NYÍREGYHÁZI EGYETEM
MATEMATIKA ÉS INFORMATIKA INTÉZET

KISVÁLLALATI SZÁMÍTÓGÉPES MENTÉSI RENDSZEREK LINUX-OS ALAPOKON

Dezső János Loránd
programtervező-informatikus

Konzulens:
Halász Attila Mihály
egyetemi tanár

2017

Tartalomjegyzék

1. BEVEZETÉS

Az 1990-es végén számítógépes hálózatok elterjedésével átalakult a biztonsági mentések elvégzésének módja. Korábban az adatokat manuálisan mentették fizikai külső adathordozókra (mágnesszalag, CD, DVD). Az internet elterjedésével lehetőség nyílt a mentések automatizálására, valamint hogy a fizikailag nagy távolságban lévő gépekhez csatolt adathordozóra való mentés közel azonos időben történjen, így csökkentve a kockázati tényezőket (pl. környezeti katasztrófák, tűz, stb.). A biztonsági mentéseket rendszerezhetjük a mentés gyakorisága szerint, a kiterjedése szerint (teljes vagy részleges). Részleges mentések esetén lehetőség nyílik az inkrementális mentésre, mely adattárolás szempontjából előnyös, mert a korábban mentésre került részek nem kerülnek újra a célnak kitűzött adathordozóra. Hátránya, hogy az alap mentés sérülése esetén nincs lehetőség visszaállítani a korábbi mentéseket. Ennek a kockázati tényezőnek elkerülése miatt célszerű akár több különböző földrészen elhelyezett szerverekre szinkronizálni a számunkra fontos adatokat. A nagyobb cégek által kínált felhőbe való mentés ezt az előnyt már magában foglalja. Szakdolgozatom témája egy önálló biztonsági mentési rendszer létrehozása kisvállalatok számára Linux-os alapokon. Az ok amiért Linux operációs rendszerre esett a választásom, annak magas fokú optimalizálhatósága, melyet 12 éves felhasználói tapasztalataim, valamint a Nyíregyházi Egyetemen végzett tanulmányaim során szerzett tudás alapján kívánom bemutatni.

2. IRODALMI ÖSSZEFOGLALÓ

2.1 Mentési módszerek

2.1.1 Teljes mentés

Minden mentésre kijelölt adat másolásra kerül, Előnye, hogy nem szükséges visszamenőleg elraktározni a korábbi mentéseket a visszaállításhoz, hátránya a nagy tárigény, és a lassú mentés és visszaállítás. Alkalmazása olyan esetekben praktikus, amikor a mentendő adatok mérete nem túl nagy. A egyéb mentési módszerek első lépéseként teljes mentést kell végezni.

2.1.2 Inkrementális mentés

Ebben az esetben mindig egy korábbi mentéshez képest történt változások kerülnek kiírásra. Két típusa van, a kumulatív illetve a differenciális mentés. A kumulatív mentés során a legutolsó teljes mentéshez képest történt változásokat raktározzuk el, míg a differenciális mentés mindig a legutolsó inkrementális mentéshez viszonyított változásokat menti el. Ez utóbbi megoldást használja a több verziókezelő rendszer, pl. git, svn.

2.2 Mentés iránya

A biztonsági mentésünket kezdeményezhetjük a a menteni kívánt adatokat tartalmazó szerverről, valamint ellenkező irányból, a mentés tárolási feladatát ellátó kliens oldalról. A két mentési közötti lényeges különbség, hogy az egyik irányban biztosítanunk kell a szerver jelszó nélküli, ssh kulccsal való elérését. Azt hogy ezt melyik irányból tesszük, függ attól, hogy milyen jogosultságaink vannak a szerveren, illetve hogy melyik irányból gyengítjük a rendszert egy esetleges váratlan behatolás során. Az elkövetkezendő leírásban azt a verziót fogom bővebben ismertetni, amikor a script fájl a kliens gépen fut.

2.3 Mentési céleszköz kiválasztása

2.3.1 RAID technológia

Ajánlott a tárolt adatokat RAID (angolul „Redundant Array of Inexpensive Disks” vagy „Redundant Array of Independent Disks”, mely magyarul annyit jelent, hogy: Független/Olcsó Lemezek Redundáns Sora) technológiával összekötött merevlemezekre menteni, így ha az egyik merevlemez meghibásodna, akkor a párhuzamosan működő merevlemezen még megtalálhatóak az adatok. Létezik szoftveres és hardveres RAID, és különböző szintjeit különböztetjük meg. A hardveres RAID lényegesen drágább, kisebb cégek számára a szoftveres megoldás is kiváló alternatívát jelent. A hardveres esetnél külön dedikált hardver elem biztosítja a merevlemezek

közötti szinkronizációt, levéve a számítógép processzoráról a terhelést. Amennyiben nem sebességkritikus az adatok kiírása, abban az esetben ez nem jelent problémát még egy Raspberry Pi esetén sem, ha megfelelően van kiválasztva a RAID szintje. A legelterjedtebb szintek:

- RAID 0: Ez a szint kizárólag teljesítménybeli előnyökkel jár, az adatblokkok elosztva kerülnek kiírva a két lemez között (tehát a tárolható adat mérete a lemezek kapacitásának összege), így az írás és olvasási sebesség is megnő azonban ha kiesik az egyik merevlemez, elvész az összes tárolt adat. Ezért ezt a megoldást csak olyan esetben érdemes használni, ahol rendszeresen történik biztonsági mentés.
- RAID 1: Itt is minimum 2 merevlemez szükséges, de itt a két lemezen tükrözve vannak az adatok (tehát a tárolható adatmennyiség az összkapacitás fele). Az olvasás gyorsabb, az írási sebesség az ugyanolyan mint RAID nélkül. Amennyiben kiesik egy lemez, a rendszer működőképes marad amíg pótlásra kerül a meghibásodott HDD.
- RAID 5: A RAID 0 és a RAID 1 előnyeit ötvözi, minimum 3 lemez szükséges, az írás és az olvasás sebessége is gyorsabb. A felhasználható tárhely a következő képlet segítségével számítható ki: (legkisebb tárhely mérete)*(felhasznált lemezszám-1). A paritás információk elosztva vannak jelen a lemezeket, ezért ha meghibásodik egy lemez és kicserélésre kerül, akkor automatikusan felépülnek rajta az adatok a rendszer leállása nélkül. Ennek kiterjesztett változata a RAID 6, ahol minimálisan 4 merevlemezre van szükségünk, viszont 2 lemez kiesését is tolerálja a rendszer.
- RAID 10: Minimálisan 4 merevlemez szükséges, szokás RAID 1+0 néven is hivatkozni rá, mert gyakorlatilag 2 pár RAID 0-ba kötött meghajtó kerül tükrözésre RAID 1-el.

A mostani példánkban a RAID 1 konfigurálása lesz bemutatva.

2.4 Linux

A munkahelyemen kizárólag Linux alapú számítógépek vannak hálózatra kötve, így egyszerűsödik a mentés során felhasznált protokollok száma.

A „linux” szó önmagában csak a linux kernelt jelenti, azonban általánosságban a GNU/Linux rendszerekre szokás használni. A linux kernel-t Linus Torvalds írta, és hozta nyilvánosságra 1991. október 5-én. A kernel az operációs rendszer magja, ami megteremti és irányítja a kapcsolatot a számítógép processzora, memóriája és a perifériák között. Ez a legalsó szintje az operációs rendszernek. [1]

Manapság a Linux rendszerek egyre jobban elterjedtek, még ha asztali számítógépek között nem is örvend nagy népszerűségnek, a háttérben rengeteg eszközön megtalálható: a webszerverek, szuperszámítógépek nagy része, okostelefonok túlnyomó többsége, és számtalanféle beépített rendszer (autók, okos lakáshoz eszközök, médiaboxok, okosTV-k, stb.) futtat különböző Linux rendszert. Optimalizálhatósága páratlan, a rendszer felépítése moduláris és átlátható, gyakorlatilag szinte bármilyen konfiguráció módosítható néhány szöveges fájl szerkesztésével, ugyanakkor a biztonságról az összetett jogosultságrendszere gondoskodik. Architektúrák és hardverek széles skálája támogatott.

2.5 SSH

A hálózaton jelen lévő gépek között SSH (Secure Shell) kapcsolattal érjük el az adatokat. Sebesség szempontjából talán kedvezőbb lenne az NFS („Network Filesystem” - Hálózati fájlrendszer) protokoll használata, de mivel a vállalat, melynek mentési rendszerének kiépítését végzem, több telephellyel is rendelkezik, valamint figyelembe véve későbbi bővítési lehetőségeket, az SSH használata nagyobb rugalmasságot biztosít (egy esetleges rendszer újratelepítés során nem kell újrakonfigurálni az NFS klienst és szervert, elég csak a jelszó nélküli SSH kapcsolat létrehozásához szükséges kulcsokat létrehozni, és importálni), és biztonsági szempontokat szem előtt tartva is jobb választásnak tűnik. Továbbá az általunk másolni kívánt adatok mennyisége sem olyan mértékű, hogy érezhető legyen a sebesség növekedés NFS használata során. Célunk az egyszerűség, a könnyebb átláthatóság és egyszerűbb karbantarthatóság.

Az SSH protokollt az OpenSSH programmal tudjuk használni. Ennek segítségével elkerülhetőek a lehallgatások, kapcsolat lopás, és különböző támadások, melyek az interneten továbbított adatokra leselkednek. Ez a program végzi az adatok titkosítását a másolás, és parancs végrehajtások során. Az OpenSSH-val létrejött kapcsolaton keresztül biztonságos csatornát tudunk létrehozni, sokféle hitelesítés közül választhatunk, és egyéb kifinomult konfigurációk segítségével emelhetjük a biztonsági szintjét a munkánknak.[2]

2.6 Bash

A mentési feladatok automatizálására egy futtatható Bash script állományt fogok létrehozni. A Bash szó a „Bourne-Again SHell” kifejezés rövidítése (a szójátékos elnevezés utalás Stephen Bourne-ra, a Unix shell „sh” ősének megírójára). A Unix shell (héj) egy parancsértelmező és egyben programozási nyelv. Parancsértelmezőként felhasználói felületet biztosít a GNU gazdag eszköztárához, melyeket vegyítve programozási nyelvi eszközökkel új parancsokat tudunk

létrehozni egy állományként („GNU is Not Unix” rekurzív kifejezés rövidítése, mely arra utal, hogy a GNU projekt nem összetévesztendő a Unix-al. A 90-es években azért jött létre, hogy az akkor még zárt forráskódú Unix rendszereket kiváltó, de azzal egyenértékű programokat kínáljon). Ezek a Bash script fájl-ok ezt követően ugyanúgy használhatóak a fájl jogosultságaiban meghatározott csoportok és felhasználók által, mint a beépített rendszer szintű parancsok.[3]

A Bash script-ek nagy előnye, hogy nem szükséges hozzá fordító program, ami lefordítja az általunk írt kódot gépi kódra, így a későbbiekben is könnyen szerkeszthető, fejleszthető akár fizikailag távoli számítógépről is. Olyankor célszerű használni, amikor nem igényel nagy számítási kapacitást az elvégzendő feladat, mert természetesen a futás lassabb az olyan programoknál, ahol a fordítás után kapunk egy bináris fájl-t, melyet a gép közvetelenül tud futtatni. Másik nagy előnye a Bash-nek, hogy szinte teljesen platform független, bármilyen GNU/Linux és Unix rendszeren futni fog a programunk.

A Bash shell alapértelmezett a legtöbb GNU/Linux disztribúcióban. Interaktív terminál felületet használva is dolgozhatunk rajtuk akár helyben, akár SSH-n keresztül egy távoli számítógépről. Ez az egyik legelterjedtebb módja pl. a szerverek konfigurálásának és üzemeltetésének, ahol legtöbb esetben nincs is szükség grafikus felületre, mivel azok csak további hibalehetőséget, és támadási felületet kínálnának a támadóknak. Az általános vélekedés programozói körökben, hogy több kód (program), több hiba.

Igény szerint felépíthetünk a Bash script-ben megírt programunkhoz egy egyszerűbb felhasználói felületet is pl. Zenity vagy YAD („Yet Another Dialog” - Zenity fork) programokat használva.

2.7 Biztonsági mentést segítő szoftverek

2.7.1 Másolás, szinkronizáció (dd, tar, cp, mysqldump, rsync, partimage)

- dd – komplett háttértárak (pl. pendrive, CD, DVD, HDD, stb.), illetve partíciók másolására szolgáló program, segítségével akár a bootszektorról is csinálhatunk biztonsági mentést
- tar – archiváló program, képes könyvtárakat és fájlokat egy állományba írni, úgy hogy azoknak a jogosultságai is megmaradjanak, gyakran valamilyen tömörítőeljárással kombináljuk használatát (pl. gz, bz, xz stb.)
- cp – fájlok, könyvtárak másolása

- mysqldump – MYSQL adatbázis archiválása, a mentéskor létrejövő sql kiterjesztésű fájl gyakorlatilag sql parancsok gyűjteménye, melyek létrehozzák a mentett adatbázis
- rsync – gyors és sokoldalú távoli és helyi fájlszinkronizációs program
- partimage – partíciók egyszerű mentése és visszaállítása

2.7.2 Tömörítés (gzip, bzip2, xz)

A mentés során célszerű tömöríteni a menteni kívánt adatokat az adatátvitel felgyorsítása és az igényelt tárolóhely méretének optimálisabb kihasználtsága érdekében. A legelterjedtebb tömörítési mód linuxos környezetben a „gzip”, „bzip2” és a „xz”. Az interneten több oldal is foglalkozik ezek összehasonlításával, többnyire a linux kernel tömörítésén szokták tesztelni őket (Pl. [4]). Az eredmények összehasonlításakor érdemes figyelembe vennünk a tömörítés idejét, és a tömörítési hányadost, mely megadja, hogy az eredeti méret hány százalékára sikerült adott beállítással összezsugorítani az eredeti adatot. Mind a 3 vizsgált tömörítőnek 1-9-es skálán megadható, a tömörítés mértéke, ahol minél nagyobb számot kapunk, annál több időt szán a program a minél jobb tömöríthetőségre. Az ábrákat vizsgálva nekünk az „xz” tömörítő 2-es mértéke meg is felel. Ha alacsony memória- és számítási igény felhasználásra, és gyorsaságra törekszünk akkor célszerű lehet a „gzip” használata. Amennyiben fontosabb számunkra, hogy kisebb helyet foglaljon el a tömörített állomány, és ezért hajlandóak vagyunk feláldozni számítási kapacitást, akkor az „xz” lesz a legjobb választás. A konkrét felhasználást a példa során fogom tárgyalni.

2.8 Script formázási elvek

A script-ünk megírásakor célszerű figyelemebe venni néhány jó tanácsot, mellyel megkönnyíthetjük a parancsfájl újrahasznosítását, és a későbbi továbbfejlesztést.

2.8.1 Megjegyzések

Amint korábban említve volt, a „#” jellel lehet megjegyzéseket hozzáfűzni a kódunkhoz. Ezeket ott célszerű használni, ahol nem egyértelmű a kód, és segítheti a laikus szemlélő eligazodását, és plusz információt nyújthat a kód mibenlétéről. Általános szokás a script elején a következőképp megadni a fájl nevét, a szerzőt és egy rövid leírást. Kivételt képez a script legelső karaktere, amennyiben felkiáltójel követi, akkor a script futtatását végzendő binárisra (interpreter) mutató útvonalat definiáljuk vele (shebang, hashbang):

```
#!/bin/bash
```

```
# Fájl: mentes-xy-szerverrol.sh
# Leírás: napi, heti és éves mentés az xy szerverről
# Szerző: Dezső János
```

2.8.2 Idézőjelek

A Bash a szimpla és dupla idézőjelek között különbséget tesz. A szimpla idézőjelben megadott szöveget egy-az-egyben szövegként kezeli. Amennyiben dupla idézőjelet használunk, akkor a szövegben elhelyezett változókat először kiértékeli. pl.:

```
$ echo 'szimpla idézőjellel $HOME'
szimpla idézőjellel $HOME

$ echo ”dupla idézőjellel $HOME”
dupla idézőjellel /home/felhasználónév
```

2.8.3 Szögletes zárójelek

A szimpla és dupla szögletes zárójelek között szintén különbséget kell tennünk. A szimpla szögletes zárójel az „sh” shell alapértelmezettje, a dupla szögletes zárójelet a Bash script kiterjesztett tulajdonságokkal ruházza fel. Amennyiben törekszünk arra, hogy a script sh kompatibilis legyen, akkor kerüljük a dupla szögletes zárójelek használatát. Tapasztalatom szerint, a Bash elég elterjedt ahhoz, hogy ne okozzon fennakadást ennek használata. A lényegi különbség, hogy a dupla szögletes zárójelekben:

- nem szükséges idézőjelebe rakni a változókat
- lehet használni a „&&” és a „||” jeleket a „<”, „>” és „=” karakterekkel összehasonlító logikai tesztekhez.
- Szintén lehet használni a „*”-ot mint joker karakter-t. Pl a következő feltétel bármilyen y-al kezdődő válasz esetén igazat ad:

```
if [[ $valasz = y* ]]
```

Kiváló összefoglaló található a BashFAQ oldalán. [5]

2.8.4 Konstansok

A használt konstansokat célszerű a script elején rögzíteni és csupa nagybetűvel megadni a nevüket. Használhatunk „`readonly`” előtagot, mely biztosítja, hogy véletlenül se írjuk felül az értékét a script futtatása során. A konstans neve legyen beszédes, a későbbi munkánkat megkönnyítve, aláhúzással válasszuk el a szavakat jobb olvashatóság miatt. Pl:

```
# Pelda konstans
readonly PELDA_KONSTANS=”konstans szöveg”
```

2.8.5 Változók

A változók neveit kisbetűvel szokás írni, aláhúzással elválasztva a szavakat, vagy úgynevezett „CamelCase”-el, mely használatakor a szavakat egybe írjuk, de minden szó első betűje kapitális. A függvényekben használhatjuk a „`local`” előtagot, hogy elkerüljük az esetleg ütközéseket a globális változókkal. A változók értékadásakor egy egyenlőségjel után írhatjuk a felvenni kívánt értéket. Amennyiben a változóban tárolt adatra szeretnénk hivatkozni, akkor a változó neve elé „$” kerül. Pl:

```
elso_valtozo=”egy szöveg”
echo $elso_valtozo
```

2.8.6 Feltételek

Az alábbi példa szerint vizsgálhatunk feltételeket amennyiben két ágú a feltétel kimenete:

```
if feltetel ; then
    parancs
else
    parancs
fi
```

Ha többféle értéket is vizsgálnunk kell akkor a „`case`” parancs alkalmazandó:

```
case kifejezés in
    eset1)
        parancs
    ;;
    eset2)
        parancs
    ;;
esac
```

Érdemes megjegyezni, hogy az egyenlőséget többféleképp is vizsgálhatjuk:

- „=” karakter a „POSIX” szabvány, karakterláncot (stringet) hasonlít össze
- „==” karakterek karakterláncot (stringet) hasonlítanak össze
- „-eq” karakterek esetén az értékek intiger-ré alakításuk után kerülnek összehasonlításra

2.8.7 Függvények

Függvényeket kétféleképp deklarálhatunk. Implicit deklaráció:

```
fuggveny_nev () {
    return 0
}
```

Explicit deklaráció:

```
function fuggveny_nev () {
    return 0
}
```

Az explicit deklaráció könnyebbé teszi a script olvasását, azonban nem POSIX szabvány, így csak a Bash (és néhány másik) parancsértelmezővel kompatibilis. Amennyiben törekszünk a kód hordozhatóságára, akkor ne használjuk.

2.9 Verziókezelés (git)

Munkám során verziókezelő rendszert használtam, melynek lényege, hogy a fejlesztés alatt álló állomány(ok) különböző változatait, verzióit a későbbiek folyamán könnyen vissza lehessen keresni. Ez lehetőségek tárházát biztosítja a kód későbbi újrahasználásra, illetve más irányú továbbfejlesztésére (fork-olás), különböző variációk teljes körű tesztelésére. Lehetőség van, hogy több ember is dolgozzon ugyanazon a programon, így elágazások (branch-ek) keletkezhetnek, melyeket tetszés szerint lehet visszaolvasztani a fő (baseline, mainline) verzióba. Amennyiben a verziókezelést egy nyílt platformon hajtjuk végre, akkor lehetőséget biztosítunk, hogy akár ismeretlen emberek is továbbfejleszthessék az általunk előállított kódot, és szélesebb körben nyílik lehetőség a tesztelésre is, így az esetleges hibákat jelezhetik a tesztelők bugreport-okban.

A verziókezelőket kezelési modell alapján 2 csoportra bonthatjuk attól függően, hogy létezik-e egy kinevezett fő szerver (központosított modell, pl. svn, cvs), vagy minden fejlesztői gép egyenrangú tárolóként jelenik meg (elosztott modell, pl. git). A központosított modell megvalósítása egyszerűbb, jó biztonsági mentésnek is minősül, könnyebb a műveletek visszavonása. Az elosztott rendszerek nagy előnye, hogy nem kell egy folyamatos online jelenlétet biztosítani a verziókezelést végző szervernek, és sokkal gyorsabb, de nehezebb menedzselni ha sok résztvevője van a fejlesztésnek.

Én személyesen a git verziókezelő rendszert részesítem előnyben. A git-et Linus Torvalds, a Linux kernel megalkotója hozta létre 2005-ben, hogy a kernel fejlesztését ennek segítségével

végezzék. A korábban használt verziókezelővel problémák adódtak (BitKeeper), és egy elosztott működésű rendszert szeretett volna.

Git szervert magunk is létrehozhatunk, de a dolgozatom nem terjed ki ennek tárgyalására. Léteznek szolgáltatók, melyek kiváló ingyenes git tárolókat nyújtanak, így egyesül a központi és a elosztott modell minden előnye. A három legnépszerűbb szolgáltató a GitHub, BitBucket és a GitLab. Mind a három szolgáltató támogat többféle verziókezelőt, az interneten sok oldalt találni ami az összehasonlításukkal foglalkozik, én most csak a leglényegesebb erősségeket említeném meg.

A GitHub a nyílt forráskódú fejlesztések között a legelterjedtebb, „korlátlan” számú közreműködő vehet részt egy-egy projektben ingyenesen amennyiben az publikus. Ha érzékeny információt tartalmaz a forráskód, azaz magán tárolókat szeretnénk, azért fizetni kell.

A BitBucket ezzel szemben ingyen kínál magán tárolókat is, azonban a részvevők számát korlátozza 5 főben. Ha ennél több ember vesz részt a fejlesztésben, akkor az pénzbe fog kerülni.

A GitLab pedig egy teljesen nyílt forráskódú megoldás, mellyel lehetőségünk nyílik saját verziókezelő rendszert kiépíteni webes felülettel. Nagyobb projektek számára lehet ez az optimális választás.

A verziókezelést elvégezhetjük természetesen parancssorból, de léteznek különböző grafikus kliensek is. Utóbbi csoportba tartozik az általam is ajánlott GitKraken, mely nem kereskedelmi célra ingyenesen használható, és egyszerű átlátható kezelői felületével gördülékennyé teszi a munkát. Továbbá grafikus felülettel rendelkező git kliensek: gitg, git-cola, GitEye. Ezek közzül a GitEye biztosítja a legtöbb funkciót.

Alapvető git parancsok:

Első futtatáskor konfiguráció:

```
$ git config --global user.name  "Minta Pista"

$ git config --global user.email "pista@minta.hu"
```

Ezt követően miután odanavigáltunk a leendő projektünk könyvtárába, git tárolónkat inicializálni kell, mely létre fog hozni egy „.git” könyvtárat a projekt gyökérkönyvtárában, ami tartalmazni fogja a verziókezeléssel kapcsolatos információkat:

```
$ git init
```

Fájl-t a következő képp tudunk hozzáadni a tárolónkhoz:

```
$ git add file1 file2
```

Amennyiben az összes fájlt hozzá szeretnénk adni az adott könyvtárban, akkor:

```
$ git add .
```

Ezt követően a hozzáadott fájlok egy úgynevezett „staging” állapotba kerülnek. Amint összegyűlt a fejlesztés során egy adott csoportba tartozó változtatás a fájlokban, és mindent „staging” állapota raktuk, ezt követően jöhet a „commit”, ami a változtatások elkönyvelése, és itt adunk egy átfogó leírást a elvégzett változtatásokról:

```
$ git commit -m "Változtatások összefoglalása."
```

Ezzel a saját tárolónkban meg is történt a változtatás, azonban, ahhoz hogy a többi fejlesztőnél is látszódjon az általunk módosított dolgok, még egy parancs kiadására lesz szükség:

```
$ git push origin branch
```

Mielőtt azonban rendszeresen használni tudjuk ezt a parancsot, első alkalommal definiálni kell az „origin”-t, azaz hogy hova akarjuk menteni a változtatásokat.

```
$ git remote add origin pista@minta.hu:proba.git
```

Ha a mások által (másik gépen) történt változtatásokat szeretnénk lekérdezni, a következő parancs lesz a megfelelő:

```
$ git pull origin master
```

Sok további parancs opció létezik, melyekkel kezelni és átlátni tudjuk a változásokat. A git honlapján részletes dokumentáció foglalkozik ezzel.[6]

Kiváló magyar nyelvű összefoglalók található az alábbi weboldalakon:

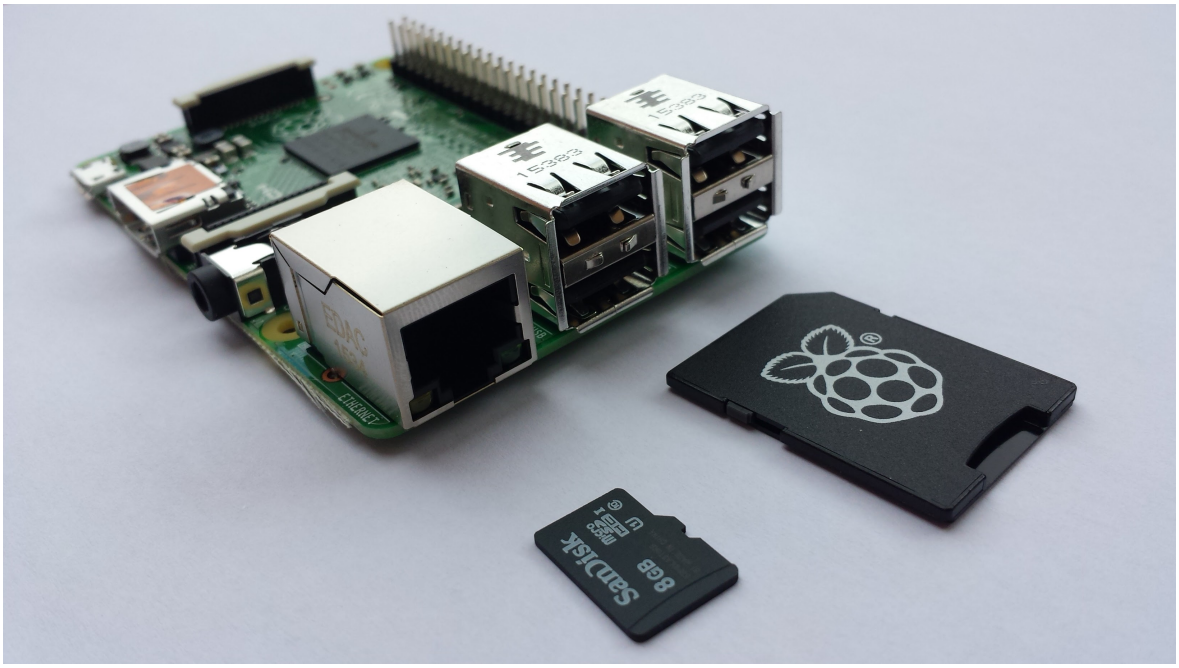
```
http://math.bme.hu/~balazs/git/gitcml.html
```

```
http://wiki.hup.hu/index.php/Hogyan_haszn%C3%A1ljuk_a_Git_verzi%C3%B3kezel
%C5%91_rendszert
```

3. RENDSZER ÉS MÓDSZEREK

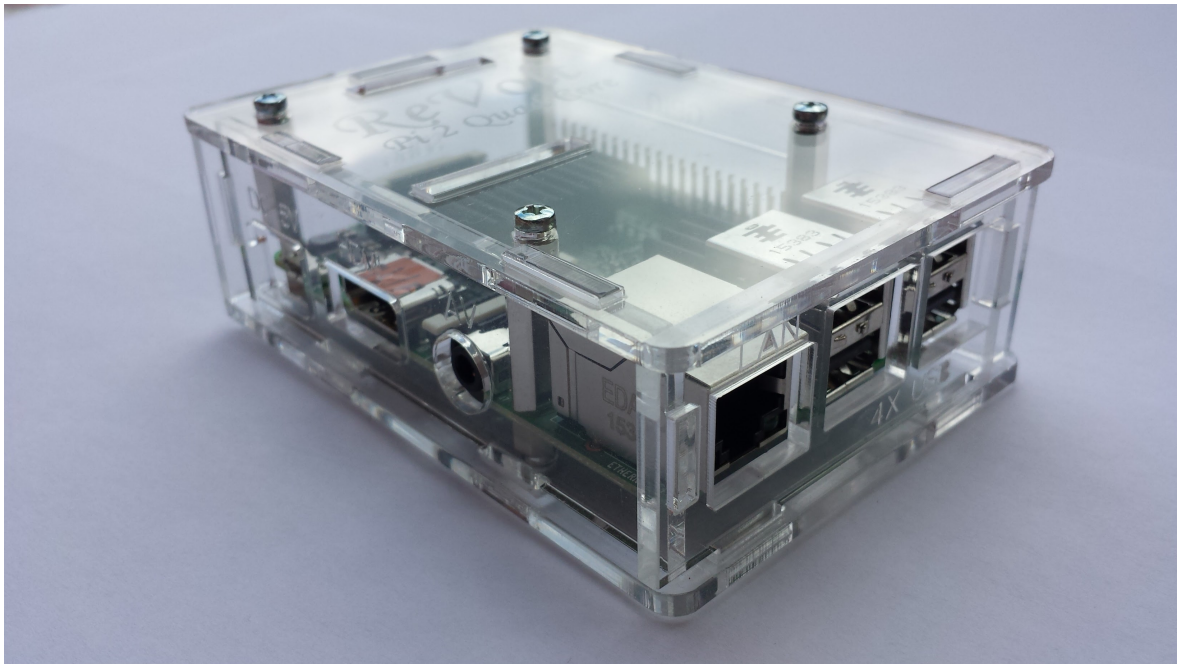
3.1 Hardver

A feladat elvégzésre 2 darab Raspberry Pi 2 Model B 1GB (900MHz-es Broadcom BCM2836 ARMv7 Quad Core CPU & 1 GB 450MHz SDRAM) egykártyás számítógép lesz üzembe helyezve (1. Ábra). A választás azért esett erre a termékre, mert ez egy mozgó alkatrész nélküli modell, mely hangtalanul és alacsony fogyasztás mellett tudja a háttérben végezni a szükséges feladatokat, és ez a legelterjedtebb típus az egylapos számítógépek közül, ehhez érhető el a legtöbb dokumentáció, és széles palettája a kompatibilis Linux alapú operációs rendszereknek és a karbantartott programok ezreinek. A 2 gép két külön telephelyen kerül beállításra, így biztosítva az adatok biztonságát.



1. Ábra: Raspberry Pi 2 Model B

Online boltból vásároltam a terméket, melyhez opcionálisan lehet vásárolni plexi keretet (2. Ábra), mely megakadályozza a por lerakódását, ezáltal a hűtés hatékonyságát tovább megőrzi, mivel nincs ventilátorral ellátva a termék processzora. Ezen kívül véd a fizikai behatások ellen is.



2. Ábra: Egyszerű plexi tokban

3.2 Operációs rendszer

A Raspberry Pi hivatalosan is támogatott operációs rendszere a Raspbian. Ennek telepítését és konfigurálását fogom bemutatni. Elérhetőek harmadik fél által karbantartott rendszerek is: Ubuntu Mate, Snappy Ubuntu Core, stb. A Raspbian-t két féle módon is fel lehet telepíteni. Az egyik mód, amikor közvetlenül a MicroSD kártyára másoljuk a rendszer fájljait, a másik mód a NOOBS által kínált megoldás, mely magában foglalja néhány másik OS telepítésének lehetőségét is, amennyiben rendelkezésre áll a telepítés során internetes kapcsolat. Ezek a következőek:

- Pidora (Fedora alapú rendszer)

- OpenELEC (Kodi Linux-on alapuló multimédiás rendszer)

- OSMC (multimédiás rendszer)

- RISC OS (teljesen önálló operációs rendszer, az ARM architektúra fejlesztőitől)

- Arch Linux (ARM-re optimalizált változata az egyre népszerűbb rendszernek)

3.2.1 Operációs rendszer telepítése (wget, unzip, SHA-1, fdisk, dd)

A legegyszerűbb telepítési mód, ha közvetlenül a MicroSD kártyára írjuk a Raspbian honlapról letölthető képfájlt. Ehhez látogassunk el a következő címre:

```
https://www.raspberrypi.org/downloads/raspbian/
```

Az oldalon a „Download ZIP” ikonra kattintva tölthetjük le az aktuális legfrissebb verziót, amely jelenleg a Raspbian Jessie. Terminálból a következő paranccsal tudjuk letölteni:

```
wget https://downloads.raspberrypi.org/raspbian_latest
```

Két Raspbian változat között választhatunk, az egyik teljes asztali környezettel érkező rendszer, a második pedig a Lite verzió, mely csak a minimális rendszert tartalmazza, asztali környezet nélkül. Mi most az egyszerűség kedvért a teljes verziót töltjük le, így kényelmesebb konfigurálni a rendszert.

A letöltési link alatt található egy SHA-1 (Secure Hash Algorithm – biztonságos hasító algoritmus) kód, mely segítségével ellenőrizhetjük, hogy a letöltött fájl megegyezik-e a szerveren található fájllal. Az SHA-1 egy kriptográfiai hash (hasító) függvény, mely segítségével egy fájlból képezhetünk egy adott hosszúságú kódot. Ez a kód különbözik, amennyiben az operandusként megadott fájl tartalma eltér. Egyfajta elektronikus aláírásként működik, az 1980-as években kezdték fejleszteni az ilyen típusú algoritmusokat. A letöltött fájl SHA-1 kódját megkapjuk az alábbi parancs kiadásával:

```
sha1sum /eleresiut/2016-05-27-raspbian-jessie.zip
```

A parancs kimenetében először a hash kód lesz, utána a fájlnév. Amennyiben megegyezik a honlapon mutatott értékkel, úgy hozzá is láthatunk a kitömörítéshez. Ennek során ügyelni kell, hogy ne felejtsünk el megadni elérési utat a kicsomagolandó fájlnak, mert alap esetben abba a könyvtárba csomagolja ki a gép, ahol kiadtuk a parancsot (a kitömörített img kiterjesztésű fájl mérete 3,7 Gbyte). Az elérési utat a „-d” kapcsolóval tudjuk megadni:

```
unzip /eleresiut/2016-05-27-raspbian-jessie.zip -d /eleresiut
```

A kitömörítés után hozzáláthatunk az img fájl MicroSD kártyára írásához. Ehhez meg kell adnunk, hogy melyik partícióra akarjuk írni. A rendszerpartíció minimálisan ajánlott mérete 8 GByte. Helyezzük a MicroSD kártyát egy olvasóba, és dugjuk be a gép megfelelő aljzatába. A rendszerünk által elérhető összes háttértárat és azon levő partíciókat az „lsblk” parancs használatával tudjuk megjeleníteni. Az itt listázott partíciók nem feltétlenül vannak felcsatolva a rendszerhez:

```
lsblk
```

Egy példa a kimenetre:

```
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda           8:0    0 465.8G  0 disk
├─sda1        8:1    0   300M  0 part /boot/efi
├─sda6        8:6    0 346.8G  0 part /media/data
├─sda8        8:8    0   6.9G  0 part [SWAP]
sdb           8:16   0  22.4G  0 disk
├─sdb1        8:17   0  22.1G  0 part /
mmcblk0     179:0    0  29.7G  0 disk
└─mmcblk0p1 179:1    0  29.7G  0 part
```

Egy másik parancs, mellyel kilistázhatóak a csatolt partíciók mérete, azokon lévő üres hely, és csatolási pontja. Itt csak a felcsatolt partíciók láthatóak:

```
df -h
```

Kimenete:

```
Filesystem      Size  Used Avail Use% Mounted on
udev            2.7G  4.0K  2.7G   1% /dev
tmpfs           542M  1.4M  541M   1% /run
/dev/sdb1        19G   16G  1.6G  92% /
tmpfs           2.7G   28K  2.7G   1% /tmp
/dev/sda6       342G  318G  5.9G  99% /media/data
/dev/sda1       296M   48M  249M  17% /boot/efi
/dev/mmcblk0p1   30G  3.3G   27G  11% /media/felhasznalo/RASPFAT
```

Míg az „lsblk” paranccsal egyszerűbb átlátni a szerkezetét az elérhető partícióknak, addig a „df -h” parancs további információt nyújt, egyértelműbben be tudjuk beazonosítani a telepítésre kiválasztott partíció elérési útját.

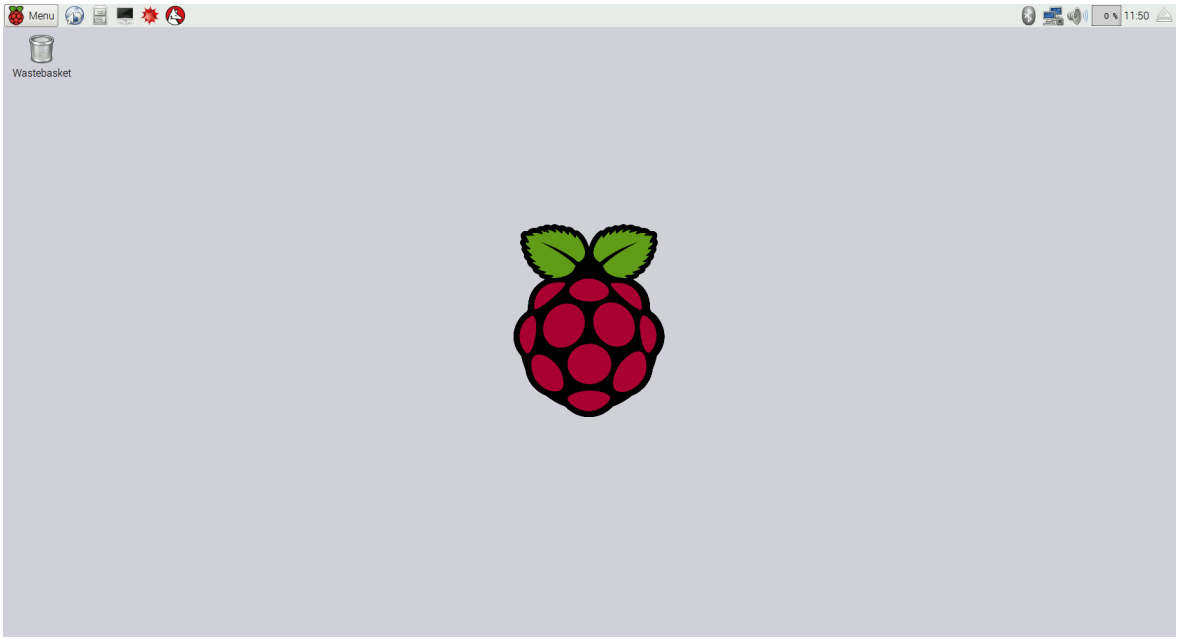
Az img képfájl tartalma az alábbi kód futtatásával másolható a MicroSD kártyánkra:

```
sudo dd bs=4M if=/eleresiut/2016-05-27-raspbian-jessie.img of=/dev/mmcblk0
```

A „sudo” előtag root jogot biztosít nekünk, a „bs=4M” kapcsoló a blokk méretet határozza meg, az „if=” az „input file” rövidítése, itt kell megadnunk elérési úttal az img képfájlt, az „of=” az „output file” rövidítése, jelen esetben a lemez azonosítót kell megadnunk, annak elérési útjával, ami a /dev könyvtár. Fontos megjegyezni, hogy a „dd” parancs az egész lemez tartalmát felül fogja írni, nincs lehetőség külön partíciót megadni neki. Vegyük észre, hogy a lemez azonosítója „mmcblk0” és az azon lévő partícióé pedig „mmcblk0p1”. Ez rendszerfüggő, lehetséges, hogy sdd illetve sdd1 néven jelennek meg. A másolás végeztével helyezzük be a

MicroSD kártyát a Raspberry-n található kártyaolvasóba, és a szükséges perifériák (Tápegység, HDMI monitor billentyűzet, egér, Ethernet kábel vagy Wi-Fi adapter) csatlakoztatása után indíthatjuk a rendszert a számítógép tápegységének konnektorba dugásával.

Az első indítás során a rendszer automatikusan bejelentkezik az előre beállított LXDE asztali környezetbe (3. Ábra)



3. Ábra: Első bootolás után

3.2.2 A „/etc/inputrc”, és a „~/.bashrc” fájlok beállítása

A következőkben viszonylag sokat fogjuk használni a terminált a telepített rendszerünkön, így néhány olyan beállítást mutatok be, amelyek megkönnyíthetik a parancssorban való eligazodást.

Mindenek előtt érdemes tisztázni, hogy a Linux alapú rendszereken a „~” hullámjel az aktuális felhasználó saját könyvtárát jelenti. A saját könyvtár a „`/home/felhasznalonev`” elérési útvonalon van.

Egy friss rendszer telepítése után első dolgom a „`/etc/inputrc`” fájl ellenőrzése, hogy tartalmazza-e a következőket, illetve engedélyezve vannak-e:

- A „Ctrl” - „kurzormozgató jobbra/balra nyíl” billentyűkombinációval a szavakhoz vagy kifejezésekhez való ugrás az aktuális sorban:

```
"\e[1;5C": forward-word
  "\e[1;5D": backward-word
  "\e[5C": forward-word
```

```
"\e[5D": backward-word
"\e\e[C": forward-word
"\e\e[D": backward-word
```

- Amennyiben elkezdünk írni karaktereket, a korábban kiadott és azzal kezdődő parancsok léptetése „le” és „fel” nyilakkal. Számomra ez sokkal triviálisabb és könnyebben használható mint a „Ctrl – R” billentyűkombináció, mely szintén a parancselőzmények visszakeresését segíti (melyek a „`~/.bash_history`” fájlban vannak eltárolva).

```
"\e[B": history-search-forward
"\e[A": history-search-backward
```

A Raspbian esetén az első megoldás alaphelyzetben engedélyezve volt, a második segédlet ugyan benne volt az „/etc/`inputrc`” fájlban, de kettős-kereszt volt előtte, tehát nem volt engedélyezve, de azokat törölve a következő bejelentkezésnél vagy ezen parancs kiadása után lehet is használni:

```
$ bind -f /etc/inputrc
```

Ha nem rendelkezünk „root” joggal egy adott rendszeren, akkor ezeket a beállításokat elhelyezhetjük a „home” könyvtárunkban is a „`~/.inputrc`” fájlban.

Ugyancsak érdemes pár szót ejteni a „`~/.bashrc`” fájlról. Itt a parancssor különböző viselkedéseit állíthatjuk, akár egy függvényt is definiálhatunk, ami egy parancsként működik, illetve „alias”-okat (egyéni neveket) adhatunk összetett parancsoknak is. Egy „alias” megadása:

```
alias ssh-x='ssh -c arcfour,blowfish-cbc -C -Y -X '
```

Ezután az „`ssh-x felhasznalonev@domain.nev`” parancsot kiadva az fentebb említett összes kapcsolót felhasználva fog indulni az „`ssh`”, így akár egy grafikus felülettel rendelkező programot is tudunk indítani a távoli gépről, mely a mi képernyőnkön fog megjelenni.

Továbbá álljon itt egy példa arra, hogy ne kelljen megjegyeznünk az összes tömörítő program kapcsolóját:

```
ex ()
{
  if [ -f $1 ] ; then
    case $1 in
      *.tar.bz2)   tar xjf $1   ;;
      *.tar.gz)    tar xzf $1   ;;
      *.bz2)       bunzip2 $1   ;;
```

```
      *.rar)       unrar x $1     ;;
      *.gz)        gunzip $1    ;;
      *.tar)       tar xf $1    ;;
      *.tbz2)      tar xjf $1   ;;
      *.tgz)       tar xzf $1   ;;
      *.zip)       unzip $1     ;;
      *.Z)         uncompress $1;;
      *.7z)        7z x $1      ;;
      *)           echo "'$1' cannot be extracted via ex()" ;;
    esac
  else
    echo "'$1' is not a valid file"
  fi
}
```

Ezt a néhány sort beillesztve a „~/.bashrc” fájlba, elég lesz ennyit írni a parancssorba ha ki akarunk tömöríteni egy fájlt helyben:

```
$ ex filenev.rar
```

Természetesen a „~/.bashrc” fájlban végrehajtott módosítások is a következő bejelentkezés után, vagy a következő parancs kiadása után lesznek aktívak:

```
$ source ~/.bashrc
```

3.2.3 Frissítés és szükséges programok telepítése

A Raspbian rendszer alapértelmezetten létrehoz egy „pi” felhasználót, melynek jelszava „raspberry”. Ezt célszerű megváltoztatni az első indítás után egy egyedi jelszóra a következő paranccsal:

```
$ passwd
```

Debian alapú rendszerek az „apt” csomagkezelőt használják. Megbízható csomagkezelő, megjelenésekor nagy előnye volt a függőségek nagyszerű kezelése, azonban napjainkban már léteznek gyorsabb és praktikusabb csomagkezlők is (pl. Az Archlinux csomagkezelője: a „pacman”). Első teendőnk, hogy frissítjük az elérhető csomagok listáját:

```
$ sudo apt update
```

Ez a parancs az „/etc/apt/sources.list” fájlban illetve a „/etc/apt/sources.list.d” könyvtár fájljaiban található címeket ellenőrzi, hogy található-e frissebb csomag a szerveren.

Mint azt korábban említettem, a „sudo” előtag biztosítja, hogy „root” jogokkal futtassunk egy parancsot. Amennyiben nem szeretnénk a „sudo” előtagot kiadni, és egymás után sorban „root” jogot igénylő teendőnk akad, akkor kiadhatjuk a „sudo -i” parancsot, és ezt követően „root” jogokkal hajthatunk végre parancsokat „sudo” előtag nélkül. Én a „sudo” használatát részesítem előnyben, mert így könnyen visszakereshetőek a kiadott parancsok a „~/.bash_history” állományból („sudo -i” használata esetén a „root-ként” kiadott parancsok a „/root/.bash_history” fájlban lesznek eltárolva).

Nézzük meg, hogy található-e frissítés a rendszerünkhöz:

```
sudo apt full-upgrade
```

Ha elérhetőek a jelenleginél frissebb csomagok, akkor a rendszer felkínálja azok telepítését, mindössze egy „Enter”-t kell nyomni.

Nagyszerű előny más típusú operációs rendszerekhez képest, hogy egyetlen sor beírásával tetszőleges számú programcsomagot tudunk telepíteni:

```
sudo apt install csomag1 csomag2 csomag3
```

Az „apt” függőség kezelésének köszönhetően, a szükséges további csomagokat automatikusan letölti és telepíti a rendszer. Telepítsük a szükséges programokat az alábbi paranccsal:

```
sudo apt install ssh rsync mc htop
```

Az „ssh” és az „rsync” sok Linux alapú operációs rendszerben alapból telepítve van, de biztos ami biztos, kiadhatjuk a parancsot, a már telepített csomagokat ki fogja listázni a rendszer, de nem telepíti őket újra, hacsak nem adjuk meg a „--reinstall” kapcsolót. Értelem szerűen az „ssh” az SSH kapcsolat létrehozásához és kezeléséhez biztosítja a parancsokat, az „rsync” csomag pedig a fájok szinkronizálásához fog segítséget nyújtani.

Az „mc” csomag a Midnight Commander programot rejti, klasszikus két paneles konzolos fájlkezelő, jól jöhet a rendszer konfigurálása során, valamint tartalmazza az „mcedit” nevű szövegszerkesztőt is, mellyel a kódkiemelésnek köszönhetően könnyebben átláthatjuk a szerkeszteni kívánt szkripteket/konfigurációs fájlokat.

A „htop” program egy konzolos rendszerfigyelő, a rendszerünk által használt erőforrásokat jeleníti meg, és a futtatott programokat listázza.

3.2.4 SSH kulcs létrehozása

A szerver, melyről a biztonsági mentést akarjuk végezni a felhasználó jelszavát fogja kérni parancsok futtatásához. Mivel a szkriptünknek automatikusan kell majd futnia a nap megadott időszakban, nem lesz módunk minden alkalommal megadni a jelszót. Ennek automatizálására létrehozhatunk egy SSH kulcsot, mellyel az adott helyi gépről jelszó beírása nélkül tudunk bejelentkezni.

Ehhez az első teendők, hogy létrehozunk egy SSH kulcsot a helyi gépen:

```
$ ssh-keygen
```

A parancs rá fog kérdezni, hogy hova akarjuk menteni a létrehozandó kulcsot. Alapból az aktuális felhasználói könyvtáron belül a „`.ssh`” könyvtárba kerül „`id_rsa`” néven. (Megj.: Linux rendszereken ha egy fájl illetve könyvtár neve előtt egy pont szerepel, az azt jelenti, hogy a fájl rejtett. Tipikusan a különböző felhasználók saját könyvtárában kerülnek tárolásra a telepített programok felhasználóra vonatkozó konfigurációs fájljai). Ezt követően megadhatunk egy jelmondatot, amellyel a kulcsunkat használni akarjuk, de ettől most eltekintünk, üresen hagyjuk.

Ha megtörtént a kulcs fájl létrehozása, akkor a helyi gépen a „`.ssh/id_rsa.pub`” fájlban található azonosítót kell átmásolnunk a szerveren található „`.shh/authorized_keys`” fájlba. Minden sor új azonosítót tartalmaz. Ezt a műveletet ellehet végezni manuálisan is, azonban ehhez a művelethez létezik egy parancs is:

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub felhasznalo@szerverdomain.nev
```

A másoláskor kérni fogja a szerver a jelszavunkat.

A következő bejelentkezést követően már jelszó nélkül be tudunk lépni a szerverre, illetve parancsokat tudunk rajta futtatni:

```
$ ssh felhasznalo@szerverdomain.nev
```

3.2.5 RAID 1 beállítása

A Raspberry Pi tervezése során elsődleges szempont volt a kis fogyasztás, emiatt viszont nem tud több külső merevlemezes háttértárat meghajtani USB-n keresztül. Célszerű tehát beruházni egy aktív USB elosztóba, mely külső táp segítségével orvosolja ezt a problémát. Pendrive-ok használata esetén nincs ilyen problémánk, megbízhatóan működik. Amennyiben a kiválasztott hátértárak mérete eltérő, a tömb létrehozásakor automatikusan a legkisebb méretével lesz

egyenlő a felhasználható terület.

Nézzük meg, hogy milyen néven tudunk hivatkozni a háttértárainkra. Nagy valószínűséggel „`/dev/sda`” és „`/dev/sdb`” lesz a két eszköznév:

```
$ sudo fdisk -l
```

Töröljük a háttértárainkainkat (opcionális):

```
$ sudo dd if=/dev/zero of=/dev/sda bs=512 count=1 conv=notrunc
```

Hozzunk létre „`msdos`” típusú partíciós táblát az eszközökön:

```
$ sudo parted /dev/sda mklabel msdos
```

Információ a háttértár állapotáról:

```
$ sudo parted /dev/sda unit s print free
```

Hozzunk létre egy partíciót, mely a háttértár teljes területét igénybe veszi:

```
$ sudo parted -a optimal /dev/sda mkpart primary 0% 100%
```

Célszerű elnevezni a partíciónkat:

```
$ sudo e2label /dev/sda1 hitachi500gb
```

Telepítsük a RAID tömb létrehozásához, és kezeléséhez szükséges programcsomagot:

```
$ sudo apt install mdadm
```

RAID 1 tömb létrehozása:

```
$ sudo mdadm -Cv /dev/md0 -l1 -n2 /dev/sd[ab]1
```

A tömb állapotáról információ megjelenítése:

```
$ sudo cat /proc/mdstat
```

RAID 1 tömb partíció fájltípusának megadása:

```
$ sudo mkfs /dev/md0 -t ext4
```

További információt kaphatunk a tömbről:

```
$ sudo mdadm --detail /dev/md0
```

Hozzuk létre a könyvtárat, ahova fel fogjuk csatolni:

```
$ sudo mkdir /media/data
```

A tömb felcsatolása:

```
$ sudo mount /dev/md0 /media/data
```

A csatolási pontot rendeljük hozzá az aktuális felhasználóhoz, és a „users” csoporthoz. Így ha több felhasználó lesz a későbbiekben, akkor elég lesz azokat hozzáadni a „users” csoporthoz az írási jogosultsághoz:

```
$ sudo chown $USER:users /media/data
```

Állítsuk be a jogosultságokat a létrehozott könyvtárnak, hogy a tulajdonos és a „users” csoportba tartozó felhasználók tudjanak olvasni, írni a könyvtárba, és legyen futtatási joguk:

```
$ sudo chmod 775 /media/data
```

Miután meggyőződtünk róla, hogy megfelelően működnek a fájlműveletek, véglegesítsük a beállításokat, hogy újraindítás után is elérhető legyen az előbb létrehozott tárterület. Illesszük be a következő sort az „/etc/fstab” fájlba, mely definiálja a bootoláskor csatolandó partíciók csatolási helyét, fájltípusát, csatolási tulajdonságait, és a fájlrendszer ellenőrzésének gyakoriságát:

```
$ sudo  nano /etc/fstab
```

```
/dev/md0      /media/data    ext4      defaults,noatime      0      0
```

3.3 Szkript leírás

Kétféle biztonsági mentést fogok részletesen bemutatni. A lényege mind a kettőnek, hogy legyen meghatározott napra visszamenőleg napi mentésünk, az utolsó egy évről heti mentésünk, és további éves mentések. A különbség a két megoldás között, hogy az egyik teljes mentést hajt végre, míg a másik inkrementális mentés lesz. Az első megoldást olyankor célszerű alkalmazni, amikor kevés állományból áll a menteni kívánt adat, illetve ha a mérete nem túl nagy az egyes mentéseknek. A mi esetünkben ez igaz a weboldalunk MYSQL adatbázisára, mindössze néhány MByte, és nem változik napi rendszerességgel sem. A másik példa egy inkrementális mentést fog végrehajtani, ez a cég honlapjának egyéb fájljait fogja szinkronizálni, valamint a helyi hálózaton a fontosabb állományok biztonsági mentése is ezzel a módszerrel lesz megoldva. A teljes mentés során új fájlok jönnek létre. Az inkrementális mentés esetén hardlink-ekkel

hivatkozunk a változatlan fájlokra. A hardlink-ek esetén pl. ha „a” fájlhoz társítunk egy „b” hardlink-et, akkor az „a” fájl törlése esetén a „b” elérhető marad, az adat nem törlődik a partíciós táblából, csupán ezután egyedül a „b” mutat majd rá. Softlink alkalmazásnál az „a” fájl törlése utána a „b” linken nem lesz elérhető az eredeti fájlunk.

A szkriptünk minden munkanap, az adott időben létrehoz egy a mentés dátumával és pontos idejével jelzett fájlt egy távoli szerveren, és szinkronizálja a helyi gépre is. Ez rövid ideig megoldhatja a biztonsági mentést, illetve bizonyos esetekben, mikor 2 mentés között sok idő telik el, akkor nincs szükség tovább bonyolítani a dolgokat. Ha azonban sok évig fut a szkriptünk, akkor célszerű rendszerezni a megkapott mentéseket, és a tárhelyigényt is figyelembe véve csak akkor mentsünk új példányát az állománynak, ha annak mérete eltér a korábbitól. A rendezés mértéke igény függő, én a következőek szerint csoportosítom a mentéseket. 3 mappába történik a mentés attól függően, hogy melyikben milyen gyakorisággal kerül újabb mentés. Ezek a mappák: „daily”, „weekly” és „yearly”, melyek rendre a napi, heti és éves mentéseket fogják tartalmazni. A készítendő script-ben a napi mentésnél fontos megjegyezni, hogy ha változik a fájl tartalma, akkor készül mentés függetlenül attól, hogy a mai nap már készült egy. Ennek az az előnye, hogy a script-et tetszőleges időben futtatva az aktuális állapotról ad egy biztonsági mentést.

Először a teljes mentést fogom ismertetni, és azt követően tárgyalom a lényegi különbségeket az inkrementális mentéshez képest.

A teljes mentés lépései címszavakban:

a) változók és konstansok deklarálása

b) sql fájl generálása mysqldump-al, fájlinformációk mentése

c) elérési utak ellenőrzése, szükség esetén létrehozása

d) fájlinformációkat kiíró függvények

e) letöltő függvény

f) felesleges régi mentéseket törlő függvény

g) összehasonlító függvény (napi, heti és éves mentésekhez)

h) összehasonlító függvény meghívása napi, heti és éves mentésekre

i) ideiglenes fájlok törlése

Inkrementális mentés felépítése:

a) változók és konstansok deklarálása

b) elérési utak ellenőrzése, szükség esetén létrehozása

c) fájlok szinkronizálása

d) másoló függvény

e) felesleges régi mentéseket törlő függvény

f) összehasonlító függvény (napi, heti és éves mentésekhez)

g) összehasonlító függvény meghívása napi, heti és éves mentésekre

3.3.1 Szövegszerkesztők

A script-et célszerű olyan fájlszerkesztőben írni, ami a különböző parancsok és környezetük szövegkiemelését elvégzi. Ilyen pl. grafikus felületen a GTK alapú Geany, a QT widget rendszerre alapuló „kate”, egy új fejlesztésű program a „CudaText”, melyhez elérhető Qt és GTK alapú változat is, illetve parancssorban az „mcedit”, mely általában az „mc” („Midnight Commander” - kétpaneles fájlkezelő) csomagban található Linux rendszereken. Nem csak tetszetősebb így a kód, sok hibát kiküszöbölhetünk segítségükkel. A legtöbb Linux alapú disztribúcióban a „nano” konzolos szövegszerkesztő telepítve van, azonban ennek a kódkiemelése nem minden esetben alapbeállítás. Ha távoli gépen levő kódot szeretnénk szerkeszteni, érdemes kipróbálni a „Krusader” kétpaneles QT alapú fájlkezelő beépített szerkesztőjét, melyet az „F4” billentyűvel lehet elérni a szerkesztendő szöveges állomány kijelölése közben. Ennek előnye, hogy az ssh-n keresztül csatlakozott rendszeren („`fish://`” előtagot használva az elérési útvonalban) a megnyitott fájlt elég menteni, és a változás azonnal megtörténik a távoli fájlrendszeren is. Erre másik mód, ha „`sshfs`” program segítségével felcsatoljuk a szerver fájlrendszerét egy megadott útvonalra. Ezt elmenthetjük a „`/etc/fstab`” állományba a RAID tömb kiépítésénél mutatott módon, és ezt követően gyakorlatilag úgy fog viselkedni rendszerinduláskor a távoli fájlrendszer, mintha a saját gépünkön lennének az adatok (megfelelő adatkapcsolati sebesség esetén). Meg kell említenünk továbbá a „`vi`” szövegszerkesztőt, amely a gyakorlottabb programozók kedvelt környezete, gyorsbillentyűk segítik a tempós munkát, hátránya azonban, hogy használatának elsajátítása türelmet és időt igényel.

3.3.2 SSH, script futtatás

Ha rendelkezünk felhasználói névvel és jelszóval az adott (web)szerverre, akkor SSH-n keresztül tudjuk elvégezni az adatbázisról a mentést következő paranccsal:

```
$ ssh felhasznalo@szerverdomain.nev "mysqldump -uadatbazisfelhasznalo
-padatbazisjelszo adatbazisnev > /eleresiut/filenev.sql"
```

A dupla idézőjelek között megadott parancs fog lefutni a szerveren az azonosítást követően, mely az SSH kulcs szerverre másolása után automatikusan megtörténik. A „`mysqldump`” parancs a megadott adatbázis teljes mentését listázza. Linuxon egy program szöveges kimenetét a „>” karakterrel tudjuk lementeni egy szöveges fájlba. Amennyiben újra lefuttatjuk a parancsot, a korábbi fájl felülírásra kerül. Ha a „>>” karakterekkel végezzük az átirányítást, akkor abban az esetben ha már létezik a célfájl, akkor a meghívott program kimenete hozzáadódik a szöveges fájlhoz.

Érdekesség, hogy a felhasználói nevet és a jelszót a „-u” és „-p” kapcsolók után szóköz nélkül kell megadni.

Amennyiben fenti parancsot rendszeres időközönként szeretnénk futtatni, célszerű egy script fájlban, eltárolni. Általánosságban a bash vagy egyéb shell script-eken értelmezett parancsokat futtató állományokat „.sh” kiterjesztéssel szokták ellátni. A kettős kereszttel kezdődő sor általánosságban megjegyzésnek minősül, de kivételt képez a fájl első sora, amiben annak a parancsértelmezőnek az elérési útját adjuk meg, amivel szeretnénk futtatni a script-ben használt algoritmusokat, parancsokat. Bash esetén ez így fog kinézni:

```
#!/bin/bash
```

Ez után egy új sorban következhet az előzőekben említett „`ssh`” parancs, és el is menthetjük a fájlunkat. Tanácsos, a szöveges fájlt egy üres sorral zárni - ez általánosságban igaz Linux környezetben.

A script fájl elmentése után, adhatunk a fájlnak futási jogot, pl. ha ütemezve akarjuk futtatni. Ennek megadása nem feltétlen szükséges, a parancssorban így is tudjuk futtatni a Bash szkriptet (amennyiben ilyen módon adjuk meg a használni kívánt parancsértelmezőt, akkor a legelső sorban megadott elérési úton található parancsértelmezőt figyelmen kívül hagyja a gép):

```
$ bash scriptnev.sh
```

Vagy ha futtathatóvá szeretnénk tenni az állomány akkor:

```
$ chmod +x scriptnev.sh
```

És futtatáshoz:

```
$ ./scriptnev.sh
```

Ebben az esetben a „pont” az előre per jel előtt arra a mappára hivatkozik, amelyikben jelenleg vagyunk. A „./” karakterek elhagyása esetén a rendszer a a PATH környezeti változóban eltárolt elérési útvonalakon fogja keresni a futtatandó állományt.

3.3.3 Rsync, cron

A korábban leírt ssh paranccsal generáltunk egy „sql” kiterjesztésű fájl-t, mely tartalmaz egy SQL adatbázis mentést. Ezt a fájlt a következő paranccsal a gépünkre tudjuk „szinkronizálni”:

```
$ rsync -HavXx
"felhasznalo@szerverdomain.nev:/eleresi/ut/adatbazis_backup_$datum.sql"
"/helyi/eleresi/ut" --progress
```

Az „rsync” parancs opcióit mint általában minden parancsnak az „`rsync --help`” futtatásával megtekinthetjük. Jelen esetben a következőket használtam:

- `H` – hard linkek megőrzése
- `a` – archív mód
- `v` – bővebb információ az előrehaladásról
- `X` – a kiterjesztett attribútumok megtartása
- `x` – fájlrendszerek közötti kompatibilitás
- `--progress` – aktuális állapot mutatása átvitel közben

Amennyiben bővebb információt akarunk megjeleníteni az egész másolás állapotáról, és nem csak az adott szinkronizálandó fájlról, akkor használhatjuk a „`--progress`” kapcsoló helyett a „`--info=progress2`” kapcsolót is.

Ha szeretnénk egy mentést végezni pl. minden munkanap, 15:30-kor, azt a „cron” beállításával érhetjük el. A „`crontab`” parancs kiadásával írjuk be az utolsó sorba:

```
30 15 * * 1-5 /eleresi/ut/scriptnev.sh
```

A cron konfigurációs fájl felépítése a következőképp alakul: az első elem a perc, a második az óra, harmadik a nap, majd a hónap és a hét napja. Ezt követően kell megadni a futtatni kívánt script elérési útját és nevét. Az időpontoknál ha konkrét értéket adunk, meg, akkor minden olyan előforduláskor lefut a szkriptünk. Pl. ha így kezdődik a sorunk: „`30 * * * *`” akkor minden óra

30 perckor, azaz óránként. A hét napját is számokkal jelöljük, intervallum megadásával lehet hivatkozni pl. az esetünkben használt munkanapokra. Ha pl. negyed óránként kívánjuk futtatni a parancsot, akkor azt így tehetjük meg, vesszőkkel elválasztva felsorolva az futtatni kívánt perceket:

```
0,15,30,45 * * * * /eleresi/ut/scriptnev.sh
```

Vagy rövidebben, megadva, hogy hány percenként fusson a script:

```
*/15 * * * * /eleresi/ut/scriptnev.sh
```

Ahhoz, hogy a parancs fusson, a fent említett módon futási jogot kell adni a szkriptnek.

3.3.4 Változók és konstansok megadása

Következő lépésként adjuk a szkriptünkhöz azt a sort, ami a „$datum” változó helyére beilleszti a ma napi dátumot:

```
#!/bin/bash
datum="$(date +'%Y%m%d_%H%M')"

ssh felhasznalo@szerverdomain.nev "mysqldump -uadatbazisfelhasznalo
  -padatbazisjelszo adatbazisnev > /eleresiut/adatbazis_backup_$datum.sql"
rsync -HavXx
  "felhasznalo@szerverdomain.nev:/eleresiut/adatbazis_backup_$datum.sql"
  "/helyi/eleresiut" --progress
```

A „date” parancs visszaadja a pontos időt és dátumot. A zárójelek közti részt futtatva megkapjuk a mai dátumot, és egy aláhúzást követően a jelenlegi időt. A „$datum” kifejezés visszaadja a „datum” változóban tárolt értéket. Használatára ügyelni kell, ha pl. így kívánnánk hivatkozni egy fájlra, melynek a nevének kezdetét egy változóból adjuk, akkor annak nem várt következménye lehet:

```
/eleresiut/$datum_adatbazis_backup.sql
```

Ez esetben egy „datum_adatbazis_backup” nevű változóra hivatkozunk. A helyzet egyértelműségének kedvéért kirakhatjuk a határoló kapcsos zárójeleket:

```
/eleresiut/${datum}_adatbazis_backup.sql
```

Ajánlott a script elején, vagy egy másik állományban (Sourcing fejezet, 38. oldal) elhelyezni azokat a konstansokat, melyek módosításra kerülnek ha szkriptünket más környezetben akarjuk felhasználni. Érdemes konstansnak deklarálni azokat a változókat, melyek a program futása során nem vesznek fel újabb értéket. Ezt a „readonly” előtaggal nyomatékosíthatjuk. Pl.:

```
#!/bin/bash
#  Szerver és azon levő adatbázis címének, felhasználó nevének és a szerveren
   lévő ideiglenes elérési út megadása
readonly SERVER_LOGIN="felhasznalo@szerverdomain.nev"
readonly SERVER_PORT="23"
readonly SERVER_TEMP_PATH="/home/felhasznalo/sqltmp/"
readonly DB_USER="root"
readonly DB_NAME="mysql"
readonly DB_PASS="MySqlP4ssWord"

#  Helyi idiglenes elérési út
readonly LOCAL_TEMP_PATH="/tmp"
readonly DATE_TODAY="$(date +'%Y%m%d_%H%M')"

ssh $SERVER_LOGIN -p $SERVER_PORT "mysqldump -u$DB_USER -p$DB_PASS $DB_NAME >
   ${SERVER_TEMP_PATH}${DB_USER}_backup_$DATE_TODAY.sql"
rsync -HavXx -e "ssh -p $SERVER_PORT" $SERVER_LOGIN:${SERVER_TEMP_PATH}$
   {DB_USER}_backup_$DATE_TODAY.sql ${LOCAL_TEMP_PATH} –info=progress2
new_backup_size=$(ssh $SERVER_LOGIN -p $SERVER_PORT "stat -c %s $
   {SERVER_TEMP_PATH}${DB_NAME}_backup_$DATE_TODAY.sql")
new_backup_md5=$(ssh $SERVER_LOGIN -p $SERVER_PORT "head -n -1 $
   {SERVER_TEMP_PATH}${DB_NAME}_backup_$DATE_TODAY.sql | md5sum" | awk
   '{ print $1 }')
```

A „`stat -c %s fájlnév`” paranccsal megkapjuk a hivatkozott fájl méretét byte-ban. Az md5 generálásnál fontos megjegyezni, hogy MYSQL adatbázis mentés esetén a generált „sql” kiterjesztésű fájlok utolsó sora mindig az adott mentés aktuális ideje. Emiatt két különböző időben változatlan adatbázisról készült mentés két különböző generált md5 hash-t fog jelenteni, így az összehasonlításuk nem lehetséges. Ennek kiküszöbölése érdekében az md5 hash-t az utolsó sor nélkül generálom a mentésekről. A „`head -n -1 fájlnév`” parancs kimenetét „`pipe`”-on keresztül („`|`” karakter) átadjuk az „`md5sum`” parancsnak, és az abból származó kimenetnek az első tagját mentjük el a változóba (önmagában futtatva az md5sum parancsot, megadja az argumentumként megkapott fájl md5 hash értékét, és utána a fájlnevet; utóbbit azért, mert több fájl is megadható egyszerre).

Az a könyvtár, ahol a futó script van, a „`dirname`” parancs segítségével határozható meg. Amennyiben a futó script nevére vagyunk kíváncsiak, akkor a „`basename`” parancsot kell alkalmaznunk. Mind a két eszközt a „Coreutils - GNU core utilities” tartalmazza.[7]

```
#  Az aktuális elérési út tárolása
readonly LOCAL_BACKUP_PATH="$( cd "$(dirname "$0")" ; pwd -P )"
```

3.3.5 Elérési utak ellenőrzése, szükség esetén létrehozása

A könyvtárak létrehozása a „`mkdir -p /elérésiút/könyvtárnév`” paranccsal történik. A „`-p`” kapcsoló gondoskodik arról, hogy ha nem létezne a könyvtár, akkor alkönyvtárakkal együtt létrehozza. Amennyiben a könyvtár létezik, érintetlenül hagyja azt.

```
#  Elérési utak biztosítása a mentések számára
mkdir -p "$LOCAL_BACKUP_PATH/daily"
mkdir -p "$LOCAL_BACKUP_PATH/weekly"
mkdir -p "$LOCAL_BACKUP_PATH/yearly"
```

3.3.6 Fájlinformációkat kiíró függvények

A függvények működése értelemszerű. A helyi fájlok vizsgálata során többször is meghívásra kerül a méret és md5 hash lekérdező parancs, ezért szükséges függvényként deklarálni őket. Az „`echo`” parancs „`-e`” kapcsolója biztosítja, hogy a konstansként deklarált színek megjelenítésre kerüljenek. Az „`${NC}`” jelölés a „`no color`” kifejezésre utal.

```
#  Függvény, mely kiírja a képernyőre a friss mentés méretét és md5 hash
  értékét
function get_new_backup_data {
    echo -e "A friss mentés (${DB_NAME}_backup_$DATE_TODAY.sql) mérete: $
  {BLUE}$new_backup_size ${NC}byte,"
    echo -e "md5 hash értéke: ${BLUE}$new_backup_md5 ${NC}"
}

#  Függvény, mely visszaadja az argumentumként megadott file méretét
function get_file_size {
    echo "$(stat -c %s $1 2>/dev/null )"
}

#  Függvény, mely visszaadja az argumentumként megadott file md5 hash értékét
function get_file_md5 {
    echo "$(head -n -1 $1 | md5sum | awk '{ print $1 }')"
}

#  Függvény, mely kiírja a képernyőre a legutolsó mentés méretét és md5 hash
  értékét
function get_old_backup_data {
    echo -e "\nAz utolsó helyi $1 mentés ($file_latest_local) mérete: ${BLUE}
  $(get_file_size $file_latest_local) ${NC}byte,"
    echo -e "md5 hash értéke: ${BLUE}$(get_file_md5 $file_latest_local) $
  {NC}"
```

```
}
```

3.3.7 Letöltő függvény

A „set -e” parancs biztosítja, hogy hibás „rsync” parancs futás esetén a script ne folytatódhasson. A letöltő függvény először egy ideiglenes könyvtárba szinkronizálja a biztonsági mentést. Azért választottam ezt a megoldást, mert a napi, heti, éves mentések közül, lehet hogy csak az egyik, vagy csak kettő fog lefutni, és így az rsync futására biztosan csak egyszer lesz alkalom, ezt követően másolás „cp” paranccsal kerül helyére a mentés.

```
#  Függvény, amely letölti a backupot az ideiglenes mappába
function download_backup {
    #  Képernyőre írjuk az új mentés adatait
    get_new_backup_data
    echo -e "${BLUE}Új backup mentése a $1 mappába${NC}"
    if [ ! "$(ls ${LOCAL_TEMP_PATH}/${DB_NAME}_backup_$DATE_TODAY.sql 2>
  /dev/null)" ]; then
        #  Ha az rsync parancs hibával áll le, akkor a script futtatása
  megállítva
        set -e
        #  Szinkronizálás a helyi ideiglenes könyvtárba
        rsync -HavXx -e "ssh -p $SERVER_PORT" $SERVER_LOGIN:$
  {SERVER_TEMP_PATH}${DB_NAME}_backup_$DATE_TODAY.sql ${LOCAL_TEMP_PATH}
  --info=progress2
    fi
    #  Az ideiglenes könyvtárból az aktuálisba másolás
    cp ${LOCAL_TEMP_PATH}/${DB_NAME}_backup_$DATE_TODAY.sql ./
    echo -e "${GREEN}Új backup mentve a $1 mappába.${NC}"
}
```

3.3.8 Felesleges régi mentéseket törlő függvény

A script elején definiáljuk a napi és heti mentésekre vonatkozó maximális tárolt biztonsági mentés számot.

```
#  A maximálisan tárolni kívánt napok és hetek száma
readonly DAYS_TO_STAY=7
readonly WEEKS_TO_STAY=8
```

A „remove_oldies” meghívás úgy történik, hogy argumentumnak megadjuk ezt a konstanst.

```
#  Függvény, mely törli $1 változóban definiált értéket meghaladó mentést
function remove_oldies {
        number_of_rows=$(wc -l < idorend)
        if [[ $number_of_rows -gt $1-1 ]]; then
```

```
            toremove=$(tail -n+7 idorend | head -n1)
            rm $toremove
            echo -e "${RED}Régi mentés törölve a daily mappából: $toremove${NC}"
      fi
}
```

3.3.9 Összehasonlító függvény (napi, heti és éves mentésekhez)

A script lényegi része. A függvény összehasonlítja az argumentumként megkapott mentés típusának megfelelő könyvtárban lévő, már meglévő legutolsó mentést a frissen elkészült mentéssel. Az összehasonlítás először fájlméret alapján történik. Amennyiben a fájlméret egyezik, akkor sor kerül az md5 generálásra, és összehasonlításra. Azért alkalmaztam két lépcsős összehasonlítást, mert a fájlméret nagy számítási igény nélkül vizsgálható, és amennyiben különbözik, már indulhat is a mentés, nem szükséges generálni így minden esetben md5 hash-t, amely hosszadalmas és számítási kapacitást igénylő feladat lehet, ha a tárolni kívánt adatbázisunk mérete megnőne.

A heti mentések esetén vizsgáljuk a már meglévő mentést, hogy hanyadik héten történt. Ezt a fájlnévből határozom meg. A fájlnév első része az adatbázis neve, ami változhat ha máshol alkalmazom a függvényt. Hogy ez ne jelentsen bonyodalmat, az évet, hónapot és napot a fájlnévből hátulról számolom. A „`${#változó}`” kifejezés megadja a hivatkozott változó karakterhosszát, ebből kivonom a hátulról számolt pozíciót, és kettősponttal elválasztva megadom, hogy hány karaktert akarok kivágni a fájlnévből. Miután rendre megvan az év, hónap, nap, kötőjellel elválasztva megadom a „`date`” parancsnak mint aktuális dátum, és „`%V`” karakterekkel jelzem, hogy az adott dátumhoz tartozó hét számát adja vissza.

```
#  Függvény, mely eldönti, hogy szükséges-e menteni, és amennyiben igen, végrehajtja a $1 értéken kapott könyvtárra
function check_backup {
    cd "$LOCAL_BACKUP_PATH/$1"
    #  Fetétel, mely vizsgálja, hogy létezik-e már mentés
    if [ ! "$(ls $LOCAL_BACKUP_PATH/$1/${DB_NAME}_backup_* 2> /dev/null)" ]; then
        echo -e "${BLUE}\nElső mentés a $1 mappába:${NC}"
        download_backup $1
    else
        #  Az "idorend" fájlba létrehoz egy listát csökkenő időrendbe a mentések fájlneveiből
        ls ${DB_NAME}_backup_* -1 | sort -r > idorend
```

```
      #  Eltárolja a legutolsó meglévő mentés nevét
      file_latest_local=$(tail -n+1 idorend | head -n1)
      #  Feltétel, mely először fájlméret alapján vizsgál, majd ha az egyezik, md5 hash alapján hasonlítja össze a legutolsó és a legújabb mentést
      if [[ "$new_backup_size" == "$(get_file_size $file_latest_local)" && "$new_backup_md5" == "$(get_file_md5 $file_latest_local)" ]]; then
          echo -e "${BLUE}\nNincs szükség új mentésre a $1 mappában (azonos md5 hash).${NC}"
      else
          case $1 in
              "daily")
                  #  Ha új mentés készül, akkor képernyőre írjuk a legutolsó mentés adatait
                  get_old_backup_data $1
                  download_backup $1
                  remove_oldies $DAYS_TO_STAY
              ;;
              "weekly")
                  # A ${#VALTOZO} kifejezés megadja a változó karakterhosszát, mert változó lehet az db név hossza, amivel a fájlnév kezdődik
                  year_latest=${file_latest_local:${#file_latest_local}-17:4}
                  month_latest=${file_latest_local:${#file_latest_local}-13:2}
                  day_latest=${file_latest_local:${#file_latest_local}-11:2}
                  week_latest=$(date --date="$year_latest-$month_latest-$day_latest" +"%V")
                  if [[ $(date +'%V') != $week_latest ]]; then
                      get_old_backup_data $1
                      download_backup $1
                      remove_oldies $WEEKS_TO_STAY
                  else
                      echo -e "${BLUE}\nNincs szükség új mentésre a $1 mappában (még nincs következő hét).${NC}"
                  fi
              ;;
              "yearly")
                  year_latest=${file_latest_local:${#file_latest_local}-17:4}
                  if [ "$(date +'%Y')" != "$year_latest" ]; then
                      get_old_backup_data $1
                      download_backup $1
```

```
                    else
                        echo -e "${BLUE}\nNincs szükség új mentésre a $1
  mappában (még nincs következő év).${NC}"
                    fi
                ;;
            esac
        fi
    fi
}
```

3.3.10 Összehasonlító függvény meghívása napi, heti és éves mentésekre

A „check_backup” függvény meghívása, argumentumnak megadjuk rendre a „daily”, „weekly” és „yearly” szavakat, melyek alapján fogja vizsgálni a függvény, hogy melyik mentést szükséges elvégezni.

```
#  Napi, heti, éves mentések ellenőrzése és végrehajtása
check_backup daily
check_backup weekly
check_backup yearly
```

3.3.11 Ideiglenes fájlok törlése

A script végére marad a helyi és a szerveren levő ideiglenes fájlok törlése:

```
#  A helyi idiglenes mentés törlése a lemezről (hibaüzenet
  figyelmenkívülhagyásával, ha nem készült mentés)
rm ${LOCAL_TEMP_PATH}/${DB_NAME}_backup_$DATE_TODAY.sql 2> /dev/null
#  A szerveren lévő idiglemes fájl törlése
ssh $SERVER_LOGIN -p $SERVER_PORT "rm ${SERVER_TEMP_PATH}$
  {DB_NAME}_backup_$DATE_TODAY.sql"
```

3.4 Inkrementális mentés

Az inkrementális mentés során a szinkronizáció előrébb kerül, és a korábbi állapothoz képest szinkronizált fájlok számát elmentjük egy változóba. A későbbiekben ez lesz az elsődleges szempont, hogy létrehozzunk-e új másolatot. A „--delete” kapcsoló hatására a helyi példányban törlésre kerülnek azok a fájlok, amik a szerveren lévő változatban eltávolításra kerültek idő közben. A „--stats” kapcsolóval generálunk egy riportot, melyből kinyerhetjük többek között, hogy hány fájl került letöltésre. Ez átirányítjuk egy szöveges állományba.

```
#  Szinkronizálás
```

```
rsync -HavXx --delete -e "ssh -p $SERVER_PORT" $SERVER_LOGIN:${SERVER_PATH} $
  {LOCAL_BACKUP_PATH}/alpha --info=progress2 --stats > $
  {LOCAL_BACKUP_PATH}/rsyncstats.txt

#  Szinkronizált fájlok számának tárolása

transferred_files=$(awk '/Number of regular files transferred/{print $NF}' $
  {LOCAL_BACKUP_PATH}/rsyncstats.txt)

echo -e "${BLUE}Szinkronizált fájlok száma: $transferred_files${NC}"
```

A másodlagos vizsgálati szempont a korábban mentett és a frissen szinkronizált mentés közötti különbség. Hasonló módon mint a teljes mentés során az md5 hash számolásnál, itt a „`diff -r könyvtár1 könyvtár2`” parancs futtatásának a számításigénye lényegesen megnőhet, ha nő a mérete a mentendő adatnak, ezért van szükség először vizsgálni egy kevésbe számítás igényes adatot, amiből kiderülhet, hogy a mentés elkezdődhet.

```
#  Feltétel, mely először a szinkronizált fájlok számát vizsgálja, majd a két
  könyvtár közötti különbséget

if [[ $transferred_files = 0 && ! $(diff -r $dir_latest_local $
  {LOCAL_BACKUP_PATH}/$BACKUP_DIR) ]]; then

   echo -e "${BLUE}\nNincs szükség új mentésre a $1 mappában.${NC}"
```

Mivel a szinkronizáció már a script futásának elején megtörténik, ezt követően már csak másolnunk kell a mentést, méghozzá a „`cp -al`” paranccsal, mely hardlink-eket hoz létre, így csak azok az újonnan szinkronizált fájlok fognak újabb tárterületet foglalni, és azok is csak egy helyen lesznek fizikailag.

```
  cp -al ${LOCAL_BACKUP_PATH}/$BACKUP_DIR/* $LOCAL_BACKUP_PATH/$1/$DATE_TODAY
```

A script végén elmaradt az ideiglenes fájlok törlése is. Egy aktuális másolat mindig elérhető marad az „alpha” könyvtárban, ehhez fog viszonyítani a következő szinkronizáláskor a script.

4. EREDMÉNYEK ÉS TOVÁBBI FEJLESZTÉSI LEHETŐSÉGEK

Szakdolgozatom eredményeképp megvalósításra került a munkahelyemen egy állandóan üzemelő, mégis kis fogyasztású számítógépes mentési rendszer. Az adatokat tetszőleges időben elérem, és újabb másolatokat hozhatok létre munka közben.

További fejlesztési lehetőségeket rejt magában a megírt bash script. Az alábbiakban említem a dolgozatomban nem vagy csak érintőlegesen tárgyalt részeket, melyek a későbbiekben támpontot nyújthatnak egy esetleges továbbfejlesztéshez.

4.1 Sourcing

Jelen esetben a script olvashatóságát növeli ha egyetlen állományt kell áttekinteni. De ez az előbb említett mód, mely szerint a futó környezettől függő konstansokat megadjuk a script fájl elején, nem minden esetben ajánlott, pl. ha jelszavakat/érzékeny adatokat is kell tárolni. Illetve abban az esetben, ha a script túl sok sorból áll, az olvashatóságot pont az segíti, ha szétdaraboljuk az állományt, és később a megfelelő helyen meghívjuk az adott részt.

Ezt kétféle képen tehetjük meg:

- A szükséges helyen futtatjuk a kívánt állományt. Ebben az esetben úgy kell elképzelnünk, hogy nyílik egy új shell, és azon belül lefut a kívánt script, a kimenetét pedig átadja a meghívott helyen. (pl. ha a meghívott script-ben könyvtárt váltunk, ez nem lesz hatással a meghívó script aktuális pozíciójára)
- A „`source`” (vagy pont és egy szóköz karakter, mely a POSIX szabvány) előszóval hivatkozunk az állományra, ebben az esetben úgy viselkedik a a meghívott script, mintha a meghívó része lenne. Ilyenkor a meghívott script-ben elvégzett változások hatással vannak a meghívó környezetére is. (pl. ha a meghívott script-ben könyvtárt váltunk, ugyanaz a könyvtár lesz az aktuális pozíciója a meghívó script-nek is a meghívás után)

4.2 Tömörítés

Miután megvan az adatbázist leíró fájl, tömöríthetjük másolás előtt így a tárolás helytakarékosabb lesz. A korábbiakban tárgyalt „`xz`” tömörítőt a következőképp használhatjuk az említett 2-es tömörítési mértékkel (a „`--threads=0`” opcióval tudjuk kihasználni ha a gépünk többmagos processzorral rendelkezik, enélkül a tömörítés csak egy szálon zajlik):

```
$ xz -2v --threads=0 tomoriteni-kivant-file.dat
```

Kitömörítéshez pedig:

```
$ unxz tomoritett-file.dat.xz
```

A végleges script-ben a tömörítéstől eltekintek, melynek oka, hogy a MYSQL adatbázis mentésekor a generált fájl utolsó sora tartalmazza a mentés idejét, így az md5 hash generálásakor figyelmen kívül hagyom az utolsó sort. Amennyiben tömörítve lenne az állomány akkor minden egyes ellenőrzéskor ki kellene tömöríteni, vagy másik opció lenne a fájl-ból lecsípni az utolsó sort, de mivel jelenleg az adatbázis fájlok nem foglalnak nagy méretet az én esetemen, így nem okoz gondot a tömörítés hiánya és így megmaradhat az „`mysqldump`” által generált fájl az eredeti formában.

4.3 Titkosítás

Titkosíthatjuk is a mentet fájljainkat, így ha netán rossz kezekbe kerülne, akkor se tudjanak egykönnyen hozzáférni a fájlban tárolt adatokhoz. Az rsync-nek köszönhetően viszonylag biztonságosan tudjuk adatainkat szinkronizálni, de ha kifejezetten érzékeny adatokat másolunk két távoli számítógép között, akkor nem árt az óvatosság. A titkosítás bevezetésekor érmes figyelembe venni, hogy amennyiben a titkosítandó dokumentum nagy méretet ér el, akkor a titkosítási folyamat elég számításigényes tud lenni. Mivel a végső script egy viszonylag gyenge hardveren fog futni, így a titkosítás a végleges script-be nem került bele, de annak módját az alábbiakban ismertetem:

A titkosítást az „`openssl`” parancs segítségével tudjuk elvégezni. Először hozzunk létre egy állományt, mely tartalmazza a titkosítás feloldásához szükséges kódot:

```
$ echo -n "M1ntaN4gyonT1tkosEsHosszuKulcs" > kulcs.txt
```

Ezt követően ezen kulcs felhasználásával tudjuk titkosítani az állományunkat a következő módon:

```
$ openssl enc -aes-256-cbc -pass file:kulcs.txt < titkositasra-varo-
allomany.dat > nagyon-titkos-adat.dat
```

Innentől kezdve a „`nagyon-titkos-adat.dat`” állományunkat fogjuk másolni távoli és helyi gépek között. Célszerű a „`kulcs.txt`” fájlt külön úton küldeni, pl. e-mailben, pendrive-on. További biztonságot nyújthat, hogyha a „`kulcs.txt`” állományt is titkosítjuk az „openssl” által létrehozott jelmondattal. Ehhez létre kell hoznunk egy kulcsot (ezt a két parancsot csak egyszer kell kiadni):

```
$ openssl genrsa -out key.pem 2048
```

```
$ openssl rsa -in key.pem -out key-public.pem -outform PEM -pubout
```

Elészült a „key.pem” és a „key-public.pem” állomány. A „kulcs.txt” állományunkat a publikus kulccsal tudjuk titkosítani:

```
$ openssl rsautl -encrypt -pubin -inkey key-public.pem < kulcs.txt >
enc.key.txt
```

Ha ezt a módot a választjuk, akkor 3 fájlt kell eljuttatni a cél gépre, ahol a visszafejtés megtörténhet: a „`nagyon-titkos-adat.dat`”, az „`enc.key.txt`” és a „`key.pem`”. Ahhoz, hogy megkapjuk a távoli gépen az eredeti állományt a következőket kell futtatni. Először visszafejtjük a titkosított jelmondatot a magán kulccsal:

```
$ openssl rsautl -decrypt -inkey key.pem < enc.key.txt > kulcs.txt
```

Majd a kulcs felhasználásával visszafejtjük a titkosított állományt:

```
$ openssl enc -aes-256-cbc -d -pass file:kulcs.txt < nagyon-titkos-adat.dat
> visszafejtett-allomany.dat
```

Kulcs fájl megadása helyett használhatjuk közvetlenül is a kulcsként használt jelmondatot:

```
$ openssl enc -aes-256-cbc -d -pass pass:M1ntaN4gyonT1tkosEsHosszuKulcs
 < nagyon-titkos-adat.dat > visszafejtett-allomany.dat
```

5. ÖSSZEFOGLALÁS

Az elkészült szakdolgozatom remélhetőleg segítheti más cégek, magánemberek adatainak tárolásának biztonságosabbá tételét, és egyúttal bevezetőt, kedvcsinálót jelenthet a Linux rendszerek és a Bash scripting rejtelmeibe.

A szakdolgozat odt és pdf formátumban az alábbi linken elérhető, és ugyanitt megtalálható a teljes és inkrementális mentést végző script is:

`https://github.com/zajnemzaj/szakdolgozat`

6. MELLÉKLETEK

6.1 1. sz. Melléklet - Teljes mentés

```
#!/bin/bash


#  Fájlnév: teljes-mentes.sh
#  Leírás: Ssh-n keresztül MYSQL adatbázis teljes mentését elvégző script,
  daily, weekly, yearly mappákba.
#  Szerző: Dezső János


#  Szerver és azon levő adatbázis címének, felhasználó nevének és a szerveren
  lévő ideiglenes elérési út megadása
readonly SERVER_LOGIN="user@domain.nev"
readonly SERVER_PORT="23"
readonly SERVER_TEMP_PATH="/home/user/sqltmp/"
readonly DB_USER="root"
readonly DB_NAME="mysql"
readonly DB_PASS="MySqlP4ssWord"


#  Helyi idiglenes elérési út
readonly LOCAL_TEMP_PATH="/tmp"


#  A maximálisan tárolni kívánt napok és hetek száma
readonly DAYS_TO_STAY=7
readonly WEEKS_TO_STAY=8


#  Az aktuális elérési út tárolása
readonly LOCAL_BACKUP_PATH="$( cd "$(dirname "$0")" ; pwd -P )"


#  Mai dátum és idő eltárolása
readonly DATE_TODAY="$(date +'%Y%m%d_%H%M')"


#  echo paranccsal használt színek megadása
readonly RED='\033[0;31m'
readonly GREEN='\033[0;32m'
readonly BLUE='\033[1;34m'
readonly NC='\033[0m' # No Color


#  Új mentés készítése a szerveren az adatbázisról, majd annak méretének
  változóba mentése
ssh $SERVER_LOGIN -p $SERVER_PORT "mkdir -p $SERVER_TEMP_PATH"
```

```
ssh $SERVER_LOGIN -p $SERVER_PORT "mysqldump -u$DB_USER -p$DB_PASS $DB_NAME >
  ${SERVER_TEMP_PATH}${DB_NAME}_backup_$DATE_TODAY.sql"
new_backup_size=$(ssh $SERVER_LOGIN -p $SERVER_PORT "stat -c %s $
  {SERVER_TEMP_PATH}${DB_NAME}_backup_$DATE_TODAY.sql")
new_backup_md5=$(ssh $SERVER_LOGIN -p $SERVER_PORT "head -n -1 $
  {SERVER_TEMP_PATH}${DB_NAME}_backup_$DATE_TODAY.sql | md5sum" | awk
  '{ print $1 }')


#  Elérési utak biztosítása a mentések számára
mkdir -p "$LOCAL_BACKUP_PATH/daily"
mkdir -p "$LOCAL_BACKUP_PATH/weekly"
mkdir -p "$LOCAL_BACKUP_PATH/yearly"


#  Függvény, mely kiírja a képernyőre a friss mentés méretét és md5 hash
  értékét
function get_new_backup_data {
    ##!!!!!!!! -e is needed to have color options.
    echo -e "A friss mentés (${DB_NAME}_backup_$DATE_TODAY.sql) mérete: $
  {BLUE}$new_backup_size ${NC}byte,"
    echo -e "md5 hash értéke: ${BLUE}$new_backup_md5 ${NC}"
}


#  Függvény, mely visszaadja az argumentumként megadott file méretét
function get_file_size {
    echo "$(stat -c %s $1 2>/dev/null )"
}


#  Függvény, mely visszaadja az argumentumként megadott file md5 hash értékét
function get_file_md5 {
    echo "$(head -n -1 $1 | md5sum | awk '{ print $1 }')"
}


#  Függvény, mely kiírja a képernyőre a legutolsó mentés méretét és md5 hash
  értékét
function get_old_backup_data {
    echo -e "\nAz utolsó helyi $1 mentés ($file_latest_local) mérete: ${BLUE}
  $(get_file_size $file_latest_local) ${NC}byte,"
    echo -e "md5 hash értéke: ${BLUE}$(get_file_md5 $file_latest_local) $
  {NC}"
}


#  Függvény, amely letölti a backupot az ideiglenes mappába
function download_backup {
```

```
    #  Képernyőre írjuk az új mentés adatait
    get_new_backup_data
    echo -e "${BLUE}Új backup mentése a $1 mappába${NC}"
    if [ ! "$(ls ${LOCAL_TEMP_PATH}/${DB_NAME}_backup_$DATE_TODAY.sql 2>
  /dev/null)" ]; then
        #  Ha az rsync parancs hibával áll le, akkor a script futtatása
  megállítva
        set -e
        #  Szinkronizálás a helyi ideiglenes könyvtárba
        rsync -HavXx -e "ssh -p $SERVER_PORT" $SERVER_LOGIN:$
  {SERVER_TEMP_PATH}${DB_NAME}_backup_$DATE_TODAY.sql ${LOCAL_TEMP_PATH}
  --info=progress2
    fi
    #  Az ideiglenes könyvtárból az aktuálisba másolás
    cp ${LOCAL_TEMP_PATH}/${DB_NAME}_backup_$DATE_TODAY.sql ./
    echo -e "${GREEN}Új backup mentve a $1 mappába.${NC}"
}


#  Függvény, mely törli $1 változóban definiált értéket meghaladó mentést
function remove_oldies {
        number_of_rows=$(wc -l < idorend)
        if [[ $number_of_rows -gt $1-1 ]]; then
            toremove=$(tail -n+7 idorend | head -n1)
            rm $toremove
            echo -e "${RED}Régi mentés törölve a daily mappából: $toremove$
  {NC}"
        fi
}


#  Függvény, mely eldönti, hogy szükséges-e menteni, és amennyiben igen,
  végrehajtja a $1 értéken kapott könyvtárra
function check_backup {
    cd "$LOCAL_BACKUP_PATH/$1"
    #  Fetétel, mely vizsgálja, hogy létezik-e már mentés
    if [ ! "$(ls $LOCAL_BACKUP_PATH/$1/${DB_NAME}_backup_* 2> /dev/null)" ];
  then
        echo -e "${BLUE}\nElső mentés a $1 mappába:${NC}"
        download_backup $1
    else
        #  Az "idorend" fájlba létrehoz egy listát csökkenő időrendbe a
  mentések fájlneveiből
        ls ${DB_NAME}_backup_* -1 | sort -r > idorend
        #  Eltárolja a legutolsó meglévő mentés nevét
```

```
      file_latest_local=$(tail -n+1 idorend | head -n1)

      #  Feltétel, mely először fájlméret alapján vizsgál, majd ha az
egyezik, md5 hash alapján hasonlítja össze a legutolsó és a legújabb
mentést

      if [[ "$new_backup_size" == "$(get_file_size $file_latest_local)" &&
"$new_backup_md5" == "$(get_file_md5 $file_latest_local)" ]]; then

          echo -e "${BLUE}\nNincs szükség új mentésre a $1 mappában (azonos
md5 hash).${NC}"

      else

          case $1 in

              "daily")

                  #  Ha új mentés készül, akkor képernyőre írjuk a
legutolsó mentés adatait

                  get_old_backup_data $1

                  download_backup $1

                  remove_oldies $DAYS_TO_STAY

              ;;

              "weekly")

                  # A ${#VALTOZO} kifejezés megadja a változó
karakterhosszát, azért így, mert változó lehet az db név hossza, amivel a
fájl kezdődik

                  year_latest=${file_latest_local:${#file_latest_local}-
17:4}

                  month_latest=${file_latest_local:${#file_latest_local}-
13:2}

                  day_latest=${file_latest_local:${#file_latest_local}-
11:2}

                  week_latest=$(date --date="$year_latest-$month_latest-
$day_latest" +"%V")

                  if [[ $(date +'%V') != $week_latest ]]; then

                      get_old_backup_data $1

                      download_backup $1

                      remove_oldies $WEEKS_TO_STAY

                  else

                      echo -e "${BLUE}\nNincs szükség új mentésre a $1
mappában (még nincs következő hét).${NC}"

                  fi

              ;;

              "yearly")

                  # A ${#VALTOZO} kifejezés megadja a változó
karakterhosszát, azért így, mert változó lehet az db név

                  year_latest=${file_latest_local:${#file_latest_local}-
17:4}

                  if [ "$(date +'%Y')" != "$year_latest" ]; then

                      get_old_backup_data $1
```

```
                        download_backup $1
                    else
                        echo -e "${BLUE}\nNincs szükség új mentésre a $1
  mappában (még nincs következő év).${NC}"
                    fi
                ;;
            esac
        fi
    fi
}


#  Napi, heti, éves mentések ellenőrzése és végrehajtása
check_backup daily
check_backup weekly
check_backup yearly


#  A helyi idiglenes mentés törlése a lemezről (hibaüzenet
  figyelmenkívülhagyásával, ha nem készült mentés)
rm ${LOCAL_TEMP_PATH}/${DB_NAME}_backup_$DATE_TODAY.sql 2> /dev/null
#  A szerveren lévő idiglemes fájl törlése
ssh $SERVER_LOGIN -p $SERVER_PORT "rm ${SERVER_TEMP_PATH}$
  {DB_NAME}_backup_$DATE_TODAY.sql"
```

6.2 2. sz. Melléklet - Inkrementális mentés

```
#!/bin/bash


#  Fájlnév: inkrementalis-mentes.sh
#  Leírás: Ssh-n keresztül weboldal inkrementális mentését elvégző script,
  daily, weekly, yearly mappákba.
#  Szerző: Dezső János


#  Szerver és azon levő adatbázis címének, felhasználó nevének és a szerveren
  lévő elérési út megadása
readonly SERVER_LOGIN="user@domain.nev"
readonly SERVER_PORT="23"
readonly SERVER_PATH="/var/www/domain.nev/www"
readonly BACKUP_DIR="alpha/$(basename $SERVER_PATH)"


#  Helyi elérési út
readonly LOCAL_BACKUP_PATH="$( cd "$(dirname "$0")" ; pwd -P )"


#  A maximálisan tárolni kívánt napok és hetek száma
```

```
readonly DAYS_TO_STAY=7
readonly WEEKS_TO_STAY=8

#  Mai dátum és idő eltárolása
readonly DATE_TODAY="$(date +'%Y%m%d_%H%M')"

#  echo paranccsal használt színek megadása
readonly RED='\033[0;31m'
readonly GREEN='\033[0;32m'
readonly BLUE='\033[1;34m'
readonly NC='\033[0m' # No Color

#  Elérési utak biztosítása a mentések számára
mkdir -p "$LOCAL_BACKUP_PATH/alpha"
mkdir -p "$LOCAL_BACKUP_PATH/daily"
mkdir -p "$LOCAL_BACKUP_PATH/weekly"
mkdir -p "$LOCAL_BACKUP_PATH/yearly"

#  Szinkronizálás
rsync -HavXx --delete -e "ssh -p $SERVER_PORT" $SERVER_LOGIN:${SERVER_PATH} $
  {LOCAL_BACKUP_PATH}/alpha --info=progress2 --stats > $
  {LOCAL_BACKUP_PATH}/rsyncstats.txt

#  Szinkronizált fájlok számának tárolása
transferred_files=$(awk '/Number of regular files transferred/{print $NF}' $
  {LOCAL_BACKUP_PATH}/rsyncstats.txt)
echo -e "${BLUE}Szinkronizált fájlok száma: $transferred_files${NC}"

#  Függvény, átmásolja a backupot az argumentumnak megkapott mappába
function copy_backup {
    echo -e "${BLUE}\nÚj backup mentése a $1 mappába${NC}"
    mkdir -p "$LOCAL_BACKUP_PATH/$1/$DATE_TODAY"
    cp -al ${LOCAL_BACKUP_PATH}/$BACKUP_DIR/* $LOCAL_BACKUP_PATH/
  $1/$DATE_TODAY
    echo -e "${GREEN}Új backup mentve a $1 mappába.${NC}"
}

#  Függvény, mely törli $1 változóban definiált értéket meghaladó mentést
function remove_oldies {
        number_of_rows=$(wc -l < ${LOCAL_BACKUP_PATH}/idorend$2)
        if [[ $number_of_rows -gt $1-1 ]]; then
            toremove=$(tail -n+$1 ${LOCAL_BACKUP_PATH}/idorend$2 | head -n1)
```

```
            rm -rf $toremove
            echo -e "${RED}Régi mentés törölve a daily mappából: $toremove${NC}"
        fi
}

#  Függvény, mely eldönti, hogy szükséges-e menteni, és amennyiben igen, végrehajtja a $1 értéken kapott könyvtárra
function check_backup {
    cd "$LOCAL_BACKUP_PATH/$1"
    #  Fetétel, mely vizsgálja, hogy létezik-e már mentés
    if [ ! "$(ls $LOCAL_BACKUP_PATH/$1/* 2> /dev/null)" ]; then
        echo -e "${BLUE}\nElső mentés a $1 mappába:${NC}"
        copy_backup $1
    else
        #  Az "idorend" fájlba létrehoz egy listát csökkenő időrendbe a mentések fájlneveiből
        ls $LOCAL_BACKUP_PATH/$1/ -1 | sort -r > $LOCAL_BACKUP_PATH/idorend$1
        #  Eltárolja a legutolsó meglévő mentés nevét
        dir_latest_local=$(tail -n+1 $LOCAL_BACKUP_PATH/idorend$1 | head -n1)
        #  Feltétel, mely először a szinkronizált fájlok számát vizsgálja, majd a két könyvtár közötti különbséget
        if [[ $transferred_files = 0 && ! $(diff -r $dir_latest_local ${LOCAL_BACKUP_PATH}/$BACKUP_DIR) ]]; then
            echo -e "${BLUE}\nNincs szükség új mentésre a $1 mappában.${NC}"
        else
            case $1 in
                "daily")
                    copy_backup $1
                    remove_oldies $DAYS_TO_STAY daily
                ;;
                "weekly")
                    # A ${#VALTOZO} kifejezés megadja a változó karakterhosszát, azért így, mert változó lehet az db név hossza, amivel a fájl kezdődik
                    year_latest=${dir_latest_local:${#dir_latest_local}-13:4}
                    month_latest=${dir_latest_local:${#dir_latest_local}-9:2}
                    day_latest=${dir_latest_local:${#dir_latest_local}-7:2}
                    week_latest=$(date --date="$year_latest-$month_latest-$day_latest" +"%V")
                    if [[ $(date +'%V') != $week_latest ]]; then
                        copy_backup $1
                        remove_oldies $WEEKS_TO_STAY weekly
```

```
                    else
                        echo -e "${BLUE}\nNincs szükség új mentésre a $1
  mappában (még nincs következő hét).${NC}"
                    fi
                ;;
                "yearly")
                    year_latest=${dir_latest_local:${#dir_latest_local}-13:4}
                    if [ "$(date +'%Y')" != "$year_latest" ]; then
                        copy_backup $1
                    else
                        echo -e "${BLUE}\nNincs szükség új mentésre a $1
  mappában (még nincs következő év).${NC}"
                    fi
                ;;
            esac
        fi
    fi
}

#  Napi, heti, éves mentések ellenőrzése és végrehajtása
check_backup daily
check_backup weekly
check_backup yearly
```

IRODALOMJEGYZÉK

1: The Linux Foundation, What is Linux? , The kernel, 2016, https://www.linux.com/what-is-linux

2: OpenBSD, OpenSSH 7.2p2, Introduction, 2016, http://www.openssh.com

3: Free Software Foundation, Inc., The GNU Bash Reference Manual 4.3, Introduction, 2014, https://www.gnu.org/software/bash/manual/bash.html#Introduction

4: Jarrod@RootUsers, Gzip vs Bzip2 vs XZ Performance Comparison 1, 1, 2015, https://www.rootusers.com/gzip-vs-bzip2-vs-xz-performance-comparison/

5: Dan Douglas, BashFAQ 1, What is the difference between test, [and [[?, 2016, http://mywiki.wooledge.org/BashFAQ/031

6: Scott Chacon, Pro Git 2nd, , 2014, https://book.git-scm.com/doc

7: Free Software Foundation, Inc., Coreutils - GNU core utilities , , 2016, https://www.gnu.org/software/coreutils/coreutils.html