10. Ismétlő feladatok a 10. gyakorlattól kezdve

1. Feladat

(**findw.sh**) Írjuk meg a következő paraméterekkel indítható shell programot:

```
./findw.sh [ -n N ] [ -r R ] dir szo
```

`dir` : könyvtárnév, ha nincs megadva akkor a pillanatnyi munkakönyvtár (ennek neve a `.`),
`szo` : egy sztring,
`-n` opció paramétere N egy egész szám, ha nincs megadva akkor értéke 1,
`-r` opció paramétere R egy egész szám, ha nincs megadva akkor értéke 1.

A szkript kilistázza azoknak a .txt típusú szöveges állományoknak a nevét a `dir` könyvtárból amelyekben a paraméterként megadott szó az első R sorban legalább N-szer fordul elő teljes szóként (és nem szó töredékként). A nevek után kiírja azt is, hányszor fordul elő a szó (tehát egy 2 oszlopos listát ír ki):

```
fájlnév1 szám1
fájlnév2 szám2
```

Azt, hogy végigjáráskor egy állomány valóban szöveges-e ellenőrizni kell a `file` paranccsal és reguláris kifejezéssel.

Az opció kezelést végezzük el a 8. előadás `case3.sh` példa mintájára.
Az opciók végigjárása közben, amikor megkapjuk a paramétereket ***vagy*** miután végigjártuk a parancssori paramétereket, és megfelelő változókba kerültek, ellenőrizni kell:

- `dir` könyvtár létezik-e?
- `N` egész szám?
- `R` egész szám?
- a `szo` nem üres sztring.

Rossz paraméter esetén ki kell lépni hibaüzenettel.

Mivel a laboron oldottunk feladatot a case3.sh típusú opció kezeléssel, alternatív meg lehet tanulni használni a shell beépített opció kezelési parancsát, ennek neve: `getopts` , példaprogram a 8. előadás `getopts.sh` szkriptje, tankönyvben a leírás itt van (B. függelék).

2. Feladat

(**watch.sh**) Írjunk egy szkriptet amelyik a parancssorán megadott könyvtárban levő .txt típusú állományokat figyeli. Az alábbi parancssorral indítjuk:

```
watch.sh  [ -k N ] dir
```

Implicit érték N=2, `dir`-t kötelező megadni. A szkriptnek ellenőriznie kell, hogy az N egész szám, illetve a `dir` könyvtár létezik.

A szkript N szekundumonként megvizsgálja a könyvtárat (megpróbálja végigjárni a könyvtár .txt fájljait), és ha .txt típusú fájlok jelentek meg benne, akkor végigjárja ezeket. A fájlokból kiveszi az egész számokat, és egy `szamok.txt` nevű fájl végére írja, utána elköltözteti a feldolgozott `.txt` fájlt a **`dir`** fölötti könyvtárban található **`files_ok`** könyvtárba.

A `szamok.txt` tartalmát üresre állítja induláskor:

```
: > szamok.txt
```

A `szamok.txt` helye a **`dir`** fölötti könyvtár.

Teszteléshez, miközben a szkript fut másoljunk a könyvtárba olyan fájlokat amelyek kis szöveget és egész számokat tartalmaznak, pl. a következő tartalommal:

```
abc 12 def
3 xyz 111
219
```

A program végtelen ciklusban fusson, és csak akkor álljon le, ha a könyvtárban megjelenik a `vege.txt` nevű fájl ***vagy*** ha leállítjuk Ctrl-C jelzéssel. Induláskor a szkript letörli a `vege.txt` fájlt ha az létezik.

A fájl legelejére írjuk be:

```
trap "echo Vége; exit" INT
```

Ez a parancs (`trap`) jelzés kezelőt állít be, így elkapja a Ctrl-C jelzést és ilyenkor végrehajtja az `echo Vége; exit`
parancsokat, az `exit`-re kilép.

3. Feladat

(**`collect.sh`**) Írjunk egy szkriptet, amely bejárja az argumentum listájában megadott könyvtárakat:

`./collect.sh  dir [dir1]...`

és a következőt végzi:

Legalább egy könyvtár kell létezzen a parancssoron, ha nincs egy sem akkor hibával kilép. A parancssoron több könyvtárat adhatunk meg, ezek számát nem kell ellenőrizni.

A bejárás során az alábbiakat végzi:
-végigjárja a könyvtárakban levő szöveges állományokat (.txt állománytípus), de ellenőrzi azt is, hogy valóban szöveg van bennük,
-a fájlokból kiveszi az alábbi adatokat, és három különböző fájlba gyűjti, minden adatot egyenként új sorba:

1. romániai autó rendszámtábla, ezek a `rendszam.txt`-be kerülnek,
2. telefonszám 0999-999-999 vagy 0999999999 alakban; a telefonszámok a `tel.txt`-be kerülnek,
4. valós szám, az alakot lásd alább, ezek a `valos.txt`-be kerülnek.

Ha olyan sort talál, amelyikben egyik sincs, akkor ezeket a `szemet.txt`-be írja.

A valós számok a szövegekben a következő alakokban jelenhetnek meg (tekintsük úgy, mintha egy C programban lennének valós típusú változóhoz rendelt számok):
12
-7
12.56
.34
0
34.
Ennek az sztringnek minden felépítő része opcionális. Ha olyan kifejezést írunk, ahol opcionális az előjel, egész rész, a pont és a tört rész is, az mindenre fog illeszkedni (az opcionális kifejezés mindenben megvan). Ezért írjunk egy olyan kifejezést ami legalább két ***vagy*** ágból áll, és mindkét ***vagy*** ágban van egy kötelező rész (pl. egyikben kötelező az egészeket megadni, a másokban a törteket).

Megoldhatjuk a feladatot csak az egész számok keresésével is, ha nem sikerül a valósra reguláris kifejezést írni.

A feladathoz saját teszt adatokat kell készíteni.

4. Feladat

(**`checkmd5.sh`**) Írjunk egy szkriptet, amelyik végtelen ideig fut, és adott időközönként ellenőrzi egy könyvtár fájljait: megnézi, hogy változott-e a tartalmuk. Azoknak a fájloknak a listáját, amelyek tartalma megváltozott egy napló fájl végére írja a pillanatnyi időbélyeggel együtt.

Az ellenőrzéshez használjuk az **`md5sum`** parancsot. Ez MD5 hash kódot számol ki a fájl tartalmából: ha a fájl tartalma megváltozik, és újraszámoljuk a kódot, ez más értékű lesz. Az angol Wikipedia szócikke részletesen bemutatja az algoritmust: https://en.wikipedia.org/wiki/MD5 .
Az MD5 128 bites hash kód, ez 32 számjeggyel adhatjuk meg (16-os számrendszer használatos) megjeleníthető olvasható formában.

Két azonos tartalmú fájlnak az MD5 hash kódja ugyanaz, így a kódot eltárolva, ellenőrizni lehet, hogy egy fájl tartalma megváltozott-e időben.

Használat:

```
$ md5sum yes/wurm.txt
fe74f086d98afac87d8fabd2596b0716  yes/wurm.txt
$ echo a >> yes/wurm.txt
$ # a megváltozott fájlnak más a hash kódja:
$ md5sum yes/wurm.txt
d70f1e7a1034a8a465c6b7bf04b32da6  yes/wurm.txt
$
```

Több fájlra meghívva:

```
$ md5sum yes/*.txt
eda14df7b6260e2bc8e89d91fe779e25  yes/longdistance.txt
8389165b3e3c48a0b40c71c5263f60c3  yes/solidtime.txt
d4b80c03daf1dd35e41124d6ee1c6c64  yes/survival.txt
57d4ce2191592b99a04f72adb4e1813f  yes/timeanda.txt
d70f1e7a1034a8a465c6b7bf04b32da6  yes/wurm.txt
$
```

A kimenetet el lehet menteni:

```
$ md5sum yes/*.txt > yes.md5
$
```

és fel lehet használni a fájlok későbbi integritásának ellenőrzésére (`-c` opció, *check*):

```
$ md5sum -c yes.md5
yes/longdistance.txt: RENDBEN
yes/solidtime.txt: RENDBEN
yes/survival.txt: RENDBEN
yes/timeanda.txt: RENDBEN
yes/wurm.txt: RENDBEN
$ echo a >> yes/survival.txt
$ md5sum -c yes.md5
yes/longdistance.txt: RENDBEN
yes/solidtime.txt: RENDBEN
yes/survival.txt: HIBÁS
yes/timeanda.txt: RENDBEN
yes/wurm.txt: RENDBEN
md5sum: FIGYELMEZTETÉS: 1 számított ellenőrzőösszeg NEM egyezett
$
```

Mivel a hash kódokat gyakran az integritás ellenőrzésére használják, az md5sum a módosult fájlt hibásnak jelzi, ez csak azt jelzi, hogy most már más a hash kódja.
A `FIGYELMEZTETÉS`-t tartalmazó sor a standard hiba kimenetre megy:

```
$ md5sum -c yes.md5 2>/dev/null
yes/longdistance.txt: RENDBEN
yes/solidtime.txt: RENDBEN
yes/survival.txt: HIBÁS
yes/timeanda.txt: RENDBEN
yes/wurm.txt: RENDBEN
$
```

A `--quiet` opcióval nem írja ki a rendben levő fájlokat, a `--ignore-missing` opcióval nem írja ki az esetleg hiányzó (pl. letörölt) fájlok nevét, így megkapjuk a módosult fájlokat:

```
$ md5sum -c yes.md5 --quiet --ignore-missing 2>/dev/null
yes/survival.txt: HIBÁS
$ echo $?
1
$
```

Ha csak a fájl neve szükséges a feladat megoldásához, azt a kimenetből a `sed`-el vehetjük ki:

```
$ echo 'yes/survival.txt: HIBÁS' | sed -r 's/^.*\/([^:]+).*$/\1/'
survival.txt
$
```

A `sed` reguláris kifejezésében a **`/`** karaktert a **`\/`** sorozattal keressük, mivel a **`/`** az **`s`** parancs elválasztójaként is használandó. A szűrés akkor is működik, ha a fájl nem az első szintű könyvtárban van, hanem mélyebben.
Ugyanazt elvégezhetjük az `egrep`-el is:

```
$ echo 'yes/survival.txt: HIBÁS' | egrep '[^/]+:' | cut -f1 -d:
survival.txt
$
```

Ha minden rendben, a szkript igazat térít vissza a **?** változóba, ha van megváltozott fájl, akkor hamisat:

```
$ md5sum -c yes.md5 --quiet --ignore-missing 2>/dev/null
yes/survival.txt: HIBÁS
$ echo $?
1
$ # újraszámolás:
$ md5sum yes/* > yes.md5
$ md5sum -c yes.md5 --quiet --ignore-missing 2>/dev/null
$ echo $?
0
$
```

A megoldandó feladat: az alábbi parancssorral indítva:

```
$ ./checkmd5.sh -t 3 yes
```

a `checkmd5.sh` szkript a következőket végezze:
A `-t` egy opció, amelynek paramétere egy maximum 3 jegyű egész szám. A paraméter alapértelmezett értéke 1, ha nem adjuk meg (legyen ez T változó a szkriptben). Az opció és paraméter után a megfigyelendő könyvtár neve következik, ezt kötelező megadni. Így is indíthatjuk:

```
$ ./checkmd5.sh yes
```

A szkript kiszámolja a könyvtárban levő .txt végződésű állományok MD5 hash kódjait, utána vár T másodpercet, majd ellenőrzi, hogy változott-e meg fájl. Ha módosult a fájl, pl. a `wurm.txt`, a `checkmd5.log` napló fájl végére írja a következő alakú sort:

```
2024-11-12 18:38:03 módosult: wurm.txt
```

és a következő alakot ha több fájl módosult:

```
2024-11-12 18:37:56 módosult: survival.txt timeanda.txt
```

A dátumot és időt állítsuk elő a **`date`** paranccsal:

```
$ date '+%F %T'
2024-11-23 12:43:38
$
```

A `checkmd5.log` napló helye a abban a könyvtárban van ahonnan a szkriptet indítjuk. A fájlt a szkript minden indulásakor nullázza, és az első sorába írja:

```
2024-11-12 19:30:07 start
```

Ha nincs változás nem ír ki semmit. Újraszámolja az MD5 kódokat, majd vár T szekundumig. Így, ha új fájl jelenik meg, a következő ciklusban már ellenőrizheti.
Ezt a folyamatot ismétli végtelen ideig. A `TERM`, `INT`, `QUIT` jelzésekre kilép, és a napló végére írja:

```
2024-11-12 19:30:08 stop, futási idő: 2 szekundum
```

Így ha nem történik a fájlokban változás, kilépés után a napló így néz ki:

```
2024-11-12 19:30:07 start
2024-11-12 19:30:08 stop, futási idő: 2 szekundum
```

Példa futtatás:

```
$ # checkmd5.sh háttérben fut:
$ ./checkmd5.sh -t 2 yes &
[1] 4731
$ #várunk 2 másodpercet
$ sleep 2
$ #  yes/wurm.txt módosul
$ echo a >> yes/wurm.txt
$ #várunk 3 másodpercet
$ sleep 3
$ # két fájl módosul
$ echo a >> yes/wurm.txt; echo a >> yes/solidtime.txt
$ #várunk 2 másodpercet
$ sleep 2
$ # visszahozzuk előtérbe a szkriptet
$ fg
./checkmd5.sh -t 2 yes
$ # Ctrl-C-t kap, kilép
^C$
```

A napló tartalma (a parancsok beírása is elvesz valamennyi időt):

```
2024-11-12 19:38:21 start
2024-11-12 19:38:39 módosult: wurm.txt
2024-11-12 19:39:26 módosult: solidtime.txt wurm.txt
2024-11-12 19:39:42 stop, futási idő: 21 szekundum
```

A naplót létrehozhatjuk úgy is, hogy elindítjuk a a szkriptet, és a `gedit`-el módosítgatjuk a fájlokat.

Hibaüzenetek induláskor:

```
$ ./checkmd5.sh
használat: bash checkmd5.sh -t T dir
$ ./checkmd5.sh -t a yes
a -t paramétere: a nem maximum háromjegyű egész szám
$ ./checkmd5.sh -t 3 yess
A yess könyvtár nem elérhető
$ mkdir ures
$ ./checkmd5.sh -t 3 ures
ures üres könyvtár
$
```

5. Feladat

A feladathoz szükségesek a következő parancsok: **`tar`** és **`date`**.

A `tar` parancs használatáról a Moodle 4. szakaszában lehet olvasni, *A tar parancs használata* dokumentumban.

A `date` parancs használatáról a 13. szakaszban lehet olvasni, a Néhány időhöz kötött parancs használata. A date parancs c. segédanyagban.

a. Feladat (`backup.sh`): írjunk egy háttér mentés (*backup*) készítő szkriptet az alábbi módon: a szkript egy könyvtárnevet kap paraméterként, előtte megadhatunk egy **`-i`** opciót, indítás:

```
$ ./backup.sh adatok
```

vagy

```
$ ./backup.sh -i adatok
```

Ha nincs `-i` opció, akkor egy tgz típusú mentést kell végezni a megadott könyvtár alatti könyvtárakról, pl. az alábbi esetben:

```
adatok$ tree
.
├── elso
└── masodik

2 directories, 0 files
adatok$
```

az `elso` és `masodik` könyvtárról, a mentett fájl nevében meg kell jelennie a dátumnak, pl. az `elso` könyvtár esetében:
`elso-2024-11-13.tgz`
ez a fájl a saját könyvtár alatti **`BACKUP`** könyvtárba kerül, ha ez nem létezik automatikusan létre kell hozni. Ha van ott ilyen nevű fájl (ugyanazon a napon másodszor futtatjuk), felül kell írni, de ki kell írni a terminálra, hogy felülírás történt.

Ha van `-i` opció, indulás után egy kiválasztható listában ki kell írnia az adott könyvtár alatti könyvtárak listáját, pl. ha adatok alatt van `elso` és `masodik` könyvtár, akkor ezeket a neveket (egy `select` szerkezettel):

```
Válassza ki a könyvtárat háttér mentésre:
1) elso
2) masodik
>
```

A felhasználó kiválaszthat ezek közül egy könyvtárat, pl. az 1. menüpont kiválasztásával, és ilyenkor csak ezt a könyvtárat kell elmenteni (`elso`), ugyanolyan névvel mint a nem interaktív esetben.

Interaktív mentésnél, ha van ugyanolyan nevű fájl a `BACKUP` könyvtárban, akkor meg kell kérdeznie, hogy írja-e felül, és ezt csak akkor tegye meg ha a felhasználó jóváhagyja (kérdés, igenlő válasz a kis vagy nagy i karakter). A program addig ismételje ezt a könyvtár mentés műveletet, amíg fájl vége karaktert (Ctrl-D) kap a bemenetről a kiírt könyvtár lista kiválasztásánál.

Ha a program `TERM` vagy `INT` jelzést kap, lépjen ki, de előtte írja ki a `BACKUP` könyvtárban található időben legfrissebb fájl nevét (amit legutoljára készített el). `USR1` jelzésre írja ki, hány

milliszekundumot futott, és ez alatt hány új tgz fájlt készített.

A szkript kezeljen hibákat, és bármilyen hiba esetén saját üzenettel lépjen ki.

Teszt könyvtár készítése:

```
$ mkdir adatok
$ cp -r yes adatok/elso
$ cp -r yes adatok/masodik
$ cp -r yes adatok/harmadik
$ tree -L 1 adatok/
adatok/
├── elso
├── harmadik
└── masodik
3 directories, 0 files
$
```

Feltételezzük, hogy az adatok könyvtárban csak könyvtárak vannak.

b. Feladat (`backupz.sh`):
Módosítsuk az a. feladatot úgy, hogy fogadjon el egy **`-z`** opciót is a parancssoron. Ha ezt megadjuk, akkor a **`tar`** helyett a **`zip`** nevű archiváló programot használja, és .zip típusú mentéseket készítsen, a mentett fájlok neve `elso-2023-12-08.zip` alakú legyen. A **`zip`** man oldalát meg lehet nézni a rendszerben `man zip` paranccsal, az angol man lap esetén a példák a *„Command format.”* címnél vannak. Legegyszerűbb használati formája:

```
$ zip -r elso.zip elso
$ ls elso.zip
elso.zip
$
```

6. Feladat

Írjunk egy szkriptet (**`doer.sh`**) , amely bejárja az argumentum listájában megadott könyvtárban levő fájlokat:

```
./doer.sh [ -m MEGA ] dir
```

Példa indítások:

```
./doer.sh kupac
./doer.sh  -m 2 kupac
```

A szkriptnek egy opciója van, a **`-m`** amivel egy fájl méretet adunk meg megabyte-ban. Ebben a feladatban 1 MB alatt 1048576 byte-ot értünk. A MEGA alapértelmezett értéke 2, legkisebb értéke 1.

A **`kupac`** teszt könyvtár letölthető a Moodle oldal 10. szakaszából (`kupac.tgz` csomag). Dolgozzunk egy **`fel6`** könyvtárban, és másoljuk alája a kicsomagolt `kupac` tartalmát.

A könyvtárban .txt és .jpg végződésű fájlok vannak, de egyes fájlok nem a kiterjesztésnek

megfelelő adatokat tartalmaznak.

Bejárás során a szkript a .txt fájlok esetében ellenőrzi, hogy a fájl tartalma valóban szöveges. Ha igen, akkor kikeresi a fájlból azokat az valós számokat, amelyek olyan sorokban vannak amelyek egy * karakterrel kezdődnek:

pl.:
```
*alma 33.66 szilva
*cseresznye 55.3
*12
*  12. alma
```

A valós számoknak mindig ki van írva az egész része, a pont és tört rész opcionális (pl. 5 5. 5.24). Ha egy sor első karaktere nem *, abból nem veszi ki a valós számokat.

Ezeket egy `fel6/szamok.txt` nevű fájlba írja, a következő alakban:

```
33.66
55.3
12
12.
```

A .jpg fájlok esetében ellenőrzi, hogy valóban JPEG képek, és ha igen, akkor:

- Amennyiben a kép kevesebb helyet foglal a lemezen mint MEGA megabyte, akkor átalakítja GIF (.gif) képpé, és ezeket a `fel6/gif` könyvtárban hozza létre,
- amennyiben több helyet foglal mint MEGA megabyte, akkor PNG (.png) képpé alakítja, és ezeket a `SCRIPT/png` könyvtárban hozza létre.

Ha a .jpg végződésű fájl nem JPEG kép, akkor a `fel6/szemet` könyvtárba viszi

A `png, gif` és `szemet` könyvtárakat létre kell hozni induláskor ha nem léteznek.

Egy fájl méretét a **`du`** paranccsal kapjuk meg (rövid leírás a 8. labor 8. feladatban). A **`-B`** opció után megadhatjuk a „blokk” méretét, amiben visszaadja a méretet. Pl. a méretet kérhetjük megabyte-ban, kétféleképpen is:

```
$ du -BM mountain.jpg
2M    mountain.jpg
$ du -B1048576 mountain.jpg
2     mountain.jpg
$
```

A méret és a fájl neve közt egy TAB van, tehát ha csak a méret kell, akkor:

```
$ du -B1048576 mountain.jpg | cut -f1
2
$
```

Figyelem, a `du` felkerekíti a méretet, a `mountain.jpg` tulajdonképpen 1,5 MB.

```
$ ls -lh mountain.jpg
-rw-r--r-- 1 lszabo lszabo 1.5M jan    9  2023 mountain.jpg
```

```
$
```

Tehát amennyiben 2-es méretet kapunk, azt jelenti, hogy a fájl „befér” 2 MB-ba (kisebb vagy egyenlő ezzel a 2MB mérettel).

A kép típus konverziókat az ImageMagick **`convert`** parancsával tudjuk elvégezni (lásd 2. házi feladatlap 10. feladatában):

```
$ convert mountain.jpg mountain.png
$ eog mountain.png
$
```