A verziókövetés

Alapfogalmak

Repository

- **Adatbázis neve**, ami a **verziókezelt fájlokat tárolja**.
- Egy mappa fájlokkal, amiben egy speciális rejtett **.git** mappa, ami a kezeléshez szükséges adatbázist tárolja.

Commit

- A fájlokról elmentett pillanatkép, amihez egy kommentet írunk, hogy éppen mit csináltunk.

Server/Origin

- Szerver, ami a **repository**-t tárolja és kiszolgálja a fejlesztőknek.
- A fejlesztők a saját gépükön rendelkezhetnek privát repository-val is, de a többi fejlesztőnek csak az lesz elérhető, ami a szerveren van.
- Mivel a Git elosztott rendszerű, ezért nem szükséges a repository-t publikálni egy másik szerverre.

Client

- Kliens gép, ami a szerverhez csatlakozik.

Working Set/Working Copy

- A fejlesztő helyi gépén tárolt változata a repository-nak.

Revision number

- Repository-ban tárolt változatokra ezzel tudunk hivatkozni.
- Git-ben ez egy hasító függvénnyel képzett érték, ami egyedileg és egyértelműen beazonosítja a változatokat.

Head

- Legutolsó revision number hivatkozó kifejezése.

Main/branch

- A main a fő ág, ebből az ágból bármelyik ponton készíthető al ág, amit branch-nek nevezünk.
- A branch létrehozása után külön változatként él tovább.
- A létrehozott branch-ek bármikor beintegrálhatóak bármelyik ágba, akár vissza a fő ágba is.
- Előnyük, hogy kísérletezhetünk vele, mint például új funkciók fejlesztésére.

Merge

- Két ág, branch összeolvasztásának folyamata.
- Általában nem igényel manuális beavatkozást.

Push

- Lokális változások feltöltése távoli repository-ba.

Tag

- Címkézés, például programverzió.

Fork

- Szerveren megosztott repository helyi másolata, amin mi dolgozhatunk.

Pull request

- Távoli repository változásainak letöltése és mergelése.

Diff

- Változások megtekintése

Reset

- Nem commitolt módosítás eldobása.

Stash

- Módosítás mentése átmeneti tárolóba.

Checkout

- Branchek közötti váltás.

Fetch

- Távoli repository változásainak letöltése.

A GIT sajátosságai

.gitignore

- Ez egy rejtett dot file, ami nem kerül feltöltésre a repository-ba, csak lokálisan van a fejlesztő gépén.
- Ebben a fájlban megadhatunk „szabályokat”, amivel megmondhatjuk, hogy mit ne töltsön fel.
- Például dot fájlok, konfig fájlok, csomagok.

Támogatottság

- Különböző operációs rendszerre és fejlesztői eszközre kiterjed, például fejlesztői környezetekben is megtalálhatóak, mint például a Visual Studio, Jetbrains, stb.

Problémák/konfliktusok kezelése

- Konfliktus akkor keletkezik, amikor a verziókezelő rendszer nem tud dönteni a változások sorsáról.
- Akkor következhet be, amikor egyszerre többen dolgoznak ugyanazon a projekten.
- Ilyenkor manuálisan kell megoldani a konfliktusokat, hogy melyik sor az, ami kell.
- A konfliktusokat parancssoron és erre kialakított szoftvereket is tudunk használni:
    - GitKraken
    - Github Desktop
    - Sourcetree

Fejlesztési modellek (Gitflow, Trunk-Based, Linux Kernel)

Gitflow

- Git munkafolyamat, ami lehetővé teszi a fejlesztők számára, hogy szabályozzák a fejlesztési folyamatot különböző ágak között.
- **Fő elemei:**
    - **Master ág:** Tartalmazza a stabil, befejezett és tesztelt kódokat.
    - **Develop ág:**
        - Minden új funkció, javítás vagy változtatás a develop ágban indul, majd a tesztelés után összeolvasztják a master ággal.
    - **Feature ág:**
        - Új funkciók a develop ágba kerülnek.
        - **Soha nem kerül interakcióba a main ággal.**
    - **Release ág:**
        - Új verzió előkészítésének ága, ami kiadásra van szánva, amit mergelünk a main ággal és egy verziószámmal látjuk el.
    - **Hotfix ág:**
        - Hibajavításokat tartalmaz, amik összeolvasztásra kerülnek a master ággal, majd a develop ággal.

Gitflow folyamata

1. Létrehozunk egy develop ágat a main ágból.
2. Létrehozunk egy release ágat a develop ágból.
3. Létrehozunk egy feature ágat a release ágból.
4. Ha egy funkció elkészült, akkor a develop ágba mergelődik.
5. Amikor a release ág elkészült, akkor mergelődik a develop és a main ággal.
6. Ha a main ágban hibát találunk, akkor a main ágból létrehozunk egy hotfix ágat.
7. Ha a hotfix elkészült, akkor mergelődik a develop és a main ágba.

Trunk-Based

- Minden fejlesztő lokálisan és önállóan dolgozik a projektjén, majd a változásait legalább naponta egyszer mergeli a main ágba.
- A mergenek függetlenül attól kell történnie, hogy a funkcióváltások vagy kiegészítések befejeződtek-e vagy sem.
- Kevesebb ágat tartalmaznak és kevesebb a merge conflict.
- Alapfeltétele a CI/CD, hogy automatizálva legyen minden.

Gitflow vs Trunk-based

- Gitflow-t érdemes komplexebb projektekben használni, ahol több ág van.
- Trunk-based kisebb team számára kedvezőbb.
- Összességében a választás függ a projekttől, team-től.

Linux Kernel

- Fejlesztők közös fejlesztési erőfeszítéseinek eredménye.
- Kis létszámú maintainerekből álló team, akik felügyelik a hozzájárulásukat és biztosítják, hogy azok összhangban legyenek a projekt általános céljaival.
- **Linux Kernel folyamata:**
    - **Ötlet/Idea:** A fejlesztőnek támad egy ötlete.
    - **Fejlesztés/Development:** Lefejleszt egy új funkciót és azt alaposan teszteli is.
    - **Benyújtás/Submission:** A fejlesztő benyújtja a kódot, hogy a maintainer felülvizsgálja.
    - **Értékelés/Review:** A maintainer ad egy visszajelzést, hogy esetleg min kellene még változtatni, amíg az nem felel meg.
    - **Tesztelés/Testing:** A maintainer és más fejlesztők is tesztelik a kódot, hogy biztosan stabil-e.
    - **Integráció/Integration:** Ha a kód átment a teszten és a maintainer elfogadta, akkor beintegrálják a kernel kódbázisába.
    - **Release:** Új kernel verzió nyilvánosságra kerül és ezzel a ciklus kezdődik elölről.