# GitKraken

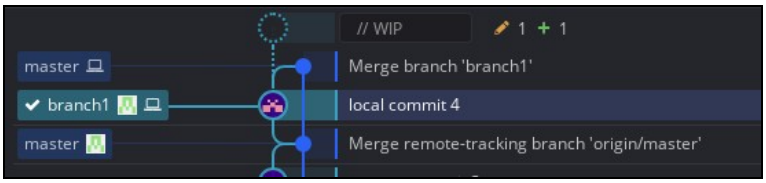
## Contents



# Fájl kezelés alapok

## Munkaterület kezelése

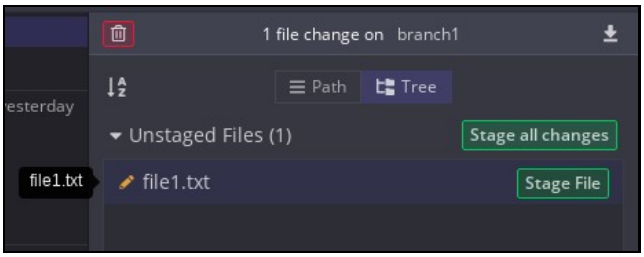
Ha van lokális módosítás a working copy-ban, akkor a commit tree legtetején megjelenik a //**WIP** (Work In Progress) sor üres körrel, ami annak a branch-nek az utolsó commit-jából származik, ami éppen a working területre be van töltve.




A //WIP mellett található a módosított, új és törölt fájlok listája, amit az utolsó commit-hoz képest kell érteni. Értelem szer?en a ceruza mögötti szám a módosított fájlok száma, a + mögött a hozzáadott és a - mögötti a törölt fájlok darabszáma. A //WIP részbe beírhatjuk kapásból a leend? commit summary részét. Nyugodtan nyomjunk enter-t, ez még nem commit-ál semmit csak a fában fog már elve a leend? commit nevével szerepelni. A fenti képen a //WIP sor a **branch1** lokális branch-b?l származik, ami a jelenlegi check-out-olt branch, mivel a **branch1** neve mellett van a kis zöld pipa.

Ha van módosított fájl, és nem a //WIP soron állunk akkor bal felül megjelenik egy figyelmeztetés egy gombbal, hogy módosított fájl a munkaterületen. A gombbal kapásból a //WIP-re ugorhatunk, ahol megjelennek a módosított fájlok a baloldalon:

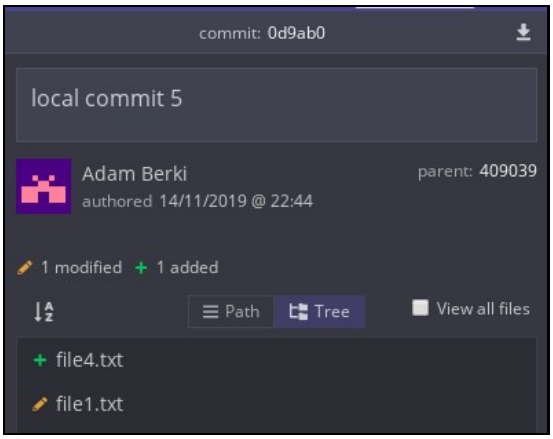

Ha a fájl neve fölé visszük az egeret, akkor megjelenik a 'Stage file' gomb, amivel csak ezt a fájlt adhatjuk a stage-hez. Az 'Unstaged files' szekció fölötti gombbal az összes még nem stage-elt fájlt hozzáadhatjuk a stage-hez:




## Commit részletek

### Fájlok listázása

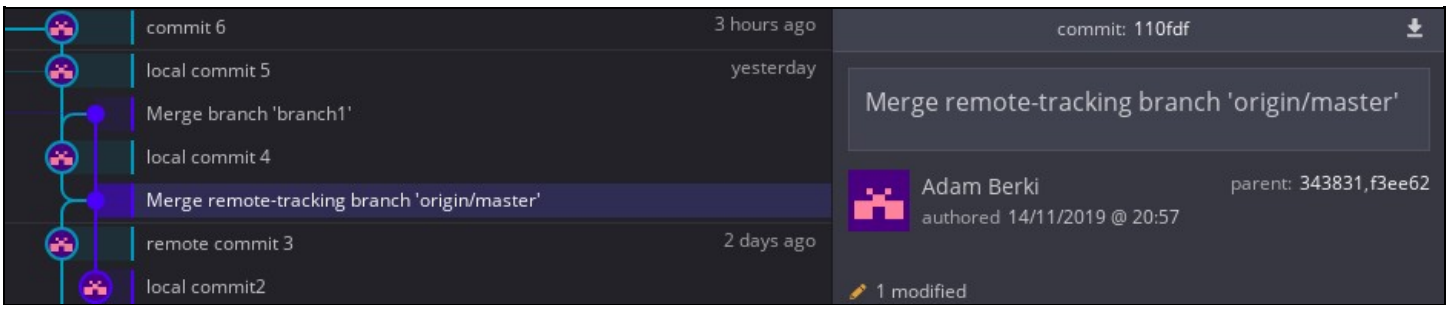
Ha a commit fában egy commit-ra kattintunk, akkor megjelenik a baloldalon a commit részletek. Tartalmazza commit címét, a commit user-t, valamit a commit-ban szerepl? fájlok listáját.




- **Commit id**: A commit száma az ablak legtetején van. A példában: 0d9ab088667b93aa99336bd966afcbf1a5823ce2, aminek az els? 6 karaktere van megjelenítve.
- **Parent**: A commit szül?je a commit fában a commit címe alatt található, ez a példában: 409039643a6d770f73b5f4327fdc25510ba73162.

A ceruza jelenti a módosított fájlt, a zöld+ a hozzáadott fájlt, míg a piros'-' az adott commit-ban törölt fájlt.

Ha egy olyan commit-t nézünk, ami merge-b?l keletkezett, akkor két parent ID lesz a commit részletekben:

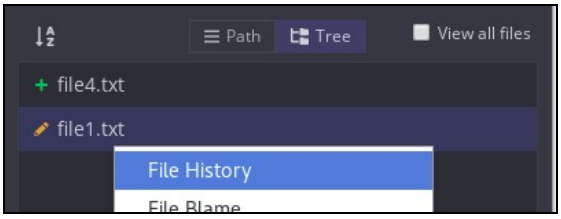



A képen kiválasztott commit egy merge commit. A jobb oldalon, a commit részletekben láthatjuk hogy két szül?je van:

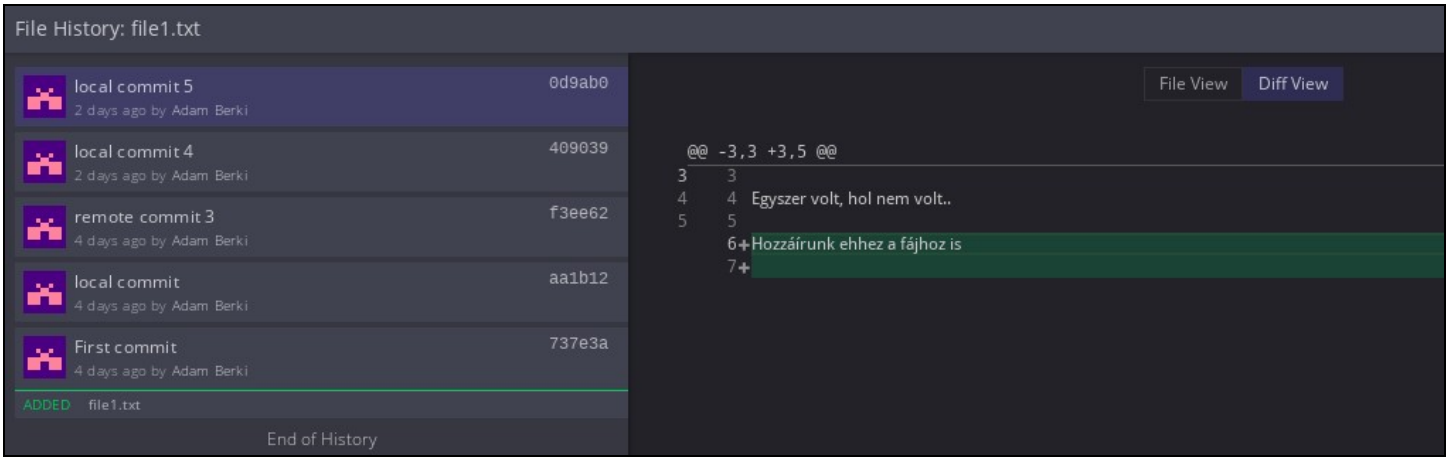
- 343831030f0d41d579270f95d9458cd845b6adaa
- f3ee62d54a8eed112530b28a1880b68575d7355b

### File history

A commit részletekben kattintsunk jobb clikkel a fájl nevére, majd válasszuk a 'File History'-t.

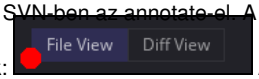


A file history-ban baloldalon láthatjuk a fájlt befolyásoló commit-okat. A legels? commit van legalul.

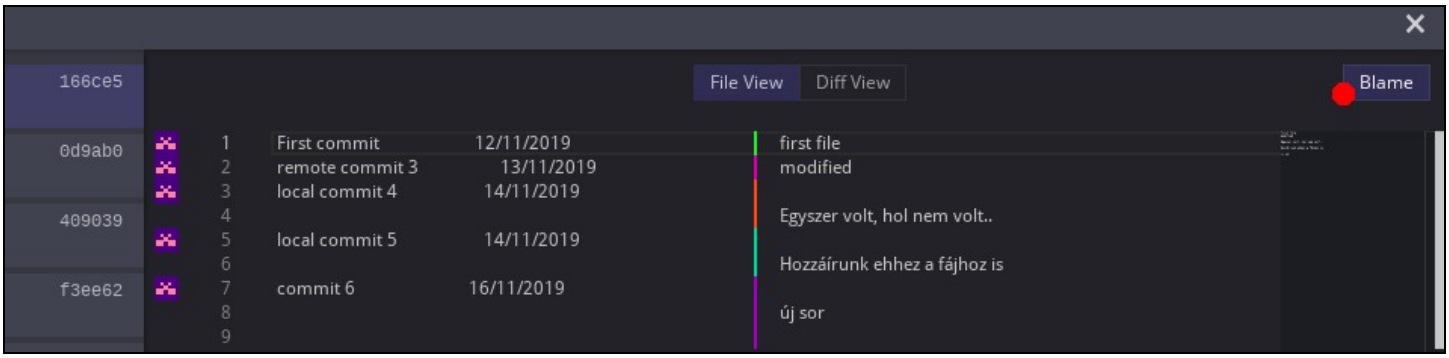



- Ha egy adott commit-ra kattintunk, akkor megjelenik a baloldalon a diff ablak, ahol az el?z? commit-hoz képest láthatjuk a diff-et.
- Ha a commit listában a commit id-ra kattintunk, akkor a commit fában az adott commit-ra ugrik

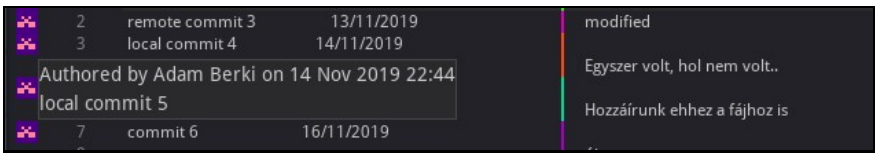
### Blame

A blame funkcióval azt nézhetjük meg, hogy egy adott fájlban melyik sort ki hozta létre. Ez megfelel az SVN-ben az annotate-el. A 'Blame' több helyr?l is elérhet?. Egyrészt a fájl commit history-ában, ha a diff ablak feletti választóban a 'File view'-t választjuk: 

A Blame-et a File view-ban külön be kell kapcsolni a jobb oldalon lév? 'Blame' gombbal.



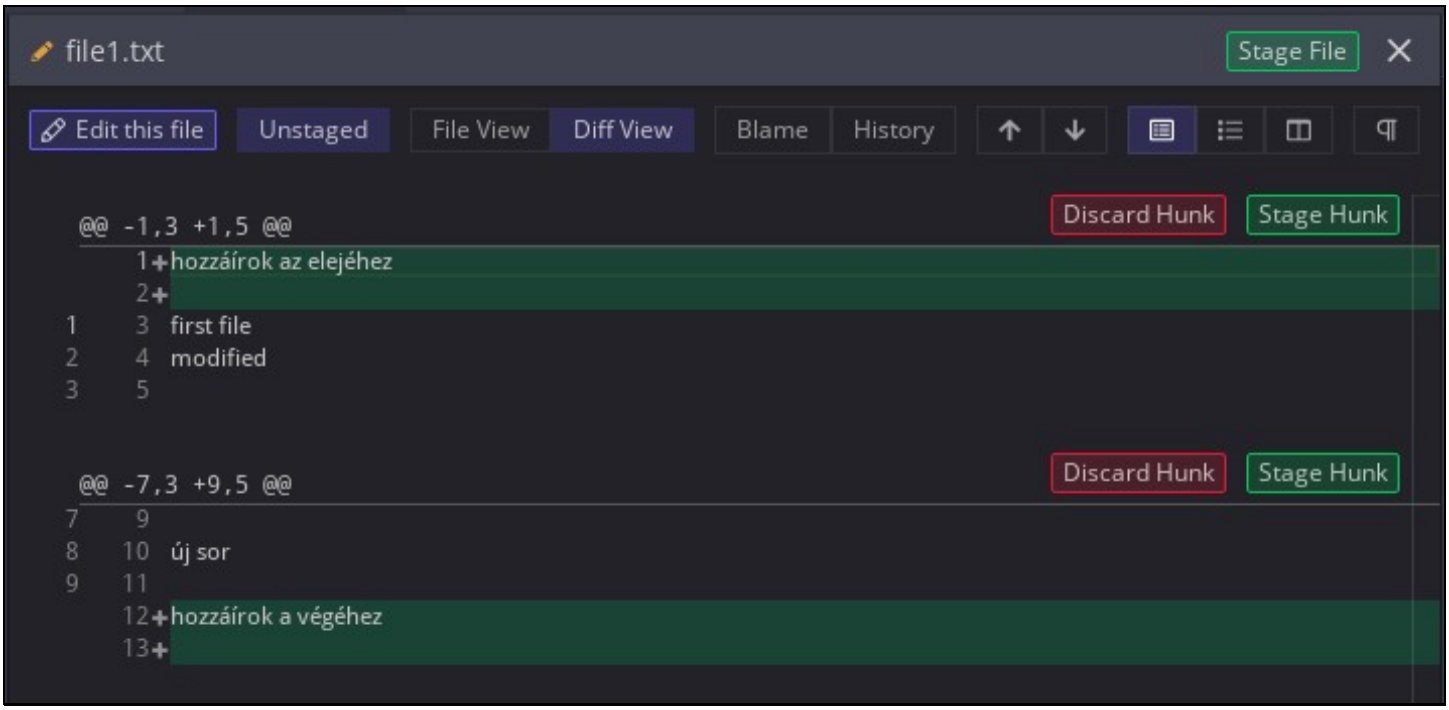

Miden egyes sor mellé balra oda van írva hogy melyik commit-ban keletkezett és a commit el?tt látható a user avatárja. Ha felé visszük az egeret, akkor láthatjuk, hogy ki hozta létre a commit-et:




A Blame és a File commit history is elérhet? minden fájl diff ablakának fejlécéb?l a 'Blame' ill 'History' gombokkal:


## Diff

A diff ablakot mindig a jobb oldali fájl lista ablakból nyithatjuj meg A GitKraken-ben a diff ablakban nagyon sok nézet elérhet?. Alapértelmezetten a szabványos diff jelölést használja, és a változtatásokat alapértelmezetten úgynevezett hunk-okba (nagy egység) csoportosítva mutatja. Minden egyes hunk csak a módosított rész pár soros környezetét tartalmazza.




A diff nézetben a két fájl tartalmát egybe írva láthatjuk minden egyes hunk-ban, nem egymás mellett minta meld-ben. Minden hunk-nak van egy @@ -al kezd?d? fejléce. A fejléc szintaktikája az alábbi:

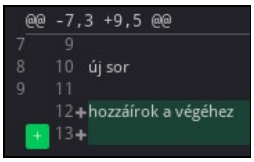
```
@@ -[<régi fájlban a hunk kezdete>, <hossza>] +[<új fájlban a hunk kezdete>, <hossza>] @@
```

- A @@ utáni részben a '-' jelöli a régi fájlt és a '+' az új fájlt. A régi és fájl jelentése attól függ hogy mit diff-elünk mivel. Ha egy commit-ban szerepl? fájlban nézzük a diff-et, akkor a '-' jelöli az új commit el?tti állapotot és a '+' az új commit-ban lév? állapotot. A diff-ben nem az egész fájl tartalmát fogjuk látni, csak azt ami változott a régihez képest.
- A - és a + után is két-két számot láthatunk. Az els? azt mondja meg, hogy az adott hunk (fájl részlet) hányadik sorban kezd?dik, míg a másik azt mondja meg hogy az adott hunk hát sort tartalmaz az els? ill. a második fájlból.

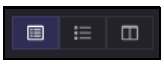
A fenti példában két hunk-ot látunk. A másodikban a fejléc az alábbi:

```
@@ -7,3 +9,5 @@
```

Ez azt jelenti, hogy a réig fájlban (-) a megjelenített sorok a 7. sorban kezd?dnek, és 3 sort tartalmaz a hunk. Az új verzióban (+) a megjelenített sorok már a 9. sorba kerültek, és 5 sort tartalmaz már az új verzió:




Minden 'hunk' ban a +zöld sorok jelölik az új részt (ami a régiben verzióban nem volt benne) és a -piros kezdet?ek a hiányzó részt, ami az új verzióban már nincs benne.
A diff nézetet a jobb fels? sarokban lév? gombokkal vezérelhetjük:



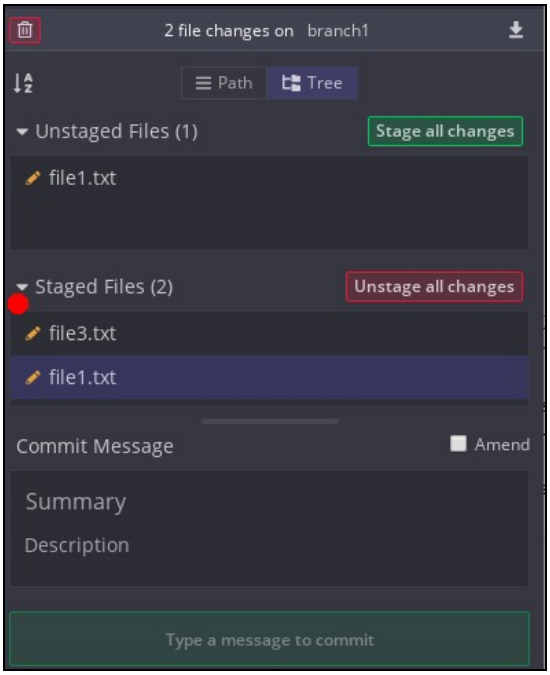
- 'Hunk view': baloldali gomb, ez az alapértelmezett, diff szabványú megjelenítés, csak a módosult szekciókat mutatja, ezt láthatjuk a fenti példában.
- 'Inline view': második gomb, itt is diff szabványban láthatjuk a különbségeket, de nem hunk-ban, az egész fájlt egyben látjuk.
- 'Split view': jobboldali gomb, hagyományos osztott képerny?s nézet, mint a meld-ben.

Ha 'Hunk view'-ban vagyunk, akkor az egyes hunk-okban lév? módosításokat elvethetjük a Hunk jobb fels? sarkában lév? 'Discard hunk' gombbal:


## Stage

## Stage alapok

A stage a commit-ra szánt fájlok gy?jt?helye, egy átmeneti lépés a munkaterület és a repository között. A munkaterületen és a stage-en lév? fájlokat a jobb oldali mez?ben láthatjuk:



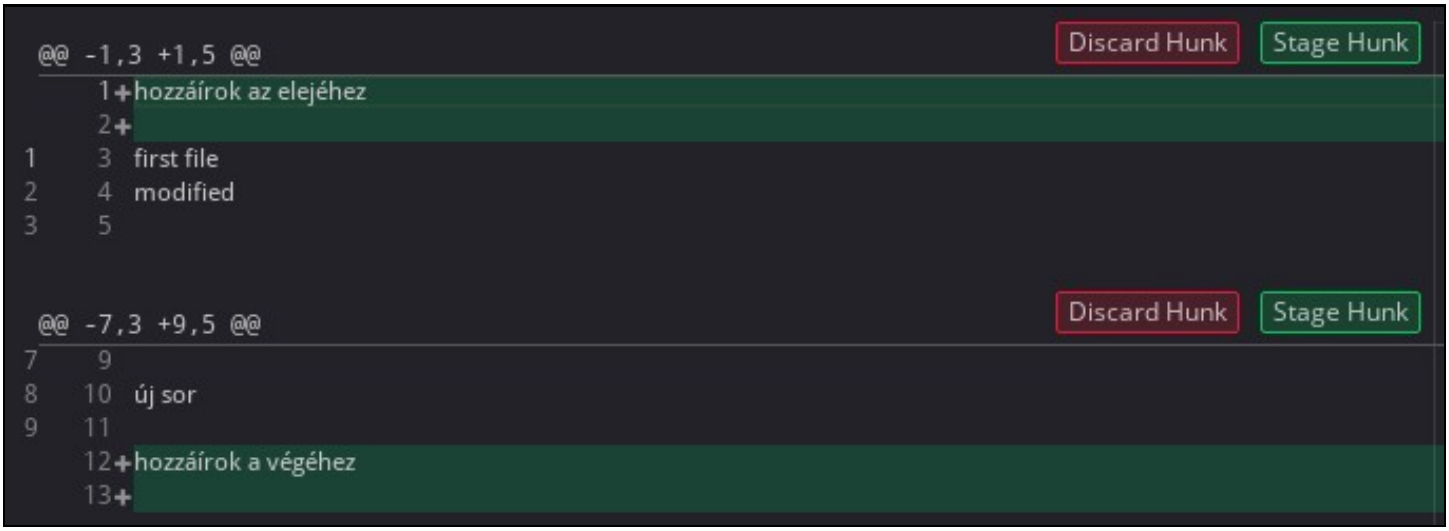

A stage-be be és ki lehet rakni a fájlokat egészen a commit-ig. Csak az lesz commit-álva ami a stagen van.

## Hunk stage-be rakása

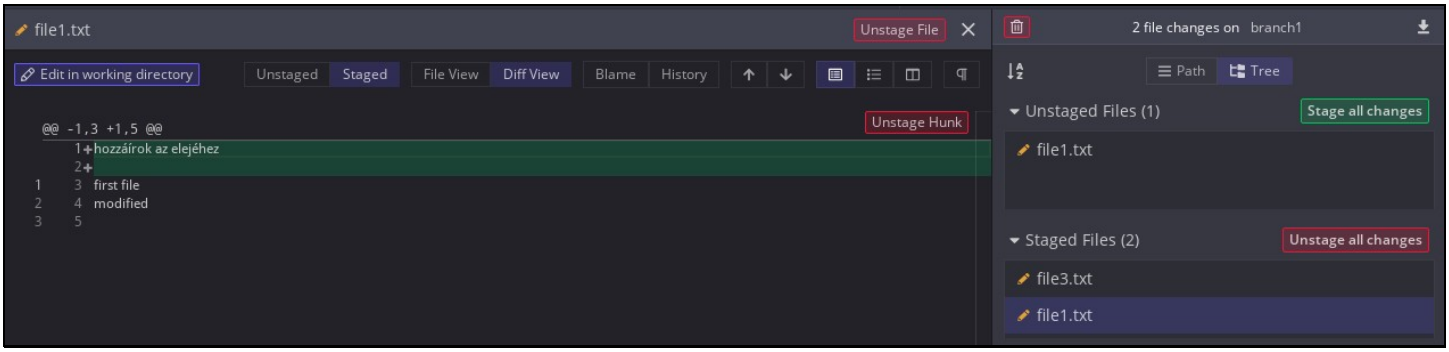
Nem csak egész fájlokat lehet a stage-hez adni, lehet csak egy fájl módosításainak részleteit is. Nyissuk meg az 'Unstaged files' szekcióban lév? fájlt a diff-ablakban:


A diff ablakban a stag-hez lehet adni egyesével a hunk-okat vagy a hunk-on belül akár egyetlen egy módosított sort is. Egy módosított fájlból egy hunk-ot a hunk jobb fels? sarkában lév? 'Stage hunk' gombbal tehetjük meg:

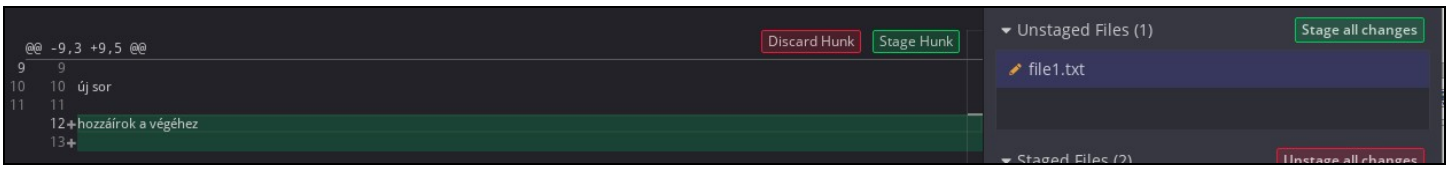

Tegyük fel, hogy a file1.txt -ben két hunk található, a diff ablak az alábbi:



Ha itt az els? hunk-ot hozzáadjuk a stage-hez a 'Stage hunk' gomb megnyomásával, akkor a stage-be bekerül a file1.txt-nek egy olyan változata, amiben csak az els? hunk-ban szerepl? változtatás, vagyis a 'hozzáírok az elejéhez' változtatás szerepelni fog, de a 'hozzáírok a végéhez' változtatás már nem fog szerepelni. Kattintsunk a file1.txt-re a jobb oldali 'Staged files' szekcióban, hogy megnyissuk a stage-ben lév? file1.txt diff ablakát:

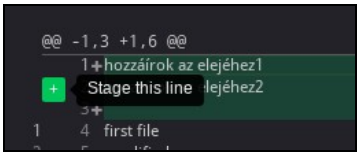



Láthatjuk, hogy csak a 'hozzáírok az elejéhez' szerepel benne. A stage-be rakott hunk-ot az 'Unstage hunk' a-l lehet visszavonni. Ha nem marad egy fájlnak olyan változtatása, ami a stage-ben van, csak akkor t?nik el a 'Staged files' szekcióból.
A file1.txt továbbra is látszik az 'Unstaged files' szekcióban, mivel van még olyan része a fájlnak, ami nem lett a stag-be rakva. Ha megnézzük a diff ablakban az Unstaged file1t.txt-t láthatjuk, hogy most már csak a 'hozzáírok a végéhez' módosítást tartalmazó hunk látszik:

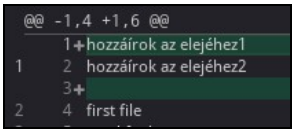



### Egy sor stage-be rakása

Lehet sorokat egyesével is a stage-be rakni. Ehhez az 'Unstaged files' szekcióból nyissuk meg diff módban a fájlt. Majd vigyük az egeret a stage-be rakni kívánt sor elé. Ekkor megjelenik egy zöld+ a sor el?tt.




A sor elé kattintva a stage-be kerül a fájlnak egy olyan változata, amiben csak ez a módosítás szerepel csak a lokális repoban lév? legfrissebb verzióhoz képest.

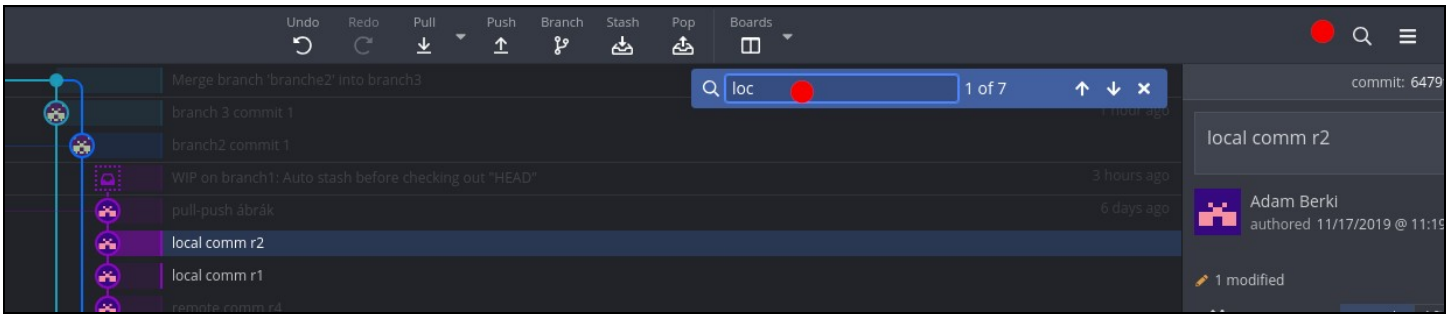



A fenit példában rákattintottam a 'hozzáírok az elejéhez2' sorra, így az már nincs kijelölve az 'Unstaged files' nézetben megnyitott fájlban.
Ha most megnyitom a stage-ben lév? file1.txt-t a diff ablakban, akkor a stage -be került sor zöld-el szerepel. Ha elé visszük az egeret, akkor egy piros '-' jelenik meg, ezzel tudjuk kivenni a stage-b?l a sort:


# Commit ügyek

### Commit-ok keresése

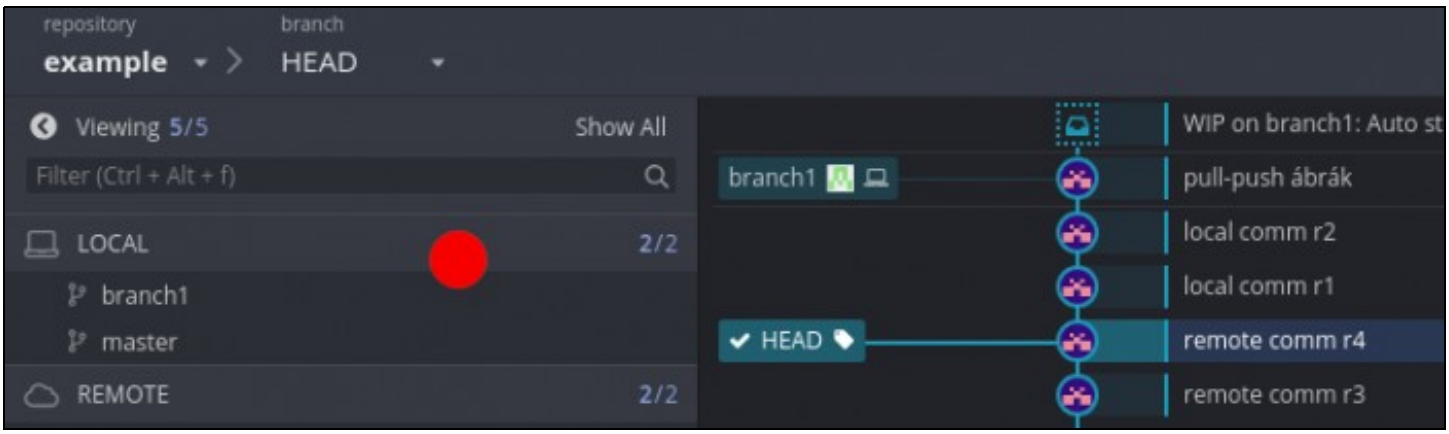
GitKraken-ben a jobb fels? sarokban lév? nagyítóra kattintva kereshetünk a commit-ok nevében.




A keresett név beírása közben csak azokat a commit-okat emeli ki a fában, amire illeszkedik a név. A keres? input melletti le-fel nyíllal ugrálhatunk a találatokon. Alapból az utolsó 2000 commit-ot mutatja a fa.

### Checkout commit

Tetsz?leges commit-ra rá lehet állítani a lokális head mutatót, nem kell hogy egy branch legutolsó commit-ja legyen. Ehhez a commit fában jobb click-menüben válasszuk a 'Check out this commit' lehet?séget. Ekkor láthatjuk, hogy a pipa mellett nem a branch neve lesz, hanem csak annyi hogy 'Head', ami külön branch-ként nem jelenik meg a bal oldali local-branch listában:




### Revert commit

...TODO...

### Reset head to commit

...TODO...

- Soft:
- Mixed:
- Hard:

### Drop commit

...TODO...

### Move commit

...TODO...

## Branch

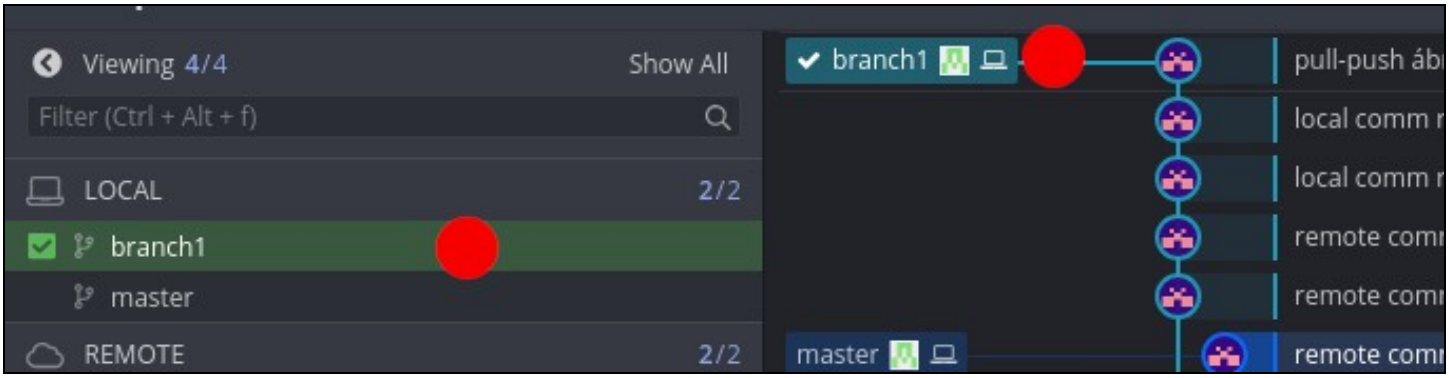
...

## Új branch létrehozása

### branch létrehozása az aktuális head-r?l

Amikor megnyomjuk a fels? menüsorban a 'branch' gombot 

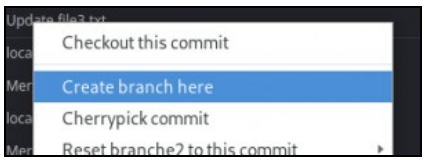
, akkor attól függetlenül, hogy mit nézegetünk éppen a commit fában, mindig a lokális repon-k aktuális branch-énak (checked out) az aktuális head mutatójánál fogja létrehozni az új branch-et, vagyis ráállít még egy head mutatót az új branch nevével. Tehát a lényeg, hogy a bal oldali branch listában és a a commit tree-ben hol van a pipa:



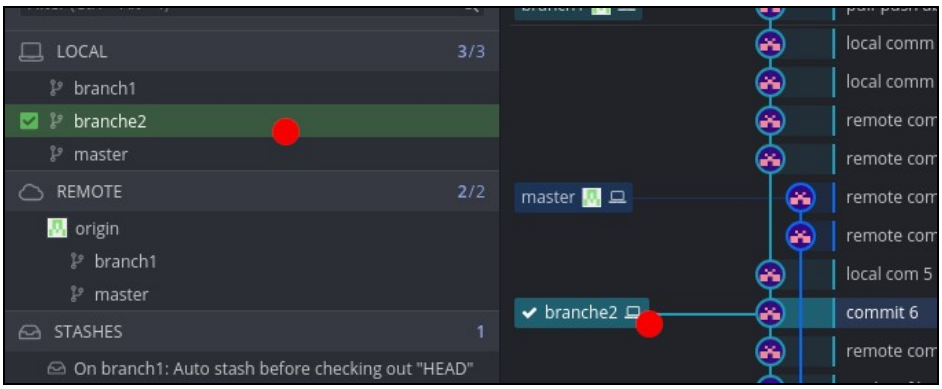

Ha a head mutatót elmozgattuk egy régebbi commit-ra, akkor oda fogja létrehozni a branch-et ahova a head mutat.

### branch létrehozása tetsz?leges commit-ról

Tetsz?leges commit-ról lehet branch-et készíteni a commit fában. Ehhez az adott commit jobb-click menüjében válasszuk a 'Create branch here' lehet?séget:




Majd írjuk be az új branch nevét. Ekkor a head az új branch-re fog ugrani, vagyis ez a branch lesz a checked out branch. Ez természetesen csak a lokális repoban fog létezni, ahhoz hogy a távoliba is bekerüljünk PUSH-olni kell.




Láthatjuk, hogy a zöld pipa rákerült az újonnan létrehozott 'branch2'-re, ami csak a lokális branch listában szerepel, a remote-ban nem.

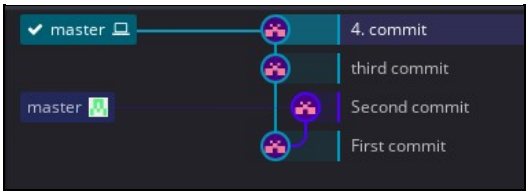
## Merge

# Pull és Push m?ködése

https://support.gitkraken.com/working-with-repositories/pushing-and-pulling/
A merge és rebase stratégiák nem csak két branch egyesítése közben értelmezett, akkor is mikor egy meglév? branch-en kiadjuk a pull ill. a push parancsot. Ne feledjük el, hogy a git lokálisan is fenntart egy repository-t, amibe commit-al tudunk változásokat bejuttatni. Tehát az SVN-el ellentétben a változásokat a lokális repository-ba kell beadni, majd a PUSH-al ill PULL-al szinkronizáljuk a lokális és távoli repo-t.
A lokális repo-nk egy másolata a távoli repo-nak, minden branch-re azt az utolsó commit-ot tartalmazza, ami a PULL pillanatában a legújabb volt. Aztán ahogy telik az id?, mind a távoli, mind a lokális repositor-nkba ugyan azon a branch-en keletkezhetnek új commit-ok ezért tekinthetjük úgy hogy hogy a push és a pull is két branch-et egyesít, a pull a távoli branch-et mergeli a lokális repo-ra míg a push a lokálisat mergel-i a távolira.

**Note**
Fontos azt is érteni, hogy a Push és Pull a lokális és távoli commit-okat szinkronizálja, a stage-ben lév? fájlokhoz és a lokálisan módosított még stage-be sem került fájlokhoz semmi köze. Szigorúan véve a pull futtatásához nem volna szabad hogy stage-elt vagy lokálisan módosított fájlok legyenek nálunk, mert a merge után fejbe fogja vágni a working tree-t is. Itt jön a képbe a **stach**, lásd itt Stash

A GitKraken a következ? képen mutatja a lokális és távoli repo-k eltérését:



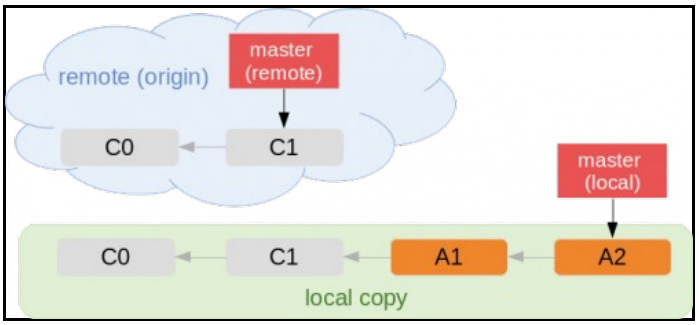

A lentebbi kék doboz az origin (original repository, vagyis ahonnan a lokálisat klónoztuk) a fentebbi zöld doboz pedig a lokális mutatónk. Láthatjuk, hogy a távoli repo-ban a master branch-en vagy a 'second-commit' ami lokálisan hiányzik, viszont lokálisan van 2 új commit, ami a távolin nincs meg.

# Push

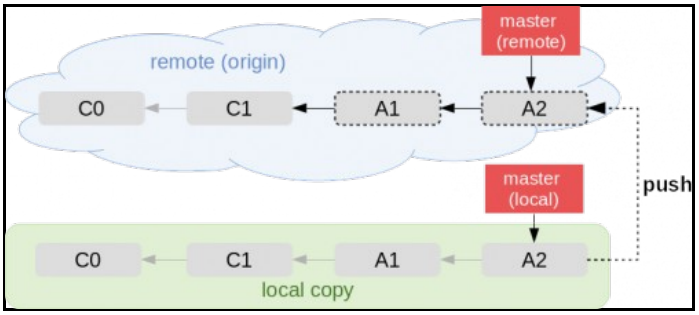
## Fast-forward push

**Note**
A push csak akkor tud lefutni, ha a csak lokálisan létez? commit-okat fast-forward merge-el egyesíteni lehet a távoli repo-val

A távoli repo-ban akkor tudunk egyszer?en push-olni, ha ott nincs olyan commit ami a lokálisan nincs meg. Nézzük a következ? példát. Láthatjuk, hogy a távoli repo-ban nincs újabb commit a branch-en mint a lokálisban, viszont a lokálisban született két új commit, az A1 és A2.

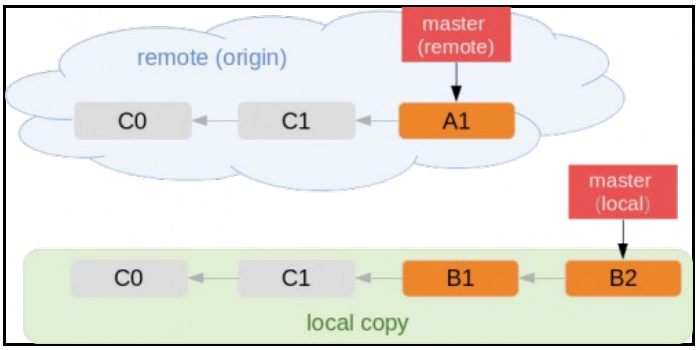



Ebben az esetben a push elvégezhet? 'fast-forward' merge-el. A git az A1 és A2 commit-okat a távoli C1 után fogja másolni, majd a remote-master head mutatót a távoli A2-re fogja állítani.

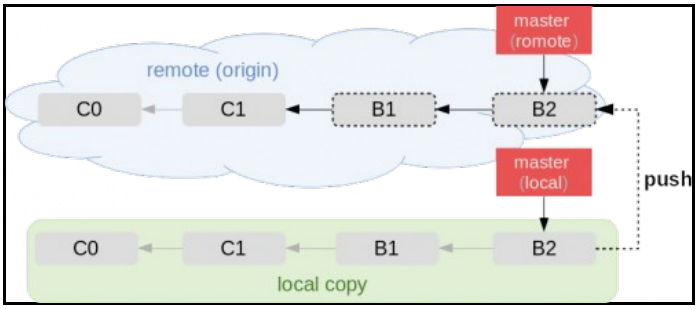



## Force push

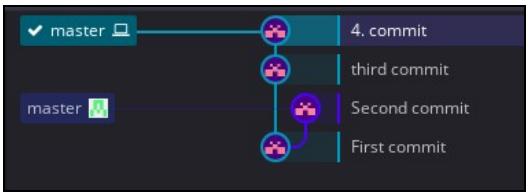
Ha a távoli és lokális branch-en is történt módosítás, akkor a push nem lehetséges, mert egy 'fast-forward' merge-el nem tudja a git egyszer?en felmásolni a lokális módosításokat. Ebben az esetben az egyik lehet?ségünk a 'Force push', ami hatására a teljes távoli branch-et és commit histor-yt helyettesíteni fogja a git a lokális branch-l és lokális commit history-val, így minden távoli olyan változtatás ami lokálisan nem volt meg el fog veszni. Tegyük fel, hogy a távoli branch-en létrejött az A1 commit, ami már lokálisan nem létezik, és lokálisan létrejött a B1 és B2 commit, ami a távoli branch-en hiányzik. Mivel mind lokálisan mind távol is vannak új commit-ok, a 'Fast-forward' push nem lehetséges.



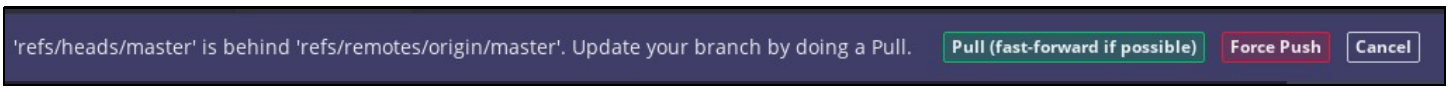

Ha a 'Force push' lehet?séget választjuk, akkor a távoli A1 commit el fog veszni, mivel az nem volt meg lokálisan.




Nézzük hogy néz ez ki GitKraken-ben. Tegyük fel hogy adott az alábbi Commit tree:

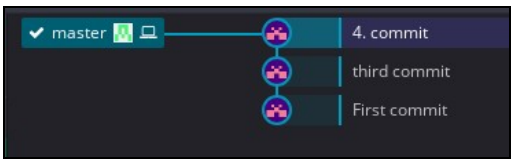



Láthatjuk, hogy lokálisan hiányzik a 'Second commit' viszont lokálisan van két extra commit-unk ('third commit' és '4.commit'). Ha most rányomunk a Push-ra a fels? menüben, akkor az alábbi figyelmeztetést fogjuk kapni:




Egyrészt a GitKraken felajánlja hogy els?ként futtassunk egy PULL-t (amib?l jelen esetben 3-way merge lenne, lásd a Pull fejezetben), vagy futtathatjuk a drasztikus **Force push**-t. A Force-push a teljes távoli repository commit history-t felülírja a lokális commit history-val, úgy hogy minden olyan commit el fog veszni, ami lokálisan nem volt meg. Ez egy visszavonhatatlan lépés. A GitKraken figyelmeztet is rá:


Ha tényleg a Force push-t választjuk, akkor a fenti példában a 'Second commit' (ami nem volt meg lokálisan, csak a távoli branch-en létezett) elt?nik, és az új log fa így néz ki:




### Konfliktus push esetén

..TODO

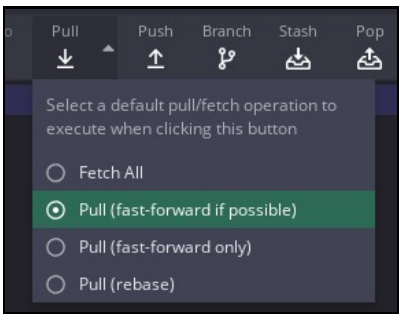
## Pull

### Pull alapok

Pull esetében 4 lehet?ségünk van. Ebb?l 3 valóban merge-li is a táli repo-t a lokális repora. Bármelyiket is választjuk ebb?l a 3 lehet?ségb?l, ha konfliktus mentesen lefut a Pull, akkor a lokális repository-nk biztosan olyan állapotba kerül, hogy Push-olni lehet a remote repo-ba, mivel ott biztosan

elég lesz a fast-forward merge. (Részletek lentebb)

GitKraken-n a négy opció az alábbi módon van felkínálva:




A kiválasztott opció lesz a default m?ködlés ha rányomunk választás nélkül a Pull-ra.

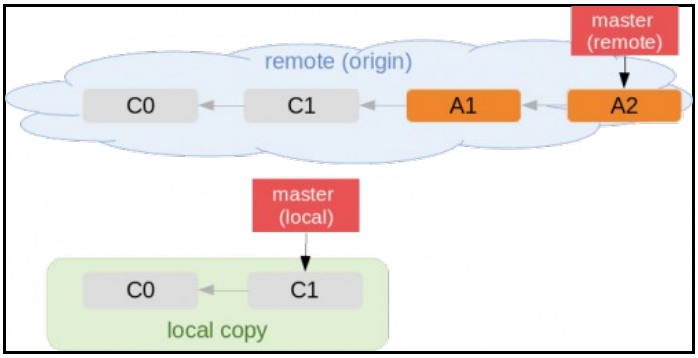
### Fetch

Nem frissíti a lokális repot, csak letölti a commit "meta" adatokat a táli repo-bol, hogy ki tudja rajzolni a commit tree-t.

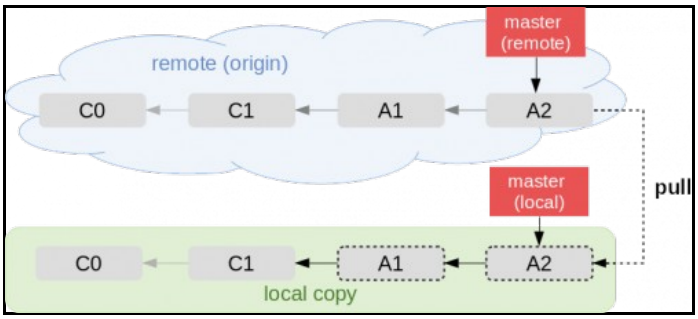
### Fast-forward only

Ez megfelel a fast-forward Push m?ködésnek, csak fordítva. Vagyis a távoli branch-ben vannak új commit-ok a lokálishoz képest, de lokálisan nincs olyan commit, ami a remote-ban ne szerepelne, s?t lokálisan semmilyen még nem commitált módosítás sem létezhet. Fast-forward merge esetében a lokálisan hiányzó commit-okat a git hozzá fogja biggyeszteni a lokális branch végéhez, majd a lokális branch mutatót át fogja állatni. (lokális fast-forward). Mikor majd Push-olni akarjuk a csak lokálisan létez? commit-okat, a git-nek nem lesz más dolga, mint hogy ezeket ráf?zze a távoli repoban a branch végére. A 'Fast-forward only' pull nem fog sikerülni ha nem lehet simán el?re mozgatni a mutatót, vagyis egy merge commit-ra lenne szükség mert lokálisan is történt módosítás.

Tegyük fel, hogy a távoli repoban lév? branch-en létrejött az A1 és A2 commit, amik nincsenek meg még a local repo-ban.

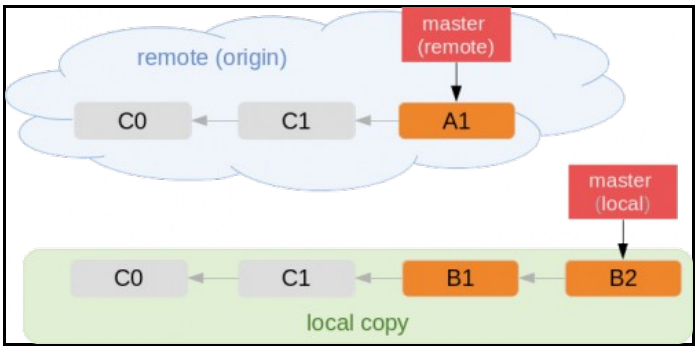



Ekkor ha végrehajtjuk a 'fast-forward' pull-t, akkor a git a lokális C1-re rá fogja f?zni az A1 és A2-t, majd a head mutatót az A2-re fogja el?re mozgatni (forward).

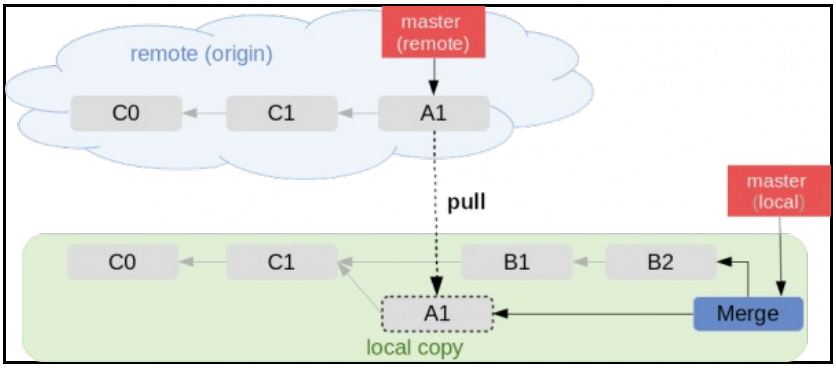



### Fast-forward if possible

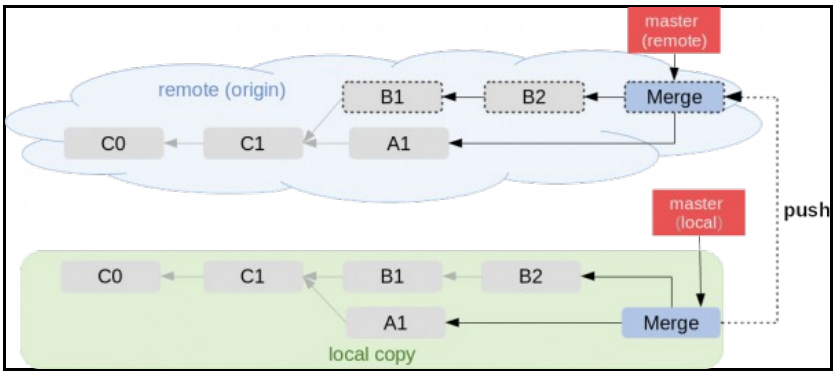
Ebben az esetben meg fogja próbálni a fast-forward-ot, de ha nem lehetséges, akkor a 3 utas merge-t fogja alkalmazni, és létre fog hozni egy merge commit-ot, aminek a nevében is benne lesz, hogy ez miért keletkezett. Tegyük fel, hogy a távoli branch-en létrejött az A1 commit, ami már lokálisan nem létezik, és lokálisan létrejött a B1 és B2 commit, ami a távoli branch-en hiányzik. Mivel mind lokálisan mind távol is vannak új commit-ok, a 'Fast-forward' pull nem lehetséges.



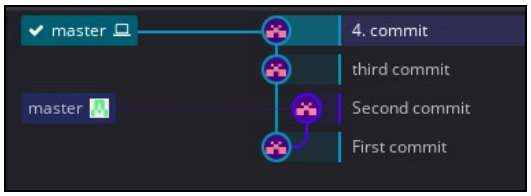

Ekkor a git els?ként le fogja tölteni a távoli A1-et majd lokálisan az A1 és B2-b?l létre fog hozni egy merge commit-ot, amire rá fogja állítani a head mutatót:




A pull után már ki tudjuk adni a push parancsot, mivel egy fast-forward merge-el felmásolható a lokális merge commit és a hiányzó B1 és B2.

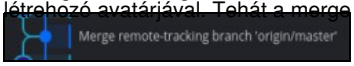



GitKrakenben ez a következ? képen néz ki. Tegyük fel, hogy adott a következ? commit tree. Láthatjuk hogy lokálisan létrejött a 'third commit' és '4. commit' míg a távoli branchen a 'remote commit'.



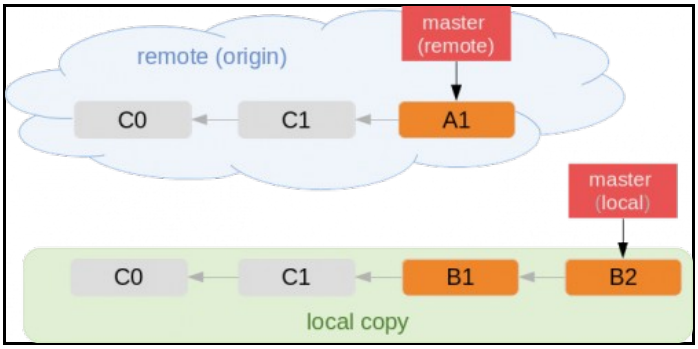

Ha erre kiadjuk a 'Fast-forward if possible' pull parancsot, akkor a git létre fogja hozni a 'Merge remote tracking branch..' merge commit-ot.


Fontos, hogy a merge commit csak lokálisan fog létezni addig amíg nem nyomunk egy push-t is rá.
A merge-öt a két ág találkozásánál csak egy pötty-el jelöli a GitKraken ellentéten a normál commit-al, amit egy nagyobb kör jelöl benne a commit-ot létrehozó avatárjával. Tehát a merge commit-ot nem látszik hogy ki hozta létre:

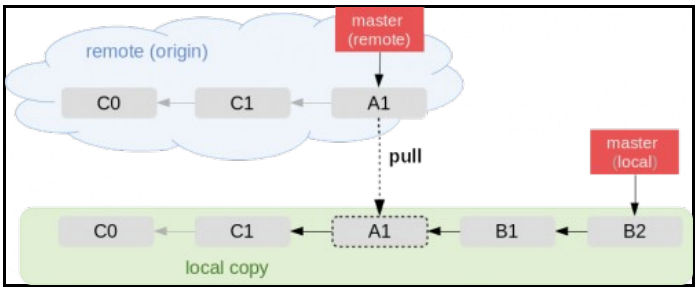


### Rebase azonos branch-en

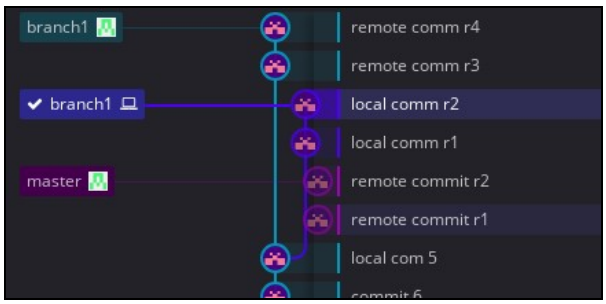
Ennek csak akkor van értelme, ha a távoli repo-ban és az újban is vannak új commit-ok, ezért nem lehet fast-forward merge-t alkalmazni. Lokálisan, a távoli utolsó commit-ra rá fogja f?zni a lokális új commit-okat, így nem lesz plusz leágazás a commit-logban, az egész egy folytonos vonal esz, viszont elveszik az az információ, hogy a remote és a local elmászott egymástól (ami egyáltalán nem baj, tisztán tartja a commit history-t). Tegyük fel, hogy a távoli branch-en létrejött az A1 commit, ami már lokálisan nem létezik, és lokálisan létrejött a B1 és B2 commit, ami a távoli branch-en hiányzik. Mivel mind lokálisan mind távol is vannak új commit-ok, a 'Fast-forward' pull nem lehetséges.




Ekkor ha a 'Pull (rebase)' lehet?séget választjuk, akkor a git le fogja tölteni a távoli új commit-okat, és ezt be fogja ékelni a lokális régi és új commit-ok közé, vagyis az A1-et beékeli a C1 és B1 közé. Úgy is mondhatjuk, hogy az új commit-okat ráf?zi a távoli módosításokra (amíg nem nyomunk push-t is, addig csak lokálisan)



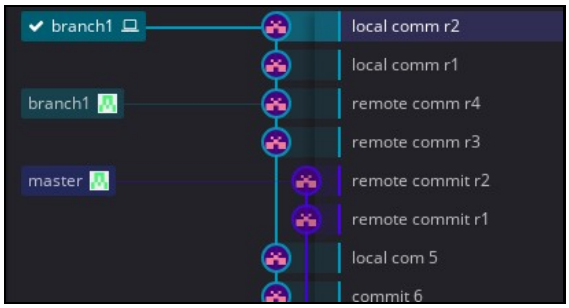

GitKraken-ben ez a következ? képen néz ki. Tegyük fel hogy mind lokálisan mind a távoli repoban 2-2 commit történt:




A távoli repo-ban a 'branch1'-en a 'remote comm r3' és r4. Ezt jelöli a fels? zöld 'branch1' téglalap: 

Lokálisan 'local comm r1' és r2 commit-ok voltak. Jelenleg a lokális repó a 'local comm r2'-re mutat, mivel mellette van a pipa és a kis monitor szimbólum: 

. A közös ?s a 'local comm 5'.
Ha most 'Pull (rebase)'-t választjuk, akkor a távoli repo 'remote comm r4'-re rá fogja ültetni a 'local comm r1'-et. (lokálisan)



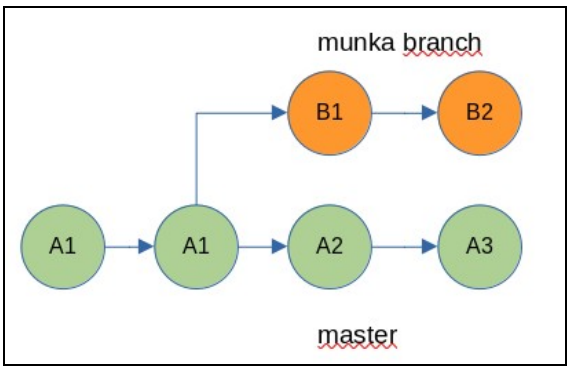

A rebase után nem fog többet látszani, hogy a 'local comm r1'-nek valójában a 'local comm 5' volt az ?se.
Ha Push-t is nyomunk, akkor a változásokat már egy egyszer? 'Fast forward' merge-el el is fel lehet juttatni, hiszen nincs más dolga a git-nek, mint el?re mozgassa a mutatót:


A 'Force push'-al ellentétben itt nem veszett el a távoli 'remote comm r3' és r4.

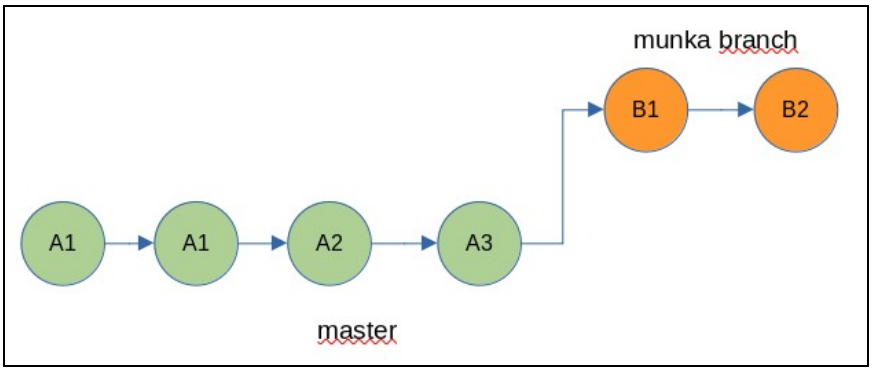
### Rebase két külön branch-en

Ha van egy master branch, ahova mindent visszavezetünk, és van egy külön munka branch-ünk, amin dolgozunk, akkor gyakori esemény, hogy a master branch-en annyira el?re haladtak a dolgok, hogy a munka branch-et már nem lehetne mergelni a masterre. Ebben az esetben a munka branhc-et rebase-leni kell a master-re, vagyis rá kell rakni a tetejére.

Alább látható, hogy a master-re rákerültek újabb comit-ok a munka branchhez képest.




Azt szeretnénk, hogy a munka branch-et rebase-eljük a master tetejére, tehát: rebase: munka -> master
A végeredmény így fog kinézni.

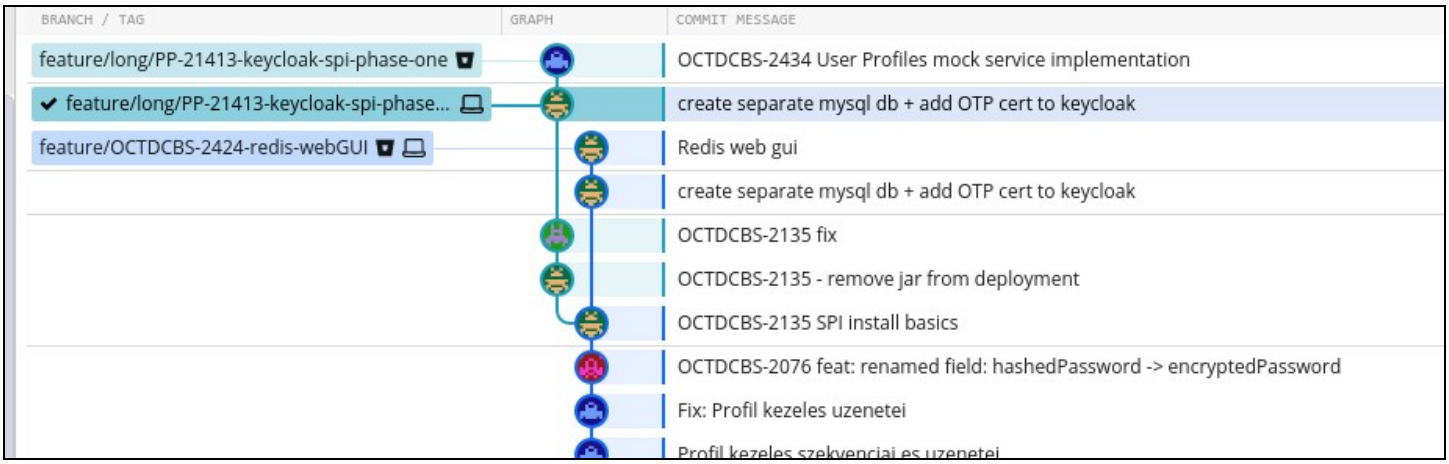



Egy él? példa:

- master: Van egy master branch-ünk: PP-21414. (a lokális mutató egyel hátrébb van mint a remote, a remote a lényeg).
- munka: És van egy munka branch-ünk: redis-webGUI

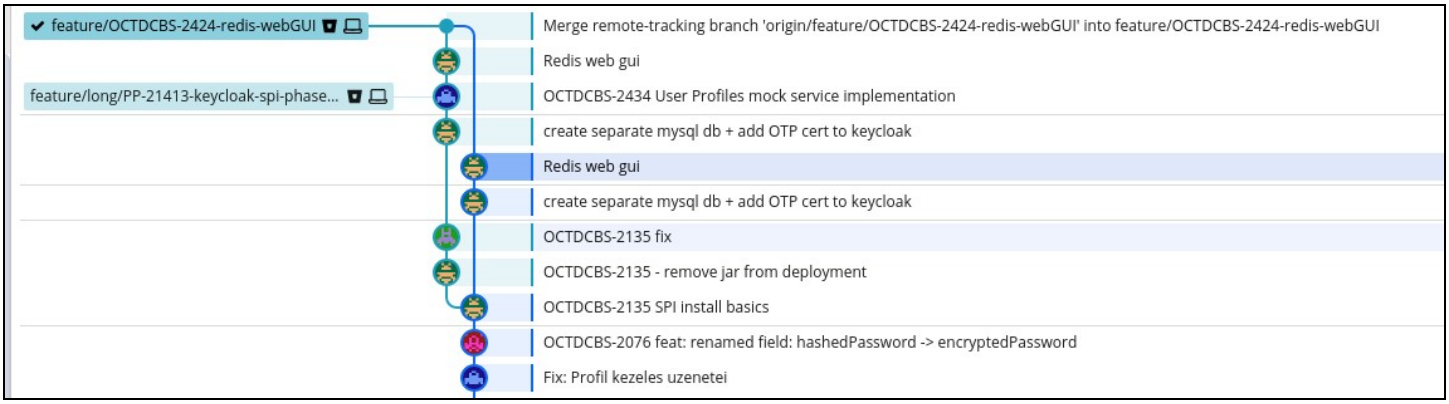
A leágazás óta 4 commit van a master-en a munka branch-hez képest. De van egy kis trükk. A master-en a 3. commit (ahol a lokális mutató épp áll) az megegyezik a munka branch 2. commit-jával 'mysql db +..'.
Ez azért van, mert a 'redis-webGUI' branch egy másik munka branch-b?l származik. Ennek a másik munka branch-nek az utolsó commit-ja a 'create separate mysql db..' volt.

1. Ki lett húzva egy munka branch a master?l az 'Install Basic' commit után. Ennek a neve az volt hogy 'munka-branch1'. Ez már nem látszik a fában, mert már törölve lett: long/PP-21.. -> 'munka-branch1'
2. A 'munka-branch1'-en született egy commit: 'create separate mysql db'.
3. Ki lett húzva egy új branch (redis-webGUI) a 'munka-branch1' branch-b?l: 'munka-branch1' -> 'redis-webGUI'
4. Az új branch-en (redis-webGUI) született egy darab commit 'Redsi web GUI' címmel.
5. A 'munka-branch1' branch mergelve lett (pull request-el) a master-re (long/PP-21..), majd törölve lett. merge 'munka-branch1' -> 'long/PP-21..'
6. A törölt 'munka-branch1' branch egyetlen commit-ja (create separate mysql db) megjelent a master tetején.
7. A master-en született egy új commit: 'User profile mock..'.
8. Elhatároztuk, hogy a 'redis-webGUI' munka branch-et rebase-eljük a master tetejére.




És mi a végeredmény:

1. A 'redis-webGUI' branch most már a master-b?l n? ki. Mivel a 'redis-webGUI' els? commit-ja már rajta volt a masteren korábban, ezért az a commit nem került ráf?zésre a master-re, csak a 'Redis web gui' commit lett ráf?zve. Most már a 'redis-webGUI' branch tartalmazza az összes változtatást ami id? közben a master-en történt (három ilyen commit van).

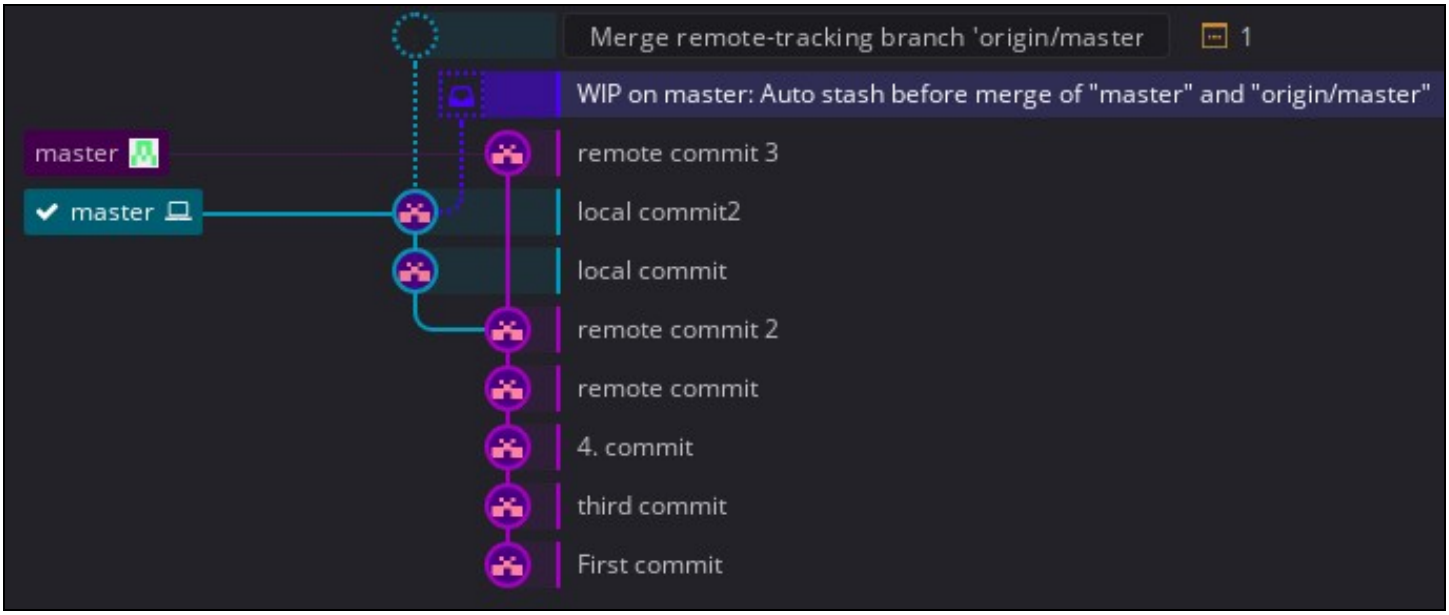



### Auto stash

... TODO ...

### Konfliktus PULL-esetén

...



A fenti képen conflit volt a pull-ban. Amit feloldottam. Most látható, hogy három águnk van. Lilával láthatjuk a remote branch-et. Zölddel láthatjuk a lokális két commit-unkat, amit még nem push-oltunk. A local-commit 2-b?l két felé ágazik a gráf. A sötétkék négyzet alakú ág az automatikusan mentett stash, amiben a pull-merge el?tti állapota található a még nem commit-ált fájloknak. Jelen esetben ebben most csak a file1.txt-ban. Ha megnézzük a tartalmát, akkor láthatjuk, hogy ebben csak a lokális módosítás van benne. A gráf legtetején zöld szaggatott körrel található a merge eredménye, ami most csak a file1.txt feloldott konfliktusát tartalmazza úgy ahogy megjelöltük a diff során. Az auto-stage-be rakott eredeti fájlon három m?veletet hajthatunk végre:

- Apply stage
- Pop stage
- Delete stage

# Stach

# Tag

# Pull request

A 'Pull request'-el kérhetjük meg a távoli repository tulajdonosát, hogy nézze át az általunk készített módosítást (review), és ha megfelel?nek találja, akkor merge-ölje be a változtatásokat a saját branch-ébe. Ugyan a 'Pull request' nem része a core git szabványnak, megtalálható mind a négy nagy git szolgáltatónál (GitHub, GitLab, Butbucket és AzureDevOps). Ha a master branchen (vagy bármilyen kiemelt, központi branchen) be van kapcsolva a 'pull request' funkció, akkor a lokális branch-böl nem lehet közvetlen Push-al feljuttatni a változtatásokat a távoli repoba.

1. Létre kell hozzunk egy új fork-ot a távoli master branch-böl
2. Clone-ozni kell az új fork branch-et, hogy lokálisan is meglegyen.
3. El kell végezni lokálisan a módosításokat, majd lokálisan commit-lni.
4. Push-olni kell a lokális fork branch-en a commit-okat a távoli fork branch-re.
5. Fel kell adni egy pull request-et a távoli fork branch-r?l a távoli master branch-re.

Ha a review-el jóváhagyja a változtatásainkat, akkor merge-ölni tudja azokat a távoli master branch-be. A review és merge felületet már mindig a git szolgáltatók adják, ez már nem része a GitKraken-nek.

# Troubelshooting

## File watcher failed to start for this repository

https://techsparx.com/blog/2018/02/gitkraken-inotify.html<brr>

Type this command:

```
$ cat /proc/sys/fs/inotify/max_user_watches
8192
```

This is the limit on your computer.

Each inotify watch consumes a modest amount of memory. On a 64-bit computer like this one, each consumes 1 KB, so 8,192 watches consumes about 8 MB of memory. On a 16GB main memory computer that's a drop in the bucket.

Temporarily increasing the limit is this simple:

```
# echo 99999 > /proc/sys/fs/inotify/max_user_watches
```

After which you'll get this:

```
$ cat /proc/sys/fs/inotify/max_user_watches
99999
```

To make a permanent change, set fs.inotify.max_user_watches= in sysctl settings. On some systems (Debian/Ubuntu/etc) those settings are in /etc/sysctl.conf and on some others there will be a file in /etc/sysctl.d.

After editing the sysctl settings, run this:

```
# sysctl -p
fs.inotify.max_user_watches = 99999
Putting it on one line:

# echo fs.inotify.max_user_watches=99999 | sudo tee -a /etc/sysctl.conf && sudo sysctl -p
```

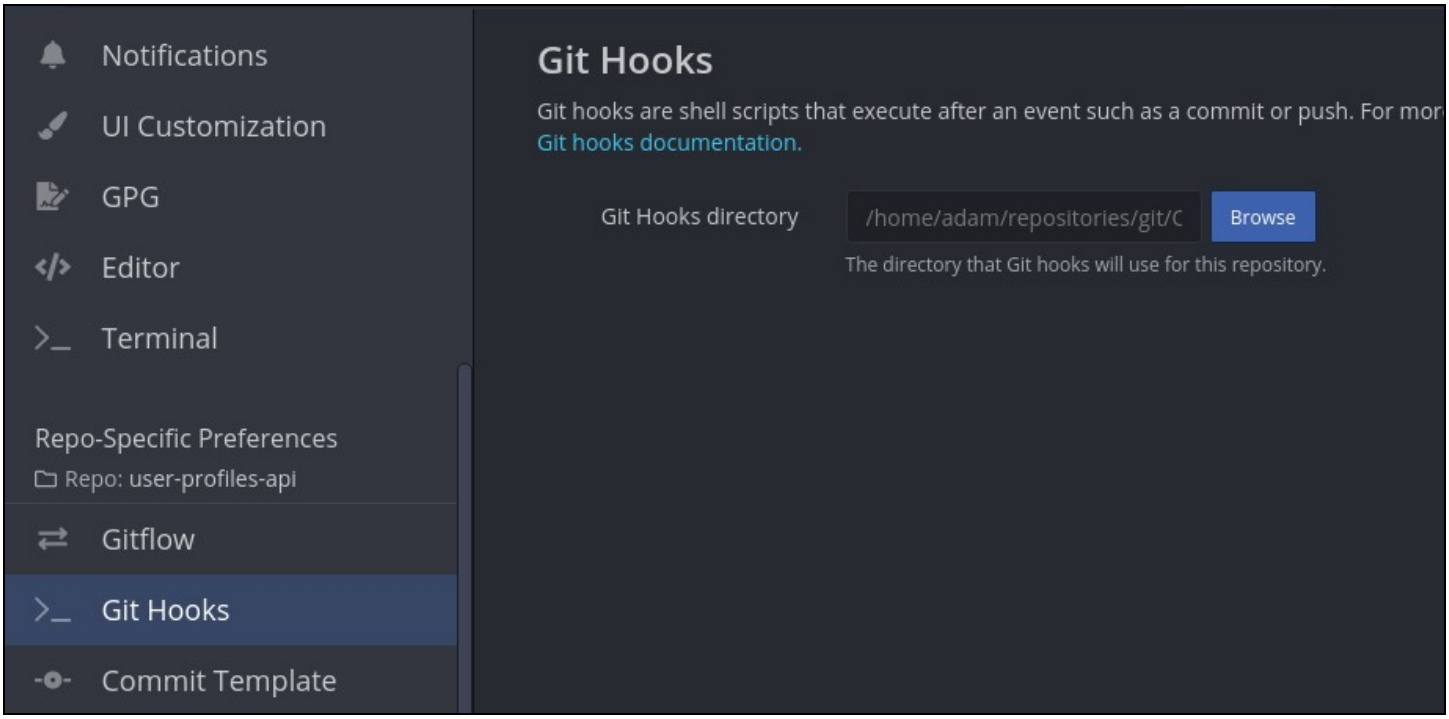
Or on certain other systems:

```
# echo fs.inotify.max_user_watches=99999 | sudo tee /etc/sysctl.d/40-max-user-watches.conf && sudo sysctl --system
```

## Pre-receive hook declined

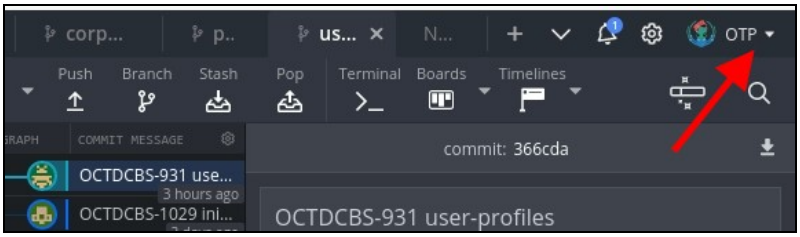
A pre-receive hook-ok két helyen lehetnek beállítva.

1. Lokálisan, az adott repository .git/hooks mappában. Itt olyan sh script-eket kell berakni, amik ellen?rzik a commit-ot. A GitKraken-ben itt van:

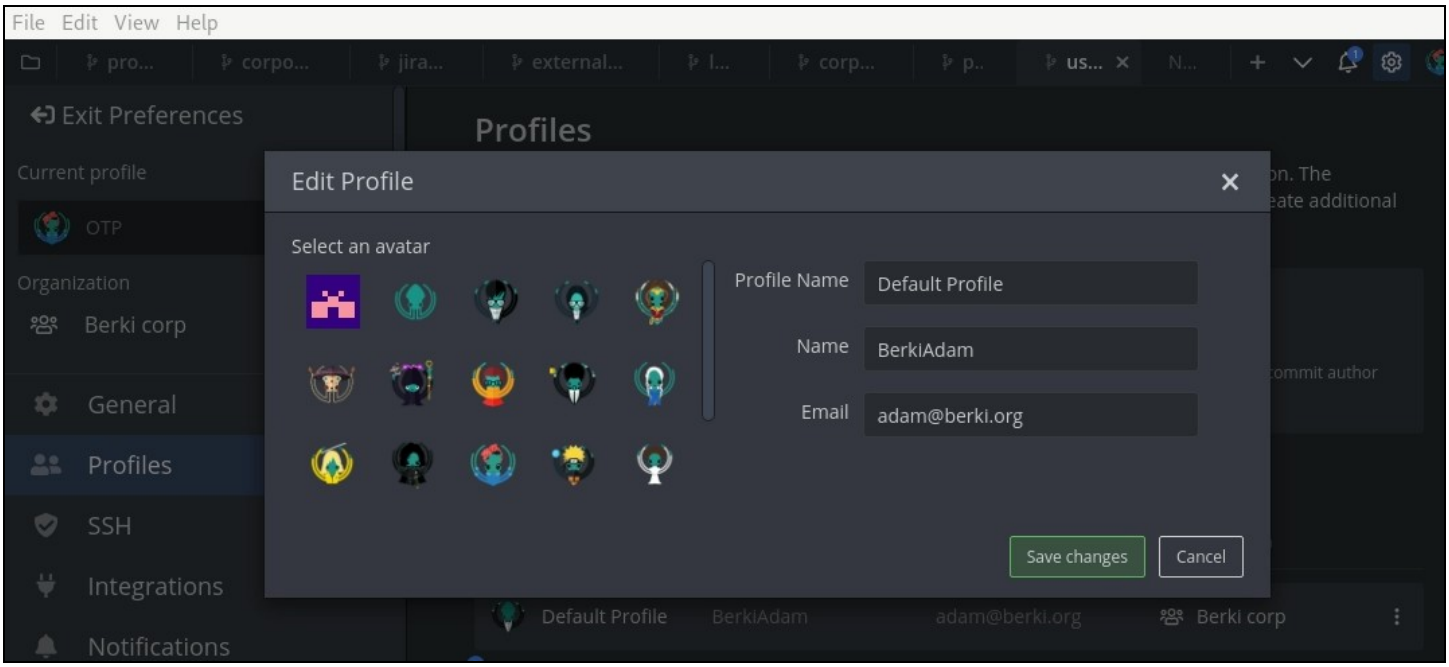



2. A távoli repoóba, ahova push-olni akarunk.

Ha ezek közül egyiket sem szegjük meg, akkor az lehet a baj, hogy a commit-ban lév? username/email nem ugyan az mint a távoli repóban, pl BitBucket-ben.




A profil-t a jobb fels? sarokban választhatjuk ki. Minden profil tartalmaz egy nevet és egy email címet. A profilokat a preferences/profiles-ban állíthatjuk be:




Bitbucket szerver esetén itt az az email cím kell és név, ami a Bitbucket profilban meg van adva.

**Warning**
A commit készítésekor kiválasztott profil számít, nem a push-kor kiválasztott profil. Ha a commit-ot rossz profillal csináltuk, akkor nem számít hogy a push alatt mi van kiválasztva

A rossz profillal készített commit-ot úgy vonhatjuk vissza, hogy az el?z? commit-ra jobb klikk, majd: