

Verziókezelés a Git használatával

Vajna Miklós

2009. október 31.

Miről lesz szó?

- Miért jó a verziókezelés?
- Miért jó az elosztott verziókezelés?
- Miért jó a Git?
- A Gitről - alulról felfelé
- A Git használata külsősként
- A Frugalware mire használja a Gitet

Miért jó a verziókezelés?

- Mindenki használ verziókezelést, legfeljebb nem tud róla (Mentés másként, tarball + patch-ek, stb.)
- Kollaborációs munkához elengedhetetlen
- Hibakeresést segíti
- Dokumentációs eszköz

- OpenOffice.org *változások követése*
- Telefon kollégának, hogy ott most ne dolgozzon
- Szövegfájloknál kézi 3-way merge
- Saját merge driverek - a legtöbb verziókezelőből hiányzik
- Összetartozó változtatások követése vs idő-alapú, utólagos próbálkozás (cvs2cl)
- Verzió: elosztott rendszerben a szám csak megtéveszt
- Kézi ChangeLog

Miért jó az elosztott verziókezelés?

- A teljes repó elérhető helyben, gyors blame, log, diff, merge
- Nincs szükség hálózati kapcsolatra
- Nincs SPoF
- Megszűnhet a *committer* fogalma
- Backup jelentősége csökken
- Branch/merge egyszerűbbé válik

- Commit early, commit often vö. testsuite
- Felesleges aszimmetria felszámolása
- Gyakorlatban: szükségessé vált korrekt merge algoritmusok implementálása
- Ütközés esetén nem veszhet el a saját munkánk vö.
Subversion `svn up -j`
- Merge ütközések feloldása nem feltétlen a karbantartó feladata

- "A CVS nem a válasz, a CVS a kérdés. Nem a válasz."

Miért jó a Git?

- A legtöbb előny természetesen az elosztottságból fakad
- merge-recursive (vö. Subversion)
- rerere
- blame - kódblokk-áthelyezés érzékelése (vö. explicit másolás/átnevezés)
- git grep
- combined diff

- Egyedülálló funkció
- Egyik legjobb példa arra, hogy mi a baj az explicit átnevezéssel
- Példa:

```
$ git blame -C b.c
607001b b.c (Miklos 20090922 32)    g_free(t);
607001b b.c (Miklos 20090922 33) }
607001b b.c (Miklos 20090922 34)
56bbfb0 a.c (Miklos 20090908 35)  int ip_get()
56bbfb0 a.c (Miklos 20090908 36) {
56bbfb0 a.c (Miklos 20090908 37)    int ip;
```

Miért (volt?) nehéz a Gietet elkezdni használni?

- Tipikus Unix eszköz, hatékony, ha megtanuljuk használni
- Hasonló példák: vim, mutt, stb.
- A Git 1.6.4-es verziójáig 600+ hozzájáruló → rengeteg beállítást támogat, bőség zavara
- Dokumentáció még 1.5 környékén is leginkább referencia jellegű
- Azóta számos könyv megjelent (Git Community Book, Pro Git, Git Magic, stb.)

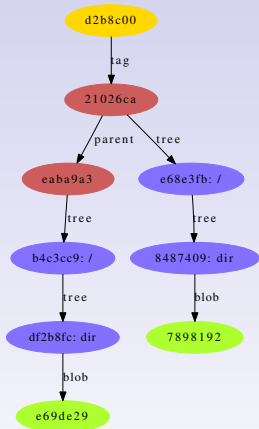
- "Szeretem a színezett diffeket, de vegyük észre, ezek a bizonyos színválasztások csak arra jók, hogy a legtöbb ember egy fondüs villával kikapja a szemét. (...) Szóval, hogy elkerüljük a vak git felhasználók sokaságát, kérlek rakjátok be ez a patch-et."

- A cím a git.git első commitjából van, de:

```
$ ls -lh git-htmldocs-1.5.1.tar.gz
666K 2007 ápr    4 git-htmldocs-1.5.1.tar.gz
$ git shortlog -s -n v1.6.4|wc -l
666
$ git pull
Generating pack...
Done counting 666 objects.
```

- Alacsony szinten egy tartalom szerint címezhető fájlrendszer
- 4 objektum-típus: blob, tree, commit, tag
- blob: egy fájl egy változata
- tree: lehet tree vagy blob, mindegyikből több, de összesen legalább egy
- commit: 0..sok parent, egy tree
- tag: van neve, és bármire mutathat (commitra szokott)

- 2 commit és egy tag esetén:



- Hash számítása:

```
>>> hashlib.sha1("blob 0\0").hexdigest()  
'e69de29bb2d1d6434b8b29ae775ad8c2e48c5391'
```

- Alacsony szinten:

```
$ git init
$ mkdir dir
$ touch dir/file
$ git update-index --add dir/file
$ git update-ref HEAD $(echo A | \
    git commit-tree $(git write-tree))
$ echo a > dir/file
$ git update-index --add dir/file
$ git update-ref HEAD $(echo B | \
    git commit-tree $(git write-tree) \
    -p $(git rev-parse HEAD))
```

- Magas szinten:

```
$ git init
$ mkdir dir
$ touch dir/file
$ git add dir/file
$ git commit -m A
$ echo a > dir/file
$ git add dir/file
$ git commit -m B
```

- ref, symref
- hook
- reflog
- config
- index

- A referencia rövidítése, egy mutató, ami egy sha1-re mutat, vagy egy másik ref-re
- Tipikus ref-ek: HEAD, legtöbbször az aktív branch-re mutat
- A branch-ek olyan ref-ek amik a refs/heads/akármi névvel rendelkeznek és commitra mutatnak
- Tag-ek: refs/tags/akármi néven futnak és tagre vagy commitra mutatnak (signed/lightweight)

- Speciális scriptek, amik bizonyos időpontokban futnak le
- Példa használatukra: push után levél küldése, vagy xml post cia.vc-re
- Commit előtt sorvégi whitespace-ek keresése
- A projekt által megszabott commit message előkészítése

- Ha az utóbbi 10 másodpercben nem változott semmi:

```
$ git checkout "@{10 seconds ago}"
$ git checkout master
$ git log -g --pretty=oneline
402de8e HEAD@{0}: checkout: moving \
    from 402de8e to master
402de8e HEAD@{1}: checkout: moving \
    from master to @{10 seconds ago}
402de8e HEAD@{2}: commit: B
c820060 HEAD@{3}: commit (initial): A
```

Merge vs. rebase

- Kiindulás:

```
      A---B---C topic
      /
D---E---F---G master
```

- rebase:

```
                A'--B'--C' topic
                /
D---E---F---G master
```

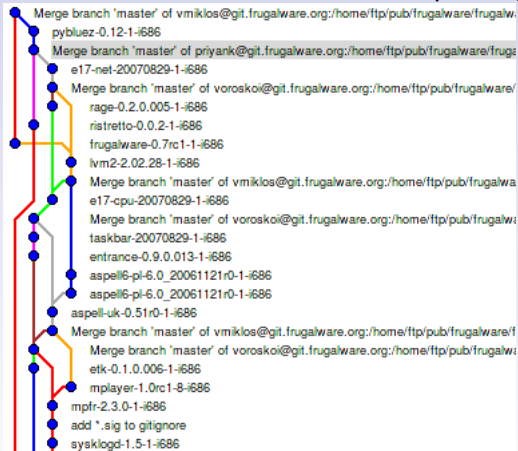
- merge:

```
      A---B---C---H topic
      /           /
D---E---F---G master
```

- AYB Git módra:



- Ha már nehezen áttekinthető a repó, segít a gitk:



- arra utal, hogy pull-kor kell-e majd merge vagy rebase
- jobb a merge ha a repónk referencia mások számára
- jobb a rebase ha szebb historyt akarunk
- git.git esetén: master/next/pu

A Git használata külsősként

- Külsős: nincs commit joga, patch-eket küld
- Helyben persze git-ben dolgozik
- Rebase-el, nem merge-öl
- Interactive rebase: squash, darabolás, rendezgetés
- git format-patch, git am
- Bundle-ök

- Ha git.project.org-on van egy project git repója
- git.or.cz (Novell)
- [gitorious](https://gitorious.org) (Nokia)
- [github](https://github.com) (nem csak opensource)

- Példa: A, B, C, D és B-t szeretnénk B1-re és B2-re
- `git reset --hard B`
- `git reset HEAD^`
- `git add <amit az első commitba szeretnénk>; git commit`
- `git add <amit a másodikba>; git commit`
- `git cherry-pick C`
- `git cherry-pick D`

Git parancsok: sok van, melyik kell nekem?

- A Git 1.6.4 esetén 145 parancs
- Fő magas szintű parancsok
- Mellék magas szintű parancsok
- Alacsony szintű parancsok

- init, clone, add, rm
- status, diff
- commit, reset
- fetch, pull, push
- branch, checkout, rebase, merge
- log, tag, mv, show, grep, bisect

Fő magas szintű parancsok (példák)

- archive, bundle, am és format-patch
- cherry-pick és revert
- describe, shortlog
- gc, clean, stash, submodule

Mellék magas szintű parancsok (példák)

- Manipulálók: config, filter-branch
- Lekérdezés: blame, fsck, verify-tag
- Interakció más rendszerekkel: fast-import, fast-export, archimport, cvsimport, cvsexportcommit, quiltimport, svn

- eredetileg leginkább svn → git
- 30 GiB Subversion repó 3 óra alatt
- mára működő import/export: git, bzz, hg, darcs

- Ha scriptelni szeretnénk
- Példa: log vs rev-list:

```
$ git log --pretty=oneline HEAD~2..
```

```
2920c0c vinagre
```

```
329aae5 gtk-vnc
```

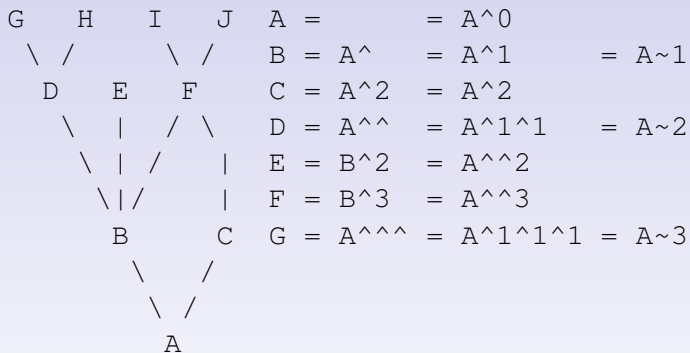
```
$ git rev-list HEAD~2..
```

```
2920c0c
```

```
329aae5
```

Commitok szimbolikus nevei

- Egy példa sokat segíthet:



$$H = D^2 = B^{^2} = A^{^^2} = A\sim 2^2$$

$$I = F^{\wedge} = B^{3^{\wedge}} = A^{^^3^{\wedge}}$$

$$J = F^{\wedge 2} = B^{3^{\wedge 2}} = A^{^^3^{\wedge 2}}$$

Szimbolikus név ha mégis verziót akarunk

- elosztott rendszerben a szám nem egyedi
- mégis van előnye ha van egy folyamatosan növekvő szám
- megoldás: tegyük bele mind a kettőt:

```
$ git describe  
1.1-478-g2920c0c
```

- <utolsó tag>-<azóta a commitok száma>-g<hash>

- Ha a commit nevére akarunk hivatkozni, azt is lehet, pl.:

```
$ git show -s :/"A"  
commit c820060  
Author: Miklos Vajna <vmiklos@frugalware.org>  
Date: Tue Sep 22 00:46:55 2009 +0200
```

A

- Probléma: egy fájlban két módosítás, de csak az egyiket szeretnénk commitolni
- Karbantartás esetén: merge-nél csak az ütközés lenne az érdekes
- Megoldás: index, mint köztes réteg
- git diff, git diff --cached, git diff HEAD

- Az előbbi parancsok térbe helyezve:

```
diff
+-----+
|       |
+-----+
| Objektum- |
|   tároló   |
+-----+
diff HEAD | diff --cached
| +-----+
| | Index |
| +-----+
| | diff  |
+-----+
| Munka- |
| könyvtár |
+-----+
```

- git add az indexhez ad
- git commit az indexből dolgozik
- → ha git add után volt szerkesztés, a git commit az index-beli verziót commitolja!

A Frugalware mire használja a Gitet

- A -current fa csomag-leíróinak tárolására
- A 1.1-516-ge0b7c1e verziónál ez 4603 scriptet jelent, összesen 51263 commit, 46 fejlesztőtől
- Csomagkezelő
- Telepítő
- Dokumentáció és annak fordításai

- Hunk-szintű checkout/reset/stash
- git fast-export --no-data
- Állítható beszédességű hibaüzenetek

- "Kitehetnénk pár hirdetést [a git honlapra] is és a befolyó pénzből egy round-robin alapon működő, fejlesztők közötti pizza-elosztót pénzelhetnénk."

- A verziókezelés jó dolog
- A elosztott verziókezelés még jobb
- A Git egyedülálló lehetőségeket ad a kezünkbe
- A sok funkció ára a bőség zavara, ezzel érdemes számolni

- Innovations in git <http://gitster.livejournal.com/16077.html>
- Git for Computer Scientists
<http://eagain.net/articles/git-for-computer-scientists/>
- Összefoglaló szimbolikus nevekről: `man git-rev-parse`

- GIT honlap: <http://git-scm.com/>
- Levelezési lista: <http://vger.kernel.org/vger-lists.html#git>
- IRC: #git @ irc.freenode.net
- A diák elérhetősége: <http://vmiklos.hu/odp/>