

Bagoly Zsolt, Papp Gábor

Bevezetés a UNIX rendszerekbe

egyetemi jegyzet
informatikus fizikus szakos hallgatóknak

1993-1994, ELTE, Budapest

Tartalomjegyzék

1. A UNIX kialakulásának története	13
2. UNIX alapok	15
2.1. A UNIX rendszerek elérése	16
2.1.1. Bejelentkezés a UNIX rendszerbe	16
2.1.2. Jelszó használata	17
2.1.3. Kijelentkezés	18
2.1.4. A használt shell beállítása (<code>chsh</code> parancs)	18
2.2. A UNIX shell parancsok	19
2.2.1. A parancs végrehajtásának megszakítása	19
2.2.2. A parancsok javítása	20
2.2.3. Az utasítások kapcsolói	20
2.2.4. Az utasítások argumentumai	22
2.2.5. A <code>man</code> parancs	22
3. Állományok használata	23
3.1. A UNIX állományok	23
3.1.1. Az állományok típusai	24
3.1.2. Állományok létrehozása	24
3.2. Állományok kilistázása	25
3.3. Állományok tartalmának megjelenítése	27
3.3.1. A <code>cat</code> parancs	28
3.3.2. Nagyobb állományok megjelenítése	29
3.4. Állományok másolása, átnevezése és törlése	31
3.4.1. Állományok másolása	31
3.4.2. Állományok átnevezése (<code>mv</code>)	32
3.4.3. Állományok törlése	32
3.5. Szöveg keresése: a <code>grep</code> parancs	33

3.6.	Állományok sorbarendezése a <code>sort</code> utasítás	33
3.7.	A UNIX helyettesítő karakterei	34
3.8.	Könyvtárak használata	37
3.8.1.	Abszolút és relatív útvonal	39
3.8.2.	A könyvtárrendszer használata	39
3.8.3.	Állományok másolása, mozgatása és linkelése	40
3.8.4.	Könyvtárak létrehozása és törlése	41
3.8.5.	Állomány megkeresése a könyvtárstruktúrában	41
3.9.	Állományok védelmének beállítása	42
4.	Állományok szerkesztése	45
4.1.	A <code>vi</code> editor használata	45
4.2.	A MicroEMACS szerkesztő	46
5.	Állományok nyomtatása	49
5.1.	A printer állapotának lekérdezése	50
5.2.	A nyomtatási kérés megszakítása	51
6.	Ki- és bemenetek átirányítása, különleges shell parancsok	53
6.1.	Parancs kimenetének átirányítása	53
6.2.	Bemenet átirányítása	54
6.3.	A cső (<i>pipe</i>) használata	56
6.4.	Parancs eredményének továbbhasználata	56
6.5.	Parancsállományok létrehozása	57
7.	Programok futtatása	59
8.	Kommunikációs programok	63
8.1.	Párbeszédés kapcsolat	63
8.1.1.	A <code>write</code> utasítás	63
8.1.2.	A <code>talk</code> parancs	64
8.1.3.	A <code>mesg</code> utasítás	65
8.2.	Elektronikus levelezés	65
8.2.1.	A mail levelező program	65
8.2.2.	Az <code>elm</code> levelező program	68
8.3.	Hálózat használata	70
8.3.1.	A <code>telnet</code> parancs	70
8.3.2.	Az <code>ftp</code> utasítás	71

8.3.3. A <code>finger</code> parancs	74
8.4. A tin elektronikus újság	75
8.5. A <code>gopher</code> elektronikus adatkereső program	78
9. A Korn shell	83
9.1. A környezetváltozók	84
9.2. A parancsok kiértékelése a shellben	84
9.3. Védőkarakterek	85
9.4. Parancssor szerkesztés	86
9.5. Visszatérési értékek	87
9.6. A Korn shell kapcsolói	87
9.7. A Korn-shell beépített parancsai	87
9.8. A függvények	89
9.9. Feltételes kifejezések	89
9.10. Speciális parancsok	91
9.11. Aritmetikai Kifejezések	97
9.12. Parancs eredményének helyettesítése	97
9.13. Állománynév helyettesítés	98
9.14. Ki- és bemenetek átirányítása	98
9.15. Helyettesítő nevek (<code>alias</code>)	99
9.16. A tilde helyettesítés	100
9.17. Paraméter helyettesítés	100
10. Állományok feldolgozása	107
10.1. A <code>grep</code> parancs ismertetése	107
10.2. A <code>sed</code> pipe szerkesztő	110
10.2.1. A <code>sed</code> parancsai	111
10.3. A <code>uniq</code> utasítás	115
10.4. Az <code>awk</code> programnyelv	117
10.4.1. Az adatállomány feldolgozása	118
10.4.2. Az <code>awk</code> programnyelv	118
10.4.3. A kiterjesztett reguláris kifejezés	118
10.4.4. Az összehasonlítások	120
10.4.5. A minták kombinációja	120
10.4.6. A <code>BEGIN</code> és <code>END</code> kifejezések	121
10.4.7. Változók	121
10.4.8. Az <code>awk</code> utasításai	123
10.4.9. Példák	126

10.5. Állományok sorrendbe rendezése: a <code>sort</code> utasítás	127
11. Archiválás és archív állományok használata	133
11.1. Könyvtárak archiválása	134
11.1.1. A <code>tar</code> parancs	134
11.1.2. A <code>dd</code> adatkonvertáló utasítás	135
11.1.3. Mágnesszalagos egységek	137
11.1.4. Példák archiválásra	138
11.2. Tömörítő programok	140
11.2.1. A <code>compress</code> eljárás	140
11.2.2. A <code>pack</code> eljárás	141
11.3. Kódolási eljárások	141
11.3.1. Az <code>uuencode/uuencode</code> parancs	141
11.3.2. A <code>btoa/atob</code> eljárás	142
11.3.3. Állományok szétDarabolása	142
12. Programok fordítása és futtatása	145
12.1. A UNIX C Fordító (<code>cc</code>)	146
12.2. A UNIX FORTRAN fordító (<code>f77</code>)	148
12.3. A Pascal fordító	149
12.4. A UNIX fordítói könyvtárkezelése (<code>ar</code> , <code>ranlib</code>)	150
12.5. A futtatható program összeszerkesztése: az <code>ld</code> parancs	152
12.6. Programok futtatása	155
12.6.1. A <code>batch</code> parancs	158
12.6.2. Feladatok rendszeres futtatása: a <code>crontab</code> parancs	158
12.7. Hibakeresés és ellenőrzött futtatás (<code>dbx</code>)	160
12.8. Programok újrafordítása: a <code>make</code> parancs	166
12.8.1. Kiterjesztések	167
12.8.2. Makrók	168
12.8.3. A <code>make</code> kapcsolói	169
13. A <code>vi</code> editor használata	173
13.1. Üzemmodok a <code>vi</code> -ban	173
13.2. A <code>vi</code> editor különböző beállításai	174
13.3. A <code>vi</code> parancsai	176
13.3.1. Mozgás az állományban	177
13.3.2. String keresés	178
13.3.3. A szövegírás üzemmód	178

13.3.4. Szövegváltoztatás a parancs módban	179
13.3.5. Másolás és mozgatás	180
13.3.6. Visszaállítás és parancs-ismétlés	180
13.3.7. Szövegrész helyettesítés	181
13.4. Állománykezelés	181
13.5. A shell elérése a vi-ből	182
13.6. Makrók és parancsok definiálása	183
14. A MicroEMACS szövegszerkesztő leírása	185
14.1. A legfontosabb MicroEMACS parancsok	185
14.2. Mozgás az állományban	186
14.2.1. Törlés és beszúrás	187
14.3. Keresés és helyettesítés	187
14.3.1. Állománykezelés	188
14.3.2. Buffer és ablak kezelés	189
14.4. Speciális utasítások	190
15. A mail program működése	193
15.1. A mail felépítése	193
15.2. Parancsok levél küldése közben	195
15.3. A mailbox alparancsok	196
16. Az elm részletes használata	203
16.1. Az elm alaputasításai	204
16.2. A postaládák	206
16.3. Aliasok	207
16.4. A konfigurációs file (elmr)	208
17. Hálózati alapismeretek és egyéb programok	209
17.1. TCP/IP alapok	209
17.1.1. A TCP/IP protokollok általános felépítése	210
17.1.2. A TCP szint	211
17.1.3. Socketek és applikációk	212
17.1.4. Egyéb protokollok: UDP, ICMP és ARP	213
17.1.5. Repeaterek, bridge-ek és routerek	214
17.2. Az rlogin és rsh utasítások	215
17.2.1. Az rlogin utasítás	216
17.2.2. Az rsh utasítás	216

17.3. Anonymous ftp használata	217
18. A tin elektronikus újság részletes leírása	219
18.1. A rovatok kiválasztása	220
18.2. Cikkék elküldése	221
18.3. A tin környezete	222
18.4. A tin parancsai	223

Előszó

Könyvünket több tanfolyam és a mindennapi élet tapasztalatai alapján két részre osztottuk: az első részt a teljesen kezdő, a UNIX (ejtsd *junix*) rendszerrel ismerkedő felhasználóknak szántuk. Ennek megfelelően az egyes fejezetekben az adott téma (program) használatáról csak a legfontosabb tudnivalókat foglaltuk össze. Megítélésünk szerint ez az egyes segédprogramokról szóló néhány oldal elegendő az elinduláshoz és a legegyszerűbb használathoz (a szerzők tapasztalata szerint az egy oldalt meghaladó leírásokat senki sem olvassa el¹...).

Az első részben ismertetett parancsok és programok részletes leírását a könyv második felében tárgyaljuk. Ezt a részt azoknak szántuk, akik jobban megszeretnék ismerni az egyes programokat és/vagy a rendszer részletesebb működését. Ebben a részben több olyan fejezet is található, amely már programozási alapismereteket is feltételez (pl. a programok fordításáról és futtatásáról szóló rész).

Mivel a könyv a fentiek értelmében két részre oszlik, a kereszthivatkozásokat tipográfiaiailag külön is jelöltük a margón.

A hivatkozott UNIX parancsokra és programokra külön betűtípus hívja fel a figyelmet. A parancsok ismertetésénél [] között jelöltük az esetlegesen megadható paramétereket, ilyenkor a paraméterek megadásához a [] jeleket nem kell beírni. Amennyiben a paramétereket | választja el, akkor közülük csak egyet lehet kiválasztani. A *dőlt betűkkel* kiemelt parancsrészeket nem kell (nem szabad) betű szerint beírni, ezeket mindig értelemszerűen helyettesíteni kell az aktuális értékekkel. Ugyanígy *dőlt betű* emeli ki a példáknál a magyarázatokat: természetesen ezeket sem kell (szabad) begépelni.

A gyakorlást segítő példákra egy másik jel hívja fel a figyelmet a margón. A példákban a felhasználó által *beírandó szövegrészt* így jelöltük. A példák különböző shellek (parancs értelmező programok) alatt másképpen működnek: mi a Korn-shell (`ksh`, l. a 9. fejezetet) használatát feltételezzük. Törekedtünk arra, hogy a példák (szinte) minden UNIX rendszerben működjenek. Tekintve, hogy nagyon

A kereszthivatkozásokat tipográfiaiailag külön is jelöltük a margón.

¹A szerzők sem olvassák el!

sok, részben különböző UNIX rendszer létezik, ezért előfordulhat, hogy az egyes parancsoknál megadott paraméterek némelyike nem, vagy nem úgy működik, mint az a könyvben szerepel.

Mivel a UNIX nyelve az angol, ezért megtartottuk a parancsok eredeti angol kiejtését (pl. `vi` ejtsd *víáj*). A számítástechnikában a köznapitól eltérően használt kifejezések és az idegen szavak magyarázatát a könyv végén, a ?? oldalán adjuk meg.

I. Alapismeretek

1. fejezet

A UNIX kialakulásának története

A UNIX operációs rendszert a Bell Laboratóriumban (AT&T) fejlesztették ki a 60-as években, eredetileg egy DEC PDP-7 típusú számítógépen. Az operációs rendszer szorosan kapcsolódik a C programozási nyelvhez, maga a rendszer döntően ezen a nyelven íródott és csupán minimális, a processor típusától függő gépi kódú részt tartalmaz. Mivel a legtöbb számítógépre ma már létezik C fordító, ezért a UNIX operációs rendszer könnyen átvihető ezekre a gépekre. Ez összecseng a UNIX eredeti céljaival. A fejlesztők, Dennis Ritchie és Brian W. Kernighan olyan operációs rendszert akartak létrehozni, amely:

- "hordozható"
- eszközfüggetlen állományrendszert tartalmaz
- modulrendszerű
- többfelhasználós
- több programot párhuzamosan futtat

Rugalmissága következtében a UNIX operációs rendszer gyors fejlődésnek indult, és egyre újabb verziói jelentek meg. Ugyanakkor egymástól független fejlesztések következtében megjelentek a különböző UNIX változatok. 1979-ben az AT&T fejlesztését felhasználva a Berkeley Egyetemen egy némileg ettől eltérő UNIX változatot dolgoztak ki (BSD UNIX), s manapság ennek a két főágnak a fejlesztése párhuzamosan folyik. Remélhetőleg azonban az eltérő változatok a szabványosítási tendenciák következtében a POSIX szabványban egyesülni fognak. A legismertebbek:

- AT&T System V R4
- BSD UNIX 4.4 (Ez a Berkeley University fejlesztése)
- AIX 3.2 (IBM)
- SCO UNIX (Santa Cruz Operation), INTERACTIVE UNIX (Kodak), Coherent UNIX, 386BSD (Ezek IBM PC kompatibilis gépeken futnak)
- SUNOS 4.1, SOLARIS 2.1 (SUN)
- ULTRIX 4.4, OSF/1 (DEC)
- HP-UX (Hewlett-Packard)
- IRIX 4.0 (Silicon Graphics)
- UNICOS (Cray)

2. fejezet

UNIX alapok

A UNIX alapvető sajátása az ún. multitasking. Ez azt jelenti, hogy minden feladatot külön eljárásként kezel, és minden más feladattól függetlenül tudja azt végrehajtani. Így egyidőben (pl. a DOS rendszerrel ellentétben) a gép több különböző programot is futtathat. Az operációs rendszer másik jellegzetessége, hogy igen nagy szabadságot ad a programok bemenő és kimenő adatainak kezelésében. Ennek keretében több futó feladat ki és bemenete össze is kapcsolható (pipe), illetve a bemenet és kimenet tetszőleges eszközre átirányítható.

A UNIX operációs rendszer egyszerre több felhasználót is ki tud szolgálni egyidőben, akár úgy is, hogy mindegyik felhasználó egyszerre több feladatot futtat alatta. A rendszer rugalmasságához tartozik, hogy mindegyik felhasználóhoz külön "méretre is szabható", mivel bejelentkezés során egy ún. shell (burok) alakul ki a felhasználó körül, melynek paramétereinek egy része szabadon állítható. Shellekből többfajta is van egy UNIX rendszeren, de a gyakorlott felhasználók akár maguk is írhatnak ilyen burkokat.

A modulrendszer következtében igen sok UNIX parancs van (300 fölött). Ráadásul mindegyiknek vannak kapcsolói, melyekkel egy-egy parancs hatását lehet módosítani. Ilyen nagyszámú parancs használata már igen körülményes, és nem tartható fejben, ezért a UNIX rendszer része egy beépített kézikönyv (l. a `man` parancsot a 22. oldalon). A UNIX rendszer talán a legnagyobb hasonlóságot a DOS rendszerrel mutatja.¹

¹Pontosabban az állítás fordítva igaz: a DOS eredetileg a UNIX és a VMS operációs rendszerek keresztezéséből és teljes „lecsupaszításából” született...

2.1. A UNIX rendszerek elérése

2.1.1. Bejelentkezés a UNIX rendszerbe

A belépéshez felhasználói névvel és jelszóval(kulcsszó,*password*) kell rendelkezünk a megadott gépen. A felhasználói név egy — általában 8 betűnél rövidebb — szó, amivel a rendszer azonosítja a felhasználókat (ez lehet a család- vagy keresztnév, becenév, stb.). A kulcsszó a felhasználói nevet védi: csak a megfelelő jelszóval lehet a rendszerbe jelentkezni. A jelszó így megvéd minket és állományainkat az illetéktelen behatólól. A felhasználókat a rendszer csoportokba osztja: mindenki egy vagy több csoporthoz tartozhat. Ennek a hozzáférési jogoknál lehet szerepe: pl. bizonyos állományokat csak egy adott csoport tagjai érhetnek el, stb.

A UNIX a felhasználói név és jelszó alapján dönti el, hogy jogosultak vagyunk-e a gép használatára. Felhasználói nevet és kezdeti jelszót a gép rendszergazdájától lehet kapni. A jelentkezés a következő lépésekből áll:

- ellenőrizzük, hogy be van-e kapcsolva a terminál vagy PC.
- hálózat használata esetén futtassuk le a megfelelő kommunikációs programot (pl. `telnet ludens.elte.hu`, amennyiben a `ludens.elte.hu` gépre szeretnénk jelentkezni). Várjunk, amíg a jelentkezési prompt meg nem jelenik (ezt megelőzheti egy rövid, pl. a gép nevét és típusát tartalmazó információ).
- A `login:` promptnál gépeljük be a felhasználói nevünket.
- Ha a felhasználói névhez jelszó tartozik, akkor azt a rendszer a `Password:` prompt megjelenítésével kéri. A jelszó biztonsági okokból begépeléskor nem jelenik meg a képernyőn. (Egyes esetekben bizonyos felhasználói nevek esetén nincs jelszó. Ilyenkor általában csak korlátozott felhasználást engedélyez az operációs rendszer).

Ha rossz felhasználói nevet, vagy jelszót adtunk meg, erről a rendszer hibüzenettel értesít minket. Ilyenkor a következő promptnál korrigálhatunk. Általában bizonyos számú hibás jelentkezés után a gép bontja a hálózati kapcsolatot. A UNIX operációs rendszer megkülönbözteti a kis és nagy betűket, ezért **nagyon figyeljünk**, hogy a felhasználói nevünkben, illetve jelszavunkban milyen betűk fordulnak elő.

Példa:

A következő bejelentkezésben a felhasználói név: Joska, a jelszó pedig: 2pistA.

A helyes bejelentkezési eljárás:

```
login: Joska 
```

```
Password: 2pistA 
```

\$ *ez természetesen nem látszik a képernyőn*

\$ *ez már a shell promptja, bejelentkeztünk*

Helytelen bejelentkezés a következő (a kis- és nagybetűt megkülönbözteti a rendszer):

```
login: joska 
```

```
Password: 2pistA 
```

```
You entered an invalid login name or password.
```

```
login:
```

A jogosulatlan belépési próbálkozások ellen kétféleképpen védekezik a rendszer: egyrészt értesíti a felhasználót, hogy mikor és melyik terminálról próbáltak meg sikertelenül belépni a felhasználói azonosítójával, másrészt a rendszergazda számára ezeket az információkat egy állományban tárolja, aki szükség esetén ennek alapján megfelelő ellenintézkedéseket tehet.

2.1.2. Jelszó használata

A jelszó megváltoztatására a UNIX rendszerben a `passwd` parancs szolgál. Az előbbi példa jelszavát a következőképpen változtathatjuk meg:

```
$ passwd 
```

```
Changing password for "Joska"
```

```
Joska's Old password: 2pistA 
```

ez nem jelenik meg

```
Joska's New password: 3juli 
```

ez sem jelenik meg

```
Re-enter Joska's new password: 3juli 
```

a második beírás a véletlen elütést védi ki

A fenti példából látható, hogy a jelszavakat csak a korábbi ismeretében változtathatjuk, ilyen módon is védve vagyunk az idegen beavatkozástól. Amennyiben hibásan gépeltük be a jelszavunkat, vagy az új jelszó kétszeri begépelése nem egyezik meg, hibüzenetet kapunk és a változtatás nem következik be.

A nevünket használó jogosulatlan belépések megakadályozásához tartsuk be a következőket:

- A jelszó nagyon fontos biztonsági eszköz a UNIX rendszerben, tartsuk jelszavunkat titokban!

- Ne használjuk jelszóként nyilvánvaló szavakat, pl. keresztnév, gyermek neve, stb.! Illesszünk a betűk közé számo(ka)t is!
- A rendszergazda beállíthatja, hogy a jelszó egy adott idő múlva érvényét veszítse. Ha ez már küszöbön áll, meg kell változtatni jelszavunkat. Gyakori, hogy új felhasználóként az első belépéskor a rendszer felszólít minket a jelszó azonnali megváltoztatására, amit ezután már a rendszergazda sem ismer.
- Amennyiben elfelejtettük a jelszót, vagy más ok miatt nem tudunk belépni, akkor keressük fel a rendszergazdát, aki vagy új jelszót ad ki a számunkra, vagy kitörli a jelszót. Ilyenkor célszerű azonnal megváltoztatni az új jelszót.

2.1.3. Kijelentkezés

Mindig jelentkezzünk ki a gépből, bármilyen rövid időre is hagyjuk azt szabadon. Kijelentkezni többféleképpen lehet a rendszerből. Lehet az `exit` paranccsal, a `Ctrl` és `D` billentyűk egyidejű lenyomásával, illetve bizonyos esetekben a `logout` utasítással².



Példa:

```
$ logout       vagy
login:
$ exit 
You have stopped jobs  a rendszer figyelmeztet, hogy a háttérben még
                        programjaink futnak
$ exit       a kilépéshez meg kell ismételni a parancsot
login:
```

A kijelentkezés után a rendszer azonnal felkészül a következő bejelentkezésre.

2.1.4. A használt shell beállítása (`chsh` parancs)

A bejelentkezés után egy ún. burok fogad minket. A rendszergazda több különböző program közül választja ki az új felhasználó kezdeti burkát. Ha ez nekünk nem felel meg, a `chsh` paranccsal átállíthatjuk ezt. A parancs felsorolja a rendszerben található burkokat, kiírja, hogy éppen melyiket használjuk. Ezután megkérdezi, hogy meg akarjuk-e változtatni. Amennyiben igen, meg kell adnunk az új shell teljes nevét (pl. `/bin/ksh`). Az új shell a következő bejelentkezéstől él. Megjegyezzük, hogy általában ezt a parancsot csak ritkán szükséges kiadni. Egy shellt a `chsh` helyett közvetlen futtatással (névének beírásával) próbáljunk ki!

²Ezt akkor kell használnunk, ha a `chsh`-t, vagy hasonló burkot futtatunk.

2.2. A UNIX shell parancsok

A UNIX shell egy olyan interaktív program, amely egyszerre tölti be a parancs értelmező (*command interpreter*) és az adatközvetítő szerepét. A különböző rendszerszolgáltatásokat, például nyomtatás, másolás, megjelenítés, keresés, szerkesztés, sorbaállítás, törlés, kommunikáció, stb. a shellnek eljuttatott parancsokkal valósíthatjuk meg.

A shell prompttal jelenkezik be, ami általában a `$` jel, ez után lehet a parancsokat begépelni. A parancsokat általában egy sorban adjuk be: először a parancsot, majd az esetleges módosító kapcsolókat és argumentumokat. A parancsot az `Enter` billentyűvel zárjuk.

Vigyázzunk, mert a UNIX rendszer **megkülönbözteti a nagy és kisbetűket!** A parancsok nagy részét kisbetűvel kell írni.

Példaként vegyük a `who` parancsot, amely kilistázza az összes bejelentkezett felhasználót, a hozzájuk tartozó terminállal, valamint a bejelentkezési időpontjukkal együtt:

```
$ who Enter
Joska          tty2          Jan 02 10:02
root           console      Jan 02 07:56
Zoli           tty4         Jan 22 22:45
```

Ha a shell parancs kiadása közben lenyomjuk az `Enter` billentyűt, akkor a rendszer vagy hibüzenettel, vagy — megkezdett idézőjelek esetén — egy ún. másodlagos prompttal (ez általában a `>` jel) jelzi. A másodlagos prompt esetén a rendszer várja annak lezárását. A másodlagos promptnál befejezhetjük a parancsot, de meg is szakíthatjuk azt a `Ctrl-C` billentyűk lenyomásával. Hibüzenet esetén a begépelte parancs elveszik.

Például az `echo` parancssal kiírhatjuk a képernyőre az idézőjelek között szereplő szöveget:

```
$ echo "Ennyi az Enter lenyomjuk az Enter-t még mielőtt végeznénk
>egesz" Enter befejezzük a parancsot
Ennyi az
egesz
$
```

2.2.1. A parancs végrehajtásának megszakítása

Bizonyos esetekben szükség lehet a parancs leállítására annak futása közben: például ha egy parancs már túl hosszú ideje fut, vagy közben eszünkbe jut, hogy nem

ezt az utasítást akartuk futtatni.

A parancs leállítására a `Ctrl` és `C` billentyűk egyidejű lenyomása szolgál. Ezzel megszakítjuk a parancs futását, és visszakapjuk az eredeti shell promptot. A parancs futását — amikor az adato(ka)t kérdez be — a `Ctrl` és `D` egyidejű lenyomásával is megszakíthatjuk. A `Ctrl-D` a UNIX operációs rendszerben az adatállomány végét jelzi. Parancssorban alkalmazva a rendszer befejezi az adott parancssorozat végrehajtását, azaz így kilépünk a rendszerből.³ Ha egymás után többször alkalmazzuk ezt a módot, akkor először a futó programot, aztán a shellt állítja le a rendszer.

2.2.2. A parancsok javítása

Ha a parancsot hibásan gépeltük be, akkor a UNIX rendszer nem tudja végrehajtani és hibaüzenetet küld. Például:

```
$ whoa Enter
whoa: not found a shell válasza: nincs ilyen parancs
$
```

A gépelési hiba kijavítására a `Backspace` billentyű (esetleg a `Ctrl-H` kombináció, vagy a `Delete` `←` felírral ellátott gomb) szolgál. Ez törli a kurzor előtt álló karaktert. Bizonyos esetekben (pl. PC-s hálózati programnál) célszerű az általunk használt törlőgomb beállítása. Ezt pl. a `Backspace` gombra a

```
$ stty erase Backspace Enter
```

utasítással tehetjük meg.

A kurzort mozgó gombok a UNIX rendszerben általában csak „különleges” shell programok (pl. `tcsh`, `bash`) esetén használhatók a parancsok javítására!

Az egész parancssor törlésére a `Ctrl-U` billentyűk szolgálnak. Ezt akkor használjuk, ha túl sokat hibáztunk a parancs beírásakor.

2.2.3. Az utasítások kapcsolói

A kapcsolók az alaputasítás végrehajtását módosítják. Szinte minden utasítás rendelkezik kapcsolóval. A kapcsolókat az utasítás után a mínusz (-) vagy ritkábban a plusz (+) jellel elválasztva adjuk meg. A kapcsolók általában egy betűből állnak. A UNIX operációs rendszer itt is megkülönbözteti a kis- és nagybetűket!

³Ez nem mindig van így, pl. `csh` vagy `tcsh` shell esetén esetén az `ignoreeof` környezeti változó értékétől függően működik.

A kapcsolókat az utasítástól illetve annak argumentumaitól szóköz(ök) választják el. Ha az egyes kapcsolókat külön-külön adjuk meg, akkor azok közé szintén szóköz(ök) kelljenek.

A következő példákban egy feltételezett felhasználói könyvtárban nézzük az állományok listáját⁴:

Az `ls` parancs az aktuális könyvtárban levő állományok felsorolását adja:

```
$ ls 
choice cm kocsi log test Mail
```

A `-l` kapcsoló módosítja az `ls` utasítás alapértelmezését, és állományok részletes listáját írja ki (ami oszloponként a következőt tartalmazza: védelmi (hozzáférési) státusz, link szám, a file birtokosa (felhasználó és csoport), méret byte-ban, az utolsó változtatás ideje és a file neve):

```
$ ls -l 
total 20
-rw-w--w--  2 galahad  lovag    3910 Oct 26 1992  choice
-rw-w--w--  1 galahad  lovag         21 Oct 26 1992  cm
-rw-w--w--  1 galahad  lovag    198 Oct 26 1992  kocsi
-rw-w--w--  1 galahad  lovag         5 Dec 05 1992  log
-rwxr-xr-x  1 galahad  lovag    31 Oct 26 1992  test
drwx--x--x  2 galahad  lovag    512 Jul 06 1992  Mail
```

A `-a-l` kapcsoló részletesen kilistázná a könyvtárban levő összes állományt (beleértve a `.`-tal – ponttal – kezdődőeket is⁵), ha az `ls -a-l` helyett az `ls -a -l` (vagy az ezzel azonos az `ls -al`) parancsot írtunk volna, azaz

```
$ ls -a-l 
bad switch
```

helyett a következő a helyes:

```
$ ls -a -l 
total 32
drwxr-xr-x  3 galahad  lovag    512 Dec 05 1992  .
drwxr-xr-x  9 galahad  lovag   1024 Mar 05 11:45  ..
-rw-r--r--  1 galahad  lovag    166 Nov 02 1992  .index
```

⁴Amennyiben ezeket a parancsokat saját könyvtárunkban kipróbáljuk, az eredmények általában nem fognak megegyezni az itt szereplőkkel.

⁵Ezek általában speciális állományok, ezért nem listázza ki őket az `ls` alapértelmezésben.

```
-rw-w--w-- 1 galahad  lovag  3910 Oct 26 1992  choice
-rw-w--w-- 1 galahad  lovag    21 Oct 26 1992  cm
-rw-w--w-- 1 galahad  lovag  198 Oct 26 1992  gral
-rw-w--w-- 1 galahad  lovag    5 Dec 05 1992  log
-rwxr-xr-x 1 galahad  lovag    31 Oct 26 1992  test
drwx--x--x 2 galahad  lovag    512 Jul 06 1992  Mail
```


 Érdekes kivétel a következő parancs:

```
$ who am i 
galahad  tty2      Jan 02 10:02
```

amely kiírja az éppen aktuális felhasználói nevünket (ez nem is olyan felesleges pl. több gép egyszerre való használatakor).

2.2.4. Az utasítások argumentumai

A kapcsolók mellett a UNIX operációs rendszer utasításainak nagy része paramétereket is elfogad. Amíg a kapcsolók módosítják az adott parancs hatását, addig a paraméterek határozzák meg, hogy mire vonatkozik az utasítás. A parancs után először a kapcsoló(ka)t, majd a paramétereket adjuk meg szóközzel elválasztva.

 Általában több paraméter is megadható. A már korábban is említett `echo` parancs például az argumentumát jeleníti meg a képernyőn:


```
$ echo haliho  parancs és paraméter
haliho          az echo parancs által megjelenített paraméter
```

2.2.5. A `man` parancs

A UNIX operációs rendszerben a parancsokról részletes (angol nyelvű) leírást kaphatunk a `man` (manual) parancs használatával. A `man` parancs után megadva a kérdéses utasítás nevét a billentyűre laponként, -re soronként írja ki az információt a rendszer. Ha abba akarjuk hagyni, a billentyűt nyomjuk meg. Ha egy *Kulcsszó*t szeretnénk keresni a kézikönyvben, akkor a

```
$ man -f Kulcsszó
```

utasítással azoknak a fejezeteknek a rövid listáját kapjuk meg, amiben ez a kulcsszó szerepel.

 Például az `ls` parancs leírását a

```
$ man ls 
```

utasítással kaphatjuk meg.

3. fejezet

Állományok használata

3.1. A UNIX állományok

A számítógépeken az információt file-ban (állományban) tároljuk. Az információ lehet szöveg, adat, kép vagy akár hang is. Állományban tárolja a rendszer a futtatható programokat is. A UNIX operációs rendszerben az állományok maximális méretét általában a rendszergazda korlátozza. A fizikai eszközök tárolókapacitása szintén határt szab az állományokban tárolható információknak.

A UNIX operációs rendszer az állományokra a nevükkel hivatkozik. Az állományok nevét bizonyos minimális megkötésekkel és fenntartott nevek kivételével szabadon választhatjuk meg. A pont (.) kezdetű file neveket a burok (shell) ill. az egyik leggyakrabban használt parancs, az `ls` különlegesen kezeli. Ezek általában ún. inicializáló állományok, és — hacsak külön nem kérjük — nem látszódnak listázáskor.

Minden állomány valamelyik felhasználóhoz tartozik, aki általában az, aki létrehozta a állományt. Az állományok tipikusan a merevlemezen helyezkednek el, de hajlékony mágneslemezen, optikai lemezen, vagy mágnesszalagon is lehetnek. A UNIX operációs rendszer feljegyzi minden file utolsó módosításának és utolsó használatának idejét.

A UNIX rendszerekben az állománytörléssel felszabadított helyet a rendszer akár azonnal is lefoglalhatja, így itt nincsen *undelete* típusú parancs a törölt állomány visszaállítására. A törlés a UNIX-ban **VÉGLEGES!**

3.1.1. Az állományok típusai

Háromfajta állomány létezik a UNIX operációs rendszerben:

- a „közönséges” állományok tartalmazhatnak szövegeket, adatokat, adatbázisokat, de akár futatható programokat is. Általában ezeket az állományokat használjuk.
- az ún. speciális állományok az operációs rendszer számára tartalmaznak információt. Ezek általában egy rendszer adott hardware eszközeinél használatosak. Az átlagos felhasználó csak ritkán használja ezeket.
- a könyvtárállományok segítik az állományok rendszerezését (a UNIX könyvtárrendszerről a 37. oldalon található részletes leírás).

A rendszer könyvtárai a / (gyökér) könyvtárból indulnak ki, és fa-struktúrát alkotnak. A felhasználó saját maga alakíthatja ki saját könyvtárállományait (általában csak a bejelentkezési könyvtára alatt, de bizonyos esetekben (pl. scratch disk¹) máshol is hozhatunk létre könyvtárat).

3.1.2. Állományok létrehozása

A legegyszerűbb módon a (a későbbiekben tárgyalandó) `cat` parancsot használhatjuk állományok létrehozásához. A későbbi példákhoz hozzuk létre a `gyakorlas` állományt a következő parancssorozattal (csak a *vastagbetűs* részt kell begépelnünk, a szóközzel együtt):

```
$ cat > gyakorlas <<ALLJ 
> Ez egy proba sor ami csak betuket tartalmaz 
> Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42 
> Ebben a sorban nehany furcsa karakter van % @ ! 
> Itt mar szamok is vannak a furcsa karakterekkel együtt 1 % @ 42 ! 
> A sorban a UNIX metakarakterei is itt szerepelnek $> ' es a * 
> Most mindent együtt szerepeltetunk ebben a sorban 'vege' $1 * 42 
> ALLJ 
```

Ellenőrzésképpen írassuk ki az állományt:

```
$ cat gyakorlas 
Ez egy proba sor ami csak betuket tartalmaz
Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42
```

¹A scratch disken az állományok általában csak néhány napig (esetleg hétig) maradnak meg, utána a rendszer törli őket.

Ebben a sorban néhány furcsa karakter van % @ !

Itt már számok is vannak a furcsa karakterekkel együtt 1 % @ 42 !

A sorban a UNIX metakarakterei is itt szerepelnek \$ > ' es a *

Most mindent együtt szerepeltetünk ebben a sorban 'vege' \$1 * 42


Ugyanígy hozzuk létre a rovidfile állományt is

```
$ cat > rovidfile <"ALLJ" 
> Ez egy egyszerű szövegfájl, ami semmire sem jó, 
> csak hogy tanuljunk vele a UNIX operációs rendszert 
> használatát. 
> ALLJ 
```


3.2. Állományok kilistázása

Az állományok kilistázására az `ls` parancs szolgál. A lista formátumát sokféle kapcsolóval módosíthatjuk.


Példák:

Hat file van az adott könyvtárban: choice, cm, kocsi, log, test és a Mail: 


```
$ ls 
choice cm kocsi log test Mail
```

Leellenőrizzük, hogy az adott choice file megtalálható-e az adott könyvtárban: 

```
$ ls choice 
choice
```

Ha az adott file nincs benne az adott könyvtárban, akkor hibüzenetet kapunk: 


```
$ ls rossz-szoveg 
rossz-szoveg: No such file or directory
```

A parancs a `-s` kapcsoló hatására kiírja a állományok nevét, és diszkblokkokban² mért méretét: 

```
$ ls -s 
```


²Ez a helyi rendszertől függ, lehet 512 byte, de lehet 1, 2 sőt 4 kbyte is.

```
total 1475
1 cm      1 log      1 Mail
4 choice  1 cm      1 test
```


-  A parancs a `-l` kapcsoló hatására kiírja az állományok, ill. könyvtárak (könyvtár esetén az első betű `d`, míg közönséges állománynál `-`) hozzáférési jogait (lásd később), a rájuk vonatkozó kapcsolatok számát, a tulajdonosuk nevét és felhasználói csoportját³, a hosszát byte-okban, az utolsó módosításuk idejét, végül a nevét:

```
$ ls -l 
total 5
-rw-w--w--  1 galahad  lovag      3910 Oct 26 1992  choice
-rw-w--w--  1 galahad  lovag           21 Oct 26 1992  cm
-rw-w--w--  1 galahad  lovag      198 Oct 26 1992  gral
-rw-w--w--  1 galahad  lovag           5 Dec 05 1992  log
-rwxr-xr-x  1 galahad  lovag      31 Oct 26 1992  test
drwx--x--x  2 galahad  lovag      512 Jul 06 1992  Mail
```

A Mail könyvtár, minden állománynak galahad a tulajdonosa, valamint minden állomány a lovag csoportba tartozik, a gral hossza 198 byte és 1992. október 26-án volt utoljára módosítva.

-  A parancs a `-a` kapcsoló hatására kiírja az aktuális könyvtárban levő összes file-t, beleértve a bizonyos szempontból rejtett, ponttal kezdődő állományokat is:

```
$ ls -a 
.      ..      .index  choice  Mail
cm     gral    log     test
```

-  A `-F` kapcsoló hasznos lehet akkor, ha meg szeretnénk állapítani az aktuális könyvtárban levő állományok típusát:

```
$ ls -F 
cm     gral    log     choice  Mail/    test*
```

A file neve után álló / arra utal, hogy a bejegyzés könyvtár, a * pedig arra, hogy az végrehajtható program.

Hasznos az állományok elnevezésekor a következőkre figyelni:

- állományok létrehozásánál ügyeljünk arra, hogy a filenév egyedi legyen egy adott könyvtáron belül, különben felülírjuk az adott név alatt már létező állományt. Különböző könyvtárakban természetesen lehetnek azonos nevű

³BSD alapú rendszerekben ehhez esetleg a `-g` kapcsolót is használni kell.

állományok. Például, ha létrehoztuk a `/usr/galahad/test` és a `/usr/galahad/Mail` könyvtárakat, akkor mind a kettőben tarthatunk egy-egy `proba` nevű állományt, melyek tartalma különböző.

- Válasszunk beszédes neveket, hogy a későbbiekben is tudjuk, mit tartalmaz az adott file.
- Használhatunk kis- és nagybetűket, illetve ezek kombinációját is az állománynevek megadásakor. Ügyeljünk arra azonban, hogy a UNIX operációs rendszer megkülönbözteti a kis- és nagybetűket, így az állománynévre való hivatkozáskor ezeket helyesen kell begépelni.
- Az állomány nevében a betűkön, számokon kívül bármelyik ASCII karakter használható, kivéve az ASCII 0, és a / karaktereket. A speciális karakterek közül leggyakrabban a pont (`.`), az aláhúzás (`_`), valamint a kötőjel (`-`) használatos.

FIGYELEM! Ne kezdjük az állomány nevét kötőjellel (`-`), mert ez zavaró lehet a burok (shell) számára, és kapcsolóként értelmezi az utána jövő karaktereket.

- Ne használjunk állománynévben olyan speciális karaktereket, mint a vessző (`,`), az idézőjelek (`'` vagy `"`), mivel ezek különleges jelentéssel bírnak a shellekben.
- Az állománynév minimum egy, maximum 256 karakter hosszú lehet (a maximumot bizonyos állományrendszerekben 14-16 karakterre korlátozzák).
- hasznos lehet a ugyanolyan végződések használata az állománynevekben. Például a levélállományoknak a `.lev` végződést adhatjuk, így pl. a `galahad` nevű felhasználótól kapott levelet a `galahad.lev` állományra keresztelhetjük el.
- néhány program automatikusan generál végződéseket. Például a C fordítóprogram létrehoz egy ún. `object` állományt, amit a `.o` végződéssel lát el. A forráskódú programokat a kialakult szokás szerint az adott programozási nyelvre utaló végződéssel látjuk el. Így például a C nyelvű programok a `.c`, a FORTRAN nyelvű programok a `.f` végződést kaphatják.

3.3. Állományok tartalmának megjelenítése

Sokszor szükségünk van az állományok tartalmának megjelenítésére a képernyőn. Erre a célra ugyan szövegszerkesztőket is használhatunk (pl. `vi`-t), de sokszor

hatékonyabb a `cat`, a `pg` vagy a `more` parancsok használata. A `cat` parancsot célszerű rövidebb állományok esetén használni, melyek tartalma elfér egy képernyőoldalon, míg hosszabb állományok megjelenítésére inkább a `pg` és a `more` parancs a célszerű.

A `pg` és a `more` parancsokat csak szövegállományokra adjuk ki. A nem szöveg típusú (bináris vagy könyvtár típusú) állományokból ilyen módon nem nagyon lehet információt kiolvasni, sőt a bináris állományokban található speciális karakterek elállíthatják terminálunkat.

3.3.1. A `cat` parancs

Kisebb állományok megjelenítésére a UNIX operációs rendszerben a `cat` parancsot használjuk az állomány(ok) nevével, mint paraméterrel. Egyszerre több állománynév is megadható, ilyenkor a rendszer a megadott sorrendben jeleníti meg az állományok tartalmát. A paraméterlistában használhatjuk a burok (shell) helyettesítő karaktereit, a csillagot (*), amely egy tetszőleges stringet helyettesít, vagy a kérdőjelet (?), amely egy tetszőleges betűt helyettesít (ezekről a 3.7 fejezetben, a 34. oldalon. található bővebb magyarázat).

Példák:

- 📎 A 24. oldalon látható módon, a `gyakorlas` állományhoz hasonlóan a `rovidfile` állomány tartalmát így írathatjuk ki :

```
$ cat rovidfile 
```

Ez egy egyszerű szövegfile, ami semmire sem jó, csak hogy tanuljuk vele a UNIX operációs rendszer használatát.

- 📎 Két (vagy több) állománynévvel a megadott sorrendben, elválasztás nélkül íródna ki az állományok:

```
$ cat gyakorlas rovidfile 
```

Ez egy próba sor ami csak betűket tartalmaz

Ebben a sorban már vannak számok is , például az 1 és a 42

Ebben a sorban néhány furcsa karakter van % @ !

Itt már számok is vannak a furcsa karakterekkel együtt 1 % @ 42 !

A sorban a UNIX metakarakterei is itt szerepelnek \$ > ' és a *

Most mindent együtt szerepeltetünk ebben a sorban 'vege' \$1 * 42

Ez egy egyszerű szövegfile, ami semmire sem jó,

csak hogy tanuljuk vele a UNIX operációs rendszer használatát.

3.3.2. Nagyobb állományok megjelenítése

Nagyobb állományok megjelenítésekor hasznos a `pg` vagy a `more` parancs, mivel ezek formázva jelenítik meg az állományok tartalmát. Egyszerre csak egy képernyőoldali anyag jelenik meg az állományból, és utána a `pg`, ill. a `more` parancs megkérdezi, hogy mit csináljon. Lehet tovább olvasni az állományt, befejezni az állomány olvasását, de akár visszafelé is mozoghatunk az állományban.

A hosszú állományok tartalmának megjelenítésére általában rendelkezésre áll a `less` parancs is, amely a `more` program kibővített változatának tekinthető. Az állományok elejének megtekintésére a `head` parancsot, végének megtekintésére a `tail` parancsot használhatjuk, amelyek alapértelmezésben 10 sorra vonatkoznak.

A `pg` utasítás

Az állomány megjelenítésére a `pg` parancs paramétereként a megjelenítendő állomány(ok) nevét kell megadni. Az első képernyőoldal után a képernyő alján egy kettőspont (`:`) prompt jelenik meg.

Például nézzük meg a `gyakorlas` állományt:

```
$ pg gyakorlas 
```

```
Ez egy proba sor ami csak betuket tartalmaz
```

```
Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42
```

```
Ebben a sorban nehany furcsa karakter van % @ !
```

```
Itt mar szamok is vannak a furcsa karakterekkel egyutt 1 % @ 42 !
```

```
A sorban a UNIX metakarakterei is itt szerepelnek $ > ' es a *
```

```
:
```

Az esetleges következő lap megjelenítéséhez nyomjuk le az billentyűt.

Átugorhatunk egyszerre több lapot is, ha a (`:`) promptnál a plusz (`+`) billentyűt és egy számot nyomunk le. A szám értéke mondja meg, hogy hány lapot óhajtunk átugrani. Például három lap átugrásához `+3`-at gépelünk be, majd leütjük az -t.

Visszalapozni az előzőhöz hasonlóan a mínusz (-) utasítással tudunk. Például két oldalt a `-2` `[Enter]` beütésével lapozunk vissza.

A megjelenítés befejezését a `q` `[Enter]` begépelésével érjük el. Ezután a shell promptnak kell megjelennie. Ha az állomány végére érünk, a `pg` parancs befejezi a megjelenítést, és visszatér a shellbe.

A `pg` parancs még számos, egyéb lehetőséget is kínál, például keresés a szövegben, file műveletek, parancsok végrehajtása a shellben, amelyeket a `:` promptnál beírt `h` `[Enter]` (help) információ lekérésével nézhetünk meg.

A `more` utasítás

A `pg` parancshoz hasonlóan működik a `more` parancs is:

```
$ more gyakorlas [Enter]
Ez egy proba sor ami csak betuket tartalmaz
Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42
Ebben a sorban nehany furcsa karakter van % @ !
Itt mar szamok is vannak a furcsa karakterekkel egyutt 1 % @ 42 !
A sorban a UNIX metakarakterei is itt szerepelnek $ > ' es a *
Most mindent egyutt szerepeltetunk ebben a sorban 'vege' $1 * 42
--More-(99%)
```

A parancs megjeleníti a file tartalmát a képernyőn, és a képernyő legalsó sorában a képernyőn már látott állományrész hosszát jelzi százalékban.

A `[Szóköz]` billentyű leütésével laponként, az `[Enter]` leütésével soronként tudunk haladni az állományban.

Az előre lapozást az `[f]`, a visszafelé lapozást pedig a `[b]` billentyű leütésével is végezhetjük. Itt is a `[q]` billentyű leütésével szakíthatjuk meg a `more` futását.

A `more`, a `pg` parancshoz hasonlóan számos egyéb funkcióval is rendelkezik (keresés, shell parancsok), amit a `[h]` (help) billentyű leütésével (help) kapott táblázatban nézhetünk meg.

Egy érdekes és sokszor hasznos lehetősége a `more` parancsnak, hogy a `[v]` billentyű leütésével a `vi` editort lehet meghívni. Ekkor a `vi`-t használhatjuk a file kezelésére, és akár át is szerkeszthetjük az állományt.

Itt hívjuk fel a figyelmet arra, hogy — bár ez rendszerről rendszerre eltérő lehet — a `man` parancs (manuál) leggyakrabban szintén a `more`-t használja a szöveg megjelenítésére.

3.4. Állományok másolása, átnevezése és törlése


3.4.1. Állományok másolása

Állományokról másodpéldány készítésére a `cp` parancs szolgál. Például a 

```
$ cp gyakorlas masolat 
```

a `gyakorlas` állomány tartalmát átmásolja a `masolat` nevű állományba.


Amennyiben létezik olyan nevű állomány, amelyekre rá akarunk másolni, használjuk a `-i` kapcsolót. Ez megerősítést kér minden olyan állománynál, amelyet felül kellene írni. Vigyázzunk a `cp` parancs használatakor, mivel a másolás során — egy régebbi állományra való másolással — könnyen megsemmisíthetünk adatokat ha nem használjuk ezt a kapcsolót!

Adjuk ki újra az előző példa parancsát a `-i` kapcsolóval 

```
$ cp -i gyakorlas masolat 
```


```
overwrite masolat? y 
```

A rendszer ilyenkor megkérdezi, hogy felülírja-e a `masolat` állományt. Amennyiben erre `y`-nal (yes) válaszolunk, akkor felülírja, egyébként a másolás nem történik meg.

A `cp` parancsnak egyszerre több állománynevet is megadhatunk, sőt a shell ún. kifejtő mechanizmusa révén helyettesítő karaktereket is használhatunk (l. 34. oldal). A parancs így használható pl. egy könyvtár tartalmának átmásolására egy másik könyvtárba⁴ (kivéve a `.`-al kezdődő állományokat). Például a 

```
$ cp * masik 
```

átmásolja az aktuális könyvtárban levő összes állományt a `masik` nevű alkönyvtárba. A parancs a könyvtárakat nem másolja.

Könyvtárakat a teljes alkönyvtárrendszerükkel együtt a `-r` kapcsoló⁵ segítségével másolhatunk át. Például a 

```
$ cp -r /usr/user /u/user/tmp 
```

átmásolja a `/usr/user` könyvtár összes állományát és az abból nyíló összes alkönyvtárat a `/u/user/tmp` könyvtárba. Ezáltal a teljes `/usr/user` könyvtárban lévő állománystruktúra megjelenik a `/u/user/tmp` könyvtárban is.

A `cp` utasítás – ugyanúgy, mint a többi állománykezelő parancs – hibajelzéssel áll meg, ha nincs engedélyünk arra, hogy az adott helyre írjunk (l. a 3.9 fejezetet, a 42. oldalon).

⁴A UNIX könyvtárrendszerrel a 37. oldalon található részletes leírás.

⁵Bizonyos UNIX rendszereknél ez a kapcsoló nem működik.

3.4.2. Állományok átnevezése (mv)

Állományok átnevezésére vagy átmozgatására az `mv` parancs szolgál. A parancs kérdés nélkül felülírja a már létező állományokat, de ennek elkerülésére használhatjuk a `cp` parancsnál már említett `-i` kapcsolót.

Ha meg szeretnénk tartani a file eredeti nevét más könyvtárba való másolásnál, akkor elegendő, ha az `mv` utasítás céljaként a könyvtárat adjuk meg. Például a

```
$ mv level /u/user/test 
```

átmozgatja a `level` állományt az aktuális könyvtárból a `/u/user/test` könyvtárba az eredeti `level` néven.

Az `mv` parancsnak — a `cp` parancshoz hasonlóan — több állománynevet is megadhatunk, de ilyenkor a legutolsó argumentumnak (a célnak) könyvtárnak kell lennie.

3.4.3. Állományok törlése

Állományokat az `rm` paranccsal törölhetünk. Egyszerre több állományt is megadhatunk, sőt a shell kifejtő mechanizmusa révén helyettesítő karaktereket is használhatunk. Az `rm` parancs kérdés nélkül töröl minden megadott állományt. A `-i` kapcsolóval a törlés előtt — a `cp`-hez hasonlóan — az `rm` rákérdez, és törlési megerősítést vár minden egyes állományra. Amennyiben biztosítani szeretnénk

magunkat a véletlen törlések ellen, akkor az

```
alias rm="rm -i"
```

parancssort kell a `$HOME/.profile` állományba beírni. Ekkor a `rm` automatikusan használni fogja a `-i` kapcsolót.

Az `rm` parancsnak egyszerre több állománynevet is megadhatunk, sőt a shell ún. kifejtő mechanizmusa révén helyettesítő karaktereket is használhatunk.


A `-r` kapcsoló rekurzív törlést eredményez, azaz a megadott könyvtártól lefelé a teljes alkönyvtár rendszerben sor kerül a törlésre, az utasítás még a könyvtárakat is kitörli. A

```
$ rm -r * 
```

a UNIX egyik „leghatásosabb” parancsa, az adott könyvtártól kezdve felszámolja a teljes állománystruktúrát, és még az alkönyvtárakat is kitörli! Csak akkor adjuk ki, ha valóban törölni akarunk mindent! A UNIX rendszerekben az állománytörléssel felszabadított helyet a rendszer akár azonnal is lefoglalhatja, így itt **nincsen undelete** típusú parancs a törölt állomány visszaállítására!

3.5. Szöveg keresése: a `grep` parancs

A `grep` parancssal gyorsan megkereshetünk egy adott szöveget (karaktermintát) az állományokban. A keresett minta lehet egy szó, vagy betűk és számok valamilyen csoportja. Ha a `grep` parancs megtalálja a mintát, akkor a mintát tartalmazó sort kiírja a képernyőre. A parancs használatához meg kell adni a keresett karaktersort és azt az állományt (vagy állományokat), amely(ek)ben keresni akarunk. Ha a karaktorsor több szóból áll, vagy szóköz(öke)t tartalmaz, akkor aposztrófok közé kell tenni (nem tévedhetünk, ha mindig aposztrófok közé tesszük a keresett szót).

 `grep` részletes leírása a 107. oldalon

Példák:

```
$ grep ebben gyakorlas 
```

```
Most mindent együtt szerepeltetünk ebben a sorban 'vege' $1 * 42
```

A parancs kilistázza azt a sort a `gyakorlas` nevű file-ból, amelyben a `ebben` szó szerepel. Természetesen a `grep` parancs is megkülönbözteti a keresett karaktersorban a kis és nagybetűket, ezt a különbségtételt azonban a `-i` kapcsolóval kikapcsolhatjuk:

```
$ grep -i ebben gyakorlas 
```

```
Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42
```

```
Ebben a sorban nehany furcsa karakter van % @ !
```

```
Most mindent együtt szerepeltetünk ebben a sorban 'vege' $1 * 42
```

A `grep` parancs alapesetben kiírja a képernyőre az állományok azon sorait, amelyekben a keresett karaktersorozatot megtalálta. Amennyiben a `-n` kapcsolót megadjuk, a sorok előtt azok sorszáma is megjelenik.

Néha arra van szükség, hogy csak a állományok nevét kapjuk meg, amelyekben ez a minta előfordul. Ilyenkor a `-l` (list) kapcsolót használjuk:


```
$ grep -l 'egy' * 
```

```
gyakorlas
```

```
rovidfile
```

3.6. Állományok sorbarendezése a `sort` utasítás

Néha szükségünk van arra, hogy egy állomány tartalmát valamilyen módon sorba rendezzük: erre a `sort` parancs szolgál.

 `sort` részletes leírása a 127. oldalon

A `sort` utasítás minden sor legelső karaktere szerint működik. Ha az első karakterek megegyeznek, akkor a második, harmadik, stb. karaktereket hasonlítja össze. A sorbarakás a szokásos ASCII kódok esetén az írásjelek – számok – nagybetűk – kisbetűk sorrend szerint történik. A `-r` kapcsolóval a rendezési sorrendet teljesen megfordíthatjuk.

Az állomány sorai néha oszlopokba rendezett adatok mezőit tartalmazzák. A mezők között valamilyen határoló vagy elválasztó jelnek kell állnia, ez alapesetben a szóköz vagy a tabulátor. Ha `sort` parancsot a `+Szám` kapcsolóval használjuk, a `Szám` által meghatározott mezőt kihagyva végzi a sorbarendezést. A `-Szám` hatására a rendezés a `Számnál` megadott mezőnél ér véget.

Például rendezzük sorba a korábban létrehozott `gyakorlas` (l. 24. oldalon)

 állományt :

```
$ sort gyakorlas 
```

A sorban a UNIX metakarakterei is itt szerepelnek \$ > ' es a *

Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42


Ebben a sorban nehany furcsa karakter van % @ !

Ez egy proba sor ami csak betuket tartalmaz

Itt mar szamok is vannak a furcsa karakterekkel együtt 1 % @ 42 !

Most mindent együtt szerepeltetunk ebben a sorban 'vege' \$1 * 42

A következő parancs a sorbarendezést az ötödik mezőn (az ötödik szónál)

 kezdi el, és a hetedik mezőnél fejezi be:

```
$ sort +4 -7 gyakorlas 
```

Ez egy proba sor ami csak betuket tartalmaz

Most mindent együtt szerepeltetunk ebben a sorban 'vege' \$1 * 42

Ebben a sorban nehany furcsa karakter van % @ !

A sorban a UNIX metakarakterei is itt szerepelnek \$ > ' es a *

Itt mar szamok is vannak a furcsa karakterekkel együtt 1 % @ 42 !

Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42

3.7. A UNIX helyettesítő karakterei

A UNIX operációs rendszerben lehetőség van arra, hogy a parancsokban szereplő egyes konkrét kifejezéseket (pl. állományneveket), illetve bizonyos stringeket felsorolás helyett egy mintával való azonosság alapján adjuk meg.


Állományokra való hivatkozáskor a kérdőjel (?) az állomány nevében egy

karaktert helyettesíthet, ennek helyén a névben bármilyen karakter (de csak egy) állhat, kivéve a névkezdő `.`-ot. A csillag (*) karakter az állomány nevében minden betűsorozattal (nem csak egy karakterrel) helyettesíthető. Egy adott karakterhalmazt a `[]` jelek között felsorolva a kifejezés a halmaz bármelyik tagjával megegyezik. Egy tartományt kijelölhetünk a két végkarakter közé tett mínusz - jellel.


Szövegben a pont (.) egyezik meg egy bármilyen karakterrel, míg a csillag (*) az őt megelőző karakter tetszőleges számú (akár nulla) ismétlődésével egyezik meg. A `^` a sor elejével, a `$` a sor végével egyezik meg. Szövegekben a egy adott karakterhalmazt az állományneveknél említett módon, a `[]` jelekkel tudunk kijelölni.

A helyettesítő karakterek használatakor a shell először kifejti a karaktereket és a kapott file listát adja tovább a parancsnak. Ha a helyettesítő karaktereket tovább akarjuk adni paraméterként egy parancsnak, akkor azokat macskaköröm (") vagy aposztróf (') jelek közé kell zárni. Az aposztróf minden speciális karaktert megvéd, míg a macskaköröm nem védi a `$`, `\` és a ``` (visszafelé aposztróf) karaktereket⁶. Speciálisan az ún. backslash (`\`) megvédi a kifejtéstől a közvetlenül utána jövő karaktert.


Az elmondottakat a következő példákon érthetjük meg:

A következő parancs kilistázza az összes `a`-val kezdődő két betűs állományt a `/bin` könyvtárban 

```
$ ls /bin/a? 
/bin/ar      /bin/as
```


A következő utasítás kilistázza az összes `a`-val kezdődő file nevet: 

```
$ ls /bin/a* 
/bin/acledit      /bin/aclput      /bin/adfutil      /bin/as
/bin/aclget       /bin/adb         /bin/ar           /bin/awk
```

A következő parancs kilistázza az összes file-t, aminek nevében az `ar` karakterek előfordulnak: 

```
$ ls /bin/*ar* 
/bin/ar /bin/sysdumpstart /bin/xargs
/bin/startsrc /bin/tar
```

⁶Ezek részletes magyarázatát a 83. oldalon, a Korn shell leírásában találhatjuk meg.


-  A `grep` parancsnál is használhatjuk a helyettesítő karaktereket. Az előző `grep` példát folytatva a következő parancs kírja a `ma` + egy karakter kifejezést tartalmazó sorokat:

```
$ grep ma. gyakorlas 
```

Ez egy próba sor ami csak betuket tartalmaz

Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42

Itt mar szamok is vannak a furcsa karakterekkel együtt 1 % @ 42 !

-  Ha az összes, számot tartalmazó sorra kíváncsiak vagyunk, akkor a következő kifejezés a célszerű:

```
$ grep '[0123456789]' gyakorlas 
```

Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42

Itt mar szamok is vannak a furcsa karakterekkel együtt 1 % @ 42 !

Most mindent együtt szerepeltetunk ebben a sorban 'vege' \$1 * 42

-  Az előző utasítás helyett használhatjuk a következő tömörebb kifejezést is:

```
$ grep '[0-9]' gyakorlas 
```

Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42

Itt mar szamok is vannak a furcsa karakterekkel együtt 1 % @ 42 !

Most mindent együtt szerepeltetunk ebben a sorban 'vege' \$1 * 42

-  Az összes soreleji `Ebb` kifejezést a következő utasítás keresi meg:

```
$ grep '^Ebb' gyakorlas 
```

Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42


Ebben a sorban nehany furcsa karakter van % @ !

-  A következőkben speciális karaktert keresünk:

```
$ grep '*' gyakorlas 
```

A sorban a UNIX metakarakterei is itt szerepelnek \$ > ' es a *

Most mindent együtt szerepeltetunk ebben a sorban 'vege' \$1 * 42

-  Használhatjuk a `\-t` is:

```
$ grep \* gyakorlas 
```

A sorban a UNIX metakarakterei is itt szerepelnek \$ > ' es a *

Most mindent együtt szerepeltetunk ebben a sorban 'vege' \$1 * 42

-  Ha a

```
$ grep '$' gyakorlas 
```

parancsot adjuk ki, ez minden sort kiír (minden sornak van vége!):

Ez egy proba sor ami csak betuket tartalmaz

Ebben a sorban mar vannak szamok is , peldaul az 1 es a 42

Ebben a sorban nehany furcsa karakter van % @ !

Itt mar szamok is vannak a furcsa karakterekkel együtt 1 % @ 42 !

A sorban a UNIX metakarakterei is itt szerepelnek \$ > ' es a *

Most mindent egyutt szerepeltetunk ebben a sorban 'vege' \$1 * 42

ezért \$ karaktert csak levédve kereshetünk:

```
$ grep '\$' gyakorlas 
```

A sorban a UNIX metakarakterei is itt szerepelnek \$ > ' es a *

Most mindent egyutt szerepeltetunk ebben a sorban 'vege' \$1 * 42

3.8. Könyvtárak használata

A könyvtárak olyan speciális állományok, amelyek más állományok szervezésére és csoportosítására használatosak. A UNIX operációs rendszer könyvtárai egy hierarchikus fastruktúrát alkotnak. Például egy elképzelt rendszerben a következő oldalon látható struktúra alakulhat ki.

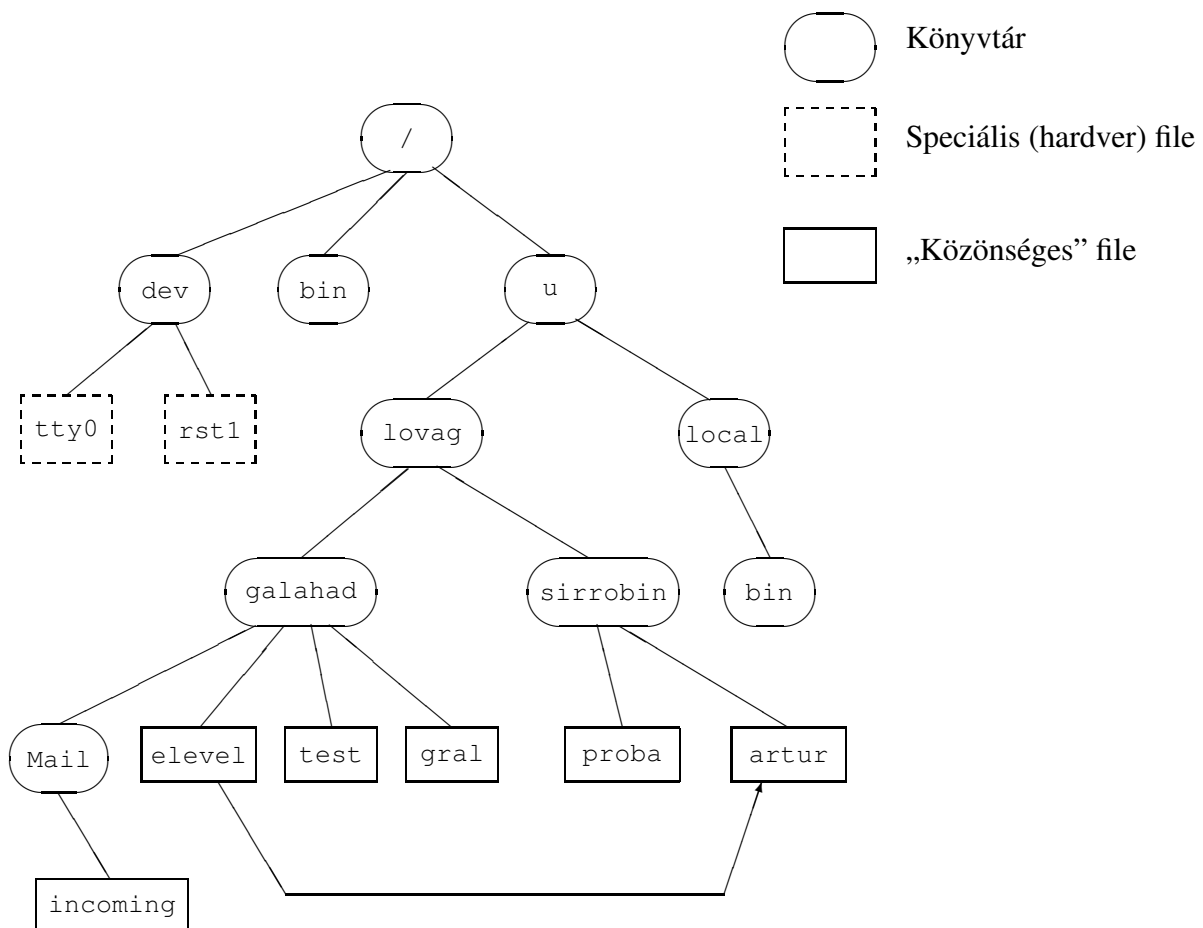
A UNIX könyvtár-rendszerének tetején helyezkedik el a gyökér könyvtár, amit a / jel jelöl. Innen kiindulva minden file elérhető a rendszerben. Az elérési utat *path*-nak (ösvény, útvonal) hívjuk. A könyvtárak más könyvtárakat vagy állományokat tartalmazhatnak. Minden könyvtárnak van *parent* ("szülő") könyvtára, amelyik a struktúrában felette álló könyvtár. Az egyedüli kivétel a gyökér (/) könyvtár.

A felhasználó HOME könyvtára és a hozzá tartozó file-ok és az alkönyvtár-rendszer csak egy kis részét alkotja a UNIX teljes könyvtárrendszerének. A HOME könyvtár az a könyvtár, amibe a felhasználó bejelentkezés után automatikusan bekerül. Az előző példában pl. galahad HOME könyvtára a /u/lovag/galahad, míg a sirrobiné a /u/lovag/sirrobin. A HOME könyvtár alatt a felhasználó létrehozhatja saját könyvtárrendszerét.

Az éppen használt könyvtárat aktuális munkakönyvtárnak nevezzük (*CWD*, *current working directory*). Ezt kiíráthatjuk, ill. a gyökér könyvtárhoz viszonyított útvonalat lekérdezhethetjük a

```
$ pwd 
```

```
/u/lovag/galahad
```



Egy elképzelt UNIX állományrendszer struktúrája.

paranccsal. A parancs információja szerint pl. most a `galahad` felhasználó `HOME` könyvtárában vagyunk⁷ (az egymás alatti alkönyvtárakat a `/` jel választja el egymástól).

A UNIX rendszerben bárhol, ahol állománynevet adhatunk meg, használhatjuk az állományt az elérési útvonallal együtt. Azaz — ha az előző példában megadott könyvtárban vagyunk — a `test` állományt a `/u/lovag/galahad/test` néven is megadhatjuk. A `/u/lovag/galahad/test` nevet bármelyik könyvtárból használhatjuk, ez mindig erre az állományra hivatkozik.

3.8.1. Abszolút és relatív útvonal

Az útvonal megadásánál kétféle leírást használhatunk. Az abszolút útvonal-név megadásakor a gyökér (`/`) könyvtárból indulunk és leírjuk a teljes útvonalat. Ilyen útvonal lehet pl. a `/u/local/bin`.

A relatív útvonal-név az éppen aktuális könyvtárból adja meg az elérési útvonalat. Például ha a `galahad` felhasználó a `HOME` könyvtárában van benn, akkor a `Mail/incoming` adja meg a `Mail` alkönyvtárbeli `incoming` állomány elérését.

Minden könyvtárban használhatjuk a `..` jelölést, ami egy olyan speciális állomány, ami az egy szinttel feljebb levő könyvtárra mutat. Amennyiben a feletünk levő könyvtárból nyíló alkönyvtárra akarunk hivatkozni, akkor a `../` után írjuk be az alkönyvtár nevét. Például, ha a `/u/lovag/galahad/Mail` könyvtárban vagyunk benn, akkor a `../../../../sirrobin/artur` adja meg a `/u/lovag/sirrobin` könyvtárbeli `artur` állomány elérését.

A `..` jelhez hasonlóan lehet használni a `.` jelölést, ami az éppen aktuális munkakönyvtárra hivatkozik.

3.8.2. A könyvtárrendszer használata

A könyvtárak között a `cd` paranccsal lehet mozogni. Ha egy könyvtárból annak egy *Alkönyvtár*ába mozgunk akkor elég a

```
cd Alkönyvtár
```

parancsot kiadni. Ha más könyvtárba mozgunk, ki kell írni a teljes elérési útvonalat. Két szinttel feljebb pl. a

```
$ cd ../ 
```

paranccsal juthatunk.

⁷Természetesen saját gépünkön valamilyen más eredményt kapunk.

3.8.3. Állományok másolása, mozgatása és linkelése

Másolni és mozgatni állományokat a már említett `cp` ill. `mv` paranccsal tudunk.

✎ Az előző példánál maradva a

```
§ cp /u/lovag/sirrobin/artur .
```

✎ parancs az aktuális könyvtárba másolja az `artur` állományt. A

```
§ mv /u/lovag/sirrobin/* .
```

parancs az összes, a `/u/lovag/sirrobin` könyvtárban lévő állományt átmozgatja az aktuális könyvtárba⁸. Ekkor — mivel az eredeti helyről az állományok kitörölődnek — természetesen törlési jogunknak kell lennie a `/u/lovag/sirrobin` könyvtárban (l. a `chmod` parancsot a 42. oldalon).

A UNIX egyik érdekes lehetősége az ún. *linkek* létrehozása: ekkor egy állománynak (ami akár könyvtár is lehet) több neve is lehet. A legegyszerűbb ilyen link, az ún. szimbolikus link egy mutatót jelent csak, amin keresztül más név alatt

✎ is elérhetjük a megadottállományt⁹. Példánknál maradva a

```
§ ln -s /u/lovag/sirrobin/artur elevel
```

parancs a `/u/lovag/sirrobin/artur` állományt `elevel` néven is elérhetővé teszi:

```
§ ls -ltr elevel
```

```
lrwxrwxrwx 1 galahad lovag 13 May 1992 elevel -> /u/lovag/sirrobin/artur
```

A szimbolikus kapcsolat csak egy mutatót jelent, így az állomány mindkét néven írható, olvasható (és esetleg futtatható). Azonban ha az `elevel` állományt **kitöröljük**, akkor az maga az `/u/lovag/sirrobin/artur` állomány nem törölődik, de az adatokat már nem tudjuk elérni `elevel` néven. Az `/u/lovag/sirrobin/artur` állomány törlésekor viszont a csak a link marad meg, az adatok az `/u/lovag/sirrobin/artur` állománnyal együtt elvesznek.

✎ Szimbolikus linket könyvtárak között is létrehozhatunk:

```
§ ln -s /u/local/bin bin
```

A parancs eredményeként pl. a `/u/local/bin/gnuplot` állományt a `bin/gnuplot` név alatt is elérhetjük.¹⁰

⁸Kivéve a `.`-tal kezdődő állományokat.

⁹Lehetőség van ún. *hard* link létrehozására is, de ez néha kényelmetlen.

¹⁰Vigyázzunk a könyvtárak szimbolikus linkjeivel a `csh` típusú shellek esetén, mivel ezek kifejtik a linkek értékét, és az eredményt állítják be az aktuális munkakönyvtárnak. Ez az jelenti, hogy ha egy alkönyvtárba belépünk, majd onnan a `cd ..` paranccsal visszalépünk, egészen máshol találjuk magunkat. (Ez egy ritkán várt „feature” ...)

3.8.4. Könyvtárak létrehozása és törlése

Könyvtárakat a `mkdir` paranccsal hozhatunk létre. Argumentumként akár több könyvtárnevet is megadhatunk. Például a:

```
$ mkdir test test/artur /u/lovag/galahad/tmp 
```

létrehozza az aktuális könyvtárból nyíló `test`, az abból nyíló `artur`, valamint a bejelentkezési könyvtár (itt a `/u/lovag/galahad`) alatti `tmp` könyvtárakat.

Természetesen nem hozhatunk létre könyvtárat egy "normális" file alatt. Ugyanígy sikertelen a parancs, ha nincs engedélyünk alkönyvtár létrehozására (beleírására) az adott könyvtárban.

Ha egy idő után nem lesz többé szükség egy *Könyvtárra*, és célszerű azt kitörölni, akkor erre a UNIX operációs rendszerben az

```
rmdir Könyvtár
```

parancsot használhatjuk. Az `rmdir` parancs a `mkdir` parancshoz hasonlóan több argumentumot is elfogad.

Amennyiben olyan könyvtárat akarunk kitörölni, amelyik nem üres, a UNIX operációs rendszer hibajelzést ad. Ilyenkor — kellő óvatossággal — használhatjuk a 32. oldalon említett `rm -r` parancsot.

Néha a kitörölni kívánt könyvtárban ún. rejtett állományok helyezkedhetnek el, amelyeket a szokásos `ls` parancs nem listáz ki, csak a `-a` kapcsoló használatával. A rejtett állományok ponttal kezdődnek, és általában bizonyos programok (`elm`, `vi`, `tin`, `xv` ...) használják őket. Ha a könyvtárban — amelyet ki akarunk törölni — rejtett állományok vannak, és egészen biztosak vagyunk benne, hogy nem kellenek, egyenként kitörölhetjük őket. **VIGYÁZAT!!** SOHA ne adjunk ki törlési parancsot a `.*` állományokra vonatkozólag, mivel egy ilyen törlés automatikusan elindul a `..` különleges jelentése miatt felfelé is a könyvtárstruktúrában! Megoldás lehet ilyenkor pl. legalább 3 karakter megadása a törlésnél (`.*??*` alakot), esetleg karaktertartomány kijelölése.

3.8.5. Állomány megkeresése a könyvtárstruktúrában

Állományokat a `find` parancs segítségével kereshetünk meg a könyvtárrendszerben. Például a

```
$ find . -name test.c -print
```

parancs az aktuális könyvtártól lefelé megkeresi az összes `test.c` állományt, és a nevüket kiírja a képernyőre. Ha nem tudjuk a teljes nevet, csak azt hogy `c`-re végződik, akkor a

```
$ find . -name '*c' -print
```

parancsot használva a `find` az aktuális könyvtártól lefelé megkeresi az összes `c` végződésű állományt, és a nevüket kiírja a képernyőre.

A parancs általánosabb alakja:

```
find KiindulásiKönyvtárak -name Állomány -Parancs
```

a *KiindulásiKönyvtárak*ból elindulva a könyvtárstruktúrában lefelé keresi a megadott nevű *Állományt* és végrehajtja rá a megadott *Parancsot*. A `find` sokoldalú parancs, részletes leírását pl. a `man find` utasítással kérhetjük elő.

3.9. Állományok védelmének beállítása

Az állományok védelmi kódját a `chmod` paranccsal állíthatjuk. A védelmi kód 3 darab *oktális* (8-as számrendszerbeli) szám, melyek szimbolikusan is jelölhetünk. Az `ls -l Állomány` parancs az első oszlop a 2-10. karakterpozíciójában kijelzi a védelmi kódokat: a kódot külön állítjuk az állomány tulajdonosára (`u` – user, 2-5 pozíció), a felhasználó csoportjára (`g` – group, 6-8 pozíció) és mindenki másra (`o` – others, 8-10. pozíció).

A védelmi kódok szabályozzák az állomány futtatását (`x` – eXecute = 1), írását (`w` – write = 2), és olvasását (`r` – read = 4). Akinek írási joga van, az törölheti is az adott állomány tartalmát (de magát az állományt nem). A könyvtárra kiadott futtatási jog engedélyezi az adott könyvtárba való belépést.

📎 Például a korábban létrehozott `gyakorlas` állomány védelme általában 644-es:

```
$ ls -l gyakorlas 
```

```
total 20
```

```
-rw-w--w-- 1 galahad lovag 3910 Oct 26 1992 gyakorlas
```

Az adott betű jelenléte a megadott hozzáférést engedélyezi, a `-` jel pedig a tiltást jelenti.

📎 A

```
$ chmod 755 gyakorlas 
```

parancs megváltoztatja a `gyakorlas` állomány védelmi kódját a következő módon: az első szám a tulajdonosra (`galahad`) vonatkozik, a következő a csoportra (`lovag`), és az utolsó a többiekre. A `7=4+2+1` alapján a tulajdonosnak megvan a futtatási (1), az írási (2) és az olvasási (4) joga is, másoknak csak a futtatási és az

📎 olvasási joga. Ha a parancsot a

```
$ chmod 55 gyakorlas 
```

alakban adjuk ki, akkor csak a csoport és a többiekre vonatkozó beállítás változik, a tulajdonosra vonatkozó védelmi kód nem változik.

Az előző parancs kiadható szimbolikusan is:



```
$ chmod u+rwx * 
```

```
$ chmod go+rx-w * 
```

Ekkor a tulajdonosnak megadtuk az összes jogot, a többiektől pedig elvettük az írási jogot. A második parancs helyett a

```
$ chmod go=rx * 
```

is használhatjuk.

4. fejezet

Állományok szerkesztése

4.1. A `vi` editor használata

A különböző szerkesztőket (editorokat) állományok létrehozására és módosítására használjuk. A UNIX operációs rendszerekben a `vi` editor az általánosan elterjedt, egész képernyős szerkesztő program, mely része az operációs rendszernek.

Egy file szerkesztéséhez egyszerűen adjunk meg a `vi` parancsnak az állomány nevét:

```
$ vi Állomány 
```

Ha az adott állomány nem létezik, akkor a `vi` létrehozza azt¹.

A szerkesztőbe ún. parancs üzemmódban lépünk be, de bármikor máskor is ide kerülünk vissza az `Esc` (vagy `Escape`) gomb – esetleg a `Ctrl-` kombináció – megnyomásával. Ha a gomb megnyomásakor esetleg egy sípolást hallunk, akkor már eleve parancs üzemmódban voltunk (ez nem hiba!).

A parancs üzemmódban különböző utasításokat adhatunk ki az egyes gombok lenyomásával. A `:` parancs után további parancsokat adhatunk ki. Pl. a `vi`-ből kilépni a parancs üzemmódból a `:x` paranccsal tudunk, a változtatások elmentésével. Ha mégsem akarjuk elmenteni a változtatásokat, akkor a `:q!` parancsot használjuk a kilépéshez!


A parancs üzemmódból az `a` és az `i` paranccsal léphetünk át a szövegírás üzemmódba: az `a` az aktuális karakter után, az `i` pedig az elé kezdi beilleszteni a szöveget. Az `o` (kis o betű) parancs az aktuális sor után, az `O` (nagy O betű) pedig az előző sorba kezdi szöveget beilleszteni. A parancs üzemmódba az `Esc` gomb megnyomásával kerülhetünk vissza.

¹Természetesen csak ha van jogunk a megadott helyre írni.

A parancs üzemmódban az `x` gombbal törölhetjük ki a kurzor helyén lévő betűt. Szavak és sorok törléséhez a `d` parancsot kell használnunk: egy szó kitörölését a `dw` paranccsal végezhetjük el, míg az aktuális sort a `dd` parancs törli ki.

A kurzort a parancs üzemmódban a `h-j-k-l` gombokkal mozgathatjuk balra-le-fel-jobbra, de általában a billentyűzeten található nyilakat is használhatjuk ehhez. Ha gyorsabban akarunk mozogni az állományban, akkor használhatjuk a képernyőoldalyival mozgó parancsokat is: a `Ctrl-F` egy teljes képernyővel előre visz, a `Ctrl-B` pedig egy képernyővel visszafelé lapoz.

A `vi` nagyon sok utasítással rendelkezik, az ez iránt érdeklődőknek ajánljuk a 173. oldalon található részletes leírást.

 `vi`
részletes
leírása a 173.
oldalon

4.2. A MicroEMACS szerkesztő

A MicroEMACS szövegszerkesztő ugyan nem része az eredeti UNIX operációs rendszereknek, de a UNIX világ egy széles körben² elterjedt többablakos editora. A szövegszerkesztő előnye, hogy a UNIX mellett VAX/VMS és IBM PC (DOS) rendszereken is futtatható. Mivel mindegyik operációs rendszerben ugyanazokat az utasításokat használhatjuk, ezért egy kezdő számára hasznos lehet ezt a szerkesztőt először megtanulni. A programnak létezik egy sokkal nagyobb, több mindenre képes változata, az ún. GNU `emacs` szövegszerkesztő, ami szintén használatos UNIX rendszereken. Ugyan a két program sok parancsa megegyezik, mégis célszerű szétválasztani a kettőt.³ A továbbiakban csak a MicroEMACS programmal foglalkozunk.

A MicroEMACS szerkesztő a
`§ uemacs` Állománynév `Enter`
parancs kiadásával hívjuk meg. Néhány gépen a `uemacs` parancs helyett az `emacs` parancsot kell használnunk.

A szerkesztőbe való belépés után beírhatjuk a kívánt szöveget, törölni a szokásos módon lehet a `Backspace` gombbal (ezt néha `Delete` vagy `←` felirat jelzi).

A szerkesztés végén az `Esc-Z` (lenyomjuk az `Esc` gombot, elengedjük, majd lenyomjuk a `Z` gombot) utasítás elmenti a változtatásokat és kilép a szerkesztőből. Ha nem akarjuk a változtatásokat megőrizni (pl. ha rossz file-t nyitottunk meg), akkor a `Ctrl-X Ctrl-C` utasítást használjuk (folyamatosan lenyomva tartva


²A program forráskódja C nyelven szabad programként (ún. *public domain*) mindenki rendelkezésre áll ingyen.

³A GNU `emacs` inkább egy programozási stílust képvisel, mint egy szövegszerkesztőt...

a `Ctrl` gombot először lenyomjuk az `X` gombot, elengedjük, majd lenyomjuk a `C` gombot).

A szövegben való mozgásra a billentyűzeten található nyilakat használhatjuk⁴. Amennyiben a nyilak nem működnek, használhatjuk a `Ctrl-Betű` kombinációkat (lenyomva tartva a `Ctrl` gombot lenyomjuk a `Betű` gombot): a `Ctrl-F` előre ugrik egy betűvel, a `Ctrl-B` visszalép egy betűnyit, míg a `Ctrl-N` a következő sorra ugrik, a `Ctrl-P` pedig az előző sorra lép. Ha gyorsabban akarunk mozogni, akkor használhatjuk a képernyőoldallyal mozgató parancsokat is: a `Ctrl-Z` egy teljes képernyővel előre visz, a `Ctrl-V` pedig egy képernyővel visszafelé lapoz.

A `uemacs` sok más utasítással és lehetőséggel rendelkezik (pl. több file egyidejű szerkesztése, ablakok, makrók, stb.), az ez iránt érdeklődőknek ajánljuk a 185. oldalon található részletes leírást.

 `emacs`
részletes
leírása a 185.
oldalon

⁴Lehet, hogy a billentyűzet nyilai nem működnek rendesen (ez általában a 84 gombos IBM PC billentyűzetenél fordulhat elő): ekkor a számbillentyűzet (*numeric keypad*) 2-4-6-8 gombjaival próbálhatjuk mozgatni a kurzort.

5. fejezet

Állományok nyomtatása

Előbb-utóbb ki szeretnénk nyomtatni állományainkat. A multitasking rendszerben a UNIX több különböző helyi, ill. távoli géphez kötött nyomtatót is tud egyszerre kezelni. A felhasználó így maga döntheti el, melyik nyomtatót óhajtja használni.

Az azonos nyomtatók közül — beállítástól függően — a rendszer automatikusan is kiválaszthatja az éppen szabad printert. Amennyiben az összes elérhető nyomtató foglalt, a UNIX operációs rendszer várakozó listába (ún. *queue*) rendezi a nyomtatási kérést, és mihelyt egy nyomtató felszabadul, a beérkezési sorrendben, vagy méret szerinti kiosztásban elkezd a következő várakozó állomány nyomtatását. Egyszerre több példányban is kinyomtathatunk egy állományt, valamint lehetőség van a már elküldött nyomtatási kérések törlésére, illetve azok állapotának lekérdezésére.

A nyomtatókra egyszerűen a várakozó lista (queue) megadásával hivatkozunk. Ha a rendszerre egyszerre több printer is rá van csatlakoztatva, ezek közül az egyik az ún. rendszernyomtató. Ez azt jelenti, hogy ha másként nem rendelkezünk, akkor a nyomtatás erre a printerre megy.

Egy adott rendszerből elérhető helyi és távoli nyomtatók használatáról (pl. a queue neve, a nyomtató típusa és helye, stb.) a rendszergazda adhat pontos információt.

Nyomtatni általában¹ az `lpr` paranccsal tudunk: a

`$ lpr gyakorlas`

parancs a rendszernyomtatón nyomtatja ki a `gyakorlas` állományt².

¹Néha a nyomtató queue nem konvertál automatikusan, ilyenkor pl. PostScript printer esetén az ASCII szövegünket lehet, hogy át kell alakítani.

²Ha nincs ilyen parancs (pl. bizonyos System V rendszerekben), akkor az `lp` parancsot használhatjuk az `lpr` helyett.

✎ A következő utasítás a `gyakorlas` állományt a `hpc214` nevű³ printeren nyomtatja ki (a `-P` kapcsoló után szóköz kihagyása nélkül kell beírni a printer nevét)⁴:

```
$ lpr -Phpc214 gyakorlas 
```

✎ A `-#` kapcsoló után megadhatjuk a kívánt példányszámot. Például ha 2-2 példányt szeretnénk `gyakorlas` és a `masolat` állományokból, akkor az előző utasításhoz hasonlóan:

```
$ lpr -#2 -Phpc214 gyakorlas masolat 
```

5.1. A printer állapotának lekérdezése

A nyomtatási várakozó-lista(printer queue) állapotát az `lpq` utasítás listázza ki. Ugyanez a parancs mutatja meg, hogy a nyomtató milyen állapotban van (például ✎ kikapcsolták, vagy kifogyott a papír, stb.). Például a `hpc214` nyomtató állapotát a

```
$ lpq -Phpc214 
```

parancs⁵ kérdezi le. A válasz például a következő lehet:

Queue	Dev	Status	Job	Files	User	PP	%	Blks	CP	Rnk
hpc214	lp0	running	42	gyakorlas	galahad	10	83	12	1	1

A parancs kilistázza a nyomtató queue nevét, a hozzá tartozó eszközt (`Dev`), a queue állapotát⁶ (`Status`), valamint az éppen a queue-ban lévő nyomtatási kérés(ek) sorszámát (`Job`), a hozzá tartozó állomány(oka)t és felhasználó(ka)t. Az éppen nyomtatás alatt álló file esetében a rákövetkező oszlopok a már kinyomtatott részt százalékban adják meg. A válaszból leolvasható még a file hossza és a kért példányok száma is.

A példában a `galahad` felhasználó `gyakorlas` 12 blokkhosszúságú állományának 83%-át kinyomtatta már a `hpc214` nyomtatón és csak 1 példány készül belőle.

³Általában a printer neve utal a típusra és az elhelyezkedésre: a `hpc214` egy, a C épület második emelet 14-es szobájában lévő HP nyomtatóra utalhat.


⁴Az `lp` parancsnál az `lp -d` kapcsolót kell használni.

⁵Ha nincs ilyen parancs, akkor esetleg az `lpstat` parancsot használhatjuk.

⁶A `running` azt jelenti, hogy a file-t éppen nyomtatja a nyomtató, a `queued` pedig azt, hogy a file arra vár, hogy rá kerüljön a sor.


5.2. A nyomtatási kérés megszakítása

A nyomtatási kérés megszakításához tudnunk kell a kérés sorszámát (ezt az előbbiekben tárgyalt `lpq` paranccsal is lekérdezhethetjük).

Az előzőekben említett példában a 

```
$ lprm -Phpc214 42 
```

parancs⁷ kitörli a 42-es számú nyomtatási kérést a `hpc214` queue-ból (már futó nyomtatás esetén azt megszakítja és úgy törli ki).

Lehetőség van az egy felhasználóhoz tartozó minden kérés kitörlésére: az 

```
$ lprm -Phpc214 galahad 
```

parancs minden, a `galahad` felhasználóhoz tartozó nyomtatási kérést kitöröl a `hpc214` queue-ból.

⁷Ha hiányzik ez a parancs, akkor esetleg a `cancel` parancsot használhatjuk az `lprm` helyett.

6. fejezet

Ki- és bemenetek átirányítása, különleges shell parancsok

A UNIX operációs rendszerben a felhasználó belépésekor egy ún. *shell*(burok, héj) program indul el. A shell program fogadja a parancsokat és — részben átalakítva — továbbadja azokat az egyes speciális programoknak. A shellen belül megteremtették a programok összekapcsolásának lehetőségét is, mivel általában minden program innen indul el.

Az átirányítás lehetővé teszi, hogy egy program végeredményét egy másik program használja, külön állomány létrehozása nélkül. Egy parancssorban több átirányítást is használhatunk, pl. az első utasítás a bemenetét egy megadott állományból veszi, majd a második parancs az első kimenetét (eredményét) dolgozza fel, végül a harmadik a második kimenetét továbbalakítva a végeredményt egy kimeneti állományba írja. Mindez nagy közbülső állományok létrehozása nélkül történik, ezért sokkal gyorsabb a parancsok egyenkénti futtatásánál.

6.1. Parancs kimenetének átirányítása

Egy program végeredményét egy állományba is beleírhatjuk a képernyő helyett. Ehhez a parancs után egy `>` jelet, majd a kívánt file nevét kell írunk. Például:

```
$ ls -l > filelista 
$ cat filelista 
total 6
-rw-w--w--  1 galahad  lovag      3910 Oct 26 1992  choice
-rw-w--w--  1 galahad  lovag       21 Oct 26 1992   cm
```


```

-rw-w--w--  1 galahad  lovag      0 Dec 05 1992  filelista
-rw-w--w--  1 galahad  lovag     198 Oct 26 1992  gral
-rw-w--w--  1 galahad  lovag      5 Dec 05 1992  log
-rwxr-xr-x  1 galahad  lovag     31 Oct 26 1992  test
drwx--x--x  2 galahad  lovag     512 Jul 06 1992  Mail

```

Az első parancs az `ls -l` utasítás eredményét a `filelista` állományba írja bele, majd a `cat` paranccsal a `filelista` file tartalmát írjuk ki a képernyőre.

Az átirányítás **felülírja** a már létező állományt. Ha ehelyett az állomány végéhez akarjuk az eredményt fűzni, akkor a `>{ }>` jelet kell használnunk. A UNIX ekkor ellenőrzi, létezi-e a megadott nevű file. Ha igen, akkor a végéhez fűzi az új adatokat, ha pedig nem létezik a file, akkor létrehozza azt. Az előző

 példát folytatva :


```

$ ls > filelista 
$ ls -l >> filelista 
$ cat filelista 
choice  cm      filelista      gral  kocsi  log  test  Mail
total 6
-rw-w--w--  1 galahad  lovag      3910 Oct 26 1992  choice
-rw-w--w--  1 galahad  lovag       21 Oct 26 1992  cm
-rw-w--w--  1 galahad  lovag       68 Dec 05 1992  filelista
-rw-w--w--  1 galahad  lovag     198 Oct 26 1992  gral
-rw-w--w--  1 galahad  lovag      5 Dec 05 1992  log
-rwxr-xr-x  1 galahad  lovag     31 Oct 26 1992  test
drwx--x--x  2 galahad  lovag     512 Jul 06 1992  Mail

```

Az első `ls` felülírja a `filelistat`, míg az `ls -l` a végéhez fűzi a részletes állománylistát.

6.2. Bemenet átirányítása

Nem csak a képernyőre való kiírást (az ún. *standard output*), hanem a billentyűzeten történő adatbevitelt (*standard input*) is átirányíthatjuk. Például a `grep`  parancs állomány megadása nélkül a bemenetről olvassa a sorokat, és keresi bennük a megadott szót:


```

$ grep 'ben' 
Itt nincs a keresett karakter 
Ebben mar van 

```

`Ctrl`-`D`

Ebben mar van

Irányítsuk át a bemenetet a `gyakorlas` állományból! Ehhez a `<` jelet kell  használnunk:

```
$ grep 'ben' <gyakorlas Enter
```

Ebben a sorban mar vannak számok is , például az 1 es a 42

Ebben a sorban néhány furcsa karakter van % @ !

Most mindent együtt szerepeltetünk ebben a sorban 'vege' \$1 * 42

✎ Másik példa a `wc -l` parancs, amelyik megszámolja a bemeneten érkező sorokat:

```
$ wc -l < gyakorlas 
6
```

6.3. A cső (*pipe*) használata

A UNIX-ban lehetőség van arra, hogy közbülső file létrehozása nélkül kapcsoljunk össze két (vagy több) programot. Ilyenkor az első program kimenete a második bemenete lesz.

A két összekötni kívánt programot az ún. *pipe* jellel (ez a `|`) kell a parancssorban elválasztani. Egyszerre több parancsot is összekapcsolhatunk a `|` jellel, ezt *pipeline*-nak (csővezeték) nevezzük.

Példák:

✎ Kिलistázzuk a `/bin` könyvtár tartalmát az `ls` parancs segítségével és az eredményt a `more` segítségével megjelenítjük a képernyőn:

```
$ ls -l /bin | more 
```

✎ A `who` parancs kiírja az aktuális felhasználókat (soronként egyet), a `wc -l` parancs pedig megszámolja a sorok számát. Így az éppen aktív felhasználók számát a

```
$ who | wc -l 
adja meg.
```


6.4. Parancs eredményének továbbhasználata

Bizonyos esetekben szükségünk lehet egy parancs eredményére, de nem bemenetként, hanem egy utasítás argumentumaként. Ilyenkor a fordított aposztróf karakterek közé kell zárni a forrásként szereplő utasítást: a fordított aposztrófok közé írt parancs végeredménye közvetlenül behelyettesítődik a parancssorba. Például a

✎ `more \grep -l 'egy' * \` parancs minden, az `egy` szót tartalmazó állományt megjelenít a képernyőn a `more` segítségével.

6.5. Parancsállományok létrehozása


Néha egymás után többször ugyanazt a parancssorozatot kell lefuttatnunk. Ilyenkor célszerű lehet ezt egy állományban, új parancsként összefoglalni.


Például hozzuk létre egy szövegszerkesztővel a `ddir` állományt, a következő  tartalommal:

```
echo "Most kovetkezik az allomanylista"  
ls -l  
echo "Vege van az allomanylistanak"
```

Ezután változtassuk meg az állomány védelmi kódját futtathatóra:

```
$ chmod +x ddir 
```

majd a `ddir` parancs egymás után lefuttatja a három parancsot. A parancsállományokról és a bennük használható parancsokról a 83. oldalon, a Korn shell  részletes leírása a 83. oldalon

 ksh
részletes
leírása a 83.
oldalon

7. fejezet

Programok futtatása

A UNIX operációs rendszerben egyszerre több parancsunk is futhat párhuzamosan. Ez, ha csak egy terminált használunk, úgy valósul meg, hogy egy parancsunk az előtérben fut (teljesen lefoglalva a terminált, billentyűzetet), miközben a többiek a háttérben futnak. A háttérben futó eljárások is írhatnak a képernyőnkre, de nem kaphatnak bemenő adatokat a billentyűzetről.

Egy végrehajtható programot (file-t) a nevével futtatunk. A shell a `PATH` környezeti változóban megadott könyvtárakat sorra végignézi, amíg meg nem találja a megadott nevű programot. Ha ez nem tartalmazza az aktuális könyvtárat, akkor a teljes elérési útvonallal kell megadni az állományt (pl. `./a.out`, `/usr/ucb/mail`).

Egy már futó programot a `Ctrl-Z` gombok egyidejű lenyomásával állíthatunk meg. Az így leállított program még benne van a gépben, csak nem fut, bármikor folytatható a futtatása. Ha egy programból ki akarunk lépni úgy, hogy az befejezze a futást, a `Ctrl-C` lenyomásával tehetjük ezt meg.

A háttérbe tett (futó vagy megállított) programokat a `jobs` paranccsal nézhetjük meg. A parancs felsorolja a programok hivatkozási számát, állapotát (Running, Stopped). Az aktuális háttérprogram mellett a `+` jel látható.

A programokat az `fg` (foreground) paranccsal előtérbe, a `bg` (background) paranccsal háttérbe tehetjük. A parancsoknak meg kell adni, hogy melyik programot akarjuk elő- vagy háttérbe tenni. Ha nem adunk meg azonosítót, akkor a parancs az aktuális programra hat. Az azonosító a `jobs` parancsnál kiírt kis egész szám. A 2-es számú program háttérben való továbbfuttatása pl. a

```
$ bg %2 Enter
paranccsal történik.
```

Egy parancsot futtathatunk azonnal is a háttérben. Ehhez a parancs legvégére

at
leírása a 156.
oldalon

batch
leírása a 158.
oldalon

tegyünk egy `&` jelet. A háttérben futó parancsok kisebb prioritással bírnak, s így terhelt rendszeren lassabban futnak le. Ha sokan futtatnak a rendszeren, célszerű más módon elküldeni a futtatásokat (lásd az `at` és `batch` parancsokat a 156. és a 158. oldalon). A háttérben indított parancsok befejezik futásukat, ha kijelentkezünk a gépről. Ha ezt el akarjuk kerülni, akkor a háttérbe küldendő parancs elé be kell írni a `nohup` parancsot:

```
$ nohup Program <BemenetiFile >KimenetiFile & 
```

Például a következő parancs a háttérben futva a `lista` állományba beleírja az összes, az adott könyvtárban és az alkönyvtárakban található `hello.c` állomány nevét:

```
$ nohup find . -name hello.c -print >lista & 
```

Minden futó eljárásnak a rendszerben van egy azonosító száma, az ún. PID (*Process IDentification number*). A PID megadja az elindított program sorszámát a gép elindítása óta, s így egyedi az adott gépen futó programokra (pl. a futó `shell`-ünknek is van ilyen száma). A bejelentkezési shellhez tartozó programokat a `ps` paranccsal kérdezhetjük le. Ha más terminálon is dolgozunk, akkor a

```
$ ps -u FelhasználóiNév 
```

paranccsal kérdezhetjük meg az adott *FelhasználóiNév*hez tartozó összes futó eljárást¹.

A felesleges programjainkat le is állíthatjuk a `kill` parancs segítségével. A

```
$ kill -9 PID-szám 
```

parancs leállítja a *PID-szám* azonosítójú eljárást. A `-9` kapcsoló a felhasználó által elindított alprogramok leállításához szükséges ún. *trap* jel értéke². Csak a saját programjainkat tudjuk leállítani. Háttéreljárásainkat leállíthatjuk más módon is, ha a *PID-szám* helyére a `jobs` parancsnál kapott kis egész azonosítószámot írjuk a `%` jel után. Például a

```
$ kill -9 %2 
```

leállítja a 2. számú háttéreljárást.

A UNIX rendszer minden futó eljáráshoz egy ún. prioritási értéket rendel: minél kisebb ez a szám, annál nagyobb százalékát kapja meg a CPU időnek az adott program. Amennyiben nagy és hosszú háttérben futó programot akarunk elindítani, ajánlatos ezt nagyobb prioritási értékkel („lassabb futással”) elindítani.³ Ha a gép üres, akkor így is megkapja a program a teljes CPU időt, míg

¹Ezt azt utasítást BSD alapú gépeken `ps -u` formában kell helyesen kiadni.

²A `-9` helyett a `-1` — `-8` kapcsolókat is használhatjuk, „kíméletesebb” leállításhoz. A `-9`-et csak végső esetben használjuk.

³Néhány gépen a rendszer automatikusan nagyon nagyra állítja a prioritási értéket, ha a prog-

pl. egy interaktív bejelentkezéskor a program futása lelassul, és lehetővé teszi az interaktív (gyorsabb) használatot.

A csökkentett prioritás beállítására a `nice` parancs szolgál⁴:

```
$ nice [-Szám] Parancs 
```

A Szám 1 és 19 közti érték, és a növekvő értékek csökkenő prioritásra utalnak. Az alapérték a 10.

A `nice` magától nem futtat a háttérben, ezért általában a háttérben a következő típusú paranccsal futtathatunk:

```
$ nohup nice Program <BemenetiFile >KimenetiFile & 
```

A prioritást egy már futó programnál is átállíthatjuk a `renice` parancs segítségével:

```
$ renice Új-prioritás PID-szám 
```

ahol a *PID-szám* az átállítandó program azonosítóját jelenti.


ram a háttérben magas prioritással sokáig fut.

⁴A C shellben van a `nice` parancson kívül egy beépített `nice` is, mely speciális esetekben másképpen viselkedik.

8. fejezet

Kommunikációs programok

A UNIX operációs rendszer szerves része a hálózati kapcsolat, melynek segítségével lehetőség nyílik különböző elektronikus információcserére. A legismertebb ilyen programok közül a `write` és a `talk` közvetlen üzenetküldő utasításokkal, a `mail` és az `elm` elektronikus levelezőprogramokkal, a `tin` elektronikus újságolvasó programmal, valamint a TCP/IP hálózatot használó `telnet`, `ftp`, `finger` utasítással és a `gopher` elektronikus adatlekérdező programmal foglalkozunk röviden ebben a részben. A könyv második részében ezen utasítások bővebb leírása mellett más hálózati programok használatát ill. leírását is megtalálhatjuk (`rlogin`, `rsh`, `nslookup`, `archie` és az u.n. *anonymous* `ftp` lehetőség).

 bővebb TCP/IP ismeretek a 215. oldalon

8.1. Párbeszédés kapcsolat

8.1.1. A `write` utasítás

A `write` utasítás segítségével rövid, egysoros üzeneteket küldhetünk a rendszerben éppen dolgozó felhasználóknak. Az üzenet ugyan csak rövid lehet, de használata kényelmes, mivel rögtön megjelenik a címzett képernyőjén, és elolvasásához nem szükséges kilépni az éppen futó programból.

A paranccsal csak a rendszert interaktívan használó felhasználóknak üzenhetünk. Például:

Először nézzük meg, be van-e jelentkezve akit keresünk (tegyük fel, hogy felhasználói nevünk `vilma`):

```
$ who   
fredi tty36 Mar 21 20:05
```



```
vilma tty4 Mar 21 19:15
irma pty/2 Mar 21 18:19
```

Ezután üzenhetünk pl. fredinek:

```
$ write fred 
Szia fred, mi ujsag 
```

A parancs kiadása után minden sor végén -t gépelve a sor tartalma megjelenik fred képernyőjén. Ha be akarjuk fejezni az írást, akkor az állományvége karaktert (Ctrl-D) kell beírnunk.

Fredi képernyőjén az éppen aktuális kurzorpozícióban a következő üzenet jelenik meg ekkor:

```
Message from vilma on tuzko (tty4) [Mar 21 21:12:05 1993]
Szia fred, mi ujsag
```

Ha fred válaszolni akar, akkor hasonlóképpen a

```
$ write vilma 
utasítással válaszolhat.
```

A write hívásokat a mesg n utasítás letiltja.

8.1.2. A talk parancs

A write utasítás csak rövid beszélgetések esetén kényelmes, mivel a képernyőn saját üzeneteink és a válaszok összekeverednek. Hosszabb beszélgetéseket éppen ezért célszerűbb a talk utasítással kezdeményeznünk. Például ha felhasználói nevünk vilma a dino gépen, akkor fredivel a tuzko gépen a

```
$ talk fred@@tuzko 
```

paranccsal próbálhatunk kapcsolatot teremteni. Az utasítás hatására fred felhasználó képernyőjén egy figyelmeztető üzenet jelenik meg, hogy vilma@dino szeretne beszélni vele:

```
Message from TalkDaemon@tuzko...
talk: connection requested by vilma@dino.
talk: respond with: talk vilma@dino
```

Ha erre fred a

```
$ talk vilma@@dino 
```

paranccsal válaszol, akkor mindkettőjük képernyője két részre oszlik: a felső részen írhatják be üzeneteiket, míg az alsó részen a partner gépelése jelenik meg folyamatosan. A beszélgetést a Ctrl-C paranccsal lehet befejezni.

Amennyiben nem adunk meg gépnevet, az automatikusan a helyi gépre vonatkozik.

A `talk` hívásokat a `msg n` utasítás letiltja.

8.1.3. A `msg` utasítás

Néha zavartalanul szeretnénk dolgozni, minden, közvetlenül a képernyőnkön megjelenő üzenet nélkül. Ilyenkor a

```
$ msg n 
```

utasítással megakadályozhatjuk, hogy zavarjanak bennünket a `write` és a (már tárgyalat) `talk` utasításokkal.¹ A hívásokat a

```
$ msg y 
```

parancs után újra fogadni fogjuk.

8.2. Elektronikus levelezés

8.2.1. A mail levelező program

A `mail` utasítás segítségével elektronikus leveleket küldhetünk más felhasználóknak. A levél elküldése után a címzett vagy azonnal értesítést kap erről (ha éppen be van jelentkezve a rendszerbe), vagy a következő bejelentkezéskor a UNIX rendszer figyelmeztetni fogja az új levélre.

A UNIX `mail` programja és az `elm`² csak két különböző, az operációs rendszer alap levelezőszolgáltatását használó program, így ugyanonnan veszik az új leveleket. Ezáltal elképzelhető, hogy valaki felváltva használja őket.

A UNIX `sendmail` levéltovábbító programja — történeti okok miatt — csak 7 bitet visz át. Magyar ékezetes szövegek elküldéséhez ezért a levelet kódolnunk kell, amire pl. a `uuencode` parancsot használhatjuk, vagy szöveget a `TeX` (esetleg az `etex`) formátumba is átalakíthatjuk.

A UNIX rendszerek különböző `mail` programokat használnak. A következőkben egy széles körben elterjedt programot, az ún. UCB `mail` programot ismertetjük. Ez a program fut alapértelmezésben a `mail` meghívásakor pl. az IBM AIX és a DEC Ultrix rendszerekben. A SUNOS/Solaris rendszerben a `mail` alapesetben

¹Természetesen ilyenkor is elérhetőek vagyunk, pl. elektronikus levéllel bárki üzenhet.


²Az `elm` nem része az eredeti UNIX rendszernek, hanem u.n. közprogram.



`uuencode`
leírása a 141.
oldalon.

ugyan a `/bin/mail` nagyon egyszerű programra vonatkozik, de a `/usr/ucb/mail` parancs itt is az UCB mail programot futtatja³.


Az UCB mail helyett — amennyiben lehetőség van rá — célszerűbb a menüvel ellátott `elm` levelező programot használni, melynek leírását a 68. oldalon találhatjuk meg.

 `elm`
leírása a 68.
oldalon


Levél küldése

Egy egyszerű, rövid üzenetet a `mail` program segítségével küldhetünk el. Ha levelünk hosszú, célszerű lehet először a `pl. vi` szerkesztő segítségével megírni, és utána elküldeni.

A `mail` utasítás sorában meg kell adni a címzett(ek) felhasználói nevét. Ezután a program először a rövid tartalmat kérdezi meg (`Subject:`), majd soronként fogadja az üzenetet. Az üzenetet egy, csak egy `.`-ot tartalmazó sorral zárjuk le (használhatjuk a `Ctrl-D` kombinációt is). A legvégén - beállítástól függően - a `mail` program felajánlja "másolatok" küldésének lehetőségét (`Cc:`) is más felhasználóknak. Például:

 `$ mail galahad`
`Subject: delutani talalkozo`
`Delutan 3 orakor megbeszeles lesz`
`a kerekasztalnal`
`Sir Robin`
`.`
`Cc:`

Ha `file` formájában már rendelkezésre áll az elküldendő információ (pl. egy program futásának végeredménye, vagy egy előre megírt levél, stb.), akkor azt egyszerűen a standard input átirányításával küldhetjük el. Például:

 `$ mail galahad <fontos.level`

A program a `fontos.level` file-t elküldi a `galahad` felhasználónak.

Egyszerre több felhasználónak is elküldhetjük ugyanazt a levelet, ha a `mail` parancs után egyszerűen felsoroljuk a neveiket szóközzel elválasztva.

Ha a címzett nem ugyanazon a gépen dolgozik, ahonnan mi szeretnénk küldeni a levelet, akkor nemcsak a felhasználói nevet, hanem a másik gép nevét is ismernünk kell. A címbe ilyenkor a felhasználói név és a gép neve közé egy `@@` jelet kell tennünk.

³Rendszeres használat esetén célszerű az `alias mail="/usr/ucb/mail"` parancssort a `$HOME/.profile` állományba beírni. Ekkor a `mail` automatikusan az UCB mail-re vonatkozik.

Példa:

```
$ mail sirrobin arthur@@anglia.elte.hu galahad <fontos.level [Enter]
```

A program a `fontos.level` file-t elküldi a az `arthur` nevű felhasználónak, aki a `anglia.elte.hu` elnevezésű gépen dolgozik, valamint `sirrobin` és `galahad` felhasználónak a helyi gépen. Természetesen mindenki ugyanazt a levelet kapja meg.

Levél fogadása

Egy új levél érkezésekor a "YOU HAVE MAIL" (levele érkezett) üzenetet írja ki általában az operációs rendszer a képernyőre. Ha éppen dolgozunk a gépen akkor ez a levél érkezésekor megtörténik, máskülönben pedig a következő bejelentkezésünkkor kapjuk az üzenetet. Például az új levelet a

```
$ mail [Enter]
```

utasítással olvashatjuk el. A UNIX rendszer ekkor a következőképpen válaszol:

```
Mail version 2.18 5/19/83. Type ? for help.
"/usr/spool/mail/lancelot": 3 messages 3 unread
R  1 arthur Tue Sep 23 11:28 24/725 "Gyules"
N  2 arthur Thu Sep 26 13:32 10/315 "Hova tetted a Galt?"
U  3 arthur Thu Sep 26 14:35 11/305 "A fecske sebessége..."
&
```

A legelső sorban, rögtön a program neve után jelzi a `mail`, hogy `?`-et gépelve segítséget kaphatunk.

A következő sorban a levelesláda file nevét írja ki a rendszer, majd ezután következik a levelek felsorolása. Minden levélre kiíródik az állapota (`R` - már olvasott, `U` - még olvasatlan, `N` - új levél), a levél sorszáma, a küldő neve, a küldés ideje, a levél hossza sorban/karakterben, és végül a levél `Subject`: sora.

Az `&` karakter után a kívánt sorszámot beütve a levél tartalma megjelenik a képernyőn. Például:

```
& 2
```

```
Message  2:
From anglia Tue Sep  3 11:28:16 1991
Date: Tue, 3 Sep 91 11:26:57 -2300
From: arthur (Gral Knight)
To: lancelot
Subject: Hova tetted a Galt?
```

Lancelot,

mar napok ota keresem a Galt. Nem lattad valahol?

Arthur

A hosszabb levelek kiírását a `Ctrl-S-Ctrl-Q` gombokkal szabályozhatjuk.

A `mail` parancs & promptjánál különböző utasításokat adhatunk meg az éppen olvasott levélre vonatkozóan.

Példák:

 Az

& `s levelem.doc`

paranccsal kiírhatjuk a levelet egy `levelem.doc` nevű file-ba.

 A

& `d`

utasítás kitörli az adott levelet a levelesládából.

 Az

& `f*`

utasítás kilistázza az adott levelesládában található leveleket, ugyanúgy, mint a `mail` programba való belépéskor.

A programból a `q` (quit) vagy az `x` (exit) parancsokkal léphetünk ki. A `q` utasítás kilép a `mail`ből, és a levelesládát megváltoztatja az esetleges törléseknek megfelelően.

Az `x` parancs NEM változtatja meg a `mail`be való belépés előtti állapotot, azaz pl. nem törli ki a `mail`ben a `d` paranccsal törölt leveleket.

Figyelem! Az `q` és az `x` parancs pontosan "fordítva" működik, mint ahogy azt esetleg megszoktuk a VAX/VMS operációs rendszerben!

A `mail` több más utasítást is ismer, a 193. oldalon egy részletes leírás található a programról.

8.2.2. Az elm levelező program


Az `elm` elektronikus levelezőrendszert a


§ `elm`

paranccsal hívjuk be. Amennyiben az első alkalommal hívjuk be a gépen az `elmet`, feltétlenül olvassuk el a 203. oldalon található részletes magyarázatot a program beindításáról.

A UNIX `mail` programja és az `elm` csak két különböző, a UNIX rendszer levelezőszolgáltatását használó program, így ugyanonnan veszik az új leveleket. Ezáltal elképzelhető, hogy valaki felváltva használja őket.

Az `elm` főmenüje egy teljes képernyőt betöltő rendszer. A képernyő tetején a program kiírja a levelesláda helyét és azt, hogy hány levél található benne.

 `mail`
részletes
leírása a 193.
oldalon

 `elm`
első
elindítása a
203. oldalon

Többféle levelesláda (folder) lehet; a bejövő új leveleken, valamint az eltárolt régi leveleken kívül létrehozhatjuk magunknak még például az elküldött levelek másolatát, vagy egy adott témával kapcsolatos levelek gyűjtőjét is. Az `elm` az új posta vizsgálatával jelentkezik be:

```

Folder is '/usr/mail/guest' with 4 messages [elm 2.3 PL2]
N      1   Aug 17 Bokor Gyorgy      (25)   targyalas
O      2   Aug 19 J.F. Kenedi      (22)   Re: a tiedet
NU     3   Sep 28 jozsi@ludens.elit (37)   Mi ez!!!
       7   Oct 21 POSTMASTER      (76)   Undeliverable mail

```

You can use any of the following commands by pressing the first character
d)elete or u)ndelete mail, m)ail a message, r)eply or or f)orward mail, q)uit
To read a message, press <return>. j=move down, k=move up, ?=help

Command:

Minden levélről egy egysoros bejegyzést láthatunk (amennyiben van levelünk). A UNIX `mail`tól eltérően az `elm` olyankor is elindul, ha nincs új levelünk. A bejegyzés elején a levél státusza áll (új, még nem olvasott, törölt, stb.), azt követi egy sorszám, a levél beérkezésének dátuma, a feladó, a levél hossza sorokban, és a levél tárgya.

A képernyő alján rövid segítséget ad a program a legfőbb, az adott menüben használható utasításokról. Kilépni a `q` gomb lenyomásával tudunk a levelező-programból. Ha volt levelünk, akkor az `elm` a kilépés során megkérdezi, hogy a törlésre kijelölt leveleket kitörölje-e, az olvasott leveleket áttegye-e az olvasott levelek gyűjtőjébe. Minden kérdésre felajánl egy alapmegoldást, amit az `Enter`-re elvégez.


Levelet úgy tudunk elolvasni, hogy rámegyünk a kurzorral (a kiválasztott levél fordított színekkel vagy a `->` jellel van megjelölve a képernyőn). Mozogni a kurzormozgató nyilakkal tudunk, de beállítástól függően a `vi` szövegszerkesztő utasításai is működnek: a `j` lefelé, a `k` felfelé mozgat. Ha a kiválasztott levélen vagyunk, az `Enter` lenyomásával kezdhethetjük el azt olvasni. Az `elm` a `more` programot használja a levél tartalmának kiírására, így a `more` parancsait is használhatjuk.

Levelet a főmenüben az `m` (mail) gomb lenyomásával küldhetünk. A beépített `mail` levelezőrendszerhez hasonlóan ilyenkor az `elm` megkérdezi, kinek akarjuk küldeni a levelet a `Send the message to:` kiírásával. Ide kell beütni a címzett E-mail címét (vagy amennyiben létrehoztuk, a becenevét). Ezek után az `elm` a `Subject of the message:` kiírásával a levél tárgyát kérdezi meg (igen rövid leírása annak, hogy milyen ügyben írtuk a levelet). Amennyiben ezt a mezőt nem

töltjük ki, az `elm` megkérdezi, hogy folytatni akarjuk-e a levél elküldését. Ha igen, válaszoljunk `y`-nal. Az `elm` beállításától függően ezek után vagy mindjárt írhatjuk a levelet, vagy még előbb megkérdezi a program, hogy küldjön-e másolatot valakinek a `cc`: (Carbon Copy) kiírásával (ha nem akarunk senkinek másolatot küldeni, nyomjuk le az `Enter` gombot, különben írjuk be a címe(ke)t).

Amikor ezeken túljutottunk nagy valószínűséggel a `vi` szövegszerkesztőben találjuk magunkat. Ebben megírhatjuk a levelet, és a `vi`-ből kilépve az `elm` megkérdezi, hogy elküldje-e a levelet. Ezt az opciót ajánlja fel (`s` – send), így `Enter`-re a levelet elküldi. Ha nem akarjuk a levelet elküldeni, akkor a `f` (forget) lenyomásával állíthatjuk le a levél elküldését.

Az `elm` részletesebb leírását lásd a 16. fejezetben, a 203. oldalon.

 `elm`
részletes
leírása a 203.
oldalon

8.3. Hálózat használata

8.3.1. A `telnet` parancs

A `telnet` parancssal jelentkezhetünk be interaktívan egy távoli gépre. Például a `$ telnet rs1`

belép az `rs1` nevű gépbe, míg a

`$ telnet 193.6.30.1`

belép a 193.6.30.1 Internet számú gépre. Ilyenkor pl. a következő jelenik meg a képernyőn:

```
Trying . . .
Connected to rs1
Escape character is '^T'.
```

```
AIX telnet (rs1)
IBM AIX Operating System
Version 3.2 (C) Copyright IBM Corp. 1992
(/dev/pts0)
login:
```

A parancs kiadása után a helyi gépünk megpróbálja a távoli gépet elérni, és sikeres kapcsolatfelvétel esetén megjelenik a távoli gép promptja, mintha terminálról jelentkezniénk be.

A `telnet` parancs képes arra, hogy a távoli géppel a kapcsolat kiépítésekor megállapodjon a használt terminál típusában (bizonyos esetekben azonban ezt kézzel kell beállítani a rendszer `/etc/profile` vagy a felhasználó `$HOME/.profile` állományaiban). Amennyiben a termináltípust sikerült megállapítania a parancsnak, a távoli gépen automatikusan beállítja erre a `TERM` változó értékét. A beál-

lítandó termináltípust felülbíráhatjuk a `telnet` parancs `-e` kapcsolójával. Ismeretlen termináltípus esetén a `vt100` típus beállítása az esetek nagy részében működik.

8.3.2. Az `ftp` utasítás

Az `ftp` parancssal állományokat vihetünk át a helyi gép és egy távoli gép között. A parancs ezt az TCP/IP File Transfer Protocol (FTP) használatával valósítja meg. A parancs formátuma a következő:

```
$ ftp GépNév
```

ahol a *GépNév* a távoli gép Internet neve vagy címe. A kapcsolat kiépülése után az `ftp` megkérdezi a távoli gépen használandó azonosítót és kulcsszót.

Az `ftp>` prompt után az `ftp` programnak különböző utasításokat adhatunk, amikkel pl. könyvtárakat listázhatunk ki, állományokat vihetünk át az egyik gépről a másikra, stb.

A parancs ugyan nagymértékben tudja követni a felhasználói igényeket, de az FTP szabvány megkötései miatt nem tud bizonyos file paramétereket átvinni (pl. tulajdonos neve, védelem, dátum, stb.). Ha ilyen típusú átvitelt szeretnénk megvalósítani (pl. alkönyvtárak kijelölése, stb.), akkor a `tar` archiváló parancsot (l. 134. oldal) célszerű használnunk. Futtatható, archív és más bináris állományok átvitelénél az FTP `binary` üzemmódját kell használni.

Az `ftp` kapcsolatot automatikusan építi ki a program a megadott gépre, ha a felhasználó `~/.netrc` állományában a *GépNév*hez beírjuk az azon a gépen használandó felhasználói azonosító adatokat. Például ha a `.netrc` állományban a következő bejegyzés van:

```
machine rs1 login tas password huha
machine rs2 login kond
```

akkor az `ftp rs1` parancs automatikusan belép a `tas` felhasználói azonosítóra `huha` jelszóval. Az `ftp rs2` parancs elvégzi a belépést, de megkérdezi a `kond` felhasználó jelszavát, tekintve, hogy ez nincs benne a `.netrc`-ben. Ha el akarjuk kerülni, hogy az `ftp` használja a `.netrc` adatait, akkor `ftp -n GépNév` formátumot használjuk.

A `~/.netrc` állomány fontos személyes információkat tartalmaz, éppen ezért a UNIX rendszer nem használja ezt, amennyiben a file elérhető a csoport vagy a külvilág számára (megfelelő biztosíték pl. a 600-as védelmi kulcs beállítása a

```
$ chmod 600 $HOME/.netrc
```

utasítással).

Egy interaktívan futó ftp használatot a quit vagy bye utasításokkal, vagy az *End of File* (Ctrl-D) billentyűvel szakíthatunk meg az ftp> promptnál. Egy file átvitelt az *Interrupt* billentyűvel (Ctrl-C) szakíthatunk félbe.

Az ftp utasításai

Az ftp értelmezi az ftp> prompt után megadott parancsokat. Ha a parancsok használata során állománynévként a kötőjelet (-) adjuk meg, akkor a standard inputot (általában a billentyűzetet) használja a program bemenetként, a standard outputot (általában a képernyőt) pedig kimenetként.

A leggyakrabban használt ftp utasítások a következők (a gyakorlott felhasználók a felsoroltakon kívül több más parancsot is használhatnak, pl. makrózáshoz):

```
user Felhasználó
```

a helyi felhasználót a megadott *Felhasználó* paraméterekkel azonosítja a távoli gépen.

```
cd TávoliKönyvtár
```

belép a megadott *TávoliKönyvtárba* a távoli gépen.

```
lcd HelyiKönyvtár
```

belép a megadott *HelyiKönyvtárba* a helyi gépen.

```
pwd
```

kiírja az aktuális könyvtárat a távoli gépen.

```
dir [TávoliKönyvtár]
```

Kiírja az aktuális könyvtár (vagy a megadott *TávoliKönyvtár*) állománylistáját.

```
ascii
```

átkapcsolja az átviteli módot ASCII formátumba (alapértelmezés).

```
binary
```

átkapcsolja az átviteli módot bináris formátumba.

```
get TávoliFile [HelyiFile]
```

átmásolja a *TávoliFile*-t a helyi gépre. Ha megadjuk a *HelyiFile* paramétert, az új állomány neve ez lesz.

```
put HelyiFile [TávoliFile]
```

átmásolja a *HelyiFile*-t a távoli hostra (az esetlegesen megadott *TávoliFile* néven).


```

prompt
    bekapcsolja/kikapcsolja az interaktív promptot. Kikapcsolása általában az
    mput/mget parancsoknál szokásos.
mdelete TávoliFile
    kifejti a TávoliFile kifejezést, és kitörli a távoli gépen ezeket az állományo-
    kat.
mget TávoliFile
    kifejti a TávoliFile kifejezést, és átmásolja a távoli gépről ezeket az állomá-
    nyokat a helyi gépre.
mput [HelyiFile]
    kifejti a HelyiFile kifejezést a helyi gépen, és a megadott állományokat át-
    másolja a távoli gépre.
!Parancs [Paraméter]
    meghívja a shellt a helyi gépen az esetleges Parancscsal és Paraméterekkel.
quit
    lezárja az ftp bejelentkezést és kilép az ftp programból.

```

Az ftp-t például a következőképpen használhatjuk:



```

$ ftp rs1 rácsatlakozunk az rs1 gépre
Connected to rs1.
220 rs1 FTP server (Version 4.1 Fri Aug 28 11:37:57 GDT 1987) ready.
ftp> user kond felhasználói nevünk pl. kond
331 Password required for kond.
Password: megadjuk jelszavunkat, de ez nem jelenik meg
230 User kond logged in.
ftp> dir kérünk egy file listát a távoli (rs2) gépen
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 60
drwx-----  2 kond  boss      512 Oct 19 23:45 Mail
-rwxr-xr-x   1 kond  boss     1564 Oct 11 10:27 drawzdata
-rw-r-----  1 kond  boss     9546 Sep 30 16:37 dwhetc.c
-rw-r--r--   1 kond  boss     3051 Dec 06 03:44 smit.log
drwxr-xr-x   2 kond  boss      512 Dec 10 18:03 tmp
-rw-r-----  1 kond  boss     9820 Oct 11 10:28 z05001.gpl

226 Transfer complete.
827 bytes received in 0.19 seconds (4.2 Kbytes/s)

```

```

ftp> get dwhet.c           áthozzuk a dwhet.c állományt, ASCII (default)
                           üzemmódban a távoli gépről a helyi gépre
200 PORT command successful.
150 Opening ASCII mode data connection for dwhetc.c (9546 bytes).
226 Transfer complete.
local: dwhetc.c remote: dwhetc.c
9903 bytes received in 0.031 seconds (3.1e+02 Kbytes/s)
                           a két méret a sorvégi konverzió miatt nem egyezik
ftp> ls                   a helyi gépen megnézzük, megérkezett-e
config.tel                dwhetc.c                gro                tmp
ftp                        plsr                    wave
ftp> put config.tel       átvisszuk a távoli gépre a config.tel állományt
200 PORT command successful.
150 Opening ASCII mode data connection for config.tel. (6122 bytes)
226 Transfer complete.
local: config.tel remote: config.tel
6043 bytes sent in 0.039 seconds (1.5e+02 Kbytes/s)
ftp> binary átkapcsolunk bináris üzembe
200 Type set to I, binary transfer mode
ftp> mget z*              áthozzuk a távoli gépről az összes z-vel kezdődő file-
                           t.
mget z05001.gpl? y       az ftp egyesével végigkérdezi, hogy át akarjuk-e
                           hozni az adott állományt
200 PORT command successful.
150 Opening BINARY mode data connection for z05001.gpl (9820 bytes).
226 Transfer complete.
local: z05001.gpl remote: z05001.gpl
9820 bytes received in 0.036 seconds (272 Kbytes/s)
ftp> quit                kilépünk a programból
221 Goodbye.

```

8.3.3. A `finger` parancs

A `finger` paranccsal egy adott gépen található felhasználó(k)ról kaphatunk információt. A paranccsal egy adott gépen (alapesetben a helyi gépen) létező felhasználóról, vagy egy adott gépet éppen használó felhasználókról kaphatunk információt.

Példák:

Az `rs2` gépen interaktívan bejelentkezett felhasználókat a

```
$ finger @@rs2 
```

parancs listázza ki:

```
kond    console Mar 15 13:19
tas     pts0     Mar 15 13:01
huba    tty0     Mar 15 13:01
```

A `kond` felhasználóról az `rs2` gépen a

```
$ finger kond@@rs2 
```

parancs a következőt írja ki:

```
Login name: kond
Directory: /u/kond Shell: /bin/ksh
On since May 8 07:13:49 on console
No Plan.
```

A `finger` program ilyenkor kiírja a felhasználó `HOME` könyvtárát, a login shellt, és a `$HOME/.plan`, valamint a `$HOME/.project` file tartalmát is.

Bizonyos gépeknél a fenti információk módosulhatnak, megjelenhet pl. a felhasználó telefonszáma, van-e új levele, stb.

8.4. A tin elektronikus újság

A UNIX rendszerekben külön programokkal (ilyen pl. a `tin`, `trn`, `rn`, `inn`) olvashatjuk az UseNet elektronikus újságot. A `tin` nem része az eredeti rendszernek, de talán az egyik legegyszerűbben használható kedvelt *public domain* újságolvasó program.

Az UseNet újság olyan rovatokba van szervezve, melyek nagy része teljesen nyilvános, mindenki számára olvasható, bárki írhat beléjük. Ezek mellett lehetőség van teljesen zárt, csak meghatározott felhasználók számára olvasható rovatok megadására is, valamint bizonyos rovatokba csak a rovat szerkesztőjén keresztül lehet írni.


Az újságot a

```
$ tin -r 
```

vagy a

```
$ rtin 
```

paranccsal indítjuk el, a rendszer beállításától függően. Ezután az egyes rovatok listájával jelentkezik a képernyőn a program. A rovat neve tartalmazza, hogy mi

 tin
első
elindítása a
220. oldalon

célt szolgál, milyen körben osztják szét, stb. Amennyiben a legelső alkalommal hívjuk be a gépen a `tin`, olvassuk el a 220. oldalon található leírást a rovatok kiválasztásáról.

A program a képernyő tetején és alján néhány sorban kijelzi az aktuális állapotot (a `Ctrl-L` gomb mindig újraírja a képernyőt). A legelső sorban kiadhatjuk a kívánt `tin` parancsot, ill. a rendszer ide írja ki az állapot- és hibaüzeneteket. A felső sorban az adott állapot kijelzése mellett a program megadja, milyen szinten dolgozunk éppen. Ez háromféle lehet:

- rovatlista szint (Selection Group Level) : A rovatlista szintre a program elindítása után kerülünk.

Az egyes rovatok sorszáma mellett megjelenik a rovat neve és a rovatban található még nem olvasott cikkek száma. A `-` jel azt jelzi, hogy az adott rovatot még egyáltalán nem olvastuk, ha pedig semmi nincs kiírva, akkor a rovat utolsó olvasása óta nem érkezett újabb cikk. Például:

```

                                Group Selection (77)                                Type 'h' for help
21  comp.graphics.gnuplot                                78
22  comp.lang.fortran                                    -
-> 23  comp.lang.idl-pvwave                               6
24  comp.lang.postscript                                -
25  comp.laser-printers                                 -
26  comp.mail.elm                                       144
27  comp.sources.misc
28  comp.sources.postscript
29  comp.sources.unix
30  comp.sources.x

```

A `[j]` és `[k]` gombok (esetleg a fel- és lefelé nyilak) segítségével kiválaszthatjuk az éppen aktuális rovatot (ezt egy nyíl vagy inverz felirat jelzi a képernyőn). Az `[Enter]` gomb lenyomásával belépünk a jelzett rovatba. Ilyenkor megjelenik a rovat cikkliséje.

- cikkliséje szint (Group Level) : A `tin` ekkor az adott rovatához tartozó cikkek adatait (sorszám, válaszok száma, tartalom és feladó) jeleníti meg. Például:

```

                                comp.lang.idl-pvwave (23 33)                                Type 'h' for help
-> 1  + 2  Could I (please) be sent the FAQ??          jjunker@sdf.lonestar.org
2    +   I'm writing an FAQ for IDL                    pat@gsfc.nasa.gov (patric
3    +   applying poly_2d to larger images            knipp@ipi.uni-hannover.de
4    +   List widget problem                          goldberg@rwja.umdj.edu (

```

```

5      RENDER in PV CL 4.0          baird@thor.hor (DB Baird)
6      test                          baird@thor.hor (DB Baird)
7      Forcing motif type windows   baird@thor.hor (DB Baird)
8      FAQ: Where is the FAQ?       hong@cs.arizona.edu (Ke G
9      Request for neural network archite black@signal.dra.hmg.gb (

```

Egy adott cikket – az előző szinthez hasonlóan – a **[j]** és **[k]** gombokkal (esetleg a fel- és lefelé nyilakkal) választhatunk ki. A kiválasztott cikket az **[Enter]** gombbal kezdetjük el olvasni.

A szintről az **[q]** (quit) parancs tér vissza az előző, rovatlista szintre.

- cikkolvasás (Article Level) :

Ezen a szinten a kiválasztott cikk jelenik meg a képernyőn. A felső három sorban az adott cikkek adatai (dátum, rovat, sorszám, tartalom, az esetleges válaszok száma és a feladó) jelennek meg. Például:

```

1 Mar 1993 14:01:45 GMT          comp.lang.idl-pvwave          Thread 1 of 23
Article 49          Re: Could I (please) be sent the FAQ??      Respno 1 of 2
zawodny@arbd0.larc.nasa.gov  Dr. Joseph M Zawodny at NASA Langley Research

```

```

In article <C2xBGr.3so@sdf.lonestar.org> jjunker@sdf.lonestar.org writes:
>Hi:
>I'm new to this group and would like a copy of the most recent FAQ.
>Please, e-mail the FAQ to the address below. Thanks.
>
>                                     Jon
>                                     jjunker@sdf.lonestar.org

```

Jon

I'm not sure there is a FAQ for this group (at least I've never seen one). Just go ahead and ask the question.

--More-- (84%) [1037/1229]

Ha a cikk hosszabb, mint egy oldal (amire a `--More-- (84%) [1037/1229]` utal), akkor a **[Szóköz]**-zel (**[Space]**) lapozhatunk a következő oldalra.


Az `tin` program egy adott cikket és a rá érkezett „válaszokat” csoportként kezeli ún. „thread” formájában, ami nagyban megkönnyíti az egy témához kapcsolódó cikkek olvasását. A **[Szóköz]** parancs a cikk vége után automatikusan az első, második, stb. válaszlevélre ugrik, de közvetlenül is megadhatjuk a válaszlevél sorszámát. A 0 (nulla) mindig az eredeti levélre vonatkozik.

A szintről a **[q]** (quit) parancs (esetleg a balranyíl gomb) tér vissza az előző, cikklista szintre.

A `tin` minden szintjén egy saját (angol nyelvű) segítség-rendszert hívhatunk elő a `[h]` (help) utasítással.

A programból a `[q]` (quit) paranccsal léphetünk ki. A program ilyenkor megkérdezi, elmentse-e az olvasott cikkek listáját (célszerű ilyenkor `[y]`-nal válaszolni).

A `tin` sok más utasítással is rendelkezik (pl. cikkek beküldése, keresés a cikklistánban, stb.), az ez iránt érdeklődőknek ajánljuk a 219. oldalon található részletes leírást. Szintén ajánljuk a `news.announce.newusers` rovatot, ahol a kezdő olvasók sok értékes információt találhatnak.

 `tin`
részletes
leírása a 219.
oldalon

8.5. A `gopher` elektronikus adatkereső program

A TCP/IP hálózatok egyik érdekes lehetősége az elosztott információforrások használata. Ezek az adatok különböző gépeken vannak, de a felhasználó általában ezt nem érzékeli, mivel a keresőprogramok automatikusan váltanak az egyes távoli adatszolgáltató gépek között.

Az elosztott információforrások egyik megvalósítása az ún. `gopher` protokoll. A `gopher` — amely általában nem része az eredeti UNIX rendszernek — egy egyszerűen kezelhető, felhasználóbarát *public domain* adatkereső program. A felhasználó egy menürendszert lát, ahonnan egy menüpontot kiválasztva a program a hálózaton keresztül lekérdezi a megfelelő gépről a kívánt információt. Az egyes adatforrások között az átkapcsolás automatikus.

A lekérdező programot a

```
$ gopher 
```

vagy a

```
$ gopher IndulóAdatforrásGépNév 
```

paranccsal indítjuk el. Az első esetben a program a keresést az alapértelmezésben beállított gépen kezdi el, míg a második esetben az *IndulóAdatforrásGépNév* Internet névvel (vagy címmel) megadott gépről kérdezi le az induló menüt.

A program ezután az egyes adatok (ill. adatcsoportok) listáját írja ki a képernyőre. A listán az egyes sorszámok után röviden le van írva, hogy az adott pont mit tartalmaz. Például a

```
$ gopher ludens.elte.hu 
```

utasítás után a következő képernyő jelenik meg:

```
Root gopher server: ludens.elte.hu
```

```
--> 1. LUDENS gopher server/  
    2. ELTE gopher server/
```

Press ? for Help, q to Quit, u to go up a menu

Page: 1/1

A program a képernyő alján egy sorban kijelzi az aktuális állapotot (a `Ctrl-L` gomb újraírja a képernyőt).

Egy adott menüpontot a kurzormozgató fel- és lefelé nyilakkal, vagy a sorszám megadásával választhatunk ki. A kiválasztott pontra az `Enter` vagy a kurzormozgató jobbranyíl (`→`) gombbal léphetünk rá. Egy előző szintre a balranyíl (ez a kurzormozgató `←` gomb, nem tévesztendő össze a `Backspace` gombbal) vagy az `u` parancs lép vissza. A programból a `q` (quit) paraccsal léphetünk ki (a program ilyenkor megerősítést kér a `y` gombbal). A `h` (help) gomb kiírja a `gopher` utasításait.

Ha az előző példánál az első pontot választjuk ki, a következő menübe jutunk:

```
Internet Gopher Information Client v1.1
LUDENS gopher server
```

- ```
--> 1. Olvass el.
 2. VMS alapismeretek/
 3. Szolgáltatások a Ludens-en/
 4. Levelezési alapismeretek/
 5. Egyetemközi Hirdetotabla/
 6. NEWS Archivum/
 7. TTK-s NYUZ, 8. felevfolyam /
 8. Hollosi Information eXchange (HIX)/
 9. ELTE csillagaszati gopher server/
 10. Caesar gopher server/
 11. Csoma gopher server/
 12. BKE gopher server/
 13. The European root Gopher Server/
```

Press ? for Help, q to Quit, u to go up a menu

Page: 1/1

Egy adatnak többféle típusa lehet: újabb menüsint, szöveges adat, kép, hang, stb. A program automatikusan megkülönbözteti ezeket: belép egy újabb menüsintre, kiírja a szöveges adatot, áthozza a bináris állományt (pl. programot) a helyi gépre, megjeleníti a képet, megszólaltatja a hangot, stb. Az előző példában az újabb menüsintekre a `/` végződés, míg az adatfile-ra a `.` végződés utal.<sup>4</sup> Megjegyezzük, hogy a HIX (Hollosi Information eXchange) faliújságot, ill. ennek egyes rovatait is ezen a szerveren keresztül lehet elérni (8. pont).

A szöveges adat lekérdezése esetén azt a `gopher` a `more` parancs segítségével jeleníti meg: ha az adat hosszabb, mint egy oldal akkor a `Szóköz`-zel lapozhatunk a következő, a `b`-vel pedig az előző oldalra. A `q` gombbal kiléphetünk az olvasásból. Ha a `PAGER` környezeti változóban más programot (pl. `less`) adunk

<sup>4</sup>Ez csak ebben a példában van így, máshol ezt másképp vagy egyáltalán nem jelölik.

meg, akkor a program a `more` helyett azt használja. Az adatsor után a program megkérdezi, elmentse-e az adatot egy állományba (`s` opció), elküldje-e levélben valakinek (`m` opció), vagy egyszerűen lépjen vissza az előző menübe (`[Enter]` opció).

Bináris állományoknál a program megkérdezi, milyen nevű állományba hozza át a helyi gépre az adatokat. Kép megjelenítésére csak grafikus felülettel (pl. X11), hang megjelenítésére csak a szükséges hang kimenettel rendelkező gépek (pl. SUN munkaállomás) képesek. A `gopher` menüsztintjén az `[O]` (Opció) paranccsal állíthatjuk át az alapértelmezésbeli értékekről azokat a különböző UNIX-beli adatkezelő parancsokat, amelyeket a `gopher` az adott információ típusok esetén használ (ez pl. `xv` lehet képek megjelenítésére).

Bizonyos menüpontoknál a program keresést hajt végre egy nagyobb adatbázisban: ilyenkor meg kell adnunk a keresendő szót (*search string*).

A `gopher` program a gyakran olvasott menüpontokat képes egy ún. bookmark (olvasójel) állományba összefoglalni. Az aktuális menüpontot az `[a]` paranccsal adhatjuk ehhez az állományhoz. Az olvasójel listát a `[v]` paranccsal nézhetjük meg: ilyenkor itt ugyanúgy mozoghatunk és kereshetünk adatokat, mint a „fő” menüben. Egy adott pontot — amennyiben az olvasójel listán vagyunk — a `[d]` gombbal törölhetünk. Ha a programot a `gopher -b` utasítással indítjuk, akkor rögtön az olvasójel listába lépünk be.

Megjegyzendő, hogy a `gopher` mellett több más, hasonló adathálózat (pl. a World Wide Web (WWW), Hytelnet, stb.) is létezik. Ezeket vagy pl. egy `gopher-WWW` kapuként üzemelő gépen keresztül, vagy a `gopher` mellett más protokollt is kezelő programok segítségével érhetjük el (ilyen nagyon jól használható program pl. az ASCII alapú `lynx`, vagy az X11 Window alapú NCSA `xmosaic`).



## **II. Túl az alapokon**



## 9. fejezet

# A Korn shell

A UNIX rendszereken többféle shell (burok) program futtatható. A legrégebbi a Bourne shell (`sh` vagy `bsh`), illetve a C nyelv szintaxisát követő C-shell (`csh`).

A továbbiakban a Korn shellt (`ksh`) ismertetjük, amely az egyik leginkább elterjedőben levő shell. A shell felülről teljesen kompatibilis a Bourne shelllel, és a C-shell számos hasznos tulajdonságát is magába ötvözi. A legtöbb forgalomban lévő UNIX operációs rendszer a Korn shellt már eredetileg tartalmazza, de mivel a program közprogramként is beszerezhető, így a többi gépen is használhatjuk ezt a shellt.

Megjegyzendő, hogy az említett három shell mellett közprogramként feltűntek más, még többet tudó shellek is (`tcsh`, `zsh`, `bash` stb.), de ezek alapértelmezésben nem részei egyetlen forgalomban levő operációs rendszernek sem.

A Korn shell maga egy interaktív parancsértelmező és programozási nyelv, amely a következő funkciókat tartalmazza:

- shell parancsok
- függvények, feltételek és aritmetikai kifejezések kiértékelése
- különleges parancsok
- job vezérlés
- parancs helyettesítés
- állománynév helyettesítés
- bemenet és kimenet átirányítása
- helyettesítő nevek alkalmazása
- ~ (tilde) helyettesítés
- paraméter helyettesítés
- parancssor szerkesztése

Ha a Korn shellt bejelentkezéskor hívja meg a rendszer, az végrehajtja a `/etc/profile` shell állomány parancsait, majd a `$HOME/.profile` állomány shell parancsait. Ezután beolvassa az `ENV` környezetváltozó által definiált állományból a parancsokat. Amennyiben a `ksh` parancs után megadunk egy állománynevet, akkor az adott állományban felsorolt shell parancsok is végrehajthatók. A megadott állománynak olvashatónak kell lenni (l. `chmod` parancs).

## 9.1. A környezetváltozók

A környezetváltozók tárolják az egy-egy program számára átadható információkat. A környezetváltozónak van egy neve és ehhez rendelhetünk értéket. Értéket egyszerűen a `Név=Érték` utasítással adhatunk a változónak<sup>1</sup>. Ez az értékadás csak az adott shellre érvényes, ha azt akarjuk, hogy az abból nyíló újabb shellek is átvegyék a környezetváltozót, azt exportálni kell<sup>2</sup>:

```
$ export Név
```

A leggyakrabban használt környezetváltozók közé tartozik a `TERM`, mely a terminálunkat azonosítja a rendszer számára, a `LOGNAME`, mely bejelentkezési nevünket tartalmazza, a `PWD`, mely az aktuális könyvtár nevét adja meg, a `PATH` pedig a parancsok keresési útvonalát tartalmazza.

Interaktív használatkor a shell a parancsokra a `PS1` környezetváltozóban definiált prompttal vár. A Korn shell a `PS1` értékét a prompt kiírása előtt mindig kiértékeli. Így a

```
$ export PS1='$PWD>'
```

parancs hatására a prompt mindig az aktuális könyvtárnév lesz (a `PS1` változó mindig az aktuális `PWD` változó értékét fogja tartalmazni). Amennyiben a parancsot nem fejeztük be az „újsor” karakter (`Enter`) kiadása előtt (pl. a `'` vagy `"` nyitó karaktereknek nem adunk meg párt), akkor a Korn shell a `PS2` környezetváltozó értékét rakja ki másodlagos promptként.

## 9.2. A parancsok kiértékelése a shellben

Minden alkalommal, amikor átadunk egy parancsot a rendszernek, a shell kiértékeli, és feldolgozza azt. Amennyiben a rendszer egy speciális beépített parancsát

<sup>1</sup>C-shellben ez `set Név=Érték` alakú.

<sup>2</sup>A C-shellben a `set` helyett a `setenv` paranccsal kell ilyenkor értéket adni a környezeti változónak.

adjuk ki, az az adott shellben hajtódik végre (például a `test` parancsra a shell beépített utasítása hajtódik végre, még akkor is, ha létezik általunk létrehozott, `test` nevű futtatható állomány). A következő lépésben a shell leellenőrzi, hogy a kiadott parancs nem egy általunk megadott függvény-e. Amennyiben igen, a pozicionális paraméterek (azaz az adott parancs parancssorban lévő argumentumai) elmentődnek, majd behelyettesítődnek a függvénybe. Amikor a függvény lefut, a shell visszatölti az eredeti paramétereket. Egy függvény mindig az utoljára végrehajtott parancs visszatérési értékével fejezi be a futását, hacsak a `return` utasítással nem adunk neki más értéket. A függvény az aktuális shellben hajtódik végre.

Amennyiben a parancs fordítás és linkelés eredménye (pl. az `a.out` állomány), akkor az adott shellben futtatja le. Ha a utasítás se nem linkelt ill. beépített parancs, se nem függvény, akkor azt a shell parancsokat tartalmazó állománynak tekinti, és alshellt indít el azok beolvasására és végrehajtására. Ilyenkor a nem exportált alias- és függvénydefiníciók nem adódnak át. A zárójelek közti parancsok is alshellben futnak le, de megkapják a nem exportált változókat is.

A parancsot a rendszer a `PATH` környezetváltozó értékeként megadott könyvtárakban keresi. A `PATH`-ban több könyvtárnevet is megadhatunk, egymástól kettősponttal elválasztva. A parancs keresése a megadott felsorolási sorrendben történik. Az aktuális könyvtárat két egymás utáni kettősponttal jelöljük, esetleg a lista végén levő kettősponttal, de ki is írhatjuk a `.` könyvtárnévvel hivatkozva. Amennyiben a parancs `/` jelet tartalmaz, akkor a shell nem használja a `PATH` értékét, hanem megpróbálja útvonalként értelmezni a parancsot. A parancsot csak akkor indítja el a shell, ha annak védelmi kódja (amit a `chmod` paranccsal állíthatunk be) lehetővé teszi a futtatást.

### 9.3. Védőkarakterek

Minden ún. metakarakter (ezek a fordított törtvonal `\`, aposztróf `'`), fordított aposztróf `^`, macskaköröm `"`, gömbölyű `()`, szögletes `[]` és kapcsos `{}` zárójelek, kisebb és nagyobb jel `<>`, függőleges vonal `|`, és-jel `&`, pontosvessző `;`, csillag `*` kérdőjel `?` és a dollár `$` jel) speciális jelentést hordoz a shell számára. Amennyiben azt akarjuk, hogy a shell ne értelmezzen egy karaktert, akkor azt le kell védeni. Egy karaktert az elé helyezett fordított törtvonal `\` védi meg a shelltől. A sor végén álló `\` azt jelzi, hogy a parancs a következő sorban folytatódik. Több karaktert aposztrófok `'` közé kell tenni (ezzel az szövegben előforduló aposztróft nem tudjuk levédeni). A macskakörömök közötti szöveg-

részben végrehajtódik a paraméter- és parancshelyettesítés, de a fordított törtvonal levédi a fordított törtvonalat, a fordított aposztrófot (```), a macskakörmöt (`"`) és a dollár (`$`) jelet. A macskakörmök és aposztrófok közötti részt a shell egy szó-nak kezeli (egy pozíciós paramétert foglal). A `$*` és `$$$` pozíciós paraméterek (az összes átadott argumentum) egyforma hatású amikor nem tesszük macskakörmök közé, de macskakörmök között a `$*` egy szóként kezeli az összes átadott paramétert, míg a `$$$` minden paramétert külön szóba ír. A fordított aposztrófok között a fordított törtjel levédi a fordított aposztrófot és a dollár jelet. Amennyiben a fordított aposztróf macskakörmök között fordul elő, akkor a fordított törtvonal levédi a fordított aposztrófok közötti macskakörmöt is.

A fenntartott szavak valamint a helyettesítő nevek (alias) felismerését megakadályozhatjuk, amennyiben bármelyik karakterüket levédjük (ezek felismerése még a védőkarakterek kiértékelése előtt történik). A beépített parancsok, valamint a függvények felismerését ilyen módon nem akadályozhatjuk meg.

## 9.4. Parancssor szerkesztés

A Korn shell az utolsó 128 végrehajtott parancsot a `$HOME/.sh_history` állományban tárolja el. Az állomány neve a `HISTFILE` környezetváltozóval megváltoztatható, az eltárolt parancsok számát meg a `HISTSIZE` környezetváltozó értéke szabja meg. Alapbeállításban a begépelte parancsok a bejelentkezések között is eltárolódnak, és amennyiben egyszerre több bejelentkezésünk is fut ugyanazon a gépen, akkor mindegyik ugyanabba az állományba írja be a begépelte parancsokat, ezért más bejelentkezésben beírt parancsainkat is visszahívhatjuk.

A parancsok visszahívására az `fc` parancs szolgál. Ennél hatékonyabb módja is van a régi parancsok visszahívásának és szerkesztésének, ezért aki ezt a parancsot akarja használni, olvassa el a számítógépen található kézikönyv (man) oldalakat. A legtöbb gépen az `fc -e -` parancs az `r` betűhöz van hozzárendelve alias-szal, és az `r l` parancs végrehajtja a legutoljára begépelte, `l`-lel kezdődő parancsot (nemcsak egy betű adható meg a parancsból). Az `FCEDIT` környezetváltozó értéke az a szövegszerkesztő, melyet a szerkesztéshez szeretnénk használni, például a `vi` vagy az `emacs`. A szövegszerkesztőket a teljes útvonallal érdemes (path) definiálni. A szövegszerkesztő felfelé lépés (előző sor) parancsa hívja vissza az előző parancsokat, a lefelé lépés (következő sor) utasítással pedig az egyre későbbi parancsokat érhetjük el (`vi` üzemmódban a `[k]` és a `[j]` gombok használatát az `[ESC]` gombbal célszerű indítani, mivel beszúrási üzemmódból indulunk). A szerkesztőkarakterekkel (törlés, helyettesítés ...) javíthatunk a parancsokon, és az

`Enter` lenyomásával a már javított parancsot adjuk át a rendszernek.

A parancssor szerkesztését az `FCEDIT` definiálása nélkül, az `EDITOR`, illetve ennek hiányában a `VISUAL` környezetváltozók beállításával is megadhatjuk. Ha a változó értéke `emacs`, `gmacs`, vagy `vi`-ra végződik, akkor a megfelelő parancsszerkesztési mód lép működésbe. Általában ezek egyike sincs definiálva, erről a felhasználónak kell gondoskodnia (például a `.profile` állományon keresztül).


Az `EDITOR` és `VISUAL` környezetváltozók helyett a `set -o vi`, ill. a `set -o emacs` parancsokkal is beállíthatjuk a parancssorok szerkesztését. Ezeket a parancsokat szintén célszerű a `.profile` állományba beleírni.

## 9.5. Visszatérési értékek

Minden lefuttatott parancs egy u.n. visszatérési értéket ad vissza az őt elindító shellnek (programnak) Ha külön, a `return` utasítással nem adunk neki más értéket, akkor egy shell scriptben ezt normális lefutás esetén az utolsó lefuttatott parancs visszatérési értéke adja meg. A futás közbeni hibákat a shell kiírja a hibát tartalmazó sor számával (egynél több sor esetén), a hívott függvény nevével és a hibáüzenettel együtt.

## 9.6. A Korn shell kapcsolói

A Korn shell lefutatható programként is (akárcsak a többi shell). A `ksh` parancsoknak megadhatók kapcsolók is (melyeket a már futó shellben is állíthatunk a `set` parancssal). A legfontosabb kapcsolókat a `set` ismertetésekor, a 93. oldalon soroljuk fel.

 `set`  
részletes  
leírása a 93.  
oldalon

## 9.7. A Korn-shell beépített parancsai

A Korn shell a következő fenntartott szavakat ismeri fel, amennyiben azok a parancs első szavai, védőkarakatererek nélkül:

```
if for case then while esac else until function
elif do select fi done time {} [[]]
```

for Azonosító [in Érték ...]; do Lista ;done

Minden alkalommal, amikor a `for` parancs lefut, az *Azonosító* értéke az

*Érték* következő szavát veszi fel. Ha nem adjuk meg az `in Érték` kifejezést, akkor a *Listában* felsorolt parancsokat a pozíciós paraméterekre hajtja végre.

```
select Azonosító [in Érték ...] ;do Lista ;done
```

A `select` parancs kinyomtatja a standard hibakimeneten az *Értékben* felsorolt szavakat, sorszámmal ellátva. Ha nem adjuk meg az *Értéket*, a pozíciós paramétereket használja a shell. A PS3-ban definiált promptot kiírva olvassa be a választ. Amennyiben a beolvasott sor tartalmaz egy számot, akkor az *Azonosító* értéke a kiválasztott szó lesz, üres sor esetén a választási lista jelenik meg újra, egyébként pedig az *Azonosító* nem kap értéket. A beolvasott sort a `REPLY` környezetváltozó tartalmazza. Minden kiválasztásra végrehajtnak a *Listában* felsorolt parancs(ok).

```
case Szó in [[(] Minta[|| Minta] ...) Lista ;;] esac
```

A `case` parancs végrehajtja a *Szóval* egyező első *Mintához* tartozó *Listát*. A *Minta* az állománynév helyettesítésben ismertetett helyettesítő karaktereket tartalmazhat.

```
if Lista ;then Lista [elif Lista ;then Lista] ... [;else Lista] ;fi
```

Az `if` parancs végrehajtja az első *Listában* felsorolt parancsokat, és ha az utolsó nulla visszatérési értéket ad, akkor az utána következő *Listát* hajtja végre. Ellenkező esetben az `elif` utáni részt értékeli az előzőhöz hasonlóan. Amennyiben egy feltételnél sem nulla a visszatérési érték az `else` utáni *Listát* hajtja végre.

```
while Lista ;do Lista ;done
```

```
until Lista ;do Lista ;done
```

A `while` parancs folyamatosan végrehajtja az utána felsorolt parancsokat, és amennyiben az utolsó parancs visszatérési értéke nulla, végrehajtja a `do` utáni parancssorozatot, különben kilép a hurokból. Az `until` parancs hasonlóan működik, csak a feltételnek nulla értékére lép ki a hurokból.

```
(Lista)
```

A *Lista* parancs(sorozat) külön környezetben hajtja végre, így a végrehajtás során létrejött változások (környezetváltozó módosítása, könyvtárváltás) nem befolyásolják a munkakörnyezetünket. Amennyiben egymásba kell ágyazni több ilyen hívást, tegyük szóközt a zárójelek közé, mivel az aritmetikai kiértékelés használja a kettős zárójelet (a `((` jelet).

```
{Lista ;}
```

Egyszerűen végrehajtja a parancsokat. Mivel a kapcsos zárójel fenntartott



szó, ezért a shell csak akkor ismeri fel, ha a sor elején vagy a parancsokat elválasztó pontosvessző után áll.

```
[[Kifejezés]]
```

Kiértékeli a *Kifejezést*, és nulla értékkel tér vissza, ha az igaz.

```
function Azonosító {Lista ;}
 vagy az
```

```
Azonosító () { Lista ;}
```

az *Azonosító*hoz rendel egy függvényt. A függvény a {} közötti *Listában* felsorolt parancsokat hajtja végre.

```
time Parancssorozat
```

Végrehajtja a *Parancssorozatot* és kiírja a felhasznált rendszer- és felhasználói időt a standard hibakimeneten.

## 9.8. A függvények

A függvényeket a shell magában tárolja el definiálásukkor. A parancshoz hasonlóan a függvényeknek is átadhatunk pozicionális paramétereket.

A függvények a hívó paranccsal egy eljárásban (process) futnak le, és megszűnik annak erőforrásait (könyvtár, állományok, nyomkövetés, változók ...).

A függvényből a `return` speciális paranccsal térhetünk vissza. A függvény végrehajtása során fellépő hiba is felfüggeszti a függvény futását.

## 9.9. Feltételes kifejezések

A feltételeket a `test`, az `[ és ]` illetve a `[[ és ]]` parancspárral adhatjuk ki, állományok jellemzőinek vizsgálatára, számok, stringek összehasonlítására. A shell nem bontja fel szavakra, és nem hajt végre állománynév helyettesítést a `[[ és ]]` jelek között. Minden kifejezést a következő kifejezésekből állíthatunk össze:

```
-a File
```

Igaz, ha a *File* létezik.

```
-d File
```

Igaz, ha a *File* létezik, és könyvtár.

```
-f File
```

Igaz, ha a *File* létezik, és közösleges állomány.

`-n String`  
Igaz, ha a *String* nem nulla hosszúságú.

`-r File`  
Igaz, ha a *File* létezik, és olvasható az eljárás számára.

`-s File`  
Igaz, ha a *File* létezik, és nem nulla hosszúságú.

`-w File`  
Igaz, ha a *File* létezik, és az adott eljárás tudja írni.

`-x File`  
Igaz, ha a *File* létezik, és a futó eljárás végre tudja hajtani. Ha a *File* könyvtár, akkor a futó eljárásnak joga van keresni a könyvtárban.

`-z String`  
Igaz, ha a *String* nulla hosszúságú.

`-L File`  
Igaz, ha a *File* létezik, és szimbolikus link.

`File1 -nt File2`  
Igaz, ha *File1* létezik és újabb, mint *File2*.

`File1 -ot File2`  
Igaz, ha *File1* létezik és régebbi, mint *File2*.

`File1 -ef File2`  
Igaz, ha *File1* és *File2* is létezik, és ugyanarra az állományra mutatnak.

`String = Minta`  
Igaz, ha a *String* egyezik a *Mintával*.

`String != Minta`  
Igaz, ha a *String* nem egyezik a *Mintával*.

`String1 < String2`  
Igaz, ha a *String1* megelőzi az ASCII alapú könyvtári sorrendben a *String2* kifejezést.

`String1 > String2`  
Igaz, ha *String1* a *String2* után következik könyvtári sorrendben.

`Kifejezés1 -eq Kifejezés2`  
Igaz, ha a *Kifejezés1* egyenlő *Kifejezés2*-vel.

`Kifejezés1 -ne Kifejezés2`  
Igaz, ha a *Kifejezés1* nem egyenlő *Kifejezés2*-vel.

`Kifejezés1 -lt Kifejezés2`  
Igaz, ha a *Kifejezés1* kisebb, mint *Kifejezés2*.

*Kifejezés1 -gt Kifejezés2*

Igaz, ha *Kifejezés1* nagyobb, mint *Kifejezés2*.

*Kifejezés1 -le Kifejezés2*

Igaz, ha *Kifejezés1* kisebb vagy egyenlő, mint *Kifejezés2*.

*Kifejezés1 -ge Kifejezés2*

Igaz, ha *Kifejezés1* nagyobb vagy egyenlő, mint *Kifejezés2*.

Összetett kifejezéseket ezekből a következőképp gyárthatunk (csökkenő prioritással):

(*Kifejezés*)

Igaz, ha a *Kifejezés* igaz. A kifejezések csoportosítására használjuk.

!*Kifejezés*

Igaz, ha a *Kifejezés* hamis.

*Kifejezés1 && Kifejezés2*

Igaz, ha a *Kifejezés1* és a *Kifejezés2* is igaz. Az && helyett a -a feltétel is használható.

*Kifejezés1 ||| Kifejezés2*

Igaz, ha vagy a *Kifejezés1* vagy a *Kifejezés2* igaz. Az ||| helyett a -o feltétel is használható.

## 9.10. Speciális parancsok

A Korn shellbe beépített parancsok a shell eljáráson belül futnak le. Használhatjuk a bemeneti és kimeneti átírányítást is. Általában a kimenet az 1. állományleírójú állományba (standard kimenet) íródik, és sikeres végrehajtás esetén az utasítás nulla értékkel tér vissza. A következőkben a legfontosabb beépített parancsokat ismertetjük:

. *File*

Beolvassa a *File*-t, majd végrehajtja az aktuális shellben és környezetben. A *File*-t a PATH környezetváltozó alapján keresi meg. A visszatérési érték az utolsó végrehajtott parancs visszatérési értéke. A . parancs beolvassa a teljes *File*-t mielőtt végrehajtaná. Ezért a *File*-on belüli alias, stb. parancsoknak már nem lesz hatása a végrehajtás során. Ha interaktív bejelentkezésből futtatunk így, akkor az állományban található exit utasítás hatására mindjárt a shellből is kilépünk.

`alias [-tx] [Név[=Érték]] ...`

Argumentum nélkül az `alias` parancs kiírja a helyettesítőnevek listáját `Név=Érték` formátumban. Argumentumot megadva beállítja a `Név=Érték` formában megadott hozzárendelést. Például az `alias dir=ls` hatására a `dir` parancs is kiírja egy könyvtár tartalmát.

Az `x` kapcsoló hatására az utasítás exportált (azaz az a lejárások számára is elérhető) lesz, illetve kiírásakor csak az exportált helyettesítő neveket írja ki. A `t` kapcsoló bekapcsolja a nyomkövetést az adott `Névre`.

`break [n]`

Kilép a `while`, `until`, `for` vagy `select` hurokból. Az opcionális `n` paraméter mondja meg, hogy egymásba ágyazott hurkok esetén hány szintet lépjen feljebb.

`continue [n]`

A `while`, `until`, `for` vagy `select` következő ciklusát kezdi el. Az opcionális `n` paraméter mondja meg, hogy egymásba ágyazott hurkok esetén hány szintet lépjen feljebb.

`eval [Argumentum ...]`

Beolvassa az `Argumentum`okat shell inputként, és az eredményként létrejövő parancso(ka)t végrehajtja. Ezzel az utasítással **kétszer** történik meg az `Argumentumban` található speciális karakterek kiértékelése.

`exit [n]`

A shell befejezi futását és az `n` értékkel tér vissza. Ha nem adunk meg visszatérési értéket, akkor az az utolsó végrehajtott parancs értékével tér vissza.

`export [Név[= Érték]] ...`

A megjelölt neveket exportálja. Ezáltal az így kijelölt változók automatikusan átadódnak a shellből indított a lejárásoknak is, még ha közben meg is változtattuk értéküket.

`getopts OpcióString Név [Argumentum ...]`

Lellenőrzi a shell parancs `Argumentumát` (pozíciós paramétereit) az opciók szempontjából. Opció a `+` vagy a `-` jellel kezdődő betű (vagy szó). Ha az `argumentum` nem plusz vagy mínusz jellel kezdődik, vagy a `--` jel fordul elő, azon túl nem keres opciót.

Az `OpcióString` tartalmazza azokat a betűket, amit a `getopts` parancs opcióként felismer. Az adott betű utáni kettőspont azt jelzi, hogy annak az opciónak `argumentuma` van. Az opció `argumentumát` vagy közvetlenül az opció után, vagy attól szóközzel elválasztva kell megadni. A `getopts` parancs

a soron következő opcióbetűt a *Név* változóba helyezi, amennyiben az opció a + (vagy -) jel előzte meg. A következő argumentum pozícióját az `OPTIND` változó, az adott opció argumentumát az `OPTARG` változó tartalmazza.

`let Kifejezés ...`

Kiértékeli a megadott aritmetikai kifejezést. Nulla értékkel tér vissza, ha az utolsó kifejezés értéke nem nulla, máskülönben eggyel.

`print [-Rnprsu[n]] [Argumentum ...]`

Kinyomtatja az argumentumokat a standard kimenetre. A kapcsolók jelentése a következő: az `R` és `r` nem hajtja végre az ún. escape szekvenciák (`\n \t \b \r`, stb.) kifejtését, az `n` nem ír ki új sor karaktert a lista végére. A `p` kapcsoló csőbe ír, az `s` kapcsoló hatására pedig az utasítás beleíródik a `HISTFILE` állományba. Az `u` kapcsoló segítségével az `n` számmal megadott állományleíróba írhatunk.

`pwd`

Kiírja az aktuális könyvtárat. Hatása azonos a `print -r - $PWD` paranccsal.

`read [-prsu[n]] [Név?Prompt] [Név ...]`

Beolvas egy sort a standard bemenetről és szavakra bontja a mezőelválasztó karaktereket definiáló `IFS` környezetváltozó alapján. A sor elejére kiteszi a *Prompt*-ot, és a *Név* változóba olvassa a sort. Az `r` kapcsoló hatására szavakat az egymás után felsorolt *Név* változóba teszi, és a sorvégi fordított törtjelet nem értelmezi folytatósor jelzésre. Ha nem adunk meg nevet, a sort a `REPLY` változóba olvassa be. A visszatérési érték nulla, kivéve ha file vége jelet olvas be.

A `p` kapcsoló csőből olvas, az `s` kapcsoló hatására a parancs beleíródik a `HISTFILE` állományba. Az `u` kapcsoló segítségével az `n` számmal megadott állományleíróból olvashatunk.

`readonly [Név [=Érték] ] ...`

A megadott *Név*-et csak olvashatónak jelöli meg, és ezáltal a változó értékét nem lehet később megváltoztatni.

`return [n]`

A shell függvény futását lezárja és a hívó eljárásba adja vissza a vezérlést `n` visszatérési értékkel.

`set [+|-aefhkmnopstuvx] [+|-o Opció ]... [+|-A Név ] [Argumentum...]`

A shell különböző paramétereit állítja. A következő parancsokat használhatjuk:

- A Tömb hozzárendelés. Törli a *Név* előző értékét és egymás után hozzárendeli a felsorolt *Argumentum*okat. Ha a +A kapcsolót használjuk nem törli előzőleg az eredeti értéket.
- a Minden további definiált változó automatikusan exportált lesz.
- f Letiltja az állománynév helyettesítést.
- n Beolvassa és leellenőrzi a parancsokat, de nem hajtja őket végre.
- o Az utána megadható, leggyakrabban használt opciók a következők:

emacs : emacs módban lehet a parancssort szerkeszteni;  
 gmacs : gmacs módban lehet a parancssort szerkeszteni;  
 vi : a vi besúrási üzemmódjában hajtja végre a parancssor szerkesztését;  
 viraw : minden begépelte karaktert vi bemeneti karakterként dolgoz fel;  
 ignoreeof : a shell nem lép ki a file vége karakterre, csak az exit parancsra;  
 markdirs : minden az állománynévkiejtés eredményeként kapott könyvtárnév végére egy törtvonalat illeszt;  
 noclobber : megakadályozza hogy a > átirányítás hatására már létező állományokat felülírjon a shell. Ilyenkor a > || jellel érthetünk el felülírást;  
 nolog : megakadályozza, hogy a függvénydefiníciókat a shell a history file-ba mentse.

Ha nem adunk meg opciókat, a beállított opciókat írja ki.

- s A pozíciós paramétereket abc sorrendbe állítja.
- t Kilép egy parancs beolvasása és végrehajtása után.
- u A helyettesítés során hibának értelmezi, ha egy változó értéke nincs beállítva.
- v Kírja a végrehajtandó parancsot ahogy beolvassa.
- x Kírja a végrehajtandó parancsot kifejtés után ahogy végrehajtja.
- Kikapcsolja az x és v kapcsolók hatását.
- A kapcsolók feldolgozásának végét jelzi. Az utána következő argumentum már kezdődhet mínusz jellel anélkül, hogy azt a shell opciónak értelmezné.

Amennyiben a fenti kapcsolókat mínusz helyett plusz jellel adjuk meg, akkor az kikapcsolja az adott kapcsoló hatását. A kapcsolók aktuális beállítását a \$- környezetváltozó tartalmazza.

shift [n]

Átszámozza a pozíciós paramétereket \$<sub>n+1</sub>-től kezdve \$1, \$2 ...-re. Az

$n$  alapértéke 1, és megadható tetszőleges aritmetikai kifejezés, mely a pozíciós paraméterek számánál ( $\$ \#$ ) kisebb vagy egyenlő, nem negatív számot eredményez.

`times`

Kiírja a shell, és a belőle indított eljárások által felhasznált rendszer- és felhasználói időt.

`trap [Parancs] [Jel] ...`

Beolvassa és végrehajtja a `Parancs`ot ha a shell megkapja a `Jel`et. (A kiadható jeleket a `kill -l` paranccsal listázhatjuk ki.) A `Jel`et akár számként, akár a jel nevével is megadhatjuk. A nyomkövetés nem működik olyan jelekre, melyeket a shell indításakor vagy később a `set` paranccsal letiltottunk. A nyomkövetések a jelek sorszama szerinti sorrendben hajtódnak végre. Ha nem adunk meg `Parancs`ot, akkor törli a nyomkövetéseket eredeti értékükre. A `trap $HOME/.logout EXIT` parancs hatására a shellből való kilépéskor a rendszer végrehajtja a bejelentkezési könyvtárunkban levő `.logout` állományt.

`typeset [+HLRZfilrtux[n]] [Név [=Érték]] ...`

Beállítja a shell változók értékeit és típusát. Függvényhíváson belül létrehozott *Név* esetén azt külön helyen tárolja, és a függvényhívás végén visszaállítja annak eredeti, shellbeli értékét. A kapcsolók jelentése a következő:

- `L` Levágja a szó elején levő szóközöket, és balra rendezi az *Érték*et. Az  $n$  definiálja a mező szélességét. Ha nem adjuk meg, akkor az első hozzárendelésből állítja be értékét a shell.
- `R` Jobbra rendezi a változó értékét.
- `Z` Jobbra rendez, amennyiben az *Érték* számmal kezdődik.
- `f` Azt jelzi, hogy a *Név* függvény.
- `i` A változó egész típusú. Ez meggyorsítja az aritmetikai műveleteket. A kiíráshoz használandó számrendszert az  $n$  értéke határozza meg.
- `l` Kisbetűkre konvertálja a nagybetűket. A nagybetű  $u$  kapcsolóját hatálytalanítja.
- `r` A kijelölt *Neve*ket csak olvashatónak deklarálja.
- `u` A kisbetűket nagygyá konvertálja. Az `l` kapcsoló ellentéte.
- `x` A felsorolt *Neve*ket exportálásra jelöli ki.

Ha a kapcsolónál a `-` jel helyett a `+` jelet használjuk, akkor az kikapcsolja az adott kapcsoló hatását. Ha nem adjuk meg a *Név* paramétert, akkor a megadott kapcsolóhoz tartozó *Neve* listáját adja vissza. Ha kapcsolót sem adunk meg, minden változót kiír az *Érték*ével.

`ulimit [-HSacdfmst] [Limit]`

Beállítja, vagy (ha nem adjuk meg a *Limitet*) megmutatja az erőforrás határokat. Nem minden UNIX rendszer támogatja az itt felsorolt erőforrásokat. A *Limit* értéke vagy a típusától függő szám, vagy `unlimited` ha nem adunk meg rá korlátot. Kétféle határ van, a kemény határ (`H` kapcsoló), és a puha határ (`s` kapcsoló). A kemény határt általában a rendszergazda állítja be (pl. a rendszer fizikai határértékeire), és ezeket a felhasználó nem tudja átállítani. A puha határ az a kemény határon belül levő érték, amit a felhasználó a kemény határértékig megváltoztathat. A parancs, hacsak nem adjuk meg a `H` kapcsolót, a puha határokat írja ki. A többi kapcsoló jelentése:

- a Kilistázza az összes határértéket.
- c A core dump (futási hiba esetén a program memóriaképe a core állományba másolódik, és ezt a `dbx` paranccsal megvizsgálhatjuk) maximális mérete 512 byte-os egységekben.
- d Az adatmezőnek fenntartott hely maximális mérete kbyte-okban.
- f Az aljárások által kiírható maximális állomány nagyság 512 byte-os blokkokban.
- m A felhasználható fizikai memória nagysága kbyte-okban.
- s A stack terület legnagyobb mérete kbyte-okban.
- t Az eljárás által felhasználható maximális rendszeridő másodpercben.

`umask [Mask]`

Az újonnan létrehozott állományok alap védelmi beállítását szabályozza. A *Mask* értéke vagy egy oktális szám, vagy a `chmod` paranccsban ismertetett szimbolikus érték. Szimbolikus megadás esetén az `umask` új értéke a komplementere lesz annak a maszknak, amit úgy kapunk, hogy az az új *Maskot* alkalmazzuk a régi *Mask* értékének komplementerére. Ha nem adjuk meg a *Maskot*, kiírja annak értékét.

`unalias Név ...`

Kiveszi a felsorolt *Neveket* az alias listából.

`unset [-f] Név ...`

A felsorolt *Nevekhez* tartozó értékeket és jellemzőket kitörli. A csak olvashatónak kijelölt változók nem törölhetők. Az `f` kapcsoló esetén a *Név* függvénynév. Az `ERRNO`, `LINENO`, `MAILCHECK`, `OPTARG`, `OPTIND`, `RANDOM`, `SECONDS`, `TMOUT`, `_` változókra alkalmazva megszünteti azok speciális jelentését, mégha a későbbiekben újradefiniáljuk is őket.

`wait [Job]`

Vár a megadott eljárásra (*Job*), és tudósít futásának befejezéséről. Ha nem



adjuk meg az eljárást, az összes aleljárásunk végére vár. A megadott eljárás visszatérési értékét adja vissza.

whence [-pv] *Név* ...

Minden megadott *Névre* megmutatja, hogyan értelmezné a shell parancsként. A `p` kapcsoló hatására végignézi a `PATH` környezetváltozó alapján, hogy hol található a megadott parancs, mégha a megadott *Név* alias, függvény vagy fenntartott szó is. A `v` kapcsoló részletesebb kiírást eredményez.

## 9.11. Aritmetikai Kifejezések

Egész típusú számolásokat a `let` speciális parancs segítségével hajthatunk végre. A kiértékelés az ún. *long integert* (a mai gépeken ez általában egy 32 bites számot jelent) használja. A konstansokat `[Bázis#] Szám` módon adhatjuk meg, ahol a *Bázis* egy 2 és 36 közötti, tízes számrendszerbeli érték, mely a *Szám* megadásához használt számrendszer alapját definiálja. Alapértéke 10.

Az aritmetikai kifejezések a C nyelv szintaxisát, prioritási sorrendjét és csoportosítási szabályait követik. A `++`, `-`, `?:` kivételével minden egész típusú művelet megengedett. Az aritmetikai kifejezésekben a változókra hivatkozhatunk a nevükkel, nem kell a paraméter helyettesítési szabályokat (dollár jel) alkalmazni.

Mivel sokszor lehet szükség az aritmetikai kifejezésekben a macskaköröm használatára, ezért a `let` parancson kívül megengedett a dupla zárójel (azaz a `(( ... ))`) használata is. A dupla zárójelek közötti részt macskakörmözött aritmetikai kifejezésként értelmezi a shell.

## 9.12. Parancs eredményének helyettesítése

Egy parancs kimenetét felhasználhatjuk egy másik parancsban is. Hagyományosan ezt a fordított aposztrófokkal lehet elérni, de a Korn shellben használható még a `$( Parancs )` alak is. Mindkét esetben a kimenetről a shell kiszűri a lezáró „új-sor” karaktert. Az aritmetikai kifejezéseket a `$( ( Kifejezés ) )` módon írhatjuk be parancssorba. A fordított aposztrófok esetén a shell kifejti a védőkaraktereket, mielőtt végrehajtaná a parancsot.

### 9.13. Állománynév helyettesítés

A parancs minden szavát a shell leellenőrzi a \* ? [ , karakterek szempontjából. Amennyiben ilyet talál, akkor a szót mintának tekinti, és behelyettesíti ABC sorrendben a mintának megfelelő állománynevekkel. Ha nem talál egy állománynevet sem, akkor változatlanul hagyja a szót. A speciális karakterek a következő helyettesítést jelentik: a csillag helyettesít minden karakterláncot, beleértve a nulla stringet is. A kérdőjel egy karaktert helyettesít. A szögletes zárójelek közt felsorolt karakterek között az egyikkel egyezni kell. Megadhatunk egy egész karaktertartományt is a - jel használatával. Ha a szögletes zárójelen belül az első karakter a felkiáltójel, az negálja a kifejezést (minden fel nem sorolt karakterrel történhet egyezés). A mínusz jelet a legelső karakterként adhatjuk meg. Az [a-zA-Z] minta az összes (kis vagy nagy) betűt jelenti.

Összetett mintákat a következő eljárással hozhatunk létre: jelölje a *MintaLista* az egy, vagy több || jellel elválasztott mintát. Ekkor

? (*MintaLista*) Az adott minták közül bármelyikkel egyezik.

\* (*MintaLista*) Az adott minták egyszer sem, vagy többször fordulnak elő.

+ (*MintaLista*) Az adott minták egyszer vagy többször fordulhatnak elő.

@ (*MintaLista*) Csak az egyik megadott mintával egyezik.

! (*MintaLista*) Egyik megadott mintával sem egyezik.

A .-tal kezdődő állományneveknél expliciten ki kell tenni a pontot a minta elejére, különben a shell nem helyettesíti be azokat.

Lehetőleg ne adjunk olyan állomány és könyvtárneveket, melyek csillagot, kérdőjelet vagy szögletes zárójeleket tartalmaznak.

A szögletes zárójelek között használhatók még az előre definiált karakterosztályok is [:*Osztály*:] alakban, ahol az osztály a C-nyelv ctype rutinjainak megfelelően a következő lehet: *alnum*, *alpha*, *digit*, *lower*, *print*, *punct*, *space*, *upper*, *xdigit*.

### 9.14. Ki- és bemenetek átirányítása

A shell — még mielőtt végrehajtaná a parancsokat — lehetőséget biztosít, hogy azok bemenetét és kimenetét átirányítsuk. Az átirányítási parancsokra is végrehajtható a shell helyettesítése, de csak a kifejtett értékek adódnak át a meghívott programnak.

<*File*

A *File* állományból veszi a standard bemenetet.

>*File*

A *File* állományba teszi a standard kimenetet. Amennyiben az állomány létezik, felülírja, ellenkező esetben létrehozza.

>||*File*

Ugyanaz, mint a >, de a `noclobber` opció esetén is felülírja a létező állományt.

> >*File*

A *File* állományba teszi a standard kimenetet úgy, hogy létező állomány esetén annak végéhez hozzáír.

<>*File*

A *File* állományt írás-olvasásra nyitja meg standard bemenetként.

<<*Szó*

A shell a *Szó* előfordulásáig vagy a file vége jelig olvassa be a standard inputot. Az így keletkezett szöveg lesz a standard bemenet. Amennyiben a *Szó* bármely karaktere le van védve, akkor a keletkezett szövegen nem történik shell helyettesítés, különben paraméter és parancssor helyettesítés zajlik le. Ha a *Szó* elé a - karaktert helyezünk, akkor ez a szövegből kivágja a sor eleji tabulátorokat.

<&*Számjegy*

A standard inputot a *Számjegy* állományleíróból kapja.

>&*Számjegy*

A standard kimenetet a *Számjegy* állományleíróba teszi.

A fenti jelek előtti szám az adott számhoz tartozó állományleíró átirányítását jelöli. A `2>&1` helyettesítés a 2-es állományleírójú eszközt (standard hibakimenet) az 1-be (standard kimenet) irányítja. Többszörös átirányítás esetén ügyelni kell a sorrendre. A `1>File 2>&1` kifejezés az 1. állományleírót a *File*-hoz rendeli, majd a 2. állományleírót az 1.-höz, ami már a *File*, azaz mindent a *File*-ba ír. Fordított sorrend esetén a hibakimenetet a standard kimenetre, az eredeti standard kimenetet pedig a *File*-ba írja.

## 9.15. Helyettesítő nevek (alias)

Minden parancs első szavát a shell kicseréli, amennyiben az adott szót helyettesítő névnek (alias) definiáltuk. A helyettesítő név első karaktere tetszőleges nyomtatható karakter lehet, de a többi karakterre ugyanazok a megkötések érvényesek,

mint a változó nevekre. A helyettesítő szöveg (amire a shell az alias-t kicseréli) tartalmazhat bármely létező shell scriptet, akár metakarakterekkel is. A shell a helyettesítő szöveg összes parancsának első szavát is megpróbálja helyettesítő névként kifejtetni. Amennyiben a helyettesítő érték utolsó karaktere szóköz, akkor a shell az alias utáni szóra is megpróbál alias helyettesítést elvégezni. Aliasokat használhatunk a beépített parancsok újradefiniálására, de nem használhatók a fenntartott szavak átdefiniálására. Helyettesítő neveket az `alias` paranccsal hozhatunk létre, és az `unalias` paranccsal törölhetünk. Ha aleljárásokban is akarunk használni egy helyettesítő nevet, azt exportálni kell.

Az alias helyettesítés a parancs beolvasásakor történik meg, és nem végrehajtásakor. Ezért időben kell gondoskodni a helyettesítő név definiálásáról.

A következő exportált aliasokat beépítették a Korn shellbe (de ezek törölhetők és átdefiniálhatók):

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
```

## 9.16. A tilde helyettesítés

Az alias helyettesítés után a shell megvizsgál minden szót, hogy nem `~` jellel kezdődik-e. Az utána a `/` jelig következő szót a shell megnézi, hogy előfordul-e a `/etc/passwd` állományban felhasználói névként. Ha igen, akkor az adott felhasználó bejelentkezési könyvtárát helyettesíti be a szó helyére, ellenkező esetben nem történik helyettesítés. Amennyiben a `~` a `/` jelet közvetlenül megelőzi, akkor a `HOME` változó értékére helyettesítődik. A `~+` a `PWD`, a `~^` az `OLDPWD` változó értékére helyettesítődik.

## 9.17. Paraméter helyettesítés

A paraméter egy azonosító, a `*`, `@@`, `#`, `?`, `-`, `$`, `!` karakterek valamelyike, illetve egy vagy több számjegy lehet. A számjegyes paraméterek az ún. pozicionális paraméterek, melyek egy shell hívásakor a shellnek átadott argumen-

tumokat tartalmazzák a megadott sorrendben. A speciális karakteres paraméterek a shell által használt és általában beállított paraméterek.

A paramétereknek általában értékük és beállított tulajdonságaik vannak. A beállítható tulajdonságokat a `typeset` parancsnál, a 95. oldalon ismertettük.

A Korn shell támogatja az egy dimenziós tömbök használatát. A tömb egy elemére annak indexével (szögletes zárójelek közti aritmetikai kifejezés) hivatkozunk. Értéket egy tömbnek a `set -A` parancssal adhatunk. A tömb indexe 0 és 512 közötti érték lehet. E határok közt a shell automatikusan létre is hozza a tömböt. Ha egy tömbre index nélkül hivatkozunk, az a 0. elemét jelöli.

Egy paraméter értékét megadhatjuk még egyszerű hozzárendeléssel:

```
§ Név=Érték [Név=Érték]
```

formában is. Amennyiben `-i` (egész) típusúnak állítottuk be a *Név* változót, akkor az *Érték* egy aritmetikai kifejezés is lehet, amit a shell kiértékel. A pozicionális paramétereknek a `set` speciális parancssal adhatunk értéket. Ha scriptet hívunk meg, akkor a `$0` a hívó parancs neve, `$1`, `$2`, ... pedig az átadott argumentumok. A `$*` és a `@@` az összes átadott argumentumot jelenti.

A shell a következő helyettesítéseket hajta végre:

```
§{Paraméter}
```

A shell beolvassa az összes két kapcsos zárójel közti karaktert egy szóként, mégha kapcsos zárójeleket vagy metakaraktereket tartalmaz is. Amennyiben definiálva van ilyen változó, annak értékét helyettesíti be. A kapcsos zárójeleket elhagyhatjuk, ha nem kell a változó neve után közvetlenül egy betűt, számot, vagy aláhúzás karaktert illeszteni, vagy ha a paraméternek indexet adunk. Az egy számjegynél nagyobb pozicionális paramétereket is kapcsos zárójelbe kell tenni.

```
§{#Paraméter}
```

A *Paraméter* értékének a hosszát helyettesíti be. A `*` illetve a `@` *Paraméternél* a pozíciós paraméterek számát adja meg.

```
§{#Azonosító[*]}
```

A tömbben található elemek száma.

```
§{Paraméter:-Érték}
```

Ha a *Paraméter* létezik, és nem nulla az értéke, akkor azt helyettesíti be, különben az *Érték*et.

```
§{Paraméter:=Érték}
```

Ha a *Paraméter* nem létezik, vagy nulla az értéke, akkor az *Érték*re állítja be, és ezután helyettesíti be.

`${Paraméter:?Érték}`

Ha a *Paraméter* létezik, és nem nulla az értéke, akkor azt helyettesíti be, különben kiírja az *Értéket* és kilép a shellből.

`${Paraméter:+Érték}`

Ha a *Paraméter* létezik, és nem nulla az értéke, akkor az *Értéket* helyettesíti be, különben semmit nem helyettesít be.

`${Paraméter#Minta}` vagy `${Paraméter##Minta}`

Ha a *Paraméter* értékének eleje egyezik a *Mintával*, akkor a *Mintának* megfelelő részt kitörölve helyettesíti be a *Paramétert*, különben a teljes értékét. A # jelölés a minimális, a ## jelölés a maximális egyező tartományt törli ki.

`${Paraméter%Minta}` vagy `${Paraméter%%Minta}`

Ugyanaz, mint az előző, csak a *Paraméter* értéke végének egyezését figyeli.

A fenti esetekben az *Érték* egy kifejezés is lehet, és nem értékelődik ki, hacsak nem őt használja a shell helyettesítésre. Például az `echo ${d:-$(pwd)}` utasításban a `pwd` parancs nem kerül végrehajtásra, ha a `d` paraméter értéke nem nulla.

A következő paramétereket a shell automatikusan beállítja:

# A shellnek átadott pozicionális paraméterek száma.

- A shell hívásakor bekapcsolt opciók.

? Az utoljára végrehajtott parancs visszatérési értéke. A legtöbb parancs sikeres végrehajtáskor 0 értékkel tér vissza.

\$ A shell processz-száma (PID). Leggyakoribb alkalmazása egyedi állománynév generálása: pl. `a /tmp/temp.$$` a processz egyedi számával ellátott állománynevet jelenti.

\_ Az előző parancs utolsó argumentuma.

! A háttérben utoljára elindított eljárás processz-száma.

ERRNO A hibakóddal utoljára visszatért eljárás hibakódja.

LINENO A script állományon vagy a függvényen belüli, éppen végrehajtás alatt álló sor száma.

OLDPWD A `cd` parancs által beállított előző munkakönyvtár.

OPTARG A `getopts` parancs által feldolgozott utolsó opció argumentuma.

OPTIND A `getopts` parancs által feldolgozott utolsó opció indexe.

PPID A shell szülőeljárásának processz száma.

PWD Az aktuális munkakönyvtár.

RANDOM Minden híváskor egy 0 és 32767 közötti "véletlen számot" generál. A kezdőértéke beállítható.

REPLY A `select` és a `read` speciális parancsok által beállított változó, mely a választ tartalmazza.

SECONDS A shell meghívása óta eltelt időt írja ki másodpercekben. Ha kezdőértéket adunk neki, onnan számolja az időt.

A shell a következő változókat használja:

CDPATH A `cd` parancs keresési útvonala.

COLUMNS A shell ennek a változónak az értékét használja a szerkesztési módokban és a `select` parancs alkalmazásakor az ablak szélességének megállapításához.

EDITOR Ha ennek értéke `emacs`, `gmacs`, vagy `vi`-ra végződik, és a `VISUAL` változó nincs beállítva, akkor a megfelelő parancsszerkesztési mód lép működésbe.

ENV Új shell meghívása előtt e változó a paraméterhelyettesítéssel kapott állománynevet futtatja le. Általában helyettesítő nevek és függvények definiálására használják.

FCEDIT Az `fc` parancs alatt használandó szövegszerkesztőt definiálja.

IFS A shell szóbetördelésére használt elválasztókaraktereket tartalmazza (szóköz, tabulátor, új sor).

HISTFILE A végrehajtott parancsokat tároló állomány neve.

HISTSIZE Értéke adja meg, hogy az utolsó végrehajtott parancsok közül hányat tárol el a shell.

HOME A `cd` parancs argumentum nélkül ebbe a könyvtárba ugrik. A bejelentkezési eljárás a bejelentkezési könyvtárra állítja az értékét. Ha több gépen dolgozunk, célszerű programjainkat úgy megírni, hogy ne a teljes file-nevekkel, hanem ennek a változónak az értékével hivatkozzunk az állományokra, mivel ezáltal programjaink áttehetők egyik gépről a másikra.

LINES A `select` használja a rendelkezésre álló sorok meghatározására. A `select` a listát a `LINES` értékének kétharmadáig írja ki.

MAIL Az új leveleket tartalmazó állomány neve. Ha a `MAILPATH` változó nincs beállítva, akkor a shell az állomány megváltozását új levél érkezéseként jelzi.

MAILCHECK Értéke meghatározza, hogy a shell milyen gyakran ellenőrizze le a `MAILPATH` vagy `MAIL` változóiban megadott állományokat. Alapértéke 600 másodperc.


MAILPATH A benne felsorolt, kettősponttal elválasztott állományneveket ellenőrzi a shell új levél érkezése céljából. Minden állománynév után közvetlenül a kérdőjel után megadhatjuk az üzenetet, amit a shell kiír, ha változás történt az állományban. Az üzenetben a shell elvégzi a paraméter helyettesítést. A

megváltozott állomány nevét a `$_` változóban tárolja. Az üzenet alapbeállítás: `YOU HAVE MAIL IN $_`.

- PATH** A parancsok keresési útvonala. Egymástól kettősponttal elválasztott könyvtárneveket tartalmaz, melyeknek megadási sorrendjében keresi a shell a kiadott parancsnak megfelelő állománynevet. A nulla könyvtárnév az aktuális könyvtárat jelöli.
- PS1** Az elsődleges rendszer promptot tartalmazza. Értéke a prompt kiírása előtt átmegegy a paraméter helyettesítésen. Ha felkiáltójelet adunk meg benne, az a parancs sorszámára helyettesítődik. Alapértéke a dollár és egy szóköz.
- PS2** A másodlagos rendszer prompt. Alapértéke a nagyobbjel és egy szóköz.
- PS3** A `select` parancsban a prompt értéke. Alapértéke a `#?` és egy szóköz.
- SHELL** A shell elérési útvonala. (Sok program ezt használja, ha ki akarunk lépni belőle a shellbe).
- TMOUT** Ha a `PS1` prompt kiadása után a `TMOUT`-ban megadott számú percre nem adunk ki parancsot, a shell befejezi futását. Akkor célszerű használni, ha elszoktuk felejtetni a kijelentkezést. 0 értéke esetén nem lép ki a shell.
- VISUAL** Ha ennek értéke `emacs`, `gmacs`, vagy `vi`-ra végződik, akkor a megfelelő parancsszerkesztési mód lép működésbe.

A shell a `PATH`, `PS1`, `PS2`, `MAILCHECK`, `TMOUT`, `IFS` változóknak ad értéket, míg a `HOME`, `SHELL`, `ENV`, `MAIL` változókat nem állítja be (a `HOME` értékét a `login` parancs állítja be).

A környezeti változók beállítására következzen egy mintaként használható

 `.profile` állomány:

```
@(#) .profile 5.1 ULTRIX 4/23/91
umask 22
export PATH=$PATH:/u/local:/usr/local/bin:/usr/bin/X11:.
PS1=' $LOGNAME '@ hostname ' $ '
VISUAL=vi
FCEDIT=vi
export TERM=vt100
export VISUAL FCEDIT
export PS1
if [-z "$DISPLAY"]
 then
export DISPLAY=\who am i || awk '{l=length($6); print substr($6,2,l-2) ":0.0"
 else
if ["$DISPLAY" = ":0.0"]
```



```

 then
 export DISPLAY=":0.0"
 else
export DISPLAY=\who am i || awk '{l=length($6); print substr($6,2,l-2) ":0.0"}'\
 fi
 fi
alias ls='ls -g '
alias archie='telnet 131.130.1.23'

```



Az állományban először beállítjuk az ún. `umask` értékét (azaz a 755-ös alap védelmi kódot), majd a `PATH`, `PS1`, `VISUAL`, `FCEDIT` és `TERM` környezeti változókat (figyeljük meg, hogy a `PS1` értéke egy shell parancsot is tartalmaz!) Ezután egy dupla `if` feltételvizsgálattal megállapítjuk, hogy honnan jelentkezünk be (helyi gépről vagy a hálózaton keresztül), és ennek megfelelően beállítjuk az X11 Window által használt `DISPLAY` változó értékét (ez adja meg az X11 ablakokat megjelenítő grafikus terminál nevét). Az állományt néhány `alias` definícióval zárjuk: az `ls` parancs a csoportot is fel fogja tüntetni a hosszabb file listákban, az `archie` parancs pedig egy bécsi `archie`-szerver gépre jelentkezik be (ott felhasználói azonosítóként szintén az `archie`-t használjuk!).

A Korn-shell parancsainak használatára vegyük a következő parancsállományt, amely helyi hirdetőtáblaként működtethető:

```


#!/usr/bin/ksh
PS3="Enter your choice> "
select ans in "exit" \ls -d /u/local/info/*/* \
 || sed "s\/u\/local\/info\/\(.*\)\.news\/1/"\
do
 if [[$ans = exit]]
 then
 exit
 fi
 more /u/local/info/$ans.news
 echo "Hit enter to continue ... "
done

```

 `archie`  
parancs  
leírása a 217.  
oldalon  


Az állomány első sorában a `#!` megjegyzés után következik a parancsot futtató shell neve. Esetünkben — tekintve, hogy a `select` parancs nincs a Bourne-shellben — ez a Korn-shell. A második sorban beállítjuk a `select` parancs által használt `PS3` prompt értékét. A hirdetőtábla a `/u/local/info` könyvtár alkönyv-

táiraiban elhelyezkedő `.info` végződésű állományokból áll: egy-egy tétel címe megegyezik az állomány nevével. A `select` parancs argumentumában az `ls` és `sed` parancsok automatikusan felépítik a pillanatnyi listát (figyeljük meg a folytatósorok használatát a sorvégi `\` karakterrel), amiből aztán a felhasználó a kívánt sorszám beütésével választhat. Az `ans` változó a kiválasztott tétellel tér vissza, amit aztán a `do` ciklusban elhelyezkedő `more` utasítás jelenít meg. A ciklus az `ans` változó `exit` értékéig ismétlődik.

 A `for` ciklus használatára példa a következő parancsállomány:

```
#!/usr/bin/ksh
for File in ls *.c
do
 cp -i $File $File.old
 echo $File
done
```

A parancssorozat az adott könyvtárban található minden `.c` végződésű állományból másolatot készít, aminek neve az eredeti név `.old`-dal kiegészítve.

# 10. fejezet

## Állományok feldolgozása


A UNIX operációs rendszerben sok parancs az állományok feldolgozását: kifejezések megkeresését, gyors szerkesztését (pipe-ot használva), rendezését, ismétlődések kiszűrését teszi lehetővé. A továbbiakban ezeknek a parancsoknak a részletes ismertetésével foglalkozunk.

A UNIX operációs rendszerben a `grep` parancs szolgál arra, hogy állományokban megadott karaktersorozatokat keressünk. A `sed` parancs a kimenet szűrő-szintű szerkesztésére `ex-` ill. `vi-`szerű utasításokkal ad lehetőséget. A `uniq` parancs kiszűri az egy állományban található ismétlődő sorokat. A UNIX általános célú állomány feldolgozó programja az `awk`, amely megadott mintájú sorokon utasításokat hajt végre.

Az állományok bizonyos szempont szerinti sorrendbe tételére a `sort` parancs szolgál.

### 10.1. A `grep` parancs ismertetése

A `grep` parancs szolgál arra a UNIX operációs rendszerben, hogy állományokban megadott karaktersorozatokat keressünk. A `grep` családjába tartozik még az `egrep` és a `fgrep` parancs is. A karaktersorozat tartalmazhat általános helyettesítő karaktereket is (l. `ed`, `vi`, `sed` parancsok). A parancsnak a kapcsolókon kívül két argumentuma van: az első a keresett minta, a második a állományok listája, amiben keressük. Az utóbbi hiányában a standard bemenetről olvassa be az állományt (szűrőként is alkalmazhatjuk). Ha csak egy állományt adunk meg, akkor csak a mintát tartalmazó sorokat írja ki, különben eléjük rakja annak az állománynak nevét is, amiben a keresett minta előfordul.

 `grep`  
rövid leírása  
a 33. oldalon

A keresendő minta megadásánál vigyázni kell arra, hogy a shell által speciálisan kezelt karaktereket (`$`, `*`, `[`, `||`, `^`, `(`, `)`, `\`) levédjük. Ha a minta nem egy összefüggő karaktersorozat, akkor azt aposztrófok (vagy macskakörmök) közé kell tenni. A főbb helyettesítő karakterek a `$`, mely a sor végét, a `^`, mely a sor elejét, a `.`, mely egy tetszőleges karaktert, a `*`, mely az utolsónak megadott karakter tetszőleges számú megjelenését jelöli és a `[]` pár, mely azt határozza meg, hogy a karakternek a zárójelek közti értékek valamelyikét kell felvennie. Itt megengedett pl. az `[a-z]` használatával egy egész tartomány kijelölése is. A zárójelben egy kifejezés előtti `^` azt jelzi, hogy azt az értéket nem veheti fel a helyettesítő karakter. A `-e` kapcsoló használatával lehetőség van több minta megadására is (l. a 34. oldalon található leírást is!).

A `grep` parancs használatakor a következő korlátozásokat célszerű szem előtt tartani: a sorok maximálisan 2048 karakter hosszúak lehetnek, a hosszabb sorokat a `grep` betördeli. A paragrafusok (l. `-p` kapcsoló) maximális hossza 5000 karakter. A parancsot ne futassuk speciális állományokban a rendszergazda és saját békénk érdekében. A bemenő állomány nem tartalmazhat NULL karaktert, és új sor karakterre kell végződnie.

A `grep` parancs kapcsolói:

`-c`

Azt írja ki, hogy hány sort talált, mely tartalmazta a mintát.

`-E`

Kiterjesztett reguláris kifejezésként értelmezi mindegyik mintát<sup>1</sup>. A semmi minta (NULL) minden sorral egyezik.

`-e Minta`

Egy vagy több keresendő mintát lehet megadni. Igen hasznos, ha a *Minta* a `-` (mínusz) jellel kezdődik. A mintákat egymástól újsor karakterrel kell elválasztani. A mintákat a `grep` a `-E` és `-F` kapcsolók hiányában egyszerű reguláris kifejezésként értelmezi. Több megadott minta esetén minden olyan sort kiír, amelyik legalább az egyik mintát tartalmazza. A `-e` kapcsoló el is hagyható, de ilyenkor a *Mintának* a `grep` parancs első argumentumának kell lennie. A kapcsoló többször is kiadható, és ekkor a mintákat nem kell az újsor karakterrel elválasztani.

`-F`

Mindegyik megadott mintát szó szerint, és nem reguláris kifejezésként ér-

---

<sup>1</sup>Ha a kapcsoló egy adott rendszerben nem létezik, akkor a `grep` helyett az `egrep` parancsot használva ugyanezt a funkciót megkapjuk.

telmezi, ezért sokkal gyorsabban lefut<sup>2</sup>.

-f *Állomány*

Megadja a keresési mintákat tartalmazó állomány nevét. Az *Állományban* minden mintának új sorban kell állnia.

-h

Hatására több állomány esetén sem írja ki az állományneveket.

-i

Az összehasonlítás során nem tesz különbséget a kis- és nagybetűk között.

-l

Csak azon állományok nevét sorolja fel, amelyekben a megadott kifejezés megtalálható.

-n

Minden sor elé kiírja, hogy az állomány hányadik sorában található a megadott minta.

-p [*Elválasztókarakter*]

A mintát tartalmazó teljes paragrafust kiírja. A paragrafusokat egymástól az üres sorok választják el. Amennyiben megadjuk az *Elválasztókaraktert*, a `grep` annak megfelelően értelmezi a paragrafust.

-q

Nem ír ki semmit a kimenetre, pusztán nulla visszatérési értéket ad, ha a megadott minta létezik.

-s

Nem írja ki a nem létező, vagy nem olvasható állományokra vonatkozó hibüzeneteket.

-v

A megadott mintát nem tartalmazó sorokat írja ki.

-w


Egész szavakat keres.


-x

Azokat a sorokat írja ki, melyeknél a megadott minta a teljes sorral egyezik.

Példák (1. még a 33. és a 36. oldalon található példákat is):

A `proba` állományban keressük meg azokat a sorokat, melyek betűvel kezdődnek:

 további `grep` példák a 33. és a 36. oldalon

<sup>2</sup>Ha a kapcsoló egy adott rendszerben nem létezik, akkor a `grep` helyett az `fgrep` parancsot használva ugyanezt a funkciót megkapjuk. 

```
$ grep '^[a-zA-Z]' proba
```

A mintában szereplő `^` (sor eleje) jel mögötti kifejezés kiválaszt minden betűt, mely ABC sorrendben a és z illetve A és Z közé esik – azaz minden kis és nagy betűt.

A `proba` állományból írjunk ki minden olyan sort, mely vagy az `abc` vagy a `xyz` karaktereket tartalmazza:

```
$ grep -e 'abc' -e 'xyz' proba
```

vagy

```
$ grep 'abc'
```

```
> xyz' proba
```

vagy

```
$ grep -E 'abc|xyz' proba
```

Ez utóbbi esetben kiterjesztett reguláris kifejezést használtunk, melyben a szűrő jel `grep` VAGY műveletet jelent.

## 10.2. A `sed` pipe szerkesztő

A `sed` parancs szolgál a kimenet szűrőszintű szerkesztésére. Argumentumként megadható egy vagy több állománynév: ilyenkor ezeknek az állományoknak a tartalmát szerkeszti át a standard kimenetre. Állománynév hiányában a `sed` a standard bemenetet dolgozza fel.

A szerkesztés során lehetőség van kiválasztani azokat a sorokat, melyekre a szerkesztést el akarjuk végezni, és csak azokat módosítani. A `sed` nagyon hasonlít az `ed` szerkesztő családra (l. [vi](#)), és ennek megfelelően a parancsok általános alakjában meg lehet adni a tartományt (sorok), melyekre a parancs vonatkozik, egy parancsot, illetve az azt módosító kapcsolókat. A `sed` beolvassa a megadott állományokból (standard bementről) a sorokat a minta munkaterületre, végrehajtja azon a megadott parancsokat sorjában, és végül kiírja a standard kimenetre. Ezek után letörli a minta munkaterületet, és ugyanezt elvégzi a következő sorral. Néhány parancs egy másik munkaterületre (tároló) el tudja menteni a minta munkaterületet vagy annak egy részét, hogy a későbbiekben azt újra felhasználhassa.

Ha a `sed` parancs nem tartalmaz címtartományt, akkor minden sorra vonatkozik, különben csak a kijelölt sorokra. A cím lehet a sor száma, a `$` jel (a bemenet utolsó sorát jelöli), vagy egy reguláris kifejezés, mely kijelöli azt a sort, amellyel a mintája illeszkedik. A szokásos reguláris kifejezésektől (l. [vi](#)) a `sed` megenged

néhány eltérést.

A mintákat nem kötelező a / törtjellel elválasztani, bármely általunk kiválasztott (nem a szóköz vagy újsor) karakter lehet, de az első előfordulásakor fordított törtjellel kell beírni. Például

```
\RmintaR
```

esetén az `R` betű az elválasztó karakter. Ha maradunk a törtjelnél, akkor nem kell az első fordított törtjelet kiírni.

A `\n` a nem a sor végét jelölő újsor karaktert fejezi ki. A pont minden karaktert helyettesít, még a nem sorvégi újsor karaktert is, eltérően az `ed` (`vi`) szerkesztőtől.

A címtartomány állhat nulla, egy vagy két címből. Ha nincs megadva cím, a parancs minden sorra vonatkozik. Amennyiben egy szám (illetve a `$` jel), vagy egy reguláris kifejezés van megadva, az utána jövő parancs a megadott számú sorra, illetve a reguláris kifejezéssel megegyező sorra vonatkozik. Ha két számot adunk meg vesszővel elválasztva, akkor a két sorszám között minden sorra vonatkozik a parancs. Amennyiben a második szám kisebb vagy egyenlő az elsőnél, akkor csak az első számmal megegyező sorszámú soron végzi el a parancsot.

A `sed` a megadott parancsokat vagy a standard bementről veszi (első argumentum, vagy `-e` kapcsoló), vagy egy állományból (`-f` kapcsoló). Egyszerre összesen 99 parancsot lehet megadni.

A `sed` kapcsolói a következők:

`-e` *Utasítás*

A megadott *Utasítás* alapján dolgozza fel a szöveget. Amennyiben nem használjuk a `-f` kapcsolót, és csak egy utasítást adunk meg, nem kell kitenni a kapcsolót.

`-f` *Állomány*

Az utasításokat tartalmazó *Állomány* nevét adja meg.

`-n`

Nem írja ki a standard kimenetre a szerkesztés eredményét (csak az *Utasítás*-ban szereplő külön parancs esetén). Ezzel lehet elérni, hogy a `sed` ne írjon ki olyan sorokat, melyeket nem szerkesztünk.

### 10.2.1. A `sed` parancsai

A parancsok sokszor valamilyen szöveget tartalmaznak. Az `a` `c` vagy `i` parancsok esetén egy sornál hosszabb szöveget is beírhatunk, ha az utolsó sor kivételével mindegyik egy fordított törtjelre végződik. Mivel a parancsállományban a `sed`

figyelmen kívül hagyja a sor elején álló szóköz és tabulátor karaktereket, azokat a megadott szövegben a fordított törtjellel kell levédeni.

A parancsok előtt zárójelben szerepel, hogy hány címet fogad el az adott parancs:

(1) a **Szöveg**

A következő sor beolvasása előtt a *Szöveget* kiírja a kimenetre. Az a parancs után fordított törtjelet kell tenni, és a *Szöveget* új sorba írni.

(2) b [*Címke*]

A :*Címke*vel kezdődő sorra ugrik a parancsállományban. Ha nem adunk meg *Címket*, a parancsállomány végére ugrik.

(2) c **Szöveg**

Letörli a minta munkaterületet, és kiírja a *Szöveget* a kimenetre. Címtartomány megadása esetén csak a tartomány végén írja ki a *Szöveget*. Az a parancshoz hasonlóan itt is a c parancs után fordított törtjelet kell tenni, és a *Szöveget* új sorba írni.

(2) d

Letörli a minta munkaterületet.

(2) D

A minta munkaterület elejéről az első új sor karakterig töröl.

(2) g

A minta munkaterületet letörli, és a tároló munkaterületet tölti át.

(2) G

A tároló munkaterület tartalmát a minta munkaterület végére illeszti.

(2) h

A tároló munkaterületre bemásolja a minta munkaterület tartalmát.

(2) H

A minta munkaterület tartalmát a tároló munkaterület végére illeszti.

(1) i **Szöveg**

A *Szöveget* a kimenetre teszi, mielőtt a minta munkaterület tartalmát kiírná. Az a parancshoz hasonlóan itt is az i parancs után fordított törtjelet kell tenni, és a *Szöveget* új sorba írni.

(2) l

A minta munkaterületet kiírja a standard kimentre úgy, hogy a nem nyomtatható karaktereket 4 számjegyű hexadecimális formában írja ki. A hosszú sorokat betördeli (folytatósor esetén az előző sor végére egy fordított törtjelet tesz). Igen hasznos, ha olyan állományunk van, melyben hosszú (pél-



dául 2048 karakternél hosszabb) sorok vannak, mivel azokat nem minden szövegszerkesztő képes kezelni.

(2) n

A minta munkaterületet a standard kimenetre írja, és annak helyére beolvassa a következő sort.

(2) N

A feldolgozandó állomány következő sorát a minta munkaterülethez fűzi egy újsor karakterrel elválasztva, és megváltoztatja az aktuális sorszámot. Olyan minták keresésekor célszerű alkalmazni, melyek két (vagy több) egymás utáni sorban lehetnek.

(2) p

A minta munkaterületet a standard kimenetre írja.

(2) P

A minta munkaterület elejét az első újsor karakterig a standard kimenetre írja.

(1) q

A parancsállomány végére ugrik, de nem indít el új ciklust, hanem kilép.

(2) r *Állomány*

Beolvassa az *Állomány* tartalmát, és a standard kimenetre teszi a következő sor beolvasása előtt. Az *Állománynak* a parancssor végén kell állnia, és pontosan egy elválasztó karakter (szóköz vagy tabulátor) lehet előtte.

(2) s/*Minta/Helyettesítés/Kapcsolók*

A munkaterületen levő, a megadott *Mintával* egyező karaktersorozatot a *Helyettesítés* karaktersorozatra írja át. Ha nem adjuk meg a *g* kapcsolót, csak az első előforduló legnagyobb mintát cseréli le. A helyettesítés szintaxisa megegyezik az *ed* és *vi* szerkesztőknél leírtakkal. A következő kapcsolókat adhatjuk meg (egyszerre többet is):

*g* Az összes előforduló, a megadott mintának megfelelő karaktersorozatot lecseréli.

*n* Csak a mintának megfelelő *n*. karaktersorozatot cseréli le.

*p* Kírja a minta munkaterületet, ha sor került a helyettesítésre.

*w* *Állomány*

A minta munkaterületet az *Állományba* írja, ha sor került a helyettesítésre. A munkaterületet az *Állomány* végéhez fűzi hozzá (vagy ha az nem létezik, akkor létrehozza).

(2) t *Cimke*

A *:Cimke* sorra ugrik a parancsállományban, amennyiben a helyettesítő

parancs hatására változtatás történt a munkaterületen. Ha nem adunk meg *Cimkét*, a parancsállomány végére ugrik.

(2) *w* *Állomány*

A munkaterületet az *Állomány* végére fűzi. Az *Állomány* a parancssor végén kell, hogy álljon, és pontosan egy elválasztókarakter előzi meg.

(2) *x*

Kicseréli a minta és a tároló munkaterület tartalmát.

(2) *y*/*Minta1*/*Minta2*/

A *Minta1*-ben felsorolt karaktereket lecseréli a *Minta2*-ben felsoroltakkal. A két mintában a karakterek számának meg kell egyeznie, az újsor karaktert a `\n` jelöli.

(2) *!**sed*-parancs

A megadott *sed* parancsot azokra a sorokra alkalmazza, melyeket a címki-választás nem jelölt ki.

(0) *:**Cimke*

A *b* és *t* parancsok számára egy címket ad meg a parancsállományban. A *Cimke* legfeljebb 8 karakterből állhat.

(1) *=*

Kiírja az aktuális sor számát a standard kimentre.

(2) *{**Parancs1* *Parancs2* *...* *}*

A *sed* utasításokat csoportosítja. Mindegyik *Parancs*nak külön sorban kell állnia.

Az üres parancssorokkal a *sed* nem törődik. Megjegyzéseket a sor elején álló *#* jellel tehetünk a parancsállományban. Amennyiben az első sorban a *#* után közvetlenül egy *n* betű áll, az ugyanazt jelenti mintha kiadtuk volna a *-n* kapcsolót.


Példák:

Teljes csere végrehajtása: átírunk minden *alma* stringet *eper*-re a

 BemenoAllomány állományban. A kimenetet az UjAllományba írjuk:

```
$ sed "s/alma/eper/g" BemenoAllomány >UjAllomány
```

A *pr* parancs által előállított lap sorszámokat zárójelbe szeretnénk tenni, és

 úgy kinyomtatni a BemenoAllományt:

```
$ pr BemenoAllomány || sed "s/Page *([0-9]*)$/(\1)/" | lpr
```

Ez az utasítás a *Page* szót tartalmazó, és számra végződő sorokat cseréli le, úgy hogy a számot írja csak ki zárójelben (az `\1` jelöli az első, `( - \)` párral kijelölt résszel egyező mintát).

Ha csak a `keresem` szót tartalmazó sorokat szeretnénk megjeleníteni a `BemenoAllomany`ban, akkor a

```
$ sed -n "/keresem/p" BemenoAllomany
```

A `-n` kapcsoló miatt nem íródik ki semmi olyan a kimenetre, amire nem szólítottuk fel a `sed`-et. Jelen esetben ez a `p` parancs (az utasítás helyett a `grep` parancsot is használhattuk volna).

A parancsállomány használatára vegyük a következő példát: azt akarjuk, hogy a `sed` a legelső sor elé a `Kezdet` szót írja ki, az utolsó sor után pedig azt, hogy: `Veg`. Az összes fordított törtjelre végződő sorokat fűzze össze az utánuk jövővel a fordított törtjel nélkül. A `parancs.sed` állományunk például ilyen lehet:

```
1i\
Kezdet
:fuzes
/\$/{N
s/\\n//
b fuzes
}
$a\
Veg
```

Az `N` parancs fűzi össze a sorokat, az `s` parancs kiszűri az újsor karaktert, és a `b` parancs ismételteti meg az eljárást (enélkül csak két sort fűzne össze a `sed`).

Fontos, hogy az `N` parancs leállítja a `sed` parancsot, ha állomány vége jelet talál, és nem másolja át ekkor a minta munkaterületet a kimenetre. Ezáltal, ha egy állomány fordított törtjelre végződik, akkor a fenti parancsállomány nem írja ki az utolsó sort.

A parancsokat a `BemenoAllomany`ra a következő utasítás alkalmazza:

```
$ sed -f parancs.sed BemenoAllomany
```

### 10.3. A `uniq` utasítás

A `uniq` parancs kiszűri az egy állományban található egymás után ismétlődő sorokat, és csak egyszer írja ki az ismétlődő sort (célszerű lehet ezért az állományt először a `sort` utasítással rendezni). A parancsot olyankor használjuk, amikor például egy listában ki akarjuk szűrni az ismétléseket. A parancsnak a kapcsolókon kívül két argumentuma lehet, a bemeneti és a kimeneti állomány. Ezek hiányában a standard kimenetre illetve bemenetről dolgozik.

A parancs alapértelmezésben teljes sorokat hasonlít össze. Lehetőség van

arra is, hogy az összehasonlítás során ne vegyen figyelembe megadott mezőket (a mező egy olyan karaktersorozat, melyet egy vagy több elválasztókarakter — általában szóköz vagy tabulátor — választ el egy másik karaktersorozattól).

Az `uniq` parancs kapcsolói a következők:

- c A kimeneten megjelenő sorok elé beírja, hogy az adott sor hányszor fordult elő a bemenő állományban.
- d Csak az ismétlődő sorokat írja ki.
- u Csak az egyszer előforduló sorokat írja ki.
- f *Szám* Az összehasonlítás során a *Számnak* megfelelő számú mezőt figyelmen kívül hagy. A kapcsolót *-Szám* alakban is használhatjuk.
- s *Szám* Az összehasonlítás során a *Számnak* megfelelő számú karaktert hagy figyelmen kívül. Ha a *-f* kapcsolóval együtt használjuk, akkor a megadott számú mezőt hagyja először figyelmen kívül, majd az első, az összehasonlításnál használandó mezőből hagy figyelmen kívül *Szám* darab karaktert. A kapcsolót *+Szám* alakban is használhatjuk.

Vegyük például a `tagok` nevű állományt, amely a következő neveket tartalmazza:

```
joska
pista
joska
lajos
pista
joska
```



**A**

§ `uniq tagok osszes`

parancs létrehozza az `osszes` nevű állományban a

```
joska
pista
lajos
```

sorokat.

## 10.4. Az `awk` programnyelv

A UNIX általános célú állomány feldolgozó programja az `awk`. Működésének alapja, hogy megadott mintájú sorokat keres, és azokon az általunk megadott utasításokat hajta végre.

Az `awk` nyelv a legelső változat kialakulása óta bővült: némely UNIX rendszerben az újabb verziót `nawk`-nak (new `awk`) nevezik. Mi csak a program újabb változatával foglalkozunk, ezért — amennyiben az adott rendszerben létezik a `nawk` parancs — a példákban mindenhol azt kell használni az `awk` utasítás helyett.

A parancs első argumentuma az `awk`-nak szóló utasításokat tartalmazza, majd ezután következik az az állomány, amit az `awk` feldolgoz. Ha nem adunk meg állományt, akkor a standard bemenetről vár adatot. Több állomány esetén az `awk` úgy dolgozza fel őket, mintha a megadott sorrendben egymás után fűztük volna őket. Mivel általában hosszabb utasítássorozatot adunk meg, ezért lehetőség van az utasításokat tartalmazó állományt (parancsállomány) megadni az `awk`-nak a `-f Parancsállomány` alakban. A feldolgozott adatokat az `awk` alapértelmezésben a standard kimenetre írja.

A feldolgozás során az utasításokban megadott kiterjesztett reguláris kifejezések szerint elemzi az adatállomány aktuális sorát, és egyezés esetén végrehajtja a mintához tartozó parancsot. Az `awk` a parancsállomány összes sorát végignézi további egyezéseket keresve, és csak utána tér rá az adatállomány következő sorára.

Az `awk` megengedi változók, számokkal vagy karakterekkel operáló függvények és logikai kifejezések használatát.

Az `awk` interpreter szinten működik, azaz a parancsállományt nem kell lefordítani (mint általában a programozási nyelvek forráskódját), emiatt azonban lassabban futnak le az utasítások.

Az `awk` kapcsolói a következők:

`-f Állomány`

Az `awk` parancs az `Állomány` parancsállományból veszi az utasításokat. A kapcsoló ismételhető, és ilyenkor a megjelenés sorrendjében összefűzött állományt használja parancsállománynak az `awk`.

`-F KRK`

A `KRK`-t (kiterjesztett reguláris kifejezés, l. 10.4.3 fejezet a 118. oldalon) használja mező elválasztó karakternek.

`-v Hozzárendelés`

Értéket ad egy változónak. A `Hozzárendelés Név = Érték` alakú. A hoz-

zárendelést az `awk` még a `BEGIN` rész lefuttatása előtt elvégzi.

### 10.4.1. Az adatállomány feldolgozása

Az `awk` programnyelvben a `BEGIN` speciális parancs definiálja azokat az utasításokat, melyeket az adatállomány feldolgozása előtt kell végrehajtani (például számlálók nullázása, stb.) Ennek végrehajtása után az `awk` beolvas egy *rekordot* az állományból. A rekord az ún. rekord elválasztójellel szeparált adattömeg. Ez az elválasztójel alapesetben az újsor karakter, de az `RS` változó értékének változtatásával átállítható (pl. a `BEGIN` utasításnál).

A következő lépésben az `awk` a beolvasott rekordot összehasonlítja a parancsállományban lévő mintákkal azok megjelenési sorrendjében, és egyezés esetén végrehajta a hozzájuk tartozó parancsokat. A parancsban a rekordokban lévő mezőkkel is operálhatunk (alapesetben a szóközzel vagy tabulátorral elválasztott karaktorsorozatok). A mező elválasztókaraktert vagy az `awk -F` kapcsolójával, vagy az `FS` változó értékével lehet beállítani. A mezők értékeire a `$1`, `$2` ... kifejezésekkel hivatkozhatunk, melyek rendre az első, második, stb. mezőt jelentik. A `$0` az egész rekordot jelenti.

Miután az `awk` a rekordot az összes mintával összehasonlította és végrehajtotta az előírt parancsokat, beolvassa az adatállomány következő rekordját. Ez egészen addig folytatódik, amíg van bemenet. Az utolsó állomány utolsó sorának feldolgozása után az `awk` végrehajta az `END` speciális parancsban megadott utasításokat.

### 10.4.2. Az `awk` programnyelv

Az `awk` parancs általános szerkezete a következő :

```
Minta { Utasítás }
```

Ha a `Minta` megegyezik a rekorddal, akkor az `Utasítás` végrehajtódik. Ha nem adunk meg `Utasítást`, akkor a teljes rekord változtatás nélkül a standard kimenetre kerül. A `Minta` nélküli `Utasítás` minden rekordra végrehajtódik.

A `Minta` vagy egy kiterjesztett reguláris kifejezést, vagy egy összehasonlítást tartalmaz, de lehet ezek kombinációja, esetleg a `BEGIN` vagy `END` speciális szavak egyike is.

### 10.4.3. A kiterjesztett reguláris kifejezés

A kiterjesztett reguláris kifejezés a reguláris kifejezésekre (RK) épül (l. az `ed` és `vi` parancsokat). A keresett mintát törtjelek közé tesszük. A kiterjesztett RK a

következő speciális karaktereket használja:

- + Az előtte levő karakter egy vagy több előfordulását helyettesíti. Hasonlít a \*-hoz, mely nulla, vagy több előfordulást helyettesít.
- ? Az előtte álló karakter nulla vagy egy előfordulását helyettesíti.
- A pont az újsor karakter kivételével mindent karakterrel egyezik. A /a..at/ helyettesíti az `allat` kifejezést.
- \* Az előtte álló karakter nulla vagy több előfordulását helyettesíti.
- ~, ~! Az előtte megadott változóra vizsgálja meg, hogy az utána megadott RK-val egyezik-e. A  $\$1\sim/n/$  kifejezés minden olyan sorral egyezik, melyben az első mező tartalmazza az  $n$  betűt.
- ^ A mező vagy a rekord elejére hivatkozik. A  $\$2\sim/^h/$  kifejezés a második mező elején álló  $h$  betű esetén ad egyezést.
- \$ A mező vagy a rekord végére hivatkozik.
- || A szűrő jel két oldalán levő kifejezés bármelyikének egyezése esetén egyezést ír elő. Például a `/kutya||macska/` minden olyan sort kiválaszt, melyben vagy a `kutya` vagy a `macska` szó előfordul.
- ( ) Csoportosítja egy RK-n belül a karaktersorozatokat. A + vagy a ? kapcsolóval együtt szokás használni. Például a `/(zi)?(zi)?/` kifejezés az üres sorral (nulla előfordulása mindkét kifejezésnek), a `zi` és a `zizi` szavakkal egyezik meg.
- {m} Az előtte levő karakter pontosan  $m$  darab egymás utáni előfordulását helyettesíti. A `/l{2}/` kifejezés például az `allat` szó esetén igaz.
- {m, n} Az előtte levő karakternek  $m$  és  $n$  közötti számszor kell egymás után előfordulnia ( $m < n$ ).

[*Karaktersorozat*]

A *Karaktersorozat*ban előforduló bármelyik karakternek kell a vizsgált rekordban előfordulnia. Egész tartományok is megadhatók, például a `/ll[a-e]/` az `allat` és az `ellet` szavakkal is egyezik (a kifejezést a `/ll[abcde]/` alakban is megadhattuk volna). A kapcsos zárójelen belüli első helyen a `^` karakter azt jelenti, hogy az utána megadott karaktereknek nem szabad előfordulniuk. A `/ll[^b-e]/` kifejezés az `allattal` egyezik, de az `ellett`el már nem.

Ha a speciális karakterek valamelyikét szeretnénk expliciten megjelölni a kifejezésekben, azokat a fordított törtjellel kell levédeni. Például törtjelet a szövegben a következő kifejezéssel lehet keresni:

```
/szoveg\blabla/
```

A minta az összes `szoveg/blabla` kifejezést tartalmazó rekorddal megegyezik.

Az `awk` ismeri a C nyelvben használatos speciális szekvenciákat is, így a `\ddd` kifejezés megadásával tetszőleges karakter ASCII kódját is megadhatjuk a `ddd` oktális formában. A fordított törtjellel levédett speciális kifejezések közül az `awk` ismeri a macskaköröm, törtjel, a fordított törtjel (önmaga levédése), a `b` (karakter vissza), a `n` (újsor), a `r` (kocsi-vissza), `t` (tabulátor) és az `a` (csengő), `f` (lapdobás) és a `v` (vertikális tabulátor) vezérlő karaktereket.

#### 10.4.4. Az összehasonlítások

Az összehasonlítás során két kifejezést mérünk össze a `<`, `>`, `<=`, `>=`, `==` és `!=` jelekkel (sorrendben: kisebb, nagyobb, kisebb egyenlő, nagyobb egyenlő, egyenlő, nem egyenlő). A

```
$1 < $4
```

minden olyan rekorddal egyezik, melyben a negyedik mező nagyobb az elsőnél (vagy mint szám, vagy karakterek esetén az ABC sorrend alapján). Hasonló mó-

don megadhatjuk, hogy az első mező ne legyen `q` betű:

```
$1 != "q".
```

#### 10.4.5. A minták kombinációja

Ha két mintát vesszővel választunk el, akkor az `awk` több rekordot is kiválaszt a megfelelő utasításra: kiválasztja az adott tartományban azt a sort, amelyik az első mintával egyezik, majd utána keresi azt a sort, amelyik a második mintával



egyezik, és rájuk, valamint minden közöttük lévő rekordra is végrehajtja a parancsot.

Alkalmazhatunk logikai műveleteket a minták között a következő kifejezésekkel: a `||` (és), `&&` (vagy), `!` (nem). A

```
$1 == "igen" && $2 == "42"
```

minden olyan sort kiválaszt, melyben az első mező az `igen`, a második pedig `42`.

A mintákat zárójelekkel is csoportosíthatjuk.

#### 10.4.6. A BEGIN és END kifejezések

A `BEGIN`-nel felsorolt parancsok az első rekord beolvasása előtt, az `END`-del felsoroltak az utolsó rekord feldolgozása után hajtódnak végre. Ha a parancsállomány csak egy `BEGIN`-ből áll, akkor az hajtódik végre, és nem olvassa be a rekordokat. Ha csak egy `END`-ből, akkor beolvassa az összes rekordot, mielőtt végrehajtaná az `END` parancsokat.

#### 10.4.7. Változók

Az `awk`-ban használhatunk skalár, mező, tömb és speciális változókat. A változók neve nem kezdődhet számjeggyel. A használt változókat nem kell előre definiálni. Ha nem adunk kezdőértéket egy változónak, akkor numerikus változó esetén `0` lesz az értéke (akár skalár, akár tömb), karakterlánc esetén pedig a nulla karakterlánc. A változók a szöveggörnyezet alapján automatikusan numerikus, vagy string értéket kapnak. A stringek esetén az értéket macskakörmök közé kell tenni. Például az

```
x = "4" + "8"
```

kifejezésben két stringet adunk össze, de mivel az `awk` ezt számként tudja értelmezni, és így az `x` változóhoz `12`-t rendel. Ha egyszerűen azt akarjuk, hogy egymás után kerüljön a `4` és a `8` számjegy (`48`-at mutatva), akkor nem kell a plusz jel. Amennyiben a változó stringet tartalmaz, akkor numerikus kifejezésekben értéke `0` lesz.

Tömbökre a változó után írt szögletes zárójelekkel hivatkozhatunk, ahol a zárójelek között a tömb indexe áll. A tömbök méretét az `awk` automatikusan változtatja. A tömb indexe lehet szám is, de **string** is! (Ez utóbbi esetben persze kicsit nehézkes végigmenni a tömb elemein, de erre való az `awk` saját `for` Index in Tömb ciklusa.) Például a `bolha`, `elefánt` és `csiga` szavak előfordulását a be-menő állományban a következő parancsokkal számolhatjuk meg (mivel stringeket keresünk, ezért pl. a `csigabiga` kifejezés is a `csiga` csoportjába tartozik):

```

/bolha/ { x["bolha"]++ }
/elefant/ { x["elefant"]++ }
/csigá/ { x["csiga"]++}
END {print "bolha ", x["bolha"];
print "elefant ", x["elefant"];
print "csiga ", x["csiga"];}

```

Egy tömbnek több indexe is lehet, ezeket a szögletes zárójelen belül vesszővel kell elválasztani.

A speciális változók az `awk` működését befolyásolják. Itt csak a legfontosabakat soroljuk fel, bővebbet a

`$ man awk`

paranccsal tudhatunk meg.

FILENAME

Az aktuális feldolgozás alatt álló állomány teljes neve.

FNR

Az éppen feldolgozás alatt lévő rekordnak az aktuális állománybeli sorszáma.

FS

A beolvasott rekordokra vonatkozó mező elválasztókarakter, alapesetben a szóköz (ilyenkor a mezőket tetszőleges számú szóköz választhatja el). Az `FS` értéke lehet `KRK` (kiterjesztett reguláris kifejezés) is.

NF

Az aktuális rekordban levő mezők száma, ami legfeljebb 99 lehet.

NR

Az éppen feldolgozás alatt álló rekord sorszáma. Egy adatállomány esetén megegyezik az `FNR` változóval, de ez utóbbi minden új állomány esetén nullázódik.

OFS

A kiírás során a mezőket elválasztó karakter. Alapértéke a szóköz. Például

`a`

`$ awk '{OFS=":"; print $1,$2}' gyakorlas`

a `gyakorlas` állomány minden sorának első két szavát `:-`-al elválasztva írja ki.

ORS

A kiírás során a rekordokat elválasztó karakter. Alapértéke az újsor. Például

`a`

```
$ awk '{ORS="+"; print $0}' gyakorlas
```

a `gyakorlas` állomány sorait egyben, csak egy + jellel elválasztva írja ki.

RS


A bemeneti rekordokat elválasztó karakter. Alapértéke az újsor. Ha nullára állítjuk értékét, akkor a rekordokat egy vagy több üres sor választja el egymástól, és az újsor mindig mező elválasztó karakter lesz, függetlenül az FS értékétől.


### 10.4.8. Az `awk` utasításai

Az `awk`nak rengeteg utasítása van. Itt csak a legfontosabbakat soroljuk fel. Az utasítások kapcsos zárójelek között állnak. Több utasítást is fel lehet sorolni, egymástól pontosvesszővel elválasztva. Megjegyezzük, hogy a felhasználó maga is írhat saját függvényeket.

A következő matematikai operátorokat használhatjuk a kifejezésekben: +, -, /, ^, \* % (összeadás, kivonás, osztás, hatványozás, szorzás, és modulo). Például

```
a
$2=$1^$3
```

a második mezőhöz az első mező harmadik mezőbeli hatványát rendeli. A ++ és - a C nyelvbeli inkrementálást és dekrementálást jelöli (állhat mind a változó előtt, mind utána). Ugyanígy működnek a C-ben megszokott rövidítések is: +=, -=, /=, \*=. 

Karakterláncokat a legegyszerűbben úgy fűzünk össze, hogy egymás után írjuk őket. Például: 

```
$3 = $1 $2.
```

### Az `awk` függvényei

A beépített függvények között található a  $\cos(x)$ ,  $\sin(x)$ ,  $\text{atan2}(y,x)$ ,  $\exp(x)$ ,  $\log(x)$ ,  $\text{sqrt}(x)$ ,  $\text{int}(x)$ ,  $\text{rand}(x)$  (cosinus, sinus, arcus tangens  $y/x$  – radiánban -, e alapú hatványozás, természetes alapú logaritmus, négyzetgyök, egészrész, és egy 0 és 1 közötti véletlenszám). A karakterláncokkal a következő függvények operálnak:

```
sub(KRK,Hely,[Változó])
```

A *KRK*-ban definiált, a *Változó*ban előforduló első egyező karakterláncot a *Hely*ben definiált módon lecseréli. Ha nem adunk meg *Változót*, akkor az egész rekordot veszi. A függvény visszatérési értéke a helyettesítések

száma. Az & jel a *Hely* kifejezésben a *Változó* azon karakterláncát veszi fel, amelyekre a *KRK* minta illet.

`gsub (KRK, Hely, [Változó])`

Ugyanaz, mint a `sub` függvény, de a Kiterjesztett Reguláris Kifejezés (*KRK*) minden előfordulását cseréli.

`index (String1, String2 )`

Megmondja azt a pozíciót (1-el kezdve), ahol a *String2* kifejezés a *String1*-ben kezdődik. Ha a *String2* nem fordul elő a *String1*-ben, nullát ad.

`length (String) ]`

A *String1* karakterlánc hosszát adja meg.

`substr (String, M, [N] )`

A *String*ben megadott karakterláncból kivág az *M*. pozíciótól *N* darab karaktert (ha *N* nincs megadva, akkor a *String* végéig).

`tolower (String)`

Kisbetűkké konvertálja a *String* értékét, és azzal tér vissza.

`toupper (String)`

Nagybetűkké konvertálja a *String* értékét.

`sprintf (Formátum, Kifejezés1, Kifejezés2, ... )`

A C nyelv szabályai szerint a megadott *Formátumban* kiírja a felsorolt *Kifejezéseket*, és az így létrejött karakterlánccal tér vissza.

Az aritmetikai és karakterfüggvényeken kívül van még néhány általános függvény is. Az egyik leghasznosabb a `system (Parancs)`, mely egy shell *Parancsot* hajt végre, és annak visszatérési értékével tér vissza.

## Feltételek

A legtöbb `awk` feltételes kifejezés szintén a C nyelv alapján épül fel. A feltételeket a { } zárójelek segítségével csoportosíthatjuk. A feltétel és az annak teljesülése esetén végrehajtandó parancs külön sorban is lehet, de egy sorba is írható. Ha egy feltétel esetén több utasítást is kiadunk, azokat kapcsos zárójelek közé, pontosvesszővel elválasztva egy sorban vagy külön sorban kell beírni.

A C nyelv feltételes utasításai közül az `awk` a következőket ismeri:

`if (Kifejezés) { Utasítás } [ else Utasítás ]`

Feltételes utasításvégrehajtás a *Kifejezés* értékétől függően.

`while (Kifejezés) { Utasítás }`

Ciklus végrehajtása a *Kifejezés* értékétől függően.

```
for (Kifejezés1; Kifejezés2; Kifejezés3) { Utasítás }
```

Ciklus végrehajtása a *Kifejezésekben* megadott értékétartományra.

`break`

A `while` vagy `for` cikluson belül a ciklusból való kilépést jelenti.

`continue`

A `while` vagy `for` cikluson belül a ciklus következő tagjára való ugrást jelenti.

A C nyelvtől eltérő feltételek a következők:

```
for(Változó in Tömb) Utasítás
```

A *Változó* értékét a *Tömb* egymás utáni indexeinek felelteti meg, és végrehajtja az *Utasítást*.

```
delete Tömb[Kifejezés]
```

Kitörli a *Tömb Kifejezéssel* megadott indexű elemét. Ha *g* egy tömb, akkor a

```
for (i in g)
```

```
delete g[i];
```

kitörli a *g* tömb összes elemét.

`next`

Befejezi az aktuális rekord feldolgozását, és a következő rekord feldolgozását kezdi el az utasítássorozat elejéről.

```
exit [Kifejezés]
```

Végrehajtja az `ösze END` által definiált parancsot, majd befejezi az `awk` futását, és a *Kifejezésben* megadott értékkel tér vissza. Az `END` parancssorozaton belül kiadva nem hajtja végre a további `END` parancsokat, hanem kilép a programból.

`#`

Megjegyzést írhatunk mögé. Bárhol kezdődhet a sorban, és az újsor karakterig tart.

## Kiíratás

Az `awk`ban két parancs szolgál a kiíratásra, a `print` és a `printf`.

A `print` a mögötte felsorolt kifejezések értékeit írja ki a standard kimenetre. A kiírás során a szóközzel vagy tabulátorral elválasztott kifejezéseket közvetlenül egymás után írja. A vesszővel elválasztott kifejezéseket az `OFS` speciális változó

értéke választja el egymástól (ez alapesetben a szóköz), a sor végén pedig az `ORS` változó értékét (alapesetben újsor) írja ki a `print`. A kimenetet átírányíthatjuk a parancs végére írt `>`, `>>`, `||` szokásos átírányító jelekkel, és utána megadjuk az állomány nevét (ha magát az állománynevet adjuk meg, tegyük idézőjelbe, hogy a `print` karakterláncnak, és ne változónak gondolja), vagy a szűrő esetén a parancs nevét.

A `printf` parancs nagyon hasonlít a `printre`, de a kifejezések előtt egy, a kiírás formátumát előíró kifejezést vár. Ennek formátuma a C nyelvbeli szabályokkal adható meg. A formátum után állhatnak a kiírandó kifejezések és esetleges átírányítások.

### 10.4.9. Példák

✎ A `prog.f` állomány hetedik sorát a következő módon írathatjuk ki:

```
$ awk 'NR == 7 { print }' prog.f
```

A `gyakorlas` állományból az összes `Eb` mintát a második mezőben tartalmazó

✎ sort a következő paranccsal írathatjuk ki a sorszámmal együtt:

```
$ awk '$2 ~ /Eb/ { print NR, $0 }' gyakorlas
```

A 72 karakternél hosszabb sorokat (FORTRAN programozók, figyelem!) a

✎ következő módon szűrhetjük ki a `prog.f` állományból:

```
$ awk 'length($0) > 72' prog.f
```

Az utasítás minden 72 karakternél hosszabb sort kiválaszt, és — mivel nincs utasítás mellékelve — kiírja a standard kimenetre.

A következő példa minden sort kiír a `minta.txt` állományból a `start` és `stop`

✎ szavakat tartalmazó sorok között:

```
$ awk '/start/,/stop/' minta.txt
```

Tegyük fel, hogy a `berszam.fejtés` adatállományban a második oszlop a bruttó, az ötödik pedig a nettó bért tartalmazza. A következő paranccsal összegezzük a második és az ötödik oszlopban:

```
$ awk -f ossz.awk berszam.fejtés
```

Itt az `ossz.awk` parancsállomány például a következőket tartalmazhatja:

```
{
 sum2 += $2
 sum5 += $5
}
END {
```

## 10.5. ÁLLOMÁNYOK SORRENDBE RENDEZÉSE: A SORT UTASÍTÁS 127

```
print "Brutto ber: ", sum2, "Netto ber: ", sum5;
printf "Brutto atlag %f, netto atlag %f\n:", sum2/NR, sum5/NR;
}
```

A programállományban két változót használunk, és mivel nem adtunk nekik kezdőértéket, ezért azok nulláról indulnak. Nincs megadva sorkiválasztás, így a program minden sorra a `sum2` változóhoz a második, a `sum5` változóhoz az ötödik oszlop értékét adja hozzá. A legvégén (l. `END` parancs) az `awk` kiírja a második és ötödik oszlop összegeit, valamint az átlagértékeket is (az `NR` változó a beolvasott rekordok (sorok) számát tartalmazza).

A következő `awk` program az egymástól tetszőleges (de legalább egy) számú vesszővel, szóközzel vagy tabulátorral elválasztott mezők esetén fordított sorrendben kiírja az első két mezőt, majd az eredeti első mezőben összegez. Legvégül kiírja az összeget és az átlagot:

```
BEGIN {FS = "[, \t]"}
 {print $2, $1}
 {s += $1}
END {print "az osszeg ",s," , atlaga ", s/NR }
```

Ha a következő sort betesszük a `$HOME/.profile` állományunkba, akkor az itt létrehozott `REMOTE` változó értéke megadja annak a gépnek a nevét, ahonnan bejelentkeztünk:


```
export REMOTE=`who am i || awk '{gsub("[()]", "", $6); print $6}'`
```

## 10.5. Állományok sorrendbe rendezése: a `sort` utasítás

Az állományok bizonyos szempont szerinti sorrendbe tételére, illetve a már rendezett állományok helyes sorrendben való összefűzésére szolgál a `sort` parancs.

A parancs az argumentumában megadott állománynevet (ennek hiányában a standard inputot) rendezi, és az eredményt a standard kimenetre írja (kivéve ha a `-o` kapcsolóval megadjuk a kimeneti állomány nevét). Több bemeneti állománynevet is megadhatunk, ilyenkor a `sort` parancs összefűzi őket, és az így kapott állományra végzi el a rendezést. Az alapértelmezésbeli rendezés az (angol) ABC sorrend szerint történik.

A rendezés mezők és oszlopok kijelölésével történhet. A mező az elválasztókarakterek (ez általában a szóköz és a tabulátor) közötti karaktersorozat. Az elválasztókaraktert a `-t` kapcsolóval adhatjuk meg. Az oszlop a mezőn belüli karakterpozíció. A mező és/vagy az oszlop számát rendezési kulcsnak hívjuk.

 `sort`  
rövid leírása  
a 33. oldalon

A `sort` parancs a rendezést a megadott kulcsok sorrendjében végzi, azaz először az első kulcs szerint rendez, és a kulcson belül megegyező sorokat a második megadott kulcs alapján rendez tovább (stb.).

A `sort` parancs kapcsolói közül ha a `-b`, `-d`, `-f`, `-i`, `-n` és a `-r` kapcsolók egy rendezési kulcs előtt állnak, akkor módosítják annak hatását. Ezek a kapcsolók nem állhatnak magukban a `-k` kapcsoló után. Ha ezek a kapcsolók a (`-k`-nál szereplő) ún. *KulcsDefiníció*hoz vannak rendelve, úgy csak az adott rendezési kulcsra vonatkoznak. Amennyiben ezen kapcsolók egyike a `+MSzám.OSzám` vagy `-MSzám.OSzám` kulcsok valamelyike után fordul elő, csak arra a kulcsra fejt ki hatását.

-A

Az ASCII szekvencia szerint végzi el a rendezést (az alapértelmezést a helyi nyelvre is át lehet állítani).

-b

Figyelmen kívül hagyja a sor elején álló szóközöket vagy tabulátorokat az oszlopok kiszámolásánál.

-c

Lellenőrzi, hogy a megadott állomány a megadott szabályok szerint sorba van-e rendezve. Ha igen, nulla értékkel tér vissza.

-d

ABC sorrend szerint végzi a rendezést. Csak a betűk, számok és a szóközök számítanak.

-f

Rendezés előtt az összes kisbetűt nagyra cseréli.

-i

Figyelmen kívül hagyja az összes nem nyomtatható karaktert az összehasonlítás során.

-k*KulcsDefiníció*

Egy keresési kulcsot definiál, a következő formátumban:

*[KezdőMező][.KezdőOszlop][Módosító],[VégMező][.VégOszlop][Módosító]*

A szögletes zárójelek azt jelzik, hogy nem kötelező megadni a bennük levő számokat. A rendezés a *KezdőMező KezdőOszlopa* és a *VégMező Végoszlopa* közötti karakterek szerint történik. A *Módosító* a `b`, `d`, `f`, `i`, `n` és `r` értékek valamelyike lehet. Hatásuk megfelel az azonos betűvel jelölt kapcsolókénak. A fenti mennyiségek alapértékei: a *KezdőMező* a sor eleje, a *KezdőOszlop* az első oszlop a mezőben, a *VégMező* az utolsó mező, és a *VégOszlop* a mező utolsó oszlopa. A





## 10.5. ÁLLOMÁNYOK SORRENDBE RENDEZÉSE: A SORT UTASÍTÁS 129

-k 3.2,3r

kapcsoló olyan rendezést jelent, melyben a 3. mező 2. pozíciójától a 3. mező végéig (nincs kitéve a *VégOszlop*) folyik a rendezés, fordított sorrendben (l. -r kapcsoló). A fenti definíció megadható a -k kapcsoló használata nélkül is, ha pl. a *KezdőMező* elé plusz, a *VégMező* elé mínusz jelet írunk. Ilyenkor a + kapcsoló jelzi az átugrandó mezők számát, míg a - kapcsolóval adjuk meg azt a legutolsó mezőt, amelyik még számít. Az előző példa ezzel a módszerrel ezért ilyen:

+2.1 -3r

-m

Több, már rendezett állományt fűz össze egy rendezett állományba.

-n

A számokat tartalmazó mezőket a számértékük szerint rendezi. Mivel az ASCII sorrend szerint pl. a 10 előbb van, mint a 2 (a 2-es számjegyet megelőzi az 1), a numerikus kifejezéseket csak így lehet sorbarendezni. A numerikus mező tetszőleges számú szóközzel és tabulátorral kezdődhet, tartalmazhat előjelet, tizedes értékeket, az ezreseket elválasztó jelet. Minden más jel a felhasználó számára is érdekes eredményt szolgáltat.

-o *Állomány*

A rendezés eredményét a megadott *Állomány*ba irányítja. Megadhatjuk itt ugyanazt az *Állomány*nevet is, amelyiket éppen rendezünk.

-r

Megfordítja a rendezés sorrendjét.

-t *Karakter*

A megadott *Karakter*t tekinti a `sort` a mezőket elválasztó karakternek.

-u

Ha a rendezés során több `sort` egyenértékűnek talál a `sort`, akkor a -u kapcsoló hatására csak egyet ír ki.


-T *Könyvtár*

Az összes, a rendezés során használatos átmeneti állományt a megadott *Könyvtár*ba helyezi. Akkor célszerű használni, ha nagy állományt rendezünk, és a mi könyvtárunkban (file rendszer) kevés a hely.

Például rendezzük a következő `minta.txt` állományt különböző szempontok szerint:

```
+Kalmanka 16
lajoska 4
Peterke 28
```

```
Jozsika 12
lajoska 34
peterke 28
Sanyika 111
```

 A legegyszerűbb sorbatétel a következő:

```
$ sort minta.txt
+Kalmanka 16
Jozsika 12
Peterke 28
Sanyika 111
lajoska 34
lajoska 4
peterke 28
```

A rendezés először a jeleket, aztán a nagybetűket, utána a kisbetűket veszi ABC sorrendben. Ha lexikográfiai rendezést hajtunk végre a `-d` kapcsolóval, akkor az figyelmen kívül hagyja a jeleket, így `+Kalmanka` a `Jozsika` és `Sanyika` közé kerül. ASCII rendezés (`-A` kapcsoló) esetén a jelek a kis és nagybetűk közé kerülnek, és `+Kalmanka` `Sanyika` és `lajoska` közé kerül.

A következő parancs lexikografikus sorrendbe teszi az állományt úgy, hogy nem tesz különbséget kis- és nagybetű között (l. `-f` kapcsoló), és az ismétlődő sorokat csak egyszer írja ki (l. `-u` kapcsoló):

```
$ sort -d -f -u minta.txt
Jozsika 12
+Kalmanka 16
lajoska 34
lajoska 4
peterke 28
Sanyika 111
```

Most nem teszünk különbséget kis- és nagybetű között, ezért hiányzik a `Peterke` 28 sor is.

Rendezzük most az állományt a második mező (számok) szerint. A következő parancs ABC sorrend szerint rendez, azaz minden 1-gyel kezdődő megelőzi a 2-vel kezdődőket:


```
$ sort +1 minta.txt
Sanyika 111
Jozsika 12
+Kalmanka 16
Peterke 28
```

## 10.5. ÁLLOMÁNYOK SORRENDBE RENDEZÉSE: A SORT UTASÍTÁS 131

```
peterke 28
lajoska 34
lajoska 4
```

A következő parancs a számsorrend szerinti rendezéshez szolgál: 

```
$ sort +1 -n minta.txt
lajoska 4
Jozsika 12
+Kalmanka 16
peterke 28
Peterke 28
lajoska 34
Sanyika 111
```

Amikor a mezőket egymástól nem szóközők, hanem például kettőspontok választják el, mint pl. a `/etc/passwd` állományban, akkor meg kell adni az elválasztókaraktert. A rendszer összes felhasználójáról a 

```
$ sort -t: +4 /etc/passwd
```

paranccsal kapunk a nevek szerinti ABC sorrendben listát (ha a rendszergazda nem volt lusta, és beírta a neveket).



# 11. fejezet

## Archiválás és archív állományok használata

A UNIX használata során előbb-utóbb szükségünk lesz nem csak egy file, hanem egész könyvtárak ill. könyvtárrendszerek archiválására (elmentésére). Erre — több más, hasonló parancs mellett (pl. `cpio`, `backup/restore`) — a `tar` parancs biztosít kényelmes lehetőséget. Gyakran az állományok — különösen az archív állományok — sok, ismétlődő részt tartalmaznak, s ezért jól tömöríthetők: erre a `pack` vagy a `compress` utasítást használva kb. felére csökkenthető az állományok mérete. Ezek az utasítások minden UNIX rendszerben ugyanúgy működnek, ezért jól használhatók különböző típusú rendszerek közötti adatcserére is: pl. az egyik gépen létrehozott `tar` állományt egy másik típusú gép is szét tud csomagolni.

A programok mellett több, elsősorban DOS-os környezetből származó public domain archíváló program is megjelent a UNIX világban (pl. `arc`, `zoo`, `zip` stb.). Ezek használata azonban csak a DOS felé történő adatátvitelnél és csak előzetes kipróbálás után ajánlott: az ilyen programokkal szinte mindig verzióbeli (inkompatibilitási) problémák lépnek fel, gyakran a DOS-ban írt tömörített állományt a UNIX nem tudja olvasni és viszont. Jó megoldás lehet viszont a `tar` és a `compress` IBM PC-n való használata.

A különböző bináris állományokat (amik lehetnek pl. archív állományok vagy fordítóprogrammal és linkerrel létrehozott programok) elektronikus levélben csak ASCII formátumba való átalakítás után tudjuk átvinni. A két leggyakoribb ilyen kódolási eljárást a tradicionális `uuencode/uudecode`, illetve a hatékonyabb `atob/btoa` parancspárral használhatjuk.

A nagy állományokat az E-levélben történő továbbításhoz célszerű feldarabolni. Erre a `split` parancsot használjuk.

## 11.1. Könyvtárak archiválása

### 11.1.1. A `tar` parancs

A `tar` parancs könyvtárak ill. könyvtárrendszerek archiválására szolgál. Az utasítás archív állományként nemcsak a filerendszer egyik állományát tudja kezelni, hanem közvetlenül is írhatja/olvashatja pl. a külső szalagos eszközt (tape).

A `tar` parancs általános alakja a következő:

§ `tar [ Opciók ] -f ArchívÁllomány Állomány(ok)`

A `tar` parancsban az *Állomány(ok)* nevének mérete nem haladhatja meg a 100 karaktert.

Leggyakoribb opcióként a következő kapcsolók használatosak<sup>1</sup>:

- c Egy új *ArchívÁllományt* hoz létre a `-f` kapcsolóval megadott néven, majd beleírja a megadott *Állomány(ok)*at.
- r A megadott *ArchívÁllomány* végéhez írja az *Állomány(ok)*at. Ez az opció a szalagos egységnél nem használható.
- u A megadott *ArchívÁllomány* végéhez írja az *Állomány(ok)*at, amennyiben azok nem szerepelnek az *ArchívÁllományban*, vagy az archiválás óta megváltoztak. Az opció a szalagos egységnél nem használható.
- t Kilistázza az *ArchívÁllomány* tartalmát.
- v Kiírja az egyes állományok nevét, ahogy azokra sor kerül a feldolgozás során.
- x Kiolvassa az *ArchívÁllományból* a megadott állományokat. Ha az *Állomány(ok)*at nem adjuk meg, akkor az összes állományt kiolvassa az archív állományból. Ha a megadott állomány ugyanazon a néven többszöt is szerepel, akkor csak a legutolsó változatot olvassa el (l. a `-u` és `-r` kapcsolókat). Vigyázat! A helyettesítő karakterek nem a megszokott módon működnek (a 139. oldalon egy jól működő változat látható).
- f *ArchívÁllomány*  
Megadja a használandó *ArchívÁllomány* nevét. Amennyiben ennek a `-` (mínusz jel) értéket adjuk, a `tar` a standard inputról olvas, ill. a standard outputra ír.

---


<sup>1</sup>Megjegyzendő, hogy a kapcsolókat egyben szoktuk megadni, azaz `-xvf` és nem `-x -v -f`. Erra azért van szükség, mivel bizonyos UNIX rendszerek másképpen nem tudják helyesen értelmezni.

*-b Blokkméret*

A rekordok mértetét adja meg 512 byte méretű blokkokban. A *Blokkméret* alapértelmezésben 20, ami egyben a maximum is egyes UNIX rendszerekben. Ez az érték megfelel hálózaton keresztüli mentésre is, mivel a rekordok ilyenkor még nem nagyon szegmentálódnak a TCP/IP átvitelben. Ha nagyobb *Blokkméretet* választunk, az adathordozó több adatot tud tárolni. Amennyiben nem a 20-as értéket használjuk, erősen ajánlott ezt az adathordozón feltüntetni a későbbi frusztrációk elkerülésére.

- B Hálózati mentés vagy visszaállítás során használni kell ezt a kapcsolót. Hatására a `tar` addig vár, amíg a kommunikációs csatornában nem gyűlik össze egy rekordnyi adat.
- h Követi a szimbolikus linkeket, azaz a linkekkel megadott állományokat és könyvtárakat is elmenti. A `tar` alapértelmezésben csak a szimbolikus link információt írja be az archív állományba.
- p A kiolvasott állományokat az eredeti, az archív állományban tárolt védelmi kódokkal, felhasználói és csoport azonosítókkal látja el.
- C *Könyvtár* Amennyiben a felsorolt állományok előtt a `-C Könyvtár` kifejezés van, a `tar` egy `cd` utasítást hajt végre a *Könyvtár*-ban. A paranccsal csak abszolút könyvtárneveket adhatunk meg, és az elmentendő állományokat is egyenként kell felsorolnunk.
- d A `tar` a kapcsoló hatására a különleges (üres könyvtár, hardver file, FIFO, stb.) állományokat is helyesen kezeli el. Ezt az opciót a `tar` nem minden UNIX rendszerben ismeri.

Példákat a `tar` parancs használatáról a 11.1.4 fejezetben, a 138. oldalon találhatunk.

 `tar`  
példák a 138.  
oldalon

### 11.1.2. A `dd` adatkonvertáló utasítás

A `dd` parancs adatok formátumának átalakítására használatos a

`dd [Opció=Érték]`

alakban. Méret megadásakor használhatjuk az érték után a `b` jelzést, ami 512 byte-os egységet, és a `k` jelzést, ami 1024 byte-os egységet jelöl. A leggyakrabban használt *Opciók* a következők:

`if=Állománynév`

A bemeneti file neve. Alapértelmezésben ez a standard input.

## 136 FEJEZET 11. ARCHIVÁLÁS ÉS ARCHÍV ÁLLOMÁNYOK HASZNÁLATA

`of=Állománynév`

A kimeneti file neve. Alapértelmezésben ez a standard output.

`bs=Méret`

A blokkok mérete byte-ban. Mérete alapesetben gépfüggő, általában 512 byte, de van, amikor 1024 byte. Felülbírálja az `ibs` és `obs` kapcsolókat.

`ibs=Méret`

A bemeneti blokkok mérete byte-ban. Amennyiben a *Méret* után egy `b-t` írunk, akkor a `tar`-nál megismert 512 byte méretű egységeket használ (pl. `ibs=20b`).

`obs=Méret`

A kimeneti blokkok mérete byte-ban. Amennyiben a *Méret* után egy `b-t` írunk, akkor a `tar`-nál megismert 512 byte méretű egységeket használ (pl. `obs=20b`).

`skip=Szám`

A másolás megkezdése előtt a `dd` átugorja *Számmal* megadott blokkot.

`conv=ascii`

Az EBCDIC-ről ASCII-ra alakít.

`conv=ebcdic`

Az ASCII-ről EBCDIC-re alakít.

`conv=block`

A változó hosszúságú rekordokat fix hosszúságúvá alakítja.

`conv=unblock`

A fix hosszúságú rekordokat változó hosszúságúvá alakítja.

`conv=lcase`

A betűket kisbetűvé alakítja.

`conv=ucase`

A betűket nagybetűvé alakítja.

`cbs=Méret`

A konverziós blokkok méretét adja meg byte-ban. Csak a `conv` opcióval együtt használjuk. A *Mérettel* megadott számú karakter kerül egyszerre konverzióra. Az `ascii` és az `unblock` konverziónál a vezető szóközőket a `dd` így levágja, és egy újsor karakter kerül a sor végére. Az `ebcdic`, `ibm` és `block` átalakításnál a *Mérettel* megadott számú karakter kerül a bufferebe, majd a `dd` üres karakterekkel tölti fel a kimeneti blokkméretre a blokkot.

Például egy `gyakorlas` ASCII állományt a következő paranccsal konvertálhatunk át az `ejnye` EBCDIC állományba:



§ `dd if=gyakorlas of=ejnye conv=ebcdic`

Egy, a `/dev/rmt0h` eszközben lévő EBCDIC szalagot, amely tíz 80 byte méretű virtuális EBCDIC lyukkártyát tartalmaz rekordonként, a `ejnye` ASCII állományba

§ `dd if=/dev/rmt0h of=ejnye ibs=800 cbs=80 conv=ascii,lcase`

parancs alakít.

A `dd` parancs használatáról további példákat a 11.1.4 fejezetben, a 138. oldalon



`dd`  
példák a 138.  
oldalon

### 11.1.3. Mágnesszalagos egységek

A szalagos egységeket az `mt` parancson keresztül vezérelhetjük:

`mt [-f Eszköz] Parancs [Szám]`

Az *Eszköz*re a `/dev` könyvtárban található speciális állomány(ok)on keresztül hivatkozhatunk, melyek neve gépenként más és más lehet. Emellett a szalagos egység különböző üzemmódjaihoz (írásűréség, stb.) is különböző állományok tartoznak. Az éppen használható eszközökről a rendszergazda adhat felvilágosítást: ennek hiányában SUN rendszereknél a `/dev/rst0`, IBM AIX rendszereknél a `/dev/rmt0`, míg DEC Ultrix rendszereknél a `/dev/rmt0h` eszközöket próbálhatjuk meg használni. A felsorolt eszközök írás vagy olvasás után azonnal visszatekerik a szalagot, s ezért általában létezik ún. nem visszacsévéző (no rewind) változat is: ehhez az egység neve elé egy `n` betűt kell illeszteni. Például ha a `/dev/rmt0h` eszköz helyett a `/dev/nrmt0h` eszközre hivatkozunk, akkor a szalagos egység nem fog automatikusan visszacsévélni.

Amennyiben nem adunk meg eszközt a `-f` kapcsolóval, akkor az `mt` megpróbálja a `TAPE` környezeti változóban definiált eszközt használni.

A leggyakrabban használatos *Parancsok* a következők (a *Szám* alapértelmezésben egy):

`bsf`

A megadott *Szám*ú file vége (EOF, End of File) karaktert lép vissza a szalagon. A szalag az EOF jel szalagkezdeti oldalára áll rá.

`bsr`

A megadott *Szám*ú rekordot lép vissza.

`fsf`

A megadott *Számú* állományt lép előre a szalagon.

`fsr`

A megadott *Számú* rekordot lép előre.

`rewind`

Visszacsevéli a szalagot.

`retension`

Teljesen előretekeri a szalagot, majd visszacsévéli. Hatására a szalag feszültségmentes lesz.

`status`

Kijelzi a szalagos egység állapotát.

`eof`

Az adott szalagpozícióra állományvég (EOF, End of File) karaktert ír.

`erase`

Letörli a szalagot.

A következő példákban a `/dev/nrmt0h` mágnesszalagos eszközt kezeljük:



Kérdezzük le a szalagos egység állapotát:

```
$ mt -f /dev/nrmt0h status
```



A következő parancs visszatekeri a szalagot:

```
$ mt -f /dev/nrmt0h rewind
```



Három file-t a következő paranccsal léphetünk vissza:

```
$ mt -f /dev/nrmt0h bsf 3
```



Két EOF karaktert a

```
$ mt -f /dev/nrmt0h eof 2
```

utasítás ír a szalagra.

#### 11.1.4. Példák archiválásra

A példákban a `/dev/nrmt0h` mágnesszalagos eszközt használjuk.

Az aktuális könyvtárról és annak alkönyvtáiról a következő utasítással ké-



szíthetünk `tar` mentést:

```
$ tar -cvf /dev/rmt0
```

Az `allomany1` és `allomany2` állományokat a következő utasítással írhatjuk szalagra:

```
$ tar -cvf /dev/rmt0 allomany1 allomany2
```

Az összes állományt a szalagról a

```
$ tar -xvf /dev/rmt0
```

parancs állítja vissza. A visszaállítás a szalagon megadott névre történik, azaz pl. a `/tmp/test.1` nevű állományt a `/tmp/test.1` állományba, míg a `tmp/test.1` nevű állományt az aktuális könyvtár alatti `./tmp/test.1` állományba írja.

A `/usr/local/doc` könyvtárat a

```
$ tar -cvf /dev/rmt0 -C /usr/local doc
```

utasítással menthetjük el. Ennek hatására a szalagon keletkező állományok nevei a `/usr/local/doc/` helyett csak a `doc/` részt fogják tartalmazni.

Az összes állományt a szalagon a

```
$ tar -tvf /dev/rmt0
```

parancs listázza ki.

Az `innen` könyvtárat a következő parancs másolja át az `oda` könyvtárba az alkönyvtárakkal együtt

```
$ cd innen; tar -cvf - . | (cd oda; tar -xBpf -)
```

A `compress` utasítással (l. a 11.2.1 fejezetet a 140. oldalon) tömörített `test.tar.Z` állományt a

```
$ zcat test.tar.Z | tar -tvf -
```

vagy az ezzel egyenértékű

```
$ cat test.tar.Z | uncompress | tar -tvf -
```

parancs listázza ki.

Mivel a kifejezéseket a `tar` nem a megszokott formában kezeli, ezért pl. az összes, a `gon` mintát tartalmazó nevű állományt a

```
$ tar -xvf... /dev/rmt0 \ tar -tf /dev/rmt0 | grep 'gon'
```

utasítással állíthatunk vissza. Megjegyezzük, hogy a parancs futása meglehetősen lassú, valamint elég könnyen túlléphetjük a parancssor maximális hosszát (ami általában néhány ezer karakter).

Az `allomany1` és `allomany2` állományokat a következő utasítással írhatjuk szalagra 20 blokkos méretben a `dd` parancs használatával:

```
$ tar -cvf - allomany1 allomany2 | dd of=/dev/rmt0 obs=20b
```

Tegyük fel, hogy a szalagon egymás után két archívum van rögzítve. Az elolvasáshoz először tekerjük vissza a szalagot (figyeljük meg, hogy itt már min-

denhol a vissza nem csévéelő /dev/nrmt0 állományt használjuk):

```
§ mt -f /dev/nrmt0 rewind
```

Ezután olvassuk ki az első állományt:

```
§ tar -xvf /dev/nrmt0
```

A szalag az olvasás végén nem a következő állományon áll, hanem az EOF (állomány vége) jelen. Ez lépjük át:

```
§ mt -f /dev/nrmt0 fsf 1
```

Ezután kiolvashatjuk az következő állományt:

```
§ tar -xvf /dev/nrmt0
```

A hálózaton keresztül is használhatjuk a más gépen elhelyezett szalagos egységeket: ehhez a célgépre rsh utasítással be kell tudnunk lépni (ennek beállítását l. a rlogin és rsh parancsokat a 216. oldalon). Például az anglia távoli gépre a következő utasítás készít archív állományt:

```
§ tar -b 20 -cvf - * || rsh anglia dd of=/dev/rmt0 obs=20b
```

A távoli gépen elhelyezett archív állományt a rsh -n anglia dd if=/dev/rmt0 bs=20b || tar -xvBb 20 -f - parancssal szedhetjük szét.

## 11.2. Tömörítő programok

### 11.2.1. A compress eljárás

A compress parancs az ún. Lempel-Ziv eljárással tömöríti az állományt. Az eljárás az ismétlődő részeket rövidebb kifejezésekkel helyettesíti, s így tipikusan kb. felére csökkenti egy állomány méretét. Az eljárás dinamikusan működik, nincs szükség a teljes állomány ismeretére, ezért pipe-ban is használható. A tömörített állomány a .z végződést kapja meg.

A tömörített állományokat az uncompress parancs segítségével állíthatjuk vissza. Sikeres visszaállítás után az uncompress kitörli a tömörített file-t.


A zcat parancs a standard outputra írja ki az eredeti file tartalmát.

Például a gyakorlas.o állományt a következő paranccsal tömöríthetjük össze:


```
§ compress gyakorlas.o
```

A gyakorlas.o állomány visszaállítása a következő paranccsal történik:

```
§ uncompress gyakorlas.o.Z
```

Egy `tar` mentést a következő utasítással tömöríthetünk és írhatunk szalagra : 

```
$ tar -cvf - allomany1 allomany2 | compress | dd of=/dev/rmt0 obs=20b
```

Az elmentett archívumot a 

```
$ dd if=/dev/rmt0 ibs=20b | uncompress | tar -xvBf -
```

parancs állítja vissza.

### 11.2.2. A `pack` eljárás

A `pack` parancs az ún. Huffman eljárással tömöríti az állományt. Az eljárás a karaktereket gyakoriságuk alapján rövidebb bitmintákkal helyettesíti, s így kb. kétharmadára (esetleg felére) csökkenti egy állomány méretét. Az eljáráshoz a teljes állomány ismerete szükséges, azaz nem működik dinamikusan ezért pipe-ban nem használható. Az eljárás lassú, és csak ritka, különleges esetekben tömörít jobban a `compress`-nél. A tömörített állomány a `.z` végződést kapja meg.

A tömörített állományokat az `unpack` parancs segítségével állíthatjuk vissza. A sikeres visszaállítás után a tömörített file kitörlődik.

A `pcat` parancs a standard outputra írja ki az eredeti file tartalmát.

## 11.3. Kódolási eljárások

### 11.3.1. Az `uuencode/uudecode` parancs

Az `uuencode` parancs segítségével egy bináris állományt (pl. `tar` archívum, futtatható file, magyar ékezetes állomány) az ASCII karaktertartományba kódolhatunk át. Erre pl. levél küldésénél van szükségünk: a UNIX elektronikus leveleket továbbító programja csak 7 bitet (azaz az ASCII tartományt) viszi át (l. a 213. oldalt), s ezért bináris állományok közvetlen küldésére nem alkalmas.

A kódolási eljárás 3 byte-ot 4 ASCII karakterre (+ a kontroll információ) alakít, azaz kb. 35%-kal megnő a file mérete. Ennek ellenére lassú vonalakon érdemes egy szöveges állományt először tömöríteni (tipikusan 50%-ra), majd ezt az állományt `uuencode`-olni: a file mérete így kb. 30%-kal csökken.

A parancs általános alakja a következő:

```
uuencode [Állomány] TávoliÁllomány
```

Az utasítás a megadott *TávoliÁllomány* néven kódolja a bemeneti *Állományt*, ill.

annak hiányában a standard inputot. A végeredmény a standard outputon jelenik meg.

✎ Például a `gyakorlas.tar` állományt a

```
$ uuencode gyakorlas.tar gyakorlas.tar >gyakorlas.uue
```

utasítással kódolhatjuk.

A kódolt állományból a `uudecode` parancs állítja vissza az eredeti file-t a `uuencode`-ban megadott *TávoliÁllomány* néven.

✎ Az előző `gyakorlas.uue` állományból az eredetit a

```
$ uudecode gyakorlas.uue
```

utasítás bontja ki.

### 11.3.2. A `btoa/atob` eljárás

A `btoa` szűrőutasítás bináris állományt olvas be a standard inputon, majd ezt ASCII karakterekkel kódolva kiírja a standard outputon. A kódból a szintén szűrőként működő `atob` utasítás állítja vissza az eredeti állományt.

A `btoa` eljárás egy kompakt, 85-ös számrendszerű eljárást használ, így 4 byte-ból 5 byte lesz. Az eljárás gazdaságosabb a tradicionális `uuencode` módszernél, de nem minden UNIX rendszer része.

### 11.3.3. Állományok szétdarabolása

Néhány elektronikus hálózat (pl. a BITNET) a nagyon nagy (kb. 100 kbyte feletti) elektronikus leveleket nem képes továbbítani. Az Interneten ugyan nincs ilyen korlát, mégis érdemes a nagyon nagy (2-300 kbyte feletti) állományokat feldarabolni, ha azokat valamilyen okból levélben akarjuk elküldeni. Egy nagyobb állományt több kisebb file-ra szövegszerkesztővel is feldarabolhatunk, ettől a fáradságos (mi több, könnyen elhibázható) művelettől a `split` parancs kímél meg minket. A parancs alakja a következő:

```
split [-Szám] [Állomány [Prefix]]
```

Az utasítás a megadott *Állományt* (alapesetben a standard inputot) *Számnyi* (alapesetben 1000) sorral rendelkező állományokra bontja. Az állományok neve a *Prefix*-szel (alapesetben ez `x`) kezdődik, és végződésük először az `aa`, majd az `ab` stb. értékeket veszi fel, a legutolsó a `zz` lesz. Ily módon legfeljebb 676 állományra bonthatjuk a bemeneti *Állományt*.

A `split` soronként képes darabolni az állományokat. Bináris állományok szétbontásához használhatjuk a hasonló `bsplit` parancsot, amely — tekintve,

hogy közprogram — nem található meg minden UNIX rendszeren.

Például a `hosszufile` állományt a következő utasítással bonthatjuk fel `rovid` kezdetű, 100 sort tartalmazó darabokra:

```
$ split -100 hosszufile rovid
```

Az utasítás létrehozza a `rovidaa`, `rovidab`, stb. állományokat. Tegyük fel, hogy nincsen más `rovid` kezdetű állomány a könyvtárban. Ekkor a

```
$ cat rovid* > ujhosszufile
```

parancs visszaállítja a `hosszufile`-t `ujhosszufile` néven.





## 12. fejezet

# Programok fordítása és futtatása

A UNIX operációs rendszerben lehetőség van arra, hogy ne csak a UNIX előre definiált parancsait és parancsállományokat (script) futtassunk, hanem más magas szintű programnyelvben írt programokat is.

A UNIX, más nagygépes rendszerekhez hasonlóan a forráskódból első lépésben egy ún. *object* állományt hoz létre. Az *object* állományból az ún. *linkelés* során jön létre a futtatható állomány, amely már a CPU-nak szóló gépi kódú utasításokat tartalmazza. A fordítás ilyen módon történő két lépésre bontásával lehetővé válik, hogy több nyelvből származó forráskódot is egy futtatható állományba egyesítsünk, mivel az *object* állományok formátuma mindig azonos.

A szétválasztás másik következménye, hogy a különböző program-könyvtárakban tárolt rendszer és felhasználói függvények hozzáfűzése (linkelés) is elválik a forrásnyelvi fordítástól.

A UNIX rendszerben lehetőség van saját könyvtár létrehozására is az *object* állományokból.

A UNIX rendszer lehetőséget nyújt a futtatható állományoknál a hibakeresésre és nyomkövetésre (*debug*). A debugnak különböző szintjei léteznek a gépi kódtól egészen a forrásnyelvben való visszakeresésig. A UNIX alapértelmezésben betesz bizonyos alapvető debug információkat a futtatható állományokba. Amennyiben biztosak vagyunk, hogy a futtatható állományban levő debug információkra nincs szükségünk, akkor helytakarékoság és sebességnövelés céljából a `strip` parancssal azokat kitörölhetjük.

Általában minden UNIX alapértelmezésben tartalmazza a C fordítót, linkert és a loadert, valamint kiegészítésként beszerezhetők C++, FORTRAN, Pascal, COBOL, Ada és más magasszintű programnyelvek fordítói is. Megjegyzendő, hogy a GNU project keretében elérhető a teljes GNU fordítócsalád (C, C++, FOR-

TRAN, Pascal, Ada, debugger, stb.). Ezek a jó minőségű programok reális alternatívái a „gyári” fordítóknak, ráadásul térítésmentesen beszerezhetők.

## 12.1. A UNIX C Fordító (cc)

A UNIX operációs rendszer standard C fordítója `cc`. Használata:

`$ cc [Opció|Állomány]` A különböző UNIX rendszerek rendelkezhetnek még más C fordítóval is (pl. az IBM RS/6000 alap C fordítója az `xlc`), de a `cc` parancsot szinte mindegyik ismeri.

A parancsok feldolgozzák az assembler és az object állományokat is. Hacsak a `-c` opciót nem állítjuk be, a parancsok behívják a linkelő programot is. Az állománynevek kiterjesztésére a következő konvenció használatos:

- `.c` C forráskód
- `.i` preprocessált C forráskód
- `.o` object állomány (az `ld` parancs dolgozza fel)
- `.a` object könyvtár (az `ld` parancs dolgozza fel)
- `.s` assembler forráskód

A parancsok opciójaként a következők állhatnak:

Egy kapcsolós opciók:

- `-#`  
Részletes információt közöl a fordítás menetéről, de maga nem hív be semmilyen programot.
- `-c`  
Csak fordítást végez, a linkelő programot nem hívja be.
- `-DNév [= Definíció]`  
A *Név* szót úgy tekinti, mintha `#define`-ban szerepelne. Ha a *Definíció*t nem adjuk meg, az 1 értéket veszi fel.
- `-E`  
Csak preprocessálást végez el, fordítást nem. A kimenet a standard kimenet.
- `-g`  
Debug információkat is belefordít.

-IKönyvtár

A nem teljes könyvtárnévvel megadott include állományokat a *Könyvtárban* keresi.

-lObjectKönyvtár

A *libObjectKönyvtár.a* fordítási könyvtárállományt keresi.

-LKönyvtár

A `-l` opcióval megadott fordítási könyvtárállományok helyét adja meg.

-oNév

A fordítás eredményeként létrejövő (általában végrehajtható) állomány neve. Alapesetben a *Név* a.out, ha futtatható állomány a cél, az object állomány pedig a .o kiterjesztést veszi fel a .c helyett.

-O

Optimalizációt hajt végre.

-p

Egyszerű profile-t (futtatás közbeni információk gyűjtését lehetővé tevő kódot) generál a forráskódhoz.

-pg

Részletesebb profile-t generál a forráskódhoz.

-P

Csak preprocesszálást végez, fordítást nem. A kimenet egy .i kiterjesztésű állomány. (A kapcsolót bizonyos UNIX rendszerek nem ismerik).

-UNév

Megszünteti a név definícióját úgy, mintha az `#undef` direktívát adtuk volna meg a programban.

-v

Részletes információt közöl a fordítás menetéről.

-w

Nem ír ki információs és figyelmeztető üzeneteket.

-yKulcs

A lebegőpontos kifejezések futás közbeni kerekítését határozza meg. A *Kulcs* a következők valamelyike lehet:

n : a legközelebbi értékre kerekít

m : mínusz végtelen felé kerekít

p : plusz végtelen felé kerekít

z : a nulla felé kerekít

Az egyéb opciókat a következő módon adjuk meg:

`-qOpció`

ahol az *Opció* ki/be kapcsolóként van megadva. Például a `-qOpció` bekapcsolja az *Opció* opciót, míg a `-qnoOpció` kikapcsolja azt (pl. ha *Opció*=source, akkor a `-qsource` kapcsoló forráskódú listát készít, míg a `-qnosource` megakadályozza ilyen lista készítését). A következőkben felsorolt opciók felülbírálják az eredeti fordítási beállítást:

- `attr` A hivatkozott neveket felsorolja attribútumaikkal.
- `extchk` Ellenőrzi az eljárásoknak átadott értékek konzekvens használatát.
- `list` Object listát generál.
- `listopt` A listába beírja az összes opció beállított értékét.
- `source` Forráskódú listát generál.
- `stat` Idő- és mérettáblázatot is generál a forráskódú listába.
- `xref` Kereszt-referencia listát készít a hivatkozott nevekről.

## 12.2. A UNIX FORTRAN fordító (f77)

A UNIX operációs rendszerben a standard Fortran fordító az `f77`:

§ `f77` [ Opció ] [ Állomány ]

A különböző UNIX rendszerek rendelkezhetnek még más FORTRAN fordítóval is (pl. az IBM RS/6000 alap FORTRAN fordítója az `xlf`).

Az `f77` parancs a FORTRAN forráskódú állományokat fordítja le. FORTRAN programokon kívül assembler gépi kódú, valamint object állományokat is feldolgoz. Amennyiben nem adtuk meg a `-c` opciót, az `f77` parancs behívja a linkelő programot is, mely a kész futtatható állományt állítja elő. A parancs bemenő állománya(i) a következő típusú(ak) lehet(nek):

- `.f` kiterjesztésű FORTRAN forráskód
- `.o` kiterjesztésű object file az `ld` parancs számára
- `.s` kiterjesztésű gépi kódú forrásprogram

A leggyakrabban használt opciók nagy része (`# c g I l L o O v w y`) megegyezik a C fordítóéval. A különbségek:

- `-1` A `DO` ciklust legalább egyszer lefutattja.

- C Futás közben ellenőrzi a tömbökre alkalmazott indexeket (nem léptük-e túl a tömb definiált méretét), illetve a karakterláncokat.
- D Azokat a sorokat is lefordítja a FORTRAN forráskódú programból, melyek első sorában D betű áll.

A -q utáni opciók jó része is (*extchk*, *list*, *listopt*, *source*, *stat*, *xref*) megegyezik a C fordítónál leírtakkal. Az eltérések:

- dpc* A lebegőpontos állandókat dupla pontosan kezeli.
- extchk* Ellenőrzi az eljárásoknak átadott értékek, valamint a common blokkok konzekvens használatát.
- extname* Minden external neve elé egy aláhúzás karaktert tesz (kivéve a MAIN programok neve). Ez olyankor hasznos, amikor különböző nyelveken írt<sup>1</sup> programokat keverünk.
- noobj* Nem hozza létre az object állományt.
- recur* Lehetővé teszi rekurzív alprogramok írását.

## 12.3. A Pascal fordító

A UNIX operációs rendszerben a Pascal fordítónak nincsen általános neve (lehetőség pl. *pc*, *pascal*). Például az IBM RS6000 rendszereken a Pascal fordító formátuma:

```
xlp [Opció] [Állomány]
```

Általában hasonló a többi gépen is a parancs kiadása. A fordító pontos nevét a

```
$ apropos pascal
```

vagy a

```
$ man -k pascal
```

paranccsal nézhetjük meg (esetleg a rendszergazdát is megkérdezhetjük). A használható opciók nagyrészt megegyeznek a C és FORTRAN részben felsorolt opciókkal.

---

<sup>1</sup>C és FORTRAN programokat használók figyelmébe ajánljuk a *cfortran* programcsomagot, amely a két nyelv egyszerre való használatát jelentősen egyszerűsíti. A programcsomag pl. az *ftp.elte.hu* gépen *anonymous ftp*-vel (l. 217. oldal) található meg.

## 12.4. A UNIX fordítói könyvtárkezelése (ar, ranlib)

A UNIXban a fordítási (object) könyvtárak kezelésére az ar parancs szolgál:

§ ar [ -Kapcsoló(k) [ File ] ArchívÁllomány Állomány(ok)


Az ar parancs egy vagy több *Állományt* ír vagy töröl az *ArchívÁllományba* speciális formátumban. Az ún. *header*ket könnyen áthelyezhető módon tárolja, *object* könyvtár létrehozásakor, illetve felülírásakor mindig újraépíti a szimbólum-táblát. Az ld(linkelő) program ezeket az archív állományokat használja fel a végrehajtható program készítésére. Az archív állomány formátumát a /usr/include/ar.h állomány írja le. Az archív állományon belül névisméltés is előfordulhat, és ilyenkor számít a sorrend. Egyes gépeken figyelni kell, hogy az egymásra hivatkozó állományok az *object* könyvtáron belül megfelelő sorrendben legyenek. Az IBM AIX nem követeli meg az *object* könyvtárállomány ilyen szigorú felépítését. A parancs kapcsolói is változhatnak gépről gépre. Itt a IBM RS6000 AIX főbb kapcsolóit ismertetjük:

- a *File* A *File*-lal megjelölt állomány után helyezi be a megnevezett állományokat.
- b *File* A *File*-lal megjelölt állomány elé helyezi be a megnevezett állományokat.
  - c Nem írja ki az *object* könyvtár létrehozásakor szokásos üzenetet.
  - d Kitérli a megadott állományokat az *object* könyvtárból.
  - l Az átmeneti állományokat az aktuális könyvtárba helyezi, és nem a /tmp könyvtárba. (Akkor célszerű használni, ha a /tmp-ben kevés a hely).
  - o Egymás után rendezzi az állományokat, és kompresszálja az archív állományt. Az archív állományban tárolt UNIX állományok nem szükségszerűen vannak egymás után elhelyezve. Például, ha törlünk vagy átmozgatunk egy állományt, fel nem használt szabad hely keletkezhet az archív állományon belül. Ez az opció újraépíti az archív állományt úgy, hogy az állományokat egymás mögé helyezi, és az összes szabad helyet megszünteti.
  - p Kijírja a standard kimenetre a megnevezett állomány, vagy az összes (ha nem adunk meg állománynevet), az *ArchívÁllományban* található állomány tartalmát.
  - q A megadott állománynevet az *object* könyvtár végéhez csatolja. A pozíciókapcsolók hatását megszünteti. Ez az opció NEM ellenőrzi le, hogy az adott állománynév alatt van-e már valami a *object* könyvtárban, így ha kétszer adunk meg egy állománynevet, az kétszer íródik be az *object* könyvtárba.

- r A megadott állományt (ha létezik az object könyvtárban) az újjal helyettesíti. Mivel az állomány helye adott, ezért a pozíciókapcsolóknak nincs semmi hatása. Ha az u(update) kapcsolóval használjuk, az r csak azokat az állományokat helyettesíti az object könyvtárban, melyek az utolsó beírás óta megváltoztak.
- s Új szimbólum-tábla felépítésére készíti az ar programot, még az esetben is, ha nem változott közben az object könyvtár tartalma. A kapcsolót a strip parancs után használjuk a szimbólum-tábla újraépítésére.
- t Az object könyvtár tartalomjegyzékét írja ki a standard kimenetre. Amennyiben megadunk állományneveket, csak azok jelennek meg.
- u Csak azokat az állományokat másolja be, melyek a legutóbbi másolás óta változtak. (lásd az r kapcsolót).
- v Részletes, állományokra lebontott információt ad az új object könyvtár felépítéséről. A t kapcsolóval együtt használva az ls -l parancshoz hasonló listát ad. Az x kapcsolóval minden állomány elé kiírja a nevét, a h kapcsolóval pedig felsorolja az állományok nevét és módosítási idejét.
- w Az archív állomány szimbólum-tábláját írja ki. Minden szimbólum annak az állománynak a nevével íródik ki, amelyben megjelenik.
- x Kimásolja az object könyvtárból a megadott állományt az aktuális könyvtárba. Az új állományok az eredeti, az object könyvtárban maradt állomány nevét veszik fel. Ha nem adunk meg állománynevet, az x minden, az archívban levő állományt kimásol.

Amennyiben a megadott *ÁllományNév* nincs az object könyvtárban, az ar hozzáadja azt az object könyvtárhoz. Ez esetben a pozíciókapcsolókkal megadhatjuk az állomány helyét az object könyvtárban. Amennyiben nem intézkedünk a helyről, az új állomány automatikusan az object könyvtár végére kerül. Ha ugyanazt az állománynevet kétszer adjuk meg, az állomány kétszeresen kerülhet be az object könyvtárba.

Példák:


A libem.a állomány létrehozása a basic.o buffer.o char.o object modulokból: 

```

$ ar vq libem.a basic.o buffer.o char.o
ar: creating archive file libem.a
q - basic.o
q - buffer.o
q - char.o


```

Amennyiben a `libem.a` object könyvtár nem létezik, akkor az `ar` létrehozza, és elhelyezi benne a `basic.o`, `buffer.o` és `char.o` állományokat. Amennyiben a `libem.a` létezik, akkor ellenőrzés nélkül hozzáadja a végéhez a két állományt. Ezért a duplázások elkerülésére célszerű letörölni a régi object könyvtárat, ha teljesen újjáépítjük. A `v` kapcsoló hatására a folyamatról részletes tájékoztatást kapunk.

 A `libem.a` állomány tartalmát a következőképpen listázhatjuk ki:

```
$ ar vt libem.a
rw-r--r-- 100/100 8860 Sep 11 18:33 1991 basic.o
rw-r--r-- 100/100 9871 Sep 11 18:34 1991 buffer.o
rw-r--r-- 100/100 3568 Sep 11 18:34 1991 char.o
```

Az `ls -l`-hez hasonló, 'hosszú' listát kapunk. Az egyszerű felsoroláshoz hagyjuk el a `v` kapcsolót.

 A `libem.a` object könyvtárhoz adjuk hozzá a `crypt.o` állományt úgy, hogy a már a könyvtárban levő `buffer.o` elé pozicionáljuk:

```
$ ar vrb buffer.o libem.a crypt.o
a - crypt.o
$ ar vt libem.a
rw-r--r-- 100/100 8860 Sep 11 18:33 1991 basic.o
rw-r--r-- 100/100 2362 Sep 11 18:35 1991 crypt.o
rw-r--r-- 100/100 9871 Sep 11 18:34 1991 buffer.o
rw-r--r-- 100/100 3568 Sep 11 18:34 1991 char.o
```

Általában a UNIX rendszerekben az `ar` nem végzi el a szimbólum-tábla legyártását (az IBM RS6000 AIX rendszeren az `s` kapcsoló ezt elvégzi). Ilyenkor a `ranlib` parancs végzi el az archív object könyvtár "random" object könyvtárrá alakítását, így az előző példában a

```
$ ranlib libem.a
```

utasítással építhetjük újra a szimbólum-táblát.

## 12.5. A futtatható program összeszerkesztése: az `ld` parancs

A különböző object állományokat, object könyvtárakat és külső változókat az `ld` parancs ötvözi egy betölthető és végrehajtható állományba, miközben kiértékeli a



## 12.5. A FUTTATHATÓ PROGRAM ÖSSZESZERKESZTÉSE: AZ LD PARANCS153


különböző hivatkozásokat. Az alapértelmezésbeli kimenet vagy az `a.out` végrehajtható állomány, vagy az `ld` parancs egy másik hívásakor felhasználható, már előkészített állományt generál (pl. ha nincs még feloldva az összes hivatkozás).

Az `ld` parancs formátuma a következő:


§ `ld` [ Kapcsolók ] Állomány(ok)

A felsorolt *Állományok* (object, object könyvtár, import lista) az `ld` a felsorolás sorrendjében szerkeszti össze. (Némely UNIX rendszerben az object könyvtárakat értékeli ki legutoljára az `ld`).

A fordítási könyvtárak `.a`-ra végződő nevű archív állományok (l. az `ar` parancsot a 150. oldalon). A fordítási könyvtárakat megadhatjuk abszolút vagy relatív útvonallal is, pl. a `-lxyz` kapcsoló a `libxyz.a` fordítási könyvtárállományt specifikálja, mely vagy a `-L` kapcsolóval megadott könyvtárakban, vagy a standard könyvtárakban (`/lib` és `/usr/lib` könyvtárak) helyezkedik el.

 `ar`  
részletes  
leírása a 150.  
oldalon

Az `ld` parancs pontos működése, kapcsolói eltérhetnek a különböző UNIX rendszerekben, de használatuk általában elkerülhető a `cc` parancssal (l. a 155. oldalon). Ha mindenáron használni akarjuk a parancsot, előtte feltétlenül olvassuk el a manuált (`man ld`).

 `cc`  
részletes  
leírása a 155.  
oldalon

### Az `ld` parancs kapcsolói

A kapcsolók sorrendje általában nem befolyásolja az `ld` parancs működését, kivéve a `-f` és `-L`, melyek az `ld` számára előkészített állományokat jelölik meg. A `-f` kapcsolók a parancssorban való megjelenés sorrendjében értékelődnek ki. A `-L` kapcsolókkal jelölt állományokat szintén a megjelenés sorrendjében dolgozza fel az `ld`, de csak az összes object állomány kiértékelése után. Ha egymásnak ellentmondó kapcsolókat használunk, az `ld` az utolsó kapcsolót veszi figyelembe, az annak ellentmondó megelőzőeket figyelmen kívül hagyja.

A leggyakrabban használt kapcsolók a következők:

`-f File` A `.o` és `.a` végződésű állományok neveit az *File* állományból veszi.

A bemenő object állományokat, valamint a `-f` kapcsolóval kijelölt object állományokat az `ld` a parancssorban való megjelenés sorrendjében dolgozza fel. A kapcsoló ismételhető.

`-lKönyvtár` A `-L` kapcsolóval megadott, vagy a standard object könyvtárak valamelyikében levő `libKönyvtár.a` fordítási könyvtárat is belefűzi a kimenetbe. A kapcsoló utáni állományokat az összes a bemeneten lévő, valamint a `-f` kapcsolóval megadott állomány feldolgozása után értékeli ki. A kap-

csoló ismételhető, a megadott fordítási könyvtárakat a megjelenés sorrendjében keresi meg az `ld`.

Az IBM RS6000 AIX alatt az `ld` parancs az összes objectet és object könyvtárállományt azonos módon kezeli. Először beolvassa a szimbólum-táblát, és kiszedi a már definiált szimbólumokat, így nem szükséges a fordítási könyvtárállományok speciális rendezése, melyet némely szerkesztőprogram igényel. Viszont ezáltal a szimbólum első előfordulása a mérvadó, ami akár meg is előzheti az illető szimbólum definícióját.

-L*Könyvtár*

A `-l` kapcsolóval megadott fordítási könyvtárállományoknak a file-rendszerben elfoglalt helyét (*path*) adja meg. A kapcsoló ismételhető. A megadott *Könyvtárakat* az `ld` a kimeneti állományba is bejegyzí az esetleges dinamikus betöltés használatához.

-o *Név*

A kimeneti állományt a *Név* nevű állományba helyezi és nem az `a.out`-ba.

-r

Lehetővé teszi, hogy a `ld` létrehozza a kimeneti állományt még akkor is, ha abban nem definiált hivatkozások vannak. Általában az így létrehozott állományt az `ld` parancs egy későbbi hívásakor használjuk állományparaméterként.

-s

Kiszedi a szimbólum-táblát, a sorszám és pozícióinformációkat a kimeneti állományból. Ezzel helyet lehet megtakarítani, de lehetetlenné teszi a debugger használatát. A művelet utólag is elvégezhető a kimeneti állományon a `strip` paranccsal.

-v

Részletes információt ad a szerkesztés menetéről.

Példák:

 Több object állomány összeszerkesztése `a.out` végrehajtható programmá:

```
§ ld -T512 /lib/crt0.o pgm.o subs1.o subs2.o -lc
```

Az `ld` használatakor nekünk kell megmondani, hogyan helyezkedjenek el az egyes szegmensek. Az IBM RS6000 AIX rendszerben a `-T512` egy általánosan működő paraméter. Utána következik az object file-ok felsorolása: a `/lib/crt0.o` a UNIX rendszer része. A felhasználói object file-ok után a `-lc` kapcsoló a C nyelv object könyvtárát linkeli még a programhoz. Azaz itt a felhasználónak kell gondoskodnia az összes olyan object könyvtár, object modul felsorolásáról, melyet egy fordítóprogram automatikusan megcsinál.

A parancs kiadásának egyszerűbb módja a `cc` parancs használata, mely a szükséges kapcsolókat automatikusan beállítja:

```
$ cc pgm.o subs1.o subs2.o
```

Példa fordítási könyvtárállomány használatra:

```
$ cc -o pgm pgm.o subs1.o subs2.o mylib.a -ltools
```

A `pgm.o`, `subs1.o`, `subs2.o` object állományokat, a `mylib.a` object könyvtár rutinjait, és a `-ltools`-szal specifikált fordítási könyvtárbeli (ami a `/lib/libtools.a` vagy `/usr/lib/libtools.a` file lehet) rutinokat szerkeszti össze `pgm` nevű végrehajtható állománnyá.

## 12.6. Programok futtatása

A kész, végrehajtható programokat a UNIX operációs rendszerben a nevükre hivatkozva futtatjuk. A program könyvtárának vagy benn kell lennie a `PATH` környezetváltozóban, vagy teljes névvel kell rá hivatkozni. Az így indított program standard bemenet és kimenete a terminál, így azt másra a program futásának ideje alatt nem használhatjuk. A ki- és bemenet a shellnél ismertetett módon átirányítható.

A program futtatásának előbbi módját az előtérben való (vagy interaktív) futtatásnak nevezzük. Lehetőség van arra, hogy a programot a háttérben futtassuk; ilyenkor annak futása alatt a terminál használható másra is. A programokat tetzés szerint rakhatjuk át az előtérből a háttérbe és viszont. Az éppen futó interaktív programot a `[Ctrl]` és `[Z]` billentyűk egyidejű lenyomásával állíthatjuk meg. A prompt megjelenése után végrehajthatunk egyszerű parancsokat, majd az eredeti program futtatását folytathatjuk az előtérben, vagy a háttérben.

A háttérfuttatáshoz a program neve, és az összes átirányítás, kapcsoló és argumentum után egy `&` jelet kell tenni (l. a 60. oldalon). A rendszer kiírja az eljáráshoz tartozó azonosító számot (`PID`), mely alapján később hivatkozhatunk az eljárásra. Bourne shell esetén az így indított eljárások befejezik futásukat, ha a szülő processz megszűnik (pl. kilépünk a rendszerből). Ezt megakadályozza a parancssor elején álló `nohup` utasítás. A `nohup` használata szabályozható a C- és Korn-shellekben is, általában nélküle sem állnak le a háttérben indított eljárások.<sup>2</sup> A rendszerbe való újbóli belépés után a `nohup`pal elindított eljárásokat a `ps` paranccsal kereshetjük meg.

<sup>2</sup>Az eltérő beállítások miatt sok UNIX rendszerben szükséges a `nohup` használata Korn-shell esetén.

A programot a háttérbe a `bg`, az előtérbe az `fg` paranccsal tehetjük. Lehetőség van a parancsok prioritásának megváltoztatására, egy parancs egy adott időben történő indítására illetve egy parancs rendszeres futtatására, valamint a rendszer egyenletes terhelése érdekében a parancsok egymás utáni futtatására.

A program, hacsak nem fordítás és linkelés eredményeként létrejött linkelt program (pl. az `a.out` állomány), egy külön ún. *alshellben* (*subshell*) indul el. Ha egy script programot az adott shellben akarunk lefuttatni, akkor használjuk az `. ScriptÁllomány` parancsot.

### A program prioritásának állítása: a `nice` parancs.

A `nice` paranccsal egy *Parancs*ot az alapbeállításnál alacsonyabb prioritással futtathatunk:

```
$ nice [-Szám] Parancs
```

A *Szám* 1 és 19 közti érték, és a növekvő értékek csökkenő prioritásra utalnak. Az alapérték a 10.

A root jogokkal rendelkező felhasználók ezzel a paranccsal növelhetik egy processz prioritását, úgy, hogy negatív értéket adnak a *Számnak*.

A C shellben van a `nice` parancson kívül beépített `nice` is, melyek bizonyos esetekben másképpen viselkednek.

A `nice` magától nem futtat a háttérben. Így a

```
$ nice Program
```

a `Program` nevű állományt futtatja le 10-es prioritással az előtérben, azaz a terminál másra nem használható, míg a

```
$ nice Program &
```

a háttérben fut le alacsonyabb prioritással.

### Programok időzítése: az `at` parancs.

Egy utasítás későbbi időpontban való végrehajtására szolgál az `at` parancs. Formátuma:

```
$ at [Kapcsolók] Idő [Dátum] [Növekmény]
```

Az `at` parancs beolvassa a standard bemenetről a későbbi időpontban végrehajtandó parancsokat. A betáblázott parancsok standard kimenetre és hibaeszközeire érkező üzeneteit postázza, kivéve, ha átirányítottuk a kimenetet (pl. a `mail` utasítással levélbe). A standard hibaeszközeire kiírja az adott eljárás számát és az indítási időt.

Az `at` parancs az aktuális eljárás környezetét veszi át (kivéve a prioritást).

A rendszergazda szabályozhatja, hogy melyik felhasználónak legyen joga használni az `at` parancsot.

Az időzített parancsokat kilistázhatjuk a

`$ at -l`

paranccsal. Az `at` használatakor minden feladat kap egy azonosítót (ez betűkből és számokból áll, pl: `ctw.635677200.a`). Ezt is kiírja a lista. Az elküldött feladatot, még mielőtt elkezdene végrehajtódni, törölhetjük a

`$ at -r Feladat`

paranccsal, ahol a *Feladat* jelöli az azonosítót.

Az időt a következőképpen adhatjuk meg:

- *Szám*, melyet opcionálisan valamely kiegészítés követ. Az egy és kétjegyű számokat az `at` órának értelmezi, a négyjegyűeket óra-percnek. Például `0852` 8 óra 52 percet jelent.
- A `noon`, `midnight`, `now` speciális szavakat szintén időnek értelmezi az `at`.
- A dátumot megadhatjuk a hónap (angol) nevével és a nap számával (esetleg vesszővel utána az évet), vagy a hét napjával. Két különleges napot is megadhatunk: a `today`-t és a `tomorrow`-t.

Az opcionális *Növekmény* az idő kifejezésében a következő lehet:

- `+` jellel ellátott szám, melyet a következő szavak valamelyike követ: `minute[s]`, `hour[s]`, `day[s]`, `week[s]`, `month[s]`, `year[s]`.
- a `next` speciális szó, melyet az előbb felsorolt szavak követhetnek. Így a `next day` a megadott időpontot másnapra értelmezi.

A már ismertetett `l` és `d` kapcsolón kívül használható még a:

`-c` A `csh` paranccsal futtatja az eljárást.

`-k` A `ksh` paranccsal futtatja az eljárást.

`-s` A `bsh` paranccsal futtatja le az eljárást.

`-m` Postázza a felhasználónak a parancs sikeres végrehajtását.

Példa:

Az (aktuális könyvtárban lévő) `uuclean` parancs futtatása péntek délután 5 órakor:


`$ at 5 pm Friday uuclean`

vagy


```
$ at 5 pm Friday
```

```
uuclean
```

```
Ctrl-D
```

 A jövő héten ugyanebben az időben történő futtatás:

```
$ at now next week uuclean
```

 Két nap múlva ugyanebben az időben:

```
$ at now +2 days uuclean
```

Január 24-én délután 3 órákor:

```
$ at 3:00 pm Jan 24 uuclean
```

Egy betáblázott feladat betáblázhatja magát újra a következő napra, úgy hogy futása végén ismételten meghívja az `at` utasítást. Ilyen rendszeres futtatásokra szolgál a UNIX `crontab` (l. 12.6.2. fejezet) parancsa. A `crontab` használata rendszeres futtatásokra előnyösebb, mivel egy esetleges hiba esetén is újraindul (az `at` futtatása ilyenkor megszakad).

### 12.6.1. A `batch` parancs

A `batch` paranccsal indíthatjuk el a programot úgy, hogy az akkor kezdjen el futni, ha a rendszer nincs túlságosan leterhelve. A `batch` parancs az általa elindított program standard kimenetét és hibakimenetét postázza, hacsak azok nincsenek átirányítva. Kijírja a feladat számát és betáblázási idejét. A program az aktuális

 környezetből és könyvtárból indul:

```
$ batch <<"vege"
```

```
nagyfeladat 2>&1 >kimenet ||@ mail $LOGNAME
```

```
vege
```

Az így elküldött `nagyfeladat` program standard kimenete az aktuális könyvtár `kimenet` nevű állománya lesz, a hibüzeneteket pedig postázza a felhasználónak.

### 12.6.2. Feladatok rendszeres futtatása: a `crontab` parancs

Feladatok rendszeres futtatásokra a `crontab` parancs szolgál:

```
$ crontab [-l|-r|-v] [Állomány]
```

A `crontab` parancs bemásolja a megadott *Állományt*, vagy annak hiányában a standard bemenetet a `crontab` állományait tartalmazó könyvtárba. Az állandóan futó `cron` daemon ezen állományok utasításai szerint jár el. A megadott parancsok standard kimenetét illetve hibakimenetét a `crontab` postázza, ha csak nincs átirányítás. A `crontab` írásakor az összes előző beírásunk törlődik.

A `crontab` parancs használatát a rendszergazda szabályozhatja (engedélyezheti, illetve letilthatja).

A `crontab` bejegyzések 6 mezőből állnak. A mezőket szóköz vagy tabulátor választja el. A mezők tartalma sorrendben:

- Perc (0 és 59 közötti szám)
- Óra (0 és 23 közötti szám)
- A hónap napja (1 és 31 közötti szám)
- Az év hónapja (1 és 12 közötti szám)
- A hét napja (0 és 6 közötti szám: vasárnaptól szombatig)
- A futtatandó shell parancs

Az időpontra vonatkozó mezők tartalmazhatnak:

- Egy, az adott határok közötti számot.
- Kötéjjellel (-) elválasztott két számot, melyek egy tartományt határoznak meg.
- Vesszővel elválasztott számokat.
- A \* az összes megengedett értéket jelenti.

A napot két helyen is meghatározhatjuk. Így például a

```
0 0 1,15 * 1 parancs
```

bejegyzés minden hónap első és 15. napján fut le, valamint minden hétfőn.


A `cron` daemon a hatodik mezőben megadott parancsot futtatja le az adott időben. A hatodik mezőben esetlegesen megjelenő % jel utáni részt a `cron` a megadott parancs standard bemenetére teszi (kivéve a \% vagy "%" használatakor). A hatodik mezőben megjelenő nem védett felkiáltójel újsor karakternek számít. A shell csak a parancsmező első sorát futtatja le. Az összes többi az adott parancs standard bemenetére kerül.


A `cron` daemon a `HOME` könyvtárunkból indítja a subshellt. Nem futtatja le automatikusan a `.profile` állományt, így ha szükségünk van a benne foglaltakra, a felhasználónak kell erről a `crontab` bejegyzésben gondoskodnia. A `cron` daemon a következő környezetváltozókat adja át a subshellnek: `HOME`, `LOGNAME`, `SHELL` (alap esetben ez a `/bin/sh`), `PATH` (alap esetben ez a `:/bin:/usr/bin`).

Kapcsolói a következők:

- l Kilstázza a crontab állományt.
- r Törli a crontab állományt a crontab könyvtárból.
- v Részletesen kilistázza a crontab állományokat.

Példák:

-  Minden hétköznap du. 5-kor írjon ki nekünk egy üzenetet:  

```
0 17 * * 1-5 echo "Itt az ideje hazamenni!" | write $LOGNAME
```
-  Minden Karácsony előtt (dec 20 és 31 között 12 és 17 óra között óránként) köszöntsön minden, a gépen dolgozó embert:  

```
0 12-17 20-31 12 5 /etc/wall%Kellemes Unnepeket!
```

## 12.7. Hibakeresés és ellenőrzött futtatás (dbx)

A `dbx` egy szimbolikus debugger program. Alkalmazható C, Pascal, FORTRAN és COBOL programok elemzésére a következő módokon:

- Kiértékeli az object illetve a (programok váratlan leállásakor keletkező) ún. `core` állományokat.
- A program futtatásához egy felhasználó által ellenőrizhető környezetet teremt.
- Leállási pontok definiálhatók, vagy soronként futtatható a program.
- A szimbolikus változók értékeit név szerint ellenőrizhetjük.

Használata:

```
dbx [-a ProcessID] [-c ParancsFile] [-d KövetésiMélység]
 [-I Könyvtár] [Kapcsolók] [ObjectFile[CoreFile]]
```

Az *ObjectÁllomány* a fordító által generált (végrehajtható) állomány. A `dbx` program használatához a fordításnál használjuk a `-g` kapcsolót (szimbólum-tábla generálása). Amennyiben ezt nem tesszük, vagy a szimbólumtáblát a `strip` paranccsal megszüntettük, a `dbx` szimbolikus lehetőségei erősen lecsökkennek.

Induláskor a `dbx` a kezdeti parancsokat az aktuális könyvtár `.dbxinit` állományából, ennek hiányában a `HOME` könyvtár ilyen nevű állományából próbálja beolvasni.

Ha a program futása valamilyen hiba miatt megállt, az összes szükséges információt a rendszer a `core` nevű állományban tárolja. Amennyiben az adott könyvtárban van ilyen állomány, vagy a `dbx` hívásakor megadtuk a *CoreFile*t, akkor a `dbx` segítségével elemezhető a program állapota a hiba idején.



**A dbx kifejezései**

A dbx kifejezéseket C vagy Pascal (Modula-2) szabvány szerint kell megadni. A \* előtag vagy ^ utótag indirekt címzést jelent. Egy tömb részét megadhatjuk, ha az első és utolsó elemét két ponttal (..) választjuk el (pl. a(4..7)). A tömbök elemeire szögletes vagy kerek zárójelben megadott index szerint hivatkozhatunk. A pointerok és rekordok mezőire a ponttal (.) hivatkozunk, így a C nyelv -> operátorát nem szükséges használni (bár lehet).

A következő műveletek megengedettek:

**Algebrai:** +, -, \*, / (lebegőpontos), div (egész), mod, exp

**Bitművelet:** -, l, bitand, xor, ~, <{}<, >{}>

**Logikai:** or, and, not, ll, &&

**Összehasonlítás:** <, >, <=, >=, !=, = vagy ==

**Egyéb:** sizeof, :: (C++ scope művelet)

**A dbx kapcsolói**

-a *ProcessID*

A debug programot az éppen futó *ProcessID* azonosítójú eljáráshoz rendeli.

A debug program aktív lesz, mihamar az eljárás beindul.

-c *Állomány*

Lefuttatja az *Állomány*beli dbx parancsokat, mielőtt a standard bemenetről várna parancsot.

-d *KövetésiMélység*

Beállítja a program blokkjainak követési mélységét. Az alapérték 25.

-f

'Gyors' módban indítja el a dbx parancsot. Csak a szükséges szimbólumtáblát olvassa be induláskor, így sokkal gyorsabban indul be a dbx, valamint kevesebb memóriát használ.

-I *Könyvtár*

A keresési útvonalhoz hozzáadja a *Könyvtár*at. Az alapkeresés az aktuális könyvtár, valamint a végrehajtható állomány könyvtára. A debugon belül a use parancssal állíthatjuk be a keresési útvonalat.

-r

Azonnal elindítja a végrehajtható állományt. Ha sikeresen ér véget, a dbx program is befejezi futását. Ellenkező esetben a dbx program elindul, és közli a hiba okát.

A `-r` kapcsoló használatának kivételével a `dbx` felhasználói utasításra vár, és nem indítja el automatikusan a végrehajtható állományt.

### A `dbx` parancsai

A `dbx` elindulás után interaktív módon futtatható. A leggyakrabban használt utasítások a következők (a parancsok pontos szintaxisáról a `help` parancs ad felvilágosítást):

|                     |                    |                                                                                                                                                     |
|---------------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
|                     | /                  | Keresés a forráskódban előre.                                                                                                                       |
|                     | ?                  | Keresés a forráskódban hátrafelé.                                                                                                                   |
| <code>alias</code>  |                    | Helyettesítő név definiálása a <code>dbx</code> parancsokra.                                                                                        |
| <code>assign</code> |                    | Változók értékének beállítása.                                                                                                                      |
|                     | <code>call</code>  | Lefuttatja a program megnevezett eljárását.                                                                                                         |
|                     | <code>clear</code> | Az adott forráskódú sorra beállított <code>stop</code> utasítást törli.                                                                             |
|                     | <code>cont</code>  | Folytatja a program futtatását az utolsó leállási ponttól.                                                                                          |
| <code>delete</code> |                    | Nyomkövetéseket és leállási pontokat töröl.                                                                                                         |
| <code>detach</code> |                    | Kilép a debuggerből, és folytatja a program futását.                                                                                                |
|                     | <code>dump</code>  | Kiírja az adott eljárásbeli neveket és értékeiket.                                                                                                  |
|                     | <code>edit</code>  | Az aktuális állományra behívja a szövegszerkesztőt. Az alapbeállítás a <code>vi</code> , de az <code>EDITOR</code> környezetváltozóval beállítható. |
|                     | <code>file</code>  | Az aktuális forráskódot cseréli ki.                                                                                                                 |
|                     | <code>goto</code>  | A program futtatását a jelzett sorra teszi át.                                                                                                      |
|                     | <code>help</code>  | Információt ad a <code>dbx</code> parancsok használatáról.                                                                                          |
|                     | <code>list</code>  | Kilistázza az aktuális forráskód sorait.                                                                                                            |
|                     | <code>move</code>  | A forráskódú listában a jelzett sorra ugrik.                                                                                                        |
|                     | <code>next</code>  | A jelzett sorig futtatja a programot.                                                                                                               |
| <code>print</code>  |                    | Kiírja a kifejezés értékét, vagy eljárás esetén lefuttatja azt, és a visszatérési értéket adja meg.                                                 |
| <code>prompt</code> |                    | Beállítja a <code>dbx</code> promptját.                                                                                                             |
|                     | <code>quit</code>  | Befejezi a <code>dbx</code> futását.                                                                                                                |
|                     | <code>rerun</code> | Elindítja a program futását az előző argumentumokkal.                                                                                               |
| <code>return</code> |                    | Folytatja a program futását, amíg az vissza nem tér a jelzett eljárásba.                                                                            |
|                     | <code>run</code>   | Elindítja a program futását.                                                                                                                        |
|                     | <code>set</code>   | A <code>dbx</code> programváltozóinak ad értéket.                                                                                                   |
|                     | <code>sh</code>    | Lefuttat egy parancsot a shellben.                                                                                                                  |

- `skip` Folytatja a program futását az aktuális leállási ponttól és az esetlegesen megadott számú leállási pontot figyelmen kívül hagyja.
- `source` A jelzett állományból beolvassa a `dbx` parancsokat.
- `status` Megjeleníti az aktív nyomkövetéseket.
  - `step` Egy forráskódú sort futtat le.
  - `stop` Leállítja a program futását adott feltételek mellett (forráskód).
- `trace` Nyomkövetési információt ír ki az adott eljárásra, sorra, kifejezésre, változóra a jelzett feltételek mellett.
- `unalias` Megszünteti az alias-szal megadott helyettesítő nevet.
  - `unset` Törli egy `dbx` változó értékét.
  - `use` A forráskódú állományok keresési könyvtárát állítja be.
- `whatis` A programbeli változók, függvények, eljárások deklarációját mutatja meg.
  - `where` Az aktív eljárásokat és függvényeket listázza ki.
- `whereis` A megadott mintának megfelelő szimbólumok teljes leírását adja meg.
  - `which` Az adott azonosító teljes leírását adja.

A következőkben egy egyszerű példán bemutatjuk a `dbx` szimbolikus debugger programjának használatát.

Hozzuk létre a következő `test.c` állományt:



```
#include <stdlib.h>
#include <stdio.h>

main(argc, argv)
int argc;
char *argv[];
{
 int a, b, c;

 a=1;
 b=2;
 putchar(65);
 for(c=0; c<10; c++)
 {
 a=a*c+b;
 b=b*2;
 printf("a: %d b: %d c: %d", a, b, c);
 }
}
```

A fordítást és linkelést a `cc` paranccsal végezhethetjük el, a `-g` opciót használva (ekkor a fordítás és linkelés során a debug információk is bekerülnek a végrehajtható állományba):

```
$ cc -g test.c
```

Ezek után a létrejött `a.out` állományt indítsuk el a `dbx` debuggerrel:

```

$ dbx a.out
dbx version 3.1 for AIX.
Type 'help' for help.
reading symbolic information ...
(dbx) list Kilistázzuk a forráskódot.
 1 #include <stdlib.h>
 2 #include <stdio.h>
 3
 4 main(argc,argv)
 5
 6 int argc;
 7 char *argv[];
 8 {
 9
10 int a, b, c;
(dbx) list Kilistázzuk a következő 10 sort.
11
12 a=1;
13 b=2;
14 putchar(65);
15 for(c=0; c<10; c++)
16 {
17 a=a*c+b;
18 b=b*2;
19 printf("a: %d b: %d c: %d", a, b, c);
20 }
(dbx) stop at 13 Beállítunk egy töréspontot a 13. sorban.
[1] stop at 13
(dbx) run Elindítjuk a programot.
[1] stopped in main at line 13 A program a 13. sorban leáll.
13 b=2;
(dbx) step Előre lépünk egy utasítást.
stopped in main at line 14
14 putchar(65);
(dbx) step 3 Előre lépünk 3 utasítást.
stopped in main at line 18

```

```

18 b=b*2;
(dbx) print a, b, c Kiíratjuk az a, b és c változók értékét.
2 2 0
(dbx) trace b at 18 Beállítjuk a nyomkövetést a b változóra a 18. sorban.
[3] trace b at 18
(dbx) step Előre lépünk egy utasítást.
stopped in main at line 19
19 printf("a: %d b: %d c: %d", a, b, c);
(dbx) step Előre lépünk egy utasítást.
stopped in main at line 17
17 a=a*c+b;
(dbx) step Előre lépünk egy utasítást.
at line 18: b = 4 A nyomkövetés megtörtént.
[3] stopped in main at line 18
18 b=b*2;
(dbx) step 3 Előre lépünk 3 utasítást.
stopped in main at line 19
at line 18: b = 8
[3] stopped in main at line 18
18 b=b*2;
(dbx) step 2 Előre lépünk 2 utasítást.
stopped in main at line 17
17 a=a*c+b;
(dbx) print a Kiíratjuk az a változó értékét.
20
(dbx) assign a=1 Megváltoztatjuk az a változó értékét 1-re.
(dbx) print a Ellenőrzésképpen kiíratjuk újra az a értékét.
1
(dbx) step Előre lépünk egy utasítást.
at line 18: b = 16
[3] stopped in main at line 18
18 b=b*2;
(dbx) skip 1 Az [1]-es számú töréspontot átugorva futtatjuk a
programot.
at line 18: b = 64
at line 18: b = 128
at line 18: b = 256
at line 18: b = 512

```


```
at line 18: b = 1024
execution completed (exit code 10)
(dbx) quit Kilépünk a dbxből.
```

## 12.8. Programok újrafordítása: a `make` parancs

A `make` a sok fileből felépülő végrehajtható állomány előállítását segíti elő:

```
$ make [Kapcsolók] [-fMakeÁllomány] [CélÁllomány]
```

A `make` egy leíróállományt használ, mely tartalmazza, hogy egy állomány megváltoztatása milyen más állományokat érint, és hogyan kell egymásból előállítani őket. Azt, hogy egy állomány megváltozott, abból veszi észre a `make`, hogy az utolsó módosítási dátuma későbbi, mint azoké az állományoké, amelyek tőle függenek.

 Nézzük azt a példát, amikor van egy `pelda.c` programunk, melyet `pelda` végrehajtható programmá a `cc` utasítással fordítunk le. A `pelda.c` file `#include`-dal behív egy `pelda.h` állományt is. A `make` leíró állományában (ez a `makefile` vagy `Makefile` alapértelmezésben) ekkor a következő bejegyzés szerepel:

```
pelda: pelda.c pelda.h
 cc -o pelda pelda.c
```

A `make` parancs hatására a rendszer újrafordítja a `pelda.c` programot, ha az utolsó fordítás óta megváltoztattuk a `pelda.c` vagy a `pelda.h` állományokat, különben nem csinál semmit. Amennyiben egy állomány hiányzik (például a `pelda`), akkor azt mindennél régebbinek tekinti, így előállítja. Ha a leíró állomány neve nem `makefile` vagy `Makefile`, akkor a `-fÁllomány` kapcsolóval vehetjük rá a `make` parancsot, hogy az *Állomány*ből értékelje ki a feladatokat.

A `make` parancssorában az összes kapcsoló feltüntetése után megadhatunk makródefiníciókat is *MakróNév=Karakterlánc* alakban (l. 168. oldalon). A parancssorban az összes, a kapcsolókat követő szó, mely nem tartalmaz egyenlőségjelet *CélÁllománynév*, azaz azt mondja meg, hogy melyik állományokat óhajtjuk előállítani a leíró file-ból. A *CélÁllománynév* a kettőspontok előtti bejegyzés, azaz példánkban a `pelda`. A leíró állomány felsorolhat több program előállításához szükséges információt is, de ha megadjuk a *CélÁllomány* neveket, akkor csak a felsorolt állományokat gyártja le.

A bejegyzések általános alakja a következő:

```
cél [cél] [::] [Állomány] [`shell parancs `] [;Parancs] [Parancs]
```

A bejegyzés első sora tartalmazza a célállományok jegyzékét. Ezt egy kettősponttal zárja, és opcionálisan követheti azon állományok neve, melyektől ezek függenek. Megadható shell parancs is fordított aposztrófok között, melynek értéke adja meg az állományok nevét (pl. `\ls *.c \`). Amennyiben az állományok előállításához szükséges shell parancsokat teszünk ebbe a sorba, azokat pontosvesszővel kell bevezetni. Minden kettősponttal, vagy új sorban tabulátorral kezdődő parancs tartalmazza azokat a shell utasításokat, melyekkel a célállomány felépíthető.

Megadhatunk több parancssorozatot is a *CélÁllományra* vonatkozóan. Ilyenkor több bejegyzést kell írni, melyeknél a *CélÁllományt* két kettőspont követi. Utána ugyanúgy kell felsorolni az állományok nevét, melytől a *CélÁllomány* függ, valamint a szükséges parancsokat.

A # után a sor végéig következő minden karakter megjegyzésnek számít. Megjegyzések bárhol elhelyezhetők a leíró állományban. Az üres sorokat a `make` figyelmen kívül hagyja, viszont egy listán belül nem szabad üres sort hagyni.

A nem # karakterrel vagy tabulátorral kezdődő új sor új bejegyzést vagy makródefiníciót jelent. A parancssorokat egyenként, külön subshellben futtatja le a `make`, így néhány shell parancs (pl. `a cd`) hatása nem érvényesül, ha közben új sort kezdtünk. Viszont használható a `\` folytatósor jelölésére.

### 12.8.1. Kiterjesztések

A `make` parancs a szabványos állományok felépítésére nézve rendelkezik bizonyos alapszabályokkal. Ezeket a szabványos állománykiterjesztések alapján alkalmazza. A szabályok definiálják a kiindulási és a kész állomány típusát. Így például egy `.o` állomány esetén feltételezi, hogy létezik egy hozzá tartozó `.c` állomány, és tudja, hogy azt a `cc -c` paranccsal kell lefordítani.

A csak egy kiterjesztést tartalmazó (azaz `.c:`) szabály azt írja elő, hogy a programot az összes forráskód alapján kell felépíteni. Szabályokat a leíró állományban is definiálhatjuk. A `make` parancs minden ponttal (`.`) kezdődő, és `/` jelet nem tartalmazó *Célt* szabálynak tekint.

A `make` parancs futtatása során több szinten is befolyásolhatjuk annak működését. A parancs (emelkedő) sorrendben a következő hierarchia szerint dolgozza fel a kapcsolókat és makrókat:

1. a `make` saját alapszabályai
2. a környezetváltozók (`MAKEFLAGS` és `MFLAGS`)

3. a leíró állomány
4. a parancssor

### 12.8.2. Makrók

A *Karakterlánc1* = *Karakterlánc2* típusú bejegyzések a leíró állományban makróknak számítanak. A *Karakterlánc2*-t a sor végéig, illetve az első # megjelenéséig értelmezi. Lehet folytatósort használni. A definíció után a `make` a leíró állományban a `$(Karakterlánc1)` minden előfordulását *Karakterlánc2*-re cseréli. Az eljárás rekurzívan is alkalmazható, a `$( $(Karakterlánc1) )` vagy `$( $( $(Karakterlánc1) ) )` belülről kifelé helyettesítődik. Legfeljebb 10 szint mélységig használható az eljárás. A makró nevét nem kell zárójelbe tenni, ha az csak egy karakterből áll. Mivel a `make` a `$`-t használja makróhelyettesítésre, ezért ezt lehetőleg állománynevekben és a leíró állománybeli parancsokban ne használjuk. Öt előre definiált makró áll a rendelkezésünkre:

- `$*` A forrásállomány neve (kiterjesztés nélkül).
- `$$$` Az aktuális célállomány teljes neve.
- `$<` A nem időszerű modul forrásállománya. A `.c.o: cc -c $<` bejegyzésben a `$<` bármely nem időszerű `.o` állomány esetén a hozzá tartozó `.c` állományra utal.
- `$?` A nem időszerű állományok listája. A `make` akkor értékeli ki, amikor a *MakeÁllományból* megkeresi a konkrét szabályokat.
- `$%` Az archív állomány tagjának neve. Csak a `lib(Állomány.o)` formájú célállomány esetén értékeli ki a `make`. Ebben az esetben a `$$$` magára az archív állományra, a `$%` pedig az *Állomány.o* tagra mutat.

A `D` az állománynév útvonalára, az `F` a név részére utal az összes belső makróban a `$?`-t leszámítva. Így a `$(@D)` a `$$$` állomány könyvtárára utal. A `$` jelet `$$` módon adhatjuk meg a leíró állományban.

Ha a célállomány neve zárójelet tartalmaz, a `make` fordítási könyvtárállományként kezeli azt. A zárójelek közti rész a könyvtárállomány megfelelő tagja. Ez lehet például `library(Állomány.o)` alakú, de ha pl. definiáltunk egy `LIB` makrót, akkor `$(LIB)(Állomány.o)` alakú is lehet. A célállományokat mindig az archív állomány nevével kell megadni. A `$(LIB) + (tag1.o tag2.o)` kifejezést a `make` NEM tudja értelmezni.



Az archív könyvtárakra vonatkozó szabályok `.x.a` formájúak, ahol a `.x` azon állomány kiterjesztése, melyet a könyvtárhoz akarunk adni. Így a `.c.a` szabály azt jelzi, hogy C forráskódú állományból hogyan kell felépíteni a könyvtárat. Például a

```
lib: lib(tag1.o) lib(tag2.o) lib(tag3.o)
 @echo A lib keszen van
```

leíró állománybeli bejegyzéshez tartozó `make` alapszabály a


```
.c.a:
 $(CC) -c $(CFLAGS) $<
 ar rv $@@ $*.o
 rm -f $*.o
```

(a `CC` makró az alap C fordítót tartalmazza, a `CFLAGS` makró pedig a fordítás során használandó kapcsolókat. A `CC` definiálva van a UNIX rendszerekben, és átírható más fordítóra is).

### 12.8.3. A `make` kapcsolói

- d Debug mód: részletes információt közöl, hogy a `make` mely állományokat mennyi ideig dolgozta fel.
- e A környezetváltozók használatára kényszeríti a `make`-et, mégha a leíró állományban újra is lenne definiálva.
- f *MakeÁllomány* Beolvassa a *MakeÁllományt*, mely tartalmazza a célállomány felépítésének módját. Ha csak - (mínusz) jelet adunk meg *MakeÁllománynak*, akkor azt a standard bemenetről veszi. Ha nem használjuk a `-f` kapcsolót, akkor az aktuális könyvtárban sorban végigkeresi a `makefile`, `Makefile`, `s.makefile` és `s.Makefile` állományokat. A kapcsoló ismételhető, így egyszerre több leíró állomány is megadható.
- i Figyelmen kívül hagyja a parancsok által visszaadott hibakódokat. Normális esetben hiba esetén a `make` befejezi futását.
- k Befejezi az aktuális célállomány feldolgozását hiba esetén, de folytatja a tőle nem függő egyéb állományok feldolgozását.
- n Kijelzi, hogy mely parancsokat hajtaná végre a `make`, de nem futtatja le őket. A `@@`-cal kezdődő sorokat kiírja. Amennyiben a leíró állomány tartalmazza a `$(MAKE)` szimbólumot, meghívja újra a `make` parancsot. Ezzel a kapcsolóval ellenőrizhetjük le a leíró állományt.
- p A parancsok lefuttatása előtt kiírja az összes makródefiníciót, és cél leírást.
- q Nulla értékkel tér vissza, ha a célállományt nem kell újrafordítani, nem nullával, ha szükséges.

- s Nem jeleníti meg a végrehajtandó parancsokat.
- t A felsorolt parancsok lefuttatása helyett pusztán átírja az állományok keletkezési idejét. Akkor használjuk, ha a forráskódban csak lényegtelen változtatást hajtottunk végre. A parancs az összes a leíró állományban, illetve a parancssorban megjelenő célállomány idejét átírja.


 **Példa :** # A prog nevű állomány legyártása 3 object állományból

```
prog: x.o y.o z.o
A prog-ot a cc paranccsal kell lefordítani
 cc x.o y.o z.o -o prog
Az x.o-t két másik állományból (x.c és defs) kell legyártani
x.o: x.c defs
Az x.o-t a cc-vel allitjuk elő
 cc -c x.c
Az y.o-t két másik állományból kell legyártani
y.o: y.c defs
Az y.o-t a cc-vel allitjuk elő
 cc -c y.c
Az z.o-t két másik állományból kell legyártani
z.o: z.c defs
Az z.o-t a cc-vel allitjuk elő
 cc -c z.c
```

A beépített szabályok felhasználásával egyszerűsíthetünk ezen a leíró állományon:

```
A prog nevű állomány legyártása 3 object állományból
prog: x.o y.o z.o
A prog-ot a cc paranccsal kell lefordítani
 cc x.o y.o z.o -o prog
Az objecteket a forráskódból és a defs állományból allitjuk elő
x.o y.o z.o: defs
```

A `make` ismeri a következő kiterjesztéseket: `.a` (archív állomány), `.c` (C forrásnyelv), `.f` (FORTRAN forrásnyelv), `.h` (C header állomány), `.l` (Lex állomány), `.o` (object állomány), `.s` (assembler forrásnyelv) forráskód, `.sh` (shell parancsállomány), `.y` (Yacc állomány). Ismeri az SCCS állományokat is, ezekre az előző kiterjesztés utáni `~-`vel kell hivatkozni.

 **Példa:** `/usr/include`-beli állományok felfrissítése az aktuális könyvtárban levők

alapján:

```
Az INCDIR könyvtarnev definalalasa
(ha mas könyvtarbol inditjuk a make-t)
INCDIR = /usr/include
A könyvtarbeli allomanyok egy csoportjanak definalalasa
INCLUDES makronev alatt
INCLUDES = \
 $(INCDIR)/stdio.h \
 $(INCDIR)/pwd.h \
 $(INCDIR)/dir.h \
 $(INCDIR)/a.out.h \
Mindegyik felsorolt allomany a hasonnevű aktualis
könyvtarbeli allomanytól függ.
$(INCLUDES): (@F)
Másolja át az újabb aktualis könyvtarbeli allomanyt
a /usr/include könyvtarba
 cp $? $@@
Allítsuk be csak olvashatóra a státuszt
 chmod 0444 $@
```




# 13. fejezet

## A `vi` editor használata

A `vi` képes több állományt is szerkeszteni, ha egymás után megadjuk az állományok nevét:

```
$ vi Állomány1 Állomány2 ...
```

 `vi`  
rövid leírása  
a 45. oldalon

A `vi` szerkesztő a UNIX operációs rendszer `ex` editorára épül, ezért az `ex` parancsok használhatók a `vi`-on belül. Az `ex`-nek megfelelő szokásos `vi` korlátok a következők:

- 2048 karakter soronként
- 256 karakter globális parancsonként
- 128 karakter az előzőleg beszúrt és törölt szöveg esetén
- 128 karakter shell parancsonként
- 128 karakter stringenként
- 30 karakter az ún. *tag* név maximális hossza
- 1 048 560 sor soronként 2048 karakterrel számolva
- 128 makró 2048 össz karakterszámmal

### 13.1. Üzem módok a `vi`-ban

A `vi` háromféle üzemmódot használ: a parancs, a szövegírás és az ún. utolsó sor üzemmódot.

A parancs üzemmódba a belépés után kerülünk be. Ekkor különböző kurzor-mozgató, kereső, törlő, állománykezelő (írás/olvasás), stb. parancsokat adhatunk a `vi`-nak. Parancs üzemmódban használjuk pl. a `Ctrl-L` parancsot, ami a képernyőt

írja újra. A `vi` mindig a parancsmódba tér vissza, amikor az alparancsok és más üzemmódok befejeződnek. Egy adott alparancs megszakításához az `ESC`, vagy a `Ctrl-C` gombot<sup>1</sup> használhatjuk.

A szövegírás módban közvetlenül változtathatunk a szövegen. A parancsmódból a szövegírás módba a szövegírás parancsok egyikével (`a`, `A`, `i`, `I`, `o`, `O`, `c`, `C`, `s`, `S` és `R` parancsok) léphetünk be. Az üzemmódból az `ESC` gombbal térhetünk vissza a parancsmódba, ill. az `Ctrl-C` gombot is használhatjuk az üzemmód megszakításához.

Az ún. utolsó sor üzemmódban a `vi` a képernyő legalján olvassa be a parancsot. Ilyenek a `:`, `/`, `?` és `!!` parancsok. A parancssor lefuttatásához azt `Enter`rel (vagy `ESC`-el) kell zárunk, míg megszakítani a `Ctrl-C` gombbal tudjuk. A parancsoknál a következőképpen hivatkozhatunk a file soraira:

- `%` minden sor
- `^` első sor
- `$` utolsó sor
- `.` az aktuális sor

`/Minta/` a *Mintát* tartalmazó sor az aktuális sor alatt

`?Minta?` a *Mintát* tartalmazó sor az aktuális sor előtt

`mBetű` az adott pozíciót megjelöli egy *Betű* jellel. A *Betű* bármilyen ASCII kisbetű karakter lehet

`' Betű` az adott *Betű* jellel jelölt sorra hivatkozik

`\ Betű` az adott *Betű* jellel jelölt karakterre hivatkozik

## 13.2. A `vi` editor különböző beállításai

Leggyakrabban a `vi`-t különböző (külső ill. belső) paraméterekkel hívjuk meg. A parancssorban a következő kapcsolókat használhatjuk:

`-l` LISP üzemmód. A `vi` ekkor a LISP nyelvű programok írására használható kényelmesen, és megváltoztatja a különböző zárójeleket a LISP parancsok írásának megkönnyítésére.

---

<sup>1</sup>A vezérlő (`Ctrl`) karakterek kiadásához nyomva kell tartanunk a `Ctrl` (esetleg `Cntrl`) gombot, és eközben kell a megadott gombot lenyomni.

- r egy előző szerkesztés véletlen megszakadása után visszanyeri az előző állapotot. Az esetleges állomány megadása nélkül csak kilistázza a visszakapható file-okat.
- R csak olvashatóra nyitja meg az állományt.
- t*Tagnév* Ha létezik a (*ctag* paranccsal létrehozott) ún. tag lista, ami pl. egy programot tartalmazó file esetén a függvénynevek és azok pozíciója lehet, akkor a file szerkesztésekor a *Tagnév*vel jelzett pozícióra (függvényre) ugrik.
- w*Szám* az ablak méretét *Szám* sornyira állítja.
- y*Szám* a maximális sorok számát a *Szám*ra állítja.
- +*ex-parancs* a szerkesztés elkezdése előtt végrehajtja az *ex-parancs*ot.

A *vi* belső változóit ideiglenesen a

```
:set Változó=Érték
```

utasítással állíthatjuk be. Az alapértelmezéstől való rendszeres eltérés esetén érdemes a változók értékét permanensen átállítani. A *vi* elindulásakor végrehajtja a *.profile* file-ban az EXINIT környezeti változóval megadott beállításokat, vagy ha ez nincs megadva, a HOME könyvtárban található *.exrc* állományban leírt parancsokat. Ezek után a *vi* végrehajtja az aktuális könyvtárban lévő *./exrc* állományban található parancsokat (feltéve, hogy létezik ilyen állomány). Mivel ez az itt lévő parancsokat külső ellenőrzés (megerősítés) nélkül lefuttatja, biztonsági problémának tekinthető, és az újabb rendszerekben ezt a *set exrc* utasítással külön engedélyezni kell.

Az *.exrc* állományban soronként fel kell sorolni az értékadásokat, pl.:

```
set cp=3.,;
set number
set noedcompatible.
```

A leggyakrabban használt, *set*tel beállítható változók a következők (a zárójelben a használható rövidítések szerepelnek):

*closepunct* (*cp*=) A szövegzáró elválasztási jeleket sorolja fel. A *vi* ezeknél a jeleknél nem darabolja fel a szöveget. Ha egyszerre több ilyen karaktert akarunk megadni, először a számukat kell megadni (pl. a *cp=3.,;* a három írásjelet állítja be). Ez az opció nem működik minden UNIX rendszeren.

*directory* (*dir*=) Az editálás során használt ideiglenes bufferhez tartozó könyvtárat definiálja. Értéke általában a */tmp*.

*edcompatible* (*ed*) a *g*, *c* és *r* parancsok működését az *ed* szerkesztőhöz teszi hasonlatossá. Általában a *noedcompatible* beállítást használjuk.

`ignorecase (ic)` a keresés során nem különbözteti meg a kis- és nagybetűket.

Alapértelmezésben `noic`.

`magic` beállítja a reguláris kifejezések használatát a keresésnél. Kikapcsolva csak a `^( ) $` jeleknek van speciális jelentése. Alapértelmezése `magic`.

`modelines` ha a szerkesztendő file első vagy utolsó 5 sora

`ex: parancs... :`

vagy

`vi: parancs... :`

formátumban szerkesztő parancsokat tartalmaz, lefuttatja azokat. Alapértelmezésben `nomodelines` az értéke, mivel ez a parancsokat külső ellenőrzés (megerősítés) nélkül lefuttatja, s így biztonsági problémának tekinthető.

`number (nu)` A sorok előtt kijelzi azok sorszámát.

`readonly (ro)` bekapcsolja a csak olvasás üzemmódot.

`scroll (scr=)` az ablak felfelé/lefelé mozgásánál megadja a sorok számát.

`shell (sh=)` a `!` parancsnál használt shell típusát tartalmazza.

`showmatch (sm)` kijelzi az egymáshoz tartozó (`{ }` és `[[ ]]`) zárójeleket.

`showmode (smd)` kijelzi az editor szövegrész (input) módját.

`term (term=)` a használt terminál típusát adja meg. Alapértelmezésben ez a `TERM` változó értéke.

`window (wi=)` a kijelzéskor használt lapméretet definiálja.

`wrapmargin (wm=)` beállítja az automatikus betördelést, azaz a megadott margó utáni szavak elé automatikusan egy újsor karaktert illeszt. Alapértelmezésben `wm=0`, ami kikapcsolja ezt a funkciót.

### 13.3. A vi parancsai

A `vi` parancsmódjában használható utasítások általános formátuma a következő:

*[Buffer-Név] [Operátor] [Szám] Cél*

ahol a *Buffer-Név* egy adott ideiglenes szövegtároló területet jelöl; az *Operátor* egy érvényes `vi` parancs lehet; a *Szám* egy egész szám, ami akár sorszámot, akár más értéket adhat meg; a *Cél* pedig egy szövegrész ill. pozíció lehet (pl. egy bekezdés, szó, stb.). Például a

"2p

parancs a 2. számú törlési bufferből az aktuális pozícióhoz beszúrja annak tartalmát (a `vi` a kitörölt szövegrészeket 1 és 9 között számozott tárolókba helyezi, a legutolsó törlött rész az 1., az az előtti a 2., stb. tárolóban található).



A parancsok elé sokszor egy számot tehetünk, amit a `vi` ismétlésként vagy paraméterként értelmezhet. Például a

```
10Ctrl-U
```

parancs 10 sorral feljebb lép, míg a

```
4dw
```

parancs 4 szót töröl. A

```
137G
```

parancs a 137. sorra ugrik, míg a

```
25|
```

parancs a 25. oszlopra lép.

### 13.3.1. Mozgás az állományban

A `vi`-ban a kurzor egyszerű mozgásához parancsmódban a `h-j-k-l` gombokat használhatjuk. Néha a billentyűzeten található nyilak is működhetnek, érdemes ezért azokat kipróbálni.


A soron belüli mozgásnál a `0` (nulla) parancs a sor elejére, a `$` a sor végére ugrik. Az `fBetű` parancs a következő `Betű` karakterre, az `FBetű` a legutolsó `Betű` karakterre ugrik. A `tBetű` és `TBetű` hasonlóan a `Betű` karakter elé ugrik. A `;` parancs a legutolsó `f`, `F`, `t` és `T` parancsok egyikét ismétli meg, a `,` parancs pedig fordított irányba mozogva ismételi. Az oszlop parancs a `|`, amely a megadott oszlopra mozogtatja a kurzort.

A `vi` lehetőséget ad a szavankénti mozgásra is. A `w` előre, a `b` visszafelé ugrik egy szót. Az `e` parancs a következő szó végére viszi a kurzort. A `w`, `B` és `E` parancsok hasonlóan működnek, de nem veszik figyelembe az elválasztó karaktereket.

A sorok közötti mozgásra a következő parancsokat használhatjuk: a `H` parancs a képernyő tetejére, az `M` a közepére, az `L` pedig az aljára ugrik. A `+` (és a `-`) parancs a következő (előző) sor első nem-szóköz karakterére lép. Egy adott sorra a `sorszámG` paranccsal ugorhatunk (természetesen ilyenkor is használhatjuk pl. a `'BetűG` parancsot, ami a `Betű`vel jelölt sorra ugrik). Hasznos még a `''` parancs, amely felcseréli az aktuális és az előzőleg megjelölt kurzorpozíciót.

Ha gyorsabban akarunk mozogni, akkor a használhatjuk a képernyőnyivel mozgó parancsokat is: a `Ctrl-F` egy teljes képernyővel előre, míg a `Ctrl-B` egy képernyővel visszafelé lapoz. A `Ctrl-U` és a `Ctrl-D` előre/hátra mozog fél képernyőnyit. A `Ctrl-E` felfelé, a `Ctrl-Y` lefelé tolja el a képernyőt egy sorral.

A lapozás után esetleg szükség lehet a képernyő újraírására, amit a `Ctrl-L` parancs végez el. A `z` `Enter` parancs szintén újraírja a képernyőt, de az aktuális sor ekkor a képernyő tetején, a `z-` parancsnál az alján, a `z.` parancsnál pedig a

-  képernyő közepén lesz. Például a `/Haho/z` parancs megkeresi a `Haho` stringet a file-ban, és a képernyő közepére helyezve kijelzi a sort.

### 13.3.2. String keresés

Az adott kurzorpozíció után a `/Minta` parancs keresi meg a `Minta` előfordulását. Visszafelé a `?Minta` paranccsal kereshetünk. Az `n` utasítás megismétli az utolsó kereső parancsot az azzal megegyező, míg a `N` parancs a paraccsal ellentétes irányban. Ha éppen egy zárójel-karakteren helyezkedik el a kurzor, a `%` jel megkeresi a hozzá tartozó záró vagy nyitó zárójel-karaktert.

A keresés során reguláris stringeket is használhatunk (de csak ha a `magic` változó be van állítva (alapértelmezés)).

### 13.3.3. A szövegírás üzemmód

A parancs üzemmódból a szövegírás módba a következő utasításokkal léphetünk át (vissza az `Esc` gombbal léphetünk):

- a Az aktuális kurzorpozíció után kezdi el beilleszteni a begépelte szöveget.
- A Az aktuális sor végére ugrik, és átkapcsol szöveg beszúrára.
- i Az aktuális pozíció elé illeszti be a szöveget.
- I Az aktuális sor elejére ugrik, majd a begépelte szöveget oda írja be.
- o (kis o betű) Az aktuális sor után új sort nyit, annak elejére lép, és ott kezdi el a szöveg beillesztését.
- O (nagy O betű) Az aktuális sor előtt nyit egy új sort, és ott kezdi el beszúrni a szöveget.

A szövegírás üzemmódban a következő speciális parancsokat használhatjuk:

- `Ctrl-H` Kitörli az utolsó karaktert. Ezt általában a `Backspace` vagy `Delete` gombokkal is megtehetjük.
- `Ctrl-U` Kitörli az utolsó beadott sort.
- `Ctrl-W` Kitörli az utolsó szót.
- `Ctrl-V` Az utána begépelte speciális karaktert beilleszti a szövegbe.

### 13.3.4. Szövegváltoztatás a parancs módban

A parancs üzemmódban a következő utasításokat is használhatjuk a file megváltoztatására (ezek egy részéből az `ESC` billentyűvel léphetünk ki, tekintve, hogy átkapcsolnak szövegírási módba):

- `rBetű` Az éppen aktuális karaktert a *Betű*vel helyettesíti.
- `R` Átkapcsol felülírás üzemmódba.
- `C` Megváltoztatja a sor többi részét.
- `cc` Megváltoztatja az egész sort.
- `cw` Megváltoztatja a következő szót.
- `x` Kitörli a kurzor helyén lévő karaktert.
- `x` vagy `dh` Kitörli a kurzor előtti karaktert.
- `D` Kitörli az aktuális pozíciótól a sor végéig eső részt.
- `dd` Kitörli az egész sort.
- `dw` Kitörli a következő szót.
- `d' Betű` Az aktuális pozíciótól kitörli a *Betű*vel jelzett részig a szöveget.
- `u` Visszaváltoztatja az előző változtatást.
- `<<` Egy tabulátor pozícióval balra tolja a sort.
- `>>` Egy tabulátor pozícióval jobbra tolja a sort.
- `<L` Egy tabulátor pozícióval balra tolja az összes sort az aktuális sortól kezdve a képernyő végéig.
- `>L` Egy tabulátor pozícióval jobbra tolja az összes sort az aktuális sortól kezdve a képernyő végéig.
- `>' Betű` Az aktuális pozíciótól a *Betű*vel jelzett pontig jobbra tolja egy tabulátorral az összes sort.
- `<' Betű` Az aktuális pozíciótól a *Betű*vel jelzett pontig balra tolja egy tabulátorral az összes sort.
- `J` Összefűzi az aktuális sort a következővel.
- `~` Megcseréli az aktuális betűt a kis/nagybetűs párjára.

A parancsok kiadásánál használhatunk szám paramétereket is, azaz pl. a `16x` parancs kitörli a következő 16 betűt, a `3dd` pedig 3 sort töröl.

### 13.3.5. Másolás és mozgatás

A `vi` editorban másolásra ill. mozgatásra buffereket (tárolókat) használunk. A `vi` automatikusan 1-től 9-ig számozva tárolja az utolsó 9 törölt értéket. Melletük a 26 ASCII kisbetűvel lehet felhasználói buffereket definiálni. A tárolókra a "*Szám*" vagy "*Betű*" paranccsal hivatkozhatunk.

Nagyobb szövegrészek másolására és mozgatására az állományon belül a következő parancsokat használhatjuk (az esetleges elhibázott, befejezetlen parancsot az `ESC` gombbal lehet megszakítani):

- `p` A legutolsó törlési paranccsal kitörölt szöveget illeszti a kurzor után (azaz a "`1`" értékét szúrja be).
- `P` A legutolsó törlési paranccsal kitörölt szöveget illeszti a kurzor elé.
- "`Bufferp`" A *Buffer* tartalmát illeszti a kurzor után (azaz pl. `p` megegyezik a "`1p`" paranccsal).
- "`BufferP`" A *Buffer* tartalmát illeszti a kurzor elé.
- `y` `Szóköz` Az aktuális karaktert bemásolja a törlési tárolóba ("`1`" tároló).
- `yw` A következő szót bemásolja a törlési tárolóba.
- `yy` Az aktuális sort bemásolja a törlési tárolóba (pl. `12yy` a következő 12 sort másolja be, `:12,40y` a 12. és a 40. sor közötti szakaszt, stb.).
- `Y` Az aktuális sort másolja be a törlési tárolóba.
- "`Betűyy`" Az aktuális sort a *Betű* tárolóba teszi.
- `y' Betű` Az aktuális pozíciótól kezdve bemásolja a *Betű* jelig terjedő részt a törlési tárolóba.

### 13.3.6. Visszaállítás és parancs-ismétlés

A `vi` editor megőrzi az utolsó 9 törlés eredményét, ezért - ugyan csak korlátozott mértékben - de lehetőség van az esetleges hibás utasítások előtti állapot visszaállítására. Erre a következő parancsok szolgálnak:

- `u` Visszaállítja az utolsó változtatás előtti helyzetet.
- `U` Visszaállítja az aktuális sort, ha nem léptünk közben más sorra a kurzorral.
- "`Szám``p`" A *Szám* törlési buffer értékét szúrja be. (A *Szám* 1 és 9 között lehet).
  - `.` Megismétli a legutolsó változtatást. Ha az a "`Szám``p`" volt, eggyel megnöveli a *Szám*ot.

### 13.3.7. Szövegrész helyettesítés

Egy adott szövegrész helyettesítését a `:` (utolsó sor) parancsmódból végezhetjük el. Például a `régi-string` soronkénti első előfordulását a következő parancs helyettesíti át az egész állományban `új-string-re`:

```
:1,$s/régi-string/új-string/
```

A `:` után meg kell adni a helyettesítési tartományt (azaz pl. itt az első sortól az utolsóig, amit a `$` jelöl), majd az `s` parancs (substitute) után a két stringet. A `régi-string` reguláris kifejezés is lehet. Az `új-string` esetén az `&` jel a keresett kifejezést (itt a `régi-stringet`) jelöli, míg a `%` az előző helyettesítő stringre hivatkozik.

A parancs csak a soronkénti első előfordulást helyettesíti. Ha az összes előfordulást ki szeretnénk cserélni, használhatjuk a

```
:1,$s/régi-string/új-string/g
```

(`g` mint global) parancsot. Lehetőség van a helyettesítés interaktív megerősítésére is, amit a `g`-hez hasonló `c` (confirm) kapcsolóval állíthatunk be.

Ha valamilyen okból nem szeretnénk a kereséskor különbséget tenni a kis- és nagybetű között, akkor érdemes beállítani az `ignorecase` változót (`:set ignorecase` parancs).

## 13.4. Állománykezelés

Egy adott állomány szerkesztése után azt a lemezre a

```
:w
```

paranccsal írhatjuk ki. Új állományként való kiírásakor használjuk a

```
:w FileNév
```

parancsot.

Egy újabb file-t a

```
:e FileNév
```

utasítással olvashatunk be. Mivel ilyenkor az előző változtatások elvesznek, ha azokat nem mentettük el, az utasítás hibüzenettel tér vissza. Ha nem szándékunk elmenteni a változtatásokat, akkor a

```
:e! FileNév
```

utasítást kell használnunk. Speciálisan a

```
:e!
```

utasítás újra beolvassa az éppen szerkesztett file-t a lemezeről. A parancs után a szerkeszteni kívánt sor számát is megadhatjuk, pl. az

```
:e +14 vi.doc
```

a `vi.doc` file 14 sorára ugrik beolvasás után.

A `vi` lehetőséget ad induláskor a shell parancssorban több file megadására is. Amennyiben a sorban következő file-ra szeretnénk lépni, akkor erre használhatjuk

```
a
```

```
:n
```

parancsot (ill. a `:n!` parancsot, ha nem akarjuk elmenteni a változtatásokat). Az `:n` parancs után egy esetleges új file-listát is megadhatunk.

Egy másik file tartalmát az éppen aktuális bufferba (file-ba) a

```
:r FileNév
```

parancssal szűrhatjuk be az aktuális sor után.

Az éppen használt file-t ill. a bufferhez tartozó információkat a `Ctrl-G` parancs használatával kérhetjük le.

A `vi` editorból a

```
ZZ
```

parancssal léphetünk ki: ekkor elmentődik az éppen szerkesztett file. Ha mentés nélkül szeretnénk kilépni, akkor erre a

```
:q
```

parancs szolgál, amennyiben nem változtattuk meg a buffert. Változtatás esetén a

```
:q!
```

parancs eldobja a változtatásokat és kilép a `vi`-ből.

A UNIX rendszerben az állományoknak nincsenek — a VMS rendszerben használthoz hasonló — verziószámai, és nincs *undelete* típusú parancs sem. Ezért a szerkesztett állományok kiírása **végleg** letörli az előző változatot!

## 13.5. A shell elérése a `vi`-ből

A `vi` editorban lehetőség van a UNIX operációs rendszer shelljének többféle elérésére. A legegyszerűbben a

```
:sh
```

parancssal léphetünk ki a shellbe, ahonnan a `Ctrl-D` vagy `exit` utasítással visszakerülünk az editorba. Érdemes a shellbe való kilépés előtt mindenképpen elmenteni a file-t.

Egy *Shell-parancs*ot a

```
:!Shell-parancs
```

utasítás hajt végre, (pl. a

```
:!ls
```

kilistázza az aktuális könyvtár tartalmát). A legutolsó ilyen parancsot a

```
:!!
```

utasítással ismételtethetjük meg.

Egy UNIX *Shell-parancs* végeredményét (a standard outputot) a

```
:Számr!Shell-parancs
```

utasítás írja be a *Szám* sortól kezdődően a file-ba. A *Szám* megadása nélkül az aktuális sorra hivatkozunk.

## 13.6. Makrók és parancsok definiálása

Gyakran szükség van bizonyos bonyolultabb parancsok többszöri végrehajtására. A UNIX operációs rendszer *vi* editora lehetőséget ad *vi* parancsok tárolására és végrehajtására.

A legegyszerűbb ilyen módszer a makró létrehozása. Ekkor a *vi* parancsokat egy *Betű*vel jelzett bufferbe kell betölteni (pl. soronként), majd a parancsokat a `@@Betű` parancssal végrehajthatjuk. Lehetőség van a legutolsó ilyen parancs megismétlésére is, amire a `@@@szolgál`. A *Betű* kis- és nagybetű egyaránt lehet, az `a-z` bufferek betöltik, míg az `A-Z` bufferek pedig hozzáfűzik a szöveget a bufferhez, így lehetőség van a makrók lépésenkénti fejlesztésére is. Például az

```
o :1,$s/haho/haliho/g ESC "Add
```

parancs-sorozat után a `@@A` parancs végrehajtja a helyettesítést. A munkaterületre írt parancsot az `A` bufferben helyeztük el. A példát parancs üzemmódból indulva, karakterenként gépeljük be!

A *vi* rendelkezik a makróknál hatékonyabb parancsismétlési eszközzel is, mégpedig az új parancsok (billentyű leütések) definiálásával. A

```
:map Betű vi-Parancsok
```

utasítás a *Betű* billentyű lenyomása esetén (parancs üzemmódban) végrehajtja a felsorolt *vi-Parancsokat*. Amennyiben szöveg beszúrásakor is szeretnénk használni ez a definíciót, ezt a

```
:map! Betű vi-Parancsok
```


utasítással tehetjük meg. A `map` parancs paraméter nélkül kiírja az éppen használt definíciós listát. A `map` parancs hatását az `unmap` parancs szünteti meg. Például a

```
:map q :q! Ctrl-V Enter Enter
```

parancs megváltoztatja az `q` gomb hatását, az `q` lenyomása után kilépünk a *vi*-ből.

Egy adott makró vagy billentyű-definíciót a `.exrc` file-ban definiálva tehetünk permanenssé.

Egy adott definíció esetén, ha az éppen használt karakterre akarunk hivatkozni, használjuk a `Ctrl-V` gombot a karakter előtt. Szintén ezt a karaktert kell használni, ha a terminál funkciógombjait szeretnénk definiálni.

A `map` parancs mellett a `:ab` (abbreviation) parancsot is használhatjuk gyakran  használt rövidítések esetén. Az

```
:ab t type
```

parancs után szövegbeszúrás üzemmódban az összes `Szóköz-t-Szóköz` kifejezésből `Szóköz-type-Szóköz` kifejezés lesz. Nem helyettesítődik viszont a `t` sem egy szón belül, sem parancsmódban. Az `:ab` parancs paraméter nélkül kiírja az éppen használt rövidítéseket, a `:unab` pedig egy előző `:ab` hatását szünteti meg.



## 14. fejezet

# A MicroEMACS szövegszerkesztő leírása

Az `uemacs` szövegszerkesztő egyszerre több állományt is tud kezelni és megjeleníteni a képernyőn (ezt csak a rendelkezésre álló memória, ill. a képernyő mérete korlátozza). A MicroEMACS szerkesztőt több állomány esetén a

§ `uemacs ÁllományNév1 ÁllományNév2 ...`


parancs kiadásával hívjuk meg. Néhány gépen a `uemacs` parancs helyett az `emacs` parancsot kell használnunk.

Lehetőség van előre meghatározható parancssorozatok beállítására (ún. *makrók*). A kényelmesebb működtetés érdekében az egyes gépeken a felhasználó bejelentkezési könyvtárában a `.emacsrc` állomány tartalmaz egy makró-sorozatot, amelyet a szerkesztőbe való belépés előtt a MicroEMACS automatikusan végrehajt.

A MicroEMACS magyar nyelvű tanító-állománnyal is rendelkezik (J. Tóth Gábor fordítása), ami „megtanítja” a szövegszerkesztő alapszintű használatát. A program a

§ `tanits_emacs`

utasítással hívható be az erre beállított gépeken.

 `emacs`  
rövid leírása  
a 46. oldalon

### 14.1. A legfontosabb MicroEMACS parancsok

A parancsok kiadásához általában a `Control` (vagy `Ctrl`) és az `Esc` (vagy `Escape`) billentyűket egy másik billentyűvel kombinációban használjuk. A parancsok jelölésére a következő MicroEMACS jelöléseket használjuk:

^Billentyű a `Ctrl` billentyűt lenyomva tartva a `Billentyű`-t kell lenyomni. Azaz `^V` jelentése: a `Ctrl` billentyűt lenyomva tartva a `V` billentyűt kell lenyomni.

M-Billentyű nyomjuk meg majd engedjük el az `Escape` (vagy `Esc`) gombot, majd a `Billentyű`-t (az `Escape` a MicroEMACS ún. *metakaraktere*). A kis- és nagybetűk között a program nem tesz különbséget. Néhány parancsnál (pl. `M-^W`) a `Billentyű` lenyomása előtt a `Ctrl` billentyűt is le kell nyomni.

Bizonyos termináloknál nincs külön `Esc` gomb, ekkor vagy a `^[` kombinációt használhatjuk helyette, vagy átdefiniálhatjuk a meta karaktert a MicroEMACS-ban egy másik gombra.

A MicroEMACS parancsait megszakítani a `^G` utasítással lehet, amikor a program paramétereiket (adatot) kérdez be. Ha nem tudjuk, hogy mi történik (vagy mit kérdez éppen a program), ezt a parancsot kell kiadnunk! Ne feledjük, hogy a programból változtatás nélkül a `^X ^C` parancssal léphetünk ki!

## 14.2. Mozgás az állományban

A szerkesztőbe való belépés után megírhatjuk a kívánt szöveget, miközben a kurzor mozgatására a billentyűzeten található nyilakat használhatjuk. Ha a billentyűzet nyilai valamilyen ok miatt nem működnek rendesen, akkor a következő parancsokat is használhatjuk a képernyőn való mozgásra:

|                                 |                               |
|---------------------------------|-------------------------------|
| <code>^F</code> egy betű előre  | <code>^N</code> következő sor |
| <code>^B</code> egy betű vissza | <code>^P</code> előző sor     |

A szövegben való gyorsabb (kényelmesebb) mozgást a következő parancsok teszik lehetővé:

|                                             |                                            |
|---------------------------------------------|--------------------------------------------|
| <code>^A</code> sor elejére ugrik           | <code>M-&lt;</code> a szöveg elejére ugrik |
| <code>^E</code> a sor végére ugrik          | <code>M-&gt;</code> a szöveg végére ugrik  |
| <code>^V</code> egy képernyővel előre       | <code>M-F</code> a következő szóra ugrik   |
| <code>^Z</code> egy képernyővel vissza      | <code>M-B</code> az előző szóra ugrik      |
| <code>M-B</code> a bekezdésre elejére ugrik | <code>M-G</code> egy adott sorra ugrik     |
| <code>M-N</code> a bekezdés végére ugrik    |                                            |

### 14.2.1. Törlés és beszúrás

A szöveg begépelésekor a kurzor előtt álló betűt a `Backspace` gombbal lehet törölni (ezt a gombot néha `Delete` vagy `←` felirat jelzi). Ha esetleg egyik gombot sem tudjuk megtalálni a billentyűzeten, akkor a `^H` parancs szintén törli az előző betűt. Ha a szövegben nagyobb részeket szeretnénk törölni, akkor hasznosak lehetnek a következő utasítások:

`M-Backspace` az előző szót törli

`M-D` a következő szót törli

`^K` Az adott kurzorpozíciótól a sor végéig töröl, második lenyomásra pedig kitörli a sorvége karaktert.

`^W` Kitörli az aktuális kurzorpozíció és az ún. „jel” közötti tartományt. A „jel”-et az aktuális kurzorpozícióban az `M-Space` (Szóköz) utasítással rögzíthetjük.

`M-W` „Eltávolja” az aktuális kurzorpozíció és az ún. „jel” közötti tartományt.

`M-^W` Az aktuális bekezdést törli ki.

`^X ^O` Kitörli az aktuális kurzorpozíció körüli üres sorokat.

Ha az előző utasításokkal esetleg olyan részt töröltünk ki, amelyet nem kellett volna, NE ESSÜNK PÁNIKBA! A MicroEMACS a legutolsó törölt egységet egy tárolóba teszi, és onnan a `^Y` utasítással bemásolhatjuk az aktuális kurzorpozícióba. Ezzel lehetséges szövegrészek mozgatása és – a `^Y` többszöri lenyomásával – másolása is, amely a következőképpen végezhető:

- bejelöljük az aktuális pozíciót az ún. „jel”-l, az `M-Szóköz` utasítással;
- a tartomány másik végére mozgunk, és kitöröljük (`^W`) vagy egyszerűen csak bemásoljuk (`M-W`) a tartományt a törlés tárolójába;
- az új helyre visszük a kurzort, és ott a `^Y` paranccsal – akár többször is – előhívhatjuk a törlési tárolóból az adatokat.

## 14.3. Keresés és helyettesítés

A MicroEMACS képes egy szövegrész (string) megkeresésére, és esetleges helyettesítésére.

Egy stringet az adott pozíció után a `^S` utasítás<sup>1</sup> segítségével találhatunk meg. A parancs kiadása után kéri a keresendő szöveget, amelyet a `meta` karakterrel

<sup>1</sup>Bizonyos esetekben a `^S` kombináció leállítja a képernyőre való írást. Ekkor a `^Q` paranccsal újra engedélyezni kell a képernyőre való írást, és a `^S` helyett a `^\`-t kell használnunk

(`ESC` gomb) kell lezárni. Ezután a kurzor a megtalált szövegrészre áll. Ha ezután újra ugyanazt a stringet szeretnénk keresni, a `^S` utasítás után már csak a `meta` karaktert (`ESC` gomb) kell leütni.

A `^R` utasítás az adott kurzorpozíció előtti keresést teszi lehetővé, használata megegyezik a `^S` paranccsal.

Ha nemcsak keresni, de helyettesíteni is akarunk egy szövegrészt, akkor az `M-R` és az `M-^R` parancsokat kell használnunk. Az `M-R` parancs kiadása után kéri a *RégiSzöveget*, majd az *ÚjSzöveget*. A szövegeket a `meta` karakterrel kell lezárni. Ezután az adott pozíciótól kezdve a *RégiSzöveg* minden előfordulását a *ÚjSzövegre* cseréli ki.

Ha nem vagyunk biztosak, hogy mindenhol el kell-e végezni a cserét, akkor az interaktív `M-^R` parancsot kell használnunk. Ez hasonlóan működik az `M-R` parancshoz, de a *RégiSzöveg* minden előfordulásánál megkérdezi, kicserélje-e, vagy sem. A válaszok a következők lehetnek:

`^G` megszakítás

`.` megszakad, és visszaugrik a kezdőponthoz

`Y` helyettesítés

`N` nem helyettesít az adott helyen, de folytatja a keresést

`!` feltétel nélkül helyettesíti az összes maradék előfordulást

`?` segítség

A keresés során a szövegek egyezésénél a MicroEMACS alapértelmezésben nem különbözteti meg a kisbetűket és a nagybetűket. Ha erre mégis szükségünk lenne, akkor az EXACT üzemmóddal (amit a `^X-M exact` parancs állít be, l. 190. oldal) ezt megtehetjük.

### 14.3.1. Állománykezelés

Egy állományt a `^X ^F` utasítás segítségével olvashatunk be. Az utasítás megkérdezi a keresett állomány nevét, amelyet a `Return` (vagy `Enter`) gombbal kell lezárni. Ha ez az állomány nem létezik, a MicroEMACS létrehozza azt.

A szerkesztett állományainkat a számítógép memóriájából ki kell írunk a diszkre, ha nem akarjuk, hogy a változtatások elveszenek a MicroEMACS-ból való kilépéssel. Ezt a `^X ^S` paranccsal<sup>1</sup> végezhetjük el munka közben. Ha a szerkesztést is be akarjuk fejezni, és ki akarunk lépni a programból a szöveg elmentésével, akkor az `M-Z` parancsot érdemes használnunk. Ha bármilyen okból

NEM akarjuk megőrizni a változtatásokat, akkor a `^X ^C` a módosítások elmentése nélkül kilép a programból (de előtte megkérdezi, hogy valóban el akarjuk-e dobni a változtatásokat).

A program ismeri még a következő állománykezelő utasításokat is:

- `^X ^R` egy új állományt olvas be az éppen aktuális bufferbe. A buffer előző tartalmát törli (de figyelmeztet, ha már változtattunk valamit rajta).
- `^X ^I` az adott kurzorpozícióhoz beszúr egy másik állományt.
- `^X ^V` beolvas egy állományt VIEW üzemmódban (azaz nem tudunk rajta változtatni).
- `^X ^W` kiírja a buffert egy másik állományba, aminek megkérdezi a nevét.
- `^X N` az éppen aktuális bufferhez tartozó állomány nevét változtatja meg.

### 14.3.2. Buffer és ablak kezelés

A MicroEMACS program nagy előnye, hogy egyszerre több állományt is képes használni, sőt a képernyőn több állományt is szerkeszthetünk egyszerre. A program minden állományhoz egy tárolót (ún. *buffert*) rendel. A buffereket külön-külön kezelhetjük.

A buffereket a MicroEMACS a képernyőn ablakokban jeleníti meg. Minden buffer alján egy állapot sor jelenik meg, amelyik tartalmazza a program nevét (pl. MicroEMACS 3.10), zárójelek között az éppen használt üzemmódokat, a buffer és a bufferhez tartozó állomány nevét. A sor legelején egy `*` jelzi, ha már végeztünk változtatásokat a bufferen, de még nem mentettük el az állományba.

A következő utasításokkal kezelhetjük az ablakokat a programban:

- `^X 2` félbevágja az éppen aktuális ablakot
- `^X 1` az éppen aktuális ablak kivételével minden más ablakot töröl a képernyőről
- `^X 0` törli a képernyőről az aktuális ablakot
- `^X O` a következő ablakra ugrik
- `^X P` az előző ablakra ugrik
- `M-^V` egyet lapoz következő ablakban előre
- `M-^Z` egyet lapoz következő ablakban visszafelé
- `^X Z` egy sorral megnagyítja az ablak méretét
- `^X ^Z` egy sorral csökkenti az ablak méretét
- `^X ^N` egy sorral eltolja az ablakot lefelé

- $\wedge X \wedge P$  egy sorral eltolja az ablakot felfelé
- $M-Szám \wedge X W$  az ablak mérete *szám* sor lesz
- $\wedge L$  újra írja a képernyőt

A bufferek kezelésére a következő parancsok szolgálnak:

- $\wedge X X$  a következő bufferre ugrik
- $\wedge X B$  egy másik bufferre ugrik
- $M-\wedge N$  megváltoztatja az aktuális buffer nevét
- $\wedge X K$  kitöröl egy buffert (ez a buffer nem lehet kijelezve a képernyőn)
- $\wedge X \wedge B$  kijelzi az éppen használt bufferek listáját

## 14.4. Speciális utasítások

A MicroEMACS-ban különleges szerepe van az aktuális kurzorpozíció és az ún. „jel” közötti tartománynak (pl. ezt a tartományt ki tudjuk törölni a  $\wedge W$  utasítással). A „jel”-et egy adott kurzorpozícióba az  $M-Space$  (Szóköz) utasítással rögzíthetjük. Az állományt a  $\wedge X <$  utasítással leszűkíthetjük az adott tartományra (azaz akkor csak ezt a darabot látjuk és érhetjük el). A leszűkített tartományból a  $\wedge X >$  utasítás állítja vissza a teljes állományt.

Használhatjuk a „jel”-et egy adott pozíció megjelölésére is, mivel a  $\wedge X \wedge X$  utasítás felcseréli a kurzorpozíciót és a „jel” helyét.

A MicroEMACS lehetőséget ad kisbetű-nagybetű cserére a következő utasítások segítségével:

- $M-U$  nagybetűssé változtatja a szót
- $M-C$  nagy kezdőbetűssé változtatja a szót
- $M-L$  kisbetűssé változtatja a szót
- $\wedge X \wedge U$  nagybetűssé változtatja a kurzorpozíció és a „jel” közötti tartományt
- $\wedge X \wedge L$  kisbetűssé változtatja a kurzorpozíció és a „jel” közötti tartományt

A MicroEMACS különböző üzemmódokban működhet. Az üzemmód beállítását a  $\wedge X M$ , törlését a  $\wedge X \wedge M$  parancs végzi egy adott buffer esetében. Ha ezt az összes bufferre akarjuk kiadni, akkor az  $M-M$  (beállítás) ill.  $M-\wedge M$  (törlés) parancsot kell használni<sup>2</sup>. A jelenleg használt üzemmódok a következők:

---

<sup>2</sup>Vigyázzunk, mert ez csak a parancs kiadása UTÁN beolvasott bufferekre határos!

- OVER bekapcsolja a felülírást a beszúrás helyett
- WRAP a sor vége után (amit az M-szám ^X-F paranccsal lehet szám szélesre beállítani) automatikusan új sorba ugrik
- VIEW csak olvasás üzemmódot állít be (azaz nem tudunk változtatni a bufferen)
- MAGIC speciális, a reguláris kifejezéseket használó keresési módot állít be
- EXACT a keresésnél és helyettesítésnél megkülönbözteti a kis- és nagybetűket
- CMODE a C programnyelvek írásakor használatos üzemmódot állít be (pl. ellenőrzi a különböző zárójelek egyezését, stb.)

Az M-Q utasítás segítségével betördelhetjük a bekezdést az M-Szám ^X-F paranccsal beállított Szám betűnyi szélességbe.

A MicroEMACS lehetőséget nyújt az operációs rendszer elérésére. A ^X ! utasítással egy UNIX parancsot tudunk elindítani az operációs rendszerben, míg a ^X C egy új UNIX shellt indít el, ahonnan a szokásos exit utasítással léphetünk ki.

A MicroEMACS az előző részekben felsorolt parancsokat bizonyos billentyű-leütésekre hajtotta végre. A parancsok hozzárendelése a billentyűkhöz megváltoztatható, a felhasználó tetszés szerint definiálhatja újra az egész billentyűzetet. A program emellett lehetőséget ad újabb parancsok (makrók) létrehozására több MicroEMACS utasítás összefűzésével. Ezeket a makrókat elhelyezhetjük külön parancs file-ban és név szerint hívhatjuk meg, vagy közvetlenül egy billentyűhöz is rendelhetjük őket. A leggyakoribb makrókezelő utasítások a következők:

- ^X ? leír egy adott gombhoz tartozó makrót
- M-A kilistázza a makrókat egy kulcs szerint
- ^X ( elkezdi rögzíteni a billentyűzeten beadott utasításokat
- ^X ) befejezi a billentyűzeten beadott utasítások rögzítését
- ^X E lefuttatja (az előző két paranccsal rögzített) billentyű-makrót
- M-^X végrehajt egy MicroEMACS utasítás sort (makrót).

A MicroEMACS az itt említetteken kívül még sok más lehetőséggel rendelkezik (pl. makró program írása, feltételes elágazások használata, lépésenkénti makró végrehajtás (debug) stb.), amit egyrészt az online helpben (M-? parancs), másrészt általában a /usr/local/doc/emacs/emacs.txt állományban találhatnak meg az érdeklődők.





# 15. fejezet

## A<sub>mail</sub> program működése

### 15.1. A<sub>mail</sub> felépítése

A<sub>mail</sub> parancs kétfajta levelesládát kezel, a rendszer és a felhasználó saját levél-állományát. Minden, a UNIX operációs rendszeren létező felhasználó számára létezik egy rendszer-levélszekrény, általában a

```
/usr/spool/mail/FelhasználóiAzonosító
```

állományként. A rendszer ebben tárolja azokat a leveleket, amelyeket a felhasználó kapott, de még nem olvasott, törölt vagy mentett el saját állományába. Alapértelmezésben, ha a felhasználó olvasta, törölte vagy elmentette az összes rendszer-mailbox üzenetet, a levélszekrény kitörlődik. Ha valamilyen okból mégis megszeretnénk tartani ezt az állományt, akkor a rendszergazda ezt beállíthatja a


```
chmod 755 RendszerLevélszekrényKönyvtár
```

paranccsal.

A rendszer levélszekrénye mellett minden felhasználónak személyi levelesládája van. Alapértelmezésben ez a \$HOME/mbox állomány (ami pl. lanceLOT felhasználó esetén /u/lancelot/mbox), de ez az MBOX shell változóval átállítható. A<sub>mail</sub> program az új levelek elolvasásakor azokat a rendszer levelesládájából ebbe az állományba mozgatja át (hacsak másként nem rendelkezünk).

A levelező program képes ún. folder („rendező”) állományok használatára. Ezeket a levelek csoportosítására használhatjuk, a mail program segítségével a leveleket az egyes folderek között mozgathatjuk, stb.. A foldereket célszerű külön alkönyvtárban elhelyezni.

Az esetleges új leveleket a UNIX<sub>mail</sub> rendszere a MAIL, MAILCHECK és a MAILMSG környezeti változók felhasználásával ellenőrzi le. A MAIL változó hatá-

 [mail](#)  
rövid leírása  
a 65. oldalon

rozza meg a rendszer levélszekrényének helyét. Interaktív bejelentkezés esetén a MAILCHECK változó mondja meg, milyen gyakran ellenőrizze az esetleges új levél érkezését az operációs rendszer (alapértelmezésben ez 600 másodperc). Új levél érkezésekor a MAILMSG környezeti változók tartalmát dírja ki (ez általában "YOU HAVE NEW MAIL").


A mail program elindításakor először végrehajtja a /usr/lib/Mail.rc állományban található mailbox parancsokat, majd a MAILRC változó által definiált felhasználói állományt, amennyiben létezik (alapértelmezésben ez a \$HOME/.mailrc).

A UNIX operációs rendszer lehetőséget ad a levelek automatikus továbbítására, amennyiben a felhasználó HOME könyvtárában létezik a .forward file. Az állománynak egyszerűen tartalmaznia kell a címzettet (vagy a címzettek listáját vesszővel elválasztva). A levelek továbbítása ilyenkor automatikus, ehhez nem kell belépni a mail rendszerbe.

A mail parancs kiadásakor különböző kapcsolókat használhatunk. A leggyakrabban használtak ezek közül a következők:

- e Leellenőrzi, hogy van-e új levél. Ez a kapcsoló különösen script file-okban használható jól, mivel az új levelet a parancs visszatérési értékében (státuszában) jelzi.
- f [ *File* ] A mail a rendszer levelesláda helyett az \$HOME/mbox (ill. a megadott *File*) állományt használja.
- f +*Folder* A rendszer levelesláda helyett a megadott folder alkönyvtárban a *Folder* folder állományt használja. A folder alkönyvtárat pl. az \$HOME/.mailrc állományban adhatjuk meg.
- i Letiltja az INTERRUPT jelet (ez különösen zajos vonalak, pl. modem estén hasznos).
- n Letiltja az /usr/lib/Mail.rc állomány beolvasását induláskor.
- N Nem nyomtatja ki a kezdeti fejléceket.
- H Csak a kezdeti fejléceket nyomtatja ki.
- s *Tárgy* Az elküldendő levél tárgya (Subject) a *Tárgy* lesz.
- v Bekapcsolja a részletes (verbose) üzemmódot.

Példák:

 Ha a személyes levelesládánk tartalmát szeretnénk megnézni, akkor ezt a  
\$ mail -f  
paranccsal tehetjük meg.

 Amennyiben leveleinket folderekbe osztottuk szét, akkor pl. a gral folder

tartalmát a

§ *mail -f +gral*

utasítással nézhetjük meg.

## 15.2. Parancsok levél küldése közben

Miközben levelet küldünk, a mail képes speciális, ún. tilde parancsok feldolgozására is. A parancsokat egy sorelejei tilde (~) katarakter előzi meg. A leggyakrabban használt ilyen parancsok a következők:

~! *[ShellParancs]*

Kilép a shellbe (és esetleg végrehajta a *ShellParancsot*).

~.

A levél végét jelzi (helyettesíti a `Ctrl-D`-t).

~?

Kilistázza a tilde parancsokat.

~: *MailParancs*

Végrehajta a *MailParancsot*. Csak akkor használható, ha előtte levélolvasásra indítottuk el a mail programot.

~a

A `$HOME/.mailrc` állományban megadható ún. aláírás stringet fűzi a levélhez.

~b *Felhasználó*

A *Felhasználót* hozzáadja az ún. vakmásolat (Bcc, blind carbon copy) listához. A *Felhasználó* ekkor megkapja a levél másolatát, de a levél fejlécében NEM jelenik meg, hogy ő is kapott másolatot.

~c *Felhasználó*

A *Felhasználót* hozzáadja a másolat (Cc, carbon copy) listához, és a levél fejlécében megjelenik, hogy ő is kapott másolatot.

~t *Felhasználó*

A *Felhasználót* hozzáadja a címzett (To) listához. A levél fejlécében megjelenik, hogy ő is a címzettek között van.

~h

megkérdezi a `Subject:`, `To:`, `Cc:` és `Bcc:` mezők tartalmát. Az esetleges előző beállításokat felül lehet írni.

~e

az EDITOR változóval (alapesetben ed) megadott szerkesztőt hívja be a levél elkészítéséhez.

~v

a VISUAL változóval (alapesetben vi) megadott szerkesztőt hívja be a levél elkészítéséhez.

~f [ÜzenetLista] Felhasználó

továbbküldi az ÜzenetListát (vagy az aktuális levelet), de csak ha levél olvasása közben vagyunk. A leveleket változtatás nélkül küldi el.

~m [ÜzenetLista]

Beszúrja az ÜzenetLista levelek tartalmát (vagy az aktuális levelet) a levélbe, de csak ha levél olvasása közben vagyunk. A beszúrt szöveget egy TAB karakterrel jobbra tolja.

~r Állomány

beolvassa a levélbe az Állományt. Használható még a ~< Állomány parancs is.

~w Állomány

a levelet beleírja az Állományba, de a fejlécek nélkül.

~q

kilép a levél elküldése nélkül. A nem üres, részben kész levelet a dead.letter állományba írja.

~x

kilép a levél elküldése nélkül, de a részben kész levelet nem menti el a dead.letter állományba.

### 15.3. A mailbox alparancsok

A levelesláda ill. a folderek olvasásakor a mail sokféle parancssal segít a felhasználónak. A parancsok megadásakor az egyes levelekre számokkal ill. listával hivatkozhatunk. A lista lehet a számok felsorolása szóközzel elválasztva (pl. 1 2 4 9), de lehet egy tartomány is (pl. 1-137). A . az éppen aktuális levélre vonatkozik (amit a > jelöl), míg a ^ az első, a \$ pedig az utolsó levélre vonatkozik. Az összes levélre a \* jellel hivatkozhatunk. A Felhasználó kifejezés az összes, az adott Felhasználótól érkező levélre vonatkozik. A /Szöveg minden levelet kiválaszt, aminek subject sorában a Szöveg előfordul (kis- és nagybetűt nem különböztet meg itt a rendszer). Ha a :Állapot parancsot adjuk ki, akkor minden

adott *Állapot*ú levélre hivatkozhatunk (az *Állapot* lehet *d* (deleted, törölt), *n* (new, új), *o* (old, régi), *r* (read, olvasott), *u* (unread, nem olvasott, de kilistázott)).

Leggyakrabban a következő parancsokat szokták használni (zárójelben a parancs esetleges rövidítését adjuk meg):

```
! ShellParancs
 Végrehajta a ShellParancsot.

shell (sh)
 Meghívja a shell programot.

?
 Röviden kilistázza a parancsokat.

list (l)
 Röviden kilistázza a mailbox utasításokat.

chdir Könyvtár (cd)
 Az aktuális könyvtár a Könyvtár lesz.

from [ÜzenetLista] (f)
 Kiírja az ÜzenetLista fejléceinek tartalmát.

z [-]
 Kilistázza az éppen aktuális levél körüli 20 levelet. Argumentum nélkül
 a következő 20 levélre ugrik, míg - argumentummal visszafelé mozog 20
 levélnyit.

top [ÜzenetLista]
 Kiírja az ÜzenetListaban található levelek első 5 sorát.

size [ÜzenetLista]
 Kiírja az ÜzenetListaban található levelek méretét.

next [Üzenet] (n)
 A következő levelet olvassa be, és írja ki a képernyőre.

print [ÜzenetLista] (p)
 Kiírja az ÜzenetLista tartalmát a képernyőre.

Print [ÜzenetLista] (p)
 Kiírja az ÜzenetLista tartalmát a képernyőre, az összes fejléc-mező tar-
 talmával együtt.

more [ÜzenetLista] (mo)
 Kiírja az ÜzenetListan található leveleket a képernyőre laponként az előre
 definiált lapozó program (alap esetben a more) segítségével.
```

`edit [ÜzenetLista] (e)`

Az `EDITOR` változóval jelzett editort (alapesetben `ed`) használva szerkesztésre átadja a leveleket.

`visual [ÜzenetLista] (v)`

A `VISUAL` változóval jelzett editort (alapesetben `ed`) használva szerkesztésre átadja a leveleket.

`copy [ÜzenetLista] Állomány (c, co)`

Az `ÜzenetLista` minden levelét hozzáfűzi az `Állomány`hoz, de nem jelzi ezt a leveleken.

`save [ÜzenetLista] Állomány (s)`

Az `ÜzenetLista` minden levelét hozzáfűzi az `Állomány`hoz, és kitörli azokat. `Állomány` megadása nélkül az `mbox` file-ra hivatkozik.

`delete [ÜzenetLista] (d)`

Az `ÜzenetLista` leveleit kijelöli törlésre. A `mail` parancsból való kilépés előtt még használhatjuk az `undelete` parancsot a levelek visszaszerzésére.

`undelete [ÜzenetLista] (u)`

Az `ÜzenetLista` leveleinek törlési jelölését megszünteti. Lista megadása nélkül az utolsó kitörölt levelet hozza vissza.

`exit (ex vagy x)`

Kilép a `mail` programból, és a levelesláda állapotát az eredeti tartalomra állítja vissza (pl. nem törli a `delete` paranccsal kijelölt leveleket).

`quit (q)`

Kilép a `mail` programból, és a levelesláda állapotát az új, módosított értékekre (tartalomra) állítja be. Törli a `delete` paranccsal kijelölt leveleket, módosítja a rendszer levelesládát és az `mbox` állományt.

`folder [ÁllományNév] (fo)`

Átkapcsol az `ÁllományNév` folderre, miután tárolta az éppen aktuálisat. Argumentum nélkül kiírja az aktuális folder értékét. Az állománynév megadásánál hivatkozhatunk az előző `Állomány`ra a `#`, a rendszer levelesládára a `%` (ez a `/usr/spool/mail/FelhasználóiAzonosító file`), a személyi levelesládára az `&` (ez a `$HOME/mbox file`), és a folder könyvtár egy tagjára a `+Folder` rövidítésekkel.

`folders`

Kilistázza a folder könyvtár tartalmát.


`touch [ÜzenetLista]`

A rendszer levelesládából átmozgatja a személyi levelesládába az `ÜzenetListát`.

mail *CímzettLista* (m)

Elindítja a levélküldő programot a *CímzettListán* szereplő felhasználóknak címezve.

alias (a)

Argumentum nélkül kiírja az éppen felismert összefoglaló neveket. Az  & *alias kerekasztal lancelet artur@@camelot galahad sirrobin* parancs kiadása után a *kerekasztal* címzés minden felsorolt felhasználónak elküldi a levelet.

reply [*Üzenet*] (r)

Válaszolhatunk az *Üzenet*et küldő felhasználónak.

Reply [*Üzenet*]

Válaszolhatunk az *Üzenet*et küldő felhasználónak. Az *Üzenet* minden eredeti címzettje szintén megkapja a válaszlevelet.

source *Állomány* (so)

Beolvassa és végrehajtja az *Állományban* található mailbox parancsokat.

ignore *Kulcsszó*

A levél fejrészében megjelenő *Kulcsszóval* megadott sorokat nem jelzi ki.

set [*Opció* [=Érték]] (se)

Argumentum nélkül kiírja az összes beállított opciót. Egy adott *Opció* megadásakor annak értékét állítja be. Egy kétállapotú, a *set*tel beállított opciót az *unset* paranccsal törölhetünk. A leggyakrabban használt változók:

append

Az új leveleket az *mbox* végéhez illeszti az alapértelmezésbeli kezdet helyett.

askcc , askbcc , asksub

Beállítása esetén a program megkérdezi a megfelelő fejléc-mezőt.

folder=*Könyvtár*

A *mail* alapkönyvtárának helyét definiálja. A parancs kiadása után az összes *+Állomány* típusú kifejezést a *mail* a *Könyvtár/Állomány* alakra fordítja.

ignore

Letiltja az *Interruptot* (megszakítást). Zajos vonal használatakor hasznos.

indentprefix=String

A *~m* parancs által használatos stringet állítja be. Alapértelmezésben ez a *TAB*.

`outfolder`

A kimenő levelekről készülő másolat állományokat a folder könyvtárba helyezi.

`record=ÁllományNév`

Az összes kimenő levélről másolatot készít az *ÁllományNév* állományba.

`showto`

A fejléc kiírása esetén, ha a levél tőlünk ered, a címzett nevét írja ki a feladó neve helyett.

`sign=Aláírás` Az aláírás string értékét állítja be, amit aztán a `~a` parancssal hívhatunk be.

#### Példák:

Ha nem akarjuk látni a levelek listázásakor a `Received` mezőt, adjuk a következő sort a `$HOME/.mailrc` állományhoz:

```
ignore received
```

A `$HOME/levelek` folder könyvtár beállítását pl. a következő utasítással végezhetjük el a `$HOME/.mailrc` állományban:

```
set folder=$HOME/levelek
```

A 2. számú levelet a `gral` folderbe a `& s 2 +gral` paranccsal menthetjük el.

A kimenő levelekről automatikusan másolatot a `$HOME/kimeno` folderbe a `& set record=$HOME/kimeno` `.mailrc` parancssorokkal kaphatunk.

A `kimeno` folder tartalmát a `$ mail -f +kimeno` shell paranccsossal olvashatjuk el.

A `/usr/lib/Mail.rc` állományban beállított



```
set askcc
parancs hatását (alapértelmezés) egy
unset askcc
paranccsal kűszöbölhetjük ki a .mailrc állományban.
```

Nagyobb csoportokat az `alias` paranccsal a következőképpen definiálhatunk a `.mailrc` állományban:

```
& alias kerekasztal lancelet artur@@camelot galahad sirrobin
& alias wizard @mordor.UUCP:gandalf@hobbit
& alias wizards merlin@camelot oz wizard
```


Ha ezek után a `$ mail wizards <level` paranccsal küldünk el egy `level` állományt, azt a `mail` a `merlin@@camelot`, az `oz` és a `@mordor.UUCP:gandalf@hobbit` felhasználóknak küldi el.



# 16. fejezet

## Az `elm` részletes használata

Az `elm` mailer nem a UNIX operációs rendszer része, viszont a beépített levelezőrendszerénél kényelmesebb felületet nyújt a felhasználó számára. Tekintve, hogy a program nem része az operációs rendszernek, csak azokon gépeken elérhető, ahol azt a rendszergazda installálta. Az `elm` alapszintű használatát a 8.2.2 fejezetben ismertettük.

 `elm`  
rövid leírása  
a 68. oldalon

Az `elm` a `$HOME/.elm/elmrc` állományon keresztül programozható. A program működik ezen állomány nélkül is, de célszerű a rendszergazdától kérni egy mintát, és azt kitölteni igényeink szerint.

Az `elm` első használatakor létre akarja hozni a `.elm` könyvtárat, és a következő felirat jelenik meg<sup>1</sup>:

```
Notice:
This version of elm requires the use of a .elm directory in your home
directory to store elmrc and alias files. Shall I create the
directory .elm for you and set it up (y/n)? y
```

Nyomjuk le a `Enter` gombot, mire a

```
Great! I'll do it now
```

```
Notice:
elm require the use of folders directory to store your mail folders in.
Shall I create the directory $HOME/Mail for you (y/n)? y
```

szöveg jelenik meg.

Nyomjuk le ismételten az `Enter` gombot, és ezzel bejutunk a főmenübe. Ezt az eljárást csak egyszer kell megcsinálni, a program későbbi használata során

---

<sup>1</sup>Az `elm` újabb verzióiban a szöveg kissé megváltozhat! Ez igaz az itt felsorolt összes példára is.

egyből a főmenübe jutunk. FIGYELEM! Amennyiben a fenti kérdéseknél `no`-t választunk, akkor az `elm` nem végzi el a file-struktúra kialakítását a későbbi használata során sem! Ilyenkor erről a felhasználónak kell gondoskodnia.

## 16.1. Az `elm` alaputasításai

Az `elm`-be belépve a képernyő alján rövid segítséget ad a program a legfőbb, az adott menüben használható utasításokról. Hogy itt pontosan mi jelenik meg, az az `elm` verziószámán kívül függ attól is, hogy a már említett `elmrC` konfigurációs file-ban kezdő, középszintű, vagy haladó felhasználónak minősítettük magunkat.

Minden, az aktuális postaládában levő levélről egy egysoros bejegyzést láthatunk a képernyőn. Ha sok levelünk van, nem látjuk egyszerre az összeset, de a kurzor mozgatásával minden levelet elérhetünk. A bejegyzés első betűje a következőt jelenti:

- E lejárt (bizonyos időnél régebbi) levél
- N új levél
- O régi, olvasatlan levél
- D törölt levél
- U sürgős levél

Ezt követi a levél sorszáma, amivel hivatkozhatunk rá, a levél beérkezésének dátuma (hónap, nap), a feladó neve vagy E-mail címe, a levél hossza sorokban, és a levél tárgya.

A képernyőn levő levelek között egy mindig az aktuális levél. Ezt vagy az különbözteti meg a többitől, hogy inverzben látszik, vagy van előtte egy nyíl: `-->` (beállítás kérdése). A levelek között a `j` gomb segítségével lefelé, a `k` gomb segítségével felfelé mozoghatunk (lehet, hogy a billentyűzeten levő nyilak is működnek). Hivatkozhatunk egy levélre a sorszámaival is, amennyiben számot kezdünk el beütni, az `elm` egyből azt gondolja, hogy egy sorszámot akarunk megadni, és a

Command: `New Current Message`                      `Set current message to:`  
választ jeleníti meg. Gépeljük be a számot, és `Enter`-rel az `elm` a megadott számú levélre ugrik. Ismételt `Enter`-re a levél tartalmát megjeleníti a képernyőn. Gyors mozgásra szolgál az `=` az első levélre és a `*` az utolsó levélre ugrás parancsa.

Levelet kiválaszthatunk valamilyen minta alapján is. A főmenüből kiadott `/` parancs hatására az `elm` egy mintát vár. Gépeljük be a keresett szöveget, és az

`Enter` lenyomására az első olyan levélre áll rá a rendszer, melynek a feladó vagy a tárgy mezőjében szerepel a megadott minta. A `//` parancs hatására a keresés a teljes levélben megtörténik, így például ha gyorsan meg akarjuk találni azt a levelet, amiben a fizetésemelésünkről értesítettek minket, a `//fizetes` parancs rááll a keresett levélre.

A már ismertett `m` (mail) főmenübeli levélküldési parancson túl az `elm` képes válaszolni is egy levélre, és azt továbbítani. Ha elolvastunk egy levelet, akkor az `r` (reply) paranccsal válaszolhatunk rá: az `elm` automatikusan létrehozza az aktuális levél alapján a (To:) címet, és a tárgy mezőbe a

Re: *AktuálisLevélTárgya*

kerül, amit átírhathatunk. Ezután az eredeti levelet beolvassa a szövegszerkesztőbe, és minden sor elé nagyobb jelet tesz (ezt meg lehet változtatni). A levél elé beír egy általunk megadott szöveget (ez `Reply to FeladóCíme` alapértelmezésben).

Ha az főmenüben az `f` (forward) parancsot adjuk ki, akkor a rendszer az `m` (mail)-hez hasonlóan megkérdezi, hogy kinek szeretnénk a levelet elküldeni, és az aktuális levél tárgya alapján automatikusan felajánlja a továbbított levél tárgyát (alapértelmezésben ez: *EredetiTárgy (fwd)* ), majd megkérdezi, hogy meg szeretnénk-e szerkeszteni az eredeti levelet:

Do you want to edit the message (y/n) `y`

Ha `y`-t válaszolunk, akkor az aktuális levelet beolvassa a szövegszerkesztőbe, és minden sor elé `>` jelet tesz. A továbbított levél feladója azonban már mi leszünk, tehát ha az, akinek továbbítottuk a levelet `r` (reply)-jal válaszol, a választ mi kapjuk. A `b` (bounce) paranccsal továbbíthatunk úgy leveleket, hogy annak feladó mezője ne változzon.

A főmenüben a `d` (delete) paranccsal törölhetjük az aktuális levelet. Valójában ilyenkor csak kijelöltük törlésre a levelet, fizikailag csak akkor törlődik, amikor kilépünk vagy foldert váltunk. A törlésre kijelölt levél előtt egy `D` betű jelenik meg, és amikor fel- vagy lefelé mozgunk a levelek között, az `elm` átugorja ezeket a leveleket. Ha mégis egy törlésre kijelölt levélre akarunk lépni, akkor sorszám szerint tehetjük meg. Ilyenkor az `u` (undelete) paranccsal visszavonhatjuk törlési szándékunkat.

Az `elm` sokféle más parancsáról angol nyelvű segítséget kaphatunk a `?` parancs segítségével.

## 16.2. A postaládák

A levelek csoportosítására a postaládák (folderek) szolgálnak. Az `elm`ben három postaláda generálódik automatikusan: az új levelek ládája, melyre a `!`-jel, a régi levelek (received) ládája, melyre a `>`-bal, és az elküldött levelek (send) ládája, melyre a `<`-bel hivatkozhatunk. Ezenkívül mi is megadhatunk postaládákat. A postaláda egy egyszerű file, melyben egymás után sorakoznak a levelek, és minden levél a `From: Feladó` sorral kezdődik. Ha egy levelet a főmenüből az `s` (save) paranccsal mentünk ki, akkor az `elm` megkérdezi a file (folder) nevét, ahova a levelet betegyje, és a mentés végén a levelet automatikusan törlésre jelöli ki (abból a postaládából csak, ahol éppen vagyunk). Ha a megadott név alatt már létezett file, akkor ahhoz hozzáírja az új levelet, ha nem létezett, akkor létrehozza az állományt. Ezáltal tulajdonképpen a folder és a file fogalma azonosnak tekinthető. A postaládának adott név az aktuális könyvtárban hoz létre egy ilyen nevű file-t; ha a megadott név legelső karaktere az `=`, akkor a `$HOME/Mail` könyvtárban (ez az a könyvtár, amit az `elm` az első használatkor létrehoz) jön létre a file, ami gyűjti a leveleket. Az `elm` automatikusan ilyen `=`-vel ellátott nevet ajánl fel, pl.: `=laca`, ha a feladóra beállítottuk a `laca` becenevet.

A postaládák használatával csoportosíthatjuk az olvasott leveleket. Az új levél elolvasása után, ha azt nem tettük át egy másik postaládába, kilépéskor az `elm` automatikusan felajánlja, hogy az olvasott leveleket a régi levelek postaládába teszi át (ez a `$HOME/Mail/received` állomány).

A levelezőrendszeren belül a főmenüben a `c` (change) paranccsal válthatunk postaládát. A ládára a nevével kell hivatkozni (a három fő ládára a `!`, `>`, `<` jellel). Az `elm` beállításától függően elképzelhető, hogy az `elm` nem generál automatikusan másolatot a kimenő levelekről, és így a kimenő postaláda mindig üres. Az `elmr`c állomány programozásával rávehetjük az `elm` levelezőrendszert, hogy tegyen másodpéldányt a kimenő postaládába.

Amennyiben levelezési címünk megváltozna, a leveleket automatikusan továbbküldhetjük az új címre ugyanúgy, mintha a UNIX beépített levelezőrendszerét, a `mailt` használnánk: a `$HOME/.forward` állományba be kell írni az új címünket. Az `elm` és a `mail` csak két különböző, az operációs rendszer alap levelezőszolgáltatását használó program, így ugyanonnan veszik az új leveleket. Ezáltal elképzelhető, hogy valaki felváltva használja őket. A különbség abban van, hogy hol tárolják a régi leveleket. Ha a UNIX `mail`ből átállunk az `elmr`e, akkor a régi leveleket a `$HOME/mb`ox postaládából kell elővenni.

## 16.3. Aliasok

A hosszú és bonyolult címek használata helyett hozzájuk rendelhetünk rövid, és kifejező "beceneveket" (alias), melyeket könnyen megjegyzünk, és a levelek küldésénél a `TO:` (címezett) mezőben ezeket is használhatjuk. Mindenki írhat magának felhasználói beceneveket (user alias). A rendszergazda ezen kívül definiálhat még mindenki számára elérhető, rendszerszintű neveket (system alias). Az főmenüből az `a` paranccsal léphetünk az alias menübe:

```
a)lias current message, d)elete an alias, check a p)erson or s)ystem,
 l)ist existing aliases, m)ake new alias or r)eturn
```

Alias:

Becenevet az `m` (make) paranccsal hozhatunk létre. Az alias rendszer először a becenevet kéri (alias, ennek egyedinek kell lennie, nem adhatunk új értéket már létező becenévnek, ahhoz azt előbb ki kell törölni), majd a felhasználó nevét, és végül levelezési címét. Ezáltal egy definíció a következő alakú:

*becenév = (Teljes) Név = cím(ek)*

Címként akár több címet, vagy már előzőleg definiált becenevet is megadhatunk<sup>2</sup>:

```
ami = Clinton = president@whitehouse.gov
rus = Boris Jelcin = boris@kreml.kremlvax.su
elnokok = Fejesek = ami, rus, tseng@beijing.china
```

A levél küldésénél is felsorolhatunk több, vesszővel elválasztott címet. Ha gyakran írunk több embernek, érdemes belőlük az `elnokok`-höz hasonló csoportot kialakítani, es egy névvel lehet mindenkinek levelet küldeni. Egy becenév több csoporthoz is tartozhat. A becenevek megőrződnek, tehát ha legközelebb bejelentkezünk, használhatjuk őket.

A már definiált beceneveket az `l` (list) paranccsal írathatjuk ki. A felesleges, vagy módosításra szoruló neveket a `d` (delete) paranccsal törölhetjük a becenév alapján.

Igen hasznos parancs az `a` (alias), mely lehetővé teszi, hogy az aktuális levél *Feladó*jának címéhez rendeljünk becenevet anélkül, hogy meg kellene jegyezni a címét. A rendszer ilyenkor a levél alapján automatikusan betölti a címet, és csak a hozzá tartozó becenevet kérdezi meg.

Az alias menüből az `r` (return) paranccsal léphetünk vissza a főmenübe.

---

<sup>2</sup>A legelső cím valóban működik!

## 16.4. A konfigurációs file (`elmrc`)

Az `elm` levelezőrendszer konfigurációs állománya a `$HOME/.elm/elmrc`. Ezen keresztül minden felhasználó "testre szabhatja" magának a rendszert. Az állomány egyszerű, olvasható file, melyben különböző változók értékeit állíthatjuk a *Változó=Érték* bejegyzéssel. Itt csak a legfontosabbak szerepét ismertetjük. A file-ban a # utáni rész megjegyzés, azt a rendszer nem veszi figyelembe:

- `editor` a használt szövegszerkesztőt állítja be. Amennyiben pl az EMACS-ot akarjuk használni az  
`editor=TeljesÚtvonal/emacs`  
 sort kell beírni. A `none` vagy `builtin` értékek a `mail`hez hasonló "szerkesztőt" nyújtanak. Amennyiben nem adunk meg semmit, akkor az `EDITOR` környezeti változó értékét, annak hiányában a `vi` editort használja.
- `fullname` A teljes felhasználói nevünk. Ha nem adjuk meg a `/etc/passwd` file-ban levő információt használja.
- `maildir` A postaládákat tartalmazó könyvtár. Ennek az alapértéke a `$HOME/Mail`.
- `pager` A levél olvasásakor alkalmazott lapozóprogram. Ha nem adjuk meg, a `PAGER` környezeti változó értékét használja, annak hiányában a beépített `builtin+lapozót`.
- `prefix` A levélre való válaszoláskor vagy továbbításkor az eredeti levél minden sora elé kerülő karakter (alapérték: "> ")
- `print` A levél kinyomtatását szabályozó parancs. A levelet tartalmazó állományra a `%s`-sel hivatkozhatunk. Alapértéke: `lpr %s`.
- `signature` Az elküldött levelekben a levél végéhez automatikusan hozzáírandó "aláírást" tartalmazó állomány nevét adja meg.
- `arrow` Ha értéke `ON`, akkor az aktuális levelet a `-->` nyíl jelöli, ha nem adjuk meg az értékét, vagy az `OFF`, akkor inverzben jelenik meg az aktuális sor.
- `askcc` `OFF` esetén nem kérdezi meg a `Cc: (Carbon Copy)` mezőt. Alapértéke `ON`.
- `autocopy` `ON` esetén automatikusan bemásolja az `r (reply)`-jal megválaszolt levelek esetén a szövegszerkesztőbe az eredeti levelet. Alapértéke `OFF`.
- `copy` `ON` esetén automatikusan másolatot csinál minden kimenő levélről. Alapértéke `OFF`.

A többi változó az `elm`hez adott minta-konfigurációs állományban angol nyelvű magyarázatokkal megtalálható.



# 17. fejezet

## Hálózati alapismeretek és egyéb programok

### 17.1. TCP/IP alapok

A TCP/IP a UNIX rendszerek által leggyakrabban használt kommunikációs protokoll, melynek segítségével a számítógépek erőforrásait megoszthatják egymással hálózaton keresztül.

A leggyakrabban a következő TCP/IP szolgáltatásokat használják a számítógépek között:

- File átvitel, amire a File Transfer Protocol (`ftp`) szolgál.
- Távoli bejelentkezés, amit a Network Terminal Protocol (`telnet`) tesz lehetővé.
- Elektronikus levelezés, ami lehetővé teszi üzenetek küldését más számítógépekre is.
- Network file rendszerek (NFS,RFS,AFS), amelyek lehetővé teszik egész file rendszerek elérését más gépek számára. A file rendszert a távoli gépek saját operációs rendszerükbe beillesztve használhatják.
- Távoli nyomtatás, ami lehetővé teszi más gépekhez kötött nyomtatók használatát. Segítségével egy adott felhasználói csoport relatíve kevés nyomtatóval is kiszolgálható.
- Távoli programfuttatás (RPC). Segítségével lehetővé válik programok futtatása a távoli gépen. Akkor hasznos, ha csak egy programot akarunk más gépen futtatni, vagy ha a másik géphez kötött eszközt szeretnénk közvetlenül a gépünkről elérni (pl. backup készítéséhez).

- Name szerverek, amelyek az egyedi gépek neveit és címeit tartalmazzák egy intézményen belül. Segítségével nem szükséges minden host címét a helyi gépen tárolni, az a központi szerverről elérhető.
- Terminál szerverek, amelyek speciális célszámítógépek, ahonnan távoli gépekre jelentkezhetünk be. A terminálok ezekhez az (általában Ethernet) hálózatra kötött eszközökhöz tartoznak, így nem szükséges egy hosthoz jelentkezni csak azért, hogy egy másik gépet `telnet`tal elérjünk.
- Network-alapú window rendszerek, mint pl. az X11 rendszer.

### 17.1.1. A TCP/IP protokollok általános felépítése

A TCP/IP protokollt több egymásra épülő protokoll alkotja. Áltában egy TCP/IP alkalmazás 4 fő rétegből áll:

- Applikációs protokoll, mint pl. az `ftp`.
- Egy olyan protokoll, mint pl. a TCP, amely az applikáció számára különböző szolgáltatásokat biztosít.
- IP, amelyik az alapvető szolgáltatás az ún. *datagram*ok átviteléhez.
- A fizikai hordozót kezelő felület, amelyik pl. az Ethernetet vagy a soros vonalat hajtja meg.

A TCP/IP az ún. *catenet* modellt használja. Ez feltételezi, hogy sok egymástól független hálózat van összekötve egymással gateway-ekkel. A csomagok ekkor több különböző hálózaton is keresztülmehetnek, mielőtt célhoz érnének. Ez a folyamat a routing, ami a felhasználó számára teljesen láthatatlan. A csomag a célgép Internet címével van ellátva, ami egyértelműen megadja a végcél. Az Internet cím egy 32-bites szám, amit gyakran 4 decimális számmal jelölnek, mint pl. a `146.113.42.42`. A felhasználók általában nem a címet, hanem a hozzárendelt szimbolikus címet (pl. `vogon.shi.edu`) használják. Az egyes gépeken futó eljárások külön-külön is kiépíthetnek TCP/IP kapcsolatot, ezeket egy Internet címen belül az ún. portszámmal különböztetjük meg. Az Internet cím és a portszám együtt egyedi kombinációt alkot. Például:

|              | Internet cím |           | TCP portok |    |
|--------------|--------------|-----------|------------|----|
| 1. kapcsolat | 128.6.4.194  | 128.6.4.7 | 1234       | 21 |
| 2. kapcsolat | 128.6.4.194  | 128.6.4.7 | 1235       | 21 |

A TCP/IP az ún. "összeköttetésmentes" technológián alapul, amelyben az információ ún. *datagram*ban terjed. A datagram (csomag) az egy üzenetben elküldött adatok összessége. Minden datagram a hálózaton egyedi módon terjed:

a hálózati összeköttetés függvényében a futási idők különbözők lehetnek, azaz egy később elküldött datagram megelőzheti egy korábban elküldöttet. Ugyanígy lehetséges, hogy valamilyen hiba folytán egy datagram elveszik, amit újra kell küldeni.

### 17.1.2. A TCP szint

A TCP/IP csomagok továbbítására két protokoll, a TCP és az IP szolgál. A TCP (Transmission Control Protocol) végzi el az üzenetek feldarabolását csomagokra az egyik oldalon, míg a másik oldalon a beérkező datagramokból összerakja az eredeti üzenetet. Ez a szint kezeli a esetlegesen elvesző csomagok újrakérését és a sorrendváltást.

Az IP (Internet Protocol) az egyedi datagramok továbbításáért felelős. Ugyan úgy tűnhet, hogy a TCP végzi el a feladat nagy részét, de ez csak kis hálózatok esetén van így. Az Internet egy nagy hálózat, és a datagramok célhoz juttatása nagyon bonyolult lehet, amennyiben több hálózaton kell azt keresztülküldeni. Az összes kapcsolat és a különböző vonalak (esetlegesen különböző fizikai hordozókon) kezelése komplex feladat, amit az IP szintnek kell elvégeznie.

Példa: Egy `ftp` processz a következő adatfolyamot kívánja a távoli gépen futó eljárásnak átküldeni (ez lehet pl. egy 1 Mbyte-os file):

.....

A TCP ezt az adatsort kezelhető darabokra szaggatja (ez illeszkedik a hálózat és a szemközti program által kezelhető maximális méretű csomaghoz):

.....

A TCP minden datagram elejére egy header részt illeszt, ami tartalmazza a forrás- és célprocessz port címét és az ún. sorozatszámot. A szemközti TCP program ezek ismeretében állítja vissza az adatfolyamot, a sorozatszám alapján eldönti, hogy vezett-e el csomag, stb. A sorozatszám nem a csomagokat, hanem a csomagokban lévő oktetteket számozza. A TCP ezek mellé még egy ún. checksumot adva hozza létre a TCP headert ( $T$ ). Az adatok tehát már így néznek ki:

T..... T..... T..... T..... T..... T..... T.....

A TCP az IP szintnek adja át az adatokat a cél Internet címével együtt. Az IP ebből és a helyi Internet címből (I) képez egy csomagot, amit egy újabb checksummal ellátva ad tovább. Az adatok ekkor már így néznek ki:

```
IT.... IT.... IT.... IT.... IT.... IT....
```

ahol az I jelenti az IP headert. A fizikai szint különböző lehet, lehet soros vonal, X25, de a leggyakoribb az Ethernet. Az Ethernet saját 6 byte-os címekkel rendelkező fizikai eszköz, amely több protokollt is képes (akár egyszerre is) használni (pl. TCP/IP, DECnet, Novell, stb.). Az Ethernet saját E headerjét (binnen a két Ethernet címmel) és C trailerjét (checksummal) illeszti az előző IP csomagokhoz:

```
EIT....C EIT....C EIT....C EIT....C EIT....C
```

Természetesen a fogadó oldal ezeket a konverziókat visszafelé végzi el. Az Ethernet interface levágja az Ethernet headert és a checksumot. Amennyiben a csomag IP típusú, akkor az Ethernet interface azt az IP-nek adja tovább, ami a protokoll mezőt figyeli. Ha protokoll mező TCP típusú, akkor az IP az IP headert kitörölve átadja a TCP-nek, ami a sorozatszám alapján visszaállítja az eredeti adatfolyamot.

Az Ethernet hálózat a csomagokat egy közös (általában koax) hordozóra helyezi. Mivel ez busz rendszer, előfordulhat csomagok ütközése, amikor is a rendszer bizonyos véletlenszerűen megadott idő múlva újra megkísérli a csomag elküldését. Az időzítések miatt a rendszer nagyon érzékeny a maximális futási időre. A hasznos sáv szélesség ugyan csak egy része a teljes fizikai sáv szélességnek, de az egyszerűség és flexibilitás kompenzálja ezt a hátrányt.

### 17.1.3. Socketek és applikációk

A felhasználói applikációk a TCP szint felett futnak. Ezek a programok a TCP/IP-t mint összekötetési felületet használják, és - a felhasználókhöz hasonlóan - nem kell tudniuk a mélyebben fekvő rétegekről (IP, Ethernet).

Az egyes programok egységes használata érdekében ún. socket számokat használnak a különböző gépek az Internet hálózaton. Ezt azt jelenti, hogy míg a hívó gép (amely pl. egy ftp programot akar futtatni egy távoli gépen) a saját ftp client programjához egy véletlen, de egyedi portszámot rendel, a szemközti gépen a 21-es portot (az ftp socketjét) adja meg célként. A kapcsolat így egyértelműen

meghatározott (két különböző kapcsolatnál a két Internet és a két portszámból legalább egy különböző).

Az egyes applikációk a hibamentes TCP/IP csatornán további saját protokollokat használhatnak, mint pl. az SMTP levelezés:

```
RED 220 RED.VOGON.EDU SMTP Service at 29 Jun 92 05:17:18 EDT
TOPAZ HELO topaz.vogon.edu
RED 250 RED.VOGON.EDU - Hello, TOPAZ.VOGON.EDU
TOPAZ mail From:<zaphod@topaz.vogon.edu>
RED 250 mail accepted
TOPAZ RCPT To:<civita@red.vogon.edu>
RED 250 Recipient accepted
TOPAZ DATA
RED 354 Start mail input; end with <CRLF>.<CRLF>
TOPAZ Date: Sat, 27 Jun 92 13:26:31 EDT
TOPAZ From: zaphod@topaz.vogon.edu
TOPAZ To: civita@red.vogon.edu
TOPAZ Subject: meeting
TOPAZ
TOPAZ Talalkozo holnap 1-kor a Vilag Vegen. Egy Pangalaktikus
TOPAZ gegepukkasztot mar rendeltem...
TOPAZ .
RED 250 OK
TOPAZ QUIT
RED 221 RED.VOGON.EDU Service closing transmission channel
```

Figyeljük meg, hogy a parancsok csak "normális" szöveget tartalmaznak, ami tipikus az Internet szabványokban (ezek az ún. RFC ajánlások, amik *de facto* szabványok). A parancsokat általában egy szám jelenti, ami után a szöveg csak az esetleges debuggolást segíti. A számokat a különböző Internet szabványok rögzítik, általában a 2-vel kezdődőek sikeres lefutást jeleznek, a 3 további adatot kér, a 4 és 5 pedig "ideiglenes" ill. végzetes hibát jelent.

#### 17.1.4. Egyéb protokollok: UDP, ICMP és ARP

Bizonyos alkalmazások esetén nem szükséges a TCP szintet használni, mivel az üzenet belefér egy datagramba. Ilyen lehet pl. a név szerinti Internet cím keresés. Ilyen típusú célokra az UDP ("User Datagram Protocol") használatos. Az UDP a TCP-hez hasonlóan saját port címeikkel rendelkezik, és a saját headerjével ellátott csomagot adja át az IP-nek továbbításra. Az IP a távoli gépen nem a TCP, hanem az UDP protokoll kezelő programnak továbbítja a csomagot.

Egy másik protokoll az ICMP ("Internet Control Message Protocol"), amit maga a TCP/IP software használ hiba- és egyéb saját üzenetek továbbítására. Az ICMP még egyszerűbb, mint az UDP, mivel nincsenek saját portcímei.

A harmadik említésre méltó protokoll az Ethernet által használt ARP ("Address Resolution Protocol"), amely szigorúan véve nem IP protokoll. A kapcsolat kiépítésekor feltétlenül szükség van az Internet cím alapján a távoli gép Ethernet címére (ez kell a kapcsolat kiépítéséhez). Amennyiben a helyi ARP táblában ez az információ nincs meg, akkor ez lesz az, amit az ARP elvégez: egy speciális csomagban szétküldi az Internet címet a hálózaton. Minden gép figyel az ARP csomagokat, és ha látja, hogy őt szólították meg, válaszol. Például az ARP 128.6.4.7 kérésre az 128.6.4.7 Internet című gép visszaküldi saját Ethernet címét (pl. 8:0:20:1:56:34). Ez az információ a helyi gép ARP cache tárolójába kerül, elkerülendő a gyakori ARP kéréseket. Az ARP kérések ún. broadcast kérések, amiket az ff:ff:ff:ff:ff:ff Ethernet címre küld az ARP.

### 17.1.5. Repeaterek, bridge-ek és routerek

Az Ethernet hálózat fizikai mérete három ok miatt maximált: a jel csillapodása és a futási idők limitálják a maximális távolságot, a véletlenszerű csomagküldés pedig a forgalom növekedése miatt a maximálisan üzemeltethető berendezések számát limitálja. A jel csillapításán ún. repeaterek használatával segíthetünk, amelyek egyszerű jelerősítők. Nagyobb távolságoknál a futási idők túlságosan nagyok lesznek, ezért ilyenkor a csomagok ütközését tárolás/továbbítás módszerrel kell elkerülni két távoli szegmens között. Erre a célra szolgálnak az ún. bridge-ek, amelyek ezen túlmenően csak a két szegmens közötti forgalmat engedik át, s így a helyi forgalom nem zavar más szegmenseket.

Az IP szint feladata a megadott datagram eljuttatása a cél címre. Ezt a feladatot routingnak hívjuk. A routing teljes egészében az Internet címen alapul. Éppen ezért a helyes routing megkönnyítése érdekében egy speciális struktúrát, az ún. domain struktúrát alakították ki az Interneten.

A domain struktúra az jelenti, hogy különböző Internet címtartományokhoz különböző maximális címtartomány kapcsolódik. Az A osztályú címek az első byte-ban a 1-127-es számot tartalmazzák, és  $256 \times 256 \times 256$  gépre kiterjedő hálózatot tennének lehetővé. Általában a B osztályú címeket használják a nagyobb intézmények, ahol  $256 \times 256$  gépet lehet így hálózatba kötni. A B osztálynak a 128.1 - 191.254-es Internet prefix tartomány felel meg. Kisebb hálózatok esetén elegendő a 256 címmel ellátott C osztályú Internet címtartomány a 192.1.1 - 223.254.254-es tartományban. Megjegyzendő, hogy az oktettekben a 0 és 255 értéket nem használják.

A 0 és a 255 különleges jelentést hordoz: a 0 azoknak a gépeknek a címét jelenti, amelyek nem tudják saját Internet címüket. A 255-as értéket ún. broad-

cast célokra használják, amikor is a (helyi) hálózaton minden gép figyeli az így megcímezett üzenetet. Pl. a 128.6.4 hálózaton a 128.6.4.255 értéket kell broadcast célokra használni. Általában lehet a 255.255.255.255 értéket is használni, a gateway feladata a broadcast helyi szinten tartása. A broadcastot a netmask megadásával is helyi szinten tarthatjuk. Egy adott osztály esetén a netmask értékét egyszerűen a fix Internet címtartomány 1-esre, míg a változó címtartomány 0-ra állításával kaphatjuk meg. Pl. a 194.43.42 hálózaton 255.255.255.0 lesz a netmask.

A legalacsonyabb szint a helyi hálózat: amennyiben az IP látja, hogy a datagramot a helyi hálózat egyik gépének címezték, akkor közvetlenül kiküldi. Amennyiben a cím nem érhető el közvetlenül, akkor egy, a helyi hálózathoz kapcsolt ún. gateway gépnek küldi el a datagramot. Általában a helyi hálózatokhoz csak egy router van illesztve, de a routerek képesek - a cím ismeretében - egymás között továbbítani a datagramot.

## 17.2. Az `rlogin` és `rsh` utasítások

Néha a `telnet` parancs használatával átjelentkezni egy távoli gépre nem elég kényelmes: például ha több gépen is van felhasználói azonosítónk, és gyakran átlépünk az egyikről a másikra (pl. segédprogramok futtatására), akkor célszerű lehet megadni a célgépen az ún. „megbízható” gépek listáját. Ezekről a gépekről ilyenkor kulcsszó használata nélkül is átléphetünk az adott gépre.

A megbízható gépek listáját kétféleképpen lehet megadni:

- A rendszergazda felsorolhatja a `/etc/hosts.equiv` állományban azokat a gépeket, amelyekről a felhasználók automatikusan használhatják az `rlogin` és `rsh` parancsokat. Ekkor a felhasználói azonosítónak meg kell egyeznie a két gépen.
- Saját címeket is hozzáadhatunk a listához, ha a `$_HOME/.rhosts` állományba soronként beírjuk az engedélyezendő gép nevét és a távoli felhasználói azonosítót (szóközzel elválasztva). Ezt az állományt csak a felhasználó írhatja, máskülönben a rendszer biztonsági okokból nem fogja használni.

Vigyázat! Ha egy felhasználói azonosítónk kulcsszavát valaki megtudja, akkor **minden**, a fenti módszerrel megadott gépünkhöz is hozzáfér!

Megjegyezzük, hogy a NIS (Network Information System, a néhai Yellow Pages) rendszerrel kényelmesebben engedélyeztethető az ilyen típusú hozzáférés

sok (kb. >5 ) gép esetén. A NIS-t elsősorban SUN operációs rendszerek használják, de több BSD alapú rendszerbe is beépítették.

### 17.2.1. Az `rlogin` utasítás

✎ Interaktív átjelentkezésre az `rlogin` utasítás szolgál. A legegyszerűbben a `$ rlogin rs1` utasítással léphetünk át az `rs1` nevű gépre. Természetesen az `rs1` gépben a kiindulási gép és felhasználó azonosítójának szerepelnie kell a fent említett módon.

Amennyiben a két gépen nem egyezik meg az azonosítónk (pl. `zaphod` az `rs1` gépen), akkor az `rs1`-en a `$HOME/.rhosts` állományt kell használnunk, valamint a kiindulási gépen az `$ rlogin rs1 -l zaphod` utasítást kell használni.

A parancs kiadása után a helyi gép megpróbálja a távoli gépet elérni. Sikeres kapcsolatfelvétel esetén megjelenik a távoli gép promptja, mintha terminálról jelentkeznénk be.

### 17.2.2. Az `rsh` utasítás

Nem mindig van szükségünk interaktív használatra, sőt pl. távoli eszközök használatánál célszerűbb egy parancs távoli futtatása. Erre a célra az `rsh` utasítás szolgál. Pl. a `df` parancsot az `rs1` nevű gépen a `$ rsh rs1 df` utasítással futathatjuk le. Természetesen az `rs1` gépben a kiindulási gép és felhasználói azonosítójának szerepelnie kell a bevezetőben említett módon.

Ha a két gépen nem egyezik meg az azonosítónk (pl. `zaphod` az `rs1` gépen), akkor — az `rs1`-en a `$HOME/.rhosts` állományt használva) — a kiindulási gépen az `$ rsh rs1 -l zaphod df` utasítást kell használni.

A parancs kiadása után a helyi gépünk megpróbálja a távoli gépet elérni, majd a távoli gépen az utasítást lefuttatni. Az utasítás különleges értéke, hogy a távoli gépen futó utasítás standard be- és kimenete a helyi gépen van. Így például a `$ rsh rs1 cat gyakorlas > > helyiproba` utasítás az `rs1` gépen található `gyakorlas` állományt a helyi gépen található `helyiproba` állományhoz fűzi.



Ha az `rs1` gépen található `gyakorlas` állományt az `rs1` gépen található `tavoliproba` állományhoz akarjuk fűzni, akkor a shell metakaraktereit macskakörmökkel le kell védenünk:

```
$ rsh rs1 cat gyakorlas "> >" tavoliproba
```

Megjegyezzük, hogy C-shell (`csh`) használatakor az `rsh` utasítás néha hibüzenettel tér vissza.

### 17.3. Anonymous<sub>ftp</sub> használata

A UNIX terjedésével egyre több és több olyan programot írnak, melyeket forráskóddal együtt, díjmentesen terjesztenek. Ezeket az ún. *public domain* programokat ugyan nem lehet árulni, de bárki szabadon használhatja őket. Ilyenek többek között pl. a GNU fejlesztés, amely egy teljes, néha a „professzionális” fejlesztőrendszereknél is jobb programokat (C, C++, Fortran-C fordító, debugger, UNIX segédprogramok, grafikus programok, stb.) tartalmazó szoftverrendszer.

Az Internet ragyogó lehetőséget biztosít az ilyen típusú programterjesztésre az ún. anonymous<sub>ftp</sub> segítségével.

Példaként vegyük a `gnuplot` programcsomagot: ennek megszerzéséhez először meg kell tudnunk azoknak a gépeknek a nevét, amelyek szerverként tárolják ezt a programcsomagot (a programcsomagok általában `compress`-el tömörített `tar` állományok, így az archívállomány neve `.tar.Z` végződik).

A szervergépek nevét az ún. `archie` szolgáltatáson keresztül kérdezhetjük le: pl. a bécsi egyetemen egy jól használható `archie`-szerver működik. Eléréséhez a `$ telnet 131.130.1.23`

parancs kiadása után az `archie` felhasználói azonosítóval léphetünk be. Ezek után már magába az `archie` programba kerülünk be. A teljes parancskészlet nem egységes, az aktuális gépen ezt általában a `help` utasítással kérdezhetjük le.

A fenti példát folytatva először válasszuk ki az első 50 találatot (ez időt takarít meg):

```
archie> set maxhits 50
majd keressünk rá a gnuplot szóra:
archie> prog gnuplot
```

A program ezek után végigkeresi az adatbázist, és kiírja azoknak a gépeknek az Internet címét, amelyek a `gnuplot` programcsomagot tárolják. A címek mellett a teljes elérési útvonalak is megjelennek.

## 218 FEJEZET 17. HÁLÓZATI ALAPISMERETEK ÉS EGYÉB PROGRAMOK

Egy adott cím kiválasztása után a `quit` (esetleg az `exit`) utasítással léphetünk ki az `archie`-ből. Ezek után az `ftp` segítségével lépünk be a kiválasztott gépre: felhasználói azonosítóként `anonymous-t`, kulcsszóként saját E-level címünket használjuk.

Ezek után lépünk be az `archie` által megadott könyvtárba (ez általában a `/pub` könyvtárból nyílik), és az `ftp` parancsait használva (l. 71. oldal) hozzuk át a programcsomagot. Vigyázzunk, az általában `.tar`, `.Z`, `.gz` végződésű állományok bináris átvitelt (azaz először a `binary` utasítást) igényelnek!

Az átvitel után a szokásos `quit ftp` utasítással zárhatjuk a kapcsolatot.


Az áthozott programcsomagot a 11. fejezetben, a 133. oldalon leírt módokon csomagolhatjuk ki. Ezek után követnünk kell a programcsomag dokumentációjában leírt utasításokat.

Megjegyezzük, hogy az ELTE az `ftp.elte.hu` gépen üzemeltet `anonymous ftp` szolgáltatást.

## 18. fejezet

# A `tin` elektronikus újság részletes leírása

Az UseNet az 1970-es évek végén jött létre. Kezdetben — az akkori UNIX lehetőségeknek megfelelően — UUCP kapcsolatot (általában modemet) használt a viszonylag kevés újságrovat terjesztéséhez. A '80-as évek közepén a UseNet egyre nagyobb és nagyobb lett, mind többen kezdték el használni. Ennek megfelelően újabb terjesztési módok is megjelentek: a közvetlen TCP/IP kapcsolat, a gyors E-levelezés és a különböző, a felesleges átvitelt elkerülő protokollok (pl. *IHAVE/SENDME*) ma már lehetőséget biztosítanak a háromezret meghaladó rovat terjesztéséhez. A UNIX mellett már más operációs rendszerek (VAX/VMS, IBM VM/CMS, stb.) is képesek futtatni a `news` programot.

 a `tin` rövid leírása a 75. oldalon

Az egyes rovatok — a sok cikk miatt — meglehetősen sok lemezterületet igényelnek, amely nem mindig áll rendelkezésre. TCP/IP hálózat használata esetén lehetőség van egy központi ún. „news server” gép megadására<sup>1</sup>. Általában csak ez a szerver tárolja a leveleket, míg a többi gépről az újságolvasó programok a hálózaton keresztül, közvetlenül érik el a leveleket, az ún. NNTP protokollt használva. A központi szerver általában mindenkinek, bármely gépről lehetővé teszi az olvasást, de csak a külön engedélyezett gépekről lehet új cikkeket elküldeni.

A `tin` program egy egyszerűen kezelhető, mégis sok feladatot ellátni képes program. A UNIX világban e mellett több más program is használatos: ilyenek a pl. a `vnews`, `xrn` (X11 Window alapú), `gnus` és `gnews` (a GNU `emacs`on alapuló), `nn`, `rn` és a `trn` újságolvasó programok. A VAX/VMS rendszeren a teljes UseNet

---

<sup>1</sup>Pl. az ELTE-n a `ludens.elte.hu` tölti be a news server szerepét.



news rendszert az ANU NEWS program valósítja meg.

## 18.1. A rovatok kiválasztása


 A tin első használatakor kiírja az adott gépen található összes rovatot. Például:

```
KILL ON Group Selection (1986) Type 'h' for help

u 1 alt.abortion.inequity -
u 2 alt.abuse-recovery -
u 3 alt.abuse.offender.recovery -
u 4 alt.abuse.recovery -
u 5 alt.activism -
u 6 alt.activism.d -
u 7 alt.adoption -
u 8 alt.aeffle.und.pferdle -
u 9 alt.agriculture.fruit -
u 10 alt.agriculture.misc -
u 11 alt.aldus.freehand -
u 12 alt.aldus.misc -
u 13 alt.aldus.pagemaker -
u 14 alt.alien.vampire.flonk.flonk.flonk -
u 15 alt.alien.visitors -
```

A tin a rovatok mindegyikét u (unsubscribed) jellel látja el: ez azt jelenti, hogy az a rovatot nem „fizettük” elő. A sorszámmal vagy a  és  gombok segítségével (esetleg a fel- és lefelé nyilakkal) kiválasztott aktuális rovatot az s (subscribe) utasítással vehetünk fel saját listánkra.

A későbbiek során is lemondhatunk egy rovatot az u (unsubscribe) utasítással (ezek a parancsok természetesen csak rovatlista szinten működnek). Egy egész rovatcsoportot (pl. alt) az s utasítással vehetünk fel a listára, és az U parancssal mondatunk le, ha kiterjesztett reguláris kifejezést használva megadjuk a csoport nevét. Például a sci.\* kifejezés az összes sci-vel kezdődő csoportot kiválasztja, míg a .\*comp.\* kifejezéssel minden olyan csoportra hivatkozunk, melynek nevében szerepel a comp string.

 A rovatok listáját, a már olvasott cikkek számával együtt a felhasználó HOME könyvtárában található .newsrc állomány sorolja fel. Ebben a rovatok neve után megtalálható a már olvasott cikkek száma, pl.:

```
:
comp.lang.idl-pvwave: 32-123
comp.laser-printers!
```

```
:
```

Itt a `comp.lang.idl-pvwave` rovat 32-től 123-ig terjedő sorszámú cikkeit olvastuk, míg a `comp.laser-printers` rovatra nem fizettünk elő (ezt jelzi a !). A rovatokat az állománybeli sorrendben írja ki a `tin` a képernyőre. Amennyiben a !-et :-ra cseréljük, a program ezt úgy tekinti, mintha előfizettünk volna az adott rovatra.

Ha a `.newsrc` állományt kézzel átszerkesztjük, a `tin` az újabb elindításkor a változtatásokat figyelembe fogja venni.

## 18.2. Cikk elküldése

A `tin` lehetőséget ad új cikkek elküldésére is. Ezt a `w` (write) utasítás segítségével tehetjük meg. Ekkor egy editor (alapértelmezésben a `vi`) segítségével megszerkeszthetjük a cikket, a cikk címét (tárgy, Subject) és a rovat (vagy esetleg rovatok) nevét, ahova küldeni szeretnénk (alapértelmezésben ez az aktuális rovat).

Egy cikk olvasásakor nemcsak új levelet küldhetünk, hanem lehetőség van egy éppen olvasott cikkre válaszolni. Erre az `F` (followup) parancs szolgál, amely ilyenkor az írásra használt szerkesztőbe betölti az eredeti cikket (így kényelmesen tudunk az egyes részekre hivatkozni).

A cikkek elküldésekor lehetőség van a terjesztési kör (pl. ELTE, Magyarország, Föld, az egész világ stb.) megadására is. A cikkek írásakor ezért ne feledjük, hogy írásunkat az egész egyetemen, sőt akár az egész világon is olvashatják! Ügyeljünk arra, hogy szokásostól eltérő terjesztési kör esetén azt pontosan, szándékunknak megfelelően adjuk meg (ellenkező esetben sok, a helyes használatra figyelmeztető levelet kaphatunk a morcos olvasóktól). Általában célszerű, ha megpróbálunk alkalmazkodni az adott rovatban megjelenő cikkek stílusához, valamint tanulmányozzuk a cikkek írásakor követendő, általánosan elfogadott szabályokat (ezek rendszeresen megjelennek a `news.announce.newusers` rovatban).

Bizonyos rovatokba csak szerkesztőkön keresztül lehet írni (így pl. elkerülhető a sok hasonló válasz-cikk egy kérdésre, ki lehet szűrni az esetlegesen nem oda tartozó kérdéseket, stb.). A küldőnek nem kell különbséget tenni a különböző rovatok között, a `tin` automatikusan postázza az esetleges rovatszerkesztőnek a megírt cikket.

A cikkek írásakor a `tin` a megírt cikk végéhez illeszti — amennyiben a felhasználó létrehozta — a `$HOME/.signature` aláírás állomány tartalmát.

Egy aktuális cikket – ha az saját írásunk – a `c` (cancel) parancs segítségével törölhetünk ki a cikklista szinten. A törlés elég sokáig tarthat, mivel a cikkek a hálózaton keresztül hamar szétterjednek a különböző gépekre.

### 18.3. A tin környezet

A tin program több állományban tárolja az újságolvasó működésével kapcsolatos információkat. Mindenki egy idő után kiválasztja azokat rovatokat, amelyeket többé-kevésbé rendszeresen olvas, ezért a program a \$HOME/.newsrc állományban rögzíti a „előfizetett” rovatok listáját a már olvasott cikkek számának feltüntetésével.

Az elektronikus újság lehetővé teszi, hogy a már korábban olvasott cikkek a következő belépéskor már ne zavarjanak. Ennek megfelelően egy rovatba történő belépés után minden cikk olvasott cikknek számít, és ez szintén rögzül kilépéskor a \$HOME/.newsrc állományban.

Lehetőség van a cikkek bizonyos csoportosítására is rovatlista szinten a t (tag) parancs segítségével további feldolgozás (nyomtatás, E-levél és pipe) számára.

A tin az állományban használt sorrendben jeleníti meg a rovatokat a képernyőn.

A tin működésével kapcsolatos állományok zömét a \$HOME/.tin könyvtár tartalmazza. A \$HOME/.tin/tinrc konfigurációs file közvetlenül is szerkeszthető, de a tin programban is előhívható az M utasítással:

#### Options Menu

```

1. Auto save : ON
3. Mark saved read : ON
5. Show Author : ON
7. Print header : OFF
9. Process type : None
10 Sort article by : Date: field (ascending)
11 Save directory : /u/galahad/News
12 Mail directory : /u/galahad/Mail
13 Printer : /usr/ucb/lpr
2. Save separate : ON
4. Kill articles : OFF
6. Draw arrow : OFF
8. Goto 1st unread : ON

```

Select option by entering number before text. Any other key to save.

Az opciószám megadása után változtathatunk az adott ponton, míg pl. az Enter gombbal elmenthetjük az adott beállítást.

Az Auto save opció beállításakor a cikk Archive-name sorát használja fel a tin az esetleges elmentéskor, míg a Save separate opcióval kiválasztható, hogy ezek a cikkek külön-külön állományba kerüljenek-e. A Mark saved read pont az elolvasott cikkek automatikus bejegyzését teszi lehetővé a \$HOME/.newsrc állományba.

A Kill articles engedélyezi a cikkek kizárását a \$HOME/.tin/kill állomány használatával a From: és Subject: sorok alapján: ez különösen hasznos

lehet bizonyos szerzőktől származó cikkek kizárására. A `kill` menübe a rovatlista és cikklista szintről a `Ctrl-K` utasítással léphetünk be.

A `Show Author` pontban be- és kikapcsolhatjuk a szerző kiírását, míg a `Draw arrow` pontban az inverz és a `-->` mutató között választhatunk. A `Print header` a teljes és a szűk (csak a `Subject` és `From:` mezőket tartalmazó) fejléc között választ. A `Goto 1st unread` kapcsoló beállítása esetén a rovatlistába belépve a gép a mutatót az első, még nem olvasott cikket tartalmazó rovatra állítja automatikusan.

A `tin` képes a leveleket különböző software terjesztési módoknak megfelelően dekódolni. A `Process type` dekódolási típus lehet `shell script` (azaz egy önszétcsomagoló program), `uuencode` (bináris file ASCII átvitele), `uuencode + zoo` (`zoo` bináris tömörített archívum ASCII átvitele) és `patch` (program javító file).

A `Sort article by` pontban a cikkek sorrendjének kritériumait adhatjuk meg (ez lehet üres, vagy a cikkek `Subject:`, `From:` vagy `Date:` mezője). A `Save directory` az elmentett cikkekhez tartozó könyvtárat állítja, míg a `Mail directory` és a `Printer a tin` program által használt mailer és a nyomtató elérését adja meg.

A konfigurációs állományok beállítását a környezeti változók módosíthatják. A két legfontosabb egyike az `NNTPSERVER`, amelyik a news server Internet címét adja meg. A másik az `EDITOR` változó, amely a cikk küldésekor használt szövegszerkesztő elérését tartalmazza.

## 18.4. A tin parancsai

Általában a rovat- és cikklistán történő mozgáshoz nemcsak a `j-k` gombokat (esetleg nyilakat) használhatjuk, hanem a `Ctrl-U` (vagy `b`) és a `Ctrl-D` (vagy `Szóköz`) parancsokkal léphetünk egy képernyőnyit felfelé ill. lefelé.

Ha egy adott sorszámú rovatra ill. cikkre akarunk hivatkozni, akkor a sorszámának begépelése rögtön odavisz minket. A `$` utasítás a lista végére ugrik. Egy adott szinten lehetőség van megadott kifejezés keresésére: erre a `vi` programnál használatos `/` (előre keresés) és `?` (visszafele keresés) utasításokat használhatjuk.

A `Ctrl-L` parancs újraírja a képernyőt.

A `tin` sok hasznos paranccsal rendelkezik, melyeket vagy az adott szinthez tartozó `h` (`help`) paranccsal, a struktúráltabb `H` (`help`) paranccsal, vagy a program man leírásában (`man tin`) találhatunk meg az érdeklődők. A leggyakrabban használt, az előzőekben nem szerepelt utasítások a következők:

- általános és rovatlista szinten használható utasítások:

`Ctrl-K` Alkalmazza a kill szűrőket az adott szintre (a kill szűrőt a 222. oldalon leírt módon állíthatjuk be).

`H` Ki- és bekapcsolja a minimenüt.

`g` Új rovatot választhatunk ki segítségével a rovat- és cikklista szintekről.

`m` Az adott rovatot áthelyezi a rovatlistában a megadott sorszámú helyre. A `§` a legutolsó pozícióra vonatkozik.

`s` „Előfizet” a megadott rovatra (csak rovatlista szinten).

`S` „Előfizet” egy megadott mintával egyező rovat(ok)ra (csak rovatlista szinten).

`u` Törli az „előfizetést” az aktuális rovatra (csak rovatlista szinten).

`U` Törli az „előfizetést” a megadott mintával egyező rovat(ok)ra (csak rovatlista szinten).

`y` Felfrissíti a `$HOME/.newsrc` állományt az operációs rendszer központi rovatlistájából (csak rovatlista szinten).

`c` Az adott rovatban minden cikket olvasottnak állít be.

`z` Az adott rovatban minden cikket olvasatlannak állít be (csak rovatlista szinten).

`z` Az adott cikket olvasatlannak állítja be (csak cikklista és cikkolvasó szinten).

- cikklista és cikkolvasó szinten használható utasítások:

`a` Előre keresi az adott szerzőt.

`A` Hátrafelé keresi az adott szerzőt.

`t` A tag parancs segítségével további feldolgozás számára jelölhetünk ki cikk(ek)et, ami lehet nyomtatás (`o`), E-levél (`m`), elmentés (`s`) és pipe (`|`) (csak cikklista szinten).

`U` A tag parancs hatását szünteti meg (csak cikklista szinten).

`m` Az aktuális cikke(ke)t (vagy a megjelölt listát, esetleg egy adott mintával kiválasztot cikkeket) elküldi E-levélként valakinek.

`o` Az aktuális cikke(ke)t (vagy a megjelölt listát, esetleg egy adott mintával kiválasztot cikkeket) kinyomtatja.

`|` Az aktuális cikke(ke)t (vagy a megjelölt listát, esetleg egy adott mintával kiválasztot cikkeket) egy adott parancsnak megadja bemenetként.

`s` Elmenti az aktuális cikke(ke)t (vagy a megjelölt listát, esetleg egy adott mintával kiválasztot cikkeket). Amennyiben egy `elm` levélszekrénybe akarunk menteni, használjuk a file nevének megadásakor az `=FileNév`



- megjelölést. A *+FileNév* egy *RovatNév/FileNév* típusú állományt ad meg a felhasználó `$HOME/News` könyvtárában.
- d A cikket az ún. *rot13* eljárással dekódolja. Az ilyen típusú cikkek esetleg felhőborítóak, gusztustalanok, stb. lehetnek az olvasók számára (ez a parancs csak a cikkolvasó szinten működik).

# Tárgymutató

- ., 39
- .., 39
- .forward, 194, 206
- .netrc, 71
- .newsrc, 220
- .profile, 104
- .rhosts, 215
- .tinrc, 222
- /dev/rmt0, 137
- /dev/rmt0h, 137
- /dev/rst0, 137
- /usr/lib/Mail.rc, 194
- <, 54, 55
- >, 53
- >>, 54
- állományok, 23
  - átnevezése, 32
  - kírása, 27
  - keresése, 41
  - létrehozása, 24
  - listázása, 25
  - másolása, 31, 40
  - mozgatása, 40
  - nyomtatása, 49
  - törlése, 32
  - típusai, 24
  - védelme, 42
  - védelmi kódja, 42
- átírányítás, 53
  - bemenet, 54
  - kimenet, 53, 54
  - példák, 56
- írási jog, 42
- a.out, 152
- alias, 99
- alkönyvtár, 37
- anonymous ftp, 217
- ar, 149
  - példák, 151
- archie, 217
- archiválás, 133, 134
- archiválási példák, 138
- argumentum, 22
- ARP, 214
- at, 156
- awk, 117
  - BEGIN, 121
  - ciklus, 124
  - END, 121
  - függvények, 123
  - feltételek, 120, 124
  - kapcsolók, 117
  - kiírás, 125
  - kifejezések, 118
  - működése, 118
  - példák, 126
  - parancsok, 118
  - utasítások, 123
  - változók, 121

- batch, 158
- bg, 59
- bridge, 214
- bsplit, 142
- btoa, 142
- C fordító, 146
- cat, 28
- cc, 146
- cd, 39
- chmod, 42
- chsh, 18
- compress, 140
- cp, 31
- cron, 158
- crontab, 158
- cső, 56
- dbx, 160
  - kapcsolók, 161
  - kifejezések, 160
  - példa, 163
  - parancsok, 162
- dd, 135
  - példák, 136, 138
- domain, 214
- E-levelezés, 68
- EBCDIC, 136
- EBCDIC átalakítás, 136
- echo, 19
- ed, 173
- egrep, 107
- elérési út, 37
- elérési útvonal, 39
  - abszolút, 39
  - relatív, 39
- elektronikus újság, 75
  - .newsrc, 220
  - cikk küldés, 221
  - cikk kizárása, 222
  - cikklista, 76
  - előfizetés, 220
  - első indítása, 75
  - környezeti változók, 223
  - olvasás, 77
  - parancsok, 223
  - rot13, 225
  - rovatlista, 76
- elektronikus levél automatikus továbbítás, 194, 206
- elektronikus levelezés
  - levélszekrény, 193
- elm, 68, 203
  - .elm, 203
  - alaputasítások, 204
  - alias, 207
  - automatikus továbbítás, 206
  - becenév, 207
  - elmr, 203
  - első használata, 203
  - főmenü, 68, 204
  - folder, 68, 206
  - küldés, 69
  - keresés, 204
  - kilépés, 69
  - konfigurációs állomány, 203, 208
  - mozgás, 69, 204
  - olvasás, 69
  - rövid leírás, 68
  - részletes leírás, 203
  - törlés, 205
  - továbbküldés, 205
  - válaszolás, 205
- elmr, 208
- Ethernet, 212
- f77, 148

- felhasználói név, 16
- fg, 59
- fgrep, 107
- find, 41
- finger, 74
- folder, 68
- fordítási könyvtár, 149
- fordított aposztróf, 56
- Fortran fordító, 148
- ftp, 71
  - anonymous, 217
  - elindítása, 71
  - kilépés, 72
  - példa, 73
  - utasítások, 72
- futtatási jog, 42
- gopher, 78
  - mozgás, 79
  - olvasójelek, 80
- grep, 33, 107
  - kapcsolók, 108
  - minta, 108
  - példák, 33, 36, 109
  - rövid leírás, 33
  - részletes leírása, 107
- héj, 19
- head, 29
- helyettesítő karakterek, 34
- ICMP, 213
- IP, 211
- jelszó, 16
  - megváltoztatása, 17
- jobs, 59
- jog
  - írási, 42
  - futtatási, 42
  - olvasási, 42
- könyvtár, 37
  - aktuális, 37
  - al-, 37
  - gyökér, 37
  - HOME, 37
  - létrehozása, 41
  - lekérdezése, 37
  - mozgás, 39
  - szülő, 37
  - törlése, 41
- kapcsoló, 20
- kijelentkezés, 18
- kill, 60
- Korn shell, 83
- ksh, 83
  - állománynév helyettesítés, 98
  - alias, 99
  - beépített parancsok, 87, 91
  - bemenete, 98
  - függvények, 89
  - feltételek, 89
  - környezeti változók, 84
  - kapcsolók, 94
  - kimenete, 98
  - metakarakterek, 85
  - paraméter, 100
  - parancsfeldolgozás, 84
  - parancsszerkesztés, 86
- kulcsszó, 16
- ld, 152
- link, 40
  - hard, 40
  - létrehozása, 40
  - szimbolikus, 40
- login, 16
- lpq, 50, 51
- lpr, 49

- ls, 21, 25
- mágnesszalagos egységek, 137
- MAIL, 193
- mail, 65
  - .mailrc, 194
  - automatikus továbbítás, 194
  - környezeti változók, 193
  - kapcsolók, 194
  - kilépés, 68
  - levél elküldése, 66
  - levél fogadása, 67
  - levelesláda, 193
  - példák, 68, 200
  - parancsok, 196
  - rövid leírás, 65
  - részletes leírás, 193
  - tilde parancsok, 195
  - UCB, 65
  - változók, 199
- MAILCHECK, 194
- MAILRC, 194
- mailrc, 194
- make, 166
  - kapcsolók, 169
  - kiterjesztések, 167
  - makró, 167
  - példák, 169
  - példa, 168
- man, 22
- msg, 65
- meta karakter, 186
- MicroEMACS, 46, 185
  - .emacsrc, 185
  - üzemmódok, 190
  - állomány kezelés, 188
  - ablakok, 189
  - beszúrás, 187
  - buffer, 189
  - helyettesítés, 188
  - keresés, 187
  - kilépés, 46
  - makrók, 191
  - meta karakter, 186
  - mozgás, 47, 186
  - mozgatás, 190
  - parancsok, 185
  - rövid leírása, 46
  - részletes leírása, 185
  - shell elérése, 191
  - törlés, 46, 187
- more, 30
- mt, 137
  - példák, 138
- multitasking, 15
- mv, 32
- news server, 219
- nice, 61, 156
- nohup, 60
- nyomtatás, 49
  - lekérdezése, 50
  - megszakítása, 51
- nyomtató, 49
- object könyvtár, 149
- olvasási jog, 42
- pack, 141
- paraméter, 22
- parancs
  - megszakítása, 19
- parancsállományok, 57
- Pascal fordító, 149
- passwd, 17
- password, 16
- path, 37
- pcat, 141

- pg, 29
- PID, 60
- pipe, 53, 56
- prioritás, 60
  - beállítása, 61
- program
  - futtatása, 59
  - háttérben futtatása, 59
  - leállítása, 59, 60
  - megszakítása, 59
- programok összekapcsolása, 56
- programok parancsok argumentumaként, 56
- ps, 60
- pwd, 39
- queue, 49
  - állapota, 50
  - megszakítása, 51
  - nyomtatás, 49
- renice, 61
- rlogin, 216
- rm, 32
- rot13, 225
- routing, 214
- rsh, 216
- rtin, 75
- script, 57
- sed, 110
  - kapcsolók, 111
  - példák, 114
  - parancsok, 111
- shell, 19, 53
- sort, 33, 127
  - kapcsolók, 127
  - példák, 34, 129
  - rövid leírása, 33
  - részletes leírás, 127
- split, 142
- standard input, 54
- standard output, 53
- szimbolikus debugger, 160
- tail, 29
- talk, 64
  - letiltása, 65
- tar, 134
  - példák, 138
- TCP/IP, 209
  - datagram, 210
  - portszám, 210
  - rétegek, 210
  - socket, 212
- telnet, 70
- tin, 75
  - első indítása, 220
  - konfigurációja, 222
- UCB mail, 65
- UDP, 213
- umask, 105
- uncompress, 140
- uniq, 115
- unpack, 141
- UseNet, 219
- user ID, 16
- uudecode, 142
- uuencode, 141
- várákozó lista, 49
  - állapota, 50
  - megszakítása, 51
  - nyomtatás, 49
- vi, 45
  - üzemmódok, 173
  - állományok beolvasása, 181
  - állományok kiírása, 181
  - beírás, 45, 178

behívása, 45  
helyettesítés, 181  
kapcsolók, 174  
keresés, 178  
kilépés, 45, 182  
másolás, 180  
makró, 183  
mozgás, 46, 177  
mozgatás, 180  
parancsok, 176  
rövid leírás, 45  
részletes leírás, 173  
set, 175  
shell elérése, 182  
sorok jelölése, 174  
törlés, 46  
változók, 175  
visszaállítás, 180  
who, 22  
write, 63  
  letiltása, 64  
zcat, 140