Git bevezető

Szabó Adrienn
Adatbányászat és Webes Keresés Kutatócsoport

2010

MTA SZTAKI
COMPUTER AND AUTOMATION RESEARCH INSTITUTE
HUNGARIAN ACADEMY OF SCIENCES

Tartalom

Áttekintés

Git: egy modern verziókezelő

Manapság a legnépszerűbb[1] szabadon használható elosztott verziókövető rendszer a *git*. Kicsit nehezebb ugyan megtanulni mint a *cvs*-t vagy az *svn*-t, de megéri, mert sokkal többet tud.

Előnyei:

- Elosztott
- Rugalmas
- Könnyű branch-eket használni
- Gyors
- Jól használható más rendszerekkel együtt is

[1] Open Source projektek körében

Egy kis történelem

Az első verziókövető redszer (VCS) a *CVS* volt (1986). Azóta sok más rendszert fejlesztettek, mert a *CVS* nem tudott minden igényt kielégíteni.
2000-ben a linux kernel fejlesztésének támogatására Linus a *BitKeeper*-t választotta, ami az első valóban elosztott VCS volt. Licenszelési konfliktusok miatt 2005-ben a kernel fejlesztői úgy döntöttek hogy létrehoznak egy saját verziókezelőt.

A célok között szerepelt:

- gyorsaság
- egyszerűség
- nem-lineáris fejlsztés (párhuzamos branchek) támogatása
- nagy projektek hatékony kezelése

Egy kis történelem

Az első verziókövető redszer (VCS) a *CVS* volt (1986). Azóta sok más rendszert fejlesztettek, mert a *CVS* nem tudott minden igényt kielégíteni.
2000-ben a linux kernel fejlesztésének támogatására Linus a *BitKeeper*-t választotta, ami az első valóban elosztott VCS volt. Licenszelési konfliktusok miatt 2005-ben a kernel fejlesztői úgy döntöttek hogy létrehoznak egy saját verziókezelőt.

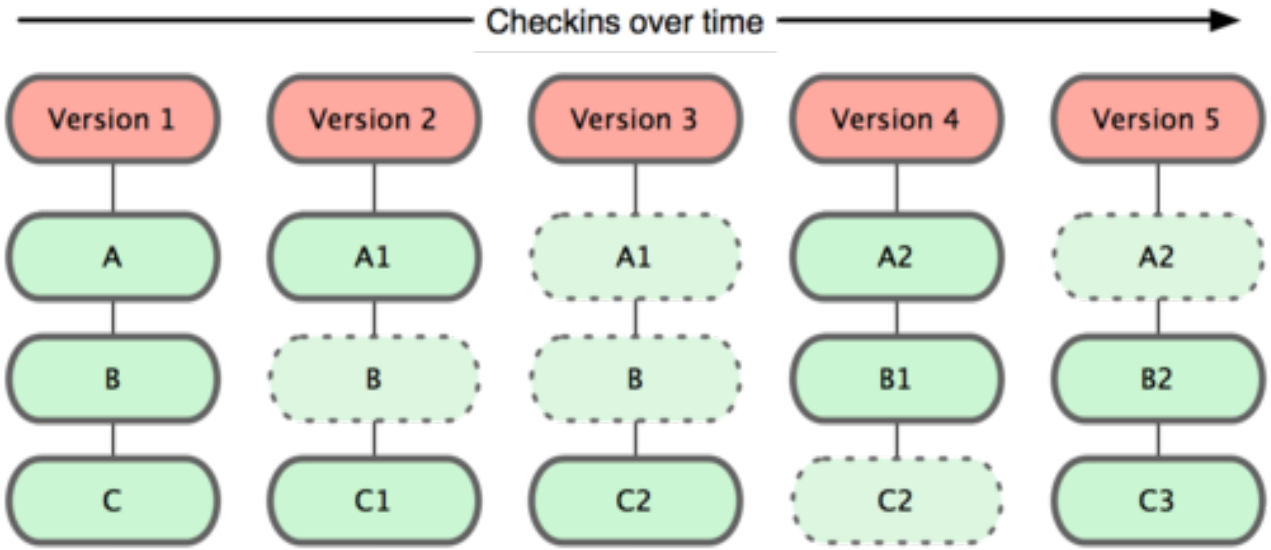
A célok között szerepelt:

- gyorsaság
- egyszerűség
- nem-lineáris fejlsztés (párhuzamos branchek) támogatása
- nagy projektek hatékony kezelése

Git alapok

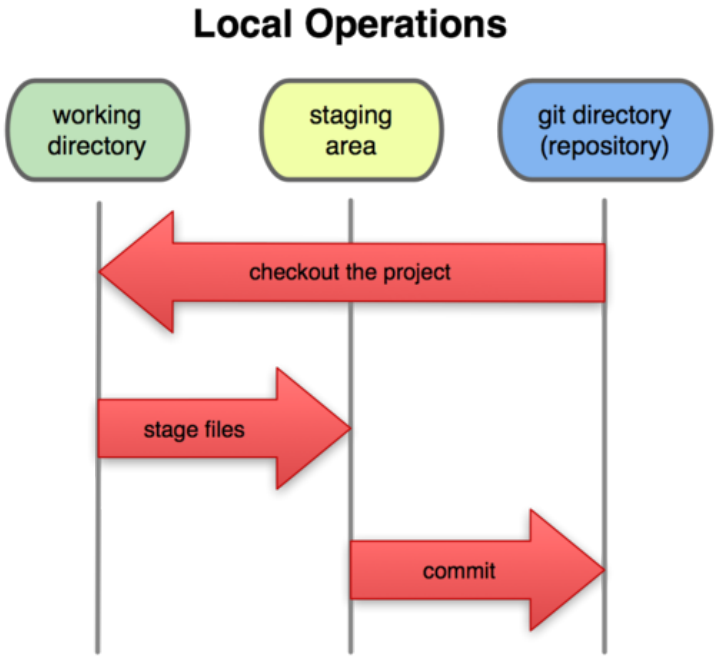
A legtöbb verziókezelő a fájlok eredeti verzióit és az ahhoz képesti változásokat rögzíti.

Ezzel szemben a *git* ún. snapshot-okat tárol. Ez lehetővé teszi hogy úgy működjön mint egy „mini fájlrendszer”.



Git alapok

A *git* rugalmasságát részben az biztosítja, hogy a fájloknak van egy közbülső állapota a commit előtt. Egy fájl háromféle állapotban lehet a *git* repository-n belül: *modified*, *staged* vagy *committed*.



Első példa

Új repository létrehozása, fájlok hozzáadása a projekthez, állapot ellenőrzése, néhány commit, átnevezés, törlés log megtekintése.

Parancsok:

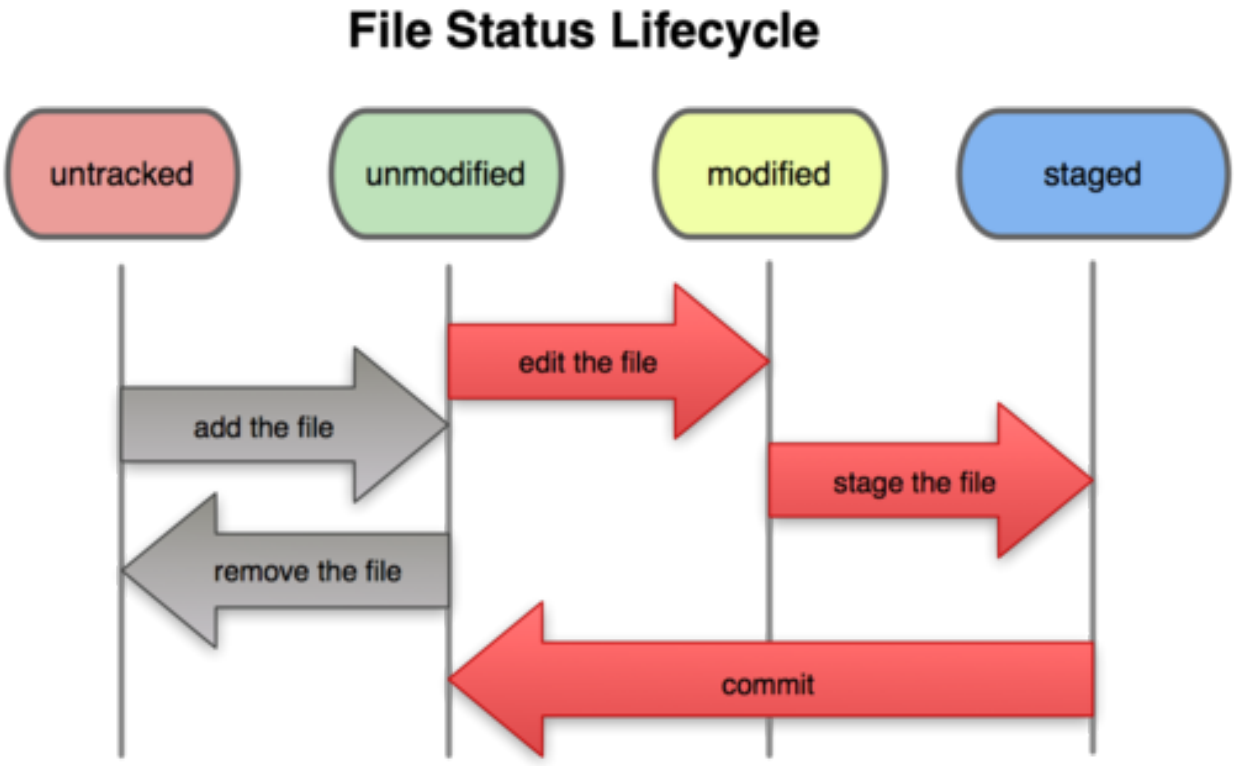
```
git init
git status
git add
git commit
git mv
git rm
git log
```

Beállítások

Néhány változó beállításával hasznosabb logokat kaphatunk, illetve alias-okat is megadhatunk hogy kevesebbet kelljen gépelni.

- `git config --global user.name adri`
- `git config --global user.email adri@sztaki.hu`
- `git config --global alias.ci commit`
- `git config --global alias.st status`
- `git config --global alias.co checkout`

Egy fájl életciklusa



Második példa

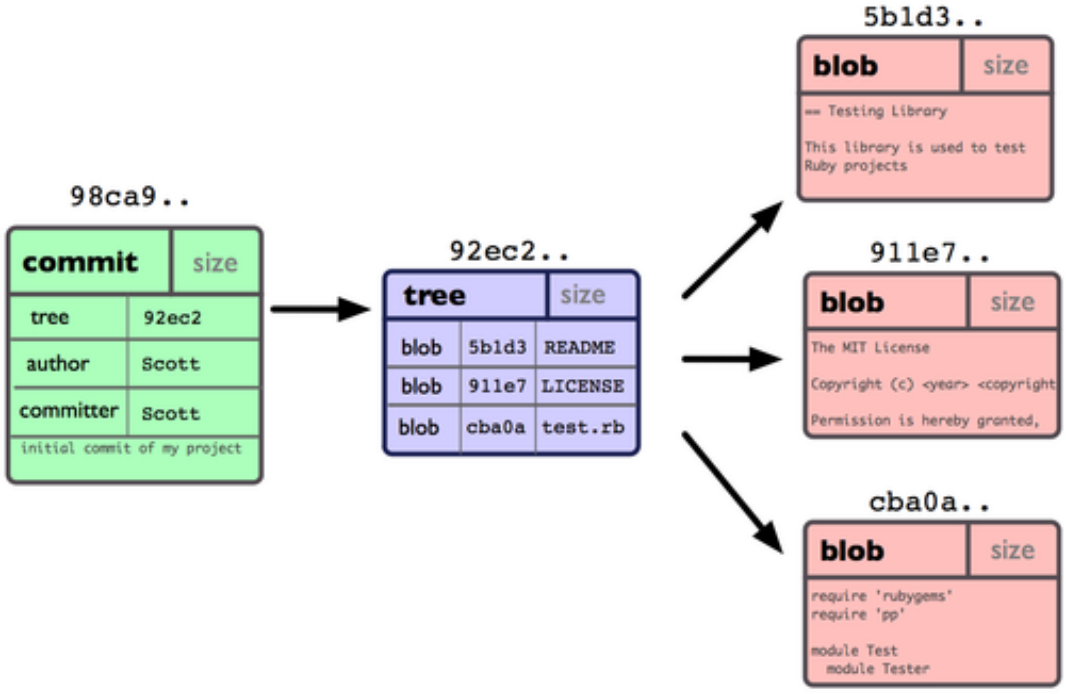
Mi változott?
Hibák kijavítása: utolsó commit módosítása, nem-commitolt változtatások eldobása (reset), commitok eldobása (revert).

Parancsok:

```
git diff
git commit --amend
git reset
git checkout
git config alias.unstage 'reset HEAD --'
git unstage
git revert
```

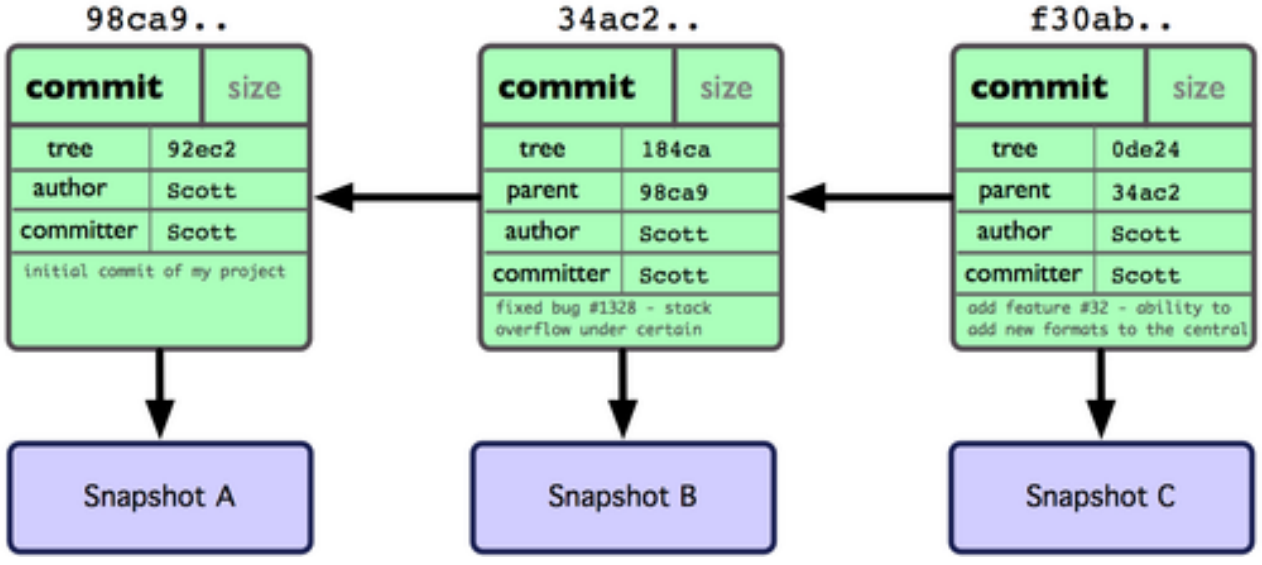
Git objektumok

A *git* tömörített objektumokban tárol és a SHA-1 hash értékkel azonosít mindent (fájlok, könyvtárfa, commitok).



A history egy DAG

Egyszerű esetben a commitok szép sorban követik egymást:



De merge esetén egy commit-nak két (vagy több) szülője is lehet majd.

Áttekintés

Centralizált vs elosztott verziókövetés

Elosztott rendszer előnyei:

- offline is használható
- lokális műveletek a helyi repository-n: gyorsabb
- a fejlesztők bátrabban commitolnak sok kicsit ha nem kell rögtön a központba tenni mindenki szeme láttára a még félkész művüket
- mindenkinél megvan az egész repo (biztonság)
- a hatékony tömörítés miatt nem lesz túl nagy a repo mérete historyval együtt sem

Centralizált vs elosztott verziókövetés

Elosztott rendszer előnyei:

- offline is használható
- lokális műveletek a helyi repository-n: gyorsabb
- a fejlesztők bátrabban commitolnak sok kicsit ha nem kell rögtön a központba tenni mindenki szeme láttára a még félkész művüket
- mindenkinél megvan az egész repo (biztonság)
- a hatékony tömörítés miatt nem lesz túl nagy a repo mérete historyval együtt sem

Centralizált vs elosztott verziókövetés

Elosztott rendszer előnyei:

- offline is használható
- lokális műveletek a helyi repository-n: gyorsabb
- a fejlesztők bátrabban commitolnak sok kicsit ha nem kell rögtön a központba tenni mindenki szeme láttára a még félkész művüket
- mindenkinél megvan az egész repo (biztonság)
- a hatékony tömörítés miatt nem lesz túl nagy a repo mérete historyval együtt sem

Centralizált vs elosztott verziókövetés

Elosztott rendszer előnyei:

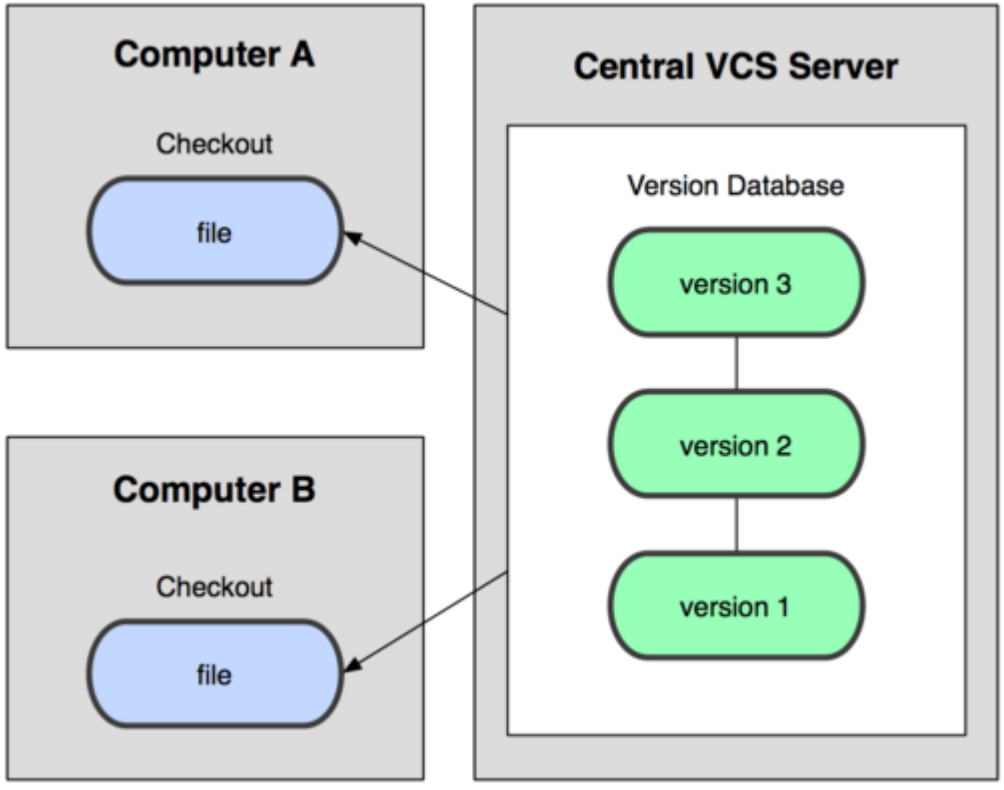
- offline is használható
- lokális műveletek a helyi repository-n: gyorsabb
- a fejlesztők bátrabban commitolnak sok kicsit ha nem kell rögtön a központba tenni mindenki szeme láttára a még félkész művüket
- mindenkinél megvan az egész repo (biztonság)
- a hatékony tömörítés miatt nem lesz túl nagy a repo mérete historyval együtt sem

Centralizált vs elosztott verziókövetés

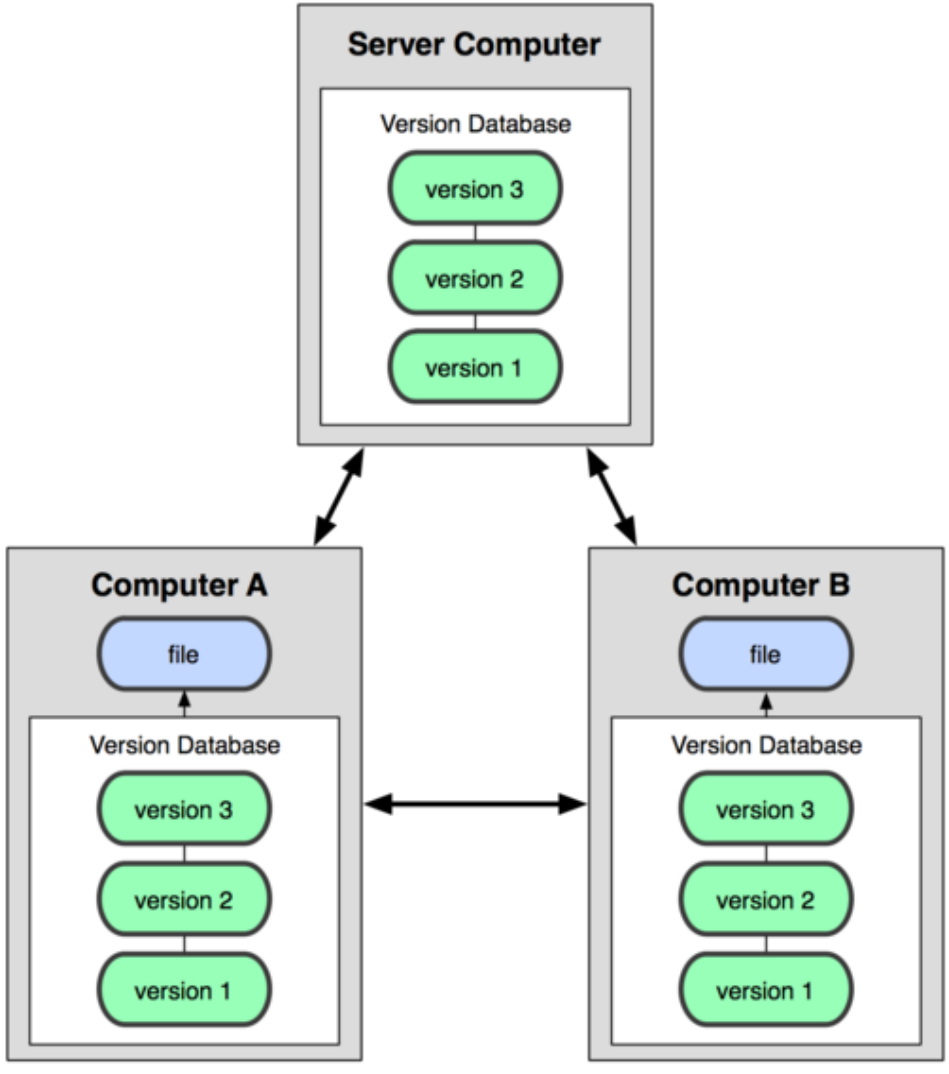
Elosztott rendszer előnyei:

- offline is használható
- lokális műveletek a helyi repository-n: gyorsabb
- a fejlesztők bátrabban commitolnak sok kicsit ha nem kell rögtön a központba tenni mindenki szeme láttára a még félkész művüket
- mindenkinél megvan az egész repo (biztonság)
- a hatékony tömörítés miatt nem lesz túl nagy a repo mérete historyval együtt sem

Centralizált rendszer



Decentralizált rendszer



Harmadik példa

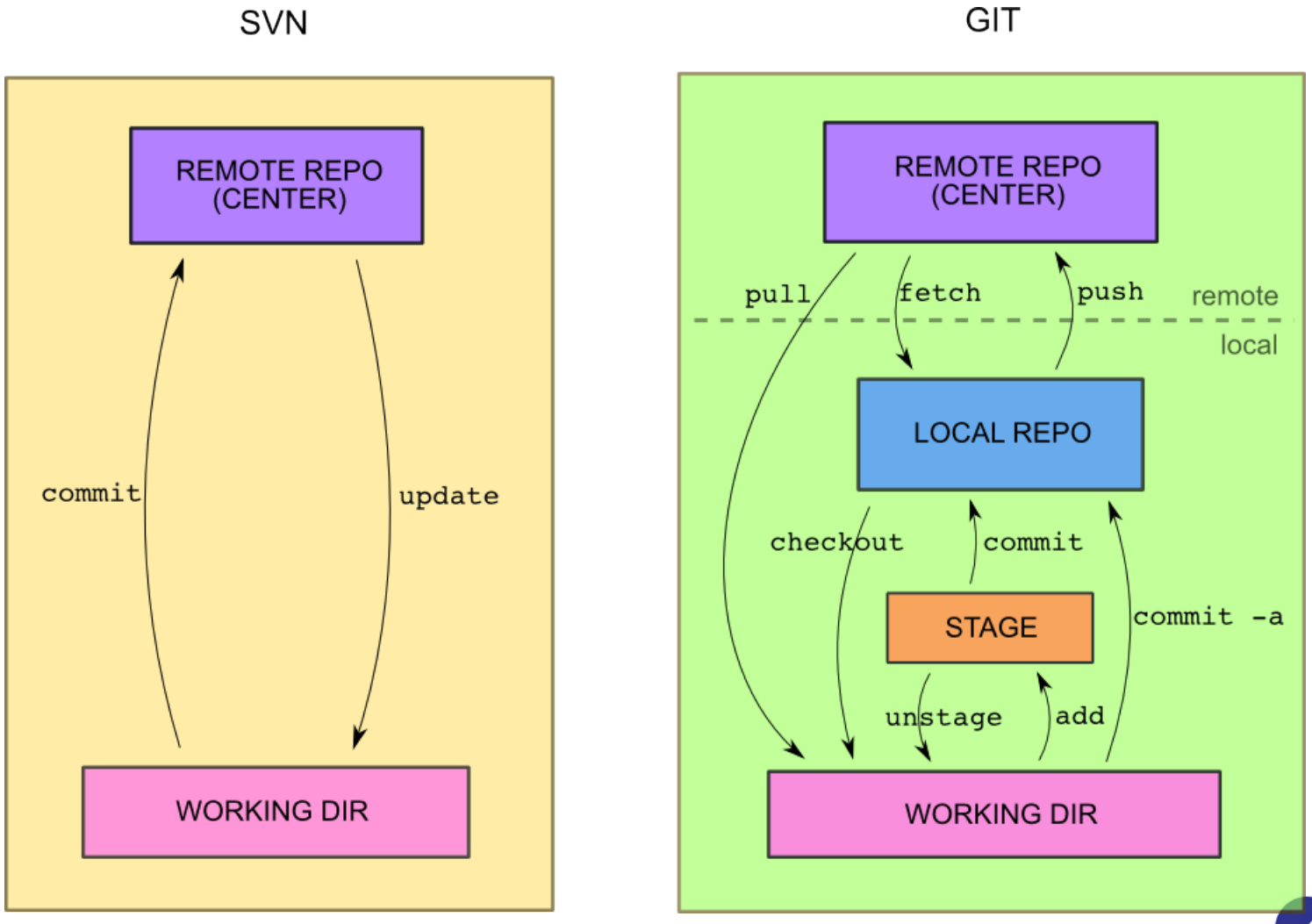
Kommunikáció a központtal:

git clone Lemásol egy projektet a központból. (Nálunk a *gitosis* kezeli a központot, lásd később.)

git pull A központból lekéri a legutóbbi állapotot (persze a historyval együtt) és merge-eli a helyi repóval.

git push Felteszi a legutóbbi commit-jaidat a központba

SVN és git parancsok



Gitosis

A *gitosis* megoldja több központi repository és több felhasználó biztonságos kezelését.

A felhasználók csak egy korlátozott jogú shellhez kapnak hozzáfárást a szerveren. Projektenként egyszerűen megadhatók írási-olvasási jogok, felhasználói csoportok is létrehozhatók.

A felhasználókat a *gitosis* SSH kulcsokkal azonosítja.

Ezért minden olyan gépen amin *git*-et szeretnél használni (illetve a központot is el szeretnéd érni), le kell futtatnod az `ssh-keygen` parancsot, és a generált publikus kulcsot Adri vagy Zsolt beteszi a gitosis-ba.

Áttekintés

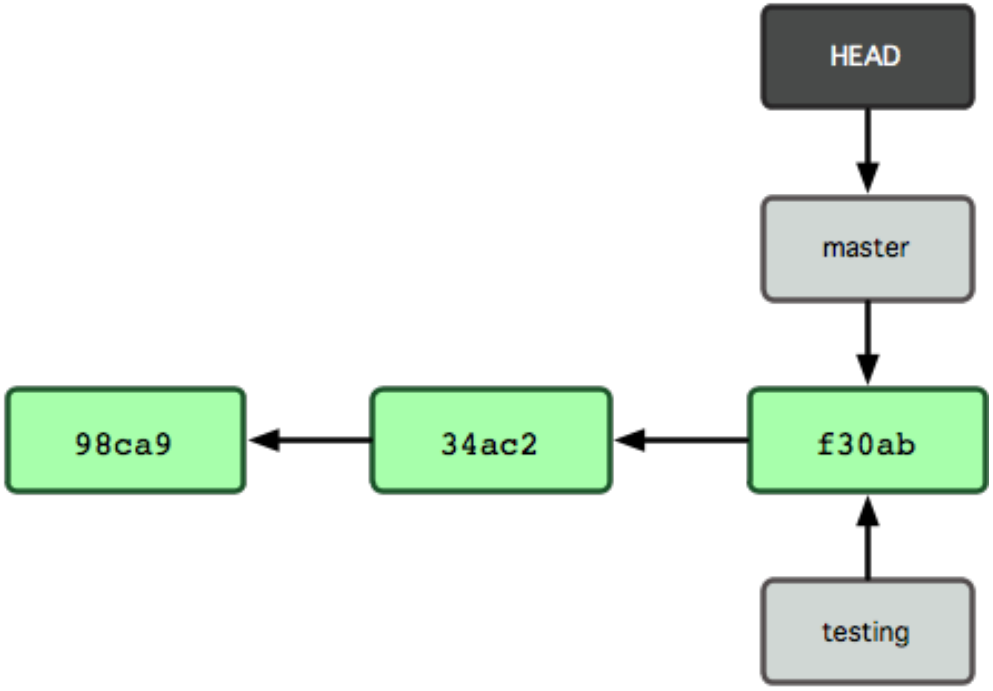
Branching

A *git* egyik legnagyobb erőssége a hatékony branch-kezelés.

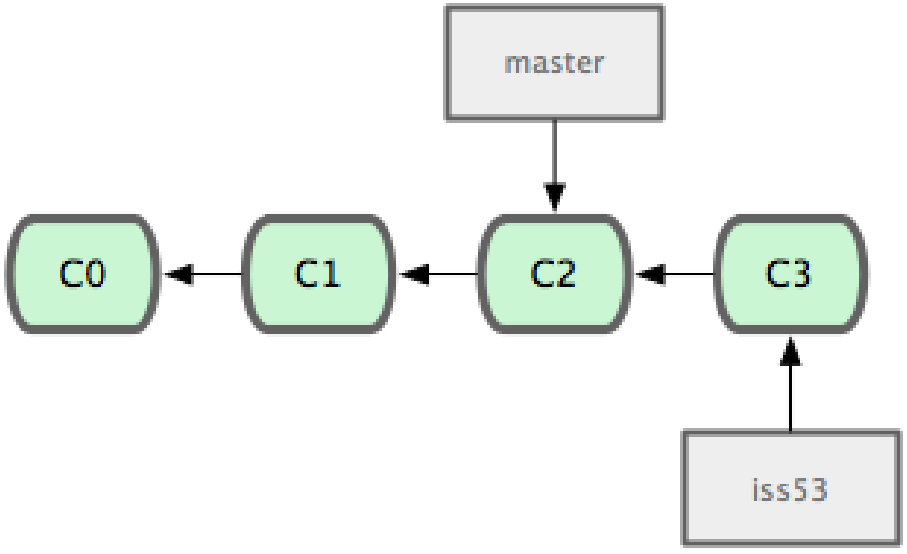
Alap parancsok:

- `git branch` – Kilistázza a brancheket.
- `git branch mybr` – Létrehozza a *mybr* ágat.
- `git checkout mybr` – Átvált a *mybr* nevű ágra.
- `git merge mybr2` – Az aktuális ágba merge-eli *mybr2*-t.
- `git branch -d mybr` – Törli a *mybr* nevű ágat. Csak akkor fog sikerülni ha az ág merge-elve van már egy másik ágba, hogy ne veszítsünk el semmit.

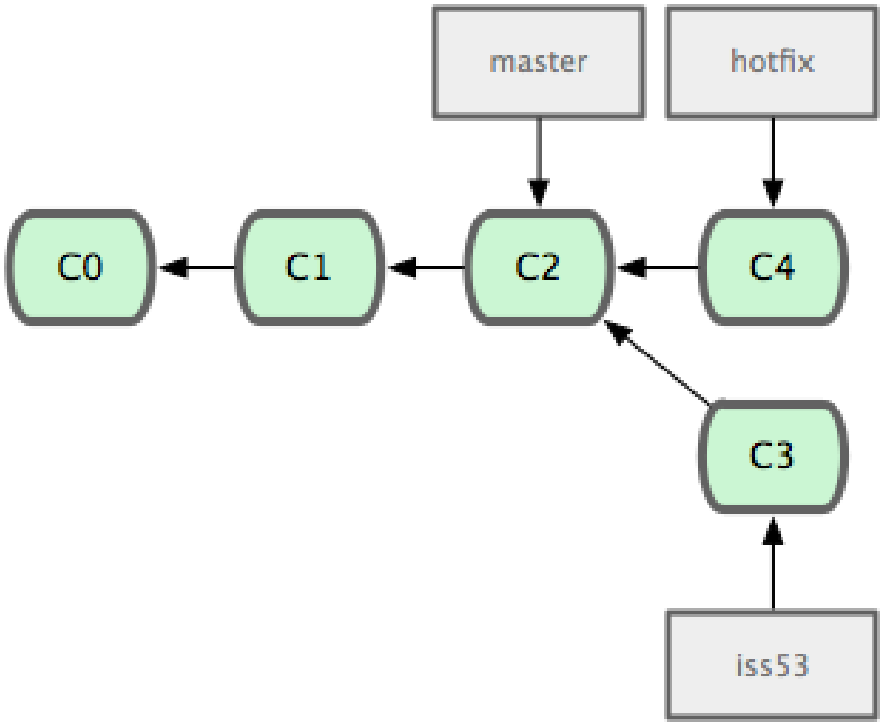
Branching



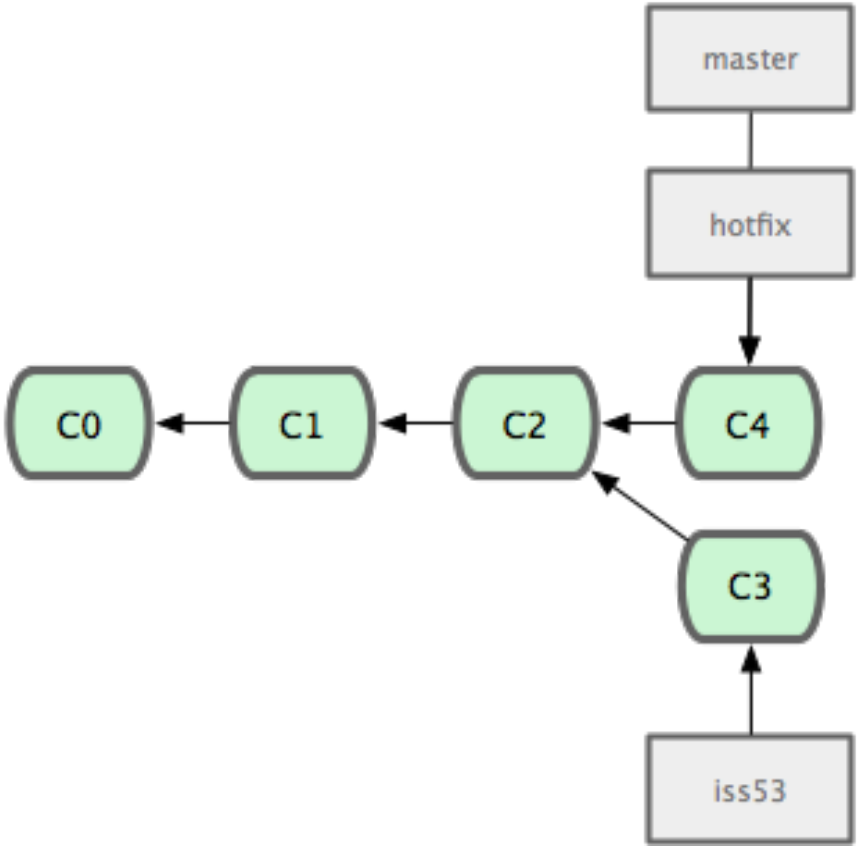
Branching



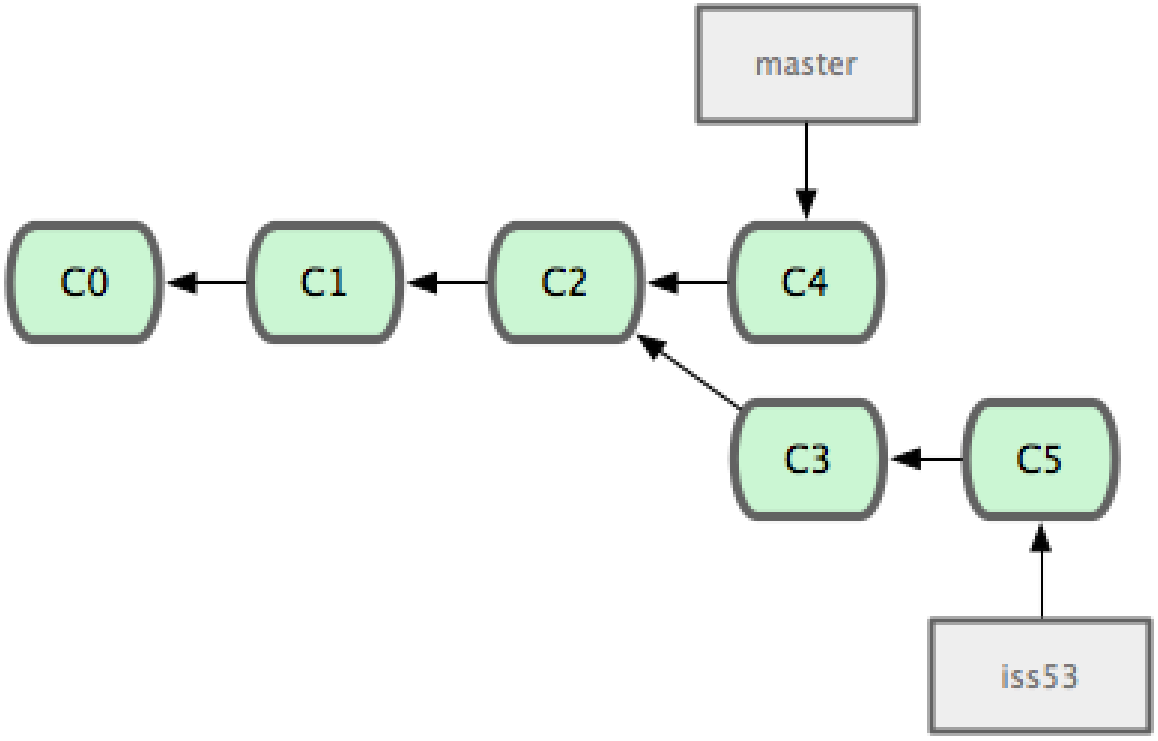
Branching



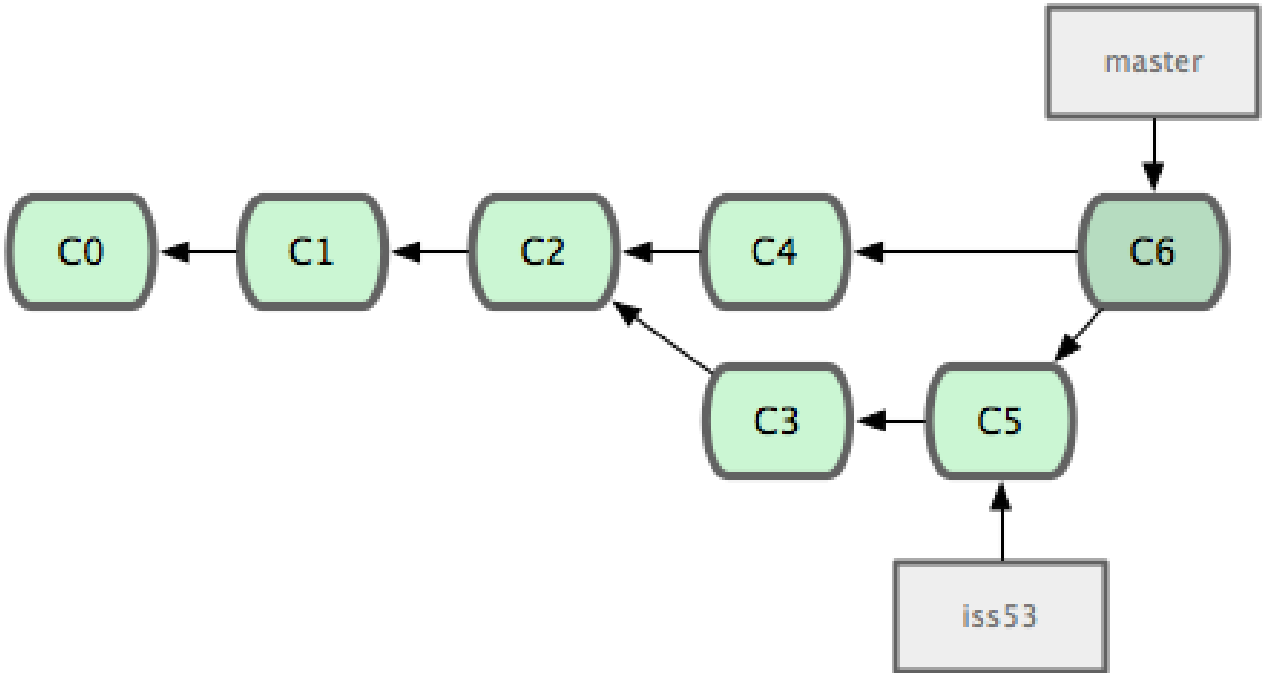
Branching



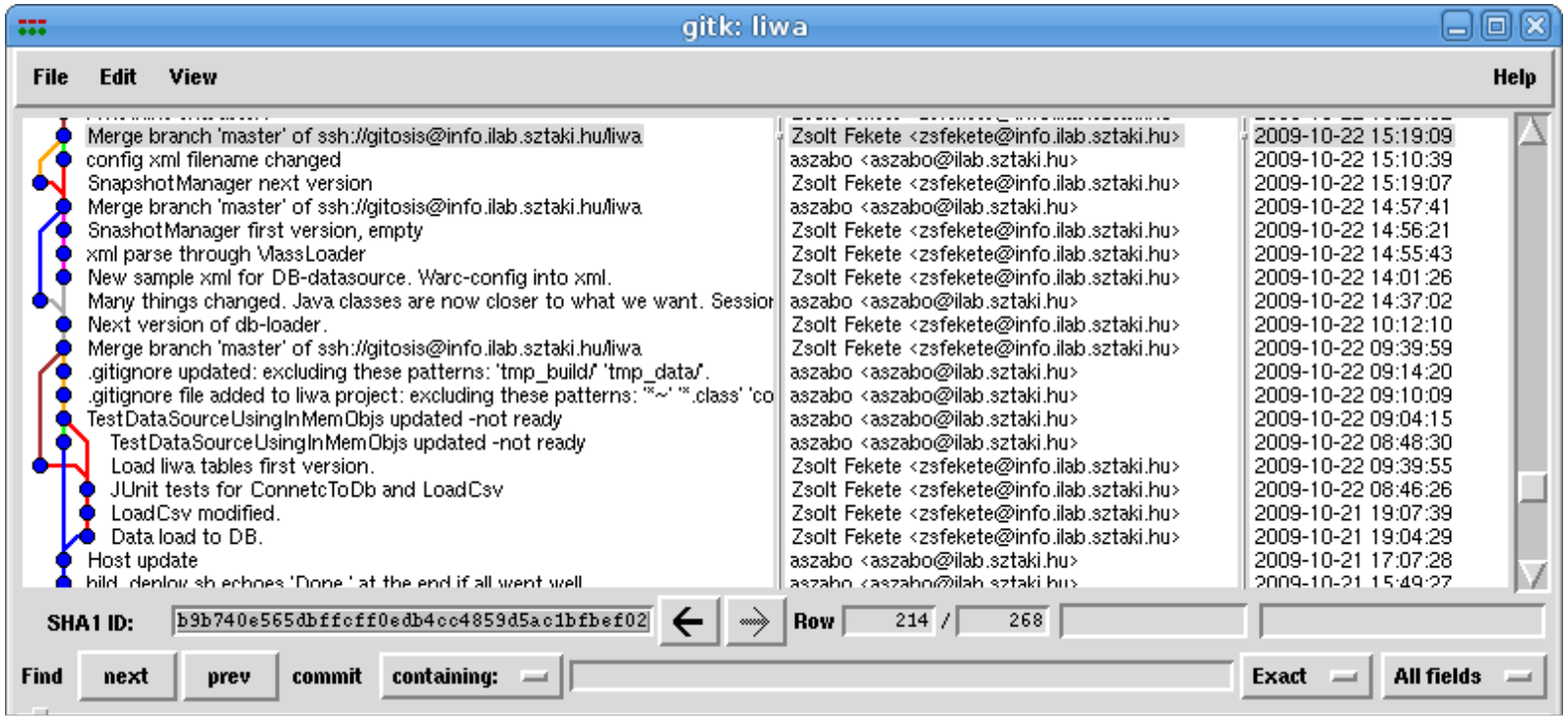
Branching



Branching



Grafikus felület: Gitk



A polc: stash

Néha branch-váltáskor (például amikor csak gyorsan szeretnél megnézni valamit a másik ágon) nem kellemes commitolni csak azért hogy válthass, pedig a váltáshoz tiszta munkakönyvtár kell.

Hogy a munkakönyvtáradat gyorsan kitakarítsd, felteheted a még nem commitolt változtatásaidat és az index(stage) tartalmát „ a polcra", ami egy stack a félkész dolgoknak.

Stash kezelése

- `git stash` – A wd változtatásait elmenti a stack tetejére.
- `git stash list` – Kilistázza a stash tartalmát.
- `git stash show -p` – Megmutatja a legfelső elem changeset-jének változtatásait.
- `git stash apply` – A legfelső mentett changeset-et alkalmazza a munkakönyvtáradra.
- `git stash pop` – A legfelső mentett changeset-et alkalmazza a munkakönyvtáradra, és törli a stack tetejéről.
- `git clear` – Mindent kidob a polcról.

Egyszerűsített history: squash

A sok kicsi kommit lokálisan hasznos, de áttekinthetőbb lesz ha egyben töltöd fel az új feature-t a központba.

Több szekvenciális kommit egybeolvasztható a `--squash` opcióval. Ha a *feature-branch* ágban kész vagy valamivel, akkor ezzel az összes feature-kommitot egyetlen kommitként is merge-elheted a *master-be*.

```
git checkout master
git merge --squash feature-branch
git commit -am 'New feature'
```

Egyebek

- `git svn` – Akkor is használhatsz *git*-et ha a projekt *SVN*-ben van
- `git rebase` – Merge helyett használható, szebb history gráfot eredményez, de nem mindig használható
- `remotes` – Több távoli "forrás" is használható egy repositoryban
- `remote branches` – Branchek létrehozása központban, követésük a helyi repóban
- `git tag` – Verziók, állapotok címkézése.
- `submodules` – Al-projektek (repository-k) egy nagy projekten belül

Összefoglalás

A *git* hátrányai:

- Bonyolult, nehezebb megtanulni
- Nem-standad elnevezések (checkout, revert)
- Nem lehet a projektnek csak egy részét `clone`-ozni
- Nagy bináris fájlokat nem kezeli hatékonyan
- A history átírható

Összefoglalás

A *git* előnyei:

- Ha már megtanultad kezelni akkor hatékony
- Gyors
- Biztonságos: elosztott, és ellenőrizhető az integritás (hash)
- Rugalmas, sokféle workflow-t támogat
- A history átírható

További olvasnivalók I

- Official Git page
  http://git-scm.com/
- Pro Git book
  http://progit.org/
- Git Howto Wiki
  https://textrend.sztaki.hu/cgi-bin/twiki/view/Main/GitHowto