A GIT verziókövető rendszer

**Tartalomjegyzék**

[**A GIT bemutatása**](#_nlimodq7vrmc) **2**

[Jellemzők](#_2r1xr1lx2sy) 2

[**Verzió követő rendszerek evolúciója**](#_69ps3l3krje8) **3**

[Local VCS - helyi verzió követő rendszer](#_9jdpjw7swv3u) 3

[Centralized VCS - központosított verzió követő rendszer](#_vjaongggqlp1) 4

[Distributed VCS - elosztott verzió követő rendszer](#_n7dksi3arnlw) 5

[**A GIT használata**](#_szy10kszta7z) **6**

[Telepítés és beállítás](#_9d3p6mj2qmtk) 6

[Munkaterület létrehozása](#_jas24xs5qyfd) 7

[GIT add - fájlok hozzáadása](#_j6i8csibv06i) 7

[GIT commit - módosítások mentése](#_jrye5jwwn7p) 8

[GIT push - változások feltöltése a szerverre](#_cqfoh95gulo6) 9

[GIT pull - változások letöltése a szerverről](#_es622up9ow3p) 10

[GIT workflow - az összes fontos lépés egyben](#_jcek5o5v5xyt) 10

[Github - a jó barát](#_7kjfw783cnxg) 11

[Branch - ágak](#_l38d005xqhgc) 12

[Két branch egyesítése](#_wpzpch8v49y5) 14

[Egy új Github tároló beállítása](#_i65ubgzi6v0h) 14

[**Összefoglalás**](#_m3h63zeyffwv) **16**

# 

# 

# A GIT bemutatása

A Git egy nyílt forráskódú, elosztott verziókezelő szoftver, vagy másképpen egy szoftverforráskód-kezelő rendszer, amely a sebességre helyezi a hangsúlyt. A Gitet eredetileg Linus Torvalds fejlesztette ki a Linux kernel fejlesztéséhez. Minden Git munkamásolat egy teljes értékű repository teljes verziótörténettel és teljes revíziókövetési lehetőséggel, amely nem függ a hálózat elérésétől vagy központi szervertől.

Számos nagy volumenű projekt használja jelenleg a Gitet verziókezelő rendszerként; a legfontosabbak ezek közül: Linux-rendszermag, GNOME, Samba, X.org, Qt, One Laptop per Child core development, MediaWiki, VLC media player, Wine, Ruby on Rails és az Android platform. A Git karbantartásának felügyeletét jelenleg Junio Hamano látja el.

A „git” nevet Linus Torvalds némi iróniával a brit angol szleng kellemetlen személyt jelentő szavából eredezteti. „Egy egoista szemétláda vagyok, és minden projektemet magam után nevezem el. Először volt a Linux és most a git.” Ez az önmarcangoló humor valójában csak fikció, hiszen Torvalds valójában külső nyomásra nevezte el a Linuxot maga után. A hivatalos Git wiki számos magyarázatot ad az elnevezésre, mint pl. „Global Information Tracker” (globális információkövető).

## Jellemzők

A Git rendszerét a BitKeeper és a Monotone ihlette. Eredetileg alacsony szintű verziókövető rendszernek tervezték, amelyhez más programok adják a felhasználói felületet, pl. a Cogito vagy az StGIT. Mindazonáltal a Git projekt azóta teljes értékű revíziókövető rendszerré szélesedett, amelyet már közvetlenül is lehet használni.  
Az utóbbi időben de-facto szabvánnyá kezd válni, olyan óriások váltottak a használatára a saját rendszerükről mint a Microsoft vagy a Google. Rendkívül rugalmas, megfelelően biztonságos és hibatűrő rendszert képez.

## 

## 

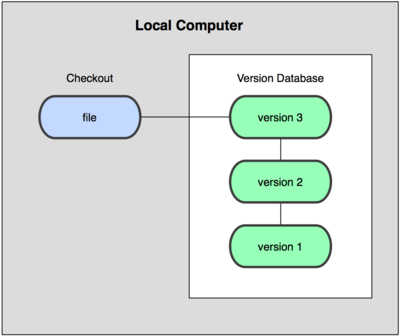
# Verzió követő rendszerek evolúciója

## Local VCS - helyi verzió követő rendszer

A helyi verziókövetésnek a legegyszerűbb formája a fájlok különböző verzióinak különböző mappákba mentése. Ez amellett hogy nagyon időigényes, rendkívül nehéz követni benne a hibákat és összehasonlítani a verziókat.

Fejlettebb megoldás ha adatbázisban tartjuk nyilván a változásokat, viszont ha megsérül a helyi tároló akkor ebben az esetben is elvesznek adataink.

Az alábbi ábra a helyi adattárolás modelljét mutatja be:



## 

## 

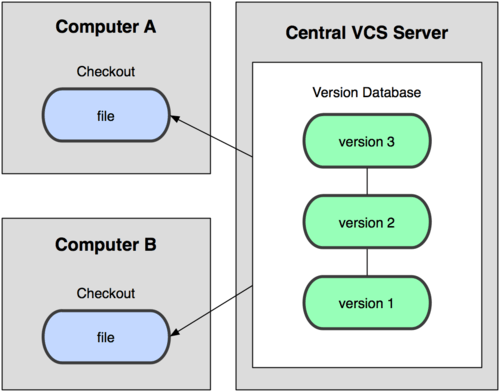
## Centralized VCS - központosított verzió követő rendszer

Ebben az esetben nagyobb az adatok biztonsága, mivel a változások egy központi szerveren vannak nyilvántartva. Itt viszont a fájlok különböző verziói nincsenek meg az összes felhasználó gépén, így a szerver kiesése esetén ugyan elméletben lehetséges a teljes helyreállítás, de az összes felhasználó által tárolt adatra szükség van hozzá. Ilyen például az SVN (Subversion), melyben nem tartalmazza az összes kliens az összes verziót, csupán a változások jegyzékét.

A GIT jellegű rendszerek megjelenése előtt ez a modell volt a döntő. Saját szerveren is gyakran futtatták az SVN-t. Képviselője például a Tortoise SVN volt. Egészen a mostani időkig használták a Microsoft termékek is, de a legújabb verziókban szintén átálltak a GIT -re.

Itt a teljes fájlkészeletet nem ismerik meg az egyes kliensek, mindenkinél csak azok a fájlok vannak amelyekkel dolgozik.

Az ábrán ennek diagramja látható:



## Distributed VCS - elosztott verzió követő rendszer

Az elosztott rendszerek, mint a Git, Mercurial, Bazaar vagy Darcs, más módszert alkalmaznak mint az előbbiek. A klienseknél nem csak az utolsó pillanatfelvétel található meg a fájlokból, hanem az egész történek. Ha bármilyen szerver kiesés is történik, mivel minden kliens a teljes tartalom birtokában van ezért csak felmásolják az új szerverre az adatokat és folytathatják a munkát. Minden checkout teljes adatmentést tartalmaz.

Az alábbi ábra az elosztott verzió követést mutatja be:

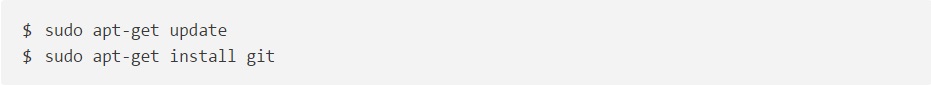


# 

# A GIT használata

## Telepítés és beállítás

A GIT telepítését és használatát Ubuntu rendszeren fogom bemutatni. Telepítéshez az apt nevű csomagkezelő rendszert fogjuk haszunálni. Először frissítsük a csomagok listáját, majd az install paranccsal kezdjük meg a telepítést:



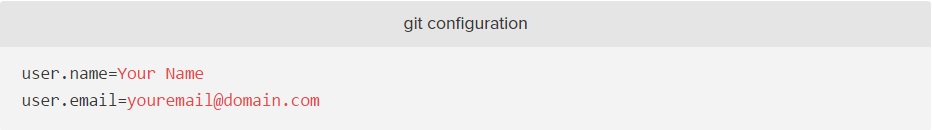
Ezután állítsuk be a felhasználó nevét és email címét akinek a nevében szeretnénk használni a GIT -et:



A helyes beállítást a következőképpen kérhetjük le:

k3.jpg

És ez az eredménye:



Ezzel a telepítés és beállítás befejeződött, megkezdhetjük a használatot.

## Munkaterület létrehozása

Mivel fájlok verzióinak követése a célja, ezért a GIT munkaterülete a fájlrendszer. Először hozzunk létre egy mappát, amit szeretnénk követni. Válasszunk egy tetszőleges helyet a fájlrendszerben és hozzunk létre ott egy mappát, majd abban egy fájlt. Ezután az alábbi példa szerint inincializáljuk a mappában a gitet. Ennek hatására létre fog jönni egy .git nevű almappa. Ebben tárolja a verziókezelő az egyes fájlok módosult verzióit.

mkdir -p git/testing ; cd ~/git/testing

touch file

git init  
Initialized empty Git repository…

## GIT add - fájlok hozzáadása

A git add parancs szolgál arra, hogy a meglévő fájlokat hozzáadjuk a verziókövetéshez. Amíg ezt nem tesszük meg, a git nem vesz tudomást az adott fájlokról, nem fogja őket követni - figyelni. Az add parancs esetén a pont (.) azt jelenti hogy az adott mappában minden fájlt rekurzívan, azaz ha több mappa mélységben helyezkedik el akkor is hozzáadunk a figyelt fájlok listájához.

git add .

Ennek a parancsnak nincs kiemenete, azaz a git nem informál arról hogy milyen fájlokat adott hozzá a listához. Később, amikor commit-olni fogjuk a változtatásokat kapunk majd egy listát erről is.

Mindig le kell futtatni az add parancsot, ha új fájlok kerültek a munkaterületre, hogy a figyelő listára felvegyük őket.

Amikor egy fájlt hozzáadunk a githez, akkor először egy köztes területre, az úgynevezett stage -re kerül. Ez tartalmazza a módosult vagy frissen hozzáadott fájlokat.

## GIT commit - módosítások mentése

A git commit parancs hatására a stage -ről átkerülnek a fájlok a .git könyvtárba és készek lesznek a feltöltésre. Ez nem tölti fel a fájlokat semmilyen szerverre, csupán a helyi gépen menti a módosításokat.

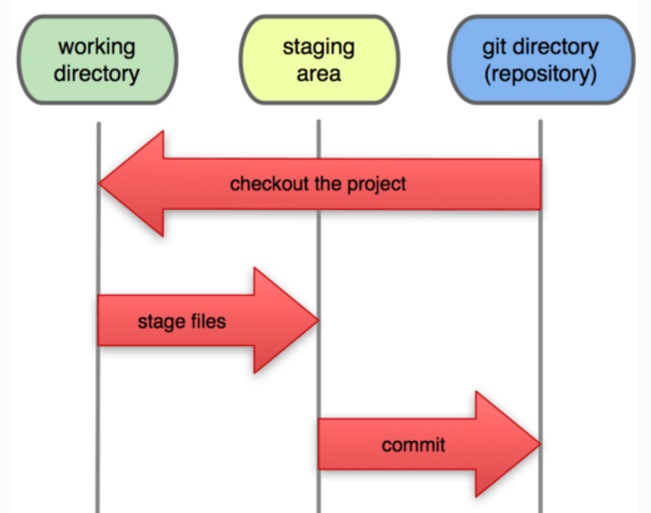
git commit -m "Initial Commit" -a  
[master (root-commit) 1b830f8] initial commit  
 0 files changed  
 create mode 100644 file

A fenti példában láthatjuk hogy több úgynevezett kapcsolót is használunk a git commit parancshoz:

1. -m: ez a message rövidítése, utána kell megadni a commithoz tartozó üzenetet
2. -a: ez pedig az all, azaz ennek hatására az összes fájtl hozzáadjuk a githez ami a stage -en van. Ha csak egy fájlt szeretnénk hozzáadni akkor itt a fájl elérési útját kell megadni a -a helyett.

A parancs kimenete is jó látható a példában. Ez arról árulkodik, hogy initial commit, azaz ez volt a tároló első commit -ja. Ezenkívül listázza hogy 0 fájl módosult és egy fájl lett hozzáadva a githez, file néven.

Az add és a commit működése:



## GIT push - változások feltöltése a szerverre

Mit sem ér a git, ha nincs hozzá távoli szerverünk, ahová fel tudjuk tölteni a fájlok verzióit. Ez lehet saját szerver is, ha telepítjük és beállítjuk valamelyik git szervert, mondjuk a gitlab -ot. Én most az ingyenes olnline szolgáltatásként igénybe vehető gitlab szolgáltatáson keresztül fogom megmutatni ezt a képességet.

Először is állítsuk be a szerveren található tároló címét:

git remote add origin ssh://git@git.domain.tld/repository.git   
git remote -v  
origin ssh://git@git.domain.tld/repository.git (fetch)  
origin ssh://git@git.domain.tld/repository.git (push)

A prarancs kimenetét is meghagytam a példában. A második sorban az url -ben be kell helyettesíteni a saját github fiókunk nevét és azon belül a tároló nevét is. Érdemesebb a könnyebben beállítható http protokollt használni az ssh helyett. Például ha a fiókunk neve okoska és a tárolónak oksoka-munkaja, akkor így fog kinézni az url http esetén:

git remote add origin https://github.com/okoska/okoska-munkaja.git

Amint az első parancsnál látható, a git remote -v paranccsal ki tudjuk listázni a szervercímeket. A fetch az olvasást - letöltést jelenti, a push a feltöltést. A kettő lehet eltérő is, most mi ugyanazt használjuk ahogy a legtöbbször a valóságban is.  
A konkrét feltöltést a git push paranccsal tudjuk megkezdeni. Ha sikeres volt a feltöltés akkor részletes riportot kapunk róla, ha nem akkor részletes hibaüzenetet:

git push origin master  
Counting objects: 4, done.  
Delta compression using up to 2 threads.  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (3/3), 266 bytes, done.

Az üzenet tartalmazza hogy melyik ágra (később tárgyaljuk), mennyi adatot töltöttünk fel. Azt olvassuk benne hogy delta compression. Ez arra utal hogy az adatok tömörítve utaznak a kliens és a szerver között. Az egyes commit -ok külön azonosítót kapnak, ez áll az üzenet végén a master -> master rész előtt. A példában a 0e78fdf ről az e6a8ddc commit -ra frissült a szerver.

## GIT pull - változások letöltése a szerverről

Ha le szeretnénk tölteni a távoli szerverről azokat a változtatásokat, amelyek még nincsenek meg a helyi tárolónkban, akkor a git pull parancsát tudjuk használni.

git pull

A parancs kimenete egy riport, amely tartalmazza, hogy milyen commit-ok érkeztek a szerverről.

## GIT workflow - az összes fontos lépés egyben

A GIT-el való munka során az egyes parancsokat szinte mindig ebben a sorrendben adjuk ki.

A forgatókönyv a következő:

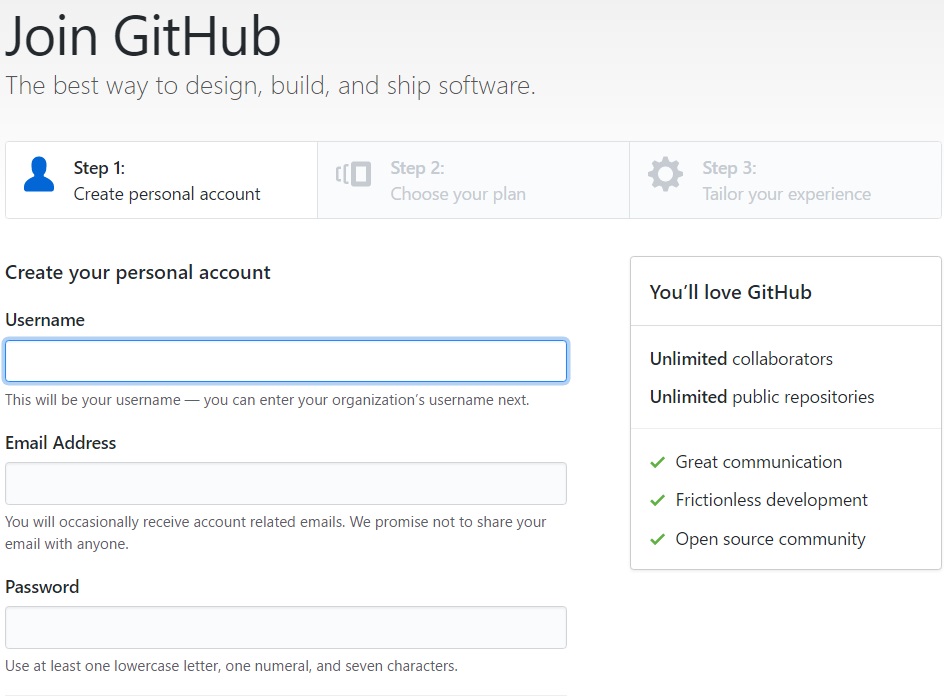
1. Módosítottunk vagy létrehoztuk fájlokat a helyi gépen.
2. Hozzáadjuk őket a stage-hez.
3. Kommit-oljuk a változtatásokat.
4. Letöltjük a módosításokat a távoli szerverről.
5. Feltöltjük az összes módosítást a távoli szerverre.

A parancsok sorrendben:

git add .  
git commit -m “a módosítások leírása”  
git pull  
git push

## Github - a jó barát

A github egy online szolgáltatás, ami git szervert biztosít számunkra. Előnye hogy nem kell megvenni a hardvert és beállítani a saját git szerverünket, mondjuk egy gitlab -ot. Egy gyors regisztrációt követően használatba is vehetjük Hátránya, hogy az ingyenes verzió esetén a tárolóink nyilvánosak lesznek. Jelenleg havi $7 a díja ha privát tárolókat szeretnénk, ami megközelítőleg 2 100 Ft -nak felel meg. A regisztráció a <http://github.com> címen érhető el:



Amikor push parancsot adunk ki akkor a git meg fogja kérdezni a felhasználó nevünket és a jelszavunkat, amit a regisztrációkor adtunk meg. Ha a jelszó megadása előtt kiadjuk az alábbi parancsot, akkor a git elmenti a jelszavunkat. Ez csak akkor ajánlott ha az adott gépet nem használja más rajtunk kívül.

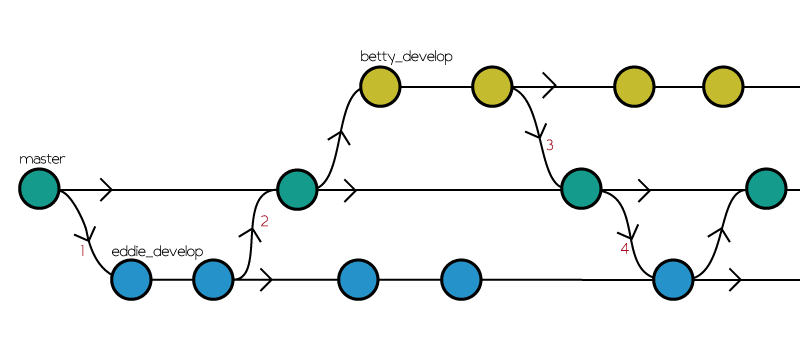
$ git config credential.helper store

Ha ezután push -olunk akkor csak az első alkalommal kell jelszó

## Branch - ágak

A branch -ok olyan párhuzamosan futó folyamatok, amelyek külön kerülnek figyelésre és mentésre. Lehetőség van őket egyesíteni a fő projekttel. Tipikusan akkor jönnek jól ha egyszerre sokan dolgoznak egy munkán. Mindenkinek van saját brancs -a, amit ha végzett a munkával vissza lehet kötni a fő ágba, ami a master.

Az alábbi példában a master branch mellett két másik is van, az egyik Betty, a másik Eddie fejlesztéseit tartalmazza. Az is jól látható, hogy miután elvégeztek egy feladatot a munkájukat visszatöltik a fő ágba, a master -be. Utána folytatják a fejlesztést és a fontosabb fejlesztéseket sorozatosan töltögetik vissza a fő ágba. Ez a való életből vett példa, ahol la GIT verziókezelő segíti a csoportos munkát.

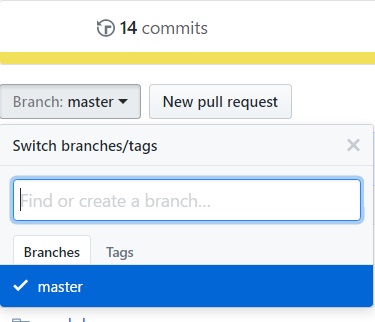


A branch -okat a branch paranccsal lehet lekérni. Példa az összes branch lekérésére:

git branch -a

\* master  
 remotes/origin/master

A kimenet is látszik a parancs után. Láthajuk hogy az egyetlen meglévő branch a master. A gihub felületén könnyen létrehozhatunk új branch -okat, a fájlok felett balra lárható kis lenyíló listában. Egyszerűen megadjuk a nevet és entert ütünkt:



Új branch -ot parancssorban is létrehozhatunk. Navigáljunk a munkakönyvtárba és adjuk ki az alábbi parancsot:

git checkout -b develop

Ha még nincs ilyen nevű branch, akkor a következő üzenetet kell látnunk:

Switched to a new branch 'develop'

Ez azt jelenti, hogy létrejött az új branch. Itt még nem kerül szinkronizálásra automatikusan a szerverrel, akkor fognak róla értesülni a projektben résztvevő fejlesztők ha kiadtuk a git commit és a git pull parancsokat.

Ha nem sikeült a létrehozása annak az a leggyakoribb oka hogy már van azonos nevű branch a tárolóban. Ekkor a következőhöz hasonló hibaüzenet fog megjelenni:

fatal: A branch named 'develop' already exists.

Váltani meglévő branch -ok között a checkout paranccsal tudunk, ebben az esetben nem szükséges a -b kapcsoló használata mint az új létrehozásánál:

git checkout master  
Switched to branch 'master'

Az üzenet ekkor is tájékoztat a sikeres váltásról.

## Két branch egyesítése

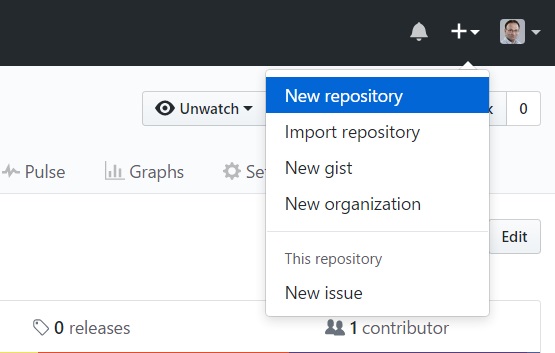
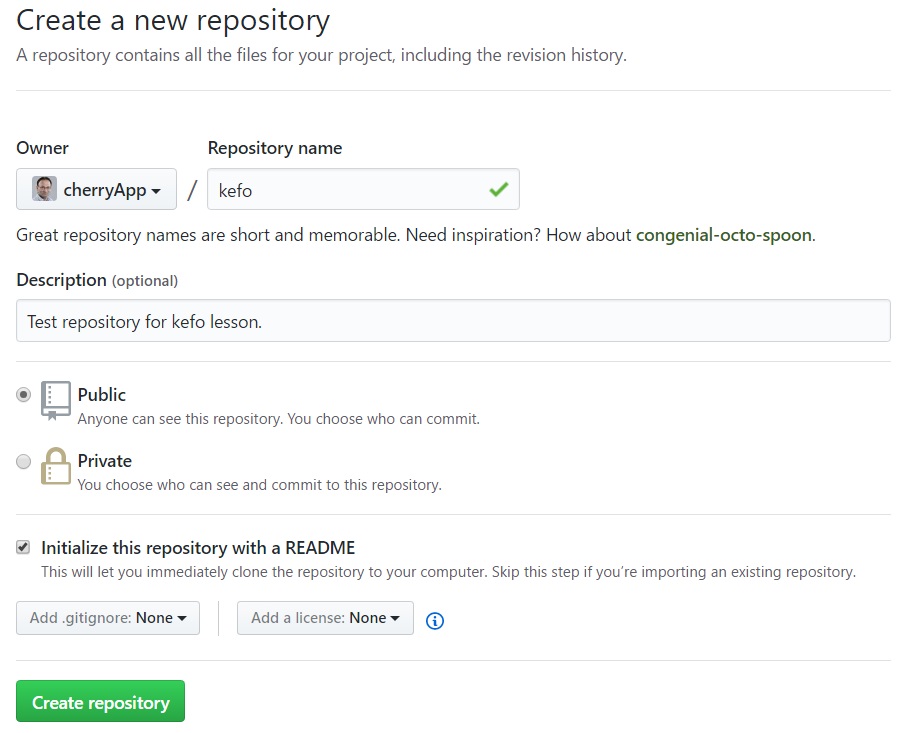
A munkánk során gyakran előfordul hogy befejezzük a fejlesztést egy branch -on belül és szeretnénk egyesíteni a különböző branch -okban lévő fájlokat. Ekkor úgymond egyesíteni kell őket, ezt merge -nek hívják. Két branch egyesítésének lépései:

1. Belépünk a fő branch -ba:  
   git checkout master
2. Kilistázzuk az elérhető branch -okat:  
   git branch
3. Itt láthatjuk a branch -ok listáját, ahol az aktuális csillaggal van jelölve:  
    develop   
   \* master
4. Most hogy a master branch az aktív, adjuk ki a következő parancsot:  
   git merge develop --no-ff
5. A --no-ff kapcsoló szinkronba hozza a két külön branch -ban található commit -okat. Mivel a merge parancsot a master branch -ban adtuk ki, így a master és a develop branch lesz egyesítve. Ha helyesen lefutott a merge, akkor erről is értesítést kapunk. Ebben az érintett fájlok száma is benne lesz:  
   Merge made by the 'recursive' strategy.  
    0 files changed  
    create mode 100644 develop
6. Végül ne feledjük el hogy fel kell tölteni a módosításokat a tárolóba:  
   git push

## 

## Egy új Github tároló beállítása

Az alábbi leírásban azt mutatom be, hogyan hozhatunk létre egy új tárolót - repository -t a github webes felületén.

1. Először lépjünk be a github fiókunkba, vagy hozzunk létre egyet. A github használata ingyenes, ha nyilvánosan akarjuk kezelni a tárolóinkat. A nyilvános tárokókban lévő fájlokat bárki letöltheti, de szerkeszteni csak mi tudjuk, vagy azok a github felhasználók akiknek erre jogot adunk.
2. A felhasználói felület jobb felső sarkában kattintsunk a plus jelre, az itt megjelenő listában válasszuk ki a “New repository” lehetőséget.  
   
3. Az itt megjelenő ablakban először adjuk meg az új tároló nevét. A rendszer validálja a nevet hogy nincsenek-e benne tiltott karakterek vagy szóköz, illetve hogy nem létezik-e már ezen a néven tárolónk. A tulajdonos felhasználói neve és a tároló neve lesz a github.com oldalon belül az elérési út.  
   Itt érdemes adni egy rövid leírást is (Description), mert akkor a rendszer automatikusan létrehozza ez alapján a kötelező readme.md fájlt. Ezt az utolsó checkboxnál még be kell állítani.  
   Azt is adjuk meg hogy publikus vagy privát legyen a tároló. Ha nincs előfizetésünk akkor a privát választása esetén az előfizetés oldalra irányít minket a rendszer.  
   Az alábbi kép részletesen bemutatja az ajánlott beállításokat, természetesen a megfelelő helyekre a projedt specifikus beállításokat kell beírni.  
   A megfelelő beállítások megadása után kattintsunk a “Create repository” feliratú gombra. Ennek hatására létrejön az új tároló és megnyílik szerkesztésre.  
   

# Összefoglalás

A modern fejlesztés során elkerülhetetlen a verziókövető rendszerek használata. Különösen igaz ez ha többen is párhuzamosan résztvesznek a mukában, illetve nagy adatbiztonságra törekednek. Erre ad megoldást az előzőekben bemutatott GIT rendszer. A jelenlegi trendek alapján a verziókezelők csatáját nagy valószínűséggel ez fogja megnyerni, tehát minden fejlesztőnek ajánlott hogy elsajátítsa a kezelését.

Végül néhány hasznos link:

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

<https://github.com/>

[Digitalocean git tutorials](https://www.digitalocean.com/community/tutorial_series/introduction-to-git-installation-usage-and-branches)