addig nem enged pusholni , amíg a central repository megfelelő ágának legfrissebb verziói nincsenek pullolva

Az ablak szerkezete:

- A **Push to groupbox**: lásd Pull ablaknál a Pull from groupbox
- **Push branches** fül: a **branch groupbox** funkciói ugyanazok, mint a pull branch groupbox funkciói
- **Push tags**: - Tag groupbox: itt tudjuk megadni, melyik tag-et akarjuk pusholni (ha a tag címkét is push-oljuk, akkor a central repo-n is pontosan ugyanazon SHA1-kóddal ellátott revision-re fog mutatni)
- A **Pull** gombra nyomva a Pull ablak jön be (lásd korábban)
- A Push gombra nyomva megtörténik a pusholás és frissül a repo-fa címkézése is (lásd következő dia)





Ágak között felajánlja a HEAD-et és All-t is.

Parancssorosan:

```
git push –progress munkaanyagok munkaanyagok
```

Push

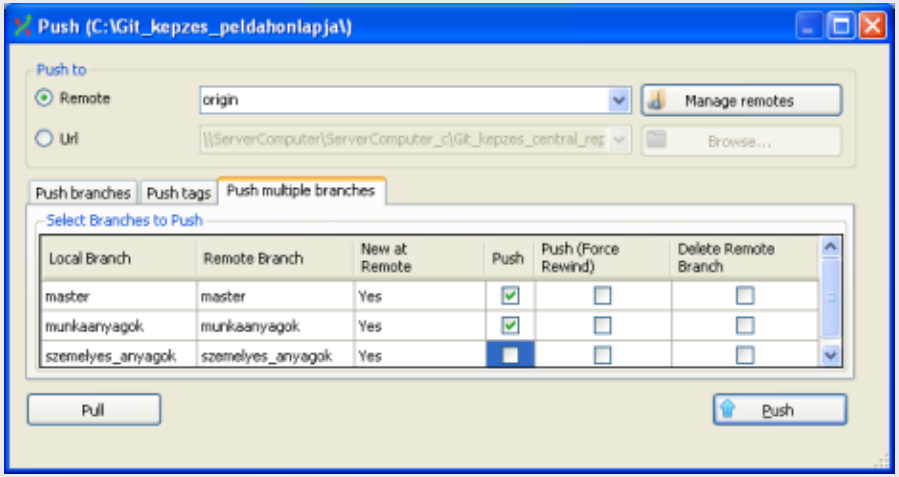
Commands->Push (vagy eszköztár)

Itt mi tesszük fel a módosításainkat a central repository-ba. Jobb megértéshez lásd a fetch és push hatása a faszerkezetre diát.

Addig nem enged pusholni , amíg a central repository megfelelő ágának legfrissebb verziói nincsenek pullolva

Az ablak szerkezete:

A **Push multiple branches fülön** a Select Branches to Push groupbox-ban egyszerre tudunk több ágat push-olni, de itt több ágat is tudunk egyszerre törölni

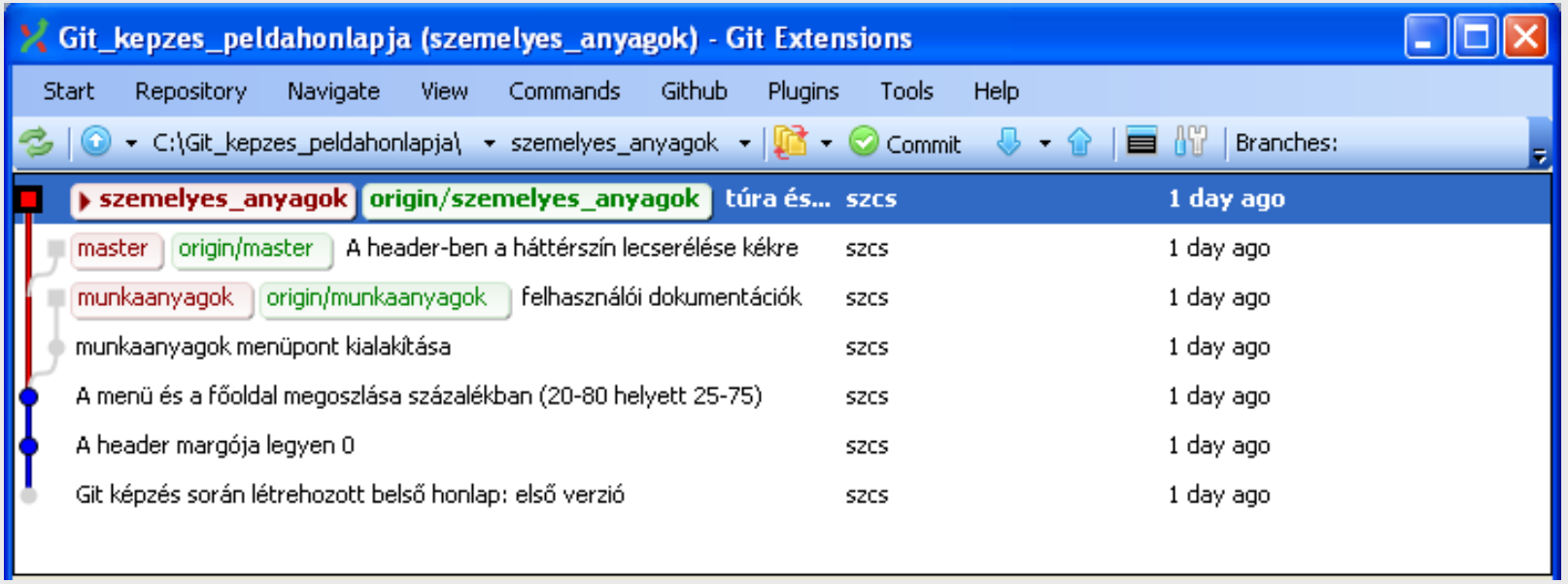


Parancssorosan:

```
git push –progress munkaanyagok munkaanyagok
```

Origin ágcímkék

A push és pull műveleténél az úgynevezett origin ágcímkék (GitExtension-ben zöld címkék) mutatója mindig szinkronizálódik.
A per előtti rész jelöli a távoli szerver Manage Remotes fülön megadott címét, ami olyan, mint egy mount-olás , hogy ne az Url-t , vagy elérési utat kelljen odaírni. Általában egyetlen távoli ágba szoktunk pusholni és onnan pullolni , amit konvencionálisan origin-nal szoktuk elnevezni (pl.: origin/FOMI_szemelyes_anyagok).
Törölni úgy tudom őket, hogy a Manage Remote Repository fülön törlöm a hivatkozást.



Git példahonlap

Jelmagyarázat a következő diapéldasorhoz

origin/
master

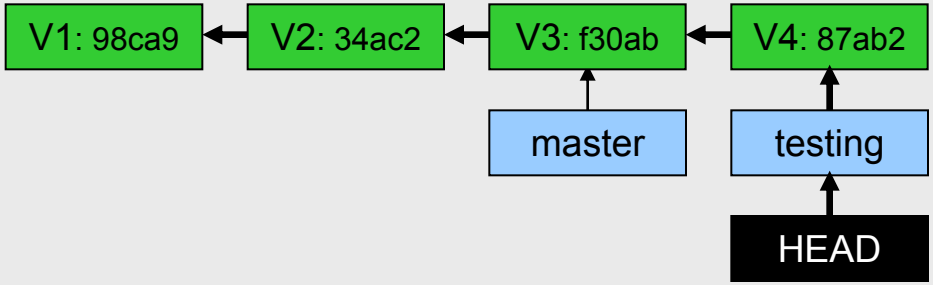
→

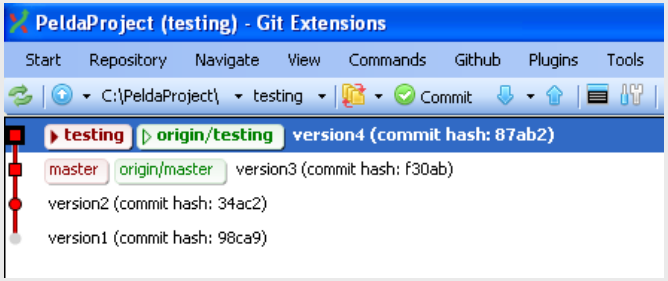
- Halványkék téglalapban vannak azok a branch-címkék , amelyek a remote / central repository ágait jelölik a helyi fában. Ezeknek az ágcímkéknek a **mutatóit** azért lettek más színnel, konkrétan kékkel bejelölve, mert **csak a push és a pull (és ezen belül is csak a fetch) résznél állítódnak másik revision-re! A HEAD címke mutatóját pedig soha nem tudjuk ráállítani az ilyen címkékre**. Ilyenkor mindig felajánlja, hogy a neki megfeleltetett lokális címkére (jelen esetben master) álljunk, rá, ha nincs, hozzuk létre, vagy csak álljunk a revision-re, stb.
- A többit lásd a lokális műveletek diapéldasorának jelmagyarázatánál!

Online műveletek – példasor
Alapállapot

Lokális repo:

Induljunk ki a lokális műveletek példasorának 6. példájából, a korábbi Commit 6 állapotból!

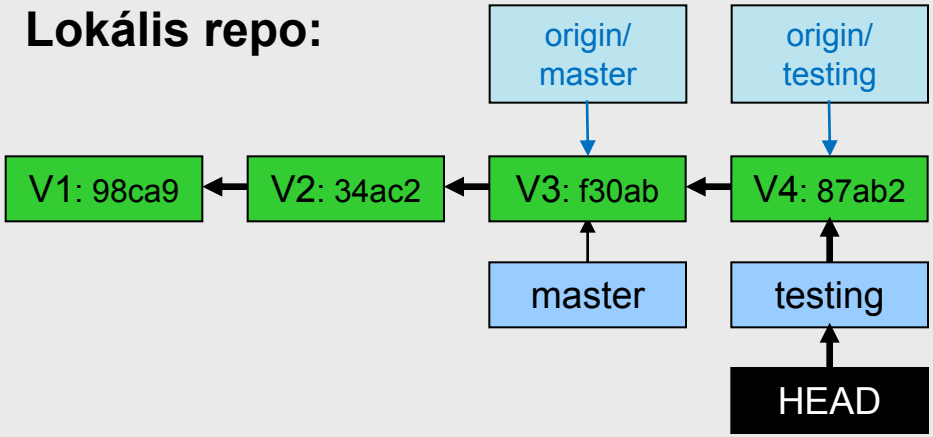




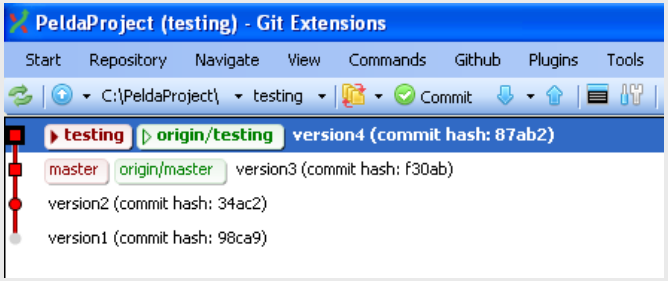
Central repo:
üres

Online műveletek – példasor
Push multiple branches

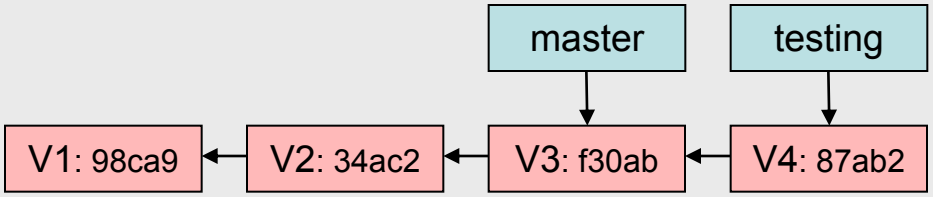
Lokális repo:



origin/master és origin/testing ágcímkék jelölik a lokális repo-ban, hogy a központi repo-ban mire mutat a master és testing ágcímke!

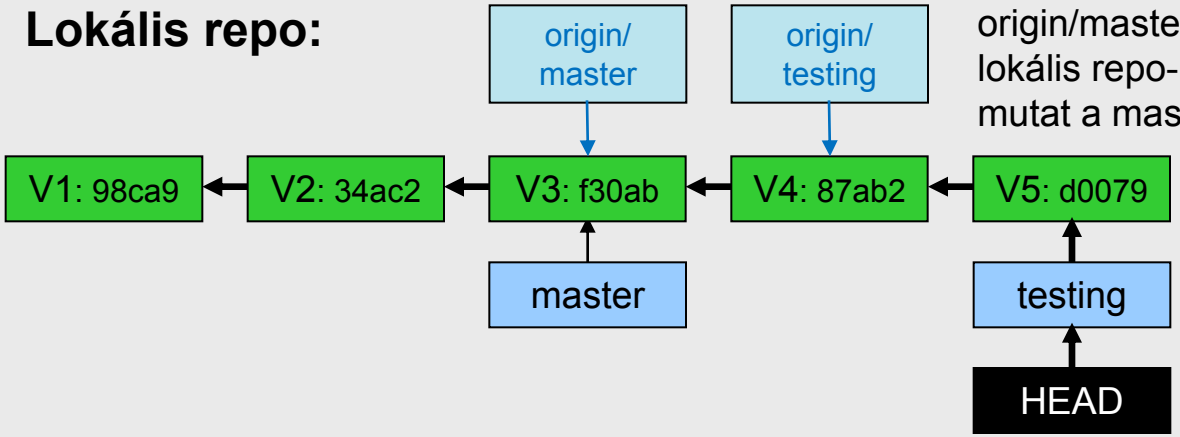


Central repo (central repo-ban nincs HEAD!)



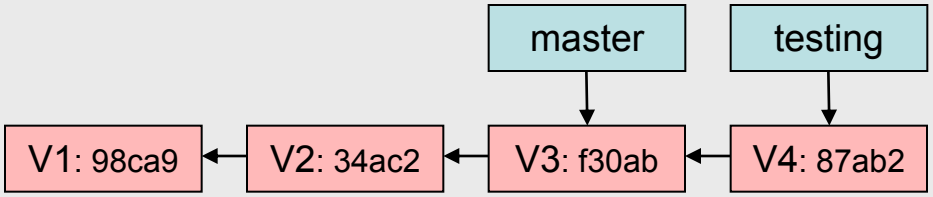
Online műveletek - példasor
V5 revision készítése (commit)

Lokális repo:



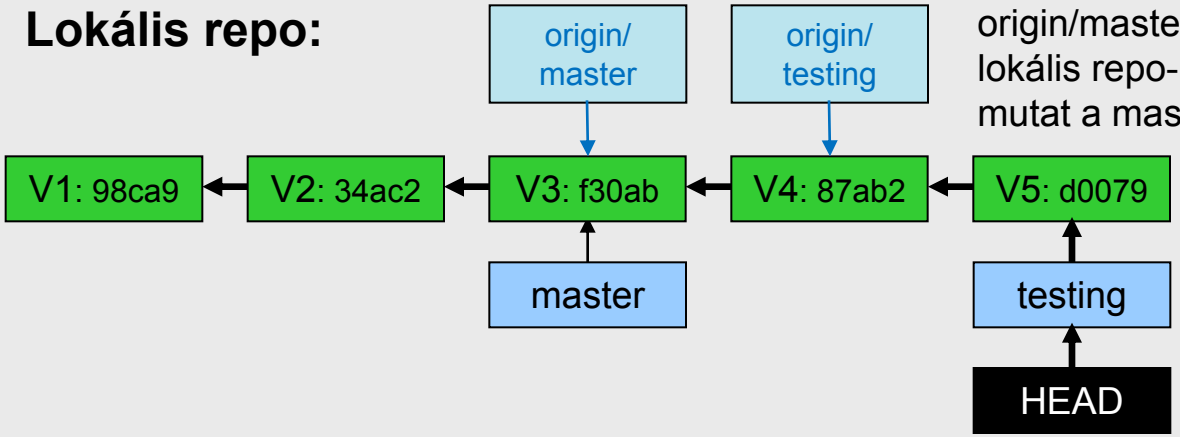
origin/master és origin/testing ágcímkék jelölik a lokális repo-ban, hogy a központi repo-ban mire mutat a master és testing ágcímke!

Central repo:

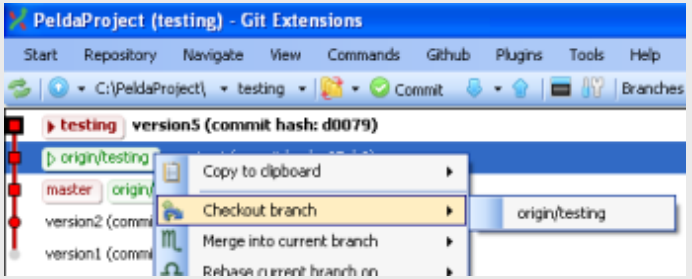


Online műveletek - példasor
Checkout remote branch kérdése

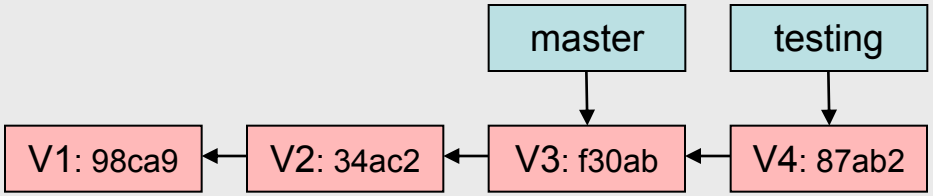
Lokális repo:



origin/master és origin/testing ágcímkék jelölik a lokális repo-ban, hogy a központi repo-ban mire mutat a master és testing ágcímke!

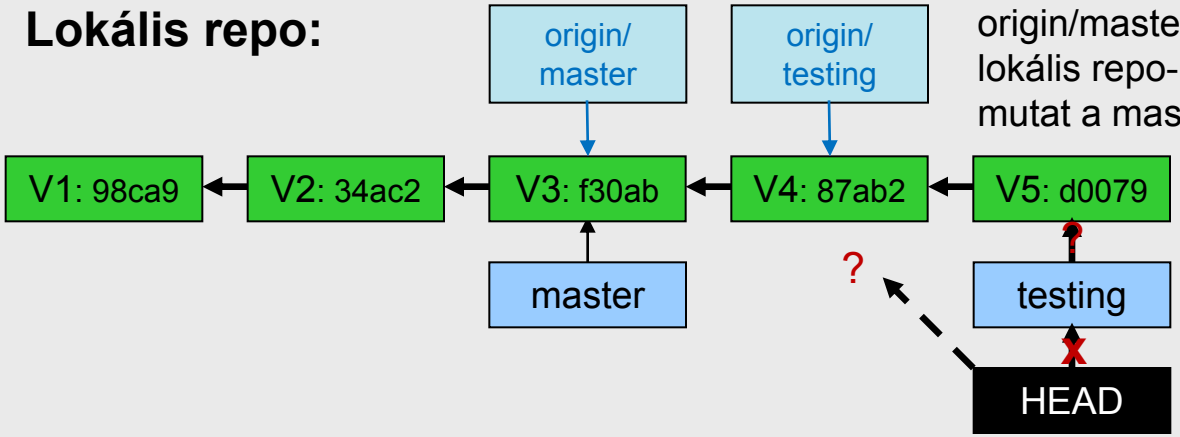


Central repo:

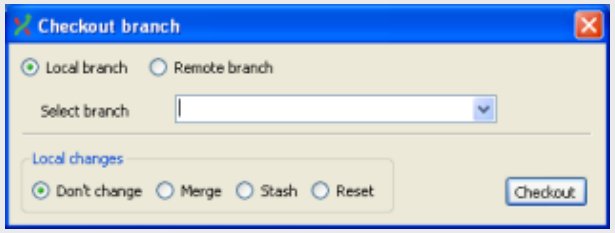


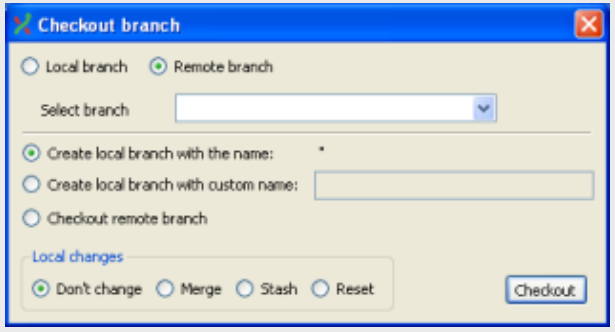
Online műveletek - példasor
Checkout remote branch kérdése

Lokális repo:



origin/master és origin/testing ágcímkék jelölik a lokális repo-ban, hogy a központi repo-ban mire mutat a master és testing ágcímke!





Sem checkout, sem reset, sem merge, rebase, fetch, pull, push, commit, stb., sem semmilyen művelet hatására nem fog a HEAD címke az origin/master vagy origing/testing címkére mutatni!

Az adott példában például a HEAD-el nem tudok az origin /testing-re checkout-olni. Ilyenkor a bal oldali ablak(ok) jönnek be.

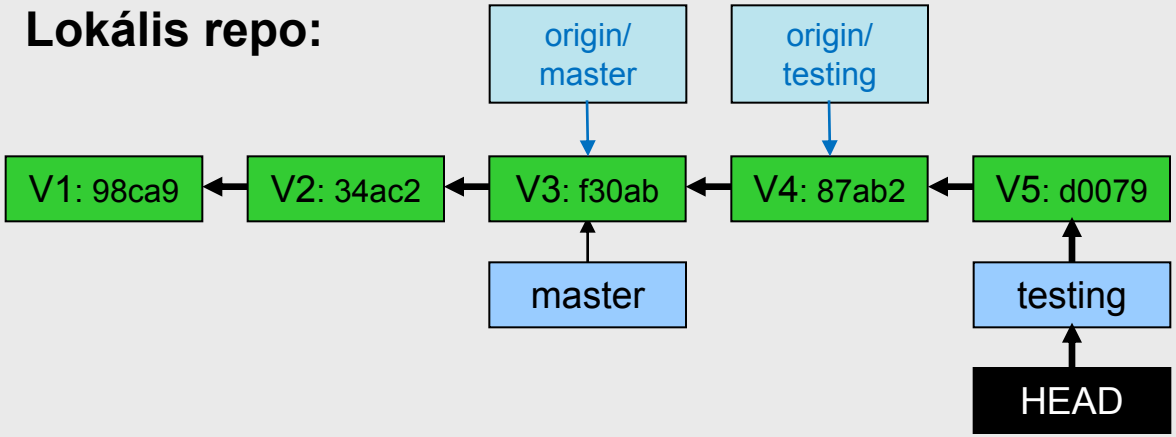
- Itt ha mutatna a V4-re helyi ág, akkor arra tudnánk állni (fölső ablak, de jelenleg nem mutat),
- Létre tudunk hozni egy új helyi ágat (Create local branch…)
- A Checkout remote branch opció pedig nem csinál semmit.

A **Local Changes** rész akkor érdekes, ha vannak nem commit-ált fájljaink (azaz a HEAD/Testing/V5 tartalma különbözik a Working dir. tartalmától). Ilyenkor a reset hard reset-et jelent, a többi 3 opciónál viszont megmaradnak a módosítások (a stash-nél még külön beteszi egy verembe. Merge-nél pedig, ha lesz merge conflict, megjeleníti)

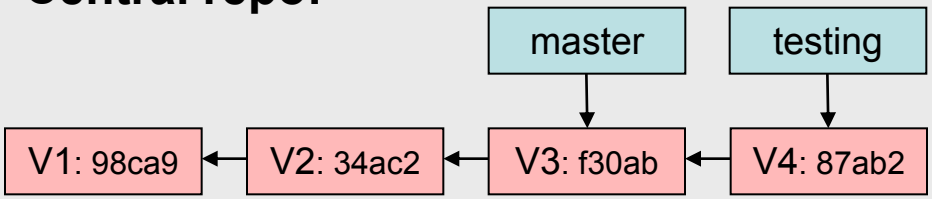
A stash műveletet nem szoktuk használni!

Online műveletek - példasor
Távoli fa (testing ág) változik

Lokális repo:



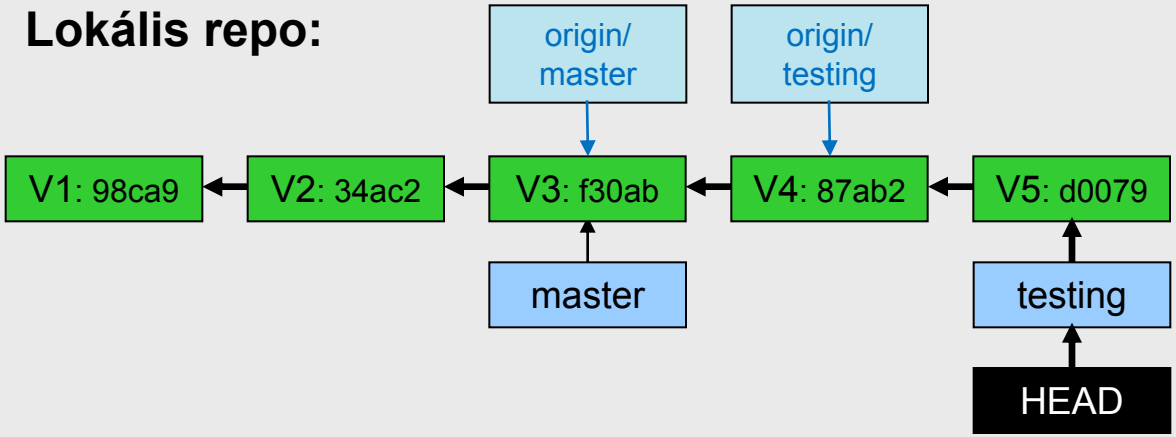
Central repo:



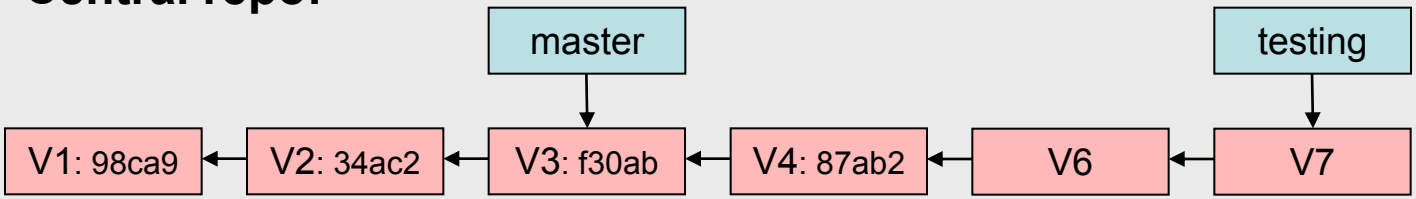
Sem checkout, sem reset, sem merge, rebase, fetch, pull, push, commit, stb., sem semmilyen művelet hatására nem fog a HEAD címke az origin/master vagy origing/testing címkére mutatni!

Online műveletek - példasor
Távoli fa (testing ág) változik

Lokális repo:



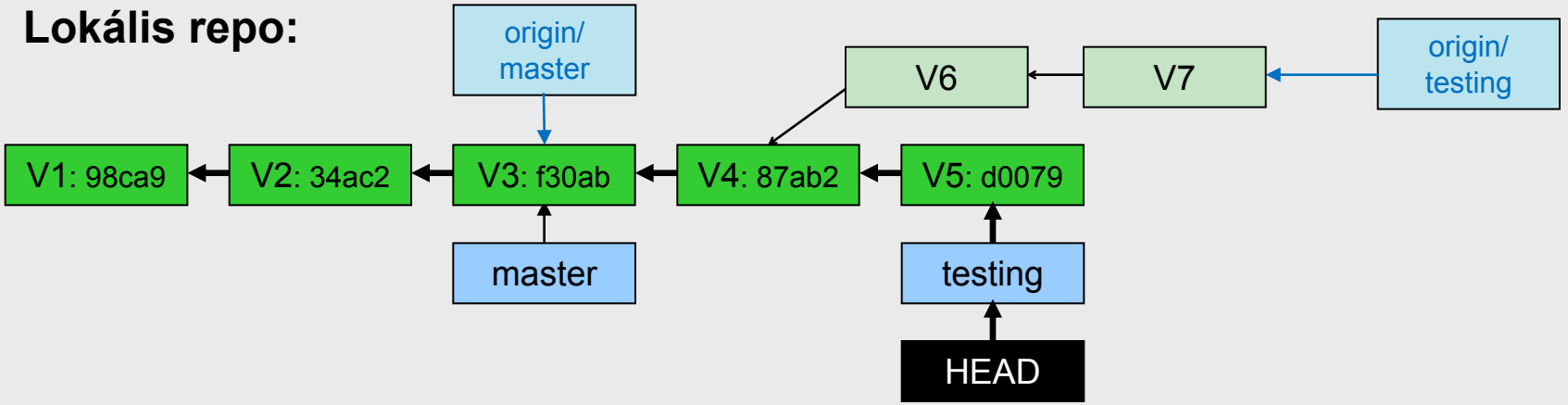
Central repo:



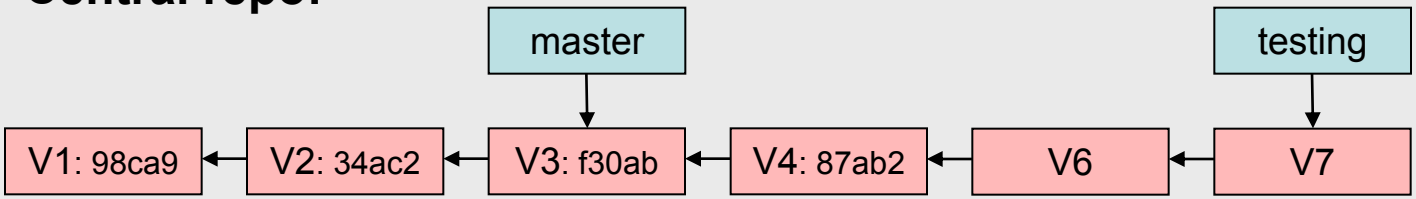
Sem checkout, sem reset, sem merge, rebase, fetch, pull, push, commit, stb., sem semmilyen művelet hatására nem fog a HEAD címke az origin/master vagy origing/testing címkére mutatni!

Online műveletek - példasor
Pull: fetch (testing)

Lokális repo:



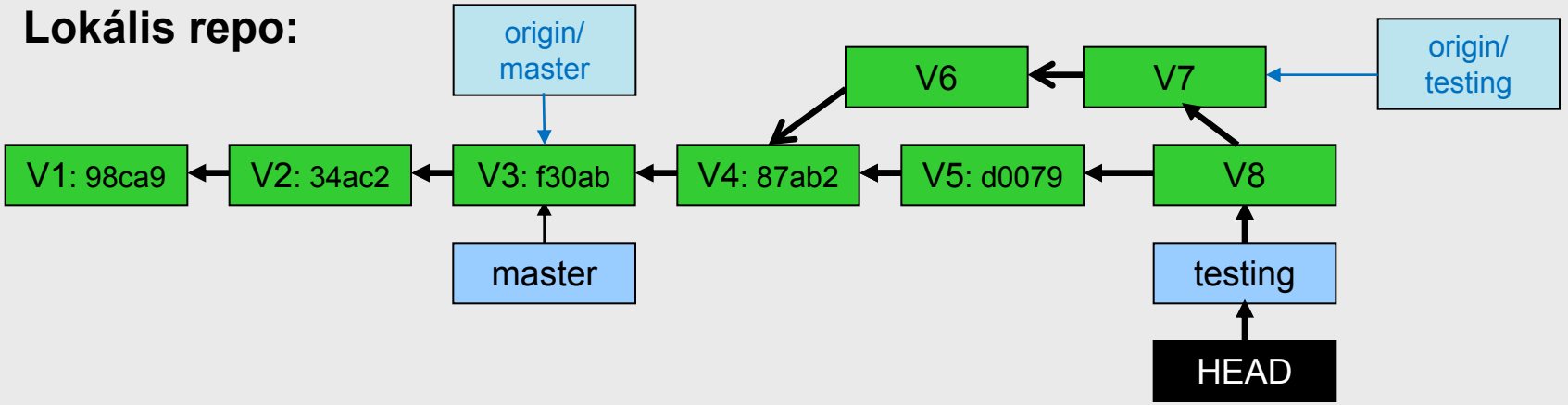
Central repo:



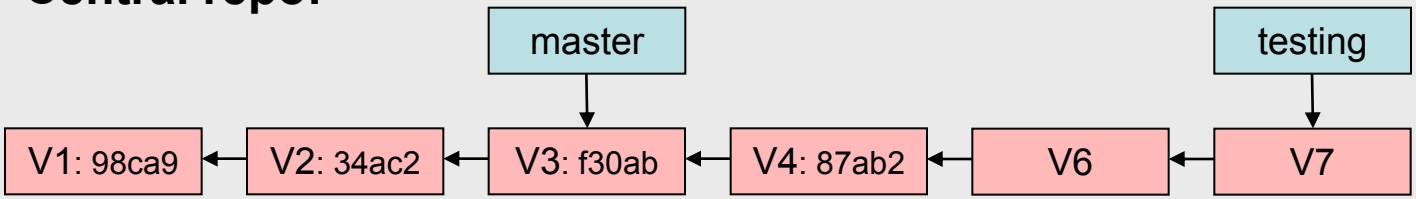
Sem checkout, sem reset, sem merge, rebase, fetch, pull, push, commit, stb., sem semmilyen művelet hatására nem fog a HEAD címke az origin/master vagy origing/testing címkére mutatni!

Online műveletek – példasor
Pull: fetch + merge (testing->testing)

Lokális repo:



Central repo:

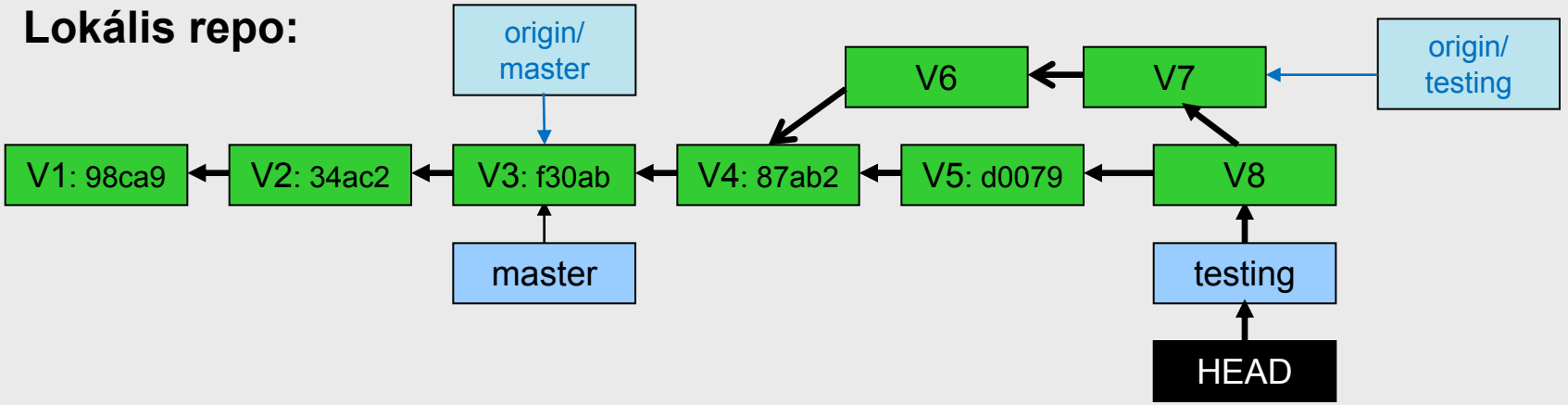


Sem checkout, sem reset, sem merge, rebase, fetch, pull, push, commit, stb., sem semmilyen művelet hatására nem fog a HEAD címke az origin/master vagy origing/testing címkére mutatni!

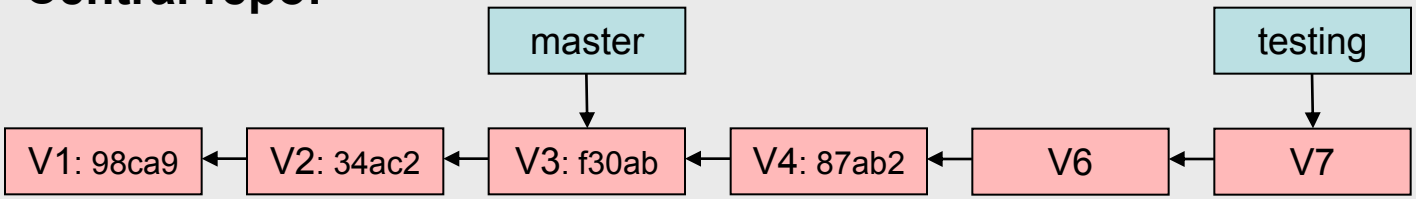
Az origin/testing mutatója nem állítódik át, csak a fetch-nél, mivel a V8 verziója nincs benne a távoli repository-ban!

Online műveletek – példasor
Push: master -> master

Lokális repo:



Central repo:

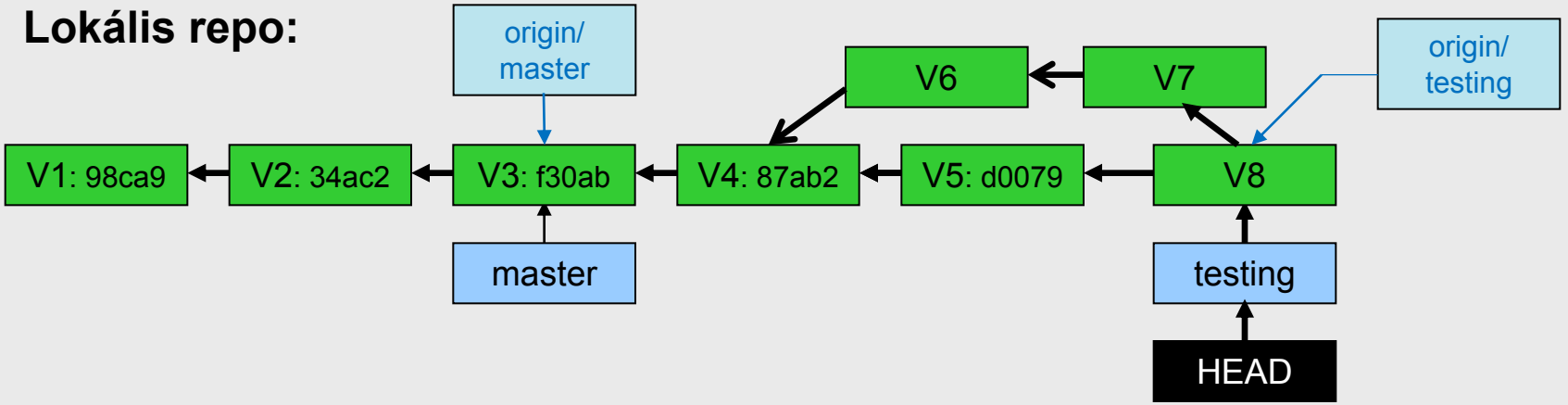


A háttérben valójában a push művelete is két részre bontható:

- Egy ág push-olásánál először felkerülnek a távoli repository hash-fájába **a lokális repo master ágának** azon revision-jei, amelyeket a távoli repo hash-fája eddig nem tartalmazott **(itt nincs ilyen).**
- Ezután a **távoli repo master ágának és a lokális repo origin/master ágának mutatója** átállítódik arra a **V3 revision-re, ahova a lokális master ág címkéje** is **mutat (mivel eddig is mindkettő a V3-ra mutatott, itt nem történik semmi)**

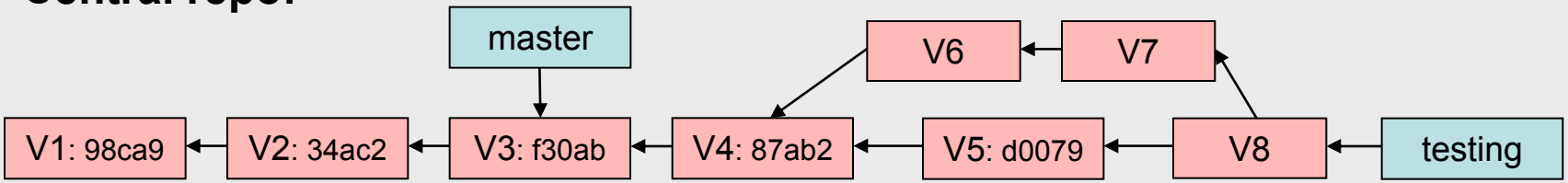
Online műveletek – példasor
Push: testing -> testing

Lokális repo:



Megjegyzés: Push műveleténél semmi jelentősége nincs annak, hogy a HEAD-el hol állok (kivéve, ha konkrétan a a HEAD-et push-olom)

Central repo:

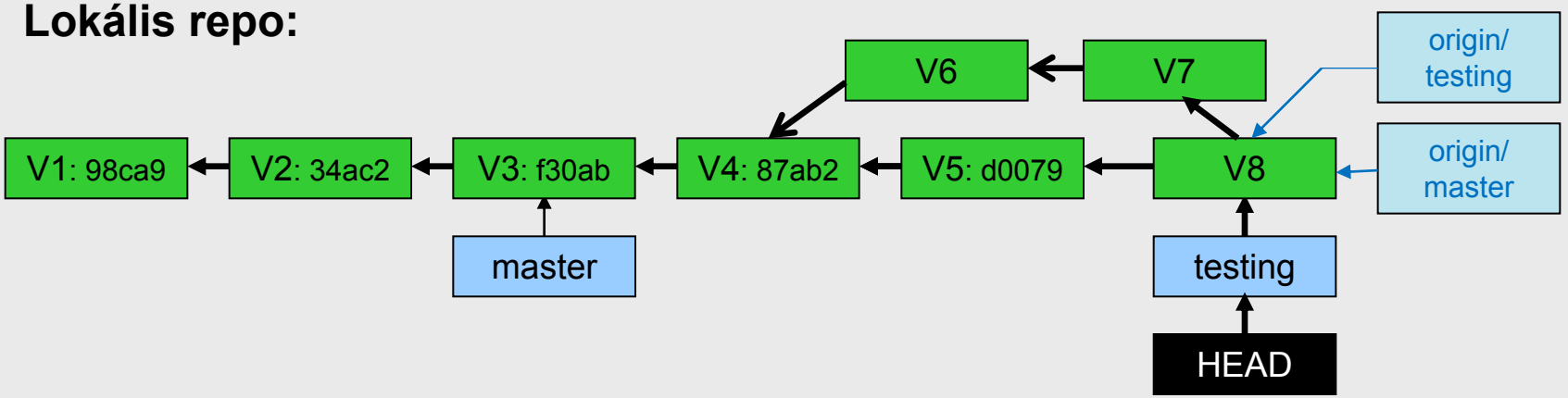


A háttérben valójában a push művelete is két részre bontható:

- Egy ág push-olásánál először felkerülnek a távoli repository hash-fájába **a lokális repo testing ágának** azon revision-jei, amelyeket a távoli repo hash-fája eddig nem tartalmazott **(<-V5<-V8 és a nyíl a V8-ból a V7-be).**
- Ezután a **távoli repo testing ágának és a lokális repo origin/testing ágának mutatója** átállítódik arra a **V8 revision-re, ahova a lokális testing ág címkéje** is mutat.

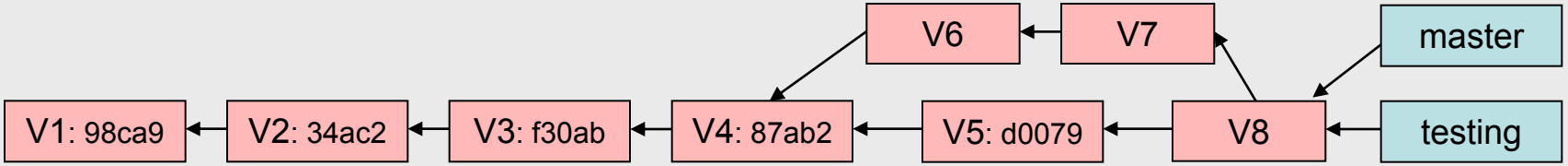
Online műveletek – példasor
Push: testing -> master

Lokális repo:



Megjegyzés: Push műveleténél semmi jelentősége nincs annak, hogy a HEAD-el hol állok (kivéve, ha konkrétan a a HEAD-et push-olom)

Central repo:

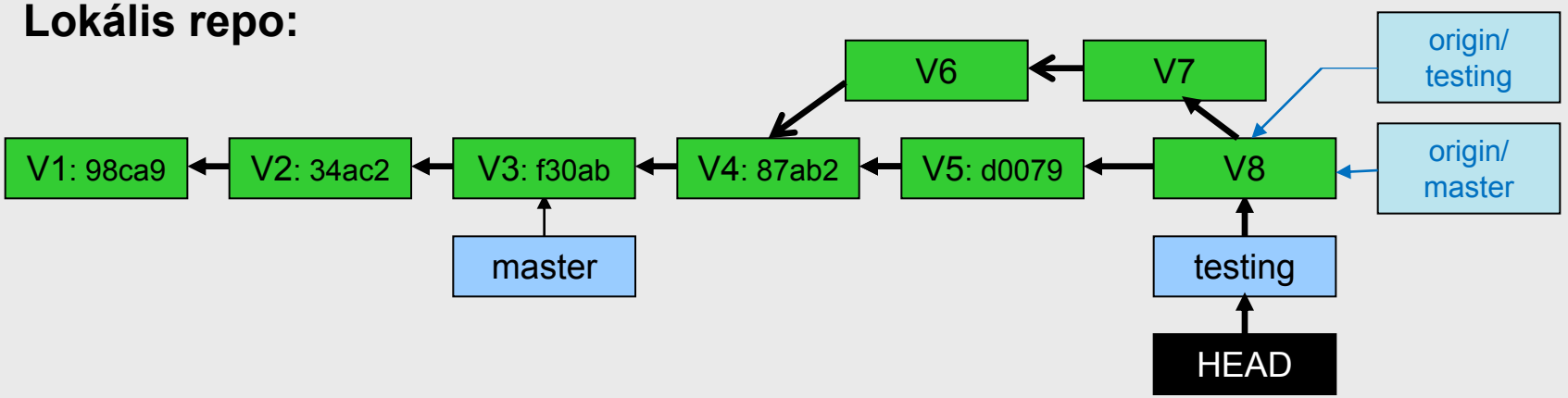


A háttérben valójában a push művelete is két részre bontható:

- Egy ág push-olásánál először felkerülnek a távoli repository hash-fájába **a lokális repo testing ágának** azon revision-jei, amelyeket a távoli repo hash-fája eddig nem tartalmazott **(itt nincs ilyen).**
- Ezután a **távoli repo master ágának és a lokális repo origin/master ágának mutatója** átállítódik arra a **V8 revision-re, ahova a lokális testing ág címkéje** is mutat.

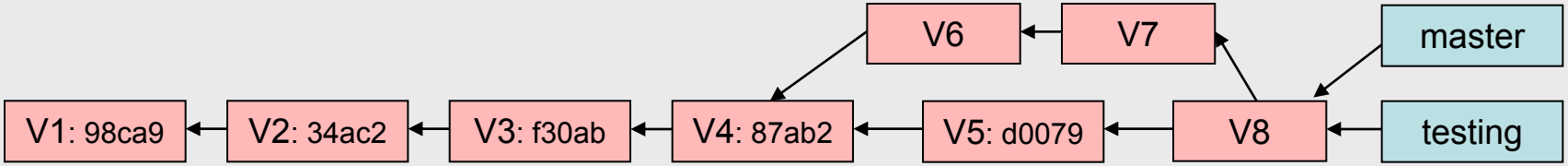
Online műveletek – példasor
Pull: fetch (master)

Lokális repo:



Megjegyzés: Push műveleténél semmi jelentősége nincs annak, hogy a HEAD-el hol állok (kivéve, ha konkrétan a a HEAD-et push-olom)

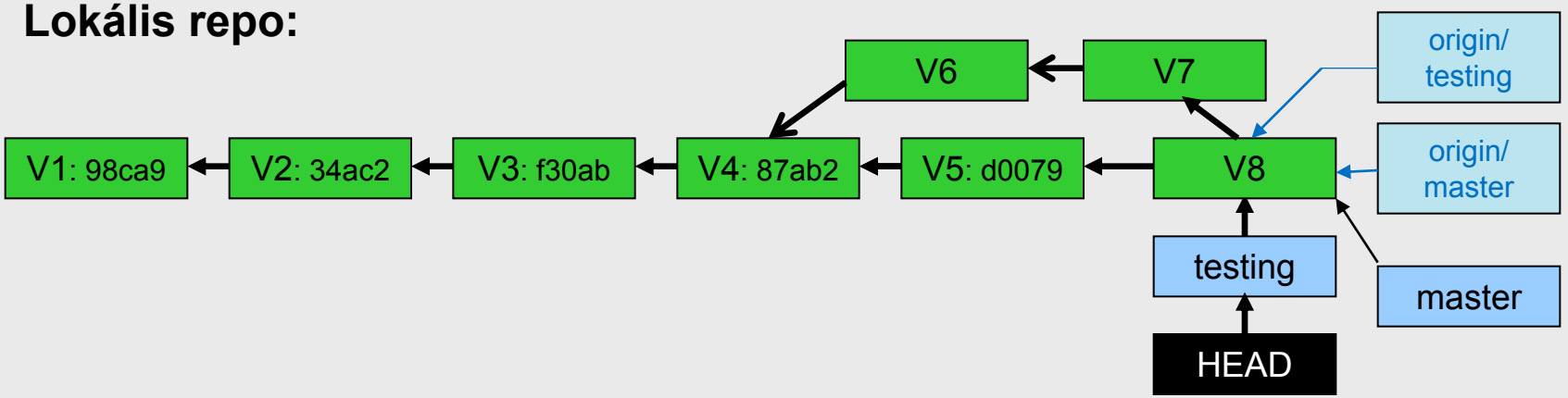
Central repo:



A pull fetch részénél jelen esetben nem történik semmi

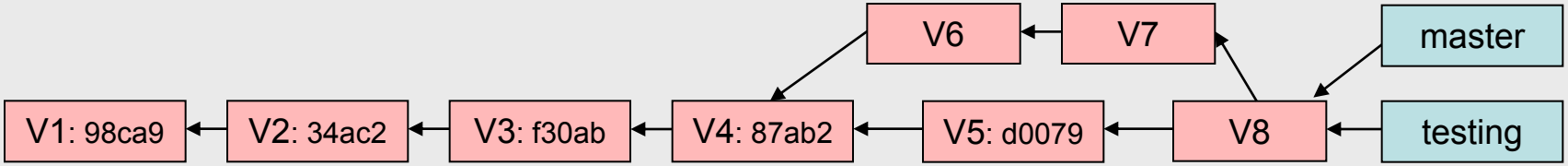
Online műveletek – példasor
Pull: fetch + merge (master->master)

Lokális repo:



Megjegyzés: Push műveleténél semmi jelentősége nincs annak, hogy a HEAD-el hol állok (kivéve, ha konkrétan a a HEAD-et push-olom)

Central repo:



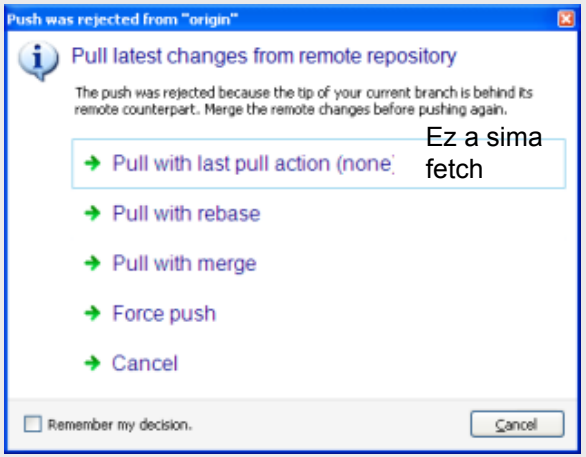
Az origin/master tartalmát merge-öljük a master ágba (jelen esetben tulajdonképpen csak egy fast-forward történik)

Force művelet jelentősége

Több műveletnél is opcionálisan bejelölhető, illetve megjelenhet ez a fogalom. Egy bármilyen műveletnél, amikor az ágcímke mutatója átállítódna, és ennek hatására lesz néhány revision, amire nem fog mutatni semmi, akkor a force opcióval ezek a revision-ök menthetetlenül törlődnek visszakérdezés nélkül!

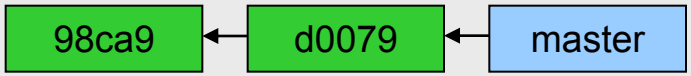
- Branch törlésénél (Delete Branch -> Force Delete)
- Force push műveletnél: amikor nincs közös részhalmaz a távoli repo master és lokal repo master ágának revision-jai között, vagy ha a lokális repo master ágcímkéjének hash-fája csak szigorúan véve részhalmaza a távoli repo megfelelő ágcímkéjének hash-fájával, azaz létezik a távoli repo ágcímkéjéhez tartozó hash-fában olyan elérhető revision, amely nincs benne a local repo megadott ágcímkéjéhez tartozó hash-fájában. Ilyenkor az se számít, ha a távoli repo-ban közvetlenül, vagy közvetve mutat arra a revision-re más ágcímke. Ezekben az esetekben a legjobb megoldás, ha először pull-olok, majd az origin ágat (a példában az origin/master-t) merge-ölöm a neki megfelelő lokális ágba (a példában a master ágba), ezután már tudok push-olni.

1. példa: push: master->master

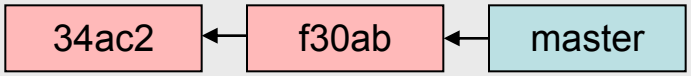


Ez a sima fetch

Lokális repo



A remote repo nem részhalmaza a lokális repo master hash-fájának (nincs benne a két zöld revision), ezért jön fel az ablak!
Megjegyzés: az alábbi esetben, ha csak sima fetch-el pullolunk, akkor lokális repo hash-fája nem lesz összefüggő



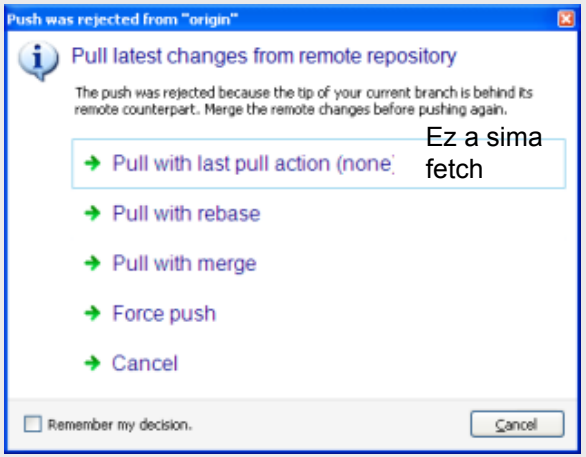
Megjegyzés: Push műveleténél semmi jelentősége nincs annak, hogy a local repo -ban a HEAD-el hol állok (kivéve, ha konkrétan a a HEAD-et push-olom)

Force művelet jelentősége *

Több műveletnél is opcionálisan bejelölhető, illetve megjelenhet ez a fogalom. Egy bármilyen műveletnél, amikor az ágcímke mutatója átállítódna, és ennek hatására lesz néhány revision, amire nem fog mutatni semmi, akkor a force opcióval ezek a revision-ök menthetetlenül törlődnek visszakérdezés nélkül!

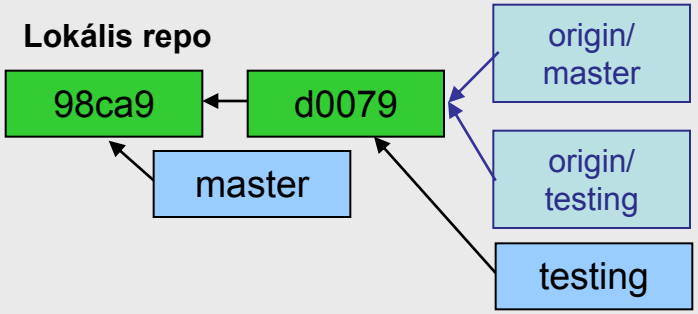
- Branch törlésénél (Delete Branch -> Force Delete)
- Force push műveletnél: amikor nincs közös részhalmaz a távoli repo master és lokal repo master ágának revision-jai között, vagy ha a lokális repo master ágcímkéjének hash-fája csak szigorúan véve részhalmaza a távoli repo megfelelő ágcímkéjének hash-fájával, azaz létezik a távoli repo ágcímkéjéhez tartozó hash-fában olyan elérhető revision, amely nincs benne a local repo megadott ágcímkéjéhez tartozó hash-fájában. Ilyenkor az se számít, ha a távoli repo-ban közvetlenül, vagy közvetve mutat arra a revision-re más ágcímke. Ezekben az esetekben a legjobb megoldás, ha először pull-olok, majd az origin ágat (a példában az origin/master-t) merge-ölöm a neki megfelelő lokális ágba (a példában a master ágba), ezután már tudok push-olni.

2. példa: push: master->master

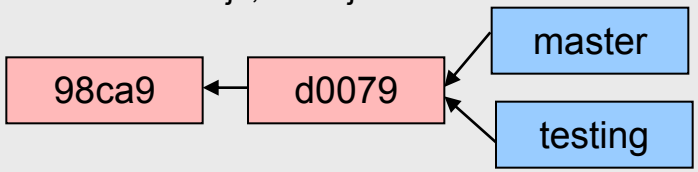


Ez a sima fetch

Lokális repo

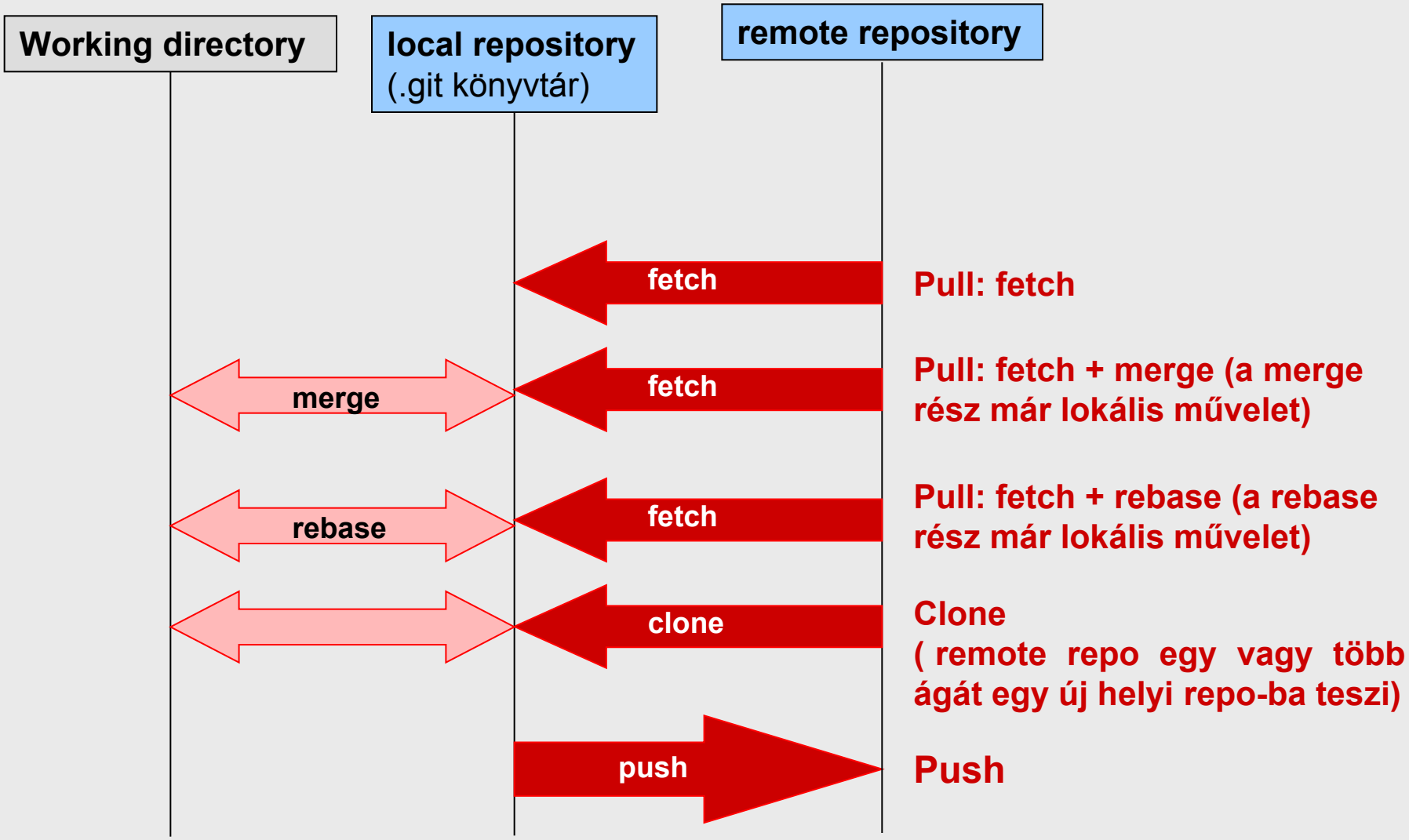


A lokális repo master-ének hash-fájában nincs benne a remote repo d0079 revision-je, ezért jön fel az ablak!



Megjegyzés: Push műveleténél semmi jelentősége nincs annak, hogy a HEAD-el hol állok (kivéve, ha konkrétan a a HEAD-et push-olom)

Online műveletek



Köszönetnyilvánítás

- Elsősorban szeretnék köszönetet mondani **László István** igazgató úrnak, hogy segített abban, hogy ez az előadás létrejöjjön.
- Szeretnék köszönetet mondani Kolesár Andrásnak, hogy a GitHub és GitLab technikájának megismertetésével teljessé tette az előadást.
- Megjegyzés: Az előadásban szereplő ábrák kb. felét a Git hivatalos oldaláról vettem (https://git-scm.com/book/en/v2), a másik fele pedig vagy saját, vagy ugyaninnen vett, de kiegészített példák.

- **Köszönöm a figyelmet!**
- **Mindenkinek békés, boldog, kellemes karácsonyi ünnepeket és boldog új évet kívánok!**