**1. modul**

Introduction to Basics

GIT

git, git bash

# Bevezetés

Képzeld el, hogy készítesz egy nagy szoftvert egy csapatban, a másik hat csapattal együtt és ugyanazon a programon dolgoztok. Képzeld el, hogy a többieknek is szükségük van az általad megírt programra és neked is a többiek munkájára. Képzeld el, hogy a megrendelő folyamatosan újabb funkciókat szeretne látni és neked ezeket határidőre részletekben kell szállítani. Képzeld el, hogy néha egy-egy hibát bejelentenek a programmal kapcsolatban és neked ezt egy megadott következő verzióban ki kell javítani.

A fenti rövid leírásból látszik, hogy egy nagyobb szoftvernél, ahol több százezer sornyi kódról beszélhetünk nem könnyű biztosítani, hogy a fejlesztők egyszerre rendezetten együtt tudjanak dolgozni úgy, hogy ne alakuljon ki káosz.

Az eszköz amit ennek megoldására meg fogsz ismerni az a git. A git-et [Linus Torvalds](https://hu.wikipedia.org/wiki/Linus_Torvalds) alkotta meg, mert nem volt megelégedve a régebbi verziókezelő szoftverekkel (pl.: SVN) . Ma a git a legnépszerűbb verziókezelő szoftver. Legismertebbek: GitHub, GitLab

# Miért?

* több verzió tárolása
* visszaállás korábbi verziókra
* kód könnyű megosztása
* közös munka
* változások nyomon követése
* backup *(biztonsági mentés)*
* conflict kezelés *(ugyanazon sort módosították többen egyszerre a fájlban)*

Olyan projekteknél *(program forráskódjának írásánál)* használatos, amelyen huzamosabb ideig több fejlesztő dolgozik párhuzamosan. Ezentúl minden új projektet, aminek a verzió kezelésére git-et használunk **repository**-nak fogunk hívni. A repository-k között különbséget teszünk, attól függően, hogy hol találjuk. Azokat a repository-kat, amiket saját gépünkről érjük el, **local repository**-nak hívjuk, azokat amelyeket valamilyen távoli helyről például a GitLab, GitHub, Bitbucket -ról érjük el **remote repository**-nak.

# Első lépések

Telepítsük a git-et, amit ezen a linken érsz el: <https://git-scm.com/>

Miután sikerült feltelepíteni, szükséges néhány konfigurációt elvégezni

Felhasználónév

|  |
| --- |
| git config --global user.name 'NEVED' |

E-mail cím (amelyikkel a GitLab-on regisztráltál)

|  |
| --- |
| git config --global user.email 'EMAIL\_CÍMED' |

# Mit ne?

Ne verzió kezeljük azokat a fájlokat, amelyek bármikor előállíthatóak a többi segítségével. Pl.: fejlesztő környezethez, programhoz tartozó fájlok vagy binárisok

# Alapfogalmak/parancsok

* commit: fájl vagy fájlok egy elmentett állapota, verziója
* branch: commitok összessége
* push: lokális változtatások, új dolgok bevezetésének feltöltése a szerverre
* pull: szerveren tárolt fájlok letöltése a lokális repository-ba
* status: megmutatja mi az állapota a lokális repository-ban lévő fájloknak
* staging area/staged files: lokális fájlok, amelyek ki vannak választva a következő commithoz

# 

# 

# 

# 

# Adatfolyam



# Új repo

Ahhoz, hogy egy teljesen új projektet készíts nyiss egy új mappát bárhol a gépeden. Miután elkészítetted a projekt mappát lépj bele és jobb klikk után nyiss egy új Git Bash-t. A Git telepítése után automatikusan elérhető lesz ez az opció. Vagy nyiss egy Terminált/Command Line-t.

**Menjünk végig az alábbi lépéseken:**

Inicializáljuk egy új üres repository-t *(készít egy új rejtett mappát .git néven, ebben lesz minden a repository-nkról)*:

|  |
| --- |
| git init |

Inicializál egy új üres repository-t *(készít egy új rejtett mappát .git néven, ebben lesz minden a repository-nkról)*.

Futassuk le az alábbi parancsot ha github-ot szeretnénk használni, hogy beállítuk a main branch-et, amin dolgozni fogunk: *(később jobban meg lesz még magyarázva ez a parancs)*

|  |
| --- |
| git branch -M main |

Készítsünk egy fájlt, hogy ne csak egy üres projektünk legyen és egyúttal írjunk is bele valamit:

|  |
| --- |
| echo 'This is my first project' > README.MD |

Egyébként íratlan szabályként, szokásként a README.md szokta tartalmazni a repository-ban a hasznos információkat, tudnivalókat.

A git feljegyzi, hogy milyen módosításokat hajtottunk végre a repository-n, esetünkben elkészítettük az első fájlt. Jelöljük meg a verziókövetést a fájlhoz:

|  |
| --- |
| git add . |

Commit-al készítünk egy új verziót a **-m** után megadott üzenettel *(ami általában egy rövid leírás szokott lenni, hogy mit tettünk az adott verzióba, mit tartalmaz, mi volt a módosítások célja)*, azaz innentől lesz egy olyan verziónk, amit később bármikor elő tudunk venni és csak a README.MD fájl aktuális verzióját fogja tartalmazni.

|  |
| --- |
| git commit -m 'created README.MD' |

Készítsünk új mappát, bele fájlokat (tehetünk ide képeket is, bármilyen fájlt) illetve módosítsuk a meglévőt README.MD-t is.

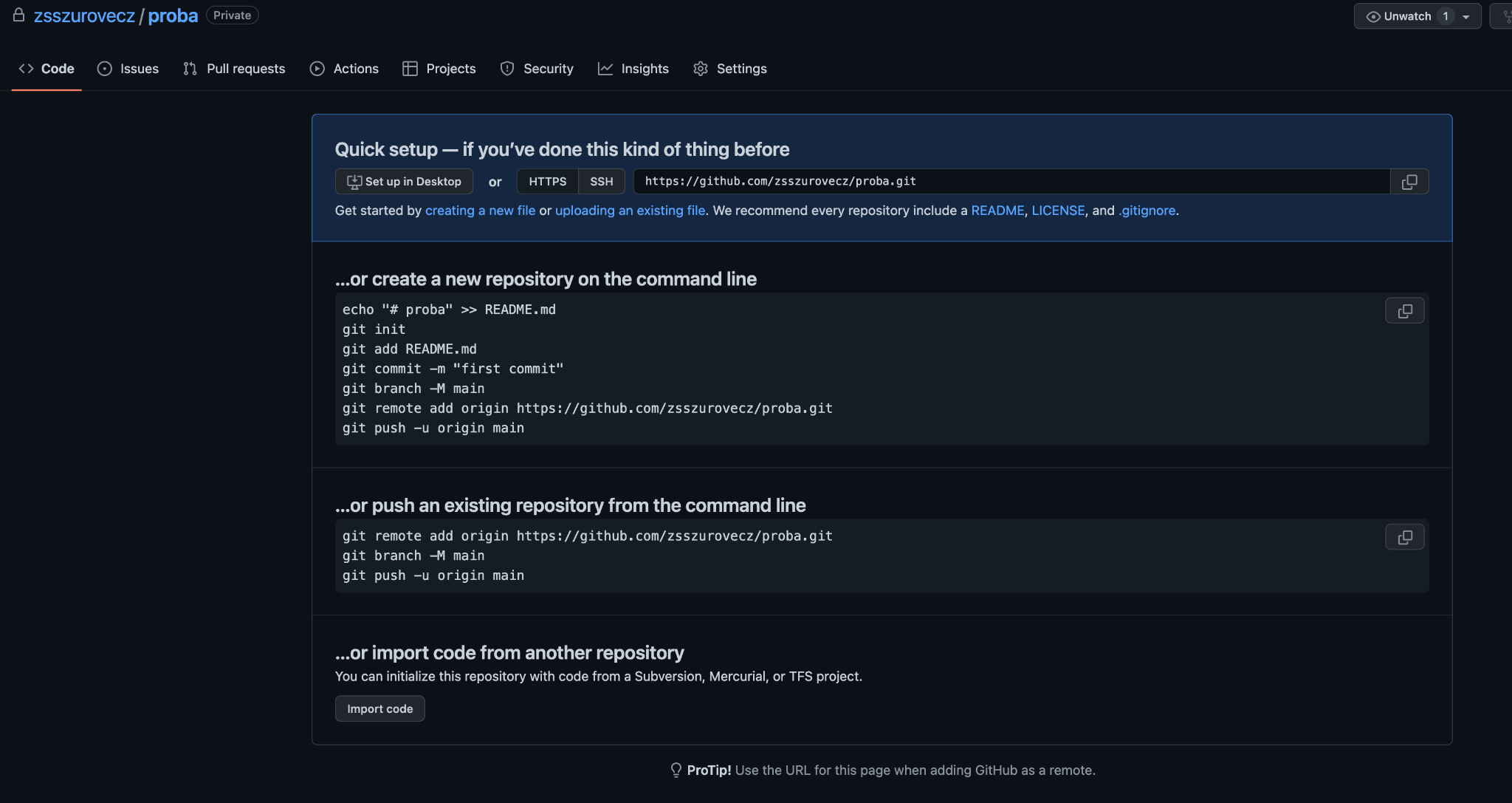
Adjuk ki a következő parancsokat, hogy létrehozzuk a következő verziót, a lépések között folyamatosan figyelve, hogy mi a státusz:

|  |
| --- |
| git status  git add .  git status  git commit -m ‘new files added and README.MD modified’  git status |

Most, hogy lokálisan rendben vagyunk, készítsünk egy remote repository-t, hogy ne csak a gépünkön legyenek meg a verziókezelt fájlok, hanem egy távoli szerveren is. Lépjünk be a Githubra *(regisztráljunk ha még nem tettük)*

* a “New” gombra kattintva tudjuk elkezdeni a létrehozást
* adjunk nevet a repository-nak
* allítsuk be, hogy private vagy public repo-t akarunk
* többi részt hagyjuk változatlanul és kattintsunk a “Create repository” gombra

Az alábbihoz hasonló képernyőt kell lássunk:



Ahhoz, hogy a saját gépünkről feltölthessük a projektünket, be kell állítanunk a remote repository elérési útvonalát, amiben a GitHub segít is nekünk, pusztán ki kell másolni a ***git remote add origin https://git...***-el kezdődő sort a felugró ablakból és lefuttatni a gépeden.

Ha ez megtörtént, ezáltal összekapcsoltuk a gépen lévő és a távoli szerveren lévő repositoryt, most már látják egymást. (és emlékeznek is, tehát ezt csak első feltöltés esetén kell lefuttatni). Most indítsuk is el a feltöltést:

|  |
| --- |
| git push -u origin main |

Ezzel a lokális és a szerver repository-ja szinkronizálva lett.

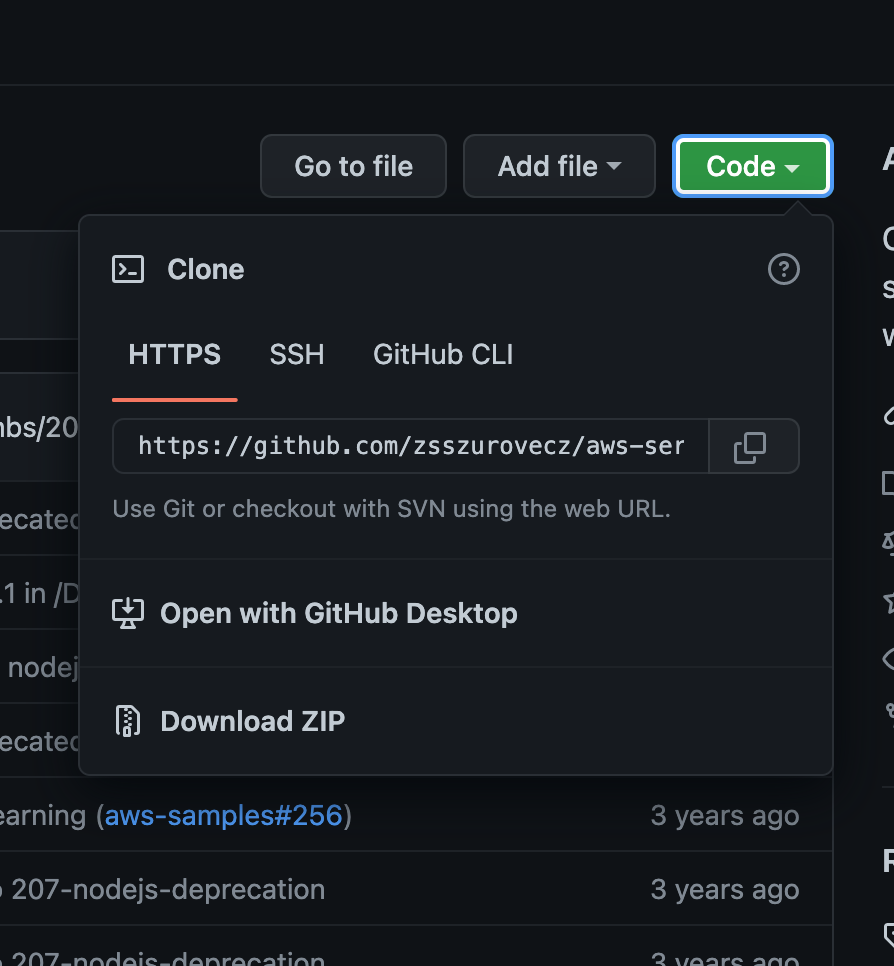
# Meglévő repo

Leggyakrabban már meglévő repository-ban folyó munkához kell becsatlakozni. Tehát első lépésként a remote repository-t kell a saját számítógépünkre letölteni, hogy ott hozza létre a lokális repositoryt.

Navigáljunk oda, ahova le szeretnénk tölteni a projektet, majd adjuk ki a

|  |
| --- |
| git clone https://gitlab.com/FELHASZNALO/A\_REPOSITORYM.git |

parancsot. Hogy mi a pontos *clone* utáni rész azt egyébként a Github/Gitlab felületén az adott repository-ban meg lehet nézni és ki lehet másolni:



Az így letöltött projekt teljes értékű repository, mely már ugyanúgy tartalmaz *.git* mappát mint amikor mi hoztuk létre a *git init* segítségével. Most már szabadon hozhatunk létre vagy módosíthatunk fájlokat, azokat a git add . / git commit / git push parancsok segítségével hozzá tudjuk adni a meglévő kódbázishoz.

Későbbiekben természetes lesz, hogy mások is push-olnak a remote repository-ba és mi is szeretnénk letölteni a változtatásokat saját local repository-nkba. Erre a **pull** parancsot használhatjuk az alábbi módon:

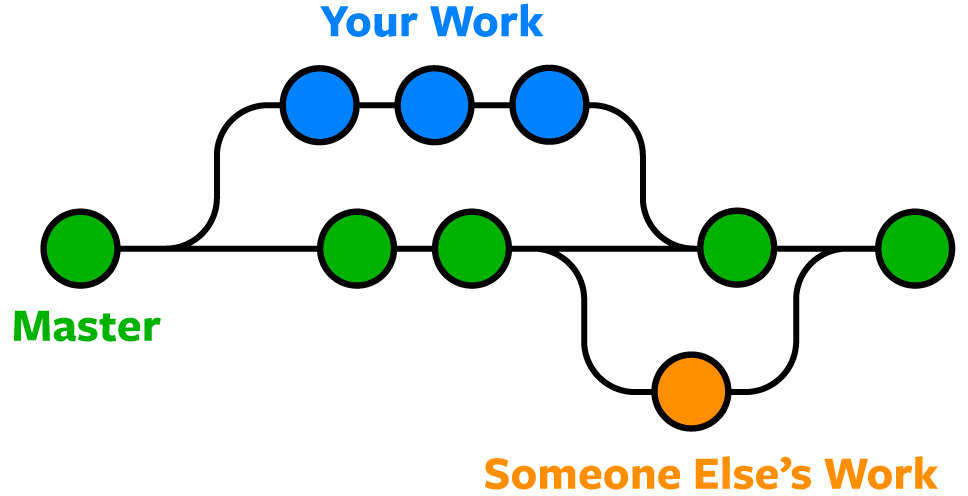
|  |
| --- |
| git pull origin main |

Megj.: ahol a *main* a branch neve, mely szabadon változtatható annak függvényében éppen melyiken dolgozunk.

Fontos, hogy mindig mielőtt ***push***-olunk először adjunk ki egy ***pull*** parancsot, így mindig megkapjuk a legfrissebb változtatásokat. Amennyiben a szerveren már újabb verziók is megjelentek, mint a mi legfrissebb verziónk, úgy a *push* hibát fog dobni, így védi meg a repository-t.

# Branch-ek

Programozásban gyakran előfordul, hogy két fejlesztő, esetleg néhány csapat eltérő részein dolgoznak a programnak, egymástól jól elszeparált részeket fejlesztenek. Ilyen esetekben a különböző funkcionalitások fejlesztésére külön branch-eket szoktak létrehozni a main-en kívül, például feature/a és feature/b. Amint a két branch-en elvégeztük a feladatokat, azokat vissza merge-eljük a branch-be. Ezt úgy kell elképzelni, hogy onnantól a változtatások láthatóak lesznek a branch-en.



A branch-elés megismeréséhez készítsünk egy új projektet, amiben egy README.MD fájl található. Futtassuk le az alábbi parancsokat:

|  |
| --- |
| mkdir branch-example cd branch-example git init echo 'Branching' > README.md git add . git commit -m 'created README.md  git push -u origin main |

Ezután elkészítünk két újabb fájlt, *bob.md* és *adam.md* néven, mindkettőt a saját *feature/bob* és *feature/adam* branch-én. Hozzuk tehát létre a branch-et majd váltsunk át rá:

|  |
| --- |
| git branch feature/adam git checkout feature/adam |

(Van lehetőség a két parancs összevonására is a ***git checkout -b feature/bob*** megcsinálja egymenetben a fenti két utasítást.)

Akkor hozzuk létre a fájlokat és commitoljuk fel:

|  |
| --- |
| echo ‘Adam’ > adam.md git add . git commit -m 'created adam.md'  git push -u origin feature/adam |

Ezt követően ugyanez a történet a bob.md-vel, *(csak előtte álljunk vissza a main-re)*:

|  |
| --- |
| git checkout main  git branch feature/bob git checkout feature/bob echo ‘Bob’ > bob.md git add . git commit -m 'created bob.md'  git push -u origin feature/bob |

Ezen a ponton kipróbálhatjuk, hogy válthatunk a branch-ek között és figyeljük a fájlrendszerben, hogy melyik branch-en milyen fájljaink vannak. Azt fogjuk látni, hogy az egyes branch-ek csak a rajtuk létrehozott fájlokat fogják tartalmazni, hiszen azokat csak oda commit-oltuk.

Tehát:

|  |
| --- |
| git checkout feature/adam |

esetében csak az adam.md látszik.

|  |
| --- |
| git checkout feature/bob |

Esetében csak a bob.md

|  |
| --- |
| git checkout main |

Esetében pedig a README.md

Az, hogy egyébként milyen branch-ek elérhetőek nem szükséges fejből tudnunk, van rá egy parancs, amely kilistázza nekünk a local és remote branch-eket:

|  |
| --- |
| git branch -a |

Az utolsó feladatunk, miután a két új "funkciót" elkészítettük, hogy vissza merge-eljük őket a branch-re. A merge-re hasonlóan a branch készítéshez két módszer létezik. A **merge** és a **rebase.**

## merge

Ezzel a módszerrel egy meglévő branch változtatásait egy másik branchre vihetjük át. Amennyiben voltak a két ágon eltérő commit-ok, úgy egy merge commit fog létrejönni, ha nincs ütközés.

|  |
| --- |
| git checkout main git merge --no-ff feature/adam -m 'merged feature/adam branch' |

A sorrendél érdemes megjegyezni, hogy először átálltunk a branch-re, ahova merge-elni akartunk, majd a merge parancsnál azt a branch-et adtuk meg, amelyiket a main-be akartuk merge-elni.

## rebase

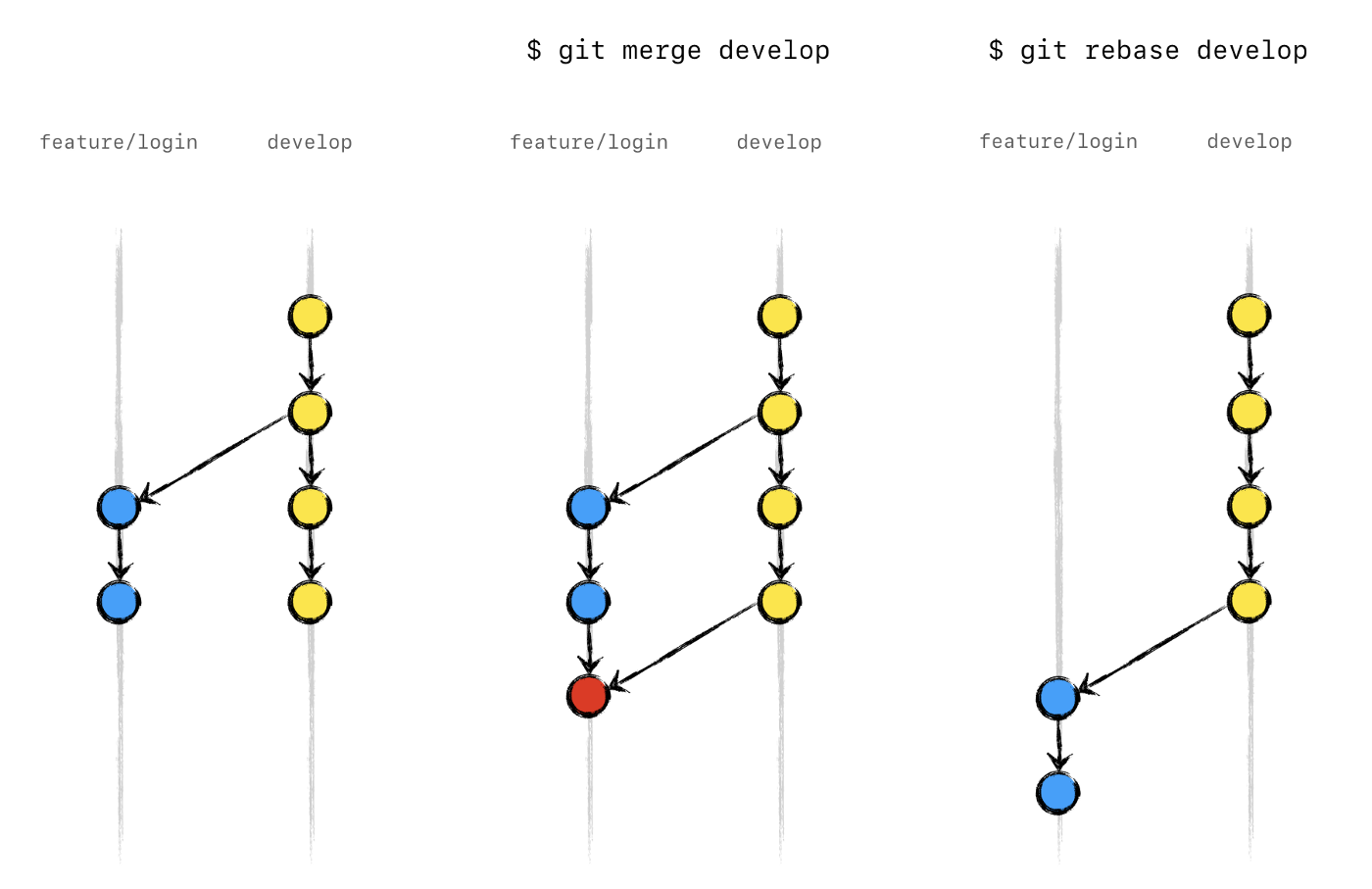
A merge-hez képest itt az irányok lesznek az eltérőek. A rebase esetében ugyanis azt mondjuk meg, hogy melyik branch változtatásait, commit-jait játssza újra az aktuális branch-en. Eközben az eredeti commit-ok eltűnnek és azok másolatai játszódnak újra esetünkben a main branch-en.

|  |
| --- |
| git checkout feature/bob git rebase main git checkout main git merge feature/bob -m 'merged feature/bob branch with rebase' |

Miután a két feature branch-en végzett módosításokat beépítettük a main ágra a branch-eket törölhetjük.

|  |
| --- |
| git branch -D feature/adam git branch -D feature/bob |

Íme egy ábra, ami a merge és rebase különbségét szemlélteti:



# Ignorált fájlok

Előfordul néha, hogy a projektekben megjelennek olyan fájlok is, amiket egyáltalán nem szeretnénk verziókezelni. Ezek a fájlok sajnos megjelennek majd a gépünkön, de mi mégis azt szeretnénk ha a git nem foglalkozna velük. Természetesen mint mindig most is van megoldás és nagyon egyszerű. A megoldás a .gitignore fájl, amit ezután verziókezelve fogunk tárolni a projektekben. Amennyiben létrehozunk egy új projektet, ebbe a fájlba kell felvennünk azokat a fájlokat, amiről azt szeretnénk ha a git nem foglalkozna velük. Ilyen fájlok lehetnek például bizonyos kódszerkesztő programok működéséhez szükséges fájlok. Tipikusan egy IntelliJ esetében például a .idea/ mappa és a project-name.iml fájl ilyenek. Más programok esetében más fájlok is megjelennek. Bármilyen programot is használunk a fejlesztéshez, nagyon fontos, hogy normálisan felvegyük az ignorálni kívánt fájlokat, hiszen ha ezt elfelejtjük, azzal csapattársainknak sok fájdalmat okozhatunk.

Pár példa a *.gitignore* fájl tartalmára:

* settings.xml,
* minden a gyökér könyvtárban lévő .iml kiterjesztésü fájl,
* target/, out/, bin/, .idea/, .settings/ mappák és bennük lévő fájlok a gyökér mappában,
* bármelyik mappában elhelyezkedő orig/ mappa,
* bármelyik mappában elhelyezkedő .bad kiterjesztésű fájl

|  |
| --- |
| setting.xml \*.iml target/ out/ bin/ .idea/ .settings/ \*\*/orig/ \*\*/\*.bad |

Figyeljük meg, hogyha nem tudjuk pontosan, hogy hogyan fogjuk nevezni a fájlt, csak a kiterjesztése szerint tudjuk, hogy ignorálni kell, akkor \*-ot használunk a fájl neve helyén, nem tudjuk, hogy melyik almappában lesz, akkor a mappa nevét \*\*-al helyettesítjük.

Próbáljuk ki, hogy néhány fájlt készítünk az ignorált fájlok, mappák közül és git status-al figyeljük, hogy a git nem foglalkozik velük, azaz nem ír ki róluk semmit, mint nem track-elt fájlok.

Az interneten találhatunk oldalakat például a <https://www.gitignore.io/> ahol a megadott eszközök után a weblap legenerálja nekünk az ignorálandó fájlokat tartalmazó .gitignore fájlt.

Ha mi magunk hozzuk létre a repository gyökerében, azt megtehetjük az echo > .gitignore paranccsal, majd egy tetszőleges szövegszerkesztővel megnyitva tudjuk szerkeszteni.

# Jó tanácsok

* összetartozó dolgokat commit-olj egyszerre
* gyakran commit-olj
* hasznos commit message-eket írj - derüljön ki belőle, hogy mi van a commit-ban
* használj branch-eket, ne mindig a main-en dolgozz
* ne commit-olj szintaktikailag hibás, nem futó kódot (és persze ne is push-old)

# Fontos lehet még

Nagyon hasznos és fontos parancs lehet még a későbbiekben a **git stash**. Valamint a **merge** **conflict**-ok feloldása. Ezekkel a tanfolyam egy későbbi pontján fogunk foglalkozni.

# Linkek

Játékos gyakorló oldal a git parancsok elsajátításához: <https://learngitbranching.js.org/>

Atlassian git tutorial: <https://www.atlassian.com/git/tutorials/setting-up-a-repository>

Git cheat sheet: <https://about.gitlab.com/images/press/git-cheat-sheet.pdf>

Hivatalos git doc: <https://git-scm.com/doc>

Legjobb grafikus felületek a git-hez:

* [GitKraken](https://www.gitkraken.com/)
* [GitHub Desktop](https://desktop.github.com/)
* [Sourcetree](https://www.sourcetreeapp.com/)

# Mellékletek





# Feladatok

1. Készítsünk egy új projektet, ahol bemutatkozunk, esetleg bemutatunk egy érdekes dolgot. A megoldást a README.MD fájlban közöljük *(a GitLab/Github ezt a fájlt jeleníti meg alapértelmezetten, amikor megnyitunk egy projektet)*. A bemutatkozó/bemutató projektünket több commit-ban készítsük el, szerepeljen/szerepelhet benne kép, videó, linkek, listák…etc.
2. Ebben a feladatban az lenne a cél, hogy lemásoljuk a git forráskódját egy saját repositoryba. Mint már korábban tárgyaltuk a git egy nyílt forráskódú szoftver, szóval az alábbi linken bárki számára elérhető minden része: [Git](https://github.com/git/git). Ha jobban megnézzük ez a forráskód egy ugyan olyan git repositoryban van tárolva, mint amilyeneket mi is létrehoztunk már GitLab-on. Klónozd le a Git forráskódját a gépedre, majd pushold fel egy általad készített privát repositoryba úgy, hogy közben töröld a repository teljes history-jának tartalmát. *(Ezt úgy tudod megtenni, hogy a clone-ozás után kitörlöd a .git mappát)*
3. A szakdolgozat verzióinak tárolása egy idő után elég sok fejfájást tud okozni. A következő feladatban szeretnénk kihasználni a Git előnyeit, és verziókezelten feltölteni egy távoli repositoryba.
   1. Inicializálj egy új Git repositoryt diploma-work néven.
   2. Készíts egy README.md file-t amiben leírod a nevedet, és a szakdolgozat témáját.
   3. Lépj be GitHub-ra, és készíts egy GITHUBUSERNAME-diploma-work nevű projektet, majd állítsd be távoli repositoryként azt. *(ahol értelemszerűen mindenki a sajátgithub userneve alapján nevezze el)*
   4. Commitold és pushold a változtatásokat create readme néven.
   5. Hozz létre egy új file-t document.txt néven, ezek után egy általad kiválasztott témában írj néhány mondatot.
   6. A dokumentumot bővítsd még párszor, és minden bővítés után commitold be, pushold fel a változtatásokat. Ha végeztél a bővítéssel pushold fel a commitokat a távoli repository-ba.
   7. Tegyük fel, hogy egy év eltelt és át kell dolgoznod a szakdolgozatot. Készíts egy új branch-et *szakdolgozat2022* néven.
   8. A létrehozott branch-en a dokumentum-ban módosíts pár sort. (írj át valamit, törölj valamit illetve adj is hozzá új tartalmat)
   9. Commitold és pushold fel a branchedre a módosításokat
   10. Merge-eld be a branch-et a main-be, hogy a módosítások ott is szerepeljenek *(itt használhatsz git bash-t vagy a github felületen a Pull request lehetőségét)*

Miután végeztél a feladatokkal, nyisd meg a projectedet Github-on, és néz meg a commit historyt. Kövesd végig a folyamatot, hogy mik történtek, és azok helyesek-e.

+1: csinálj új branch-et *szakdolgozat2023* néven, csináld meg a g-j pontokat úgy, hogy ha git bash-t használtál akkor most használj pull requestet ha pedig PR-t használtál akkor használj git bash-t a branch merge-hez.