## Verziókövető rendszerek

### Verziókövető rendszerek általánosan:

A verziókövető rendszerek arra szolgálnak, hogy a programkódunk minden változatát elmentsék és segítsék a korábbi verziókra való visszatérést és a csoportmunkát. Láthatjuk ki melyik fájlt módosította, mikor, miért, a változtatások pedig könnyen összevethetőek egymással. A szakdolgozatom során egyedül csináltam az alkalmazást, mégis jó hasznát vettem a verziókövetésnek, mert így két gépen is tudtam haladni a fejlesztéssel, attól függően, hogy épp hol voltam, valamint utólag is nyomon tudom követni, mikor mit csináltam.

Csoportmunka során pedig elengedhetetlen a verziókövetés, mert egy remek segítség abban, hogy ne kelljen egymásra várni, hanem egyszerre lehessen dolgozni különböző részeken, sőt, akár ugyan azon a fájlon is.

A verziókövetéshez szükséges egy szerver, ami a verziókezelésbe vont fájlokat tárolja. Első alkalommal feltöltünk egy induló változatot és utána minden egyes újbóli feltöltésnél a rendszer azt fogja eltárolni, hogy melyik fájlok változtak és miben. Ezzel a módszerrel biztosítja azt, hogy lépésenként láthassuk, hogy hogyan változtak a fájlok és bármelyik verzióra vissza tudjunk térni.

A verziókövető rendszerek általában kliens-szerver alapúak: a fejlesztők a szerveren tárolt ún. *repository*-ból jutnak hozzá a fájlokhoz és a változtatásaikat is először a repository-ba küldik be (ez a *commit* művelet), a többi fejlesztő már a *repostory*-ból jut hozzá a változtatásokhoz.

### Verziókövető rendszerek típusai:

Két nagy csoportra bonthatjuk a verziókövetőket: elosztott és központosított.

A központosított verziókövetők a régebbiek, ma már kevésbé népszerűek, itt ugyanis minden változás csak egy helyen, a szerveren van tárolva. Ilyen például a *Subversion*.

Az elosztott verziókövető azt jelenti, hogy bár van egy központosított szerver, minden felhasználónál megtalálható a központi szerveren található *repository* tulajdonképpeni teljes archívuma (biztonsági mentése). Egy üzemzavar vagy meghibásodás esetén ezek bármelyike visszatölthető a szerverre, hogy lecserélje a központi szerveren levő példányt vagy pótolja hiányát.

Az elosztott verziókövető rendszer további jellemzője, hogy nem igényel állandó, megbízható internet-kapcsolatot a központi szerverrel, és emiatt sokkal kevesebbszer fordul a szerverhez, mint nem elosztott társai. A legtöbb művelet a helyi *repository*-ra hat, így jóval kisebb hálózati forgalmat generál és az átlagos válaszideje is kisebb. A két legnépszerűbb és legelterjedtebb elosztott verziókövető a *Git* és a *Mercurial*.

### Verziókövető rendszerek alapfogalmai:

Az alábbiakban a legfontosabb alapfogalmakat és műveleteket ismertetem: [29]

* **Repository** (*tároló*): egy távoli szerver, ezen találhatóak a fájlok
* **Revision** (*változat*): a szerveren található fájlok egy adott változata
* **Commit** (*elkönyvelés*): a helyi változtatások elkönyvelése a továbbításhoz
* **HEAD revision** (*fő változat*): a szerveren található legújabb állapot
* **BASE revision** (*alapváltozat*): a munkakönyvtárban található legfrissebb állapot
* **Differences** (*eltérésmutatás*): két különböző állapot összevetésének naplója
* **Merge** (*összefűzés*): két különböző változat egybefűzése (előfordul, hogy nem sikerül tökéletesen, ilyenkor összetűzés (*conflict*) alakul ki)
* **Conflict** (*összetűzés*): tökéletlen összefűzés (*merge*) során kialakuló helyzet
* **Resolve** (*feloldás*): az összetűzés (*conflict*) megoldása (ilyenkor a munkatárs könyvelés előtt átnézi a konfliktusba került módosításokat és végrehajt egy sikeres összefűzést)
* **Checkout**: a távoli szerver egy adott állapotának lemásolása a gépeden található munkakönyvtárba
* **Working copy** (*munkakönyvtár*): a fájlokat tartalmazó mappa a számítógépeken, ennek tartalma kerül könyvelésre a szerverre
* **Trunk** (*törzs*): a fő fejlesztési ág
* **Branch** (*ág*): a fő ággal és egyéb ágakkal párhuzamosan fejlesztett ágak
* **Tag** (*megjelölt változat*): egy lezárt fejlesztési ág vagy kiadás mappája

### A verziókövető rendszer kiválasztása:

A fenti ismertetésből már látszik, hogy mindenképp érdemes elosztott rendszert használni, még akkor is, ha csak egyedül fejlesztünk valamit. Én személy szerint használtam már mind a három fentebb említett verziókövetőt, és a szubjektív véleményem alapján a Git használata mellett döntöttem. A Subversion azért esett ki a választásból, mert nem elosztott és hálózati kapcsolat nélkül nem használható. A Git és a Mercurial között az döntött, hogy Git-et áthatóbban ismertem meg, valamint gyorsabbnak és gyorsabban tanulhatóbbnak mutatkozott, valamint ennek a kezelése számomra szimpatikusabb és egyszerűbbnek tűnik, mint a Mercurial kezelése.

### Git-hez használatos eszközök bemutatása:

A Git használatához a Windows-ra elérhető támogató eszközöket tartalmazó programcsomagot telepítettem.[33]

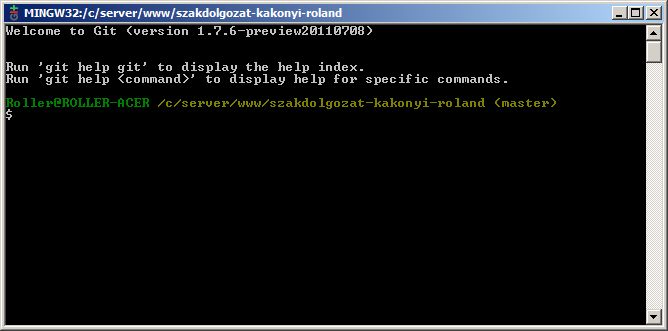
### Git bash:

A Git bash-t a 15. ábra mutatja, parancssori eszköz, használata a Linux alapú rendszerekből ismert *BASH*[34] parancsértelmező működése szerint történik.

A Git bash-t használhatjuk a Git Gui vagy a Gitk grafikus felületek elindításához a "git gui" vagy a "gitk" parancsok begépelésével és futtatásával, vagy parancssorból használhatjuk a már említett grafikus felülettel rendelkező segédprogramokban is elérhető funkciókat. A parancsokat a „git” parancssó után begépelve érhetjük el. A Git bash-ben használható parancsokat a „git help” parancs futtatásával listázhatjuk, ezek rövid ismertetése az alábbi táblázatban található:

|  |  |
| --- | --- |
| add | Fájl hozzáadása a verziókövetéshez |
| branch | A fejlesztési ágak listázását, létrehozását és törlését valósítja meg |
| checkout | Fejlesztési ágak, branchek közötti váltásra szolgál |
| clone | Lemásol egy repository-t egy új könyvtárba |
| commit | Változások elkönyvelése a verziókövezéshez |
| diff | Megmutatja a küönbségeket adott commitok vagy branchek között |
| fetch | Letölti csak a változások listáját a távoli repositoryból |
| grep | Kilistázza a megadott mintára illeszkedő sorokat |
| init | Új repositoryt hoz létre vagy újrainicializál egyet |
| log | Megjeleníti a commit-ok történetét |
| merge | Összevon egy vagy több fejlesztői ágat |
| mv | Átmozgat vagy átnevez egy mappát vagy fájlt |
| pull | Letölt és összefűz egy fejlesztői ágat a helyi változatával |
| push | Elküldi az elkönyvelt változásokat a távoli tárolónak |
| reset | A jelenlegirl visszaáll egy korábbi verzióra |
| rm | Eltávolít egy fájlt a fájlrendszerből és a verziókövetésből |
| status | Megmutatja a jelenlegi állapot változott, de nem commit-olt fájljainak listáját |
| tag | Pillanatnyi állapotról mentést, pillanatképet, úgynevezett címkét készít |

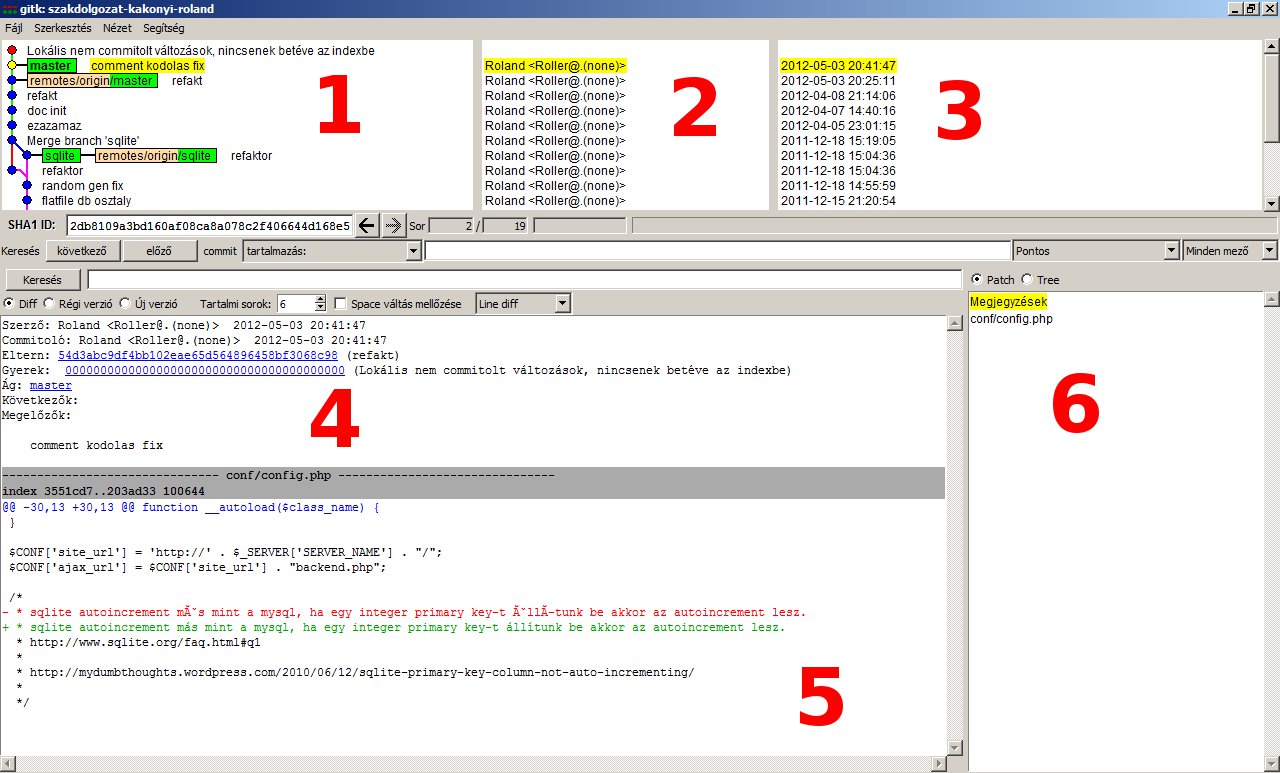
1. táblázat Git bash parancsok



. ábra Git bash működés közben

## Gitk:

A fejlesztési történet vizualizálására szolgál. A verziókövetés során eltárolt változások között böngészhetünk, kereshetünk vagy adott esetben visszatérhetünk a kívánt verzióra.

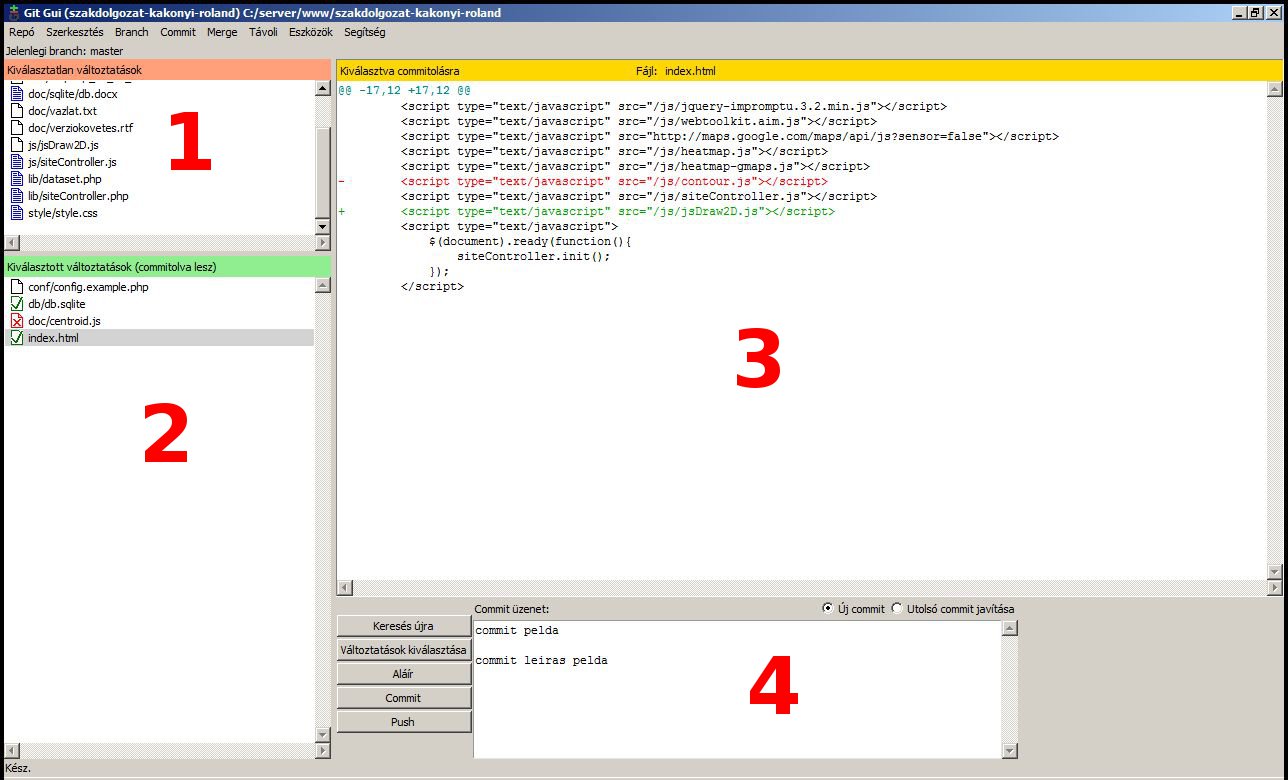


. ábra Gitk működés közben

A Gitk működését a 16. ábra szemlélteti. A felület részei a következők:

1. Verzió fa: elkönyvelt változások listája fejlesztési áganként
2. Az adott változást beküldő felhasználó neve
3. Az adott változás beküldési ideje
4. A változás részletes adatai, fejlesztési ág, szülő commit, gyerek commit
5. Az adott változásban érintett fájlok tartalma
6. Az adott változásban érintett fájlok listája

## Git Gui:



. ábra Git Gui működés közben

A Git Gui működését a 17. ábra szemlélteti. A felület részei a következők:

1. A legutóbbi commit óta változott fájlok listája
2. A jelenlegi commit-ba beválasztott fájlok listája
3. Az 1. vagy a 2. felületen kiválasztott fájlban történt változások
4. A jelenlegi commit-hoz tartozó üzenet

### Verziókövető használata és előnyei a szakdolgozatom fejlesztése során:

Ahhoz, hogy használni tudjuk a verziókövetőt, szükségünk van egy központi szerverre, ahol létrehozhatjuk a repository-t (tárolót). Három népszerű, ingyenes tároló megoldást ismerek, ezek közül választottam ki a fejlesztés során használandót. A *GitHUB* [31] és a *Bitbucket* [32] nagy népszerűségnek örvend, én mégis a *Google code* [30] mellett döntöttem egyszerű, gyors és biztonságos használata miatt. A *Google code* támogatja a Git-et és a Mercurial-t, tehát ebből a szempontból is megfelelő számomra.

Regisztráció után létre tudunk hozni egy repository-t, majd a saját gépünkre le kell *clone*-oznunk (vagyis egy lokális másolatot csinálunk). Innentől kezdve a Git Shell, a Gitk és a Git Gui segítségével végezhetjük a verziókövetést.

Ezek segítségével több lokális ágat (branch-et) tarthatunk fenn, amelyek teljesen függetlenek egymástól. Ezen fejlesztési vonalak létrehozása, egyesítése és törlése csupán másodpercekbe kerül.

Ez azt jelenti, hogy olyan dolgokat tehetünk, mint például:

* Létrehozhatunk egy új ágat, hogy kipróbálhassunk egy új ötletet, néhány változtatás (commit) után visszaválthatunk oda, ahonnan nyitottuk az új ágat, összefésülhetjük az azóta fejlesztett dolgokkal, majd visszaváltunk a kísérletező helyre, és beolvasztjuk.
* Fenntarthatunk egy ágat, amiben mindig csak az éles rendszerbe kerülő dolgok vannak (ezt szokás *master* ágnak hívni), egy másikat, amibe a tesztelésre kerülő munkákat olvasztjuk be, és több kisebbet a mindennapi feladatokra.
* Minden újabb feladatra, amelyen dolgozunk, új ágat hozhatunk létre, hogy gond nélkül váltogathassunk közöttük. Ezek után mindegyiket törölhetjük, amikor az adott dolog beolvasztásra kerül a főágban.
* Létrehozhatunk egy új ágat kísérletezésre; ha esetleg nem jött be a kísérletezés, egyszerűen kitörölhetjük, lemondva az abban végzett munkáról. Ezt az egészet senki más nem látja (akkor sem, ha közben más ágakat feltöltöttünk).
* Amikor egy távoli tárolóba (repositroy-ba) feltöltjük a változásokat, akkor nem kell minden ágat továbbítani, elég csak azt, amin épp dolgoztunk.

A projekteken belül általában vannak olyan fájlok, amiket nem szükséges követni. Ilyenek például a lokális konfigurációs fájlok, amik minden fejlesztőnél mások (pl. adatbázis-kapcsolat, elérési utak, stb.), vagy az ideiglenes létrehozott fájlok. Egy *.gitignore* nevű fájlban beállíthatjuk, hogy ezek a fájlok ne legyenek követve, így mielőtt commit-olnánk, nem is látszódnak, hogy ezek változtak.

Az én szakdolgozatom esetében ilyen nem követett állomány a *config.php*, ami az adatbázis-elérés konfigurációja és az elérési utak beállításait tartalmazza. Helyette egy *config.example* fájl van követve, amiben mindig jelzem a változásokat.

[33] <http://git-scm.com/>

[34] <http://linux.die.net/man/1/bash>