# Git és GitHub alapok kezdőknek

## Tartalomjegyzék

1. [Bevezetés](#bevezetes)  
   1.1 [Mi az a Git?](#mi-az-a-git)  
   1.2 [Mi az a GitHub?](#mi-az-a-github)  
   1.3 [Miért hasznos a verziókezelés?](#miert-hasznos-a-verziokezeles)
2. [Első lépések a Gittel](#elso-lepesek-a-gittel)  
   2.1 [Git telepítése](#git-telepitese)  
   2.2 [Alapvető fogalmak](#alapveto-fogalmak)
3. [Lokális projekt létrehozása Git-tel](#lokalis-projekt-letrehozasa-git-tel)  
   3.1 [Helyi repó létrehozása (git init)](#helyi-repo-letrehozasa)  
   3.2 [Fájlok követésbe vétele (git add)](#fajlok-kovetesbe-vetele)  
   3.3 [Változások rögzítése (git commit)](#valtozasok-rogzitese)
4. [Távoli repók és GitHub](#tavoli-repok-es-github)  
   4.1 [GitHub regisztráció](#github-regisztracio)  
   4.2 [Új repó létrehozása a GitHubon](#uj-repo-letrehozasa-a-githubon)  
   4.3 [Kapcsolat létrehozása a távoli repóval (git remote add origin)](#kapcsolat-letrehozasa-a-tavoli-repoval)  
   4.4 [Változások feltöltése (git push)](#valtozasok-feltoltese)  
   4.5 [Frissítések lekérése (git pull)](#frissitesek-lekerese)
5. [Branch-kezelés (ágak)](#branch-kezeles)  
   5.1 [Branch létrehozása](#branch-letrehozasa)  
   5.2 [Branch váltása](#branch-valtasa)  
   5.3 [Merge (ágak egyesítése)](#merge-agak-egyesitese)  
   5.4 [Konfliktuskezelés](#konfliktuskezeles)
6. [GitHub funkciók](#github-funkciok)  
   6.1 [Pull requestek](#pull-requestek)  
   6.2 [Issues](#issues)  
   6.3 [Projekt boardok](#projekt-boardok)
7. [Alap Git parancsok és gyakori kapcsolók](#alap-git-parancsok-es-gyakori-kapcsolok)
8. [Gyakori hibák és megoldások, best practices](#X3c359d8f5270c894e43c37846af21d3cb1ad8da)
9. [Összefoglalás](#osszefoglalas)

## 1. Bevezetés

### 1.1 Mi az a Git?

A **Git** egy nyílt forráskódú verziókezelő rendszer, amelyet Linus Torvalds (a Linux-kernel létrehozója) fejlesztett ki. Célja, hogy a szoftverfejlesztők (és más, fájlokkal dolgozó szakemberek) képesek legyenek **minden változást** nyomon követni a projektjeikben.

* Lehetővé teszi a különböző verziók könnyű visszaállítását.
* Támogatja a csapatmunkát, több felhasználó ugyanazon projekten való párhuzamos munkáját.

### 1.2 Mi az a GitHub?

A **GitHub** egy online platform, amely Git-alapú projektek tárolására és kezelésére specializálódott.

* Ingyenes nyilvános tárhelyeket (repository-kat) kínál.
* Segíti a csapatok együttműködését (például Pull Requestek, Issue-k, Projekt boardok).
* További funkciókat nyújt a fejlesztési folyamat gördülékenyebbé tételéhez, például folyamatos integráció (GitHub Actions), kódáttekintés, stb.

### 1.3 Miért hasznos a verziókezelés?

* **Biztonsági mentés:** Bármikor vissza tudsz térni egy korábbi verzióhoz, ha elrontasz valamit.
* **Csapatmunka:** Többen egyszerre dolgozhattok ugyanazon a projekten, és követhető lesz, ki milyen módosítást végzett.
* **Átláthatóság:** Minden commitnál (rögzített változásnál) látszik, mikor és ki mit változtatott.
* **Konfliktuskezelés:** Ha egyszerre módosítotok egy fájlt, a rendszer segít feloldani az esetleges ütközéseket.

## 

## 2. Első lépések a Gittel

### 2.1 Git telepítése

1. **Windows:**
   * Töltsd le a [git-scm.com](https://git-scm.com/download/win) oldalról a telepítőt.
   * Futtasd a telepítőt, a beállítások többségét hagyd alapértelmezett módon.
   * A telepítés részeként feltehető Git Bash nevű parancssori eszköz is.
2. **macOS:**
   * Terminálban futtasd: xcode-select --install, vagy töltsd le a [git-scm.com](https://git-scm.com/download/mac) oldalról.
   * Kövesd a telepítési utasításokat.
3. **Linux (pl. Ubuntu):**
   * Terminálban írd be:
   * sudo apt-get update  
     sudo apt-get install git
   * Más disztribúciókon a megfelelő csomagkezelő parancsát használd.

Telepítés után ellenőrizd a verziót:

git --version

### 2.2 Alapvető fogalmak

* **Repository (röviden: repó):** Maga a projekt, a benne található fájlok és azok előzményei.
* **Working Directory (munkakönyvtár):** A könyvtár, ahol aktuálisan dolgozol a fájlokon.
* **Stage/Staging area (gyakran index):** Egy átmeneti terület, ahova a következő commitba szánt változásokat “felkészíted”.
* **Commit:** Egy “pillanatkép” a projekt állapotáról, amit a Git megjegyez. Az üzenet (commit message) leírja, hogy mit változtattál.
* **Branch:** Egy párhuzamos munkafolyamat a fő ághoz (main/master) képest.
* **Merge:** Külön ágakon végzett munkák egyesítése.

## 3. Lokális projekt létrehozása Git-tel

### 3.1 Helyi repó létrehozása (git init)

* Válaszd ki a projekted mappáját, vagy hozz létre egy új könyvtárat.
* Nyisd meg a terminált/Parancssort ebben a mappában.
* Futtasd:
* git init
* Ez létrehoz egy rejtett .git mappát, amely tartalmazza a verziókövetéshez szükséges fájlokat.

### 3.2 Fájlok követésbe vétele (git add)

* Ha új fájlokat hoztál létre, a Git még nem követi őket automatikusan.
* Példa:
* git add index.html
* Ezzel a index.html fájl bekerül a staging area-ra (vagyis készen áll a commitra).
* Ha minden változtatást hozzá akarsz adni egyszerre, használhatod:
* git add .
* (A . jelenti, hogy az aktuális könyvtár összes változott fájlját hozzáadod.)

### 3.3 Változások rögzítése (git commit)

* Miután hozzáadtad a fájlokat a staging area-ra, commitold őket:
* git commit -m "Első commit üzenetem"
  + A -m kapcsoló jelzi, hogy a commit üzenetét egy sorban adod meg.

## 4. Távoli repók és GitHub

### 4.1 GitHub regisztráció

1. Nyisd meg a [GitHub.com](https://github.com/) oldalt, és regisztrálj egy ingyenes fiókot.
2. Igazold vissza az e-mail címedet.
3. Bejelentkezés után a jobb felső sarokban találod a menüpontokat az új repók létrehozásához, a profilod eléréséhez, stb.

### 4.2 Új repó létrehozása a GitHubon

1. Kattints a **New** (vagy “+” ikon) gombra a repók listájában.
2. Adj nevet a repónak (pl. teszt-projekt).
3. Beállíthatod, hogy nyilvános (public) vagy privát (private) legyen.
4. Kattints a **Create repository** gombra.

### 4.3 Kapcsolat létrehozása a távoli repóval (git remote add origin)

* Ha már létezik egy lokális repód, és szeretnéd összekötni a GitHub-on lévő repóval, használd a következő parancsot:
* git remote add origin https://github.com/felhasznalonev/teszt-projekt.git
  + Az origin jelöli az elsődleges (általában GitHubon lévő) távoli repót.
* Ezután érdemes az “main” ágra átnevezni (ha szükséges) vagy beállítani:
* git branch -M main
* Ezzel a helyi “master” ágat átnevezheted “main”-re. Manapság a GitHub alapból a “main” elnevezést használja.

### 4.4 Változások feltöltése (git push)

* Az első feltöltésnél (ha létrehoztad a távoli repót, de még nem töltötted fel rá a helyi változatot) futtasd:
* git push -u origin main
  + A -u (vagy --set-upstream) beállítja, hogy a lokális “main” ág melyik távoli ághoz kapcsolódjon alapértelmezésként.
* Következő alkalommal már elég lesz a:
* git push
* parancs, és a Git tudja, hogy hova töltsön fel.

### 4.5 Frissítések lekérése (git pull)

* Ha valaki más is dolgozott ugyanezen a repón és push-olta a változásait, a lokális repót frissítened kell a legújabb állapotra:
* git pull
* A git pull valójában két műveletet kombinál: **fetch** (lekéri az új commitokat) és **merge** (összefésüli őket a te lokális ágadba).
* Ha esetleg ütközés (merge conflict) adódik, kézzel kell feloldani, majd újra commitolni.

**Gyakori kapcsolók:**

* git pull --rebase: a változtatásokat “rebase-eli” a lokális commitokra, ami tisztább commit történetet eredményezhet, de kicsit összetettebb lehet a konfliktusok feloldása.
* git pull origin main: egyértelműen megmondja, melyik távoli repóból és melyik ágról húzza le a változásokat.

## 5. Branch-kezelés (ágak)

### 5.1 Branch létrehozása

* Új fejlesztési vonal elindításához hasznos lehet egy új ág:
* git branch uj-funkcio
* Ellenőrizheted az ágak listáját:
* git branch
* A csillag (\*) jelzi, melyik ágban tartózkodsz éppen.

### 5.2 Branch váltása

* Átváltás az “uj-funkcio” ágba:
* git checkout uj-funkcio
* Vagy egyszerre létrehozás és váltás:
* git checkout -b uj-funkcio

### 5.3 Merge (ágak egyesítése)

* Ha befejezted az új funkciót az “uj-funkcio” ágban, visszatérsz a fő ágba (pl. “main”):
* git checkout main
* Ott egyesíted az új ág változtatásait:
* git merge uj-funkcio
* Ekkor a főág átveszi az “uj-funkcio” ág commitjait.

### 5.4 Konfliktuskezelés

* Ha mindkét ágban ugyanazon sorok változnak, merge conflict lép fel. A Git jelölni fogja a konfliktusos részeket a fájlban, pl.:
* <<<<<<< HEAD  
  Ez a main ág változata  
  =======  
  Ez az uj-funkcio ág változata  
  >>>>>>> uj-funkcio
* Kézzel oldd fel a konfliktusokat úgy, hogy a megfelelő verzió maradjon meg (vagy mindkettőt beolvasztod).
* Ezután add hozzá a módosított fájlt (git add) és készíts egy új commitot (git commit).

## 6. GitHub funkciók

### 6.1 Pull Requestek

* A Pull Request (PR) lehetővé teszi, hogy az “uj-funkcio” ágban végzett módosításokat átnézésre és elfogadásra elküldd a fő ágba.
* Egy csapattárs vagy projektgazda ellenőrzi a kódodat, megjegyzéseket fűzhet hozzá, és ha minden rendben van, merge-eli azt a fő ágba.
* GitHub webfelületén a repód oldalán az “Pull requests” fül alatt hozhatsz létre új PR-t.

### 6.2 Issues

* Az **Issues** funkció a feladatok, hibák, ötletek nyilvántartására szolgál.
* A projekttel kapcsolatos problémákat, javaslatokat, kérdéseket itt lehet felvetni és követni.
* Lehet hozzárendelni felelőst (assignee), címkéket (labels), határidőt.

### 6.3 Projekt boardok

* Használhatod a **Project** funkciót (táblákat, kanban rendszert) a teendők kezeléséhez, hasonlóan Trellohoz.
* Létrehozhatsz oszlopokat (pl. To do, In progress, Done), és az Issues kártyákat idehúzva követheted a munkafolyamatot.

## 7. Alap Git parancsok és gyakori kapcsolók

Íme néhány gyakran használt Git parancs rövid magyarázattal és fontosabb kapcsolókkal:

1. **git init**
   * Létrehozza a .git mappát a verziókezeléshez.
2. **git status**
   * Megmutatja az aktuális állapotot: mely fájlok módosultak, melyek várnak commitra.
   * Kapcsolók: -s (rövid output).
3. **git add [fájl]**
   * Stage-re (következő commitba) rakja a kiválasztott fájlt.
   * git add . → minden módosított/új fájl stage-re kerül.
4. **git commit -m "üzenet"**
   * Új commit létrehozása a stage-en lévő fájlokból.
   * Kapcsolók:
     + -a: automatikusan hozzáad minden módosított fájlt a commithoz (az újonnan létrehozottakat nem).
     + --amend: a legutolsó commit üzenet vagy tartalom módosítása.
5. **git log**
   * Listázza a commitokat.
   * Kapcsolók:
     + --oneline: rövidített egy soros nézet.
     + --graph: grafikus ágat mutat.
6. **git branch**
   * Ágak kilistázása.
   * git branch [ág-név]: új ág létrehozása.
   * -d [ág-név]: ág törlése.
7. **git checkout [ág-név]**
   * Átváltás a megadott ágra.
   * -b [ág-név]: új ág létrehozása és átváltás rá.
8. **git merge [ág-név]**
   * A megadott ág változtatásainak egyesítése az aktuális ágba.
9. **git push**
   * Változtatások feltöltése a távoli repóba.
   * Első alkalom: git push -u origin main.
10. **git pull**
    * Változások letöltése és egyesítése a lokális ággal.
    * --rebase: a letöltött commitokat a lokális commitok előttre fűzi.
11. **git remote**
    * Kezeli a távoli repókat.
    * git remote add origin [URL]: hozzáad egy új távoli repót.
    * git remote -v: kilistázza a távoli repókat.
12. **git stash**
    * Ideiglenesen “elteszi” a munkádat, ha gyorsan át kell váltanod másik ágra.
    * git stash apply: visszaállítja az eltett munkát.

Ez a lista korántsem teljes, de az alap funkciók 90%-át lefedi, amire a kezdéshez szükséged van.

## 

## 8. Gyakori hibák és megoldások, best practices

1. **Elfelejtettél commitolni, mielőtt váltanál egy másik ágra**
   * Megoldás: Használd a git stash parancsot, mielőtt átváltasz. Később a git stash apply visszaállítja a módosításokat.
2. **Konfliktusok merge vagy pull során**
   * Megoldás: Terminálban/GitHubon látod, mely fájlok ütköznek. Nyisd meg a fájlokat, oldd fel a konfliktus-jelöléseket, majd git add, git commit.
3. **Rosszul fogalmazott commit üzenet**
   * Megoldás: git commit --amend -m "Új üzenet" parancs, ha még nem pusholtad a commithoz a változásokat.
4. **Nem frissítetted a lokális repót, mielőtt újra dolgozni kezdtél**
   * Mindig érdemes git pull-lal kezdeni a munkát, hogy az aktuális legfrissebb állapoton dolgozz.
5. **Változások elvesztése**
   * Git mindig megőrzi a commitok előzményeit, tehát általában semmi sem “vész el” véglegesen. A staging area, a stash, és a commitok révén visszakereshetők a verziók.

### Best practices

* **Kis, gyakori commitok**: Minden logikus változtatás után commitolj, hogy a commit üzenet egyértelműen tükrözze, mi történt.
* **Beszédes commit üzenetek**: Mindig adj egy rövid, tömör, de érthető leírást.
* **Branch használata új funkciókhoz vagy hibajavításokhoz**: Ne dolgozz minden funkción a main ágon.
* **Rendszeres Pull kérések**: Ha csapatban dolgoztok, gyakran kérj visszajelzést Pull Request formájában.
* **Kódellenőrzés (Code Review)**: A GitHub erre kiváló eszközöket ad; mindig nézzétek át egymás kódját.

## 9. Összefoglalás

A Git egy rendkívül sokoldalú, megbízható verziókezelő rendszer, amely manapság szinte elengedhetetlen a szoftverfejlesztésben (és egyre inkább más területeken is). A GitHub pedig segít abban, hogy online tárold a projektjeidet, csapatban dolgozz, és követhetően dokumentáld a fejlesztési folyamatot.

**Legfőbb lépések, amiket érdemes megjegyezned kezdőként:**

1. **Git telepítése** (Windows, Linux, macOS).
2. **Lokális repó létrehozása** – git init
3. **Fájlok hozzáadása** – git add .
4. **Commit készítése** – git commit -m "commit üzenet"
5. **GitHubos repó létrehozása**, majd **távoli kapcsolat beállítása** – git remote add origin git@github.com/username/repo
6. **Feltöltés** – git push -u origin main
7. **Frissítések letöltése** – git pull
8. **Branch létrehozása és merge**.
9. **Pull Requestek**, **Issues** használata GitHubon a csapatmunka során.