GIT elosztott verziókezelő rendszer

Timár András

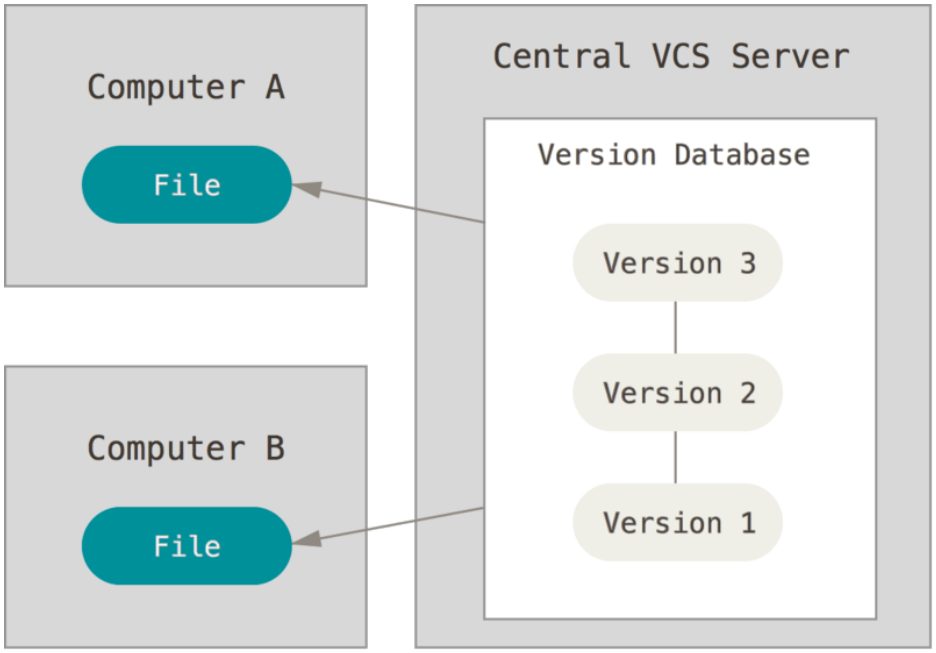
Alapok

Fogalmak

- Repo(sitory): verziókövetett tároló
- Commit: egy új verzió rögzítése a repóban
- Branch: egy ág a verziókövetési fában

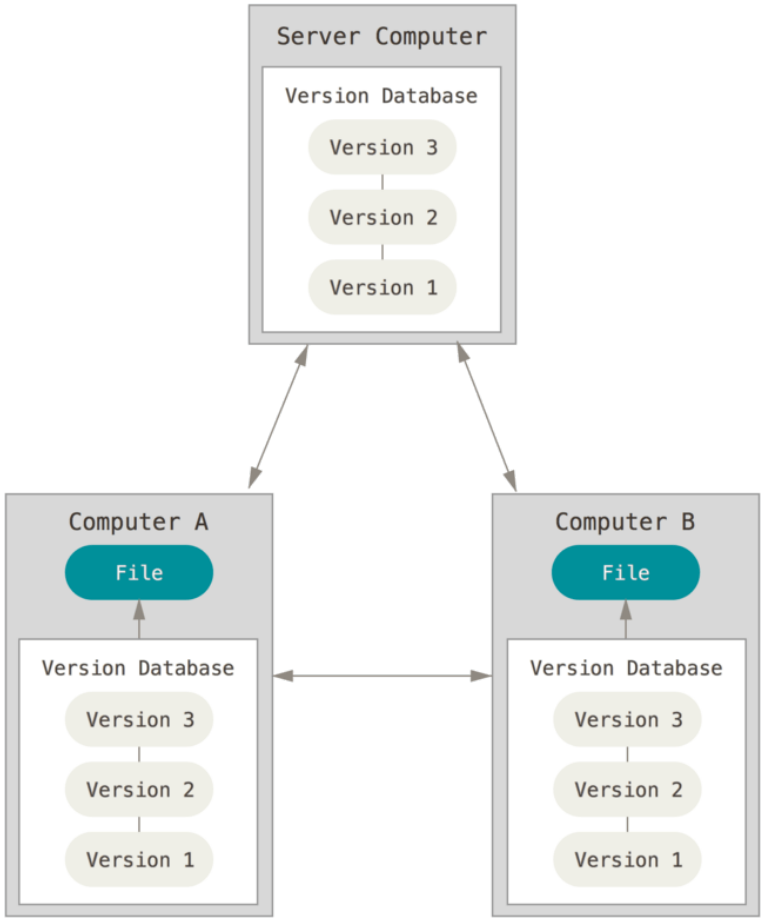
Centralizált verziókezelő rendszerek

- CVS, SVN, Perforce
- Egy központi szerveren tárolódik a verziótörténet
- Ha kiesik a szerver, senki nem tud commitolni
- Ha hardver hiba van, minden elvész
- *Single Point of Failure*



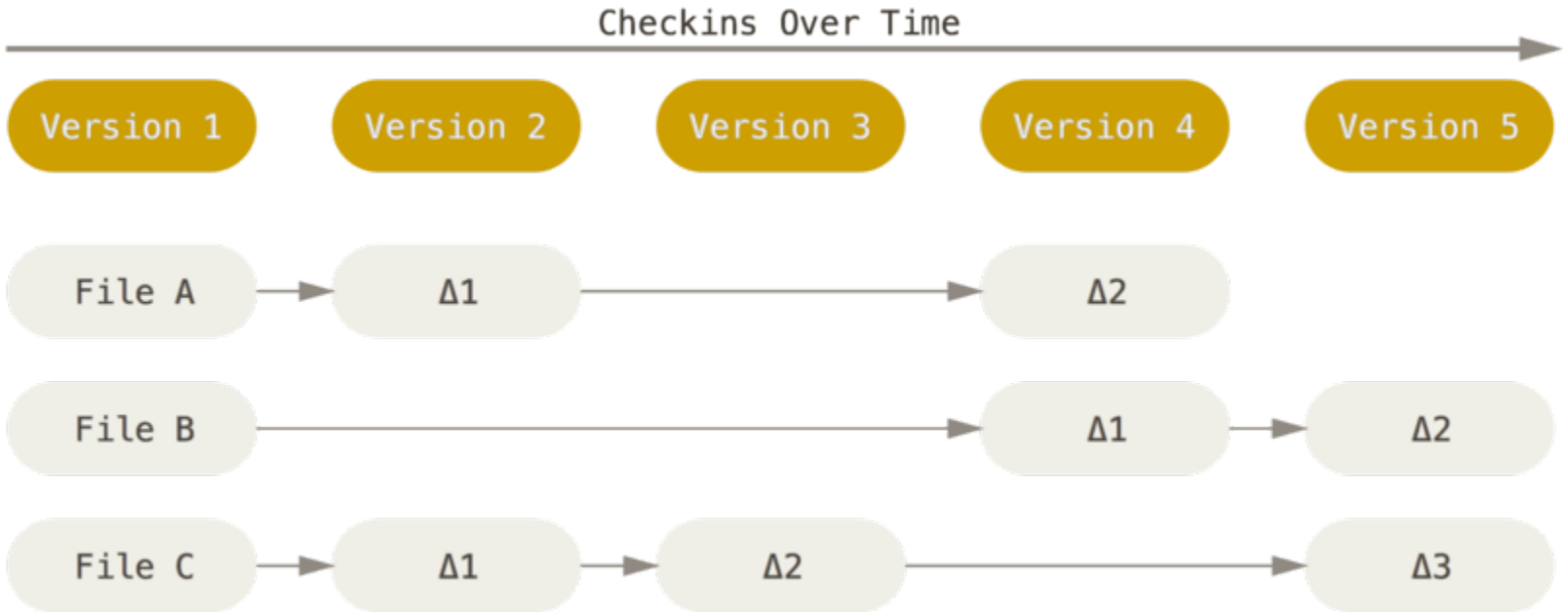
Elosztott verziókezelő rendszerek

- Git, Mercurial (Hg), Bazaar, Darcs.
- A teljes repó jelen van mindenkinél, nincs *single point of failure.*
- Bárkinek a repójából visszaállítható a teljes verziótörténet.



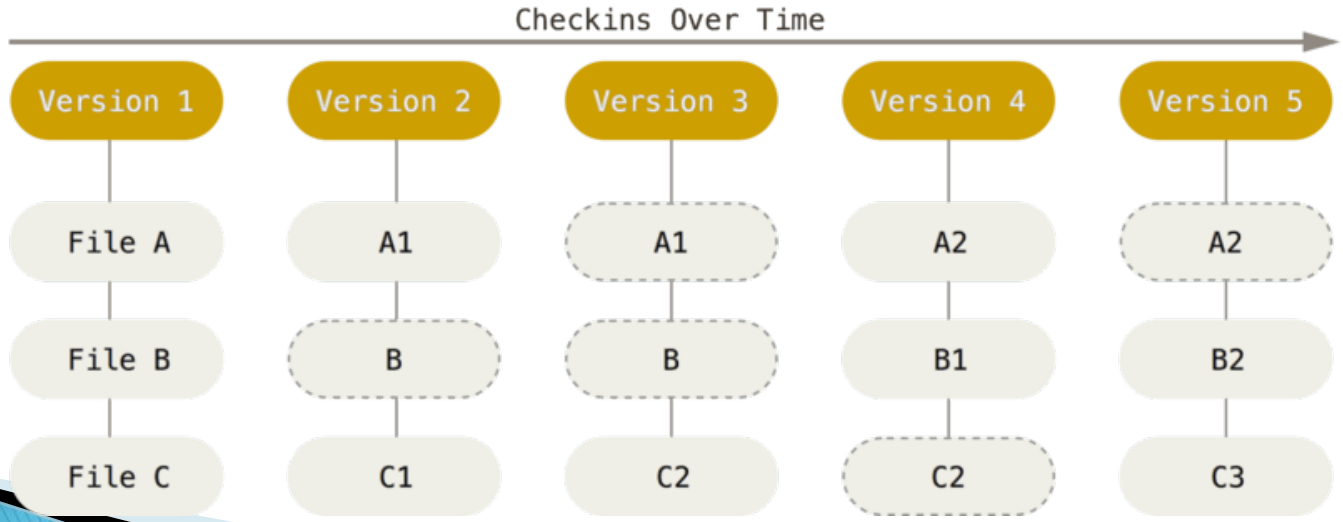
A Git működése

- Nem diffeket tárolunk egy első verzióhoz képest, mint itt…



A Git működése

- …hanem pillanatképeket (snapshot)
- A verziótörténetet a snapshotok láncolata adja
- Ami nem változott, annak a legutóbbi verziójára hivatkozunk



A műveletek lokálisak

- Szinte minden művelet helyileg történik
- A teljes verziótörténet a helyi gépen van
- Gyors műveletek
- Hálózati kapcsolat nem kell
- Lokálisan commit-oljuk a változásokat, és amikor lesz net, akkor frissítjük a szervert

Integritás

- A Git mindenről ellenőrző-összeget (checksum) készít commit-kor
- Később mindenhol erre a checksum-ra hivatkozik, pl:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

- Vagy röviden:

```
24b9da6
```

Biztonság

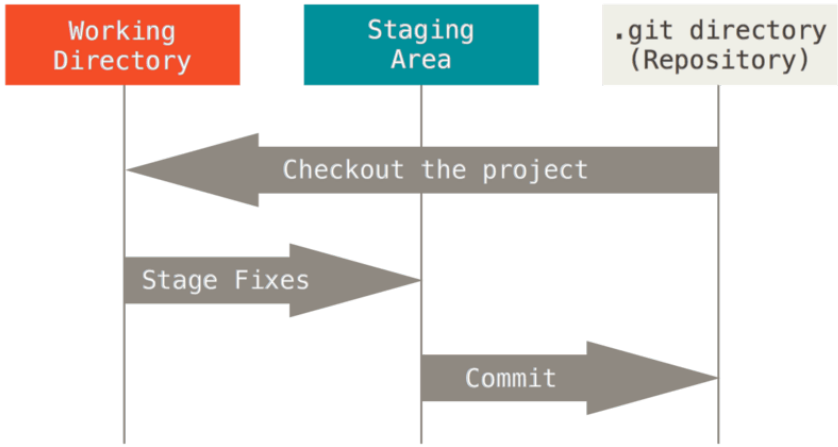
- Minden Git művelettel újabb adat kerül a rendszerbe (nincs törlés)
- Ezért minden visszafordítható
- Lehet nyugodtan kísérletezni, nem tudunk elrontani semmit
- Mindig vissza lehet állítani az előző állapotot

A három állapot!

- Git-ben egy fájlnak három állapota lehet
  - Committed
  - Modified
  - Staged
- Committed: az adat el van mentve a helyi repóban
- Modified: a fájl megváltozott, de nincs még elmentve
- Staged: a fájl aktuális állapota megjelölve, hogy kerüljön be a következő commitba

Területek

- .git könyvtár: metaadatok és objektumok (ez a repó valójában)
- Working directory (working copy): a projekt egy verziójának példánya, az adott verziójú fájlok halmaza
- Staging area (INDEX): Ez egy fájl a .git könyvtárban, ami azt tartalmazza, hogy mi fog bekerülni a következő commitba



Munkamenet

1

- A munkakönyvtárban (working copy) megváltoztatunk fájlokat

2

- Fájlok megjelölése commit-ra (=staging). Az aktuális tartalom pillanatképe bekerül a staging területre (INDEX)

3

- Commit. Ekkor az INDEX-ben lévő pillanatkép mentésre kerül a repóba.

Beállítások

- Rendszer szintű (`/etc/gitconfig`)
  - `git config --system`
- Felhasználó szintű (`~/.gitconfig`)
  - `git config --global`
- Repó szintű (repóban kiadva)
  - `git config`
- A mélyebb szint felülírja a magasabb szint beállításait

Beállítások

- Név
  - `git config --global user.name ”John Doe”`
- Email
  - `git config --global user.email ”johndoe@example.com”`
- Szerkesztő
  - `git config --global core.editor geany`

Alapvető Git használat

Meglévő könyvtár verziókövetése

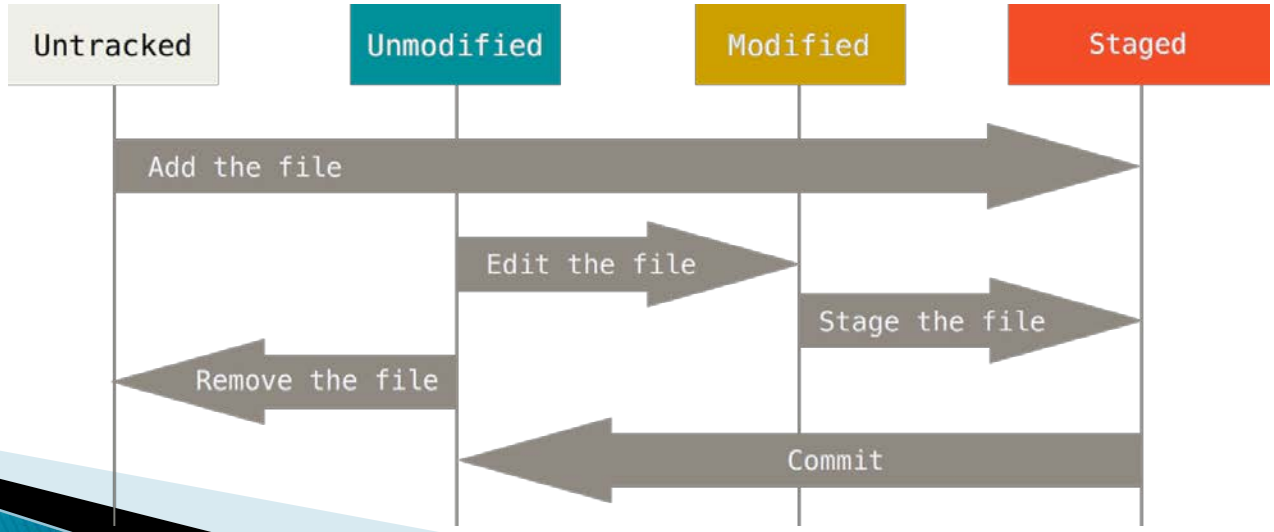
- `git init`
- Létrejön a `.git` könyvtár (rejtett)
- Az aktuális tartalom verziókövetése
  - `git add *.c`
  - `git add README`
  - `git commit –m ”initial commit”`

Meglévő repó másolása

- `git clone https://github.com/libgit2/libgit2`
  - A libgit2 könvtárban létrejön a teljes másolat
- Adott könyvtárba:
  - `git clone https://github.com/libgit2/libgit2 <könyvtárnév>`
- Mindent tartalmaz, ami a szerveren van, akár vissza is állítható belőle.
- A szervernek nincs kitüntetett szerepe

Változások mentése

- A repóban a fájlok kétféle állapotban lehetnek
  - Követett (tracked)
    Azok a fájlok, amik az utolsó pillanatképben (commit) szerepeltek. Ezeket lehet módosítani (modify), visszaállítani (unmodify), és megjelölni mentendőként (stage)
  - Minden más nem követett (untracked)



Státusz

- A repóban lévő fájlok állapota
  - `git status`
- Mutatja
  - a változott fájlokat
  - Az INDEX-ben fájlokat
  - A nem követett (untracked) fájlokat
  - Az INDEX-ben lévő fájlokat, amiket a stage után még módosítottunk

Státusz példa

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:

        modified:   main.c

Changes not staged for commit:

        modified:   README
        modified:   main.c

Untracked files:

        newfile.txt
```

Változott és stagelt, commitkor mentésre kerül

Változott és NEM stagelt, commitkor nem kerül mentésre

Új fájl, még nincs verziókövetve, nem kerül mentésre

Új fájlok követése

- `git add <file|könyvtár|wildcard>`
- Mostantól ezek a fájlok verziókövetettek lesznek
- Ha egy könyvtárat adunk meg, akkor a tartalma verziókövetett lesz
- A `git add` parancs új tartalmat ad hozzá az INDEX-hez. Ez lehet új fájl, vagy módosítás is.

Fájlok kihagyása

- `.gitignore` fájl a repó gyökérben, vagy bármelyik könyvtárban
- Az automatikusan generált fájlokat nem kell verziókezelni, ezért kihagyhatóak.
- Például object fájlok, logok, stb.
- `*.[oa]` -minden `.o` és `.a` kiterjesztésű fájl kihagyása
- *~ - minden ~-mal végződő fájl kihagyása

.gitignore tartalma

- Üres és #-kal kezdődő sorok nem számítanak
- Alap regexek működnek: pl. *?[abc][A-Z]
- / a sor elején: rekurzív bejárás tiltása
  - Pl. /TODO.txt – csak az aktuális könyvtárban lévő TODO.txt-t hagyjuk ki, a subdir/TODO.txt-t nem
- / a sor végén: az adott könyvtár teljes tartalmát kihagyjuk
- ! a sor elején: minta inverze
- a/**/z: minden útvonal a és z között:
  - Pl. doc/**/*.pdf: minden PDF fájl mellőzése a doc könyvtárban és azon belül

Változások összehasonlítása

- `git diff`
  - Az INDEX-et és a munkakönyvtárat hasonlítja össze. Amit változtattunk, de még nem stage-eltünk.
- `git diff --staged`
  - Az INDEX és a legutóbbi commit különbsége
- Ha minden változást stage-eltünk, akkor a `git diff` nem mutat semmit
- `git diff --staged`, hogy lássuk milyen változások kerültek az INDEX-be
- `--cached == --staged`

Külső diff alkalmazás használata

- `git difftool` a `git diff` helyett
  - Külső alkalmazásban nyitja meg a változásokat
  - Pl. kdiff3, diffuse, meld, vimdiff, stb.
- Beállítás
  - `git config --global diff.tool meld`
- Lehetséges eszközök listája
  - `git difftool --tool-help`

Változások mentése

- `git commit`
  - Megnyílik a beállított szerkesztő
- Commit üzenet a parancssorban
  - `git commit -m ”Bug #123 solved”`
- Mindig csak az kerül mentésre, ami az INDEX-ben volt!

Az INDEX kihagyása

- git commit -a
  - A már verziókövetett fájlok változásait automatikusan stage-eli, nem kell `git add`-dal hozzáadni.
  - A nem követett fájlokat ez sem adja hozzá a commithoz, ott kell `git add`!

Fájlok törlése

- `git rm <fájl>`
- Az INDEX-ből eltávolítja a fájlt és le is törli a munkakönyvtárból.
- Ezután a fájl már nem lesz verziókövetett
- Ha csak az INDEX-ből akarjuk kivenni, de a munkakönyvtárból nem akarjuk törölni
  - `git rm --cached <fájl>`

Commit történet

- `git log`
- `git log -2 --pretty=oneline`
  - `-2` : csak az utolsó két bejegyzést mutassa
  - `--pretty=oneline`: formázási opció, egysoros tömör kimenet
- Fa megjelenítés
  - `git log --pretty=format:"%h %s" --graph`

Túl korai commit

- Utolsó commit üzenet megváltoztatása
- Kimaradt egy fájl a commitból
- `git commit --amend`

Eltávolítás az INDEX-ből

- „Ezt a fájlt mégsem akarom, hogy bekerüljön a következő commit-ba”
- `git reset HEAD <fájl>`
  - Kiveszi az INDEX-ből és visszaállítja
  - A következő commitba nem fog bekerülni
- != `git rm <fájl>`
  - a fájl nem lesz többé verziókövetett

Változások visszaállítása

- `git checkout -- <fájl>`
- Visszaállítja az adott fájlt a legutóbbi commit szerinti állapotra a munkakönyvtárban
- Vigyázat! Amit a commit óta módosítottunk a fájlon az elvész!
- (stash és branch később)
- Minden helyi változtatás eldobása
  - `git checkout -f`

Távoli repók

- `git remote`
  - Kilistázza a távoli repókat
- `git remote -v`
  - Mutatja az URL-t is
- `origin`: klónozáskor a távoli repó alapértelmezett neve
- `git remote add <név> <URL>`
  - Új távoli repó hozzáadása

Fetch és pull

- `git fetch [távoli név]`
  - Nem változtat semmin a munkakönyvtárban
  - Nincs automatikus összeolvasztás (merge), kézzel kell csinálni, ha jónak látjuk
- `git pull [távoli név]`
  - Mint a `fetch`, de automatikusan össze is olvaszt
  - Ha ütközés van, azt előbb fel kell oldani

Feltöltés a szerverre

- `git push [remote-name] [branch-name]`
- Pl. `git push origin master`
- Ha valaki időközben push-olt a szerverre, akkor előbb „le kell húzni” az ő változtatásait, mielőtt push-olni tudnánk.
- Itt ütközések lehetnek, ezeket nekünk kell feloldani (mindig annak, aki pusholni akar)

Információk a távoli repóról

- `git remote show [távoli név]`
- Pl. `git remote show origin`
  - Fetch és Push URL-ek
  - Melyik távoli branchek vannak követve
  - push/pull parancsokhoz milyen távoli branch van beállítva automatikusan

Címkézés

- Fel lehet címkézni jelentős állomásokat a Git történetben
- Pl. v1.0 verzió
- A címkék nem mozgathatóak!
- Címkék listázása
  - `git tag`
- Címke létrehozása
  - Egyszerű: `git tag v1.2`
  - Összetett (annotated): `git tag –a v1.4 –m "verzió 1.4"`
- Adott commit címkézése
  - `git tag –a v1.2 <commit hash>`

Címkék megosztása

- Alapból a címkék nem kerülnek fel a szerverre push-kor, azt külön kell kérni
- Egyet: `git push origin v1.5`
- Mindet: `git push origin --tags`

Álnevek

- `git config --global alias.co checkout`
- `git config --global alias.br branch`
- `git config --global alias.unstage 'reset HEAD --'`
  - Pl. `git unstage fileA`

Szokásos nevek

- Összefoglalásképpen
- master: az első commit-kor keletkező ág alapértelmezett neve
- origin: a távoli repó alapértelmezett neve
- HEAD: az aktuális commit, ahol éppen vagyunk a fában (amit éppen checkout-oltunk)
- INDEX: a stage terület másnéven

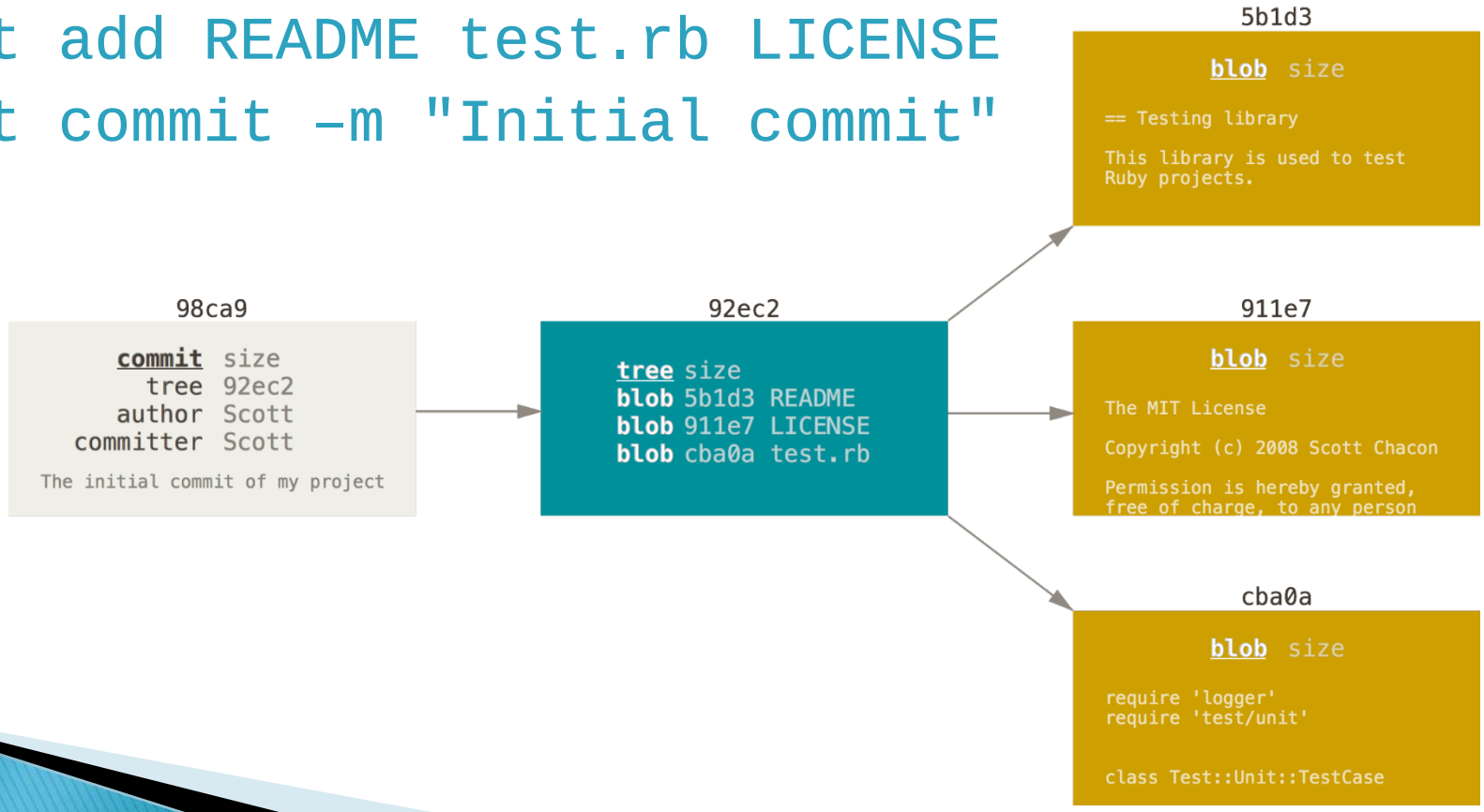
Ágak (branch)

Az ágakról általában

- Az ágak segítségével függetlenné tudjuk tenni a fejlesztést a mainline-tól
- Git-ben a branchek (ágak) sima mutatók egy commitra
- Ezért nagyon gyorsan lehet új ágakat létrehozni, törölni, ágak között váltani
- Tipp: „Branch early, branch often”

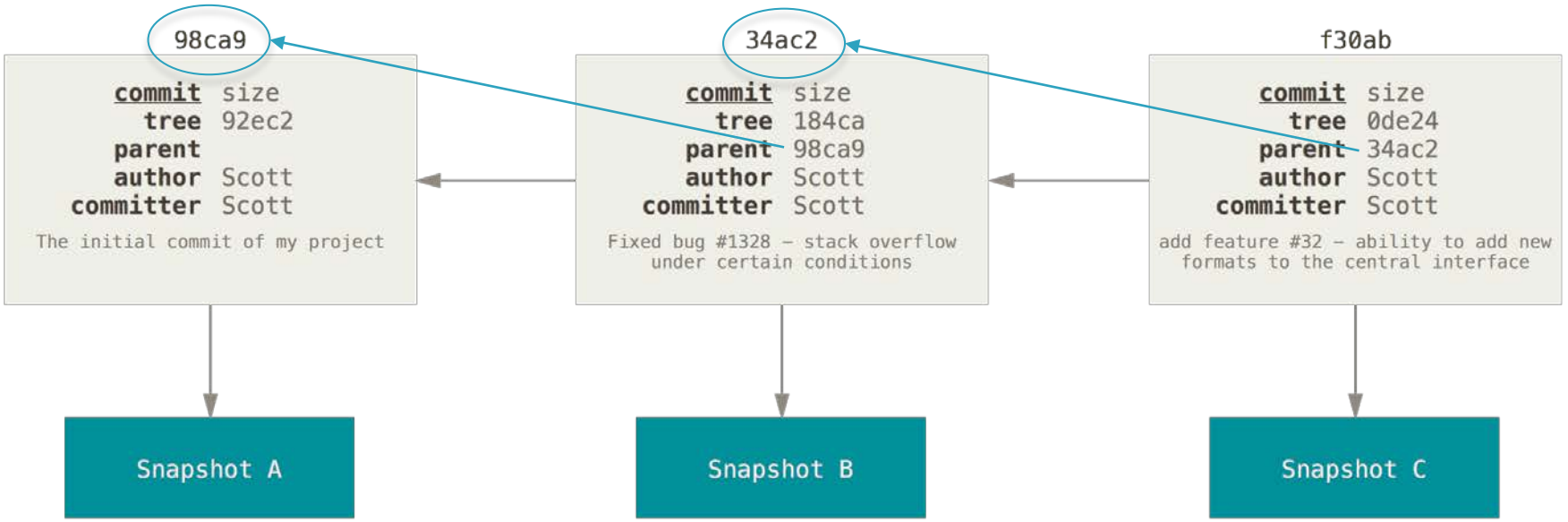
Az ágak belső működése

- Pl. 3 fájl egy könyvtárban (README,test.rb,LICENSE)
- `git add README test.rb LICENSE`
- `git commit -m "Initial commit"`



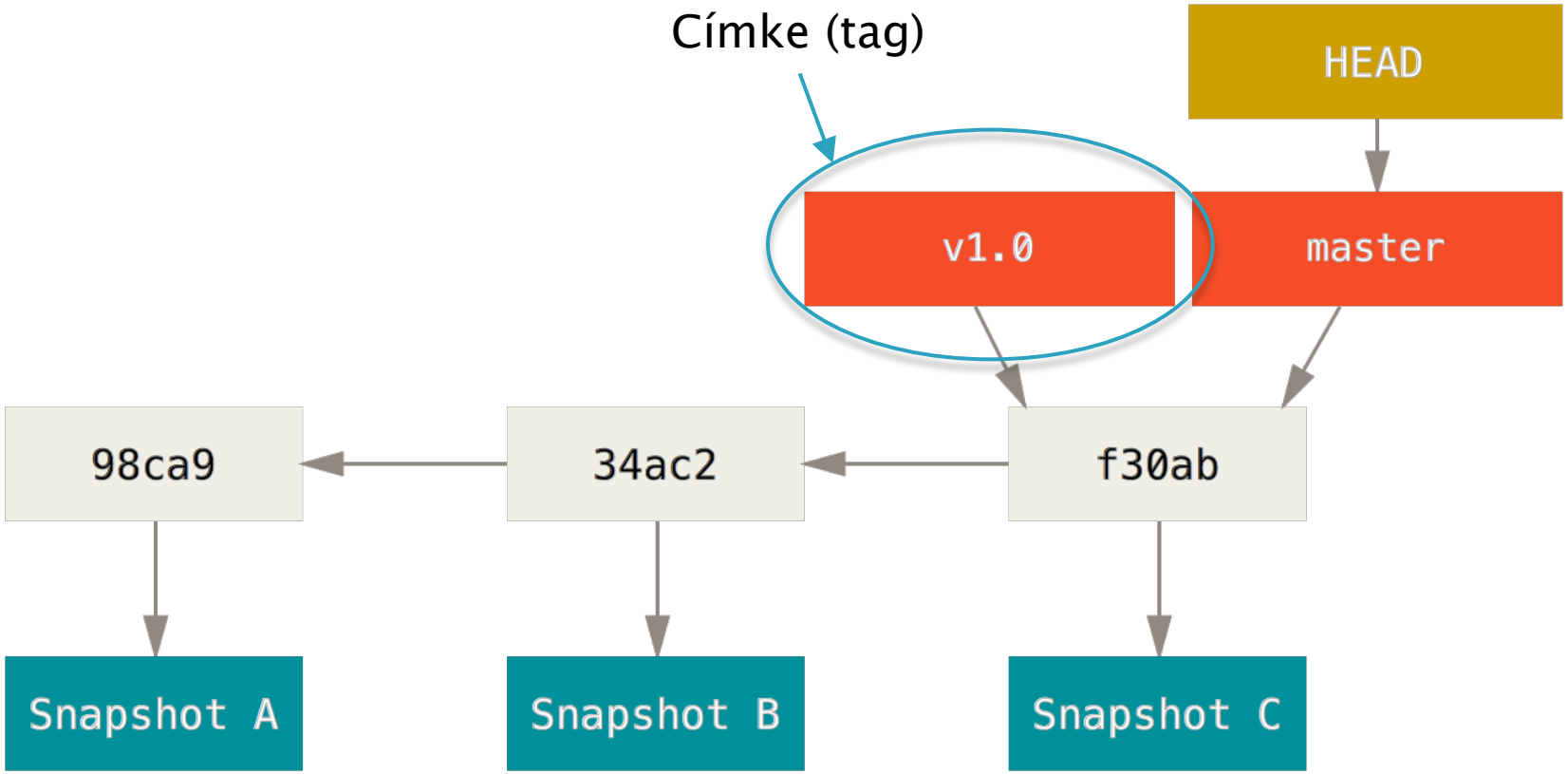
Az ágak belső működése

- Még két commit után…



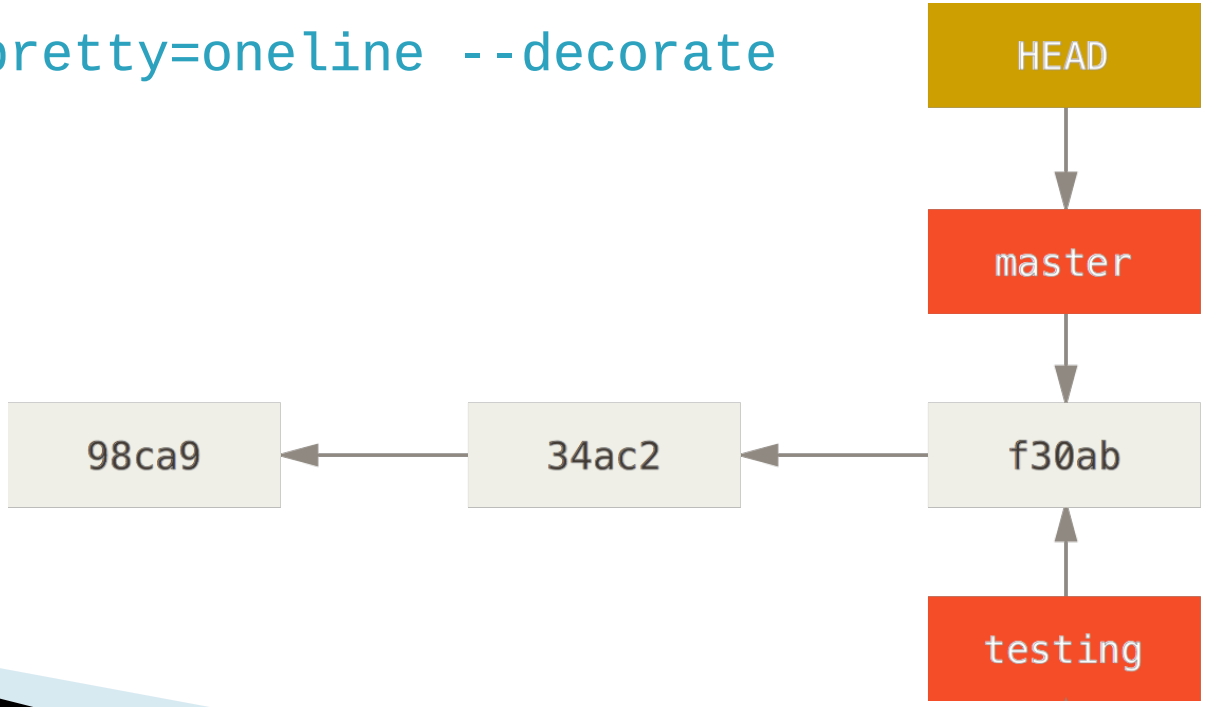
- master: ez az alapértelmezett neve az első ágnak

Az ágak belső működése



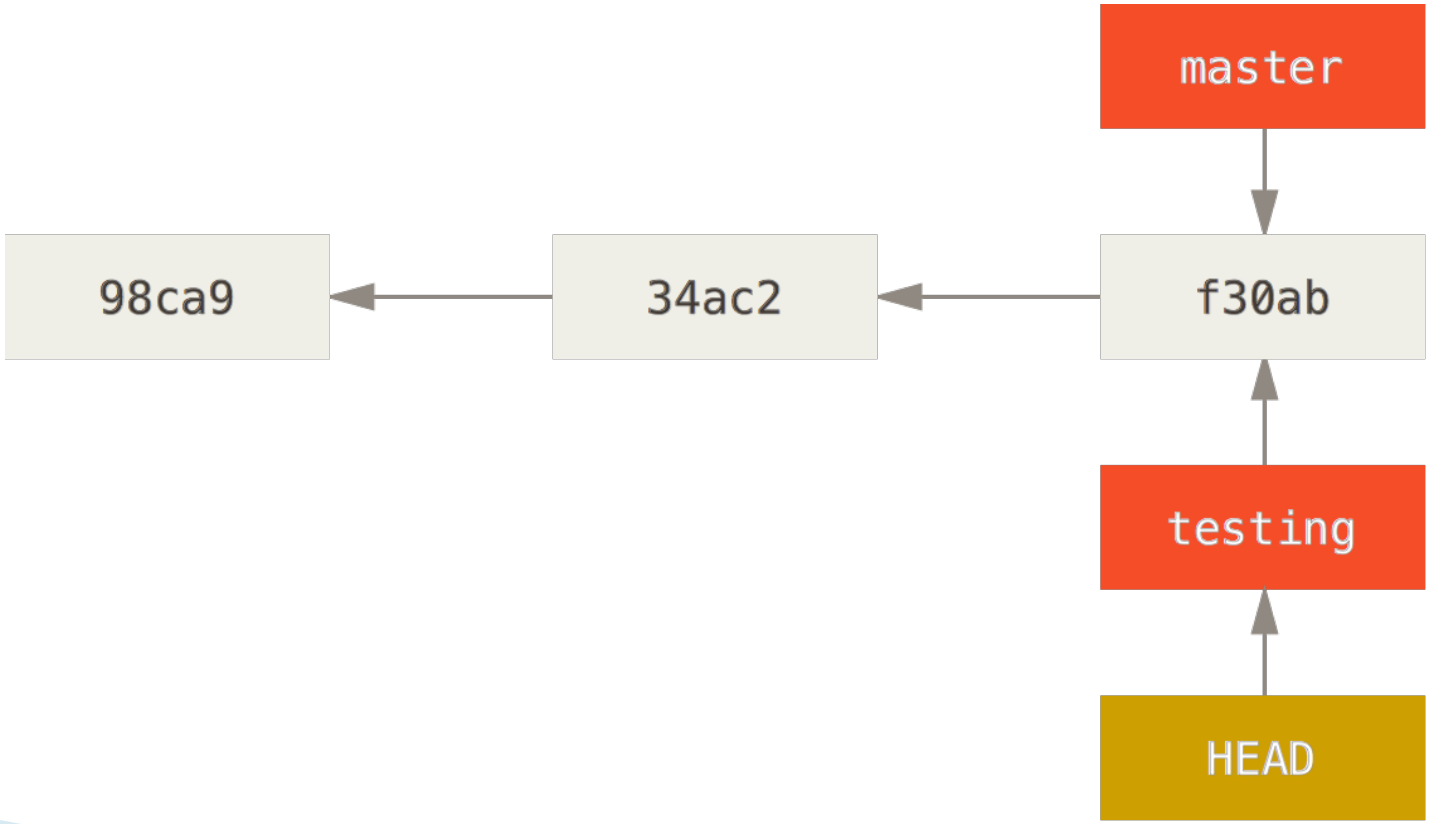
Új ág létrehozása

- Egy új pointer jön létre
- git branch testing
  - Csak létrejön az új branch, de nem vált át rá
- Hol állunk?
  - `git log --pretty=oneline --decorate`



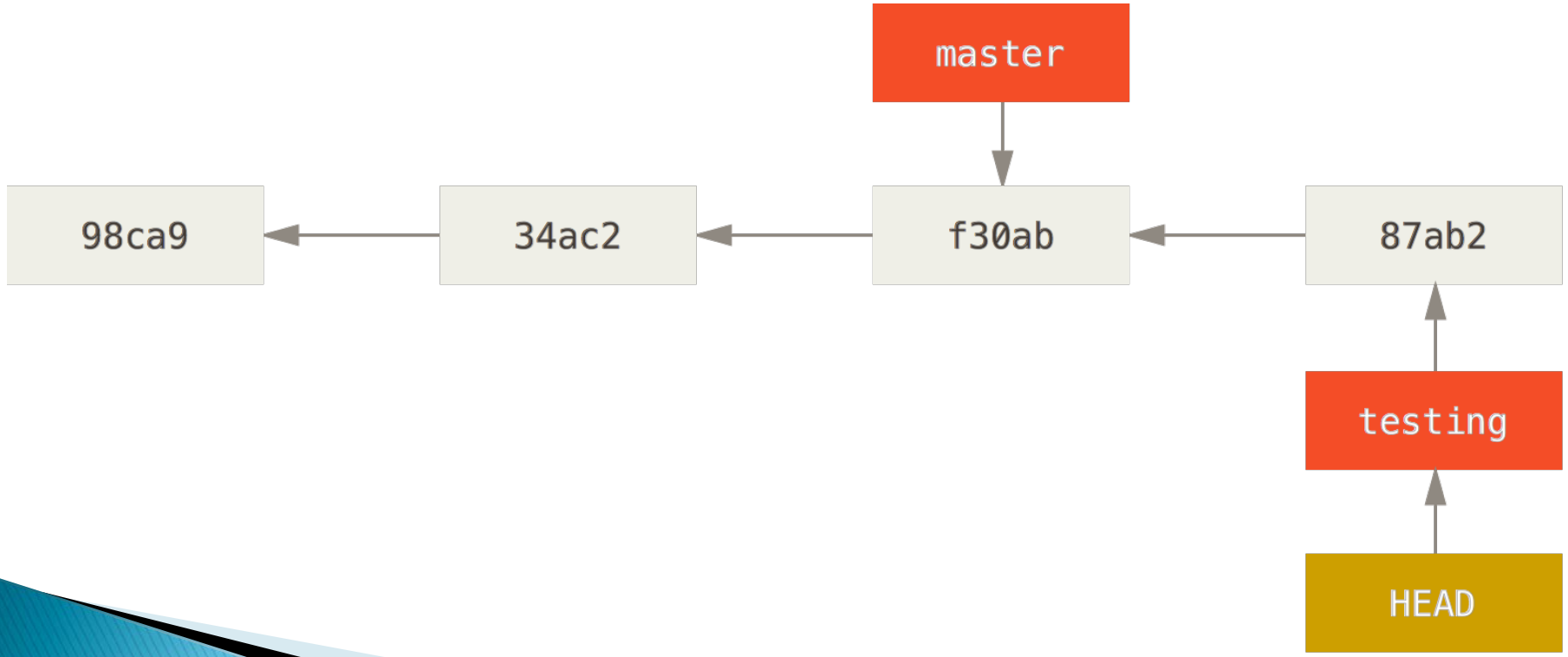
Ág váltása

- `git checkout testing`



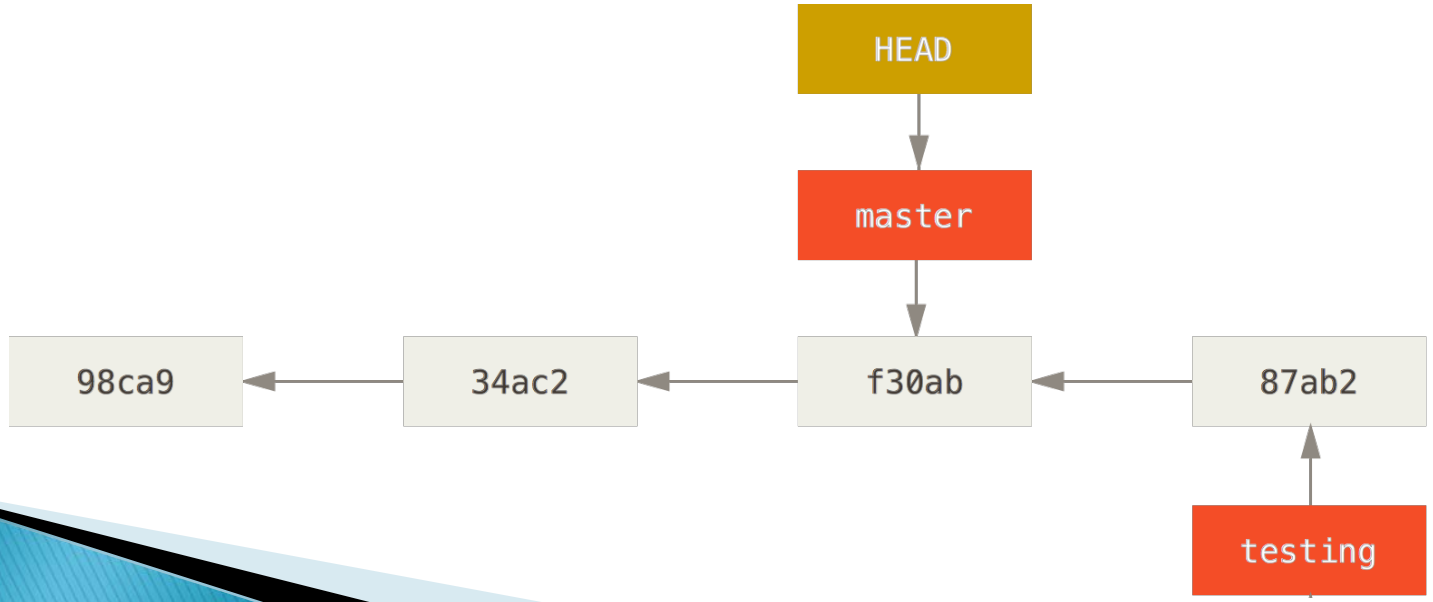
Változtatunk valamit...

- `echo "Hello" > README`
- `git commit –am "Valtozas"`



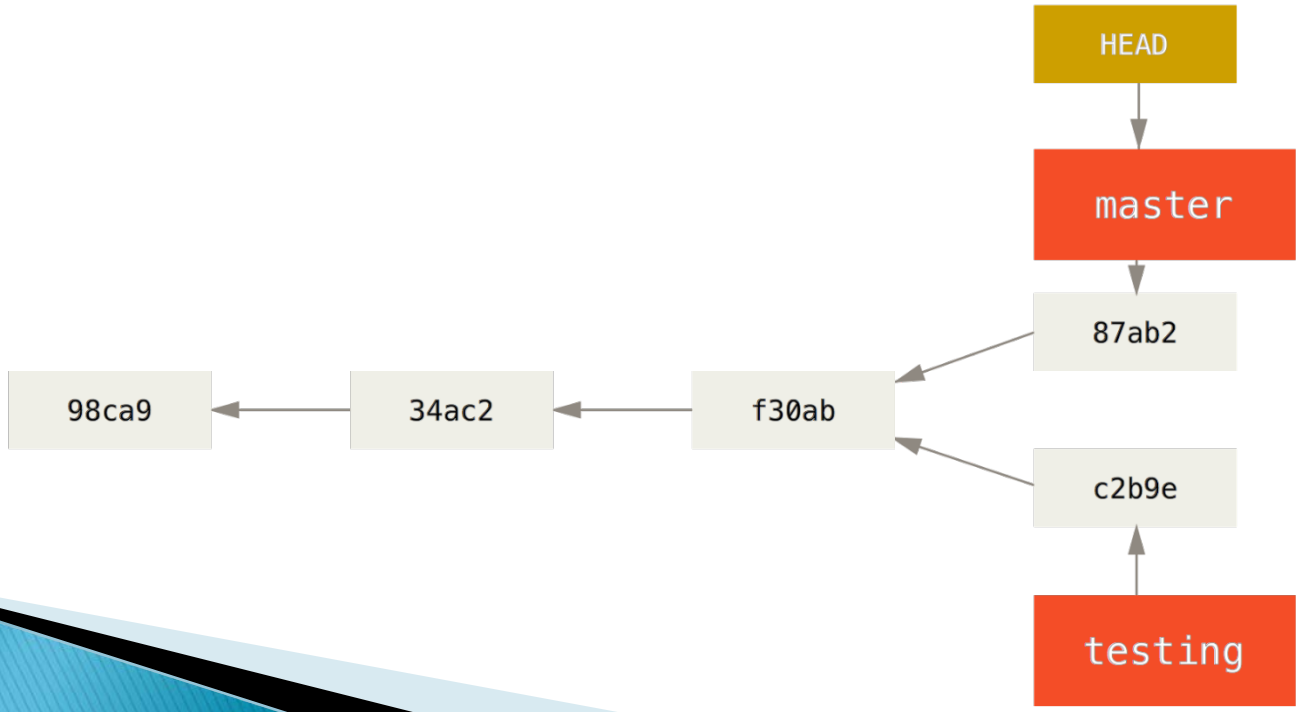
Vissza a master-re

- `git checkout master`
- Két dolog történt:
  - HEAD a master-en áll
  - A fájlok a munkakönyvtárban visszaálltak a master pillanatképére



Elágazás

- `master`-ben is változtatunk
- `echo „Back to master" > README`
- `git commit –am "Valtozas a masteren"`
- `git log --oneline --decorate --graph --all`



Példa munkafolyamat

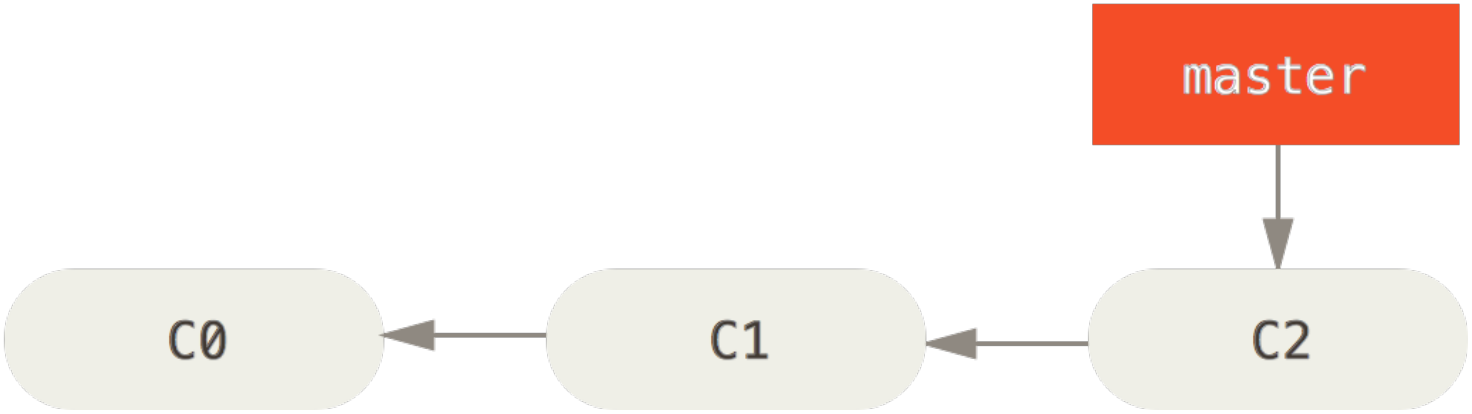
Áttekintés

1. Egy weboldalon dolgozunk
2. Készítünk egy ágat egy új feature-nek
3. Ezen dolgozunk

Itt jön egy kérés, hogy valamit sürgősen meg kell javítani a kódban

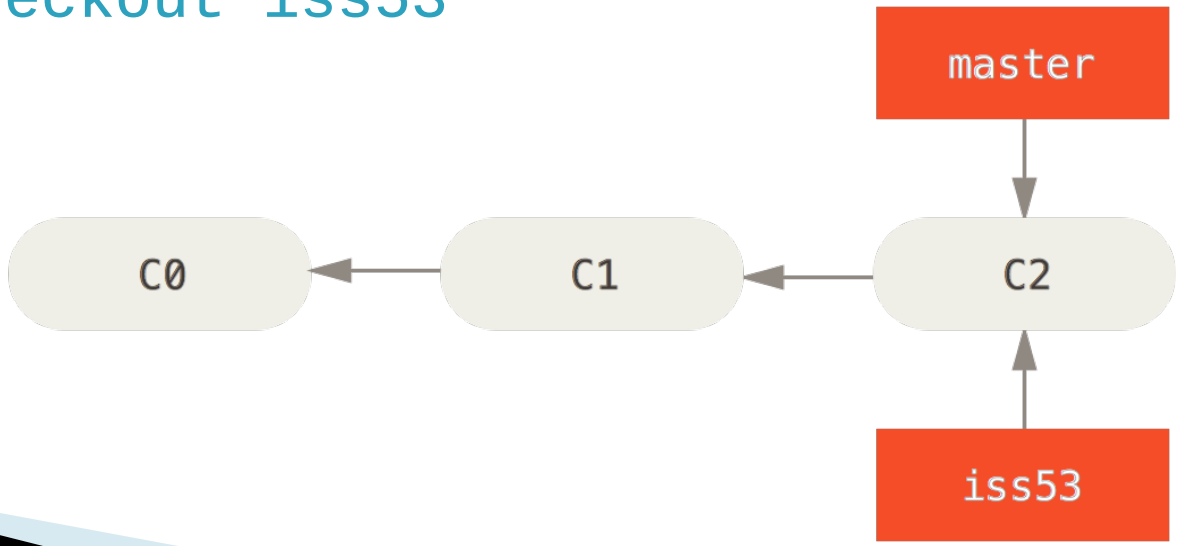
1. Átváltunk a fő ágra
2. Készítünk egy új ágat a hotfix-nek
3. Ha kész, beolvasztjuk (merge) a hotfix ágat és felküldjük (push) a fő ágba
4. Visszaváltunk az előző munkánk ágára és folytatjuk

Alaphelyzet



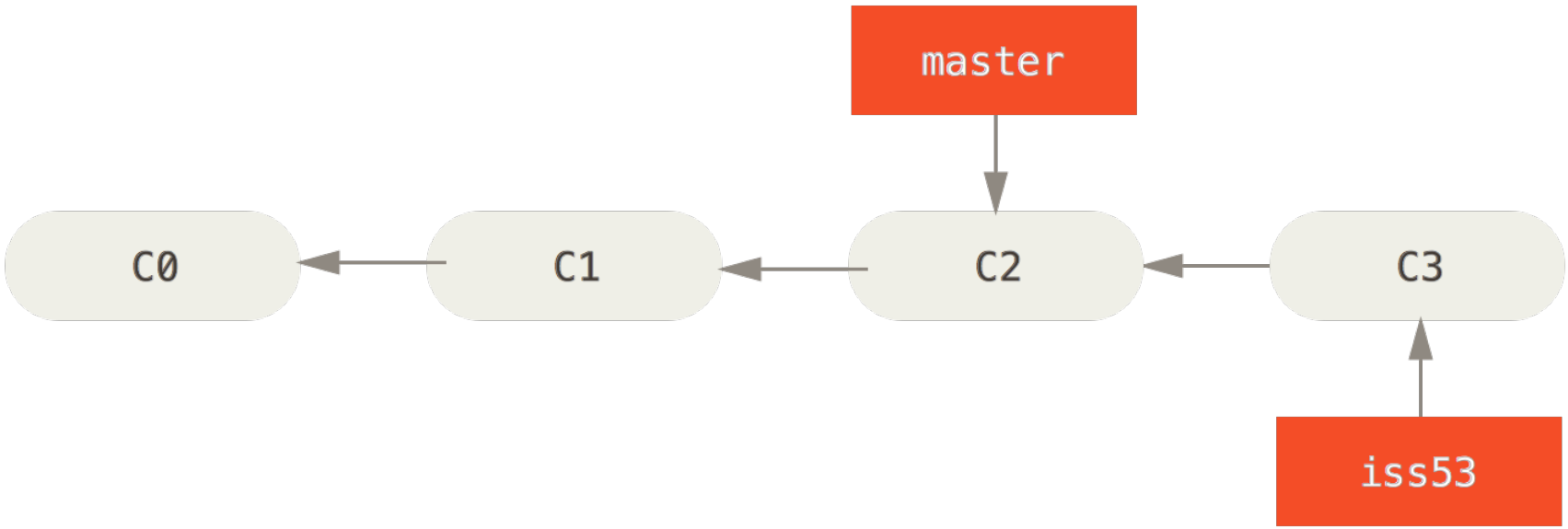
Új ág létrehozása

- Új ágat hozunk létre a feature-nek és át is váltunk
- `git checkout -b iss53`
  - Hosszabban:
  - `git branch iss53`
  - `git checkout iss53`



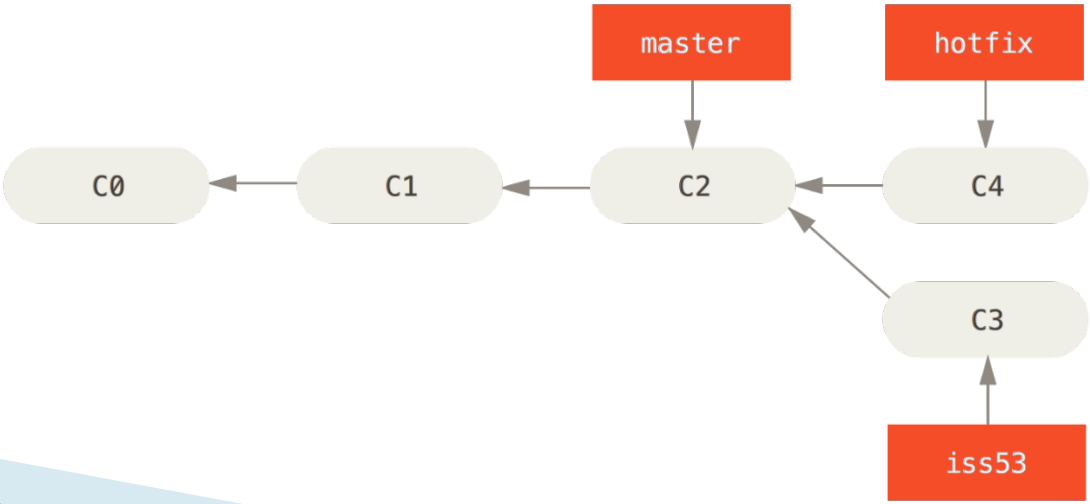
Dolgozunk...

- `echo "<p>A new feature!</p>" >> index.html`
- `git commit --am „Added new feature [issue 53]"`



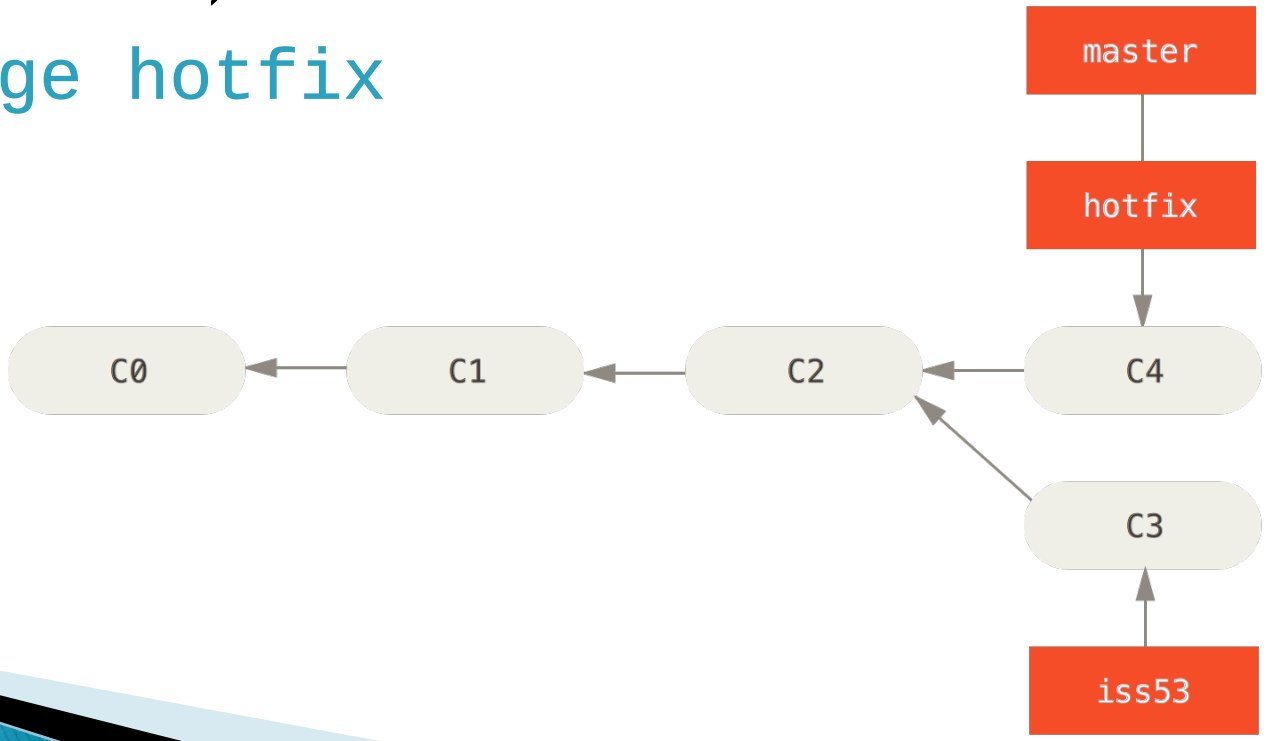
Most jön a hívás, hogy baj van…

- Elmentjük ahol eddig tartunk
- `git commit -am "In the middle of feature"`
  - Vagy git stash – lásd később
- `git checkout master` — Visszaváltunk master-re
- `git checkout -b hotfix` — Új branch és átváltunk
- Kijavítjuk a hibát
- `git commit -am "Hiba javítva"`



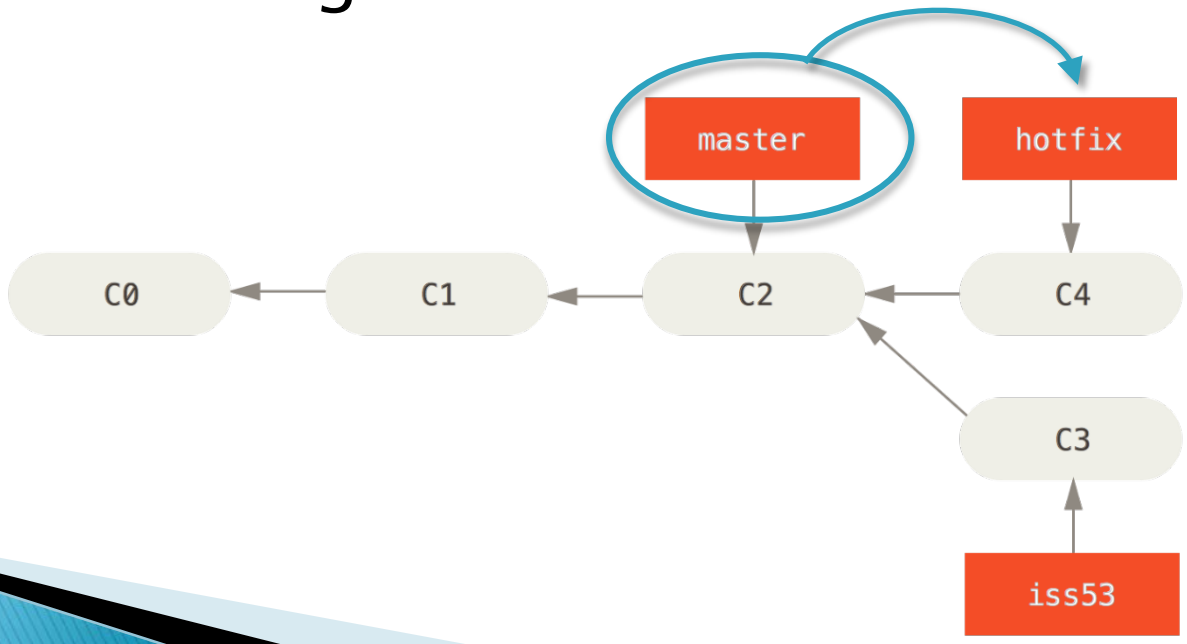
Beolvasztás (merge)

- Visszaváltunk a master-re
  - `git checkout master`
- Beolvasztjuk a hotfix ágat a jelenlegi ágba (HEAD=master)
- `git merge hotfix`



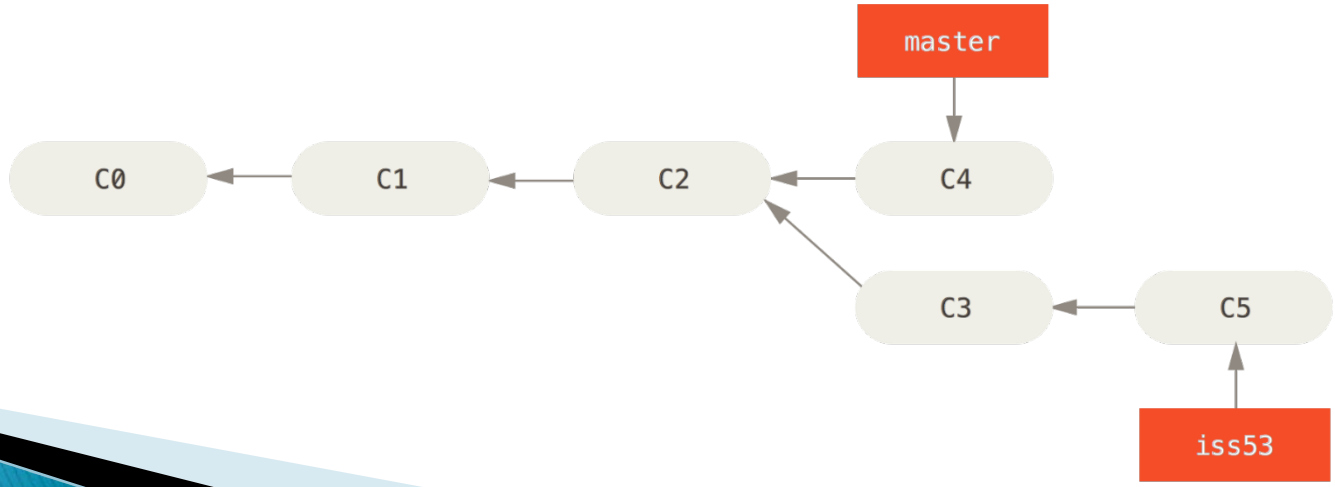
Fast-forward

- Ha egy commit-ot akarunk egy olyan commit-tal összeolvasztani ami az elsőnek az őse, akkor a Git egyszerűen előre mozgatja a pointert
- Mert nincs elágazott változtatás



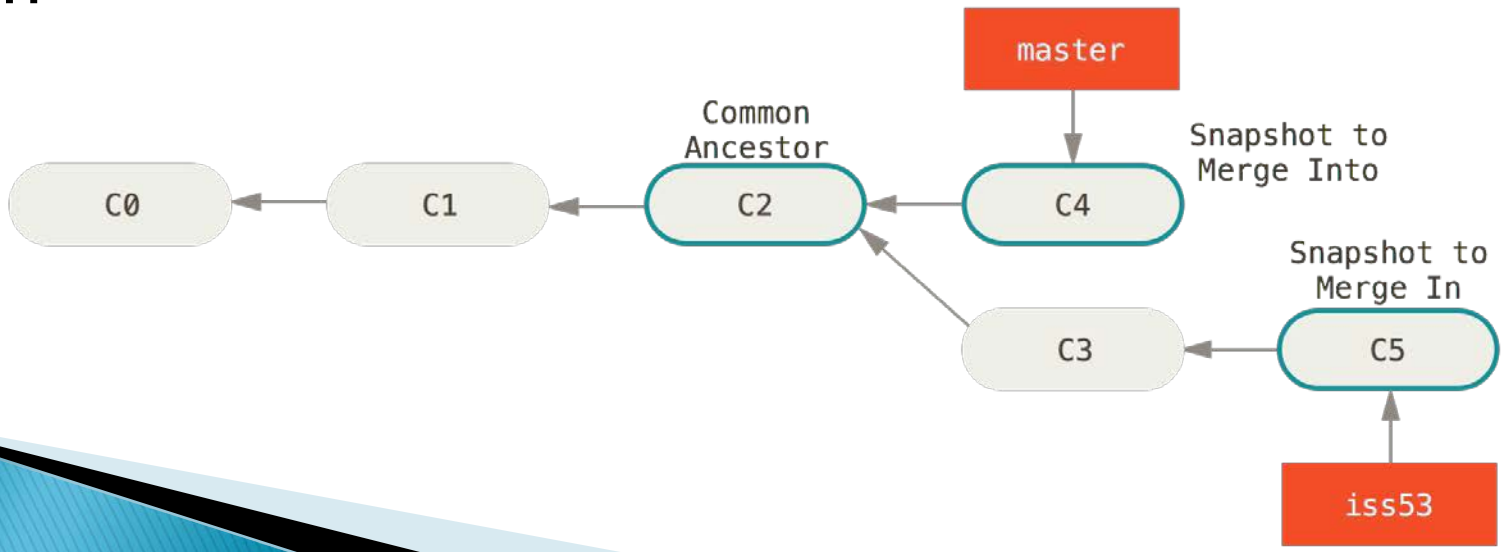
Takarítás és vissza…

- Töröljük a hotfix ágat, már nem kell
  - `git branch -d hotfix`
- Vissza az eredeti munkához
- `git checkout iss53`
- Befejezzük a munkát
- `git commit -am "Végeztünk a feature-rel"`



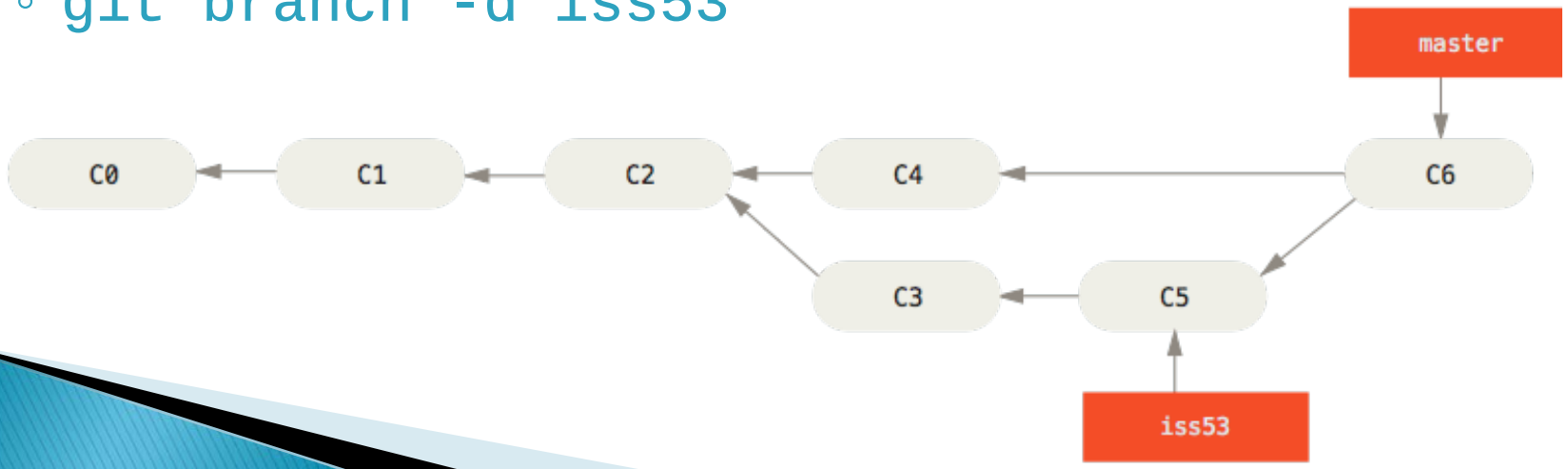
Beolvasztás

- Ha az eredeti munkát befejeztük, be is akarjuk olvasztani
- master-t és iss53-at összeolvasztjuk
- …ebből egy új commit lesz automatikusan
- merge commit=aminek egynél több szülője van



A beolvasztás menete

- Először átváltunk arra az ágra *amibe* be akarunk olvasztani
  - `git checkout master`
- …majd beolvasztjuk
  - `git merge iss53`
- Végül törölhetjük a beolvasztott ágat
  - `git branch -d iss53`



Ütközés

- Ha `iss53`-ban és a `hotfix`-ben egy fájl ugyanazon részét módosítottuk, akkor fel kell oldani az ütközést kézzel
- `git merge iss53`
  - Hiba! Nem tudja automatikusan megcsinálni
  - Megáll a beolvasztás amíg nem oldjuk fel az ütközést
  - `git status` megmondja, hogy éppen hogyan állunk

Ütközés

- Az ütköző fájlokba beleír az ütközés helyére
- Szövegszerkesztővel javítható
- Vagy GUI eszközzel
  - `git merge` → `git mergetool`

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
   please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

Ágak kezelése

- `git branch`
  - Milyen helyi ágak vannak és melyiken állunk most
- `git branch -a`
  - Minden helyi és távoli (követett) ágat listáz
- `git branch -r`
  - Csak a távoliakat listázza
- `git branch -v`
  - Commit üzeneteket is mutatja
- `git branch --merged`
  - Csak azokat mutatja, amik már be lettek olvasztva az aktuális ágba (ami előtt nincs * azt le lehet törölni)
- `git branch --no-merged`
  - A be nem olvasztott ágakat mutatja

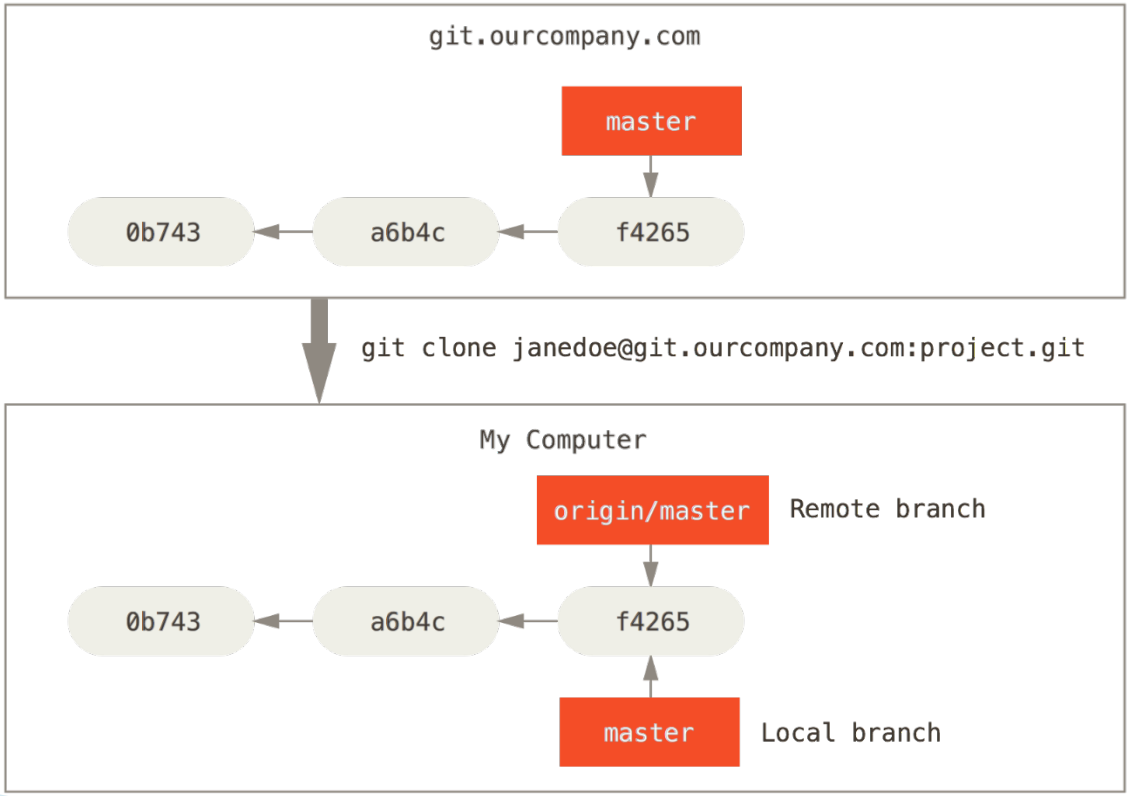
Távoli ágak

Távoli ágak

- Listázás
  - `git ls-remote [távoli név]`
  - `git remote show [távoli név]`
    - Pl. `git remote show origin`
  - `git log --oneline --graph --decorate --all`
    - Helyi, távoli ágak és commit-ok fa struktúrában
    - Érdemes ezt elmenteni aliasként:
      - `git config --global alias.l "log --oneline --graph --decorate --all"`
      - Ezután „`git l`”-lel lehet ezt a parancsot kiadni

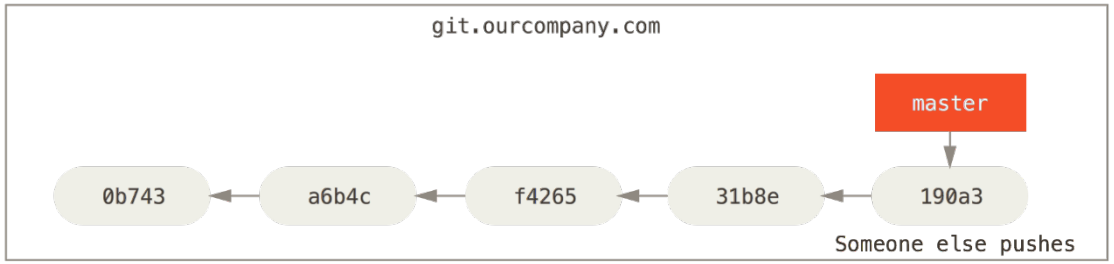
Klónozás után…

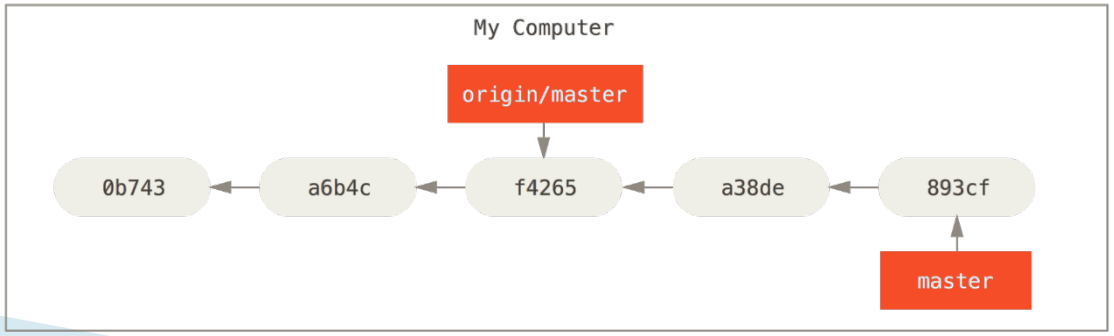
- Lokálisan két águnk lesz, melyek ugyanoda mutatnak
  - master
  - origin/master



Mindeközben a szerveren…

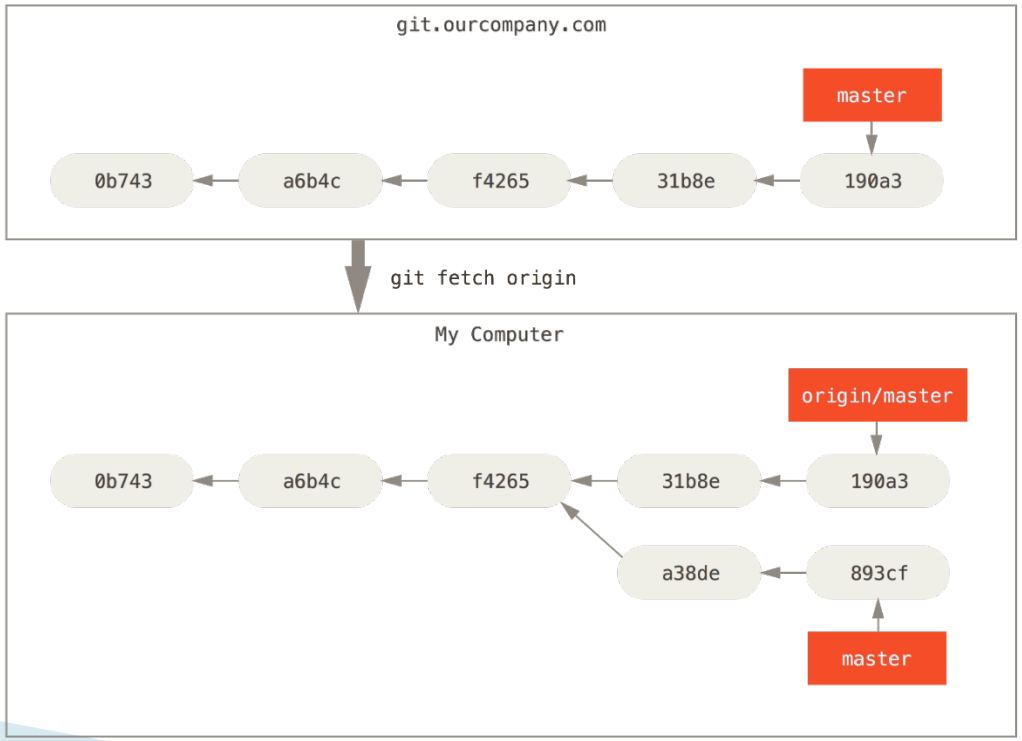
- …valaki újabb munkát küldött fel (`git push`)
- A távoli és a helyi repó elágazott
- origin/master ág addig nem változik, amíg nem kérdezzük le újból a távoli repót (`git fetch origin`)





Szinkronizálás

- `git fetch [origin]`
  - Letölt minden commitot, ami a helyi repóban nem szerepel
  - origin/master mutatót átállítja a legutóbbi commitra
  - Nem olvaszt össze!



Saját munka feltöltése

- `git push [távoli hely] [helyi ág]`
  - Pl. `git push origin serverfix`
- A távoli repóban is megjelenik a `serverfix` ág a feltöltött commitokkal

Frissítés

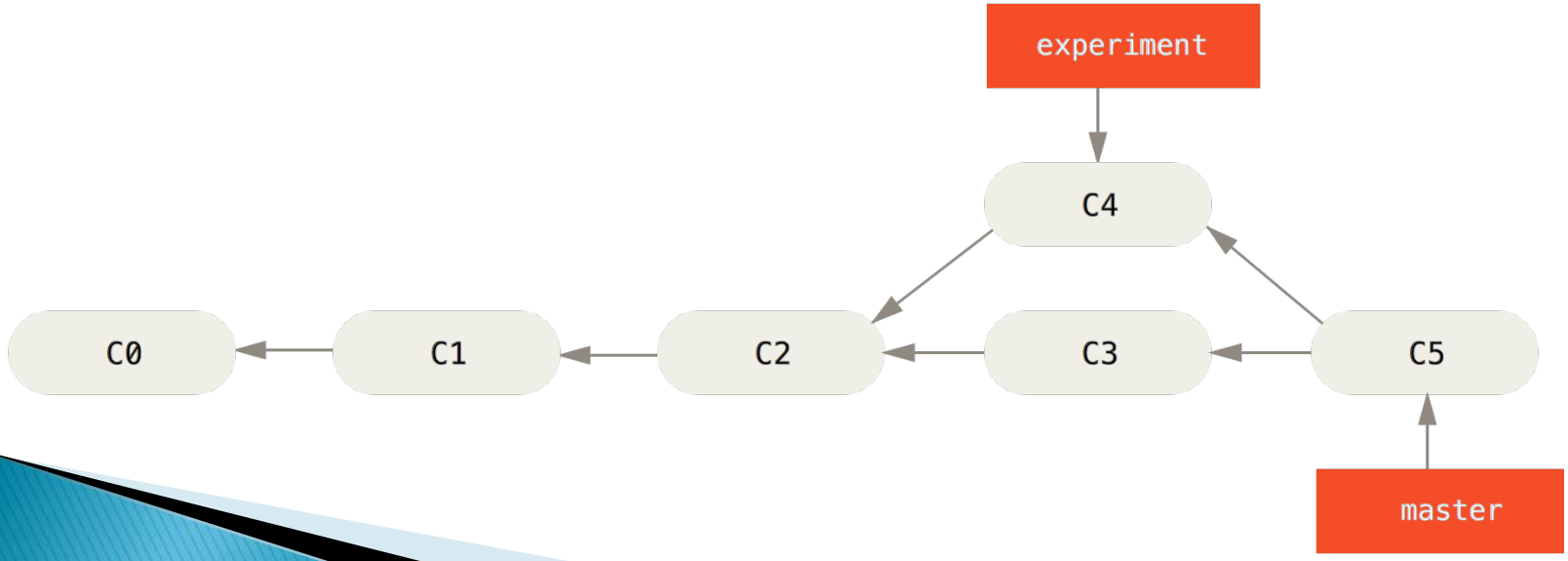
- `git fetch [origin]`
- Nem olvaszt be automatikusan
  - Beolvasztjuk…
    - `git merge origin/serverfix`
  - Vagy új ágat hozunk létre neki
    - `git checkout –b serverfix origin/serverfix`
    - = `git checkout --track origin/serverfix`
- `git pull` =
  - `git fetch`
  - `git merge origin/<ág>`

Távoli ág törlése

- `git push origin --delete serverfix`
  - (`git push origin :serverfix`) – Advanced!

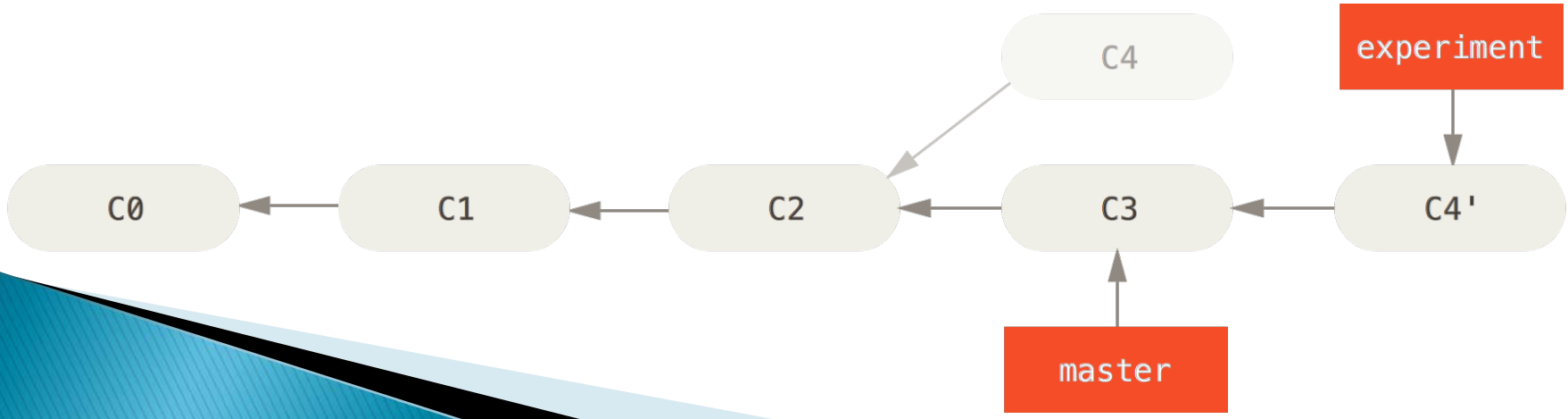
Merge helyett rebase

- Elágazást kétféleképpen lehet összeolvasztani
  - merge
  - rebase
- Hagyományos merge



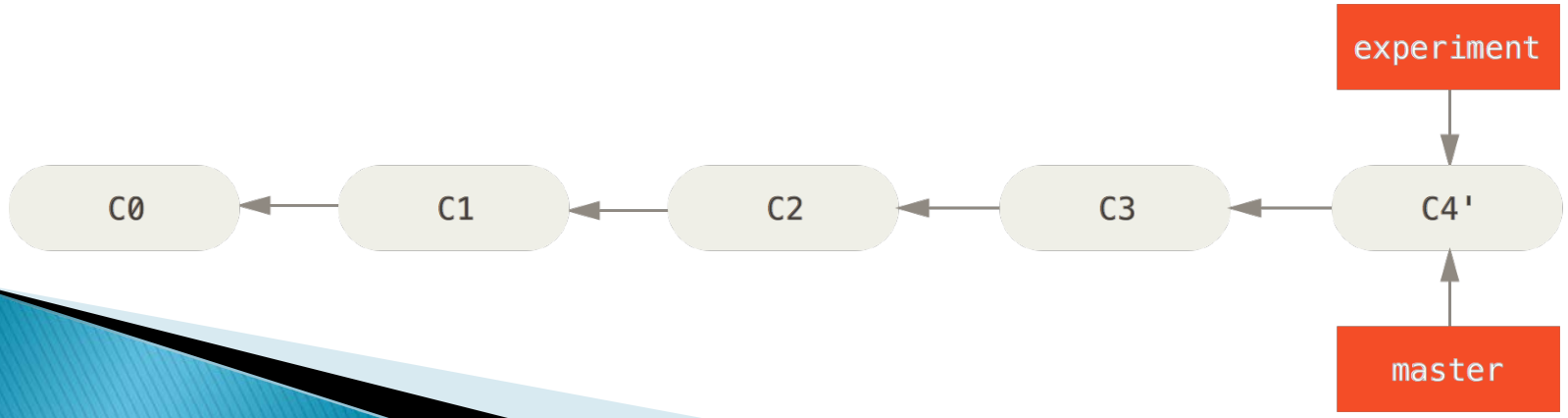
Rebase

- A szerveren történt változásokra „ráhúzzuk” a saját változásainkat
- `git checkout experiment`
- `git rebase master`
- C4-nek egy másolata (C4’) kerül a master tetejére
- Ez egy teljesen új commit (más hash-sel), de ugyanazokkal a változásokkal, amit C4 tartalmaz



Frissítsük a mastert

- Ezután a master ágat is átállíthatjuk a legújabb commit-ra
  - `git checkout master`
  - `git merge experiment` (=fast-forward)
- C4' pontosan ugyanazt tartalmazza, mint a merge példában C5
- Lineáris történet, merge commit-ok nélkül



Rebase alapszabály!

SOHA NE REBASE-ELJÜNK OLYAN COMMIT-OKAT AMIK MÁR TÁVOLI REPÓBAN IS VANNAK!

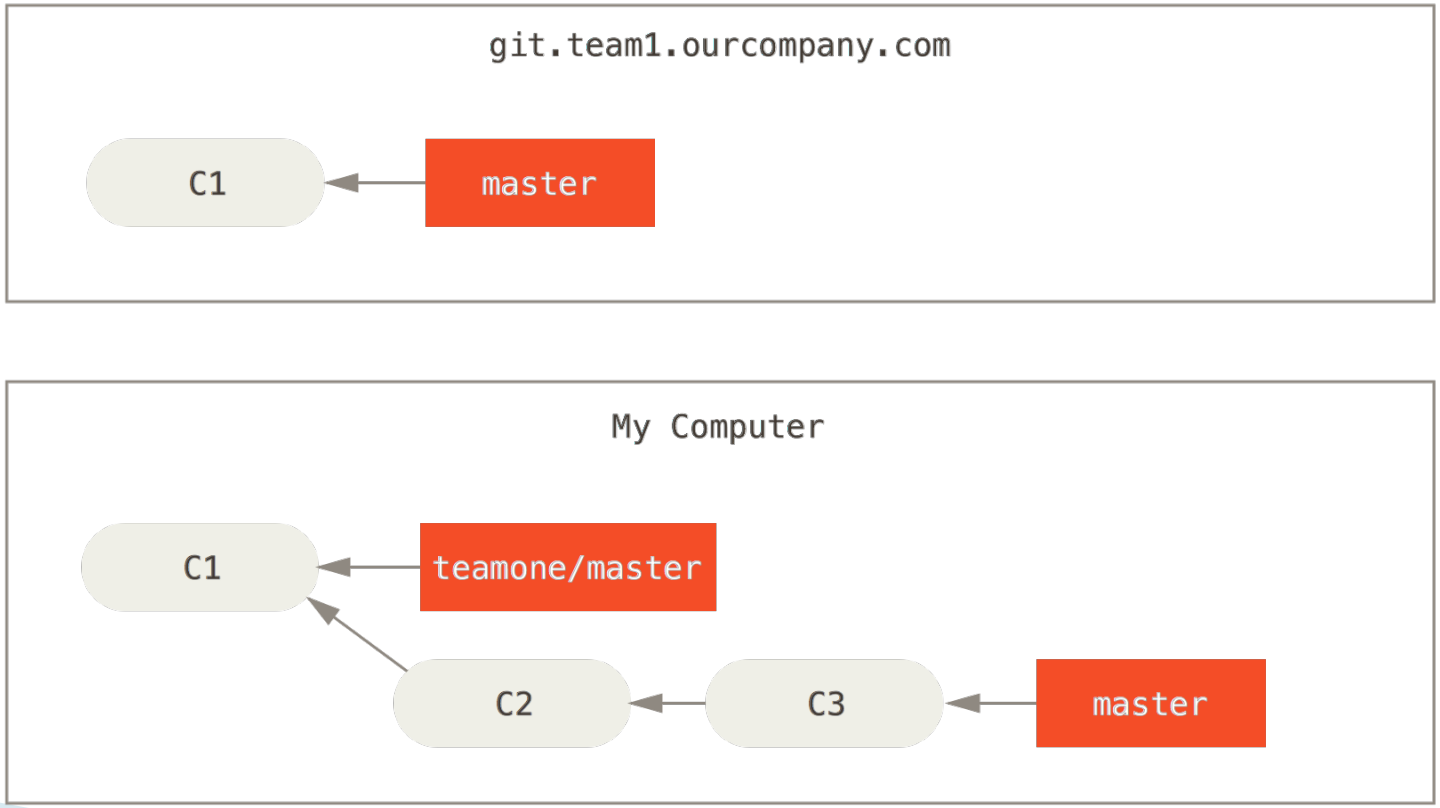
…miért?

- A rebase hátrahagy meglévő commit-okat, és újakat készít, amik tartalma ugyanaz, de mégis új commit-ok
- Ha valaki ezeket a hátrahagyott commit-okat letölti és dolgozik rajtuk…
- És eközben mi újraírjuk ezeket a commit-okat rebase-zel, majd feltöltjük
- Akkor a többieknek feltöltés előtt be kell olvasztania a mi változásainkat
- És akkor lesz baj, amikor mi le akarjuk tölteni a többiek munkáját (amik olyan commit-okon alapulnak, amiket mi átírtunk/hátrahagytunk)

Példa a rossz rebase-re

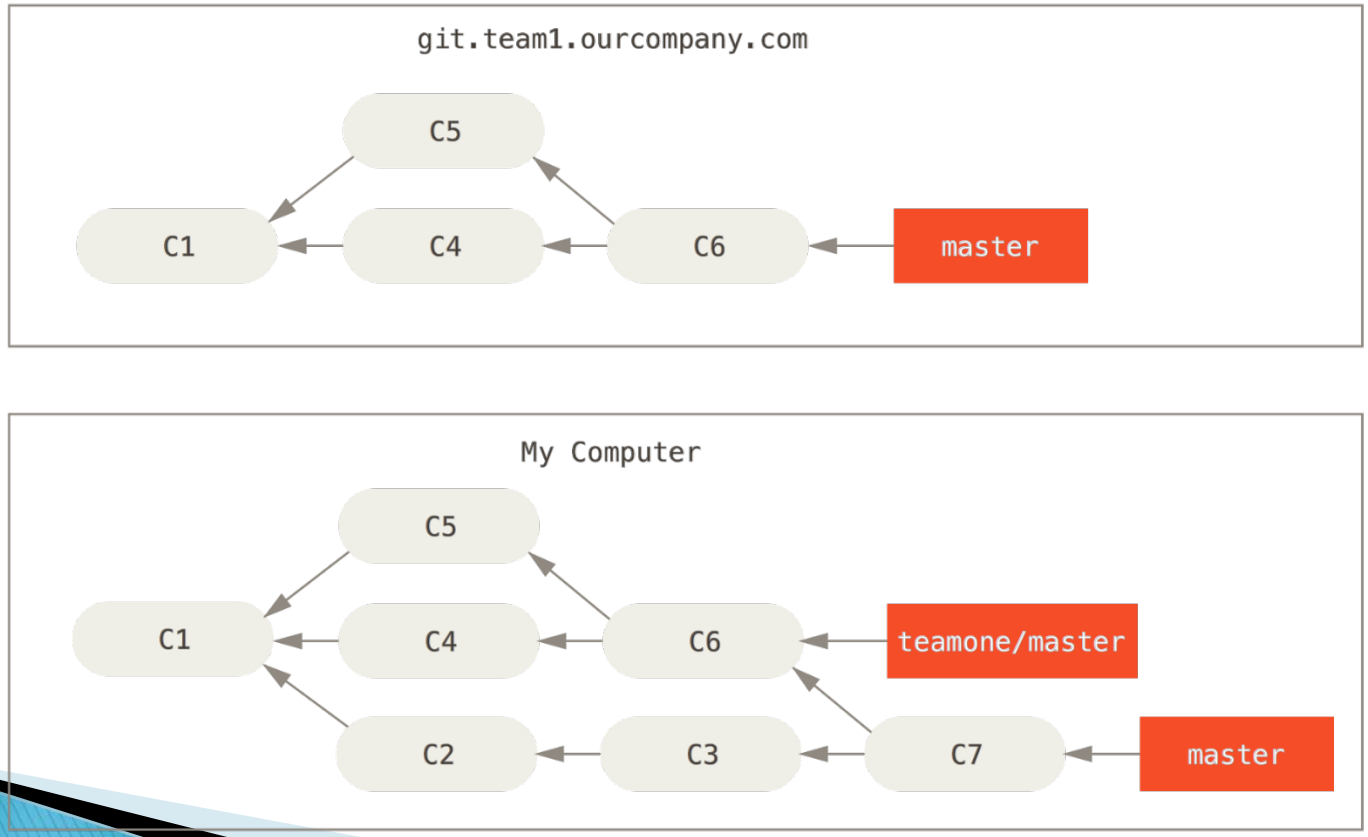
Klónozunk és dolgozunk rajta

- teamone/master követi a távoli master-t



Más is dolgozik és feltölti

- Ezeket mi letöltjük és beolvasztjuk a saját munkánkba (C7) és tovább dolgozunk

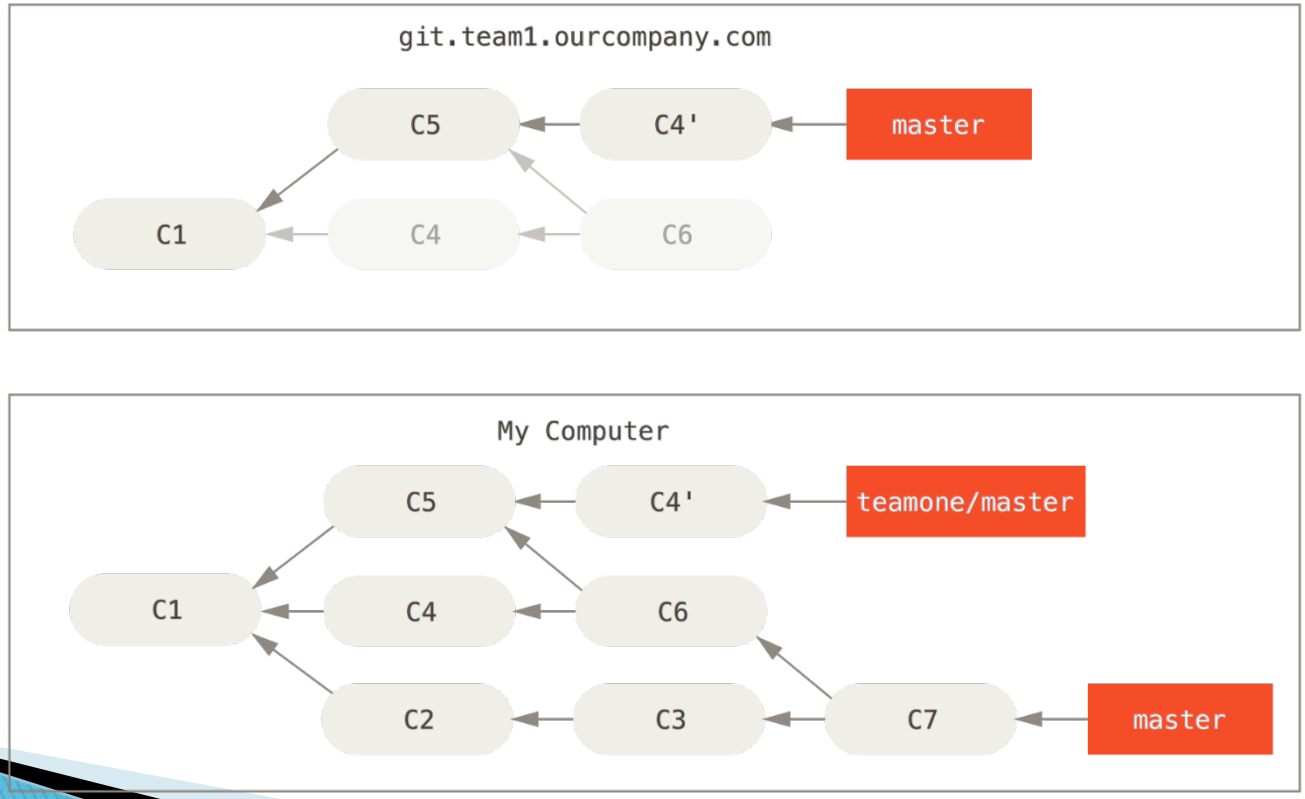


Majd…

- Aki az előző feltöltést csinálta, úgy dönt, hogy merge helyett rebase-t akar használni (a szerveren is), ezért
- Helyben visszavonja a merge-öt, rebase-eli azokat a változásokat, majd `git push --force` -szal felülírja a történetet a szerveren
- Mi megint letöltjük (fetch) az új commitokat

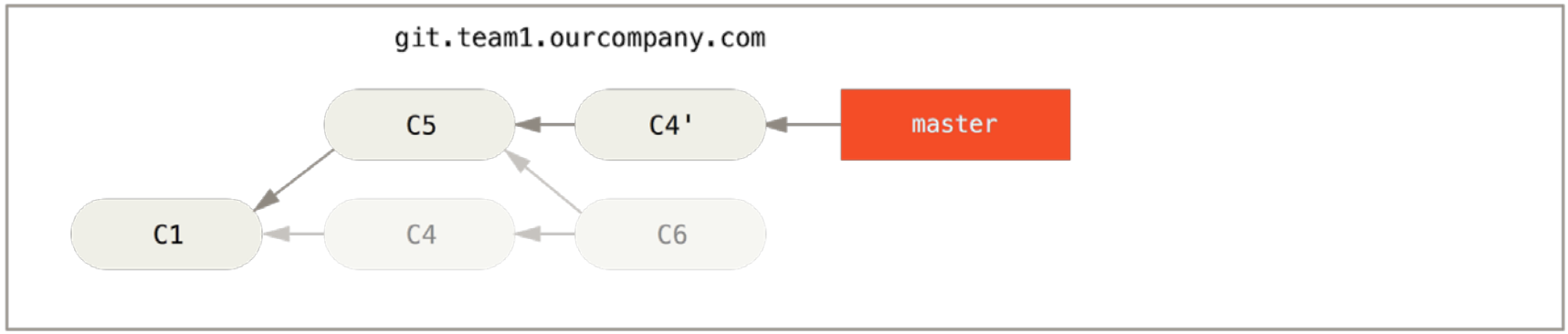
És itt tartunk…

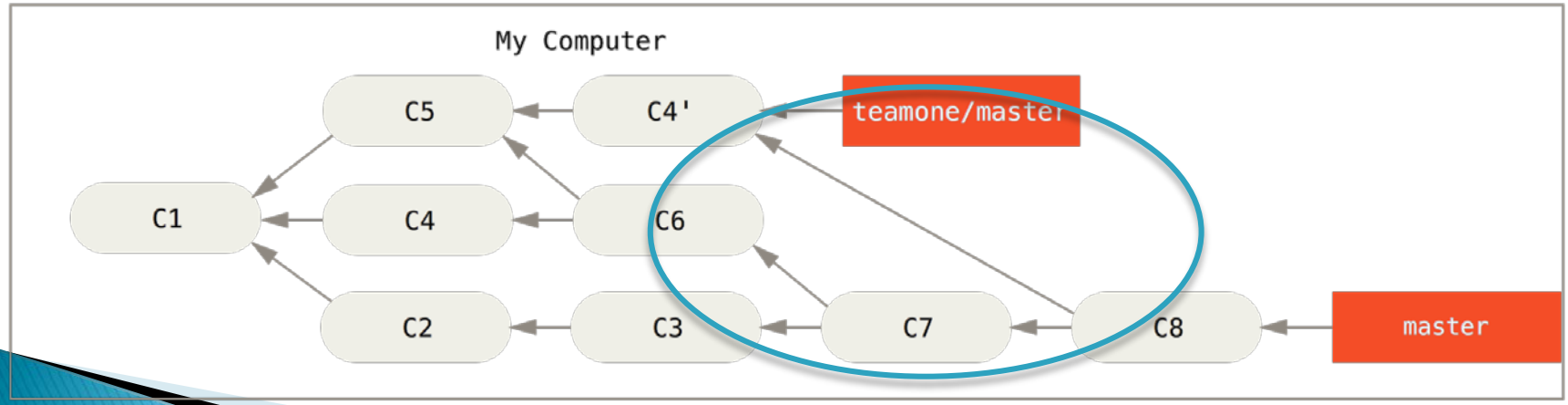
- Olyan commit-ra épül C7, ami a szerveren nem is létezik!



Ezután…

- Egy `git pull` után így néz ki a fa





Tanulság

- A rebase jó, mert lineáris történetet láthatunk a szerveren
- Azt mutatja meg, „Hogyan készült a projekt”
- Mindig használjuk lokálisan
- De nem szabad rebase-elni olyan commitokat (ágat), amit már feltöltöttünk (push)
- (mert arra más alapozhat és kihúzzuk alóla a szőnyeget)
- `git pull --rebase`
- Rebase legyen az alapértelmezett
  - `git config --global pull.rebase true`

Hasznos eszközök

Stash

- Ha egy munka közepén vagyunk, és nem akarunk még commitolni
- De át kell váltani egy másik ágra
- Egy átmeneti területre elmenthetőek a jelenlegi változások
- Majd később onnan visszaállíthatóak
- `git stash [save]`
- Egy elmentett állapotot egy másik ágon is alkalmazhatunk

Stash paraméterek

- Visszaállítás
  - `git stash apply`
    - Nem törli ki a tárolóból
  - git stash apply stash@{2}
    - Egy bizonyos stash-t alkalmaz
  - `git stash pop`
    - Kitörli a tárolóból
- Elmentett változások listázása
  - `git stash list`
- Törlés
  - `git stash drop <stash név>`
- Mentés egy új ágba
  - `git stash branch testchanges`

Változások visszaállítása

- Csak helyi repóban!
  - `git reset --soft [commit]`
    - Mozgatja a HEAD által mutatott ágat
  - `git reset [--mixed] [commit]`
    - + az INDEX-et is visszaállítja
  - `git reset --hard [commit]`
    - + a munkakönyvtárat is visszaállítja
    - Vigyázat! Ami nem volt commitolva elvész!
- Ha a visszaállítandó commitok a távoli repóban is szerepelnek!
  - `git revert`
    - Új commitot hoz létre, aminek a tartalma pontosan a visszaállított tartalom

Helyi változások eldobása

- Ha el akarjuk dobni a helyi változásokat és vissza akarjuk állítani a HEAD által mutatott állapotot
- `git checkout -f`
- `git reset --hard`

Több commit összeolvasztása

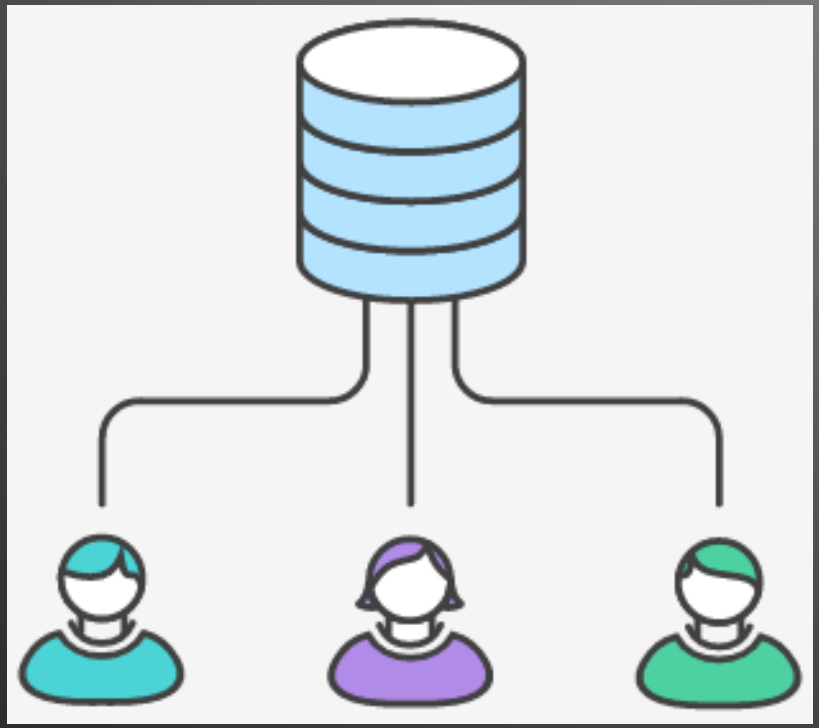
- Interaktív rebase
  - Sok más dologra is jó
    - Commit-ok sorrendjének megváltoztatása
    - Commit-ok kihagyása
- `git rebase -i`
- Pl. van 3 commit-ünk, amit egy commit-ként akarunk feltölteni

git-rebase-todo

```
1  pick 18c84db Added f()
2  pick 5a0cff0 Fixed fact()
3  pick dec216b Added .gitignore
4
5  # Rebase a314047..dec216b onto 9ba36a5
6  #
7  # Commands:
8  #  p, pick = use commit
9  #  r, reword = use commit, but edit the commit message
10 #  e, edit = use commit, but stop for amending
11 #  s, squash = use commit, but meld into previous commit
12 #  f, fixup = like "squash", but discard this commit's log message
13 #  x, exec = run command (the rest of the line) using shell
```

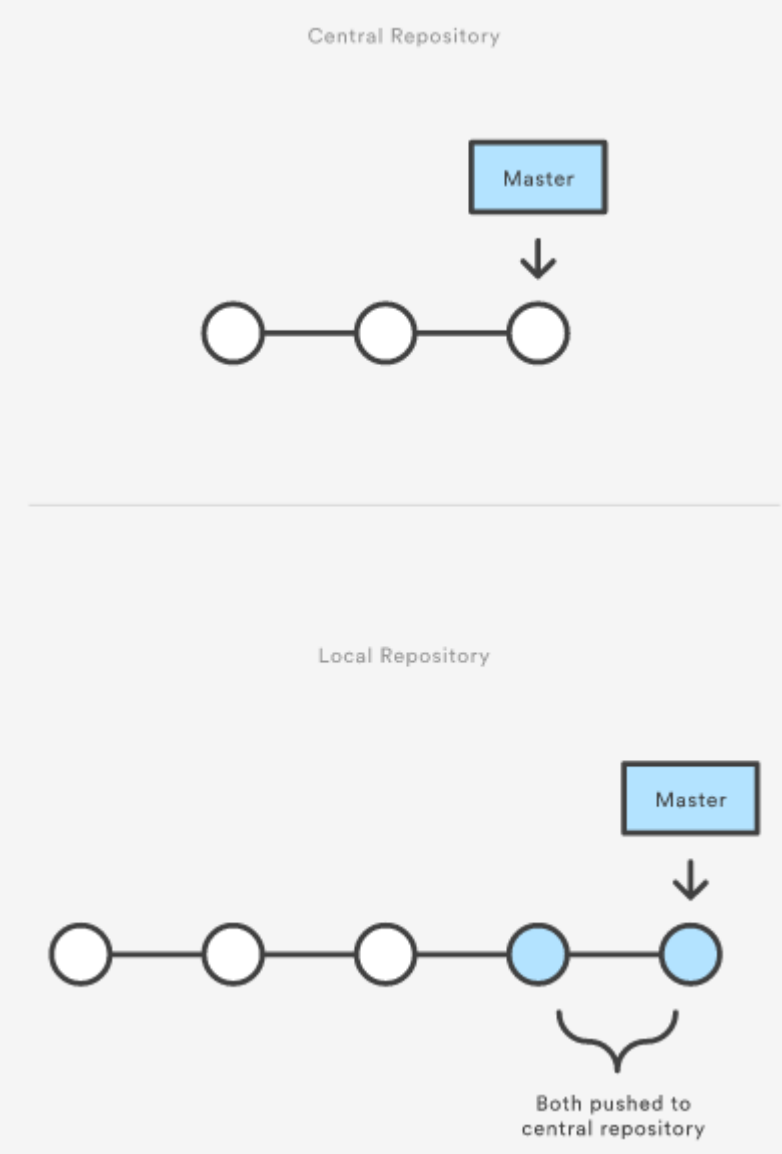
Minták Git használatra

Központosított munkamenet



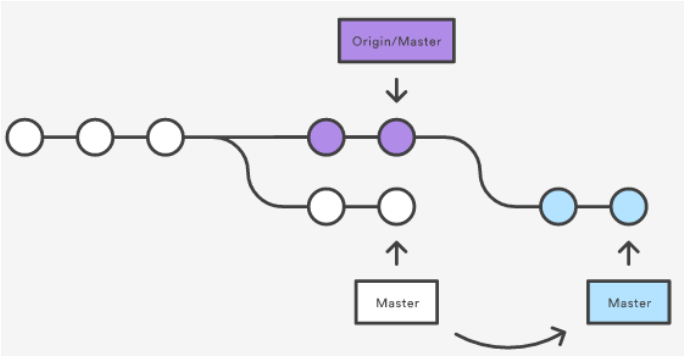
Központosított munkamenet

- Egy központi repón dolgozik mindenki
- `git clone`
- Helyi munka
  - Mindenki a saját példányán dolgozik
  - Amikor kész, megosztja a többiekkel
- `git push`

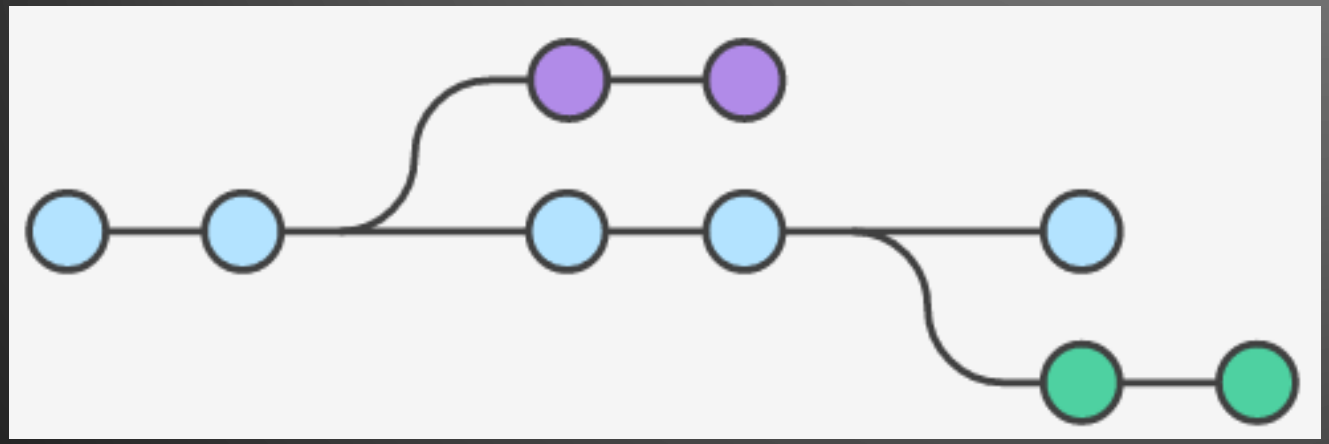


Ütközés feloldása

- Többen dolgoznak, és valaki már feltöltött újabb verzió(ka)t
- Mielőtt mi is „push”–olni tudnánk a munkánkat, le kell húzni a központi repó változásait
  - `git pull --rebase` =
    - `git fetch`
    - `git rebase origin/master`
  - Ütközés!
  - `git mergetool`
  - `git rebase --continue`
  - `git push`

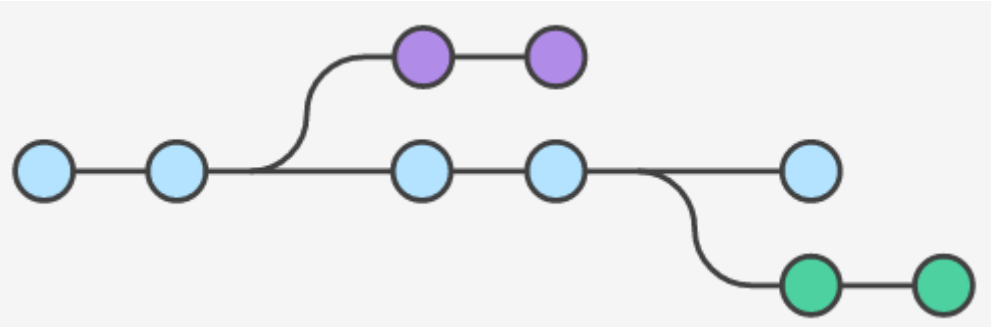


Feature branch munkamenet

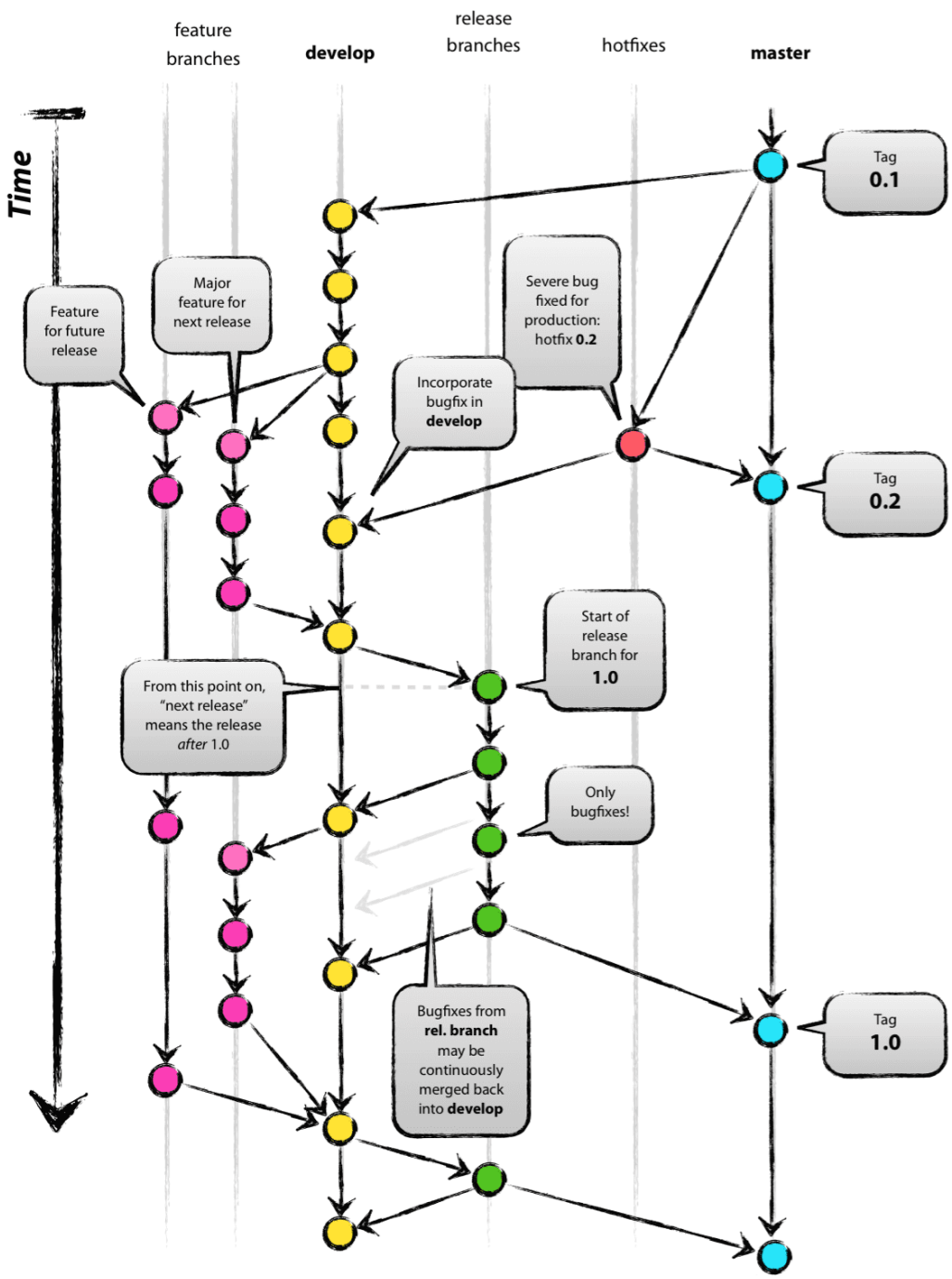


Feature branch munkamenet

- Minden új funkcióhoz egy új ágat nyitunk
- Több fejlesztő tud egy funkción dolgozni a fő ágba avatkozás nélkül
- Egy funkció elkészültekor „pull request” a fő ágat karbantartó kollégá(k)nak
- Beolvasztás előtt meg lehet vitatni, hogy mi kerül be (code review)



Gitflow munkamenet

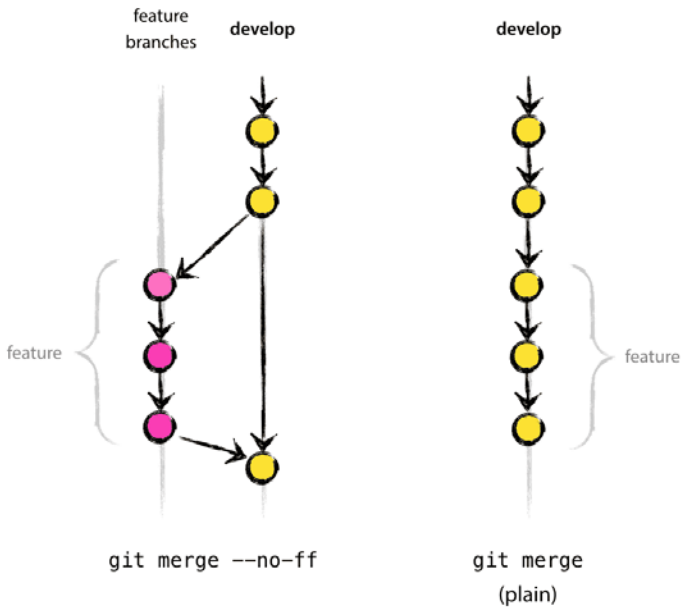

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Gitflow

- Két fő ág
  - master = a termék fő ága
    - Mérföldkövek verziószámozva, címkézve
    - Ide kerül a működő, tesztelt termék forrása
  - develop = fejlesztői ág
    - A fejlesztők innen ágaznak le
      - Új funkciókkal (feature branch)
      - Ha kész a funkció, ide olvasztják be
      - Pull request, code review lehetséges
- Kiegészítő ágak
  - Release ágak
    - Új funkciók már nem, kisebb hibajavítások, metaadatok frissítése (pl. verziószám növelés)
  - Hotfix ágak
    - master hibáinak javítására – a master ágból közvetlenül
    - Beolvad develop-ba és master-ba is

Új funckció

- Feature branch
- `git checkout -b myfeature develop`
- Új funkció kész…
- `git checkout develop`
- `git merge --no-ff myfeature`
- `git branch -d myfeature`
- `git push origin develop`



Kiadási ág

- Release branch
- `git checkout –b release-1.2 develop`
- Verziószám növelése, stb…
- `git commit –am „Verzió növelés…”`
- `git checkout master`
- `git merge --no-ff release-1.2`
- `git tag –a 1.2`

Kiadási ág vissza a develop-ba is

- `git checkout develop`
- `git merge --no-ff release-1.2`
- Töröljük az ágat, már nem kell
- `git branch –d release-1.2`

Hibajavítási ág

- `git checkout –b hotfix-1.2.1 master`
- Növeljük a minor verziót…
- `git commit –am „Version bumped to 1.2.1”`
- Javítjuk a hibát
- `git commit –am „Fixed the bug [ABC123]”`
- Beolvasztjuk `master`-ba…
- `git checkout master`
- `git merge --no-ff hotfix-1.2.1`
- `git tag –a 1.2.1`

Hibajavítás a develop-ba is

- `git checkout develop`
- `git merge --no-ff hotfix-1.2.1`
- Töröljük az ágat, már nem kell
- `git branch –d hotfix-1.2.1`

Hasznos parancsok – összefoglalás

- Szép fa diagramm a helyi és távoli történetről
  - `git log --oneline --graph --decorate –all`
- Munkakönyvtár visszaállítása a legutóbbi commitra
  - `git checkout –f` vagy `git reset –hard`
- Kérdezzük le a szervert, de ne olvasszunk be semmit
  - `git fetch`
- „Mi az új a szerveren a miénkhez képest?”
  - `git difftool origin/master`
- Frissítés a szerverről, rebase-zel
  - `git pull --rebase`

Hasznos parancsok – összefoglalás

- Nem követett fájlok törlése a munkakönyvtárból
  - `git clean`
  - …kihagyott fájlok is (ami .gitignore-ban van)
  - `git clean -x`

Köszönöm a figyelmet!