

# Operációs Rendszerek Példák

Kiegészítés a gyakorlati jegyzethez

Összeállította: Rodek Lajos

Szegedi Tudományegyetem

Képfeldolgozás és Számítógépes Grafika Tanszék



# A chmod parancs I.

---

- A „pe1da” állomány futtathatóvá tétele a tulajdonos számára (a többi jog nem módosul).  
`chmod u+x pe1da`
- A „pe1da” állomány olvasási és írási jogainak tiltása az állomány csoportja és az egyéb felhasználók számára (a többi jog nem módosul).  
`chmod go-rw pe1da`
- A „szovegek” könyvtár, valamint az abban levő állományok és az alkönyvtárak teljes tartalmának futtathatóvá tétele mindenki számára (a többi jog nem módosul). A futtatási jogot csak könyvtáraknak és az eleve futtatható állományoknak adjuk meg.  
`chmod -R a+X szovegek`
- A „pe1da” állomány összes jogának megvonása az egyéb felhasználók számára (a többi jog nem módosul).  
`chmod o= pe1da`

## A chmod parancs II.

---

- A „pe1da” állományt mindenki számára olvashatóvá tesszük, a többi jogot pedig letiltjuk.

```
chmod a=r pe1da
```

Egy ekvivalens megoldás a jogok numerikus alakjának használatával:

```
chmod 444 pe1da
```

- A „pe1da” állomány a következő jogokkal fog rendelkezni: a tulajdonos olvasni, írni és futtatni is tudja, a csoport képes olvasni és futtatni, az egyéb felhasználóknak pedig semmilyen joguk sincs.

```
chmod 750 pe1da
```

A szimbolikus jogok alkalmazásával kicsit hosszabb megoldást kapunk:

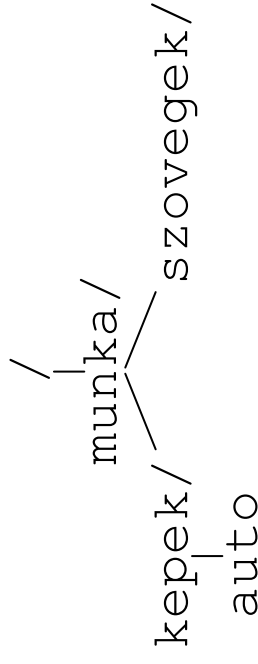
```
chmod u=rwx pe1da
```

```
chmod g=rx pe1da
```

```
chmod o= pe1da
```

# Elérési utak

- Tegyük fel, hogy a gyökérkönyvtárból nyílik egy „munka” nevű alkönyvtár, amely tartalmazza a „kepek” és „szovegek” alkönyvtárakat. Továbbá a „kepek” könyvtárban legyen egy „auto” nevű állomány. Szemléletesen:



Legyen az aktuális könyvtár a „kepek”. Ekkor:

- a „kepek” könyvtár relatív elérési útja: . vagy . /
- az „auto” állomány abszolút elérési útja: /munka/kepek/auto
- az „auto” állomány relatív elérési útja: auto vagy . /auto
- a „munka” könyvtár abszolút elérési útja: /munka vagy /munka /
- a „munka” könyvtár relatív elérési útja: .. vagy .. /
- a „szovegek” könyvtár relatív elérési útja: .. /szovegek vagy .. /szovegek/

# Munka állományokkal, könyvtárakkal I.

---

- A `cd` parancs használatának személtetése:
  - `cd, cd ~`: belépés az aktuális felhasználó saját könyvtárába
  - `cd ~root`: belépés a `root` felhasználó saját könyvtárába
- Legyenek „level” és „masolat” állományok, „szovegek” és „dokumentumok” pedig könyvtárak. A `cp` parancs használatának személtetése:
  - `cp level masolat: a,level`” állomány (tartalmának) átmásolása a „masolat” állományba
  - `cp level szovegek: a,level`” állomány bemásolása a „szovegek” könyvtárba változatlan néven
  - `cp -r szovegek dokumentumok: a,szovegek`” könyvtár teljes tartalmának átmásolása a „dokumentumok” könyvtárba

## Munka állományokkal, könyvtárakkal II.

---

- Legyenek „szoveg” és „level” állományok, „dokumentumok” pedig egy könyvtár. Az `mv` parancs használatának személtetése:
  - `mv szoveg level: a,szoveg`” állomány átnevezése „level”-re
  - `mv szoveg dokumentumok: a,szoveg`” állomány átmozgatása (áthelyezése) a „dokumentumok” könyvtárba
- Legyen „level” egy állomány, „szovegek” pedig egy könyvtár. Az `rm` parancs használatának személtetése:
  - `rm level: a,level`” állomány kitörlése
  - `rm -rf szovegek: a,szovegek`” könyvtár törlése annak teljes tartalmával együtt
- A `basename` és `dirname` parancsok használatának személtetése:
  - `basename /munka/kepek/auto: az eredmény az auto útvonal`
  - `dirname /munka/kepek/auto: az eredmény a /munka/kepek útvonal`

# Az `ls` parancs

- Legyen `level` egy állomány, „szovegek” pedig egy könyvtár. Az `ls` parancs használatának személtetése:
  - `ls -l level` („`ell`” opció): a „`level`” állomány adatainak kiírása (bővített listázás)
  - `ls -d szovegek`: a „szovegek” könyvtár mint speciális állomány adatainak kiírása
  - `ls szovegek`: a „szovegek” könyvtár tartalmának kiírása
  - `ls -l szovegek` („`egy`” opció): mint előbb, de az egyoszlopos módot használva
  - `ls -R szovegek`: a „szovegek” könyvtár teljes tartalmának kiírása, az alkönyvtárakat is beleértve
- A „kepek” és a „szovegek” könyvtárak tartalmának kilistázása.  
`ls kepek szovegek`

# Állomány- és könyvtárnevek megadása

---

- `kep?`: olyan négybetűs nevek, amelyek a „kep” szóval kezdődnek, és az utolsó karakterük tetszőleges
- `kep*`: olyan nevek, amelyek a „kep” szóval kezdődnek, amit bármi egyéb követhet (akár az üres szó is)
- `*kep`: olyan nevek, amelyek a „kep” szóra végződnek, amit bármi egyéb megelőzhet (akár az üres szó is)
- `kep[A12]`: olyan négybetűs nevek, amelyek a „kep” szóval kezdődnek, és az utolsó karakterük „A”, „1” vagy „2”
- `kep[A-Z0-9]`: mint előbb, de az utolsó karakterük nagybetű vagy számjegy
- `kep[^A-Z]`: mint előbb, de az utolsó karakterük nem nagybetű
- `kep[^A-Z]*[23]`: olyan nevek, amelyek a „kep” szóval kezdődnek, amit egy nagybetűtől eltérő karakter követ, utána bármi állhat, az utolsó karakterük pedig „2” vagy „3”



# Átírányítás és csővezeték I.

---

- A könyvtárlista elmentése a „lista” állományba.  
`ls > lista`
- Mint előbb, de ha már létezik az állomány, akkor hozzáfűzést alkalmazunk.  
`ls >> lista`
- A „level” állomány tartalmának kiírása. Az esetlegesen megjelenő hibáüzeneteket (pl. ha „level” egy könyvtár lenne, vagy ha nem lenne rá olvasási jogunk) a „hiba” állományba irányítjuk.  
`cat level 2> hiba`
- A „teszt” nevű alkönyvtár létrehozása úgy, hogy ha már létezett ilyen nevű könyvtár, akkor ne jelenjen meg hibáüzenet a képernyőn.  
`mkdir teszt 2> /dev/null`

Kihasználtnak, hogy a /dev/null speciális állomány minden bele írt adatot elnyel.

# Átírányítás és csővezetek II.

---

- A „kepek” és „szovegek” könyvtárak listájának eltárolása a „lista” állományba. Az esetlegesen keletkező hibáüzeneteket a „hiba” állományba irányítjuk. Gyakorlatilag tehát semmilyen látható kimenetet nem produkálunk.

```
ls kepek szovegek 2> hiba > lista
```

- Mint előbb, de most a „lista” állományba irányítjuk a hibáüzeneteket is.
- ls kepek szovegek &> lista
- A könyvtárlistát elmentjük a „lista” állományba, de a képernyőn is szeretnénk látni az eredményt.

```
ls | tee lista
```

- Mint előbb, de a listát továbbra is többszóloposként szeretnénk látni.

```
ls -C | tee lista
```

# Átírányítás és csővezeték III.

---

- A „kepek” és „szovegek” könyvtárak listájának eltárolása a „lista” állományba úgy, hogy az eredmény a képernyőn is megjelenik. Az esetlegesen keletkező hibáüzeneteket ugyanígy kezeljük, azaz az állományba beírjuk és a képernyőn is megjelenítjük.

```
ls kepek szovegek 2>&1 | tee lista
```

- Az „uzenet” állomány tartalmának kiírása a szabványos hibakimenetre.

```
cat uzenet 1>&2
```

- Mint előbb, de a hibáüzenetet a „hiba” állományba írányítjuk.

```
cat uzenet 2> hiba 1>&2
```

Az átírányítások sorrendje most fontos! Ha felcserélnénk őket, akkor a szabványos kimenet tartalma oda menne, ahová a hibakimenet akkor éppen írányítva van, azaz a képernyőre!

# Üzenetek megjelenítése, kiírás

- Kiírja a „A pontos ido: ” szöveget, majd közvetlenül utána ugyabba a sorba az aktuális dátumot és időt.

```
echo -n 'A pontos ido: ' ; date
```

A -n opcióra azért van szükség, hogy a date parancs kimenete ugyanabba a sorba íródjon ki. A pontosvessző leírását ld. a shellnél.

- Kiírja az „egy”, „ketto” és „harom” szavakat külön-külön sorba.

```
echo -e 'egy\nketto\nharom'
```

A -e opció segítségével használható a \n páros, ami a szövegbe egy sortörést szúr be.

- Kiírja a „Adat: tomeg 0063.80” szöveget.

```
printf 'Adat: %6s %07.2f' tomeg 63.8
```

A %s egy szöveges paramétert (szöveges konverziót), a %f pedig egy valós tizedestörtet jelöl. A százalékjel utáni számok az illető paraméter kiírásához használt karakterek számát adják meg. A %f esetén a .2 a tizedesjegyek számát közli, a százalékjel utáni 0 pedig azt eredményezi, hogy a szám elé kiíródnak a bevezető nullák is.

# Szűrők I.

---

- A „Kovács Jancsi” nevű felhasználóról nyilvántartott kritikus információkat tartalmazó sor megjelenítése az `/etc/passwd` állományból.  
`cat /etc/passwd | grep ' :Kovács Jancsi:'`
- Az összes bejelentkezések számának kiírása. A többször bejelentkezett felhasználókat többször számoljuk.  
`who | wc -l`
- A bejelentkezések felhasználói azonosító szerint rendezett listája.  
`who | sort`
- A „h123456” azonosítójú felhasználó bejelentkezéseinek kiírása. Annyi sor fog megjelenni, ahányszor be van jelentkezve.  
`who | grep 'h123456'`
- Kiírja, hogy hányszor van bejelentkezve a „h123456” azonosítójú felhasználó.  
`who | grep 'h123456' | wc -l`

## Szűrők II.

- A „szoveg” állomány sorai számának kiírása.

```
cat szoveg | wc -l
```

Fontos, hogy a következő megoldás már nemcsak a sorok számát írja ki, hanem az állomány nevét is! Ezért inkább az előzőt célszerű használni.

```
wc -l szoveg
```

Vegyük észre, hogy mind a két esetben a szám előtt szóközők is kiíródnak! Ezek eltüntetésére a shellnél látunk majd módszert.

- Tegyük fel, hogy a „lista1” és „lista2” állományok minden sora egy-egy szót tartalmaz. Készítünk egy rendezett listát, amely a két állományban előforduló szavakat tartalmazza fordított (csökkenő) sorrendben. Minden sor pontosan egy szót tartalmaz, a többször előforduló szavaknak csak egyetlen példányát hagyjuk meg, továbbá a kisbetűket és a nagybetűket nem különböztetjük meg!

```
sort -fru lista1 lista2
```

## Szűrők III.

---

- A „szoveg” állomány harmadik sorának megjelenítése.

```
head -n 3 szoveg | tail -n 1
```

**Egy ekvivalens megoldás:**

```
head -n 3 szoveg | tail -n +3
```

# A shell I.

---

- A „20\$” szöveg kiírása. A három megoldás ekvivalens egymással.

```
echo '20$'
```

Második megoldás:

```
echo "20\ $"
```

Harmadik megoldás:

```
echo 20\ $
```

- A „Hello” szöveg kiírása a „trukkos nev” nevű állományba:

```
echo Hello > trukkos\ nev
```

Mivel a szóköz speciális karakter (hiszen szóhatároló), így csak akkor lehet állománynévben szerepeltetni, ha előtte közönségessé tesszük.



## A shell II.

---

- A sor szavakra tördelésének szemléltetése:

```
echo 1 2 3
```

A három számjegy között eredetileg 5 szóköz állt, a kimenetben mégis csak egy-egy szóközt látunk. Ha valóban 5 szóközt szeretnénk kapni, akkor használjuk az idézőjelet:

```
echo "1 2 3"
```

- A „lista” állományba beleírja az aktuális dátumot és időt, majd az aktuális könyvtár listáját is.

```
date > lista ; ls >> lista
```

Egy ekvivalens megoldás:

```
(date ; ls) > lista
```

Kihasználtuk, hogy a csoport kimenete is átirányítható.

## A shell III.

---

- Kiírja a „A pontos ido: ” szöveget, majd közvetlenül utána ugyabba a sorba az aktuális dátumot és időt.

```
echo -n 'A pontos ido: ' ; date
```

Egy alternatív megoldás a parancs-behelyettesítés használatával:

```
echo -n 'A pontos ido: '`date`
```

- A „123”-at ennél bonyolultabban már nehéz lenne kiírni...

```
echo 1`echo 2`\`echo 3`\``
```

Itt a második és a harmadik jegyet parancs-behelyettesítéssel jelenítjük meg, sőt a külső parancs-behelyettesítés egy újabb, beágyazott parancs-behelyettesítést is tartalmaz.

- A következő parancs azt szemlélteti, hogy a szavakra bontás a parancs-behelyettesítés eredményén is megtörténik:

```
echo `ls -l`
```

# Shell scriptek I.

---

- Most az „`első második`” szöveget írjuk ki, de úgy, hogy először egy környezeti változóban eltároljuk azt.

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

```
SZOVEG="első második"
```

```
echo $SZOVEG
```

Az egyenlőségjel jobb oldalán szereplő szöveget itt mindenképpen idézőjelek vagy aposztrófok közé kell zárni, mert az egyenlőségjel után legfeljebb egyetlen szó állhat!

## Shell scriptek II.

---

- A „h123456” azonosítójú felhasználó bejelentkezéseinek eltárolása a BEJELENTKEZETT környezeti változóba, majd a lista kiírása egymás után kétszer. (Ennek nem sok gyakorlati haszna van, de hasonló esetekben alkalmazható ez a módszer.)

```
#!/bin/bash
BEJELENTKEZETT=`who | grep 'h123456'`
echo "$BEJELENTKEZETT"
echo "$BEJELENTKEZETT"
```

Az idézőjelekre azért van szükség, mert a lista több soros is lehet. Ha az idézőjeleket elhagynánk, akkor a sortörések mentén szavakra tördelné a shell a listát, és így egy szóközökkel tagolt szóhalmazt kapnánk. Az egyenlőségjel jobb oldalan álló kifejezést most nem kötelező idézőjelek közé tenni, mert a parancs-behelyettesítés eredményét egy szónak fogja tekinteni a shell (ami persze szóközöket és sortöréseket is tartalmazhat).

# Shell scriptek III.

---

- Először a „szoveg” állomány sorainak számát eltároljuk a SORSZAM környezeti változóban. Utána kiírjuk a „Sorok:N.” szöveget, ahol N a sorok száma.

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

```
SORSZAM=`cat szoveg | wc -l`
```

```
echo Sorok:$SORSZAM.
```

A sorszám elé egy szóköz is kiíródik, hiszen a `wc` parancs kimenetében a szám előtt szóközök voltak. Ha ettől szeretnénk megszabadulni, akkor a harmadik sort cseréljük le a következőre:

```
echo Sorok:`echo $SORSZAM`.
```

Ebben az esetben a parancs-behelyettesítésen belül a `$SORSZAM` elején levő szóközök eltűnnek, hiszen azok itt egyszerűen a parancsnévnek és a paraméternek (a kiírandó számnak) az elhatárolására szolgálnak.

# Shell scriptek IV.

---

- Tegyük fel, hogy a „`kiir`” állomány a következőket tartalmazza:

```
#!/bin/bash
echo \"$UZENET\"
```

Továbbá tegyük fel, hogy ez az állomány az aktuális könyvtárban található, és a végrehajtási jog legalább a tulajdonos számára engedélyezve van. A következő néhány sor az `export` parancs használatát szemlélteti:

```
#!/bin/bash
unset UZENET
UZENET=Hello
./kiir
export UZENET
./kiir
```

A „`kiir`” script első meghívása során egy üres idézőjelpárt kapunk, míg az `export` parancs utáni lefutásnál a kívánt szöveg is megjelenik. Az `unset` parancs biztosítja, hogy az `UZENET` változó biztosan ne öröklődjön automatikusan.

# Shell scriptek V.

---

- A pozícionális paraméterek, a speciális paraméterek és a `shift` parancs működésének szemléltetése.

```
#!/bin/bash
echo $1, $*, $#
shift
echo $1, $*, $#
echo $0
```

- A script egyetlen paraméterként egy állomány nevét kapja. Feladat, hogy jelenítsük meg az állomány tartalmát!

```
#!/bin/bash
cat $1
```

# Shell scriptek VI.

---

- Mint előbb, de az állomány nevét most ideiglenesen az ALLOMANY változóban tároljuk:

```
#!/bin/bash
ALLOMANY=$1
cat $ALLOMANY
```

- Mint előbb, de ha az első paramétert a felhasználó nem adta meg, akkor alapértelmezésként a „szoveg” állomány tartalma jelenik meg.

```
#!/bin/bash
ALLOMANY=${1:-szoveg}
cat $ALLOMANY
```

- Persze ha az állomány nevére más parancshoz nincs szükségünk, akkor a második és harmadik sor egyetlen sorral helyettesíthető:

```
cat ${1:-szoveg}
```



# Shell scriptek VII.

---

- Az  $1+2$  és  $5/2$  (egész osztás) kifejezések értékének kiírása.

```
#!/bin/bash
expr 1 + 2
expr 5 / 2
```

Egy ekvivalens megoldás az aritmetikai-behelyettesítés használatával:

```
#!/bin/bash
echo $( (1+2) )
echo $( (5/2) )
```

- A „szoveg1” és „szoveg2” állományok összes sorai számának kiírása.

```
#!/bin/bash
SOR1=`cat szoveg1 | wc -l`
SOR2=`cat szoveg2 | wc -l`
echo Osszesen $( ($SOR1+$SOR2) ) sor.
```

Az utolsó sorban a változók neve előtti dollárjelekre nincs feltétlenül szükség (ld. „Aritmetikai kifejezések II.”).

# Shell scriptek VIII.

---

- (Az előbbi példához kapcsolódva.) A `SOR1` változó értékének megnövelése eggyel.

```
SOR1=$((SOR1+1))
```

- (A kettővel korábbi példához kapcsolódva.) Megvizsgálja, hogy melyik említett állomány tartalmaz több sort, és ezt üzenet kiírásával jelzi.

```
if test $SOR1 -gt $SOR2
then
    echo szoveg1 a hosszabb.
else
    echo szoveg2 a hosszabb.
fi
```

Most nem foglalkoztunk azzal az esettel, amikor a két állomány azonos számú sort tartalmazna.

# Shell scriptek IX.

---

- Az „Ures” szöveg kiírása, amennyiben a „szoveg” állomány semmit sem (vagy legfeljebb csak üres sorokat) tartalmazna.

```
#!/bin/bash
if test -z "`cat szoveg`"
then
    echo Ures
fi
```

Az idézőjelekre azért van szükség, mert a -z művelet után legfeljebb csak egy szó állhat.

# Shell scriptek X.

---

- A script hívása során megadott pozicionális paraméterek listájának kiírása. Mindegyik paraméter értéke külön sorban jelenik meg a természetes sorrendben.

```
#!/bin/bash
for p in $*
do
    echo $p
done
```

# Shell scriptek XI.

---

- Mint az előbb, de most mindegyik paraméter értéke elé kiíródik annak sorszáma is.

```
#!/bin/bash
PARSZAM=$#
for ((i=1;i<=PARSZAM;i++))
do
    echo $i. $1
    shift
done
```

A `shift` parancs módosítja a `$#` speciális paraméter értékét, ezért azt a `PARSZAM` környezeti változóban tároljuk el.

# Shell scriptek XII.

---

- Az aktuális könyvtárban levő összes közönséges állomány tartalmának kiírása.

```
#!/bin/bash
for a in *
do
    if test -f $a
    then
        cat $a
    fi
done
```

# Shell scriptek XIII.

---

- A script feladata, hogy 5 másodpercenként írja ki a „Hello” szöveget. A program futása magától nem fejeződik be.

```
#!/bin/bash
for ( ( ; ) )
do
    echo Hello
    sleep 5
done
```

- Az aktuális könyvtárban levő összes olyan bejegyzés kiírása, amelyek neve a „szoveg” szóval kezdődik.

```
#!/bin/bash
for a in szoveg*
do
    echo $a
done
```

# Shell scriptek XIV.

---

- A script egyetlen paraméterként egy könyvtár nevét (elérési útját) kapja. Feladat, hogy írjuk ki a könyvtárban levő bejegyzések nevét az elérési úttal együtt, mindegyiket külön sorba!

```
#!/bin/bash
for a in $1/*
do
    echo $a
done
```

A `for` utasításban az `ls` parancs kimenete is használható (a `-d` opció azért szerepel, hogy az esetleges alkönyvtáraknak csak a neve jelenjen meg, a tartalmuk viszont ne):

```
for a in `ls -d $1/*`
```



## Shell scriptek XV.

---

- Írjuk ki az egész számokat 0-tól 9-ig, mindegyiket külön sorba!

```
#!/bin/bash
for ((i=0;i<10;i++))
do
    echo $i
done
```

- A feladat ugyanaz, mint előbb, de most a `while` utasítást használjuk:

```
#!/bin/bash
i=0
while test $i -lt 10
do
    echo $i
    i=$((i+1))
done
```

# Shell scriptek XVI.

---

- (Az előző példához kapcsolódva.) Tegyük fel, hogy a számsorozatban a 6-ot nem szeretnénk látni!

```
#!/bin/bash
i=0
while test $i -lt 10
do
    if test $i -eq 6
    then
        continue
    fi
    echo $i
    i=$((i+1))
done
```

# Shell scriptek XVII.

- A script először megjelenít egy kérdést, majd egy szöveg begépelésére vár. A válasz a VALASZ környezeti változóba kerül. Most csak az „I” , „I” , „N” és „N” válaszokat fogadjuk el, és kiírjuk a nekik megfelelő döntést. Más válasz esetén kiírjuk a „Hiba!” üzenetet, majd kilépünk 1-es hibakóddal (kilépési státusszal).

```
#!/bin/bash
echo -n "Biztos benne? "
read VALASZ
case $VALASZ in
    i|I) echo Igen;;
    [nN]) echo Nem;;
    *)
        echo Hiba!
        exit 1;;
esac
```

# Reguláris kifejezések és a grep szűrő I.

- Az aktuális könyvtár közönséges állományainak hosszú (bővített) kilistázása.

```
ls -l | egrep '^-'
```

Kihasnáltuk, hogy a könyvtárbejegyzések típusát a sor első karaktere jelzi, és ez közönséges állományok esetén egy mínuszjel.

- A tulajdonos által futtatható közönséges állományok hosszú kilistázása az aktuális könyvtárban.

```
ls -l | egrep '^-..x'
```

Kihasnáltuk, hogy a tulajdonos hozzáférési jogait a 2. (olvasási jog), 3. (írási jog) és 4. (végrehajtási jog) karakterek mutatják, és hogy a futási jogot az „x” betű jelzi. A sor második és harmadik karaktere számunkra most érdektelen, ezért ott tetszőleges karakter illesztését megengedjük.

- A „h0” szóval kezdődő felhasználói (hallgatói) azonosítók kilistázása.

```
ls -l /home | egrep '^h0'
```

# Reguláris kifejezések és a grep szűrő II.

- Az aktuális könyvtárban található alkönyvtárak számának kiírása.

```
file * | egrep ' : directory$' | wc -l
```

Könyvtárak esetén a `file` parancs kimenetében a nevet egy kettőspont és egy szóköz után csak a „`directory`” szó követi, így zárva le a sort.

- A „`szoveg`” állomány mindazon sorainak megjelenítése, amelyek tartalmaznak (legalább) egy „`a`” betűt (és esetleg azon kívül még bármi egyebet is).

```
egrep ' a' szoveg
```

- Mint előbb, de az „`A`” betűt is találatnak tekintti.

```
egrep -i ' a' szoveg
```

- Azokat a sorokat jeleníti meg, amelyek nem tartalmaznak „`a`” betűt.

```
egrep -v ' a' szoveg
```

- Csak azok a sorok íródnak ki, amelyek tartalmazzák magát az „`a`” szót is.

```
egrep -w ' a' szoveg
```

## Reguláris kifejezések és a grep szűrő III.

---

- Az olyan sorok megjelenítése, amelyek tartalmaznak (legalább) egy nagybetűt (és esetleg azon kívül még bármi egyebet is).

`egrep '[A-Z]'` szoveg

Az előzővel ekvivalens megoldás, de most hangsúlyozzuk az ismétlésszámot:

`egrep '[A-Z]{1}'` szoveg

A következő megoldás is ekvivalens az előzőkkel, hiszen a nagybetűt tetszőleges karakterek bármilyen hosszú sorozata ( `*` ) követheti:

`egrep '[A-Z].*'` szoveg

Egy újabb ekvivalens megoldás, szintén az ismétlésszám megadásával:

`egrep '[A-Z]+'` szoveg

# Reguláris kifejezések és a grep szűrő IV.

- Az olyan sorok megjelenítése, amelyek kizárólag egyetlen karaktert tartalmaznak, mégpedig egy nagybetűt.  
egrep ' ^ [A-Z] \$ ' szoveg
- Az olyan sorok kiírása, amelyek egymás mellett (legalább) 2 nagybetűt tartalmaznak (és rajtuk kívül esetleg bármi egyebet is).

egrep ' [A-Z] [A-Z] ' szoveg

Az alábbi megoldás jelentése megegyezik az előzővel, de most a karakterek számát ismétlésszámmal adtuk meg:

egrep ' [A-Z] {2} ' szoveg

- Az olyan sorok kiírása, amelyek valahol tartalmaznak egy nagybetűt, amit egy szóköz és egy újabb nagybetű követ.

egrep ' [A-Z] [A-Z] ' szoveg

# Reguláris kifejezések és a grep szűrő V.

- Az olyan sorok megjelenítése, amelyek tartalmaznak legalább egy, nagybetűtől eltérő karaktert.  
egrep '[^A-Z]' szoveg
- Azoknak a soroknak a megjelenítése, amelyek tartalmaznak egy nagybetűt, majd valahol később tartalmaznak egy számjegyet is.  
egrep '[A-Z] . \* [0-9]' szoveg
- Azoknak a soroknak a megjelenítése, amelyekben előfordul egy ponttal lezárt ( \ . ) számjegysorozat. A sorozat hossza legalább 1.  
egrep '[0-9]+\.' szoveg
- Azoknak a nem üres soroknak a megjelenítése, amelyek csak kisbetűket, nagybetűket, számjegyeket és szóközt tartalmaznak.  
egrep '^[ A-Za-z0-9 ]+\$' szoveg



# Reguláris kifejezések és a grep szűrő VI.

- A pontosan 1 számjegyet tartalmazó sorok kiírása. A számjegy a sorban bárhol előfordulhat, és rajta kívül bármi más is tartalmazhat.

```
egrep '^[^0-9]*[0-9][^0-9]*$' szoveg
```

- Az olyan sorok megjelenítése, amelyek legalább három olyan szóval kezdődnek, amelyek első betűje „a”, ezt a „bc” vagy „de” páros követi, majd egy „f” és „g” betűtől eltérő karakter zárja.

```
egrep '^(a(bc|de)[^fg]){3,}' szoveg
```

A csoportosításra két ok miatt is szükség van: egyrészt a konkatenáció erősebb az alternációnál (a (bc|de) kifejezés előtt és után is konkatenáció áll), másrészt az iteráció erősebb mindkettőnél (a külső zárójelek elhagyása esetén az ismétlésszám csak a [^fg] kifejezésre vonatkozna).

# Az `awk` szűrő I.

- Az aktuális könyvtár olyan bejegyzéseinek hosszú kilistázása, amelyeknek tulajdonosa a „h012345” azonosítójú felhasználó.

```
ls -l | awk ' $3 == "h012345" { print } '
```

Az `ls -l` kimenetében a 3. oszlop (mező) tartalmazza a tulajdonos felhasználó azonosítóját. A paraméter nélküli `print` az egész sor (rekord) tartalmát kiírja.

- Az aktuális könyvtár augusztusi keltezésű alkönyvtárai nevének kiírása, mindegyiket külön sorba.

```
ls -l | awk ' /^d/ && ($6 == "Aug") { print $9 } '
```

Az alkönyvtárak típusát a sor elején álló „d” jelzi. A dátum hónapja a hatodik, míg a bejegyzés neve a kilencedik oszlopban (mezőben) található.

## Az awk szűrő II.

---

- Az aktuális könyvtár olyan állományai nevének kiírása, amelyek mérete nagyobb, mint 100 bájt, és a nevük tartalmaz „b” betűt. Mindegyik nevet külön sorba írjuk ki.

```
ls -l | awk ' ($5 > 100) && ($9 ~ /b/) { print $9 } '
```

A méretet az ötödik oszlop (mező) tárolja. A minta második rész kifejezése csak akkor teljesül egy sorra (rekordra), ha az állománynév illeszkedik a b reguláris kifejezésre (azaz tartalmaz „b” betűt).

- Az üres – pontosabban legfeljebb csak szóközöket és tabulátort tartalmazó – sorok kihagyása a bemenetből.

```
NF > 0
```

Ne feledjük, hogy a hiányzó akció ekvivalens a `{ print }` akcióval.

## Az awk szűrő III.

---

- Kiírja, hogy mennyi sorból állt a bemenet.

```
END { print NR }
```

Emlékezzünk vissza, hogy az END mintájú szabály akciója csak a bemenet feldolgozása után fut le.

- A bemenet páratlan sorszámú sorainak kiszűrése, azaz csak a páros sorszámú sorok jelennek meg.

```
NR % 2 == 0
```

- A bemenet minden sora elé annak sorszámát is kiírjuk. Több bemeneti állomány esetén a sorszám folytatólagos lesz.

```
{ print NR, $0 }
```

Kihasználtuk, hogy a minta elhagyása esetén az illető akció minden rekordra (most sorra) végrehajtható. Az aktuális sor (rekord) tartalmát \$0 tartalmazza.

## Az awk szűrő IV.

- A bemenet soraiban felcseréli az első két szót.

```
{ szo = $1; $1 = $2; $2 = szo; print }
```

Az első szó (mező) tartalmát ideiglenesen a szo változóba tároltuk el. Vigyázzunk, mert a mezők értékének megváltoztatásakor a \$0 értéke újraszámítódik (ld. egy későbbi példában)!

- A 10 karakternél hosszabb sorok kiírása.

```
length($0) > 10
```

- Minden sorból csak az utolsó 2 szó kiírása.

```
NF < 2
```

```
NF >= 2 { print $(NF - 1), $NF }
```

Mindenképpen meg kell vizsgálni, hogy a sor legalább 2 szót (mezőt) tartalmaz-e, különben a mezőhivatkozás érvénytelen lehetne, ill. a \$0 esetén az egész sort (rekordot) megjelenítenénk.

# AZ awk SZŰRŐ V.

- A bemenet minden sorát csupa nagybetűssé varázsolja.  

```
{ print toupper($0) }
```
- A bemenet minden szavát külön sorba írja ki.  

```
{ for (i = 1; i <= NF; i++) print $i }
```
- Most a bemenet szavait megszakítás nélkül ugyanabba a sorba írjuk ki úgy, hogy a szavak közé még szóközt sem teszünk.

```
{ for (i = 1; i <= NF; i++) printf "%s", $i }  
END { printf "\n" }
```

A kiíratáshoz a `printf` utasítást használtuk, hogy elkerüljük a sortörés kiírását. A formátumban a `%s` egy szöveges paramétert (pontosabban szöveges konverziót) jelöl. A bemenet vége után a `rend` kedvéért egy sortörést (`\n`) is megjelenítünk. (Az utóbbi tevékenység helyettesíthető lenne a `print ""` utasítással is.)

## Az `awk` szűrő VI.

---

- Előfordulási (gyakorisági) statisztika készítése a bemenet szavairól. A bemenet feldolgozása után megjeleníti a begyűjtött adatokat: kiírja a szavakat és azok gyakoriságát, mindegyik szó-szám párost külön sorba.

```
{ for (i = 1; i <= NF; i++) stat[$i]++ }  
END { for (szo in stat) print szo, stat[szo] }
```

A `stat` változó egy (asszociatív) tömb, amelynek elemeit a megtalált szavakkal (mezőkkel) indexeljük, és a `stat[szo]` elem értéke a szó előfordulásainak a száma.

## Az awk szűrő VII.

- Tegyük fel, hogy a bemenet egy olyan szöveges állomány, amelynek minden sora azonos felépítésű. A sorok két oszlopot tartalmaznak: egy hallgató nevét, illetve egy osztályzatot (1–5). A két oszlopot kivételesen nem szóközzel választjuk el egymástól, hanem egy kettősponttal, hiszen maga a név is tartalmazhat szóközt. A feladat, hogy írjuk ki a hallgatók nevét és jegyüket, és közben számítsuk ki a jegyek átlagát, amit szintén ki kell írni a végén. A név és a jegy ugyanabba a sorba kerüljön! Feltesszük, hogy a bemenetben nem fordul elő üres sor, továbbá hogy legalább 1 sorból áll.

```
BEGIN { FS = ":" }  
      { print $1,$2; atlag += $2 }  
END   { print atlag / NR }
```

A mezőhatároló karaktert az FS változó tárolja, így először ennek az értékét állítjuk át az alapértelmezett szóközről. Fontos, hogy ezt még a bemenő adatok feldolgozásának megkezdése előtt kell megtenni, így a BEGIN minta akciójába tettük.



## Az awk szűrő VIII.

- A bemenet minden sora előtt feltünteti a sor számát és a sorban levő szavak számát. A két számot egymástól ill. a sor tartalmától a „@” karakter választja el.

```
BEGIN { OFS = "@" }  
{ print NR, NF, $0 }
```

Kihasználjuk, hogy a `print` utasítás a kiírt értékek közé az `OFS` változó tartalmát (ami egyébként tetszőleges szöveg lehet) szúrja be.

- Kiírja a bemenet sorait úgy, hogy a szavakat a „@” karakter tagolja, a sorok végére pedig a „<sortvege>” szöveget szúrja be a sortörés elé.

```
BEGIN { OFS = "@"; ORS = "<sortvege>\n" }  
{ $1 = $1; print }
```

A `$1 = $1` értékadásnak „csak” annyi a következménye, hogy a `$0` tartalma újraszámítódik oly módon, hogy a mezőket (most a szavakat) az `OFS` értéke fogja tagolni. A `print` utasítás a kiírást mindig az `ORS` tartalmának megjelenítésével zárja.

*Vége*