

## 6. SED, AWK

# Visszatekintés

- Számítógépek, információk, számábrázolás, kódolás
- Felépítés, kliens-szerver szerep, fájlrendszerek
- Alapvető parancsok, folyamatok előtérben, háttérben
- I/O átirányítás, szűrők, reguláris kifejezések
- Változó, parancs behelyettesítés, aritmetikai, logikai kifejezések
- Vezérlési szerkezetek

# Mi jön ma?

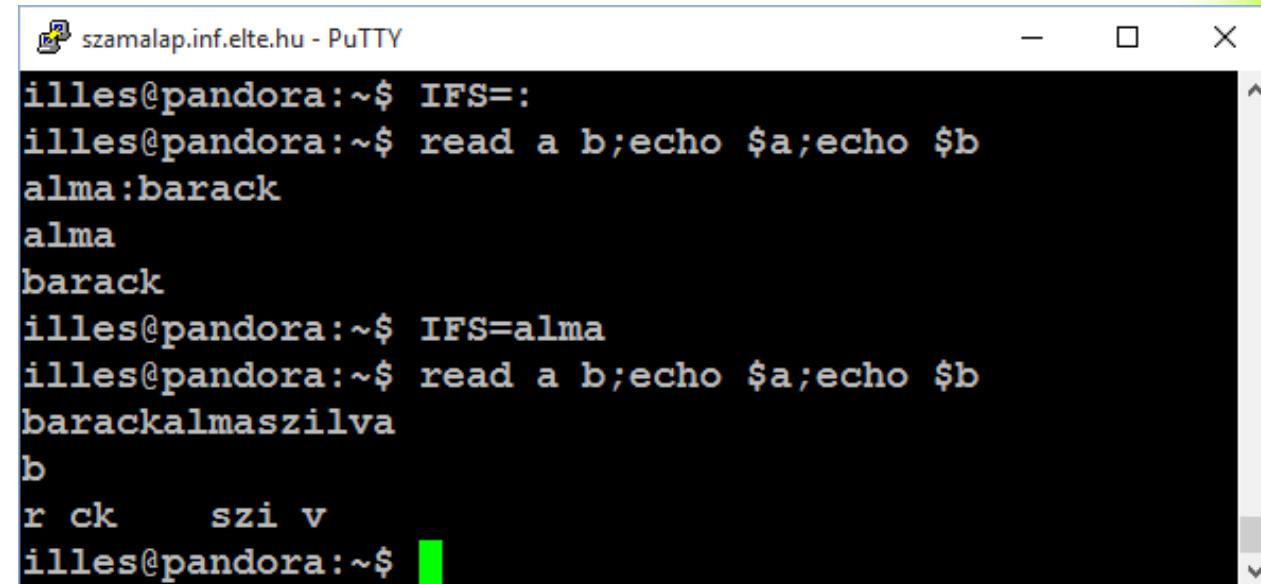
- Még több szűrés!
  - SED
- Még több programozási lehetőség!
  - AWK

# Input –Output utasítások

- Legtöbb parancs szabvány kimenetre ír
  - echo parancs, -n nincs új sor
- Bemenetről olvas:
  - read parancs
    - Példa: read alma #alma változóba olvas enterig
    - read korte; echo \$korte # ok
    - echo szia|read a; echo \$a # üres, a read nem szűrő
  - line parancs - szűrő
    - Standard inputról olvas egy teljes sort, amit a standard outputra ír
    - echo szia|`line`
    - echo alma barack|for i in `line` ; do echo \$i; done # eredmény: alma, barack külön sorban

# IFS - Internal Field Separator

- Több Unix implementációnak része!
  - BASH tartalmazza.
- Az alapértelmezett elválasztó helyett(helyköz, tab) új elválasztó karakter definiálása!
  - Célszerű egy karaktert megadni!
  - IFS=: #kettőspont az új elválasztó



```
szamalap.inf.elte.hu - PuTTY
illes@pandora:~$ IFS=:
illes@pandora:~$ read a b;echo $a;echo $b
alma:barack
alma
barack
illes@pandora:~$ IFS=alma
illes@pandora:~$ read a b;echo $a;echo $b
barackalmaszilva
b
r ck      szilva
illes@pandora:~$
```

# Szűrjünk még...

- Mit tudunk eddig szűrni?
  - Sorból kivágni karakter, mező elemeket (cut)
  - Teljes sorokat (grep, reguláris kifejezések)
- Elég ez?
- Mit nem tudunk?
  - Sok mindent....például nem tudunk soron belül szövegrészeket keresni, cserélni!

# SED – Stream EDiTOr

- Szűrő – a bemeneti sorokon a megadott szerkesztési műveleteket, módosításokat végzi (edit)
- Feladata: Komplex behelyettesítések, cserék a szabványos bemenetre érkező sorokon, eredmény a szabványos kimeneten. Veszi az összes sort (ciklus), majd minden egyes soron a parancsai végrehajtódnak, a módosított sor a kimenetre kerül!
- Példa: `cat osztaly | sed 's/3/9/g' # A "s/3/9/" idézőjel is jó!`
  - Keressük meg a sorokban az összes (g) 3-t, majd ezt cseréljük ki 9-re.

# sed fontosabb paraméterek

- -n, Nincs automatikus kiírás a szabvány kimenetre. Csak a sed script kiíró utasításai írnak a kimenetre.
- -f scriptfile, A paraméterül megadott fájlban van a program, a sed script. Jellemzően egy sorba egy parancsot írunk, de a ; elválasztóval több parancs is kerülhet egy sorba.
- -e A sed parancssorában több script van. Egy esetén elhagyható.



# sed script utasítások

- Több parancs megadása közvetlen script parancsban:
  - ; Pontosvessző a parancsok között!
    - Példa: `cat osztaly|sed 's/3/9/g ; s/9/21/'` # Cseréljük az osztály soraiban az összes 3-at 9-re, majd az első kilencest 21-re minden sorban!
  - Külön sorba írhatók a parancsok, másodlagos promptot kapunk!

```
$ cat osztaly | sed 's/3/9/g  
>s/9/21/' [enter]  
zoli 21 4 2 5 4  
feri 2 21 4 5 9  
juli 4 21 2 9 9  
$
```

# Sed script készítés

- Parancssorban több parancs elhelyezése lehetséges, de nem szerencsés!
- Készítsünk scriptet!
  - `chmod +x sedpelda`
  - Futtatás: `sedpelda osztaly`

```
teszt@pandora:~/> cat sedpelda
#!/bin/sed -f
#
s/3/9/g #3-9 csere mindegyik sorban, összes 3-es csere
s/9/10/ #9-10 csere mindegyik sorban, csak első 9-es csere
```

# sed fontosabb parancsok I.

- Sed parancs alakja: `sed [par] [cím] s /minta/új_minta/[jelző]`
  - Feladata: Keressük mintát, cseréljük új mintára!
  - A [par] a sed paraméterei, a fontosabbakat láttuk, többi: `man`
- Jellemzően a minta reguláris kifejezés, ezt keressük.
- A cím, amire vonatkozik a sed parancs, ez lehet reguláris kifejezés is, jellemzően szám, vagy intervallum `1,10 s/.../`.
- `$` jel az utolsó sort jelenti      `# 5,$s/alma/körte/`
- `!` jel, a tagadás operátora
  - `2!s/alma/körte/`      `# a második sor kivételével`  
                                 `# minden sorban csere`

# Egyéb cím meghatározás

- Ha nincs megadva, akkor minden sorra vonatkozik s parancs!
- $N$  – az  $N$ . sorra vonatkozik, Pl:  $4/3/9/g$  # a 4. sorban cserélünk
- $x \sim y$  –  $x$ . sor majd utána minden  $y$ . sor!
  - Példa:  $3 \sim 2/3/9/g$  # a 3. sortól kezdve minden második sor!
- $x+y$  –  $x$ . sor majd utána a következő  $y$  sor!
  - Példa:  $3+2/3/9/g$  # a 3. sortól kezdve két sor
- A teljes cím megadáshoz lásd referencia!

# A keresés-csere parancs jelző értékei

- A jelző értékei:
  - n: sorszám, a cserét az n. mintán kell elvégezni, ha elmarad,  $n=1$ . Ha n nagyobb mint utolsó előfordulás, a csere nem csinál semmit!
  - g: Az összes mintát le kell cserélni.
  - p: Kilistázza az aktuális sort! (pg együtt is lehet)
  - w fájl: Menti fájlba(hozzáfűz) az aktuális sort!
  - r fájl: Beolvassa a bemenetre a fájl tartalmát!

# sed fontosabb parancsok II.

- /új\_minta &/ Új mintát a minta elejére teszi
  - Pl: echo fradi|sed 's/fradi/hajra &/' #hajra fradi
  - A & jel jelenti a régi mintát, bárhol szerepelhet!
- \szám használata,
  - \1 az 1. reguláris kifejezés,
  - \ elnyomja a következő metakarater hatását(\.ali)
  - \n , soremelés beszúrása (\n >\2/'), a példában nincs n mert közvetlenül parancsként írjuk be!

```
illes@panda:~$ echo .ali 4 Ali baba|sed 's/\.ali \([0-9][0-9]*\) \(.*)^1. rész\  
> \2/' (enter)  
4. rész  
Ali baba  
illes@panda:~$
```

# sed fontosabb parancsok III.

- Törlés: [címtartomány]d
  - d parancs törli a címtartomány sorait
  - Pl: cat osztaly|sed '1d; s/3/9/pg; s/9/21/' #első sort törli, majd a többi soron a másik két parancs
- Hozzáfűzés: a
  - Pl: cat osztaly|sed 'a\alma' # új sorként az alma, minden sor után
- Beszúrás: i
  - Pl: cat osztaly|sed '2,3i\alma' # 2,3 sor elé alma

# sed fontosabb parancsok IV.

- y – minta karakter csere: `echo papagáj | sed 'y/ag/uh/' #pupuháj`
  - A cserélendő karakterszám azonos a cserélt karakterek számával!
- q – kilépés, adott címsor után a sed kilép
- :cimke – címke készítés
- b címke – feltétel nélküli ugrás
- t címke – feltételes ugrás, ha volt sikeres csere



# Sed példa I.

- 1. példa:

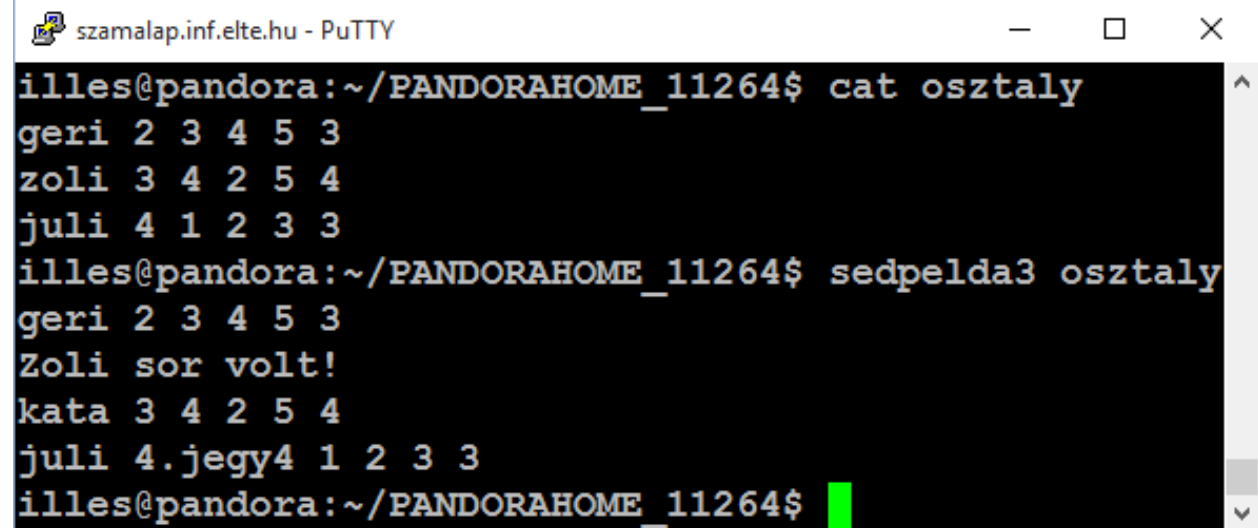
```
#!/bin/sed -f
#
# Az első és második sorban számok mögé írunk
1,2s/\([1-9][1-9]*\)ate.kívánság\1/g # minden talált szám után!!!
# A második sor után quit!
2q
```

# SED program idézőjelek

- A SED paraméterként megadott programot idézőjelek között kell megadni!
- Ez lehet az egyszeres ' , és a dupla " is!
- Van különbség!
- `x=Tibi; echo Laci ügyes! | sed "s/Laci/$x/"`
  - Eredmény: Tibi ügyes!
- `x=Tibi; echo Laci ügyes! | sed 's/Laci/$x/'`
  - Eredmény: \$x ügyes!

# Sed példa II.

```
#!/bin/sed -f
#
s/zoli/kata/ # zoli csere katára
t zolisor # feltételes ugrás, ha volt sikeres csere
# Mindegyik sorban a számok mögé írunk
2,$s/\([1-9][1-9]*\)ate.jegy\1/ #csere csak első
számnál
# A 2,$ a címzés megadása, $ jelenti az utolsó sort
b vege # feltétel nélküli ugrás
#zolisor cimke
:zolisor
i Zoli sor volt! # a feldolgozott sor elé beszúrás!!
:vege #vege cimke
```



The screenshot shows a terminal window titled 'szamalap.inf.elte.hu - PuTTY'. The user 'illes@pandora' is in the directory '~/PANDORAHOME\_11264'. They first run 'cat osztaly', which displays the following content:

```
geri 2 3 4 5 3
zoli 3 4 2 5 4
juli 4 1 2 3 3
```

Then, they run 'sedpelda3 osztaly'. The output shows the result of the sed script:

```
geri 2 3 4 5 3
Zoli sor volt!
kata 3 4 2 5 4
juli 4.jegy4 1 2 3 3
```

The terminal cursor is at the end of the last line.

# SED leírások

- A korábbi diák nem adnak teljes leírást!
- GNU SED leírás- teljes referencia
  - <https://www.gnu.org/software/sed/manual/>
- Tankönyvtár sed leírás:
  - [http://www.tankonyvtar.hu/hu/tartalom/tamop412A/2010-0011\\_szamalap1/lecke8\\_lap6.html](http://www.tankonyvtar.hu/hu/tartalom/tamop412A/2010-0011_szamalap1/lecke8_lap6.html)
- Internet ... még több leírás!

# Mi van a sed után? AWK

- A sed nagyon jó, nagyon hasznos, de ...
  - Jellemzően mintacserére használható.
  - Sorokon belüli tevékenység korlátozott.
  - Nincs aritmetikai lehetőség. (Valami van csak nem az igazi...)
  - Vezérlési szerkezetek hiányoznak. (Van ugró, feltételes ugró utasítás...,)
  - ...
- A kiút: AWK

# AWK

- Alfred V. **A**ho, Peter J. **W**einberger, Brian W. **K**ernighan
- Shell hiányosságai szövegfeldolgozáskor
- Gyakorlatilag C nyelvi lehetőségek
- Tipikus szűrő
- Gyakran shell script elemként használt
- Soronkénti szövegkezelés, végrehajtódó program
- awk –gawk (GNU AWK)
  - Referencia:  
<http://www.gnu.org/software/gawk/manual/gawk.html>

# AWK működése, szerkezete

- Parancs helye: whereis awk      # /usr/bin/awk, ...
- Az awk vagy paraméterként vagy a szabvány bemenetén várja az átdolgozandó adatokat.
- Soronkénti feldolgozás. Első sor előtti, utolsó utáni inicializálási blokk.
- A parancsblokkok {} jelek közti utasítás
- Parancsblokk előtt minta definiálható: Példa: /f.\*/
  - /reguláris kifejezés/
  - A minta egy logikai kifejezést tartalmaz: \$2 == „alma”

# AWK használata

- Program, közvetlenül mintegy paraméter
  - `awk '{ print ;}' adatfile`
    - A program minden sorra vonatkozik, kiírja azt
- File-ban a program
  - `awk -f programfile adatfile`
  - Helyette gyakran az awk programfile a parancs!
    - `#!/usr/bin/awk -f`
    - Ez az első sor parancsa.
    - pl: `$ awk_program1 adatfile`      # jellemző parancs alakja
- Szűrőként
  - `Parancs1 | awk-parancsfile`



# Bemeneti sorok elemei

- \$0 – a teljes sor
- \$1, \$2, ... - a sor első, második eleme
- Mezőelválasztó: FS (alap: helyköz vagy tab)
  - Lehet több karakter is: FS=„abc”
- Egy sor mezőinek száma: NF
- Sorrelválasztó karakter: RS (alap: újsor)
- Az eddig beolvasott sorok száma: NR
  - Több bemeneti fájl esetén: FNR, egy fájl sorszáma

# AWK példa

- Programkód:

- Futtatás:

```
$cat osztaly|elsoawk  
Kezdem a programot!  
A Geri sor tartalma: geri 2 3 4 5 3  
Itt a vége!  
$
```

```
#!/usr/bin/awk -f  
# Kezdő blokk!  
BEGIN {  
    print "Kezdem a programot!";  
    # ide jöhetnek a további kezdő, először  
    # végrehajtandó parancsok, pl mezőelválasztó  
    FS=":"  
}  
# Minden sorban keresi a geri-t, ha találja kiírja  
/geri/ {  
    print "A Geri sor tartalma: "$0;  
}  
# Befejezés  
END {  
    print "Itt a vége!";  
}
```

# AWK kimeneti sorok elemei

- print utasítás: pl: print \$0 #teljes sor kiírása
- OFS változó
  - Output Field Separator
- ORS változó
  - Output Row Separator

```
BEGIN {  
    OFS=„.”;  
    ORS=„Vége.\n”;  
}  
  
    print „Alma”, $0;  
}
```

# AWK változók, kifejezések

- Beépített változók nagybetűsek! Pl: NF
- Nincs típus: név=érték, érték lehet:szám, „szöveg”
  - `v="alma"; print v;        #alma`
- Szövegösszefűzés: nincs operátor, egymás után kell írni a változókat!
  - Pl: `{v="alma";f="fa"; print "5 "v f;} #5 almafa`
- Konverzió automatikus, ha úgy érzi, hogy kell:
  - `{v="alma";f="fa";print v+f} #0`
  - `{v="3alma";f="2fa"; print v+f} # 5`

# AWK tömbök

- `t[0]=3`, stb. megadjuk az index elemeket.
- Tömb elemei különböző típusúak lehetnek.
- Valójában asszociatív tömbök:
  - `t[„egy”]=1; print t[„egy”];`
- Tömb hossza: `length(t)`
- Egy index bent van-e a tömbben: `4 in t`
- Elem törlése: `delete t[4]`
- Többdimenziós tömbök: `tt[1,2]=3;`

# AWK műveletek, függvények

- +, -, ++, --, \*, /, %, ... - szokásos műveletek
- Logikai érték mint C-ben
- !=, ==, <, > - logikai operátorok
- && logikai és, || logikai vagy, ! tagadás
- ~, !~ minta illeszkedés, nem illeszkedés
  - A jobb oldali operandus lehet reguláris kifejezés is /.../.
  - A \$0 ~ /reg.kif/ alak rövidítve: /reg.kif/ , emiatt szerepel az awk blokkok előtt csak így!
- \*\* vagy ^ hatványozás, pl: 2\*\*3 #8

# AWK matematikai függvények

- Fontosabb beépített függvények:
  - `int(szám)` # egészrész , `print int(3.7)` #3
  - `sqrt(szám)` # négyzetgyök
  - `sin(x)`, `cos(x)` # `sin`, `cos` függvények, `x` radiánban van!
  - `rand()` # `]0,1[` intervallumban véletlenszámot generál
    - `x=int(10*rand())` # 0,1,...9
  - `exp(x)`, `log(x)` # `e ** x`, `ln(x)`
  - `atan2(x,y)` # arc. tangens `x/y`

# Egyéb hasznos awk függvények

- `{v=length("almafa");print v}` #szöveg hossza
- `split` – szöveg darabolás
  - Pl: `split(„ali:pali:robi”, n, „:”); print n[1] #ali`
- `sprintf(sz, „minta”,változók);` # mint C-ben az `sprintf`
- `system(„parancs”)` – op. rendszer par. futtatás
  - Pl: `{system(„date >datum”) }`
- File olvasás:
  - `getline <"/home/adat”; print $0;`
  - Következő `getline`, következő sor.



# AWK vezérlési szerkezetek

- Elágazás
  - `if (x % 2 == 0) print "x páros,,"; else print "x páratlan";`
- Többirányú elágazás
  - `Switch (c) { case „a”:... }; # mint C-ben`
- Ciklusok, mint C-ben
  - `while (kif) utasítás`
  - `do ... while (kif)`
  - `for(kif1;kif2;kif3) utasítás`
  - `for (i in tömb)`  
tömb[i] feldolgozása

# Példa

```
#!/usr/bin/awk -f
#
# feladat: a tanulók átlagának meghatározása
# hívása: awk_atlag osztaly
# BEGIN minta először végrehajtódik
BEGIN {
    print "Ciklus-elágazás használata!";
    if (ARGC !=2)
    {
        print "Adjon meg egy fájlnévet";
        exit 1;
    }
}
```

```
{
    print "A tanuló adatai:" $0;  # kinyomjuk az aktuális sort
    # tanuló átlag kiszámolása
    # egy tanuló jegyeinek összeget kiszámoljuk
    osszeg=0;                    # kezdőérték 0
    for (i=2;i<=NF;i++) # végigvesszük a jegyeket
    {
        osszeg+=$i;            # osszeghez adjuk az aktuális jegyet
    }
    # i. átlagot megjegyezzük
    atlagok[NR]=osszeg/(NF-1);
    print $1 " jegyeinek átlaga: " atlagok[NR];
}
# END minta a végén fut le
END {
    print NR " tanuló átlagát határoztuk meg!";
}
```

# AWK összegzése

- Help: Google awk, gawk
- BEGIN blokk, a soronkénti feldolgozás előtt hajtódik végre
- END blokk, a soronkénti feldolgozás után hajtódik végre
- Minta {soronkénti blokk}
- Ez a diasor a parancs legfontosabb elemeit foglalta össze!
- Teljes referencia:  
<https://www.gnu.org/software/gawk/manual/>

Köszönöm a figyelmet!

