Linux felhasználói ismeretek (0.12-6)

Pipás

1998. július 13.

Megjegyzés

Ez a dokumentum egy könyv kezdetleges formában. Lehetnek benne olyan részek, amelyek már most is hasznosak, de inkább csak arra jó, hogy felcsigázza az érdeklődést – aztán kielégítetlenül hagyja az olvasót. Igen sok stilisztikai és helyesírási hiba van benne, mivel vannak olyan részek, amelyet még egyszer sem olvastam el, nem javítottam ki. Legfőképpen azért érhető el, hogy a nagyközönség eldöntse milyen fejezetekkel, programokkal és információkkal bővüljön ki.

Kérek tehát mindenkit – akinek kedve és energiája van –, hogy írja meg, milyen részeket látna szívesen egy ilyen könyvben. A cím is jelzi, hogy ez egy „felhasználói" kézikönyv lesz, ezért rendszeradminisztrátori feladatok leírása nem lesz benne!

Szívesen veszem a hibákra való figyelmeztetést is, ha azok nem a helyesírási és fogalmazásbeli hiányosságokra mutatnak rá. Ezeket magam is megtalálom :).

pipas@ajk.jpte.hu

Kérdések

- Milyen ingyenes program van amely magyar elválasztást képes csinálni.
- Hogyan lehet rávenni a hion -t, hogy együttműködjön a LyX -el.
- Hogyan lehet LyX -el (avagy LaTeX -el) a képeket körülfolyatni szöveggel.
- Milyen ingyenes táblázatkezelő van Linuxra X Window alá.
- Miért nem pakol néhány sort a LaTeX bizonyos lapokra, ahová még férne. Ahol kép van, ott iszonyatos méretű üres helyek bírnak lenni....

Tartalomjegyzék

Táblázatok jegyzéke

Fejezet 1

Az alapok

1.1 Információforrások

E fejezetben azokat az információforrásokat vesszük sorra, amelyek a felhasználó rendelkezésére állnak a LINUXról.

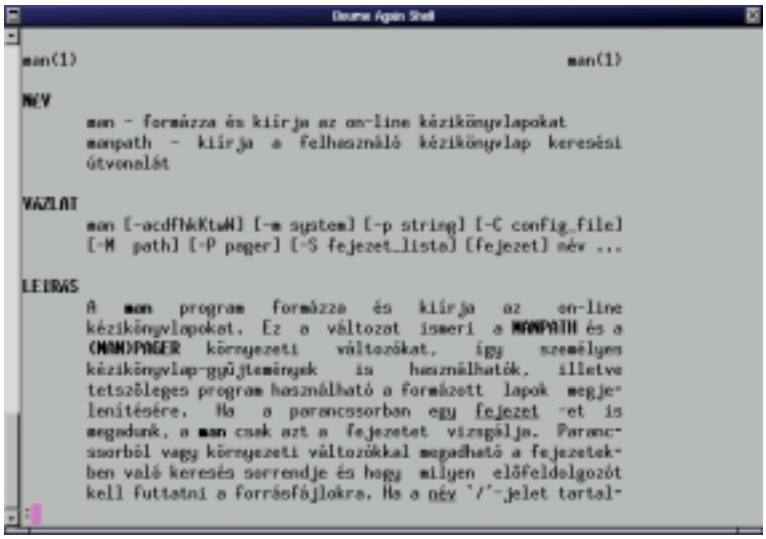
1.1.1 A man

A UNIX rendszerek – így a LINUX is – rendelkeznek egy beépített kézikönyvvel, amely parancssorból a `man` (*manual*, kézikönyv) begépelésével érhető el. Ha kíváncsiak vagyunk valamelyik parancsra vagy programra, a man után csak meg kell adnunk a nevét. Ha pl. magáról a `man` parancsról akarjuk megnézni a kézikönyvben található információkat, akkor a következő módon kell eljárnunk.

```
[root@mad /root]# man man
```

A `man` által bemutatott kézikönyvoldalakat (1.1 ábra) a kurzormozgató gombokkal fel- és lefelé görgethetjük, valamint a `q` billentyűvel kiléphetünk.

A kézikönyv fejezetekre van osztva s a `man` -nak megmondhatjuk, hogy mely fejezetekben keressen. Ez különösen akkor fontos, ha az adott parancsról több fejezetben is található információ:



ábra 1.1: A man

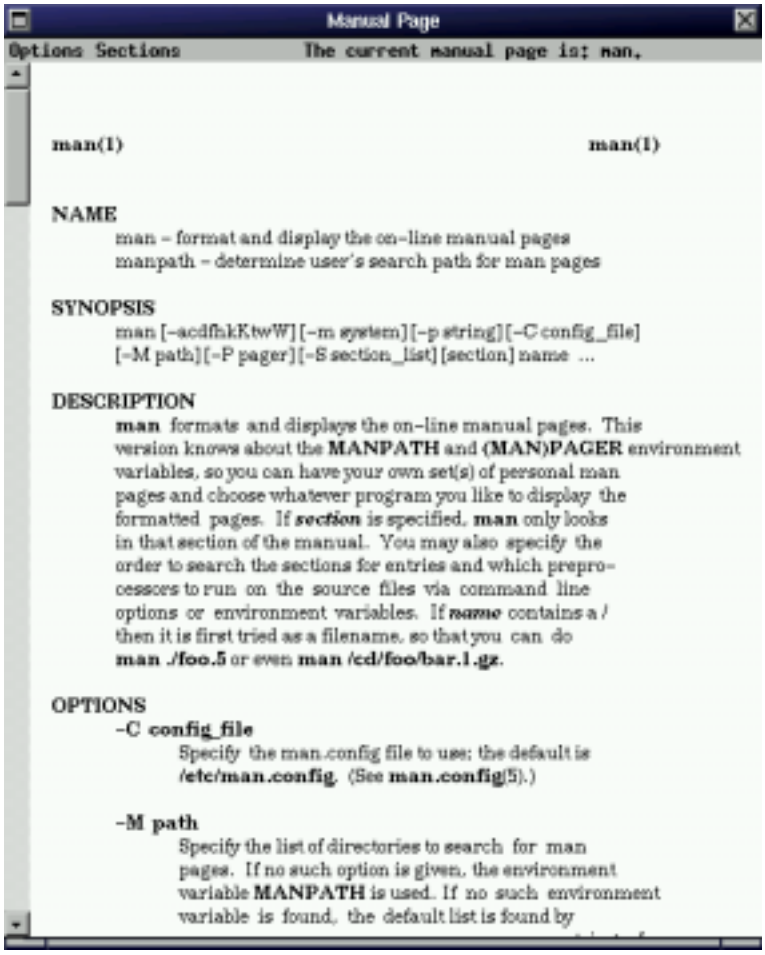
```
[root@mad /root]# man 1 man
```

Ha nem ismerjük a parancsot, amelyről leírást szeretnénk szerezni, akkor lehetőségünk van a keresésre is. A -K opció hatására a man végignézi a teljes kézikönyvet és a szövegben keres. A találatokat rendre felsorolja, választhatunk, hogy megnézzük az adott kézikönyvoldalt (y), kihagyjuk és tovább keresünk (n) vagy kilépünk a keresésből (q).

```
[root@mad /root]# man -K internet
/usr/man/mann/TclX.n? [ynq]
```

Grafikus felület – az X WINDOW – alatt rendelkezésünkre áll az xman, amely szebb megjelenést és menüs kezelőfelületet biztosít, ahogy ezt az 1.2 ábrán látjuk.

A man oldalakat POSTSCRIPT formában ki is nyerhetjük nyomtatás céljára, ehhez azonban ismernünk kell a kézikönyvoldal pontos helyét. A kézikönyv általában a /usr/man, /usr/local/man könyvtárakban található, alkönyvtárakban, ahol minden kézikönyvoldal számmal végződő



ábra 1.2: Az xman

állomány, s a szám a fejezetnek megfelelő. Ha megtaláltuk a számunkra érdekes állományt, akkor a `groff` szövegformázót használhatjuk arra, hogy azt POSTSCRIPT formátumban alakítsuk.

```
[root@mad /root]# groff -t -e -mandoc -Tps /usr/man
/hu/man1/split.1.hu >kimenet.ps
```

Az így elkészített állományt a `ghostview` programmal megnézhetjük vagy az `lpr` paranccsal kinyomtathatjuk.

A `groff` képes más formátumban is elkészíteni a kézikönyvet, a `-T` opció után megadható a `ps` helyett `dvi` (*device independent format*, eszközfüggetlen formátum), `ascii` (*American Standard Code for Information Interchange*, szabványos amerikai szöveges adatformátum), `lj4` (*LaserJet 4 Printer Control Language*, A HP nyomtatók által használt formátum) is. Az állományformátumokról részletesebben olvashatunk a **??** fejezetben.

1.1.2 Dokumentáció

A Linux alatt a programcsomagok dokumentációjának szokásos helye a /usr/doc könyvtár. Ha valamely programcsomagról szeretnénk többet megtudni, akkor itt érdemes körülnézni.

A dokumentációk általában a következő állományformátumban találhatóak meg:

`txt, ascii` Egyszerű szöveges állományok. Igen sokféleképpen lehet őket olvasni, legegyszerűbben talán a Midnight Commander (3.1 fejezet) segítségével az F3 billentyűvel.

`gz, tgz, tar.gz` A gzip nevű tömörítőprogrammal archivált állományok szokásos végződése. A már említett Midnight Commander képes ezen állományok tartalmát megmutatni, az Enter billentyű lenyomásának hatására.

`ps` Postscript állományok, amelyek tipográfiailag tördelt szöveget – esetleg képekkel – tartalmaznak. Megnézni őket a ghostview programmal lehet.

`dvi` A Postscriphez hasonló állományok, megtekintésükre az xdvi program szolgál.

`html` A WWW hipertext rendszerének formátumát használó állományok. Bármely böngészővel megnézhetőek, pl. használhatjuk a netscape nevű programot.

Általában elmondható, hogy a Midnight Commander képes megmutatni a /usr/doc könyvtárban található állományokat, de ehhez érdemes grafikus felületen – X Window alatt – elindítani. Erre azért van szükség, mert a Midnight Commander egyes állománytípusok megmutatásához grafikus felületen használható programokat indít el.

1.1.3 Interneten fellelhető dokumentációk

A LINUX az Interneten világszerte megtalálható. A LINUX fejlesztése az Interneten spontán alakult csoportok által történik, ezért nem csak a felhasználóknak szóló dokumentációt találhatjuk meg itt, hanem a fejlesztés során használt információkat is. Ezek a dokumentumok kereskedelmi programok esetében általában a fejlesztő vállalat ipari titkait képezik, a felhasználók számára hozzáférhetetlenek.

Az Interneten egyre több dokumentáció található a LINUXhoz magyar nyelven is, bár vitathatalan, hogy az angolul értők igen nagy előnyben vannak.

Az Interneten való keresésről és a Netscape böngésző használatáról a fejezetben olvashatunk.

Fejezet 2

Általános ismeretek

2.1 Bekapcsolás és kikapcsolás

A LINUX nagygépes múlttal rendelkezik, ezért a kikapcsolása és az indítása nem egyszerű feladat. Általában elmondhatjuk, hogy a nagygépes rendszerek indítása és leállítása mindig az operátor vagy rendszergazda feladata, vagyis szakembert igényel. Mindazonáltal a LINUX személyi számítógépeken is futtatható, ezért előfordulhat, hogy a felhasználónak segítség nélkül kell azt indítania és kikapcsolnia, ezért szükséges néhány szót szólnunk a témáról.

A gép bekapcsolásakor valamilyen program jelentkezik be, amely a rendszer indításának első fázisát felügyeli. Ez az indítóprogram (*boot manager*, indításvezérlő) lehet a Linux részét képező `lilo` (LINUX *loader*, LINUX betöltő), de – mivel a LINUX barátságosan megfér más operációs rendszerek mellett – más program is[1].

A `lilo` kezelése igen egyszerű. Amikor a gép elvégezte a bekapcsoláskor esedékes önellenőrzést, a `lilo` bejelentkezik a képernyőre és arra vár, hogy a felhasználó, megjelölje az indítani kívánt operációs rendszert:

```
lilo:
```

[1] Más operációs rendszerek betöltőprogramjairól – amelyek képesek a Linux indítására is – a gyártó által biztosított kézikönyvben találhatunk információkat.

Ilyenkor a következő választási lehetőségeink vannak:

- Megvárjuk, hogy a lilo elindítsa az alapértelmezés szerinti operációs rendszert – amelyet a rendszergazda beállított. Természetesen ekkor jó ha tudjuk, hogy mennyi időt kell várnunk és milyen operációs rendszerre számíthatunk – vagyis a gép üzembe helyezésekor milyen értékeket állított be a rendszergazda.
- Leütjük az Enter billentyűt és ezzel kérjük az alapértelmezés szerinti operációs rendszer azonnali indítását.
- A Tab billentyűvel kérjük az indítható operációs rendszerek listáját és kiválasztjuk a nekünk megfelelőt. A választott rövid nevet begépeljük és az Enter billentyúvel indítjuka rendszerbetöltést.

fontos megjegyeznünk, hogy a rendszer indításakor az operációs rendszer számára különleges jelentőségű parancsokat is átadhatunk, amelyekkel általában a biztonsági rendszer is megkerülhető. Hacsak a rendszer telepítésekor nem hoztak üzembe különleges biztonsági rendszereket, az indításkor a még fel nem épített jelszavas végelmi rendszer megkerülhető. Megnyugtató, hogy a rendszer indításába csak az tud beavatkozni, aki fizikailag hozzáfér a számítógéphez, vagyis az elzárt gép biztonságosan képes tárolni adatainkat.

A `lilo` mindezek után megkezdi a LINUX indítását, a vezérlést az operációs rendszernek adja át, az pedig bonyolult feladatsort elvégezve üzemkész állapotba hozza a számítógépet. Mindeközben lépésről lépésre tájékoztat a képernyőn az egyes részfeladatok elvégzéséről. Ezen részfeladatok egyik legfontosabbika a gépbe épített háttértárolókon található adatok rendszerbe illesztése. Az adatok beillesztése automatikus, de sokszor megzavarja a felhasználót, aki pánikba esve vagy elkeseredésében a legrosszabbat teszi amit el lehet képzelni: kikapcsolja a gépet.

Tudnunk kell, hogy az adatrendszer beillesztése általában néhány másodperces folyamat – mivel a LINUX nem tartja szükségesnek a tüzetes ellenőrzést – néha azonban viszonylag sokáig tart. A hosszabb időt igénybe vevő alapos ellenőrzés a következő esetekben válik szükségessé:

- A legutóbbi kikapcsolást nem előzte meg szabályos leállítás.
- A legutóbbi használat során a Linux logikai hibákat észlelt az adatok tárolását kísérő nyilvántartási rendszerben. Az ilyen hibák következhetnek a gép fizikai meghibásodásából vagy a nem

megfelelően végzett leállításokból, mindenesetre alapos ellenőrzést igényelnek.

- Az indítások és leállítások száma elérte az ellenőrzésekre előírt maximális számot. Ilyenkor óvatosáágból végez a Linux alaposabb ellenőrzést.
- A legutóbbi ellenőrzés a gép beépített órája szerint igen régen volt, ezért szükséges a megelőző céllal végzett ellenőrzés.

Bármely ok miatt is következzék be az ellenőrzés, azt türelmes végig kell várnunk, a gépet nem szabad kikapcsolnunk.

A rendszer indítása után a felhasználó azonosíthatja magát – 2.2 fejezet – és a munkát megkezdheti. A munka befejezése után szükséges lehet a kikapcsolás, a gép leállítása, amelyet körültekintően kell elvégeznünk.

Mindenek előtt az adatok mentéséről kell gondoskodnunk. Ha bármely olyan programot használtunk, amellyel adatokat hoztunk létre – szöveget, képet vagy pl. zenét – akkor ezeket az adatokat mentenünk kell, ellenkező esetben a leállításkor valószínűleg megsemmisülnek. Az egyes programokban használatos mentési – vagy automatikus mentési – és kilépési eljárásokról a program leírásánál ejtünk szót. Nem kell minden programból kilépnünk, a gép kikapcsolása előtt, általában csak azokból amelyekkel adatokat hozunk létre, vagy adatokat módosítunk, az óvatosság azonban nem árthat.

Amennyiben a számítógépeket általában távolról is használni lehet. A személyi számítógépek felépítése azt sugallja, hogy a gépet csak az használhatja, aki előtte ül és hozzáfér a billentyűzethez, látja a képernyőn megjelenő betűket – ez azonban a Linux esetében nem így van. Amennyiben a gépet üzembe helyező személy nem kérte a távoli szolgáltatások tiltását – s a számítógép működőképes hálózathoz van kapcsolva – a Linux távolról is elfogad feladatokat nyilvántartott felhasználóitól. *Ha tehát a gépet ki kívánjuk kapcsolni, előtte meg kell győződnünk arról, hogy ez mások munkáját nem szakítja meg*. Mivel ezt meglehetősen bonyolult megállapítani, a szakember általában korlátozó intézkedéseket kell, hogy bevezessen a következő módon.

A számítógépeket két csoportra osztja: az egyik csoportba azok a gépek kerülnek amelyeket bármely felhasználó kikapcsolhat, a másikba pedig azok, amelyek el vannak zárva – kikapcsolni őket így csak az arra

feljogosított személy képes. Az első csoportot alkotó gépeket a felhasználók bármikor kikapcsolhatják ezért távolról használni őket nem biztonságos. A rendszer gazdája ezért a távoli hozzáférést tiltja vagy figyelmeztetéssel ellenjavallja. Ezeket a gépeket kliens (*client*) gépeknek vagy munkaállomásoknak (*workstation*) nevezzük. A távolról használható gépeket elzárják és a leállításukkal képzett személyt bíznak meg. Az ilyen gazda (*host*) vagy szolgáltató (*server*) gépeket addig tartják bekapcsolva amíg a helyi viszonyok szükségessé teszik, akkor is, ha azokat éppen nem használja senki. Ez igen sokszor folyamatos működést jelent a nap 24 órájában.

Ha a gépet mindezen ismeretek tudtával ki akarjuk kapcsolni, akkor azt előtte le kell álítanunk – ellenkező esetben a tárolt adatok megsérülhetnek, megsemmisülhetnek. A leállítást a rendszergazda végezheti a megfelelő parancs segítségével, ez azonban a felhasználók számára tilos, mivel távoli gépről is használható. A gép leállítására a különleges jogokkal nem rendelkező felhasználó a Ctrl+Alt+Del billentyűket használhatja. A véletlen érintés elkerülése érdekében e gombokat egyszerre kell megnyomni. A Ctrl+Alt+Del billentyűk egyszerre történő megnyomása általában csak akkor hatásos, amikor karakteres – csak betűket és írásjeleket tartalmazó – üzzemódban van a gép. Ezt a Ctrl+Alt+F1 gombokat egyszerre megnyomva érhetjük el[2].

A Ctrl+Alt+Del gombok megnyomása után meg kell várnunk, hogy a gép az adminisztratív feladatait elvégezze és megkezdje az újraindulást. Akkor kapcsolhatjuk ki tehát a gépet, amikor a bekapcsoláskor már megtapasztalt képernyőt látjuk. Amikor a gép a memóriát kezdi ellenőrizni vagy a `lilo` parancskérő jelét írta a képernyőre, a gépet kikapcsolhatjuk.

2.2 A felhasználó azonosítása

A LINUX többfelhasználós operációs rendszer, ezért a munka megkezdése előtt a felhasználónak mindig igazolnia kell magát. Ez a felhasználói név (*login name*) és a jelszó (*password*) begépelésével történik. A jelszó nem jelenik meg e képernyőn – biztonsági okokból –, így „vakon” kell begépelnünk. Amennyiben jelszavunkat hibásan adtuk meg belépéskor,

[2] A Ctrl+Alt+F1 mindenképpen karakteres üzemmódba kapcsol, vagyis minden veszély nélkül megnyomhatjuk.

a LINUX elutasítja azt és nem enged dolgozni. Újra a felhasználói nevet és a jelszót kéri.

A UNIX rendszerek mindig megkülönböztetetten kezelik a rendszergazdát, amelynek neve `root`. A rendszergazda teljhatalommal rendelkezik a többi felhasználó felett. Bárkinek a fájljaihoz hozzáfér, felhasználót törölhet és hozhat létre, a futó programok közül bármelyiket leállíthatja, a gépet kikapcsolhatja. A `root` felhasználói jogai olyannyira kiterjednek mindenre, hogy a gépet bármikor működésképtelenné teheti, az oparációs rendszert megrongálhatja. Megfontolandó lehet mindezek miatt az a tanács, hogy gépünket ne rendszergazdaként használjuk – gondolva a véletlenül elkövetett rongálásokra –, csak akkor ha valóban rendszergazdai feladatokat kívánunk végezni.

A LINUX képes a felhasználó adatait – felhasználó név, jelszó, stb. – más gépekkel megosztani, NIS (*Network Information Service*, Hálózati Információs Szolgáltatás) szabvány szerint. Ez lehetővé teszi, hogy a felhasználó egységes néven és ugyanazon jelszóval használjon akár több gépet is, és adatait csak egy helyen, a NIS kiszolgálón kelljen nyilvántartani.

2.3 A képernyő kezelése

A LINUX képes egyszerre több programot is futtatni, általában azonban csak egyetlen monitor van a számítógéphez kapcsolva. A kényelem érdekében ezért lehetőségünk van az átkapcsolásra logikai képernyők között.

Az Alt+Ctrl+F1, Alt+Ctrl+F2... billentyűzetkombinációkhoz egy-egy logikai képernyő van rendelve, amelyek bármikor elérhetőek és egymástól teljesen függetlenül alkalmasak programok futtatására. Alapbeállítás esetén *karakteres képernyők* érhetőek el az első hat képernyőn, a hetediken azonban már *grafikus* rendszer, a ??? fejezetben ismertetésre kerülő X WINDOW. Az Alt+Ctrl+F7 feletti – F8, F9, stb. – kombinációk hatására a felhasználó által elindított további grafikus képernyők jelentkeznek be. Ha lenyomva tartjuk az Alt+Ctrl billentyűket, akkor végigjárhatjuk az F1-F6 billentyűk lenyomásával a karakteres képernyőket, a grafikus képernyőről azonban tpvább csak akkor jutunk, ha az Alt+Ctrl gombokat felengedjük majd újra lenyomjuk.

Az egyes logikai képernyők megőrzik a billentyűzeten található CapsLock és NumLock fények állapotát, vagyis amikor másik logikai képernyőre kapcsolunk, akkor a fények

A karakteres képenyőt használva a képernyő tetején tovatűnt szöveg egy részét megtekinthetjük a Shift+PageUp kombinációt lenyomva. A visszafelé görgetésre ilyenkor a Shift+PageDown szolgál. Ha átkapcsolunk a logikai képernyők eyg másikára

tippek:

A programok általában újrarajzolják a karakteres képernyőt a Ctrl+l kombináció megnyomására. Azt akkor használhatjuk, ha valamilyen oda nem illő dolgot látunk a képernyőn.

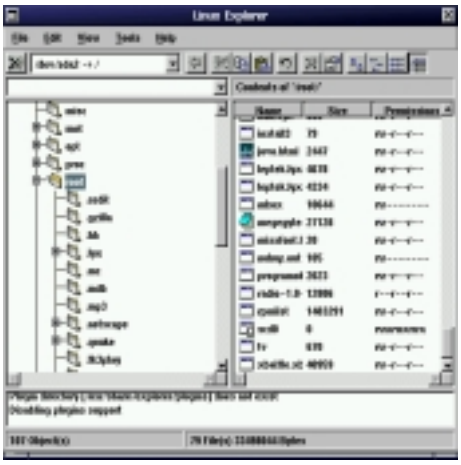
2.4 A LINUX adattárolási rendszere

A LINUX az adatokat – amelyekkel dolgozunk – állományokban, más néven fájlokban (*file*, akta) tárolja. Az állomány nem más mint az adatoknak egy halmaza, melynek önálló neve, tulajdonosa, stb. van. Ilyen értelemben az állomány igen hasonló egy iratgyűjtőhöz, amelynek a tartalma ugyan felbontható lapokra, mégis egy egységet képez, hiszen a címkéjén fontos adatok vannak feltüntetve, amelyek a teljes tartalomra – az összes lapra – érvényesek.

Mivel a számítógépen akár több tízezer ilyen állományt is elhelyezhetünk, mindenképpen szükséges egyfajta rendet tartanunk, hogy eligazodhassunk. A rend érdekében a könyvtár (*directory*, címtár) intézményét kell használnunk. A könyvtár hasonló tulajdonságokkal rendelkezik – tulajdonos, dátum, stb. – mint maga az állomány, de benne nem közvetlen adatokat tartunk, hanem állományokat. Ha irodai analógiát keresünk, akkor a könyvtár az a keményfedelű szalagos irományfedél, amelyben nem közvetlen tárolnak iratokat, hanem a dossziékat teszik bele.

Mivel a könyvtárak tartalmazhatnak könyvtárakat is, az egymásbaágyazás a végtelenségig folytatható. Tetszőleges mélységben halmozhatjuk fel az állományok csoportjait annak érdekében, hogy minél logikusabb és átláthatóbb rendszert építsünk fel.

A 2.1 ábrán egy LINUX állománykezelő segédprogram látható, ahogyan az az állományokat és könyvtárakat a felhasználó számára láttatja. A



ábra 2.1: A Linux explorer

bal oldalon vannak a könyvtárak, a jobb oldalon pedig az éppen nyitott könyvtár tartalma.

A felhasználó szempontjából két kitüntetett könyvtár létezik. Az első az *aktuális könyvtár*, az amelyikben a felhasználó éppen „tartózkodik". Sok parancs az aktuális könyvtárra vonatkozik – ha a felhasználó másképpen nem rendelkezik. A másik kitüntetett könyvtár a felhasználó *saját könyvtár*a (*home directory*, hazai könyvtár), amelyben azokat az adatokat és programokat tarthatja, amelyek a sajátjai.

2.4.1 Állománynevek

A Linux az általa tárolt állományokat névvel látja el, amelyet általában a felhasználó határoz meg. E nevekre érvényesek bizonyos megkötések, amelyeket a felhasználónak ismernie kell, hogy a szabályoknak megefelelő nevet adhasson állományainak[3]. A Linux az állományok közt a

[3]Fontos figyelembe vennünk azt, hogy amennyiben a Linux más operációs rendszerekkel közösen használ adatokat, akkor sok esetben az állománynevekre vonatkozó megkötések szigorodnak. Ha pl. a hajlékonylemezt más operációs rendszer formátumában használjuk, lehetséges, hogy az állományneveket rövidebbre kell választanunk, a gyártó által meghatározott módon.

név alapján tesz különbséget, ezért ugyanazon helyen nem lehet több állomány azonos néven.

A Linux állománynevek maximális hossza 255 írásjel, amely kevéssel több mint három sor a normál karakteres képernyőn.

Az állománynevekben szerepelhetnek az angol nyelv nagy- és kisbetűi, számjegyek és az írásjelek egy része. A nagy- és kisbetűk közt a linux különbséget tesz, ezért azok egymással fel nem cserélhetőek.

Amennyiben a Linux a magyar nyelv fogadására fel van készítve, az állománynevek tartalmazhatnak ékezetes magyar karaktereket. Ajánlatos ezek használatát kerülni, mivel az állományok hordozhatóságát veszélyeztetik, hiszen előfordulhat, hogy olyan gépre kívánjuk másolni állományainkat, amely nincs felkészülve a magyar nyelvű állománynevek fogadására.

Szóközt az állománynevek tartalmazhatnak, használatuk azonban kerülendő. Ennek praktikus oka az, hogy szóköz használata esetén a Linuxnak minden esetben jelezni kell, hogy nem két különálló névről van szó, hanem olyan állományról, melynek nevében szóköz található. Ezt megtehetjük az állománynevet határoló idézőjelekkel, de mivel a szóköz jól helyettesíthető az aláhúzás karakterrel, kerülendő. A `"space in name"` beírása helyett javasoljuk tehát a `space_in_name` formát, kényelmi okokból.

A - karakter a programoknak szóló parancssori üzenetet vezeti be, ezért azt állományok nevében nem szerencsés az első karakterként használni.

Bármennyi . karakter szerepelhet a névben. A Linux azokat az állományokat, amelyek . karakterrel kezdődnek (*dot files*, pontozott állományok), rejtett állományoknak tekinti.

2.4.1.1 Helyettesítőkarakterek

Amikor állományok nevére hivatkozunk, mindig lehetőségünk van állományok csoportjára érvényes kifejezést használni. Erre az ún. helyettesítőkarakterek segítségével nyílik lehetőségünk. A helyettesítőkarakterek a következőek:

?	Állhat egyetlen karakter helyén, bármi is legyen az. A `main.?` kifejezés illeszkedik a `main.c` állománynévre épp úgy, mint a `main.h` -ra.

`*` Bármennyi karakter helyén állhat. A `main.*` illeszkedik a `main.c` -re, `main.h` -ra, de a `main.c.gz` -re, vagy a `main.tgz` -re is.

`[a-z]` Betűk egy csoportjára illeszkedő kifejezés. A - jel két oldalán található jelen kívül illeszkedik a betűrendben köztük található összes karakterre. A `main.[0-9]*` kifejezés pl. illeszkedik minden olyan állománynévre, amely `main.` -al kezdődik és ezt legalább egy szám követi.

`[!a-z]` Minden olyen betűre illeszkedik, amely a - jel által elválasztott határoló karaktereken kívül esik. A `[!A-Z]` pl. minden karakterre illeszkedik, amely nem eleme az angol nyelv nagybetűinek.

A Linux igen korrekt módon kezeli a helyettesítőkaraktereket. A `*htm*` kifejezés pl. minden olyan állománynévre illeszkedik, amelyben valahol szerepel a `htm` karaktersorozat. A kifejezés kiértékelése nem áll le az első * vizsgálatánál – a * nem csak a név végének csonkolására alkalmas.

2.4.2 Tulajdonosok és jogok

Mivel a LINUX többfelhasználós operációs rendszer, szüksége van az állományok és könyvtárak tulajdonosainak nyilvántartására, hogy gondoskodhasson az adatok védelméről. Az állományok és könyvtárak mindegyike rendelkezik egy – és mindig csak egy – tulajdonossal, akinek személyéhez rendelődik három lehetséges kapcsoló, amelyek rendre *a)* olvasási jog, *b)* írási jog, *c)* futtatási jog. A három kapcsoló állapota a tulajdonos jogait jelzi az adott állományon.

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

A *tulajdonos írási jog kapcsolója* megmutatja, hogy a számára engedélyezett -e az adatok írása, vagyis módosítása. Ha ez be van kapcsolva, akkor a tulajdonos módosíthatja, bővítheti és törölheti az adott állományban található adatokat, sőt törölheti akár az egész állományt.

Ha e kapcsoló ki van kapcsolva, akkor a tulajdonos nem módosíthatja az adatokat, még az állomány végére sem fűzhet újabb információkat.

A *tulajdonos futtatási jogát jelző kapcsoló* azt mutatja meg, hogy a tulajdonos elindíthatja -e az adott állományt. Magától értetődik, hogy ennek a lehetőségnek csak programok esetében van értelme, adatokat tartalmazó állomány esetében e kapcsoló ki van kapcsolva.

A LINUX az állományok tulajdonságai közt tárolja a többi felhasználó jogait is. Ezek az egyszerűség érdekében ugyanígy három kapcsolóval vannak nyilvántartva, vagyis a többi felhasználó is rendelkezhet írási, olvasási és futtatási joggal, aszerint, hogy az állomány adatai közt rájuk vonatkozó három kapcsoló melyik állapotában van. Ha irodai példát keresünk, akkor az iratgyűjtő címkéje most valahogy így nézne ki:

Kép: név , tulajdonos és többiek jogai ixelje be amelyiket

A Linux lehetőséget ad munkacsoportok létesítésére, amely nagymértékben megkönnyíti a munkát. Minden felhasználó tartozhat egy – vagy több – csoporthoz, az állományok pedig csoporttulajdonossal is rendelkeznek. Az állománynak a tulajdonosa mellett tehát nyilván van tartva az is, hogy ő milyen minőségében birtokolja az adott állományt, vagyis melyik csoporttal közösen kívánja azt használni. Azt pedig, hogy hogyan kívánja közösen használni, az előzőekhez hasonlóan három kapcsoló mutatja, amelyek olvasási-, írási- és futtatási jogokat jelölnek. Most már kibővíthetjük az iratgyűjtő címkéjét a végleges formára:

kép, értelemszerűen.

A könyvtárak esetében a jogosultságok értelmezése némiképpen módosul.

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

A *könyvtáron értelmezett írási jog* újabb fájlok létrehozására és törlésére ad lehetőséget. Ha írási joggal nem rendelkezik valaki az adott könyvtárban, esetleg még írhat a benne levő állományokba vagy azokból adatokat törölhet. Egész állományokat nem hozhat létre és egész állományokat viszont nem törölhet. Amennyiben a könyvtárból az ott található állományokat nem tudja a felhasználó kitörölni, a könyvtárat

magát sem tudja törölni, hiszen csak üres könyvtárakat lehet kitörölni. Ha a könyvtár már üres, a felhasználónak nincs szüksége írási jogra a könyvtárra nézve ahhoz, hogy azt kitörölhesse.

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

Meg kell jegyeznünk, hogy a felhasználó, természetesen bármikor megváltoztathatja az általa birtokolt könyvtárakra és állományokra vonatkozó jogosultsági kapcsolókat – azokat is, amelyek saját magára vonatkoznak –, ezáltal megkerülheti a rá vonatkozó tiltásokat. Ennek ellenére e korlátozások nem feleslegesek, hiszen sokszor megóvnak a véletlen hibázásoktól.

2.4.3 Láncok és kötések

A LINUX lehetőséget ad arra, hogy egyazon állomány több néven is szerepeljen – esetleg más-más helyen. A más helyen vagy néven megjelenő állományok két fajtáját kötéseknek és láncoknak (link, lánc – symbolic link, szimbólikus lánc) nevezzük.

A láncok és kötések nem elengedhetetlenül fontosak a rendszer üzemelése szempontjából, használatuk azonban világosabbá teszi az adattárolási rendszert, ugyanakkor helytakarékossági szempontból is hasznos.

A Linux konvenció szerint könyvtárakba és alkönyvtárakba szétválogatva tartja az állományait, ahol szokás szerint az azonos funkciójú állományok kerülnek egyazon könyvtárba. A mágneslemezen található program beállításait tartalmazó állományok pl. abban a könyvtárban vannak, hol általában a beállításokat tároljuk, a futtatható állományok az egyéb futtatható állományok közt, stb. Ez az elv nagymértékben átláthatóvá teszi a egész rendszert, ugyanakkor segít abban is, hogy az egyes programok integrált egészként egységes rendszert alkossanak.

2.5 A BASH

A LINUX módot ad arra, hogy bizonyos feladatokat parancsok begépelésével végezzünk el. Így dolgozhatunk olyan képernyőn is, amely nem alkalmas grafikus megjelenítésre, lassú adatátviteli csatornát használva távoli gépen pedig sokszor egyszerűbben, gyorsabban fogalmazhatjuk

meg igényeinket a gép felé. Az számítógép ilyen használati módjának hátrányaként felróható, hogy szaktudást igényel, mivel a felhasználónak el kell sajátítania a parancsokat és azt, hogy hogyan lehet őket hoszabb kifejezésekké, mondatokká fűzni.

A parancsokat – amelyeket a felhasználó begépel – egy parancsértelmező kapja meg, és hajtja végre az Enter billentyű lenyomásakor. Több ilyen parancsértelmező – shell, vagyis burok – is létezik, melyek közül a legelterjedtebben, a BASH -t (*Bourne Again Shell*, már megint Bourne burok[4]) fogjuk részletesebben megvizsgálni.

A BASH nevét STEVEN BOURNEről kapta, aki eredetileg megalkotta.

2.5.1 Parancsok általános formája

A parancssor indítása után – amely általában a felhasználó bejelentkezésekor történik meg – egy parancskérő jelet ír a képernyőre, amely jelzi a felhasználó számára, hogy begépelheti parancsait. Ez a parancskér jel szinte bármi lehet, hiszen a felhasználó szabadon megváltoztathatja, alapesetben azonban formája a következő:

```
[pipas@pip src]$
```

A parancskérő jelben található információk a következőek:

felhasználó Az első szó – példánkban pipas – a felhasználó azonosítására szolgáló felhasználói név.

gép A második szó – a példában pip – a gépet azonosító név.

könyvtár A harmadik szó – itt src – annak a könyvtárnak a neve, amely az aktuális könyvtár éppen, vagyis amely könyvtárban a felhasználó „tartózkodik”. A parancskérő jelben nem telálható meg a teljes könyvtárnév, csak annak utolsó része.

Amennyiben a felhasználó a parancs begépelésekor idézőjelet – " vagy ′ – írt és azt nem zárta le, a BASH úgy dönt, hogy a parancs még nem komplett, ezért a további részek beírását várja. Azért, hogy jelezze a beírt kiegészítés az előző sorral együtt kerül majd kiértékelésre, a másodlagos parancskérő jelet írja a képernyőre:

[4]Az angol kifejezés így is fordítható a kiejtés szerint: „Lángolj ismét kagyló”.

```
[pipas@pip src]$ ls -l "l
>
```

Ha ebben az esetben megnyomjuk a Ctrl+c billentyűzetkombinációt, a félig beírt parancs nem hajtódik végre, az elsődleges parancskérő jel jelenik meg a képernyőn.

A Linux parancsok használata – kevés kivételtől eltekintve – egységes. Lássunk egy egyszerű példát!

```
[pipas@pip src]$ ls -l string.c
```

A parancs a fenti példa esetében az `ls`.

Az ezt követő `-l` a parancs *opciója*, kiegészítője. Az opciók olyan kapcsolók, amelyek a parancs viselkeését megváltoztatják, befolyásolják. Használatukkal ugyanazon parancs viselkedése igényeinknek megfelelően befolyásolható. Az opciók adják a parancs rugalmasságát. Ugyanazon parancs többféle opcióval is indítható, ilyenkor általában nem szükséges a - karakter többszöri kiírása, használhatjuk pl. a fenti esetben a `-la` opciót a `-l -a` helyett.

A parancs után írt string.c egy állománynév, amely azt határozza meg, hogy a parancsot mely állományon kívánjuk végrehajtani[5]. Általában elmondható, hogy ahol állományok neveit várja egy parancs, ott használhatunk az állománynevek általánosítására szolgáló helyettesítő karaktereket, vagy több állományt:

```
[pipas@pip src]$ ls -l string*
```

Amikor elindítunk egy parancsot, az általában három ponton épít ki kapcsolatot környezetével:

szabványos bemenet Sok program a szabványos bemenetről kapja az adatokat. A szabványos bemenet alapértelmezésben a billentyűzet.

szabványos kimenet A szabványos kimenet – amely általában a képernyő – a program eredményeit fogadja.

[5] Természetesen az, hogy mi történik az állománnyal a parancs jellegéből következik.

szabványos hibacsatorna A szabványos hibacsatorna fogadja a program üzeneteit. Habár a szabványos hibacsatorna a szabványos kimenethez hasonlóan a képernyő felé van irányítva alapesetben, attól külön kezelendő, hiszen átirányításával az adatok az üzenetektől elválaszthatóak.

Mindhárom szabványos csatorna átirányítható állományba vagy más programhoz.

A szabványos bemenet átirányítására a < jel szolgál. A parancs használatakor a < után írt áloományt tekinti bemeneti csatonának, nem pedig a billentyűzetet. A következő példánál a levelzőprogram bemenetét irányítjuk egy állományba, amelynek közönhetően nem kell a billentyűzetről begépelnünk a levelet – a levelezőprogram a main.c állományt küldi el a címzettnek:

```
[pipas@pip src]$ mail root <main.c
```

A szabványos kimenetet a > karakter segítségével iányíthatjuk át állományba. A következő példában az ls parancs kimenetét irányítjuk át a képernyőről a lista nevű állományba.

```
[pipas@pip src]$ ls -l >lista
```

A > használatakor az állomány automatikusan létrejön, vagy ha már létezik, *tartalma megsemmisül.* Ha azt szeretnénk, hogy a létező állomány ne semmisüljön meg, akkor a >> kettős jelet kell használnunk. Ilyenkor a létező állomány végére kerül a parancs kimenetéről származó minden adat:

```
[pipas@pip src]$ echo "vege" >>lista
```

A parancsok szabványos hibacsatornája a 2> jellel irányítható át állományba. Ha az állomány tartalmát meg szeretnénk őrizni, s az üzeneteket a végéhez hozzáfűzni, a szabványos kimenethez hasonlóan kettőzni kell a > jelet:

```
[pipas@pip src]$ ls kjh 2>>~/hiba
```

A parancsok három szabványos csatornája egyszerre is átirányítható a fenti módszerek együttes alkalmazásával.

Ha valamely program szabványos kimenetéről érkező adatokat szeretnénk más program bemenetére irányítani további feldolgozás céljából, akkor a | jelet kell használnunk. Így olyan „csővezetéket” kapunk, amelyben az adatok balról jobbra végighaladnak minden felsorolt programon:

```
[pipas@pip src]$ ps aux | grep root | less
```

A Unix rugalmasságában igen nagy szerepe van ezeknek a csöveknek, hiszen segítségükkel olyan feladatokat oldhatunk meg, amelyekre nincsen külön parancs, de a meglévő építőelemekből könnyedén összeállítható a megfelelő kifejezés.

2.5.2 A UNIX állománykezelő parancsai

A LINUX szabványos UNIX parancsokat használ, sok helyütt kibővítve azok képességeit. Ha e parancsokat megismerjük, akkor nem csak a LINUX kezelésére leszünk képesek, hanem más gyártók kereskedelemben kapható operációs rendszereit – amelyek szintén szabványos UNIX felületet adnak – használni tudjuk.

2.5.2.1 Az aktuális könyvtár lekérdezése

A felhasználó számára az aktuális könyvtár különleges jelentőséggel bír, hiszen a parancsok – ha a felhasználó másképpen nem rendeli – az aktuális könyvtárban található állományokra és alkönyvtárakra vonatkoznak. Az aktuális könyvtárat a `pwd` (*print working directory*, a munkakönyvtár kinyomtatása) paranccsal lehet lekérdezni.

```
[root@mad forprint]# pwd
/mnt/mobil1/forprint
[root@mad forprint]#
```

2.5.2.2 Az aktuális könyvtár megváltoztatása

A `cd` (*change directory*, könyvtárváltás) parancs segítségével a felhasználó megváltoztathatja az aktuális könyvtárat, mondhatnánk „másik könyvtárba

léphet". A parancs után meg kell adni annak a könyvtárnak a nevét amit aktuális könyvtárrá akarunk tenni. A könyvtárakat „/" jellel elválasztva kell begépelnünk:

```
[root@mad forprint]# cd /usr/X11R6/lib/X11/
[root@mad X11]#
```

A fenti példában a könyvtárnevek előtt „/" szimbólum található. Ez a megadási mód az *abszolút könyvtárleíró*, amely nevét onnan kapta, hogy a LINUX ebben az esetben a gyökérkönyvtárból indulva keresi a könyvtárat. Használhatunk *relatív könyvtárleírót* is, ekkor nem kell „/" jellel kezdenünk azt.

```
[root@mad X11]# cd doc/
[root@mad doc]#
```

A relatív módszer esetében az új aktuális könyvtár mindig a jelenlegi munkakönyvtárból kiindulva adható meg. Ez némi gépelést megtakarít, de ügyelnünk kell arra, hogy pl. a `cd lib` parancs egészen más könyvtárba visz mindket a `/usr/X11R6` vagy `/usr` könyvtárakból kiindulva.

A relatív könyvtármegadás egy speciális formája a `..` könyvtárra való hivatkozás. Minden könyvtárban – kivéve a gyökérkönyvtárat – található egy `..` bejegyzés, amely az adott alkönyvtár szülőjére vonatkozik. Így a `cd ..` parancs a könyvtárstruktúrában egyel feljebb visz minket. A `cd` parancs magában kiadva a felhasználót a saját könyvtárába viszi.

2.5.2.3 Állománynevek listázása

Az `ls` (*list*, lista) parancs a lemez állományainak neveit és tulajdonságait listázza. A parancs magában megadva az aktuális könyvtár állományneveit nyomtatja a képernyőre.

```
[root@pip bin]# ls
After         kde.sh          magyar
us.kyb        Fvwm2           kwm
magyar.kyb    win95           bookitar
ma            magyar.kyb.old  x
enl           mac             us
xt
```

Ha részletesebb információkat kívánunk kapni a fájlokról, akkor az `ls` parancs `-l` (*long*, hosszú) kapcsolóját kell használnunk.

```
[root@pip bin]# ls -l
total 30
-rwxr-xr-x 1 root root   10 Dec 22 00:59 After
-rwxr-xr-x 1 root root   23 Oct 11 1997  Fvwm2
-rw-r--r-- 1 root root   45 May 17 14:55 bookie-1.85.tar.gz
-rwxr-xr-x 1 root root   48 Jan 25 03:27 bookitar
-rwxr-xr-x 1 root root   35 Oct 28 1997  enl
-rwxr-xr-x 1 root root  129 Nov  4 1997  kde.sh
-rwxr-xr-x 1 root root  131 Nov  3 1997  kwm
-rw-r--r-- 1 root root    0 Oct 23 1997  ma
-rwxr-xr-x 1 root root   24 Oct 11 1997  mac
-rwxr-xr-x 1 root root   30 Mar 20 1997  magyar
-rw-r--r-- 1 root root 5416 Oct 23 1997  magyar.kyb
-rw-r--r-- 1 root root 5435 Oct 23 1997  magyar.kyb.old
-rwxr-xr-x 1 root root   25 Mar 20 1997  us
-rw-r--r-- 1 root root 4807 Oct  3 1997  us.kyb
-rwxr-xr-x 1 root root   22 Oct 11 1997  win95
-rwxr-xr-x 1 root root   43 Nov 29 14:10 x
-rwxr-xr-x 1 root root  110 Oct 26 1997  xt
```

Az *első betű* jelzi számunkra, hogy fájlról vagy könyvtárról van -e szó. Állomány esetén – jel, könyvtár esetében d betű áll itt. A *következő kilenc betű* rendre a tulajdonos, csoporttulajdonos és a kívülállóak *jogait* jelzi három hárombetűs csoportban. Ha a megfelelő helyen – jel áll, akkor az adott személy nem rendelkezik a jogosultsággal, ha pedig `r`, `w`, vagy `x` betű, akkor az rendre olvasási, írási és futtatási jogot jelent. A *következő számoszlop* az állományról készített *csatolások számát* mutatja. Amennyiben itt `1` -et látunk az állományról a rendszeren nincsen láncolt másolat. A *következő két oszlop* a *tulajdonos* és a *csoporttulajdonos* neve. Ha itt számot látunk, akkor az állomány tulajdonosa vagy csoporttulajdonosa már nem létezik a rendszer nyilvántartásában. Az *ötödik oszlop* az *állomány méretét* jelzi byteban. Az *utolsó oszlop az állomány nevét* mutatja, *előtte pedig az utolsó módosítás dátuma* látható.

Az `ls` parancsnak megadhatjuk, hogy mely állományt, vagy állományokat szeretnénk kilistázni:

```
[root@pip bin]# ls -l /etc/*.sh
```

Ha az ls -nek könyvtárnevet adunk meg a parancssorban, akkor az a tartalmát listázza. Ha ehelyett magának a könyvtárnak az adataira vagyunk kíváncsiak, akkor a -d (*directory*, könyvtár) opciót kell megadnunk:

```
[root@pip /root]# ls -ld /bin
drwxr-xr-x   2 root     root         2048 May 16 21:53 /bin
[root@pip /root]#
```

2.5.2.4 Állományok létrehozása és törlése

A touch (*touch*, megérinteni) és rm (*remove*, leszedni) parancsok fájl létrehozására és törlésére vonatkoznak[6].

A touch egy 0 hosszúságú állományt hoz létre azon a néven, amelyet a parancssorban megadunk – értesítést a képernyőre csak hiba esetén küld:

```
[root@pip /root]# touch file
[root@pip /root]#
```

Alapesetben az rm parancs megerősítést kér a felhasználótól az állomány törlése előtt – amint azt a péda mutatja – de ez a -f (*force*, erőltetni) opcióval kikapcsolható.

```
[root@pip /root]# rm file
rm: remove ‘file’? y
[root@pip /root]#
```

Az rm parancs képes egyszerre több állományt törölni, ekkor a -f opció igen hasznos lehet[7]:

```
[root@pip /root]# rm -f *dvi
[root@pip /root]#
```

[6] A felhasználónak megfelelő jogkörrel kell rendelkeznie, ellenkező esetben a művelet hibaüzenettel leáll.

[7] Óvatosan kell bánnunk az rm paranccsal, mert a törölt állományok visszaállítására általában nincsen lehetőség.

Az rm parancs képes arra, hogy teljes könyvtárstruktúrát törölön – azok tartalmával egyetemben – ehogyan ezt a 2.5.2.6 fejezetben látni fogjuk.

2.5.2.5 Állományok másolása

A cp (*copy*, másolás) és mv (*move*, átrakás) parancsok egyaránt fájlok átmásolását teszik lehetővé, de míg a cp parancs esetében az eredeti megmarad, addig az mv parancs azt megsemmisíti[8]. A cp és mv parancsok a képernyőre csak hiba esetén nyomtatnak üzenetet.

```
[root@pip /root]# cp file /home/pipas/
[root@pip /root]#
```

Amennyiben a másolás vagy mozgatás már létező állomány megsemmisítésével jár – két egyforma nevű bejegyzés nem lehet ugyanazon könyvtárban – az mv és cp parancsok megerősítést kérnek.

```
[root@pip /root]# cp file file1
cp: overwrite ‘file1’? n
[root@pip /root]# mv file file1
mv: replace ‘file1’? y
[root@pip /root]#
```

A -f (*force*, erőltetni) opció segítségével a megerősítés kikapcsolható, ilyenkor a felülírás figyelmeztetés nélkül megtörténik.

2.5.2.6 Könyvtárak létrehozása és törlése

Az mkdir (*make directory*, könyvtár létrehozása) és rmdir (*remove directory*, könyvtár törlése) a háttértár könyvtárstruktúrájának megváltoztatására szolgál.

Az mkdir parancs alapértelmezésben csak egy könyvtár létrehozására használatos, nem létező könyvtárban nem lehet könyvtárat létrehozni:

[8] Érdemes megjegyezni, hogy az mv parancs ugyanazon partíción belül sokkal gyorsabb lehet mint a cp és rm parancsok egymás utáni alkalmazása, ilyenkor ugyanis az adatok nem kerülnek fizikailag más helyre csak a fájl jellemzőit tartalmazó bejegyzés.

```
[root@pip mobil1]# mkdir  nem.letezik/ilyen.sincs/konyvtar
mkdir: cannot make directory
‘nem.letezik/ilyen.sincs/konyvtar’: No such file or directory
[root@pip mobil1]#
```

A -p (*parent*, szülő) opció megadásával az összes szükséges könyvtárat megkísérli létrehozni:

```
[root@pip mobil1]# mkdir -p nem.letezik/ilyen.sincs/konyvtar
[root@pip mobil1]# tree nem.letezik
nem.letezik
‘-- ilyen.sincs
    ‘-- konyvtar

2 directories, 0 files
[root@pip mobil1]#
```

Könyvtárat törölni az rmdir paranccsal csak akkor lehet, ha az egyébként nem tartalmaz állományokat. Ha a könyvtárban található állományokat is törölni akarjuk, akkor az rm parancsot használhatjuk -R (*recursive*, ???) opcióval. Ilyenkor a -f (force, erőltetni) opciót is érdemes bekapcsolni.

```
[root@pip /root]# rm -fR directory
[root@pip /root]#
```

Vigyáznunk kell, mert a törölt állományok visszaállítására általában nincs lehetőségünk.

2.5.2.7 A tulajdonos megváltoztatása

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

```
[root@pip /root]# ls -l file
-rw-r--r--   1 root     root            0 May 24 22:57 file
[root@pip /root]# chown pipas file
[root@pip /root]# ls -l file
-rw-r--r--   1 pipas    root            0 May 24 22:57 file
[root@pip /root]#
```

A példán látható, hogy az állomány tulajdonosa eredetileg a root volt, de a chown parancs megváltoztatta azt. A csoport nem változott. Ha a csoporttulajdonost is meg akarjuk változtatni, akkor a tulajdonos neve után kettősponttal elválasztva kell az új csoportot megadnunk:

```
[root@pip /root]# chown pipas:gazd file
[root@pip /root]#
```

A chown parancs képes egyszerre több állomány tulajdonosát megváltoztatni, a -R opcióval akár alkönyvtárakkal – és azok tartalmával – egyetemben.

```
[root@pip /root]# chown -R pipas /home/pipas
[root@pip /root]#
```

Óvatosan kell bánnunk a tulajdonos megváltoztatásával, mert könnyen előfordulhat, hogy ha egy „saját” állományt valakinek „odaadunk”, akkor már nem tudunk hozzáférni – visszavenni sem –, vagyis „kizárhatjuk” magunkat.

2.5.2.8 A csoporttulajdonos megváltoztatása

A chgrp (*change group ownership*, csoporttulajdonos megváltoztatása) paranccsal megváltoztathatjuk az állományok és könyvtárak csoporttulajdonosát. Erre alkalmas a 2.5.2.7 fejezetben bemutatott chown parancs is, ha azonban csak a csoporttulajdonost kívánjuk megváltoztatni – a tulajdonost nem – akkor a feladatra a chgrp használható.

```
[root@pip /root]# chgrp -R pipas /home/pipas/
[root@pip /root]#
```

Amint látható a chgrp parancsnak a csoporttulajdonos nevét kell megadnunk majd az állományokat – vagy könyvtárakat – amelyeken a változtatást végre akarjuk hajtani. Az is látszik a példán, hogy a chgrp is

értelmezi a `-R` opciót, amely az összes alkönyvtárra – és tartalmukra – kiterjeszti a hatását.

2.5.2.9 Az állományjogok megváltoztatása

Az állományokra vonatkozó jogokat a `chmod` (*change mode*, állapotváltás) paranccsal változtathatjuk meg. A parancsnak a jogosultságokat jelző kapcsolók új állását és az állomány(ok) nevét kell megadnunk. A jogok a következőek lehetnek:

r Olvasási jog (*read*, olvasás).

w Írási jog (write, írás).

x Futtatási jog (*execution*, végrehajtás)

A jogok vonatkozhatnak a következő személyekre:

u Tulajdonos (*user*, felhasználó).

g A csoporttulajdonos (*group*, csoport).

o Mindenki más, vagyis azok, akik nem tulajdonosok és nem tartoznak a csoporttulajdonossal jelzett csoportba (*other*, mások).

A `chmod` parancsnak a személyeket vesszővel elválasztva kell megadni, +, − vagy = jellel pedig a jogokat utánaírni, ahogyan azt a példa mutatja:

```
[root@pip /root]# chmod u+r+w-x,g+r-w-x,o-r-w-x file
[root@pip /root]# ls -l file
-rw-r-----   1 root     root            0 May 24 23:21 file
[root@pip /root]#
```

A + jelzi a jog hozzáadását még a − a jog visszavonását. Megadhatunk hiányos formában is jogokat, ez esetben a nem érintett kapcsolók nem változnak meg:

```
[root@pip /root]# chmod u+x file
[root@pip /root]# ls -l file
-rwxr-----   1 root     root            0 May 24 23:21 file
[root@pip /root]#
```

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

```
[root@pip /root]# chmod +x file
[root@pip /root]# ls -l file
-rwxr-x--x   1 root     root            0 May 24 23:21 file
[root@pip /root]#
```

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

```
[root@pip /root]# chmod u=rw file
[root@pip /root]# ls -l file
-rw-r-----   1 root     root            0 May 24 23:21 file
[root@pip /root]#
```

2.5.2.10 Állományok típusának megállapítása

A felhasználó számára sokszor nem világos az első pillanatban, hogy egy állomány milyen formában tartalmaz adatokat. Az elnevezés általában beszédes – a `txt` végződés szöveges állományt jelez, a `ps` POSTSCRIPT formátumot sejtet, stb. -, de nem mindig segít. A `file` parancs a parancssorban megadott állomány típusát kísérli megállapítani és a képernyőre nyomtatni.

```
[root@mad forprint]# file pfgguide.ps
pfgguide.ps: PostScript document text conforming at level 2.0
[root@mad forprint]# file latex2e.dvi
latex2e.dvi: TeX DVI file (TeX output 1997.01.02:0137)
[root@mad forprint]# file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1,
dynamically linked, stripped
[root@mad forprint]#
```

Amint látható a file parancs részletes információkkal szolgál a leggyakrabban használt állományformátumokról. Erre azért képes, mert nem az állomány nevéből következtet a típusra, hanem az állomány tartalmát is megvizsgálja.

2.5.2.11 Állományok tartalmának listázása

Szöveges állományt lehetőségünk van a képernyőre listázni, és elolvasni. A legegyszerűbb parancs a `cat` (*concatenate*, összefűz):

```
[root@mad forprint]# cat file
A file tartalma
[root@mad forprint]#
```

A `cat` parancs a képernyőre listázza az állományt, ha az nem fér a képernyőre, akkor a felső részen a sorok eltűnnek. A `cat` azért kapta a nevét, mert a segítségével könnyen lehet állományokat összefűzni egyetlen fájlba:

```
[root@mad forprint]# cat *.txt >>kimenet.txt
[root@mad forprint]#
```

A példában az összes txt végződésű állományt listázzuk, a kimenetet pedig a `kimenet.txt` állományba irányítottuk.

Ha bináris állományt listázunk a képernyőre a `cat` parancs segítségével, akkor előfordulhat, hogy a betűk mindenféle kriksz-kraksz grafikus karakterekké válnak. Ekkor szükséges lehet a terminál alapbeállításának visszaállítása a `reset` (visszaállít) paranccsal[9].

```
[root@mad forprint]# reset
Erase set to backspace.
Kill is control-U (^U).
Interrupt is control-C (^C).
[root@mad forprint]#
```

[9] A reset parancsot „vakon” kell begépelnünk, mivel a betűk olvashatatlanul jelennek meg ilyenkor.

Ha hosszú állományt akarunk a képernyőn olvasni, akkor valószínűleg szeretnénk elkerülni hogy a sorok „elszaladjanak” a képernyő tetején. Erre a megoldást a `more` (több) vagy `less` (kevesebb) parancsok adják. A `more` segítségével az állományban csak előre haladhatunk – az Enter billentyűvel -, a `less` mindkét irányba képes görgetni – a kurzormozgató billentyűkkel. Mindkét programból a `q` betű lenyomásával léphetünk ki. A használat:

```
[root@mad forprint]# more file
```

illetve

```
[root@mad forprint]# less file
```

2.5.2.12 Könyvtárak mérete

A LINUX a `du` (*disk usage*, lemezhasználat) parancs segítségével információkat szolgáltat arról, hogy az adott alkönyvtár – és tartalma – összesen mekkora méretű. A `-k` (kilobyte) opciót megadva az eredményt kilobyteokban kaphatjuk meg.

```
[root@mad mobil1]# du -k Linux*
24823   Linux-book
1       Linux-magazin/src/stage002
5       Linux-magazin/src/stage001
754     Linux-magazin/src
19302   Linux-magazin
200     LinuxDAP.tar.gz
[root@mad mobil1]#
```

Látható, hogy a `du` felsorolja az összes alkönyvtárat, valamint azt, hogy mekkora területet foglalnak el. A `-s` (*summarize*, összegzés) opcióval kérhetjük, hogy a részeredmények ne kerüljenek a képernyőre:

```
[root@mad mobil1]# du -ks Linux*
24823   Linux-book
19302   Linux-magazin
200     LinuxDAP.tar.gz
[root@mad mobil1]#
```

2.5.2.13 Tömörítés

Az állományok mérete általában csökkenthető bonyolult matematikai eljárásokkal úgy, hogy az eredeti tartalom halyreállítható legyen. Ezt a műveletet tömörítésnek nevezük, míg a helyreállítást kicsomagolásnak. Az, hogy a tömörítés során milyen mértékben sikerül az állomány méretét csökkenteni, az állomány jellegétől és a használt program képességeitől is függ.

A tömörítés és a helyreéllítás ugyanazon módszer szerint kell, hogy történjék, ezért a Linux alatt többféle tömörítő – és kicsomagoló – program is elterjedt. Ha más operációs rendszer alatt végeztük a tömörítést, a kicsomagolás LINUX alatt is elvégezhető – ha rendelkezünk a megfelelő programmal.

A Linux alatt általában a `gzip` nevű programot használjuk, az általa előállított állományt pedig a `gunzip` segítségével csomagolhatjuk ki. Valójában a `gzip` és `gunzip` nevű program ugyanaz, csak más néven. Azért szerepel két néven, mert indításakor megvizsgálja, hogy mi a saját neve és ettől függően tömörítést vagy kicsomagolást végez:

```
[root@pip /root]# ls -l /bin/gunzip /bin/gzip
-rwxr-xr-x   3 root     root        45420 Oct  3  1997 /bin/gunzip
-rwxr-xr-x   3 root     root        45420 Oct  3  1997 /bin/gzip
[root@pip /root]#
```

Ez az érdekes módszer elterjedten használatos LINUX alatt.

Ha egy állományt helytakarékosság miatt tömöríteni szeretnénk, akkor a `gzip` parancs után csak be kell írnunk a nevét, s megvárni, amíg a tömörítés megtörténik[10]. A következő példán látható, hogy az `rpmlist` nevű állomány tömörítése hogyan megy végbe:

```
[root@pip /root]# ls -l rpmlist
-rw-r--r--   1 root     root      1483291 Feb 14 22:37 rpmlist
[root@pip /root]# gzip rpmlist
[root@pip /root]# ls -l rpmlist.gz
-rw-r--r--   1 root     root       185921 Feb 14 22:37 rpmlist.gz
[root@pip /root]#
```

[10] A tömörítés bonyolult matematikai eljárás, ezért – különösen nagyméretű állományok esetében – hosszú idít vehet igénybe.

A példában látható mintegy 1.4 Mbyte méretű `rpmlist` állomány tömörítésének végeredménye az `rpmlist.gz`, amely 185 Kbyte, vagyis jóval kisebb az eredetinél. A `gzip` letörölte az eredeti – tömörítetlen – állományt, s egy `.gz` végződésű tmörített állományt hozott létre.

A `.gz` végződés konvenció szerint a `gzip` paranccsal tömörített állományok számára van fenntartva. Ha a konvenciótól eltérünk és nem ismerjük fel az állomány jellegét, a 41 oldalon 2.5.2.10 fejezetben ismertetett `file` parancs segítségével is lekérdezhetjük, hogy milyen típusú az állomány:

```
[root@pip /root]# file rpmlist.gz
rpmlist.gz: gzip compressed data, deflated,
original filename, last modified: Sat Feb 14 22:37:46 1998,
os: Unix
[root@pip /root]#
```

A kicsomagolás a `gunzip` paranccsal történik:

```
[root@pip /root]# gunzip rpmlist.gz
[root@pip /root]#
```

A kicsomagolás során az eredeti állomány – az eredeti néven – létrejön, a `.gz` végződésű tömörített pedig megsemmisül.

Ha egyszerre több állományt kívánunk tömöríteni, akkor a `gzip` parancsnak megadható az állománynév helyettesítő karakterekkel is:

```
[root@pip src]# gzip *.c
[root@pip src]#
```

Ekkor azonban a `gzip` nem egyetlen állományt hoz létre, hanem minden egyes állomány helyett – amelyre a helyettesítőkarakterek érzényesek – egy `.gz` végződésű tömörített állományt hoz létre. Ha a célunk az, hogy több állományt egyetlen tömörített fájlba tegyünk, akkor a 47 oldalon 2.5.2.14 fejezetben ismertetett tar nevű programot kell használnunk.

A `gzip` képes többféle sebességgel is tömöríteni. Amennyiben előírjuk számára, hogy a tömörítést hosszabb ideig végezze, a tömörítés jobb minőségű lehet, vagyis a tömörített állomány kisebb méretű lesz:

```
[root@pip /root]# gzip -9 rpmlist
[root@pip /root]# ls -l rpmlist.gz
-rw-r--r--   1 root     root       178271 Feb 14 22:37 rpmlist.gz
[root@pip /root]#
```

A tömörítés sebességének meghatározásához a -1 és -9 közti számokat adhatjuk meg, ahol a nagyobb szám lassabb, de jobb minőségű tömörtést eredményez. Alapértelmezésben a gzip a hatos szintű tömörítést használja.

LINUX alatt elérhető a compress nevű tömörítőprogram is, amely a UNIX rendszerek elterjedt programja, s ezért szükséges lehet. A compress a tömörített állományokat a .Z végződéssel látja el:

```
[root@pip /root]# compress rpmlist
[root@pip /root]# ls -l rpmlist.Z
-rw-r--r--   1 root     root       336837 Feb 14 22:37 rpmlist.Z
[root@pip /root]#
```

A kicsomagolás ez esetben történhet az uncompress nevű paranccsal, de a gunzip is használható:

```
[root@pip /root]# gunzip rpmlist.Z
[root@pip /root]#
```

Ha LINUX alatt szeretnénk kicsomagolni a népszerű arj nevű programmal tömörített állományokat, akkor használhatjuk az unarj nevű programot:

```
[root@pip block]# unarj x block.arj
UNARJ (Demo version) 2.41a Copyright (c) 1991-93 Robert K Jung

Processing archive: block.arj
Archive created: 1998-07-06 13:09:28, modified: 1998-07-06 13:09:28
Extracting blscore.dat                Binary file!  CRC OK
Extracting blockout.set               Binary file!  CRC OK
Extracting bl.exe                     Binary file!  CRC OK
  3 file(s)
[root@pip block]#
```

A zip kiterjesztést viselő tömörített állományok kicsomagolására az unzip parancs alkalmas LINUX alatt:

```
[root@pip test]# unzip block.zip
Archive:  block.zip
  inflating: bl.exe
  inflating: bl2.ovl
  inflating: blockout.set
[root@pip test]#
```

Az `lzh` végződést viselő tömörített állományokat az `lha` nevű program csomagolja ki:

```
[root@pip test]# lha x block.lzh
bl.exe  - Melted
bl2.ovl - Melted
[root@pip test]#
```

2.5.2.14 Archiválás

Biztonsági másolatot készíteni szalagos egységre LINUX alatt a `tar` (*tape archive*, szalagos arhívum) paranccsal lehetséges. A `tar` arra is alkalmas, hogy segítségével az archívumot állományba mentsük, segítségével egész könyvtárstruktúrák egyetlen állományba másolhatóak.

A `tar` képes a `gzip` tömörítőprogrammal együtt dolgozni, amely esetben könyvtárakat alkönyvtáraikkal és tartalmukkal együtt egyetlen tömörített állományba másolhatóak biztonsági mentés vagy szállítás céljából.

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

```
[root@pip /root]# tar -cf bookie-1.85.tar bookie-1.85
[root@pip /root]# ls -l bookie-1.85.tar
-rw-r--r--   1 root     root      1884160 Jul 11 12:06 bookie-1.85.tar
[root@pip /root]#
```

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

A tar – mint láttuk – létrehozta az archívumot egyetlen állományban, speciális archív formátumban tartalmazza a bookie-1.85 nevű könyvtárat és teljes tartalmát:

```
[root@pip /root]# file bookie-1.85.tar
bookie-1.85.tar: GNU tar archive
[root@pip /root]#
```

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

```
[root@pip /root]# tar -xf bookie-1.85.tar
[root@pip /root]#
```

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

A tar alkalmas a gzip tömörítőprogrammal való együttműködésre is. Amennyiben a -z (gzip) opciót kapja, az archív állományt tömöríti – helytakarékosság céljából. A következő példában látható, hogy a z opció az f elé került, mivel az f után mindenképpen az állomány nevének kell következnie:

```
[root@pip /root]# tar -czf bookie-1.85.tar.gz bookie-1.85
[root@pip /root]#
```

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

```
[root@pip /root]# tar -xzf bookie-1.85.tar.gz
[root@pip /root]#
```

Ha az archívumot szalagos egységre kívánjuk létrehozni, akkor használnunk kell az `mt` (*magnetic tape*, mágneses szalag) parancsot is a szalagos egység vezérlésére. Az mt segítségével parancsokat adhatunk a számítógéphez kapcsolt mágnesszalagos egységnek – vagy egységeknek – s a parancsok végrehajtásáról a képernyőn értesülhetünk.

Az `mt` számára meg kell adnunk a megcélzott szalagos egységet s egy parancsot, amelyet végre kívánunk hajtani:

```
[root@pip /root]# mt -f /dev/tape status
SCSI 2 tape drive:
File number=0, block number=0, partition=0.
Tape block size 0 bytes. Density code 0x0 (default).
Soft error count since last status=0
General status bits on (40050000):
 BOT DR_OPEN IM_REP_EN
[root@pip /root]#
```

Az `mt -f` opciója (*file*, állomány) szolgál a szalagos egység azonosítására. A `/dev` könyvtárban minden szalagos egységhez egy bejegyzés – látszólagos állomány – tartozik, amelyre a parancs kiadásakor hivatkoznunk kell. Itt is érvényes, hogy a `-f` opciót mindig egy állománynévnek kell követnie.

A példában látható status (állapot) szó a parancs, amelyet a szalagos egységnek kiadunk. A status hatására a szalagos egység jelenti állapotát s az mt számunkra a képernyőre írja. Példánkban megtalálható a „*Tape block size 0 bytes.*” (Szalag adatblokk 0 byte méretű.) szöveg, amely arra enged következtetni, hogy az egységben nincsen szalag – vagy kazetta. Ha a szalagos egységet feltöltjük, a blokkméret nullától különböző értéket vesz fel:

```
[root@pip /root]# mt -f /dev/tape status
SCSI 2 tape drive:
File number=0, block number=0, partition=0.
Tape block size 512 bytes. Density code 0x13 (DDS (61000 bpi)).
Soft error count since last status=0
General status bits on (41010000):
 BOT ONLINE IM_REP_EN
[root@pip /root]#
```

Fontos megjegyeznünk a blokkméretet, mivel az archiválás során szükségünk lesz rá.

Az mt által sokféle parancs adható ki a szalagos egységnek, melyek közül a legfontosabbak a következőek:

törlés `erase` (törlés). A szalagos egység törlését végzi el. A törlés általában meglehetősen hosszú folyamat, ráadásul mellőzhető, hiszen az új adatok mentése megsemmisíti a szalag eredeti tartalmát. Csak akkor használjuk, ha valóban törölni kívánjuk a szalag tartalmát.

visszecsévélés `rewind` (vissza). A szalagot a szalagos egység visszacsévéli az elejére. Óvatosnak kell lennünk, mert ha a visszacsévélt szalagra írunk, akkor a szalag tartalma felülíródik.

kikapcsolás `offline` (leválasztás). Hatására a szalagos egység a szalagot az elejére csévéli, majd – ha arra képes – kifűzi és kiadja a kazettát.

jelentés `status` (állapot). A szalagos egység állapotáról ad információt a képernyőre.

Ha archívumot kívánunk létrehozni szalagos egységen, először fel kell jegyeznünk a szalag blokkméretét, amelyet a status parancs ír a képernyőre. Szükség esetén vissza kell csévélnünk a szalagot az elejére:

```
[root@pip /root]# mt -f /dev/tape rewind
[root@pip /root]#
```

Most létrehozhatjuk az archívumot a `tar` program segítségével:

```
[root@pip /root]# tar -cf /dev/tape -b 1 bookie-1.85
[root@pip /root]#
```

Amint látjuk, a `tar` számára megadott fájlnév most a szalagos egységre mutató látszólagos állomány a `/dev` könyvtárban. A `-b` opció a blokkméret megadására szolgál, amelyre szalagos egység használatakor van szükség. A tar a `-b` opció után egy számot vár, amely megmondja,

hogy a szalagon található blokkok az 512 byte hányszorosát tartalmazzák. Ha a szalag 512 byteot tárol egy blokkban, akkor a -b után 1 áll, ha 1024 byteos a szalag blokkmérete, akkor 2, stb. (A szalagra készített archívum esetén is használható a `-z` opció a tömörítés bekapcsolására.)

Az archívum ezzel elkészült, a szalag kifűzhető és elzárható:

```
[root@pip /root]# mt -f /dev/tape offline
[root@pip /root]#
```

Ha a szalagról a mentett adatokat helyre akarjuk állítani, akkor a `tar` programmal tehetjük azt meg a már ismert `-x` opcióval:

```
[root@pip /root]# tar -xf /dev/tape -b 1
[root@pip /root]#
```

Szalagra készített mentés esetében viszonylag lassú lehet az adatátvitel sebessége, s mivel általában nagy mennyiségű adatról van szó, számítanunk kell a hosszú mentési időre. Nem ritka az órákig tartó mentés sem, de ez szerencsére nem okoz fennakadást, mivel a LINUX közben más feladatokat is végezhet, hiszen számára a mentés nem megterhelő feladat.

2.5.2.15 Meghajtók beillesztése és kikapcsolása

A meghajtók beillesztése és kikapcsolása általában a rendszergazda előzetes hozzájárulását igényli!

A Unix a különféle meghajtókat – winchesterek, hajlékonylemezek, CD-ROMok, stb. – egységes fájlrendszerbe foglalva teszi elérhetővé a felhasználó számára. A gyökérkönyvtárból indulva bejárható a teljes fájlrendszer, az egyes meghajtók tartalma egy-egy alkönyvtárban jelenik meg. Amikor pl. a felhasználó a /mnt/cdrom könyvtárra hivatkozik, akkor a gépbe épített CD-ROM meghajtó tartalmát használja.

Az egyes meghajtókat a fájlrendszerbe lehet illeszteni – ettől a pillanattól kezdve látszik a tartalmuk az adott könyvtárba –, valamint onnan el lehet venni. A fájlrendszerből eltávolított meghajtó tartalma nem látszik az adott könyvtárban.

A fontosabb meghajtók bekapcsoláskor a fájlrendszerbe automatikusan beépülnek, azokat onnan a felhasználó eltávolítani nem tudja. Ez

természetes, hiszen nélkülük a Linux nem lenne működőképes. A cserélhető médiával felszerelt meghajtókat azonban nem csak a rendszergazdának kell kezelnie, hanem a felhasználónak is, ezért meg kell ismernie ezeknek a használatát, vagyis képesnek kell lennie a rendszerbe építést és eltávolítást.

Cserélhető adathordozót általában a hajlékonylemez – vagy lemezek – és a CD-ROM használ. Ezeknek az eszközöknek a helye Linux alatt általában a /mnt könyvtárban található `floppy` és `cdrom` könyvtárak. A meghajtók beillesztésére a `mount`, kikapcsolására az `umount` szolgál.

```
[root@mad hw]# mount /mnt/floppy/
[root@mad hw]#
```

A `mount` parancs fenti formájában csak a rendszerbe illesztéshez használandó könyvtárat kapja meg a parancssorban[11]. A fenti parancs után a `/mnt/floppy` könyvtárban a hajlékonylemez tartalmát érhetjük el.

Általánosságban elmondható, hogy a mount paranccsal beépített meghajtókból az adathordozót kivenni nem szabad. Bizonyos eszközök esetén – pl. CD-ROM – a LINUX képes megakadályozni az eszköz kivételét elektronikus úton[12], de ez a mechanikus elven működő kivetőszerkezetekkel – pl. hajlékonylemez – nem így van. Ha a felhasználó a rendszerbe illesztett eszközből az adathordozót eltávolítja, azon a fájlrendszer sérülhet, a rajta található adatok elvesznek.

Mielőtt eltávolítanánk az adathordozót, a meghajtót tehát ki kell emelnünk az umount parancs segítségével a fájlrendszerből. Ez a `mount` parancshoz igen hasonló módon megy végbe:

```
[root@mad hw]# umount /mnt/floppy/
[root@mad hw]#
```

Ha tudni akarjuk, hogy milyen meghajtók vannak az állományrendszerbe építve, a `mount` parancsot opciók nélkül kell kiadnunk:

```
[root@mad hw]# mount
```

[11] A rendszergazda feladata, hogy a LINUX megfelelő információval rendelkezzen arról, hogy az adott könyvtárba milyen eszköz kerüljön beépítésre.

[12] A felhasználó hiába nyomja meg a CD-ROM lemezkiadó gombját, a lemez benn marad.

```
/dev/hda1 on / type ext2 (rw)
none on /proc type proc (rw)
/dev/hdc3 on /root type ext2 (rw)
/dev/hdc1 on /mnt/dosdisk type msdos (rw)
leila.jpte.hu:/home on /home type nfs (rw,addr=193.6.49.45)
/dev/sdb1 on /mnt/mobil1 type ext2 (rw)
[root@mad hw]#
```

A felsorolásban az on (vmin) szó után láthatjuk, hogy az egységek hova vannak beillesztve. A példában nem található /mnt/floppy, vagyis nincsen a rendszerbe építve a hajlékonylemez, az a gépből eltávolítható.

Az umount parancs sok esetben kudarcot vall, az alábbi hibaüzenettel figyelmeztetve:

```
[root@mad floppy]# umount /mnt/floppy/
umount: /mnt/floppy: device is busy
[root@mad floppy]#
```

A *device is busy* (meghajtó foglalt) üzenet jelzi, hogy a meghajtót nem lehet az állományrendszerből kivenni, mivel valamelyik program azt használja. Ez igen sokszor nem jelent fizikai használatot, csak annyit, hogy valamelyik futó feladatnak az aktuális könyvtára az adott eszközön van. A fenti példában is ez a helyzet, a pwd parancsot kiadva látható, hogy az umount parancsot a mágnezlemezen található könyvtárból adtuk ki. Ilyenkor elég az aktuális könyvtárat megváltoztatni és a parancsot ismételten kiadni:

```
[root@mad floppy]# pwd
/mnt/floppy
[root@mad floppy]# cd ..
[root@mad /mnt]# umount /mnt/floppy/
[root@mad /mnt]#
```

Előfordulhat, hogy a sok futó eljárás közt nem találjuk meg, melyik használja az adott meghajtót – pedig azt mindenképpen el szeretnénk távolítani. Ekkor az fuser parancs segítségével kereshetjük meg azt a feladatot:

```
[root@mad /mnt]# fuser /mnt/floppy/
/mnt/floppy/:          942c
[root@mad /mnt]#
```

Az `fuser` (*file user*, állomány felhasználója) a képernyőre nyomtatja azokat a feladatazonosítókat amelyek alapján a parancssorban megkapott állománynak vagy könyvtárnak a használatával elfoglalt programok megkereshetőek. A feladatazonosítók kezeléséről bővebben olvashatunk a ??? fejezetben olvashatunk.

2.5.2.16 A lemezfoglaltság

A `df` (disk free space, lemez szabad terület) parancs segítségével lekérdezhetjük az egyes lemezeken található szabad területeket. A parancs listát készít a rendszerbe illesztett meghajtókról és felsorolja, hogy mennyi a szabad terület rajtuk.

```
[root@pip man1]# df -k
Filesystem         1024-blocks  Used Available Capacity Mounted on
/dev/sda2            1087765  910021   121541     88%   /
/dev/sdb1            1908771 1318103   492013     73%   /home
/dev/sda3             592364   92574   469193     16%   /root
/dev/sdc1            1052064       0  1052064      0%   /mnt/mobil
/dev/sdd1            2043351  628720  1309007     32%   /mnt/mobil1
/dev/scd0             650774  650774        0    100%   /home/ftp/pub/cdrom
[root@pip man1]#
```

Az első oszlopban található az egyes meghajtók jele, majd utána azt olvashatjuk, hogy mennyi az adott egységen tárolható adatmennyiség. A „Used” fejléccel ellátott harmadik oszlop arról tájékoztat, hogy mekkora a foglalt terület nagysága a meghajtón. Az „Available” oszlop az üres – még felhasználható – lemezkapacitást mutatja. A „Capacity” oszlop százalékosan mutatja a terhelést, vagyis azt, hogy hány százalékban van kihasználva az adott meghajtó tárolókapacitása. A „Mounted on” feliratot viselő utolsó oszlop azokat a könyvtárakat sorolja fel, amelyekbe a meghajtó illesztve van (lásd 2.5.2.15 fejezet).

A -k opciónak köszönhetően az egyes értékek kilobyteokban vannak felsorolva.

2.5.2.17 Hajlékonylemez használata

Linux alatt a hajlékonylemezt használni általában nem minden felhasználónak van joga, ezért valószínű, hogy az e fejezetben található parancsok a rendszergazda előzetes hozzájárulását igénylik.

Mielőtt a hajlékonylemezt használnánk azt formázni szükséges[13]. A formázást az `fdformat` (*floppy disk format*, hajlékonylemez formázás) paranccsal végezhetjük el, mégpedig a kívánt formátumnak megfelelő eszközkezelőn (*device*, eszköz).

Néhány példa az eszközkezelők elnevezésére:

```
/dev/fd0d360
/dev/fd0h1200
/dev/fd1h720
/dev/fd1H1440
```

Az elnevezés mindig az „fd" -vel kezdődik, majd a meghajtó sorszáma következik – 0 vagy 1. Ez után a „h" (high density, nagysűrűségű írás) vagy „d" (double density, duplasűrűségű írás) betűk következnek. Ha a betű nagy – H vagy D – a lemez 3.5″ méretű, ha kicsi – h vagy d -, akkor a lemez 5.25″ méretű. Az elnevezés végén található szám a lemez kapacitását mutatja kilobyteokban.

A paranccsal pl. a következőképpen tudjuk a kisméretű lemezt 1.4 megabyteosra formázni – ha az az első lemezmeghajtóban van:

```
[root@mad mobil1]# fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
[root@mad mobil1]#
```

Látható, hogy a lemez formázás után ellenőrzésen esett át. Ezt az ellenőrzést a `-n` (*no verify*, ellenőrzés nélkül) opcióval kikapcsolhatjuk.

Amikor a lemezt formáztuk, akkor szükséges fájlrendszert (*file system*) létrehoznunk rajta. Erre – ha a Linux saját fájlrendszerét kívánjuk

[13] Ma már kapható a kereskedelemben előre formázott hajlékonylemez, amelyeket nem kell formázni a vásárlás után.

használni – az mke2fs *(make extended 2 filesystem*, extended 2 fájlrendszer létrehozása) parancs szolgál. A parancsnak csak a meghajtót kell megadnunk – ahol a fájlrendszert létre kívánjuk hozni –, mivel a hajlékonylemez a formázásnak köszönhetően tartalmazza a média típusát. Elég tehát az első és második hajlékonylemez-meghajtó közt választanünk a /dev/fd0 vagy /dev/fd1 használatával.

```
[root@mad mobil1]# mke2fs /dev/fd0
mke2fs 1.10, 24-Apr-97 for EXT2 FS 0.5b, 95/08/09
Linux ext2 filesystem format
Filesystem label=
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Writing inode tables: done
Writing superblocks and filesystem
accounting information: done
[root@mad mobil1]#
```

A fájlrendszert később bármikor ellenőrizhetjük az e2fsck (*extended 2 file system check*, extended 2 fájlrendszer ellenőrzése) parancs segítségével. Ha az e2fsck parancsot a -c opcióval indítjuk, akkor végigolvassa az egész hajlékonylemezt alapos fizikai ellenőrzést végezve.

```
[root@mad mobil1]# e2fsck -c /dev/fd0
e2fsck 1.10, 24-Apr-97 for EXT2 FS 0.5b, 95/08/09
Checking for bad blocks (read-only test): done
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
```

```
/dev/fd0: ***** FILE SYSTEM WAS MODIFIED *****
/dev/fd0: 11/360 files (0.0% non-contiguous), 63/1440 blocks
[root@mad mobil1]#
```

2.5.3 Szűrők

A szűrők olyan programok, amelyek a bemenetükön érkező adatokon valamilyen átalakítást végeznek automatikus módon.

2.5.3.1 Állományok eleje és vége

Az állományok első néhány sorát a `head` (fej), utolsó sorait pedig a `tail` (mese) parancs segítségével listázhatjuk szűrhetjük ki. Mindkét parancs rendelkezik -n opcióval, amelynek segítségével beállíthatjuk, hogy hány sornyi szöveget szeretnénk szűrni:

```
[root@pip /root]# tail -n 2 ~/test.tex
akármi, akárhol, akármikor \dots
\bye
[root@pip /root]#
```

Bár meglehetősen haszontalan dolognak tűnik az első – vagy utolsó – sorokat kiszűrni bárhonnan is, e két parancs igen hasznos lehet. Ha az utolsó néhány üzenetre vagyunk kíváncsiak, az első néhány névre a névsorban, vagy pl. ellenőrizni akarjuk, hogy a kötelező \bye parancsot nem felejtettük -e le egy TEXállomány végéről, használhatjuk e két parancs valamelyikét.

2.5.3.2 Rendezés

A `sort` (rendez) szűrő a bemenetére érkező szöveget soronként betűrend szerinti sorrendbe rendezi.

2.5.3.3 Keresés

A grep szövegben keres részleteket. Szétválasztja azokat a sorokat, amelyekben a keresett részlet megtalálható azoktól, ahol nem.

A grep alapesetben csak azokat a sorokat választja ki, amelyekben a keresett mintát megtalálta:

```
[root@pip /root]# ps aux | grep lyx
root       727  0.5 10.4  4936  4084  ?  S   00:10   0:07 lyx
[root@pip /root]#
```

A példában a grep bemenetére a ps a futó feladatokról küld információkat – minden sorban egy feladat jellemző tulajdonságait rendezve. A grep egyetlen parancssori opciót kap, azt a szót, amelyet keresnie kell. Amint látjuk a grep egyetlen sort választott ki, amely az egyetlen példányban futó lyx nevű program jellemzőt tárolja.

A példában szereplő kifejezést többször elindíthva könnyű produkálni a következő érdekes esetet:

```
[root@pip /root]# ps aux | grep lyx
root       727  0.4 10.5  4960  4108  ?  S   00:10   0:07 lyx
root       885  0.0  0.9   824   356  p1 S   00:38   0:00 grep lyx
[root@pip /root]#
```

Amint látjuk, a grep önmagát is megtalálta. Ez azért lehetséges, mert a csövekbe rendezett feladatok végrehajtását a Linux párhuzamosan hajtja végre. A fenti esetben amikor még futott a `ps` – amely a futó eljárások adatait listázza – már futott a grep – amely azokat szűri. A ps tehát listázta a grep parancs futására jellemző adatokat is, s mivel ebben is megtalálható a lyx szó, láthatjuk a képernyőn.

A `-v` (*revert*, ellentétes) opció hatására a grep csak azokat a sorokat engedi tovább, amelyekben a keresett minta nem található meg. A következő pédában a felhasználókról kérünk listát – a `who` parancs segítségével –, amelyből aztán kiszűrjük azokat a sorokat, amelyekben a `root` szó megtalálható:

```
[root@pip /root]# who | grep -v root
pipas    tty2     Jul 11 00:46
postgres tty3     Jul 11 00:46
[root@pip /root]#
```

A `-i` (*ignore*, figyelmen kívül hagy) opció hatására a `grep` nem veszi figyelembe a nagy- és kisbetűk közti különbséget, vagyis egyezésnek

veszi a mintát és az átvizsgált sorokat akkor is, ha a minta – vagy egyes részei – az egyikben nagybetűvel, a másikban pedig kisbetűvel jelenik meg.

2.5.3.4 Betűcsere

A `tr` (*translate*, lefordít) betűk cseréjére szolgál. A bemenetére érkező szöveges adatokban bizonyos karaktereket szisztematikusan kicserél, míg másokat változatlanul hagy.

Tegyük fel, hogy egy szöveges állományban a következő sor található:

```
akármi, akárhol, akármikor \dots
```

Ha a képernyőre listázásánál a tr parancsot használjuk, akkor kicserélhetjük az összes á betűt az angol nyelvben is megtalálható a karakterre:

```
[root@pip /root]# cat test.tex | tr á a
akarmi, akarhol, akarmikor \dots
[root@pip /root]#
```

Látható, hogy ebben az egyszerű esetben a tr opciójaként csak két karaktert kellett megadnunk, azt, hogy mit kívánunk kicserélni, s azt, hogy mire.

2.5.4 A Unix feladatvezérlő parancsai

A Unix rendszerek – így a Linux is – többfeladatos (*multitasking*) operációs rendszer, vagyis időosztásos (time sharing) módszert használva „egyszerre” több programot is képes futtatni. Az egyidejűség látszólagos, a felhasználó ugyanis nem látja, hogy valójában gyors váltásokkal osztoznak meg a futó programok a processzoron. Annak ellenére, hogy egy időpillanatban csak egy program vezérli a számítógépet, mi az egyidejűséget feltételezzük, hiszen a felhasználó szemszögéből vizsgáljuk a gép működését.

A Linux lehetőséget ad a felhasználónak az egyidőben futtatott programok vizsgálatára és befolyásolására, s ezeket a műveleteket feladatvezérlésnek (job control) nevezzük. Feladat minden elindított programpéldány, vagyis ha ugyanaz a program több példányban fut, akkor mindegyik példány külön feladatként jelenik meg.

A Unix minden futó feladathoz külön azonosítót rendel – feladat azonosító (process identificator, PID) –, hogy az egyszerre futó programpéldányok közt különbséget tudjon tenni. A feladat azonosító egy egész szám, amely a gép bekapcsoásakor 1 -ről indul és minden elindított feladat esetében eggyel növekszik.

2.5.4.1 Parancsok indítási módjai

A Linux többféle módot ad a programok indítására. A legegyszerűbb esetben a program az előtérben fut, vagyis amíg nem fejezi be futását, addig a felhasználó az adott parancssorban nem indíthat újabb programot. Az előtérben futó programpéldány birtokolja a billentyűzetet, vagyis a parancsorba beírt betűket feldolgozásra megkapja. A parancssor képernyőjét is az előtérben futó feladat használja, oda üzeneteket írhat[14]. Ha az előtérben kívánunk futtatni egy programot, akkor semmiféle különleges dolgot nem kell tennünk, az előző fejezetekben bemutatott módon kell elindítani programjainkat.

A feladatok futhatnak a háttérben is. Amikor egy programot a háttérben futtatunk, akkor a billentyűzetet azok nem ragadhatják magukhoz – azt újabb parancsok beírására használhatjuk. A felhasználó szemszögéből ez úgy nyilvánul meg, hogy a háttérben futó feladat végét nem kell megvárnia, az indítás után visszakapja a parancsra várakozást jelző parancskérő jelet (*prompt*, súgó).

A felhasználó az általa elindítani kívánt feladatot igen egyszerűen utasíthatja, hogy a háttérben fusson, ehhez nem kell mást tennie, mint egy & jelet tennie a begépel parancssor után. A következő példán egy másolást indítunk el a háttérben:

```
[root@mad /root]# cp  *.txt /mnt/mobil1/&
[1] 743
[root@mad /root]#
```

Láthatjuk, hogy a parancssor a képernyőre nyomtatott két számot, majd azonnal visszadta a parancskérő jelet. A két szám közül az első

[14] Különleges esetben – sürgős üzenet, hibajelzés, stb – előfordulhat, hogy a parancssor képernyőjére más – a háttérben futó – program is betűket küldhet.

– amelyik szögletes zárójelek közt van – az adott parancssor saját feladatazonosítója, míg a második az egész rendszeren érvényes egyedi feladatazonosító.

A felhasználó most folyatathatja a munkát, vagyis újabb parancsokat gépelhet be, s ezzel újabb – az előtérben vagy a háttérben futó – feladatokat indíthat el. Amikor a másolás – amely az 1 parancssori azonosítót kapta – befejeződik, a parancssor üzenetet küld a felhasználónak.

```
[1]+  Done                    cp -i *.txt /mnt/mobil1/
[root@mad /root]#
```

A befejezés vizsglatát a parancssor az előtt végzi, mielőtt a képernyőre újabb parancskérő jelet küldene, ezért – amikor már befejeződött a háttérben futó feladat – a felhasználó addig nem kap értesítést, amíg az Entert le nem nyomja.

Ha a parancs elé a nohup (no hangup) kulcsszót tesszük, akkor az indított program futása nem szakad meg a kilépéssel.

Parancsok indításának újabb módja a time előtag alkalmazása:

```
[root@mad saved]# time ls
grab.c   grab1.c  grab2.c
0.00user 0.00system 0:00.21elapsed 0%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (98major+18minor)pagefaults 0swaps
[root@mad saved]#
```

Ha a parancs elé a time kulcsszót írjuk, akkor a parancssor az indított program befejezése után statisztikai adatokat ad az feladat által felhasznált erőforrásokról.

Műveletek a parancssor feladatazonosítójával Lehetősége nyílik a felhasználónak arra is, hogy előtérben futó feladatot a háttérbe folytatódjon és fordítva.

Az előtérben futó programot be kell fagyasztani ahhoz, hogy a begépelt parancs segítségével a háttérbe lehessen irányítani[15]. Ezt a befagyasztást a Ctrl+z kombinációval lehet:

[15] Pusztán azért van erre szükség, hogy felhasználó által leütött billentyűk ne az előtérben futó feladathoz kerüljenek.

```
[root@mad /root]# medit

[1]+  Stopped                 medit
[root@mad /root]#
```

Látható, hogy amikor megnyomjuk a Ctrl+z kombinációt, akkor üzenetet kapunk, arról, hogy a parancsor milyen belső feladatazonosítót rendelt a befagyasztott feladathoz. Ezt a belső azonosítót felhasználhatjuk arra, hogy a feladatot – az előtérben vagy a háttérben – továbbindítsuk. Az előtérben az fg (foreground, előtér), míg a háttérben a bg (background, háttér) paranccsal folytathatjuk a befagyasztott feladatot.

```
[root@mad /root]# bg %1
[1]+ medit &
[root@mad /root]#
```

Látható, hogy az fg és bg parancsok után a belső feladatazonosítót % jellel kell bevezetni.

A belső feladatazonosítókat bármikor lekérdezhetjük a jobs parancs segítségével:

```
[root@mad /root]# jobs
[1]   Running                 medit &
[2]-  Running                 xterm &
[3]+  Stopped                 find / -name magyar.kyb -print
[root@mad /root]#
```

A jobs parancs kinyomtatja a képernyőre a feladatok állapotát és a parancssor belső feladatazonosítóját.

A parancssor belső feladatazonosítójáról szóló rész végén szeretnénk felhívni a figyelmet arra, hogy ezek az azonosítók minden parancssor esetében egyediek, így tehát nem alkalmasak a más parancssorban indított feladatok kezelésére. A jobs mindig csak az adott parancssorból indított feladatokat listázza.

2.5.5 Kommunikáció más felhasználókkal

E fejezetben azok a Unix parancsok kerülnek sorra, amelyek a rendszeren párhuzamosan dolgozó felhasználókat kapscolják össze egymással.

Mivel a Unix rendszerek már a kezdetektől többfelhasználósak (multi user), a felhasználók közti kommunikáció mindig is része volt a Unixnak és így a Linuxnak is. Igaz ugyan, hogy a számítástechnika a Unix kezdetei óta a döbbenetes mértékű árcsökkenés miatt lehetővé tette, hogy minden felhasználó saját számítógépet kapjon, a Linux ma is támogatja a több felhasználó egyidejű munkáját. A felhasználók karakteres terminálról vagy – számítógéphálózaton keresztül – más számítógépről kapcsolódhatnak a közös géphez és ott programokat futtathatnak.

Eredetileg a közös gépet használók közti kommunikációs lehetőség csak mellékszolgáltatás volt, mára azonban sokak kizárólag azért haszználnak számítógépet, hogy más felhasználókkal kapcsolatba kerülhessenek. A Unix számításokat végző eszközből sok helyütt kommunikációs eszköz lett, éppen úgy, mint a rádió vagy a televízió. Ennek nyilvánvalóan az Internet előretörése és a Unix rugalmas felépítése volt az oka.

A Unix – s így a Linux is – a felhasználók közti kommunikációnak két alapvető formáját ismeri. Az egyszerűbb esetben az egymással kapcsolatot kereső felhasználók egyazon gépen – mint programgazdán (application server) – dolgoznak, a köztük felépített adatcsatorna tehát nem lépi át az operációs rendszer határait.

Bonyolultabb a helyzet akkor, ha a felhasználók más-más gépen dolgoznak, és a számítógéphálózatot felhasználva kívánnak egymással kapcsolatot kiépíteni. Természetesen van lehetőség erre is, figyelembe véve két tényezőt, amely a kapcsolatot bonyolítja, mégpedig a) távoli gép esetén számítanunk kell arra, hogy az ellenállomás nem Unix (Linux) operációs rendszert használ, és b) szükségünk van a gép azonosítására is, hiszen a kommunikációs csatorna a saját gépünkön kívülre mutat. A két kommunikációs módszert a ??? ábra mutatja be.

A Linux által használt kommunikációs parancsok egy része csak a belső kommunikációt támogatja, más része pedig kifelé is működik. E fejezetben azokat a parancsokat mutatjuk be, amelyek tipikusan a belső kommunikáció során használatosak. A külső kommunikáció esetén hasznos programokat a 4 fejezet megfelelő részeiben találhatjuk meg. Azoknál a parancsoknál, ahol lehetőség van mind a belső, mind pedig a külső kapcsolat létrehozására, azt külön jelezzük.

A kommunikáció ilyen jellegű szétválasztását azért választottuk, hogy az egyébként nagy érdeklődésre számot tartó felhasználók közti kapcsolat megismeréséhez ne legyen szükség a számítógéphálózatok megis-

merésére.

2.5.5.1 Felhasználók listája

A gépen dolgozó – „belépett” – felhasználók listáját a who parancs segítségével kérdezhetjük le:

```
[pipas@pip /root]$ who
root     tty1     Jun  7 22:41
root     tty2     Jun  7 23:14
root     ttyp0    Jun  7 22:51 (:0.0)
root     ttyp2    Jun  7 22:47 (:0.0)
root     ttyp7    Jun  7 23:42 (:0.0)
[pipas@pip /root]$
```

A parancs által a képernyőre nyomtatott lista első oszlopa tartalmazza a belépett felhasználók azonosítónevét (*login name*)[16]. Látható, hogy amennyiben a felhasználó több kapcsolatot is felépített a rendszerrel, akkor a listában többször is szerepel. A több példányban létező felhasználók közt különbséget a második oszlop felhasználásával tehetünk, vagyis az itt található *terminál azonosító* mutatja meg, hogy honnan használja a felhasználó a rendszert. Az utolsó oszlop is azonosításra szolgál, innen olvashatjuk le, hogy *melyik távoli gép*ről jelentkezett át a felhasználó. Fontos felfigyelnünk a második és utolsó oszlop közti különbségre. Amíg az utolsó oszlop egyazon felhasználó esetén is ismétlődhet, addig a második nem, hiszen azt a Linux éppen a kapcsolat azonosítása érdekében generálta. A második oszlopban látható terminálazonosító tehát akkor is különbséget tesz két kapcsolat között, ha azokat ugyanaz a felhasználó hozta létre ugyanazon távoli gépről.

A harmadik oszlop mutatja meg, hogy az adott felhasználó mikor épített ki kapcsolatot. Amennyiben a kapcsolat már régóta kiépült, félő, hogy a felhasználó valójában nem foglalkozik a képenryővel – neki üzenni nem érdemes. Természetesen a who parancs nem tudja megállapítani, hogy a monitor előtt ülő felhasználó mit csinál éppen, azt viszont megtudhatjuk a segítségével, hogy mikor adta az utolsó feladatot:

[16] Itt jegyezzük meg, hogy a Linux nem tartalmaz mesterségesen beépített korlátot az egyszerre kiszolgálható felhasználók számának tekintetében. Ezt sok kereskedelmi program megteszi, ezzel is védve az operációs rendszer készítőjének üzleti érdekeit.

```
[root@pip /root]# who -iH
USER     LINE     LOGIN-TIME   IDLE  FROM
root     tty1     Jun  7 22:41 00:04
root     tty2     Jun  7 23:14 00:05
root     ttyp0    Jun  7 22:51 00:57 (:0.0)
root     ttyp2    Jun  7 23:44   .   (:0.0)
root     ttyp7    Jun  7 23:42 00:18 (:0.0)
[root@pip /root]#
```

A -i opció hatására megjelenik egy újabb oszlop, amely a haszontalan – lazsálással töltött – időt (*idle time*) mutatja s amelyből megtudhatjuk mióta nem dolgozik felhasználó az adott végponton. A példában szereplő -H hatására a who a lista főlé magyarázó fejlécet nyomtatott.

A who speciális felhasználása a saját név lekérdezése:

```
[root@pip /root]# who am i
pip.jpte.hu!root     ttyp2    Jun  7 23:44 (:0.0)
[root@pip /root]#
```

Látható, hogy a who am i forma használatával az első oszlopban a számítógép neve is kinyomtatásra került.

A who aparancs nem alkalmas távoli gépen dolgozó felhasználók listázására, arra a 4.4.3 fejezetben bemutatásra kerülő finger parancs szolgál.

2.5.5.2 Információkérés más felhasználóról

Amennyiben meg akarjuk tudni, hogy egy felhasználó mely csoportok tagja, a groups parancsot kell használnunk. A groups a parancssorban a felhasználói nevet várja, s kinyomtatja a felhasználói név után – kettősponttal elválasztva – az összes csoportot, amelynek az adott felhasználó tagja:

```
[root@pip /root]# groups pipas
pipas : pipas friends
[root@pip /root]#
```

2.5.5.3 Üzenet küldése más felhasználónak

2.5.5.4 Beszélgetés más felhasználóval

2.5.6 A BASH programozása

Amikor egy szöveges állományt futtathatóvá teszünk (fejezet) és elindítjuk, a parancssor megkísérli kiderteni, hogy milyen programmal kívánjuk a szöveg értelmezését elvégezni.

2.5.6.1 A BASH változói

A BASH lehetőséget ad arra, hogy a felhasználó változókat vezessen be, amelyek névvel rendelkeznek és valamilyen aktuális értéket hordoznak. Változók létrehozására minden gépi nyelvben vannak lehetőségek – nélkülük a munka elképzelhetetlen volna –, ezért amikor BASH számára készítünk programokat, fontos megismernünk a változók kezelését is.

Szükséges a változók megismerése akkor is, ha nem kívánunk BASH programokat készíteni, mivel igen sok program a beállításait adott nevű változók értékeiből veszi.

A BASH változói mindig szövegesek – karaktersorozatként kerülnek feldolgozásra – és konvencionálisan mindig nagybetűvel írjuk őket.

Változókat bevezetni az értékadással lehet, amelynek legegyszerűbb fomája az egyenlőségjellel írható:

```
PS3="   Menu: "
```

Az egyenlőségjel bal oldalán található a változó neve – amely mint említettük szokás szerint csupa nagybetűből áll –, jobb oldalán pedig az értéke. Amennyiben a változó értéke nem tartalmaz szóközöket vagy különleges jeleket, az idézőjelek elhagyhatóak:

```
SZ=nincsszokoz
```

Változók értékére hivatkozni a változónév elé tett $ karakterrel lehet. A BASH minden helyen, ahol a változó neve előtt $ jelet talál, behelyettesíti a változó aktuális értékét. A következő példában a echo parancs az SZ változó aktuális értékét a képernyőre nyomtatja.

```
echo $SZ
```

Lehetőségünk van arra is, hogy a változónak a futás közben adjunk értéket a programot elindító felhasználó megkérdezésével. Erre a read (olvas) parancs alkalmas, mely után a változó nevét kell megadnunk[17].

read TEL

hoppla: read -p

A read parancs hatására a képernyőn megjelenik egy villogó kurzor, és a BASH megvárja amíg a felhasználó begépeli a változó új értékét, majd megnyomja az Enter billentyűt. A begépelendő adat jellegéről értesítést nem küld, ezért azt előzetesen kell a képernyőre nyomtatnunk. Erre használható az echo parancs, legtöbbször a -n opcióval, amely az echo számára azt jelenti, hogy a szöveg kinyomtatása *után* nem kell új sort kezdenie. A kérdés tehát általában a következő formában fogalmazható meg a BASH nyelvén:

```
echo -n "Adja meg a nevet: "
read NEV
echo -n "A kovetkezo nevet adta meg: "
echo $NEV
```

A fenti utasítások futtatásakor a következő képernyőképet kapjuk:

```
Adja meg a nevet: Zaphod Beeblebrox
A kovetkezo nevet adta meg: Zaphod Beeblebrox
```

Jól látható, hogy a BASH a felhasználó által begépelt teljes szöveget beteszi a változóba, függetlenül attól, hogy az hány szóból áll.

A BASH a változókat általában csak abban a programban tartja meg, amelyekben azokat létrehoztuk, még az adott programból indított újabb programoknak sem adja át azokat. Az export parancs segítségével előírhatjuk, hogy a programból indított újabb programok megkapják a változó értékét.

[17] Figyeljük meg, hogy mivel a read parancsnak a változó nevét kell ismernie, nem pedig az értékét, a név előtt nem áll $ karakter!

2.5.6.2 A feltételes utasításvégrehajtás

2.5.6.3 A kiválasztás

2.5.6.4 A hurok

2.5.6.5 Az elöltesztelő ciklus

2.5.6.6 A hátultesztelő ciklus

2.5.6.7 A menü

2.5.7 A BASH testreszabása

2.5.7.1 Ékezetes betűk

A BASH alapállapotban nem fogadja el a magyar ékezetes karaktereket, ami abban jelentkezik, hogy a lenyomott billentyű hatására nem jelenik meg semmi a képenryőn. A z ékezetek használatához a felhasználónak létre kell hoznia egy .inputrc nevű állományt a saját könyvtárába és abba a következő sorokat beírnia:

```
set meta-flag on
set convert-meta off
set output-meta on
```

A parancssor a következő indításkor már helyesen fogja kezelni az ékezetes karaktereket.

Fejezet 3

Állománykezelő programok

3.1 A Midnight Commander

3.2 X file manager

Fejezet 4

Számítógéphálózatok

4.1 Általános ismeretek

Az erőforrások – számítógépes eszközök – megosztása érdekében sokszor előnyös a számítógépeket hálózatba kapcsolni. Így a felhasználó több helyről is elérheti adatait, programjait, más felhasználókkal közösen dolgozhat, azokkal kommunikálhat, távoli perifériákat – nyomtatókat, háttértárakat, egyéb eszközöket – használhat.

Amikor számítógéphálózatokról beszélünk, akkor tudnunk kell, hogy igen bonyolult, komplikált felépítésű műszaki berendezésekről van szó, amelyekről az átlagos felhasználónak csak néhány alapfogalomat kell ismerni.

A kommunikáció mindig valamilyen *fizikai eszközön* történik – legyen az telefonvonal, optikai kábel, rádió adó vevő, vagy akár mikrohullámú lánc – és mindig valamilyen *kommunikációs szabvány* szerint megy végbe.

Az átviteli csatornának a felhasználó számára csak egyetlen tulajdonsága fontos, nevezetesen a csatorna sebessége – vagy más szóval sávszélessége. A sebességet az egységnyi idő alatt átvitt információval jellemezzük, figyelembe véve azt, hogy általában több felhasználó szimultán használja a csatornát, így egyikük sem használhatja az átviteli csatornát teljes sebességgel. A szimultán használatból következik, hogy csúcsidőben – amikor sokan használják a számítógéphálózatot – lassabb

adatátvitelt tapasztal a felhasználó, míg azokban az időpontokban, amikor vélhetőleg kevesen használják számítógépeiket, az átvitel meggyorsulni látszik.

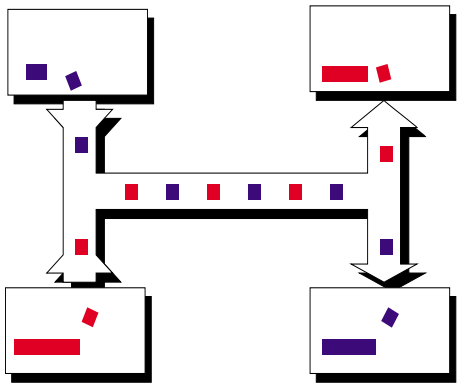
A kommunikációs szabvány az a kommunikáció formáját leíró szabvány vagy ajánlás, amely lehetővé teszi, hogy más-más gyártótól származó programok egymással kommunikálni tudjanak. Mint látni fogjuk a LINUX sokféle szabvány szerinti kommunikációra képes – összekapcsolódhat Netware kiszolgálókkal, Macintosh számítógéppel, lehet része egy Microsoft Networknak – de saját kommunikációs rendszere az IP (*Internet Protokoll*). Nem csak, hogy támogatja az Internetet tehát, hanem egyenesen az a saját protokollja, belső rendszere, amelyet sokszor még akkor is használ, ha nincsen hálózatba kötve.

Az informatikában nyílt rendszereknek nevezzük azokat a megoldásokat, amelyek belső felépítésükből fakadóan minden lehetőséget megadnak a moduláris átszervezésre. Ez leginkább a hálózatok használatakor igen hasznos, hiszen azt eredményezi, hogy mindent amit a gép előtt ülve megtehetünk, megtehetjük hálózaton keresztül, távoli eléréssel is. A LINUX esetében nincsen olyan program, amelyet megfelelő eszközökkel ne tudnánk hálózatot használva távolról is működtetni, és ez a nagyszerű lehetőség káprázatos hatalommal ruházza fel a LINUX felhasználóit.

Gondoljunk csak bele: a világ egy másik nagyvárosából belépünk munkahelyi gépünkre, átjavítjuk éppen készülőben levő cikkünket, kinyomtatjuk azt ottani nyomtatónkon, hagyunk egy üzenetet kollégánk levelesládájában, hogy mit tegyen a nyomtatóból éppen előkúszó papirokkal, majd hálából lejátsszuk neki kedvenc együttese egyik albumát az asztalunkon található hangszórókon. Mindezt csak nyílt rendszereket használva tehetjük meg, amelyek nem korlátozzák a távoli eléréssel felépített kapcsolatok lehetőségeit.

4.2 Csomagkapcsolt hálózatok

Számítógéphálózatok fizikai megvalósítására ma már szinte kivétel nélkül csomagkapcsolt hálózatokat használunk. A „csomagkapcsolt” kifejezés arra utal, hogy a hálózaton átvinni kívánt adatokat mindig csomagokra bontjuk, majd minden csomag önállóan – a többi csomagtól függetlenül – kerül átvitelre. Ez annak érdekében történik, hogy ugyanazt az átviteli



ábra 4.1: Az időosztásos módszer

csatornát időosztásos (*Time Sharing*) módszerrel látszólag egyidőben többen is használhassák (4.1 ábra).

4.3 Az Internet Protokoll

Amikor két helyi érdekeltségű számítógéphálózatot összekapcsolunk, akkor internetről, globális hálózatról szokás beszélni. Az amerikai nemzetvédelmi kutatásokat végző központ (Advanced Research for ?????) volt az első, aki ilyen globális hálózatot kezdett kiépíteni, miközben legfontosabb feladatának egy atomcsapás esetén is részlegesen működőképes informatikai hálózatot próbált kifejleszteni.

Az elkészült hálózatra egyre több egyetemet – majd később már más intézményeket és magánszemélyeket is – kapcsoltak be, mígnem eljutottunk egy az egész világot magába foglaló rendszerhez. Az így felépített világhálózatot a megkülönböztetés érdekében nagy kezdőbetűvel „Internetnek” kezdték emlegetni.

Az Internet tehát egy olyan világméretű szabványos számítógéphálózat, ameleyet az amerikai nemzetvédelmi hivatal és a felsőoktatás fejlesztett ki, de ma már az egyes részei más-más szervezetek kezelésében és tulajdonában vannak. Az IP (Internetworking Protocoll, Számítógéphálózatok közti kommunikációs szabvány) az Internet alapjait írja le és mint

ilyen a LINUX alapjait is adja, ezért – ha csak felületesen is – a felhasználónak ismernie kell.

4.3.1 A számítógépek azonosítása

Minden IP kommunikációra képes számítógép rendelkezik egy egyéni számmal, amely azonosítja őt a hálózaton. E szám – az internet cím – négy számjegyből áll, ahol minden számjegy egy byteon kerül tárolásra – tehát értéke 0 és 255 között kell, hogy legyen[1]. A számjegyeket pontokkal elválasztva írjuk, valahogy így:

```
193.6.62.56
```

Amikor a LINUX átvitelt kezdeményez egy másik számítógéppel, akkor minden csomagot ellőt a saját Internet címével, sőt ráteszi a címzett azonosítószámát is, hogy az átvitel során mindig világos legyen a csomag útja.

Ha olyan számítógéppel kezdeményezünk átvitelt, amely nem a közelünkben – nem a belső hálózaton – van, akkor a LINUX a csomagot nem közvetlenül küldi el, hanem az útválasztónak (*router*) továbbítja a megfelelő címmel ellátva. A rendszergazda feladata, hogy az útválasztó minden körülményekl közt továbbítani tudja a csomagot[2]. Az információ így, csomagokra bontva útválasztóról útválasztóig haladva végül – ha lehetséges – eljut a céljához.

A számítógépeknek ilyen módon való azonosítása műszakilag igen jó rendszer, de kezelni nehéz. A felhasználótól nem várható el, hogy bonyolult számokat jegyezzen meg, ezért létezik egy „emberi fogyasztásra alkalmas” elnevezési rendszer is. Ebben a rendszerbe minden internet címhez egy nevet rendelünk hozzá. Ilyen – könnyen megjegyezhető – név pl.:

```
beeblebrox.marketing.lzsb.hu
```

Az elnevezés hierarchikus rendszert követ, a pontokkal ellátott részek egyre nagyobb és nagyobb szervezeti egységeket jelölnek, jelen példában:

[1] Valójában 0 és 255 speciális jelentéssel van felruházva, így azok általában nem használhatóak gépek azonosítására.

[2] Fontos megjegyeznünk, hogy a Linux az útválasztó szerepét is betöltheti egy vagy több belső hálózat összekapcsolásával.

- hu Magyarországot jelöli.
- lzsb A Lekvárzsibbasztó Gépgyár megjelölése.
- marketing A gyár egy szervezeti egysége.
- beeblebrox A gép fantázianeve[3] .

A hierarchikus felépítésnek köszönhetően könnyű megjegyezni a neveket, ráadásul a nevek többszöri felhasználását könnyen elkerülhetjük, hiszen minden egységnek csak (Ország, intézmény, csoport, stb.) csak arról kell gondoskodnia, hogy saját területén belül ne használja fel ugyanazt a nevet többször[4].

A nevek ilyen használata esetén az elsőtagot - esetünkben beeblebrox - szokás hostnévnek (*host*, gazda), a fennmaradó névsorozatot - itt marketing.lzsb.hu - körzetnévnek (*domain*) nevezni. Nyilvánvalóan minden körzetnek kell, hogy legyen rendszergazdája, aki a nevek kiosztását és internet címek - számok - hozzárendelését végzi, gondosan ügyelve arra, hogy ugyanazt a nevet vagy címet ne kaphassa meg több gép is.

Az Interneten létezik egy szolgáltatás, amely a nevek és a hozzájuk tartozó címek tárolására szolgál. Ez a DNS (*Domain Name Service*, Körzeti Névszolgáltatás), teszi automatikussá a nevek „feloldását” címekké. Amikor a megfelelő helyen egy névre hivatkozunk, akkor számítógépünk automatikusan megkeresi a DNS kiszolgálóját és megtudakolja tőle, hogy az adott névhez milyen szám tartozik, majd a továbbiakban ezt használja[5]. Ha nincsen DNS kiszolgálónk, vagy az nem működik, akkor a gépekre csak számukkal hivatkozhatunk, mivel a LINUX nem képes „feloldani” a nevet számmá, nem tudja megállapítani mi a címe az adott nevű gépnek.

Amikor egy gépet távolról megkeresünk adatátviteli célból, akkor az képes arra, hogy a csomagokon található Internet címünkből a gépünk nevét visszakeresse. Erre e Reverse DNS (visszfelé történő névkeresés) szolgál. Erre tulajdonképpen nem volna szüksége, mert a választ az Internet szám ismeretében el tudja küldeni, jó lehetőség ez azonban

[3] Zaphod Beeblebrox kitalált személy, Douglas Adams egyik figurája.

[4] Természetesen lehet beeblebrox nevű számítógépe a finnországi illetőségű Jam and Juice műveknek is.

[5] A Linux képes DNS kiszolgálóként működni, vagyis más gépek számára a neveket nyilvántartani és továbbítani.

a küldő ellenőrzésére. Ha a küldő nem szrepel a DNS azonosításba, akkor sok gép - a reverse DNS folyamat után - a kapcsolatot megszakítja, mivel nem bízik kellőképpen a küldőben. Ebből következik, hogy gépünketmég akkor is be kell jegyeztetnünk a körzet rendszergazdájával, ha esetleg nem akarunk nevet. A bejegyzéssel tulajdonképpen hivatalosan is teljes értékű tagja lesz a gép az Internetnek.

Fontos megemlítenünk azt is, hogy léteznek olyan számítógépek amelyek nem rendelkeznek állandó Internet címmel, hanem azt dinamikusan kapják meg minden bekapcsoláskor. Erre a DHCP (*Dinamic Host Control Protocoll*) teszi alkalmassá a LINUXot[6]. Ha valami okból ezt a módszert használjuk, akkor a számítógép bekapcsoláskor Internet címet kap, amelyet kikapcsolásig használ, de a következő bekapcsoláskor már esetleg teljesen más néven - és címen - kapcsolódik az Internethez. Ezt a módszert általában nem nevesített gépek esetében használjuk, amikor sok, teljesen egyenrangú gépről van szó, amelyek közt különbséget tennünk felesleges - például többmunkahelyes laborokban. A rendszer előnye az, hogy újabb számítógépek különösebb munka nélkül kapcsolhatóak a hálózathoz.

4.3.1.1 Zárt belső hálózatok

Bizonyos esetekben szükséges lehet zárt belső hálózatok kialakítására. Ezt megteheti a rendszergazda biztonsági okokból, ha a célja a külső számítóges támadások kivédése, ekkor tűzfal (*firewall*) rendszerről beszélünk.

kép

A tűzfal olyan számítógép, amely igen szigorú biztonsági megkötésekkel és ellenőrzés mellett végzi munkáját. A tűzfal a rendszergazda feszült figyelmének kereszttüzében felügyeli az egész belső hálózat összes külső kommunikációját és megkísérli a támadások kivédését. Bizonyos szolgáltatásokat esetleg eleve lehetetlenné tesz - korlátozva a külső vagy belső felhasználókegyes tevékenységeit, esetleg adott körzetekből megtagadja az elérést. Tipikusan akkor tagadja meg egyes körzetekből a belső hálózat elérését, ha onnan már a rendszergazda betörési kísérletet tapasztalt[7].

[6] A Linux alkalmas DHCP segítségével dinamikusan Internet címeket kiosztani is, ilyenkor DHCP kiszolgálóként működik.

[7] A biztonsági rendszerekről a továbbiakbanmég olvashatunk.

Hasonló technikára (*IP Masquerading*, Jelmezes, álarcos Internet) akkor is szükség lehet, ha a biztonság a leválasztást nem követeli ugyan meg, de a rendszergazda nem rendelkezik a megfelelő számú Internet címmel. Mint láttuk minden gépnek egyedi Internet számra van szüksége amely - hogy ne használják sehol e kerek világon máshol - amelyet felettes szervezet osztja ki. Mivel a lehetséges Internet számok halmaza véges, előfordulhat, hogy nem kapunk annyi címet ahány gépet üzemeltetni akarunk. Ebben az esetben a zárt belső hálózat olyan Internet címeket használ, amelyek a hálózatba kifelé titkosak. A masquardingot végző gép „becsapja” a külső hálózat összes számítógépét, mivel úgy tesz, mintha a teljes belső hálózat adatforgalma az ő Internet címén történne. A kifelé haladó csomagokat a saját címével hamisítja - mivel csak neki van kifelé is érvényes Internet címe - a befelé jövő csomagokat pedig saját belső nyilvántartásának megfelelően a megfelelő cél felé küldi.

A LINUX képes masqueradingot végezni más gépek számára és képes ilyen környezetben dolgozni. A masqueradingról elmondottak megmutatják, hogy nem igaz a legenda, miszerint az Internet címek szabványa korlátozná az Internet lehetséges méretét ezért annak fejlődése a közeljövőben megtorpanna.

4.3.2 Hálózati adatfolyamok azonosítása

Az előző részben láttuk, hogyan történik a gépek azonosítása. Ez azonban még kevés, hiszen egyszerre több szálon is futhat kommunikáció egy gép felé. Előfordulhat, hogy több géppel állunk kapcsolatban egyszerre, ezért szükség ezeknek a párhuzamosan haladó kommunikációknak az egymástól való elválasztására. Erre az Interneten a port (portál, kirakat) került bevezetésre.

Minden kommunikáció amely két Internet Protocollt használó gép közt jön létre, kap két számot, amelyek látszólagos kapcsolódási pontok a gépen belül. A párhuzamosan haladó adatfolyamokat tehát nem csak a két gép címe, hanem a forrás és cél port száma is azonosítja. Mivel más jellemző alapján nem lehet a bejövő adatcsomagokat szétválogatni, ugyanazon két gép két portja közt mindig csak egy átvitel lehet folyamatban.

kép.

A portok intézménye egy praktikus további feladatot is ellát, nevezetesen a számítógép abból tudja kitalálni, hogy mi a célja az adatátvitelt kezdeményező ellenállomásnak, hogy az melyik portjára küldi az adatokat. Amikor a 25 -ös porton keresztül adatokat kap, akkor például azonnal tudja, hogy elektronikus levelet akar küldeni egy másik gép, tehát a csomagokat a levéltovábbító alrendszernek adja át. A portoknak a funkcióhoz rendeléséről szabvány rendelkezik, amely leírja melyek azok a portok, amelyek speciálisan eleve bizonyos alrendszerekhez kapcsolódnak.

4.3.3 Egy kapcsolat felépítése

Összefoglalásul lássuk mit tesz a gép, amikor pl. Internet kapcsolatot akar kiépíteni egy elektronikus levél továbbítására:

1. Megtudakolja a DNS kiszolgálótol, hogy mi az adott nevű gép Internet címe.
2. Kiválaszt egy saját portot, amelyet nem használ semelyik alrendszer vagy már élő kapcsolat, ezt lefoglalja a kommunikáció idejére.
3. Ha a célgép belső hálózaton van, akkor a megszólítandó gépet közvetlenül keresi fel, egyébként pedig az útválasztót, tűzfalat vagy a masqueradingot végző gépet.
4. Jól tudja, hogy a másik gép szabvány szerint a 25 -ös porton várja az elektronikus leveleket, ezért azt csomagokra bontva erre a portra irányítva elküldi a célgép fellé.
5. Minden csomagról visszajelzést kap - az előzőleg általa lefoglalt belső portra -, ha az nem érkezne meg időben, a csomagokat megismétli.
6. Amennyiben a csomagokra rendre nem jönnek válaszok, értesítést küld a felhasználónak, vagy az átvitelt a későbbiekben - esetleg napok múlva is - újra megkísérli.

Látható, hogy az adatátvitel még ilyen vázlatosan is meglehetősen bonyolult művelet. Nem szabad azonban elfelejtenünk, hogy az átvitel szabványosításánál a hibatűrés volt az elsődleges szempont, vagyis az,

hogy néhány eszköz meghibásodása ne vonja maga után az egész Internet működőképtelenségét.

4.4 A LINUX hálózati programjai

E fejezetben azok a legegyszerűbb programok kerülnek bemutatásra, amelyek a LINUX - és más UNIX rendszerek - felhasználói számára a hálózat elérését lehetővé teszik. Nem szerepelnek itt azok a nagyméretű - általában grafikus *alkalmazások*, amelyek a hálózaton való munka és szórakozás eszközei.

4.4.1 Keresés a DNS adatbázisban

Az `nslookup` (*name service lookup*, névszolgáltatás keresése) segítségével kideríthetjük, hogy egy adott Internet címhez milyen elnevezés tartozik. Az nslookup megszólítja a DNS kiszolgálót, majd megtudakolja tőle a beírt Internet cím hivatalos nevét és ezt a képernyőre kiírja.

```
[root@mad bin]# nslookup ftp.kfki.hu
Server:  leila.jpte.hu
Address:  193.6.49.45

Non-authoritative answer:
Name:    sunserv.kfki.hu
Address:  148.6.0.5
Aliases:  ftp.kfki.hu

[root@mad bin]#
```

Az nslookup képes a névből a címet kideríteni, vagyis fordított keresést végezni a DNS szerveren.

```
[root@mad bin]# nslookup 148.6.0.5
Server:  leila.jpte.hu
Address:  193.6.49.45

Name:    sunserv.kfki.hu
```

```
Address:  148.6.0.5

[root@mad bin]#
```

4.4.2 A ping

A ping igen egyszerű program, neve az asztalitenisz közismert elnevezéséből származik. A ping akkor használatos, ha egy távoli gépről meg akarjuk tudni, hogy az elérhető -e.

Ilyenkor a ping segítségével csomagokat küldünk a számítógép felé, amelyekre az egyenként válaszol.A ping megmutatja, hogy visszaérkeztek -e a csomagok, esetleg azokból mennyi veszett el hiba során és milyen gyors volt a válasz, vagyis hány ezredmásodperc alatt tették meg a csomagok az utat oda-vissza.

```
[root@mad bin]# ping ftp.kfki.hu
PING sunserv.kfki.hu (148.6.0.5): 56 data bytes
64 bytes from 148.6.0.5: icmp_seq=0 ttl=247 time=35.1 ms
64 bytes from 148.6.0.5: icmp_seq=1 ttl=247 time=37.8 ms
64 bytes from 148.6.0.5: icmp_seq=2 ttl=247 time=42.5 ms
64 bytes from 148.6.0.5: icmp_seq=3 ttl=247 time=31.3 ms
64 bytes from 148.6.0.5: icmp_seq=4 ttl=247 time=27.7 ms
64 bytes from 148.6.0.5: icmp_seq=5 ttl=247 time=75.2 ms

--- sunserv.kfki.hu ping statistics ---
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 27.7/41.6/75.2 ms
[root@mad bin]#
```

A ping folyamatosan küldi a csomagokat addig, amíg a felhasználó le nem nyomja a Ctrl+c billentyűzetkombinációt. Ekkor kiszámolja a statisztikai adatokat a küldött és visszakapott csomagokra, majd azokat a képernyőre írja.

4.4.3 Felhasználók listázása távoli gépen

A finger (ujj) a távoli gépen éppen dolgozó felhasználók adatait adja meg. Rákérdezhetünk egy bizonyos felhasználóra is, de általánosság-

ban is kérdezhetünk. A szolgáltatott adatok ennek megfelelően részletesebbek vagy általánosabbak.

Általános esetben a finger után a gép nevét kell megadnunk, de úgy, hogy egy „@” jellel vezetjük be azt.

```
[root@pip /root]# finger @pip
[pip.jpte.hu]
Login Name Tty Idle Login Time  Office Office Phone
root  root *1  7:54  May 17 14:44
root  root *p1       May 17 14:32 (:0.0)
root  root *p0 2:36  May 17 14:42 (:0.0)
root  root *p3 1:59  May 17 20:40 (:0.0)
```

Ha részletesebb adatokat kívánunk kapni, akkor a lehasználó nevét és a gép nevét egy „@” jellel elválasztva kell beírnunk:

```
[root@pip /root]# finger root@pip
[pip.jpte.hu] Login: root         Name: root
Directory: /root                  Shell: /bin/bash
On since Sun May 17 14:44 (CEST) on tty1 8 hours idle
    (messages off)
No mail.
No Plan.
```

Ha az adott felhasználó a saját könyvtárában elhelyez egy .plan nevű állományt, akkor azt a finger mindenkinek kiírja aki fingerrel érdeklődik felőle. A fenti példában a rendszergazdának nem volt ilyen állománya, erre utal a „No Plan.” felirat.

Látható, hogy a finger segítségével igen fontos adatokat nyerhetünk távoli gépek felhasználóiról, és mivel ezek az adatok esetleg egy hálózaton keresztüli támadás kiindulópontjául is szolgálhatnak ez káros is lehet. Sok helyütt ezért a finger kéréseit megtagadják, ami azt eredményezi, hogy az adott gép nem szolgáltat ki adatokat a felhasználóiról. A korlátozás lehet részleges is, ha konkrét felhasználóról érdeklődünk - vagyis az imént bemutatott módszerek közül a másodikat használjuk - esetleg több sikerrel járunk.

A chfn parancs segítségével megváltoztathatjuk a – finger által is felhasznált – adatokat amelyet a rendszer rólunk nyilvántart. A chfn mielőtt megváltoztatná adatainkat a jelszavunkat kéri.

4.4.4 Távoli parancssor

LINUX alatt lehetőség van arra, hogy távoli gépre bejelentkezzünk és azon parancssor segítségével dolgozzunk. A távoli gépre belépett felhasználó programjait ott futtathatja, az ottani környezetet, állományokat, erőforrásokat használva. Természetesen név és jelszó kombinációval azonosítania kell magát mielőtt bármit tenne.

A belépés egyszerű formája a `telnet` (*telephone network*, telefonos hálózat) parancs használata:

```
[root@mad /root]# telnet roger.jpte.hu
Trying 193.6.49.2...
Connected to roger.jpte.hu.
Escape character is '^]'.

Red Hat Linux release 5.0 (Hurricane)
Kernel 2.0.32 on an i586
login: pipas
Password:
[pipas@roger pipas]$
```

Ahogyan láthajuk a felhasználó bejelentkezett a roger.jpte.hu nevű gépre és ott ugyanúgy parancsokat adhat ki, ahogyan a saját gépén is megteheti. A „login:” kérdésre a felhasználói nevet kellett megadni, míg a „Password:” után a jelszavát - ami biztonsági okokból nem jelenik meg a képernyőn.

Nagyon fontos tudnunk, hogy a `telnet` parancs által végrehajtott kommunikáció nem titkosított adatcsatornát használ, ezért azt speciális programokkal le lehet hallgatni. Mivel a jelszó is titkosítás, sifrírozás nélkül kerül továbbításra, a `telnet` kapcsolat nem biztonságos. Épp ezen okokból a rendszergazda nem jelentkezhet be a LINUXra telnettel, mivel az ő jelszavát ellopva teljhatalomra tehet szert az esetleges behatoló.

Biztonságos kapcsolat létrehozására a telnethez igen hasonló `ssh` (*secure shell*, biztonsági parancssor) alkalmas. Ez olyan kódolt átvitelt valósít meg a két gép közt, amelyet igen nehéz - valójában lehetetlen - megfejteni. Az esetleges támadó le tudja hallgatni ugyan az `ssh` kapcsolatot, de a titkosítás miatt azt értelmezni már nem.

Az ssh használata kissé különbözik a telnetétől, amennyiben alapesetben nem kérdez felhasználói nevet, hanem feltételezi, hogy a távoli gépen is ugyanaz a nevünk mit a helyi LINUXon.

```
[root@mad /root]# ssh leila.jpte.hu
root@leila.jpte.hu's password:
Last login: Thu May 21 13:17:31 1998 from mad.jpte.hu
No mail.
[root@leila /root]#
```

Látható, hogy az ssh nem kért felhasználói nevet, csak jelszót. Ha ez nem felel meg céljainknak - mert a távoli gépen más a nevünk mint helyben -, akkor a -l opciót használva adhatjuk meg a távoli kiszolgálón érvényes nevünket.

```
[root@mad /root]# ssh -l pipas leila.jpte.hu
pipas@leila.jpte.hu's password:
Last login: Wed May 13 08:31:54 1998 from ajkterm06
You have new mail.
[pipas@leila pipas]$
```

A példában a root nevű rendszergazda a távoli gépen pipas néven jelentkezett be.

Megjegyezzük, hogy az ssh nem a telnet titkosított változata, hanem az rsh (*remote shell*, távoli parancssor) módosításával készült. Nem csak a titkosításban különbözik a telnettől, hanem néhány plusz szolgáltatást is tartalmaz - például belépéskor kiírja képernyőre, hogy érkezett -e levelünk.

4.4.5 Az elektronikus levelezés

Az elektronikus levelezés egyike az Internet leggyakrabban használt szolgáltatásának. Talán annak köszönhető ez, hogy a elektronikus levelezés nem különbözik alapvetően a megszokott postai levelezéstől, ezért a használatát mindenki könnyedén megérti.

LINUX alatt minden felhasználónak - akinek a rendszergazda valamilyen különleges okból nem tiltotta meg - joga van leveleket küldeni és

leveleket fogadni, vagyis van levelesládája. A levelek továbbítása az Interneten automatikus, minden levél eljut a felhasználó levelesládájába, függetlenül attól, hogy a felhasználó az adott pillanatban a gépre be van jelentkezve vagy sem. A levelek tehát csendben várakoznak a gépen és megvárják amíg a felhasználó elolvassa őket majd kitörli közülük azokat, amelyekre továbbá már nincs szüksége.

Minden felhasználó azonosítása a felhasználói nevével és a gép Internet nevével történik, e kettő megfelelő formában az elektronikus levélcímet (*E-mail address*). A cím formája felhasználónév@gépnév, vagyis az elöl álló felhasználói nevet és az utána következő gépnevet egy speciális jel, a „@" választja el[8]. Az elektronikus levélcím elégséges információ ahhoz, hogy valakit az Interneten azonosítsunk, levelet küldjünk számára.

Az elektronikus levelek alapállapotban kódolatlanul haladnak az Interneten, vagyis bárki aki a levél haladásának útjában a megfelelő ismeretekkel és jogkörökkel rendelkezik, alevelet elolvashatja. Ez ellen - indokolt esetben - sifrírozással, kódolással védekezhetünk, mely LINUX esetében igen nagy biztonságot adhat a kéretlen olvasókkal szemben. A levelek titkosításáról később ejtünk szót, itt csak annyit jegyzünk meg, hogy e titkosítás mai ismereteink szerint semmi módon nem megkerülhető.

Nem csak a hivatalos leveleket lehet ellőtni a hitelességet igazoló pecséttel, hanem az elektronikus levelet is. Ez szintén kriptográfiai eszközökkel történhet és a hatékonysága is hasonló. A titkosítás és a hitelesség garantálása lehetővé teszik, hogy az elektronikus levelezés teljes egészében felváltsa a papír alapú irodai levelezést, ezzel igen olcsóvá és gyorssá téve a kommunikációt.

Az elektronikus levelezésről elmondható, hogy a levél kézbesítése - ha valamilyen hiba azt nem akadályozza - percek alatt megtörténjen még akkor is, ha a címzett egy másik kontinensen tartózkodik.

4.4.5.1 Az elektronikus levelezés a `mail` segítségével

Az elektronikus levelezéshez meglehetősen sok program áll rendelkezésünkre, amelyek közül nem egy egérrel kezelhető, széles körű szolgáltatást nyújt, könnyen használható. Az itt bemutatásra kerülő levelezőprogram ezekkel

[8] A „@" kilvasva „et" vagy a magyar szlengben elterjedt „kukac" lehet.

ellentétben igen egyszerű, nehézkesen kezelhető, mindazonáltal bármely Unixot futtató gépen elérhető, ezért használatának legegyszerűbb elemeit érdemes elsajátítani.

A `mail` (posta) nevű Unixos levelezőprogram hasznos lehet abban az esetben, ha egy meglévő állományt akarunk gyorsan továbbítani, vagy a levélküldést automatizálni akarunk. Ebben az esetben a program indítása a következő forma szerint történhet:

```
[root@pip /root]# mail root@pip.jpte.hu -s "Ez a targy." <test.txt
[root@pip /root]#
```

A `mail` parancs után a címzett neve következik – ez esetben `root@pip.jpte.hu`. A `-s` (*subject*, tartalom) opció után idézőjelek közt található szöveg a levél tárggya. A parancssor végén található `<test.txt` nem más, mint a mail bemenetének átirányítása. Ennek hatására a `mail` nem a billentyűzetről várja a levelet, hanem a `test.txt` nevű állományból. Nem kell tehát begépelnünk semmit, a `mail` automatikusan elküldi a `test.txt` tartalmát a címzettnek.

Meglévő állomány küldésére ez igen gyors és egyszerű módszer, ráadásul automatikusan is elvégezhető, hiszen nincsen szükség a felhasználó jelenlétére a parancs végrehajtásához.

4.4.6 Állományok másolása távoli gépről

Az FTP (File Transfer Protocol) olyan adatátviteli protokoll, amely állományok másolását teszi lehetővé az Internetre kötött számítógépek között. Az `ftp` (*File Transfer Program*) a LINUX része, segítségével kihasználhatjuk ezt az adatátviteli lehetőséget. Természetesen az FTP átvitelt más alkalmazásokkal is használhatjuk - amelyek kényelmesebbek, színesebbek és könnyebben kezelhetőek - az `ftp` azonban minden gépen rendelkezésünkre áll ezért hasznos lehet megismerkedni vele.

4.4.6.1 Az `ftp` alapvető parancsai

Amikor parancssorból indítjuk az `ftp` -t, azonnal megadhatjuk a kiszolgálót - a távoli gép Internet nevét vagy címét -, amellyel kapcsolatba

kívánunk lépni:

```
[root@mad /root]# ftp sunserv.kfki.hu
Connected to sunserv.kfki.hu.
220 sunserv.kfki.hu FTP server
(Version wu-2.4.2-academ[BETA-15](1)
Mon Nov 3 10:53:01 MET 1997) ready.
Name (ftp.kfki.hu:root):
```

Ahogyan látható az ftp program felvette a kapcsolatot a sunserv.kfki.hu nevű géppel és felhasználói nevet kér. Ha a kiszolgálón login névvel rendelkezünk, akkor azt megadhatjuk majd a jelszavunk begépelése után állományokat tölthetünk le a saját könyvtárunkból. Ha nincs saját felhasználói nevünk a távoli gépen, akkor is lehetőségünk van sok gépen állományokhoz - programokhoz, irományokhoz, stb. - jutnunk, „anonymous" felhasználói néven. Azokat a gépeket, amelyekről így - név nélkül, anonim módon - tudunk állományokat letölteni „anonymous FTP szervereknek" vagy röviden „anon FTP szervereknek" nevezzük.

```
Name (sunserv.kfki.hu:root): anonymous
331 Guest login ok, send your complete e-mail address
as password.
Password:
```

Amint látjuk az „anonymous" megadása után, a kiszolgáló biztosít róla, hogy vendég (*guest*) hozzáférés is lehetséges, majd felszólít arra, hogy adjuk meg az E-mail címünket mintha jelszó volna. (A jelszó begépelés közben biztonsági okokból nem jelenik meg a képernyőn, ezért „vakon" kell begépelni.) Ez érthető kívánság a gépet üzemeltetőktől, hiszen ingyenes szolgáltatást nyújtanak számunkra, ezért jogosan várják el tőlünk, hogy legalább az E-mail címünkkel azonosítsuk magunkat.

Amikor beírtuk E-mail címünket, a kiszolgáló általában üdvözlő szöveget nyomtat a képernyőre, majd várakozik a parancsainkra.

```
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Az „ftp>” felirat jelzi, hogy a gép olyan parancsokat vár, amelyek az `ftp` program sajátjai. Meglehetősen sok ilyen parancs létezik, itt csak az elengedhetetlenül fontosakat vesszük sorra.

Első parancs a `dir` (*directory*), amellyel a felhasználó találkozik. E parancs a képernyőre listázza az FTP kiszolgálón található állományok közül azokat, amelyek az aktuális könyvtárban vannak:

```
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls. total 8
drwxr-xr-x 7  root root 1024 May 18 19:57 .
drwxr-xr-x 7  root root 1024 May 18 19:57 ..
d--x--x--x 2  root root 1024 Sep 16 1997  bin
d--x--x--x 2  root root 1024 Sep 16 1997  etc
drwxrwxrwx 2  root ftp  1024 May 18 20:01 incoming
drwxr-xr-x 2  root root 1024 Sep 16 1997  lib
dr-xr-sr-x 18 root ftp  1024 May  4 18:03 pub
-rw-r--r-- 1  root ftp   560 Jan 30 19:53 welcome.msg
226 Transfer complete.
ftp>
```

Látható, hogy a lista a már ismertetett UNIX formában tárul szemünk elé, éppen olyan módon, ahogyan az az `ls -l` parancs esetén a parancssorban megjelenne a saját gépünkön[9]. A különbség mindösszesen annyi, hogy ezek az állományok és könyvtárak nem a saját gépünkön, hanem a kiszolgálón vannak.

Az ftp parancsok közt megtalálhato a cd (change directory, könyvtárváltás) is, a távoli gép könyvtárai közti közlekedés eszközeként.

```
ftp> cd pub
250-Please read the file README
250- it was last modified on Fri Jan 30 19:58:38 1998
- 110 days ago
250 CWD command successful.
ftp>
```

[9]Megjegyzendő, hogy az `ls -l` formát az ftp is elfogadja.

A parancs most a „pub” (*public*, nyilvános) könyvtárba vitt minket. Az anon FTP szervereken konvencionálisan ebben a könyvtárban vannak azok az állományok, amelyek bárki számára elérhetőek.

A `dir` és `cd` parancsok segítségével bebarangolhatjuk az FTP kiszolgáló könyvtárait és megkereshetjük azokat az állományokat, amelyeket le kívánunk tölteni a saját gépünkre. Mielőtt azonban a letöltést elkezdenénk, be kell állítanunk az átviteló módot.

Kétféle átviteli módszert ismer az FTP rendszer: *bináris* és *szöveges* módszert. A bináris módszer változtatás nélkül tölti át az állományt a helyi gépre, a szöveges pedig konverziót végez áttöltés közben. Konverzióra akkor van csak szükség, ha szöveges állományt másolunk két különféle operációs rendszer között. Ha az áttölteni kívánt állomány nem szöveges, akkor semmiképpen ne használjuk a szöveges módszert, mivel az az állományt átalakítja és használhatatlanná teszi.

Szöveges átviteli módba az `ascii` *(American Code For Information Interchange*, Amerikai Szabványos Információátviteli Kódolás), binárisba pedig a `bin` (*binary*, bináris) parancsokkal kapcsolhatunk:

```
ftp> ascii
200 Type set to A.
```

Visszakapcsolás:

```
ftp> bin
200 Type set to I.
```

Állományok letöltése előtt mindenképpen legyen gondunk a megfelelő átviteli mód beállítására, ellenkező esetben lehetséges, hogy a letöltött anyag hasznavehetetlenné válik a konvertálás miatt.

Mindezek után nincs más feladatunk, mint megkezdeni a kiválasztott állomány letöltését. Erre a `get` (megszerez) parancs használható:

```
ftp> get tanfolyam.txt
local: tanfolyam.txt remote: tanfolyam.txt
200 PORT command successful.
150 Opening BINARY mode data connection for
tanfolyam.txt (12903 bytes).
226 Transfer complete.
```

```
12903 bytes received in 0.066 secs (1.9e+02 Kbytes/sec)
ftp>
```

A `get` végrehajtása után az állomány a saját gépünkre másolva megtalálható abban a könyvtárban, ahonnan az ftp programot indítottuk, a képernyőn pedig statisztikai adatokat látunk az átvitel folyamatáról.

Ha egyszerre több állományt akarunk áttölteni, akkor használhatjuk az `mget` parancsot az áttöltésre. Ez megegyezik a get paranccsal, de itt már használhatjuk a joker karaktereket (* és ?) állományok csoportjainak kijelölésére is.

```
ftp> mget *.zip
mget ssh-1-2windows.zip? y
200 PORT command successful.
150 Opening BINARY mode data connection for
ssh-1-2windows.zip (192733 bytes).
226 Transfer complete. 192733 bytes received in
0.64 secs (2.9e+02 Kbytes/sec)
mget ssh-1.2.14-win32bin.zip?
```

Amint látjuk, az `mget` parancs az összes állományra egyenként rákérdez, ami valljuk be meglehetősen bosszantó dolog. A megoldás az `ftp` „`-i`” opcióval való indítása. A `-i` kapcsoló megadása esetén az `ftp` az `mget` parancsot külön kérdés nélkül azonnal végrehajtja. Az indításra láthatunk példát a következő sosrokban:

```
[root@mad forprint]# ftp -i leila.jpte.hu
Connected to leila.jpte.hu.
220 leila.jpte.hu FTP server
(Version wu-2.4.2-academ[BETA-15](1)
Mon Sep 22 20:49:48 EDT 1997) ready.
Name (leila.jpte.hu:root):
```

Amikor az összes állományt áttöltöttük, kiléphetünk a programból. Erre a `quit` (mentes), parancs szolgál:

```
ftp> quit
221 Goodbye.
[root@mad forprint]#
```

Látható, hogy az `ftp` megszakította a kapcsolatot az FTP kiszolgálóval, majd befejezte futását. Visszakaptuk a szokásos parancssort, legközelebb csak akkor használhatunk `ftp` parancsokat, ha újra elindítjuk.

4.4.7 Az `ftp` időzített használata

Amikor nagy távolságból akarunk állományokat másolni, akkor bizony az átvitel akár órákig is tarthat. Ez főleg akkor jelent sok várakozást, ha csúcsidőben próbálkozunk a letöltéssel, amikor a sok felhasználó miatt nagy a számítógéphálózat terhelése. Megoldást jelenthet, ha az áttöltést automatikussá tesszük és arra utasítjuk a gépet, hogy azt olyankor hajtsa végre, amikor a hálózat terheltsége alacsony - pl. éjszaka vagy vasárnap.

Az időzített használathoz mindenekelőtt létre kell hoznunk egy `.netrc` nevű állományt a saját könyvtárunkban. Az `ftp` ugyanis indítás után megvizsgálja, hogy ebben az állományban benne van -e annak a gépnek az Internet neve, amelyet a parancssorban kapott. Ha itt az adott gép nevét megtalálja, akkor az utána található parancsokat megkísérli végrehajtani.

A `.netrc` -ben a következő formában írhatjuk be az automatikus letöltéshez szükséges információkat:

```
machine leila.jpte.hu
login anonymous
password pipas@ajk.jpte.hu
macdef init
cd /pub/Linux
bin
mget ssh*
exit
```

Az állomány elején látható machine bejegyzés tartalmazza az `ftp` kiszolgáló nevét. A következő sorban látható `login` után azt adhatjuk meg, hogy milyen felhasználói néven kívánunk bejelentkezni. A harmadik sorban látható `password` bejegyzés után adhatjuk meg a jelszót - ami anonymous ftp esetében egyszerűen az E-mail címünk[10].

[10] Óvatosan kell bánnunk a lejegyzett jelszóval! Győződjünk meg mindig arról, hogy az állomány mások számára nem olvasható!

A negyedik sorban található macdef (*macro definition*, csoportutasítás készítése) az induláskor (*initialization*, előkészítés) lefuttatandó paracsszekvenciát készít. Ez után a sor után beírhatjuk, hogy mit szeretnénk az `ftp` automatikus bejelentkezése után végrehajtani. Az utolsó utasítás az `exit` legyen, mert ez az `ftp` program futását befejezi.

Ha most az `ftp` programot elindítjuk, az csak abban az esetben fogja végrehajtani a `.netrc` tartalmát, ha parancssorban megadjuk a `leila.jpte.hu` nevet – vagyis a `.netrc` állomány `machine` kulcsszava után írt gép Internet címét. Mivel az automatikusan végrehajtandó parancsok közt szerepel az `mget`, az `ftp` -t a `-i` opcióval kell indítanunk. Az `ftp` indítása ezek szerint a következő lesz az automatikus indításhoz:

```
[root@mad /root]# ftp -i leila.jpte.hu
```

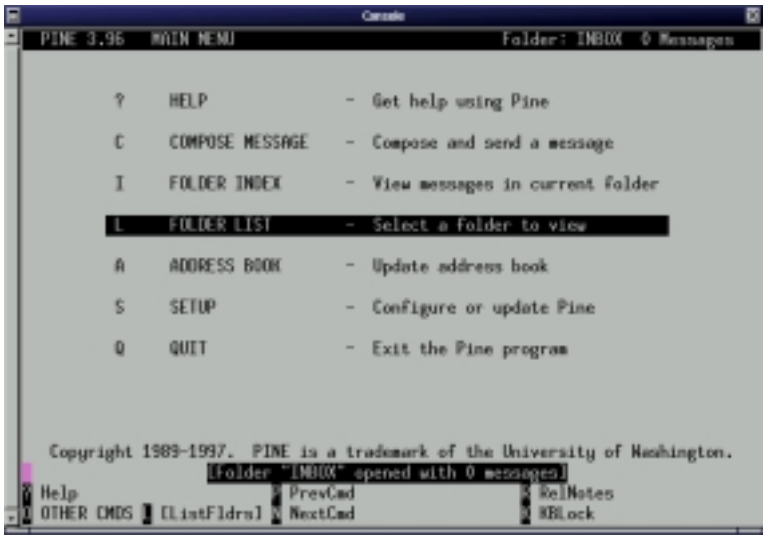
Most már csak arról kell gondoskodnunk, hogy az `ftp` elinduljon a megfelelő időben – amikor mi már nem vagyunk ott. Erre az `at` parancs szolgál:

```
[root@mad /root]# at 18:02
at> ftp -i leila.jpte.hu
at> <EOT>
warning: commands will be executed using /bin/sh
job 2 at 1998-05-26 18:02
[root@mad /root]#
```

Az `at` parancs után meg kell adnunk, hogy mely időpontban szeretnénk a parancsot letöltést indítani, majd le kell nyomnunk az Enter billentyűt. Ekkor az `at` parancs elindul, majd az `at>` jellel várakozik. Ekkor kell beírnunk a parancsokat amelyeket az adott időpontban el szeretnénk indítani – jelen esetben `ftp -i leila.jpte.hu`. Ha a parancsokkal végeztünk, a Ctrl+D kombinációval kell kilépnünk.

4.5 Elektronikus levelezésa a `pine` segítségével

A `pine` (*Program for Internet News and Email*, program az internetes újság és levelezés kezelésére) egy karakteres, menüvezérelt program, amelyet sokan használnak elektronikus levelezésre.



ábra 4.2: A pine főkérpernyője

A `pine` alkalmas elektronikus levelek fogadására, szerkesztésére és küldésére, valamint levelezőpartnerek címének nyilvántartására. A leveleket iratrendezőkbe (*folder*) tartja, így a felhasználónak lehetősége van a jobb áttekintehőség érdekében leveleit az általa létrehozott struktúra szerint csoportosítani.

A `pine` indítása a `pine` szó begépelésével lehetséges karakteres felületen vagy grafikus parancssorból. A `pine` indítás után a 4.2 ábrán látható képernyővel jelenik meg.

A képernyő felső sorában egysoros mező értesít arról, hogy melyik iratrendezőt használjuk éppen és ebben mennyi levél található. A képernyő közepén található a `pine` főmenüje, alul pedig mindig azoknak a billentyűknek a felsorolása található, amelyeket az adott pillanatban használhatunk használhatunk.

A `pine` főmenüje a következő elemekből áll:

`help` A pine kezeléséről olvashatunk egy néhány oldalas dokumentumot e pont alatt.

`compose message` Itt írhatunk és küldhetünk el új levelet.

ábra 4.3: A pine levélszerkesztője

`folder index` Az aktuális iratgyűjtő tartalmát nézhetjük meg e menüben.

`folder list` Az iratgyűjtök közül válogathatunk itt.

`address book` A levelezőpartnerek címeinek karbantartása történhet itt.

`setup` A pine beállításait változtathatjuk meg ebben a menüpontban.

`quit` A pineból való kilépésre szolgál.

4.5.1 Levél küldése

Ha a `pine` segítségével akarunk levelet küldeni, akkor az indítása után a „*compose message*” menüpontot kell választanunk. Ekkor a 4.3 ábrán látható képernyőhöz jutunk.

A képernyőn a kurzormozgató nyilakkal tudunk felfelé és lefelé haladni, hogy felül kitöltsük a levél fejlécét, alul pedig beírjuk a levél szövegét. A fejlécben található rövidítések a következők:

`To` A levél címzettje. Ide kerül az az elektronikus levélcím, ahova a levelet küldeni szeretnénk. Ha a levélnek több – egyenrangú – címzettje is van, akkor ide mindanyiuk levélcímét be kell írnunk, vesszővel elválasztva.

`Cc` *Carbon Copy* (indigómásolat). Ide is beírhatunk egy vagy több levélcímet, éppen úgy, ahogyan a To mezőbe. Az ide beírt címre szintén elküldésre kerül a levél. Tudnunk kell, hogy az eredeti címzett láthatja, hogy a neki szánt levélből ki kapott másolatot.

`Attchmnt` *Attachment* (melléklet). Ebbe a mezőbe állományneveket írhatunk be vesszővel elválasztva. A `pine` gondoskodik arról, hogy a mellékletben megjelölt állományok a levéllel együtt kézbesítésre kerüljenek az összes címzett számára.

`Subject` Tárgy. A levél tartalmáról írhatunk egy rövid – egy soros – kivonatot, amely alapján a címzettek azt listázhatják a képernyőre. Érdemes jól érthető tárgyat írnunk, mert a címzett e mező tartalmát látja a beérkezett levelek listájában.

Ha a levél fejlécében a To vagy Cc mezőkben állunk, akkor a Ctrl+t billentyűzetkombináció hatására a személyes címtárunkhoz jutunk, ahonnan lehetőségünk nyílik ismerőseink címét kikeresni. Ehhez természetesen előbb fel kell tölteni a címtárunkat adatokkal.

Az Attchmnt mezőben is használható a Ctrl+t kombináció. Amint megnyomtuk, a pine egy állománykezelő képernyőt ad, amelyből a küldeni kívánt állomány kiválasztható.

Ha befejeztük a levél megírását, a Ctrl+x kombinációval küldhetjük azt el. Ha a levél írása közben úgy döntenénk, hogy a levél elküldésétől visszalépünk, a Ctrl+c kombinációval megszakíthatjuk a levélírást.

4.5.2 Levelek fogadása

A `pine` alapesetben három iratrendezőt használ a levelek kezelésére, amelyek *a*) `INBOX` a bejövő levelek tárolására, *b)* `sent-mail` az elküldött levelekről készült másolatok részére, valamint *c)* `saved-messages` az elolvasott levelek számára.

A főmenüben

Fejezet 5

Az X WINDOW és alkalmazásai

5.1 Bevezetés

Az X WINDOW egy ablakozós rendszer, amely igen sokféle gépen es operaciós rendszeren elérhető.

Ablakozós rendszerről (*Windowing System*) akkor beszélünk, amikor a számítógép képernyőjét több kisebb területre - ablakokra - osztjuk, hogy így a felhasználó párhuzamosan több programmal dolozhasson. Az ablakozós rendszerek párhuzamos fejlődésük során olyan - egyszerűen kezelhető - grafikus alapelemeket dolgoztak ki, amelyek a grafikus mutatóeszközzel - pl. egér - könnyen és gyorsan kezelhetőek. Ilyen elemek például a nyomógombok, listák, menük, szerepük pedig az, hogy a felhasználónak ne kelljen az adott program kezelését megtanulni, parancsokat és billentyűzetkombinációkat magolni. Ablakozós, grafikus rendszerben - mint egy műszerfalon vagy irányítópulton - a képernyőre pillantva minden eszköz a szemünk elé tárul, s csak válogatnunk kell belőlük.

Természetesen a UNIX alapú operációs rendszerek esetében is hasznos a könnyű kezelhetőség, ezért a 80-as évek közepén indult fejlesztés keretében kidolgozták a UNIX ablakozós rendszerét az X WINDOW SYSTEM -et. Máig a programrendszer sokat fejlődött, a LINUX ma a 11 -es

változat 6. kiadását használja (X11R6), mivel a MIT (*Massachusetts Institute of Technology*) az általa birtokolt rendszert publikussá tette.

Az X WINDOW nem csak egy programrendszer, hanem egy szabvány is, így többféle gépre, többféle operációs rendszerre elkészíthető az inplementációja, vagyis gyártófüggetlen rendszerbe illeszthető, kapocs a különféle gyártók rendszerei közt.A felhasználónak végső esetben nem kell tudnia, hogy az általa használt operációs rendszer milyen gyártótól származik, hogy programjai milyen számítógépen futnak, hiszen szabványos felületen - az X WINDOWn - keresztül kommunikál azokkal.

5.1.1 A grafikus megjelenítés alapfogalmai

Ha grafikus üzemmódban használjuk számítógépünket, néhány alapfogalommal meg kell ismerkednünk.

Elsőképpen ismerkedjünk meg a *felbontás* fogalmával. felbontáson értjük a képernyőn elhelyezhető képpontok számát mind szélességben mind magasságban. A számítógépek grafikai rendszere minőségtől függően több kevesebb képpontot képesek ábrázolni a monitoron, vagyis más-más felbontásúak. Minél nagyobb a felbontása egy grafikus megjelenítőnek, annál olvashatóbb képet tárnak szemünk elé, annál olvashatóbb rásképet produkálnak. Szokásos meghjelenítési mód a 600x800 -as (600 képpent vízszintesen, 800 képpont függőlegesen) vagy az 1024x768.

Az egyszerre megjeleníthető színek számát *színmélységnek* nevezzük. A grafikus rendszer felépítéséből következik, hogy korlátozott az egyszerre megjeleníthető színek száma, vagyis azoknak az árnyalatoknak a mennyisége, amelyek egyidőben a monitoron lehetnek. A színek számát általában nem közvetlenül adják meg, hanem - az egyszerűség kedvéért - azoknak a biteknek a számával, amelyen egy képpont színét tárolja a számítógép. Így 8 bpp (*bit per pixel*, bit képpontonként) 256 színt, 16 bpp pedig 65536 színt jelent. Természetesen a grafikus megjelenítőrendszer ennél több képpontot ismer, de csak ennyit fajta színt képes *egyidőben* a képernyőre rajzolni. Azt mondhatjuk, hogy a színek közül egy véges *paletta* tárolja az aktív - ténylegesen használt - színeket, s ennek a palettának a méretét jelzi a színmélység.

A monitor a képet képpontonként rajzolja fel és - változatlan képtartalom mellett is - folyamatosan frissíti azt. A monitor minőségére jellemző a frissités sebessége, vagy frekvenciája. Gyors frissítésű monitor es-

etén a kép vibrálásmentes, nem fárasztja oly mértékben a szemet mint régebbi, lassabb monitorok esetén.

A grafikus rendszer e három jellemzője - felbontás, színmélység és frissítési frekvencia - határozza meg a minőségét. A mai eszközeink már képesek többféle felbontásban, színmélységben és frissítési frekvenciával dolgozni, amely üzemmódok közt szoftveresen - program segítségével - választhatunk. Erre a váltioztatásra képes az X WINDOW, amely az üzemmódok igen sokféle kombinációjában képes működni, igényeinknek és anyagi lehetőségeinknek megfelelően.

5.1.2 Kliens szerver architektúra

Az X WINDOW kliens-szerver architektúrájú, vagyis létezik X szerver és kliens, amelyek egymással kommunikálva teszik lehetővé grafikus programok futtatását. Habár általában a kliens és a szerver ugyanazon a gépen fut, lehetőségünk van arra is, hogy a kliens(ek) más gép(ek)en fusson(sanak), így a felhasználónak nem kell fizikailag annál a gépnél tartózkodnia, ahol a programja fut.

kép

Az alkalmazásokkal az X szerver áll kapcsolatban, mig a felhasználóval az X kliens, így lehetséges - a kettejüket összekötő információs csatorna révén - a földrajzilag más területen tartozkódó felhasználó igényeinek kielégítése.

A liens szerver architektúra teszi azt is lehetővé, hogy egyazon gépen egyszerre több felhasználó is futtasson grafikus programokat.

kép

5.1.3 Az X WINDOW indítása

Az X WINDOW többféleképpen is indítható, de bármelyik módot is használjuk az indításra, végső soron mindig ugyanaz a program indul el.

Az X WINDOW legegyszerűbben a `startx` paranccsal indítható el. Karakteres kepernyőn bejelentkezés után gépeljük be:

```
[pipas@pip /]$ startx
```

ábra 5.1: X kiszolgá

A `startx` parancs elinditja az `X` nevű programot, amely tulajdonképpen az X WINDOW szerver. Az X -nek paramétereket átadni "–" után tudunk, mert így a `startx` azokat nem értelmezi, hanem változtatás nélkül átadja az X -nek. Ha pl. azt szeretnénk, hogy 256 szinű üzemmódban induljon az X szerver, akkor a következő opciót kell megadnunk:

```
[pipas@pip /]$ startx -- -bpp 8
```

Hasznos lehet a :1 opció is, amely a második képernyőn indítja el az X WINDOWt. Ez a képernyő az Alt+Ctrl+F8 billentyűzetkombinációval érhető el. Értelemszerűen az Alt+Ctrl+F9, Alt+Ctrl+F10 billentyűkombinációval elérhető újabb X képernyőket is indíthatunk.

```
[pipas@pip /]$ startx -- :1
```

Amennyiben más géphez szeretnénk kapcsolódni - vagyis egy távoli gépen futtatni programokat úgy, hogy az a mi gepünkön, mint X terminálon jelenjék meg - akkor közvetlen indíthatjuk a "X" programot a megfelelő opcióval:

```
[pipas@pip /]$ X -query leila.jpte.hu
```

A fenti peldánál a leila.jpte.hu nevű géphez kapcsolódunk – X Window segítségével grafikusan – az Interneten keresztül[1]. Ilyenkor először egy bejelentkező képernyőn kell beírnunk a felhasználói nevünket es a jelszavunkat, vagyis a távoli gépen azonosítani kell magunkat. A távoli gépen az xdm nevű programnak kell futnia, amely fogadja a X WINDOW alóli bejelentkezéseket.

Lehetőségünk van arra is, hogy a hálózaton létező összes lehetséges – X Window kapcsolatot támogató – szerverből válogassunk egy listázóprogram (*chooser*, választó) segítségével. Ehhez az `-indirect` (nem közvetlen)kapcsolót kell megadnunk, valamint annak a gépnek a nevét, amelytől a szerverek összegyűjtését kérjük. Ez a – listát szolgáltató – gép legegyszerűbb esetben lehet a saját gépünk is, vagyis gépnévként használhatjuk a localhost -ot:

[1]Kissé zavaró lehet, hogy szokatlan módon a szerver keresi meg a klienst kapcsolatfelvétel céljából. Azért nevezzük mégis szervernek az x -et, mert a kapcsolat kiépülése után az ő az, amelyik a kéréseket - amelyek grafikus elemek megjelenítésére vonatkoznak - végrehajtja, az alkalmazás pedig - amelyik a Linux szerveren fut - e kéréseket generálja, vagyis kliensként viselkedik.

```
[pipas@pip /]$ X -indirect localhost
```

Amennyiben olyan X Window kapcsolatot kívánunk felépíteni, amely nem a saját belső hálózatunkon van, akkor a távoli alhálózat egyik gépét kérhetjük meg a lista összeállítására. Tulajdonképpen az indirect (nem közvetlen) kapcsolat lényege az, hogy X Window kapcsolatot építünk ki egy közvetítő – listát összeállító – gép segítségét igénybe véve.

Az X fenti opciói együttesen is használhatóak:

```
[pipas@pip /]$ X -bpp 8 :1 -indirect libun.jpte.hu
```

A fenti példa 256 színű üzemmódban indítja az X Window -t (`-bpp 8`), az Alt+Ctrl+F8 billentyűzetkombinációval elérhető képernyőn (`:1`) és a `libun.jpte.hu` -t kéri meg, hogy a körzetében lévő – X WINDOW kapcsolatra felkészült – gépekből adjon választási lehetőséget (`-indirect libun.jpte.hu`).

Az X WINDOW indításának harmadik módja az `xdm` indítása, ezt azonban csak a rendszergazda teheti meg.

```
[roor@pip /]# xdm
```

Ha a gépen az `xdm` fut, akkor az távoli bejelentkezéseket is fogadhat, más gépekről amlyek a már említett „`X -query`” parancs hatására a kapcsolatot kezdeményezik.

kép egyik gép xdm, többi X -query

Az indítás egy újabb módja az `xdm` automatikus indítása a gép bekapcsolásakor, ezt azonban csak a rendszergazda kezdeményezheti.

5.1.4 Kilépés az X WINDOW rendszerből

A rendszerkonfigurációtól függően a kilépés esetenként más-más helyen található, ezért nem mindig könnyű megtalálni a menük között, de a Ctrl+Alt+Backspace billentyűkombinációval bárhol kiléphetünk. Vigyáznunk kell, mert az X alatt futó programokból is kilépünk ilyenkor, a nem mentett adatok pedig elvesznek.

ábra 5.2: Az Afterstep ablakkezelő

Nem kell kilépni az X WINDOW -ból, ha csak időlegesen szeretnénk használni a karakteres terminálokat. A Ctrl+Alt+Fx billentyűkkel válthatunk a képernyők között, ahol a karakteres terminálok x=1,2, ... ,6 számú funkcióbillentyűkkel érhetőek el, az első X kepernyő az F7, a továbbiak pedig ennél feljebb találhatóak.

5.1.5 Ablakkezelők

Az X WINDOW maga nem rajzolja meg az ablakok kereteit. Ezt a feladatot - néhány más dologgal együtt - az ablakkezelő (*window manager*) végzi el. LINUX alá sokféle Window manager érhető el, melyek mindegyik más - más stílusú megjelenést biztosít.

Minden felhasznalo maga dönthet arról, hogy melyik ablakkezelőt kívánja használni. Amennyiben több ablakkezelő is telepítve van , előfordulhat, hogy minden felhasználó más-más ablakkezelőt használ. Ha meg hozzátesszük, hogy az ablakkezelők rugalmasan konfiguralhatóak - szintén felhasznalónként - akkor jól láthatjuk, hogy mennyire változatos és szabad grafikus környezetet biztosít az X WINDOW.

A `startx` a felhasználó saját könyvtárában a `.xinitrc` rejtett fájlt keresi és ha megtalálja, akkor ebből olvassa ki, hogy milyen ablakkezelő induljon. (Az ablakkezelő neve utan nem szabad & jelet tenni!) Amennyiben ilyen file nincs, a rendszergazda által beállított ablakkezelő indul el. Egy példa az `.xinitrc` fájlra:

```
# Inditunk egy naptarat
ical&
# Az ablakkezelo kovetkezik
/usr/local/bin/fvwm95
```

Amint a példában látható az `.xinitrc` fájl tartalmazhat X WINDOW alatt futó programokat, amelyek után „&” jellet kell írni, valamint - a fájl végén - az ablakkezelőt, amely után „&” jelet nem szabad tennünk[2]. Azt is láthatjuk, hogy a „#” jel után megjegyzéseket szúrhatunk be a fájlba. A megjegyzés végét a sor vége jelzi.

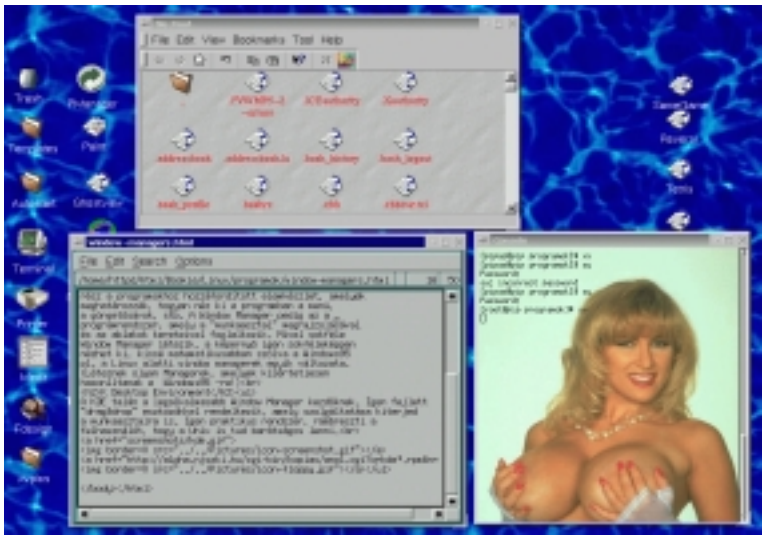
Azt, hogy távoli gépről történő X kapcsolat esetén milyen ablakkezelő induljon el, a felhasznaló könyvtárában található `.xsession` fájl mutatja. Ezt az állományt az `xdm` használja, tehát akkor is ez a mérvadó, ha a helyi gépen jelentkezünk be az `xdm` által indított grafikus bejelentkező ablakon - tehát nem a `startx` paranccsal indítjuk az X WINDOWt. A `.xsession` állomány formája megegyezik a már bemutatott `.xinitrc` állományéval, azzal a különbséggel, hogy a `.xsession` állománynak futtathatónak kell lennie.

Latható, hogy ugyanaz a felhasznaló kétféle ablakkezelőt használhat, attól függően, hogy honnan indítja az X WINDOWt. Ez akkor lehet igen hasznos, ha a gépet tavolról is használja lassú hálózatról vagy számítógépről is, hiszen az egyszerűbb ablakkezelők kevesebb erőforrással is beérik.

Ha már elindult az ablakkezelő, akkor általában lehetőseg van menüből másikat választani, méghozzá úgy, hogy a futó programokból sem kell kilépni.

---

[2] A „&” jel azon szándékunkat jelzi, hogy az adott programot a háttérben szeretnénk futtatni, vagyis - jelen esetben - a rendszer a fájlban továbbmehet nem kell megvárnia az `ical` programból való kilépést, azonnal indíthatja az `fvwm95` ablakkezelőt.



ábra 5.3: A KDE ablakkezelő

5.1.5.1 Afterstep

Az Afterstep ablakkezelő indítása felhasználó könyvtárában elhelyezett `.xinitrc` vagy `.xsession` állományba írt `afterstep` paranccsal történhet - ahogyan azt az előző fejezetben láttuk.

Ha más fájlt nem adtunk meg indításkor a `-f` opcióval, az `afterstep` először a felhasznaló saját könyvtárában keresi a `.steprc` állományban a beállíttásokat, majd ha itt nem találja, akkor a `/usr/X11R6/lib/X11/afterstep/system.steprc` nevű állományban. Így minden felhasználónak lehetnek saját beállításai, de ha nincsenek, az sem baj, hiszen akkor használja a program a rendszergazda által elkészített alapbeállításokat. Ha tehát saját kívánalmainknak megfelelően akarjuk beállítani az `afterstep` ablakkezelőt, akkor a `system.steprc` állományt kell a saját könyvtárunkba másolni `.steprc` néven, majd azt megfelelően átalakítani.

Az `afterstep` képes virtuális képernyők kezelésere, ami igen hasznos dolog, hiszen sokszor kevésnek bizonyul a kepernyő mérete a munkához. Ilyenkor a virtuális képernyők segítségével nagyobb területhez jutunk, ahol jobban elférnek az ablakok. Alapesetben a virtuális képernyők

ábra 5.4: A pager

közt az egérrel válthatunk az ún. pagerre kattintva (5.4 ábra).

Nem trivialis a hatterkep beallitasa. LINUX alatt legtobbszor az xv nevu grafikus programot hasznaljuk képek böngeszesere. A -quit -root opcioval elerheto, hogy a hatterben jelenjen meg a kep es ott is maradjon.

```
[pipas@pip /]$ xv -quit -root kepfile.gif
```

5.1.6 Widgetek

Bizonyos alapveto keprajzolo programok az ablakkezelotol fuggetlen rendszerben vannak elhelyezve, dinamikusan linkelheto fileokban. Ezeket a fileokat az X ala irott programok aktivan hasznaljak.

Ha e fileokat lecsereljuk, akkor megvaltoznak bizonyos kepernyoelemek – nyo- mogombok, gorditosavok –, meghozza tobb programban egyszerre. Ugyelnunk kell azonban, a dinamikus konyvtarak installalasakor le kell futtatnunk az ldconfig nevu programot.

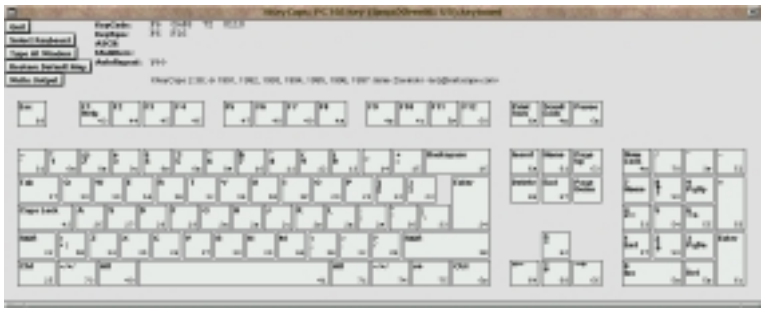
5.1.7 A billentyűzet X alatt

Az X WINDOW billentyűzetkiosztását az `xmodmap` programmal állthatjuk. A program képes előre elkészített szöveges állomány alapján beállítani a billentyűzetkiosztást;

```
[pipas@pip /]$ xmodmap magyar.kyb
```

Az xmodmap alkalmas az egér nyomogómbjainak átállítására is. Az alábbi parancscsal balkezesre allítja az egeret[3]:

[3] A bemutatott parancs tulajdonképpen a billentyűzetkiosztás definíciós állományából egy részletet ad át a `-e` opcióval az `xmodmap` számára. Ebből következik, hogy a bemutatott „`pointer`” ugyanígy használható a definíciós állományban is.



ábra 5.5: Az xkeycaps

```
[pipas@pip /]$ xmodmap -e "pointer = 3 2 1"
```

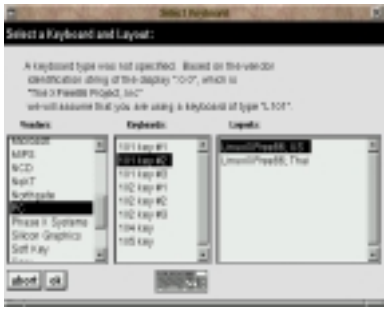
Ha nincsen előre elkészített billentyűzetkiosztásunk amellyel tökéletesen meg volnánk elégedve, saját magunknak kell azt elkeszítenünk. Erre a legalkalmasasabb az xkeycaps nevű program (5.5 ábra). Mivel az xkeycaps régebbi verziói a standard kimenetre küldték a kész billentyűzetdefiniáló állományt, a kimenetet már indításkor át kellett irányítani:

```
[pipas@pip /]$ xkeycaps >magyar.kyb
```

A program indítása után megpróbálja felismerni a billentyűzetünket majd választási lehetőségeket ajánl fel (5.6 ábra). Igen kellemes meglepetésben lehet részünk, amikor pl. VXT2000 diskless grafikus terminálon elindítva azonnal tudja, hogy a billentyűre pillantva mit is látunk.

Erre szükség is van, mert a billentyűzetkiosztást definiáló állományok más típusú billentyűzetre nem használhatóak. Ha pl. a fent nevezett gépünkön akarjuk használni az előzőleg PC -hez készített magyar billentyűzetdefinicíónkat, akkor az bizony nem fog menni!

Ha már beállítottuk, hogy milyen a billentyűzet, amihez kiosztást szeretnénk definiálni, akkor kezdődhet a tulajdonképpeni munka. Állítsuk az egérkurzort a nyomógombot szimbolizáló négyzet fölé, majd nyomjuk meg az egér jobb gomját és a felbukkanó menüből válasszuk az *„Edit KeySyms of Key”* menüpontot. Ekkor egy dialógusablakhoz jutunk, amelyet a 5.7ábra mutat.



ábra 5.6: Az xkeycaps billentyűzetválasztó ablaka

ábra 5.7: Az xkeycaps szimbólumválasztó ablaka

Itt történik annak a megadása, hogy milyen hatást kívánunk elérni adott billentyű lenyomásával. Ha a bal oldalon található "*KeySyms of KeyCode*" listában az elsőre kattintva kiválaszthatjuk a "*Character Set*" és "*KeySym*" listákból azt, hogy a gombot leütve milyen betűt lássunk a képernyőn. Ha a "KeySyms of KeyCode" listából most a másodikat választjuk, akkor azt adhatjuk meg, hogy a Shift billentyűvel együtt nyomva milyen legyen a hatás. Szokványos esetben a KeySym 1 helyén kisbetűt, a KeySym 2 helyén pedig ugyanannak karakternek a nagybetűs változatát kell beállítanunk.

A leggyakrabban használt karakterek elnevezése a következő:

acute, egyszeres hosszú ékezet, mint pl. oacute: ó, Oacute: Ó

diaeresis, dupla pont ékezet, pl. udiaeresis: ü

tilde, hullámos ékezet, pl. otilde: ő

circumflex, kalapos ékezet, pl. utilde: ű

doubleacute, kétszeres hosszú ékezet, mint pl. odoubleacute: ő

Ez utóbbi - vagyis a magyar kétszeres hosszú ékezet, amelyet máshol Hungarian umlautnak is neveznek - használatához Latin2 kódkészletre is szükségünk van, ezért ezeket a "*Character Set*" lista "*Latin 2*" pontjában találjuk meg.

A billentyűzet definiálás ennél természetesen sokkal bonyolultabb is lehet, hiszen lehetőség van pl. a vezérlőbillentyűk átdefiniálására is vagy a Shiften kívüli más módosítóbillentyűk használatára. Mindazonáltal az itt leírtak elégséges megoldást jelentenek a billentyűzet magyarítására és minden bizonnyal ez a legfontosabb.

Amikor végeztünk a munkával, annak eredményét szöveges fileba menthetjük a "*Write Output*" megnyomásával. Az xkeycaps jelenlegi verziói a felhasználó saját könyvtárába mentik ez az állományt egy fix néven, míg a régebbiek a standard kimenetre küldik azt.

7. Programok X alatt

Az X WINDOW alatt futtatható programok száma igen nagy. Ezen alkalmazások igen sokrétűek, változatosak mind kezelésükben mind kinézetükben. Néhány dologban azonban általaban hasonlítanak egymáshoz, ezeket vesszük most sorra.

Elsőképpen említésre méltó közös jellemző a képernyő átirányíthatósága. Bármelyik gépen indítsunk is X alatt programot, annak kepernyője más X WINDOWt futtató gépre átirányítható. Így a program ott jelenik meg, ott használható. A megjelenítést végző gépen először engedélyezni kell a programot futtató gépet az `xhost` nevű programmal a kovetkező módon;

```
[cel@pip /]$ xhost forras.jpte.hu
forras.jpte.hu being added to access control list
[root@mad /root]#
```

Ezek után a `cel` nevű gép számára engedélyezett a hozzáférés az „forras” képernyőjéhez. Ha most a `-display` opcióval indítunk el programot a „forras” -on, akkor az a "cel" nevű gép képernyőjén jelenik meg.

```
[forras@pip /]$ xeyes -display cel.jpte.hu:0.0
```

A példában látható `xeyes` nevű program két szemet jelenít meg a kepernyőn, amelyek követik az egérkurzort.

Fejezet 6

A Netscape

6.1 A WWW

A WWW (*World Wide Web*, világszéles pókháló) az Internet legismertebb területe, olyannyira, hogy sokan az egész Internetet azonosítják vele. Technikailag a WWW egy – az egész világot behálózó – hipertext rendszer.

A hipertext

Fejezet 7

A szövegszerkesztés

Szövegszerkesztés alatt betűkből, írásjelekből és számokból álló állományok begépelését és átalakítását értjük. Fontos figyelembe vennünk, hogy a szövegszerkesztés folyamata nem tartalmazza a szöveg tipográfiai megmunkálását, vagyis a nyomdai termék külalakjának megtervezését és elkészítését. Leginkább úgy lehetséges megérteni a szövegszerkesztés és kiadványszerkesztés közti különbséget, ha figyelembe vesszük, hogy a szövegszerkesztés a szerző szándékát hivatott kifejezni, míg a kiadványszerkesztés a nyomdás, a tipográfus feladata.

Ma már olyan nagy a számítógépek teljesítménye, hogy képesek a szövegszerkesztést és a kiadvány tipográfiai tervezését egyszerre – egyetlen kombinált program keretében – lehetővé tenni a felhasználó számára, de a szövegszerkesztő programok ideje azért még nem járt le. Több esetben is hasznos lehet az olyan program, amely a szövegszerkesztésen túl nem vállal más feladatot:

- Sokszor olyan szöveget készítünk, amely soha nem kerül nyomdába. Programokat készíthetünk, adatokat vihetünk be nyilvántartóprogramok részére, gépi úton továbbfeldolgozott szöveges állományt hozhatunk létre a szövegszerkesztővel anélkül, hogy valaha is papírra kívánnánk vinni a kész állományt.
- Sok esetben nem vállalja a szerző, hogy tipográfiai feladatokat is elvégezzen, ez esetben a szövegszerkesztő program számára ideális,

mivel sokkal egyszerűbb kezelni mint egy bonyolult tördelőprogramot.

- Sok professzionális tördelőprogram létezik, amely egyáltalán nem alkalmas arra, hogy szövegszerkesztőként használjuk. Ilyenkor kedvenc szövegszerkesztőnkkel dolgozhatunk a szöveg elkészítésekor.

Igen sok szövegszerkesztő használható Linux alatt, némelyek pl. a programírásnál használhatóbbak, mások a rugalmasságukkal tűnnek ki vagy éppen az egyszerű kezelhetőséggel nyerik el tetszésünket. Aki sokat – és sokféle dologra – használja a Linuxot, az általában több szövegszerkesztő programot használ rutinosan, a feladat jellegéhez – és a környezet adta lehetőségekhez – igazodva választ közülük az adott pillanatban. Megteheti, hiszen a szövegszerkesztők az adatok tekintetében egymással felcserélhetőek, nincsen saját állományformátumuk. Amit az egyik szövegszerkesztővel elkezdtünk, azt egy másikkal folytathatjuk.

Minden Linuxos szövegszerkesztő az ASCII (American Standard Code for Information Interchange) kódolást használja. E kódolási forma igen régi, a Unix születésekor már ismeretes volt. Az ASCII kódolás 128 jel pontos leírására alkalmas, amely elegendő az angol nagy- és kisbetűk, az írásjelek, a számok és jónéhány különleges jel egyértelmű reprodukálására. A későbbiekben bevezetett ún. kiterjesztett ASCII kódolás további jeleket vezetett be, így már – 256 féle jellel – az angol nyelvben nem használatos nemzeti karaktereket is képes volt kezelni. Sajnos a kiterjesztett ASCII már koránt sem annyira egyértelműen értelmezett szabvány, ezért az egyes operációs rendszerek között zavaró ellentmondások léptek fel. A kiterjesztett ASCII használatakor – pl. magyar nyelven írt szöveg esetében – kódkonverzióra van szükség ha a szöveget más operációs rendszerre akarjuk átvinni. Szerencsére a Unix és Linux egyes változatai között példás az egyetértés...

7.1 A `vi`

Az első szövegszerkesztő amelyet ismertetünk, a `vi`, melynek nevét hagyományosan betűzve ejtjük ki az angol fonetikai szabályok szerint. A `vi` rövidítés, amely a *visual editor* (látható szövegszerkesztő) elnevezésből ered, nevét onnan kapta, hogy a `vi` volt az első olyan UNIX

alapú szövegszerkesztő, amely a szerkesztett szöveg több sorát is megmutatta a munka közben[1].

A `vi` jónéhány előnye indokolja a kezelésének – legalább alapfokú – elsajátítását:

- A `vi` minden Unix alapú rendszeren megtalálható, ezért ha ismerjük, akkor nem érhet meglepetés – bárhová sodojon sorsunk.
- A `vi` lassú telefonvonalakra készült, azért kiválóan alkalmas távoli gépeken történő munkára, így a segítségével lehetségessé válik hosszú dokumentumok módosítása akkor is, ha az egy másik kontinensen található s csak egy telefonvonal áll a rendelkezésünkre.
- A `vi` tervezésénél figyelembe vették, hogy a kereskedelemben kapható számítógépbillentyűk a legváltozatosabb formát mutatják. Használatakor pl. akkor sem kell kétségbe esnünk, ha a billentyűzet egyáltalán nem tartalmazza azt a nyomógombot, amellyel a hibásan begépelt betűket szoktuk kitörölni.
- A `vi` igen komplex módon is használható. Ha professzionálisan kezeljük hatékony eszközzé válik kezünkben.
- Végül, de talán nem utolsó sorban a vi a UNIX felhasználók beavatási szertartásának része. Ha nem tudjuk kezelni – legalább elemi szinten – akkor biztosan lesznek olyanok a környezetünkben, akik megmosolyognak és elmés megjegyzéseket tesznek az igazi tudás mibenlétéről. Másrészről aki idegen helyen a `vi` segítségével lát munkához, méltán számíthat a környezet elismerésére.

A `vi` rendelkezik egy apró hibával is, nevezetesen igen nehéz megtanulni a használatát. Szerencsére ez inkább az alapfunkciókon túl beépítésre került extrákra vonatkozik, ameleyek nem elengedhetetlenül fontosak. Mindezeket figyelembe véve a `vi` legfontosabb eszközeit vesszük sorra, a bonyolult parancskombinációkat pedig mellőzzük – hiszen a legtöbb feladat nélkülük is elvégezhető.

[1] A vi elődei olyan soralapú szövegszerkesztők voltak, amelyek egyszerre csak egy sort mutattak meg a szövegből, s amelyeket ebből adódóan igen nehéz volt kezelni.

7.1.1 A vi indítása és leállítása

A vi a vi parancs begépelésével indítható s a parancssorban mindjárt megadhatjuk a szerkeszteni kívánt állomány nevét is:

```
[root@pip /root]# vi allomany
```

Ekkor a vi elindul, s ha létezik a parancssorban beírt állomány a képernyőre írja. Amennyiben a beírt néven nem található állomány, akkor üres szöveges állományt kezd – amelyet nem hoz létre csak akkor ha parancsot kap rá az első mentés során. Mivel a vi az üres sorokat a ~ karakterrel jelzi, üres állomány esetén a következő kép tárul elénk:

```
~
~
~
~
~
~
~
"allomany" [New File]
```

A vi leállítása – most még magyarázat nélkül – kétféle módon történhet. Ha menteni kívánunk, akkor le kell ütnünk kétszer az Esc billentyűt, aztán begépelnünk a :wq betűket, majd lenyomni az entert. Ha a mentésről le kívánunk mondani, akkor az Esc kétszeri lenyomását kövesse a :q! begépelése, majd az Enter lenyomása.

7.1.2 A vi állapotai

A vi többféle állapottal rendelkezik, amelyek meghatározzák, hogy a leütött billentyű milyen változást okoz a szövegken. A legelemibb munka – egyetlen betű beszúrása egy meglévő szövegbe – sem képzelhető el a vi három alapvető állapotának megismerése nélkül, ezért fontos, hogy megértsük ezeket. A három alapvető állapot a következő:

normál mód A vi indítás után normál módban van. Ilyenkor a szöveg a képernyőn látszik, a kurzorral a szövegben mozoghatunk,

szöveget azonban begépelni nem tudunk. Normál állapotban ugyanis a leütött betűket a vi parancsként értelmezi, amelyek a szövegben változást idéznek elő – pl. kitörölhetünk egy sort vagy egy betűt a segítségükkel.

beszúrás mód Ebben az állapotban írhatunk be szöveget az állományba, vagyis ekkor működik a vi írógépként.

parancs mód Ebben az állapotában a vi az alsó sorban parancsot vár, amely általában több betűből áll. A parancs állapotban menthetjük pl. az állományunkat vagy léphetünk ki a `vi` -ból.

A vi indításkor normál állapotba kerül – gépelni tehát nem lehet ilyenkor. Ha gépelni szeretnénk, akkor meg kell nyomnunk az `i` betűt[2], ez szolgál ugyanis a neszúrás állapotba váltásra – normál állapotból. Ha a beszúrás állapotba lépünk, akkor – hacsak nincs kikapcsolva ez a szolgáltatás – az alsó sorban megjelenik a `--INSERT--` szöveg, jelezve, hogy a gépelést megkezdhetjük.

Beszúrás állapotból visszatérni a normál állapothoz az Esc billentyű lenyomásával lehet. Ekkor az alsó sorból eltűnik az `--INSERT--` felirat jelezve, hogy visszatértünk a normál állapotba. Ha a billentyűzeten nincsen Esc feliratú nyomógomb, akkor használhatjuk a Ctrl+[kombinációt helyette.

Parancs állapotba a normál állapotból juthatunk olymódon, hogy kettőspontot gépelünk. Ekkor a kurzor a képernyő aljára ugrik s a vi parancsot vár. Amikor a parancs begépelése után lenyomjuk az Enter billentryűt, a vi a parancsot végrehajtja és automatikusan visszatér a normál állapothoz. Ha a parancsot – amit parancs állapotban az alsó sorba gépeltünk be – nem akarjuk végrehajtani, hanem vissza akarunk térni a normál állapotba, kétszer egymás után le kell nyomnunk az Esc billentyűt (vagy az Esc -t helyettesítő Ctrl+[kombinációt).

7.1.2.1 A normál mód

Normál állapotban a szövegben mozoghatunk, sorokat, szavakat vagy betűket törölhetünk. A normál állapotban használható billentyűparancsok egyszerű betűkből és írásjelekből állnak, így alkalmazhatjuk őket

[2] Az i betű az *insert* (beszúrás) rövidítése.

akkor is, ha a billantyűzeten nem találhatóak vagy nem működnek a különleges billentyűk. Különösen hasznos ez azért, mert a Unix világban igen sokfajta számítógépbillentyűzet használható, amelyek között sajnos sokszor fellépnek inkompatibilitások. Ha egy távoli gépre belépve – a beállítás hiányosságai miatt – a Delete, Backspace, Home stb. billentyűk nem működnek megfelelően, a `vi` normál állapotát bizton használhatjuk.

A legfontosabb műveletek – amelyeket normál állapotban elvégezhetünk –, a következőek:

mozgás betűnként A betűnként való mozgásra a `h`, `k`, `l`, `j` betűk lenyomása alkalmas. A `h` betű balra, az `l` jobbra mozgat, míg a `k` betűvel felfelé a `j` betűvel pedig lefelé haladhatunk.

mozgás szavanként A `b` betűvel a szöveg eleje felé, az `e` betűvel pedig a vége felé haladhatunk szavanként.

ugrás sorban A sor elejére a `0` megnyomásával, végére pediga `$` jel begépelésével ugorhatunk.

görgetés A képernyőn található szöveget felfelé és lefelé götgethetjük soronként. A Ctrl+y kombinációval a szöveg eleje felé haladhatunk, míg a Ctrl+e segítségével a vége felé.

törlés betűnként A kurzor felett található karaktert az `x` betű lenyomásával törölhetjük. Ha lenyomjuk az `r` betűt, akkor a vi a kurzor felett található karaktert kicseréli arra a beűre, amelyet az `r` betű után nyomunk le.

törlés soronként Egész sort a `d` betű kétszeri lenyomásával törölhetünk.

visszavonás Ha valamely változtatást vissza szeretnénk vonni, akkor az `u` betűt kell lenyomnunk. A vi lehetőseéget ad többváltoztatás – fordított sorrendben történő – visszavonására az u betű többszöri lenyomásával. A visszavonást is vissza lehet vonni, ha úgy döntünk, hogy a változtatást mégis érvényesíteni akarjuk, akkor a Ctrl+r kombinációt kell lenyomnunk.

újrarajzolás Ha valamely üzenet a képernyőn látható szöveget összezavarta, akkor használhatjuk a Ctrl+l kombinációt a képernyő újrarajzolására.

beszúrás Az `i` betű megnyomásával kapcsolhatunk át beszúrás üzemmódba, amely szöveg begépelésére szolgál.

A vi lehetőséget ad arra, hogy a szöveg bizonyos részletét megjelöljük, majd azon végezzünk műveletet. A kijelölt szöveg ellentétes színezéssel jelenik meg, a vi megcseréli az előtér és háttér színét a kijelölt szöveg területén.

Néhány parancs – pl. a betűk törlésére szolgáló `x` –, értelme annyiban változik ha a szövegben kijelölés van, hogy a művelet automatikusan a kijelölt szövegen hajtódik végre.

kijelölés A kijelölést normál módban kell megkezdenünk. A kurzorral a kijelölni kívánt szöveg elejére kell állnunk és a v betű megnyomásával megkezdenünk a kijelölést. ha ezek után a mozgató parancsokkal a kurzort a kijelölendő szövegrészlet végére visszük, a kijelölést elvégezhetjük. Megkezdett kijelölést félbeszakítani az Esc gomb megnyomásával lehet.

blokkos kijelölés Blokkos kijelölésre a Ctrl+v kombinációval van lehetőségünk. Ekkor a kijelölt szöveg a képernyőn található téglalap lehet.

A vi segítségével egyszerre több szöveges állomány is szerkeszthető, sőt arra is lehetőségünk van a hhasználatakor, hogy egyszerre több szöveges állományt egyetlen képernyőn szerkesszünk.

A képernyő több részre osztására a vi Ctrl+w parancsai szolgálnak, amelyek két részből állanak. Először mindig meg kell nyomnunk a Ctrl+w kombinációt – jelezve, hogy az ablakokra vonatkozó parancs következik –, majd még egy betűt külön. A legfontosabb ablakkezelő parancsok – amelyeket normál módban használhatóak – a következőek:

Ctrl+w n Új ablak nyitása.

Ctrl+w c Az aktuális ablak – amelyben a kurzor éppen tartózkodik – bezárása.

Ctr+w w A következő ablakra ugrás. E parancs segítségével juttathatjuk át a kurzort másik ablakba.

Ctrl+w + Az aktuális ablak méretének növelése.

Ctrl+w - Az aktuális ablak méretének csökkentése.

Ctrl+w = Az ablakok méretének elrendezése olymódon, hogy minden ablak egyenlő mértékben osztozzék a képernyőn.

7.1.2.2 Beszúrás mód

Amikor normál módban megnyomjuk az `i` billentyűt, a `vi` átkerül beszúrás állapotba. Ez az állapot szolgál a gépelésre, vagyis ilyenkor a `vi` írógépszerűen viselkezik.

A beszúrás állapotban is lehetőségünk van általában a szövegben való haladásra és javításra a különféle különleges billentyűkkel. Ha ezek a billentyűk nem a várt hatást produkálják, esetleg egyáltalán semmi változást nem produkálnak, kénytelenek vagyunk a normál mód betűparancsait használni.

Beszúrás üzemmódból normál módba az Esc megnyomásával juthatunk.

7.1.2.3 Parancs mód

Parancs mód használatakor a vi lehetőséget ad az alsó sorban komplex parancsok begépelésére, amelyek általában az állománykezelő műveletekre adnak módot.

Parancs módba normál módból a : megnyomásával juthatunk. Amikor a parancs begépelését befejeztük és azt az Enter gombbal végrehajtottuk, automatikusan visszakerülünk nomál módba. Ha a parancs végrehajtása nélül szeretnénk visszajutni normál módba, az Esc gombot kell kétszer egymás után lenyomnunk.

x Az állomány mentése – ha az módosítva lett –, majd kilépés.

w Az állomány mentése.

w nev Az állomány mentése a `nev` néven.

w! Az állomány erőltetett mentése. Akkor használhatjuk, ha az állomány írásvédett, de egyébként a mentése lehetséges.

wa Az összes módosított állomány mentése.

wq Az állomány mentése és kilépés.

q Kilépés.

q! Kilépés mentés nélkül.

help Az angol nyelvű dokumentáció megnyitása egy új ablakban.

ascii A kurzor felett található karakter kódját írja ki az alsó sorban – az ASCII kódolás szerint.

7.2 Az xemacs

Az xemacs igen fejlett és sokrétű szövegszerkesztő program – a szövegszerkesztők svájci bicskája. Nem csak szövegszerkesztésre, hanem az Interneten való böngészésre, elektronikus levelezésre, Internet újság olvasásra, a Linux dokumentáció olvasására sőt játékra is használható.

7.2.1 Mozgás a szövegben

Az xemacs használata közben a következő billantyűzetkombinációkkal haladhatunk a szövegben:

Sor eleje Home billentyű vagy Ctrl+a

Sor vége End vagy Ctrl+e

Jobbra Kurzormozgató nyíl vagy Ctrl+f

Balra Kurzormozgató nyíl vagy Ctrl+b

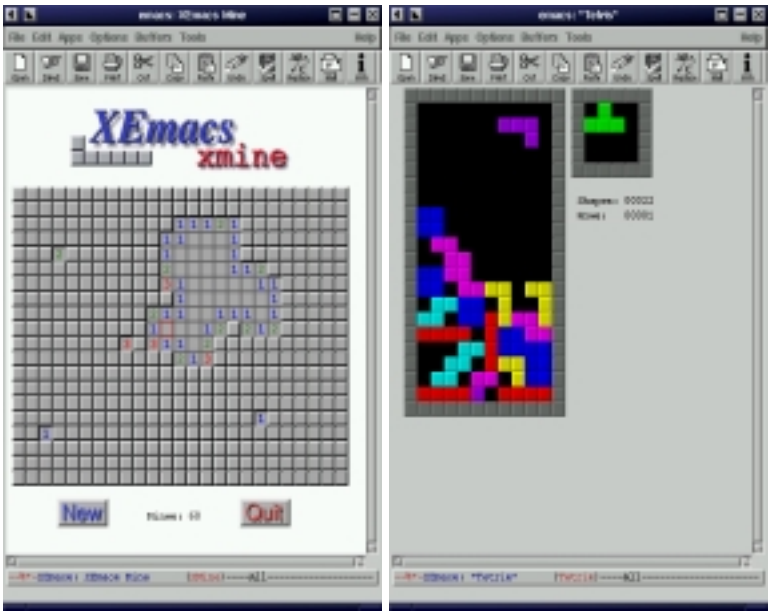
Egy szó előre Ctrl+Kurzormozgató nyíl vagy Alt+f

Egy szó vissza Ctr+kurzormozgató nyíl vagy Alt+b

Lefelé Kurzormozgató nyíl vagy Ctrl+d

Felfelé Kurzormozgató nyíl vagy Ctrl+p

Egy lap lefelé PageDown vagy Ctrl+v



ábra 7.1: Az xemacs

Egy lap felfelé PageUp vagy Alt+v

Első betűhöz Ctrl+Home

Utolsó betűhöz Ctrl+End

Ugrás sorra Alt+x. A billentyűzetkombináció megnyomása után az xemacs a felhasználótól a sor számát kéri, majd oda ugrik.

7.2.2 Törlés a szövegből

Delete A kurzor előtti karakter törlése.

Ctrl+d A kurzor utáni karakter törlése.

Ctrl+k A kurzor utáni szöveg törlése a sor végéig.

Alt+d A kurzor utáni szó törlése.

Alt+Delete A kurzor előtti szó törlése.

Fejezet 8

Kiadványszerkesztés

8.1 Bevezetés

A kiadványszerkesztés az a feladat, amelyre a számítógépet leggtöbbször használják napjainkban. Természetesen a LINUX is rendelkezik kiadványszerkesztő rendszerrel, amellyel professzionális nyomdatechnikai termékeket állíthatunk elő.

LINUX alatt a leghasználhatóbb tördelőprogram a TEX, amely ingyenesen elérhető és felhasználható. A programot DONALD E. KNUTH, világhírű matematikus fejlesztette ki, majd később sokan javítgatták, bővítették. A szerző célja az volt, hogy személyi számítógépeken - amelyek akkoriban még korántsem voltak olyan fejlettek mint ma - használható tördelőprogramot alkosson, amely felveszi a versenyt a professzionális rendszerekkel és alkalmas matematikai szövegek szedésére is. Elmondhatjuk, hogy aki ma igényes matematikai kiadványt akar szerkeszteni, az a TEX rendszert használja.

Mindazonáltal a TEXsokban különbözik a személyi számítógépen elterjedt megoldásoktól, amennyiben nem kimondottan támogatja a WYSIWYG (*What You See Is What You Get*, Azt Kapod Amit Látsz) elképzelést. A TEX tulajdonképpen egy nyelv, amely alkalmas dokumentumok leírására, a felhasználónak pedig el kell sajátítani a nyelvet ha dokumentumokat kíván létrehozni. A TEX nyelvén megfogalmazott dokumentumból a TEX egy fordítási folyamat során - ha abban szintaktikai hibát nem talál -

egy dvi (Device Independent, Eszközfüggetlen) állományt generál, amelyet megnézhetünk a képernyőn vagy kinyomtathatunk a nyomtatón.

Mindebből következik, hogy az alkotó - miközben a szöveget készíti - nem látja olyan formában a szöveget, ahogyan a végző nyomtatásban meg fog jelenni. Az is látható, hogy a TEX segítségével csak akkor vagyunk képesek kiadványokat létrehozni, ha annak nyelvét elsajátítjuk, ami nem könnyű feladat. Amit viszont cserébe kapunk, az azonban megéri a fáradtságot. A TEXkorrekt nyomdaképes eredményt produkál, képes több száz oldalas képletekkel teletűzdelt könyv kezelésére, kottát, sakktáblát, grafikonokat előállítani, minden nyelv karakterkészletét tartalmazza, logikai, nyelvészeti szimbólumokkal éppen úgy dolgozik, ahogy a vonalkódok előállítására alkalmas szimbólumokkal.

Ehelyütt a TEXnem teljes egészében kerül ismertetésre - hiszen az maga is több kötetnyi tárgyalás igényelne - csak a legfontosabb részei. Szó esik azonban a LATEX rendszerről, amely a TEX egy bővített változata, valamint a LYX kiadványszerkesztő programról.

A LYX egy olyan kiadványszerkesztő program, amely a LATEX segítségével állít elő dokumentumokat, de többé kevésbe WYSIWYG rendszerű, vagyis képes a szerkesztés közben „élvezhető” módon megjeleníteni a készülő dokumentumot. A LYX ráadásul menüs-grafikus program, amely könnyen kezelhető azok számára is, akik nem töltöttek éveket a LATEXés a TEXmegismerésével. A LYX nagy erőssége, hogy a készülő szövegbe LATEXparancsokat szúrhatunk be a segítségével, így nem korlátozza a felhasználó képességeit.

8.2 A TEX

8.2.1 A munkamenet

A TEX bemenete egy szöveges állomány, amelyet bármely szövegszerkesztővel előállíthatunk. vegyük tehát elő a kedvenc szövegszerkesztőnket és készítsünk el egy filet, amelynek neve `elso-file.tex`[1], a tartalma pedig valahogy így néz ki.

```
Ez az elso file, amit letrehoztam.
```

[1] A TEX bemenetét képező fileok a konvenció szerint .tex végződésűek.

```
Fajdalom, meg ekezetek nelkul irok.
\bye
```

Ügyeljünk arra, hogy ékezetes betűket ne használjunk, az első és az utolsó sor után kihagyjunk egy üres sort és irományunk végét egy „\bye” szó zárja le.

Amint elmentettük a szöveget, indítsuk el a TEX programot és mindjárt adjuk meg neki a feldolgozni kívánt szöveges állomány nevét:

```
[root@pip Linux-book]# tex elso-fileom.tex
This is TeX, Version 3.14159 (C version 6.1)
(elso-fileom.tex
Babel <v3.6h> and hyphenation patterns for
american, german, loaded.
[1] )
Output written on elso-fileom.dvi (1 page, 304 bytes).
Transcript written on elso-fileom.log.
[root@pip Linux-book]#
```

Amint látjuk a TEX rendben lefutott, hibát nem talált és minden bizonnyal előállította a dvi állományt a szövegből. Az eszközfüggetlen dvi állományt megtekinthetjük az xdvi programmal az X WINDOW alatt.

```
[root@pip Linux-book]# xdvi elso-fileom.dvi
```

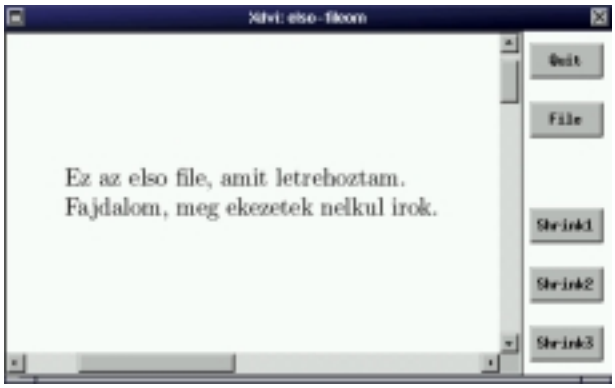
Ekkor a 8.1 ábrán látható látvány tárul szemünk elé.

Ha a képernyőn megcsodált állományt ki akarjuk nyomtatni, akkor a dvips nevű programmal azt megtehetjük:

```
[root@pip Linux-book]# dvips elso-fileom.dvi
```

Esetleg átalakíthatjuk POSTSCRIPT formátumúra is, hogy aztán a nyomdába küldhessük.

```
[root@pip Linux-book]# dvips -o nyomdanak.ps elso-fileom.dvi
```



ábra 8.1: Az xdvi

Ezzel be is mutattuk a TEX használatának alapelemeit. Létrehoztunk egy szöveges állományt, abban elhelyeztük a szöveget amelyet nyomdai formára kell hoznunk, majd azt feldolgoztuk a `tex` parancs begépelésével. A végeredményt megtekintettük a képernyőn majd kinyomtattuk azt. A további vizsgálódásaink a TEX nyelvének megismerését célozzák, amely lehetővé teszi a legváltozatosabb nyomtatási képek előállítását.

8.2.2 A TEX nyelv alapelemei

Első szövegünk - amely kissé igénytelenre sikeredett - már néhány tanulsággal járt. Az első amit meg kell szoknunk, az az, hogy a TEX a bemeneti állománya - a forrás - végén egy `\bye` parancsot vár. Ha ezt nem találja meg, akkor hibajelzéssel leáll, nem készíti el a kimeneti állományt.

Második megtanulni való az, hogy a *bekezdéseket* egy üres sor beszúrásával választjuk el egymástól, vagyis úgy, hogy *kétszer egymás után lenyomjuk az Enter billentyűt*. A sorok végén nyugodtan lenyomhatjuk az Entert, a szövegszerkesztőben új sor kezdve, a TEX nem veszi ezt figyelembe. Csak akkor kezd új bekezdést, ha két új-sor karaktert talál közvetlen egymás után.

Nem csak a sortöréseket - amelyeket az Enter beszúrásával a saját kényelmünkre elhelyeztünk - nem veszi figyelembe, de a *többlet szóközöket*

\'o=ó	\.o=ȯ	\b o=o̲
\‘o=ò	\H o=ő	\d o=ọ
\^o=ô	\u o=ŏ	\c o=ǫ
\"o=ö	\v o=ǒ	
\=o=ō	\t oo=o͡o	

Táblázat 8.1: Ékezetes betűk a TEX ben I.

is *figyelmen kívül hagyja*. Ha tehát két szó közé kettő vagy több szóközt teszünk, a végeredmény éppen az lesz, mintha csak egy szóközzel választottuk volna el e szavakat.

Mindezekből látjuk, hogy ha ékezetek nélküli - tehát pl. angol nyelvű - folyamatos szöveget kívánunk előállítani a TEX segítségével, akkor elegendő a fentiek figyelembevételével egyszerűen begépelni a szöveget, és a többit a gép elvégzi helyettünk.

A TEX ennél sokkal többre is képes, hiszen a szövegben parancsokat helyezhetünk el számára, amelyek a kiadvány formájára vonatkozó kívánalmainkat közvetítik számára. Az első parancsot már ismerjük is, ami nem más mint a dokumentum végét jelző \bye szócska. A parancsok minden esetben a \ jellel kezdődnek és minden ami e jellell kezdődik a TEX számára parancs - különleges értelemmel bír. Lássuk ezen parancsok egy csoportját, azokat amelyek lehetővé teszik számunkra, hogy ékezetes betűket hozzunk létre!

8.2.3 Ékezetes betűk és különleges karakterek

A TEX képes különleges betűk összeépítésére azok alapelemeiből, ezzel lehetővé teszi, hogy olyan betűformákat használjunk, amelyet esetleg még senki sem használt. Most néhány módszert fogunk megvizsgálni, amely lehetővé teszi ékezetes betűformák létrehozását.

Az első módszer amit sorraveszünk az *ékezet és mellékjel rátoldása* lesz. Ennek a műveletnek a során „hozzátoldunk” a már meglévő betűkhöz, hogy újabb betűformákhoz jussunk. A módszer roppant egyszerű: egy „\” jel után valamilyen ékezetet helyettesítő írásjelet írunk, majd azt a betűt, amelyet fel kívánunk ékesíteni ezzel az ékezettel. A lehetséges ékezetek a 8.1 táblázat mutatja.

\'v=v́	\.v=v̇	\b v=v̲
\‘v=v̀	\H v=v̋	\d v=ẏ
\^v=v̂	\u v=v̆	\c v=ȳ
\”v=v̈	\v v=v̌	
\=v=v̄	\t vv=v͡v	

Táblázat 8.2: Ékezetes betűk a TEX ben II.

Ez már így is több, mint amit sok kiadványszerkesztőtől kaphatunk, de a TEX nem áll meg itt. Mivel az ékezetet a betűre úgy tettük rá „kézzel”, másfajta betűre is rátehetjük azt. Olyan betűre is, amelyen talán még soha nem volt ez az ékezet (8.2 táblázat).

Az ékezet rátoldásának módszere különleges módon alkalmazható az í és Í betűk előállítására, mivel ezekről el kell távolítanunk előbb a pontot.

\' i{}=í

\' I{}=Í

E módszer után a következő amit megismerünk a *betűk összetoldása* lesz. Ezt már nem tehetjük meg bármely két betűvel, csak az alábbi kombinációk használhatóak.

\oe=œ	\ae=æ	\aa=å
\OE=Œ	\AE=Æ	\AA=Å

Amit pedig ezekkel a módszerekkel nem tudunk előállítani, arra általában egyedi parancsok vannak:

\o=ø	\L=Ł	\ddag=‡
\O=Ø	\ss=ß	\S=§
\l=ł	\dag=†	\P=¶

Természetsen senki nem várhatja el, hogy amikor magyarul írunk, akkor ilyen ákombákomokat tegyünk minden ékezetes betűre.

nos ezt itt ki kell találni, legyen itt a HION?

8.2.4 Csoportok képzése

A TEX számára a { és } jeleknek különleges jelentősége van. A kapcsos zárójelekkel csoportokat alkothatunk, amelyek sok tekintetben egyetlen egységként viselkednek. A kapcsos zárójelek sok parancs esetében használatos, velük jelezhetjük a határokat, vagyis, hogy a parancs hatásköre meddig terjedjen.

A csoport alkalmazásának legegyszerűbb esete az, amikor egy-egy sort kívánunk rendezni. A `\centerline{Középre igazított sor}` hatására a sor középre kerül, a `\rightline{Jobbra igazított sor}` egy jobbra igazított sort hoz létre és értelemszerűen a `\leftline{Balra igazított sor}`balra rendez:

Középre igazított sor

Jobbra igazított sor

Balra igazított sor

A fenti példákon bemutatott három parancs csak sorok rendezésére használható, vagyis a kapcsos zárójelek közé írt szövegnek ki kell férnie egyetlen sorban.

A kapcsos zárójelek egymásbaágyazhatóak, ahogyan azt a matematikában a zárójelektől már megszoktuk. A következő példa ilyan egymásbaágyazást mutat a következő fejezetben bemutatásra kerülő `\bf` parancs használatával. A `\centerline{{\bf Középre} igazított sor}` parancs hatására olyan - középre igazított - sor jelenik meg, amelynek egy része félkövér betűvel jelenik meg a `\bf` miatt:

Középre igazított sor

8.2.5 Betűtípusok

Kapcsos zárójeleket használunk általában a betűforma megváltozatatásánál is, hiszen amikor betütípust váltunk, akkor ezt általában valamilyen szövegrészletre kívánjuk alkalmazni. Ki akarunk emelni egy szót, dőlten kívánunk szedni egy bekezdést, a megoldás mindig csoportok segítségével a legegyszerűbb. Ha { és } jelek közt megváltoztatjuk a betűformát, akkor az új betűforma csak e jelek közt - a csoportra - lesz érvényes, a záró }jel után a TEX visszzatér eredeti betűformájához.

Az betűforma a *roman*, amely alkalmas hoszabb szöveg írására, mivel jól olvasható, nem fárasztja a szemet. Áttérni erre a betütípusra a `\rm`

paranccsal lehetséges, bár a TEX mindig ezt a betütípust használja ha másképp nem rendelkezünk.

Második forma a *bold face*, vagyis a kövérített, **vastag** betű. Ez típus általában címekben használatos, mivel meglehetősen figyelemfelkeltő, harsány. Sokan szöveg közben is használják szavak, mondatrészek kiemelésére, bár ez csúnya írásképet ad, nehezen olvashatóvá teszi a szöveget, ezért általában nem ajánlott. A vastagított betűt a TEXesetén a `\bf` paranccsal használhatjuk. A fenti szövegrészlet a csoporthatároló jelek segítségével csak a „vastag” szót tette kövérré: „`...kövérített, {\bf vastag} betű.`”

Mondatrészek, szavak kiemelésére az *italic* betűforma használható, amelyet *magyarul kurzív szedésnek* nevezünk. Nem használható a kurzív szedés hosszú szövegrészletek, egész bekezdések szedésére, mert elveszti figyelemfelkeltő hatását és fárasztja az olvasót. Kurzív betűket az `\it` paranccsal készíthetunk, a `{\it italic}` hatása pl. *italic* lesz a tördelés után.

Ha hoszabb szövegeket kívánunk kiemelni, akkor a slanted, vagy dőlt betűhöz folyamodunk. A parancs `\sl` lesz, melynek hatására a szöveg dőlt lesz ugyan, de olvashatóbb mint a kurzív szedésnél. A `{\sl slanted}` parancs hatására tördelés után *slanted* formában jelenik meg a szöveg.

Különleges esetben szükségünk lehet olyan betűformára, amelynek minden betűje ugyanakkora szélességű helyet foglal el a papíron. Ilyen *monospace* betűforma a \tt paranccsal kapcsolható be, neve pedig typewriter (írógép). A `{\tt typewriter }` parancs hatása: `typewriter`. E betűformát használhatjuk számítógépes példák szedésére, mivel a számítógép - ha nem grafikus üzemmódban van - szintén egyforma szélességű betűket ír a képernyőre.

A számok írására használható külön stílus az ún. „*ugráló számok*”. Normális esetben a számok mindegyike egyforma szélességű és magasságú (12234567890), ami könnyűvé teszi ugyan a számok helyiérték szerinti egymás alá rendezését, de nem túl szép. A TEX esetében használhatjuk a {\oldstyle 1234567890} számokat, mégpedig az `\oldstyle` parancs segítségével. A fenti példa tehát a következőképpen készült: `{\oldstyle 1234567890}`

Általában a fent bemutatásra kerülő betütípusok elegendőek a szöveg szedésére, mindazonáltal használhatunk extra írásképet, amely eltér a vázolt formáktól.

8.2.6 Különleges bekezdések

A TEX lehetővé teszi különleget bekezdések létrehozását, amelyek igen praktikusan használhatóak szokványos írási kép előállítására.

Ilyen különleges bekezdés a *lábjegyzet*, melyet igen egyszerűen helyezhetünk el a szövegben. A következő parancs egy automatikusan sorszámozott lábjegyzetet szúr a szövegbe:

```
\footnote{A lábjegyzet szövege...}
Az eredmény pedig valahogy így[2] néz ki.
```

Könnyedén hozhatunk létre felsorolásokat az \item paranccsal.

```
\item{1.} A felsorolás
  \itemitem{a)} A felsorolás
  \itemitem{b)} A felsorolás
\item{2.} A felsorolás
```

8.2.7 A matematikai mód

Ahhoz, hogy matematikai képleteket szerkesszünk a szövegbe át kell térnünk matematikai üzemmódba. matematikai módból mindjárt kettőféle is rendelkezésünkre áll, egyik a sorközi mód, (pl: $a^2 + b^2 = c^2$) másik pedig a kiemelt matematikai mód, ahol a matematikai kifejezések nem a szövegben jelennek meg. Az előbbi képlet kimenelt módban:

$$a^2 + b^2 = c^2$$

Sorközi matematikai módba jelenik meg minden, amit két „$” jel közé írunk, valamint kiemelt módban a „$$” jelek közti képletek lesznek. A fenti képlet sorközi módban „$a^2+b^2=c^2$”, kiemelt módban pedig „$$a^2+b^2=c^2$$” formában adható meg.

E kétfajta matematikai mód csak megjelenésében különbözik egymástól, ugyanazokat a képleteket kiszedhetjük sorközi és kiemelt matematikai módban is, a különbség a megjelenés csekély módosulásában fog csak jelentkezni. Erre azért van szükség, mert a sorok közé írt matematikai

[2] A lábjegyzet szövege...

\alpha=α	\iota=ι	\varrho=ϱ
\beta=β	\kappa=κ	\sigma=σ
\gamma=γ	\lambda=λ	\varsigma=ς
\delta=δ	\mu=μ	\tau=τ
\epsilon=ϵ	\nu=ν	\upsilon=υ
\varepsilon=ε	\xi=ξ	\phi=ϕ
\zeta=ζ	\o=	\varphi=φ
\eta=η	\pi=π	\chi=χ
\theta=θ	\varpi=ϖ	\psi=ψ
\vartheta=ϑ	\rho=ρ	\omega=ω

Táblázat 8.3: Görök kisbetűk a a TEX matematikai üzemmódjában

\Gamma=Γ	\Xi=Ξ	\Phi=Φ
\Delta=Δ	\Pi=Π	\Psi=Ψ
\Theta=Θ	\Sigma=Σ	\Omega=Ω
\Lambda=Λ	\Upsilon=Υ	

Táblázat 8.4: Görök nagybetűk a TEX matematikai üzemmódjában

képleteknek sokkal kisebb helyen kell elférniük, mint kiemelt társaiknak. A kiemelt módban írt képletek sorszámozhatjuk, hogy később hivatkozhassunk rájuk.

Fontos megjegyeznünk, hogy minden matematikai kifejezést matematikai módban kell írni, akkor is, ha látszólag normál szövegként is be tudnánk írni amit szeretnénk. A matematikai mód ugyanis más formában jeleníti meg a betűket, így az „a” nem tehető „a” helyére, mert értelemzavaró lehet.

8.2.8 Különleges jelek

Ha matematikai módban vagyunk, akkor jónéhány új szimbólummal dolgozhatunk. A 8.3, 8.4, 8.5, 8.6, 8.7, 8.8 táblázatok a használható szimbólumokat – és a hozzájuk tartozó parancsokat – mutatják be.

\pm=±	\cup=∪	\wedge=∧
\mp=∓	\uplus=⊎	\land=∧
\setminus=\	\sqcap=⊓	\oplus=⊕
\cdot=·	\sqcup=⊔	\ominus=⊖
\times=×	\triangleleft=◁	\otimes=⊗
\ast=∗	\triangleright=▷	\oslash=⊘
\star=⋆	\wr=≀	\odot=⊙
\diamond=⋄	\bigcirc=◯	\dagger=†
\circ=∘	\bigtriangleup=△	\ddagger=‡
\bullet=•	\bigtriangledown=▽	\amalg=∐
\div=÷	\vee=∨	
\cap=∩	\lor=∨	

Táblázat 8.5: Operátorok a TEX matematikai üzemmódjában

\le=≤	\ge=≥	\sim=∼
\prec=≺	\succ=≻	\simeq=≃
\preceq=⪯	\succeq=⪰	\asymp=≍
\ll=≪	\gg=≫	\approx=≈
\subset=⊂	\supset=⊃	\cong=≅
\subseteq=⊆	\supseteq=⊇	\bowtie=⋈
\sqsubseteq=⊑	\sqsupseteq=⊒	\propto=∝
\in=∈	\ni=∋	\models=⊨
\vdash=⊢	\owns=∋	\doteq=≐
\smile=⌣	\dasv=⊣	\perp=⊥
\frown=⌢	\mid=∣	\parallel=∥
\geq=≥	\equiv=≡	

Táblázat 8.6: Relációs jelek a TEX matematikai üzemmódjában

\not\le=$\not\le$	\not\ge=$\not\ge$	\not\sim=$\not\sim$
\not\prec=$\not\prec$	\not\succ=$\not\succ$	\not\simeq=$\not\simeq$
\not\preceq=$\not\preceq$	\not\succeq=$\not\succeq$	\not\asymp=$\not\asymp$
\not\ll=$\not\ll$	\not\gg=$\not\gg$	\not\approx=$\not\approx$
\not\subset=$\not\subset$	\not\supset=$\not\supset$	\not\cong=$\not\cong$
\not\subseteq=$\not\subseteq$	\not\supseteq=$\not\supseteq$	\not\bowtie=$\not\bowtie$
\not\sqsubseteq=$\not\sqsubseteq$	\not\sqsupseteq=$\not\sqsupseteq$	\not\propto=$\not\propto$
\not\in=$\not\in$	\not\ni=$\not\ni$	\not\models=$\not\models$
\not\vdash=$\not\vdash$	\not\owns=$\not\owns$	\not\doteq=$\not\doteq$
\not\smile=$\not\smile$	\not\dashv=$\not\dashv$	\not\perp=$\not\perp$
\not\frown=$\not\frown$	\not\mid=$\not\mid$	\not\parallel=$\not\parallel$
\not\geq=$\not\geq$	\not\equiv=$\not\equiv$	

Táblázat 8.7: Inverz relációs jelek a TEX matematikai üzemmódjában

\uparrow=$\uparrow$	\Uparrow=$\Uparrow$	\longmapsto=$\longmapsto$
\downarrow=$\downarrow$	\Downarrow=$\Downarrow$	\mapsto=$\mapsto$
\updownarrow=$\updownarrow$	\Updownarrow=$\Updownarrow$	\hookleftarrow=$\hookleftarrow$
\nearrow=$\nearrow$	\nwarrow=$\nwarrow$	\hookrightarrow=$\hookrightarrow$
\searrow=$\searrow$	\swarrow=$\swarrow$	\rightharpoonup=$\rightharpoonup$
\longrightarrow=$\longrightarrow$	\rightarrow=$\rightarrow$	\rightharpoondown=$\rightharpoondown$
\longleftarrow=$\longleftarrow$	\leftarrow=$\leftarrow$	\rightleftharpoons=$\rightleftharpoons$
\longleftrightarrow=$\longleftrightarrow$	\leftrightarrow=$\leftrightarrow$	\leftharpoonup=$\leftharpoonup$
\Longrightarrow=$\Longrightarrow$	\Rightarrow=$\Rightarrow$	\leftharpoondown=$\leftharpoondown$
\Longleftarrow=$\Longleftarrow$	\Leftarrow=$\Leftarrow$	
\Longleftrightarrow=$\Longleftrightarrow$	\Leftrightarrow=$\Leftrightarrow$	

Táblázat 8.8: Nyílszerű jelek a TEX matematikai üzemmódjában

8.2.9 A matematikai mód parancsai

Matematikai módban a képletek alakját parancsokkal határozzuk meg. Tekintve, hogy a matematika mára hihetetlen mennyiségű eszközt és jelölésrendszert vezetett be, ez igen sok parancs megtanulását jelenti. A felhasználó általában elborzad a kézikönyvek és használati útmutatások alapján - amelyek a TEX teljes matematikai eszköztárát mutatják be. Ha még hozzátesszük, hogy ebből a nagyszámú paracsból igen bonyolult kifejezések - képletek - készíthetőek, akkor értehjük, hogy miért övezi a félelemteljes tisztelet azokat a felhasználókat, akik rutinosan - és megelégedve - használják a TEXet bonyolult matematikai munkák tördelésére.

Feltehetjük a kérdést, hogy ha ilyen bonyolult a matematikai szöveg tördelése TEX segítségével, akkor miért használjuk? A választás sokféleképpen magyarázható.

A TEX minden matematikai képlet kiszedésére alkalmas. Nem érhet bennünket az kellemetlen meglepetés, amikor a munkánk közepén vesszük észre, hogy az éppen szedésre kerülő képlet nem elkészíthető, mert a használt program nem ismeri azokat a speciális jeleket - vagy tördelési formát - amelyet mindenképpen használnunk kellene.

A TEX szedésképe jól átgondolt, stílusos és igen jól olvasható - egyszóval szép. Ha helyesen szedjük a matematikai jeleket, akkor a végeredményben a legtüzetesebb vizsgálat sem talál hibát. Minden jel századmilliméter pontossággal a megfelelő helyen van. Lássunk egy példát:

$$| x + 2 | + | sin\alpha + 1 |= 0$$

$$|x + 2| + |\sin\alpha + 1| = 0$$

A felső példa a rossz szedés miatt nehezen olvasható, csúnya, az alsó pedig pontos és jól áttekinthető, pedig gyakorlatilag ugyanazokat a jeleket tartalmazza mindkettő.

A TEX használata mellett szól az is, hogy a matematikai kifejezéseket egy parancsokból és szabályokból álló nyelv segítségével írja le. Mivel ez igen közel áll a matematikai gondolkodásmódhoz, hozzáértő számára nem okozhat gondot az elsajátítása. Aki a matematikában kiművelte magát, az valószínűleg két napi gyakorlás után örömet talál a TEX

matematikai rendszerével való munkában - ha a kézikönyvet még évekig az asztalán tartja is.

Ehelyütt nem vállalkozhatunk a teljes parancskészlet ismertetésére - hiszen az maga megtöltene egy terjedelmesebb könyvet -, de a legfontosabb parancsokon keresztül egy jól használható alapot kívánunk biztosítani, amellyel a matematikai képletek legtöbbje elkészíthető.

8.2.9.1 Az indexek

A matematikában igen sok jelet látunk el felső- és alsó indexekkel. Sokszor indexelünk indexben szereplő kifejezést, a tördelés közben gondoskodva róla, hogy az olvasó azonnal lássa melyik index hova tartozik.

Az index elkészítésére a „^” (felső index) és „_” (alsó index) jelek szolgálnak. A már bemutatásra került $a^2 + b^2 = c^2$ (`$a^2+b^2=c^2$`)példa jól mutatja a felső index használatát, ezúttal hatványozásra. A $n_i = k_j$ (`$n_i = k_j$`) az alsó index használatát példázza.

Amennyiben több karaktert - betűt vagy számot - kívánunk indexbe tenni, a csoportot kell képeznünk a már ismertetett módon - kapcsos zárójelekkel. Az $x_{12} \neq \pm k^{\sin \alpha}$ (`$x_{12}\eq\pm k^{\sin \alpha}$`) kifejezés ilyen csoprtokat használ felső- és alsó indexben.

Az indexek egymásba ágyazhatóak, ekkor azonban mindenképpen használnunk kell a kapcsos zárójelet, hogy egyértelműen jelezzük melyik index hova tartozik. $x_{k_i} \neq x^{sin^2 \alpha}$ (`$x_{k_i}\ne x^{\sin^2 \alpha}$`)

A TEX az indexek elkészítéséhez kétfajta betűméretet használ. Az első index készítéséhez csökkentett betűméretet használ, a második indexet - az index indexét - tovább kicsinyítve képezi. A további indexek már nem kisebbek, mivel a miniatűr betűk és számok olvashatatlanok volnának. Példa (`$$x^{(1 +x^2)^2}= 0$$`)

$$x^{(1+x^2)^2} = 0$$

Indexet tehetünk különféle kifejezésekre, arra azonban érdemes figyelnünk ebben, hogy a szövegközi matematikai módban helytakarékosság miatt nem úgy jelennek meg az indexek mint kiemelt módban. A $\sum_{i=1}^{n} \int_b^a f_i(x)dx$ (`$\sum_{i=1}^n \int^a_b f_i(x)dx$`) kifejezés kiemelt módban több helyet foglal, de olvashatóbb:

$$\sum_{i=1}^{n} \int_{b}^{a} f_i(x)dx$$

8.2.9.2 Az osztás

Az egyszerű osztást az `\over` paranccsal jelezhetjük. E parancs az előtte és utána található kifejezéseket egymás fölé írja és az osztást törtvonallal jelzi. Ha az \over előtt és/vagy után kapcsos zárójellel összefogott csoportok szerepelnek, akkor az osztás azok egészére vonatkoznak.

$$\frac{1+x}{1-x}$$

Az osztások egymásba ágyazhatóak, ekkor emeletes törtekhez jutunk. Példa:

$$\frac{1+\frac{1}{x}}{1-\frac{1}{x}}$$

Amennyiben az osztást valamilyen zárójelek közé kívánjuk tenni használhatjuk az `\overwithdelims` parancsot, amely után a zárójeleket meg kell adnunk. Az `$$1 \overwithdelims () {1+x}$$` hatására kerek zárójel kerül az osztás köré, de használható a <> és a [] páros is, amint azt a példa mutatja. (`$${1 \overwithdelims () {1+x}}\ne {1 \overwithdelims <> {1+x}}\ne {1 \overwithdelims [] {1+x}}$$`).

$$\left(\frac{1}{1+x}\right) \neq \left\langle\frac{1}{1+x}\right\rangle \neq \left[\frac{1}{1+x}\right]$$

Ha kapcsos zárójeleket kívánunk a tört köré rakni, akkor a kapcsos zárójeleket a \ jellel kell beírnunk, hogy a TEX azokat ne csoporthatároló jelként értelmezze (`$$1 \overwithdelims \{\}{1+x}$$`).

$$\left\{\frac{1}{1+x}\right\}$$

Az `\over` parancshoz hasonlóan használható a `\atop` parancs is, amely elhagyja a törtvonalat (`$$ n \atopwithdelims <> k+j $$`).

$$\left\langle \begin{matrix} n \\ k+j \end{matrix} \right\rangle$$

8.2.9.3 A gyökvonás

A TEX kétféleképp képes a gyökjelet rajzolni. A négyzetgyök esetében a kitevőt nem kell külön jelölni, ekkor az `\sqrt` (*square root*, négyzetgyök) parancsot használhatjuk. `$$ \sqrt {1+x} = \sqrt {1+x^2}$$`

$$\sqrt{1+x} = \sqrt{1+x^2}$$

Látható, hogy ha a gyökjel alá több jelből álló kifejezést akarunk tenni, akkor csoportot kell alkotnunk a kapcsos zárójelek segítségével. A gyökjelek egymásba ágyazhatóak, ügyelnünk kell azonban a helyes csoportosításra.

$$\sqrt{\sqrt{1+z}+z^2}$$

Ha a gyökjel kitevőjét is ki akarjuk írni, akkor a `\root` parancsot kell alkalmaznunk, amely esetben az `\of` paranccsal írhatjuk be a kitevőt.

$$\sqrt[3]{x} = 8$$

Természetesen mind a gyökjel alatti, mind pedig a kitevő lehet bonyolult kifejezés, ha kapcsos zárójeleket használunk.

$$\sqrt[n+1]{\sin\beta}$$

8.2.9.4 Ékezetek, csoportékezetek

8.2.10 A lapok kialakítása

8.3 A LATEX

A TEX igen jól bővíthető rendszer, parancsaiból kialakítható olyan csomag, amely nem tartalmaz tördelendő szöveget, csak olyan - a TEX nyelvén megfogalmazott - utasításokat, amelyek megkönnyítik a használatát.

Az évek során számtalan ilyen bővítés napvilágot látott, melyek közül kiemelkedik az általános célokra szolgáló LaTeX csomag. Nevét LESLIE LAMPORTról kapta, aki a LaTeX első változatát készítette.

A LaTeX célja az, hogy a felhasználó számára magasszintű parancsokat adjon, amelyek segítségével kevés munkával, egyszerűen lehet a dokumentumok formázását elvégezni. Ennek érdekében sílusokat deklarál, amelyek tartalmazzák a beállításokat, amelyek bizonyos dokumentumfajtáknál általánosan elfogadottak. A LaTeX a következő stílusokat ismeri:

`article` Az `article` (cikk) stílus rövid dokumentumok előállítására szolgál, amelyek a tudományos publikációknál megszokott formát követik. Az article stílus címe az első oldal tetején helyezkedik el, a szöveg pedig alatta - folytatólagosan -, vagyis a dokumentum nem tartalmaz külön fenntartott címlapot.

`report` Hosszabb cikkek számára készített stílus. A cím külön oldalon helyezkedik el, valamint lehetőségünk van felyezetekre osztani a dokumentumot `report` (riport) stílus használata esetén.

`book` A `book` (könyv) stílus hoszabb terjedelmű művek készítésére szolgál. Az oldalak formája a könyveknél megszokott kétoldalas nyomtatásra alkalmas.

`letter` A `letter` (levél) stílus mindent tartalmaz, amit levelek írásánál használunk. Címzés, dátum és aláírás könnyen elkészíthető e stílust használva.

A dokumentumstílusok *opciókkal* változtathatóak, hogy minél jobban megfeleljenek a kívánalmaknak. A dokumentum csak egyfajta stílust használhat - a fentiekből -, de többféle opcióval is módosíthatjuk azt. Az opciók a következőek:

`11pt` A dokumentum alap betűméretét 11 pontosra változtatja. Mivel az eredeti méret 10 pontos, a méretnövekedés 10% -os.

`12pt` Az alap betűméret 12 pontos méretű lesz.

`twoside` A dokumentum kétoldalas lesz, vagyis olyan formájú, hogy a bal és jobb oldalon rendre a páros és páratlan sorszámú oldalak jelenjenek meg. A book stílusnak ez az alapértelmezés szerinti formája.

`twocolumn` A dokumentum két hasábra tördelve készül el.

`titlepage` A cím külön oldalon jelenik meg. Ez az alapértelmezés szerinti formája a `book` és a `report` stílusoknak.

A LaTeX a fenti stílusokkal és stílusmódosulatokkal képes dokumentumokat előállítani a felhasználót nagymértékben segítő, magasszintű parancsokkal. Természetesen semmi olyasmit nem tesz, amelyet a TeX segítségével nem lehet megcsinálni - hiszen a LaTeX nem külön program, hanem a TeX nyelvén írt makrócsomag - de nagymértékben megkönnyíti a munkát. A kezdő felhasználó olyan dokumentumokat állíthat elő vele, amelyeket - hiányos ismeretei miatt - a TeXel nem tud elkészíteni.

8.3.1 A munkamenet

A dokumentum tördelése hasonlóképpen történik, mint ahogyan azt a TeXnél láttuk. A különbség csak annyi, hogy a tex parancs helyett most a latex paranccsal kell indítanunk a tördelést:

```
[root@pip Linux-book]# latex elso.latex.fileom.tex
This is TeX, Version 3.14159 (C version 6.1)
(elso.latex.fileom.tex
LaTeX2e <1996/12/01> patch level 1
Babel <v3.6h> and hyphenation patterns for
american, german, loaded.
(/usr/lib/texmf/texmf/tex/latex/base/article.cls
Document Class: article 1996/10/31 v1.3u Standard LaTeX
document class
(/usr/lib/texmf/texmf/tex/latex/base/size12.clo))
No file elso.latex.fileom.aux.
[1] (elso.latex.fileom.aux) )
Output written on elso.latex.fileom.dvi (1 page, 236 bytes).
Transcript written on elso.latex.fileom.log.
[root@pip Linux-book]#
```

Látható, hogy minden a megszokott módon történt, a szöveget a TeX tördelte a LaTeX bővítést felhasználva. A tördelt dokumentum az `elso.latex.fileom.dvi` eszközfüggetlen állományba került, amely a már ismert módon megtekinthető és feldolgozható.

A LaTeX fileok formátuma némiképp eltér attól, amit a TeXesetében megszoktunk. A minális LaTeXállománynak tartalmaznia kell három parancsot, ahogyan a következő példán látjuk:

```
\documentclass[12pt]{article}
\begin{document}

What's all about?

\end{document}
```

A `\documentclass` parancsnak mindig az állomány elején kell lennie, és a használni kívánt stílust beállítani. Ha a stílust módosítani kívánjuk a módosítószót is itt kell megadnunk kapcsos zárójelek közt - ahogy a példa mutatja -, de módosítót beírni nem kötelező.

A `\begin{document}`parancs a dokumentum kezdetét jelöli. Ettől a parancstól kezdve írhatjuk a tördelni kívánt szöveget és a külalakot módosító parancsokat.

Az `\end{document}`parancs a dokumentum végét jelöli, ami ezután következik, azt a LaTeXmár nem veszi figyelembe.

A példában látható „`What's all about?`" szöveg a dokumentum helyét jelöli.

8.3.2 Bekezdésstílusok

A LaTeXelőre definiál néhány bekezdésstílust, amelyek meghatározzák a bekezdés külalakját. A stílusok a következőek:

`quote` A quote (idéz) stílus akkor használatos, ha valaki szavait idézzük s ez az idézet rövid, egyetlen sorból vagy bekezdésből áll.

`quotation` A quotation (idézet) stílus akkor alkalmazható, ha az idézet több bekezdésből áll.

`center` A center (közép) stílus középre igazítja a sorokat.

`itemize` Az itemize (tételre bont) olyan felsorolások esetében hasznos, ahol nem akarjuk kiemelni a sorrendet. Minden bekezdés egy jellel kezdődik. A tételre bontást a bekezdések elején elhelyezett `\item` paranccsal kell kérnünk.

`enumerate` Olyan felsorolás készíthető az enumerate (megszámoz) stílussal, amely sorszámozott tételekből áll. A bekezdéseket - amelyeket számozni kívánunk - itt is az `\item` paranccsal kell kezdenünk.

`description` A description (leírás) olyan felsorolás amelynél a listaelemek szöveges jelöléssel kezdődnek, olyanok mint amilyen ez a felsorolás.

`verbatim` A verbatim (szó szerinti) stílus egyenlő szélességű betűket használ és nem végez tördelést. Arra szolgál, hogy a számítógép karakteres képernyőjén megejelenő szövegeket változatlan formában kinyomtathassuk.

A stílusok használata igen egyszerű. Arra a pontra, ahonnan át akarunk térni valamely stílus használatára, be kell szúrnunk a `\begin{stílusnév}`parancsot, ahonnan pedig a stílust már nem kívánjuk tovább használni oda az `\end{stílusnév}` parancsot kell betennünk.

példák.

8.3.3 Címhierarchia

A LaTeXképes a dokumentumban elhelyezett címeket hierarchikus rendben megjeleníteni, és kezelni - pl. tartalomjegyzésk készítéséhez. A címeket rendre `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\paragraph`, `\subparagraph` parancsokkal jelölhetjük csökkenő fontosságú sorba.

példa

8.3.4 Betütípusok és módosulatok

A LaTeXkönnyen hozzáférhető módon használ három alapvető betűcsaládot, amelyek a Roman, Sans Serif és Typewriter - valamint ezek módosulatait.

A szöveg alap betütípusa *Roman*. Hoszabb szöveget általában ilyen stílusban írhatunk, mert jól olvasható, nem fárasztja a szemet. Ez a szöveg is Roman betütípussal szedett, ahogyan azt már a TeXnél láttuk. Roman betütípusra a \rmfamily paranccsal kapcsolhatunk.

A \sffamily paranccsal kapcsolhatunk *Sans Serif* stílusú betűkre. E betütípusnak a jellegzetessége, hogy a betűk végződésének nincsen „talpacskája”, mint mint ahogyan azt itt láthatjuk. Ez a betűforma nehezebben olvasható, ezért hosszú szöveget szedni nem szokás a felhasználásával.

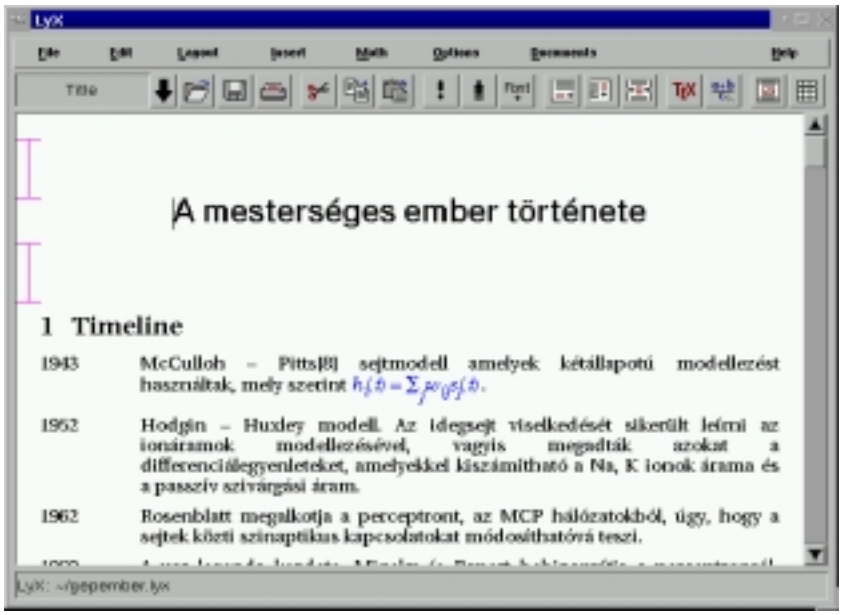
A *Typewriter* betűcsalád monospace (egyméretű) betűcsalád, amelynek az a különlegessége, hogy minden betűre ugyanakkora szélességű hely jut a papíron. Mivel nehezen olvasható csak akkor használjuk, ha valóban szükségünk van rá. Ez a betűcsalád a \ttfamily paranccsal érhető el.

A betűcsaládok *normál vastagságban és félkövéren* is elérhetőek. Az \mdseries parancs kapcsol normál vastagságra, míg a \bfseries fékövérré teszi a betűket. Mivel minden betűcsalád használható normál és félkövér formában - tehát létezik normál Roman és **félkövér Roman**, normál Sans Serif és **félkövér Sans Serif** valamint normál Typewriter és **félkövér Typewriter** - a félkövér módosító megduplázza a használható betűformák számát.

A betűk alakja lehet Normál, *Kurzív*, *Dőlt* és KIS KAPITÁLIS. A normál betűk alkotják az alap betűformát, a kurzív változattal egy-egy szót vagy mondatrészt emelünk ki, a dőlt betű hosszabb - több mondatnyi - szöveg kiemelésére szolgál, a kis kapitális pedig nevek szedésére elterjedten használt. Az \upshape, \itshape, \slhape, \scshape parancsok rendre normál, kurzív, dőlt és kis kapitális változatokat állítanak be.

8.3.5 A betűk mérete

A betűméret fokozatosan változtatható a \tiny, \scriptsize, \footnotesize, \small, \normalsize, \large, \Large, \LARGE, \huge és \Huge paranc-



ábra 8.2: A LyX

sokkal:

tiny, scriptsize, footnotesize, small, normalsize, large Large, LARGE, huge, Huge.

8.4 A LyX

A LyX egy kiadványszerkesztő program Linux alatt, amely nagymértékben támaszkodik a LaTeXtördelő rendszerre. Valójában a LyX önmagában nem is képes dokumentumokat tördelni, ehhez a LaTeXre van szüksége. A LyX erőssége minden bizonnyal abban van, hogy egy jól áttekinthető praktikus felületet ad a kezdő LaTeXfelhasználóknak ahhoz,

hogy ne kelljen a legegyszerűbb dokumentum előállításához parancsokat magolniuk[3].

Aki már használt kiadványszerkesztőt, valószínűleg kissé idegen lesz a LyX, mivel az alapkoncepciója meglehetősen egyéni. A szöveget többé kevésbé WYSIWYG módban ábrázolja, de nem teljesen abban a formában, ahogyan az a nyomtatás után megjelenik. A LyX készítői szerint programjukat használva több időt szentelhetünk a dokumentum tartalmára, mert a formával nem kell foglalkoznunk - azt a LaTeXhelyettünk megtervezi.

8.4.1 A LyX indítása

Ha elindítottuk a programot, akkor első lépésként meg kell nyitnunk egy már létező LyX dokumentumot vagy újat kell kezdenünk. A legutóbb használt dokumentumok a File menü alsó részén megtalálhatóak, tehát azokat igen könnyen meg tudjuk nyitni. Ha az általunk használni kívánt dokumentum itt nem szerepel, akkor a File menü Open menüpontjának segítségével megkereshetjük és megnyithatjuk azt.

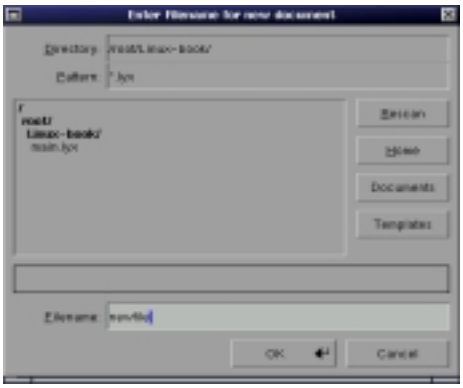
Ha még nincsen dokumentum amelyen dolgozhatunk, akkor újat kell létrehoznunk. Erre a File menü New... menüpontja szolgál. Ekkor a 8.3 ábrán látható ablak jelenik meg, amelynek segítségével új LyX dokumentumot kezdhetünk használni. Ha hoszabb dokumentumot tervezünk, amelyben esetleg képek is szerepelnek, akkor mindenképpen érdemes egy könyvtárat készíteni - amelyben a dokumentumhoz tartozó minden állomány helyet kap majd - és az új állományt ott létrehozni. Amikor a későbbiekben a dokumentumot szállítani kívánjuk, ajánlatos az egész könyvtárat lemásolni - mivel a LyX nem másolja be dokumentum minden részét egyetlen egy állományba.

8.4.2 magyarítása

8.5 A dokumentum utómunkálatai

A psutils *(postscript utilities*, postscript felhasználói programok) csomag segítségével lehetőségünk van POSTSCRIPT állományok manipulálására.

[3] A LaTeX parancsokat a LyX egyébként a dokumentumba képes illeszteni, ezért a profi felhasználók érdeklősdésére is számot tarthat.



ábra 8.3: Új LyX dokumentum

Sokszor szükségünk van arra, hogy a nyomtatás előtt *felére kicsinyítsük* a dokumentum fizikai méretét. Így az A4 méretű (210x297mm) oldalra két oldal fér rá és egyben felére csökken a szükséges papír mennyisége. A `pstops` programmal a következő módon tehetjük ezt meg:

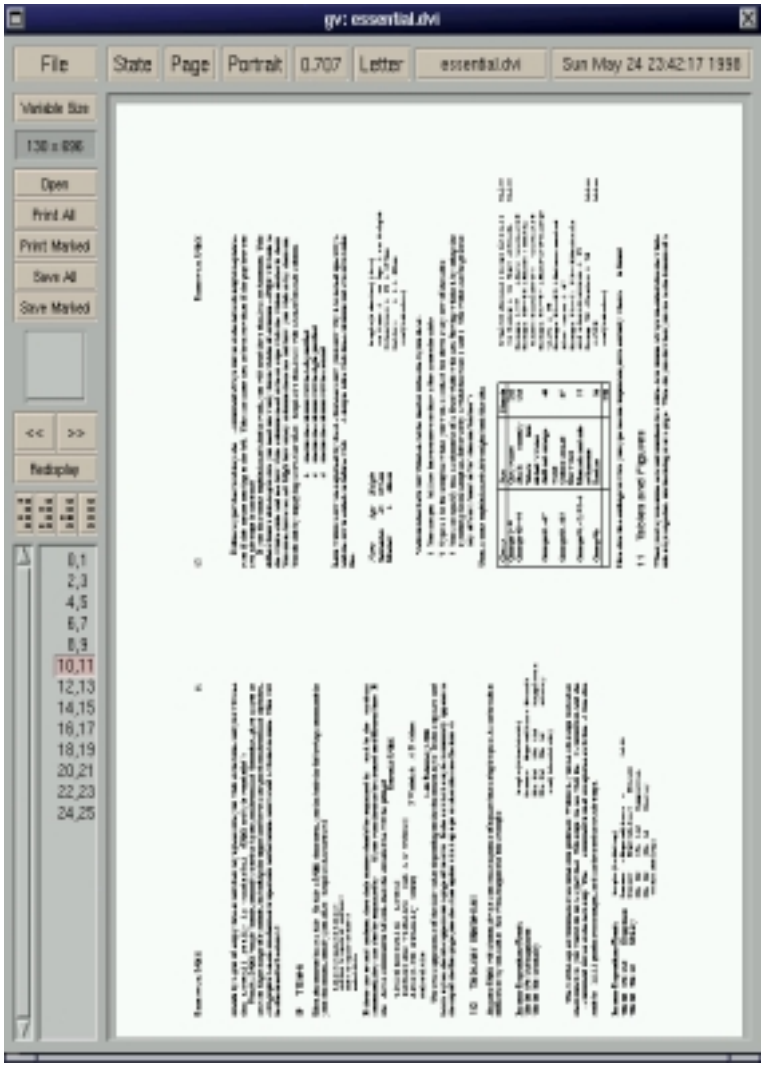
```
[root@pip forprint]# pstops "2:0L@.7(21cm,0)+1L@.7(21cm,12.85cm)"
essential.ps essential.kicsi.ps
[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13]
Wrote 13 pages, 583368 bytes
[root@pip forprint]#
```

A végeredményt a `ghostview` programmal tekinthetjük meg X WINDOW alatt (8.4 ábra).

```
[root@pip forprint]# ghostview essential.kicsi.ps
```

Látható, hogy a lapok fél méretben elforgatva fognak a papíron megjelenni, mégpedig minden oldalon két lap. A betűk is kisebbek lettek a kicsinyítés során, pontosan úgy kicsinyítette le a lapot a `pstops`, ahogyan azt fénymásolóval szokás.

Ha azt szeretnénk elérni, hogy a kinyomtatott könyv lapjainak mindkét oldalán legyen szöveg, akkor át kell alakítanunk a Postscript dokumentumot. Az átalakításra azért van szükség, hogy a lap két oldalán



ábra 8.4: Félméretre kicsinyített lapok

található szöveg könnyvvé rendezhető legyen – a nyomdai kilövésnek[4] megfelelően. A lapok elrendezésére a psbook és psnup parancsok használhatóak. Ha pl. A4 formátumú lapra két oldalt szeretnénk nyomtatni, akkor a következő formát használhatjuk:

```
[root@mad bin]# psbook -q eredeti.ps  | psnup -2 >kesz.ps
```

Ha a nyomtatónk nem alkalmas kétoldalas nyomtatásra, akkor két menetben tudunk könyvet nyomtatni. Az első menetben kinyomtatott lapokat megfelelő módon forgatva vissza kell tennünk a nyomtatóba, hogy a másik oldalra is felkerüljön a szöveg. Ehhez a művelethez a könyvet tartalmazó POSTSCRIPT állományt két részre kell választani páros és páratlan lapokra:

```
[root@mad bin]# psselect -o kesz.ps a.ps
[root@mad bin]# psselect -e kesz.ps b.ps
```

[4] A nyomdában egy lapra több oldal kerül, a lapokat hajtogatják, kötik majd a szélét levágják. Az ívekre a könyvlapok bonyolult szabályok szerint kerülnek fel, hogy a hajtogatás után a sorrend helyes legyen. A lapok elrendezési módját nyomdai szakszóval „kilövésnek” nevezik.

Fejezet 9

Adatbáziskezelés

9.1 POSTGRES95

A POSTGRES95 – továbbiakban POSTGRES – olyan adatbáziskezelő rendszer, amely SQL (*Structured Query Language*, rendszerezett lekérdezőnyelv) szabványú felületen elérhető adatbáziskezelést tesz lehetővé. Habár a POSTGRES nem tartalmazza az SQL szabvány teljes parancskészletét, a legfontosabb kifejezéseket értelmezni és végrehajtani képes, ezért a munkára alkalmas.

9.1.1 Az adatbázis létrehozása

Az adatok a POSTGRES alatt adatbázisokba vannak rendezve, amelyek általában egy-egy teljes feladatcsoportot felölelnek. Ilyen adatbázis lehet pl. a munkaügyi adatbázis, ahol az adott vállalat dolgozóinak adatait tároljuk, vagy az ügyfelek adatbázisa, ahol a megrendelők és megrendelések adatait tartjuk nyilván.

A *adatbázis rendszergazda*[1] feladata beállítani, hogy ki dolgozhat az adott – LINUXot futtató – gépen a POSTGRESsel és milyen adatbázisokhoz férhet hozzá. Engedélyezheti a felhasználónak azt is, hogy újabb adat-

[1] A feladatok megoszthatóak, nem biztos, hogy a *rendszergazda* egy személyben az adatbáziskezelésért felelős *adatbázis rendszergazda* is.

bázisokat hozzon létre, ezért a felhasználónak tisztában kell lennie azzal, hogyan hozhat létre adatbázist:

```
[root@pip pstricks]# createdb szotar
[root@pip pstricks]#
```

Amint az látható adatbázist létrehozni igen egyszerű. A `createdb` (*create database*, adatbázis létrehozása) parancs csak egy szót vár, amely az újonan létrehozott adatbázis neve lesz. Amennyiben az adatbázis létrehozása a felhasználónak engedlélyezett és a művelet sikeres volt külön üzenet nem jelenik meg a képernyőn.

Az adatbázist megsemmisíteni a `destroydb` (*destroy database*, adatbázis elpusztítása) paranccsal lehetséges, éppen olyan formában, ahogy azt a létrehozásánál láttuk.

```
[root@pip /root]# destroydb szotar
[root@pip /root]#
```

Az adatbázisban található adatok az adatbázis törlésekor megsemmisülnek, helyreállításukra általában nincsen mód.

9.1.2 Az SQL parancssor

Amennyiben SQL rendszerben kívánunk adatokat feldolgozni két út áll előttünk: parancsok begépelésével kezeljük adatbázisunkat vagy pedig programot készítünk, amely speciálisan az adott adatokon az adott feladatnak megfelelő módon képes dolgozni.

Mindkét módszernek vannak előnyei. A parancsok begépelésével – SQL parancssort használva – igen rugalmas és hatékony eszközhöz jutunk, míg a speciális programok kezelését könnyű megtanulni. A speciális program és a nehezebben kezelhető hatékony parancssor közül mi a parancssort tárgyaljuk részletesebben, mivel a felhasználótól nem várható el, hogy *a)* minden feladatra programozót szerződtessen vagy *b)* speciális programokat készítsen.

A POSTGRES SQL parancssora a következőképpen indítható:

```
[root@pip latex]# psql szotar
```

```
Welcome to the POSTGRESQL interactive sql monitor:
Please read the file COPYRIGHT for copyright terms
of POSTGRESQL

   type \? for help on slash commands
   type \q to quit

type \g or terminate with semicolon to execute query

You are currently connected to the database: szotar
 szotar=>
```

A `psql` parancs után az adatbázis nevét kell megadnunk amelyen dolgozni kívánunk. A `szotar=>` parancskérő jel – amely a mindenkori aktív adatbázis nevét tartalmazza – mutatja, hogy SQL parancsokat gépelhetünk be.

Amíg az SQL parancssorban vagyunk, annak belső – nem SQL – parancsokat is adhatunk a \ jellel bevezetve. Ilyen belső parancs a `\q`, amellyel az SQL *parancssorból kiléphetünk*:

```
szotar=> \q
[root@pip latex]#
```

A `\?` segítségével a lehetséges belső parancsokról kérhetünk listát. Igen hasznos belső parancs pl. a `\d` amely kilistázza az adatbázisban található összes táblázatot és indexet a tulajdonosaikkal együtt.

9.1.3 A Postgres SQL nyelvjárása

E fejezetben azokról a parancsokról lesz szó, amelyek a POSTGRES számára értelmes SQL parancsok. E parancsokat – a megfelelő formában – a `psql` parancssorba gépelhetjük, vagy speciális programba építhetjük azokat, hogy képzetlen felhasználók is használhassák az adatbázist. (Ilyen módszerre láthatunk példákat a 9.1.7 fejezetben.)

A POSTGRES parancsok összetettek, szinte megegyeznek a beszélt nyelvvel. A kulcsszavak – parancsok – a nagybetű-kisbetű különbségre érzéketlenek, a nevek (*identifyers*, azonosítók) – amelyeket mi vezetünk be – esetében

azonban ügyelnünk kell a pedáns írásmódra[2]. Ehelyütt a parancsokat mindig nagy-, az azonosítókat pedig kisbetűvel írjuk az átláthatóság kedvéért, de ez természetesen nem kötelező.

A POSTGRES parancsait mindig pontosvessző zárja le. Amíg az SQL parandcssor nem talál pontosvesszőt, addig a parancsot nem értelmezi, nem kezdi el végrehajtani. Amikor olyan sort után ütjük le az Entert amelynek nem zárja le a végét pontosvessző, az SQL parancssor várakozik a parancs további részeire. Ezt a másodlagos parancskérő jel kiírásával jelzi, mely abban különbözik a parancskérő jeltől, hogy benne a = jel helyett - áll:

```
szotar=> CREATE TABLE szavak (
szotar-> magyar varchar(36),
szotar-> angol  varchar(36)
szotar-> );
CREATE
szotar=>
```

Ha a parancs begépelésekor hibát vétettünk, akkor a Backspace billentyűvel törölhetjük a hibás betűket. Ha a hiba valamelyik előző sorban van, akkor használhatjuk a \e belső parancsot, amely az aktuális parancs szerkesztését teszi lehetővé a vi szövegszerkesztőben. Amint kimentettük a szerkesztés eredményét és kiléptünk a szövegszerkesztőből, a parancs automatikusan végrehajtódik.

Bármikor törölhetjük az éppen beírás alatt álló parancsot, akkor is ha az több sorból áll. Erre a \r belső parancs szolgál. Használatakor visszakapjuk az elsődleges parancskérő jelet és a parancsot az elejéről kezdve írhatjuk be:

```
root=> rossz parancs
root-> \r
buffer reset(cleared)
root=>
```

A felfelé és lefelé mutató kurzormozgató billentyűkkel az előzőleg begépelt sorok közt válogathatunk.

[2] A Nev és nev elnevezés nem egyezik meg, ezért ügyelnünk kell arra, hogy a nagy- és kisbetűket ne cseréljük fel.

A példák bemutatásánál a parancskérő jeleket elhagytuk, hogy átláthatóbb legyen a begépelt parancs. Szintén a könnyű olvasást segíti a tördelés, amelynek során a parancsok struktúráját a sorok egymás alá rendezésével próbáljuk meg jelezni. A fenti példa ehelyütt tehát így jelenik meg:

```
CREATE TABLE szavak
 (
   magyar varchar(36),
   angol  varchar(36)
 );
```

9.1.3.1 Tábla létrehozása

A POSTGRES az adatokat – az adatbázisokon belül – táblázatokban (*classes*, osztályok)[3] tartja. Egy adatbázis több táblázatot is tartalmazhat – a dolgozók adatbázisa pl. egy táblázatban tartalmazhatja a munkatársak címét és telefonszámát, míg egy másik táblázat az általuk teljesített munkákat.

A táblázat *oszlopai* egymástól független de azonos formájú adatokat tartalmaznak – pl. neveket – míg a *sorai* együvé tartozó de formában és jelentésben különböző információknak adnak helyet – pl. egy dolgozó összes adata. Az oszlopokat szokás mezőknek (*fields*) a sorokat pedig rekordoknak (*records*, feljegyzés) nevezni.

Mielőtt használni kezdenénk egy táblát, azt létre kell hoznunk. Ehhez meg kell adnunk a létrehozandó *tábla nevét*, a benne található *oszlopok nevét* és az egyes oszlopokban tárolt *adatok formátumát*. Az SQL nyelvben a dolgok létrehozására a CREATE (létrehozni) parancs szolgál, amelyet táblázat létrehozásakor a TABLE (táblázat) szó egészít ki. A parancs teljes formája pl. a következő lehet:

```
CREATE TABLE szavak
 (
   magyar varchar(36),
   angol  varchar(36)
 );
```

[3] A szakirodalomban elterjedt kifejezés helyett a táblázat vagy tábla kifejezést használjuk, mivel az SQL parancsok is a TABLE kifejezésre épülnek.

A parancsra sikeres végrehajtását a POSTGRES a CREATE szó kiírásával jelzi. Ha a táblázatot valamilyen oknál fogva nem lehet létrehozni, a postgres angol nyelvű hibaüzenetet ad.

A CREATE parancs formája tehát tablázat létrehozásakor mindig a TABLE parancs beírását és az egyes oszlopok zárójelek közti – vesszőkkel történő – felsorolását jelenti. Az oszlopok megadásakor meg kell mondanunk, hogy mi az oszlop *neve*, és milyen a *típusa*. A következő oszloptípusok közül válogathatunk[4]:

`bool` Logikai értéket hordozó típus, amely két értéket vehet fel. Megadható – igaz érték esetén – TRUE, `'t'`, `'true'`, `'y'`, `'yes'`, `1`, vagy – hamis érték esetén – FALSE, `'f'`, `'false'`, `'n'`, `'no'`, `0` formában.

`char` Egyetlen betűt hordozó adattípus (*character*, betű). Az érték megadásánál egyszeres idézőjeleket kell használni, pl: `'m'`.

`char()` Fix hosszúságú szöveg. A szöveg hossza a zárójelben található szám, amelyre – ha a tényleges szöveg hossza kisebb – a POSTGRES szóközök hozzáadásával kiegészíti azt. Az érték megadásánál egyszeres idézőjeleket kell használni.

`date` Dátum (*date*, dátum), amely évet, hónapot és napot tartalmaz. Habár a dátum igen sokféle formában adható meg, mindig az előre beállított módon jelenik meg. A beíráshoz használhatjuk a `'éééé-hh-nn'` formát[5] , amennyiben a magyar szokásnak megfelelő beállítást használjuk. A dátumforma beállításáról bővebben a 9.1.5 fejezetben olvashatunk a 168. oldalon. A dátum megadható szövegesen is, a `'now'` (most), `'tomorrow'` (holnap), `'yesterday'` (tegnap) formák valamelyikét használva.

`datetime` Időpont tárolására szolgáló adattípus, amely beépített öröknaptárral rendelkezik. A legkésőbbi ábrázolható időpont kb. az időszámításunk elötti negyedik évezredre tehető. Az adatok megadási formája lehet pl. a következő: `'4013-Dec-31 23:59:59 BC GMT'`, ahol a GMT (*Greenwich mean Time*) az időzóna rövidítése.

[4] Csak a legfontosabb adattípusok leírását adjuk meg.

`float4, float8` Lebegőpontos (*float*, lebegő) számokat hordozó típus. A „lebegőpontos" kifejezés arra utal, hogy az ábrázolt számban, a tizedes vessző bárhol lehet[6] , korlátozva csak az ábrázolt számjegyek száma van. Eztrém nagy vagy kicsiny értékek megadhatóak exponenciális alakban, pl.: `1.2e-307`, ahol az `e` után található szám a tizes hatványkitevője: $1.2 \cdot 10^{307}$.

`int2, int4` Egész típusú változó, amely előjellel is rendelkezhet.

`time` Óra, perc, másodperc és századmásodperc hordozására alkalmas adattípus. Az ábrázolható érték `00:00:00.00` és `23:59:59.99` között van. A `time` típus megadásának módja: `'23:59:59.99'`. Az időpont megadásánál használható a `'now'` rövidítés, amely az óra által mutatott értéket jelenti.

`timespan` Időintervallum tárolására szolgáló adattípus. Az értéket pl. másodpercben adhatjuk meg, kiolvasáskor pedig mindig szöveges formában kapjuk: `8 days 7 hours 2 mins 3.00 secs`.

`timestamp` A teljes időpontot hordozó adattípus, melybe az év, hónap, nap, óra, perc és másodperc, ráadásul az időzóna is hordozható. Az adattípus beviteli módja: `'1997-12-30 11:55:59+1'`, de itt is használható a `'now'`.

`varchar()` Változó hosszúságú karaktersorozat hordozására alkalmas adattípus. Az egyszeres idézőjelek közt megadandó karakterláncot ennél a típusnál a POSTGRES nem egészíti ki szóközökkel. A karakterlánc maximális hosszát a zárójelek közt meg kell adnunk. Amennyiben a bevitelnél ennél a hosszúságnál nagyobb karaktersorozatot írunk be, a POSTGRES figyelmeztetés nélkül a maximális hosszra csonkolja.

---

[5] A jelölésben találhato „n" betűk helyére a nap kerül sorszámmal, a „h" jelöli a hónapot, az „é" pedig az évet. Pl.: `'23-12-1998'`

[6] Tekintve, hogy a POSTGRES a számokat normál alakban tárolja és az exponens tárolása is korlátozott területen történik, a lebegőpontos számok sem képesek bármekkora számok tárolására. A korlát a 10 háromszázadik hatványa körül van mindkét irányban, a legkisebb ábrázolható szám tehát kb. 10^{-300}, a legnagyobb pedig 10^{300}.

Az adattípusok meglehetősen széles skálája áll rendelkezésünkre és az adatok megadási módja is igen változatos lehet – különösen a dátumok és időpontok tekintetében. Az itt bemutatott típusok általában a legtöbb feladat megoldására elégségesek, de nem szabad elfelednünk, hogy egyébb típusok és változatok is rendelkezésre állnak.

9.1.3.2 Táblázat törlése

A POSTGRES a törlésre a `DROP` parancsot használja. A `TABLE` szóval jelölhetjük, hogy táblázatot kívánunk törölni – és ilyenkor a táblázat nevét is meg kell adnunk:

```
DROP TABLE szavak;
```

A POSTGRES sikeres törlés esetén a `DROP` szóval jelzi, hogy a művelet befejeződött. Óvatosan kell eljárnunk a táblázat törlésével, mivel az adatok visszanyerésére nincs mód.

9.1.3.3 Táblázat feltöltése adatokkal

Ha már létezik egy táblázat, akkor oda adatokat tehetünk, vagyis elkezdhetjük az adatbázis feltöltését. Természetesen felhasználhatjuk a más formában már meglévő adatbázisainkat, de az SQL parancssort használva is rögzíthetünk a táblázatba új sorokat.

Az adatok bevitelét parancssorból az `INSERT INTO` parancs segítségével végezhetjük. Ennek formája egyszerű példánkban a következő:

```
INSERT INTO szavak VALUES
 (
   'nap',
   'sun'
 );
```

Amint láthatjuk az `INSERT` parancs a `CREATE` paranccsal igen nagy hasonlóságot mutat. Itt az `INSERT` után a az `INTO` következik, majd a `VALUES` kiegészítő szó után zárójelben a táblázathoz hozzáadni kívánt sor adatait kell felsorolnunk, a forma azonban már ismerős. Itt is vesszőkkel kell elválasztani az egyes oszlopokat, bár nem a nevüket és

típusukat kell megadnunk – hiszen az már ismert a POSTGRES számára –, hanem a tartalmukat. A tartalom megadásánál – ha az oszlop típusa szöveges – az adatokat egyszeres idézőjelek, aposztrófok közt kell beírnunk.

Azt, hogy a a felsorolt adatok közül melyik kerül az első, a második, stb. oszlopba, kizárólag a sorrend dönti el, ezért mindig szigorúan ügyelnünk kell arra, hogy az adatokat olyan sorrendben adjuk meg, ahogyan azokat a táblázat létrehozásánál deklaráltuk.

Ha az adatok már rendelkezésünkre állnak valamilyen szöveges állományban, akkor természetesen azokat nem kell újra begépelnünk. Az SQL parancssor `\copy` paranccsa szolgál adatok másolására állományok és táblázatok közt[7]. Az adatokat kinyerhetjük pl. a `szavak` nevű táblázatból a `kimenet` állományba:

```
\copy szavak to kimenet
```

Ha tabulátorjellel elválasztott hasábokba rendezzük adatainkat és a `bemenet` nevű állományba mentjük, akkor a következő paranccsal azt a `szavak` SQL táblázatba tölthetjük:

```
\copy szavak from bemenet
```

9.1.3.4 Adatok listázása a képernyőre

Az adatokat a `SELECT` (kiválaszt) paranccsal listázhatjuk a képernyőre az SQL parancssort használva:

```
SELECT magyar, angol FROM szavak;
```

Ekkor a parancssor egy egyszerű táblázatban felsorolja a táblázatban található adatokat:

[7] Ez a parancs – mint minden \ jellel kezdődő parancs – az SQL parancssor beépített parancsa. Habár létezik hasonló SQL parancs is azt használni kissé körülményes. Ennek az oka az, hogy az SQL parancsok nem a saját nevünkben futnak, az SQL adatgazda állományhozzáférési jogaival vannak felruházva. Mivel az SQL rendszergazda nem rendelkezik írási és olvasási joggal a saját könyvtárunkhoz, oda az POSTGRES állományokat nem tehet és onnan állományokat nem olvashat.

```
magyar|angol
------+-----
nap   |sun
(1 row)
```

Az első sorban az oszlopnevek találhatóak, a további sorok pedig a táblázat sorait mutatják meg. Az utolsó sorban – (1 row) – azt látjuk, hogy hány sornyi a kilistázott táblázat (*row*, sor).

A SELECT parancs után azoknak az oszlopoknak a nevét kell megadni, amelyeket a képernyőn látni szeretnénk majd a FROM (-ból, -ből) kulcsszó után a táblázat nevét.

Az oszlopneveket nem szükséges felsorolni abban az esetben, ha mindegyiket látni szeretnénk – ahogy az előző példában is –, ilyenkor azok a * jellel helyettesíthetőek.

```
SELECT * FROM szavak;
```

Valójában a SELECT parancs sokkal többre képes mint amit első pillanatban gondolnánk róla. A továbbiakban azokat a toldalékokat vesszük sorra, amelyek a SELECT hatékonyságát, rugalmasságát adják.

9.1.3.5 Szűrések

A SELECT parancs tovább bővíthető szűrési feltételekkel, amelyek azt határozzák meg, hogy mely sorokat kívánjuk listázni a táblázatból. A szűréseket a WHERE (ahol) kiegészítővel vezetjük be:

```
SELECT * FROM szavak WHERE magyar='csillag';
```

A parancs hatására a POSTGRES kikeresi azokat a sorokat, ahol a magyar nevű oszlopban a csillag szó szerepel, majd ezeket a sorokat a képernyőre listázza:

```
magyar |angol
-------+-----
csillag|star
(1 row)
```

Ha a kísérleti szótárunkat az ellenkező irányban szeretnénk használni – vagyis angol szavakat keresünk –, nem kell mást tennünk, mint a WHERE kiegészítőben az angol nevű oszlopnevet megadni a feltételben:

```
szotar=> SELECT * FROM szavak WHERE angol='planet';
```

A válasz a megszokott formában érkezik:

```
magyar|angol
------+------
bolygó|planet
(1 row)
```

A feltétel nem csak az = lehet, hanem a <> (nem egyenlő), > (nagyobb szöveg esetében „a névsorban utána álló"), < (kisebb, „a névsorban előtte álló") is.

Szöveges esetben használható a részleges egyezést jelző LIKE (hasonló) szó is:

```
szotar=> SELECT * FROM szavak WHERE angol LIKE 'mo%';
```

A szavak csonkolására a % jel használható. A % jel bármennyi betűt helyettesíthet, a fenti kifejezés tehát listázza az összes „mo" kezdetű angol szót és a magyar fordítását:

```
magyar|angol
------+------
hold  |moon
hétfô |monday
(2 rows)
```

A % nem csak a szavak végén használható, hanem az elején és belsejében is. Ennek megfelelően a „%net" kifejezés az összes „net" végződésű szóra keres, a „we%k" pedig minden szóra illeszkedik, amely „we" kedetű és k betűvel végződik. A WHERE kiegészítőnél használható relációs jeleket a 172. oldalon a 9.1 táblázatban soroltuk fel.

9.1.3.6 Sorrendbe rendezés

A lekérdezéskor megadthatjuk az is, hogy milyen sorrendben kívánjuk a sorokat a képernyőre listázni. Erre az ORDER BY (sorrend szerint) kiegészítő szolgál. A sorrendbe rendezés az ORDER BY után felsorolt oszlopok tartalma szerint történik:

```
SELECT * FROM szavak WHERE angol LIKE 's%'
  ORDER BY angol;
magyar |angol
-------+-----
hét    |seven
csillag|star
nap    |sun
(3 rows)
```

A ORDER BY használata esetén a USING (használva) kulcsszó után megadható a művelet, amelyet a POSTGRES fel fog használni a sorrendiség eldöntésére. Ennek a lehetőségnek leggyakrabban használt formája a csökkenő sorrend beállítása:

```
SELECT * FROM szavak WHERE angol LIKE 's%'
  ORDER BY angol USING <;
```

Az ORDER BY parancskiegészítő megadható akár a WHERE parancskiegészítővel együtt is.

9.1.3.7 Oszlopfüggvények

A SELECT parancs használatakor nem csak a táblázatban található oszlopok közül válogathatunk, hanem számított értékeket tartalmazó oszlopokat is listázhatunk. Tételezzük fel, hogy a arucikk táblázatban létezik elnevezes, netto és afa nevű oszlop, s közülük a netto és afa számadatokat tartalmaz. Ekkor listázhatunk a következő módon:

```
SELECT arucikk, netto, netto*(1+afa/100)
  FROM aruk;
```

A SELECT három oszlop listázására kapott parancsot, abból azonban csak az első kettő szerepel a táblázatban. A harmadik oszlop a táblázat oszlopaiból kerül előállításra, matematikai műveletek segítségével. A végeredmény a következő formájú:

```
arucikk|netto|?column?
-------+-----+--------
ollo   |   50|    62.5
villa  |  250|   312.5
(2 rows)
```

Amint látjuk a harmadik – számított – oszlop is szerepel a listában. A POSTGRES a táblázat minden sorára elvégezte a kijelölt műveleteket, az adott sorban éppen aktuális értékekkel. A vázolt módon beírt képleteket oszlopfüggvényeknek nevezzük.

Az oszlopfüggvények használata esetén megadhatjuk, hogy milyen néven jelenjen meg a kiszámolt érték. Erre az AS (úgy mint) parancs használható:

```
SELECT arucikk, netto*afa AS afa,
                netto*(1+afa/100) AS brutto
  FROM arucikk;
```

Amint látjuk minden egyes kiszámolt érték esetén külön-külön nevet adhatunk az AS kulcsszó segítségével.

9.1.3.8 Csoportosítás

A lekérdezésekben használható a csoportosítás egy igen hatékony eszkoze, a GROUP BY (csoportosítani vmi szerint) kiegészítés. Ez a kiegészítés arra ad lehetőséget, hogy a táblázat soraiból az ismétlődő elemeket kiszűrjük.

Tételezzük fel, hogy egy igen egyszerű táblázatban a következő adatok vannak:

```
arucikk|netto|afa
-------+-----+---
ollo   |   50| 25
```

```
villa  |  250| 25
villa  |  238| 25
(3 rows)
```

Amint látjuk a „villa” kétszer szerepel az adatbázisban, egyszer 250, egyszer pedig 238 netto értékkel. A GROUP BY kiegészítő segítségével ezt az ismétlődést kiszűrhetjük – ha pl. csak az árucikkekről szeretnénk egy névklistát;

```
SELECT arucikk FROM aruk
  GROUP BY arucikk;
arucikk
-------
ollo
villa
(2 rows)
```

A példában a GROUP BY kiegészítő után egy oszlop nevét adtuk meg. A POSTGRES ezen oszlopban keresi az ismétlődéseket. Ez az oka annak, hogy az adatbázisban egyébként kétszer szereplő „villa” most csak egyszer jelentkezik. Fontos megjegyeznünk, hogy a csoportosítás csak olyan oszlop szerint történhet, amely a listában egyébként megjelenik.

Vizsgáljuk meg, mi történik ha a a listában olyan oszlopokat kérünk, amelyek a GROUP BY által kiszűrt ismétlődő sorokban különböző értékekkel jelentkezik:

```
SELECT arucikk, netto FROM arucikk
  GROUP BY arucikk;
arucikk|netto
-------+-----
ollo   |   50
villa  |  238
(2 rows)
```

Amint látjuk, a felsorolásba belekerült az egyik „villa” és a netto értéke. Történetesen éppen 238 áll a „villa” mellett, de ez nem jelent túl sokat, hiszen a lista netto oszlopa véletlenszerűen kiválasztott értékeket hordoz. A táblázat így használhatatlan.

Ha a GROUP BY kiegészítést használjuk, akkor a különbözőséget hordozó oszlopokra ügyelnünk kell. Használható információt ezekből a különböző értékeket felsorakoztató oszlopokból ilyen esetben a *csoportosítható oszlopfüggvények*kel nyerhetünk. Lássunk egy példát:

```
SELECT arucikk, SUM(netto) FROM arucikk
  GROUP BY arucikk;
arucikk|sum
-------+---
ollo   | 50
villa  |488
(2 rows)
```

A példában szereplő SUM (*summa*, összeg) függvény numerikus oszlopok összegét adja. Mivel a SUM csoportosítható, az egyes csoportokra külön kiértékelésre kerül és így helyes eredményt adja[8]. A csoportosítható oszlopfüggvények a következőek:

SUM — *Summa* (összeg). A numerikus oszlop összegét adja csoportonként.

MIN — *Minimum* (legkisebb). A legkisebb elem értékét adja csoportonként.

MAX — *Maximum* (legnagyobb). A legnagyobb elem értékét adja csoportonként.

COUNT — *Counter* (számláló). Az csoportosítás előtti előfordulások számát adja csoportonként.

AVG — *Average* (átlag). Az elemek számtani átlagát adja csoportonként.

9.1.3.9 Új tábla létrehozása másolással

A SELECT utasítás alkalmas arra is, hogy újabb táblázatot hozzunk létre az adatokkal már feltöltött táblázatok felhasználásával. Ebben az esetben a INTO TABLE (táblázatba) kulcsszóval kell kiegészítenünk a parancsot:

[8] A táblázat a kifejezés szerint az összes arucikket tartalmazza – azonban mindegyiket csak egyszer –, mellettük pedig a netto érték összege szerepel.

```
SELECT * INTO TABLE draga FROM arucikk
  WHERE netto>1000;
```

Az INTO TABLE használata esetén is használhatjuk a SELECT parancs eddig megismert kiegészítéseit, tehát szűréseket, csoportosításokat és sorbarendezéseket is előírhatunk az új táblázat elkészítésekor.

Az új táblázat azokat az oszlopokat fogja csak tartalmazni, amelyeket a SELECT után előírunk. Ha a SELECT után * áll – mint a fenti példa esetében – akkor az összes oszlop átkerül az új táblázatba.

Amennyiben oszlopfüggvényekkel számolt értékeket is listázunk, azok átkerülnek az új táblázatba. A következő parancs hatására pl. létrejön egy új oszlop, amely az AS hatására a brutto nevet viseli:

```
SELECT arucikk, netto*(1+afa/100) AS brutto
  INTO TABLE brutto FROM arucikk;
```

A parancs hatására létrejött új táblázat a következő:

```
arucikk   |brutto
----------+------
ollo      |  62.5
asztal    |  1875
monitor   |  6875
winchester|  4375
pointer   |  4375
telefon   | 43750
pipa      |   375
(7 rows)
```

Az új tábla a másolás után ugyanúgy viselkedik, mintha azt „kézzel” hoztuk létre, tehát adatokkal bővíthető, törölhető, listázható, stb. Amennyiben olyan áblázatot akarunk létrehozni, másolással, amely követi a forrásul szolgáló táblázatban bekövetkező változásokat, a 166. oldalon található 9.1.3.13 fejezetben ismertetett CREATE VIEW parancsot kell használnunk.

9.1.3.10 Sorok törlése táblázatból

Törölni sorokat a DELETE FROM (törlés vhonnan) parancs segítségével lehet. Egyszerű formájában a DELETE minden sort töröl a táblázatból, óvatosan kell tehát eljárnunk a használatakor:

```
DELETE FROM arucikk;
```

A DELETE parancs elvégése után a Postgres a képernyőre nyomtatja, hogy hány sort törölt a táblázatból.

Ha csak bizonyos sorokat kívánunk törölni, a WHERE parancskiegészítővel kell szűrnünk a DELETE hatókörét:

```
DELETE FROM arucikk WHERE arucikk='ollo';
```

Amint a parancs formájából következik, a DELETE nem képes olyan sorok közt különbséget tenni, amelyek minden oszlopukban ugyanolyan értékeket hordoznak[9].

9.1.3.11 Sorok módosítása

Adatok módosítására az UPDATE (frissít) parancs használatával van lehetőségünk. Az UPDATE képes az adatokat úgy módosítani, hogy az új értékek kiszámításához felhasználja az adott sorban található aktuális értékeket.

```
UPDATE arucikk SET netto=netto*1.2 WHERE netto<50;
```

A fenti példában – amely 20% -al megnöveli minden netto értékét, amely kevesebb mint 50 – látható, hogy az UPDATE esetében is használható a WHERE kiegészítő.

A DELETE parancshoz hasonlóan az UPDATE sem képes különbséget tenni azonos értékeket hordozó sorok közt, ezért az egyedi kulcs használata itt is igen fontos:

```
UPDATE arucikk SET netto=1200 WHERE cikkszam=1235412 ;
```

[9] Ez azonban nem jelenthet problémát, hiszen az adatbázisszervezés egyik alapelve, hogy minden táblázatban kell, hogy legyen olyan egyedi kulcs, amely garantáltan különbözik minden sorban. Ilyen kulcs lehet a cikkszám, számlaszám, stb.

9.1.3.12 Új oszlopok létrehozása

A már elkészített és adatokkal feltöltött táblázatokba is lehetőségünk van új oszlopok beszúrására. Erre az ALTER TABLE (táblázat módosítása) parancs felhasználásával van mód.

```
ALTER TABLE arucikk
 ADD
 (
  cikkszam int
 );
```

Az ALTER TABLE parancs végrehajtását az ADD szó kiírásával jelzi a POSTGRES. Láthatjuk, hogy új oszlop létrehozásakor hasonlóképpen kell megadnunk az oszlop nevét és típusát, ahogyan azt a táblázat létrehozásakor megtettük. Lényeges különbség, hogy oszlop beszúrásakor egyszerre csak egy új oszlop hozható létre.

A létrehozott oszlop sorrendben mindig az utolsó lesz.

9.1.3.13 Nézettábla létrehozása és törlése

9.1.3.14 Index létrehozása és törlése

A POSTGRES sokkal gyorsabban képes végrehajtani bizonyos utasításokat, ha a táblázathoz indexet készítünk. Az index olyan kiegészítő eleme az adatoknak, amely új információt már nem tartalmaz, csak a táblázatban lévő sorok kikeresését gyorsítja meg. Indexek létrehozásakor mindig háttértárkapacitást áldozunk fel a sebesség növelése érdekében.

Indexeket létrehozni a CREATE INDEX parancs segítségével lehet:

```
CREATE INDEX nev ON cikkek (nev, brutto);
```

Látható, hogy minden index nevet kap az elkészítésekor. Az index létrehozásához az ON kulcsszó után a táblázatot kell megadnunk amelyen az indexet használni szeretnénk, valamint zárójelek közt felsorolva azokat az oszlopokat, amelyek szerint a rendezés az indexben történik.

Az elkészített indexet a POSTGRES folyamatosan frissíti, ezért az adatbevitel az index létrehozásával lassulhat. Az indexek használata automatikus, ha tehát a Postgres olyan indexet talál, amely az éppen végrehajtásra kerülő műveletet gyorsíthatja, akkor az indexet használni fogja a gyorsaság növelése érdekében. Mivel az indexek használata automatikus, jól meg kell gondolnunk, hogy milyen indexeket hozzunk létre.

Az adatbázis szerkezetéből nem állapítható meg, hogy milyen jellegú indexeket érdemes létrehozni, hiszen a jó index a leggyakrabban használt adatkezelési feladatokat gyorsítja, az indexelés milyenségét ezek a feladatok határozzák meg. Ha pl. sokszor keresünk az ügyfelek adatait tartalmazó táblázatban város szerint, akkor a város nevét tartalmazó oszlop szerint érdemes indexet készíteni.

Az egyedi kulcsok szerint mindenképpen érdemes indexet létrehozni. Ekkor az `UNIQUE` (egyedi) kulcsszóval egészíthetjük ki a `CREATE INDEX` parancsot, amelynek hatására a POSTGRES az index létrehozásakor és új adatok beszúrásakor ellenőrzi, hogy az egyedi kulcs aktuális értéke szerepel -e már valahol a táblázatban:

```
CREATE UNIQUE INDEX csz ON cikkek (cikkszam);
```

Ha megpróbálunk olyan sort beszúrni a táblázatba, amelynek cikkszáma – egyedi kulcsa – már szerepel az indexben, a POSTGRES megtagadja a beszúrás végrehajtását és hibaüzenetet ír a képernyőre.

Indexet törölni a DROP INDEX paranccsal lehet.

```
DROP INDEX nev;
```

Az indexeket bármikor újra elő lehet állítani az adatokból , bár nagyméretű adatbázisok esetén ez akár órákat is igénybe vehet.

9.1.4 Rendszertáblázatok

A POSTGRES a belső működéséhez szükséges adminisztratív táblázatokat a felhasználó adataival megegyező módon kezeli. Ez igen könnyű hozzáférést biztosít a felhasználó számára olyan adatokhoz, amelyek a rendszer belső állapotát tükrözik. A felhasználó számára elérhető rendszertáblázatok közül z legfontosabbak a következőek:

`pg_database` A rendszeren található adatbázisokat tartalmazó táblázat. Ha a felhasználó elfeledte, hogy milyen néven hozott létre adatbázist, e táblázat lekérdezésével könnyedén lekérdezheti azt. A `psql` indításához ugyan szükséges egy adatbázist megadni, de mivel normális esetben mindig létezik a `template1` adatbázis, ez nem jelent akadályt. Az adatbázisok lekérdezésének e formáját a 9.1 tábla mutatja.

`pg_class` Az adatbázisban található táblázatok adatait találhatjuk ebben a táblázatban. Amikor lekérdezzük a `pg_class` táblázatot, akkor a POSTGRES rendszertáblázatait is itt találjuk. Ha csak a fehasználó adatait tartalmazó táblázatokat kívánjuk megjeleníteni, akkor a `pg_class` táblázatot szűrnünk kell. Ehhez felhasználhatjuk azt, hogy a Postgres minden adminisztratív távbla nevét „`pg_`” kezdettel látja el: `SELECT * FROM pg_class WHERE relname NOT LIKE 'pg_%';`

`pg_operator` Az értelmezett operátorok – műveleti és relációs jelek – listáját tartalmazó táblázat. A `pg_operator` tartalmazza azokat az operátorokat is, amelyeket a felhasználó vezetett be.

A rendszertáblázatok itt bemutatott listája csak a legtöbbször használt táblákat mutatja be, amelyek a felhasználó számára a legfontosabbak.

9.1.5 Dátumforma

A POSTGRES képes a dátumokat többféle formában is bemutatni. A dátumforma beállítására a `SET DATESTYLE TO` parancs szolgál, ahol a felhasználható formákat az `ISO`, `SQL` és `Postgres` szavak jelölik:

```
SET DATESTYLE TO 'ISO';
```

Az `SQL` típus rendelkezik `US` és `European` változattal, amelyek a hónapok és napok sorrendjében különböznek egymástól.

```
SET DATESTYLE TO 'SQL,European';
```

A *magyar írásmód*nak – éééé-hh-nn – az ISO dátumformátum felel meg.

```
[root@pip /root]# psql template1
Welcome to the POSTGRESQL interactive sql monitor:

Please read the file COPYRIGHT for copyright terms of POSTGRESQL

   type \? for help on slash commands
   type \q to quit

type \g or terminate with semicolon to execute query

You are currently connected to the database: template1

template1=> SELECT * FROM pg_database;
datname  |datdba|datpath
---------+------+---------
template1|   100|template1
egyetem  |     0|egyetem
szotar   |     0|szotar
(3 rows)

template1=>
```

Táblázat 9.1: Adatbázisok lekérdezése a POSTGRESben

Jel	*Jelentés*	*Példa*
!	Faktor	5!
%	Maradék. A bal oldalon álló szám az osztandó, a jobb oldali az osztó, az eredmény pedig az osztás maradéka	27%5
%	Egészrész. A Postgres a tizedeseket elhagyja – tehát *nem* kerekít	%4.86
*	Szorzás	4*3
+	Összeadás	3+2
-	Kivonás	3-1
/	Osztás	5/3
:	Hatványozás természetes alapra (:3.2= $e^{3.2}$)	:3.6
;	Természetes alapú logaritmus. A zárójelet használni kell!	(;5.0)
@	Abszolútérték	-12@
^	Hatványozás	2.0^3.0
|/	Négyzetgyök	|/9.0
|//	Köbgyök	|//27.0

Táblázat 9.2: A POSTGRES matematikai műveletei

9.1.6 Műveletek az adatokkal

A táblázatokban található adatokkal különféle műveletek végezhetünk az adattípustól függően.

A 9.2 táblázatban soroljuk fel azokat a *matematikai műveleteket*, amelyeket a POSTGRES felismer és elvégez. A műveletek elvégzésének sorrendje követi a matematikai szabályokat, ezektől eltérni a zárójelek alkalmazásával lehet.

A POSTGRES nem minden matematikai műveletet értelmez mindegyik számábrázolási formára. A hatványozás műveletét pl. csak lebegőpontos számokkal képes elvégezni, ezért pl. amikor konstans értékeket adunk meg, ennél a műveletnél, mindig ki kell tennünk a tizedespontot (2.0^3.0). Ha változón szeretnénk műveletet végezni és a típus nem megfelelő, akkor típuskonverziót kell végrehajtanunk a 9.3 táblázatban bemutatott függvények valamelyikével.

A *típuskonverziós függvények* a változók értékét adják vissza – változ-

Jel	***Jelentés***	***Példa***
`float()`	`int` típus konvertálása `float8` -ra	
`float4()`	`int` típus kovertálása `float4` -re	
`integer()`	lebegőpontos típus konvertálása `int` -re	
`abstime()`	datetime konvertálása `abstime` -ra	
`timespan()`		
`datetime()`		
`<#>`	időintervallummá konvertálás	

Táblázat 9.3: A POSTGRES típuskonverziós függvényei

Jel	***Jelentés***	***Példa***
`AND`	Logikai igaz értéket ad vissza, ha mindkét oldalán igaz érték szerepel.	`a=20 AND b<12`
`OR`	Igaz értéket ad, ha legalább az egyik oldalán igaz érték szerepel.	`a>20 OR a<100`
`NOT`	Az utána álló logikai értéket ellentettjére változtatja.	`a<10 AND NOT a=3`

Táblázat 9.5: A POSTGRES logikai műveletei

tatás nélkül – a típust azonban átalakítják.

A *relációk* olyan speciális műveletek, amelyek mindig logikai értéket adnak eredményül. Relációk segítségével a változók közti viszonyokat vizsgálhatjuk, az eredményt pedig felhasználhatjuk feltételként az egyes parancskiegészítőkben. A relációkat a 9.1 táblázatban soroljuk fel. A kisebb, nagyobb, stb. relációk szöveges változókon is értelmezettek, ekkor a névsorban előbb elhelyezkedő szöveg lesz a kisebb.

A *logikai műveletek* olyan – a kételemű Boole algebra szabályait követő – műveletek, amelyek logikai változókon értelmezhetőek és logikai értéket is adnak eredményül. A legfontosabb felhasználási területük a relációk összekapcsolása, ezért a 9.5 táblázatban is ilyen eseteket sorolunk fel példaképpen.

9.1.7 SQL és BASH

E fejezetben olyan BASH programra látunk egyszerű példát, amelyek SQL adatbázison végez manipulációkat.

A bemutatásra kerülő csekélyértelmű telefonszámnyilvántartás két BASH programból áll. Az első az adatbázis létrehozására és törlésére szolgál, neve lehet pl. telcd.

A program első sora arra utasítja a parancssort, hogy a futtatáshoz a /bin/bash nevű programot használja fel. Mivel Linux alatt általában ez az alapértelmezés, e sort tulejdonképpen el is hagyhatnánk, akkor azonban nem volna hordozható a programunk.

```
#! /bin/sh
```

Jel	*Jelentés*	*Példa*
<	Kisebb?	o<23
<=	Kisebb vagy egyenlő?	o<=23
<>	Nem egyenlő?	o<>12.4
=	Egyenlő?	o=12.4
>	Nagyobb?	o>12.4
>=	Nagyobb vagy egyenlő?	o>=12.4
!!=		o<>12.4
~~, LIKE	Csonkolt egyezés. Egy betűt _ jellel, legalább egy betűt % jellel helyettesíthetünk.	e ~~'_lso'
!~~, NOT LIKE	Csonkolt egyezés ellentéte.	m !~~ '%lso'
~	Regex típusú egyezés.	s ~ '*lso'
~*	Regex egyezés a kis- és nagybetűk közti különbség figyelembevétele nélkül.	s ~* '*lso'
!~	Regex egyezés ellentéte.	s !~ '*lso'
!~*	Regex egyezés ellentéte a kis- és nagybetűk közti különbség figyelembe vétele nélkül.	s !~* '*lso'

ábra 9.1: A POSTGRES által értelmezett relációs jelek.

A következő sorban beállítjuk a BASH menükezelőjének választást kérő jelét. A PS3 -ban tárolt szöveget arra használja a BASH, hogy a kinyomtatásával jelezze a felhasználónak, hogy a felsorolt pontok közül kell választania.

```
PS3="   Valasztas: "
```

A következő sortól kezdve felépítjük az egyszerű szöveges menüt, amelyből a felhasználó választhat. A select után a fejezetben tárgyalt módon felsoroljuk a menüpontokat, majd a do szóval jelöljük, hogy az egyes menüpontok választása során végrehajtandó feladatokat részletezzük.

```
select NEV in "Adatbazis torlese" \
              "Adatbazis letrehozasa" \
              "Kilepes"
do
```

Az első menüpont az adatbázis teljes törlésére használatos. Ilyenkor nem kell mást tennünk csak a `destroydb` parancs segítségével megsemmisítjük a telefon nevű adatbázist.

```
  case $NEV in
    "Adatbazis torlese")
         destroydb telefon
        ;;
```

A második menüpont az adatbázis létrehozása. Ha a felhasználó ezt választja, a createdb paranccsal létrehozzuk a telefon adatbázist. Az adatbázisban egy táblázatot is létre kell hoznunk ezután, amelyek a telefonszámokat és a neveket tartalmazzák.

A táblázatot a POSTGRES képes létrehozni ha arra SQL parancsot kap. SQL parancsok kiadására a `psql` nevű program, az SQL parancssor alkalmas – eddig is ezt használtuk. A psql -c opcióval alkalmas BASH parancssorból közvetlenül SQL kifejezések fogadására. Ilyenkor nem ad bevitelt kérő jelet a felhasználónak, hanem közvetlenül végrehajtja az SQL parancsot, majd befejezi futását.

```
        "Adatbazis letrehozasa")
             createdb telefon
             psql telefon -c "CREATE TABLE telszam \
    (nev varchar(32), telefon varchar(32));"
            ;;
```

A program végén alkalmat adunk a felhasználónak a kilépésre a Kilépés menüponttal. Az exit parancs hatására a BASH program futása befejeződik, egyébként a BASH a menüt ismét kinyomtatja és az újabb választásig vár.

```
         Kilepes)
             exit
            ;;
```

A program utolsó két sora jelzi, hogy a menüpontok választásakor végrehajtott feladatok felsorolása véget ért.

```
       esac
    done
```

Lássuk most a programot egyben:

```
    #! /bin/sh

    PS3="   Valasztas: "

    select NEV in "Adatbazis torlese" \
                  "Adatbazis letrehozasa" \
                  "Kilepes"
    do
      case $NEV in
        "Adatbazis torlese")
             destroydb telefon
            ;;

        "Adatbazis letrehozasa")
             createdb telefon
             psql telefon -c "CREATE TABLE telszam \
```

```
    (nev varchar(32), telefon varchar(32));"
            ;;

         Kilepes)
             exit
            ;;
       esac
    done
```

Ha mindezt a telcd állományban elhelyezzük és a chown parancs (2.5.2.7 fejezet 38. old.) segítségével futtathatóvá tesszük, indításakor a következőt látjuk:

```
[root@pip bash]# ./telcd
1) Adatbazis torlese
2) Adatbazis letrehozasa
3) Kilepes
   Valasztas:
```

Az egyes menüpontok előtt álló számok beírásával[10] választhatunk a menüből. Egyszerű programunk minden különösebb figyelmeztetés nélkül végrehajtja az egyes műveleteket, majd a menüt újra a képernyőre nyomtatja.

Lássuk most a program másik részét. Ez az adatok bevitelére, és a keresésre – tehát a mindennapi használatra – készült. A BASH program neve legyen tel, mivel ezt a nevet egyetlen UNIX parancs sem foglalja még le[11].

A program hasonlóképpen kezdődik mint az előző, itt is kikötést teszünk a szöveges programunk értelmezésre használt programra:

```
#! /bin/sh
```

Ez a program már kissé bonyolultabb mint az előző, ezért részekre – eljárásokra – tagoljuk. Legelőször ezeket az eljárásokat kell a programnak tartalmaznia, ezért mielőtt egyetlen utasítást is leírnánk, definiáljuk a tarolas, kereses és torles eljárást.

[10] Az Enter billentyűt le kell nyomni a szám beírása után.

[11] Természetesen ettől még lehetne telefon, de akkor minden indításkor négy betűvel többet kellene begépelnünk.

A tarolas fogja az új nevek rögzítését elvégezni. A feladat egyszerű, kérnünk kell a felhasználótól a read parancs segítségével egy nevet és egy telefonszámot, ezeket az INSERT INTO SQL parancs keretében át kell adnunk a psql parancsnak azonnali végrehajtásra:

```
tarolas() {
  echo
  echo
  echo -n "Tarolando nev: "
  read NEV
  echo -n "Tarolando telefonszam: "
  read TEL
  psql telefon -qc "INSERT INTO telszam \
VALUES ('$NEV', '$TEL');"
  echo
}
```

Figyeljük meg, hogy mindkét változót – NEV és TEL – egyszeres idézőjelek között adunk át. Ennek az az oka, hogy a táblázat létrehozásakor két szöveges oszlopot hoztunk létre, vagyis a telefonszámokat is szövegként tároljuk. Ha nem így tennénk, hanem pl. egész típusú oszlopot használnánk, akkor nem tudnánk tárolni a telefonszámok lejegyzésekor általában használt extra karaktereket, amilyenek pl. a kötőjel vagy a zárójel.

A második eljárás a keresésre szolgál. Kereséskor az összes olyan sort kinyomtatjuk a képernyőre, amelyben a felhasználótól kért szó – vagy szavak – megtalálhatók. Erre használhatjuk a LIKE SQL kulcsszót és a % csonkoló karaktereket. A bekért szó elejére és a végére egy-egy csonkoló jelet teszünk. Mindezen ügyködésünk prózai oka a kényelemszeretet: így nem kell a teljes nevet beírni.

A `psql` parancssor kellemes tulajdonsága, hogy amikor parancssori opciókét kapja meg az SQL kifejezést amit végre kell hajtania, akkor minden különösebb körítés nélkül kinyomtatja a képernyőre a SELECT eredményét. Ez most éppen kapóra jön.

```
 kereses() {
  echo
  echo
  echo -n "Keresendo nev: "
  read NEV
  echo
  psql telefon -qc "SELECT * FROM telszam \
WHERE nev LIKE '%$NEV%';"
}
```

A törlés eljárása csak egyetlen információt kér a felhasználótól, azt viszont pontosan. A DELETE parancs felhasználásával törli a megadott névvel pontosan megegyező nevet tartalmazó oszlopokat. Ha a felhasználó egyetlen betút is téveszt, a törlés nem történik meg – vagy ami még rosszabb, más név törlődik.

A törlés eljárásból látható, hogy az egyedi kulcs a nev mező. Kicsiny programunk tehát nem képes egyforma nevű, de különböző telefonszámú ismerőseink közt különbséget tenni.

```
torles() {
  echo
  echo
  echo -n "A pontos nev: "
  read NEV
  psql telefon -qc "DELETE FROM telszam \
WHERE nev='$NEV';"
}
```

A három eljárás definiálása után[12] már elkezdhetjük a parancsokat felsorolni. Beállítjuk a tehát a menükérő szöveget, sőt extra szolgáltatásként le is töröljük a képernyőt:

```
PS3="   Menu: "
clear
```

Az előző programhoz hasonlóan most is egy egyszerű menüt készítünk, amelyből a felhasználó választhat. A menü négy pontja rendre az új név felvitelére, törlésre, keresésre és az elmaradhatatlan kilépésre szolgál.

[12]Ne feledjük, hogy a BASH eljárásokat mindig a BASH program elején, minden egyéb előtt kell felsorolnunk. Ha nem így teszünk a program a BASH számára értelmét veszti.

Hála a gondos előkészítő tevékenységünknek, a menü feladatainak felsorolásakor már csak az előzőleg elkészített eljárásokra kell hivatkoznunk. A menüpontok kiválasztásakor a BASH automatikusan hívja az állomány elején található eljárásokat, majd azok befejeztével visszatér és folytatja a menünél félbehagyott munkát.

```
select MENU in "Uj nev felvitele" \
               "Torles" "Kereses" \
               "Kilepes"
do
  case $MENU in
     "Uj nev felvitele")
        tarolas;;
     "Torles")
        torles;;
     "Kereses")
        kereses;;
     Kilepes)
        exit;;
   esac
done
```

A `tel` állomány tartalma tehát a következő:

```
#! /bin/sh

tarolas() {
  echo
  echo
  echo -n "Tarolando nev: "
  read NEV
  echo -n "Tarolando telefonszam: "
  read TEL
  psql telefon -qc "INSERT INTO telszam VALUES\
('$NEV', '$TEL');"
  echo
}
```

```
kereses() {
  echo
  echo
  echo -n "Keresendo nev: "
  read NEV
  echo
  psql telefon -qc "SELECT * FROM telszam \
WHERE nev LIKE '%$NEV%';"
}

torles() {
  echo
  echo
  echo -n "A pontos nev: "
  read NEV
  psql telefon -qc "DELETE FROM telszam \
WHERE nev='$NEV';"
}

PS3="   Menu: "
clear

select MENU in "Uj nev felvitele"\
               "Torles" \
               "Kereses" \
               "Kilepes"
do
  case $MENU in
     "Uj nev felvitele")
        tarolas;;
     "Torles")
        torles;;
     "Kereses")
        kereses;;
     Kilepes)
        exit;;
   esac
done
```

Az állományt futtatva választhatunk az egyes menüpontok közül:

```
1) Uj nev felvitele  3) Kereses
2) Torles            4) Kilepes
   Menu: 3


Keresendo nev: Gizella

nev        |telefon
-----------+------------
Gaz Gizella|(23) 345-456
(1 row)

1) Uj nev felvitele  3) Kereses
2) Torles            4) Kilepes
   Menu: 4
[root@pip bash]#
```

Fejezet 10

Grafika

10.1 Sugárkövetés a Povray segítségével

Valósághű háromdimenziós képek előállítására a legelterjedtebb módszer a sugárkövetés (*ray-tracing*). A sugárkövetés igen számításigényes folyamat, hiszen e módszert használva minden képpont színének meghatározásához végig kell követnünk az összes oda beérkező fénysugarat, akár több visszaverődésen és fénytörésen keresztül. A számításigény némiképpen csökkenthető a fénysugár útjának megfordíthatósága miatt[1]. Mivel csak azok a fénysugarak érdekelnek minket, amelyek a képet alkotják, ezért visszafelé haladva csak a kamerába bejutott fénysugarakat követjük végig.

A LINUX alatt a POVRAY (*Persistence of Vision*, A Látomás Örökléte)nevű sugárkövetéses rendszerű grafikai programot vizsgáljuk most meg.

10.1.1 A munkamenet

A POVRAY bemenete egy szöveges állomány, amely egy sosemvolt világot ír le, a kimenete pedig a látvány ami ebben a világban a szemünk elé tárulna[2]. A virtuális (látszólagos) világot leíró szöveges állomány

[1] A fizika szerint a fény ugyanazt az utat járja be mindkét irányba.
[2] A Povray alkalmas mozgóképek készítésére is.

speciális nyelven (*scene description language*, színhelyleíró nyelv) készíthető el, a program használatának az elsajátítása pedig e megtanulását jelenti. A munkamenet elsajátításához most készítsünk egy ilyen állományt, melynek tartalma:

```
#include "colors.inc"
 sphere
 {
   <0, 1, 2>, 2
   texture
    {
      pigment { color Red }
    }
  }
background { color White }
camera
 {
   location <0, 2, -3>
   look_at  <0, 1,  2>
 }
light_source { <2, 4, -3> color White}
```

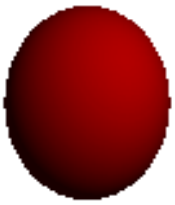
A Povrayt pedig indítsuk a következő módon:

```
[root@pip /root]# povray +L/usr/lib/povray3/include
+i pov01.pov +o pov01
```

A művelet eredménye a `pov01.tga` nevű állományba kerül (10.1 ábra), amelyet megtekintehtünk pl. a `gimp` nevű programmal.

A parancssorban a povray után a példában három dolgot adtunk meg, rendre a kiegészítő (include files) állományok elérési útvonalát (`/usr/lib/povray3/include`) valamint a feldolgozandó szöveges állomány (`pov01.pov`) és a kimeneti állomány (`pov01`) nevét.

A +L opció után megadott könyvtárban azok az állományok találhatóak, amelyeket a povray nyelvén készített kiegészítők vannak. Ezeket a kiegészítőket a program készítői hozták létre – a Povray programcsomag részei –, és munka közben az #include (beleértve) paranccsal fűzhetjük



ábra 10.1: Egyszerű kép a Povray kimeneteként

a saját munkánkhoz őket. A kiegészítő állományok könyebbé teszik munkánkat.

A bemeneti és kimeneti állományok nevének megadása rendre a +i és +o opciókkal történik a példában látható módon.

Természetesen nem csak ily kicsiny képek létrehozására alkalmas a POVRAY, a parancssorban megadhatjuk, hogy hány képpontnyi legyen a végeredmény szélessége és magassága:

```
[root@pip /root]# povray +L/usr/lib/povray3/include +W256 +H256 +i pov01.pov +o pov01
```

A W opció után a szélesség (*width*) a H után pedig a magasság (*height*) adható meg képpontokban. Nagyméretű kép készítése természetesen tovább tart, hiszen több fénysugarat kell végigkövetnie a programnak. Bonyolultabb bemeneti állomány esetében akár több órás programfutásra is számíthatunk, mozgókép esetében pedig nem ritka a napokig tartó folyamatos számítás sem.

10.1.1.1 A színhelyleíró nyelv alapelemei

A POVRAY nyelve meglehetősen komplex, de mivel jól átgondolt és konvencionális[3], az elsajátítása nem lehetetlen.

A nyelv erősen támaszkodik a csoportok képzésére, amely kapcsos zárójelek egymásba ágyazott hierarchiájával lehetséges. A jó áttekinthetőség érdekében a csoportokat sortörésekkel és beljebb tördeléssel jelezzük

[3] A nyelv alapszabályai a C programozási nyelvet ismerők számára hamar átláthatóak.

az olvasónak, ezért a Povray az extra sortöréseket, szóközöket és tabulátor jeleket megengedi. Az átalunk használt írásképben a gyakorlott szem azonnal látja,. hogy az egymáshoz tartozó zárójelek hogyan csoportosítják a nyelv kifejezéseit.

A nyelv alapelemének tekinthető a „<>” jelek közt megadott három koordináta, amely a povray háromdimenziós világában egy pont helyét határozza meg egyértelműen. A koordinátákat x, y, z sorrendben kell megadnunk.

A következőkben a színhelyleíró nyelv néhány alapelemét tekintjük végig, azok egyszerű formájában. Az egyes kifejezéseket a későbbiekben bonyolultabb formájukban is megmutatjuk.

10.1.2 Kamera

A `camera` (kamera) e virtuális világban a szemünket helyettesíti. Mindig szükségünk van egy kamerára, amelyhez legegyszerűbb esetben csak két pont megadására van szükségünk. Az első pont a kamera pontos helyét határozza meg (location, hely), a második pedig azt, hogy pontosan mi van a kamera látómezejének középpontjában (look at, nézni vhová). A parancs formája lehet pl az előzőekben vázolt példának megfelelő:

```
camera
 {
   location <0, 2, -3>
   look_at  <0, 1,  2>
 }
```

A kamera képe a look_at után megadott pontban éles, attól közelebb és távolabb életlen, homályos. Ez a valóságos kamerákkal is így van, sőt a szemünk sem viselkedik másképpen.

10.1.3 Fényforrás

A POVRAY többféle „lámpát” ismer, melyek közül a legegyszerűbb a pontszerű fényforrás. Munkánknak mindig tartalmaznia kell valamilyen fényforrást – vagy fényt sugárzó egyéb testet – ellenkező esetben az elkészített kép teljesen fekete lesz, vagyis nem látunk semmit.

Első próbálkozásaink alkalmával pontszerű fényforrást használunk, amelyekből többet is felhasználhatunk, annak érdekében, hogy a kép egyes részletei ne kerüljenek sötét árnyékba.

A fényforrást a light_source paranccsal hozhatjuk létre, legegyszerűbb esetben a fényforrás helyét és színét megadva.

```
light_source
 {
  <2, 4, -3>
  color White
 }
```

10.1.4 Testek és felületek

10.1.4.1 A gömb

Az egyik legegyszerűbb mértani test a gömb, melyet a `sphere` (gömb) parancs segítségével hozhatunk létre. A gömb egyértelmű megadásához a geometriában a gömb középpontját és sugarát kell ismernünk, és ez a színhelyleíró nyelvben is így van.

A Povraynek szüksége van e két információn kívül arra is, hogy milyen a gömb optikai szempontból. A gömbnek színe lehet mint a zöldborsónak, a fényt visszaverheti mint a vasgolyó, áthatolhat és megtörheti mint az üveggolyó, sugározhatja mint a lámpa vagy akár mintát is láthatunk a felszínén mint a márványon. A POVRAY minderre lehetőséget ad, segítségével olyan gömböket – és egyéb testeket – készíthetünk, amelyek soha nem léteztek.

Kezdjük a legegyszerűbb esettel, adjunk a felületnek speciális, névvel ellátott mintát:

```
sphere
 {
   <0, 1, 3>, 2
   texture { T_Stone23 }
 }
```

A sphere parancs után a gömb középpontja következik a három koordináta már ismert megadásával. Ezután vesszővel elválasztva következik

a gömb sugara. A gömb elkészítéséhez felhaszált harmadik jellemző az optikai megjelenést írja le. Erre a texture (szövet, szerkezet) parancsot használtuk, amellyel a Povray által ismert T_Stone23 textúrára hivetkoztunk.

Ha a T_Stone23 mintázatot használni akarjuk, akkor a POVRAY #include (beleértve) paranccsával a színhelyleíró állományunkba be kell emelnünk a stones.inc állományt, amely kövek mintázatait tartalmazza és a Povray csomag része. A gömb megtekinthető a 10.2 ábrán a létrehozásához felhasznált – fényforrással és kamerával kiegészített – színhelyleíró állomány pedig a következő:

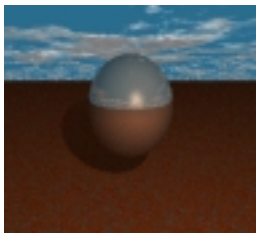
```
#include "colors.inc"
#include "stones.inc"

sphere
 {
   <0, 1, 3>, 2
   texture { T_Stone23 }
  }

background { color White }

camera
 {
   location <0, 2, -3>
   look_at  <0, 1,  3>
 }

light_source { <2, 4, -3> color White}
```

Ha megfigyeljük a képet, akkor látható, hogy a fény megcsillan a gömb felszínén. Ez azért lehetséges, mert az előre definiált T_Stone23 textúra fényvisszaverő tulajdonságokkal is rendelkezik, ezért visszaveri a kamera mögött található pontszerű fehér fényforrás fényét.

10.1.4.2 A sík

A plane (sík) parancs segítségével végtelen sík lapok készíthetőek, amelyek alkalmasak pl. az égbolt megrajzolására.

ábra 10.2: A `T_Stone23` textúra



ábra 10.3: Végtelen síkok

A síkok megadhatóak egy vektor és egy szám segítségével, ahol a vektor a síkra merőleges – ún. irányvektor – a szám pedig a síknak a koordinátarendszer középpontjától mért távolsága. Az irányvektort – mivel annak csak az iránya fontos számunkra – egyetlen pont által megadható, ha feltételezzük, hogy a másik végpont az origó. A parancs formája a sphere parancshoz igen hasonló:

```
plane
 {
   <0, 1, 0>, 5
   texture { Shadow_Clouds  }
 }
```

A `plane` után előbb az irányvektort adjuk meg, majd a távolságát az origótol. Az eredmény egy végtelen sík, amely a $ax+bx+cx-d\sqrt{a^2+b^2+c^2}=0$ összefüggésnek megfelelő pontokat foglalja magában, amennyiben a `plane { <a, b, c>,d }` értékekkel hoztuk létre.

A 10.3 ábra példát mutat két végtelen sík közt elhelyezett gömbre, a

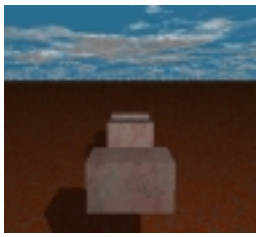
készítéséhez használt színhelyleíró állomány pedig a következő:

```
#include "colors.inc"
#include "stones.inc"
#include "textures.inc"
sphere
 {
   <0, 1, 6>, 2
   texture { Gold_Nugget }
  }
plane
 {
   <0, 1, 0>, 5
   texture { Shadow_Clouds  }
 }
plane
 {
   <0, 1, 0>, -3
   texture { Rust }
 }
camera
 {
   location <0, 2, -3>
   look_at  <0, 1,  3>
 }
light_source { <2, 4, -3> color White}
```

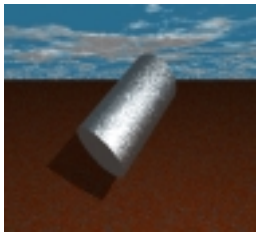
Látható, hogy újfajta textúrákat használtunk, amelyek a `textures.inc` állomány beillesztését teszik szükségessé a `#include` parancs segítségével.

10.1.4.3 A Téglatest

Téglatesteket a `box` (doboz) parancs segítségével hozhatunk létre. Minden téglatest létrehozásánál meg kell adnunk a két legtávolabbi sarokpont koordinátáit, valamint a már ismert optikai jellemzőket:



ábra 10.4: Téglatestek



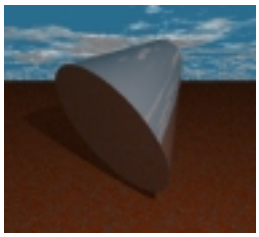
ábra 10.5: Henger

```
box
 {
   <-1, -.5, 1>, <1, .5, 2>
   texture { T_Stone5 }
 }
```

A téglatestek mindig a koordinátarendszerrel párhuzamosan – függőleges és vízszintes élekkel – jönnek létre a box parancs hatására. Ha ezen változtatni akarunk, akkor a fejezetben leírt művelettel el kell forgatnunk őket.

10.1.4.4 A henger

Hengert (10.5 ábra) a cylinder (henger) paranccsal hozhatunk létre. A henger létrehozásához meg kell adnunk a hengert záró két körlap középpontjának koordinátáit és a henger átmérőjének felét – vagyis a körlapok sugarát.



ábra 10.6: Csonka kúp

```
cylinder
 {
   <-1, 0, 3>, <1, 2, 5>, 1
   texture { Brushed_Aluminum }
 }
```

Amint látható előbb a körlapok középpontjainak koordinátáit kell megadnunk, majd a sugarat.

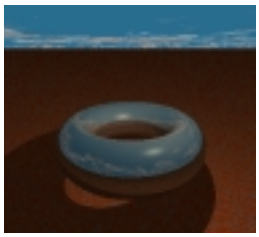
A hengerből cső készíthető, ha az open (nyitott) szóval a febőlapokat eltávolítjuk:

```
cylinder
 {
   <-1, 0, 3>, <1, 2, 5>, 1
   texture { Brushed_Aluminum }
   open
 }
```

10.1.4.5 A kúp és a csonka kúp

Kúpokat a cone (kúp) paranccsal hozhatunk létre. A kúp lehet csonka, vagyis hiányozhat a vége, ahogy azt a 10.6 ábrán láthatjuk.

A kúpok megadásához a kúpot határoló két körlap középpontját és a sugarukat kell megadnunk. A 10.6 ábra kúpja a következő paranccsal készült:



ábra 10.7: Tórusz

```
cone
 {
   <-1, 0, 5>, 3, <2, 3, 8>, 1
   texture { New_Penny }
 }
```

Az első három koordináta (-1, 0, 5) a kúp alaplapja, az utána következő szám (3) pedig az alaplap sugara. A <2, 3, 8>, 1 kifejezés a fedőlap középpontját adja meg a sugarával egyetemben.

Lehetőségünk van arra is, hogy a körlapokat eltávolítsuk az open (nyitott) kulcsszó segítségével, így egy csőszerű testhez jutva – amely leginkább egy szócsőhöz hasonlít. Az open kulcsszó megadásának módja:

```
cone
 {
   <-1, 0, 5>, 3, <2, 3, 8>, 1
   texture { New_Penny }
   open
 }
```

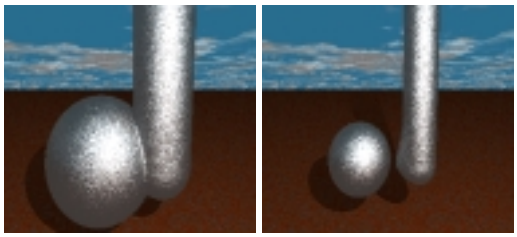
10.1.4.6 A tórusz

A tórusz (10.7 ábra) a `torus` paranccsal hozható létre. Meghatározásához a nagyátmérőt (a teljes test átmérőjét) és a kisátmérőt (a tóruszt alkotó henger átmérőjét kell megadnunk):

```
torus
 {
   3, 1
   texture { Bright_Bronze }
 }
```

A tórusz mindig origó középponttal vízszintesen jön létre, ezért ha használni akarjuk, akkor általában szükségünk van a 10.1.6 és 10.1.7fejezetekben bemutatott mozgatásra és forgatásra. A példa előállításánál – az egyszerűség kedvéért – a kamerát mozgattuk olyan helyzetbe, ahonnan az egész tórusz jól látható. A teljes állomány a következő:

```
#include "colors.inc"
#include "stones.inc"
#include "textures.inc"

torus
 {
   3, 1
   texture { Bright_Bronze }
 }
plane
 {
   <0, 1, 0>, 5
   texture { Shadow_Clouds  }
 }
plane
 {
   <0, 1, 0>, -3
   texture { Rust }
 }
camera
 {
   location <0, 4, -10>
   look_at  <0, 0,  3>
 }
light_source { <2, 4, -3> color White}
light_source { <2, -4, -3> color White}
```



ábra 10.8: Az aura

Látható, hogy a kamera az előző példákhoz képes elmozdult, valamint a jobb megvilágítás érdekében két pontszerű fényforrást használtunk.

10.1.4.7 Aura

A `blob` (paca) parancs segítségével igen különös testeket hozhatunk létre. Az alapgondolat a következő: mérteni testek körül egyfajta aurát hozunk létre, amely a testtől távolodva egyre veszít erejéből. Ha megrajzoljuk azt a felületet, amely mentén az aura bizonyos erősséggel bír, akkor új, speciális formájú testet vagy testeket kapunk (10.8 ábra). Kissé matematikusan fogalmazva az aura egy ekvipotenciális felület, amelyet egy skalártérben rajzolunk meg. Az aura egy speciális test, összetettebb mint az eddig megismert testek. Annak ellenére, hogy több résztestből áll nem a testek egyesítésének egy válfaja (10.1.5 fejezet), mivel felépítéséhez csak gömb és henger használható.

```
blob
  {
    threshold 0.5
    sphere
     {
      <-1, 0, 1.5>, 2, 1
     }
    cylinder
     {
      <+1, 0, 1>, <+1, 4, 1>, 1, 1
     }
    texture { Brushed_Aluminum }
  }
```

A blob – amint a példa mutatja – a treshold (határ) megadásából és a hengerek, valamint gömbök felsorolásából áll. A treshold határozza meg, hogy a testek körül lévő tér milyen erősségű pontajai alkossák a blob felületét. Ha e számot nagyobbra veszzük a blob kisebb lesz, sőt résztestekre szét is eshet (amint azt a 10.8 ábra második képe mutatja).

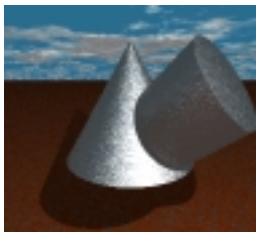
A hengerek és gömbök mindegyikénél megadható egy szám – a térerő –, amely megadja, hogy az adott test mennyivel járul hozzá a testek körül található tér erősségéhez. Az egyes számok jelentése a következő[4]:

```
sphere
 {
  <a gömb középpontja>, a sugár, térerő
 }
cylinder
 {
  <alaplap középpontja>, <fedlap középpontja>,
  a henger sugara, térerő
 }
```

Minél nagyobbra vesszük az adott test térerejét, annál erősebb lesz a körülötte található tér, ezért az aura annál távolabb kerül tőle.

[4] A <> jelek közt mindig három szám, a három koordináta található – ahogy azt már láttuk.

ábra 10.9: Összeolvadó gömbök



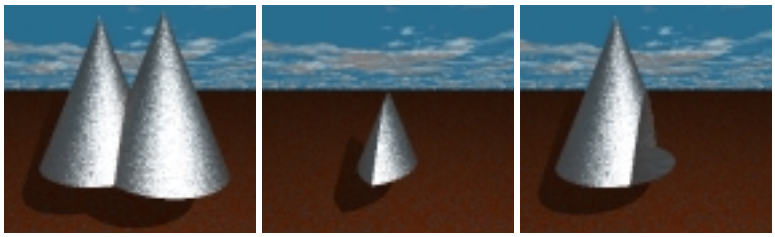
ábra 10.10: Testek áthatása

10.1.5 Testek egyesítése

Ha két vagy több gömböt hozunk létre, amelyeknek közös pontjaik vannak, akkor azok egymásba hatolnak és egyetlen test érzetét keltik. A 10.9 ábrán látható gömbök egymástól eltérő textúrával készültek.

Remészetesen nem csak egyforma testek foglalhatják el a térnek ugyanazon részét, hanem egymástól különbüzőek is. Erre mitat példát a 1.1 ábra.

Ha a több testből álló átfedő alakzatot együtt szeretnénk használni – közösen forgatni, mozgatni, stb. – akkor az `union` (egyesítés) parancs segítségével azokat össze kell kapcsolnunk. A union parancs után a kapcsos zárójeleken belül fel kell sorolnunk az összes testet, amelyet együvé akarunk kapcsolni:



ábra 10.11: A `union`, `intersection` és a `difference` hatása

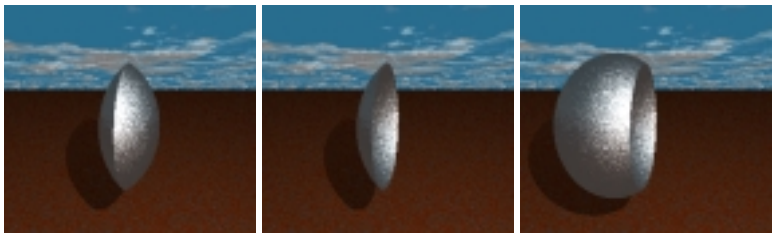
```
union
 {
    cone
     {
      <-1, 0, 1.5>, 2, <-1, 4, 1.5>, 0
     }
    cone
     {
      <+1, 0, 1>, 2, <+1, 4, 1>, 0
     }
    texture { Brushed_Aluminum }
}
```

Lehetőségünk van arra is, hogy a két test közös pontjait tartsuk meg, vagyis a két test közös részéből egy új testet hozzunk létre, amint az a ábrán látható. Erre az `intersection` (metszet) parancs szolgál, hasonló formában.

Két testet úgy is egyesíthetünk egymással, hogy az egyiket csonkoljuk a másikkal. Erre a difference parancs szolgál, amely az első testet megrajzolja, de csak azon részen, ahol a második test nincsen jelen. A difference parancs esetében nem mindegy, hogy melyik testet írjuk előre s melyiket másodiknak, hiszen az első test a képen megjelenik, amásodik pedig nem. A 10.11 ábrán láthatjuk a parancsok hatását.

Meglehetősen zavaró lehet, hogy a POVRAY által modellezett testek üregesek és amikor a valamelyiket csonkoljuk, akkor belelőtunk üreges belsejébe. Ez látható a 10.11 ábra utolsó képén is.

A megoldás egy „fedlap” elkészítése lehet, amelyet ugyanannak az állománynak a bővítésével, több lépésben mutatunk be. Mivel a zárt test



ábra 10.12: „Fedlap” készítése

elkészítéséhez többször is felhasználjuk a két testet, azokra új jelölést vezetünk be a `#declare` parancs segítségével.

```
#declare test = sphere{ <-1, 1, 1.5>, 2 }
#declare ollo = sphere{ <+1, 1, 1>, 2   }
```

A `#declare` parancs segítségével bármilyen rövidítést bevezethetünk, nem kell mást tennünk, mint beírni egy nevet, amely után egyenlőségjellel beírjuk azt, hogy mit akarunk vele rövidíteni. A fenti két sor hatására minden helyen ahol a `object{test}` és `object{ollo}` szerepel, automatikusan a két gömb kerül felhasználásra.

Első lépésben készítsük el a két test metszetét (10.12 ábra első kép):

```
intersection
{
  object{test}
  object{ollo}
  texture { Brushed_Aluminum }
}
```

Vegyük el ebből a lencse alakú metszetből azt a részt, amelynek nem szabad a képen lennie, mivel az ollóként használt test része (10.12 ábra második kép):

```
difference
 {
   intersection
   {
     object{test}
     object{ollo}
   }
   object{ollo}
   texture { Brushed_Aluminum }
 }
```

Így megkaptuk a fedőlapot, amellyel a csonkolt testet be tudjuk fedni. Tegyük ezt lapot a csonkolt testhez, hogy megkapjuk a végeredményt (10.12 ábra harmadik kép):

```
union
{
  difference
   {
      object{test}
      object{ollo}
   }
  difference
   {
     intersection
     {
       object{test}
       object{ollo}
     }
     object{ollo}
   }
  texture { Brushed_Aluminum }
}
```

ábra 10.13: A GIMP főablakai

10.1.6 Elmozdítás

10.1.7 Forgatás

10.2 A GIMP

A GIMP (*GNU Image Manipulation Program*) egy igen hatékony és komplex grafikus szerkesztőprogram, amely alkalmas fotorealisztikus képek, grafikák készítésére és manipulálására. A GIMP használatával lehetőségünk nyílik fotók retusálására, reklámgrafikák készítésére vagy akár a Povray által készített képek további feldolgozására.

10.2.1 A munkamenet

A GIMP indítása a `gimp` parancs begépelésével lehetséges X WINDOW alatt. Ha elindítottuk, a 10.13 ábra baloldalán látható ablak jelenik meg a képernyőn. A megjelenő puritánul egyszerű ablak alapján igen könnyű a GIMPet alábecsülni. Ne hagyjuk, hogy ez a kicsiny ablak megtévesszen bennünket, mivel a GIMPpel való munka során igen hamar kevésnek bizonyulhat a képernyő – a megjelenő újabb és újabb ablakok miatt.

A gimp indításakor egy képet tartalmazó állomány nevét is megadhatjuk a parancssorban, ez esetben azt a GIMP szerkesztésre megnyitja. Új kép készítésére a File menü New menüpontja alkalmazható. Új kép létrehozásakor be kell állítanunk a létrehozni kívánt kép tulajdonságait, erről a fejezetben olvashatunk bővebben.

A tulajdonképpeni munka a szerkesztőablakban végezhető el, amelye a 10.13 ábra bal oldalán látható. A felhasználó ebben az ablakban dolgozik, miközben folyamatosan figyelemmel kíséri a kép fejlődését, ezért a program használatához jóminőségű monitor és grafikus rendszer ajánlható, amely legalább 16 bpp színmélységgel és 124x768 képpontos felbontásban képes múködni.

A kép szerkesztés közben több egymáson fekvő rétegből áll, amelyek egymástól függetlenül szerkeszthetőek és megjeleníthetőek. A munka

ábra 10.14: Rétegek a GIMP használata közben

során több ilyen réteget használunk a kép egyes elemei számára. Általában a kép elkészülte után a rétegeket egyesítjük valamilyen szabályok szerint és a kész képet mentjük. Ha munka közben kívánunk mentést végezni – mert pl. a képet csak másnap fejezzük be –, akkor mindenképpen a GIMP saját képformátumát (xcf) kell használnunk, mert ez lehetővé teszi, hogy az összes réteget egyetlen állományba mentsük. Ha a mentés során, az egyes rétegeket nem egyesítettük és nem xcf formában mentettünk, akkor a képnek csak egyetlen rétege – az éppen aktív – kerül mentésre, a többi elvész.

Függelék A

Szintaktikai összefoglaló

A könyvben többször használtunk gépi nyelveket a számítógéppel való kommunikációra. Ehelyütt ezeket a nyelveket foglaljuk össze, remélve, hogy jól használható emlékeztetőt adunk az olvasó kezébe. Azokat a nyelvi elemeket is felsoroljuk itt, amelyeket az egyes fejezetekben nem említettünk, ellentétben a részletes leírásokkal itt megpróbálunk az egészlegességre törekedni. Tesszük ezt azért, hogy a programok részletes leírásának segítségével megtanult ismereteket kiegészítsük az adott program kezelésében már járatos felhasználó számára.

A nyelvek leírásánál bevett számítástechnikai jelölésrendszert használunk – a nyelvet leíró *metanyelv*ként. E metanyelvben legtöbbször a szögletes zárójellel találkozhatunk, amellyel opcionális elemeket zárunk közre, vagyis ami szögletes zárójelekben van, azt nem kötelező használni:

```
túrós csusza [tepertővel]
```

A fenti kifejezés megjelenhet „túrós csusza” és „túrós csusza tepertővel” formában is. A szögletes zárójelek egymásba ágyazva is előfordulnak, ekkor értelemszerűen további opcionális elemeket jeleznek:

```
túrós csusza [tepertővel [sóval]]
```

Ha az opcionális elemeket jelző szögletes zárójelben a függőleges vonal (|) karaktert látjuk, akkor választási lehetőségre kell gondolnunk. A | jellel elválasztott elemek közül mindig csak egyet választhatunk:

```
túrós csusza [tepertővel | cukorral]
```

A fenti kifejezés jelöli a „túrós csusza”, „túrós csusza tepertővel” és „túrós csusza cukorral” kifejezéseket.

A gépi nyelvek elméleti alapját a generatív grammatika adja, ezért a leírásukban is sokszor alkalmazzuk a kifejezések kategóriákból való felépítését. Nagyobb nyelvi elemek behelyettesítése esetén a kapcsos zárójeleket használjuk: (hopplá, lehet, hogy inkább a kurzív szedést?)

```
savanyúság
  uborkasaláta [ tejfellel ] |
  csemege uborka |
  ecetes paprika
köret
  galuska |
  rizs |
  főtt burgonya
rendszergazda mai ajánlata
  zúzapörkölt { körettel } és { savanyúsággal }
```

Amint látható az ebéd leírásához először a lehetséges alapelemek definiálásával láttunk neki. Amikor az alapelemeket egyértelműen leírtuk, azokat a kifejezésbe kapcsos zárójelekkel építettük be, jelezve, hogy nagyobb egységekről van szó – amelyeket már ismerünk. A kifejezés ilyen hierarchikus rendszerbe foglalása átláthatóvá teszi a nyelvet, mivel az előre deklarált kifejezéseket több helyen is felhasználhatjuk. A kapcsos zárójelek is egymásba ágyazhatóak abban az esetben, ha az egyes részkifejezések definiálásához már nevesített részeket használunk fel.

A.1 A BASH scriptek nyelve

```
if feltetel; then
  utasitasok;
fi
case szo in
  [( szo [| szo]...) parancsok;;]...
esac
```

```
until feltétel;
  do
    ciklusmag;
  done
while feltétel;
  do
    ciklusmag;
  done
for név [in nevek];
  do
    ciklusmag
  done
```

A.2 A Postgres SQL nyelve

A felhasznált rövidítések

felhjog

```
ALL | SELECT | INSERT | UPDATE | DELETE | RULE
```

típus

```
bool | box | char(n) | circle | date | float4 | float8 | int2 | int4 | line | lseg | mor
```

állandó

Az állandó megjelenési formája a típustól függ. Szöveges és dátumos állandókat egyszeres idézőjelekkel bezárva kell megadnunk.

érték

Az érték megjelenési formája a típustól függ.

feltétel

A feltétel valamilyen oszlopnév és állandó közt fennálló reláció vizsgálatát jelenti, mely lehet igaz és hamis. Pl.: `netto<1000`

Táblázat létrehozása

```
CREATE TABLE táblanév
 (
   oszlopnév típus [DEFAULT állandó] [NOT NULL]
[, oszlopnév típus [DEFAULT állandó] [NOT NULL] [, ...] ]
 )
   [INHERITS ( táblanév [, táblanév] )]

[CONSTRAINT cnév CHECK ( feltétel ) [, CHECK ( feltétel ) ] ]
   [ARCHIVE = NONE | LIGHT | HEAVY ];
```

A DEFAULT kulcsszó után megadhatunk egy értéket, amelyet a mező akkor vesz fel, ha a sor hozzáadásakor nem írunk elő értéket. A NOT NULL kiírásával jelezhetjük, hogy az adott mezőt mindenképpen meg kell adni ahhoz, hogy új sort hozzunk létre a táblában. Az INHERITS kulcsszó után felsorolt tábláktól az újonan létrehozott táblázat „örökli” az oszlopneveket és oszloptípusokat. Ha emiatt több oszlopot kellene létrehozni azonos névvel, a POSTGRES hibaüzenettel megtagadja az új tábla létrehozását. A CONSTRAINT kulcszó után megadott feltétel(ek) minden egyes új sor létrehozásakor ellenőrzésre kerülnek. Ha az újonan beszúrásra kerülő sorok valamelyik feltételt sértik, a POSTGRES megtagadja az adatok beszúrását és indoklásul a cnév kinyomtatásra kerül.

Táblázat törlése

```
DROP TABLE táblanév { , táblanév };
```

Táblázat átnevezése

```
ALTER TABLE táblanév
            RENAME TO táblanév;
```

Adatok hozzáadása

```
INSERT INTO táblanév
           [(att.expr-1 [,att_expr.i] )]

{VALUES (expression1 [,expression-i] ) |
           select expression1 [,expression-i]
            [from from-list] [where qual];
```

Adatok listázása

```
SELECT [DISTINCT]
           expression1 [AS oszlopnév]
           {, expression-1 [AS oszlopnév]}
           [INTO TABLE táblanév]
           [FROM from-list]
           [WHERE where-clause]
           [GROUP BY oszlopnév [, oszlopnév ]
           [ORDER BY oszlopnév

[USING művelet] [, oszlopnév [USING művelet] ];
```

Sorok törlése

```
DELETE FROM táblanév [ WHERE feltétel ];
```

Sorok módosítása

```
UPDATE táblanév SET oszlopnév = érték
           [, oszlopnév = érték ]
           [FROM from-list]
           [WHERE feltétel];
```

Oszlopok hozzáadása

```
ALTER TABLE táblanév [ * ]
           ADD ( oszlopnév típus );
```

Oszlopok átnevezése

```
ALTER TABLE oszlopnév [*]

RENAME [COLUMN] oszlopnév TO oszlopnév;
```

Index létrehozása

```
CREATE [UNIQUE] INDEX indekszn év
           ON táblanév [USING am-name]
           ( oszlopnév [type_class], ... );
```

Index törlése

```
DROP INDEX indekszn év;
```

Felhasználói jogok megadása

```
GRANT felhjog [, felhjog,...]>
        ON táblanév [,...<táblanév>]

TO [PUBLIC | GROUP csoportnév | felhnév ];
```

Felhasználói jogok elvétele

```
REVOKE felhjog [, felhjog,...]>
        ON táblanév[,...<táblanév>]

FROM [PUBLIC | GROUP csoportnév | felhnév ];
```

Rendszertisztítás

```
VACUUM [VERBOSE] [ANALYZE] [táblanév [(oszlopnév,...)]];
```

A.3 A POV-RAY színhelyleíró nyelve

Ezek a holmik mindenképpen kellenek ide, mert nincs ember aki ezen eligazodik enélkül. Ráadásul ez még legalább háromszorosára bővül. Valahogy képletessé kell tenni, jó volna a kifejezések mellé képeket tenni...

Az utasításokat ehelyütt kisbetűvel írtuk, mivel a színhelyleíró nyelv szintaxisa így követeli meg. Azok a szavak, amelyek legalább egy nagybetűvel rendelkeznek, nem a színhelyleíró nyelv elemei, s ezt az is jelöli, hogy magyarul vannak írva.

A három értékkel jellemezhető koordinátákat <> jelek közt adtuk meg, ahogyan azt a színhelyleíró nyelvben is kell. Ahol tehát <> között

azonosító szót látunk <x, y, z> formában meg kell adnunk a három koordinátát. Az azonosítószó ezeken a helyeken utal a jelöletre, vagyis a három koordináta szerepére.

Testek

```
box { <Sarok>, <Sarok> }

cone {

<BASE_POINT>, BASE_RADIUS, <CAP_POINT>, CAP_RADIUS
    [ open ] }

cylinder {
    <BASE_POINT>, <CAP_POINT>, RADIUS
    [ open ] }

sphere {
    <CENTER>, RADIUS }

text {
    ttf "FONTNAME.TTF",
    "STRING_OF_TEXT",
    THICKNESS_FLOAT, OFFSET_VECTOR
}
torus {
    MAJOR, MINOR
    [ sturm ]
}
disc {
    <CENTER>, <NORMAL>, RADIUS [, HOLE_RADIUS ]
}
plane { <NORMAL>, DISTANCE }
```

Fényforrás

```
    light_source {
        <LOCATION>
        color <COLOUR>
        [ spotlight ]
        [ point_at <POINT_AT> ]
        [ radius RADIUS ]
        [ falloff FALLOFF ]
        [ tightness TIGHTNESS ]
        [ area_light <AXIS1>, <AXIS2>, SIZE1, SIZE2 ]
        [ adaptive ADAPTIVE ]
        [ jitter JITTER ]
        [ looks_like { OBJECT } ]
        [ fade_distance FADE_DISTANCE ]
        [ fade_power FADE_POWER ]
        [ atmospheric_attenuation BOOL ]
    }
    light_source {
        <LOCATION>
        color <COLOUR>
        [ looks_like { OBJECT } ]
        [ fade_distance FADE_DISTANCE ]
        [ fade_power FADE_POWER ]
        [ atmospheric_attenuation BOOL ]
    }
    light_source {
        <LOCATION>
        color <COLOUR>
        spotlight
        point_at <POINT_AT>
        radius RADIUS
        falloff FALLOFF
        tightness TIGHTNESS
        [ looks_like { OBJECT } ]
        [ fade_distance FADE_DISTANCE ]
        [ fade_power FADE_POWER ]
        [ atmospheric_attenuation BOOL ]
    }
```

Textúra

```
texture {
   TEXTURE_IDENTIFIER
   pigment {...}
   normal {...}
   finish {...}
   halo {...}
   TRANSFORMATIONS
}
pigment {
    PIGMENT_IDENTIFIER
    PATTERN_TYPE
    PIGMENT_MODIFIERS...
  }
finish {
    FINISH_IDENTIFIER
    [ ambient COLOR ]
    [ diffuse FLOAT ]
    [ brilliance FLOAT ]
    [ phong FLOAT ]
    [ phong_size FLOAT ]
    [ specular FLOAT ]
    [ roughness FLOAT ]
    [ metallic [ FLOAT ] ]
    [ reflection COLOR ]
    [ refraction FLOAT ]
    [ ior FLOAT ]
    [ caustics FLOAT ]
    [ fade_distance FLOAT ]
    [ fade_power FLOAT ]

[ irid { thickness FLOAT turbulence VECTOR } ]
    [ crand FLOAT ]
}
```

```
halo {
    attenuating | emitting | glowing | dust
    [ constant | linear | cubic | poly ]
    [ planar_mapping | spherical_mapping |
      cylindrical_mapping | box_mapping ]
    [ dust_type DUST_TYPE ]
    [ eccentricity ECCENTRICITY ]
    [ max_value MAX_VALUE ]
    [ exponent EXPONENT ]
    [ samples SAMPLES ]
    [ aa_level AA_LEVEL ]
    [ aa_threshold AA_THRESHOLD ]
    [ jitter JITTER ]
    [ turbulence <TURBULENCE> ]
    [ octaves OCTAVES ]
    [ omega OMEGA ]
    [ lambda LAMBDA ]
    [ colour_map COLOUR_MAP ]
    [ frequency FREQUENCY ]
    [ phase PHASE ]
    [ scale <VECTOR> ]
    [ rotate <VECTOR> ]
    [ translate <VECTOR> ]
  }
warp
  {
    black_hole <CENTER>, RADIUS
    [falloff VALUE]
    [strength VALUE]
    [repeat <VECTOR>]
    [turbulence <VECTOR>]
    [inverse]
  }
```

```
    atmosphere {
        type TYPE
        distance DISTANCE
        [ scattering SCATTERING ]
        [ eccentricity ECCENTRICITY ]
        [ samples SAMPLES ]
        [ jitter JITTER ]
        [ aa_threshold AA_THRESHOLD ]
        [ aa_level AA_LEVEL ]
        [ colour <COLOUR> ]
    }
    fog {
        fog_type FOG_TYPE
        distance DISTANCE
        colour <COLOUR>
        [ turbulence <TURBULENCE> ]
        [ turb_depth TURB_DEPTH ]
        [ omega OMEGA ]
        [ lambda LAMBDA ]
        [ octaves OCTAVES ]
        [ fog_offset FOG_OFFSET ]
        [ fog_alt FOG_ALT ]
        [ up <FOG_UP> ]
        [ TRANSFORMATION ]
    }
    rainbow {
        direction <DIR>
        angle ANGLE
        width WIDTH
        distance DISTANCE
        color_map { COLOUR_MAP }
        [ jitter JITTER ]
        [ up <UP> ]
        [ arc_angle ARC_ANGLE ]
        [ falloff_angle FALLOFF_ANGLE ]
    }
```

Tárgymutató