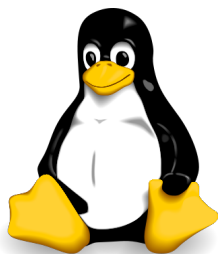


# OPERÁCIÓS RENDSZEREK gyakorlati jegyzet

Összeállította: *Griechisch Erika*

Utoljára frissítve: 2019. március 13.



debian



ubuntu



mandriva



fedora

© A képek forrása:

<http://tatice.deviantart.com/art/Operating-Systems-affiliates-80146648>

# Tartalomjegyzék

<b>1. Alapok</b>	<b>1</b>
1.1. Terminálok	1
1.2. Alapparancsok	3
1.3. basename, dirname	7
1.4. Mintaillesztés	8
1.5. Keresés	9
1.6. Állománynév-kiegészítés	11
<b>2. Jogosultságok, csatornák, szöveges fájlok, felhasználók</b>	<b>11</b>
2.1. Jogosultságok	11
2.2. Standard csatornák és az átirányítás	16
2.3.  , a csővezeték	16
2.4. tee	17
2.5. Szöveg kiírása, szöveges fájlok kezelése	19
2.6. Felhasználókkal kapcsolatos parancsok	23
<b>3. Linkelés, archiválás, csere</b>	<b>25</b>
3.1. Linkelés	25
3.2. Archiválás, tömörítés	26
3.3. Fájlok összehasonlítása	27
3.4. Csere	28
<b>4. BASH</b>	<b>31</b>
4.1. Alapok	31
4.2. A környezeti változók	35
4.3. Parancssori paraméterek	38
4.4. Matematikai kifejezések	39
4.5. for	40
4.6. if	43
4.7. Logikai műveletek	44
4.8. Logikai állandók	46
4.9. A test program	46
4.10. while	49
4.11. do-until	51
4.12. case	52
4.13. Vágókifejezések	52
4.14. Függvények	53
4.15. Tömb	55

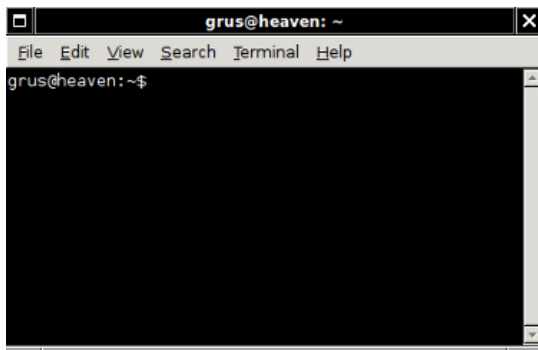
<b>5. Reguláris kifejezések</b>	<b>56</b>
5.1. Elemi kifejezések	57
5.2. Összetett kifejezések	57
5.3. grep (ismét)	58
5.4. Példák	59
<b>6. AWK</b>	<b>61</b>
6.1. Forrásprogram felépítése	61
6.2. Minták	62
6.3. Konstansok	64
6.4. Változók	64
6.5. Beépített változók	65
6.6. Mezők	66
6.7. Tömbök	69
6.8. Kifejezések felépítése	69
6.9. Vezérlési szerkezetek	72
6.10. Összetett példák	73
<b>7. Irodalom</b>	<b>75</b>

# 1. Alapok

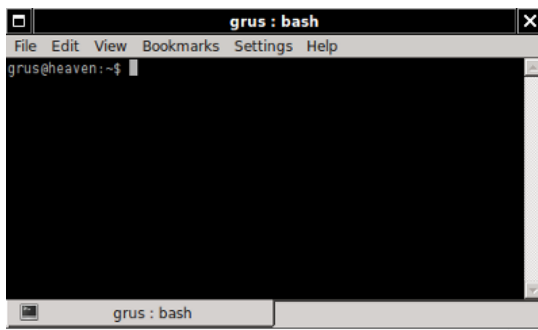
## 1.1. Terminálok



xterm



gnome-terminal



konsole

## 1.2. Alapparancsok

**Segítség** – `man`, `info`

`man` - formázza és kiírja az on-line kézikönyvlapokat

Példa: `man man` (kilépés: `q`), `man ls` (lásd alább)

```
NAME
    ls - list directory contents
SYNOPSIS
    ls [OPTION]... [FILE]...
DESCRIPTION
```

**Aktuális könyvtár** – `pwd`, `cd`

`pwd` - kiírja az aktuális (munka-) könyvtárat

`pwd`

`cd` - az aktuális könyvtár megváltoztatása

`cd` könyvtár

```
joe@localhost:~$ pwd
/home/joe
joe@localhost:~$ cd Documents/
joe@localhost:~/Documents$ pwd
/home/joe/Documents
```

**Létrehozás** – `mkdir`, `touch`

`mkdir` - könyvtár létrehozása

`mkdir` [ kapcsolók ]

könyvtár

A `mkdir` Létrehozza a megadott nevű könyvtár(ak)at.

*Kapcsolók*

**-p** Létrehozza a szülőkönyvtárakat is

**-m jog** A megadott hozzáférési joggal hozza létre a könyvtárakat (később majd ezekről lesz szó)

**touch - fájl időbélyegének megváltoztatása**

touch

`[-acm] [-r reffájl] [-t idő ] fájl`

A touch megváltoztatja a megadott fájl(ok) utolsó elérésének és/vagy utolsó módosításának idejét. Ha a fájl nem létezik, a touch létrehozza.

*Kapcsolók*

**-c, --no-create** Nem hozza létre a fájlokat, ha nem léteznek

**-d, --date= idő** Az idő argumentumot használja az aktuális idő helyett. Ebben lehetnek hónapnevek, időzóna, am vagy pm, stb.

```
joe@localhost:~/Documents$ ls -lh
total 4.0K
-rw-r--r-- 1 joe joe 827  2011-01-24 01:06 Vicc.txt
joe@localhost:~/Documents$ touch Vicc.txt
joe@localhost:~/Documents$ ls -lh
total 4.0K
-rw-r--r-- 1 joe joe 827  2011-pics-05 11:44 Vicc.txt
joe@localhost:~/Documents$ mkdir -p newdir/01/pics
joe@localhost:~/Documents$ ls newdir
01
joe@localhost:~/Documents$ ls newdir/01/
pics
```

**Másolás, áthelyezés – cp, mv**

**cp - fileok másolása**

cp [kapcsolók] forrás cél

*Kapcsolók*

**-f (force)** A létező célfájlok törlése

**-i (interactive)** A felhasználó megkérdezése arról, hogy felülírhatók-e a létező célfájlok

**-r, -R (recursive)** A könyvtárak rekurzív másolása, A nem-könyvtár fájlokat reguláris fájlként másolja

- u (update) Nem másolja azokat a nem-könyvtár fájlokat, amelyeknek azonos vagy újabb módosítási idővel rendelkező célfájla létezik

**mv - fileok mozgatása/áthelyezése**

cél

mv [kapcsolók] forrás

*Kapcsolók*

- f A létező célfájlok törlése kérdés nélkül
- i A felhasználó megkérdezése arról, hogy felülírhatók-e a létező célfájlok
- u Nem mozgatja azokat a nem-könyvtár fájlokat, amelyeknek azonos vagy újabb módosítási idővel rendelkező célfájla létezik

**Törlés – rm, rmdir**

**rm - állományok eltávolítása**

rm [ kapcsolók ] fájl(ok)

*Kapcsolók*

- f Figyelman kívül hagyja a nem létező állományokat és nem kérdezi meg a felhasználót
- i Minden fájl eltávolítása előtt megkérdezi a felhasználót, hogy törölheti-e az adott állományt
- r, -R A könyvtárak tartalmát rekurzívan törli
- v (verbose) Kiírja minden fájl nevét mielőtt törölné

**rmdir - törli az üres könyvtárakat**

könyvtár(ak)

rmdir [ kapcsolók ]

*Kapcsolók*

- p a szülőkönyvtárakat is törli



## Kiírás – cat, less, more

**cat** - fájlokat fűz össze és kiírja a szabványos kimenetre

cat [ kapcsolók ] [ file(ok) ]

A cat program minden argumentumként megadott fájlt a szabványos kimenetre ír. Amennyiben nincs fájlnev megadva, vagy a megadott fájlnev a '-'-jel, a szabványos bemenetet olvassa.

**more**

more [ kapcsolók ] [ -méret ] [ +/- minta ] [ +kezdősor ] [ file(ok) ]

**less**

A more egy egyszerű szűrőprogram, egy adott szövegből csak egy képernyőnyit mutat. A less egy sok új és hasznos szolgáltatást nyújtó more-emuláció.

```
cat fajl.txt
less fajl.txt
more fajl.txt
cat fajl1.txt fajl2.txt
```

## Filetípus

**file** - filetype meghatározása

file [ kapcsolók ] [ -f lista ] [ -m bővísfájl ] fájl(ok)

A file parancs teszteli minden argumentumát és megpróbálja kategorizálni ezeket.

```
joe@localhost:~/Documents$ ls
Kep001.jpg  Logo.png  newdir    SzegedTreebank.pdf  Vicc.
txt
joe@localhost:~/Documents$ file *
Kep001.jpg:      JPEG image data, JFIF standard
                1.01, comment: "GIMP"
Logo.png:        PNG image, 180 x 120, 8-bit/color
                RGBA, non-interlaced
newdir:          directory
SzegedTreebank.pdf: PDF document, version 1.4
```

```
Vicc.txt:          UTF-8 Unicode text, with very long
lines
```

## Fileok listázása

**ls - könyvtár tartalmának listázása**      `ls [ kapcsoló(k) ] [ file(ok) ]`

*Kapcsolók*

- l** (egy) minden sorban csak egy név látszik (egyszlopos mód)
- l** (kis L) hosszú avagy bővített lista
- a** a listában a rejtett állományok/könyvtárak is megjelennek  
*Megjegyzés:* rejtett állományok a . (pont)-tal kezdődőek
- R** a megadott könyvtár(ak) minden alkönyvtárának és azok teljes tartalmának listázása (rekurzív listázás)
- h** olvashatóbb formában írja ki a fileok méreteit

## 1.3. basename, dirname

**basename útvonal**

A könyvtárak neveit eltávolítja a megadott útvonalból (csak az utolsó / utáni állománynév marad meg), majd kiírja az eredményt. Nem ellenőrzi az útvonal valóságát!

**dirname útvonal**

Az állomány nevét eltávolítja a megadott útvonalból (csak az utolsó / előtt álló könyvtárak listája marad meg), majd kiírja az eredményt. Ha az útvonal nem tartalmaz / jelet, az eredmény a . lesz. Nem ellenőrzi az útvonal valóságát!

```
joe@localhost:~$ dirname /usr/bin/nemletezofilenev
/usr/bin
joe@localhost:~$ basename /usr/bin/nemletezofilenev
nemletezofilenev
```

## 1.4. Mintaillesztés

Hasonló felépítésű állomány- vagy könyvtárnevek listájának megadására használhatunk ún. *állománynév mintákat* (filename pattern). Ezek a közönséges karakterek mellett helyettesítő, mintaillesztő avagy Joker-karaktereket is tartalmaznak.

*Eredmény:* a mintának megfelelő (mintára illeszkedő) létező nevek szóközzel tagolt rendezett listája

### Mintaillesztő karakterek

\* tetszőleges karakterekből álló, tetszőlegesen hosszú szó (üres szó is)

? egyetlen tetszőleges karakter

[HALMAZ] A halmaz bármely karakterének egy példánya. A halmazt a karakterek egymás mellé írásával adhatjuk meg.

[ELSŐ-UTOLSÓ] mint előbb, de itt egy tartományt adunk meg

[^HALMAZ] a halmazban nem szereplő bármely karakter egy példánya

### Speciális esetek

Mindig ki kell írni a rejtett állományok/könyvtárak nevének kezdő pont (.) karakterét, ill. könyvtárak esetén a könyvtárnév után a / jelet.

A pont karakter egyéb esetekben nem számít speciálisnak. Néhány program azonban az állománynevekben az utolsó pont utáni részt, az ún. *kiterjesztést* (filename extension) különlegesen kezeli. Ezt általában az állomány tartalma típusának jelzésére használják (pl. kép, video, hang).

#### Példák

\* az összes nem rejtett állomány és alkönyvtár

\*/ az összes nem rejtett alkönyvtár

\*/\* az összes nem rejtett alkönyvtár teljes tartalma

.\* az összes rejtett állomány és alkönyvtár

.\*/\* az összes rejtett alkönyvtár

**\*.jpg** a .jpg kiterjesztésű állományok (JPEG formátumú képek)

**\*.\*** az összes nem rejtett állomány és alkönyvtár, amelynek neve tartalmaz legalább egy pontot

```
joe@localhost:~/dir$ ls -a
.                ebay_fanshop.png  hello.sh
.               pepita_sakk.png  ubigraph1.png
..              file.txt          Judy.png        .rejtett1
.               ubigraph2.png
BatteryLinux.png final.png          Logo.png        .rejtett
joe@localhost:~/dir$ ls *
BatteryLinux.png file.txt          hello.sh        Logo.png
.               ubigraph1.png
ebay_fanshop.png final.png        Judy.png        pepita_sakk.png
.               ubigraph2.png
joe@localhost:~/dir$ ls ?ello.sh
hello.sh
joe@localhost:~/dir$ ls ubigraph[12].png
ubigraph1.png  ubigraph2.png
joe@localhost:~/dir$ ls ubigraph[0-9].png
ubigraph1.png  ubigraph2.png
joe@localhost:~/dir$ ls .*
.rejtett1      .rejtett2
joe@localhost:~/dir$ ls *.png
BatteryLinux.png  final.png  Logo.png          ubigraph1
.png
ebay_fanshop.png  Judy.png   pepita_sakk.png   ubigraph2
.png
```

## 1.5. Keresés

**locate – mintához illeszkedő fájlokat nyomtat a fájlnev adatbázis(ok)ból**

```
locate [ -d elérési út ] [ -database= elérési út ] [ -version ] [ -help ] minta...
```

A locate parancs végignézi a megadott fájlnev-adatbázis(oka)t és ki-nyomtatja azokat a fájlneveket, melyek illeszkednek a mintá(k)ra. A minták tartalmazhatnak shell-stílusú speciális karaktereket is (metakarakterek). Ezek a: '\*', '?', és '['. A metakarakterek nem kezelik a '/'

vagy '.' karaktereket speciálisan, emiatt például a 'foo\*bar' minta illeszkedik a 'foo3/bar' karaktersort tartalmazó fájlnevre, hasonlóan a '\*duck\*' minta is illeszkedik a 'lake/.ducky' karaktersort tartalmazó fájlnevekre. A metakaraktereket tartalmazó mintákat idézőjelek közé kell tenni jelezve, hogy azok nem a parancsértelmezőnek (shell) szólnak.

## find – fájlokat keres egy könyvtárstruktúrában

`find [útvonal...] [kifejezés]`

A find parancs rengeteg kapcsolóval rendelkezik, emellett operátorokat is használhatunk vele, részletesebben lásd man find.

Álljon itt néhány példa a teljesség igénye nélkül<sup>1</sup>. Keressünk...

- .jpg fájlokat az aktuális könyvtárakban így: `find . -name *.jpg`
- 20 évnél idősebb állományokat keresünk az aktuális könyvtárban így:  
`find ./ -mtime +7300`
- az utolsó 3 napban módosított állományokat így: `find . -mtime -3.`
- az utolsó 3 napban módosított txt állományokat így: `find . -name '*.txt' -mtime -3`
- 10000 kbytenál nagyobb állományokat így: `find . -size +10000k`
- rc.conf nevű állomány keresése az aktuális könyvtárban

```
find . -name "rc.conf" -print
```

- ha megtalálta a find az rc.conf nevű állományt akkor azon végre hajtja a chmod utasítást

```
find . -name "rc.conf" -exec chmod o+r '{}' \;
```

- komplex keresés ami kihagyja az eredményből a \*.v vagy \*.v nevű állományokat. Egy kis extra magyarázat: -not negálást jelent, -o a logikai OR műveletet, \ ( a logikai művelet kezdetét jelöli, \) pedig a végét. Egyébként a kihagyott állományok egy verzió kezelő állományai...

<sup>1</sup>A legtöbb példa a <http://linuxbox.hu/find> oldalról származik

```
find /usr/src -not \( -name "*.v" -o -name ".*,v" \) '{}' \; -print
```

- keresés a linuxbox szóra a \*.html nevű állományokban, állomány név kiírás találat esetén.



```
find . -name "*.html" -exec grep "linuxbox" '{}' \; -print
```

- Keresés az aktuális könyvtárból indulva kis és nagybetű nem figyelembe vételével és kihagyni a .svn nevű könyvtárak tartalmát:

```
find . -iname "*old*" -a -not -path "*.svn*" -print
```

Itt igazából az -and -or -not keresési feltételek közötti logikai művelet megadás lehetősége a lényeg!

## 1.6. Állománynév-kiegészítés

A hosszabb nevek begépelését könnyíti meg az *állománynév-kiegészítés* (filename completion). A név első pár betűjének beírása után üssük le a  <sup>2</sup> billentyűt. Ha csak egy állomány neve kezdődik így, akkor a név kiegészül. Különben még egyszer üssük le a  -ot, hogy egy listát kapjunk a szóba jöhető nevekről. Ezután folytassuk a gépelést a kívánt karakterrel. Ez a szolgáltatás könyvtár- és programneveknél is működik.

## 2. Jogosultságok, csatornák, szöveges fájlok, felhasználók

### 2.1. Jogosultságok

- minden állománynak van tulajdonosa és csoportja
- mindezekhez tartozik

r olvasási jog

---

<sup>2</sup>Tab

Ha a tulajdonosnak az olvasási jogát jelző kapcsoló be van kapcsolva, akkor az adott állományban található adatokat a tulajdonos megnézheti, olvashatja, kinyomtathatja, lemásolhatja, egy szóval az adatokhoz hozzáférhet. Ha e kapcsoló ki van kapcsolva, akkor az állomány nevét látja ugyan a tulajdonos, de a tartalmához nem férhet hozzá.

A *könyvtáron értelmezett olvasási jog* a benne található állományok nevének kilistázását jelenti. Ha ezzel a joggal rendelkezik valaki akkor listázhatja az állományokat az adott könyvtárban, ha nem, akkor ezt nem teheti meg. Ötletes módon, az állományokat esetleg létrehozhatja, és akár módosíthatja is, attól függetlenül, hogy róluk listát nem kaphat.

## w írási jog

A tulajdonos írási jog kapcsolója megmutatja, hogy a számára engedélyezett-e az adatok írása, vagyis módosítása. Ha ez be van kapcsolva, akkor a tulajdonos módosíthatja, bővítheti és törölheti az adott állományban található adatokat, sőt törölheti akár az egész állományt. Ha e kapcsoló ki van kapcsolva, akkor a tulajdonos nem módosíthatja az adatokat, még az állomány végére sem fűzhet újabb információkat.

A *könyvtáron értelmezett írási jog* újabb fájlok létrehozására és törlésére ad lehetőséget. Ha írási joggal nem rendelkezik valaki az adott könyvtárban, esetleg még írhat a benne levő állományokba vagy azokból adatokat törölhet. Egész állományokat nem hozhat létre és egész állományokat viszont nem törölhet. Amennyiben a könyvtárból az ott található állományokat nem tudja a felhasználó kitörölni, a könyvtárat magát sem tudja törölni, hiszen csak üres könyvtárakat lehet kitörölni. Ha a könyvtár már üres, a felhasználónak nincs szüksége írási jogra a könyvtárra nézve ahhoz, hogy azt kitörölhesse.

## x futtatási jog

A tulajdonos futtatási jogát jelzőkapcsoló azt mutatja meg, hogy a tulajdonos elindíthatja-e az adott állományt. Ennek a lehetőségnek csak programok esetében van értelme.

A *könyvtáron értelmezett futtatási jog* a könyvtár megnyitását jelenti, vagyis azt a képességet, hogy a könyvtárba belépjünk.

- fájl(ok) futtatásához és mappa megnyitáshoz is rx (olvasási és futtatási) jog szükséges

## chmod - fájlok elérési jogainak megváltoztatása

chmod

+|-<mód> <fájlnev>

Egy fájl tulajdonosi (hozzáférési) jogait csak a fájl tulajdonosa, vagy a rendszergazda tudja megváltoztatni. A továbbiakban tegyük fel, hogy a `pelda.txt` file tulajdonosa mi vagyunk.

**+/-:** ezzel jelezzük, hogy adunk vagy elveszünk jogot.

**mód** : két dolgot kell ezesetben meghatározunk

*kinek adunk*

**u** pl. `chmod u+w pelda.txt` - saját magunknak írási jog

**g** pl. `chmod g+r pelda.txt` - csoportnak olvasási jog

**o** pl. `chmod o+x pelda.txt` - másoknak futtatási jog

**a** (all – mindenki) `chmod a+rw pelda.txt` - mindenkinek megadjuk az olvasási, írási és futtatási jogot

*...és milyen jogot*

**r,w,x** (lásd feljebb)

Lehet kombinálni is a fentieket, pl. `$chmod ug+w pelda.txt`

Ha a csoportot – akire az adott jog vonatkozik – nem adjuk meg, akkor az mindhárom csoportra vonatkozik:

```
joe@localhost:~$ ls -l fajl.txt
-rw-r--r-- 1 joe joe 178 2011-02-11 18:45 fajl.txt
joe@localhost:~$ chmod +x fajl.txt
joe@localhost:~$ ls -l fajl.txt
-rwxr-xr-x 1 joe joe 178 2011-02-11 18:45 fajl.txt
```

Ha nem a + vagy - jeleket használjuk, hanem az = jelet, akkor azok a jogok kapcsolódnak be, amelyeket felsorolunk, a többi pedig visszavonásra kerül:



```
joe@localhost:~$ ls -l fajl.txt
-rwxr--r-- 1 joe joe 178 2011-02-11 18:45 fajl.txt
joe@localhost:~$ chmod u=rw fajl.txt
joe@localhost:~$ ls -l fajl.txt
-rw-r--r-- 1 joe joe 178 2011-02-11 18:45 fajl.txt
```

A jogosultságok megváltoztatásakor nem csak a fenti mód használható, hanem számszerű formátumban is megadhatjuk a jogokat.

A jobb oldalon látható táblázat adja meg melyik joghoz milyen szám tartozik. Ha több jogot szeretnénk alkalmazni, akkor össze kell adnunk az értékeket, például az 5 az olvasási és futtatási jogot jelenti, a 6 az olvasást és írást.

Jogok	<b>u</b> tulajdonos	<b>g</b> csoporthoz	<b>o</b> mindenki más
<b>Olvasás</b>	r	r	r
<b>Írás</b>	w	w	w
<b>Futtatás</b>	x	x	x

---

Jogok	<b>u</b> tulajdonos	<b>g</b> csoporthoz	<b>o</b> mindenki más
<b>Olvasás</b>	4	4	4
<b>Írás</b>	2	2	2
<b>Futtatás</b>	1	1	1

*Példák* Az alábbi táblázatban az egymás mellett levő parancsok ekvivalensek.

chmod u=rw,g=r,o=r fajl.txt	chmod 644 fajl.txt
chmod ugo=rw fajl.txt	chmod 666 fajl.txt
chmod a=rwx fajl.txt	chmod 777 fajl.txt
chmod ugo-rwx fajl.txt	chmod 000 fajl.txt

## chown - fájlok felhasználói és csoport tulajdonosának megváltoztatása

```
chown [ kapcsolók ] [ tulajdonos ][:[ csoport ]] file(ok)
```

A `chown` (change owner, tulajdonosváltás) parancs segítségével megváltoztathatjuk az állomány tulajdonosát és csoporttulajdonosát. Alapesetben – ha csak a tulajdonost változtatjuk – a parancsnak a tulajdonos nevét és az állománynevet kell megadnunk:

```
joe@localhost:~$ ls -l fajl.txt
-rw-r--r-- 1 joe      joe 178 2011-02-11 18:45 fajl.txt
joe@localhost:~$ chown andrea fajl.txt
joe@localhost:~$ ls -l fajl.txt
-rw-r--r-- 1 andrea   joe 178 2011-02-11 18:45 fajl.txt
```

Ha a csoporttulajdonost is meg akarjuk változtatni, akkor a tulajdonos neve után kettősponttal elválasztva kell az új csoportot megadnunk:

```
joe@localhost:~$ chown joe:csoportnev fajl.txt
joe@localhost:~$ ls -l fajl.txt
-rw-r--r-- 1 joe csoportnev 178 2011-02-11 18:45 fajl.
txt
```

### Kapcsolók

**-R** rekurzívan változtatja meg a (paraméterként megadott) könyvtár és annak tartalmának tulajdonosát

### chgrp - fájlok tulajdonosi csoportjának megváltoztatása

A `chgrp` (change group ownership, csoporttulajdonos megváltoztatása) paranccsal megváltoztathatjuk az állományok és könyvtárak csoport tulajdonosát. Erre alkalmas a `chown` parancs is, ha azonban csak a csoporttulajdonost kívánjuk megváltoztatni (a tulajdonost nem) akkor erre a feladatra a `chgrp` használható.

```
joe@localhost:~$ chgrp csoportnev fajl.txt
joe@localhost:~$ ls -l fajl.txt
-rw-r--r-- 1 joe csoportnev 178 2011-02-11 18:45 fajl.
txt
```

## 2.2. Standard csatornák és az átirányítás

A Unixban 3 standard be- és kimeneti csatorna van definiálva:

**standard bemenet** (stdin, 0)

**standard kimenet** (stdout, 1)

**standard hibacsatorna** (stderr, 2)

Alapértelmezésben a standard bemenet a billentyűzet, a kimenet és a hibacsatorna pedig a képernyő (többnyire a terminál). Az átirányítás lényege, hogy a programot utasíthassunk arra, hogy a bemenetet ne a billentyűzetről várja, illetve eredményeit ne a képernyőre (terminálba) írja ki.

**bemenet átirányítása** < például: `cat < fajl.txt`

**kimenet átirányítása** > például: `cat fajl.txt > file2.txt`

A `fajl.txt` tartalmát a `file2.txt`-be irányítjuk.

A fenti parancs tulajdonképpen egyenértékű a `cp fajl.txt file2.txt` paranccsal

**hibacsatorna átirányítása** 2> például: `du -h / 2>/dev/null`

„Speciális” átirányítások:

**2>&1:** a `stderr`-t ugyanoda irányítja, ahová a `stdout` irányítva lett

**1>&2:** a `stdout`-ot ugyanoda irányítja, ahová a `stderr` irányítva lett

**Megjegyzés:** a > átirányítás felülír. Ezért ha egy létező file végéhez szeretnénk hozzáfűzni átirányítás során, akkor használjuk a >> formában megadott *adaptív* átirányítást.

## 2.3. |, a csővezeték

Ha egy parancs kimenetét szeretnénk egy másik parancs bemenetére irányítani, arra az ún. csővezeték (pipe) alkalmas, melyet a | karakter reprezentál.

*Példák*

- `ls | wc -w`

Hány állomány van az aktuális könyvtárban?

- `ls | sort | less`

Lapozható, sorbarendezt állománylista

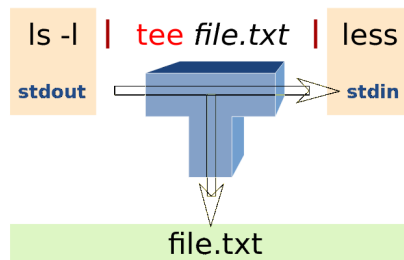
## 2.4. tee

```
tee [-ai] [-ignore-interrupts] [ fájl... ]
```

A tee parancs a standard bemenetén kapott adatokat a standard kimenetre és valamennyi argumentumként kapott fájlba másolja. Ez akkor hasznos, ha az adatokat nemcsak a csővezetéken szeretnénk továbbítani, hanem szükségünk van egy másolatra is.

### Kapcsolók

- a, --append** A standard bemenet tartalmát a célfájlok végéhez fűzi, és nem írja felül azokat.
- i, --ignore-interrupts** Figyelmen kívül hagyja a megszakításra vonatkozó jelzéseket.
- help** Használati útmutatót ír a standard kimenetre, majd sikeres visszatérési értékkel kilép.
- version** A program verziójáról ír ki információt a standard kimenetre, majd sikeres visszatérési értékkel kilép.



Kép forrása: [http://hu.wikipedia.org/wiki/Tee\\_%28parancs%29](http://hu.wikipedia.org/wiki/Tee_%28parancs%29)


### Példa (1)

Nézzünk a csővezeték és a tee együttes használatára egy példát! Tegyük fel, hogy van egy fájlunk, melynek minden sorában egy név áll. A sort parancs segítségével a nevsor\_kevert.txt tartalmát sorbarendezzük. A sorbarendezés eredménye a tee parancs hatására megjelenik a standard kimeneten (terminál) illetve a nevsor.txt fájlban is.

```
joe@localhost:~$ cat nevsor_kevert.txt
Kocsis Attila
Dobi Imre
Toth Zoltan
Kormanyos Jozsef Sandor
Lok Arpad
Botas Zoltan
Baranyi Peter
Buza Endre Csongor
joe@localhost:~$ cat nevsor_kevert.txt | sort | tee nevsor
.txt
Baranyi Peter
Botas Zoltan
Buza Endre Csongor
Dobi Imre
Kocsis Attila
Kormanyos Jozsef Sandor
Lok Arpad
Toth Zoltan
```

## Példa (2)

```
joe@localhost:~/tmp/Documents$ ls
BatteryLinux.png  final.png  Judy.png  pepita_sakk.png
ubigraph2.png
fajl.txt         hello.sh  Logo.png  ubigraph1.png
joe@localhost:~/tmp/Documents$ ls | tee fajl.txt
BatteryLinux.png
fajl.txt
final.png
hello.sh
Judy.png
Logo.png
pepita_sakk.png
ubigraph1.png
ubigraph2.png
joe@localhost:~/tmp/Documents$ cat fajl.txt
BatteryLinux.png
fajl.txt
final.png
hello.sh
Judy.png
Logo.png
pepita_sakk.png
ubigraph1.png
```

ubigraph2.png

## 2.5. Szöveg kiírása, szöveges fájlok kezelése

### echo – kiír egy szövegsort

Az `echo` kiír minden megadott karakteláncot a szabványos kimenetre, szóközzel elválasztva és egy újsor karakterrel a végén, hacsak nem volt megadva a `-n` opció.

#### Kapcsolók

- n** Nem írja ki a sor végére a soremelést karaktert.
- e** Engedélyezi a következő speciális karakterek értelmezését a karakterláncokban:
  - `\a` riadó (csengő)
  - `\b` egy karakter törlése visszafelé
  - `\c` nem ír ki újsor karaktert
  - `\f` lapdobás
  - `\n` új sor
  - `\r` koci vissza
  - `\t` vízszintes tab
  - `\v` függőleges tab
  - `\\` backslash
  - `\nnn` a karakter ASCII kódja `nnn` (oktálisan)
- E** backslash (`\`) karakterrel megadott karakterek értelmezésének tiltása (alapértelmezett)

```
joe@localhost:~$ echo "Hi all"
Hi all
joe@localhost:~$ echo 'Hi all!'
Hi all!
```

**printf – formátumozott adatkirás**

A `printf` kinyomtatja a formátum szöveget, értelmezi a `''` és `'\'` escape szekvenciákat ugyanúgy, mint a C `printf` függvény. A formátum argumentumot használja az összes kapott argumentum formázásához.

```
joe@localhost:~$ printf "%s\n" "Hello World"
Hello World
```

**wc – fájlokban található bájtok, szavak és sorok számát írja ki**

A `wc` program bájtok, szavak és újsor-jelek számát számolja meg az argumentumként megadott fájlokban. Ha nem adunk meg fájlnevet, illetve a fájlnévként a `'-'` jelet adjuk meg, akkor a szabványos bemenet olvassa a program.

Alapértelmezés szerint a `wc` mindhárom számot kiírja. Az opciókkal lehet megadni, hogy csak bizonyos számok legyenek kiírva. Az opciók nem semlegesítik egymás hatását, így pl. `wc --bytes --words` a bájtok és a szavak számát egyaránt kiírja. Minden fájlról egy-sornyi információt ír ki, és az argumentumként megadott fájl(ok) nevét is kijelzi. Több fájlnev esetén egy összesített sort is megad a lista végén `total` fájlneven. A megadott adatok sorrendben a következők: sorok, szavak, bájtok száma.

*Kapcsolók*

- c, --bytes, --chars** Csak a bájtok számát írja ki.
- l, --lines** Csak a sorok számát írja ki.
- w, --words** Csak a szavak számát írja ki.
- L, --max-line-length** Csak a fájlban előforduló leghosszabb sor hosszát írja ki, illetve ha egynél több fájl volt megadva, akkor kiírja még a legnagyobbat az előző értékek közül (nem az összegüket írja ki).

```
joe@localhost:~$ wc os05.tex
266  1152 11304 os05.tex
```

### grep – mintához illeszkedő sorokat nyomtat<sup>3</sup>

A grep szétválasztja azokat a sorokat, amelyekben a keresett részlet megtalálható azoktól, melyekben nem. A grep alapesetben csak azokat a sorokat választja ki (írja ki), melyekben a keresett mintát megtalálta:

```
joe@localhost:~$ ps -e | grep firefox
3531 ?          00:11:22 firefox-bin
joe@localhost:~$ ps -e | grep kde
3342 ?          00:00:00 kdeinit4
3347 ?          00:00:00 kded4
```

A -v (revert, ellentétes) kapcsoló hatására a grep csak azokat a sorokat tengedi tovább, melyekben a keresett minta nem található meg.

```
joe@localhost:~$ who | grep joe
joe      tty7          2011-02-26 19:17 (:0)
joe      pts/0           2011-02-26 19:29 (:0.0)
joe      pts/3           2011-02-26 19:45 (:0.0)
joe      pts/1           2011-02-26 20:42 (:0.0)
joe      pts/2           2011-02-26 20:55 (:0.0)
joe@localhost:~$ who | grep -v pts
joe      tty7          2011-02-26 19:17 (:0)
```

A -i (ignore, figyelmen kívül hagy) kapcsoló hatására a grep nem veszi figyelembe a kis- és nagybetűk közti különbséget.

### head – fájlok első részének kiírása

<sup>3</sup>A grep parancs neve a sed parancs /g/re/p utasításából ered, ahol a re a regular expression rövidítése, A grep ugyanis pontosan azt teszi, amit a sed erre az utasításra



A **head** a megadott fájlok első részét (alapértelmezésben első 10 sorát) írja ki. Ha nincs megadva fájlnev, vagy a fájlnev '-', a bemenetét a szabványos bemenetről veszi. Ha egynél több fájl adott, a fájl nevét '==>' és '<==>' jelek közé téve minden fájl első része előtt kiírja.

A **head** kétfajta opciómegadást fogad el: az újat, amikor a számok az opciókat jelző betűknek argumentumok és a régít, amikor a számok megelőzik az opciókat jelző betűket.

**-c N, --bytes N** Az első N bájtot írja ki. N nem-nulla egész, amit opcionálisan követ a következő karakterek közül egy, kijelölendő az egységet:

**b** 512 bájt hosszú blokk

**k** 1 kilobájt hosszú blokk

**m** 1 megabájt hosszú blokk

**-n N, --lines N** Az első N sort írja ki.

**tail** – kiírja a meghatározott fájl utolsó részét

A **tail** parancs a megadott fájl(ok) utolsó sorait (10 sor az alapértelmezett) írja ki; a szabványos bemenetről olvas, ha nincs fájl megadva, vagy, ha a fájl nevet '-' követi. Ha több, mint egy fájl van megadva, kiír egy fejléct, ami tartalmazza a fájl nevét '==>' és '<==>' jelek közé zárva, a többi fájl kimenetei előtt.

Kapcsolóit lásd a **head** parancsnál.

**uniq** – egy rendezett fájlból kiszedi a duplikált sorokat

A **uniq** kiírja az egyedi sorokat egy rendezett fájlból, és eldobja az egyezőket egy kivételével. Opcionálisan, mutathatja csak azokat a sorokat is, amelyek pontosan megegyeznek, illetve azokat, amelyek egynél többször fordulnak elő. A **uniq**-nak rendezett bemenetre van szüksége, mivel csak az egymás után következő sorokat hasonlítja össze.

Ha a kimenet nem specifikált, a **uniq** a szabványos kimenetre ír. Ha a bemeneti file nincs megadva, a standard input-ot olvasza.

*Kapcsolók*

**-u, --unique** Csak a nem azonos sorokat írja ki.

**-d, --repeated** Csak a duplikált sorokat írja ki.

dos2unix – szöveges fájlok átalakítására használható DOS/MAC → UNIX

*Fontosabb kapcsolók*

**-k -keepdate** a kimeneti fájl időbélyege egyezni fog a bemeneti-vel

**-c -convmode convmode** az átalakítás módja. Lehet: ASCII, 7bit, ISO, Mac, alapértelmezett az ASCII.

unix2dos – szöveges fájlok átalakítására használható UNIX → DOS/MAC

*Fontosabb kapcsolók*

**-k -keepdate** a kimeneti fájl időbélyege egyezni fog a bemeneti-vel

**-c -convmode convmode** az átalakítás módja. Lehet: ASCII, 7bit, ISO, Mac, alapértelmezett az ASCII.

## 2.6. Felhasználókkal kapcsolatos parancsok

A **who** parancs kilistázza a képernyőre a számítógépre bejelentkezett felhasználókat.

Amennyiben az opciókon kívül nincs argumentuma, a **who** program kinyomtatja minden, pillanatnyilag bejelentkezett felhasználóról a következő információkat:

- bejelentkezési név (login name)
- terminál vonal (terminal line)
- a bejelentkezés ideje (login time)
- távoli gépnév vagy X kijelző (remote hostname or X display)

```
joe@localhost:~$ who
joe      pts/0      2011-02-19 11:03 (:0.0)
joe      pts/1      2011-02-19 12:03 (:0.0)
joe      pts/2      2011-02-19 12:04 (:0.0)
```

## A whoami parancs kiírja a felhasználó nevét.

A whoami program kiírja a bejelentkezett felhasználó nevét.

```
joe@localhost:~$ whoami
joe
```

## A finger parancs a felhasználói információk megjelenítésére szolgál.

finger [user]

### Kapcsolók

- s A finger megmutatja a felhasználó belépési nevét, valódi nevét, terminálját és hogy az írható-e (a terminál neve mögött „\* jelenik meg, ha nem írható), mióta nem csinált semmit, mikor lépett be, valamint irodájának helyét és telefonszámát. A belépés idejét hónap, nap, óra, perc formában adja meg, kivéve ha hat hónapnál régebben lépett be; ezesetben az óra és a perc helyett az évet jelzi ki. Az ismeretlen eszközök és a nemlétező belépési valamint nyugalmi időt csillaggal jelzi.
- l Több soros megjelenítés, amely magában foglalja az -s kapcsoló által mutatott adatokat, valamint a felhasználó home mappáját, otthoni telefonszámát, belépési shelljét, leveleinek állapotát és a home mappájában található .plan, .project valamint .forward nevű fájlok tartalmát.

```
joe@localhost:~$ finger
Login      Name            Tty      Idle   Login Time
  Office    Office Phone
joe        Joe C.         pts/0          4 Feb 19 11:03 (:0.0)
joe        Joe C.         pts/1       1:05 Feb 19 12:03 (:0.0)
joe        Joe C.         pts/2          Feb 19 12:04 (:0.0)
```

## 3. Linkelés, archiválás, csere

### 3.1. Linkelés

A *linkelés* arra szolgál, hogy egy adott állományt több néven is el lehessen érni az állományrendszerben.

Ha az `ls -l` paranccsal kilistázzuk a könyvtárunkat, láthatjuk a második oszlopban a *linkelési szám* (link count) oszlopot. Ez mutatja, hogy egy fizikai állományra hány néven hivatkozunk a fájlrendszerben. Ez a fajta linkelés egyrészt helyet takarít meg, másrészt a felhasználó számára teljesen láthatatlan.

Ezen linkelés az ún. *hard link*, mert közvetlenül az adott fájl inode-tábla bejegyzésére mutat a fájlrendszerben.<sup>4</sup> A hard link csak egy fájlrendszeren belül működik; nem linkelhetünk be például floppy-ról egy fájlt.

#### *Hard link létrehozása*

```
joe@localhost:~$ ls -l szoveg.txt
-rw-r--r-- 1 joe joe 12 2011-02-19 14:30 szoveg.txt
joe@localhost:~$ ln szoveg.txt link_szovegre.txt
joe@localhost:~$ ls -l szoveg.txt link_szovegre.txt
-rw-r--r-- 2 joe joe 12 2011-02-19 14:35 link_szovegre.
txt
-rw-r--r-- 2 joe joe 12 2011-02-19 14:30 szoveg.txt
```

Létezik még a *szimbolikus linkelés* (soft link) is. Lényege, hogy a szimbolikus link nem a fájl inode-tábla bejegyzésére mutat, hanem egy olyan különleges fájlra, ami a linkelt fájl nevét tartalmazza. Szimbolikus linket szintén az `ln` paranccsal hozunk létre, de a `-s` opciót is meg kell adni.

#### *Soft link létrehozása*

```
joe@localhost:~$ ls -l /etc/hosts
-rw-r--r-- 1 root root 367 2011-02-18 19:54 /etc/hosts
joe@localhost:~$ ln -s /etc/hosts
joe@localhost:~$ ls -l hosts
lrwxrwxrwx 1 joe joe 10 2011-02-19 14:32 hosts -> /etc/
hosts
joe@localhost:~$ ls -l /etc/hosts
```

<sup>4</sup>Minden egyes fájlhoz ill. könyvtárhoz tartozik egy egyedi számozósító, ez az inode [index node, listázása: `ls -li`]. A partíció elején található az ún. inode-tábla, ami megmondja, hogy hányas inode-ú fájl a merevlemezen fizikailag hol található, illetve, hogy milyen jogok és egyéb attribútumok érvényesek rá.

```
-rw-r--r-- 1 root root 367 2011-02-18 19:54 /etc/hosts
```

A linkszám ebben az esetben nem változott (az csak a hard link esetén nő), a fájltypusnál egy 'l' betű szerepel, jelezvén, hogy szimbolikus linkről van szó, és a fájlnevnél a '->' karakterek jelzik, hogy melyik fájlhoz van linkelve az állomány.

### 3.2. Archiválás, tömörítés

A tar parancs archiválást tesz lehetővé: segítségével egész könyvtárstruktúrákat egyetlen állományba tudunk menteni. Képes a gzip tömörítő programmal együtt dolgozni, amely esetben könyvtárakat alkönyvtárakkal és tartalmukkal együtt egyetlen tömörített állományba másolhatóak biztonsági mentés céljából.

Legegyszerűbb esetben a tar segítségével egy könyvtárat teljes tartalmával egyetlen állományba mentünk:

```
joe@localhost:~$ tar -cf Documents.tar Documents
joe@localhost:~$ ls -l Documents.tar
-rw-r--r-- 1 joe joe 286720 2011-02-19 14:17 Documents.
tar
```

A példában a tar -c (create, létrehoz) opciója jelezte, hogy archívum létrehozása a célunk, a -f (file, állomány) pedig a létrehozni kívánt állomány neve előtt áll. A -f után mindig egy állomány nevének kell következnie, amely konvenció szerint a .tar végződést kapja.

*Megjegyzés:* A .tar végződésű állomány nem csak a könyvtárban található fájlokat és tartalmakat hordozza, hanem az egyes állományok tulajdonosainak, csoporttulajdonosainak és jogosultságot jelző kapcsolóinak értékét is. A teljes könyvtár e kiegészítő információk segítségével helyreállítható a következő módon:

```
joe@localhost:~$ tar -xf Documents.tar
```

A -x (extract, szétszedés) opció jelzi, hogy a tar archívumot újra szét kívánjuk bontani, míg a már ismert -f opció a fájlnev előtt áll. A kicsomagolás során a tar az eredeti archív állományt nem semmisíti meg, csak helyreállítja

az eredeti könyvtárstruktúrát. A tar alkalmas a gzip tömörítő programmal való együttműködésre is.

Amennyiben a `-z` (gzip) opciót kapja, az archív állományt tömöríti – helytakarékoság céljából. A következő példában látható, hogy a `z` opció az `f` elé került, mivel az `f` után mindenképpen az állomány nevének kell következnie:

```
joe@localhost:~$ tar -czf Documents.tar.gz Documents
joe@localhost:~$ ls -l Documents.tar.gz
-rw-r--r-- 1 joe joe 267638 2011-02-19 14:19 Documents.
tar.gz
```

Azoknak az állományoknak, amelyek tar archívokat gzip tömörített formában tartalmaznak, a konvenció szerint `.tar.gz` vagy egyszerűen `.tgz` végződést adunk. Ezen állományokat a tar a következő módon képes kicsomagolni:

```
joe@localhost:~$ tar -xzf Documents.tar.gz
```

### 3.3. Fájlok összehasonlítása

#### cmp – két fájl összehasonlítása

A `cmp` program összehasonlít két tetszőleges típusú fájlt és kiírja az eredményt a szabványos kimenetre. Alapértelmezés szerint a `cmp` nem ír ki semmit, ha a két fájl megegyezik. Ha különböznek, akkor kiírja a byte- pozíció és a sor számát, ahol az első különbség előfordult.

A byte-pozíciók és a sorszámok számozása egytől kezdődik.

- l Minden előforduló különbségnél kiírja a byte-pozíciót (decimális) és a különböző byte-értékeket (oktális).
- s Nem ír ki semmit különböző fájlok esetén, csak a kilépési kódot adja vissza.

#### diff – állományok összehasonlítására használható

kapcsolók ] file1 file2

diff [

A `diff` parancs segítségével két szöveges állományt hasonlíthatunk össze. Segítségével megtudhatjuk, hogy a két állomány megegyezik-e, és ha nem, akkor miben különböznek egymástól. A `diff` sor alapú mintaillesztést végez, azaz ha két sor egyetlen betűben is különbözik egymástól, a teljes sort kiírja.

### cut – sorok kiválasztott részeit írja ki

A `cut` program minden megadott fájl minden sorának a kiválasztott részeit írja ki a szabványos kimenetre. Amennyiben a bemeneti fájl nincs megadva vagy az a '-', a szabványos bemenetet dolgozza fel.

```
cut [-ns] [-b BÁJTLISTA] [-c KARAKTERLISTA] [-d ELVÁLASZTÓ] [-f MEZŐLISTA]
```

A BÁJTLISTA, a KARAKTERLISTA és a MEZŐLISTA egy vagy több számból, illetve tartományból áll, melyeket vesszők választanak el (a tartományokat két, egymástól '-' jellel elválasztott szám határozza meg). A bájtok, karakterek és mezők számozása 1-től indulva történik. Nem teljes tartomány megadása is lehetséges: '-M' azonos '1-M'-mel, míg 'N-' jelentése: az N-től a sor végéig vagy az utolsó mezőig.

```
joe@localhost:~$ cat matrix.txt
11 12 13 14 15
21 22 23 24 25
31 32 33 34 35
joe@localhost:~$ cat matrix.txt | cut -d " " -f 2
12
22
32
joe@localhost:~$ cat matrix.txt | cut -d " " -f 4-5
14 15
24 25
34 35
```

## 3.4. Csere

### tr – karakterek lecserélése, tömörítése és/vagy törlése

```
tr [-cdst] [--complement] [--delete] [--squeeze-repeats]
    [--truncate-set1] string1 [string2]
```

A `tr` átmásolja a szabványos bemenetet a szabványos kimenetre végrehajtva egyet a következő feladatok közül:

- cserél, és választhatóan tömöríti az eredményben az ismétlődő karaktereket
- tömöríti az ismétlődő karaktereket
- karaktereket töröl
- karaktereket töröl, majd tömöríti az eredményben az ismétlődő karaktereket.

```
joe@localhost:~$ echo "var" | tr a e
ver
joe@localhost:~$ echo "abcdefghijklmnpq" | tr a-j 0-9
0123456789klmnpq
joe@localhost:~$ echo "abcdef" | tr abc ABC
ABCdef
```

Egy gyakori alkalmazása a `tr` parancsnak a kisbetűk nagybetűvé alakítása. Ez megoldható több módon. Itt van közülük három:

```
tr abcdefghijklmnopqrstuvwxyz
  ABCDEFGHIJKLMNOPQRSTUVWXYZ
tr a-z A-Z
tr '[:lower:]' '[:upper:]'
```

### sed – szöveg cseréje

```
sed [-n] [-g] [-e script] [-f sfile] [file] ...
```

Sokszor szükségünk van arra, hogy egy állományban bizonyos szövegrészeket kicseréljünk valami másra. A `sed` használható erre az alábbi módon:



```
sed -e s/ezt/erre/g <bemenet.txt >kimenet.txt
```

A fenti parancs a `bemenet.txt` fájlt olvassa, a `kimenet.txt` fájlba írja az eredményt és az ezt előfordulásait az `erre` szövegre cseréli ki.

## 4. BASH

### 4.1. Alapok

A BASH héj, mint a legtöbb héj, nemcsak egy felhasználói felület, de kifinomult, magasszintű programozási nyelvet megvalósító értelmező program (interpreter) is. A BASH ezzel a nyelvvel alkalmas a napi feladataink automatizálására, a munkakörnyezet bővítésére, testreszabására.

*Miért kell megismernünk?*

- Az összetettebb feladatokat akkor tudjuk elvégezni, ha a megfelelő vezérlő szerkezeteket ismerjük.
- Az automatizálás igen fontos eleme a számítógéphaszúlatnak.
- A Unix rendszerek felépítésében komoly szerepet kapnak a héjprogramok (glue).

*Az értelmező*

- BASH programok szöveges állományok, amelyek futtatását rendszer-mag végzi bash program segítségével.
- Amikor elterjedten kezdtek több héjprogramot használni, szükségessé vált az értelmezőprogram meghatározása. Ha a fájl első két karaktere `#!` a mag az utána következő programnevet használja futtatásra.

### Változók

Mint minden parancsokra épülő programozási nyelv, a BASH nyelve is rendelkezik változókkal. BASH változókat a parancssorban is használhatunk, de programokban mindenképpen szükségünk van rájuk.

*A változók*

- névvel és értékkel rendelkező eszközök, általában szöveges érték tárolására használjuk
- a hagyomány szerint nagybetűkkel írjuk a nevüket

## Az értékadás

változónév=kifejezés

Az értékadás baloldalán egy változó neve, jobboldalán egy kifejezés áll. Az értékadás hatására a változó értéke felveszi a kifejezés értékét. Az egyenlőségjel bal és jobb oldalán nem lehet szóköz!

```
#!/bin/bash

KONYVTAR=tmp
SZOVEG="hello vilag"
```

Ha a szöveges értékben szóköz szerepel, a szöveget kettős idéző jelek (") közé kell zárunk. Ha ezt nem tesszük meg, a BASH a szóközők mentén szétválasztja a szöveget és listaként kezeli.

## Behelyettesítés

...\$változónév...

A változóbehelyettesítés használatakor a változónév elé egy \$ karaktert írunk. Az adott helyre a BASH a változó értékét helyettesíti be.

```
#!/bin/bash

KVT=tmp
ALKVT=elso
ALALKVT=masodik
mkdir -p $KVT/$ALKVT/$ALALKVT
```

**\$VALTOZO** A megadott nevű környezeti változó aktuális értékének behelyettesítése. Ha a változó nem létezik, üres szót kapunk.

**\${VALTOZO}** Hatása megegyezik az előzővel, de ez akkor is használható, ha közvetlenül a kifejezés után betű, számjegy vagy aláhúzásjel áll (máskülönben azt a név részének tekintené a shell).

### `${VALTOZO:+ERTEK}` alternatív érték használata

Ha a VALTOZO üres (nulla hosszúságú) vagy nem kapott még értéket, akkor nem történik semmi, különben az ERTEK-kel helyettesítődik a fenti kifejezés.

### `${VALTOZO:-ERTEK}` alapértelmezett érték használata

Ha VALTOZO üres, a kifejezést ERTEK-kel helyettesíti. Különben a kifejezés értéke \$VALTOZO A VALTOZO értéke változatlan marad.

### `${VALTOZO:=ERTEK}` alapértelmezett érték hozzárendelése

Ha VALTOZO üres, a változó értékét ERTEK-re állítja.

### `${!VALT}` indirekció

A \$VALT változónevű változó értékét kéri le, tehát a !VALT helyére a VALT értéke kerül.

```
#!/bin/bash

STR="A"
A="Hello"
echo "${!STR}"
echo "$A"
```

`${VALTOZO:?UZENET}` Ha a VALTOZO-nak nincs értéke, akkor megszakad a futás az UZENET-ben megadott szöveg kiírása után

Futtassuk a `scriptek/bash/default.sh` szkriptet!<sup>5</sup>

## Érték beolvasása

`read változónév`

A `read` utasítás a felhasználó által írt szöveget helyezi el az utána írt változóban. A felhasználó a szöveg beírása közben használhatja a szerkesztő billentyűket és a szövegben tetszőleges karaktereket elhelyezhet.

<sup>5</sup>Megtalálható kitömörítés után a <http://www.inf.u-szeged.hu/~grerika/teaching/os/scriptek.tar.gz> fájlban, kitömörítéshez segítség a 26. oldalon

```
#!/bin/bash
read KVT
mkdir "$KVT"
```

Az idézőjelek (") miatt a parancs akkor is egy paramétert kap, ha a felhasználó által beírt szövegben szóköz van.

## Idézőjelek hatása

A *kettős idézőjelek* (" [Alt] + 2)) részleges elszigetelést jelölnek, ami azt jelenti, hogy a héj értelmezi és behelyettesíti a benne található, számára értelmes karaktereket. Ha például egy \* karaktert talál, akkor azt a munkakönyvtárban levő fájlok neveinek listájával fogja helyettesíteni. Ha \$ karaktert „lát”, a közvetlenül utána következő szót változónévnek vagy parancssori paraméternek fogja tekinteni és a megfelelő értéket behelyettesíti.

A *szimpla (egyszeres) idézőjelek* ( ' [Alt] + 1)) használata teljes elzárást jelent. Az ilyen karakterláncokban a héj semmiféle értelmezést nem végez, azok tartalmát betű szerint kezeli.

A *visszafelé hajló* ( ` [Alt] + 7)) idézőjelek kifejezetten parancsvégrehajtást jelölnek. Az ilyen szöveget a héj parancssornak tekinti, végrehajtja és az eredménnyel helyettesíti. Ez a parancsbehelyettesítésnek nevezett eljárás lehetővé teszi, hogy egy változónak azonnal átadjuk egy esetleg meglehetősen bonyolult parancssor kimenetét.

"..."	változó behelyettesítés van
'...'	változó behelyettesítés nincs
"..."	a " jelek elvesztik különleges jelentésüket
'...'	a ' jelek elvesztik különleges jelentésüket
`...`	A közrezárt szöveg végrehajtásra kerül

1. táblázat. Idézőjelek fajtái és használatuk

```
! /bin/bash
#NEV=Gizi
echo -n "Kerek egy nevet: "
read NEV
echo Nev: $NEV
echo "Nev: $NEV"
```

```
echo 'Nev: $NEV'
echo "Nev: '$NEV'"
echo '"Nev: $NEV"'
```

scriptek/bash/behelyettesites.sh

*Eredmény (Ha a NEV értéke Gizi)*

```
Nev: Gizi
Nev: Gizi
Nev: $NEV
Nev: 'Gizi'
"Nev: $NEV"
```

## A beágyazott utasítás

```
...$(utasitas)...
```

Beágyazott utasítást bárhová elhelyezhetünk, ahol változó értékét behelyettesíthetjük. A beágyazott utasítás is behelyettesítés. A BASH a `$()` kifejezésen belül található utasítást parancsként (programként) végrehajtja. A kifejezés behelyettesítési értéke a program szabványos kimenetén megjelenő lista lesz. A beágyazott utasítás egy formája a visszafele hajló idézőjel, lásd a(z) [1. táblázatot](#) ([34. oldal](#)). Tehát a(z) `$(utasitas)` ekvivalens a(z) `'utasitas'`-al.

## 4.2. A környezeti változók

A környezeti változók névvel és értékkel rendelkező eszközök, amelyek nevüket onnan kapták, hogy a munkakörnyezetet írják le a programok számára. Minden folyamat rendelkezik környezeti változókkal, nemcsak a BASH, nem csak a héjprogramok. (Ahogyan minden folyamat rendelkezik munkakönyvtárral is!)

\$BASH	A futtató bináris állomány
\$HOME	A felhasználó saját könyvtára
\$USER	Felhasználó login neve
\$USERNAME	A felhasználó teljes neve
\$HOSTNAME	A gép neve
\$PWD	Aktuális könyvtár
\$PATH	A parancsok keresési útja
\$COLUMNS	Betűoszlopok száma a képernyőn
\$LINES	Sorok száma képernyőn
\$TERM	Terminál típusa
\$EDITOR	Alapértelmezés szerinti szövegszerkesztő

2. táblázat. Fontosabb környezeti változók

A környezeti változók kezelése egyirányú:

- A szülőfolyamat meghatározza a gyermekfolyamat környezeti változóit, de a gyermek nem változtathatja meg a szülő környezeti változóit.
- Amikor egy folyamat egy másik folyamatot indít, másolat készül a környezeti változóiról.
- A folyamat futása közben használhatja, megváltoztathatja a környezeti változóit.
- A folyamat futása közben újabb környezeti változókat hozhat létre.
- Amikor egy folyamat befejeződik, a környezeti változói megsemmisülnek.

### A környezeti változók használata

A BASH programban a környezeti változó értékét ugyanúgy helyettesíthetjük be, mint a BASH saját változót.

```
#!/bin/bash  
echo $USER@$HOSTNAME
```

Ezeket a változókat a BASH program a szülőfolyamattól örökölte.

### A környezeti változó megváltoztatása

Ha megváltoztatjuk egy környezeti változó értékét, az utána indított programok már az új értékét kapják meg. (A BASH változók értékét azonban nem kapják meg az indított programok.)

```
#!/bin/bash  
  
USER=gizi  
firefox
```

Ha a környezeti változó létezett, az új értéket kapják meg a gyermekfolyamatok; ha nem, akkor csak egy BASH változót hoztunk létre.



## Környezeti változó létrehozása

Környezeti változót a BASH változóból, az `export` parancs segítségével hozhatunk létre.

```
#!/bin/bash
IZE="mintamokus"
export IZE
```

Az `export` parancs kiadása után több változónevét is írhatjuk, mindegyikből környezeti változó lesz. Az egyszerűsített írásmód esetén az értékadást és az `export` kulcsszót egy sorban helyezük el. Az `export` parancs kiadása után több értékadást is írhatuk, mindegyik környezeti változót hoz létre.

```
#!/bin/bash
export IZE="mintamokus"
```

## 4.3. Parancssori paraméterek

A héjprogramok meghívásakor átadhatunk egy vagy több paramétert. Több paraméter esetén azokat egy vagy több szóköznek kell elválasztania. Ha az átadandó paraméter maga is tartalmaz szóközt, kettős idézőjelbe kell tenni. A parancssori paraméterek értékére a `$1`, `$2`, ... szimbólumokkal hivatkozhatunk. A szám a kérdéses paraméter sorszáma. A 0 sorszámú paraméter minden esetben maga a meghívott program neve.

<code>##</code>	A parancssori paraméterek száma
<code> \$?</code>	A legutoljára végrehajtott parancs visszatérési értéke
<code> \$\$</code>	A futó program folyamatazonosítója
<code> \$n</code>	Az $n$ -edik parancssori paraméter értéke
<code> \$0</code>	A pillanatnyi héjprogram neve
<code> \$*</code>	Valamennyi parancssori paraméter egyben, egyetlen karakterláncként (" <code>\$1</code> <code>\$2</code> ... <code>\$9</code> ")
<code> @\$</code>	Valamennyi parancssori paraméter egyben, egyenként idézőjelbe téve (" <code>\$1</code> " " <code>\$2</code> " ...)

3. táblázat. A héj névvel nem rendelkező belső változói

## 4.4. Matematikai kifejezések

Számos esetben szükségünk lehet egyszerű matematikai műveletekre a héj-programozás során.

Az `expr` egy négy alapműveletes számológép, de kizárólag egész számokkal képes műveleteket végezni.

	Logikai VAGY operátor. Visszatérési értéke az első paraméter, ha az nem nulla vagy nem üres karakterlánc, egyébként a második.
&	Logikai ÉS operátor. Visszatérési értéke az első paraméter, ha egyik argumentuma sem nulla vagy üres karakterlánc. Ellenkező esetben nullát ad vissza.

4. táblázat. Az `expr` logikai operátorai

<	Kisebb
<=	Kisebb vagy egyenlő
>	Nagyobb
>=	Nagyobb vagy egyenlő
=, ==	Egyenlő
!=	Nem egyenlő

5. táblázat. Az `expr` által ismert relációs jelek

+	Összeadás
-	Kivonás
*	Szorzás
/	Osztás
%	Maradékképzés

6. táblázat. Az `expr` műveletei

A relációs jelek (6. táblázat) használatakor az `expr` 1-et ad vissza, ha az összehasonlítás igaz, nullát ha hamis. Az összehasonlítás elvégzése előtt megkísérli számokká alakítani a megadott paramétereket. Ha ez sikeres, aritmetikai összehasonlítást végez. Ha bármelyik paramétert nem képes átalakítani, akkor az összehasonlítás betűrend szerinti (lexikografikus) lesz. Ilyenkor az a paraméter számít nagyobbnak, amelyikben előbb következik magasabb ASCII kódú karakter.

Aritmetikai műveletek csak számokon hajthatóak végre, így ha valamilyik paraméter nem alakítható számmá, hiba keletkezik.

*Buktató:* ügyeljünk arra, hogy az `expr` az egyszerű műveleti jeleket is csak akkor hajlandó értelmezni, ha azokat szóközök választják el a tényezőktől. Így a `SZAM='expr 3+2'` forma például helytelen. A helyes írásmód:

`SZAM='expr 3 + 2'`

Arra is figyelni kell, hogy az `expr` egyes műveletei a héj számára is értelmesek és ha elfelejtjük levédeni ezeket a `\` karakterrel, furcsa mellékhatásokat tapasztalhatunk.

### Példa az `expr` használatára

`scriptek/matematika/expr.sh`

Nem csak az `expr` használható matematikai műveletek elvégzésére, hanem a `let` parancs is.

```
#!/bin/bash
COUNTER=10
echo "$COUNTER"

let COUNTER -=1
echo "$COUNTER"

let COUNTER=COUNTER-1
echo "$COUNTER"

echo "$((COUNTER-=1))"
echo "$((COUNTER=COUNTER-1))"

echo "$[COUNTER-=1]"
echo "$[COUNTER=COUNTER-1]"
```

Példa `let` használatára (`scriptek/matematika/let.sh`)

## 4.5. for

```
for valtozo in lista; do
    utasitasblokk
done
```

```
for valtozo in lista
do
    utasitasblokk
done
```

A változó rendre felveszi a lista elemeinek értékét és minden értékkel végrehajtodik az utasítás blokk minden utasítása.

*A változó lehet:*

- BASH változó neve, vagy
- Környezeti változó neve.

A változó neve elé nem kell \$ jelet írunk! Az utasításblokk tetszőlegesen sok utasításból állhat, amelyekben használhatjuk a változó értékét. A változó neve elé ilyenkor \$ jelet kell írunk.

*A lista lehet:*

- Szavak szóközökkel elválasztott listája, vagy
- Állománynév helyettesítő karakterekkel megadott állománylista, vagy
- változók listája, vagy
- végrehajtandó parancs a \$(...) vagy '...' szerkezettel.

A listát lezárja:

- a pontosvessző, vagy
- az újsor karakter.

```
#!/bin/bash

for DAY in hetfo kedd szerda; do
    echo $DAY
    mkdir $DAY
done
```

```
#!/bin/bash

for FILE in *.dvi; do
    echo "Nyomtatás: $FILE"
    dvips $FILE
done
```

```
#!/bin/bash

MENTENI="$HOME/bin $HOME/munka"
for KONYVTAR in $MENTENI;do
    echo "Mentes: $KONYVTAR"
    cp -r $KONYVTAR /mnt/pendrive
done
```

```
#!/bin/bash

for ORA in $(seq 1 24);do
    echo "Letrehoz: $ORA"
    mkdir /mnt/pendrive/$ORA
done
```

Beágyazott utasítást (lásd [35. oldal](#)) és for ciklust hatékonyan használhatunk együtt. A ciklus egyenként végigjárja az utasítás kimenetén megjelenő listát.

```
#!/bin/bash

for F in $(ls -l | grep ^d | awk '{print $9}'); do
    echo "$F archivalasa"
    tar -czf $F.tar.gz $F
done
```

Beágyazott utasítás segítségével változók értékét is beállíthatjuk:

```
#!/bin/bash

for F in *; do
    KISBETUS=$(echo $F | tr A-Z a-z)
    mv $F $KISBETUS
done
```

## Egymásba ágyazott for ciklusok

```
for valtozo1 in lista1; do
    for valtozo2 in lista2; do
        utasitasblokk
    done
done
```

```
#!/bin/bash

for EMAIL in $(cat ~/cimek); do
    echo -n "Kuldes $EMAIL cimre:"
    for FILE in ~/LEVELEK/*.xt; do
        echo -n "$FILE"
        mail $EMAIL <$FILE
    done
    echo "[OK]"
done
```

## Gyakorlasképp

1. Írj szkriptet, mely beolvas egy szöveget, eltárolja azt, majd kiírja a konzolra.
2. Írj szkriptet, mely kiírja a paraméterként kapott fájlok típusát és tartalmát.

### 4.6. if

```
if utasitas; then
    utasitasblokk
fi
```

```
if utasitas
then
    utasitasblokk
fi
```

A BASH végrehajtja az utasítást és ha igaz (o a visszatérési értéke), lefuttatja az utasításblokk utasításait is.

```
if utasitas; then
```

```
        utasitasblokk1
    else
        utasitasblokk2
    fi
```

A BASH az utasítást végrehajtja: Ha igaz, az igaz ág, ha hamis (nem 0), a hamis ág utasításait hajtja végre.

A feltételként szereplő utasítást pontos vessző, vagy újsor karakter zárja le.

*Példa*

```
#!/bin/bash

if mount /mnt/floppy; then
    echo "Beillesztes megtörtént."
else
    echo "Beillesztes sikertelen."
fi
```

## 4.7. Logikai műveletek

Programok visszatérési értékein használhatjuk az alábbi logikai operátorokat.

!	tagadás(prefix)
	logikai vagy(infix)
&&	logikai és(infix)

7. táblázat. Logikai műveletek

## Tagadás

```
#!/bin/bash

if ! mkdir $HOME/tmp2;
then
    echo "Nem sikerult
        létrehozni."
fi
```

## ÉS

```
#!/bin/bash

if mkdir tmp && cp ak tmp;
then
    echo "Sikerult!"
fi
```

## VAGY

```
#!/bin/bash

if mkdir tmp || mkdir tmp2; then
    echo "Legalabb egyik könyvtar létrejött!"
fi
```

Utasításokból logikai operátorok segítségével képzett kifejezések szintén utasítások, de elsőre úgy tűnhet, hogy talán kissé különösen viselkednek.



## Lusta kiértékelés mellékhatásai

Az alábbi esetben, ha az első utasítás értéke hamis, az ÉS művelet miatt a végeredmény mindenképp hamis lesz, felesleges tehát futtatni a második utasítást!

```
utasitas1 && utasitas2
```

Az alábbi esetben pedig ha az első utasítás értéke igaz, a VAGY művelet miatt a végeredmény mindenképp igaz lesz, felesleges tehát futtatni a második utasítást!

```
utasitas1 || utasitas2
```

### *Példa*

```
#!/bin/bash

ls -ld ak || echo "Nem erhető el!"
```

Az ilyen szerkezetek úgy működnek, mint az if szerkezet: a második utasítás végrehajtása az első eredményétől függően következik be.

## 4.8. Logikai állandók

Logikai állandót a `true` és `false` programok segítségével használhatunk a BASH-ban.

**true** A program nem csinál semmit, visszatérési értéke igaz.

**false** A program nem csinál semmit, visszatérési értéke hamis.

## 4.9. A test program

Az aritmetikai relációk kiértékelésére és az állományvizsgálatra általában a `test` parancsot használjuk.

```
#!/bin/bash

if 8<9;then
    echo "Nyolc kisebb,
        mint kilenc."
fi
```

```
#!/bin/bash

if 8 < 9;then
    echo "Nyolc kisebb,
        mint kilenc."
fi
```

**Nagyon rossz!** Ez azt jelenti, hogy a 8 nevű program szabványos bemenetére irányítsuk a 9 nevű állományt!

## Helyesen

```
#!/bin/bash

if test 8 -lt 9; then
    echo "Nyolc kisebb, mint kilenc."
fi
```

## A test kettős viselkedése

```
#!/bin/bash

if test 8 -lt 9; then
    echo "Nyolc kisebb,
        mint kilenc."
fi
```

```
#!/bin/bash

if [ 8 -lt 9 ]
then echo "Nyolc kisebb,
        mint kilenc."
fi
```

**FONTOS:** a jobb oldali szintaxis esetén, a [ után illetve a ] előtt mindenképp használnunk kell szóközőket!

## Fájlvizsgálat

```
#!/bin/bash

if [ ! -d $HOME/bin ]; then
    mkdir $HOME/bin
```

```
fi
```

## Karakterlánc példa

```
#!/bin/bash

echo -n "Könyvtarnev [bin]: "
read KVT
if [ -z "$KVT" ];then
    KVT=bin
fi
```

*Megjegyzés:* ha a felhasználó nem ad meg karakterláncot és az idézőjeleket nem használjuk, akkor innen hiányzik a paraméter!

-b fájlnev	blokkeszköz-meghajtó
-c fájlnev	karaktereszköz-meghajtó
-d fájlnev	könyvtár
-f fájlnev	szabályos állomány
-L fájlnev	közvetett hivatkozás
-p fájlnev	csővezeték
-e fájlnev	létezik
-G fájlnev	saját csoportba tartozik
-O fájlnev	saját tulajdon
-r fájlnev	olvasható
-w fájlnev	írható
-x fájlnev	futtatható
-s fájlnev	nem üres
fájl1 -nt fájl2	a fájl1 újabb, mint a fájl2
fájl1 -ot fájl2	a fájl1 régebbi, mint a fájl2
fájl1 -ef fájl2	a fájl1 és fájl2 azonos állományt jelöl

8. táblázat. Fájlvizsgálat a test paranccsal

Kif1 -eq Kif2	Egyenlő
Kif1 -ne Kif 2	Nem egyenlő
Kif1 -lt Kif2	Kisebb
Kif1 -le Kif2	Kisebb vagy egyenlő
Kif1 -gt Kif2	Nagyobb
Kif1 -ge Kif2	Nagyobb vagy egyenlő
Kif1 -a Kif2	Logikai és
Kif1 -o Kif2	Logikai vagy
!Kif	Logikai tagadás


9. táblázat. Numerikus és logikai operátorok

-z String	o hosszúságú
String	nem o hosszúságú
String != String	nem egyenlők
String = String	egyenlők

10. táblázat. Numerikus és logikai operátorok

## 4.10. while

```
while utasitas; do
    utasitasblokk
done
```

Amíg a for szerkezet esetében lehetetlen volt, addig a while szerkezet esetében lehetséges az utasítások végtelen ismétlése, a végtelen ciklus. A végtelen ciklusba került programot a  + c billentyűkombinációval vagy a kill programmal szakíthatjuk meg.

```
#!/bin/bash
while true; do
    echo "Futok..."
    sleep 3
done
```

A sleep parancs felfüggeszti a program futását a paraméterre által megadott másodpercre.

```
#!/bin/bash
SZOVEG="You have a new message."
while [ ! -s "$MAIL" ]; do
    sleep 3
done
echo $SZOVEG
play /usr/share/sounds/email.wav
echo $SZOVEG | festival --tts
```

### Végtelen ciklus segítségével

```
#!/bin/bash
SZOVEG="You have a new message."
while true;do
    while [ ! -s "$MAIL" ];do
        sleep 3
    done
    echo $SZOVEG
    play /usr/share/sounds/email.wav
    echo $SZOVEG | festival --tts
    while [ -s "$MAIL" ];do
        sleep 3
    done
done
```

```
done  
done
```

## 4.11. do-until

```
until utasitas; do  
    utasitasblokk  
done
```

Az until szerkezet addig ismétli az utasításblokkot, amíg az utasítás visszatérési értéke hamis. Az until tehát while szerkezethez képest a feltételként adott utasítás értelmét az ellentettjére változtatja.

## 4.12. case

```
case szo in
    minta1a|minta1b)
        utasitasblokk1;;
    minta2)
        utasitasblokk2;;
    *) utasitasblokk0
esac
```

A case szerkezet illeszti a szót a megadott mintákra, majd azt az utasításblokkot hajtja végre, amelyik az illeszkedő minták közül az elsőhöz tartozik. A case a minták kezelésénél az állománynév helyettesítő karakterek kezelésének szabályait használja.

*Példák*

Példa:

scriptek/bash/case.sh

```
#!/bin/bash

read K

case $K in
    *.jpeg)
        mv $K $(basename
            $K jpeg)jpg;;
    *.gif)
        convert $K $(
            basename $K gif
            )jpg
esac
```

```
#!/bin/bash

# Peldaprogram a BASH
# verziójának
# megállapítására
# Forrás:
# http://conectiva.com.br
# ~aurelio/programas/
# bash/txt2regez

case "$BASH_VERSION" in
    4.0[4-9]*|4.[1-9]*) :
        ;;
    *) echo "bash version
        >=4.04 required
        ..."; exit 1 ;;
esac
```

## 4.13. Vágókifejezések

A vágókifejezések nem kötelező anyagrész, de hasznos eszközök bash programozáskor.

<code>\${STRING#minta}</code>	Ha a minta illeszkedik a string elejére, akkor levágásra kerül a legrövidebb illeszkedő rész
<code>\${STRING##minta}</code>	Ha a minta illeszkedik a string elejére, akkor levágásra kerül a leghosszabb illeszkedő rész
<code>\${STRING%minta}</code>	Ha a minta illeszkedik a string végére, akkor levágásra kerül a legrövidebb illeszkedő rész
<code>\${STRING%%minta}</code>	Ha a minta illeszkedik a string végére, akkor levágásra kerül a leghosszabb illeszkedő rész

```
#!/bin/bash
# Forras: Pere Laszlo, Linux felhasznaloi ismeretek I.
# PARAMETEREK SZAMANAK ELLENORZESE
if [ $# -ne 2 ]; then
    echo "'basename $0' - Allomany kiterjesztesenek
        csereje"
    echo "Hasznalat:"
    echo "      'basename $0' <regi kit> <uj kit>"
    exit 1
fi
# TENYLEGES MUNKA
for FILES in *. $1; do
    echo -n "$FILES -> ${FILES%$1}$2"
    if [ -e "${FILES%$1}$2" ]; then
        echo " Hiba: A ${FILES%$1}$2 mar letezik"
    else
        echo
        mv $FILES ${FILES%$1}$2
    fi
done
```

scriptek/bash/vago.sh

## 4.14. Függvények

```
#!/bin/bash

function fuggvenynev(){
    utasitas1
    ...
}

fuggvenynev [parameterek] # hasznalat
```



A függvények paramétereire – hasonlóan a héjprogramokhoz – a \$1, \$2, ... ki-  
fejezésekkel hivatkozhatunk. A függvények hozzáférnek valamennyi a fő-  
programban meghatározott változóhoz és maguk is létrehozhatnak újakat.  
Ez utóbbiakat a főprogram is látni fogja, azok nem lokálisak a függvényre  
nézve.

### Lokális változók, rekurzió

Függvényekben létrehozhatunk lokális változókat is, a `local` kulcsszó se-  
gítségével.

```
#!/bin/bash

function faktorialis ()
{
    local n=$1;

    if [ $n = 0 ]; then
        echo 1
        return ;
    fi;

    echo $(( $n * $(faktorialis $(( $n - 1 )) ) ))
}

for n in $(seq 1 20); do
    echo "$n! = " $(faktorialis $n)
done
```

## 4.15. Tömb

Nem kötelező anyagrész, de hasznos tudni. Forrás: <http://www.cyberciti.biz/faq/bash-for-loop-array/>

### Deklarálás

```
declare -a arrayname
```

vagy

```
array=( one two three )
files=( "/etc/passwd" "/etc/group" "/etc/hosts" )
limits=( 10, 20, 26, 39, 48)
```

vagy

```
array[1]=one
array[2]=two
array[4]=three
```

A tömbök indexei egészek lehetnek, nem feltétlenül folytonosak.

### Kiírás, használat

A printf használatával kiirathatjuk a tartalmunkat, például így

```
printf "%s\n" "${array[@]}"
printf "%s\n" "${files[@]}"
printf "%s\n" "${limits[@]}"
```

vagy számláló ciklussal végigjárhatjuk a tömb elemeit

```
for i in "${tombnev[@]}"
do
    echo $i
done
```

Ha az indexek értékeire vagyunk kíváncsiak, azt is lekérhetjük, azt a `${!tombnev[*]}` tartalmazza

Ha szeretnénk kilistázni az indexeket a hozzájuk tartozó tömbértékekkel, akkor azt így tehetjük:

```
for I in ${!array[*]}; do
    echo $I: ${array[$I]}
done
```

```
array=( "/etc/passwd" "/etc/group" "/etc/hosts" )
array[5]=5
array[7]=7
array[8]="nyolc"
array[100]="szaz"
```

```
for I in ${!array[*]}; do
    echo -e "$I\t${array[$I]}"
done
```

## 5. Reguláris kifejezések

Sok program (főleg szűrők) használ *mintaillesztést* (pattern matching), *mintakeresést* (pattern scanning) és *mintafeldolgozást* (pattern processing). Ilyen esetekben a – legtöbbször szöveges – bemeneti adatok azon részével foglalkozni a program, amely egy megadott mintának megfelel, azaz a minta illeszkedik (vagy amire a minta illeszkedik). Az ilyen komplex minták egyik gyakran alkalmazott formája a szabályos avagy *reguláris kifejezés* (regular expression, regexp, RE).

A reguláris kifejezésekkel mélyebben a formális nyelvek elmélete (theory of formal languages) foglalkozik.

**Leírás:** man 7 regex, man grep, info grep, man awk/gawk, info gawk

Egy reguláris kifejezés a szövegnek mindig a legkorábban elkezdődő, és ezen belül a leghosszabb részére illeszkedik. Ez a részkifejezésekre is igaz. Az illeszkedő rész a szövegen belül bárhol – akár egy szó belsejében is – előfordulhat, kivéve néhány esetet (pl.

Alapesetben a kisbetűk és nagybetűk különbözőnek számítanak illeszkedéskor.

A reguláris kifejezésekben néhány karakternek speciális jelentése van. Mivel ezek közül sokat a shell is speciálisan kezel, így a parancssorban megadott reguláris kifejezést érdemes aposztrófok közé zárni.

### 5.1. Elemi kifejezések

**(KIF)** csoportosítás (műveleti sorrend felülbírlása), KIF-re illeszkedik

**()** az üres szóra illeszkedik

**[HALMAZ]** A halmaz bármely karakterének egy példányára illeszkedik.  
A halmazt a karakterek egymás mellé írásával adhatjuk meg.

**[TOL-IG]** mint előbb, de itt egy tartományt adunk meg

**[^HALMAZ]** a halmazban nem szereplő bármely karakter egy példányára illeszkedik (a sortörést kivéve)

**.** bármilyen karakter egy példányára illeszkedik (a sortörést kivéve)

**^** a sor elejére illeszkedik

\$ a sor végére illeszkedik

\**KARAKTER** a \ után írt speciális jelentésű karaktert közönségesként kezeli

**KARAKTER** bármely közönséges karakter saját maga egy példányára illeszkedik

## 5.2. Összetett kifejezések

**KIF<sub>1</sub>KIF<sub>2</sub>** (két kifejezés egymás mellé írása): Összefűzés, konkatenáció (concatenation). Olyan szövegre illeszkedik, amelynek első fele KIF<sub>1</sub>-re, második fele KIF<sub>2</sub>-re illeszkedik. Több kifejezést is összefűzhetünk.

**KIF<sub>1</sub>|KIF<sub>2</sub>|...** Logikai MEGENGEDŐ VAGY (diszjunkció), alternáció (alternation). Olyan szövegre illeszkedik, amely legalább az egyik kifejezésre (alternatívára) illeszkedik. ismételt illesztés, ismétlésszám megadása, iteráció (repetition, iteration):

**KIF\*** KIF akárhány egymást követő példányára illeszkedik (0 is)

**KIF+** KIF legalább 1 egymást követő példányára illeszkedik

**KIF?** KIF 0 vagy 1 példányára illeszkedik (azaz KIF opcionális)

**KIF{I}** KIF pontosan I egymást követő példányára illeszkedik

**KIF(I,)** KIF legalább I egymást követő példányára illeszkedik

**KIF(I,J)** mint előbb, de legfeljebb J példányra illeszkedik ( $I \leq J$ )

Műveleti erősség csökkenő sorrendben: iteráció, konkatenáció, alternáció

## 5.3. grep (ismét)

grep 'REGKIF' ÁLLOMÁNY(OK)

Kiírja a megadott állomány(ok) mindazon sorait, amelyek illeszkednek a REGKIF reguláris kifejezésre. Szűrőnek tekinthető. A korábban említettek miatt az aposztrófok kiírása ajánlott. Ha nem adunk meg állományt, akkor a szabványos bemenetről olvas.

*Kapcsolók*

- c Az illeszkedő sorok tartalma helyett csak azok darabszáma jelenik meg.  
A -v opció esetén a nem illeszkedő sorok száma íródik ki.
- E Teljes értékű, kibővített (extended) kifejezések használata. Ha ezt elhagyjuk, akkor a reguláris kifejezéseknek egy régebbi változatát kell megadnunk. Ez utóbbi jelentősen eltér a korábban bemutatottól!
- e 'REGKIF' Akkor kell használni, ha a reguláris kifejezés - jellel kezdődik.  
Közvetlenül a REGKIF előtt kell állnia!
- F REGKIF-ben minden karaktert közönségesként értelmez
- f KIFFÁJL A KIFFÁJL minden sorát egy-egy REGKIF-nek tekinti. Ilyenkor a bármelyik kifejezésre illeszkedő sorok jelennek meg
- i a kisbetűket és a nagybetűket azonosnak tekinti
- n az illeszkedő sorok tartalma elé a sorszámukat is kiírja
- o a sorokból csak az illeszkedő részt jeleníti meg
- R, -r Ha könyvtárat adtunk meg, akkor a keresés az alkönyvtárakban és azok teljes tartalmában történik (rekurzív keresés).
- v illeszkedés helyett nem-illeszkedést vizsgál (inverzió)
- w Csak olyan sort ír ki, amelyben legalább egy egész szó (nemcsak egy részlet) illeszkedik a reguláris kifejezésre

A gyakorlatban a -E opció vagy a vele ekvivalens egrep parancs használata ajánlott! Ha nem így tennénk, vegyük figyelembe, hogy ezek nélkül a reguláris kifejezéseknek egy régebbi (basic) változatát kell használnunk, ahol pl. a ( és ) közönséges karakterek, a csoportosításra pedig a \( és \) jelölések szolgálnak (tehát a korábban látotthoz képest pont fordítva működnek). Ugyanez érvényes a {, }, |, ? és + karakterekre is.

## 5.4. Példák

- **alma** azt jelenti, hogy a minta a soron belül bárhol előfordulhat
- **^alma** előírja, hogy a mintának a sor elején kell előfordulnia
- **^[mh]?alma** azt jelenti, hogy a sor elején alma, malma vagy halma mintának kell előfordulnia

- `^[^mh]alma` azokra a sorokra illeszkedik, melyek elején nem szerepel az alma, malma vagy halma sorozat

```
$ ls -l | grep ^d
```

kiírja a munkakönyvtárban levő összes katalógust

```
$ grep '^main' *.c
```

kiírja a .c végű állományok azon sorait, amelyek a main karaktersorral kezdődnek.

## Feladatok

1. Írjunk reguláris kifejezést, ami az egész számokra illeszkedik!
2. Írjunk reguláris kifejezést, ami a szabályos e-mail címekre illeszkedik!  
Tegyük fel, hogy e-mail cím általános alakja: `local@domain`, ahol a **local** az angol abc kisbetűit, számokat és kötőjelet, pontot illetve aláhúzást (`_`) tartalmazhat, de aláhúzással nem kezdődhet **domain** részben az abc kisbetűi, számok és kötőjel, illetve pontok lehetnek.  
  
A fenti egy egyszerűsített leírása a ténylegesen érvényes e-mail címeknek. Nem lehet például pont a `@` előtt, illetve nem lehet két pont egymás után (sem a `local`, sem a `domain` részben)<sup>6</sup>.
3. Próbáljunk szabályos magyar címekre szűrni, ahol tegyük fel, hogy a cím az alábbihoz hasonló formában fordulhat elő:  
`XXXX. Város, Közterületnév utca/út/körút/sétány/tér/park szám.`  
Figyeljünk arra, hogy közterület neve állhat több szóból is!

<sup>6</sup>További infó: [http://en.wikipedia.org/wiki/Email\\_address#Syntax](http://en.wikipedia.org/wiki/Email_address#Syntax)

## 6. AWK

`awk 'PROGRAM' ÁLLOMÁNY(OK)`

- Leírás: `man awk/gawk`, `info gawk`
- Mintakereső és -feldolgozó program saját programozási nyelvvel
- Sorban beolvassa a bemeneti állomány(ok) tartalmát, miközben az AWK nyelven írt PROGRAM-ban leírt műveleteket végrehajtja. Szintén szűrő.
- Ha nem adunk meg állományt, akkor a szabványos bemenetről olvas.
- A forrásprogram szövegét érdemes aposztrófok közé zárni, hogy a benne szereplő karaktereket a shell ne tekintse speciálisnak.
- **-f PROGFÁJL**: a végrehajtandó programot PROGFÁJL-ból olvassa

gawk Az eredeti awk program GNU változata, GNU/Linux alatt ezt használhatjuk.

`#!/usr/bin/awk -f`: Ha az AWK forrásprogramot állományban tároljuk el, az állomány első sorába ezt a megjegyzést (parancsértelmező fejléce) írjuk, valamint futtathatóvá tesszük az állományt, akkor az AWK programot a shell scriptek mintájára a `./PROGFÁJL ÁLLOMÁNY(OK)` paranccsal is lefuttathatjuk.

### 6.1. Forrásprogram felépítése

Minden AWK forrásprogram **szabályok** sorozata. Minden szabály tartalmazhat egy **mintát** és egy hozzá tartozó tevékenységet avagy **akciót**. Az akciót különféle utasításokból állíthatjuk össze.

- A szabályok alakja: `MINTA{AKCIÓ}`
- A szabályokat egymástól sortöréssel vagy pontosvesszővel lehet elválasztani.
- A feldolgozás során a bemenet tartalmát *rekordokra* (record) bontja, ezek alapesetben a bemenet sorai lesznek. A rekordokat szintén továbbbontja *mezőkre* (field), amiket alapesetben az illető sor szavai képviselnek.



- A bemenet feldolgozása rekordonként történik. Minden rekordot megpróbál illeszteni sorban az összes szabály mintájára, az első szabálytól kezdve. Ha a rekord illeszkedett egy szabály mintájára, akkor végrehajtódik a hozzá tartozó akció. Végül az összes szabály ellenőrzése után rátér a következő rekord feldolgozására.
- A szabályok sorrendje fontos, hiszen a mintákra való illeszkedés ellenőrzése, s így az akciók végrehajtásának sorrendje ettől függ! Hiányzó minta esetén az illető akció minden rekord esetén lefut.
- A szabályokból az akciót is el lehet hagyni a kapcsos zárójelekkel együtt. A hiányzó akció ekvivalens a `{print}` akcióval, ami kiírja az egész rekord tartalmát.
- Vigyázat! A `{ }` páros az üres akciót jelöli, tehát nem egyezik meg az előbb említett esettel (ti. az akció elhagyásával)!
- Bármely mintát vagy utasítást folytathatjuk a következő sorban, ha az aktuális sort a `\` jellel zárjuk.
- Az akciók utasításlistája akár több sorból is állhat. Egy sorba több utasítást is írhatunk, ha őket pontosvesszővel (;) választjuk el egymástól. Hasonlóan, a pontosvessző használatával több szabályt is írhatunk egy sorba.
- Szóközöket és tabulátorokat tetszés szerint használhatunk a műveleti jelek, operandusok, utasítások, paraméterek, stb. között. Üres sorok szintén megengedettek.
- `#`: A sor végéig tartó megjegyzés kezdetét jelzi.
- Az AWK is különbséget tesz a kisbetűk és nagybetűk között!

## 6.2. Minták

Minden minta egy logikai feltételt fogalmaz meg. Ha a feltétel teljesül egy konkrét rekord esetén, akkor azt mondjuk, hogy a rekord illeszkedik a mintára. Fontos, hogy olyan feltételt is megfogalmazhatunk, amely nem (vagy nemcsak) a rekord tartalmától függ, hanem például valamely változótól!

## Elemi minták

(MINTA) csoportosítás (műveleti sorrend felülbírálnak), MINTÁ-ra illeszkedik

!MINTA logikai tagadás (negáció)

/REGKIF/ igaz, ha az egész rekord illeszkedik a reguláris kifejezésre

KIF~/REGEX/ igaz, ha a KIF kifejezés mint szöveg illeszkedik a reguláris kifejezésre

KIF!~/REGEX/ igaz, ha a kifejezés nem illeszkedik a REGKIF-re

**relációs kifejezések** tetszőleges kifejezés, amely relációs jelet tartalmaz

BEGIN csak a bemenet feldolgozása előtt teljesül

END csak a bemenet feldolgozása után teljesül

- A BEGIN és END mintákhoz mindig meg kell adni az akciót is! Továbbá ezek a speciális minták nem kombinálhatók semmilyen más mintával, valamint nem alkalmazható rájuk a csoportosítás és a negáció sem!
- A BEGIN mintához tartozó akció pontosan egyszer hajtódik végre, mégpedig a legelső bemeneti rekord feldolgozása előtt. Ez akkor is így történik, ha több bemeneti állományt adtunk meg.
- Hasonlóan, az END mintához tartozó akció is pontosan egyszer, az utolsó bemeneti rekord feldolgozása után hajtódik végre. Ezt az awk program befejeződése követi.

## Összetett minták

MINTA1&&MINTA2 logikai ÉS (konjunkció)

MINTA1| |MINTA2 logikai MEGENGEDŐ VAGY (diszjunkció)

MINTA1,MINTA2 Rekordok tartományára illeszkedik, kezdve egy olyan rekorddal, amely MINTA1-re illeszkedik, egészen egy olyan rekordig, amely MINTA2-re illeszkedik. Nem kombinálható semmilyen más mintával!

## 6.3. Konstansok

### Szám avagy numerikus konstansok

- egész számok (pl. 12)
- valós törtszámok tizedesponttal (pl. 25.3)
- egész vagy valós szám hatványkitevővel (pl.  $1.234e+2=123.4$ )

### Szöveges avagy sztring konstansok

- "SZÖVEG"
- "": üres sztring (0 karakter hosszúságú szöveg)
- A szövegben a \ speciális (az ún. escape-karakter), így használhatók pl. a következő escape-szekvenciák: \\ (közönséges \), \" (közönséges idézőjel), \n (sortörés), \t (tabulátor).

### Konstans reguláris kifejezések

- /REGKIF/
- A reguláris kifejezésen belül a \ speciális, így használhatók a \\ (közönséges \) és \/ (közönséges /) karakterpárosok.

## 6.4. Változók

- A változók *élettartama dinamikus*, az első használatkor automatikusan létrejönnek (nem kell őket deklarálni).
- A változók neve betűket, számokat és aláhúzásjelet (\_) tartalmazhat, és nem kezdődhet számjeggyel.
- **Változók típusai**
  - numerikus változók (valós számokat tárolnak)
  - szöveges változók avagy sztringek (string)
  - egydimenziós tömbök
- A tömböket kivéve minden változó *típusa dinamikus*, azaz a használatától függően változik!  
Ez a tömbelemekre is vonatkozik
- Egy változó típusát nem lehet tömbről numerikusra vagy sztringre változtatni, és viszont!

- A változók értékét az awk automatikusan konvertálja számmá vagy szöveggé, szintén a használati módtól (művelettől, függvényről) függően. Ha a szöveget nem lehet számmá konvertálni (mert nem egy érvényes alakú számot tartalmaz), nullát kapunk.
- **Manuális konverzió**
  - szövegből szám: adjunk hozzá o-t
  - számból szöveg: fűzzük hozzá az üres sztringet (" ")
- **NÉV=ÉRTÉK**
  - Értékadás egy létező változónak, vagy új változó létrehozása. A változó típusa ÉRTÉK típusa lesz.
  - A C programozási nyelv egyéb értékadó, növelő és csökkentő műveletei is használhatók
  - Az ÉRTÉK természetesen nemcsak konstans, hanem kifejezés is lehet.
  - Többszörös értékadás (NÉV1=NÉV2=ÉRTÉK) is megengedett.
- **NÉV**
  - a változó aktuális értékét jelöli
  - definiálatlan (ti. amelyiknek eddig nem adtunk értéket) változó értéke az üres sztring (" ") ill. o.

## 6.5. Beépített változók

Az awk program indulásakor már létezik jónéhány különleges, **beépített változó** (built-in variable). Ezek neve egységesen csupa nagybetűből áll, és tartalmuk egyrészt a felhasználónak szóló fontos információkat hordoz, másrészt némelyikük az awk program működését ill. a bemenet feldolgozásának módját vezérli.

**FILENAME** Az aktuális bemeneti állomány neve, illetve - a szabványos bemenet esetén.

A **BEGIN** minta (lásd 63. oldal) akcióján belül definiálatlan.

**FNR** az aktuális rekord sorszáma az aktuális bemeneti állományon belül

FS bemeneti mezőhatároló karakter (input field separator), kezdetben a szóköz

IGNORECASE Ha értéke nemzérus, akkor a sztringek összehasonlítása ill. a reguláris kifejezések illesztése nem különbözteti meg a kisbetűket a nagyoktól. Alap esetben értéke definiálatlan (effektíve nulla).

NF az aktuális rekord mezőinek száma (number of fields)

NR Az aktuális rekord sorszáma az eddig feldolgozott bemenet tekintetében.

Egy bemeneti állomány ill. a szabványos bemenet esetén egyenlő az FNR-rel.

OFS Kimeneti mezőhatároló (output field separator), kezdetben a szóköz.

Értéke tetszőleges szöveg lehet, nemcsak egy karakter.

ORS Kimeneti rekordhatároló (output record separator) kezdetben a sortörés.

Ez is tetszőleges szöveget tartalmazhat.

RS bemeneti rekordhatároló karakter (input record separator), kezdetben a sortörés

## 6.6. Mezők

A bemenet rekordokra bontását, ill. azoknak mezőkre bontását két beépített változó vezérli. Az RS változó tartalma egy karakter (alap esetben sortörés), ez jelzi a rekordokat elválasztó karaktert. Hasonlóan, az FS változó tartalma (alap esetben szóköz) határozza meg, mi határolja a mezőket a rekordokon belül. Ha az FS értéke a szóköz (alap esetben), akkor a mezőket legalább egy szóköz vagy tabulátor választja el.

- Az aktuális rekord mezőinek a számát az NF beépített változó tárolja.
- A mezők típusa ugyancsak numerikus vagy szöveges lehet, az aktuális használatától függően. Összehasonlításakor a mezők tartalmát számnak tekinti az awk, ha az valóban egy érvényes számot tartalmaz, továbbá ha a másik tag szám konstans, numerikus változó vagy mezőhivatkozás.
- \$KIF

- Az aktuális rekord megadott sorszámú mezőjének tartalma. Ezt a jelölést mezőhivatkozásnak nevezzük.
- Tetszőleges kifejezést is használhatunk, például  $\$(2*3)$  a hatodik mezőt jelzi. Természetesen a negatív értékek nem megengedettek.
- \$NF az aktuális rekord utolsó mezőjének tartalma
- \$0 (dollárjel és nulla): az aktuális rekord teljes tartalma
- \$KIF=ÉRTÉK
  - egy adott mező – ill. KIF = 0, esetén a rekord – értékének módosítása
  - Ha \$0 tartalmát változtatjuk meg, akkor minden mező új értéket kap. Ha viszont egy mező tartalmát módosítjuk, akkor \$0 értéket az awk újraépíti oly módon, hogy a mezőket az OFS értéke határolja majd el.
  - Ha KIF > NF, akkor a mezők számát kibővíti, és NF-et is módosítja. Szükség szerint a közbülső helyekre új mezőket szűr be, ezek értéke az üres sztring ("" ) lesz. Végül pedig \$0 tartalmát is újraszámítja az előbb leírt módon.

## Gyakorló feladatok

Dolgozzuk fel az ls -l parancs kimenetét egy tetszőlegesen választott könyvtár esetén!

1. Írjuk ki csak a könyvtárbejegyzések nevét és méretét! Vegyük figyelembe, hogy névben előfordulhat szóköz is, tehát a név nem minden esetben a 8. mező maga! Emellett a linkeket, mint speciális eseteket kezelni kell, hiszen a linkeknél a 8. mezőtől kezdve az utolsóig nem csak a link nevét, hanem magát a célfájl útvonalát is tartalmazza linknev -> celfajl formában. Segíthet a megoldásban a [71.](#) oldalon található *Szöveges függvények* alfejezet.
2. Számoljuk meg hány link, hány közönséges fájl és hány könyvtár szerepel a parancs kimenetén!
3. Összegezzük a fájlok méretét és vessük össze egyezik-e az eredmény azzal, amit kapunk a du -s paranccsal!

4. Állapítsuk meg melyik a legkisebb / legnagyobb fájl (a nevét és a méretét is írjuk ki), és számoljunk átlagos fájl méretet!

+1 Dolgozzunk fel egy szöveges fájlt, ami egy mátrixot tárol, ahol az elemeket egy sorban pontos vessző választja el. *Ötlet:* generáláshoz használjuk a bash-t és a RANDOM változó értékét!

5. Számoljuk sorösszeget és -átlagot minden rekordra és oszlopösszeget és -átlagot minden oszlopra.

## 6.7. Tömbök

Lehetőség van egydimenziós tömbök használatára is. Fontos, hogy a tömb méretét nem kell előre lerögzíteni, továbbá a tömbelemek indexe tetszőleges szöveg lehet (a számokat is szöveggé konvertálja)! Az ilyen tömböket asszociatív tömböknek nevezik.

- A tömbök nevét a változónevek mintájára adhatjuk meg.
- A tömb vegyesen tartalmazhat numerikus és szöveges elemeket is!
- `NÉV[INDEX]=ÉRTÉK`
  - Értékadás egy létező tömbelemnek, vagy új elem beszúrása. Az elem típusa `ÉRTÉK` típusa lesz. A tömb is létrejön, ha még nem létezett.
  - A C programozási nyelv egyéb értékadó, növelő és csökkentő műveletei is használhatók
  - Az `INDEX` és az `ÉRTÉK` konstans és tetszőleges kifejezés is lehet.
- `NÉV[INDEX]`
  - a megadott indexű tömbelem aktuális értékét jelöli
  - Definiálatlan elem értéke az üres sztring (`"`) ill. `0`.
- `INDEX in NÉV` Ez a logikai reláció csak akkor igaz, ha a tömbnek van `INDEX` indexű eleme. Lásd még: `for`, `while`, `do ... while`, `if` utasítások.
- `delete NÉV[INDEX]` a megadott indexű tömbelem kitörlése
- `delete NÉV` A tömb összes elemének kitörlése. Vigyázat, a tömb továbbra is létezni fog, csak üres lesz!

## 6.8. Kifejezések felépítése

A minták és az utasítások megadásához használhatunk különféle kifejezéseket. Az ezeket felépítő építőkövek: konstansok, változók, műveleti jelek, függvények, segédjelek (például zárójelek, vessző).



## Műveletek, relációk

- Aritmetika valós számokon: +, -, \*, /, %, ^ (hatványozás)
- Növelés (increment), csökkentés (decrement): ++, -- (mindkettő prefix és postfix használatban is)
- Sztring összefűzés, konkatenáció: egymás mellé írás, illetve szóköz
- Mező értékének használata (mezőhivatkozás): \$KIF
- Értékadás (assignment): =, +=, -=, \*=, /=, %=, ^=
- Összehasonlító relációk: <, <=, >, >=, ==, !=

Az összehasonlítás csak akkor történik numerikusan, ha a reláció mindkét oldalán szám konstans, numerikus változó vagy mezőhivatkozás áll. Máskülönben az értékek szövegesen (lexikografikusan, azaz az ábécé rendet követve) lesznek összehasonlítva!

A relációk numerikus értéke igaz esetén 1, különben 0. Ez az összehasonlító és mintaillesztő relációkra, továbbá az in relációra és a logikai műveletekre is vonatkozik.

- Mintaillesztő relációk: ~/REGKIF/, !~/REGKIF/

A mintaillesztő relációk igazak, ha a bal oldali kifejezés mint szöveg illeszkedik (~) ill. nem illeszkedik (!~) a jobb oldali reguláris kifejezésre.

- Tömbem létezésének vizsgálata: INDEX in NÉV
- Logikai műveletek: ! (negáció), && (konjunkció), || (diszjunkció)

A logikai műveletek, a feltételes kifejezés és a vezérlési szerkezetek szempontjából hamisnak (false) minősül az üres sztring ("") és a nulla. Minden más érték igaznak (true) számít.

- Feltételes kifejezés: KIF1?KIF2:KIF3 (mint a C nyelvben)

A feltételes kifejezésben először KIF1 lesz kiértékelve. Ha igaz, akkor KIF2, különben KIF3 lesz kiszámolva, s ők adják a kifejezés értékét is

## Numerikus függvények

- Trigonometria:  $\sin(KIF)$ ,  $\cos(KIF)$
- Gyökvonás:  $\sqrt[KIF]{}$
- Exponens, logaritmus:  $\exp(KIF)$ ,  $\log(KIF)$
- Egésszé konvertálás csonkolással (truncation):  $\text{int}(KIF)$

## Szöveges függvények

`index(SZÖVEG, RÉSZ)` a RÉSZ szöveg legelső előfordulásának pozíciója SZÖVEGben.

Ha nincs ilyen rész, akkor nullát ad vissza.

`length(SZÖVEG)` a megadott sztring hossza karakterekben

`split(SZÖVEG, TÖMB, HAT)` a SZÖVEG-et a HAT határolójel mentén darabokra bontja, a darabokat a megadott tömbben eltárolja, majd visszaadja a darabok számát. A SZÖVEG változatlan marad. A tömb elemei a darab sorszámaival (pont nélkül) lesznek indexelve. HAT reguláris kifejezés is lehet.

`substr(SZÖVEG, IND)` a szöveg IND sorszámu karakterén kezdődő részét adja vissza

`substr(SZÖVEG, IND, HOSSZ)` mint előbb, de legfeljebb HOSSZ karakterből álló részt ad vissza

`tolower(SZÖVEG)` visszaadja a SZÖVEG kisbetűssé konvertált értékét

`toupper(SZÖVEG)` visszaadja a SZÖVEG nagybetűssé konvertált értékét

`getline nev < "-"` Standard inputról olvas a nev változóba

```
BEGIN {
    printf "Irja be a nevet:"
    getline nev < "-"
    printf "%s kora:",nev
    getline kor < "-"
    print nev, ", kovetkezo evben " kor + 1 "eves lesz"
}
```

## 6.9. Vezérlési szerkezetek

UTASÍTÁSOK összetett utasítás, utasításblokk/-lista

if (FELTÉTEL) UTASÍTÁS else UTASÍTÁS0 szelekciós vezérlés

while (FELTÉTEL) UTASÍTÁS előfeltételes ismétléses vezérlés

do UTASÍTÁS while (FELTÉTEL) végfeltételes ismétléses vezérlés

for (KIF1;KIF2;KIF3) UTASÍTÁS számlálós ismétléses vezérlés

for (INDEX in NÉV) UTASÍTÁS Diszkrét ismétléses vezérlést valósít meg. Az INDEX változó sorban felveszi a NÉV nevű tömb elemeinek indexét, miközben a megadott utasítás végrehajtodik.

break, continue Kilépés a ciklusból, ill. rátérés a ciklus következő iterációjára (for, while és do...while esetén használhatók). Mindig az őket körbevevő legbelső ciklusra vonatkoznak!

exit A bemenet feldolgozásának azonnali befejezése. Ha nem az END minta akciójában használjuk, akkor az esetleges END minta akciója végrehajtodik, különben az awk rögtön befejezi működését.

print LISTA Kiírja a vesszővel tagolt kifejezéslista tagjainak értékét, majd az ORS tartalmát (alap esetben egy sortörést). A kiírt értékek közé az OFS tartalma kerül (alap esetben egy szóköz).

print ekvivalens a print \$0 utasítással (az aktuális rekord teljes tartalmát kiírja)

printf FORMÁTUM,LISTA formázott kiírás (mint a C prog. nyelvben)

next Azonnal nekikezd a következő bemeneti rekord feldolgozásához, a legelső szabály mintáját tesztelve.

## Gyakorló feladatok

6. A korábbi ls -l feldolgozó szkriptünket, ami átlagos fájl méretet számolt, egészítsük ki, számoljuk ki a fájl méret szórását is!  
+ feladat: Oldjuk meg a feladatot tömb használata nélkül! (matematikusok előnyben!)

Használjuk a *scriptek/telefon.txt* fájlt. A fájl az egyetemi telefonkönyv egy részlete.

7. Állapítsuk meg, hogy melyik beosztásból hány munkatárs szerepel a fájlban (a beosztás a 4. oszlopban szerepel)!
8. Számoljuk össze mennyi személynél szerepel e-mail cím!
9. Számoljuk össze mennyi személy nevében szerepel a dr. egyszer és mennyi alkalommal kétszer!
10. Számoljuk össze mennyi személy telefonszámában szerepel a 42-es szám! Ebből mennyinek végződik 42-re a telefonszáma?

## 6.10. Összetett példák (ls -l outputjára)

```
BEGIN{
    FS=" ";
    N=80
    for (i=0;i<N;i++)
        printf("-")
    print ""

    printf("\t%-35s\t%-12s\t%+10s\n","Fajlnev","Tipus+
        Jogok","Meret")
    for (i=0;i<N;i++)
        printf("-")
    print ""
}

$1!~/^l/{
    if(NR>1){
        fajlnev=""
        for(i=8;i<NF;i++)
            fajlnev=fajlnev $i " "
        fajlnev=fajlnev $NF

        printf("\t%-35s\t%-12s\t%+10s\n",fajlnev,$1,$5)
        db+=1
    }
}

END{
    for (i=0;i<N;i++)
        printf("-")
    print ""
}
```

```
} print NF-1,"sor összesen (" db " nem link)"
}
```

scriptek/awk/ls.awk

```
#!/usr/bin/awk -f

BEGIN{
    FS=" ";
    N=100
    for (i=0;i<N;i++){
        printf("-")
    }
    print ""
    printf("\t%-30s\t%-6s\t%-12s\t%+10s\n","Fajlnev","
        Tipus","Jogok","Meret")
    for (i=0;i<N;i++){
        printf("-")
    }
    print ""
}

$1!~/^l/{
    if(NR>1){
        fajlnev=""
        for(i=8;i<NF;i++){
            fajlnev=fajlnev $i " "
        }
        fajlnev=fajlnev $NF

        printf("\t%-30s\t%-6s\t%-12s\t%+10s\n",fajlnev,
            substr($1,0,1),substr($1,2,9),$5)
        NEMLINK+=1
    }
}

END{
    for (i=0;i<N;i++){
        printf("-")
    }
    print ""
    print NF-1,"sor összesen (" NEMLINK " nem link)"
}
```

scriptek/awk/ls2.awk

## 7. Irodalom

### Felhasznált irodalom

A jegyzet az alább feltüntetett könyvek, jegyzet illetve weboldal alapján készült, némi átcsoportosítással, itt-ott példákkal kiegészítve, vagy épp átfogalmazva.

- A parancsok kézikönyv (man) oldalai (1.-4. fejezet)
- PERE LÁSZLÓ: *Linux felhasználói ismeretek I.*, Kiskapu kiadó (5.-6. fejezet)
- RODEK LAJOS féle diasorozat (4-6. fejezet)
- BÜKI ANDRÁS: *Héjprogramozás*, Kiskapu kiadó
- <http://www.fsz.bme.hu/~szebi/slides/U2/sld033.htm> (5. fejezet, regex példák)

### Héjprogramozás Linux alatt (Linuxvilág)

1. rész: [http://www.linuxvilag.pbk.hu/content/files/cikk/28/cikk\\_28\\_61\\_61.pdf](http://www.linuxvilag.pbk.hu/content/files/cikk/28/cikk_28_61_61.pdf)
2. rész (héjváltozók): [http://linuxvilag.hu/content/files/cikk/29/cikk\\_29\\_59\\_59.pdf](http://linuxvilag.hu/content/files/cikk/29/cikk_29_59_59.pdf)
3. rész (feltételek): [http://www.linuxvilag.pbk.hu/content/files/cikk/30/cikk\\_30\\_60\\_61.pdf](http://www.linuxvilag.pbk.hu/content/files/cikk/30/cikk_30_60_61.pdf)
4. rész: [http://www.linuxvilag.pbk.hu/content/files/cikk/31/cikk\\_31\\_62\\_63.pdf](http://www.linuxvilag.pbk.hu/content/files/cikk/31/cikk_31_62_63.pdf)
5. rész (AWK): [http://www.linuxvilag.pbk.hu/content/files/cikk/32/cikk\\_32\\_68\\_69.pdf](http://www.linuxvilag.pbk.hu/content/files/cikk/32/cikk_32_68_69.pdf)
6. rész (sed): [http://www.linuxvilag.pbk.hu/content/files/cikk/33/cikk\\_33\\_54\\_55.pdf](http://www.linuxvilag.pbk.hu/content/files/cikk/33/cikk_33_54_55.pdf)
7. rész: [http://www.linuxvilag.pbk.hu/content/files/cikk/34/cikk\\_34\\_50\\_51.pdf](http://www.linuxvilag.pbk.hu/content/files/cikk/34/cikk_34_50_51.pdf)

8. rész (átmeneti fileok): [http://www.linuxvilag.pbk.hu/content/files/cikk/35/cikk\\_35\\_50\\_51.pdf](http://www.linuxvilag.pbk.hu/content/files/cikk/35/cikk_35_50_51.pdf)
9. rész: [http://www.linuxvilag.pbk.hu/content/files/cikk/36/cikk\\_36\\_54\\_55.pdf](http://www.linuxvilag.pbk.hu/content/files/cikk/36/cikk_36_54_55.pdf)

## Kódok elérhetősége

<http://www.inf.u-szeged.hu/~grerika/teaching/os/scriptek.tar.gz>

## Jegyzet elérhetősége

[http://www.inf.u-szeged.hu/~grerika/teaching/os/  
OperaciosRendszerek\\_jegyzet.pdf](http://www.inf.u-szeged.hu/~grerika/teaching/os/OperaciosRendszerek_jegyzet.pdf)