

A GNU/Linux alapparancsai  
1.0.2  
A Mithrandir Kft. nyelvi ellenőrzésével

Balsai Péter  
Kósa Attila

2002. június 20.



Copyright © 2001-2002 Linux-felhasználók Magyarországi Egyesülete

E közlemény felhatalmazást ad önnek jelen dokumentum sokszorosítására, terjesztésére és/vagy módosítására a Szabad Szoftver Alapítvány által kiadott GNU Szabad Dokumentációs Licenz 1.1-es, vagy bármely azt követő verziójának feltételei alapján. Nem Változtatható Szakaszok nincsenek, Címlap-szövegek nincsenek, a Hátlapszövegek neve pedig „hátlapszöveg”. E licenz egy példányát a GNU Szabad Dokumentációs Licenz elnevezésű szakasz alatt találja.

A módosított változat közzétételéért felelős személyek:

*Sári Gábor* saga@lme.linux.hu

Javítások: Sári Gábor

## **Szerző**

*Szűjjártó László* laca@janus.gimsz.sulinet.hu

## **Szakmai lektor**

*Kis Tamás* dozer@lme.linux.hu

## **Nyelvi ellenőrzés**

*Sári Gábor* saga@tux.hu

*Kósa Attila* atkosa@shinwa.hu

## **Formázás (L<sup>A</sup>T<sub>E</sub>X)**

*Kósa Attila* atkosa@shinwa.hu

## **Előzmények**

**A GNU/Linux alapparancsai**  
**A Mithrandir Kft. nyelvi ellenőrzésével**

A kiadás éve: 2002.

## **Szerző**

*Szűjjártó László* laca@janus.gimsz.sulinet.hu

## **Szakmai lektor**

*Kis Tamás* dozer@lme.linux.hu

## **Nyelvi ellenőrzés**

*Sári Gábor* saga@tux.hu  
*Kósa Attila* atkosa@shinwa.hu

## **Formázás (L<sup>A</sup>T<sub>E</sub>X)**

*Kósa Attila* atkosa@shinwa.hu

Az LME által elkészített Pingvin füzeteken a Mithrandir Kft. az olvashatóság érdekében nyelvi, helyesírási javításokat végzett.

A Mithrandir Kft. – valamint a nyelvi javítást végző természetes személyek – szakmai ellenőrzést, javítást nem végeztek.

Nem tették ezt (szakmai javítás), akkor sem – a szerzők és a szakmai lektorok munkája iránti tiszteletből –, ha a leírtak nem feleltek meg szakmai meggyőződésüknek.

A Mithrandir Kft. javítást végző szakemberei, illetve a Mithrandir Kft. mint jogi személy a leírtak helyességéért, esetleges avultságáért semmilyen felelősséget nem vállal.

# Tartalomjegyzék

<b>1. A GNU/Linux alapparancsai</b>	<b>5</b>
1.1. Bevezetés . . . . .	5
1.2. Általános parancsok . . . . .	6
1.3. Segítségkérés. A kézikönyv ( <i>man</i> ) és az <i>info</i> használata . . . . .	7
1.4. Belépés, kilépés (és egyebek...) . . . . .	9
1.5. Tájékozódás a rendszerben . . . . .	10
1.6. Szövegszerkesztők . . . . .	12
1.7. A rendszer működését befolyásoló parancsok . . . . .	14
1.8. Állománykezelő parancsok . . . . .	16
1.9. A könyvtárkezelés parancsai . . . . .	17
1.10. Fájlkezelő parancsok . . . . .	20
1.11. Tulajdonjogok, hozzáférés szabályozása . . . . .	24
1.12. Egyéb állománykezelési parancsok . . . . .	27
1.13. Állományrendszerekkel kapcsolatos parancsok . . . . .	31
1.14. Szövegkezelő parancsok . . . . .	37
1.15. Folyamatkezelő parancsok . . . . .	40
1.16. Hálózati eszközök kezelése, ügyfélprogramok . . . . .	46
1.17. Kernel és program építés, kernel modulok . . . . .	48
1.18. A héjprogramozás segédeszközei . . . . .	53
1.19. Az X Window rendszer alapparancsai . . . . .	55
1.20. Csomagok kezelése . . . . .	58
1.21. Egyéb parancsok . . . . .	60
1.22. Zárszó . . . . .	61

# Ábrák jegyzéke

1.1. A jed működés közben . . . . .	14
1.2. A Midnight Commander . . . . .	30
1.3. A cfdisk képernyője . . . . .	32
1.4. Egy grafikus archívumkezelő, a GnoZip . . . . .	36
1.5. A top működés közben . . . . .	42
1.6. Egy mindentudó hálózati beállítóprogram . . . . .	46
1.7. A make xconfig eredménye . . . . .	51
1.8. Néhány dialog-gal készült program . . . . .	55
1.9. Különböző terminálprogramok . . . . .	58

# 1. fejezet

## A *GNU/Linux* alapparancsai

### 1.1. Bevezetés

A *GNU/Linux* az utóbbi években igazi alternatívát jelent a felhasználók számára a többi elterjedt operációs rendszer mellett. Előretörését több mindennek köszönheti:

- mivel a *UNIX* operációs rendszerek családjába tartozik (amelyek immár 30 éve bizonyítanak), megbízható még az egyébként nem éppen stabilitásukról híres PC-ken is;
- programokkal való ellátottsága napról-napra jobb, gyakorlatilag ma már minden fontosabb területre megtalálható linuxos program is;
- dokumentáltsága angol nyelven nagyon jó, mivel a rendszer teljes mértékben nyílt forráskódú, nincsenek „nem dokumentált beállítások” vagy „Easter Egg”-ek;
- és még sok egyéb vonzó tulajdonsága is van, a végére egy nagyon erős érv mellette: ingyenes!

Mindezek figyelembevételével azt kell mondanunk, hogy ma már nem csak a számítógépes „guruk” operációs rendszere, hanem – reményeink szerint – hamarosan beköltözhethet az irodákba, és az egyszerű felhasználók gépein is vizionálhatjuk. Hálózati alkalmazása már ma is olyan méretű, hogy a számítástechnikával kicsit haladóbb szinten foglalkozóknak mindenképpen hallaniuk kell róla, találkozniuk kell vele.

A füzet célja, hogy azokat a fontosabb parancsokat összefoglalja – alkalmazási példák segítségével –, amik a Linuxszal való ismerkedés első időszakában előfordulnak. A parancsok neveit a folyó szövegen belül ilyen betűkkel találjuk meg, esetleges paramétereikkel. A példákban egy készenléti jel (általában `[root:~]>`) után látható az, amit be kell gépelni, majd a gép válasza következik, ezt az írógépes szöveghez hasonló betűtípus jelzi. Néhol fájlok, könyvtárak neve található más betűtípussal. A füzet szövege feltételezi bizonyos alapfogalmak ismeretét (például terminál, domain-név). Ahol lehet, hivatkozunk a *DOS/Windows* rendszereken megszokott dolgokra, hogy azok számára akik ezeket már ismerik, könnyebb legyen a munka. Mivel a szöveg példákat is tartalmaz, ajánlatos azonnal gépen is kipróbálni az itt leírtakat.

## 1.2. Általános parancsok

Mielőtt belekezdenénk, szükség van némi alapismeretre a *UNIX*ot illetően. A türelmetlenek kedvéért megígérem: nem lesz túl hosszú!

### 1.2.1. A UNIX-ok főbb tulajdonságai

A Linux, mint már említettük, a *UNIX* operációs rendszerek egy verziója, azoknak PC-re írt változata (bár létezik Linux más géptípusra is, például *Alpha*). Alapjaiban követi azokat a jellemzőket, amik egy *UNIX*nak sajátjai, ezek közül a legfontosabbak:

- egy *UNIX* rendszerben minden adatot egyetlen könyvtárfában, az azon belül elhelyezkedő alkönyvtárakban, és ezen belül fájlokban tárolunk. Sokszor hallani, hogy a *UNIX*ban minden fájl: egy szöveg, egy bináris program, de a háttértár egy partíciója vagy az első soros port is fájlként kezelendő. Ez első ránézésre furcsa lehet, de ha jobban meggondoljuk, ennek az „egyszerűsítésnek” több előnyös oldala van;
- a könyvtárfában kialakult rendszere van az alkönyvtárak elnevezéseinek és tartalmának is. Például a `/etc` tartalmazza a konfigurációs állományokat, rendszerindító szkripteket (ezek parancsokat tartalmazó, végrehajtható szöveges állományok, nagyjából a *DOS batch* fájljainak felelnek meg), a `/tmp` az átmeneti fájlokat. A példákából az is látható, hogy a könyvtáraknak a `/` az elválasztó jele;
- az eddigiekből már következik, hogy nincsenek külön „meghajtók”: ha egy CD-ROM-ot akarok használni, akkor azt (illetve a rajta lévő fájlrendszert) be kell illeszteni a könyvtárfába, ha már nincs rá szükség, akkor pedig le kell választani (tulajdonképpen ez *DOS* alatt is így van, csak ez a háttérben zajlik);
- a fájlnevek tartalmazhatnak szinte bármilyen karaktert, a kis- és nagybetűket megkülönbözteti egymástól a rendszer, több pontot is lehet rakni a névbe, és a fájl típusát nem tudjuk mindig a kiterjesztésből megállapítani, mert ez sokszor hiányzik is! Tehát például a következő fájlnevek használhatók Linux alatt:
  - `KissPista.levele.txt`
  - `johosszunevuprogram`
  - `Ez.is.program.lehet-[pedig.nem.exe.a.kiterjesztese]`
  - `.rejtett_file`

Ez utóbbiból látszik, hogy a *DOS/Windows* alatt megismert fájl-attribútumok sem az ott megszokott módon léteznek (más típusúak viszont igen, akit behatóbban érdekel, a `chattr` és az `lsattr` parancsok környékén kutasson), a rejtett fájlok nevét ponttal kell kezdeni;

- az alapfilozófia az, hogy a rendszerben minden egyes programnak legyen meg a jól behatárolt feladata, és azt (de csak azt!) jól lássa el. A legtöbb *UNIX* parancs emiatt önmagában csak bizonyos részfeladatot tud megoldani, annak viszont minden lehetséges előfordulását le tudja kezelni azáltal, hogy rengeteg paraméterrel lehet meghívni. Érdekes már most kipróbálni az `ls --help` begépelését, látható, hányféleképpen használható a legelemibb, könyvtártartalmat



kilistázó parancs is! Az egyes programokat aztán a *DOS*ból is ismert úgynevezett csővezetékekkel (angolul pipeline) összekapcsolhatjuk, és így egyszerű eszközökkel meglepően bonyolult dolgokat is meg tudunk oldani. Ezt a módszert nevezik „szerszámoszláda-modellnek”. A programok paramétereit a `-` jel után kell megadni, általában több paramétert össze is lehet vonni, az előző példánál maradva az `ls -a -l -i` ugyanaz, mint az `ls -ali`;

- a *DOS*ban is létezik egy parancsértelmező program, a `COMMAND.COM`, ami az általunk begépzelt parancsokat átadja feldolgozásra a rendszernek, esetleg hibaüzeneteket küld, és alapszinten programozni is lehet a már emlegetett *batch* fájlok segítségével. Linux alatt többféle parancsértelmező, idegen néven *shell*, magyarul héj- vagy burokprogram közül választhatunk. A leggyakoribb a `bash` (*Bourne Again Shell*), ami sok kellemes szolgáltatásával és nagyon jó programozhatóságával tűnik ki. Néhány egyszerűbb feladatra nem kell külön program, azt a *shell* végzi el.

Rövid összefoglalásként tehát a Linux alatt kiadott parancsaink vagy bináris, végrehajtható programok, vagy a *shell* belső parancsai, esetleg *shell*-szkriptek. A legtöbb program a következő formában használható:

```
program_neve -opció1 -opció2 ... paraméterek
```

Majdnem mindegyik rendelkezik úgynevezett *POSIX*-formátumú opciókkal is, ezeket `--` jellel (két mínusz) vezetjük be, és nem egybetűs, hanem hosszabb nevűek van, a jobb olvashatóság végett – cserébe sajnos kicsit többet kell gépelnünk. Tehát ha kíváncsiak vagyunk egy parancs működésére röviden, akkor kísérletezhetünk a `-h`, de a `--help` kapcsolóval is. Mindjárt látjuk azonban, hogy ennél részletesebb leírást is kaphatunk. Az elérési útvonalunkon lévő parancsokat közvetlenül a nevük begépelésével, az aktuális könyvtárban találhatóakat pedig a `./parancsnév` beírásával indíthatjuk.

### 1.3. Segítségkérés. A kézikönyv (*man*) és az *info* használata

Aki először ül le egy Linux elé, elsőre elbizonytalanodhat attól a rengeteg parancstól, amit talál a gépen. Ráadásul mindegyiknek több paramétere is van, és ezeket mind fejben tartani lehetetlen. Nem kell viszont megijedni, mert még teljesen ismeretlen és újszerű feladatok megoldásánál is nagyon jól használhatóak a következőkben tárgyalt eszközök, amikhez nem kell az Internet sem, mert nagy valószínűséggel már a helyi gépen vannak a hozzávalók.

Általában minden programhoz tartozik legalább egy kézikönyv, angolul manual, ebben a program használatát, paramétereit írják le. Ezt a parancsot tehát mindenképpen jegyezzük meg! Használata röviden: `man parancsnév`. Egy kézikönyv-oldal szabványos részekből épül fel: a parancs rövid leírása, összes paramétere, esetleges hibái, más hasonló típusú programokra való hivatkozások, stb. A mannak természetesen saját magának is van ilyen súgója, máris próbáljuk ki a `man man` parancsot! Nézzük meg a fontosabb paramétereit:

**-f parancsnév** a `what is` parancsnak felel meg, egy rövid összegzést ad az adott programról;

**-k kulcsszó** az *apropos* parancsnak felel meg, az adott kulcsszóhoz tartozó bejegyzéseket keres a man oldalakon. Az *apropos directory* például kilistáz minden olyan parancsot, ami valamilyen módon kapcsolatos a könyvtárakkal;

**-K string** a *stringet* az összes(!) man oldalon keresi, emiatt nagyon sokáig is tarthat a folyamat.

A fentiekben olvasható ... *parancsnak felel meg* elsőre érdekesnek tűnhet: ugyanazt a funkciót többféle paranccsal is meg lehet oldani? Hogy is van akkor ez a bizonyos „szerszámosláda-modell”? Nos, a *what is* és az *apropos* igazándiból egy shell-szkript vagyis héjprogram, ami kényelmi funkciókkal bővíti ki az alapparancsot, de ha nagyon akarjuk, akkor persze mindent lehet. A legtöbb héjprogram támogatja ugyanis az álnevek (*alias*) használatát. Ezekkel megoldható például az, hogy ha egy parancsot általában mindig ugyanazokkal a paraméterekkel használunk, és már unjuk a sok gépelést, vagy hajlamosak vagyunk elfelejteni a megfelelő paramétert, akkor csak ki kell találnunk egy rövid azonosítót, olyat, amilyen néven még nincs semilyen utasítás. Tegyük fel, hogy a már emlegetett *ls -ali* helyett egyszerűbbnek tűnik a *lls* begépelése. Nézzük, mi a teendő:

Először próbáljuk ki, nincs-e véletlenül egy ilyen nevű, más célra szolgáló program:

```
[root:~]>lls
bash: lls: command not found
```

Ha, mint látszik, nincs, akkor a következő lépés:

```
[root:~]>alias lls="ls -ali"
```

Nem kapunk ugyan semmilyen visszajelzést (megjegyezhetjük, hogy a legtöbb helyen egy UNIX rendszer csak akkor üzen vissza, ha baj van!), de adjuk ki az *lls* parancsot, és lássunk csodát! Arról, hogy milyen álneveink vannak beállítva, a paraméter nélküli *alias* tudósít, ha pedig meg akarunk szüntetni egy ilyen összerendelést, akkor az *unalias aliasnév* a kiadandó utasítás. Térjünk azonban vissza a segítségkéréshez!

A mannál újabb és sokszor pontosabb információt ad az *info* parancsnév. A paraméter nélkül kiadott *info* egy menürendszerben, úgynevezett *node*okban, csomópontokban jeleníti meg a fontosabb témákat, illetve parancsokat. Ezen belül billentyűkombinációkkal mozoghatunk:

**space** előre lapoz

**backspace** visszalapoz

**n** következő node

**p** előző node

**CTRL-L** frissítés

**b** legelejére ugrás

**e** legvégére ugrás

**?** help

**q** kilépés

Az info kezelése elsőre nem túl könnyű, ezért ajánlatos beszerezni a pinfo programot. Ez a lynx nevű, szöveges web-böngészőhöz nagyban hasonlít a navigálás módját tekintve, tehát a nyílbillentyűk, az ENTER és a Backspace segítségével majdnem mindent meg tudunk találni.

Információkat olvashatunk ezeken kívül a /usr/doc (esetleg /usr/share/doc) könyvtárban is egyes programokról. Itt is érdemes körülnézni, mert néhány programról esetenként minket jobban érdeklő információkat találhatunk. Nagyon sok rendszerezett anyag található a HOWTO alkönyvtárban, ezek az úgynevezett *HOGYAN*ok egy-egy speciális területet ölelnek fel, és lépésről-lépésre igyekeznek azokat elmagyarázni. Igazán hasznos olvasmányok.

Most már bátran nekivághatunk az igazi munkának, mindig tudjuk hol a segítség! A következőkben tárgyalt parancsok mindegyikének ajánlatos elolvasni a kézikönyvét, mert sok egyéb lehetőséget is rejtenek magukban.

## 1.4. Belépés, kilépés (és egyébek...)

A Linux többfelhasználós (multiuser) operációs rendszer. Még ha otthoni, vagy hálózatba nem kapcsolt gépen használjuk is, használatbavétele előtt be kell lépünk a rendszerbe. Ez egy felhasználói azonosító és a hozzá tartozó jelszó megadásával történik. Ez a gép beállításától függően történhet szöveges terminálon, vagy grafikus felületen is. A belépés kezdeményezésére nem kell külön parancsot kiadnunk, de jó tudni, hogy a beléptetés oroszlánrészét a login program végzi az előbbi, míg az xdm vagy kdm, stb. programok az utóbbi esetben. Ha szöveges terminálon lépünk be, akkor a belépés előtt már látható az issue, ami általában tartalmazza a gép főbb adatait, vagy olyan információkat, amit a rendszergazda jónak lát közölni a gépről (tartalma a /etc/issue fájlban található). Sikeres belépés után kapunk egy készenléti jelet, és olvashatjuk a „nap üzenetét” (message of the day).

Egy olyan rendszertől, amit sokan használnak (főleg, ha fontos adatokat is tárolnak benne), joggal várható el, hogy az egyes felhasználók csak azokat az adatokat érhesék el, amihez jogosultságuk van. Ennek alapfeltétele, hogy mindenki rendelkezzen az egyedi azonosítójához tartozó jelszóval, ami alapján a rendszer elsődlegesen azonosítani tudja őt. A jól megválasztott és megfelelően kezelt jelszó sok esetben elegendő védelmet nyújthat az illetéktelenek ellen.

Amikor a Linux rendszeren kapunk egy felhasználói azonosítót (*account*), ehhez tartozik egy jelszó is, amit vagy a rendszergazda ad meg, vagy azonnal mi magunk. Az előbbi esetben az első bejelentkezésnél meg kell változtatnunk ezt olyanra, ami megfelel a biztonsági követelményeknek: megfelelő hosszúságú (általában minimum 6 karakternek kell lennie kötelezően, hossza a régebbi rendszereken 8 karakterre volt korlátozva, a mai, ún. *shadow-password*öt használó rendszereken már jóval több is lehet), vegyesen tartalmaz kis- és nagybetűket, számokat (sőt, szinte mindegyik általában használatos karaktert is), de azért megjegyezhető legyen számunkra. A legrosszabb amit tehetünk, ha a felhasználói nevünkkel azonos jelszót választunk!

A jelszóváltást a passwd parancs kiadásával tehetjük meg. A program rákérdez a régi jelszavunkra, majd kétszer egymás után be kell gépelnünk az újat. Attól függően, hogy a rendszergazda hogyan szabályozta, bizonyos időközönként meg kell változtatnunk, általában havonta.

Karakteres terminálunk hamar megtelhet szöveggel. Ha ez zavar bennünket, a clear parancs a megoldás. Előfordulhat az is, hogy valamit nem úgy csináltunk, ahogy kellett volna (például egy bináris fájlt akartunk szöveggént kilistázni, és min-

denféle zagyvasággal telt meg a képernyő), ekkor a `reset` parancs kiadása segíthet. Lehet, hogy vakon kell begépelnünk, de ha sikerül, alapállapotba állítja vissza a beviteli eszközünk.

Lehetőség van arra, hogy menet közben váltsunk parancsértelmezőt, csak a nevét kell beírni. Az ismertebbek a `bash` mellett a `csh`, `tcsh`, `ash`, `zsh`. A `chsh` utasítással ez a változtatás hosszabb távra szól.

Munkánkat befejezve ki kell jelentkeznünk. A kilépés az `exit` vagy `logout` paranccsal, illetve a `CTRL-D` billentyűkombinációval is kezdeményezhető (ez utóbbi nem mindig és mindenhol működik).

Kiemelendő dolog, hogy **RENDSZERGAZDAKÉNT CSAK AKKOR DOLGOZZUNK, HA ELENEDHETETLENÜL SZÜKSÉGES!** Ekkor is csak a lehető legkevesebb időt töltsük bejelentkezve.

## 1.5. Tájékozódás a rendszerben

Sikerült bejelentkeznünk? Akkor ideje, hogy megismerjük a munkakörnyezetünket. Elsőként a saját adatainkat derítsük ki, hogy tisztában legyünk azzal, milyen jogok illetnek meg bennünket. A szénilitás korai jelentkezése ellen a `whoami` parancs ajánlható; ez kilistázza a felhasználói azonosítónkat, valamint azt, melyik gépen, illetve melyik terminálon jelentkezünk be, ha már elfelejtettük volna (akkor nagyon hasznos, ha a készenléti jel, a *prompt* nem mutatja ezeket az adatokat). Ehhez kapcsolódva sokszor lehet szükség arra, hogy megtudjuk, melyik könyvtárban is állunk éppen, ekkor jön jól a `pwd` utasítás, ami megadja azt a gyökérkönyvtártól kiindulva.

Kíváncsiak vagyunk arra, ki jelentkezett még be a gépre velünk együtt, és az adott pillanatban milyen programot futtat? Adjuk ki a nemesen egyszerű `w` parancsot! Még azt is láthatjuk, hogy honnan lépett be az illető: helyi terminálról vagy távolról. Paramétereik közül megemlíthető a `-s`, ami rövid formában listázza ki az adatokat, illetve a `-f`, ami nem listázza ki a *FROM* mezőt, vagyis hogy honnan csatlakozik a felhasználó.

A `tty` begépelésével azt tudhatjuk meg, hogy éppen melyik terminált használjuk. Ha konzolon dolgozunk, akkor `tttyx` a parancs válasza, ahol `x` 1-től 6-ig terjed általában, a 7-es sorszámú terminál már a grafikus rendszer, az *X Window* számára fenntartott. A konzolon a terminálok között az *ALT* és a terminál sorszámának megfelelő funkcióbillentyű egyidejű lenyomásával lehet váltani.

Felhasználói- és csoportazonosítónk lekérdezésére az `id` szolgál. Egy példa a futtatására:

```
[root:~]>id
uid=0(root) gid=0(root) groups=0(root)
```

Az első mező a *user ID* (felhasználói azonosító, *UID*), névvel és számszerűen is megadva. Majd a *Group ID*, a csoportazonosító (*GID*) következik, végül azoknak a csoportoknak a felsorolása, amelyeknek még tagja az illető. Ezek közül csak a *UID* kiírására az `id -u`, a *GID* kiírására az `id -g` használható. Ha mindezt nem számmal, hanem névvel kiírva akarjuk vizsgálni, akkor még a `-n` paraméter is kell, például:

```
[root:~]>id -un
root
```

A csoporttagságok lekérdezésére a `groups` parancs külön is használható.

A kíváncsi felhasználó az `uname`, `hostname` parancsokkal tudhat meg többet a gépről. Az előbbi ad több információt, szintén a `-a` paraméterrel kombinálva. Az utóbbi a nevéből adódóan a gép nevét adja vissza. Szintén nézzünk egy példát:

```
[root:~]>uname -a
Linux janus.gimsz.sulinet.hu 2.2.16 #5 Thu May 17 11:48:49 CEST 2001 i586 unknown
```

vagyis: az operációs rendszer neve, a gép teljes neve, a kernel (rendszermag) verziószáma, a kernel készítésének pontos dátuma, a gép hardvertípusa, és végül a processzor típusa (ez utóbbit a fenti példában nem sikerült beazonosítani). Ezek mindegyike külön-külön is lekérdezhető, például a `-n` opcióval a domain-név, a `-m` opcióval a hardvertípus.

A dátummal és idővel kapcsolatos adatok is fontosak lehetnek számunkra, ekkor adjuk ki a `date` parancsot, ez megad minden információt. Használatára nézzünk néhány példát!

A dátum- és időadatok alkotórészei (év, hó, nap, óra, perc stb.) mind külön-külön kiírathatóak oly módon, hogy mindegyiknek van egy azonosító betűjele, amit egy % jellel bevezetve kell megadni, és így egy úgynevezett formátumsztringet kell átadni paraméterként. Tehát ha a mai dátumot akarjuk a szokásos formában látni, írjuk be a következőt:

```
[root:~]>date +%Y.%m.%d
2001.05.29
```

Írassuk ki most a napot betűvel, majd a pontos időt:

```
[root:~]>date +%a,%r
Tue,11:59:07 AM
```

Mivel a gépen most az angol területi beállítások érvényesek, ezért látjuk a nap nevét Tue-ként (kedd), és az AM utal arra, hogy délelőtti időpontról van szó. Természetesen 24 órás formátumban is dolgozhatunk, ekkor a `%T` használandó. Érdekességképpen még nézzük meg, hány másodperc telt el 1970. jan. 1. 0 óra óta:

```
[root:~]>date +%s
991130612
```

A rendszergazda ezeken felül a `-s` paraméterrel tudja beállítani a pontos dátumot és időt. Itt kell említést tenni a `time` parancsról, ami más célra szolgál, mint a *DOS*-ban. Egészen pontosan megmérhetjük vele egy tetszőleges parancs végrehajtására fordított gépidőt, csak be kell utána írni az adott program nevét, és a futtatásához szükséges paramétereket. Az `uptime` pedig elsősorban arra ad választ, mennyi ideje működik a gép (egy Linux szervernél nem szokatlan a több száz nap sem).

Ezenkívül van még egy naptárprogramunk is kéznél, a `cal`. Egy adott évet paraméterként megadva szinte öröknaptárként is használható. Két paramétert megadva az első hónapnak tekint, a másodikat évnek, tehát ha csak egy adott hónap érdekel bennünket részletesen, akkor nem kell végigbogarászni a teljes évet. Feladatként keressük meg például a születésnapunkat, és nézzük meg, milyen napra esett.

Alapvető feladat annak kiderítése is, milyen fájlokat találunk az adott könyvtáron belül. A *UNIX*ok erre szolgáló parancsa az `ls`, de a *DOS* zord világából érkezők találnak egy régi ismerőst, mivel a `dir` is használható, de azért persze nem teljesen ugyanúgy. Már találkozhattunk az `ls` rengeteg paraméterével, nézzünk egy pár olyat, amit talán illik megjegyezni:

`-a` mindent – még a rejtett fájlokat is – listáz;

- l „long”, azaz hosszú formátumban listáz. Talán nem minden tanulság nélküli egy ilyen lista:

```
[root:/bin]>ls -l a*
-rwxr-xr-x 1 root root 2712 Oct 3 2000 arch*
-rwxr-xr-x 1 root root 65340 Oct 3 2000 ash*
-rwxr-xr-x 1 root root 263144 Oct 3 2000 ash.static*
lrwxrwxrwx 1 root root 4 Apr 29 14:12 awk -> gawk*
```

Mindjárt a parancs kiadásánál látható, hogy a *DOS*ban megszokott helyettesítő karakterek közül a \* itt is alkalmazható, itt is egyidejűleg több karakter helyettesítésére, de nem tökéletesen azonos az ottanival! Míg *DOS* alatt az első \* után már nem törődik a kiértékelő program a sor többi részével. Itt igen, tehát ha olyan fájlneveket akarunk kilistázni, amikben az első karakter „a”, majd valahol van benne ezután „b”, és még később „c” is, akkor az a\*b\*c\* mintát kell megadnunk.

A lista első oszlopa a fájl hozzáférési jogait jelenti, a következő oszlop (ami itt mindenhol 1-et tartalmaz) a láncolási szám, a következő két oszlop a fájl tulajdonosának és annak csoportjának neve, majd a fájl méret, a legutóbbi módosítás időpontja és végül a név következik. A legutolsó sorban egy szimbolikus link látható. Ezekre a jórészt ismeretlen fogalmakra természetesen vissza fogunk térni.

- i a fájlok úgynevezett *inode*-számát adja meg, ez az állománynak a háttértáron való elhelyezkedésével kapcsolatos adat.
- h a méreteket könnyebben értelmezhetően, k (kilobyte), M (megabyte) utótaggal írja ki (és nem sugót ad, arra a --help szolgál).
- l egy oszlopba egymás alá írja ki az adatokat. Itt is látható az, hogy a könyvtárak neve után megjeleníti a / jelet, vagy hogy a végrehajtható fájlok után \* áll, tehát valamilyen módon különbséget tudunk tenni így is a nevek között. Akkor pedig főleg, ha a --color paramétert használjuk, mert ha megfelelő terminálunk van, akkor színesben láthatjuk a parancs kimenetét.

## 1.6. Szövegszerkesztők

Alapvető fontosságúak a mindennapi munka során az egyszerű szöveges fájlok. Nemcsak abban kapnak szerepet, hogy a köznap értelemben vett szöveges információt (leveleket, leírásokat) tároljanak, hanem a Linuxban mint konfigurációs állományok is igen fontosak. Szöveges adatok keletkezhetnek a parancsok működése során is. A szöveg szerkesztésére szolgáló programok tehát nagyon fontosak, vegyünk sorra néhányat.

Aki a *Microsoft Word*ön nevelkedett, az a kőkorszakban érezheti magát az *ed* program láttán. Ő még abból az időből maradt ránk, amikor nemhogy grafikus felületek nem voltak, de még olyan szerkesztők sem, amik a szöveget megjelenítik a teljes képernyőn. Az *ed* sorszerkesztő: egyszerre a szöveg egy sorát képes feldolgozni. Ma már inkább csak héjprogramokban való használatra lehet jó.

Szintén őskövület, de nagyon jól tartja magát a *UNIX*os szövegszerkesztők nagy öregje, a *vi*. Sokan azt mondják, hogy bár elsőre ő is rémisztő, de a tudása még ma is lepipál sok látványosabb programot. Három üzemmódja van: a parancs mód, a beviteli mód és az *ex*-üzemmód. Indításakor a parancsmódba kerülünk, ahol a nevéből is

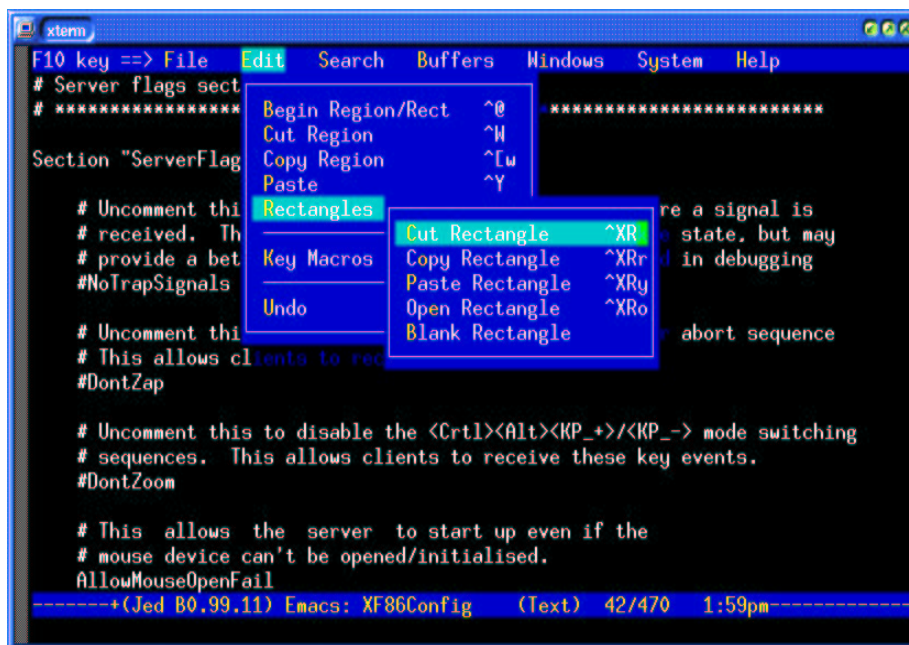
következően csak parancsokat tudunk kiadni. Ilyen például a listázás, keresés, csere, mozgás a szövegben. A parancsmódba bármikor átválthatunk az *ESC* billentyű lenyomásával. Szöveget beírni csak beviteli módban tudunk, ezt az *Insert*- billentyűvel, vagy az *a*, *i*, *o* billentyűkkel is kezdeményezhetjük. Az *ex*-üzemmódban pedig a mentés vagy a kilépés valósítható meg többek között, elérése a parancsmódból történik. Lássunk egy egyszerű példát. Írjunk egy rövid szöveget az *uj* nevű fájlba! Adjuk ki a *vi uj* parancsot, majd nyomjunk *Insert*-et, és gépeljünk be pár sort. A még üres sorok előtt ilyenkor a *~* jel látható, alul pedig az *INSERT* felirat olvasható. Bevitel közben a máshol megszokott módon a nyílbillentyűk, a *Home* és *End*, *Backspace* és a *Delete* is használható. Lépünk vissza parancsmódba az *ESC* billentyűt lenyomva, és gépeljük be a következő három karaktert: *:wq*. A kettőspont beléptet az *ex*-módba, a *w* hatására a fájl kiíródik a lemezre, és a *q* kiléptet a programból (ha úgy akarunk kilépni, hogy nem kell mentés, a *w* helyett egy felkiáltójelet használjunk). Akinek kedve és ideje van, érdemes elmélyedni a program rejtelmeiben, nem is hinnénk, mennyi mindenre jó!

Kényelmesebben használható a *joe* program, ami már valódi teljes képernyős szerkesztő. A legtöbb funkció a *CTRL* és valamelyik másik billentyű egyidejű lenyomásával érhető el. Máris érdemes kiadni a *CTRL-K-H* kombinációt (tehát tartsuk lenyomva a *CTRL*-t, és nyomjunk *K*-t majd *H*-t). Ez a sügőt jeleníti meg, ami alapján már el tudunk boldogulni. Itt mindenhol a *^* jel jelenti a *CTRL*-t. Használhatunk blokkműveleteket is, ami azt jelenti, hogy a szöveg bizonyos részeit ki tudjuk jelölni, és át tudjuk másolni vagy mozgatni – mint egy egységet – más helyre. A *CTRL-K-B* a kijelölés kezdetét, a *CTRL-K-K* a végét jelenti, ezek után a másolás a *CTRL-K-C*, a mozgatás a *CTRL-K-M*, a törlés pedig a *CTRL-K-Y* kombinációkkal valósítható meg.

A *jed* már menüvezérelt, emiatt még könnyebben használható. A menük a máshol már megszokott módon, az *ALT* és a menüpont nevében kiemelten szereplő betű lenyomásával nyithatók le, de az *F10*, majd az *Enter* lenyomásával is. A program képes egyszerre több fájl szerkesztésére, ezeket ún. pufferekben tárolja, és a *Buffers* menüpont alatt érhetjük el őket. A *Windows* menüponton belül akár több részre (ablakra) is oszthatjuk a képernyőt. Megoldható az is, hogy a programból való kilépés nélkül végrehajtsunk héjparancsokat (*System -> Shell command*), ezek eredménye bekerül egy új pufferbe, és ott tovább szerkeszthető vagy átmásolható másik szövegbe is. Az *Edit* menüben a *Begin Region* parancs kijelöli, a *Copy region* kimásolja, a *Paste* pedig beilleszti az adott részt. Az 1.1. képen egy *Mandrake 7.2*-es verzión futtatott példány látható.

Aki sokat dolgozott a *Norton Commander* szövegszerkesztőjével, annak ismerős lesz az *mcedit*. A használatos billentyűkombinációk megegyeznek az *NC*-beliekkel, az *F9* hatására megjelenő menüsorban azonban még további hasznos dolgokat is felfedezhetünk. Szintén könnyű megszokni a *pico* használatát, ami a *pine* levelezőprogram szövegszerkesztője, de külön is használható. Ebben is nagy szerephez jut a *CTRL* billentyű, a képernyő alján olvasható sügő segítségével könnyen birtokba vehető.

Említést kell tenni végül, de nem utolsósorban az *emacs* programról, ami tulajdonképpen nem is csak egy szövegszerkesztő mivel programozható, és ezáltal egy nagy tudású eszköz. Kezdőknek nehéz vele bánni, de érdemes megtanulni a használatát, főleg, ha programfejlesztésre adjuk a fejünket.



1.1. ábra. A jed működés közben

## 1.7. A rendszer működését befolyásoló parancsok

Elvárható, hogy egy több felhasználót kiszolgáló rendszer biztosítson egy jól használható alapbeállítást, munkakörnyezetet, amit aztán a felhasználók teljesen egyénire szabhatnak át, ha úgy látják jónak. A Linux ebből a szempontból is mintaszerű. Talán kicsit túl sok mindent is be lehet állítani ahhoz, hogy egy idő után az ember elveszen a sok lehetőség között, de ez is egy jó játék. Néhány dologgal azonban mindenképpen kell foglalkozni, mert annyira alapvetőek a gép működtetésére nézve. Némelyeket a rendszergazdának kell megcsinálnia, ha nem akarja a felhasználók rosszallását, de az itt felsoroltak zömével a mezei felhasználó is biztosan találkozik.

Mindjárt a parancsok és egyéb adatok bevitelénél szükség van a billentyűzetkiosztás megfelelő beállítására. Az alapértelmezettől eltérő kiosztás betöltése a `loadkeys` paranccsal történik. Paraméter nélkül az alapértelmezettet kapjuk, ha pedig megadunk egy országnévre utaló rövidítést, akkor az annak megfelelőt. A *hu* a hagyományos magyar *QWERTZ*, a *hu101* az angol billentyűzetekhez jobban illeszkedő, az *us* pedig az amerikai angol kiosztás rövidítése. Sőt, teljesen egyéni kiosztást is készíthetünk, vagy bizonyos billentyűkombinációkhoz parancsokat rendelhetünk. Nézzük, hogyan!

Elsőként a `dumpkeys` parancsot kell használnunk. Ez alapértelmezés szerint a képernyőre írja ki az aktuális billentyű-megfeleltetéseket. Minden lenyomott billentyű egy kódot generál, ezt a `showkey` programmal meg is nézhetjük (a programból máshogy nem lehet kilépni, mint 10 másodperc várakozással). Ezekhez a kódokhoz társulnak a megjelenő karakterek, figyelembevéve a váltóbillentyűket is. Ezen kell tehát változtatnunk, példának okáért szeretnénk, ha az *F11* funkcióbillentyű lenyomására lefutna az `uname -a` parancs. Ehhez a `dumpkeys` parancs kimenetét át kell irányítanunk egy fájlba:



```
[root:~]>dumpkeys > saját.bill
```

majd pedig az így keletkezett *saját.bill* állományban kell megkeresni azt a részt, ami valahogy így kezdődik:

```
string F11 = ''
```

és az idézőjelek közé beírni a parancsot a következő módon:

```
string F11 = 'uname -a\n'
```

A `\n` karaktersorozat hatására nem kell a végén majd az *ENTER*t lenyomni, mert azt is belefoglaltuk a kódba. Utolsó lépésként ezt a fájlt adjuk meg a *loadkeys* paramétereként, és már ki is próbálhatjuk, mit végeztünk!

A jobb oldali numerikus billentyűzet rész fölötti *LED*-ek vezérlésére is van program, a *setleds*. Kapcsoljuk be például a *CAPS LOCK* jelzőjét, a *NUM LOCK*-ot pedig ki :

```
[root:~]>setleds +caps -num
```

A képernyőn megjelenő betűk képe is fontos lehet számunkra. Szöveges üzemmódban is többféle lehetőségünk van, dolgozhatunk a hagyományos 80 oszlopos és 25 soros területen, de a betűk méretének változtatásával tágasabbá tehetjük a képernyőt. Igaz, ekkor jobban kell erőltetni a szemünket. A képernyőn megjelenő betűk úgynevezett *font* fájlokban tárolódnak a gépünkön, és a *consolechars* parancs segítségével lehet közöttük váltani. A `-f` paraméter után kell megadnunk egy ilyen font nevét. Mivel biztosan szeretnénk magyar ékezetes karaktereket is, részesítsük előnyben azokat a fontneveket, amelyek *lat*-tal kezdődnek, tehát latin 1-es vagy 2-es kódolásúak. Próbáljuk ki például:

```
[root:~]>consolechars -f latlu-12.psf
```

A fontok helye nem egységes, kezdjük a keresést a `/usr/lib/kbd` könyvtár környékén, ha *Red Hat* vagy hozzá hasonló disztribúciónk van. A *consolefonts* könyvtárat kell megtalálnunk, és ekkor próbálkozhatunk többféle kinézettel is. A fontok neveit, a példából is láthatóan, nem kell végig beírni.

Az egér használható szöveges módban is, a rendszergazdának kell jól beállítania, a *gpm* nevű háttérprogram (démon) kezeli. Megoldható vele többek között szövegek kijelölése és átvitele más terminálra: egyszerűen ki kell jelölni lenyomott bal gombbal a kívánt szöveget, majd átváltani a célterületre, és ott a jobb egérgommbal kattintva be lehet illeszteni (az is lehet, hogy 3 gombos egér esetén máshogy működik, olvassuk el a *gpm* manuálját).

A *DOS* alatt sem ismeretlen fogalom a környezeti változó. Ez egy névvel azonosított szöveges információ, egy vagy több program megfelelő futtatásához tartalmaz adatokat. Leggyakrabban az elérési útvonalak vagy a készenléti jel beállításánál találkozunk velük – a *DOS* alatt, mert a Linuxban jóval nagyobb szerepet kapnak. Erről mindjárt meggyőződhetünk, ha (innentől végig *bash* héjprogramot feltételezve) kiadjuk a *set* parancsot. Rengeteg, jórészt nagybetűvel kezdődő változónevet látunk ... vagy nem? Akkor máris meg kell említeni a konzolnak ama jó tulajdonságát, amivel a szöveges képernyő tartalmát „visszagördíthetjük”, a gyorsan elszaladó szöveg újra láthatóvá tehető: nyomjuk le a *SHIFT-Page Up* billentyűkombinációt néhányszor! Amíg videokártyánk memóriája bírja, tárolja az előző képernyőtartalmakat. A *SHIFT-Page Dn* persze lefelé lapoz. Így már biztosan látjuk a minket érdeklő információt. Nézzünk néhányat azok közül, amit biztosan máris tudunk értelmezni:

**HOME** a felhasználó alapkönyvtára;

**HOSTNAME** a gép Internetes neve;

**HISTFILE, HISTSIZE** a kiadott parancsainkat rögzítő fájl helye és maximális mérete (ha nem gépelünk sokat, még a néhány héttel ezelőtt kiadott utasításaink is megtalálhatóak itt);

**KEYTABLE** a billentyűzetkiosztás rövid jele (nem minden disztribúcióban van meg);

**PATH** a végrehajtható fájlok elérési útvonalai;

**PS1** az elsődleges készenléti jel formátuma;

**SHELL** az aktuális parancsértelmező;

**TERM** a terminál típusa;

**UID, USERNAME (vagy USER)** felhasználói azonosító kód és név.

Látható tehát, hogy nagyon sok környezeti változónk van, de ezeken felül még bármikor hozhatunk létre újabbakat:

```
[root:~]>export valami="semmi"
[root:~]>echo $valami
semmi
```

A példából két dolog derül ki: az `export` változónév=érték segítségével lehet létrehozni, az `echo $változónév` segítségével pedig kiírni a változót. Az `echo` parancs még elő fog kerülni, már most látszik a szerepe: szövegek, szöveges változók kiírása. Azt, hogy most egy környezeti változót kell kiírnia, a `$` jel jelzi. Nézzük meg, mit fog csinálni a következő parancs:

```
[root:~]>export valami=
```

Ellenőrizve az előbbi módon, a változónk értéke most már tényleg a semmi, pontosabban az üres sztring, de azért a neve még szerepel a többi változó között.

Többször emlegettük már a készenléti jelet vagy promptot, ami nem más, mint egy rövid szöveges üzenet, melyben a gép közli, hogy készen áll parancsaink fogadására, illetve tájékoztató adatokat nyújt. Közvetlenül utána ott villog a kurzor.

Gyakorlásképpen a készenléti jelet állítsuk be a példákban látható formátumra, tehát mutassa a felhasználói nevünket és az aktuális könyvtárat szögletes zárójelek közt, kettősponttal elválasztva, de most kövesse egy `#` jel:

```
[root:~]>export PS1='`[\u:\w]#`'
[root:~]#
```

A lehetséges beállításokról a `bash` kézikönyv-oldalain olvashatunk.

## 1.8. Állománykezelő parancsok

Adatainkat a számítógépen valamilyen rendszer szerint kell tárolnunk, hogy később is fel tudjuk őket használni. Említettük már, hogy a Linux az adatállományokat, a fájlokat egy könyvtárrendszerben tárolja, ennek kiindulópontja a gyökérkönyvtár (*root directory*). Az egyes adathordozókon (merev- és hajlékonylemez, CD-ROM, stb.) található adatokat a rendszer ebbe illeszti be, ez a folyamat (idegen szóval `mount` olás, magyarul

befűzés vagy beillesztés) bizonyos adathordozóknál már a rendszer indításakor megtörténik, de a cserélhető típusúaknál menet közben kell ezt elvégeznünk. A következő fejezetben még részletesebben is visszatérünk erre a témára.

Arról is volt szó, hogy a UNIX (így a Linux) is gyakorlatilag mindent fájlként kezel. Számára egy fájl olyan adattárolási eszköz, amibe byte-ok formájában adatot tudunk beírni, illetve belőle kiolvasni. Ennek a végletekig leegyszerűsített modellnek a következetes alkalmazásával dolgozik a rendszer. Ez magyarázza azt, amiről már történt említés: fájlként kell kezelnünk egy bináris kódot tartalmazó programot, egy kép- vagy szövegfájlt, de fájlként lehet felfogni magát a könyvtárat is, sőt, azt a lemezipartíciót is, amin a gyökérkönyvtár elhelyezkedik. Ennek megfelelően több típusú fájlal dolgozhatunk:

**Normál fájl** – közönséges, adatot tartalmazó állomány.

**Könyvtár** – olyan fájl, amiben további állományok tárolódnak.

**Link** – egy, a Windows '9x parancsikonzaihoz hasonló típus, két fajtája is van: *hard link* és *soft* (vagy szimbolikus) *link*.

**Karakteres eszköz (character device)** – olyan speciális fájl, amin keresztül valamilyen hardvereszközhöz férhetünk hozzá mint fájlhoz; tipikus példája valamilyen soros port vagy a PS/2 csatlakozó.

**Blokk eszköz (block device)** – az előbbihez hasonlóan speciális fájl, amivel olyan eszközöket tudunk kezelni, amik egyszerre nagyobb mennyiségű adatot tudnak átvinni, példa erre a merevlemez (ez utóbbi két fájltypust az `mknod` paranccsal mi magunk is létrehozhatjuk).

**Csővezeték (named pipe)** – egy úgynevezett first-in-first-out (*FIFO*) típusú fájl; egyszerre több program írhatja és olvashatja a tartalmát, így adatokat tudnak cserélni egymás között. Az `mkfifo` `pipe`-név paranccsal hozható létre.

**Socket** – elsősorban a hálózati kommunikáció során létrejövő fájl.

A mindennapi munka során általában az első három típussal van dolgunk, nézzük át a kezelésükkel kapcsolatos parancsokat!

## 1.9. A könyvtárkezelés parancsai

A Linux könyvtárfája első ránézésre sokkal bonyolultabb, kiterjedtebb, mint például a *DOS*-é. Idővel azonban rájöhethetünk, hogy mindennek megvan a jól meghatározott helye, ezt ráadásul egy *File Hierarchy Standard* nevű szabvány igyekszik is rögzíteni. Ennek ellenére, mivel nagyon sok UNIX verzió van, és a Linux disztribúciók is eltérhetnek egymástól, csak a főbb szabványos könyvtárakat és tartalmukat tekintsük át:

**/boot** a rendszerindításhoz szükséges állományok, betöltésvezérlők adatait tartalmazza.

**/bin** azokat a bináris programokat találjuk itt, amikre a betöltés korai fázisában már szükség van, és később is alapvetőek.

**/dev** a hardvereszközkhöz hozzáférést biztosító karakteres és blokk eszközök tárhelye.

**/etc** a konfigurációs állományok központi lelőhelye.

**/home** a felhasználók saját könyvtárainak bejárat pontja.

**/lib** a tárgyalkönyvtárak helye. Ezek nagyjából a *Windows*-ból ismert *DLL* fájlokhoz hasonló szerepet töltenek be.

**/mnt** a cserélhető vagy később beillesztett meghajtók tartalma található meg ennek könyvtáraiban, például `/mnt/cdrom`.

**/proc** különleges tartalmú könyvtár, benne a működő rendszerről látunk információkat.

**/root** a rendszergazda „főhadiszállása”.

**/sbin** általában csak a rendszergazda által futtatható programok.

**/tmp** átmeneti fájlok könyvtára.

**/usr** felhasználói programok fő tárháza, további nagyon kiterjedt könyvtárrendszere van.

**/var** a futás közben keletkezett állományok (például naplófájlok) helye.

A `/usr` alatti főbb könyvtárak közül meg kell említeni a `local`-t, amiben vizsgáljuk az összes fontosabb könyvtárnevet, elsősorban helyi fejlesztésű és/vagy telepítésű programok és azok minden adata kerül ide. Az `X11R6` alatt található a grafikus felület, az *X Window* rendszer futtatásához szükséges állományok szinte minden eleme. A `share` könyvtárai tartalmazzák a több program számára is felhasználható adatokat, például ikonokat, betűfontokat. A `doc` könyvtárról és annak hasznos tartalmáról már esett szó. Az `src` tartalmazza a forrásnyelven meglévő programokat, amiket még le kell fordítani gépi kódra, többek között itt a helye a Linux kernel, vagyis rendszermag forrásának is. Ennek segítségével az operációs rendszer alapvető, központi részét (ami tulajdonképpen maga a Linux, de az elnevezés a többi programmal együtt maradt rajta) újrafordíthatjuk a gépünknek leginkább megfelelő beállításokkal. Ezzel gyorsabb és stabilabbá, nem utolsósorban biztonságosabbá is tehetjük munkaeszközünket. Végül, de nem utolsósorban a `bin` könyvtár tartalmazza a felhasználó által elérhető programok zömét.

Ebben a rendszerben valahogy navigálnunk kell, ennek alapparancsa a `cd`. Ha csak önmagában adjuk ki, akkor is értelmes: a felhasználói könyvtárunkba röpit vissza bennünket, ahol belépéskor is landolunk. A *home* könyvtárunk nevét nem kell mindig teljesen kiírni, elég egy `~` jellel rövidíteni (egyébként a példákban látható készenléti jelben is ezt fedezhetjük fel). Könyvtárnevet megadva paraméterként beléptet oda, ez megadható abszolút módon, de relatívan is. Tehát ha mondjuk a `/usr/doc` könyvtárból akarok eljutni a `/etc` könyvtárba, akkor kétféleképpen is megtehetem:

```
[root:/usr/doc]>cd /etc
```

vagy:

```
[root:/usr/doc]>cd ../../etc
```

az eredmény ugyanaz lesz. Arra oda kell figyelnünk, hogy a slamos `cd .` parancskiadás itt hibához vezet!

Unjuk a sok gépelést? Vegyük igénybe a `bash` azon jó tulajdonságát, hogy ki tudja helyettünk egészíteni azokat a parancs- vagy fájlneveket, amelyek egyértelműek. Tegyük fel, hogy a `/tmp` alatt vagyunk, és szeretnénk belépni a `/usr/local/src` könyvtárba. Kezdjük el begépelni a parancsot:

`cd /u` – de itt álljunk meg, és nyomjuk le a tabulátor billentyűt! Ha a főkönyvtárban nincs több `u` betűvel kezdődő alkönyvtár, akkor már ki is egészül a parancssor így: `cd /usr/`. Ha most ismét tabulátort nyomunk, az első alkalommal nem történik semmi látványos, esetleg egy hangjelzést hallunk, de másodikra felsorolja az összes lehetőséget, amiből választani lehet (ha túl sok lenne, akkor rákérdez, hogy mindet kilistázza-e). Most nyomjunk egy `l` betűt, újra `TAB`ot, ekkor már kevesebből kell választani, és így tovább haladva értelemszerűen, gyorsabban be tudjuk fejezni a dolgot. Akkor különösen gyors ez a módszer, ha csak egyetlen alternatíva van. Hosszabb nevű parancsok kiadását is fel lehet gyorsítani így, például a már ismert `hostname` parancsot a `ho-TAB-n-TAB` billentyűleütésekkel is elő lehet varázsolni.

Meglévő könyvtárakban már otthonosan mozgunk, hozzunk létre mi is újakat. A `mkdir` könyvtárnév parancs lesz ebben segítségünkre. Vele, valamint a `mkdirhier` parancssal sem csak egyetlen könyvtárat, hanem egész alkönyvtárrendszert létre tudunk hozni. Nézzük, hogyan:

```
[root:~]>mkdir -p egy ketto harm/a/magyar/igazsag ; ls -R egy ketto harm
egy:
harm/:
a
harm/a:
magyar
harm/a/magyar:
igazsag
harm/a/magyar/igazsag:
ketto:
```

Látható, hogy több egymás után következő parancs pontosvesszővel elválasztva egymástól, egy sorban is kiadható. Magyarázatként még annyit, hogy az `mkdir` akkor hozza létre az alkönyvtárakat is, ha a `-p` paramétert is megadjuk, viszont egyszerre több könyvtárnevet is felsorolhatunk. Az ellenőrzésképpen kiadott `ls -R` pedig rekurzívan, vagyis az alkönyvtárakba is belépve listázza ki azok tartalmát, minden alkönyvtárnál kettőspont után sorolja fel a neveket.

A könyvtárak törlésére az `rmdir` használható, a könyvtárnak üresnek kell lennie. Ha azonban egy alkönyvtár-ág már üres, a hierarchiában legalsóval kezdve ki lehet törölni, csak a `-p` paraméter kell. Az előbbi példára visszatérve:

```
[root:~]>rmdir -p harm/a/magyar/igazsag
```

hatására az egész ág törlődik, mivel egyenként mindegyik üres.

A fájlok teljes elérési útjával kapcsolatos két parancs a `basename` és a `dirname`. Az előbbi a paraméterében megadott teljes útvonalból a könyvtárat jelentő részt hagyja el, míg a második éppen ezt hagyja meg. Példaként nézzük a `/usr/bin/ar` fájlt:

```
[root:~]>basename /usr/bin/ar
ar
[root:~]>dirname /usr/bin/ar
/usr/bin
```

Itt ismét alkalmazhatunk egy kis gépelést megtakarító trükköt: a héjprogramunk képes arra, hogy tárolja az előzőleg kiadott parancsainkat. A *FEL* és a *LE* billentyűkkel lépegethetünk ezek között, tehát az előbbi esetben a második parancs beírása előtt csak egy *FEL* billentyűt kell nyomni, majd a *base* szövegrészt átjavítani *dir*-re. Természetesen egy soron belül használható a *HOME* és az *END* billentyű is.

## 1.10. Fájlkezelő parancsok

Fájlokkal több mindent kell csinálnunk. Először is, ha kíváncsiak vagyunk egy adattáblomány típusára, mivel a kiterjesztésből nem tudunk esetleg rájönni, adjuk ki a `file` *filenév* parancsot! A program egy `magic` nevű fájl adatait veti össze az adott állomány elején található adatokkal, és ez alapján adja meg a típust. Ezt a fájlt mi magunk is ki tudjuk egészíteni újabbakkal, ha ismerjük, milyen byte-sorozat jellemző a felvenni kívánt fájltypus elejére. Próbáljuk ki néhányszor:

```
[root:~]>file /usr/bin/as
/usr/bin/as: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked (uses shared libs), stripped

[root:~]>file /etc/XF86Config
/etc/XF86Config: ASCII text
```

Az első esetben egy programot adtunk meg, mégpedig egy assembler fordítóprogramot. Az adatokból kiderül, hogy úgynevezett *ELF* típusú bináris (ez a Linux jelenleg elfogadott típusa, kezdetben az *a.out* formátumot használta), 386-os processzort igényel. Dinamikusan szerkesztett, megosztott tárgykódkönyvtárakat használ, ami azt jelenti, hogy nem minden funkciója van fixen a programba fordítva, hanem bizonyos ismétlődő vagy más programokban is előforduló kódrészleteket ezekből a *lib*-ekből vesz futás közben. Ezzel a programok méretét és a fejlesztésükre szánt időt is csökkenteni lehet, nem szólva a mindig értékes memóriáról és a tárhelyről. A *stripped* jelentése pedig az, hogy a hibakereséshez szükséges, jól működő program esetén felesleges kódrészletek hiányoznak belőle. A második példa ennél jóval egyszerűbb: sima *ASCII* szöveges állomány.

Maradjunk ez utóbbinál, és nézzük meg, mit tartalmaz a fájl. Több módon is meglehetjük ezt. Vegyük először elő a `cat` parancsot! Ennek elsődleges célja több fájl összefűzése (*conCATenate*) volt, de sokszor inkább arra használjuk, hogy fájlok tartalmát a képernyőre írassuk vele. Vigyázzunk, ha bináris fájlt adunk meg paraméterül, azt is kiírja, de mivel ilyenekben nem csak szöveges, hanem vezérlő karakterek is előfordulhatnak, sok jóra ne számítsunk!

Hosszabb szövegek gyorsan elfutnak a képernyőről, sokszor a már ismert *SHIFT-Page Up* sem segít, ilyenkor jön jól a `more` vagy a `less`. Az előbbi ismerős a *DOS*-ból is, ugyanúgy működik: vagy egy szűrőkarakterrel (`|`) egy másik program kimenetét adjuk az ő bemenetére, vagy csak egyszerűen megadunk neki egy paramétert:

```
[root:~]>cat /etc/XF86Config | more
Section "Files"
RgbPath "/usr/X11R6/lib/X11/rgb"
ModulePath "/usr/X11R6/lib/modules"
FontPath "/usr/X11R6/lib/X11/fonts/75dpi:unscalled"
...
--More--(19%)
```

Az alsó sorban látható, hogy még nem mutatja az egész fájlt, egy billentyű leütésére hozza következő képernyőoldalt.

A `less` (bár neve nem ezt sugallja, ugyanis „kevesebb” a jelentése) többet tud ennél. Legfontosabb különbség, hogy visszafelé is lehet vele a szövegben lapozni, de ezen felül még keresni is tud adott szövegdarabot. Futása közben egy `/` jelet ütve meg kell adnunk a keresendő szöveget, és már rá is áll az első találatra. Az `n` billentyűvel az esetleges következő előfordulásokat is gyorsan megkereshetjük. A kilépés belőle a `q` billentyűvel történik, de előtte még egy `h` lenyomása is tanácsos, hogy végignézhessük, mi mindenre képes.

Egy szövegből gyakran csak az első pár sor, vagy éppenséggel csak a vége érdekel bennünket. Erre az esetre célszerű szerszám a `head` illetve a `tail`. Kipróbálásukhoz készítsünk egy pár soros szöveget, mégpedig meglepő módon a `cat` felhasználásával! Ugyanis ez úgy működik, hogy ha nem adunk meg kiíratni valót, akkor úgy veszi, hogy az alapértelmezett bemenetről (*STDIN*) fog érkezni a jelsorozat, és ez a billentyűzet jelen esetben. A kimenettel is bűvészkedünk, mert azt nem a képernyőre (*STDOUT*) küldjük, hanem átirányítjuk egy fájlba. Tehát:

```
[root:~]>cat > proba.txt
Boci, boci tarka
Se füle
Se farka
Oda megyünk lakni
Ahol tejet kapni!
```

Ha közben hibázunk, és esetleg a visszatörlés nem működik, próbáljuk ki a *CTRL-H* kombinációt, ez a *Backspace* funkcióját látja el. Amikor végeztünk a gépeléssel, nyomjunk az utolsó üres sorban *CTRL-D*-t, ez a legtöbb parancs számára az adatbevitel végét jelenti. Most lássuk, mi a szöveg első két sora:

```
[root:~]>head -2 proba.txt
Boci, boci tarka
Se füle
```

majd pedig nézzük meg a végét:

```
[root:~]>tail proba.txt
Boci, boci tarka
Se füle
Se farka
Oda megyünk lakni
Ahol tejet kapni!
```

Mivel alapértelmezés szerint mindkét program 10-10 sornyi adatot ír ki a szövegből, és próbánk ennél rövidebb, ezért teljes egészében viszontlátjuk.

A `tail` egy nagyon érdekes trükkre is módot ad. Olyan állományok végét is képes figyelni, amelyek folyamatosan bővülnek. Ilyenek például a naplófájlok, amikből a `/var/log` könyvtár tartalmaz jónéhányat. Rendszergazdaként ezekhez biztosan van hozzáférésünk, ha nincs, a kipróbáláshoz például a `wget` nevű Internetes letöltőprogram naplófájlja is megfelelő. Feltéve, hogy a naplófájl neve `naplo.txt`, adjuk ki a `tail -f naplo.txt` parancsot! Ekkor azt látjuk, mintha a fájl eleve a képernyőre íródna, tehát menet közben bele tudunk nézni a folyamatba.

Az előbbiekben a `cat` parancsnál már láttuk, hogy lehet szövegszerkesztő nélkül is létrehozni fájlt. Néha szükség van arra is, hogy 0 byte méretű adatállományt készítsünk. A `touch` parancs kiválóan alkalmas erre a célra. Eredeti funkciója az, hogy egy fájl legutóbbi hozzáférési és módosítási dátumát az éppen aktuális időpontra változtassa meg. Azonban ha a megadott fájl még nem létezik, akkor létrehozza azt, a fent említett módon.

Ideje kicsit bővebben szólni a már említett linkekről. Alkalmazásuk fő célja a hely-megtakarítás, hiszen egy linkkel tulajdonképpen ugyanarra a fizikai adatállományra tudunk hivatkozni több néven és több helyről. A hard link vagy kemény link segítségével ugyanazon a fájlrendszeren belül tudjuk ezt megtenni, és csak fájlokra működik. A soft link, vagy lágy link illetve szimbolikus link ennél rugalmasabb, több fájlrendszeren át is él, és könyvtárakra is alkalmazható. Lássuk, hogyan használjuk őket!

Készítsünk egy könyvtárat `linkproba` néven, lépünk bele, majd gépeljük be következőket:

```
[root:~/linkproba]>touch csingi.link
[root:~/linkproba]>ls -l
total 0
-rw-r--r--    1 root    root          0 Jul  8 15:38 csingi.link
```

Létrejött tehát a 0 byte-os fájl, a hozzáférési adatai után következő második oszlopban egy 1-est látunk. Használjuk most a link-készítés parancsát, az `ln-t`, és nézzük meg, mi történik:

```
[root:~/linkproba]>ln csingi.link ezuj.link
[root:~/linkproba]>ls -li
total 0
 20071 -rw-r--r--    2 root    root          0 Jul  8 15:38 csingi.link
 20071 -rw-r--r--    2 root    root          0 Jul  8 15:38 ezuj.link
```

Elemezzük a látottakat: az `ln` használatakor először a régi, már létező állomány nevét kell megadni, majd a készítendő linkét. Ez itt most hard link-ként jött létre, tehát gyakorlatilag ugyanazt a fizikai adatállományt illethetjük most már két névvel is. Hogy valóban ugyanarról a fájlról van szó, azt látjuk abból, hogy minden adata tökéletesen megegyezik: nemcsak a méret, a dátum és idő, de még az ún. *inode*-szám is, ami azt jelenti, hogy a merevlemezen ugyanott található a két fájl. Ez pedig csak úgy lehet, ha azonosak. Ami még feltűnő, hogy a fent említett, itt már harmadik oszlopban szereplő láncolási szám is megnövekedett 2-re, jelezve, hogy ez a fájl valóban már két néven is elérhető. Ha most galád módon letöröljük a `csingi.link` állományt, a benne lévő adatok nem vesznek még el (amúgy se vesztettünk volna sokat...), hiszen megmaradt az `ezuj.link` néven való hivatkozás lehetősége. Ezzel a módszerrel tehát fontosabb adatainkat védhetjük is véletlen törlés ellen.

Ennél gyakrabban használjuk a szimbolikus linkeket. Használatukkal nem kell egy fájl vagy alkönyvtárat átmásolni egy másik helyre, elég csak a linkeket kialakítani. Nézzük először az egyszerűbb esetet, készítsünk az `ezuj.link` fájlra egy `szim.link` nevű linket:

```
[root:~/linkproba]>ln -s ezuj.link szim.link
[root:~/linkproba]>ls -li
total 0
 20071 -rw-r--r--    1 root    root          0 Jul  8 15:38 ezuj.link
 20068 lrwxrwxrwx    1 root    root          9 Jul  8 15:54 szim.link -> ezuj.link
```

Az `ln-t` most el kell látni egy `-s` paraméterrel, egyébként a használata ugyanaz. Kilistázva a dolgainkat több újdonság látható. A fájlok már nem ugyanazok fizikailag, hiszen *inode*-számuk különbözik. A másodiknál a hozzáférések oszlopában látható a legelső 1 betű, ez utal a fájl típusára (egyébként – a normál fájl, *d* a könyvtár, *c* a kataliteres, *b* a blokkos eszközfájlokat takarja). Ezen kívül látványosan kijelzi az `ls` azt is, milyen viszony van a két fájl között, a `szim.link` tulajdonképpen az `ezuj.link` fájlra „mutat”. Érdekes módon a 0 byte-os fájlra való linkelés is 9 byte-os méretet eredményezett<sup>1</sup>, de lehetne az eredeti fájl 1 gigabyte is, a link akkor is ennyi

<sup>1</sup>Ez azért van így, mert az eredeti fájlnev 9 karakteres volt, és a szimbolikus link valójában ezt tárolja.



lenne nagyságrendileg – és itt már valóban jelentős a helymegtakarítás. Az előbbi módon, ha most letöröljük az eredetit, akkor a link ugyan megmarad, de elárvul, viszont bármikor „újraéleszthető” azzal, hogy létrehozuk a hozzá tartozó célfájlt, vagy, mivel könyvtárakra is alkalmazható, célkönyvtárat.

A fájlokkal végzett leggyakoribb műveletek a másolás, mozgatás, átnevezés és törlés. A másolásra a `cp` utasítás használatos, a máshol már megszokott módon a forrás, majd a cél megjelölésével, és persze esetleges kiegészítő paraméterekkel. Meg kell említeni, hogy a `-d` paraméter megtartja a linkek kapcsolatát, a `-p` megőrzi a fájlok attribútumait, a `-R` rekurzívan, tehát az adott könyvtárárat bejárva másol, és mivel ezeket gyakran együtt alkalmazzuk, a `-a` egyedül is megfelel mindezen kapcsolók hatásának. Olyan esetben, amikor egy fájlról másolatot akarunk készíteni valahol, de más néven, nem kell az átnevezéshez külön parancs. Készítsünk például az `XF86Config` fájlról egy biztonsági másolatot. Ez nagyon jó ötlet minden olyan esetben, ha konfigurációs állományokat akarunk módosítani. Túl jól sikerült „módosítás” miatt lehet, hogy az adott program nem is indul el, ekkor legalább az eredeti példány megmarad:

```
[root:/etc]>cp ./XF86Config ./XF86Config.orig
```

A példából kiderül, hogy ha abban a könyvtárban állunk ahol a fájl is van, az aktuális könyvtárra utaló `.` könyvtárbejegyzést használhatjuk az elérési út megadásához. Használhattuk volna a `/etc/` megadást is, de lusták voltunk gépelni... Aki igazán lusta, az pedig tudja, hogy teljesen elhagyható az útvonalmegadás, ha a szóban forgó könyvtárban állunk.

A `dd` használatával is másolási műveleteket végezhetünk, de ezt inkább akkor használjuk, ha valamely állomány byte-onkénti átmásolását kell elvégezni. Tipikus alkalmazási területe indítólemezek készítése, ahol a rendszermagot tartalmazó fájlt kell kiírni a floppyra:

```
[root:~]>dd if=kernel of=/dev/fd0 bs=1k
353+1 records in
353+1 records out
```

A fenti példában a másolandó fájl neve `kernel`, a kimenet a `/dev/fd0` eszköz, ez az első floppy-meghajtó, és a másolás 1 kB-os blokkonként zajlik (megjegyzendő, hogy ez még nem elég működő indítólemez készítéséhez, használjuk inkább az `mkbootdisk` parancsot).

Az átnevezés, és egyben a mozgatás parancsa is az `mv`. A megfelelően kiadott utasításból úgyis kiderül, mi a teendő. Nézzük azt az esetet, amikor a `home` könyvtárunkban található `proba.txt` állományra kiadjuk a parancsot, feltéve, hogy `proba.txt.uj` nevű fájl vagy könyvtár nem létezik itt:

```
[root:~]>mv proba.txt proba.txt.uj
[root:~]>ls p*
proba.txt.uj
```

Láthatóan új nevet kapott a fájl. Nevezzük most vissza, majd készítsünk egy `proba.txt.uj` nevű könyvtárat, és adjuk ki ismét az előbbi parancsot. Ugye tudjuk, hogy nem kell újra begépelni?

```
[root:~]>mv proba.txt.uj proba.txt
[root:~]>mkdir proba.txt.uj
[root:~]>mv proba.txt proba.txt.uj
[root:~]>ls proba.txt.uj/
proba.txt
```

Az mv most átmozgatást csinált, hiszen volt ilyen nevű könyvtárunk, amit célként kezelt.

A törlés mindig és mindenhol veszélyes művelet, hiszen adatvesztéssel jár. Néhol még van esélyünk a meggondolatlanul törölt állományok visszaállítására (lásd a *DOS undelete* parancsa), de jól jegyezzük meg: A UNIX-BAN A TÖRÖLT ADATOK VISSZANYERÉSÉRE NAGYON KEVÉS ESÉLY VAN! Igaz, ugyan, hogy léteznek módszerek a Linux fájlrendszerében, az ext 2-ben is arra, hogy ha szerencsénk van (és süt a nap, valamint nem 13-a van), akkor visszaállíthatjuk a törölt fájlt, de ez nem egyszerű. Olyan gépen, ahol viszonylag nagy az adatforgalom, sok felhasználó dolgozik, vajmi kevés esélyünk van erre. Mindezt azért hangsúlyozom ennyire, mert Murphy törvényei alapján biztosan azt a fájlt töröljük le, amit nem kéne, tehát előtte jól gondoljuk meg, mit csinálunk! Ha mindezek ellenére néha mégis erre adjuk a fejünket, akkor az `rm` parancsot használjuk. Paraméter nélkül jólnevelten rá is kérdez minden esetben, hogy valóban törölni akarjuk-e a fájlt, ha nagyon biztosak vagyunk a dolgunkban, akkor használhatjuk a `-f` kapcsolót, ezután csak végzi csendben a munkáját. Különösen csínján kell bánni a `-r` paraméterével, ami (lassan már leírnom se kell), rekurzívan töröl. Aki igazán paranoiás, azt az előbb elhangzottak sem nyugtatják meg. Neki ajánlható a `shred` parancs, ami igyekszik úgy törölni az állományt, hogy valóban senki se tudja azt reprodukálni: többször is felülírja azt a lemeztartományt, ahol a fájl volt.

## 1.11. Tulajdonjogok, hozzáférés szabályozása

Többfelhasználós operációs rendszerekben minden fájl rendelkezik olyan adatokkal, amik alapján eldönthető, ki és mit csinálhat vele. Mindegyiknek van egy tulajdonosa, aki létrehozta és rendelkezik felette. A tulajdonos mindig egy csoportnak is tagja, és ezeket a csoporttagokat megkülönböztethetjük a többi felhasználótól, a „külvilágtól” a hozzáférési jogok tekintetében. Így három csoport alakul ki: maga a tulajdonos (User), az ő csoportja (Group), és az összes többi felhasználó (Others). Egy adatállománnyal leggyakrabban a következő három műveletet csinálhatjuk: írhatjuk, olvashatjuk, és a programokat még végre is hajthatjuk (Read, Write, Execute). Ezeket összevetve nyer értelmet a már sokszor látott oszlop az `ls` kimenetében, ahol például azt láthatjuk:

```
-rw-r--r--    1 pistike  users          559 Jul  8 17:41 .bashrc
drwxr-xr-x    7 pistike  users        4096 Jul  8 17:41 .kde/
```

Az első állomány, a `.bashrc` normál fájl, erre utal a legelső `-` karakter, tulajdonosa a `pistike` nevű felhasználó, aki a `users` csoport tagja. A következő három karakter a tulajdonos jogait jelzi: `rw-`, jelentése az, hogy olvashatja és írhatja, és mivel ez nem programfájl, nem kell végrehajtani. A következő hármas a csoportjogoké, itt már csak az olvasási jog adott, ugyanígy az összes többi felhasználó számára is. A `.kde` egy könyvtár, mutatja ezt a `d` betű a legelső helyen, majd ugyanígy hármas tagolásban következik a jogok felsorolása. Könyvtáraknál kicsit máshogy kell értelmezni azonban ezeket a dolgokat: olvasási jog esetén kilistázható a könyvtár tartalma, írási jog esetén módosítható a tartalma, tehát például tudunk benne fájlt létrehozni, végrehajtható attribútum esetén pedig bele tudunk lépni. Jelenleg `pistike` bármit tehet a könyvtárral, a csoporttagok és az „egyebek” viszont csak beléphetnek és listázhatják a tartalmát, létrehozni, vagy pláne törölni nem tudnak semmit. Játsszunk most el ezekkel a lehetőségekkel, és ismerjük meg közben a kapcsolódó utasításokat!

Először is változtassuk meg a tulajdonviszonyokat. Megint készítsünk egy új könyvtárat, jogok néven, lépünk be, és hozzunk létre egy fájlt, de most írunk bele valamit, mondjuk az `echo` paranccsal:

```
[root:~/jogok]>echo Veni, vidi, vici! > julius
```

Kilistázva látjuk, hogy root, a root csoport tagja bitorolja most ezt a fájlt. Ő azonban nagyon jóban van pistikével, és át akarja adni a rendelkezés jogát neki. Mit kell tennie? Lássuk:

```
[root:~/jogok]>chown pistike julius
[root:~/jogok]>ls -l
total 4
-rw-r--r--  1 pistike root          18 Jul  8 18:02 julius
```

A chown a megoldás tehát: első paramétere a felhasználó, akié lesz a második paraméterben megadott fájl. A csoporttagság megváltoztatására a chgrp alkalmas, hasonló módon használva. Adjuk át gyakorlásul a users csoportnak a fájlt. Megjegyzendő a következő: ha átadtuk a tulajdonjogot, akkor vissza már nem vehetjük! Egy kivétel van persze most is, a rendszergazda, de hát ő mindent megtehet...

Most ki kellene próbálnunk pistike nevében bejelentkezve a rendszerbe, hogy mit nyertünk a root barátságával? Itt először tegyünk említést egy olyan lehetőségről, amivel nem muszáj a szokott módon belépni, hanem „menet közben” felvehetjük egy másik felhasználó képét, ez a su paranccsal megy. Ha paraméter nélkül hívjuk meg, akkor a root jelszavát begépelve máris rendszergazdai jogokkal dolgozhatunk tovább, egészen a megszokott kilépési parancsok valamelyikének kiadásáig. Felhasználói nevet és a hozzá tartozó jelszót megadva neki az illető bőrébe bújhatunk. A rendszergazdának még a jelszó begépelésével sem kell bajlódnia. Akárhogy is, most már pistike nevében próbálkozzunk tovább:

```
[pistike:~]>cd /root/jogok
bash: cd: /root/jogok: Permission denied
```

Nem mind arany, ami fénylik, a hozzáférés megtagadva, de ha jobban megnézzük, nem a fájlhoz magához, hanem az őt tartalmazó könyvtárhoz! Segítsünk ezen (persze root-ként) a chmod parancs alkalmazásával. Ez az az utasítás, amivel a hozzáférési jogok állíthatók. Többféleképpen is használható, először nézzük a talán olvashatóbb megoldást:

```
[root:~/jogok]>chmod o+rx .
```

lefordítva: az others (nem a root csoport tagjai) részére engedélyeztük (+) az olvasási és végrehajtási jogot, ez ugye könyvtárak esetén belépési és tartalomlekérési privilégiumot jelent. A . jelenti az aktuális könyvtárat. Viszont még így sem megy a dolog. Jobban megvizsgálva az esetet, rájövünk, hogy azért nem, mivel pistike a jogok/ szülőkönyvtárába, a /root könyvtárba be sem tud lépni! Korrigáljuk ezt (vigyázat, egy élesben működő rendszeren ezzel NE kísérletezzünk biztonsági megfontolásokból):

```
[root:~]>chmod o+x .
```

Most már pistike látja a fájlt, bele is nézhet, de nem nevezheti át, nem törölheti. Persze, hiszen olyan könyvtárban van, amiben neki semmi joga írási műveletekre. Adjuk hát meg neki a lehetőséget a jogok/ könyvtárra kiadott chmod ow+ utasítással!

Dolgozzunk tovább pistike nevében. Állítgassuk a hozzáféréseket a julius fájlra, adjunk meg magunknak minden jogot, a csoportnak az írás-olvasás jogát, a többieknek pedig csak végrehajtási jogot (nem életszagú a példa...):

```
[pistike:/root/jogok]>chmod u+rwx,g+rw,o=x julius
[pistike:/root/jogok]>ls -l
total 4
-rwxrw--x  1 pistike users          18 Jul  8 18:02 julius*
```

Több minden is leolvasható az első sorból: a három felhasználócsoportha egyszerre is megadhatók a jogok, csak vesszővel el kell őket választani egymástól. Ha valamelyiknél pontosan akarjuk beállítani ezeket, tehát függetlenül attól, mik voltak az előző attribútumok, használjuk az egyenlőségjelet, különben lehet, hogy marad vissza olyan jog, amit nem akarunk megtartani. Itt például a csoport olvasási joga eleve megvolt, tehát nem is kellett volna állítani, hasonlóan a tulajdonos írási-olvasási jogát sem. Vagyis az `o=x` jelentése: az „others”, vagyis a csoporton kívüliek joga a könyvtárra pontosan a belépési joggal lesz egyenlő.

Ez a megoldás is jó, de sok esetben egyszerre több jogot is kell állítani, ilyenkor sokat kell gépelni. A másik járható út annak alkalmazása, hogy minden jog megfeleltethető egy oktális, 8-as számrendszerbeli számkódnak, nevezetesen a 4 az olvasási, a 2 az írási, az 1 a végrehajtási jogot jelenti, és ezek közül a megfelelőeket összegezve alakul ki egy adott felhasználócsoporthoz pozíciójában az a kód, amit meg kell adni. Állítsuk be ezek figyelembevételével a fájlt úgy, hogy a tulajdonos írni-olvasni tudja, a csoport csak olvasni és végrehajtani, a többiek pedig csak olvasni!

```
[pistike:/root/jogok]>chmod 654 julius
[pistike:/root/jogok]>ls -l
total 4
-rw-r-xr--    1 pistike  users           18 Jul  8 18:02 julius*
```

Számoljunk utána: a 6 a 4+2 (olvasás+írás) összege, ezt alkalmazzuk a tulajdonosra, az 5 a 4+1-ből (olvasás + végrehajtás) következik, ezt adjuk a csoportra, míg az olvasási jog önmagában a 4-es kóddal szerepel a többiekénél. Ami már az előző példában is látható volt, mivel valahol szerepel egy végrehajtási jog is, az `ls` egy csillagot tesz a fájl neve után.

A felsoroltakon kívül is létezik még három olyan jogosultság, amikkel néha találkozunk. Az ún. `setuid` beállítású, célszerűen programfájl a futási ideje alatt azokkal a privilégiumokkal rendelkezik, mint az a felhasználó, akinek a tulajdonában van. Tipikus példa erre a jelszóváltást végző `passwd` program. A felhasználók fontosabb adatait, így a jelszót is a tradicionális UNIX rendszerek egy szöveges, bárki számára olvasható fájlban, a `/etc/passwd` fájlban tárolják. A fájl azonban csak a rendszergazda számára írható – hogyan lehet akkor a jelszóváltást megoldani? Úgy, hogy ha egy mezei felhasználó futtatja a programot, az futási ideje alatt a `root` jogaival fog futni a `setuid` miatt, tehát így már tudja írni is a jelszófájlt. Több helyen is alkalmazzák ezt a trükköt, de sajnos a crackerek is ismerik, és emiatt egy biztonságos rendszerben minél kevesebbre kell szorítani az ilyen programok számát. Hasonló értelmű a `setgid` jog, ami csoportra jelenti ugyanezt. Létezik még ezenkívül az ún. `sticky` jog, amit manapság már inkább csak könyvtárakra alkalmaznak. Segítségével megoldható, hogy az ilyen beállítású könyvtárban csak az tud módosítani, törölni egy fájlt, aki létrehozta. Nézzük meg a `/tmp` könyvtárat, itt találkozunk ezzel a beállítással, gondoljunk bele, miért?

Azt, hogy egy fájl milyen jogosultságokkal keletkezik, befolyásolni lehet az `umask` utasítással, ami a `bash` belső parancsa. Paramétere egy, a fentiekben megismerthez hasonló kód. Kialakításához bitszintű műveletek elvégzésére, tehát kis fej-számolásra van szükség. Általában már az indulásnál illetve a felhasználó belépésekor beállítódik, később ritkán van szükség a módosítására. Bővebb leírása a `bash` kézikönyvében található.

## 1.12. Egyéb állománykezelési parancsok

Néhányszor már biztos feltettünk hasonló kérdést: „hol lehet az a szövegfájl, amiben tegnap megkaptam a hiányzó bizonyítékokat arról, ki ölte meg Kennedyt...?” Ismernünk kell azokat a módszereket, amikkel nem kell aggódnunk ilyen problémák miatt sem. Nézzük először a legfontosabb idevágó parancsot, ez a `find`. Nagyon sokrétűen használható: kereshetünk a fájl neve vagy annak csak egy töredéke alapján, a dátum- és időadatok, de a jogosultsági beállítások szerint is. Leggyakrabban csak a fájl pontos helye nem jut eszünkbe, nevét részben vagy egészen tudjuk:

```
[root:/usr]>find . -name abc*  
./share/vim/syntax/abc.vim
```

Használata tehát: meg kell adni a kiindulási könyvtárat, majd azt, milyen módszer szerint akarunk keresni, és az ehhez szükséges adatokat. Alapesetben rekurzívan bejárja a teljes alkönyvtárat, ezen változtatni a `-maxdepth n` paraméterrel lehet, ahol `n` egy egész szám, ekkor csak `n` alkönyvtár mélységig ás le. Ha időadatok alapján akarunk keresni, a `-name` helyett próbáljuk ki a `-ctime n` illetve a `-cmin n` kapcsolókkal: előbbinél azokat a fájlokat keresi, amiket `n`-szer 24 órája, míg utóbbi azokat, amelyet `n` perce módosítottak. Tehát ha tudjuk, hogy a keresett fájlt 5 perce még szerkesztettük, de már nem tudjuk, hova mentettük (ismét a szenilitás...), próbálkozunk a `find / -cmin -5` paranccsal. A `-` jellel azt fejezzük ki, hogy 5-nél kisebb a vizsgált időintervallum, ha pontosan az ellenkezőjét akarjuk, akkor persze a `+` kell. Ki akarjuk deríteni, hol vannak a fájlrendszerben `setuid` beállítású programok? A megoldás a `find / -perm 4000` kiadása (a `setuid`, `setgid` és `sticky` miatt ugyanis igazából nem csak három, hanem négy jegyből áll a teljes jogosultsági kód; ebben az első helyen álló 4 jelenti a `setuid`, a 2 a `setgid` és az 1 a `sticky` jelzőbitet, a további három már ismert). Keresni tudunk a fájl típusa alapján is, az összes felsorolt típus megadható a `-type` után írt egybetűs kóddal. A `/usr/local` alatt lévő összes könyvtárat kiírathatjuk például így: `find /usr/local/ -type d`

Keresésünket ezeken kívül a méret, a tulajdonviszonyok alapján is elindíthatjuk, kinyomozhatjuk, melyik fájlokhoz nem tartozik felhasználó (valószínűleg amiatt, hogy megszűnt a hozzáférése a géphez), mindezeket ismét minden lehetséges előfordulásban. Lássunk egy keresést olyan fájlokra, amik tulajdonosa nem a 0-s felhasználói azonosítójú felhasználó (aki a rendszergazda egyébként), mégis a `/root` könyvtáron belül vannak:

```
[root:~]>find /root ! -uid 0  
./jogok/julius
```

Igen, most van egy ilyen fájl, de hogy is adtuk ki a keresést? Tagadnunk kell azt, hogy az azonosító 0, ezért eléje egy felkiáltójelet írunk. Ez a megoldás nagyon sok helyen visszaköszön, érdemes megjegyezni.

Keressünk most 30 byte-nál kisebb, a `users` csoporthoz tartozó fájlokat a `/home` könyvtáron belül:

```
[root:/home]>find . -group users -size -30c  
./pistike/.bash_logout
```

Talán a `-size` szorul kis magyarázatra: a `30c` jelenti a 30 byte-nyi adatot (megadhatjuk a méretet blokkban, kB-ban is), a `-` pedig – már tudjuk – azt, hogy ennél kisebb mennyiséget keresünk.

A `find` mindezekén túl nyújt egy olyan lehetőséget is, hogy a megtalált fájlokkal műveleteket végezhetünk. Tegyük fel, hogy a következő a feladatunk: a `file` parancs

segítségével állapítsuk meg, milyen típusúak a `/etc` könyvtárban lévő, maximum 50 byte méretű normál fájlok. Láthatóan több paraméter is kell, ezek nagy része ismerős, a legfontosabb azonban még ismeretlen, ez a `-exec`. Lássuk előbb a gyakorlatban, és aztán elemezzük:

```
[root:~]>find /etc -maxdepth 1 -size -50c ! -type l -exec file {} \;  
/etc/ld.so.conf: ASCII text  
/etc/exports: empty  
/etc/filesystems: ASCII text  
/etc/host.conf: ASCII text  
...
```

Kezdjük tehát: a `-maxdepth 1` garantálja, hogy csak az `/etc` könyvtárban keres, mélyebbre nem megy; a `-size` nem szorul magyarázatra; a link típusúakat ki kell zárunk, ezért kell a `! -type l`; és végül a legizgalmasabb a `-exec`. Első paramétere a minden egyes megtalált fájl végrehajtandó parancs neve, ez itt most a `file`. A `{ }` jelsorozatba helyettesítődnek be futás közben sorban a fájlnevek, és a végén le kell zárunk a parancsot, erre szolgál a `\;`. Ugye, így már nem is olyan bonyolult...?

Nézzük meg most azt, hogy lehet még összetettebb kereséseket végrehajtani. Meg kell határoznunk, milyen fájlok mérete esik 100 és 200 byte közé a `/etc` könyvtárban! Ehhez láthatólag nem lesz elég egyetlen `-size` kapcsoló, mást kell kitalálnunk. Itt jönnek képbe a logikai műveletek, hiszen a kérdést úgy is feltehetnénk, hogy szükségünk van a 100-nál nagyobb és a 200-nál kisebb méretű fájlokra. Vagyis mindkét feltételnek egyidejűleg teljesülni kell, erre használható az *ÉS* logikai függvény, amivel már megfogalmazható a parancs:

```
[root:/etc]>find . -size +100c -a -size -200c -maxdepth 1  
./hosts  
./hosts.allow  
./pwdb.conf  
./amd.net  
./ftphosts
```

Látható, hogy a két `-size` paraméter közé egy `-a` kapcsoló ékelődik, az angol *AND* rövidítéseként, és ez kapcsolja össze a két elemi feltételt egy összetetté. Mivel az alapértelmezés szerint is ilyen logikai kapcsolatot feltételez a `find`, a kapcsoló el is hagyható. A *VAGY* kapcsolat kifejezéséhez egyébként a `-o` használható.

Végezetül lássuk, hogyan szabályozható az, hogy milyen adatokat és hogyan írjon ki a `find`. A használandó paraméter a `-printf`, ami sok helyen visszaköszön, szintén ismerős lehet a C-ben programozóknak – nem más, mint formátumozott kírításra szolgáló függvény. A rengeteg lehetséges opció közül nézzük meg azokat, amivel megoldható a következő probléma: a `/etc` könyvtár üres fájljait kell kírítani fájlnevéhez hozzáférési kód formában. Nézzük:

```
[root:/etc]>find /etc -empty -maxdepth 1 -printf "%p-%m\n"  
/etc/exports-644  
/etc/motd-644  
/etc/opt-755  
/etc/modules-644  
...
```

A `printf` a `date` parancshoz hasonlóan egy formátumsztringet vár idézőjelek között, itt is a `%`-jel és egy utána következő egybetűs kód jelenti a fájl valamely elemi adatát (itt most a `p` a nevét, az `m` pedig az oktálisan adott hozzáférési kódját). Végül, mivel a függvény magától nem emel sort, nekünk kell erről gondoskodni a `\n` (newline, új sor) vezérlőkarakter beiktatásával.

A `find` személyében tehát egy nagyon sokrétű szerszámmal kaptunk, de használható fájlkeresésre más eszköz is. A `locate`, illetve az `slocate` akkor használatos, ha gyorsan akarjuk megtudni egy bizonyos fájl helyét. Próbáljuk máris ki:

```
[root:~]>slocate hungary
/usr/local/so52/share/gallery/flags/hungary1.wmf
/usr/local/so52/share/gallery/flags/hungary2.wmf
```

Más keresési mintát is megadhatunk, ami feltűnő, hogy szinte azonnal, a `find` parancsnál megszokott heves meghajtózörgés nélkül adja az eredményeket. Mindez úgy lehetséges, hogy egy előre elkészített adatbázisban keres, és így nem kell a fájlrendszert végignéznie. Ezt az adatbázist az `slocate -u` hozza létre, de megadható, hogy bizonyos könyvtárakat, vagy akár egész becsatolt fájlrendszereket ne vegyen bele (például a `/mnt/cdrom` tartalma elég gyakran változik, vagy a `/root` tartalma nem túl publikus). A `-e könyvtár1, könyvtár2 ...` hatására a felsorolt könyvtárakat, a `-f frt1, frt2 ...` hatására pedig a felsorolt fájlrendszer típusokat nem nézi végig (ez utóbbi lehet például *iso9660* a CD-ROM-okhoz, *nepfs* a becsatolt Novell Netware kötetekhez). Az adatbázisnak viszont naprakésznek illik lennie, különben mit sem ér az egész, de erről a `cron` vagy hasonló célú rendszerprogramok, démonok gondoskodnak; ezek bizonyos időnként lefutnak, általában ezt a feladatot naponta végzik el.

Ide kapcsolódik még az, hogy ha egy programról nem tudjuk pontosan, hol is található, akkor alkalmazzuk a `which` programnév parancsot, ez megadja (ha tudja) a helyét teljes elérési útvonallal együtt.

Néha szükség van fájlok tartalmának összehasonlítására. Többféle parancs is használható e témakörben, elsőként vegyük a `cmp` nevűt. Próbáljuk ki úgy, hogy a `/etc/host.conf` állományt bemásoljuk egy könyvtárba, csinálunk róla itt egy másolatot, majd ennek valamelyik sorában átjavítunk néhány karaktert (például ha `multi on` szerepel benne, akkor azt `multi off`-ra). Lássunk egy lehetséges eredményt:

```
[root:~/jogok]>cmp host.conf host1.conf
host.conf host1.conf differ: char 25, line 2
```

A másodikként megadott fájl tehát először a 2. sorban, a 25. karakterpozíción tér el az elsőtől.

A `diff` már jóval több lehetőséget ad. A programfejlesztők hasznos eszköze, mivel legtöbbször ennek a segítségével állítják elő az ún. patchfájlokat, amikkel egy program forrásnyelvi kódját lehet módosítani, javítani. A megfelelő javítások után már csak a különbségeket tartalmazó állományt kell hordozni, és nem a teljes, sokszor több MB-os forrást. A `patch` program tudja aztán a különbségfájl adatainak felhasználásával a módosítást végrehajtani. Nézzünk végig most egy ilyen műveletet, igaz nagyon egyszerű lesz, de az elv talán ebből is látszik. Először győződjünk meg róla, hogy az előző példabeli `host.conf` fájlban valóban a `multi on` sor szerepel, a `host1.conf` fájlban pedig `multi off`. Célunk az, hogy az eredeti fájlban is lecseréljük erre a szöveget. Ehhez adjuk ki a

```
diff -u host.conf host1.conf > javitas
```

parancsot. A `-u` egy, a `patch` program által elfogadott formátumot produkál, ezt „belepumpáltuk” a *javitas* nevű fájlba. Végül használjuk a *patchet*:

```
[root:~/jogok]>patch -p0 < javitas
patching file host.conf
```

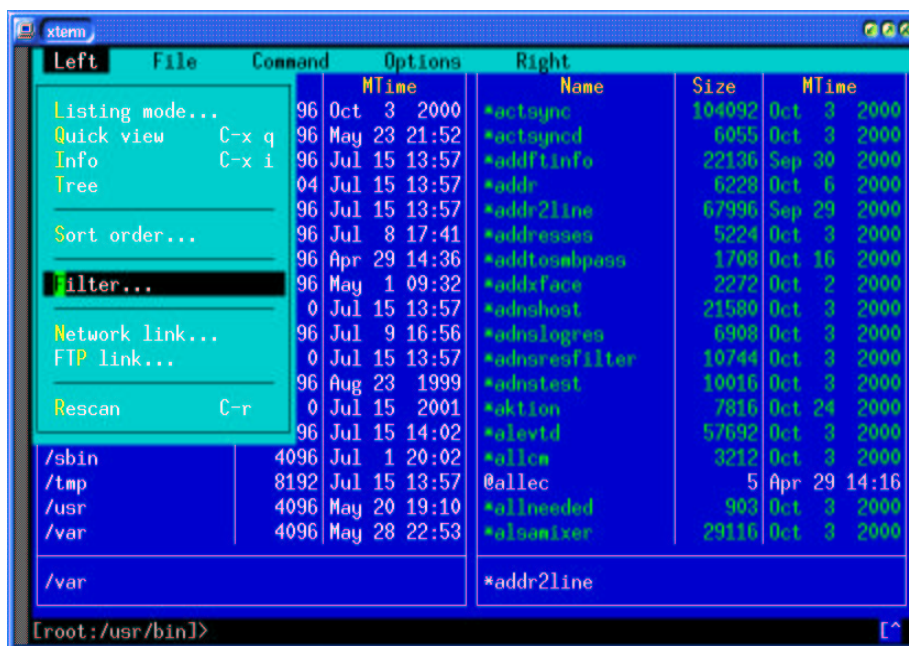
A példában a leggyakrabban kiadott kapcsolót, a `-p0` opciót használtuk, és az is látható, hogyan kell egy programnak megmondani, hogy ne az alapértelmezett bemenetről, hanem egy fájlból várja az adatokat: a `<` jel segítségével. Az üzenet szerint a

„foltozás” (patching) megtörtént, most már a `host.conf` is a `multi off` sort tartalmazza. A `diff` ezen felül nem csak egy-egy fájlt, hanem akár egész könyvtárakat tud összehasonlítani a `-r` kiadásával. Másoljuk le példának okáért a jogok/ könyvtárat `jogok1/` néven, de hagyjuk ki ebből a könyvtárból a `javitas` nevű fájlt. Ekkor:

```
[root:~]>diff -r jogok1 jogok
Only in jogok: javitas
```

vagyis az üzenet szerint csak a jogok/ könyvtárban szerepel a `javitas`.

Aki már dolgozott a *DOS/Windows* rendszerek alatt a *Norton Commander* programmal vagy annak klónjaival, biztos értékelni fogja a *Midnight Commander* programot, ami igyekszik megkönnyíteni a fájlokkal végzett mindennapi munkát. A fontos billentyűkombinációk megegyeznek a *Norton*-féle programban használtakkal, természetesen a két rendszer eltérései miatt vannak mások is (például a meghajtóváltás a Linux alatt nem létezik). Indítása az `mc` begépelésével, a kilépés az `F10` funkcióbillentyűvel (vagy az `exit` begépelésével) történik, a két lépés közötti lehetőségeket azonban inkább próbáljuk ki magunk. Az 1.2. ábrán a 4.5.51-es verziójú változat látható működés közben.



1.2. ábra. A Midnight Commander

Mivel valószínűleg még egy ideig ismertek lesznek a *Microsoft*-féle rendszerek, van egy olyan parancscsoport, ami az ott használatos fájlkezelési utasításokat igyekszik minél jobban utánozni. Ezek mindegyike az `m` betűvel kezdődik, és utána az ismert parancsnevek használhatók, például `mdir`, `mcd`, `mcopy`, `mdel` a szokásos módon. Használatuk előtt a rendszergazdának illik a `/etc/mtools.conf` fájl segítségével megfelelően beállítani a gépen található meghajtók adatait.



## 1.13. Állományrendszerekkel kapcsolatos parancsok

### 1.13.1. Fájlrendszerek kezelése

A Linux fájlrendszere a Second Extended, vagy rövid nevén `ext2` rendszer. Mindenképpen említést érdemelnek azonban az újítások, így főként a naplózó fájlrendszerek. Az érdeklődőknek annyit ezekről, hogy használatukkal a háttértárat még jobban kihasználva, még biztonságosabban lehet adatainkat tárolni. Ilyen például a `Reiserfs`, amely a 2.4.x kernelekben már benne van. A naplózás miatt egy esetleges lefagyás utáni újrainduláskor sem kell a lemezellenőrző `fsck` lefutására várni. Főbb tulajdonságai között meg kell említeni a hosszú fájlnevek használatának lehetőségét, a töredetességre, fragmentációra való nagyfokú immunitást, a jó helykihasználást, és a hálózatos munkához való nagyon jó illeszkedést. Tekintsük röviden végig, milyen módon lehet egy ilyen fájlrendszert feléleszteni és karbantartani főleg rendszergazdai oldalról nézve, de az egyszerű felhasználó által használható dolgokat is felsorolva.

Az `ext2` ugyan hajlékonylemezen is létrehozható, igazi erejét azonban egy merevlemez partíción mutathatja meg. A partíció a merevlemeznek egy logikailag egyetlen fájlrendszerhez rendelt része. Két alaptípusa az elsődleges (*primary*) és a kiterjesztett (*extended*), az `ext2` fájlrendszert bármelyikre telepíthetjük. A partíciók adatai a merevlemez partíciós táblában találhatók, ennek az adatait megvizsgálva tud az operációs rendszert betöltő program dolgozni. A partíciókra való felosztás a telepítésnél már lezajlik, később csak nagy ügyvel-bajjal lehet rajta változtatni adatvesztés vagy újratelepítés nélkül (bár vannak erre szakosodott segédprogramok, például a *Partition Magic*, amik akár „élő” partíciót is képesek átméretezni). A későbbiekben a már említett `/dev` könyvtárban található különleges eszközfájlokra keresztül tudunk rájuk hivatkozni a következő szabályok alapján: a mai PC-kben általában maximum négy *IDE* szabványú adattároló eszköz csatlakoztatható a gépre (*SCSI* szabványú meghajtókból több is, de mivel még az olcsóbb gépekben ritkábban fordulnak elő, ezekkel itt ne foglalkozunk). Ezek közül van egy elsődleges (*primary*) és egy másodlagos (*secondary*) adatsatorna, és mindkettőn lehet egy „mester” és egy „szolga” (master, slave) adathordozó. Az elsődleges mester `hda`, az elsődleges szolga a `hdb`, és ez alapján a másodlagos mester `hdc`, szolga pedig a `hdd` eszköznévvel érhető el. Merevlemezek esetén ez még partíciókra oszlik, amiket számokkal különböztetünk meg, tehát a `hdc2` jelenti a másodlagos mester merevlemez második partícióját.

A *DOS/Windows*-ból ismert nevű, de annál jóval többet tudó és rugalmasabb `fdisk` az alapvető eszköz ezek kialakítására. Kezelése egybetűs parancsokkal történik, emiatt nem túl látványos – de hatékony! Ahogy indításakor is látható, az `m` parancs ad súgót a funkcióiról. Nekünk általában a következők kellenek:

**p** az aktuális partíciós tábla kilistázása;

**n** új partíció létrehozása;

**d** partíció törlése;

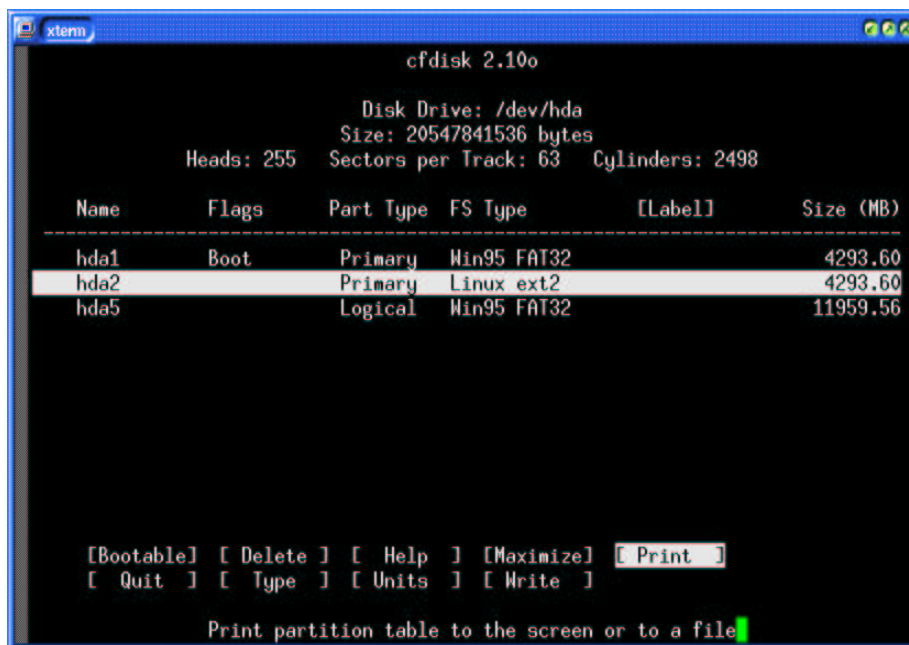
**t** partíciótípus váltása;

**l** partíció-kódok listájának megjelenítése;

**w** a partíciók adatainak mentése, a partíciós tábla kiírása a merevlemezre, mivel a *DOS*-os `fdisk`-kel ellentétben ez addig nem ír ki semmit, amíg erre nem utasítjuk (és még csak újraindítás se kell...!);

q kilépés.

Akinek nem szimpatikus a program, használhatja a `cdisk` programot is, aminek bejelentkező képernyője látható az 1.3. képen.



1.3. ábra. A cfdisk képernyője

A navigálás a képernyő alján látható parancsok között könnyen megoldható a nyíl-billentyűkkel vagy a menüpontok neveinek első betűjét leütve.

Akinek még ez sem elég, van egy `sfdisk` nevű program is, amit főleg héjprogramokban való alkalmazásra terveztek, és érdekes dolgokat tud: például a partíciók információit ki tudja írni egy másik adathordozón lévő fájlba, és ha valami baj van, onnan vissza lehet tölteni azokat.

A partíciók kialakítása után jöhet a fájlrendszer kiépítése, amit köznapi nyelven formázás néven szoktunk emlegetni. Az `mkfs`, vagy inkább az `mke2fs` használandó erre. Ez utóbbi nagyon sok paraméterrel rendelkezik, mégis általában csak egyszerűen az `mke2fs /dev/hdxy` formában használjuk, ahol `x` és `y` értéke a fent leírt módon helyettesítendő be (például: `hda2`).

A létrejött fájlrendszert a `mount` parancs illeszti be a könyvtárába. Kapcsolók nélkül beírva kilisztazza a jelenleg beillesztett rendszereket. Igyekszik kitalálni a becsatolandó rendszer típusát, de ha ez nem sikerül neki, a `-t` kapcsoló segíthet. Megadandó még az eszközfájl neve, amin keresztül a fájlrendszer elérhető, illetve az a könyvtár, ami alatt látni akarjuk majd a tartalmát. Lássunk egy köznapi példát erre, illesszünk be egy CD-ROM-ot az előzőleg már meglévő `/mnt/cdrom` alá:

```
[root:~]>mount /dev/cdrom /mnt/cdrom
mount: block device /dev/cdrom is write-protected, mounting read-only
```

Bízunk abban, hogy a `mount` felismeri a CD-n lévő *iso9660* típusú rendszert, és ez így is volt. A gépen a `/dev/cdrom` igazából egy szimbolikus link arra az esz-

közfájllra, ami a CD-ROM meghajtót azonosítja, ez rugalmasan lehetőséget ad arra, hogy a meghajtót fizikailag áthelyezve csak ezt a linket kell módosítani egy helyen, az összes rá hivatkozó program (és felhasználó) észre sem veszi a változást. Mivel nevéből adódóan egy CD csak olvasható, ezért ilyen módon is csatolta be (*read-only*). Viszont ebből következően bármilyen fájlrendszert becsatolhatunk így (a *-r* kapcsolóval). Szokás a */usr* alatti könyvtárfát egy külön partícióra telepíteni és így becsatolni, ennek tartalma úgylis ritkán változik, viszont így véletlen törlés ellen védve van.

A Linux alatt a becsatolt CD-ROM addig nem vehető ki, amíg az `umount /dev/cdrom` parancsot ki nem adjuk. Ez először kicsit szokatlan lehet (a floppy-t persze kivehetjük, de egyáltalán nem tanácsos). Az `umount` tehát csak azt az eszköznevet (esetleg azt a csatolási pontként megadott könyvtárnevet, például `/mnt/cdrom`) várja paraméterként, ami az adott háttértárhoz tartozik.

A fájlrendszerek épségéről néha illik meggyőződni, erre Linux alatt az `fsck` (`e2fsck`) használható. Ajánlatos előtte az ellenőrizendő rendszert leválasztani az `umount` paranccsal. Használata viszont egyszerű, például az `fsck /dev/hda3` leellenőrzi, és hiba esetén megpróbálja javítani az elsődleges merevlemez harmadik partícióján lévő adatokat.

Érdeklődéssel nézzük meg, hogy lehet akár menet közben ún. *ramdisk*-et, vagyis a memóriában lévő, de hagyományos háttértárhoz hasonlító (csak annál jóval gyorsabb) adattárolót kialakítani. Csak azt kell tudnunk, hogy van több olyan eszköz is *ram0*-tól felfelé számozva, amivel a memória bizonyos szabad részeit lefoglalhatjuk ilyen célra. Most egy ilyen eszközön keresztül egy 4 MB méretű tárat alakítunk ki, amit a `/mnt/ramdisk` könyvtár alatt tudunk elérni:

```
[root:/mnt]>mkdir ramdisk
[root:/mnt]>mkfs -t ext2 /dev/ram0 4096
mke2fs 1.19, 13-Jul-2000 for EXT2 FS 0.5b, 95/08/09
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1024 inodes, 4096 blocks
204 blocks (4.98%) reserved for the super user
First data block=1
1 block group
8192 blocks per group, 8192 fragments per group
1024 inodes per group
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
[root:/mnt]>mount /dev/ram0 ramdisk/
```

...és már van is egy új „meghajtónk”, kezdhethetjük feltölteni adatokkal!

A `mount` még egy izgalmas lehetőséget rejt magában, mégpedig egy fájlrendszernek az ún. visszacsatoló eszközön (*loopback*) keresztüli beillesztését. A UNIX-filozófiából következően egy adathordozó tartalmát byte-ról byte-ra le tudjuk másolni egy fájlba, ez az image-fájl vagy tükörkép-fájl. Leginkább hordozható tárolóknál (floppy, CD-ROM) alkalmazzuk, például a CD tartalmát, ami több ezer fájl is lehet, így egyetlen adatállományban lehet tárolni, és a felírás is könnyebben elvégezhető. Ma már a Linux terjesztéseket is egyre inkább így szoktuk az Internetről letölteni. Egy ilyen fájl tehát tulajdonképpen egy teljes fájlrendszer. Viszont ha CD-re való felírás után derül ki, hogy valami gond volt a tükörkép-fájlban, akkor gyártottunk egy nagyon szép díszet az autó visszapillantójára. Ezt elkerülendő, még a felírás előtt be tudjuk csatolni a fájlt, bele tudunk nézni, mintha már CD-n lenne, sőt, ha nagyon akarjuk, még ki sem kell írni, így is lehet róla telepíteni a rendszert! Tegyük fel, hogy van egy ilyen image-ünk *cdimage.iso* néven az aktuális könyvtárban, ahol van egy *cd-rom* nevű könyvtár is, ekkor a

```
mount -o loop cdimage.iso ./cd-rom
```

parancs után már tallózhatunk is a *cd-rom/* alatt! Ezt a könyvtárat a hálózaton *NFS*, *Samba* vagy más hálózati megosztási módszerrel elérhetővé téve a munkaállomások számára, indulhat a telepítés.

A *mount* parancsról még tudni illik, hogy tudását befűzés esetén a */etc/fstab* állományból veszi. Ebben szerepelnek a különböző becsatolandó rendszerek adatai és olyan paraméterek, amik szabályozzák a műveletet. Tanulmányozzuk bátran!

Hálózatos környezetben nemcsak a helyi gépen, hanem a hálózat szerverein is lehetnek olyan megosztott állományok, amikhez szeretnénk hozzáférni. A *Samba* programcsomag segítségével a Linux része tud lenni egy *Microsoft Network* hálózatnak, ahol az azon belül megosztott erőforrásokat, úgymint állományok vagy nyomtatók (összefoglaló néven *share*-ek – magyarul megosztások), el tudja érni. Állományrendszerek esetén először az *smbclient* paranccsal ki kell deríteni, milyen megosztások léteznek, majd ezeket az *smbmount* illetve *smbmnt* parancsok használatával tudjuk becsatolni. *Novell Netware* fájlkiszolgálók esetén az *slist* adja a szerverek listáját, majd az *ncpmount* végzi a beillesztést. Természetesen mindegyiknek megvan a leválasztást végző társparancsa is.

Sokszor ütközünk abba a problémába (még nagy merevlemez esetén is), hogy elfogy a tárhelyünk. Aki nem akarja ezt, használja gyakrabban a *df* parancsot. Arra szolgál, hogy megmutassa, az egyes partíciókon még mennyi szabad hely van, és mennyi foglalt. Célszerű a *-h* kapcsolóval együtt használni, mivel így kilo- illetve megabyte-ban írja ki az adatokat. Arra pedig, hogy egy könyvtár állományai mennyi helyet foglalnak, a *du* ad választ. Abban a könyvtárban kezd el az összegzést, ahol kiadtuk, és rekurzívan dolgozik, minden alkönyvtárról külön kiírja a benne lévő állományok méretét, majd a végén egy összegzést is. Szintén a *-h* paraméter segít olvashatóbbá tenni az eredményeit. Ha csak a teljes végösszegre vagyunk kíváncsiak, használjuk a *-s* opcióval együtt. A *-S* segítségével pedig minden alkönyvtár helyfoglalása külön lászik.

A Linux működés közben sok adatot tárol a memóriában, ezért fontos, hogy mindig szabályosan állítsuk le (ez egyébként a *shutdown* illetve *halt*, az újraindítás pedig a *reboot* parancs segítségével történik), mert különben a fájlrendszer is sérülhet. Ennek oka többek között az, hogy nem ír azonnal minden adatot a merevlemezre, hanem ideiglenes tárolókat, puffereket használ a memóriában. Ezek tartalmát azonban mi magunk is kiírathatjuk a lemezre a *sync* parancs használatával.

### 1.13.2. Tömörítés, archiválás

Az adatállományok mérete és a rendelkezésre álló tárhely sajnos gyakran nincs összhangban. A tömörítőprogramok használatával ez a gond csökkenthető. Fontos adataink biztonságos tárolása pedig megköveteli a biztonsági mentések készítését, az archiválást. A Linuxnak ezekre a feladatokra is fejlett eszközei vannak.

Valószínűleg a legrégebben létező tömörítőprogramok egyike a *compress/uncompress* páros, ami minden UNIX-on megtalálható. Az általa készített állomány a *.Z* végződésről ismerhető fel a legkönnyebben. Manapság már kevésbé használják, inkább utódját, a *gzip* csomagot alkalmazzák. Használata egyszerű, legtöbbször csak a tömörítés erősségi fokát szoktuk megadni, ez egy 1-től 9-ig (a legnagyobb fokú tömörítésig) terjedő skálán mozoghat, illetve a tömörítendő állomány nevét. Ellentétben viszont a *DOS/Windows* alatt jól ismert *zip* paranccsal, csak egy fájlt tud egyszerre kezelni. Ezért szinte elválaszthatatlan párja

a tar, ami annak idején szalagos egységekre való archiváló programként született (*Tape ARchiver*). Nagyon jól kiegészítik egymást, mivel a tar több fájlt, akár egész könyvtár-struktúrákat tud összefűzni egyetlen fájlba, és ezt már át lehet adni a gzip parancsnak. Olyannyira egybeépültek a használat során, hogy a tar külön kapcsolóval rendelkezik, ami a végén meghívja a gzip parancsot a tömörítési lépés végrehajtására. Lássuk őket működés közben! Talán még nem dobtuk a sutba a fentebb használt jogok/ könyvtárunkat és tartalmát. Ebben elvileg több fájl is van, lépjünk tehát be ide, és írjuk be a következőket:

```
[root:~/jogok]>tar cvf archiv.tar *
host1.conf
host.conf
javitas
julius
[root:~/jogok]>gzip -9 archiv.tar
[root:~/jogok]>ls a*
archiv.tar.gz
```

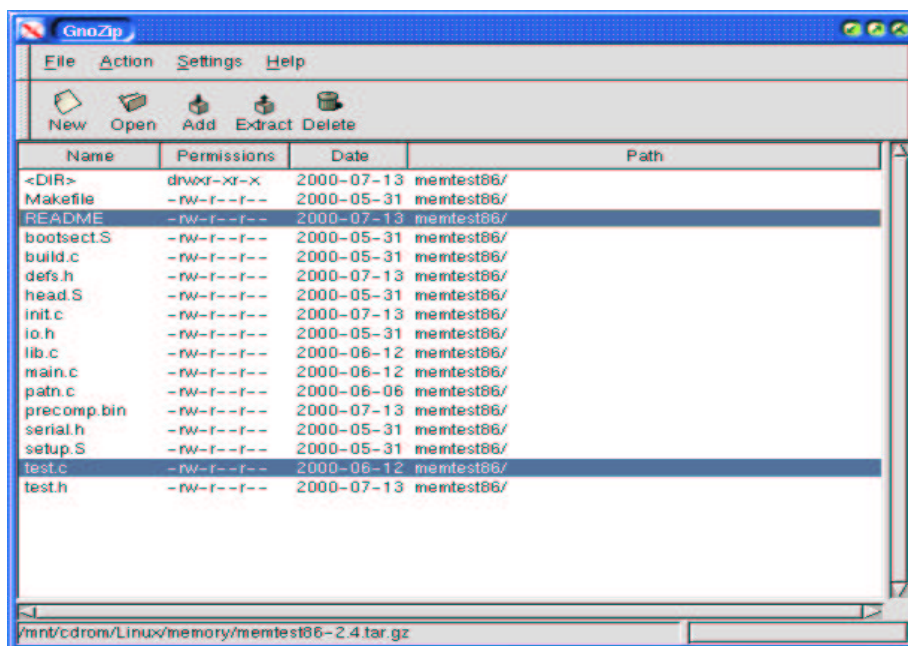
A tar kapcsolói sorban (figyeljük meg, hogy nem kell a - jel): a c jelzi, hogy készíteni akarjuk az állományt, nem kicsomagolni, a v kicsit bőbeszédűbb üzemmódra kapcsol (ezért listázza ki közben a fájlneveket), az f fájlnev pedig maga az elkészítendő állomány neve. A kiterjesztés megállapodás szerint .tar. Mivel mindent be akarunk csomagolni, ezért áll a sor végén a \* (és nem \*. \*, hiszen itt a fájlnévben több pont is lehet). A gzip parancsot aztán a legerősebb fokozatban szabadítjuk rá az elkészült fájlra, és megkapjuk a .gz kiterjesztésű, immár tömörített állományt. Mindezt megtehetjük sokkal rövidebben is, ha a tar kapcsolóit kiegészítjük még egy z opcióval, megspórolhatjuk a gzip parancs kiadását.

A kicsomagolás is többféleképpen történhet: vagy először a gunzip archiv.tar.gz, majd a tar xvf archiv.tar parancsok kiadásával, vagy egyszerűbben csak a tar zxvf archiv.tar.gz parancs beírásával. A tar természetesen megőrzi a könyvtárak elrendezését is, erre mindig figyeljünk kicsomagolás előtt. Ha bele akarunk kukucskálni egy archívumba, többféle módon járhatunk el. Csak a becsomagolt fájlok listájára vagyunk kíváncsiak? Használjuk a tar ztvf archiv.tar.gz parancsot (listázás)! Van egy nagy méretű szövegünk, és nem akarjuk kicsomagolni, csak elolvasni? A zcat vagy még inkább a zless parancsoknak vesszük hasznát.

A tar lehetőséget ad arra is, hogy ha a kicsomagolandó fájl nem azon a helyen van, ahová szánjuk, és meg akarjuk spórolni a másolást, akkor egy -C célkönyvtár paraméter közbeiktatásával először beléptessük a célkönyvtárba, és ott kezdje el a kibontást.

Az Interneten a Linuxos források többsége ilyen formátumban található meg, a .tgz kiterjesztés is ilyen archívumokat takar. Létezik jó ideje egy, a gzip programnál jobb hatásfokú tömörítő is, a bzip2. Kezelése nagymértékben hasonlít elődjére. Az általa készített állományok ismertetőjegye a .bz2 kiterjesztés. Az ismert és elterjedt ZIP-formátum is képviselteti magát a zip/unzip programokon keresztül. Ezek közül főleg a kicsomagolót használjuk, főbb paraméterei:

- l kilistázza az archívum tartalmát;
- t teszteli az archívumot;
- d könyvtár a megadott könyvtárba csomagol ki;
- v bőbeszédű üzemmód bekapcsolása.



1.4. ábra. Egy grafikus archívumkezelő, a GnoZip

A szintén nagyon elterjedt *ARJ*, *RAR*, újabban *ACE*, de a régebbi harcosok közül a *ZOO*, *LZH* formátumokat is tudjuk kezelni az *unarj*, *unrar* stb. programokkal. Ezek használata szintén nem okozhat gondot.

A Linuxos terjesztések, disztribúciók programcsomagjai általában nem így érhetőek el (kivétel azért van a jó öreg *Slackware* személyében), hanem más archiválókat használnak. A *Red Hat* csomagformátuma, az *rpm* a *cpio* programon alapszik. Ennek alapvetően három üzemmódja van: a *copy-out* becsomagol, a *copy-in* kicsomagol az archívumból, a *copy-pass* pedig egyik helyről a másikra visz át állományokat. Nézzünk példát mindháromra. Maradva a *jogok/* könyvtáron belül, adjuk ki az `ls | cpio -oF archiv.cpio` parancsot. A *cpio -o* paramétere kapcsolja be az archiválási üzemmódot, a *-F* után kell megadni a készítendő fájl nevét. A becsomagolandó fájlok listáját azonban trükkös módon kell megmutatni neki, most erre az `ls` látszott célszerűnek. Az `ls` kimenetét átadva egy szűrőn keresztül, már tudja a *cpio*, mivel kell dolgoznia. A kicsomagolás a `cpio -i < archiv.cpio` parancsra történne, de ez most nem fog rendben lefutni, mivel ugyanazok a fájlok már szerepelnek a könyvtáron belül. Végül pedig a `ls | cpio -pd /root/jogok.1` hatására az összes fájl átkerül a *jogok.1/* könyvtárba, természetesen minden tulajdonságát (tulajdon- és hozzáférési viszonyok) megőrizve; ez a *tar* esetében is így van.

A *Debian .deb* csomagjai az *ar* parancs felhasználásával készülnek. Általánosan *ar* kapcsoló archívum lista formában használatos, itt a főbb kapcsolók:

**r** becsomagolás;

**x** kicsomagolás;

**t** listázás.

Gyakorlásul próbáljuk ki azt, melyik program a legjobb hatásfokú!

## 1.14. Szövegkezelő parancsok

A szöveges adatok fontos szerepéről a szövegszerkesztők kapcsán már szóltunk. Kezelésük, manipulálásuk mindennapos feladat. Az erre szolgáló fontosabb eszközök áttekintése következze most.

A szövegfájlok létrehozási módjait, tartalmuk kilistázását már ismerjük. Ide kívánczik még a teljesség kedvéért a `strings` parancs, ami bármilyen fájlból ki tudja írni a nyomtatható karaktereket; ki lehet próbálni bináris fájlkon is. Néha szükségünk van annak megállapítására, hogy hány karakterből, szóból vagy sorból állnak szövegeink. Erre a `wc` parancs adhat választ, bár nevét látva először biztos nem erre asszociálunk. Próbáljuk ki egy állományon, legyen ez mondjuk a `/etc/passwd`:

```
[root:~]>wc /etc/passwd
 34      42    1260 /etc/passwd
```

A fenti fájl jelenleg 34 sort, 42 szót és 1260 karaktert tartalmaz. Ha ezekre egyenként vagyunk kíváncsiak, alkalmazzuk sorban a `-l`, `-w` illetve a `-c` kapcsolókat. Számoltassuk meg, hány fájl van az aktuális könyvtárunkban! Ehhez nyilván felhasználjuk az `ls` parancsot is, majd az általa adott szöveges kimenetben lévő sorokat megszámlolva kapjuk az eredményt:

```
[root:~]>ls | wc -l
 39
```

A szövegek átalakítását végző parancsok közül a `tr` szolgál bizonyos átalakításokra, nevezetesen a bemenetén érkező karaktersorozatot át tudja alakítani, transzformálni valamilyen szabály alapján. Gyakran alkalmazzuk kis- és nagybetűk cseréjére, mint az alábbi példában is:

```
[root:~]>echo VegyEs | tr a-z A-Z
VEGYES
```

Láthatóan az összes kisbetűt a-tól z-ig átalakította a nagybetűs megfelelőjére. Ami már eleve nagybetűs volt, azon persze ezt nem vesszük észre. Nemcsak átalakítást, hanem törlést is kérhetünk tőle:

```
[root:~]>echo egyesek | tr -d e
gysk
```

Ehelyütt csak említést teszünk a `sed` programról, ami ennél jóval többet tud, részletes ismertetése egy másik füzet célja.

Szövegeink sokszor valamilyen táblázathoz hasonló felépítésben fordulnak elő. Láttunk ilyet például a `df` parancs kimeneténél, a `w` parancsnál, de rengeteg egyéb helyen is. Nemcsak ilyen, hanem igazándiból tetszőleges szövegből való kivágásra szolgál a `cut`. Megadható, hogy karaktereket vagy valamilyen jellel elválasztott mezőket szedjen ki nekünk. Példaként használjuk fel a `/etc/passwd` fájlunkat. Ez úgy épül fel, hogy egy-egy sorában egymástól kettősponttal elválasztva szerepelnek az egyes felhasználók nevei, jelszavai majd egyéb fontos adatai. Oldjuk meg most azt, hogy csak a felhasználók neveit írjuk ki ebből:

```
[root:~]>cut -d: -f1 /etc/passwd
root
bin
daemon
adm
lp
...
```

A `-d` után következő karakter szerepel a mezők elválasztó jeleként, a `-f` után pedig az első mező sorszámát adtuk meg, így azt írja csak ki. Ha fix szélességű oszlopaink vannak, akkor használhatjuk a `-c` kapcsolót: utána azt kell beírni, hogy melyik karakterpozíciótól meddig akarjuk kivágni az adott sorbeli szöveget. Szűrjük meg ennek segítségével a `df -h` parancs kimenetét úgy, hogy csak az első oszlop maradjon, valamint a szabad helyet feltüntető oszloptól jobbra eső rész:

```
[root:~]>df -h | cut -c1-11,34-
Filesystem Avail Use% Mounted on
/dev/hda2    1.9G  49% /
/dev/hda1    2.1G  47% /mnt/win_c
/dev/hda5    5.9G  47% /mnt/win_d
```

Itt először az első és a 11. karakter közötti, majd pedig a 34. karakterpozíciótól a sor végéig terjedő részt vágtuk.

Szintén szeletelési műveletet hajt végre a `split`, de ez úgy működik, hogy egy nagyobb fájlt több, általunk szabályozható méretű, kisebb részre oszt szét. Csináljunk egy munka/ könyvtárat, másoljuk ide a `/etc/passwd` fájlt, majd alkalmazzuk a `split -l 5 passwd proba` parancsot! Ennek hatására olyan fájlok jönnek létre, amelyek maximum 5 sorosak, és nevük eleje `proba`, majd a program automatikusan ehhez hozzáilleszti az `aa`, `ab`, `ac` stb. karaktereket is. Tartsuk meg ezek közül az első kettőt, hogy a következő parancsot is ki tudjuk próbálni, illetve, hogy be tudjunk mutatni egy olyan lehetőséget, amire eddig még nem került sor.

A helyettesítő (joker) karakterek közül már használtuk a csillagot, ami többféle és több számú karaktert helyettesít. A *DOS*-ban megszokott módon a kérdőjel is használható. Van azonban ezeken kívül más lehetőség is. Nézzük pontosan, mi is a feladat. Az összes `probaa` kezdetű, de nem a vagy `b` végű fájlt kell kijelölnünk törlésre. Megoldásként kínálkozik a szögletes zárójelek alkalmazása: azon karakterek bármelyike előfordulhat az adott karakterpozíción, amit ebben felsorolunk. Most `c`-től egészen `z`-ig engedünk meg a fájlok végén karaktereket, tehát a kiadandó parancs így néz ki:

```
[root:~/munka]>rm -f probaa[c-z]
```

A terep tiszta a `paste` parancs alkalmazásához. Működése abban áll, hogy két vagy több fájl tartalmát egybeillessze oly módon, hogy a megfelelő sorszámú soraikat egymás mellé rakja. Tehát a `paste probaaa probaab` utasítás eredménye olyan fájl lesz, amiben az első sor első fele a `probaaa` első sora, második fele a `probaab` első sora, és így tovább – de inkább nézzük meg, elmondva bonyolultabban hangzik.

Miután darabolni már nagyon jól tudunk, fogjunk rendrakásba, egészen pontosan rendezésbe a `sort` segítségével. Feladata ABC sorrendbe rakni a fájlok tartalmát. A `probaaa` egyszerű névsorba rendezéséhez a felhasználói nevek alapján nem is kell más, csak egy `sort probaaa`, fordított sorrendbeli rendezéshez pedig egy `-r` paraméter. Ha a karakter felfogható számként is, mint jelen esetben a harmadik mezőbeli *User ID*, akkor a `-n` segít. Nem muszáj az első karakter vagy éppen mező szerint rendezni, erre példaként lássuk azt az esetet, amikor fájlkat a harmadik mezője alapján akarjuk csökkenő sorrendbe rakni. Az eredeti fájlt írassuk ki a `cat` paranccsal, hogy össze tudjuk hasonlítani, majd írjuk be:



```
[root:~/munka]>sort -t: +2 -n -r probaaa
```

Elemelve: a mezőelválasztó jel a kettőspont, ez szerepel a `-t` után; a mezők sor-számozása itt 0-tól indul, ezért kell a `+2`; a `-n` numerikus rendezést ír elő, a `-r` pedig megfordítja a sorrendet.

A `sort` gyakran közösen használatos a `uniq` paranccsal: ez a bemenetére érkező szövegből kiszűri az ismétlődőket, és csak egyszer jelenít meg minden sort.

Szövegekben való keresésre használatos a `grep`. Mielőtt használatáról beszélénk, szót kell ejteni az ún. reguláris kifejezésekről, amivel nagyon bonyolult szövegmintákra is lehet hivatkozni. Több szabálya van, ezek közül csak a fontosabbakat ismertetjük:

- minden karakter, ami nem a `[ ] $ ^ . * \ +` karakterek valamelyike, önmagát jelenti egy reguláris kifejezésben, ezt úgy is mondjuk, hogy „önmagára illeszkedik”;
- a fenti karaktereket úgy lehet megadni saját, literálisan vett értelmükben, hogy `\` jelet rakunk eléjük, tehát például a `\$` jelenti a dollárjelet;
- a `.` (pont) karakter bármely karakterre illeszkedik; emiatt a `pist.` reguláris kifejezés jelentheti a `pista`, `pisti`, `pistuka` szöveget is;
- a `*` karakter az öt megelőző reguláris kifejezés 0 vagy többszöri előfordulását jelenti. Emiatt a `pist.*` lehet `pista`, de lehet `pistuka` is, sőt a `pist` is, mivel ekkor pontosan 0-szor fordul elő az utolsó helyen vett karakter;
- a szögletes zárójeleken belül felsorolt karakterek bármelyike előfordulhat az adott pozíción, kivéve, ha a legelső karakter `^`;
- a reguláris kifejezés után írt `$` jel a mintát a sor végére, míg a kifejezés elé írt `^` jel a sor elejére illeszti;
- több egymás után írt reguláris kifejezés is reguláris kifejezés lesz.

Ne ijedjen meg senki, néhány példa után jóval érthetőbb lesz ez a dolog. Lássunk is hozzá! Készítsünk egy példafájlt `proba` néven, és töltsük fel a következő tartalommal (a hatodik sor üres):

```
joska
jozsika
jolika
jocika
pist

pista
pistu
pistuka
pistike
juliska
```

majd adjuk ki a következő parancsot:

```
[root:~/munka]>grep pist. proba
pista
pistu
pistuka
pistike
```

Ha a `pist.*` reguláris kifejezést használjuk, kiíródik a `pist` is a fent említettek értelmében. Írassuk ki most a `j` betűvel kezdődő neveket:

```
[root:~/munka]>grep ^j proba
joska
jozsika
jolika
jocika
juliska
```

Most azokat, amik nem a betűre végződnek (itt a szögletes zárójel után következő ^ az őt követő a betű tagadását jelenti):

```
[root:~/munka]>grep [^a]$ proba
pist
pistu
pistike
```

Próbáljuk meg azokat a neveket kiíratni, amelyek kezdete jo, majd nem c betű következik, vége pedig ika:

```
[root:~/munka]>grep jo[^c]ika proba
jolika
```

Adjuk meg az üres sor sorszámát! Ehhez a -n paraméter szükséges, ami minden, a mintának megfelelő sor elé kiírja annak sorszámát is. Valamint azt kell tudni, hogy a ^\$ kifejezés jelenti az üres sort:

```
[root:~/munka]>grep -n ^$ proba
6:
```

Végül írjuk át a szövegben jozsika és jolika nevét nagybetűsre, hogy kipróbálhassuk azt, amikor nem tesz különbséget kis- és nagybetű között:

```
[root:~/munka]>grep -i jo.* proba
joska
Jozsika
Jolika
jocika
```

Mind a reguláris kifejezések, mind a grep rejteget még jócskán kipróbálni valót, járjunk utána!

## 1.15. Folyamatkezelő parancsok

### 1.15.1. Folyamatkezelés

A többfeladatos (idegen szóval multitaszkos) operációs rendszerekben egy felhasználó több programot is futtathat egyszerre. Bár egy processzor egy időpontban csak egy feladatot tud ellátni, a megoldás az, hogy a végrehajtás nagyon gyorsan egyik programról a másikra kerül át, és ezáltal úgy tűnik, hogy minden párhuzamosan zajlik. A felhasználó által elindított programokat az összes hozzájuk tartozó adattal közös néven processznek vagy folyamatnak hívjuk. Mindegyik rendelkezik egy egyedi azonosító kóddal, ennek rövid neve a *PID* (*Process Identifier*). Később ezzel a kóddal tudunk a folyamattal több dolgot csinálni, például leállítani, de egyéb jelzéseket is küldhetünk számára. Van azonban egy olyan folyamat, név szerint az *init*, amit nem szokás így leállítani: ez az 1-es *PID* tulajdonosa, ő indul el elsőnek a rendszer betöltésekor, és emiatt őt nevezhetjük „Minden Processzek Atyjának”. Egy processz „szülhet” újabbakat is, tipikus példa erre a héjprogramunk, amiből további programokat indítunk. Ha viszont egy ilyen, ún. gyermek-folyamat még nem futott le, de szülője megszakad, akkor bevett kifejezéssel zombivá válik. Némely folyamatok nem kötődnek szorosan egy

terminálhoz, ezek valamilyen rendszerszintű feladatot látnak el. Ide tartoznak például a hálózati kiszolgálók (többek között FTP- vagy WWW-szerverprogramok), ezeket *démon*oknak hívjuk. Nemhiába írta egy szakíró, hogy ezek a dolgok „változtatják vidám pokollá a UNIX belsejét”.

A futó folyamatokról való tájékozódás alapvető eszköze a `ps`, amivel kilistázhatjuk ezeket. Nagyon sok kapcsolója van, párat próbáljunk ki! Először lássuk, jelenleg milyen folyamatok futnak a terminálunkon. Ehhez nem kell külön paraméter, ha viszont több helyről is be vagyunk jelentkezve, akkor kérdezzük le valamelyik másikat! A `w` megmutatja a terminál kódját, majd, ha történetesen a `ttty1` szerepel a listán, akkor:

```
[root:~]>ps -t ttty1
  PID TTY          TIME CMD
  655 ttty1      00:00:00 login
  663 ttty1      00:00:00 bash
  719 ttty1      00:00:00 xinit
  727 ttty1      00:00:04 icewm
```

A példában látható, hogy az első oszlopban találhatók a folyamatazonosítók, és a végén a processzek nevei. Egy program processzazonosítója könnyebben is lekérdezhető a `pidof` programnév utasítással. Kíváncsiak vagyunk a más felhasználók által futtatottak adataira is? Legyen mondjuk ismét pistike a kiszemelt áldozat, leleplezhetjük minden mesterkedését a `ps -u pistike` kiadásával. A rendszerben futó összes folyamatot a `ps aux` listázza ki, ha kibővítjük még a `w` kapcsolóval is, akkor az esetleges hosszabb parancssorok is láthatóvá válnak. Ilyenkor több oszlopnyi adatsor jelenik meg, láthatjuk többek között a folyamat által igénybe vett CPU-időt, memóriai igényt, mikor indult, és milyen állapotban van éppen: fut, bevitelre vár, leállt vagy netán zombi. Látványos eredményt ad a `--forest` vagy egyszerűen `f` kapcsoló:

```
[root:~]>ps f -u pistike
  PID TTY      STAT   TIME COMMAND
 1081 ttty2      S       0:00 -bash
 1140 ttty2      S       0:00 /usr/bin/mc -P
 1141 ?         S       0:00 \_ cons.saver /dev/tty2
 1142 pts/3     S       0:00 \_ bash -rcfile .bashrc
```

Amint látható, *ASCII*-karakterek segítségével igyekszik ábrázolni a folyamatok függőségi viszonyait. A `pstree` is hasonló célt szolgál, paraméter nélkül kiadva az `init` processztől kiindulva az összes futó folyamatot kijelzi.

Arra, hogy folyamatosan figyelhessük a processzek állapotát, a `top` használható. Bizonyos időközönként frissíti az adatait, és ezt listázza ki, így jobban képen lehetünk a gépen folyó viszonyokról. Menet közben egybetűs parancsokkal vezérelhető, hogy mit és hogyan írjon ki. Ezek közül csak kettőt érdemes nagyon megjegyezni: a `h` mutatja meg, milyen parancsok használhatók, a `q` pedig kiléptet. Futás közbeni állapotát ábrázolja az 1.5. ábra:

A processzek rendelkeznek egy olyan azonosítóval, ami megmondja az operációs rendszer feladat-ütemezője számára, hogy milyen fontosságú a többihez viszonyítva, vagyis milyen prioritással rendelkezik. Ez az azonosító a *nice-level* nevet viseli, és a `top` kimenetében a `NI` oszlop alatt láthatók az értékei az egyes folyamatokra. Egy programot -20-tól 19-ig terjedő *nice*-szinttel tudunk elindítani. Minél nagyobb a negatív szám, annál nagyobb a prioritás, tehát annál többet és többször foglalkozik vele a rendszer. Egy programot a `nice --20` programnév utasítással tudunk úgy elindítani, hogy most ezzel legyen a legjobban elfoglalva a gépünk. Futó folyamatnak pedig a `renice` segítségével lehet megváltoztatni ezt a jellemzőjét (de erre csak a rendszergazda, másnéven a root képes). Például a `renice -10 1124` az 1124-es processz-azonosítójú programhoz a -10-es *nice*-szintet rendeli.

```

2:09pm up 11 min, 4 users, load average: 0.02, 0.07, 0.04
53 processes: 52 sleeping, 1 running, 0 zombie, 0 stopped
CPU states: 1.7% user, 0.7% system, 0.0% nice, 97.4% idle
Mem: 65120K av, 60988K used, 4132K free, 0K shrd, 3928K buff
Swap: 0K av, 0K used, 0K free, 27560K cached

```

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	%CPU	%MEM	TIME	COMMAND
1007	root	17	0	888	888	688	R	1.1	1.3	0:00	top
726	root	14	0	9832	9832	2156	S	0.7	15.0	0:11	X
729	root	1	0	2452	2452	1468	S	0.1	3.7	0:02	iceum
733	root	1	0	2284	2284	1684	S	0.1	3.5	0:00	xterm
938	root	10	0	8432	8432	6992	S	0.1	12.9	0:07	ksnapshot
1	root	0	0	468	468	404	S	0.0	0.7	0:04	init
2	root	0	0	0	0	0	SW	0.0	0.0	0:00	kflushd
3	root	0	0	0	0	0	SW	0.0	0.0	0:00	kupdate
4	root	0	0	0	0	0	SW	0.0	0.0	0:00	kswapd
5	root	-20	-20	0	0	0	SW<	0.0	0.0	0:00	mdrecoveryd
380	root	0	0	580	580	484	S	0.0	0.8	0:00	automount
403	root	0	0	616	616	508	S	0.0	0.9	0:00	automount
451	root	0	0	568	568	444	S	0.0	0.8	0:00	syslogd
461	root	0	0	780	780	388	S	0.0	1.1	0:00	klogd
469	root	0	0	1120	1120	876	S	0.0	1.7	0:01	sshd
483	root	0	0	912	912	712	S	0.0	1.4	0:00	xinetd
519	root	0	0	440	440	380	S	0.0	0.6	0:00	gpm

1.5. ábra. A top működés közben

A folyamatok jobb esetben rendben lefutnak, de gyakran van szükség arra, hogy működésüket felülbíráljuk. Ez általában a leállítás, főleg ha lefagyott, rendetlenkedik. A gyilkosan hangzó nevű `kill` parancs a fegyver a kezünkben ilyen esetekre. Kötelező paramétere egy folyamatazonosító, de gyakran kiegészítjük ezt egy olyan paraméterrel, ami szabályozza, milyen brutálisan akarunk elbánni a processzel. A legdurvább módszer a `-9` kapcsoló, ami a `SIGKILL` nevet viseli (az összes kiadható jelzés, szignál leírása megtalálható a `man 7 signal` parancsra megjelenő kézikönyvben). Vicceljük meg szegény pistikét! Mialatt dolgozik, lőjük ki a parancsértelmezőjét, ez persze azonnal kilépteti őt a rendszerből:

```

[root:~]>ps -u pistike
  PID TTY          TIME CMD
 1196 tty3      00:00:00 bash
[root:~]>kill -s KILL 1196

```

A `kill -9 1196` parancs is ugyanerre az eredményre vezet (mindez persze csak rendszergazdaként működik). Ezzel a legerősebb fokozattal, ha lehet, csak végső esetben éljünk lefagyott processz esetén, először mindig próbálkozzunk gyengédebb módszerekkel is.

Feltéve, hogy egyszerre több azonos nevű programnak szeretnénk jelzést küldeni, használható a `killall` parancs. Azt az esetet nézzük meg, amikor több példány is fut a gépen az `xbill` nevű játékprogramból, és mindet le szeretnénk állítani. Először megnézzhetjük a `pidof` felhasználásával, melyek is ezek a processzek, majd takarodót fújunk nekik:

```

[root:~]>pidof xbill
1681 1566
[root:~]>killall xbill

```

Amikor elindítunk egy programot, akkor az az előtérben fut, a billentyűzeten adunk át neki adatokat, ő pedig a képernyőn üzen vissza. A parancsértelmezők lehetőséget adnak azonban arra is, hogy az elindított programjainkat a háttérben futtassuk, ami azt jelenti, hogy ha nem feltétlenül szükséges a futásához az állandó beavatkozás, akkor amíg dolgozik, addig mi mással is tudunk foglalkozni. Az ilyen háttérben futtatott folyamatok elnevezése *job* vagy feladat. Próbáljuk ki azt az esetet, amikor a *find* segítségével az összes *t* betűvel kezdődő nevű fájlt meg akarjuk keresni! Ez időigényes feladat, de az idő pénz, mi addig mást is szeretnénk csinálni. Mi sem egyszerűbb: a *find / -name t\** & parancs kiadásával a program a háttérben indul el, erre az & karakter utasítja, és azonnal visszakapjuk a készenléti jelet. Illetve előtte két kódot látunk: az első, szögletes zárójelben lévő szám a feladat azonosítója, mellette pedig a program processz-azonosítója szerepel, hogy a későbbiekben tudjunk rájuk hivatkozni. Menetközben, mivel máshogy nem rendelkezünk, a *find* az eredményeket kiírja a képernyőre. Ez néha zavaró lehet, ilyenkor használjunk fájlba való átírányítást. A *jobs* parancs kiadásával láthatjuk, éppen milyen állapotban van a feladatunk:

```
[root:~]>jobs
[1]+  Running                  find / -name t* &
```

vagyis még javában fut. Amikor lefutott, arról a

```
[1]+  Exit 1                    find / -name t*
```

üzenet tudósít. Indítsuk most el ugyanezt a *job*ot úgy, hogy időlegesen megszakítsuk a futását. Ez úgy érhető el, hogy az indítás után pár másodperccel lenyomjuk a *CTRL-Z* billentyűkombinációt (ilyenkor persze nem írjuk ki a parancssor végére az & jelet!). A következő felíratot kell látnunk:

```
[1]+  Stopped                  find / -name t*
```

Most dolgozhatunk valami máson, de a feladat addig nem folytatódik az előbbi esettel ellentétben, amíg újra az előtérbe nem hozzuk, mégpedig az *fg* parancs segítségével. Ennek paramétere egy százalékjel után a *job* azonosítója, ami most 1:

```
[root:~]>fg %1
find / -name t*
```

Újra kijelezte a parancssort, és ismét látjuk, hogy dolgozik. A következő lépésben újra állítsuk meg a *CTRL-Z* segítségével, de most majd azt szeretnénk, hogy az első esetben látottakhoz hasonlóan végezze a háttérben a feladatát! Ehhez a *bg* parancs szükségeltetik, ami ismét csak a feladatazonosítóval – mint paraméterrel – kiadva *job*unkat a háttérbe küldi:

```
[root:~]>bg %1
[1]+  find / -name t* &
```

A kijelzett parancssor ugyanaz, mint amit legelőször kiadtunk. Ne várjuk végig ismét, amíg végez, hanem hozzuk az előtérbe, majd ha a *CTRL-C* billentyűkombinációt használjuk, ettől azonnal meg fog állni. Ez egyébként a legtöbb programnál hatásos fegyver, ha meg akarjuk szakítani.

Abban az esetben, ha egy hosszadalmas feladatot szeretnénk még elindítani, de nincs több időnk a gép előtt maradni, jön jól a *nohup* parancs. Célja az, hogy az utána írt parancsot akkor is folytassa, ha kiléptünk. Próbáljuk ki tehát, hogy belépünk a rendszerbe, kiadjuk a

```
nohup find / -name t* > ~/nohup.log &
```

parancsot, majd egyből ki is lépünk! A keresés eredménye a *home* könyvtárunkban keletkező `nohup.log` fájlban vár majd bennünket, de elmenőben még hallhatjuk, hogy nem szakad meg a `find` tevékenysége.

A Linuxban lehetőség van arra, hogy ún. futási szinteket, *runleveleket* használjunk. Ezek nem mások, mint adott processzek halmazai, amiket elindítva meghatározott feladatokra lehet ráállítani a gépet. Az 1-es futási szint az egyfelhasználós üzemmód, ekkor csak a rendszergazda tud bejelentkezni a konzolon, és főleg beállítási, hibajavítási feladatokat tud elvégezni. A 2-es és a 3-as a szöveges módú bejelentkezésre alkalmas, teljes hálózatos üzemmód. A 4-est nem nagyon használják, az 5-ös a grafikus felület automatikus indítására szolgál (ekkor már a bejelentkezés is ilyen módon történik), a 0 a *HALT*, vagyis megállás, a 6 pedig a *REBOOT*, vagyis újraindítás szintje. Indulásakor az `init` program megvizsgálja a `/etc/inittab` fájl tartalmát, ebből olvassa ki az alapértelmezett *runlevel* értékét, és az ezen a szinten meghatározott programokat indítja el. Az aktuális futási szint a *runlevel* paranccsal kérdezhető le; ha indulás után adjuk ki, egy `N` betű és a megfelelő számérték látszik. Menet közben is lehet váltani az `init` processznek átadott kóddal, tehát az `init 5` parancs az 5-ös szintre viszi a gépet: azokat a programokat leállítja, amik ott nem kellenek, és el is indít néhányat. Ezek után a *runlevel* mutatni fogja az előző és a mostani szint értékét is.

### 1.15.2. Időzített parancsvégrehajtás

Néha az a célravezető, hogy bizonyos feladatokat egy adott időpontban végezzünk el. Teszem azt egy nagyobb méretű fájl letöltése az Internetről az éjszakai órákban várhatóan gyorsabb, mint napközben, de ha nem vagyunk éjjeliőrök, máris gondban vagyunk. Aki viszont ismeri az `at` parancs lehetőségeit, máris fellelégezhet. Ezt felhasználva egy, a rendszeren futó démon-programot, az `atd`-t tudjuk arra utasítani, hogy a megbízásunkból adott időben indítson el egy feladatot. Ha ez a feladat összetettebb, vagy a parancssora hosszú, bonyolult, célszerű ezt beírni egy szövegfájlba, majd ezt átadni neki paraméterként, hogy onnan olvassa ki a teendőket. Próbáljuk ki tehát úgy, hogy beírjuk egy `todo` nevű fájlba a már jól bevált parancssorunkat, és szeretnénk, ha éjfél előtt egy perccel látna neki:

```
[root:~]>at 23:59 -f todo
warning: commands will be executed using /bin/sh
job 4 at 2001-07-11 23:59
```

Az üzenet szerint majd egy `sh` parancsértelmező fogja végrehajtani, 4-es számú feladatként a kért időben. Annak a kilistázása, hogy milyen feladatokat kell még elvégezni, hogyan áll a várakozási sor, az `atq` feladata:

```
[root:~]>atq
4          2001-07-11 23:59 a
```

Egy feladat van most a sorban, a végén álló `a` betű jelzi, hogy még várakozik. Ha itt egy egyenlőségjelet látunk, akkor az a feladat éppen végrehajtás alatt áll. A várakozási sorból való kivételre az `atrm` szolgál, csak meg kell neki adni a *job* sorszámát (akár többet is lehet szóközzel elválasztva).

Az `at` egyébiránt nagyon sokféle formában elfogadja az időpont megadását, néhány a lehetséges variációk közül:

**noon, midnight** déli, illetve éjjeli 12 órakor;

**2pm tomorrow** holnap délután 2 órakor;

**1am Sun** vasárnap hajnali egykor;

**13:13 01.01.02** 2002. január elsején 13 óra 13 perckor;

**now + 2 minutes (hours, days, weeks)** mostantól számított 2 perc (óra, nap, hét) múlva.

Ebbe a családba tartozik még a `batch`, ami akkor hajtja végre a feladatot, ha azt a rendszer terhelése megengedi. A feladat elvégzéséről a felhasználó egy levelet kap, amiből információt nyerhet az esetlegesen felmerülő hibák okairól is.

Van, akinek még ez is kevés, mivel egy feladatot többször, periodikusan akar elvégeztetni. Példaképpen lehet említeni, hogy a leveleit naponta szeretné egy másik levelezőszerverről letölteni a saját gépére. Neki sem kell a szomszédba mennie megoldásért, mert a `cron`d démon erre találta ki. Ez percenként „felébred”, és megnézi a hivatalosan a `/var/spool/cron` alatt lévő fájlokat, amiket a felhasználók egyéniileg a `crontab` programmal hozhatnak létre és módosíthatnak, és amik tartalmazzák azt, hogy milyen parancsokat és milyen időközönként kell végrehajtania. Ennek a fájl-nak a kinézete kötött, és így épül fel:

```
esetleges környezeti beállítások
perc óra hónap_napja hónap hét_napja parancs
```

Az első részben tehát olyan környezeti változókat adhatunk meg, amik a parancsok hibátlan lefutásához esetleg szükségesek. Ugyanis ezeket egy új héjprogramot nyitva, annak környezeti beállításával futtatja a démon. Majd következik az, hogy mikor és milyen parancsot kell lefuttatni. A percek 0-tól 59-ig, az órák 0-23-ig és a többi adat is értelemszerűen sorolandó fel (a hét napjainál a 0 vagy a 7 jelenti a vasárnapot, de lehet az angol rövidítéseket is használni). Néhány példából mindez világosabb lesz:

**0 12 1 \* \* parancs1** minden hónap elsején 12 órakor;

**30 0,12 \* \* Wed parancs1** szerdánként 0:30 és 12:30 órakor;

**0-59/20 \* \* Jul 1-3 parancs1** júliusban minden héten, hétfőtől szerdáig 20 percenként fut le a `parancs1`.

Látható, hogy itt is nagyon rugalmasan adhatóak meg az időintervallumok. Teljes leírásuk a `man 5 crontab` parancs kiadása után elérhető. Lássuk a gyakorlatban is ezt, adjuk ki a `crontab -e` parancsot, ezzel editálni tudjuk a fájlt (ha az `EDITOR` környezeti változónk nincs másra beállítva, akkor a `vi` programmal), és írjuk bele:

```
0-59/2 * * * * echo Hello >> ~/hello
```

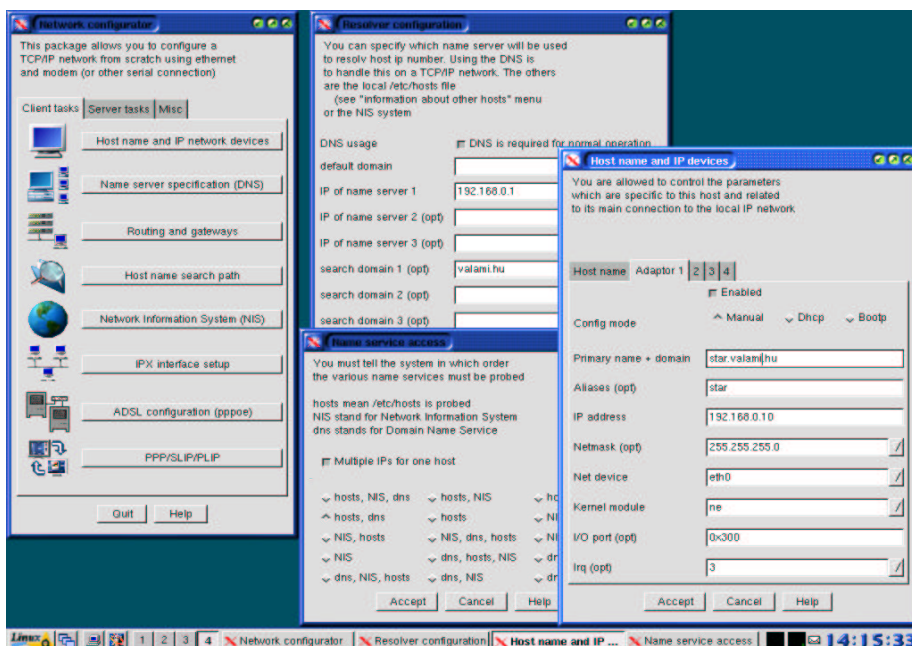
majd figyeljük a `/hello` fájlunkat, akár a `tail -f` igénybevételével! Kétpercenként egy újabb `Hello`-nak kell megjelenni ámuló szemeink előtt. Arról, hogy a fájl ne íródjon felül, hanem mindig hozzátcsolja az újabb szöveget a végéhez, a `>>` jelek gondoskodnak, tehát nem egyszerű átirányítás, hanem hozzáfűző átirányítás szükséges.

Mind az `atd` mind a `cron`d működéséről – jólnevelt démon-programokhoz méltón – a `/var/log` könyvtár alatti naplófájlok tájékoztatják a rendszergazdát, de ha úgy látja jónak, bármely felhasználót.

## 1.16. Hálózati eszközök kezelése, ügyfélprogramok

A Linux az Interneten született és fejlődik ma is, nem véletlen tehát, hogy igen hatékony eszközöket és lehetőségeket biztosít a hálózatokban előforduló minden problémára. Nemcsak *TCP/IP* alapú, vagy *Microsoft Network*, *Novell Netware* hálózatok aktív részese, hanem *OS/2*-t futtató vagy *Apple Macintosh* gépekkel is jól tud kommunikálni. Egy hálózatba kapcsolt gépen az alapvető beállításokat a rendszergazda végzi, de a felhasználók is tájékozódhatnak ezekről, és a nyújtott szolgáltatásokat is ők élvezik, fussunk át tehát ezeken a parancsokon, bár csak érintőlegesen, hiszen ezzel főleg egy másik füzet foglalkozik.

Arról, hogy milyen hálózati eszközök vannak a gépen, az `ifconfig` kiadásával szerezhetünk tudomást. A `lo`, vagyis *loopback*, magyarul visszacsatoló eszköz mindenképpen szerepelni fog a listán, és *Ethernet* hálózat esetén, ami a leggyakoribb, egy vagy több `ethx` eszköz, ahol `x` ezek sorszáma, 0-tól indulva. Mindegyik eszközről kilistázza annak típusát, IP címét, az alhálózati maszkot, a broadcast-címet és egyéb információkat. A `route` parancs a gép ún. útválasztási táblázatának kiírására és módosítására szolgál; ezen táblázat alapján tudja azt, hogy a hálózatban az adatokat merre kell továbbítani.



1.6. ábra. Egy mindentudó hálózati beállítóprogram

A fentieknél jóval gyakrabban használatos a `ping` program, amivel hálózati kapcsolatok működőképességét tudjuk tesztelni. Paramétere általában csak egy IP címmel vagy teljes domain-névvel adott gép, aminek olyan üzenetet küld, amire az köteles válaszolni (bár ez is letiltható, de általában nem szokták). Ebből látszik a két gép közti adatkapcsolat minősége is, mivel a válaszidőket is megjeleníti. Ha a `-c` szám paramétert megadjuk, akkor pontosan `szám-szor` küldi el a rövid kis adatcsomagját, egyébként a `CTRL-C` megnyomásáig folyamatosan bombázza a célgépet. A `traceroute`



ennél is érdekesebb eredményt ad, mivel a kiindulási helytől kezdve a megadandó cél-gépig felsorolja azokat az állomásokat, amiken az üzenet keresztülmegy. Nagyon jól használható arra, hogy lássuk, hol vannak szűk keresztmetszetek a hálózatban.

A `netstat` segítségével a működő hálózati kapcsolatokról lehet felvilágosítást nyerni. Paraméter nélkül kiadva valami hasonlót láthatunk:

```
[pistike:~]>netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 star.valami.hu:1042     king.valami.hu:ftp      ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags               Type                   State                  I-Node Path
unix    1      0 [ ]                 STREAM                 CONNECTED              917      @0000005d
unix    1      0 [ ]                 STREAM                 CONNECTED              920      @0000005e
unix    0      0 [ ]                 STREAM                 CONNECTED              220      @00000020
unix    1      0 [ ]                 STREAM                 CONNECTED             29463     @00000063
...
```

Itt éppen egy FTP-kapcsolat van a harmadik sor tanúsága szerint egy „king.valami.hu” nevű szerverrel. Az alsó részen a hálózati kapcsolatok számára nyitva lévő ún. *socket*ek láthatók. Ezek a kommunikáció végpontjai a gépen, minden adatcsatorna számára nyílik egy ilyen. A `-a` paraméterrel minden *socket*et kilistáz, még az éppen nem aktív, csak „figyelő” állapotban lévőket is, a `-c` pedig folyamatos figyelemre szólítja fel.

A `tcpdump` a hálózaton átmenő adatcsomagokat figyeli. Beállítható, hogy ezek közül csak bizonyos gépekre, szolgáltatásokra, protokollokra vagy a kapcsolat irányára stb. vonatkozó adatokat szűrjön ki számunkra. Rendszergazdáknak ajánlatos beszerezni az `iptables` programot is, ami menüvezérelt rendszerben, rengeteg szolgáltatást nyújt hasonló témakörben.

Más felhasználókról való információszerzési parancsokat már láttunk, akad azonban még néhány. Jelesül, ha valakiről szeretnénk megtudni, mikor lépett be legutóbb a rendszerbe, akkor a `last` felhasználónév gépelendő be, például:

```
[root:~]>last pistike
pistike  tty3          Tue Jul 10 21:23 - 22:15  (00:51)
pistike  tty2          Tue Jul 10 21:23 - 22:14  (00:51)

wtmp begins Sun Jul  1 19:01:52 2001
```

A legutolsó sorban látható, hogy ezeket az adatokat a rendszer egy `/var/log/wtmp` nevű fájlban tárolja, aminek indulási időpontja is leolvasható, tehát tulajdonképpen csak az azóta jegyzett belépéseket láthatjuk itt. Ha minden egyes felhasználóról látni akarjuk ezeket az adatokat, ahhoz pedig a `lastlog` parancs kell; ez csak a legutolsó belépés dátumát adja meg, vagy ha az illető a `/var/log/lastlog` frissítése óta nem lépett be, akkor a `Never logged in` felíratot.

Nagyon sok információt kaphatunk valakiről, aki akár egy távoli gépen lehet regisztrált felhasználó, a `finger` program segítségével. Ahhoz, hogy működjön – azon a gépen, amin a célszemélyünknek van azonosítója – futnia kell a `fingerd` démonnak, de ha ez teljesül, akkor hasonló kép tárulhat elénk:

```
[root:~]>finger root@king.valami.hu
[king.valami.hu]
Login: root                Name: root
Directory: /root          Shell: /bin/bash
On since Wed Jul 16 13:30 (CEST) on tty1  2 seconds idle
      (messages off)
No mail.
Plan:
Break the gates!
```

Tehát csak az adott személy e-mail címe kell, és máris sok mindent megtudunk róla, még azt is, mit tervez (a *Plan:* után látható, ez az illető saját könyvtárában lévő *.plan* fájl tartalma). Éppen emiatt sok helyen az első dolga a rendszergazdának, hogy a többi nem túl biztonságos vagy felesleges szolgáltatás mellett ezt is leállítsa a szerverén, mivel a crackerek számára jó kiindulópont lehet.

Néha jólesik pár szót szólni valakihez, még ha elektronikus formában is. Rövid üzenetet küldhetünk a *write* parancs felhasználásával egy velünk együtt bejelentkezett felhasználónak. Lépünk be megint pistike nevében, majd rootként írjuk be:

```
[root:~]>write pistike
<userinput>hello!!!
```

és nyomjunk a végén *CTRL-D* kombinációt (ami – tudjuk – az adatbevitel végét jelenti). Ennek hatására (hacsak pistike nem tiltotta le az üzenetküldést a *msg n* parancssal), a terminálján megjelenik az üzenet, valamint az, hogy ki küldte és mikor, valamint az EOF, az „üzenet vége” jel. Ez a kommunikáció egyirányú, ha valóban „csevegni” akarunk egymással, akkor ezt a *talk* usernév utasítással kezdeményezhetjük. Szintén szükséges hozzá a gépen futó *talkd* vagy *ntalkd* nevű démon. A kiválasztott felhasználó terminálján olyan tartalmú üzenet jelenik meg, amiben látható, ki hívja csevegésre, és hogyan lehet vele felvenni a kapcsolatot. Ha például a rendszergazda akar velünk beszédbbe elegyedni, ne tétovázzunk válaszolni neki a *talk root* begépelésével, ekkor mindkét fél képernyőjén a *Connection established*, vagyis a „Kapcsolat létrejött” látható. Innentől kezdve amit begépelünk, az a két részre osztott képernyő felső, míg a partnertől érkező válaszok az alsó részén olvashatók. Kilépni a *CTRL-C* megnyomásával lehet. Végül, a rendszergazda a minden felhasználónak szóló üzeneteket a *wall* parancssal küldheti ki. Példának okáért ha az „Ideje hozni a söreimet!” kell hogy megjelenjen mindenkinél, akkor a teendője a következő: a parancs kiadása után egy *ENTER*t kell nyomni, beírni a szöveget (minden sor után *ENTER*rel), majd az utolsó üres sort *CTRL-D*vel lezárva már várhatja is jól megérdemelt jutalmát. . .

Az Internet szolgáltatásainak igénybevételére szolgáló programok tömkelege használható Linux alatt, az egyszerű levelezéstől kezdve a manapság divatos fájlmegosztási programokig. Anélkül, hogy részletesen ismertetnénk a használatukat, álljon itt mégis azoknak a főbb programoknak a listája, amiket jórészt minden gépen elérhetünk:

**mail, pine, mutt** elektronikus levelezésre;

**ftp, ncftp** FTP szolgáltatás igénybevételére;

**lynx** web-böngészésre karakteres üzemmódban;

**tin, rtin, trn** a *newsgroup*ok, hírcsoportok olvasására;

**telnet, ssh** távoli gépek elérésére;

**irc** az *Internet Relay Chat*, vagyis világméretű csevegés igénybevételére;

**wget** WWW- és FTP-szerverek tartalmának rekurzív letöltésére, tükrözésére.

## 1.17. Kernel és program építés, kernel modulok

A Linux és alkalmazásai, szabad forráskódú rendszer lévén, teljes egészében hozzáférhető az Interneten olyan formában, hogy a programozók könnyen tudják módosítani,

javítani. Ezek a forrásnyelvű állományok az esetek többségében `.tar.gz` vagy újabban `.tar.bz2` kiterjesztéssel forgalomba kerülő archívumok, amelyek kezelését már ismerjük. Kicsomagolásuk után rendelkezésünkre áll az adott program kódja azon a programnyelven, amin megírták. Ez döntő többségben a `C` vagy `C++`, bár néha mellékelik a bináris, lefordított változatot is. Ezek után mindig olvassuk el a mellékelt `README` illetve `INSTALL` fájlokat, mert ezek írják le a pontos telepítési menetet: milyen egyéb programok, tárgykódkönyvtárak megléte szükséges a futáshoz, hova kerül telepítés után a program, hogyan kell indítani, valamint esetlegesen azt, hogy milyen szolgáltatásokat lehet a fordításnál engedélyezni vagy tiltani. A következő lépés, hogy ha van a forrásfában egy `configure` nevű állomány, akkor annak a lefuttatása. Ez ugyanis végignézi a gép aktuális beállításait, összegyűjti a szükséges adatokat, és ezek alapján készíti el az ún. `Makefile` nevű állományt, ami alapján a fordítás menetét vezérlő program, a `make` dolgozni tud. A programok kódja nagyon ritkán érkezik egyetlen fájlban, a legtöbbször rengeteg részből áll, ezek a `.c` illetve a `.h` kiterjesztést viselik (ez utóbbiak az ún. header-fájlok, amik a programrészekben több helyen is előforduló, felhasznált konstansokat, deklarációkat hordozzák). A `make` feladata, hogy a programot a megfelelő sorrendben a megfelelő programokkal lefordíttassa. Ezek lehetnek a `gcc`, a Linux `C`-fordítója, az `as` assemblerprogram, a `strip`, ami a kész binárisból a nyomkövetéshez szükséges információkat távolítja el, vagy az `install`, ami a végén a megfelelő helyre másolja a lefordított programot és komponenseit, és még több más segédprogram is. Amikor tehát a `configure` szkript elkészült a `Makefile` létrehozásával, utána a legtöbbször csak egy `make`, majd egy `make install` parancs szükséges ahhoz, hogy települjön a program, leginkább a `/usr/local/` alatti könyvtárban belül.

Van, amikor ez nem megy ilyen egyszerűen, ennek oka legtöbbször az, hogy hiányzik egy olyan programcsomag, amire az új alkalmazásnak szüksége van. Ekkor azt is be kell szerezni, lefordítani, és remélhetőleg ezek a függőségi problémák megszűnnek. Ha egy olyan programot szereztünk, ami eleve bináris formátumban volt, és szeretnénk meggyőződni arról, hogy minden hozzávaló megvan-e a gépünkön, akkor adjuk ki az `ldd` program parancsot, ez a dinamikusan szerkesztett programról megjeleníti a hozzá szükséges tárgykódkönyvtárak neveit, mint az alábbi példában is:

```
[root:~]>ldd /bin/sh
libreadline.so.4.1 => /lib/libreadline.so.4.1 (0x40020000)
libhistory.so.4.1 => /lib/libhistory.so.4.1 (0x40048000)
libtermcap.so.2 => /lib/libtermcap.so.2 (0x4004f000)
libdl.so.2 => /lib/libdl.so.2 (0x40053000)
libc.so.6 => /lib/libc.so.6 (0x40057000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Abban az esetben, ha a telepítendő program tartalmaz ilyen új tárgykódkönyvtárakat, és a telepítője nem gondoskodik erről, adjuk ki az `ldconfig` parancsot rendszergazdaként! Ennek hatására frissül az az adatbázis, amiben a rendszer tárolja a *libek* elérhetőségeit, ez a `/etc/ld.so.cache` állományban található.

Nézzük most végig az operációs rendszer lelkét képező rendszermag, vagyis a kernel újrafordításának és telepítésének menetét. Ere a lépésre akkor van szükség, ha a „gyárilag” érkező kernelben nincsenek, vagy nem megfelelően vannak meg azok a szolgáltatások, amik nekünk kellenek, új lehetőségeket építettek be, biztonsági réseket kell betömni vagy egyszerűen csak szeretnénk jobban kézben tartani a rendszerünket. A kernel fejlesztése folyamatosan zajlik, mégpedig két fő szálon: van egy fejlesztői ág, amelyen a legfrissebb, de emiatt még nem kipróbált technológiák, új vagy javított szolgáltatások épülnek be, és van az „éles” felhasználásra szánt stabil verziók ága; ide akkor kerül új verzió, ha már valamennyi ideig tesztelték. Jelen pillanatban

(2002. január közepén) a 2.4.17 a legfrissebb stabil verzió, ezt fogjuk a példában használni. A verziók számozására egyébként az a szabály, hogy az első a főverziószám, ez most a 2, utána páros szám következik ha stabil, páratlan, ha fejlesztői változatról van szó (még nem állt le a 2.2.x széria fejlesztése sem), a végén pedig az ezen belüli sorszám található. A forrás hivatalosan az `ftp://ftp.kernel.org/` szerverről tölthető le, de van magyar tüköroldala is, például az `ftp://ftp.kfki.hu/` kiszolgálón. A fájl neve `linux-2.4.17.tar.gz`, ezt másoljuk be a `/usr/src/` alá, majd adjuk ki a `tar zxvf linux-2.4.17.tar.gz` utasítást a kicsomagoláshoz. Ezek után csak egy `linux` nevű könyvtár keletkezik, de célszerűbb (amennyiben több kernelverziót is őrzünk itt), hogy átnevezzük a verziószámának megfelelően, majd készítünk rá egy szimbolikus linket `linux` néven. Ezek után tehát hasonlót kell látnunk:

```
[root:/usr/src]>ls -l
total 164
lrwxrwxrwx    1 root    root          11 Dec 21 20:07 linux -> linux-2.4.17/
drwxr-xr-x    17 root    root        4096 Apr 29 14:35 linux-2.2.17/
drwxr-xr-x    14 1046    utmp       4096 Dec 21 20:39 linux-2.4.17/
```

Lépünk még be a `./linux/include` alkönyvtárba, és ellenőrizzük, hogy van-e `asm` nevű alkönyvtár, ami a gép kiépítésének megfelelően PC-n az `asm-i386` könyvtárra mutat. Ha megvan, következhet a konfigurálás, de előtte szedjük össze minden ismeretünket a gép hardverére vonatkozóan, mert szükség lesz rá! Álljunk be tehát a `/usr/src/linux` könyvtárba, és adjuk ki a `make mrproper` parancsot. Ez nem életbevágó, de érdemes használni, mivel utána tiszta lappal indulhatunk neki a beállításoknak, illetve olyan alapvető problémák is kiderülhetnek, hogy például nincs is telepítve a `make`. Következhet a `make config`, de inkább használjuk a `make menuconfig`, vagy `make xconfig` parancsokat, mindegyik a beállítási lépéseken való végigvezetést szolgálja.

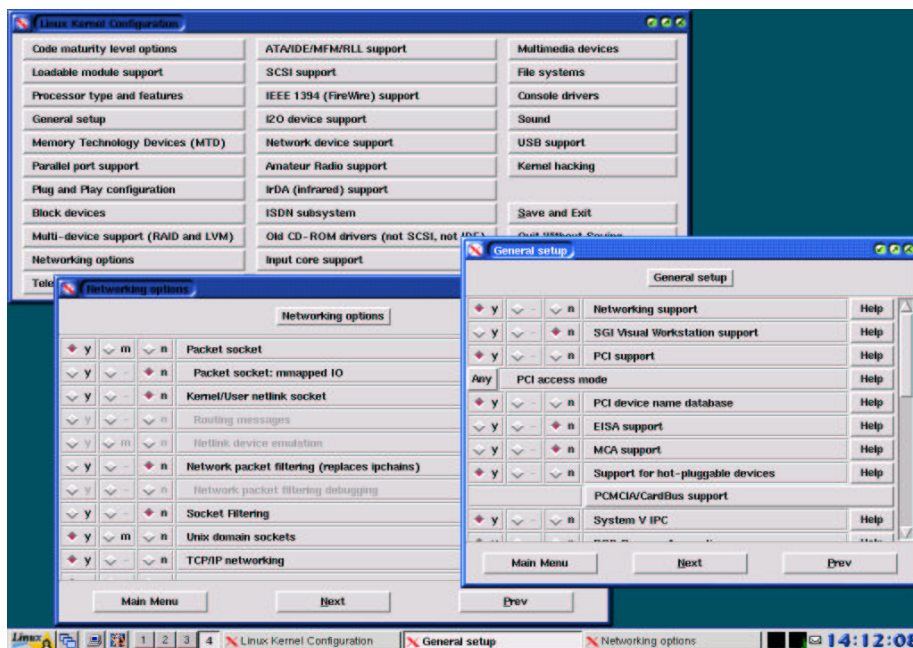
Arany középutat választva a `make menuconfig` kezelést tárgyaljuk. Ez szöveges módban, a nyílbillentyűk és az *Enter* használatával kezelhető. Több egymás alatti menüponton keresztül férünk hozzá az egyes konfigurálható elemekhez, mivel nagyon sok van belőlük, csak a főbb pontokat nézzük át:

**Code maturity level options** – ezen belül válasszuk ki a `Prompt for . . .` kezdetű sort, ekkor több olyan beállítási lehetőséget is elérünk majd később, amit a fejlesztők még nem ítélték 100 százalékosan biztonságosnak és működőnek. A beállítást egy `*` karakter jelzi.

**Loadable module support** – érdekes és hasznos lehetősége a kernelnek, hogy bizonyos részei ún. modulokba fordíthatók. Ezek lehetnek olyan hardvereszközök meghajtói, amik csak időlegesen használatosak, vagy ritkábban igénybe vett szolgáltatások kódjai. A memóriába csak akkor töltődnek be, ha valóban szükség van rájuk, ezért nagyon gazdaságos a használatuk. Most engedélyezzük őket! Ezek után majd sok eszköznél felkínálja a program a modulba való fordítás lehetőségét is.

**Processor type and features** – a központi egység pontos típusa adható meg itt. Gyorsíthatja a gép működését, ha kihasználjuk az újabb processzorok adta lehetőségeket. Itt kell a többprocesszoros rendszerekhez is a kernel támogatását engedélyezni.

**General setup** – legyen-e hálózati támogatás (mindig válasszuk ki!), *PCI*, *EISA*, *MCA* eszközök támogatása, milyen programkód-formátumokat ismerjen a kernel (álítsuk be mindig fixen az ELF-et!).



1.7. ábra. A make xconfig eredménye

**Parallel port support** – párhuzamos port és paramétereinek beállítása.

**Plug and Play support** – a „magától működő” eszközök kezelésének lehetősége.

**Block devices** – többek között a hasznos *loopback* és *ramdisk* eszközök beépítési lehetősége. Fordítsuk ezeket modulba, ehhez egy *m* betűt kell nyomni az adott eszköz sorában, vagy a szöveget lenyomni párszor, amíg az *M* betű meg nem jelenik.

**Multi-device support** – a szervereken alkalmazott *RAID* és *LVM* technikák támogatása.

**Networking options** – a hálózati működéssel kapcsolatos, rendszerszintű beállítások, amik közül a *TCP/IP* mindig legyen bekapcsolva.

**ATA/IDE/MFM/RLL support** – a felsorolt típusú adathordozók, és bizonyos alaplapi chipset-ek támogatása.

**SCSI support** – önmagáért beszél...

**Network device support** – a hálózati kártya típusának megadása található ezen belül. Ha nem tudjuk pontosan, próbálkozhatunk az *NE2000* kompatibilis típussal, a legtöbb kártya ismeri ezt a szabványt.

**Character devices** – a virtuális terminálok beállításai, valamint a nem soros porti egerék támogatása.

**File systems** – a különböző fizikai és hálózati fájlrendszerek támogatása. Állítsuk be a *vfat* és *msdos* típusokat, célszerű még a *minix*et is, ha Linuxos floppykat használunk, de az *ext2* fix beállítását sose feledjük! A hálózati részen az *smbfs* jöhet jól *Samba*, illetve *ncpfs* *Netware* fájlkiszolgálók használata esetén.

**Sound** – a hangkeltő eszközök támogatása.

A többi menüpont leginkább a különleges eszközök (*ISDN*, *USB*- és *infraport*, *FireWire*, stb.) támogatásának beállítására használható. Elmenthetjük az így elkészült konfigurációt egy külön állományba is, majd az *Exit* gomb segítségével kilépve, a *.config* fájlban rögzítődnek a beállítások. Következhet a program részei közötti függőségek megállapítása (*make dep*), majd utána egy kis takarítás a *make clean* kiadásával, és aztán a tulajdonképpeni fordítás a *make bzImage* parancs segítségével. Ennek hatására a forráskönyvtár *arch/i386/boot* alkönyvtárában kapjuk majd meg a kernelt, mint egy *bzImage* nevű fájlt. Hiba remélhetőleg nem fordul közben elő, a géptől függően pár perc (óra...) alatt végez is ezzel a részével a dolognak. Ha rendben lefutott, látjuk, hogy mekkora méretű lett, általában 6-700kB-nyi a szokásos. Folytassuk a *make modules* parancs kiadásával, ez azokat a részeket fordítja le, amiket modulba szánunk. Sikeres lefutás esetén kell még a *make modules\_install* is, ezzel a telepítésük is lezajlik, mégpedig rendszerint a */lib/modules/kernelverzió* könyvtárba kerülnek az ezen belüli megfelelő alkönyvtárakba szétszétva, *.o* kiterjesztéssel.

Már csak az van hátra, hogy ezzel a kernellel tudjuk indítani a gépet. Ehhez a *lilo* programot használjuk fel, ami a Linux betöltésvezérlője, bootmanagere. A kernel fájlt előbb a */boot* könyvtárba kell másolni, majd a */etc/lilo.conf* állományt szerkeszteni. Ebben a meglévő kernelt (mint biztonsági másolatot) mindenképpen hagyjuk meg először, tehát másoljuk le a rá vonatkozó részt, ami valahogy így nézhet ki:

```
image=/boot/vmlinuz
label=linux
root=/dev/hda1
read-only
```

majd javítsuk át úgy, hogy hasonló legyen ehhez:

```
image=/boot/bzImage
label=linux
root=/dev/hda1
read-only

image=/boot/vmlinuz
label=old
root=/dev/hda1
read-only
```

Végezetül a */etc* könyvtárban állva adjuk ki a *lilo* parancsot, ez elkészíti az új konfigurációt, és ki is írja, milyen lehetőségek közül választhatunk majd a rendszerindításkor, ha a megjelenő *LILO boot:* feliratnál a *TAB* billentyűt nyomjuk le; ekkor ugyanis kiírja az indítható rendszerek címkéit, tehát ott kell látnunk az *old* és a *linux* feliratot is, ezek közül alapértelmezés szerint a *linux* töltődik be, remélhetőleg minden gond nélkül. Természetesen mindez csak akkor igaz, ha a fő betöltésvezérlő a *lilo*. Ha nem ez a helyzet, akkor a *lilo.conf* fájlban nem a *boot=/dev/hda* feliratot látjuk, hanem például a *boot=/dev/hda1* sort, de a fent leírtakra ez gyakorlatilag nincs hatással: ha már a *lilo* megkapta valahogy a vezérlést, be tudja tölteni az új kernelt is.

A modulokat menet közben, alapelvüknek megfelelően betölthetjük/eltávolíthatjuk az `insmod/rmmod` párossal, feltéve, ha semelyik más modul működését nem sérti, ami velük kapcsolatban van (ugyanis itt is megfigyelhető egymásraépülés). A `modprobe` szintén a beillesztéshez használható, igyekszik az összes olyan modult is beilleszteni a kernelbe, ami annak működéséhez szükséges. Mindháromnak egy paramétere van, az adott modul neve. Az `lsmod` pedig a betöltött modulokat listázza ki. Megemlítendő még a `/etc/modules.conf` konfigurációs állomány, amelyben a modulok paramétereit lehet megadni. Ha kernelünkben megtalálható a `kerneld` (vagy az újabb nevén `kmod`) démon támogatása, akkor pedig mindezen feladatokat automatikusan elvégzi. Tehát ha például egy zeneprogramot indítunk, beilleszti a hangkártya meghajtóit. Sőt miután a modulokra már nincs szükség (több mint egy percig nem használja őket semmi), akkor el is távolítja őket.

## 1.18. A héjprogramozás segédeszközei

Szó volt róla, hogy a `bash` (és persze a többi parancsértelmező is) nagyon jól programozható, saját programnyelvre segítségével. Kicsit mélyedjünk el ezekben a lehetőségekben!

A héjprogramok vagy shell-szkriptek mind olyan szöveges állományok, amiknek végrehajtási jogot adva önállóan is futtathatók, de a shell segítségével is megoldható a végrehajtásuk. Kezdjük egy egyszerű példával, aztán majd bonyolítsuk kicsit: írja ki a programunk a „Halihó!” szöveget! Ehhez gépeljük be a `progi` nevű szövegfájlba a következőt:

```
#!/bin/bash
echo "Halihó!"
exit 0
```

Az első sor értelmezése: a `#` jel után következő részt a parancsértelmező megjegyzésként kezeli, kivéve, ha egyből a felkiáltójel következik utána, mert akkor azaz a programmal próbálja meg végrehajtani a szkriptet, ami ezután áll. A második sor ismerős, a harmadikban lévő kóddal pedig úgy lépünk ki a végrehajtásból, hogy egy ún. visszatérési értéket is beállítunk. A legtöbb program lefutása után beállít egy ilyet, és ezt egy programmal lekérdezve megtudhatjuk, hogy végződött a parancs. A 0 érték általában azt jelenti, hogy rendben lefutott. Adjunk ennek a fájlunk a `chmod +x progi` utasítással mindenki számára végrehajtási jogot, majd futtassuk a `./progi` kiadásával! Ez még így nem túl látványos, ezért bővítsük úgy, hogy előtte képernyőt törölünk, és az üdvözlő szöveg után kiírjuk, hogy „Most óra:perc van.”! A program második sora legyen tehát a `clear`, az utolsó előtti pedig:

```
echo -n "Most "`date +%T`; echo " van."
```

Itt több újdonság is előbukkant. Először is, a `-n` paraméter hatására az `echo` nem emel sort a kiírás végén, tehát majd a `" van"` nem kerül új sorba. A pontos idő beillesztése a szövegbe a `date` feladata lesz, és ezt most úgy oldjuk meg, hogy fordított aposztrófok között adjuk meg a parancsot. Ennek hatására futáskor a `bash` be fogja helyettesíteni a megfelelő értéket, és azt fogja kiírni; vagyis a `date` kimenetét beágyaztuk egy másik parancssorba! Ezt nagyon sok helyen ki lehet használni, például találjuk ki, mi lesz az eredménye a

```
echo "Most `w -h | wc -l` felhasználó van bejelentkezve."
```

parancsnak?

A programokban szinte mindig vannak elágazások, logikai döntéshozatalok. Valamilyen változó értéke befolyásolhatja, hogyan folytatódjon a végrehajtás. A `test` programnak nagy szerep jut ilyenkor, mivel megvizsgál bizonyos dolgokat, és az előbb emlegetett visszatérési értékek segítségével értesít annak eredményéről. El tudja dönteni, hogy két (számszerű vagy szöveges) érték hogyan viszonyul egymáshoz, egy adott állomány létezik-e és milyen típusú, méretű, és ezeket felhasználva sok hasznos dolgot tehetünk. Nézzük meg próbaképpen, hogy létezik-e a `/home/kispista` nevű könyvtár:

```
[root:~]>test -d /home/kispista && echo van || echo nincs  
nincs
```

A `-d` kapcsolóval tehát azt vizsgáljuk, van-e ilyen. Két eset lehetséges: ha van, akkor 0 a visszatérési érték, nincs hiba, ekkor a `&&` jelek után következő parancs fog lefutni. Kedvezőtlen esetben azonban a visszatérési érték eltér nullától, ekkor viszont a `||` jelek utáni parancs indul el. Ilyen egyszerű, két kimenetelű elágazást tehát már tudunk is készíteni. Ugyanezt nézzük meg úgy is, ahogy inkább használni szoktuk héjprogramokon belül:

```
[root:~]>if [ -d /home/kispista ]; then echo van; else echo nincs; fi  
nincs
```

A programozásban jártasak ismerik már az *if-then-else* (magyarul ha-akkor-különben) szerkezetet, itt is ezt látjuk viszont, egyes részei egymástól a pontosvesszővel vannak elválasztva, és a `fi` kulcsszó zárja le. De mit jelent a szögletes zárójel, és hova tűnt a `test`? Nos, a `[` egy szimbolikus link a `test` állományra, ami figyel ilyenkor arra is, hogy legyen záró `]` jel is. Most teszteljük le, hogy nagyobb-e nullánál a `/etc/motd` mérete:

```
[root:~]>if [ -s /etc/motd ]; then echo nagyobb; else echo zerus; fi  
zerus
```

Ezen a gépen tehát éppen a „különben” ág futott le, mivel a `-s` akkor ad *IGAZ* eredményt, ha a vizsgált fájl létezik, és mérete nagyobb mint zérus. Vizsgáljuk most meg, hogy pistike *User ID*-je nagyobb-e 500-nál! Ehhez első lépésként az értéket beillesztjük egy környezeti változóba, majd ezt adjuk át a `test` parancsnak:

```
[root:~]>export pistiid='id -u pistike'  
[root:~]>echo $pistiid  
502  
[root:~]>if [ $pistiid -gt 500 ]; then echo nagyobb; fi  
nagyobb
```

Látható, hogy a parancsok kimenetének beágyazásával is lehet környezeti változót feltölteni értékkel, és ezt a jelen esetben számszerű értéket a `-gt` (*Greater Than*, nagyobb mint...) kapcsoló segítségével vetettük össze az 500-as számmal.

A `bash` programokon belül használható vezérlőszervezetek az *if-then-elsen* kívül még a többágú elágaztatást végző *case*, a ciklusok létrehozására szolgáló *for*, *while*, *until* és *select*. Ezek közül most csak a léptető típusú *for* ciklusra álljon itt egy példa:

```
[root:~]>for i in *.mp3; do mpg123 "$i"; done
```

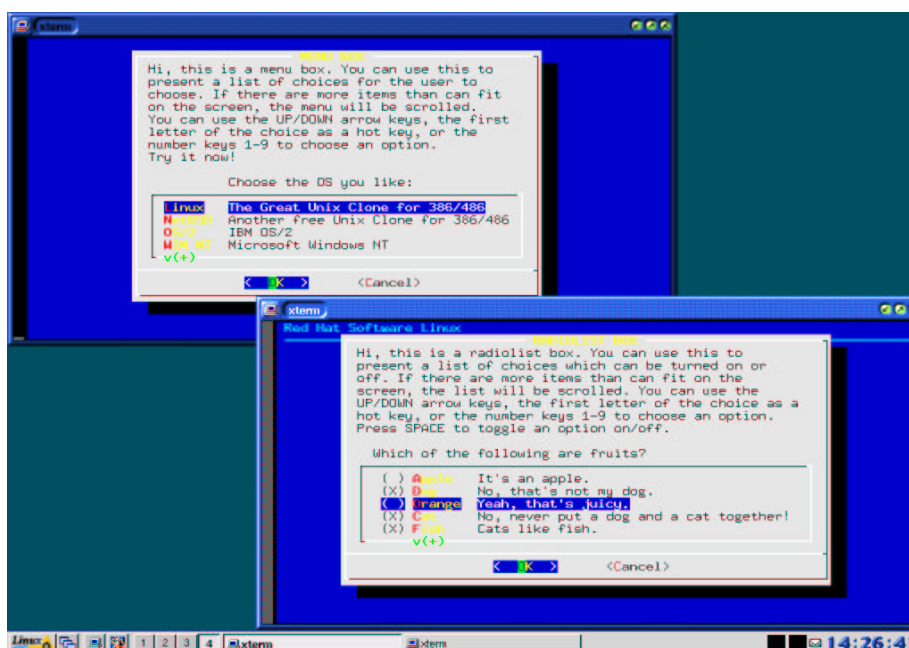
Ennek hatására az aktuális könyvtárban lévő minden *.mp3* kiterjesztésű fájlt le fogja játszani az `mpg123` nevű program, még akkor is, ha szóköz van a nevükben. Az `i`



ciklusváltozó tulajdonképpen egy szöveges környezeti változóként viselkedik, értékei sorban jönnek létre, majd azokat olvassa ki az `mpg123`.

A `sleep` parancsot is gyakran használjuk fel héjprogramokban, ha időzíteni problémák lépnek fel. Az utána írt szám és kód jelenti azt az időtartamot, ameddig vár és nem csinál semmit: tehát a `sleep 5` hatására a program 5 másodpercig várakozik, csak utána fut tovább. Megadható még az `m` utótag a percben, a `h` az órában és a `d` a napban mért időtartamokhoz.

A programba való adatbevitelre szolgál a `read` változó utasítás. A billentyűzetről vett értéket a környezeti változóban tárolja. Ha ennél kicsit látványosabb módot akarunk a felhasználónak biztosítani, akkor a `dialog` programot használjuk. Ennek különböző típusú, adatbekéréshez használt elemek a fő paraméterei, nevezetesen az adatbeviteli mező, a jelölőnégyzet, rádiógomb, lista, stb. A használata ezek miatt bonyolultabb, de valóban szép és könnyebben használható felhasználói felületet tudunk kialakítani a segítségével (1.8. ábra).



1.8. ábra. Néhány dialog-gal készült program

## 1.19. Az X Window rendszer alapparancsai

A UNIX rendszerek nagyon jól használható grafikus felülettel is rendelkeznek, amit *X Window System*nek hívnak. Ezt a rendszert hálózati alkalmazás céljára fejlesztették ki, emiatt olyan trükköket lehet vele csinálni, amit egy hagyományos PC-s környezettől nem szokhattunk meg. Működésére az jellemző, hogy a grafikus megjelenítővel rendelkező gépen fut egy ún. X szerver, aminek annyi a dolga, hogy a grafikus hardver képességeit messzemenőkéig kihasználva a képernyőre rajzolja mindazt, amit a hozzá kapcsolódó kliensek – maguk a programok – kérnek tőle. A klienseknek nem muszáj a

helyi gépen futnia, sőt megoldható az is, hogy a dologba egy harmadik gépet is bevonnak: egy nagy kapacitású gépet kér meg az egyikük (a kliens) arra, hogy elvégezze a bonyolult számításokat igénylő műveletet, és az eredményt grafikus formában küldje át egy harmadik (az X szervert futtató) gép képernyőjére. . .

A Linux alatt ingyenesen hozzáférhető X verzió az XFree86, ami nemrég esett át egy nagyobb verzióváltáson. Jelenleg a 4-es főverziószámú kiadások a legújabbak, de még sok videokártyához jobban használhatók a régi, 3.3.x verzió speciálisan hozzájuk írt szerverei (például az S3 kártyacsaládhoz). Az itt felsorolt parancsok működnek mind az új, mind a régebbi verziókban.

Mielőtt nekilátnánk az X felélesztésének, pontos adatokkal kell rendelkezniünk a videokártyánk és a monitorunk képességeiről és azokról a jellemzőkről, amik nélkül nem tudjuk teljes egészében kihasználni a képességeiket. A videokártyák legfontosabb jellemzői a felbontás és a színmélység valamint a videomemória mérete, a monitoré pedig a vízszintes és a függőleges képfrissítési frekvencia.

A videokártya paramétereinek lekérdezésére használható a SuperProbe program, ami igyekszik kitalálni a beállításához szükséges adatokat. A kártya gyártójára és a típusra utaló chipset meghatározása általában sikerül neki, de az már nem mindig, hogy mennyi a videomemória mérete. Viszont ha teljesen tanácstalanok vagyunk ezekkel kapcsolatban, akkor valamit ez is segít.

A konkrét beállításhoz többféle program is használható, a Linux terjesztések is sokféle megoldást kínálnak, ezek többségével ma már könnyen belőhető az X. Meg kell említeni itt a Red Hat-beli XF86Setup programot, a SuSE disztribúció SaX, illetve SaX2 nevű alkalmazását, vagy a Mandrakeben ismert XFdrake programot. Van azonban egy olyan eszköz, ami mindegyiknek része, és a legrégebben használatos, ez pedig az `xf86config`. A programon végighaladva először a digitalizáló eszközt kell beállítanunk, ez nagyrészt valamilyen egér. Majd a billentyűzet következik, itt használhatunk többféle kiosztást is, az újabb megoldás az ún. *XKB* használata. A monitor vízszintes frissítési frekvenciáját kell ezután kiválasztani, itt már szükség lehet a fent említett pontos adatokra, mert nagyobb érték megadása, mint amit bír, károsíthatja a monitort. Ugyanígy kell kiválasztani a függőleges frekvenciát is, majd lehet egy nevet adni a monitorunknak, hogy később, a konfigurációs fájl esetleges szerkesztésekor könnyebben megtaláljuk.

Következhet a videokártya kiválasztása. Ha nem szerepelne a miénk a hosszú listán, akkor sem kell feladni, egy SVGA szerverrel nagy valószínűséggel menni fog. Meg kell adnunk a videomemória méretét, és a kártyának is egy azonosító szöveget. A színmélység beállításával azt adjuk meg, hogy egy-egy képpont, pixel hányféle színt vehet fel: ajánlott legalább 16 bites vagy annál nagyobb értéket megadni, ha szép képet akarunk. Végül pedig a régi verziónál rákérdez arra, hogy elkészítse-e azt a szimbolikus linket, ami a `/usr/X11R6/bin` könyvtárban belül a kiválasztott X szerverre mutat és X a neve. Az új verzióban egyetlen X szerver van, így itt ez a lépés kimarad. Végül pedig felajánlja az XF86Config konfigurációs fájl mentését, ez legtöbbször a `/etc` könyvtárban vagy annak X11 alkönyvtárában található.

Ha jól dolgoztunk, máris indíthatjuk a felületet a `startx` parancs kiadásával. Ez egy héjprogram, és tulajdonképpen az `xinit` programot hívja meg a megfelelő paraméterekkel. Jó esetben elindul az X szerver, az esetlegesen automatikus futásra beállított egyéb programok, és végül az ún. ablakkezelő. X alatt ugyanis az a helyzet, hogy nem maga a szerver gondoskodik az ablakok kinézetéről, dekorációjáról, hanem egy erre szakosodott program, az ablakkezelő. Ilyenből nagyon sokféle létezik, kezdve a nagyon egyszerű, de villámgyors fajtáktól (*icewm*) a középkategórián át (*Windowmaker*, *Afterstep*) egészen a csillogó-villogó, de nagyon erőforrásigényes verziókig, amik

inkább már teljes ablakozó rendszernek nevezendők. Jelenleg ezekből két nagyon jó változat van, a *KDE* és a *Gnome*. Végül kapunk egy munkafelületet, amin elkezdhetünk dolgozni, de a jó öreg szöveges konzolokról sem kell lemondanunk, mert a *CTRL-ALT-Fx* kombinációkkal átválthatunk rájuk, vissza a grafikus felületre pedig az *ALT-F7* billentyűvel. Megjegyzendő még a *CTRL-ALT-PLUSZ* és *CTRL-ALT-MINUSZ* billentyűhármas, ezekkel a felbontást lehet váltani, valamint a *CTRL-ALT-BACKSPACE*, amivel vészhelyzetben gyorsan leállítható az X szerver. Első futtatáskor szükség lehet az *xvidtune* programra, aminek a segítségével a monitoron megjelenő képet lehet korrigálni, például mind a négy irányba elcsúsztatni. A beállításokhoz tartozik az is, hogy a billentyűzet jól működjön, erre – ha nem az *XKB* típusú billentyűzetkezelést választottuk az alapszintű konfigurálásnál – az *xmodmap* program használható. A *loadkeys* parancsnál látott módon X alatt is minden billentyűhöz tartozik egy kód, de ezeket máshogy értelmezi, viszont itt is szabadon átkonfigurálható a kiosztás, lementhető egy fájlba, és ezt kell megadni paraméterként az *xmodmap* számára.

Az *xset* és az *xsetroot* parancsok segítségével szintén beállításokat végezhetünk. A második az egyszerűbb, mert a szerepe csak az, hogy a háttérablak, az ún. *root-window* színét vagy éppenséggel a rajta megjelenő képet szabályozza, beállítsa. Az *xset* viszont jóval több mindent állít: a hibajelzés csipogásának hangerejét, az energiatakarékos üzemmód ki-bekapcsolását a monitorra, a megjelenített képernyőfontok elérési útjának szabályozását, a képernyővédő paramétereit mind-mind el lehet vele érni.

Az X használata közben ablakban futó terminálokat is nyithatunk, ezekből több típus létezik, általában mindenhol jelen van az *xterm* (1.9. ábra). Sokféle módon konfigurálható a megjelenése és a viselkedése, akár menet közben is, ha az ablaka fölött a *CTRL* billentyűt lenyomva tartva kattintunk az egér gombjaival, és az így megjelenő menüből választunk. A többi terminálemulációs program közül szerepeljen még itt az *rxvt*, ami szintén elég sok helyen fellelhető.

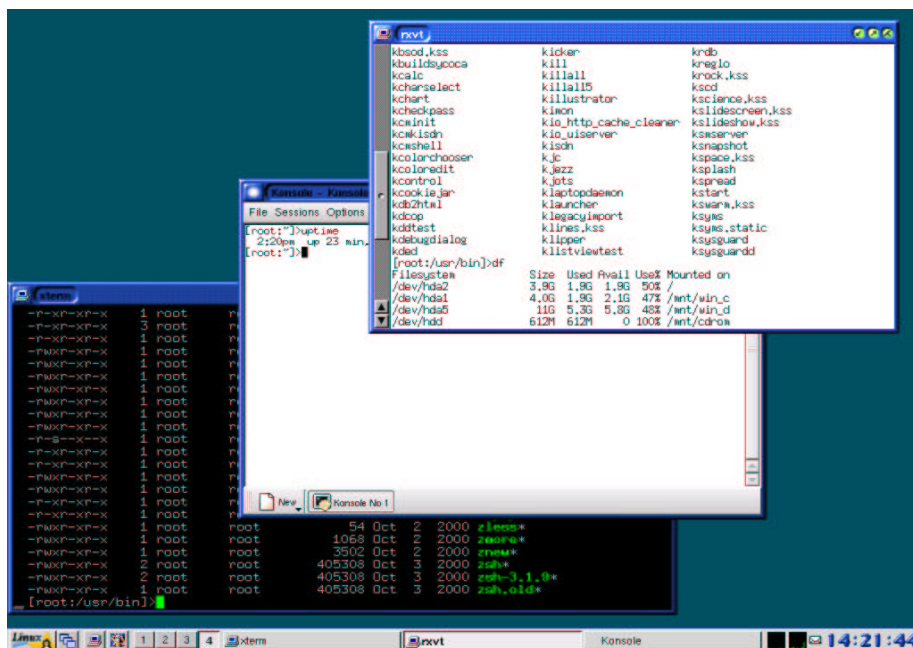
Sokszor ezeket a programokat már a rendszer indítása után egyszerre szeretnénk használni, ekkor még az ablakkezelő elindulása előtt kell őket futtatni. Ez úgy oldható meg, hogy a felhasználó a saját könyvtárában lévő *.xinitrc* állományt szerkeszti. Ennek tartalma például így nézhet ki:

```
imwheel &  
xterm &  
xmodmap ~/xmodmap.hu  
exec icewm
```

Az első sorban elindítjuk az *imwheel* programot, ami görgővel ellátott egerekhez nyújt támogatást. Az *&* jel most azért kell, hogy elindulása után ne lépjen ki azonnal, hanem végig fusson a háttérben. Indítunk még egy *xterm* programot és az *xmodmap* segítségével magyar billentyűkiosztást töltünk be. Végül pedig az ablakkezelő program az utolsó sorban szerepel.

A futó programok ablakairól az *xwininfo* ad információt, főleg a méretükről és elhelyezkedésükről a képernyőn. Az *xmag* felnagyítja a képernyő egy részét, főleg nagy felbontásnál lehet hasznos. Az *xkill* pedig máris sejthető, mire jó: azt a programot, amire rákattintunk a megjelenő kurzorral, azonnal leállítja, kilövi.

Végül nézzük meg az *xhost* program alkalmazását, amivel az X szerverhez való hozzáférés szabályozható. A *+* jel (mint paraméter) után felsorolt gépeket hozzáadja, a *-* utániakat leveszi arról a listáról, amin azok szerepelnek, amiken a futó programok használhatják adataik megjelenítésére a szervert. Lásuk, mi a teendő, ha a *star.valami.hu* gépen fut az X, ez előtt ülünk, de a



1.9. ábra. Különböző terminálprogramok

*king.valami.hu* gépen akarunk futtatni egy programot, aminek grafikus kimenetét a mi gépünkön szeretnénk látni! Először is nyissunk egy *xterm* ablakot, és adjuk ki benne az *xhost +king.valami.hu* parancsot, ezzel megengedjük neki, hogy a monitorunkra írhasson. Ezután lépünk be a *king.valami.hu* gépre telnet vagy *ssh* segítségével, és ott adjuk ki a következő utasítást: *export DISPLAY="star.valami.hu:0"*. Ez arra szolgál, hogy az azon a gépen futtatott programjaink tudják, hova kell küldeni az adataikat, egész pontosan a *DISPLAY* környezeti változó által meghatározott gép első X szervertére (mivel egy gépen több is futhat, és az elsőnek 0 a sorszáma). Végül már csak egy grafikus programot kell indítani, és látható is az eredmény!

## 1.20. Csomagok kezelése

A Linux terjesztések, disztribúciók rengeteg programot tartalmaznak, amik a kernel köré épülve alkotnak egységes operációs rendszert. Ezek az alkalmazások a bináris programokkal, dokumentációval és minden egyéb szükséges alkotóelemükkel együtt érkeznek, ezért összefoglaló néven csomagoknak hívjuk őket. A csomagok egymásra is épülnek, mivel némely csomagot addig nem érdemes (vagy nem is lehet egyáltalán) használni, amíg egy másik nincs telepítve. A csomagon belül ezért nemcsak a programok alkotórészei, hanem olyan információk is megtalálhatók, amik ezeket a függőségi információkat tartalmazzák. Valamint olyan héjprogramok is, amik telepítésük/eltávolításuk előtt és után bizonyos beállításokat tudnak végezni.

Az egyes disztribúciók más csomagformátumokat használnak, de van két fő típus, ami a legtöbbször megtalálható. Az egyik a *Red Hat* cég által kifejlesztett *RPM* formátum

(*Red Hat Package Management*), ami a *SuSE*, *Caldera*, *Mandrake*, *BeroLinux* és egyéb más terjesztésekben is használatos. A másik pedig a *Debian* által használt, `.deb` ki-terjesztéséről könnyen felismerhető csomagformátum, amit többek között még a *Corel* terjesztése is használ. Nézzük meg röviden ezek kezelését!

Az *RPM* csomagokkal való munka megkönnyítésére ma már sokféle, legtöbbször grafikus program is használható, de az alapprogram `rpm` névre hallgat. Főbb üzemmódjai a telepítés (`install`), frissítés (`upgrade`), törlés (`erase`), lekérdezés (`query`), és csomagkészítés (`build`). A következőkben egy `valami.i386.rpm` nevű fájlra ke- resztül mutatjuk be ezeket.

Teljesen új csomag telepítése az `rpm -ih valami.i386.rpm` parancs kiadá- sával történik, itt a `-i` az installálásra utal, a `-h` hatására pedig kettőskeresztek je- lennek meg a telepítés során mint folyamatjelzők. Ilyenkor mindig történik függőség- ellenőrzés is, és ha valami hiányzik azok közül, amikre a program épül, az `rpm` meg- tagadja a telepítést. A `--force` kapcsolóval azonban kieroszakolhatjuk, hogy mégis másolja fel a csomagot, a `--nodeps` pedig kikapcsolja ezt az ellenőrzést (ezekhez csak végső esetben folyamodjunk). Már meglévő, de régebbi verziójú csomag frissíté- séhez az `rpm -Uvh valami.i386.rpm` használható, itt még egy `-v` paraméterrel egészítettük ki, ami bőbeszédűbb üzemmódra kapcsol. A program eltávolítása bizton- ságosan az `rpm -e valami.i386.rpm` kiadásával történik, ekkor is ütközhetünk abba az üzenetbe, hogy bizonyos programok még igényelnék ennek a jelenlétét, de a `--force` itt is végleges megoldást kínál. A csomagok lekérdezésére szolgál az `rpm -q valami` (tehát csak a neve kell!), de ez így önmagában csak arra jó, hogy megtudjuk, egyáltalán telepítve van-e. A `-q` után írt kapcsolók tovább bővítik a lehetősé- geket, például a `-qa` megjeleníti az összes telepített csomag nevét, a `-ql` kilistázza a telepített csomagban található összes fájlt, a `-qf /valahol/valami` megmondja, hogy az adott fájl melyik csomag része, a `-qi` a csomagról szóló információkat (rövid leírás, méret, verzió, stb.) adja, ha pedig mindezek mellé még a `p` paramétert is be- tesszük, akkor a még nem telepített csomagokról is kérhetünk információt. Végül, ha egy eddig csak forrásszöveggel elérhető programot szeretnénk *RPM* csomaggá alakítani, akkor először is készítenünk kell egy ún. *spec*-fájlt, az ebben foglalt utasítások alap- ján az `rpm` előbb lefordítja az alkalmazást, megnézi, milyen függőségeket kell majd előírni, és végül elkészíti a bináris csomagot, mégpedig a `cpio` közreműködésével. A *Midnight Commander* segítségével az `rpm` csomagok kezelése egyszerű, csak *En- tert* kell nyomni a csomag nevén, így belenézhetünk, elolvashatjuk az információkat, és telepíthetjük/frissíthetjük is.

A *Debian* csomagok is hasonlóan igyekeznek könnyebbé tenni a rendszergazdák életét. A `.deb` csomagok kezelésére több program is használható. Az alapvető a `dpkg`, ami parancssori kapcsolókkal vezérelhető, de létezik hozzá egy menüs elötét- program a `dselect` személyében. A `dpkg` főbb üzemmódjai: installálás a beállítá- sok elvégzésével (`-i`), kicsomagolás beállítások nélkül (`--unpack`), újrakonfigurálás (`--configure`), eltávolítás (`-r`), a rendelkezésre álló csomagok listájának frissítése (`--update-avail`). Az utolsó eset kivételével természetesen meg kell adni a telepít- endő csomag nevét is, mint paramétert. A `dselect` hét ponton keresztül vezet végig a főbb lépéseken. Elsőként az *Access* menüpontban meg kell adni, milyen módszer szerint férhetünk hozzá a csomagokhoz: *NFS* vagy *FTP*-szerverről, hajlékonylemezek- ről vagy más médiumról. Itt a leggyakrabban az *APT* (*Advanced Package Tool*) mód- szer szerinti hozzáférést választjuk. Az *Update* menüponton belül a rendelkezésre álló csomagok listáját frissíthetjük. A *Select* ponton belül választhatjuk ki a tele- pítendőket. A program figyeli a függőségeket is, tehát azokat is felkínálja telepítésre, amik kellenek a működéshez. Az *Install* választásával indul el a tulajdonkép-

peni telepítés. A `Config` a telepített, de még nem konfigurált csomagokon végzi el a beállításokat, a `Remove` pedig eltávolítja a fölöslegeseket. Végül a `Quit` léptet ki a programból.

A másik lehetőségünk az újabb fejlesztésű *APT* módszer szerinti csomagkezelés, amihez az `apt-get` parancs a legfőbb eszköz. Szintén parancssori kapcsolókkal vezéreljük az `apt-get` opciók parancs csomagnév formában. A parancsok közül a legfontosabbak:

**upgrade** – a legfrissebb verzióra való frissítés, felhasználva a `/etc/apt/sources.list` fájlban fellelhető forrásokat, amik között lehetnek Internetes szerverek is, sőt, ez az egyik legjobb lehetősége.

**install** – telepítés.

**remove** – eltávolítás.

**source** – a csomagok nemcsak bináris formában létezhetnek, hanem még forrásnyelvi állapotban is, ilyenek letöltéséhez szükséges ez az opció.

**check** – ellenőrzés.

**clean** – a letöltött csomagfájlok törlése.

Az opciók közül a `-b` szolgál arra, hogy a letöltött forrásokból kész csomagot állítsunk elő.

## 1.21. Egyéb parancsok

Az eddigiekben felsorolt parancsokon kívül vannak még olyanok, amik nem maradhatnak említés nélkül. Ide sorolhatók azok a programok, amik matematikai feladatok megoldását segítik, mint a `bc`, amivel a négy alapműveleten felül jóval bonyolultabb dolgokat is kiszámolhatunk. Indítsuk el, majd a megjelenő üres sorba gépeljük be:

```
a=4^2+3^2
print sqrt (a)
```

Az eredmény 5 lesz a *Pitagorasz-tétel* alapján. Itt a `^` a hatványozás jele, az `sqrt` függvény pedig a gyökvonást végzi. Hagyományos számológépet kapunk ezen felül az `xcalc` grafikus program személyében.

A `watch` segítségével egy program kimenetét figyelhetjük. Próbáljuk ki úgy, hogy egy könyvtárban, ahol kevés fájl van, kiadjuk a `watch ls -l` parancsot, majd egy másik terminálon belépve (célszerű mindezt X Window alatt csinálni) létrehozunk ott egy fájlt. A `watch` alapértelmezés szerint 2 másodpercenként frissíti a képernyőjét, tehát kisvártatva látjuk, hogy megjelenik az új fájl. Ha most írunk hozzá valamit, akkor a méretváltozás is meg fog jelenni hamarosan.

Multimédiás fájlok kezelésére is sok program van. Képek nézegetésére a konzolon is használható a `zgv`. Ha van hangkártyánk, akkor a `WAV` hangfájlok lejátszásához a `playwave`, a `MIDI` fájlokéhoz a `playmidi` használható, a népszerű `MP3` formátumhoz pedig az `mpg123` vagy az `mp3blaster`. Audio CD-k lejátszásához is sokféle alkalmazás található, illetve le lehet szedni róluk a zenei adatokat digitális formában, ezt hívják `grab`-belésnek, erre többek között a `cdparanoia` szolgál.

## 1.22. Zárszó

A Linux a felsoroltakon kívül még nagyon sok kisebb-nagyobb alkalmazást tartalmaz, ha mindet fel akarjuk térképezni, nagyon sok szabadidővel kell rendelkezünk. Az itt megemlített parancsok azonban remélhetőleg jó keresztmetszetét adták annak, mi mindenre és hogyan is használható a rendszer, és kiindulási pontként szolgáltak ahhoz, hogy jobban megértsük a működését, nem utolsósorban használni is tudjuk. Mint említettem az elején, itt mindegyik parancs vázlatosan, sokszor csak az említés szintjén szerepelt, ezért mindenképpen javasolt a kézikönyvoldalak és a róluk fellelhető mindenfajta információ áttanulmányozása. Ezenfelül pedig a gyakorlás, a kipróbálás vihet előre. Ezért is igyekeztem minél több helyen kipróbálendő dolgokat beiktatni, hogy legalább az első lépéseket megkönnyítsem, és mert sok leírásból hiányoznak az alkalmazási példák. Pedig ha egy-két ilyen lát az ember, könnyebben megérti a sokszor száraz, és csak a lényegre szorítkozó dokumentációkat is. Tehát abban a reményben, hogy ez a füzet valamelyest hasznos volt, biztatom mindenkit a Linux használatára, és ha elsőre nem sikerül valami, akkor se adjuk fel, kísérletezzünk, egyszercsak rá fogunk jönni a megoldásra. Végezetül álljon itt egy olyan idézet, amit választhatunk jelmondatunknak is: „Egy pingvinre gyakorlatilag lehetetlen haragosan nézni!”

# GNU Szabad Dokumentációs Licenz 1.1 verzió, 2000 március

Copyright ©2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Jelen licenz szó szerinti sokszorosítása és terjesztése bárki számára megengedett, változtatni rajta ugyanakkor nem lehet.

## 0. ELŐSZÓ

Jelen Licenz célja egy olyan kézikönyv, tankönyv, vagy effajta írott dokumentum megalkotása, mely a szó szoros értelmében „szabad”: annak érdekében, hogy mindenkinek biztosítsa a szöveg sokszorosításának és terjesztésének teljes szabadságát, módosításokkal, vagy anélkül, akár kereskedelmi, akár nem-kereskedelmi úton. Másfelől, a Licenz megőrzi a szerző, vagy kiadó munkája elismeréséhez fűződő jogát, s egyúttal mentesíti őt a mások által beiktatott módosítások következményei alól.

Jelen Licenz egyfajta „etalonnak” tekinthető, ami nem jelent mást, mint hogy a dokumentumból származtatott munkák maguk is szabad minősítést kell, hogy kapjanak. E dokumentum egyben a GNU Általános Felhasználói Licenz kiegészítőjeként is szolgál, mely egy a szabad szoftverekre vonatkozó etalon licenz.

E Licenzet a szabad szoftverek kézikönyveiben való használatra alkottuk, hiszen a szabad szoftver egyben szabad dokumentációt is igényel: egy szabad programot olyan kézikönyvvel kell ellátni, mely ugyanazon szabadságokat biztosítja, mint maga a program. Jelen Licenz, mindazonáltal, nem korlátozódik pusztán kézikönyvekre; feltételei tetszőleges tárgykörű írott dokumentumra alkalmazhatók, függetlenül attól, hogy az könyvformában valaha megjelent-e. Mindamellet e Licenzet főként olyan munkákhoz ajánljuk, melyek elsődleges célja az útmutatás, vagy a tájékoztatás.

## 1. ALKALMAZHATÓSÁG ÉS DEFINÍCIÓK

E Licenz minden olyan kézikönyvre, vagy más jellegű munkára vonatkozik, melyen megtalálható a szerzői jogtulajdonos által feltüntetett figyelmeztetés, miszerint a dokumentum terjesztése jelen Licenz feltételei alapján lehetséges. A „Dokumentum” alább bármely ilyen jellegű kézikönyvre, vagy egyéb munkára vonatkozik. A lakosság minden tagja potenciális licenztulajdonosnak tekinthető, és mindegyikük megszólítása egyaránt „ön”.



A Dokumentum „Módosított Változata” bármely olyan munkára vonatkozik, mely tartalmazza a Dokumentumot, vagy annak elemeit akár szó szerint, akár módosításokkal, és/vagy más nyelvre lefordítva.

A „Másodlagos Szakasz” egy egyedi névvel bíró függelék, esetleg a Dokumentum egy megelőző szakasza, mely kizárólag a kiadóknak, vagy az alkotóknak a Dokumentum átfogó tárgyköréhez (vagy kapcsolódó témákhoz) fűződő viszonyáról szól, és nem tartalmaz semmi olyat, ami közvetlenül ezen átfogó témakör alá eshet. (Ha például a Dokumentum részben egy matematika tankönyv, úgy a Másodlagos Szakaszban nincs lehetőség matematikai tárgyú magyarázatokra.) A fenti kapcsolat tárgya lehet a témakörrel, vagy a kapcsolódó témákkal való történelmi viszony, illetve az azokra vonatkozó jogi, kereskedelmi, filozófiai, etikai, vagy politikai felfogás.

A „Nem Változtatható Szakaszok” olyan speciális Másodlagos Szakaszok számításának, melyek illetően való meghatározását az a közlemény tartalmazza, miszerint a Dokumentum jelen Licenz hatálya alatt lett kiadva.

A „Borítószovegek” olyan rövid szövegrészek, melyek Címlap-szöveggé, illetve Hátlap-szöveggé kerülnek felsorolásra abban a közleményben, miszerint a Dokumentum jelen Licenz hatálya alatt lett kiadva.

A Dokumentum „Átlátszó” példánya olyan géppel-olvasható változatot jelöl, mely a nyilvánosság számára hozzáférhető formátumban kerül terjesztésre, továbbá melynek tartalma szokványos szövegszerkesztő-programokkal, illetve (pixelekből álló képek esetén) szokványos képmegjelenítő-programokkal, vagy (rajzok esetén) általánosan hozzáférhető rajprogramok segítségével azonnal és közvetlenül megtekinthető, vagy módosítható; továbbá olyan formátumban mely alkalmas a szövegszerkesztőkbe való bevitelre, vagy a szövegszerkesztők által kezelt formátumokba való automatikus átalakításra. Egy olyan, egyébként Átlátszó formátumban készült példány, melynek markujja úgy lett kialakítva, hogy megakadályozza, vagy eltántorítsa az olvasókat minden további módosítástól, nem tekinthető Átlátszónak. A nem „Átlátszó” példányok az „Átlátszatlan” megnevezést kapják.

Az Átlátszóság kritériumainak megfelelő formátumok között megtalálható például a markup nélküli egyszerű ASCII, a Texinfo beviteli formátum, a  $\text{\LaTeX}$  beviteli formátum, az SGML vagy az XML egy általánosan hozzáférhető DTD használatával, és a standardnak megfelelő, emberi módosításra tervezett egyszerű HTML. Az Átlátszatlan formátumok közé sorolható a PostScript, a PDF, a szabadalmaztatott és csak fizetős szövegszerkesztőkkel olvasható formátumok, az olyan SGML vagy XML, melyhez a szükséges DTD és/vagy egyéb feldolgozó eszközök nem általánosan hozzáférhetők, és az olyan gépileg-generált HTML formátum, melyet egyes szövegszerkesztők hoznak létre, kizárólag kiviteli célra.

Egy nyomtatott könyv esetében a „Címlap” magát a címlapot, illetve bármely azt kiegészítő további oldalt jelöl, amely a jelen Licenzben definiált címlap-tartalmak közzétételéhez szükséges. Az olyan formátumú munkáknál, melyek nem rendelkeznek effajta címlappal, a „Címlap” a munka címéhez legközelebb eső, ám a szöveg törzsét megelőző szövegrészeket jelöli.

## 2. SZÓ SZERINTI SOKSZOROSÍTÁS

Önnek lehetősége van a dokumentum kereskedelmi, vagy nem-kereskedelmi jellegű sokszorozására és terjesztésére, bármely médiumon keresztül, feltéve, hogy jelen Licenz, a szerzői jogi figyelmeztetés, továbbá a Dokumentumot jelen Licenz hatálya

alá rendelő közlemény minden példányban egyaránt megjelenik, és hogy e feltételeken kívül semmi mást nem tesz hozzá a szöveghez. Nem alkothat olyan technikai korlátokat, melyek megakadályozhatják, vagy szabályozhatják az ön által terjesztett példányok elolvasását, vagy sokszorosítását. Mindazonáltal elfogadhat bizonyos összeget a másolatok fejében. Amennyiben az ön által terjesztett példányok száma meghalad egy bizonyos mennyiséget, úgy a 3. szakasz feltételeinek is eleget kell tennie.

A fenti kritériumok alapján kölcsönbe adhat egyes példányokat, de akár nyilvánosan is közzéteheti a szöveget.

### **3. SOKSZOROST ÉS NAGYOBB MENNYISÉGBEN**

Amennyiben 100-nál több nyomtatott változatot tesz közzé a Dokumentumból, és annak License feltételül szabja a Borítószovegek meglétét, úgy minden egyes példányt köteles ellátni olyan borítólappal, melyeken a következő Borítószovegek tisztán és olvashatóan fel vannak tüntetve: Címlap-szövegek a címlapon, illetve Hátlap-szövegek a hátlayan. Mindkét borítólapra egyértelműen és olvashatóan rá kell vezetnie a kiadó, vagyis jelen esetben az ön nevét. A címlapon a Dokumentum teljes címének jól láthatóan, továbbá minden egyes szónak azonos szedésben kell megjelennie. Ezen felül, belátása szerint, további részleteket is hozzáadhat a borítólapokhoz. Amennyiben az esetleges módosítások kizárólag a borítólapokat érintik, és feltéve, hogy a Dokumentum címe változatlan marad, továbbá a borítólapok megfelelnek minden egyéb követelménynek, úgy a sokszorosítás ettől eltekintve szó szerinti reprodukciónak minősül.

Abban az esetben, ha a borítólapok bármelyikén megkövetelt szövegrészek túl hosszúnak bizonyulnának az olvasható közzétételhez, úgy csak az elsőként felsoroltakat kell feltüntetnie (amennyi józan belátás szerint elfér) a tényleges borítón, a továbbiak pedig átkerülhetnek a következő oldalakra.

Amennyiben 100-nál több Átlátszatlan példányt tesz közzé, vagy terjeszt a Dokumentumból, úgy köteles vagy egy géppel-olvasható Átlátszó példányt mellékelni minden egyes Átlátszatlan példányhoz, vagy leírni minden egyes Átlátszatlan példányban egy a módosítatlan Átlátszó példányt tartalmazó nyilvános hozzáférésű számítógép-hálózat elérhetőségét, ahonnan bárki, anonim módon, térítésmentesen letöltheti azt, egy közismert hálózati protokoll használatával. Ha az utóbbi lehetőséget választja, köteles gondoskodni arról, hogy attól a naptól kezdve, amikor az utolsó Átlátszatlan példány is terjesztésre került (akár közvetlenül ön által, akár kiskereskedelmi forgalomban), a fenti helyen közzétett Átlátszó példány még legalább egy évig hozzáférhető legyen a felhasználók számára.

Megkérjük, ámde nem kötelezzük önt arra, hogy minden esetben, amikor nagyobb példányszámú terjesztésbe kezd, már jóval ezt megelőzően lépjen kapcsolatba a Dokumentum szerzőivel, annak érdekében, hogy megkaphassa tőlük a Dokumentum esetleges felújított változatát.

### **4. MÓDOSÍT ÉS**

Önnek lehetősége van a Dokumentum Módosított Változatának sokszorosítására és terjesztésére a 2. és 3. szakaszok fenti rendelkezései alapján, feltéve, hogy a Módosított

Változatot kizárólag jelen Licenz feltételeivel összhangban teszi közzé, ahol a Módosított Változat a Dokumentum szerepét tölti be, ezáltal lehetőséget biztosítva annak terjesztésére és módosítására bárkinek, aki csak hozzájut egy példányához. Mindezen felül, a Módosított Változat az alábbi követelményeknek is meg kell, hogy feleljen:

- A Címlapon (és ha van, a borítókön) tüntessen fel egy a Dokumentumétól, illetve bármely korábbi változatától eltérő címet (melyeknek, ha vannak, a Dokumentum Előzmények szakaszában kell szerepelniük). Egy korábbi változat címét csak akkor használhatja, ha annak szerzője engedélyezte azt.
- A Címlapon szerzőkként sorolja fel a Módosított Változatban elvégzett változtatásokért felelős személyeket, vagy entitásokat, továbbá a Dokumentum fő szerzői közül legkevesebb ötöt (vagy mindet, ha nincsenek öten).
- A Címlapon a Módosított Változat közzétételéért felelős személyt tüntesse fel kiadóként.
- A Dokumentum összes szerzői jogi figyelmeztetését hagyja érintetlenül.
- Saját módosításaira vonatkozóan is tegyen közzé egy szerzői jogi megjegyzést, a többi ilyen jellegű figyelmeztetés mellett.
- Rögtön a szerzői jogi figyelmeztetéseket követően tüntessen fel egy közleményt, az alábbi Függelék mintájára, melyben engedélyezi a Módosított Változat felhasználását jelen Licenz feltételei alapján.
- A fenti közleményben hagyja érintetlenül a Nem Változtatható Szakaszok és a szükséges Borítósövegek jelen Dokumentum licenszében előírt teljes listáját.
- Mellékelje jelen Licenz egy eredeti példányát.
- Az „Előzmények” szakaszt, illetve annak címét szintén hagyja érintetlenül, emellett adjon hozzá egy új elemet, amely minimálisan tartalmazza a Módosított Változat címét, kiadási évét, továbbá az új szerzők, illetve a kiadó nevét, a Címlapon láthatókhoz hasonlóan. Amennyiben a Dokumentum nem tartalmaz semmiféle „Előzmények” elnevezésű szakaszt, úgy hozzon létre egyet, mely tartalmazza a Dokumentum címét, kiadási évét, továbbá a szerzők, illetve a kiadó nevét, a Címlapon láthatókhoz hasonlóan; majd ezt követően adjon hozzá egy új, a Módosított Változatra vonatkozó elemet, a fentiekkel összhangban.
- Ne tegyen változtatásokat a Dokumentumban megadott Átlátszó példány nyilvános hálózati elérhetőségét (ha van ilyen) illetően, vagy hasonlóképp, a Dokumentum alapjául szolgáló korábbi változatok hálózati helyére vonatkozóan. Ezek az „Előzmények” szakaszban is szerepelhetnek. Csak abban az esetben hagyhatja el egyes korábbi változatok hálózati elérhetőségét, ha azok legkevesebb négy évvel a Dokumentum előtt készültek, vagy ha maga az alkotó engedélyezi azt.
- Bármely „Köszönetnyilvánítás”, vagy „Ajánlások” szakasz címét hagyja érintetlenül, továbbá gondoskodjon arról, hogy azok tartalma és hangvétele az egyes hozzájárulókat, és/vagy az ajánlásokat illetően változatlan maradjon.
- A Dokumentum összes Nem Változtatható Szakaszát hagyja érintetlenül, úgy címüket, mint tartalmukat illetően. A szakaszok számozása, vagy bármely azzal egyenértékű jelölés nem tartozik a szakaszcímek közé.

- Töröljön minden „Jóváhagyás” elnevezésű szakaszt. Effajta szakaszok nem képezhetik részét a Módosított Változatnak.
- Ne nevezzen át semmilyen létező szakaszt „Jóváhagyás”-ra, vagy olyasmire, mely címében a Nem Változtatható Szakaszokkal ütközhet.

Ha a Módosított Változat új megelőző szakaszokat tartalmaz, vagy olyan függelék-eket, melyek Másodlagos Szakasznak minősülnek, ám nem tartalmazznak a Dokumentumból származó anyagot, abban az esetben, belátása szerint, e szakaszok némelyikét, vagy akár az összeset nem változtathatóként sorolhatja be. Ehhez nem kell mást tennie, mint felsorolni a szóban forgó címeket a Módosított Változat licenszének Nem Változtatható Szakaszok listájában. E címeknek határozottan el kell különülnie minden egyéb szakaszcímtől.

„Jóváhagyás” elnevezésű szakaszt csak akkor adhat a Dokumentumhoz, ha az kizárólag a Módosított Változatra utaló megjegyzéseket tartalmaz – például mások recenzióira vonatkozóan, vagy hogy egy szervezet a szöveget egy standard mérvadó definíciójaként ismerte el.

Címlap-szöveg gyanánt egy legfeljebb öt szóból álló szövegrészt adhat meg, a Hátlap-szöveg esetén pedig 25 szót fűzhet a Módosított Változat Borítószövegeinek végéhez. Bármely entitás csak és kizárólag egy Címlap- és egy Hátlap-szövegrészt adhat (akár közvetítőn keresztül) a Dokumentumhoz. Ha a dokumentum már eleve rendelkezik Borítószöveggel, akár azért, mert azt korábban ön adta hozzá, vagy mert valaki más önön keresztül gondoskodott erről, abban az esetben nincs lehetőség újabb Borítószöveg hozzáadására; a régít mindazonáltal lecserélheti, abban az esetben, ha annak kiadója egyértelműen engedélyezi azt.

A Dokumentum szerzője/i és kiadója/i jelen Licensz alapján nem teszik lehetővé nevük nyilvános felhasználását egyetlen Módosított Változat támogatása, vagy támogatottsága érdekében sem.

## 5. KOMBINÁLT DOKUMENTUMOK

Önnek lehetősége van a Dokumentum egyéb, e Licensz hatálya alatt kiadott dokumentumokkal való kombinálására a 4. szakasz módosított változatokra vonatkozó rendelkezései alapján, feltéve, hogy a kombináció módosítás nélkül tartalmazza az eredeti dokumentumok összes Nem Változtatható Szakaszát, és hogy azok mind Nem Változtatható Szakaszként kerülnek felsorolásra a kombinált munka licenszében.

A kombinált munkának jelen Licensz mindössze egy példányát kell tartalmaznia, az egymással átfedésben lévő Nem Változtatható Szakaszok pedig kiválthatók egy összegzett példánnyal. Amennyiben több Nem Változtatható Szakasz szerepelne ugyanazon címmel, ám eltérő tartalommal, úgy alakítsa át minden egyes szakasz címét olyan módon, hogy mögéírja zárójelben az eredeti szerző és kiadó nevét (ha ismeri), vagy egy egyedi sorszámot. Ha szükséges, a Nem Változtatható Szakaszok címeivel is végezze el a fenti módosításokat a kombinált munka licenszében.

A kombinált munkában az eredeti dokumentumok összes „Előzmények” elnevezésű szakaszát össze kell olvasztania, miáltal egy összefüggő „Előzmények” szakasz jön létre; hasonlóképp kell eljárnia a „Köszönetnyilvánítás”, illetve az „Ajánlások” szakaszok tekintetében. Ugyanakkor minden „Jóváhagyás” elnevezésű szakaszt törölnie kell.

## 6. DOKUMENTUMGYŰJTEMÉNYEK

Önnek lehetősége van a Dokumentumból, illetve bármely egyéb, e Licenz hatálya alatt kiadott dokumentumból gyűjteményt létrehozni, és az egyes dokumentumokban található licenzeket egyetlen példánnyal kiváltani, feltéve, hogy a gyűjteményben szereplő összes dokumentum esetén minden más tekintetben követi jelen Licenz feltételeit, azok szó szerinti sokszorosítására vonatkozóan.

Tetszése szerint ki is emelhet egy meghatározott dokumentumot a gyűjteményből, továbbá terjesztheti azt jelen Licenz feltételei alapján, feltéve, hogy a szóban forgó dokumentumhoz mellékeli e Licenz egy példányát, és minden egyéb tekintetben betartja jelen Licenz előírásait a dokumentum szó szerinti sokszorosítására vonatkozóan.

## 7. ÖSSZEFŰZÉS FÜGGETLEN MUNKÁKKAL

A Dokumentum és annak származékainak különálló, vagy független dokumentumokkal, illetve munkákkal való összefűzése egy közös tárolási, vagy terjesztési egységen, egészében nem tekinthető a Dokumentum Módosított Változatának, feltéve, hogy az összefűzés nem lesz szerzői jogvédett. Az effajta összefűzés eredményeként „összegzés” jön létre, ám jelen Licenz nem érvényes az abban a Dokumentummal együtt szereplő önálló munkákra, hacsak azok nem a Dokumentum származékai.

Amennyiben a 3. szakasz Borítószövegekre vonatkozó rendelkezései alkalmazhatók a Dokumentum e példányaira, és a Dokumentum a teljes összegzésnek kevesebb, mint egynegyedét teszi ki, úgy a Dokumentum Borítószövegeit olyan módon is el lehet helyezni, hogy azok csak magát a Dokumentumot fogják át. Minden más esetben a teljes összegzés borítólapjain kell feltüntetni a fenti szövegeket.

## 8. FORDÍTÁS

A fordítás egyfajta módosításnak tekinthető, így hát a Dokumentum lefordított példányai a 4. szakasz rendelkezései alapján terjeszthetők. A Nem Változtatható Szakaszok lefordítása külön engedélyt igényel a szerzői jogtulajdonostól, mindazonáltal közzéteheti a lefordított változatokat is abban az esetben, ha az eredeti Nem Változtatható Szakaszokat is belefoglalja a munkába. E Licenz lefordítására ugyanezek a feltételek érvényesek, vagyis a lefordított változat csak akkor jelenhet meg, ha mellette ott van az eredeti, angol nyelvű Licenz szövege is. Amennyiben eltérés mutatkozna az eredeti változat, illetve a fordítás között, úgy a Licenz angol nyelvű eredetije tekintendő mérvadónak.

## 9. MEGSZŰNÉS

A jelen Licenzben egyértelműen kijelölt kereteken kívül tilos a Dokumentum bármilyen sokszorosítása, módosítása, allicenszelése, vagy terjesztése. Minden ezzel szembeni sokszorosítási, módosítási, allicenszelési, vagy terjesztési kísérlet a jelen Licenzben meghatározott jogok automatikus megszűnését vonja maga után. Azok a fe-

lek, ugyanakkor, akik önön keresztül jutottak másolathoz, vagy jogosultságokhoz, nem veszítik el azokat, amíg maradéktalanul betartják e Licenz előírásait.

## 10. JELEN LICENZ JÖVŐBENI JAVÍTÁSAI

Megtörténhet, hogy a Szabad Szoftver Alapítvány időről időre felülvizsgálja és/vagy új verziókat bocsát ki a GNU Szabad Dokumentációs Licenzből. E verziók szellemisége hasonló lesz jelen változathoz, ám részleteikben eltérhetnek, új problémák, új aggályok felmerülése okán. Vö.: <http://www.gnu.org/copyleft/>

A Licenz minden változata egyedi verziószámmal van ellátva. Ha a Dokumentum jelen Licenz egy konkrét, számozott verziójára, „vagy bármely újabb verzióra” hivatkozik, úgy önnök a szóban forgó változat, vagy bármely újabb a Szabad Szoftver Alapítvány által (nem vázlatként) publikált verzió feltételeinek követésére lehetősége van. Ha a Dokumentum nem ad meg semmilyen verziószámot, úgy bármely a Szabad Szoftver Alapítvány által valaha (nem vázlatként) publikált változat megfelel.

## FÜGGELÉK: A Licenz alkalmazása saját dokumentumaira

Ha e Licenzet egy ön által írt dokumentumban kívánja használni, akkor mellékelje hozzá a Licenz egy példányát, továbbá vezesse rá az alábbi szerzői jogi és licenz közleményeket, rögtön a címlapot követően:

Copyright © ÉV AZ ÖN NEVE.

E közlemény felhatalmazást ad önnek jelen dokumentum sokszorosítására, terjesztésére és/vagy módosítására a Szabad Szoftver Alapítvány által kiadott GNU Szabad Dokumentációs Licenz 1.1-es, vagy bármely azt követő verziójának feltételei alapján. A Nem Változtatható Szakaszok neve SOROLJA FEL A CÍMLAP-szövegek neve LISTA, a Hátlap-szövegek neve pedig LISTA. E licenz egy példányát a „GNU Szabad Dokumentációs Licenz” elnevezésű szakasz alatt találja.

Ha a szövegben nincsenek Nem Változtatható Szakaszok, úgy írjon „nincs Nem Változtatható Szakasz”-t, helyett, hogy egyenként felsorolná azokat. Ha nincsenek Címlap-szövegek, akkor írjon „nincs Címlap-szöveg”-et, helyett, hogy „a Címlap-szövegek neve LISTA”, és hasonlóképp járjon el a Hátlap-szövegek esetében is.

Amennyiben a dokumentum haladó programkód-példákat is tartalmaz, úgy azt javasoljuk, hogy e példákat egy választása szerinti szabad szoftver licenz alatt közölje – mint például a GNU Általános Felhasználói Licenz –, hogy lehetővé tegye a kódok szabad szoftverekben való alkalmazását.

# Hátlapszöveg

Ezen dokumentum eredetije készült 2001-2002-ben a *Linux-Felhasználók Magyarországi Egyesülete* gondozásában a *MEH IKB* pénzügyi támogatásával. A dokumentum szabadon terjeszthető és másolható a *GNU Szabad Dokumentációs Licenz* feltételei alapján.