Git gyorstalpaló

> Mi is az a Git?

> Telepítés

v Használat (alapok)

> Git Extensions

> Git Gui

v Parancssorból

Az alábbiak mind működnek linux és Windows alatt is. Linuxban egy terminált kell kérni, Windowsban a Git bash-t.

Először is megmondjuk a Git-nek, kik is vagyunk mink:

```
$ git config --global user.name "Saját Nevünk"

$ git config --global user.email "saját@email.címünk"
```

Ezután Git adatbázist szeretnénk. Git adatbázis mindig egy teljes könyvtárhoz (alkönyvtárakkal együtt) készül. Adatbázisunk kétféleképp lehet: szerzünk egyet, vagy csinálunk egyet a saját könyvtárunkról.

Git adatbázist csinálni egyszerű: a kívánt könyvtárban azt mondjuk, hogy

```
$ git init
```

Ez létrehozza a könyvtárunkban a .git/ adatbázist.

Szerezni is egyszerű. Mi egy *Szerver* nevű gépéről ssh-n keresztül fogjuk hozni-vinni a cuccainkat, ezért ezt írom le (de megjegyzésképp a Git működik http-n, rsync-en, és egy saját git protokollján keresztül is). Feltesszük, hogy ezen a szerveren van már egy *proba* nevű adatbázis (ssh git szerver leírása lejjebb). Amikor először szedjük le a *proba* nevű adatbázist,

```
git clone projfelh@valami.szerver.hu:proba
```

Ez létrehozza a *proba* nevű könyvtárat a *proba* project filejaival, és egy *.git* könyvtárral, ami a *proba* teljes adatbázisát tartalmazza. (Ékezetet semilyen filenévben és felhasználónévben sem érdemes használni. :-))

Van tehát egy könyvtárunk egy git adatbázissal. Ebben a könyvtárban módosítunk valamit, elmentjük ahogy szoktuk a munkánkat, viszont ettől még nem kerül be a Git adatbázisába. Ehhez először ki kell jelölnünk, hogy mely fileokat ajánljuk a Git figyelmébe:

```
$ git add file1 file2 file3
```

vagy

```
$ git add .
```

ha minden file-t a figyelmébe ajánlunk.

Ezután azt kell mondani, hogy

```
$ git commit
```

ekkor egy editort nyit meg, amibe pár szóval érdemes leírni mit dolgoztunk. Humanitárius okokból itt ne használjunk ékezeteket. :-) (Ha nem tetszik az editor amit a Git használ, hanem inkább *emacs*-et használnánk, akkor

```
$ git config --global core.editor emacs
```

) Ha kész, akkor az editor bezárása után a változás el lett mentve a Git helyi adatbázisába. (Megjegyzés: meg lehet mondani a Git-nek, hogy bizonyos fileok engem sosem érdekelnek, részletek itt.)

Most a megváltozott adatbázisunkat szeretnénk visszatölteni a *Szerver*re. Ha klónozással szereztük az adatbázist, akkor ehhez csak annyit kell mondanunk, hogy

```
$ git push
```

Ha viszont a *proba* könyvtárunkról az adatbázis újonan készült, akkor a *Szerver*en a *projfelh* felhasználó üzemeltetőjének el kell küldeni emailben a *proba* nevet, hogy tudathassa a *Szerver*rel, hogy lesz egy ilyen új adatbázis rajta. Ha tudatta vele, akkor a következőket kell tenni:

```
$ git remote add origin
projfelh@valami.szerver.hu:proba.git
```

ezzel megspórolhatjuk, hogy mindig ki kelljen írni a projfelh@valami.szerver.hu:proba.git -et, helyette csak annyit kell majd írni, hogy *origin*. Ezután pedig

```
$ git push origin master
```

ami szabad fordításban annyit tesz, hogy légy szíves az előbb definiált *origin* nevű központi helyre töltsd fel az adatbázisunkat. A *master* egyelőre az egyetlen *ága* a központi adatbázisnak, de ezt is be kell írni.

Ha ezután valamikor máskor dolgozni szeretnénk a könyvtárban, akkor (ha épp nincs meg a könyvtárunk benne a ./git adatbázissal, klónozzuk, lásd fent, ha pedig megvan, akkor)

```
$ git pull origin master
```

az *origin* rövidítésű központi adatbázis *master* ágával (egyelőre ez az egyetlen) frissíti a saját könyvtárunkat (ha kell).

Ha sokan piszkálták egyszerre a *proba* könyvtárat és fel-le töltögettek a központi adatbázisba, a Git legtöbbször olyankor is tudja frissíteni a saját könyvtárunkat (pull) vagy a központi könyvtárat (push), kitalálja hogyan egyesítse a változásokat. Ha ez mégsem sikerülne, klikk ide, ide vagy ide.

Ha nem vagyunk elég bátrak, pull helyett lehet a következőket is csinálni:

```
$ git fetch
```

```
$ git diff master..origin/master
```

ez nem írja át a könyvtárunkban a fileokat, de megmutatja mi a különbség a *Szerver* verziója és a mi verziónk között. Ha a helyzet jó, akkor

```
$ git merge master origin/master
```

átvezeti a változásokat. Ha ez mégsem sikerülne, klikk ide, ide vagy ide.

Ha el vagyunk tévedve, vagy valami nem világos, segíthetnek a következők:

`$ git status`	
`$ git diff`	megmutatja mi változott, de még nem volt rá *add*,
`$ git diff --cached`	megmutatja mi lesz *commit*-olva.

Verziókezelés. Történelem:

`$ git log`	történelem (bővebben).

Minden egyes *commit*nak lesz egy ronda neve, ezt pl. a *git log* megmutatja. Ha kíváncsiak vagyunk milyen fileok tartoznak az adatbázisunkhoz,

```
$ git ls-tree master

$ git ls-tree [a ronda név első pár karaktere]
```

Ha nem adtunk meg elég sok karaktert, a Git panaszkodik, adjunk meg még párat a ronda névből. Ha kíváncsiak vagyunk két *commit* közti különbségre,

```
$ git diff [ronda név]..[ronda név]
```

Ha valamelyik korábbi verziót megint megnéznénk,

```
$ git checkout [a ronda név első pár karaktere]
```

Ekkor a könyvtárunkban visszaáll a *ronda nevű commit*nak megfelelő állapot. Ne ijedjünk meg, nem vesztettük el amit azóta írtunk, egy

```
$ git checkout master
```

visszaállítja a legutóbb *commit*olt verziót a könyvtárunkban.

Elágazás. Tegyük fel, hogy kipróbálnánk valamit, de nem vagyunk biztosak benne, hogy jó lesz a vége. Ekkor elágazhatunk:

```
$ git branch kiserleti
```

ez létrehozott egy *kiserleti* ágat a könyvtárunkból. Ha ezen akarunk dolgozni,

```
$ git checkout kiserleti
```

majd változtassunk a könyvtárunkon, ezután a szokásos

```
$ git add .

$ git commit
```

elkönyveli a változásokat a kísérleti ágban. Bármikor visszatérhetünk a fő ágra:

```
$ git checkout master
```

Melyik ágban vagyunk is tulajdonképpen:

```
$ git branch
```

Mi különbözik az ágakban:

```
$ git diff master..kiserleti
```

Ha írtuk kicsit a kísérleti ágat, esetleg a fő ágat is, és a kísérlet jól sikerült, akkor beolvaszthatjuk a fő ágba:

```
$ git merge kiserleti
```

Ha a merge mégsem sikerülne, klikk ide, ide vagy ide. Sikeres merge után a kísérleti ágat kitörölhetjük (persze nem a történelemből):

```
$ git branch -d kiserleti
```

Ha viszont a kísérlet befuccsolt, és nyomtalanul el akarjuk tüntetni (a történelemből is), akkor

```
$ git checkout master

$ git branch -D kiserleti
```

Tisztítás / tömörítés. Előfordulhat, hogy a *.git* könyvtárunk túl nagyra nőtt, vagy már felesleges objektumokat tartalmaz. (Ez ténylegesen meg is történik, ha

különböző gépeken tárolt *.git* könyvtárakat a Git tudtán kívül szinkronizálunk.) Ezen segít egy tömörítés/tisztítás:

```
$ git gc
```

Elkerülendő a "Jaj mégsem kellett volna kitörölnöm" problémákat olyan *.git*-beli objektumokkal amiket amúgy a Git a hatóköréből már kiiktatott, a *git gc* csak a két hétnél régebbi **és** a Git adatbázisában már feleslegessé vált objektumokat törli. (Tehát nem a két hétnél regebbi verziókat, azok természetesen megmaradnak *git gc* után is.) További részletek itt.

További olvasnivaló: Git könyv (az első fejezet ijesztő, de nem kell); privát ssh repo.

> Központi ssh repository telepítése (elsősorban az ssh Git *Szerver* üzemeltetőjének szól)