MISKOLCI EGYETEM
GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR

Szoftvertechnológia gyakorlatok

GEIAL316-B2

Verziókezelés

Dr. Tompa Tamás
egyetemi adjunktus
Általános Informatikai Intézeti Tanszék

Miskolc, 2025

Verziókezelés

- A **verziókezelés** olyan eljárások összessége, amelyek lehetővé teszik egy **adathalmaz változatainak (verzióinak) együttes kezelését**

  - **Adathalmaz:**
    - **forráskód, dokumentumok stb.**
    - szoftverek estében, a szoftver életciklusa során **elsősorban a forráskódban végzett módosítások** tárolását jelenti
    - **de nem csak a forráskód**ot lehet verziókövetni!

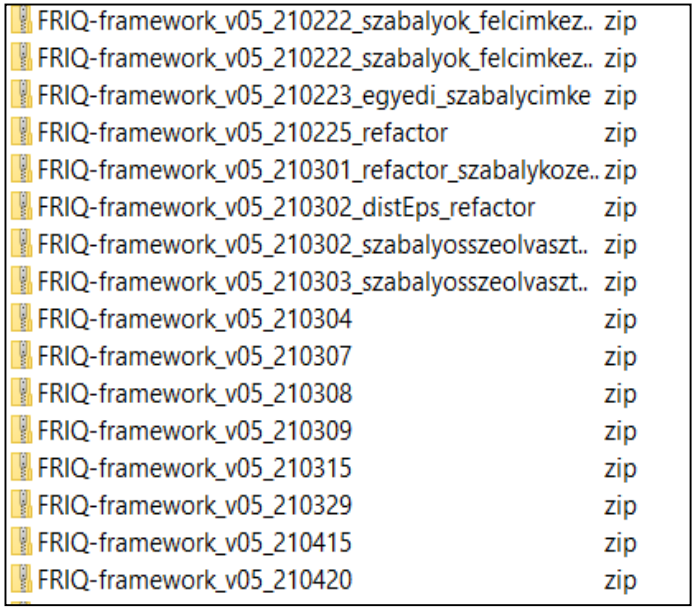
Verziókezelés

- A fejlesztés során **a forráskód sok iteráción megy keresztül**
  - esetleges probléma esetén **vissza kellene térni egy korábbi verzióra, állapotra**
- Nagy segítséget nyújt **csapatmunkában** és **egyénileg** történő szoftverfejlesztés esetében is
  - kódbázis megosztása, az éppen aktuális állapot mindenkinél
- Elnevezések: Revision Control, Version Control System (VCS), Source Control

Verziókezelés

- **Egyszerű „verziókövetés”**
  - minden változtatást külön jegyzékben elmentünk
    - megvalósítható, de nehézkes
    - időidényes, nem hatékony
    - nem átlátható
    - és ha ezek is elvesznek?
    - → erősen nem javasolt…

hogy ne… →



- **Megoldás: verziókövető rendszerek használata**
  - pl. Git, Mercurial, SVN, stb.

Alapfogalmak

- **Repository:** maga a **tároló**, röviden csak repo
  - lehet használni helyi (local) illetve távoli (remote) repokat, ilyen távoli repok például a Github, Bitbucket stb.
  - a tárolók pillanatképként tartalmazzák a változásokat, illetve a projekt aktuális állapotát
- **Working copy:** a **kód egy részének egy példánya**, amelyen a fejlesztő éppen dolgozik a saját gépén
- **Push:** adatok/commit-ok **feltöltése** a remote repoba
- **Pull:** változások **letöltése** a remote repo-ból
- **Diff/Change:** két file között **változás** megjelenítése

Alapfogalmak

- **Commit:** a kommit egy **lényegretörő megjegyzés**, ami összefoglalja, hogy **milyen módosítás történt**
  - a kódon eszközölt változtatásokat kommitok formájában lehet érvényesíteni
  - célszerű minden kisebb módosítást követően kommitolni
    - ne 1000 soronként, de ne is karakterenként…

hogy ne…:

pici css móka committed 3 months ago	elfelejtettem ezt a portot committed 4 months ago
css committed 3 months ago	amúgy sem működik... committed 4 months ago
pár apróbb ság amit kért committed 3 months ago	d(*.*)b committed 6 months ago
little css committed 4 months ago	szeretek kétszer dolgozni committed 6 months ago
egy újabb committed 4 months ago	még egy fix committed 6 months ago
elkeseredett próbálkozás committed 4 months ago	fix the fix committed 6 months ago
ismét committed 4 months ago	

Alapfogalmak

- **Commit típusok (scope)**
  - **build** - build folyamatával kapcsolatos változások
  - **ci - CI** - (Continuous Integration) folyamattal kapcsolatos változások
  - **chore** - build folyamattal kapcsolatos változások
  - **docs** - dokumentációs változtatások
  - **feat** - új funkció
  - **fix** - hibajavítás
  - **perf** - kódbeli változtatás, amely javítja a teljesítményt
  - **refactor** - kódbeli változtatás, amely sem hibát nem javít, sem új funkciót nem ad hozzá, csak refaktorálás
  - **revert** - visszavonások
  - **style** - szóköz, formázás, hiányzó pontosvesszők stb.
  - **test** - tesztek hozzáadása

Alapfogalmak

- **Commit szabályok**
  - „**scope: message**” formátum
  - angol, jelen időben
  - 50 karakternél ne legyen hosszabb
  - rövid, lényegretörő
  - **mit miért**, és nem mit hogyan

  - Példák:
    - `build: Update webpack configuration`
    - `ci: Add GitHub Actions workflow for automated testing`
    - `chore: Update dependencies using npm audit fix`
    - `feat: Add user authentication feature`
    - `fix: Fix issue with form validation not working on Safari`
    - `revert: Revert "feat: Add user authentication feature„`
    - `test: Add unit tests for login component`

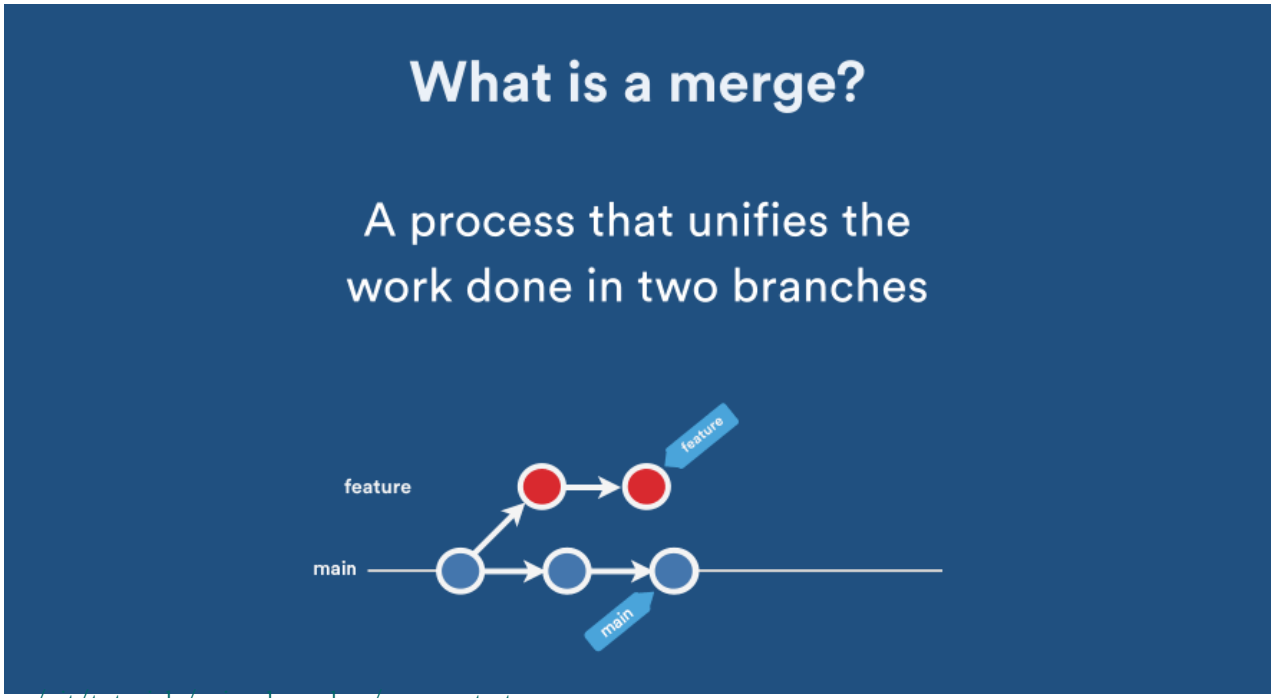
Alapfogalmak

- **Revision:** verzió
- **Checkout:** branch váltás, commit visszaállítás
- **Head:** a legfrissebb kommitot (verziót) jelöli, az aktuális ág teteje
- **Snapshot:** adott időpillanatban fájlok aktuális állapota
- **Clone:** repo tartalmának letöltése (háttértárra lokálisan)
  - megadott jegyzékbe

Alapfogalmak

- **Merge:** összefésülés, fejlesztési ágak (branch) egyesítése
  - új commit (mindkét ág változásai)
  - megmarad a branch-ek története
  - conflict-okat jelzi

(animáció)

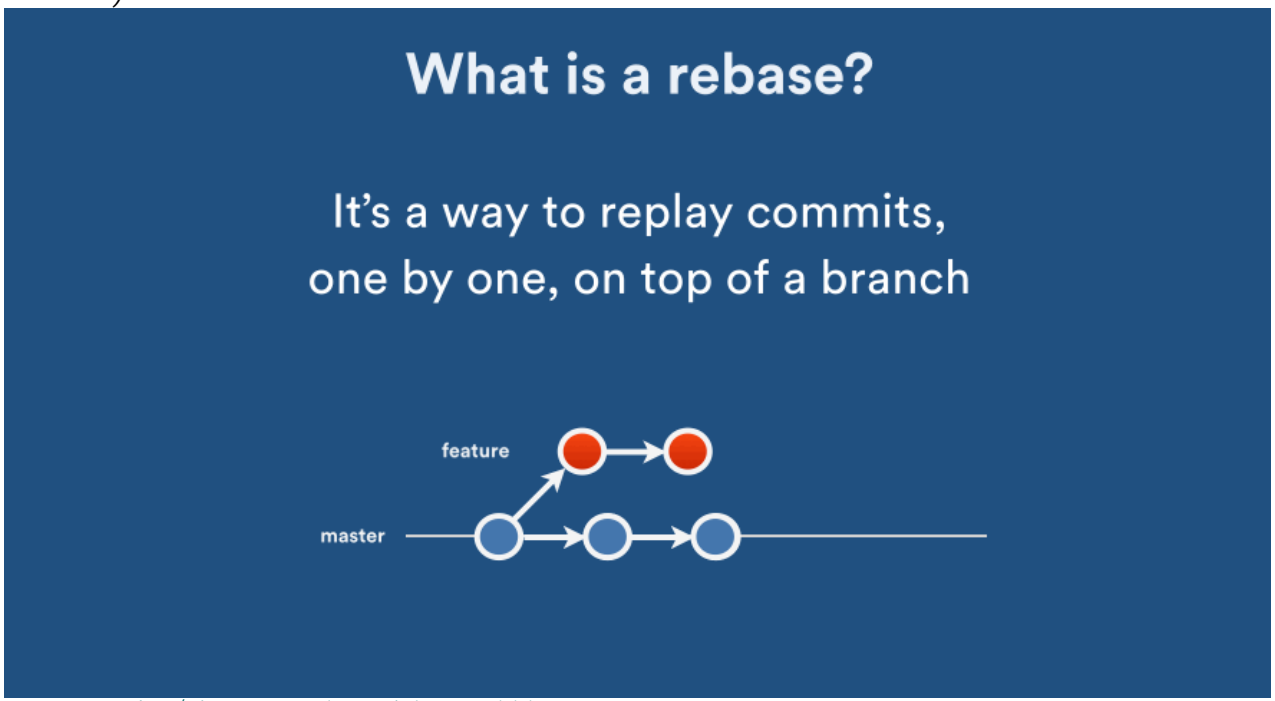


Forrás: https://www.atlassian.com/git/tutorials/using-branches/merge-strategy

Alapfogalmak

- **Rebase:** az aktuális ágban lévő commitok érvényesítése egy másik ágra
  - commit-okat áthelyezi a másik ágra
  - conflict esetén felfüggeszti a folyamatot (manuális feloldás)

(animáció)



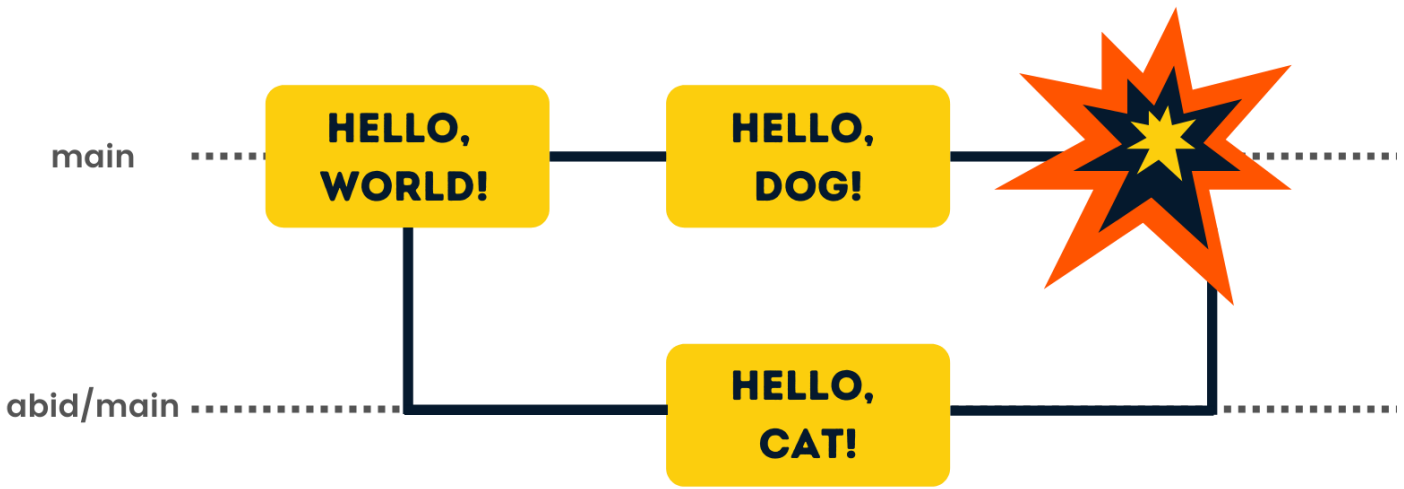
Forrás: https://medium.com/pranayaggarwal25/git-merge-rebase-d8b91825bbb1

Alapfogalmak

- **Merge vs. Rebase:**
  - **A merge:**
    - új commit-ot hoz létre amiben egyesíti a 2 ágat
    - az összes előzmény érintetlen marad, így az eredeti elágazás és az egyesítés folyamata látható a commit történetben
    - megőrzi a fejlesztés teljes történetét
    - akkor ha meg akarjuk őrizni az eredeti fejlesztési történetet, és fontos látni honnan ágaztak el a változások
  - **A rebase:**
    - áthelyezi a fejlesztési ág commitjait egy másik ág legfrissebb állapotára, újraírás nélkül
    - nem hoz létre külön merge commitot
    - akkor ha egy tiszta, lineáris commit történetet akarunk
  - fontos: konfliktok jöhetnek létre, óvatosan velük!

Alapfogalmak

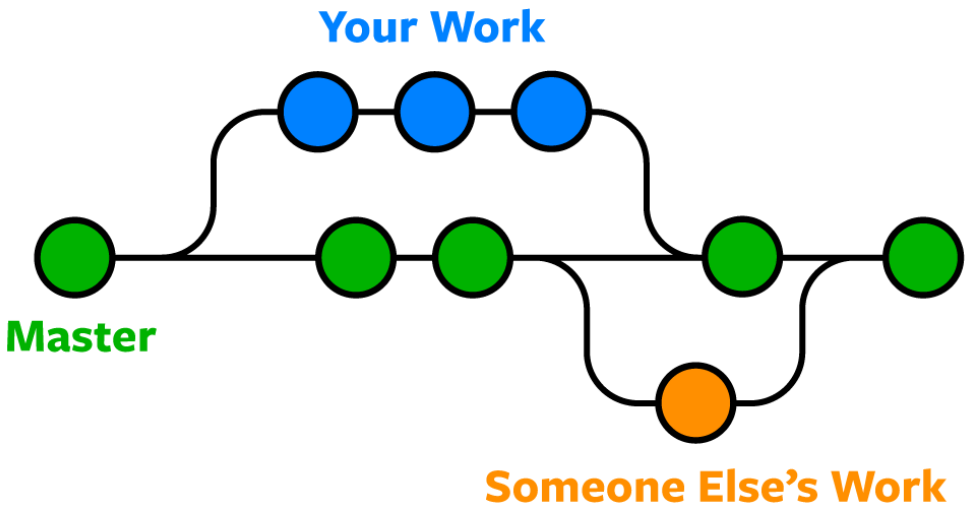
- **Conflict:** fejlesztési ágak (branch) összefésülése során keletkező jelenség, az adat ugyanazon részén történt módosítás több szereplő által, nem lehet automatikusan összefésülni
  - ha előfordul → fel kell oldani
  - feloldás: konfliktus jelzők által manuálisan vagy IDE-ben



Forrás: https://www.datacamp.com/tutorial/how-to-resolve-merge-conflicts-in-git-tutorial

Alapfogalmak

- **Branch:** fejlesztési ág, a fejlesztés ne csak egy „szálon” történjen
  - elterjedtebb branch-ek: master (main), develop, feature, release, hotfix stb.



  - ha lehet egy branch-en minél kevesebb időt kell tölteni
    - addig sokminden történik a master-en → sok conflict
    - → kis (1 napos) feladatok/ticket-ek

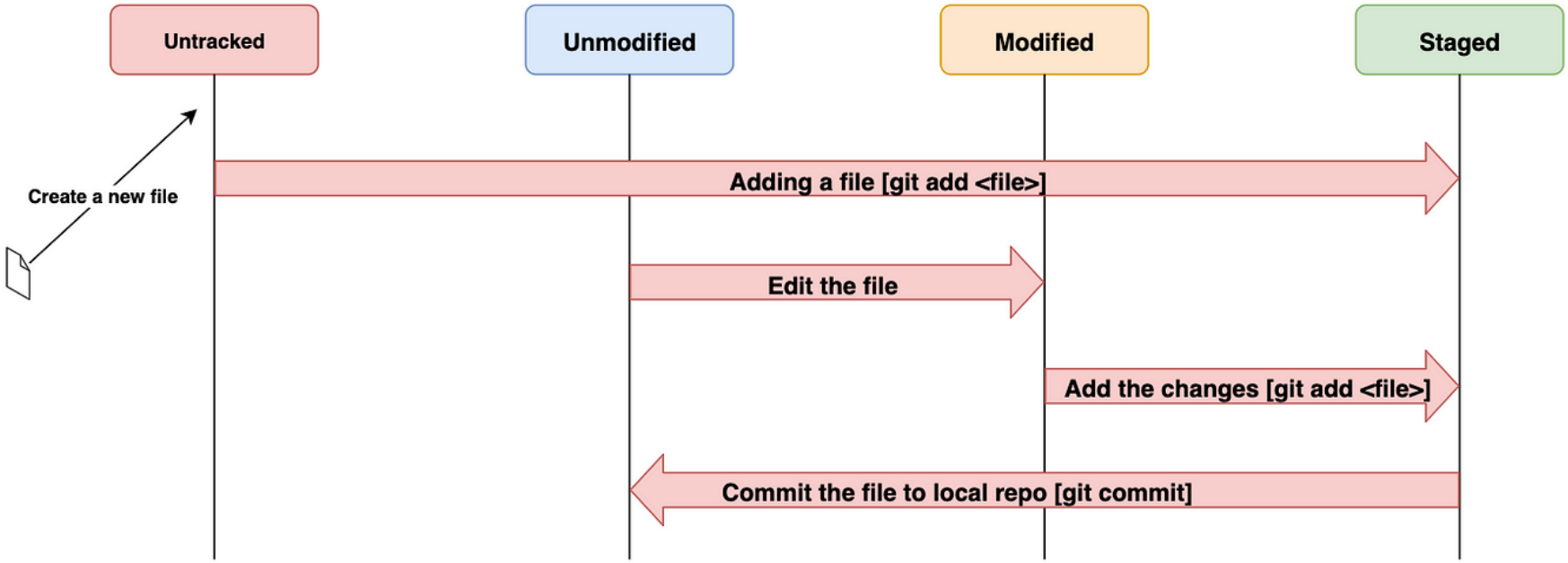
Forrás: https://www.nobledesktop.com/image/gitresources/git-branches-merge.png

Fájlok állapotai

- **Untracked:** azok a fájlok melyek változásai nincsennek nyomonkövetve

- **Tracked:** azok a fájlok melyek változásai nyomon vannak követve
  - modified: a fájl módosult a legutóbbi commit óta
  - unmodified: a fájl nem módosult a legutolsó commit óta
  - staged: a fájl módosult a legutóbbi commit óta, a következő commit során el lesz tárolva a változása

Fájlok állapotai

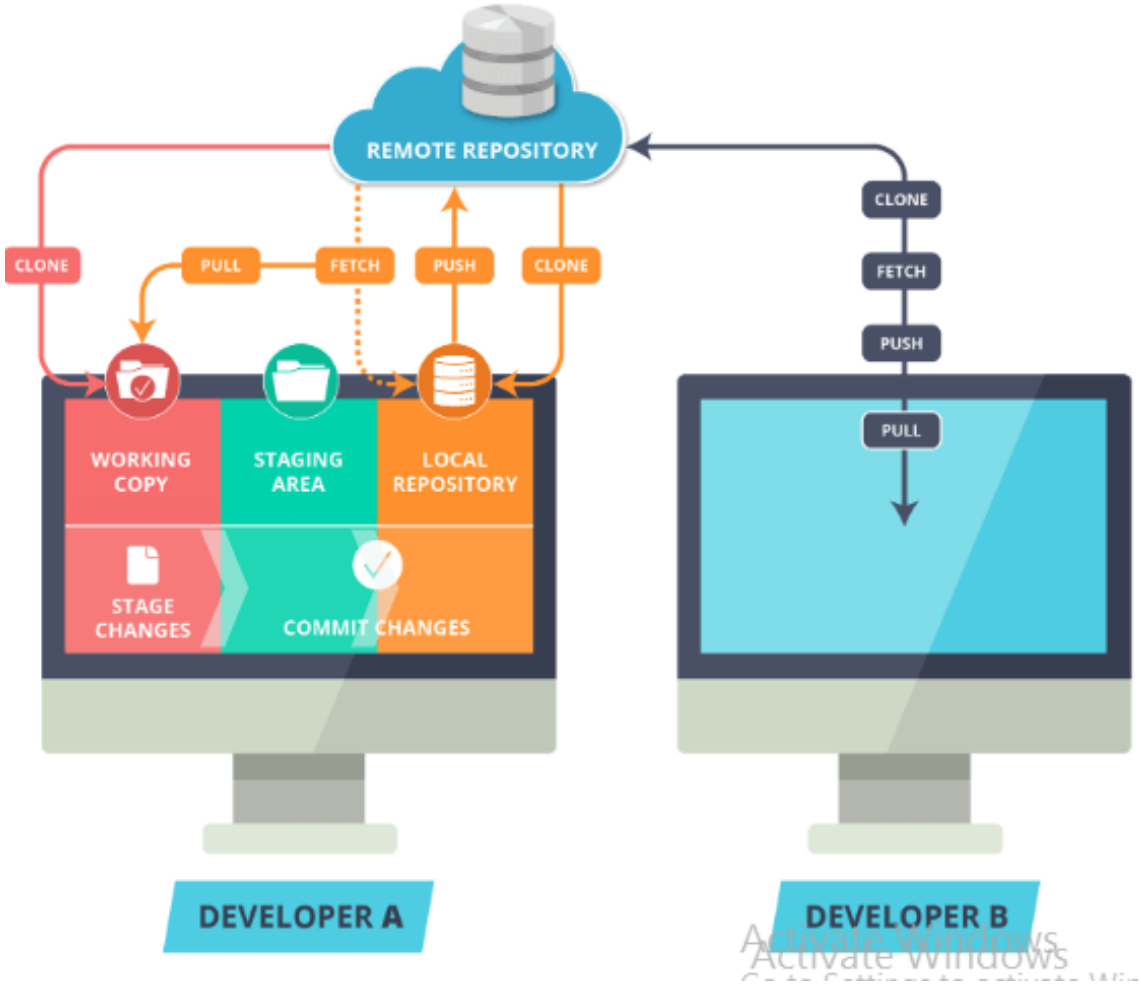


Forrás: https://miro.medium.com/v2/resize:fit:1100/format:webp/1*VlEZHRPT7vumT_505jLaHg.png

Repository-k típusai

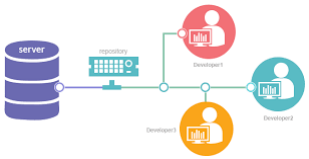
- **local:** változások a helyi fájlrendszeren, helyi repo-ban
- **remote:** változások a távoli szerveren, távoli repo-ban

- **public:** bárki hozzáférhet
- **private:** csak engedélyezett felhasználók férhetnek hozzá

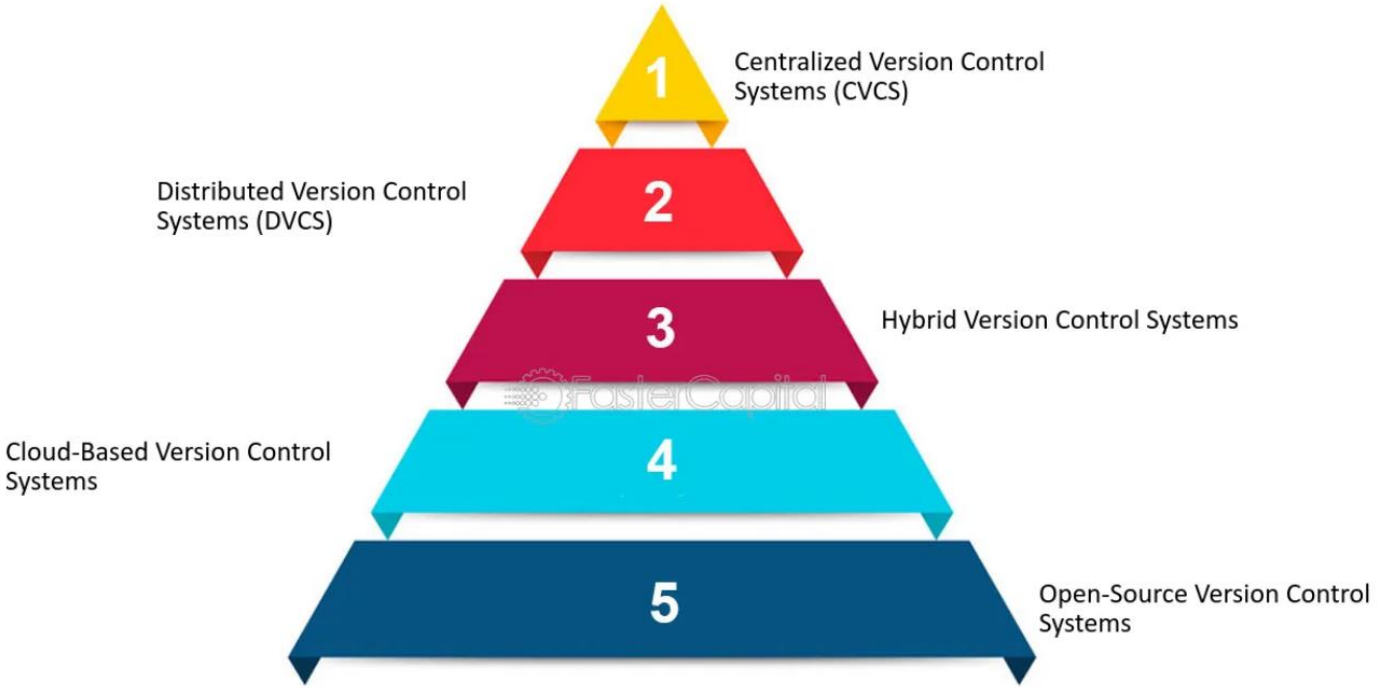


Forrás: https://www.c-sharpcorner.com/article/git-and-github-version-control-local-and-remote-repository/Images/Git%20And%20Github%20Version%20Control10.png

Verziókezelő rendszer típusok

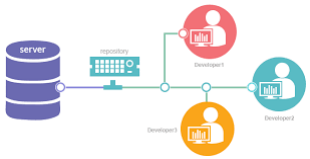


- **Alapvetően 2 féle**
  - **Centralizált (CVCS, központosított)**
  - **Decentralizált rendszer (DVCS, elosztott)**



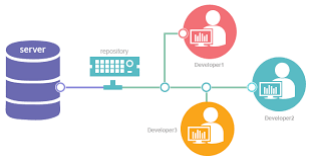
Forrás: https://fastercapital.com/topics/types-of-version-control-systems.html

Verziókezelő rendszer típusok

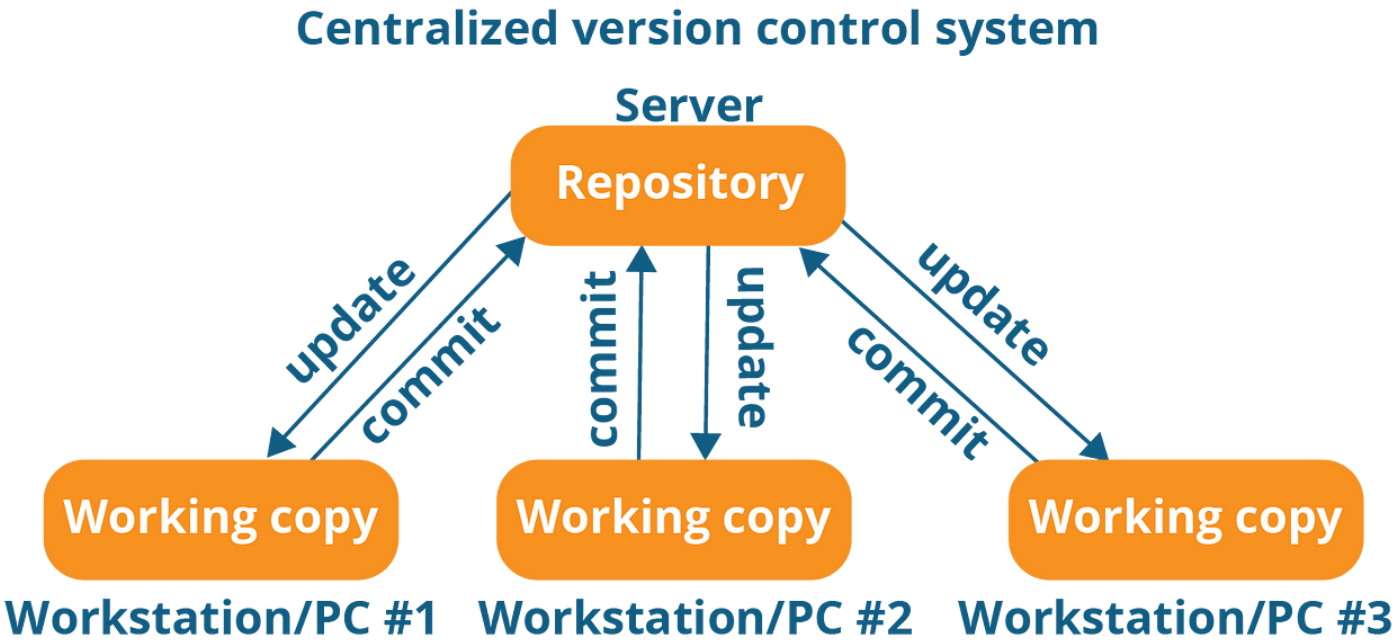


- **Centralizált (CVCS, központosított)**
  - **közös központi repo**
    - minden fejlesztő közvetlenül ehhez kapcsolódik
  - egységes verziókezelés
    - a fejlesztők mindig a központi repo változatával dolgoznak
  - **minden művelet szerveren**
    - mindig van kapcsolat a szerverrel
    - minden commit után frissítés kell, hogy a változások mindenkinél megjelenjenek
    - minden változtatást a központi repo-ba kell feltölteni, így az erősen kötődik a központi szerver elérhetőségéhez

Verziókezelő rendszer típusok

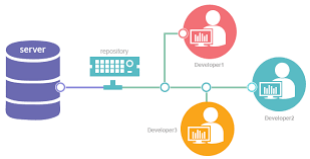


- **Centralizált (CVCS, központosított)**



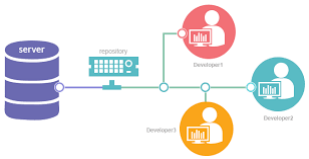
Forrás: https://faun.pub/centralized-vs-distributed-version-control-systems-a135091299f0

Verziókezelő rendszer típusok

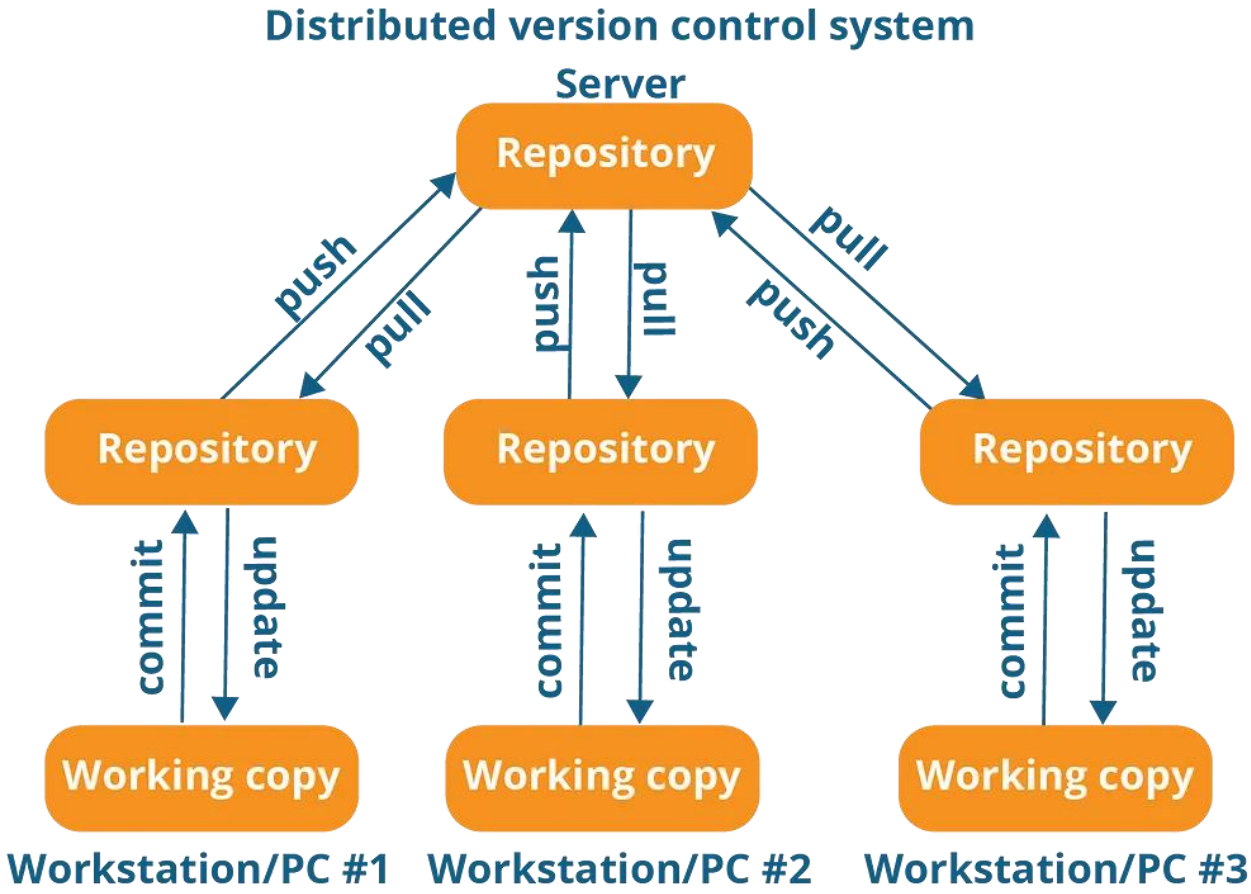


- **Decentralizált rendszer (DVCS, elosztott)**
  - **nincs központi tároló**
    - mindenkinek saját helyi repo-ja van
    - felhasználói gépek a tárolók (local repo)
    - de mellette lehet távoli repo is
    - a fejlesztők a saját helyi repository-jukban dolgoznak, és csak akkor szinkronizálják a változtatásokat a távoli repository-kkal, amikor készen vannak
  - **nem kell központi szerverrel kommunikálni**
    - gyors
    - nem függ a szerver elérésétől → nagyobb rugalmasság

Verziókezelő rendszer típusok

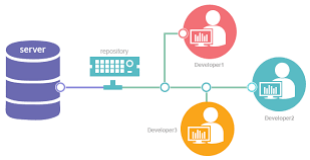


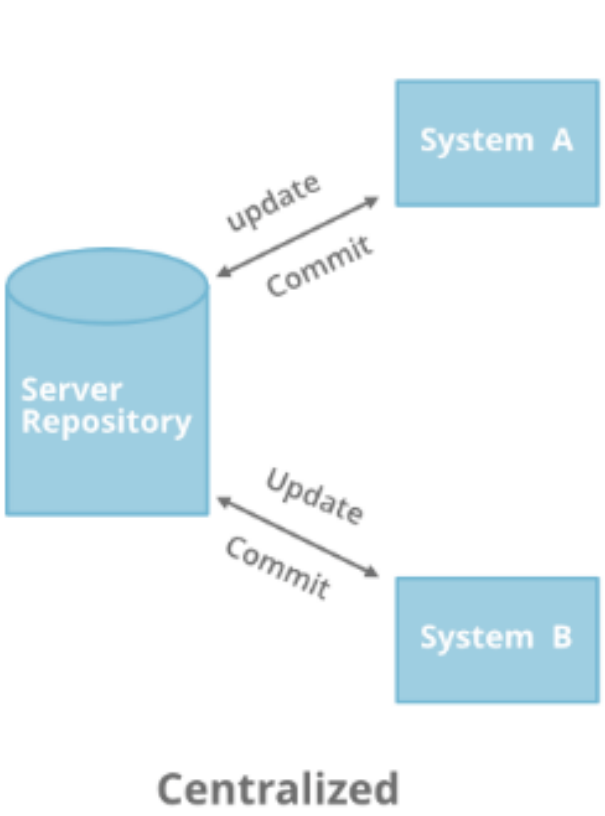
- **Decentralizált rendszer (DVCS, elosztott)**

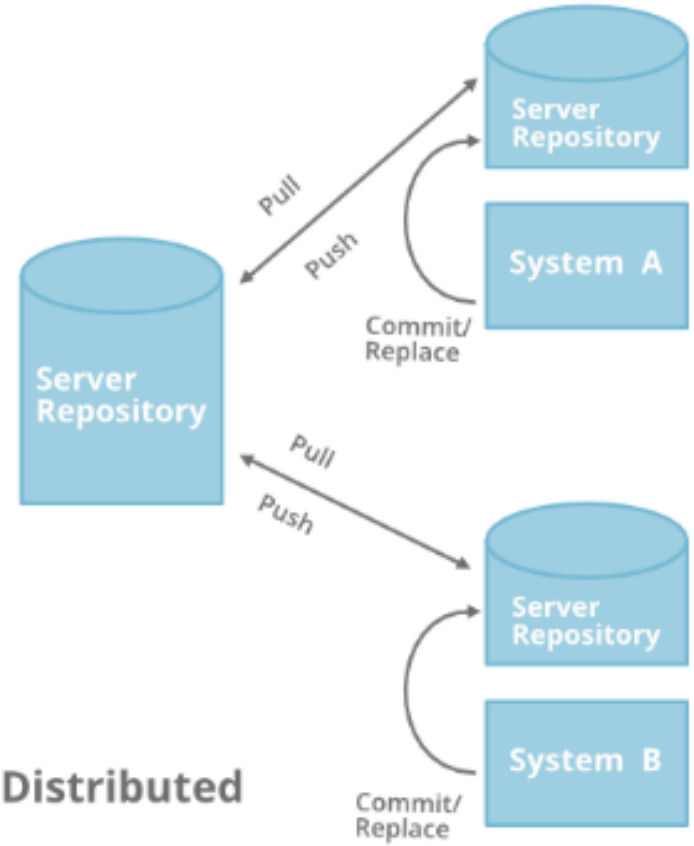


Forrás: https://faun.pub/centralized-vs-distributed-version-control-systems-a135091299f0

Verziókezelő rendszer típusok

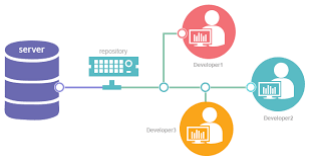




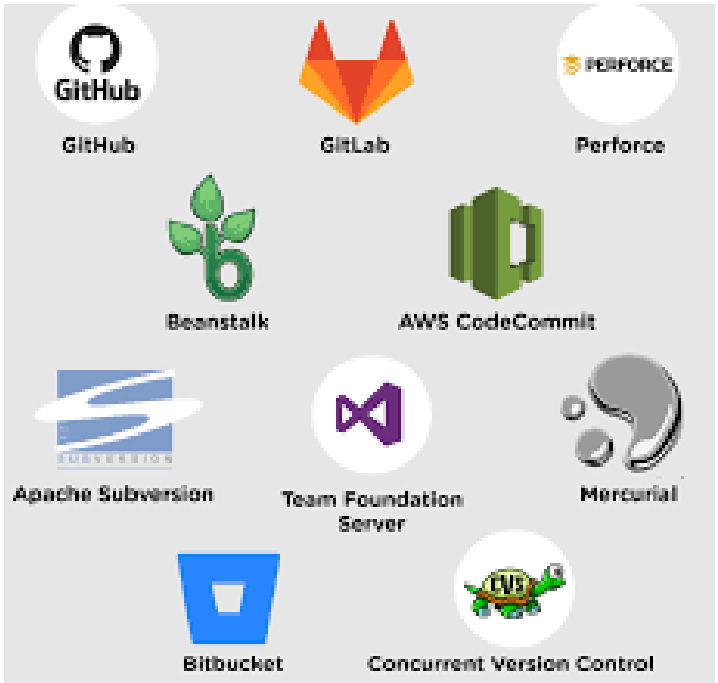


Forrás: https://media.geeksforgeeks.org/wp-content/cdn-uploads/20190820174942/CVCS-vs-DVCS.png

Verziókezelő rendszerek



- **Alapvetően 3 fő nagyobb:**
  - **Git**
  - **Mercurial**
  - **SVN**



Forrás: https://bacsoftwareconsulting.com/general/top-free-version-control-systems/

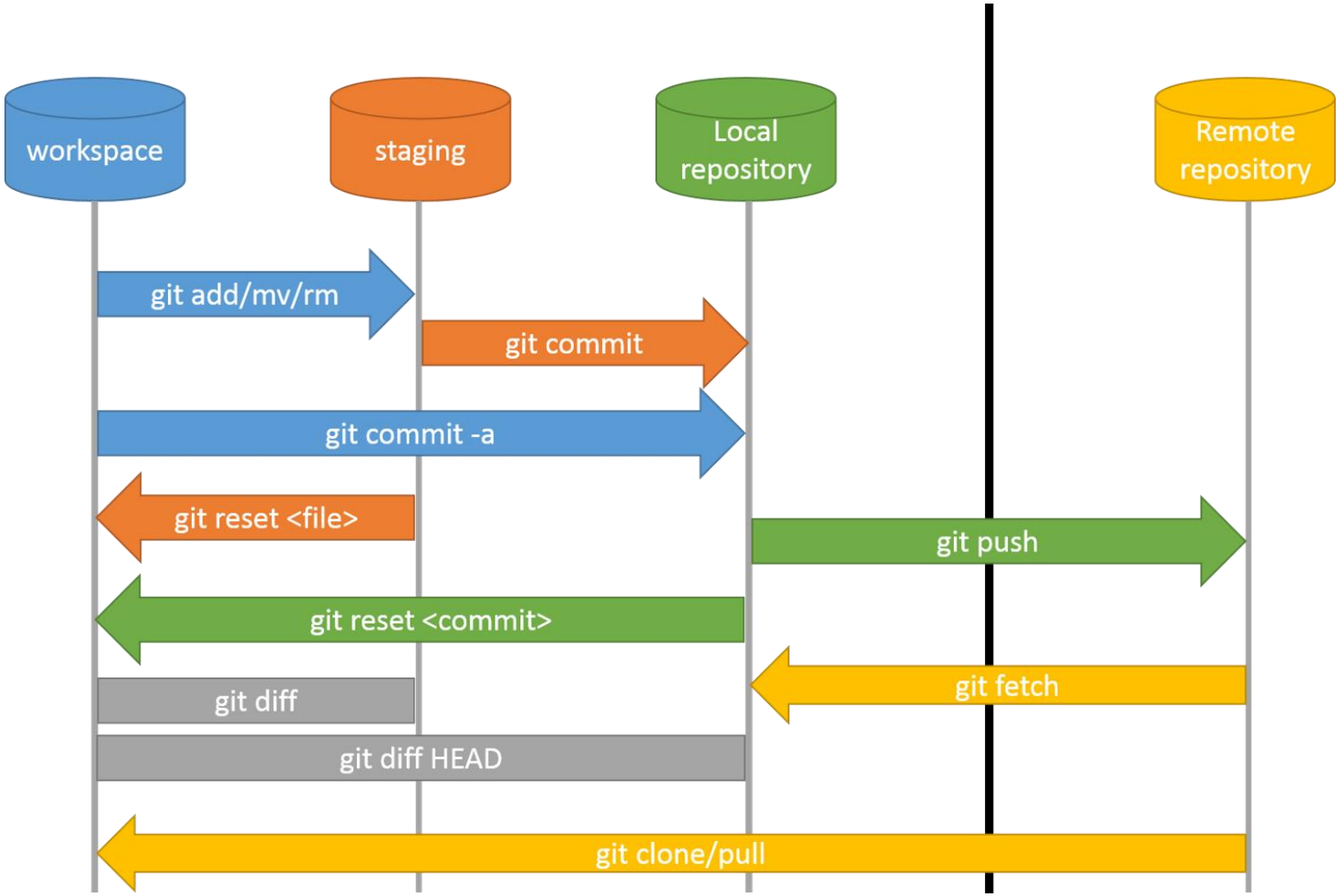
Git

- elosztott verziókezelő (DVCS)
  - minden fejlesztő teljes repo másolattal rendelkezik
- gyors, mert a legtöbb művelet lokálisan történik
- erőteljes branching és merging (ágkezelés), lehetőség párhuzamos fejlesztésre
- GitHub, GitLab, Bitbucket támogatja
- parancssoros és grafikus kliensek (pl. GitKraken, SmartGit)
- alkalmas nagy és kis projektekhez is
- Hook-ok támogatása

Git

- **Git hook-ok**
  - **egy adott esemény (pl. commit, push, update, stb.) során egy előre meghatározott parancs vagy script automatikus lefutása**
  - **Client-side (helyben futnak a fejlesztő gépén)**
    - pl. commit előtt ellenőrzi a kódstílust
  - **Server-side (a szerveren futnak, pl. GitHub, GitLab)**
    - pl. push után küld egy Slack üzenetet vagy elindít egy CI/CD pipeline-t
  - .git/hooks/ könyvtárban
  - shell script vagy más programozási nyelveken
  - pre-hook: esemény előtt, post-hook: esemény után
  - Célja:
    - automatikus ellenőrzések, kódformázás, statisztikus kódelemzés, CI/CD folyamatok indítása, automatikus tesztelés minden push után, biztonsági szabályok betartása, titkos kulcsok és jelszavak kiszűrése commit előtt stb.

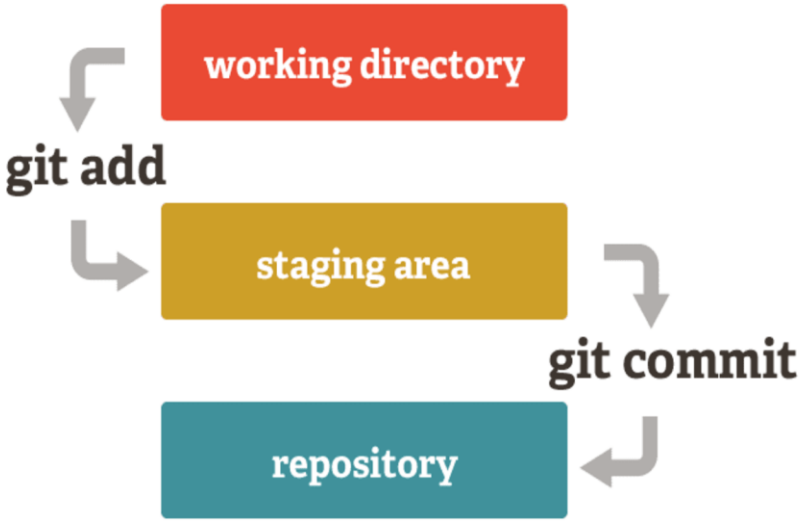
Git felépítése



Forrás: https://blog.isquaredsoftware.com/2021/01/coding-career-git-usage/

Git felépítése

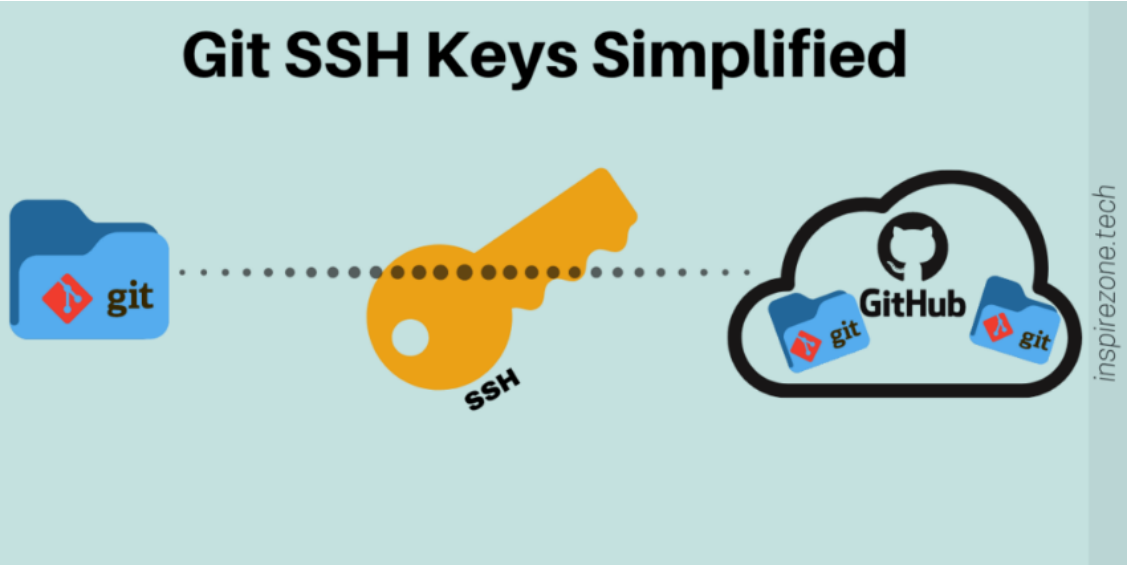
- **Working copy (/directory)**
  - azok a fájlok összesége amiket verziókövet a rendszer (.git)
- **Staging area**
  - azok a változások melyek a következő commit esetében érvényesítve lesznek
- **Local repository**
  - helyi repo
- **Remore repository**
  - távoli repo



Forrás: https://res.cloudinary.com/practicaldev/image/fetch/s--Si7ksd-d--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https://cdn-images-1.medium.com/max/800/1%2AdiRLm1S5hkVoh5qeArND0Q.png

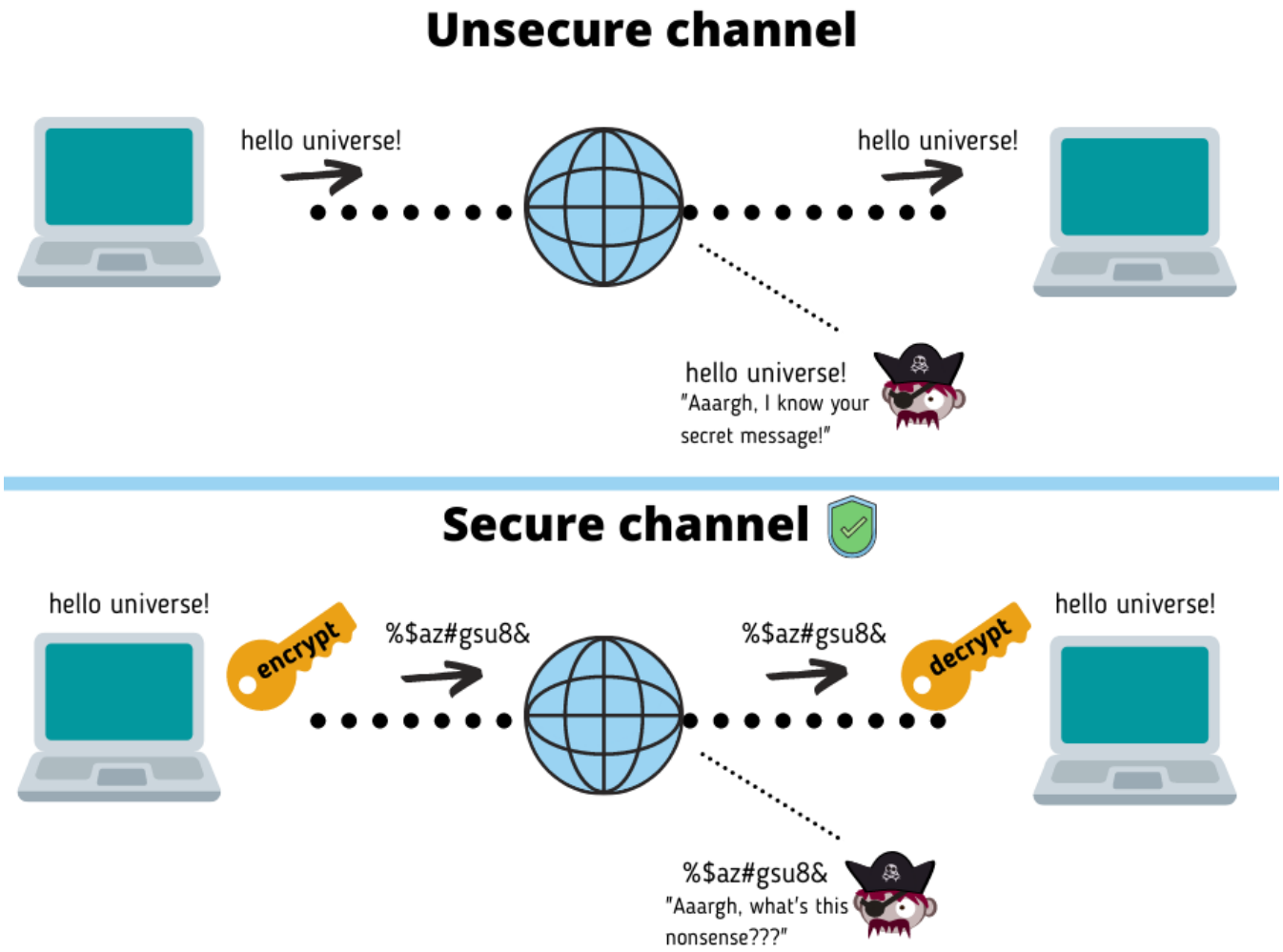
Repository-k közötti kommunikáció

- local repo és remote repo között (Github, Bitbucket stb.)
  - csak szinkronizáláskor szükséges kommunikáció
- HTTPS vagy SSH (mindkettő titkosított)
  - `git clone https://szolgaltato/felhasznalonev/repo-neve.git`
  - SSH esetében kulcspáros generálása: `ssh-keygen -t rsa`



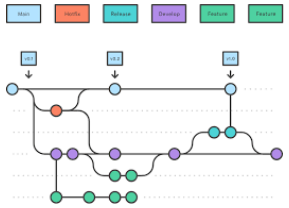
Forrás: https://inspirezone.tech/git-ssh-keys-simplified/

Repository-k közötti kommunikáció



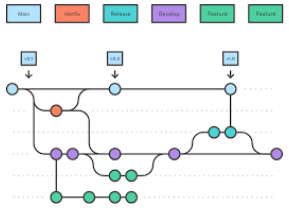
Forrás: https://inspirezone.tech/git-ssh-keys-simplified/

Gitflow



- Workflow model
  - Vincent Driessen, 2010, blogbejegyzés
- egy strukturált és **jól definiált munkafolyamat**ot határoz meg
- alapja a **különböző típusú ágak használata**, amelyek különböző fejlesztési szakaszokat képviselnek
  - Master
  - Develop
  - Feature
  - Release
  - Hotfix

Gitflow

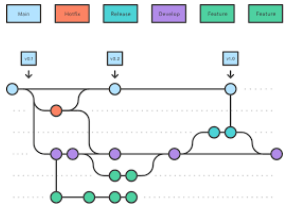


- **Master ág (master branch):** ez az ág a **stabil verziók**at tartalmazza, amelyek készen állnak a kiadásra

- **Develop ág (develop branch): a fejlesztés fő ága**, amelyben az aktív fejlesztési munka zajlik. Az új funkciók és fejlesztések a develop ágon vannak fejlesztve

- **Feature ágak (feature branches):** Minden **új funkciót vagy fejlesztés**t külön feature ágon fejlesztenek ki a develop ágból kiindulva. Amikor a funkció kész, a feature ág visszakerül a develop ágba

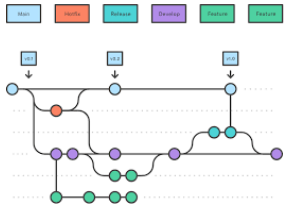
Gitflow

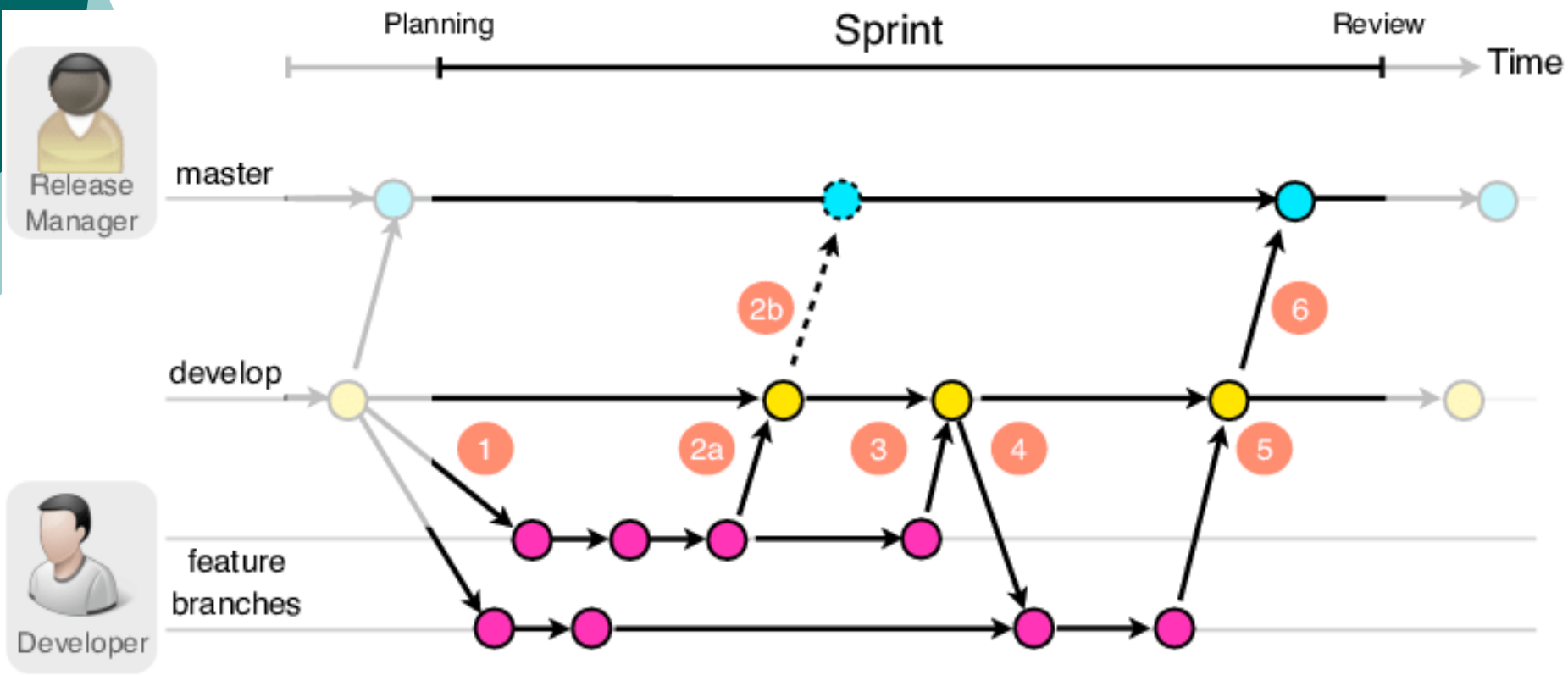


- **Release ág (release branches):** Amikor egy **új verzió kész** a kiadásra, létrehoznak egy release ágat a develop ágból, ahol a kiadásra vonatkozó utolsó finomítások és javítások történnek. A release ág végül össze lesz vonva a master ággal, és a kiadás létrejön

- **Hotfix ágak (hotfix branches):** Amikor **hirtelen és sürgős javítások**ra van szükség egy már kiadott verzióban, létrehoznak egy hotfix ágat a master ágból. Ez a javítás elvégzése után visszakerül a master és a develop ágba is

Gitflow





Ábra forrása:
Krusche, Stephan & Alperowitz, Lukas. (2014). Introduction of Continuous Delivery in Multi-Customer Project Courses. 36th International Conference on Software Engineering, ICSE Companion 2014 - Proceedings. 10.1145/2591062.2591163.

Mercurial

- Teljesen elosztott: minden fejlesztő teljes repo másolattal rendelkezik
- Egyszerű parancskészlet: pl. hg commit, hg pull, hg push
- Belső adatkezelés: nem használ hash alapú objektumtárolást, mint a Git, hanem egy stabilabb tárolási modellt, ami megakadályozza a korrupt repókat
- Közvetlenül támogatja a fájlszintű változásokat: fájlszintű revíziókat tárol, míg a Git teljes fákról készít pillanatképeket
- Több fejlesztői platform is támogatta: a Bitbucket korábban natívan támogatta a Mercurialt, de 2020-ban megszüntette
- Hookok és kiterjesztések: támogatja a testreszabható hookokat és pluginokat, de kevesebb lehetőséggel, mint a Git

SVN

- Központosított verziókezelő (CVCS)
- Egyszerűbb modell, mint az elosztott rendszerek
  - nincs mindenkinél teljes másolat
- Egyes fájlok külön-külön is letölthetők, nem kell az egész repót klónozni
- Jobb nagyméretű bináris fájlok kezelésére, mint a Git
- Nincs fejlett ágkezelés
  - branching, merging nehezebb
- Használata visszaszorulóban
- Régebbi vállalati rendszerekben és játékfejlesztésben (pl. Unreal Engine) használták elsősorban

VCS-ek összehasonlítás

	Git	Mercurial	SVN
Típus	Elosztott (DVCS)	Elosztott (DVCS)	Központosított (CVCS)
Branching, merging	Fejlett	Jó	korlátozott
Használat	Teljes repo elérhető offline	Teljes repo elérhető offline	Központi szerver kell
Adattárolás	Lokális és távoli	Lokális és távoli	Központi szerver
Alkalmazási terület	Nagy projektek, open source, startupok	Egyszerűbb fejlesztések	Régi vállalati projektek, játékfejlesztés

Felhasznált irodalom

- *Dr. Mileff Péter, Szoftverfejlesztés, Verziókövetés, Verziókövető rendszerek, jegyzet*
  - https://users.iit.uni-miskolc.hu/~mileff/szf/Verziokezeles_V5.pdf
- *Mark Groves: Introducing Gitversion control*
- https://www.atlassian.com/git/tutorials
- https://git-scm.com/docs/gittutorial

Köszönöm a figyelmet!