Shell scriptek

Áttekintés

- A shell script a parancsértelmező (esetünkben Bash) saját programnyelve. Használatával rövid, tömör megoldások adhatók számos feladatra.
- A shell scriptek szöveges fájlok, amik a program forráskódját tartalmazzák. Fordításra nincs szükség, a shell magát a forráskódot értelmezi.
- A normál szövegfájloktól abban tér el, hogy rendelkezik futási joggal. (Lásd chmod parancs.)
- A kitejesztés általában .sh, de ez csak konvenció.

Általános formátum és futtatás

- A shell script kezdete:
  - #!/bin/bash
  - Az első sorban a shell elérési útját adjuk meg, ami majd értelmezni fogja a programot.
  - A programot a parancsértelmező egy másik példánya (egy új folyamat) fogja futtatni.
- A shell script indítása:
  - A script elérési útjának beírásával történik.
    - ./elso.sh
    - /home/antiemes/proba/elso.sh

A shell script létrehozása

- A cat parancs fájlba irányításával
  - cat > elso.sh
  - Amit beírunk, az lesz majd a fájlban. A bevitel végét Ctrl-D-vel jelezzük.
  - Akkor hasznos, ha a vágólapon van a program, így egy fájlba tudjuk másolni a vágólap tartalmát.
- Szövegszerkesztővel
  - mcedit elso.sh
  - vi elso.sh

Az mcedit

- Egyszerűen használható szövegszerkesztő.
- Fontosabb funkciók:
  - Mentés: F2 (Esc 2 kombinációval helyettesíthető)
  - Kilépés: F10 (Esc 0 kombinációval helyettesíthető)
  - Visszavonás: Ctrl-U
  - Másolás:
    - a másolandó szöveg elejére állunk, majd F3
    - a másolandó szöveg végére állunk, majd újra F3
    - a kijelölt szöveg beszúrása a kurzor után: F5
    - A kijelölt szöveg törlése: F8 (ha nincs kijelölve semmi, akkor az aktuális sort törli)

Sourcing

- A programok úgy is futtathatók, mintha azokat most írnánk be
  - source elso.sh
  - ./elso.sh
  - /home/antiemes/proba/elso.sh
  - Ilyenkor nem indul új parancsértelmező (új folyamat).
  - Többek közt a változók kezelése eltérő.

Változók – I.

- A legtöbb programnyelvhez hasonlóan itt is használhatunk változókat.
- A változók típus nélküliek. Tartalmazhatnak szöveget, számot.
- A változókat nem kell deklarálni.

Változók – II.

- Értékadás
  - valtozo=ertek — Alapvető szintaxis
  - szoveg=hello — Egy egyszerű példa
  - fnev=valami.txt — A tartalom egy fájlnév
  - sz1=”Ebben a szovegben van szokoz” — Dupla idézőjel
  - sz2='Egy masik szoveg' — Szimpla idézőjel
- A szimpla idézőjel a tartalmat teljesen szó szerint véve adja át.
- A dupla idézőjel majdnem mindent szó szerint ad át. (A részletekről később.)

Változók – III.

- A változó értékének lekérdezése
  - $valtozo alakban történik.
  - echo $szoveg
  - cat $fnev
  - A változónév esetleg összefolyhat egy, a változónevet követő stringgel. Ezt a ${valtozo} alakkal tudjuk elkerülni
    - echo $szoveghello A shell a szoveghello nevű változót keresi
    - echo ${szoveg}hello A shell a szoveg nevű változót keresi és a hello-t stringként kezeli.

Változók – IV.

- A $valtozo kifejezés a változó értékére cserélődik. Ez a legtöbb esetben megfelelő működést biztosít, de a változó felhasználásakor is célszerű dupla idézőjelet használni, vagyis a $valtozo helyett a ”$valtozo” alakot. A dupla idézőjel a változókat feloldja, de minden mást szó szerint hagy.

Változók – V.

- Több változót, illetve stringet egymás után írva azok tartalma összefűzhető. A változót el kell választani az utána írt stringtől
  - echo $valt1$valt2$valt3 Alapeset.
  - echo ”$valt1$valt2$valt3” Felkészülünk arra, hogy lehetnek a változókban speciális karakterek is.
  - echo ${valt1}szoveg$valt3 Csak az összefolyás ellen védekezünk.
  - echo ”$valt1”szoveg”$valt3” A két változót külön idézőjelezzük, ami egyben el is választjat őket a stringtől.
  - echo ”${valt1}szoveg${valt3}” Az összefolyás ellen is védekezünk és a speciális karakterekre is figyelünk.

Parancssori argumentumok – I.

- Az eddigi parancsok paramétereihez hasonlóan a saját programunk is fogadhat parancssori paramétereket.
- Argumentumok átadása:
  - ./elso.sh elso_argumentum masodik_argumentum
- Az argumentumok változók formájában érhetők el a programon belül. Az első argumentum a $1 változóban tárolódik, a második a $2-ben, stb.
- Ezeket a változókat is célszerű ”$1”, ”$2”, stb. alakban használni.

Parancssori argumentumok – II.

- Írjuk ki az első paramétert
  - echo ”$1”
- Írjuk bele az első paraméterben megadott stringet a második paraméterben megadott fájlba
  - echo ”$1” > ”$2”
- Másoljuk az első paraméterben megadott fájlt a második paraméterben megadott könyvtárba
  - cp ”$1” ”$2”

Parancssori argumentumok – III.

- Feladat:
  - Keressünk az 1. paraméterben megadott könyvtárban olyan fájlokat,
  - amik illeszkednek a 2. paraméterben megadott mintára,
  - szűrjük a fájlok tartalmát a 3. paraméterben megadott mintával
  - majd ezt irányítsuk át a 4. paraméterben megadott könyvtárban levő, 5. paraméterben megadott nevű fájlba
- Megoldás:
  - find ”$1” -type f -name ”$2” -exec grep ”$3” > ”$4/$5” ”{}” \;

Parancssori argumentumok – IV.

- Feladat:
  - Az 1. paraméterben megadott fájl 2. paraméterben megadott sorszámú sorát (tehát azt az egy sort) írjuk ki a képernyőre.
  - Számoljuk meg, hogy a 3. paraméterben megadott könyvtárban levő .txt fájloknak összesen hány sorában szerepel a 4. paraméterben megadott szó és ezt fűzzük hozzá az 5. paraméterben megadott fájlhoz.
- Megoldás:

```
head -n ”$2” ”$1” | tail -n 1

find ”$3” -type f -name ”*.txt” -exec grep ”$4” ”{}” \; | wc –l
        >> ”$5”
```

Változók átadása – I.

- Ha a programunk indítása előtt egy változónak értéket adunk,
  - azt a program nem fogja tudni kiolvasni,
  - illetve nem fogja tudni módosítani.
- Az export paranccsal a változót exportálhatjuk,
  - így a program ki fogja tudni olvasni,
  - megváltoztatni viszont csak a programon belüli lokális másolatot tudja.

Változók átadása – II.

- Feladat:
  - Adjunk értéket a DIR1, DIR2, F1, F2, SZ változóknak, majd hozzuk létre a DIR1, azon belül a DIR2 könyvtárakat, a DIR2-n belül az F1 fájlt, amibe az SZ szöveg kerüljön, és az F2 fájlt, amibe egy rekurzív hosszú lista kerüljön a DIR1-ből indulva.
- Megoldás:
  - export DIR1=egyik #Hasonlóan a többivel is, de a programon kívül! A program csak olvasni fogja ezeket a változókat!

```
mkdir -p ”$DIR1/$DIR2”
echo ”$SZ” > ”$DIR1/$DIR2/$F1”
ls -lR ”$DIR1” > ”$DIR1/$DIR2/$F2”
```

Programok visszatérési értéke – I.

- A C programok main függvényének return utasítása a program visszatérési értékét állítja be.
  - Hibátlan esetben 0.
  - Hiba esetén hibakód.
  - A program futtatása után a $? speciális változóban kapjuk meg ezt a számot.

ls van_ilyen_file	ls nincs_ilyen_file
echo $?	echo $?
0	2

  - Az ls parancs a nem létező fájlra egy 2-es hibakódot adott vissza.

Programok visszatérési értéke – II.

- A 0 jelenti a helyes lefutást, vagyis az igaz értéket
- Bármely nem 0 valamilyen hibát jelent, tehát hamis értéket.
- true parancs: A visszatérési értéke mind 0
- false parancs: A visszatérési értéke mindig 1

Elágazás (if) – I.

- A visszatérési érték alapján a program el tud ágazni.
- Szintaxis:
  - if parancs ; then parancs_igaz [; else parancs_hamis] ; fi
  - A ; helyett újsor is írható. (És ez ajánlott is.)
- Működés:
  - Lefut a parancs.
  - Ha a visszatérési értéke 0 (igaz), akkor a parancs_igaz hajtódik végre.
  - Egyébként a parancs_hamis.

Elágazás (if) – II.

- Feladat:
  - Írjuk ki, hogy egy fájlban szerepel-e egy adott minta, vagy nem:
- Megoldás:

```
if grep -q minta fájl
    then
            echo szerepel
    else
            echo nem szerepel
fi
```

A test utasítás – I.

- A test utasítás meg tudja vizsgálni, hogy:
  - Egy string hossza 0-e.
  - Egy fájl létezik-e.
  - Egy fájl milyen típusú (sima fájl, könyvtár, link, stb.).
  - Milyenek a fájl jogai.
  - Két szám egyenlő-e, illetve melyik a nagyobb.
  - Két string egyenlő-e.
- Ezekre a választ a visszatérési értékben adja meg (0: igen, 1: nem).
- A test az if utasítással együtt használható hatékonyan

A test utasítás – II.

- Feladat:
  - Nézzük meg, hogy a bejegyzés létezik-e.
- Megoldás:

```
if test -e file_name
    then
            echo Letezik
    else
            echo Nem letezik
fi
```

A test utasítás – III.

- Feladat:
  - Bővítsük ki az előző feladatot úgy, hogy vizsgálja meg, hogy a bejegyzés könyvtár-e, vagy nem
- Megoldás

```
if test -e file_name
   then
        echo Letezik
        if test -d file_name
           then
                echo Es konyvtar
           else
                echo Es nem konyvtar
        fi
   else
        echo Nem letezik
fi
```

A test utasítás – IV.

- A test utasítás helyett használható a [is, de ilyenkor a] használata is szükséges
- Feladat:
  - Döntsük el, hogy a $1, vagy a $2 a nagyobb szám.
- Megoldás:

```
if [ ”$1” -gt ”$2” ]
    then
            echo Az elso a nagyobb.
    else
            echo A masodik a nagyobb, vagy egyenlőek.
fi
```

- Fontos, hogy a szintaktikai elemeket legalább egy szóközzel válasszuk el egymástól.

Többszörös elágazás (case) – I.

- Egy kifejezés (általában változó) lehetséges értékeire tudunk eseteket felsorolni és aszerint többféle működést előírni.
- Szintaxis:

```
case kifejezes in
    minta1)
            parancsok1
    ;;
    minta2)
            parancsok2
    ;;
    *)
            parancsok_egyeb
    ;;
esac
```

  - Ha a kifejezés a minta1-gyel egyezik meg, akkor a parancsok1 blokk fog végrehajtódni. Ha a minta2-vel egyeik meg, akkor parancsok2 blokk (stb.), egyébként a parancsok_egyeb blokk. Utóbbi opcionális.

Többszörös elágazás (case) – II.

- Feladat:
  - A hét napjainak kiírása szám alapján
- Megoldás:

```
case napszam in
    0)
            echo hetfo
    ;;
    1)
            echo kedd
    ;;
    #Analóg módon a többi napot is felsorolhatjuk...
esac
```

Többszörös elágazás (case) – III.

- Feladat
  - Decimális-bináris konverzió (kis számokra)
- Megoldás

```
case decszam in
    0)
                        binszam=”000”
    ;;
    1)
            binszam=”001”
    ;;
    2)
            binszam=”010”
    ;;
    3)
            binszam=”011”
    ;;
esac
```

Többszörös elágazás (case) – IV.

- Feladat:
  - Készítsünk a case segítségével menüt.
- Megoldás:

```
case menu in
    egyik)
                        #Az egyik tipusu mukodes...
    ;;
    masik)
                        #A masik tipusu mukodes...
    ;;
    kilep)
                        echo Viszlat\!
    ;;
    *)
            echo Nincs ilyen menupont.
    ;;
esac
```

Szöveg bekérése

- Szintaxis:
  - read valtozo1 [valtozo2 […]]
- Egy sor szöveget olvas be, amit szavanként a megadott változókba tesz. Az utolsó változóba a sor végéig bekerül az összes tartalom.
- Példa
  - read x y z Az első szó az x-be, a második szó az y-ba, a sor összes többi része a z-be kerül.
  - read a Az egész sort az a-ba teszi.

For ciklus – I.

- Más nyelvekhez hasonlóan itt is használhatunk for ciklust. Itt megadott értékeken tudunk végigmenni.
- Szintaxis:
  - for valtozo in ertek1 [ertek2 [...]]
    do
    utasítások
    done
- A valtozo egyenként felveszi a megadott értékeket.
  - for nap in hetfo kedd szerda csutortok pentek szombat vasarnap
    do
    echo $nap
    done

For ciklus – II.

- Az elemek felsorolása nem túl rugalmas, de szerencsére a shell a shell mintákat be tudja helyettesíteni.
- Példa: Menjünk végig az összes fájlon és mindegyik fájlt másoljuk be egy könyvtárba, aminek a nevét előzőleg bekértük. Az új fájlok neve legyen .bak-kal kiegészítve.

```
read konyvtar
mkdir $konyvtar
for file in *
do
        cp $file ”$konyvtar/$file”.bak
done
```

For ciklus – III.

- A shell minta lehet összetettebb is, illetve több mintát is használhatunk. Minden a-val kezdődő .sh, vagy .pl fájl sorait számoljuk meg. (Külön.)

```
for file in a*.sh a*.pl
do
    wc -l ”$file”
done
```

For ciklus – IV.

- A for ciklust a C-ben megszokotthoz hasonlóan is használhatjuk

```
for ((i=0; i<10; i++))
do
    utasítások
done
```

- Fontos, hogy dupla zárójelet használunk.
- Szintén fontos, hogy a dupla zárójelen belül nem kell $ jel.

For ciklus – V.

- Rajzoljunk ki egy téglalapot. A szélességet és a magasságot a billentyűzetről kérjük be.

```
read szeles
read magas
for ((y=0; y<magas; y++))
do
   for ((x=0; x<szeles; x++))
   do
            echo -n ”*”
   done
   echo
done
```

C-hez hasonló szintaxis

- Nem csak a for ciklus kezelhető a C-ben megszokotthoz hasonlóan.
- Aritmetikai műveletek:

```
((z=x+y))
((i++))
```

- Aritmetikai műveletek, ha az eredményt le akarjuk kérdezni

```
echo $((x+5))
```

- Logikai kifejezések az if utasítás feltételeként

```
if ((x<6))
```

Parancs-behelyettesítés (backtick)

- Az eddig megismert idézőjeleken (' és ") kívül van egy harmadik is, ez a backtick (`).
- Ha két ` jel közé írunk valamit, azt a shell lefuttatja és a kifejezést arra cseréli le, amit a lefuttatott parancs kiírt.
- Szintaxis:
  - `futtatandó parancs`
- Illetve:
  - $(futtatandó parancs)

Az expr parancs

- Az expr paranccsal aritmetikai és logikai kifejezéseket értékelhetünk ki. Ismeri a négy alapműveletet, a zárójeleket, illetve néhány logikai műveletet is:
  - expr 3 + 2 5-öt ír ki. Figyeljünk rá, hogy mindegyik tokent lealább egy szóközzel elválasszuk.
  - expr 9 / 2 4-et ír ki. Lefelé kerekít, illetve csak az egészeket ismeri.
  - expr 5 * 3 A *-ot a shell behelyettesítené, ha nem használnánk \-t.
  - expr \(2 + 5 \) * 6 A zárójelnél is kell a \.

Elöltesztelős ciklus (while) – I.

- Szintaxis:

  while feltétel
  do
  utasítások
  done

- A feltétel itt is egy parancs (mint az ifnél). A ciklus addig működik, amíg a feltétel igaz (tehát a parancs visszatérési értéke 0).

Elöltesztelős ciklus (while) – II.

- Egyszerű naptár/óra

```
while true
do
    clear
    date
done
```

- Új parancsok:
  - clear Törli a képernyőt
  - date Kiírja vagy beállítja a dátumot. A dátum kiírását sokféleképpen formázhatjuk.

Elöltesztelős ciklus (while) – III.

- Olvassunk be sorokat a billentyűzetről addig, amíg lehet:

```
while read sor
do
    utasítások, amikben a sor változót feldolgozzuk
done
```

- Olvassunk be egy fájlból minden sort és dolgozzuk fel a sorokat (fájl neve a file változóban):

```
while read sor
do
    utasítások, amikben a sor változót feldolgozzuk
done < $file
```

Elöltesztelős ciklus (while) – IV.

- Számoljuk meg az a-val kezdődő, a b-vel kezdődő és az egyéb sorokat az 1. paraméterben megadott fájlban.

```
aval=0; bvel=0; egyeb=0
while read sor
do
    if echo ”$sor” | grep -q ^a
            then
                    ((aval++))
    elif echo ”$sor” | grep -q ^b
            then
                    ((bvel++))
            else
                    ((egyeb++))
    fi
done < ”$1”
echo $aval $bvel $egyeb
```

Illeszkedik-e egy sor egy regexpre?

- A példában a grepet használtuk:
  - if echo ”$sor” | grep -q ^a
- De az expr is tudja ugyanezt:
  - if expr ”$sor” : ^a > /dev/null
- Vagy kombinálva a C szintaxist a backtick-kel:
  - if (($(expr ”$sor” : ^a)))

Tetszőleges mennyiségű paraméter – I.

- A paraméterek számát egy speciális változó tárolja:
  - echo A programnak $# darab parametert adtal at.
- Az összes paraméterhez is tartozik egy speciális változó: $@
- A paramétereken for ciklussal tudunk végigmenni:

```
for p in ”$@”
do
    echo Atadott parameter: ”$p”
done
```

Tetszőleges mennyiségű paraméter – II.

- A shift parancs segítségével is feldolgozhatjuk a paramétereket. A shift levágja a paraméterlista elejéről az első paramétert, a többit pedig eggyel előrébb csúsztatja. Opcionálisan egy szám is megadható a shiftnek, akkor annyi paramétert fog kicsúsztatni.

```
while [ $# -ne 0 ]
do
    echo Az aktualis parameter: $1
    shift
done
```

A tr parancs

- A tr paranccsal megadott karaktereket másik karakterekre cserélhetünk:
  - tr ”[a-z]” ”[A-Z]” A kisbetűket nagybetűre cseréli.
    tr ”[A-Za-z]” ”[N-ZA-Mn-za-m]” rot13 kódolás.
- Megadott karaktereket törölhetünk is:
  - tr -d [aouei] Minden magánhangzót töröl.
- A megadott karakterek közül csak az ismétlődéseket törli:
  - tr -s [A-Z] Minden ismételt nagybetűből csak egy marad.

Basename és dirname

- A basename a paraméterként megadott elérési útból csak az utolsó elemet hagyja meg.
  - basename /home/antiemes/proj
    proj
- A dirname a paraméterként megadott elérési útból az utolsó elemet levágja.
  - dirname /home/antiemes/proj
    /home/antiemes

Függvények – I.

- Bash-ben a függvények hasonlóan működnek, mint a programok:
  - A függvény nevével tudjuk meghívni.
  - Paramétereket tudunk neki átadni a függvény neve után felsorolva.
  - A paramétereket a $1, $2, … változókon keresztül érhetjük el.
- Szintaxis:
  - function fuggveny_neve
    {
    utasítások
    }

Függvények – II.

- Valósítsuk meg a rot13 függvényt és az 1. paraméterben megadott fájlt kódoljuk, majd írjuk bele a 2. paraméterben megadott fájlba és a képernyőre is. Majd kódoljuk is vissza a képernyőre.
  - function rot13
    {
    tr ”[A-Za-z]” ”[N-ZA-Mn-za-m]”
    }

    cat ”$1” | rot13 | tee ”$2”
    cat ”$2” | rot13

A sed (stream editor) – I.

- A sed segítségével szövegeket lehet előre megadott módon szerkeszteni.
- Hatáskörök (pl):
  - első n, utolsó n, intervallum
  - páros, vagy páratlan sorok
  - bizonyos mintára illeszkedő sorok
  - egy mintára illeszkedő sortól egy másik mintára illeszkedő sorig
- Műveletek
  - Bizonyos sorok elhagyása, vagy kiírása
  - A sorokon átalakítások elvégzése
- Egy soros Sed parancsok (sed oneliners) gyűjteménye: http://www.catonmat.net/blog/wp-content/uploads/2008/09/sed1line.txt
- A sed gyakorlatilag univerzális programnyelvnek tekinthető
- Érdekesség: Sed-ben írt játékok, pl. Sokoban: http://sed.sourceforge.net/local/games/

A sed (stream editor) – II.

- A sed-et azonban 99%-ban cserére használják:
  - sed 's/mit/mire/'
  - s: substitution, vagyis csere
  - mit: regexp, amire keresünk
  - mire: amire kicseréljük
  - A / jel helyett bármilyen más karakter is megadható. Olyat célszerű választani, ami nem szerepel a mintában.
  - Az utolsó / után megadható még néhány kapcsoló is:
    - g: Globális csere, vagyis nem áll meg az első lehetőségnél
    - i: Kis-nagybetű-nem-érzékeny csere

Egy hosszabb példa: Thumbnailek – I.

- Egy small nevű könyvtárba készítsük el az aktuális könyvtárban levő jpg képek kicsinyített verzióját (thumbnail-jét). Pl. a foo.jpg thumbnail-je legyen foo-small.jpg
- A kicsinyítéshez a convert parancsot használjuk (imagemagick csomag)
- Külön függvénybe tegyük
  - a -small utótagú név előállítását,
  - és a kicsinyítést.

Egy hosszabb példa: Thumbnailek – II.

- A -small utótag:
  - function small
    {
    echo ”$1” | sed 's,.jpg$,-small.jpg,i'
    }
- A fájl neve az első paraméterben érkezik.
- A fájlnév végén levő .jpg végződést lecseréljük a -small.jpg-re. ($ a sor végére illeszt.)

Egy hosszabb példa: Thumbnailek – III.

- A kicsinyítés:
  - function thumb
    {
    convert ”$1” -resize \>128x128 small/”$(small ”$1”)”
    }
- A convert parancs 1. paramétere a kicsinyítendő fájl neve.
- A 2. paraméter mondja meg, hogy mit tegyen a képpel (átméretezés), a 3. paraméter jelentése: töltsünk ki egy 128x128-as négyzetet az eredeti arányok megtartásával.
- A 4. paraméter a kimeneti fájl neve.

Egy hosszabb példa: Thumbnailek – IV.

- Vizsgáljuk meg, hogy létezik-e a small bejegyzés.
  - Ha nincs, hozzuk létre.
  - Ha könyvtár, akkor használjuk.
  - Ha valami más, akkor adjunk hibaüzenetet.
  - ```
    if [ -e small ]
    then
            if [ -d small ]
                    then
                            echo A konyvtar mar letezik
                    else
                            echo Van mar valami mas itt small neven
                            exit 1
            fi # az if [ -d small ] vege
    else
            mkdir small
    fi # az if [ -e small ] vege
    ```
    ```

Egy hosszabb példa: Thumbnailek – V.

- Majd a függvények használatával következik a tényleges munka:
  - for file in *.[jJ][pP][gG]
    do
    thumb ”$file”
    done
- A [jJ][pP][gG] minta a jpg, JPG, Jpg, stb-re illeszkedik.

Reguláris kifejezések – I.

- Grep
  - Global Regular Expression Print
- Használat
  - grep <minta> <fájlnév>
  - Alap esetben a minta pontos előfordulását keresi.
  - A keresés case sensitive.
- Opciók
  - -i (--ignore-case) a keresés nem case sensitive
  - -v (--invert-match) a minta NE forduljon elő
  - -n írja ki az illeszkedő sor számát
  - -E egrep üzemmód
  - -P perl üzemmód

Reguláris kifejezések – II.

- A minta azonban lehet kifejezés is, reguláris kifejezés
  - Alap (basic)
  - Bővített (extended)
  - Perl (perl)
- Reguláris kifejezés (regexp)
  - Egy olyan karaktersorozat, amely egy adott keresendő karaktersorozatot ír le, arra illeszkedik.
- Literal match – szó szerinti egyezőség
  - A regexp karaktersorozat egy az egyben kell illeszkedjen.
  - A sort karaktersorozatként és nem szavanként kezeli!
  - grep ”az” <fájlnév>

Reguláris kifejezések – III.

- Horgony kifejezések
  - ^ sor eleji egyezőség
  - grep ”^Az ” <fájl> kiír minden ”Az” névelővel kezdődő sort.
  - $ sor végi egyezőség
  - grep ”.$” <fájl> kiír minden pontra végződő sort.
  - grep ” az$” <fájl> kiír minden „az” névelőre végződő sort.
- Tetszőleges karakter illeszkedése
  - A . segítségével lehet tetszőleges karakterre illeszkedni.
  - grep ”..len” <fájl> közvetlenül, egyetlen
    de élen NEM

Reguláris kifejezések – IV.

- Zárójeles kifejezések – ’[’ és ’]’
  - Karaktercsoportok illesztésére
  - grep ” [ea]z ” <fájl> megtalál minden önálló ez és az szót
  - grep ”^[A-Z]” <fájl> minden nagybetűvel kezdődő sor

  Ugyanez Perl szintaxisban (POSIX karakter osztályok)
  - grep ”^[[:upper:]]” <fájl>
- Ismétlődés (nulla vagy tetszőleges alkalommal) – ’*’
  - grep ”([A-Za-z]*)” <fájl> zárójelek között tetszőleges szöveg

Reguláris kifejezések – V.

- Escape-zés – ’\’
  - Speciális karaktereket '(', ')', '[', ']', '.', '*', '?', '+', '^' és '$' szó szerint (literal) értelmezi.
  - grep "^[A-Z].*\.$" <fájl> minden csak karaktert tartalmazó nagybetűvel kezdődő és pontra végződő teljes sor

Bővített reguláris kifejezések – I.

- -E kapcsolóval vagy egrep paranccsal
- Csoportosítás – ()
  - Kifejezéseket lehet csoportosítani.
  - grep -E "(kifejezés)" <fájl> egy csoport megadása
- Választás – |
  - grep –E ”(az | ez)” <fájl> ’ az ’ vagy ’ ez ’ illesztése
- Többszöri illesztések – ’*’ ’+’ ’?’ ’{<szám>} ’{szám,szám}’
  - grep –E ”(A(z)? | a(z)?)” <fájl> határozott névelők illesztése
  - grep –E ” program[a-z]+ ” <fájl> program ragozva vagy összetett szó első részében illesztése

Bővített reguláris kifejezések – II.

- grep –E ” [a-z]{4} ” <fájl> négybetűs szavak illesztése
- (POSIX osztályok)

POSIX osztály	Megfelel	Jelentése
[:upper:]	[A-Z]	Nagybetűk
[:lower:]	[a-z]	Kisbetűk
[:alpha:]	[A-Za-z]	Nagy- és kisbetűk
[:alnum:]	[A-Za-z0-9]	Számjegyek, nagy- és kisbetűk
[:digit:]	[0-9]	számjegyek
[:xdigit:]	[0-9A-Fa-f]	hexadecimális számjegyek
[:punct:]	[.,!?:…]	elválasztó karakterek
[:blank:]	[\t]	szóköz és TAB
[:space:]	[\t\n\r\f\v]	üres karakterek
[:cntrl:]		vezérlő karakterek
[:graph:]	[^ \t\n\r\f\v]	nyomtatásvezérlő karakterek
[:print:]	[^\t\n\r\f\v]	nyomtatásvezérlő karakterek és szóköz

Beugró feladatok – I F.

- Feladat:
  - Írjon sorszam.sh néven Bash scriptet, ami az aktuális könyvtárból indulva rekurzívan megkeresi azon fájlokat, amelyek kiterjesztése .txt, majd kiírja, hogy ezek egyenként hány sort tartalmaznak, valamint azt is, hogy összesen hány sort tartalmaznak.
- Példa:
  - $ ./sorszam.sh
    23 ./almafa.txt
    45 ./kortefa.txt
    12 ./kert/szilva.txt
    80

Beugró feladatok – I M.

- Megoldás
  - #!/bin/bash

    find . -type f -name ”*.txt” -exec wc -l ”{}” \;
    find . -type f -name ”*.txt” -exec cat ”{}” \; | wc -l

Beugró feladatok – II F.

- Feladat:
  - Írjon leghosszabb.sh néven Bash scriptet, ami az aktuális könyvtárból indulva rekurzívan megkeresi a leghosszabb fájlt (vagy az egyik leghosszabbat) és kiírja ennek adatait.
- Példa:
  - $ ./leghosszabb.sh
    -rw-r--r-- 1 root root 5290 Mar 25 18:23 ./bcd/libreoffice.sh

Beugró feladatok – II M.

- Megoldás:
  - #!/bin/bash

    find . -type f -exec ls -l ”{}” \; | sort -n -k 5 | tail -n 1

Beugró feladatok – III F.

- Feladat:
  - Írjon vegennev.sh néven Bash scriptet, ami az aktuális könyvtárból kiindulva rekurzívan az összes fájlban megkeresi és kiírja a tetszőleges névre végződő sorokat. Minden név kéttagú és tagjaik nagybetűvel kezdődnek és szóközzel vannak elválasztva.

Beugró feladatok – III M.

- Megoldás:
  - #!/bin/bash

    find . -type f -exec cat ”{}” \; | grep -E '[A-Z][a-z]+ [A-Z][a-z]+$'

Beugró feladatok – IV F.

- Feladat:
  - Írjon Bash héjprogramot etlapok.sh néven, mely kiírja az összes lehetséges főétel-köret párosítást az alábbi módon: Az első paraméterben megadott könyvtárban levő fájlok nevei a lehetséges főételek, a második paraméterben megadott könyvtárban levő fájlok nevei pedig a lehetséges köretek
- Példa:
  - $ ./etlapok.sh foetelek koretek
    porkolt – csigateszta
    porkolt – hasabburgonya
    porkolt – rizs
    rantotthus – csigateszta
    rantotthus – hasabburgonya
    rantotthus – rizs

Beugró feladatok – IV M.

- Megoldás:
  - #!/bin/bash

```
for f in $1/*
do
   for k in $2/*
   do
      echo $(basename $f) - $(basename $k)
   done
done
```

Beugró feladatok – V F.

- Feladat:
  - Írjon sorlistaz.sh néven bash scriptet, amely a paraméterként megadott fájlok paraméterként megadott sorszámú sorát listázza ki a példában megadott formátumban.
    - Első paraméter: a fájlok hányadik sorát írja ki
    - A többi (tetszőleges számú) paraméter: fájlok nevei
- Példa:
  - ./sorlistaz.sh 4 alma.txt korte*
    A(z) alma.txt 4. sora:
    zoldseg gyumolcs
    A(z) korte.txt 4. sora:
    szep szinarany botjat
    A(z) kortefa.doc 4. sora:
    Sor vagy nem sor, ez itt a kerdes...

Beugró feladatok – V M.

- Megoldás:
  - #!/bin/bash

```
n=$1
shift
while [ $# -gt 0 ]
do
        echo 'A(z) '”$1 $n. sora:”
        head -n $n $1 | tail -n 1
        shift
done
```