Unix

Pásztor György

Utolsó módosítás: 2008. január 22.

Tartalomjegyzék

0. A Unix története 1

1. A boot folyamat, kernel 2
1.1. A boot média . . . 2
1.1.1. Bootmanagerek . . . 2
1.2. BootManager . . . 3
1.3. A kernel és az initrd . . . 3
1.4. init, inittab . . . 3

2. Felhasználók, csoportok 4
2.1. A rendszer felhasználókezelése . . . 4
2.2. parancsok . . . 5

3. A fájlrendszer 7
3.1. Merevlemezek felosztása, partíciók . . . 7
3.1.1. A linux eszköznévelnevezési sémái . . . 7
3.1.2. Partícionálás . . . 8
3.1.3. A partíciók eszköznevei . . . 9
3.2. Eszköz-fájlrendszer kapcsolat . . . 9
3.2.1. fájlrendszer-típusok . . . 9
3.2.2. Eszközök csatlakoztatása . . . 10
3.2.3. Lemezhasználtság . . . 10
3.2.4. Az fstab . . . 12
3.3. Fájlrendszer . . . 13
3.3.1. i-node tábla . . . 13
3.3.2. Fájlkezelő parancsok . . . 14
3.3.3. Speciális eszközök . . . 17
3.3.4. Jogosultságok . . . 18
3.4. A Unix könyvtárai . . . 22
3.4.1. /boot . . . 22
3.4.2. /bin, /sbin, /lib . . . 22
3.4.3. /etc . . . 22
3.4.4. /tmp . . . 22
3.4.5. /home, /root . . . 22
3.4.6. /usr, /usr/bin, /usr/sbin, /usr/lib . . . 23
3.4.7. /usr/src . . . 23
3.4.8. /usr/local . . . 24
3.4.9. /var, /var/spool, /var/log . . . 24
3.4.10. /opt . . . 24

4. Folyamatok **25**
4.1. A processzlista . . . 25
4.1.1. A parancs . . . 25
4.1.2. Paraméterek . . . 25
4.1.3. Szülő folyamat . . . 25
4.1.4. Környezeti változók . . . 25
4.1.5. Tulajdonosi jogosultságok . . . 26
4.1.6. Effektív jogosultságok . . . 26
4.1.7. Prioritás . . . 26
4.1.8. Aktuális könyvtár . . . 26
4.1.9. Root könyvtár . . . 27
4.1.10. Egyéb erőforrások . . . 27
4.2. A folyamatok kezelésére szolgáló parancsok . . . 27
4.2.1. ps . . . 27
4.2.2. top . . . 28
4.2.3. kill . . . 28
4.2.4. jobs, fg, bg . . . 29
4.2.5. export . . . 29
4.2.6. nice . . . 29

5. A parancsértelmező **30**
5.1. Jobvezérlés . . . 30
5.1.1. Vezérlőbillentyűk . . . 30
5.1.2. Vezérlő operátorok . . . 31
5.1.3. Átirányítások . . . 31
5.2. Programozási lehetőségek . . . 32
5.2.1. Feltételes elágazások . . . 32
5.2.2. Számlálásos ciklus . . . 33
5.2.3. Elöltesztelős ciklus . . . 34
5.2.4. Többágú elágazás . . . 34
5.2.5. Feltételek . . . 35
5.2.6. Változók, behelyettesítések . . . 36
5.2.7. Függvények . . . 37

6. Unixos segédprogramok **38**
6.1. grep . . . 38
6.2. sed . . . 38
6.3. cut . . . 39
6.4. head . . . 39
6.5. tail . . . 40
6.6. wc . . . 40
6.7. sort . . . 41
6.8. uniq . . . 41

7. Virtuális Memóriakezelés **42**

8. VI és regexpek **43**
8.1. Reguláris kifejezések . . . 43
8.2. Az alap vi bemutatása . . . 43
8.2.1. A command mód fontosabb parancsai . . . 44
8.2.2. Az ed mód fontosabb parancsai . . . 45

8.3. A VIM-ben megjelent új parancsok 45
8.4. A VIM-ben megjelent új ed-szerű parancsok 46
8.5. Makrózás a vim-ben 46
8.6. Néhány hasznos alapbeállítás, .vimrc 46
8.7. További vim információk 47

Jegyzékek **48**
Ábrajegyzék 48
Példák jegyzéke 49
Irodalomjegyzék 50

0. fejezet

A Unix története

Ebben a fejezetben egyelőre nincs részletezés a unix történetéről, a két unix ágról (BSD és SysV) és a többi történetről, csak néhány link, hogy hol lehet részletesebben utánanézni.

- http://hup.hu/1941-1979
- http://hup.hu/1980-1999
- http://hup.hu/2000-napjainkig
- http://hup.hu/bsd_tortenet_1
- http://hup.hu/bsd_tortenet_2
- http://hup.hu/bsd_tortenet_3
- http://hup.hu/bsd_tortenet_4

Amit még illik tudni a GNU és a Linux története.

1. fejezet

A boot folyamat, kernel

A számítógép bekapcsolásától az operációs rendszer által nyújtott szolgáltatások elindulásáig sok minden történik, több különböző lépcsőben. Az első lépés általában bármilyen gépről is legyen szó, az hogy a gép firmware-je inicializálja a hardvereszközöket. Ez a személyi számítógépek esetén a BIOS amely a CPU-t és a memóriát ellenőrzi, alapállapotba hozza a kártyákat és a konfigurációjának megfelelő sorrendben megpróbálja az operációs rendszert valamelyik perifériáról elindítani.

1.1. A boot média

A BIOS képességeitől függően több különböző médiáról is indulhat az operációs rendszer. Ezek közül a főbb csoportok:

- merevlemez[1]
- optikai meghajtó (CD-ROM, DVD)
- hálózat

A BIOS attól függően, hogy milyen médiáról van szó, megpróbál betölteni valamilyen bootmanagert. A merevlemezek esetén egy boot managernek (illetve azt betöltő kódnak) kell a lemez ún. Master Boot Recordjában lennie, ami a lemez első szektora. Optikai adathordozók esetén az Eltorito szabvány ad alapot a bootmanagereknek, hálózat esetén pedig a PXE szabvány terjedt el.

A GNU/Linuxot is betölteni képes boot managerek közül a két legelterjedtebb a GRUB (GRand Unified Bootloader) és a LILO (LInux LOader). Mindkettő telepíthető közvetlenül a lemez MBR-jébe, de telepíthető partíción belülre is, valamint képesek másik partícióban levő bootloadernek átadni a vezérlést, így akár másik partícióba telepített alternatív operációs rendszereket is képesek elindítani.

1.1.1. Bootmanagerek

LILO Csak merevlemezes bootolásra használható. Telepíthető szoftveres raid eszközbe, mbr-be, partícióra.

GRUB Telepíthető MBR-be, partícióba, swraid tükör partíciójába. Létezik pxegrub nevű változata hálózati boothoz, és van Eltoritos változata natív BootCD készítéshez.

[1] A hajlékonylemezt (floppy) is felsorolhatnánk, de manapság az operációs rendszerek már akkorára híztak, hogy a floppyk kapacitása ehhez képest csak minimális feladatokra elég

syslinux A syslinuxot tipikusan FAT-os partícióra szokás telepíteni, hogy onnan töltse be a Linux kernelt. Van isolinux nevű BootCDhez használható verziója, és pxelinux nevű PXEhez használható verziója.

1.2. BootManager

Legyen bármilyen bootmanagerről is szó, képesnek kell lennie a linux kernelt, vagy választhatóan menüből/konfigurációból felkínálva választhatóan kernelt betöltenie, annak paramétereket átadnia, stb. A boot managerhez tartozó konfigurációs fájl ezt tartalmazza, hogy milyen nevű fájlban is van a kernel, illetve a hozzá tartozó initrd, illetve milyen paramétereket kell a kernelnek átadjon.

További különbség a bootmanagerek közt, hogy a lilo a telepétésekor/újratelepítésekor olvassa be, hogy a kernel fájljának részei a lemez mely blokkjaiban vannak, és erről csinál egy térképet, addig az syslinux képes értelmezni a fat fájlrendszert, és asszerint betölteni a kernelfájl tartalmát. A grub ismeri az ext2/3, xfs, reiserfs, jfs, fat fájlrendszereket és értelmezni azokat, így azok bármelyikéről képes betölteni a kernelt. Az isolinux és isogrub ismeri az iso9660 fájlrendszerű cdromokat, valamint a pxelinux és pxegrub pedig a tftp hálózati protokollt, hogy azon keresztül töltse be a kernelt a memóriába.

1.3. A kernel és az initrd

A kernel az a rész, amely...

- az számítógépünkben található hardverelemek vezérlőprogramjait tartalmazza (driverek),...
- felel a memóriakezelésért (swap, virtuális memória),...
- kezeli a párhuzamosan futó feladatokat (process/task),...
- intézi a folyamatok ütemezését (scheduler),...
- intézi a számítógép hálózatban való részvételét (tcp/ip),...
- kezeli a rendszer tűzfalát (netfilter),...
- megvalósít virtuális fájlrendszer kezelést (vfs: ext2, ext3, xfs, fat, jfs, reiserfs, ...),...
- vezérli a folyamatok közti kommunikációt (ipc),...

Mivel azonban a kernel ennyi mindennel együtt óriási nagyra nőtt, ezért ezen funkciók egy részét betölthető modulként is lehet használni. A bootmanager által betöltendő kernel méretének lecsökkentésére ezek jórészét kihagyják belőle és csak a legszükségesebbeket tartalmazza. Azok a driverek amelyek a rendszer elindításához is kellenek, belekerülnek egy a bootkor a kernellel egyszerre betöltésre kerülő virtuális memórialemezbe (ramdisk), amelyet **initrd**nek hívunk.

A kernel mint fájl, tipikusan a gyökérkönyvtárban, vagy a `/boot` könyvtárban található meg, és `vmlinuz`, vagy `bzImage` nevet visel. Szokás még a fájlnévbe a verziószámát is beletenni. Hasonlóan az indítást segítő initrd-t `initrd` néven találhatjuk meg a kernel mellett.

1.4. init, inittab

A kernel, és a meghajtó programok után az `/sbin/init` parancs az első ami elindul, és indít el minden szolgáltatást, alfolyamatot, stb. a `/etc/inittab`-ban meghatározott konfiguráció alapján.

2. fejezet

Felhasználók, csoportok

2.1. A rendszer felhasználókezelése

Az operációs rendszert mindent számokkal tart nyilván. Minden egyes fájlnak, futó folyamatnak van egy tulajdonosa (és tulajodonos csoportja). A felhasználói nevek és számok egymásnak való megfeleltetéséről az azt kezelő segédprogramoknak kell gondoskodnia. Hogy ezek egységes módon legyenek kezelhetők ezen információk a felhasználókról a `/etc/passwd` fájlban vannak. A `passwd` fájl egy-egy sora kb. így néz ki:

```
root:x:0:0:root:/root:/bin/bash
...
pasztor:x:1000:1000:PASZTOR Gyorgy,,,:/home/pasztor:/bin/bash
```

A hét oszlop jelentése a következő:

pasztor A felhasználó loginneve

x A felhasználó elkódolt jelszavának helye

1000 A felhasználó user ID-je (uid)

1000 A felhasználó elsődleges csoport ID-je (gid)

SZTOR Gyorgy Gecos információk: A felhasználó valódi neve, telefonszáma, stb.

/home/pasztor A felhasználó home könyvtára

/bin/bash A felhasználó alapértelmezett shellje

A csoportokról szóló információk a `/etc/group` fájlban vannak. Egy-egy átlagos sora kb. így néz ki:

```
root:x:0:
...
admin:x:109:pasztor
```

admin A csoport neve

x A kódolt jelszó helye

109 A csoport ID-je (gid)

pasztor A csoport tagjainak felsorolása, vesszővel elválasztva

A biztonság növelése érdekében a kódolt jelszavak átkerültek a `shadow` és `gshadow` fájlokba, amelyekhez csak rendszergazdai jogosultságokkal lehet hozzáférni. A jelszót ellenőrizni képes folyamatok (pl. sshd) ezért rendszergazdai jogosultságokkal futnak.

A **passwd** parancs úgy képes megváltoztatni a jelszavunkat, hogy maga a parancsfájl *setuid*es, így amikor fut rendszergazdai jogosultságokkal fut, de képes ellenőrizni, hogy melyik felhasználó indította el, és csak az ő jelszavát hajlandó megváltoztatni, és azt is csak akkor, ha az eredeti jelszót már helyesen megadtuk.

2.2. parancsok

whoami Kiírja a felhasználói nevem.

id Kiírja a felhasználói információim. Beleértve a csoporttagsági információkat is.

w (work)Kiírja, melyik felhasználó milyen folyamatot futtat épp a terminálján.

who Kiírja kik vannak a rendszerre belépve.

who am i Kiírja, hogy a belépések közül melyik is az, amelyik az épp aktuális parancssorhoz tartozik

finger A w-hez hasonló. Ha egy felhasználónevet paraméterként megadva indítjuk el, akkor az adott felhasználó információit részletezi.

Példák:

```
pasztor@clyde:~$ whoami
pasztor
pasztor@clyde:~$ id
uid=1000(pasztor) gid=1000(pasztor) csoportok=4(adm),6(disk),20(dialout),24(cdrom),25(fl
pasztor@clyde:~$ who am i
pasztor  pts/1        2007-12-02 18:04 (:0.0)
pasztor@clyde:~$ who
pasztor  :0           2007-12-02 18:04
pasztor  pts/0        2007-12-02 18:04 (:0.0)
pasztor  pts/1        2007-12-02 18:04 (:0.0)
pasztor  pts/2        2007-12-02 18:04 (:0.0)
pasztor  pts/3        2007-12-02 18:04 (:0.0)
pasztor@clyde:~$ w
 22:58:30 up  4:56,  5 users,  load average: 1,49, 1,06, 0,92
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
pasztor  :0       -                18:04   ?xdm?  13:32m  0.31s /usr/bin/gnome-session
pasztor  pts/0    :0.0             18:04   44:01m  1.52s  1.52s ssh linux
pasztor  pts/1    :0.0             18:04    0.00s  1.00s  0.02s w
pasztor  pts/2    :0.0             18:04    1:39   0.30s  0.30s bash
pasztor  pts/3    :0.0             18:04    2:19   0.01s  0.01s bash
pasztor@clyde:~$ finger
Login     Name             Tty      Idle  Login Time   Office     Office Phone
pasztor   PASZTOR Gyorgy  *:0             Dec  2 18:04
pasztor   PASZTOR Gyorgy   pts/0      44  Dec  2 18:04 (:0.0)
pasztor   PASZTOR Gyorgy   pts/1          Dec  2 18:04 (:0.0)
pasztor   PASZTOR Gyorgy  *pts/2    1:40  Dec  2 18:04 (:0.0)
pasztor   PASZTOR Gyorgy   pts/3    2:20  Dec  2 18:04 (:0.0)
```

```
pasztor@clyde:~$ finger pasztor
Login: pasztor                          Name: PASZTOR Gyorgy
Directory: /home/pasztor                Shell: /bin/bash
On since Sun Dec  2 18:04 (CET) on :0 (messages off)
On since Sun Dec  2 18:04 (CET) on pts/0 from :0.0
   45 minutes 19 seconds idle
On since Sun Dec  2 18:04 (CET) on pts/1 from :0.0
On since Sun Dec  2 18:04 (CET) on pts/2 from :0.0
   1 hour 41 minutes idle
     (messages off)
On since Sun Dec  2 18:04 (CET) on pts/3 from :0.0
   2 hours 20 minutes idle
New mail received Sun Dec  2 18:13 2007 (CET)
     Unread since Fri Mar 23 19:17 2007 (CET)
No Plan.
```

3. fejezet

A fájlrendszer

3.1. Merevlemezek felosztása, partíciók

A Unixban minden eszközt, hálózati kapcsolatot, másik helybéli programmal való (socket alapú) kommunikációs csatornát, úgyanolyan módon érnek el a programok mint egy fájlt. Vagyis meg lehet nyitni, le lehet zárni, lehet bele írni, lehet belőle olvasni. Az eszközökhöz kapcsolódó fájlok a `/dev` könyvtáron belül vannak.

3.1.1. A linux eszköznévelnevezési sémái

A Linux a merevlemezeket aszerint nevezi el, hogy milyen vezérlőn keresztül csatlakoznak. A hagyományos *(P)ATA* merevlemezeket a `/dev/hda`, `hdb`, `hdc`, ... nevekkel illeti. A PATA vezérlők egyik fontos jellemzője, hogy egy vezérlőre pontosan két eszközt (merevlemezt, vagy optikai meghajtót) lehet kötni: Egy master és egy slave egységet. Továbbá, attól függően, hogy hanyadik vezérlőre van kötve egy egység szokás Primary, Secondary, Tertiary, ... vezérlőknek hívni azokat a vezérlőket, amire az egységek kötve vannak. Így az eszközök nevei a következő séma szerint alakul:

/dev/hda Primary master

/dev/hdb Primary slave

/dev/hdc Secondary master

/dev/hdd Secondary slave

/dev/hde Tertiary master

... ...

Az ATA eszközöknél, függetlenül attól, hogy CD/DVD olvasó, vagy merevlemez van kötve az adott vezérlőre, mindig ez a nevezéktan.

A szerver-számítógépekben elterjedt SCSI vezérlők már 7 ill. 15 egység csatlakoztatására képesek egy-egy vezérlőre, valamint támogatják már a kezdetektől fogva az egységek menet közbeni hozzáadását és egyéb plusz tulajdonságokat, ezért ezen eszközök nevezéktana a következőképp alakult:

/dev/sda Elsőnek felismert SCSI merevlemez

/dev/sdb Másodiknak felismert SCSI merevlemez

/dev/sdc Harmadiknak felismert SCSI merevlemez

… …

/dev/scd0 Elsőnek felismert SCSI Optikai meghajtó (CD/DVD)

/dev/scd1 Másodiknak felismert SCSI Optikai meghajtó

… …

A *SATA* vezérlők és a *pendrive*-ok már a SCSI alrendszeren keresztül érhetőek el a linuxban, mert csak ez biztosít a rendszer felé olyan szolgáltatásokat, mint pl. egy merevlemez menet közbeni csatlakoztatása, vagy leválasztása.

3.1.2. Partícionálás

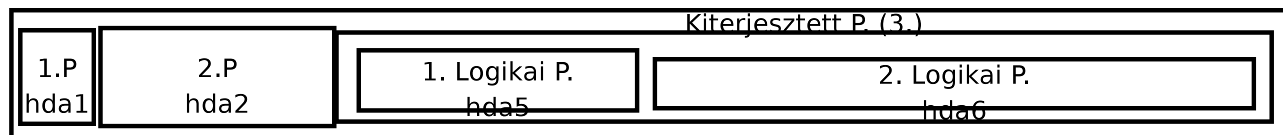
A merevlemezek teljes adatkapacitását általában nem egyben szokás felhasználni. Aki csak otthoni célra használja a gépét is legalább két partíciót létrehoz: Az egyiket az operációs rendszer programjainak és adatainak, a másikat ún. swap célra. Ha pedig rendszer biztonságát és megbízhatóságát[1] is növelni szeretnénk, akkor még érdemes lehet különválasztani a rendszer indításához szükséges fájlokat, az operációs rendszer központi részét, a felhasználói programokat, a rendszerfeladatok adatait, naplófájlokat és a felhasználók adatait.

Hogy ezeket az adatokat különválaszthassuk egy lemezen, ezért a lemezt ún. partíciókra lehet osztani. Erre szolgál a partíciós tábla, amely ahogyan a neve is mutatja, egy táblázat. A lemez elején az ún. MBR-ben található. A partíciós tábla ezen részében pontosan négy sor van, ide kerülhetnek olyan bejegyzések, amelyek egy-egy partícióról a következő információkat mondják el:

1.	típus	kezdete	vége
2.	típus	kezdete	vége
3.	típus	kezdete	vége
4.	típus	kezdete	vége

Fontos tény, hogy a partíciók nem tartalmazhatják a lemez egymást átfedő területeit! A partíciók típusa általában utal a partíció tartalmára, ill. annak formátumára. Ilyen típusok pl.: NTFS, Fat32, Linux, Linux swap, Linux raid, Linux lvm, …

Mivel azonban ez a négy sor a táblázatból sok esetben kevesebb, mint ahány részre szeretnénk osztani a lemezünket, ezért lehetőség van ún. kiterjesztett típusú partíciót létrehozni, amiben logikai partíciókat hozhatunk létre. Ilyenkor a négy ún. elsődleges partíció egyikét kiterjesztett típusúnak kell létrehozni[2], és ezen belül a legelső használható adatblokkba jegyzi be a partícionáló program, hogy milyen típusú partíció, és mettől meddig tart. Itt már nincs fix sorszám, hanem egy indirekt hivatkozással van jelezve, hogy a következő logikai típusú partícióra vonatkozó információk hol találhatók.



3.1. ábra. Egy példa partícionálásra

[1] NB.: Két külön fogalom a biztonság és a megbízhatóság!
[2] Csak egy ilyen lehet!

3.1.3. A partíciók eszköznevei

Miután az operációs rendszerben fel tudunk olvasni egész lemezeket, szükség van arra is, hogy „csak” egy-egy partícióra hivatkozzunk. A példa kedvéért legyen a felparticionált lemezünk neve `/dev/sda`, ekkor az első elsődleges partíció eszközneve `/dev/sda1`, a második elsődleges partíció neve `/dev/sda2` lesz..., valamint az első logikai partíció neve (függetlenül attól, hány érvényes elsődleges partíciónk van!) mindig `/dev/sda5`, a második logikai partíciónk neve `/dev/sda6` lesz...

3.2. Eszköz-fájlrendszer kapcsolat

3.2.1. fájlrendszer-típusok

Miután vannak eszközeink az adattárolásra, valamilyen módon rendszerezni kell az adatokat rajta. Erre szolgálnak a fájlrendszerek. Alapvetően a fájlrendszer arról szól, hogy ha van egy adatterületünk, akkor azon milyen formában tároljunk el információkat a fájljainkról, milyen információkat tároljunk el róluk, és hogyan tároljuk el azt az információt, hogy az adatterület mely részén (és milyen sorrendben) vannak a fájl részei.
Ilyen fájlrendszertípusok: fat, **ntfs**, ext2, **ext3**, **reiserfs**, **xfs**, **jfs**, **ufs**...
Az első kettő kivételével, mind alkalmas unix-típusú rendszer fájlrendszeréül szolgálni. Amivel a unix-típusú fájlrendszerek többet tudnak:

- a létrehozás, módosítás, és az utolsó hozzáférés dátumát is eltárolják
- képes speciális node-ok metaadatait tárolni: socket, named pipe, blokkos eszköz, karakteres eszköz
- lehet rajta szimbolikus linkeket létrehozni
- lehet rajta hardlinkeket létrehozni
- a fájlhoz tulajdonost, tulajdonos csoportot, és jogosultságokat eltárol
- képes kvótainformációkat hordozni

A fájlrendszerek egy másik kiemelkedő képessége az úgynevezett naplózás. Ez annyit jelent, hogy nem a lehető leggyorsabb / legoptimálisabbnak tűnő „durr-bele” módon tárolja el az adatokat és a változásokat az adatterületen, hanem egyfajta speciális adatbázisnak tekintve a fájlrendszert, tranzakciókban hajtja végre a műveleteket, amely tranzakciók vagy lezártak, vagy visszagörgethetőek. Ez a tulajdonság azért fontos, mert lehetővé teszi, hogy nagyon kevés művelettel, bármelyik pillanatban a fájlrendszert a le nem zárt tranzakciók visszagörgetésével konzisztenssé tegyünk, ami a gyakorlatban annyit jelent, hogy pl. egy áramszünet után a számítógépünk a bootolás során legroszabb esetben is néhány másodperc alatt konzisztens állapotba tud hozni akár több terabájtnyi információt tároló fájlrendszereket.

A különböző fájlrendszerek létrehozására általában az *mkfs*-el kezdődő nevű parancsok szolgálnak. Pl. xfs fájlrendszerűre formázni egy eszközt az **mkfs.xfs** paranccsal lehet. A fájlrendszerek ellenőrzésére pedig az *fsck*-val kezdődő nevű paracsok szolgálnak. Pl. egy xfs-fájlrendszert az **fsck.xfs** paranccsal lehet leelenőrizni.

3.2.2. Eszközök csatlakoztatása

Miután vannak valamilyen típusú fájlrendszerrel megformázott fájlrendszereink, feltehetően el is szeretnénk érni azokat. A mai modern rendszerekben, amikor egy-egy eszköz menetközbeni csatlakoztatása, mint pl. egy pendrive bedugása, beindít olyan automatizmusokat, amelyek ezt a műveletet végrehajtják, azonban mégsem árt tisztában lennünk a háttérben lezajló művelettel!

Fájlrendszer csatlakoztatására a mount, leválasztására pedig az umount parancs szolgál. Fontos tudni, hogy a Unix-típusú operációs rendszerek nem használnak, az alternatív rendszereknél megszokott, betűjeles meghajtókat, lévén az angol abc betűi is könnyen el tudnak fogyni, ha valaki sok különböző eszközön tárol adatot, és mindet egyszerre szeretné elérni. Az alapelv szerint egyetlen egy gyökérkönyvtár van, amelyet a `/`-el jelölünk. Ha a gyökérkönyvtártól kiindulva szeretnénk egy könyvtár nevére hivatkozni, akkor pedig a `/` jellel kezdjük a nevét leírni. Az eszközöket rendszerint valamelyik alkönyvtár alá[3] csatlakoztathatjuk fel. A használatának módja a következő:
mount *eszköz_neve könyvtár_neve*
umount *könyvtár_neve*

Példa, ha egy pendrive-ot bedugok, ahol a rajta lévő fájlrendszer a `/dev/sda1` nevű eszközön keresztül válik elérhetővé, akkor ezt a `/mnt/pendrive` nevű könyvtárba a
mount /dev/sda1 /mnt/pendrive paranccsal tudom csatlakoztatni, valamint később a
umount /mnt/pendrive paranccsal tudom leválasztani.

Ha a **mount** parancsnak nem adunk semilyen paramétert, akkor megmutatja a jelenleg csatlakoztatott eszközöket.

```
pasztor@voyager:~$ mount
/dev/mapper/voyager-ustab on / type xfs (rw)
proc on /proc type proc (rw,noexec,nosuid,nodev)
/sys on /sys type sysfs (rw,noexec,nosuid,nodev)
varrun on /var/run type tmpfs (rw,noexec,nosuid,nodev,mode=0755)
varlock on /var/lock type tmpfs (rw,noexec,nosuid,nodev,mode=1777)
udev on /dev type tmpfs (rw,mode=0755)
devshm on /dev/shm type tmpfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
lrm on /lib/modules/2.6.24-2-generic/volatile type tmpfs (rw)
/dev/sda1 on /boot type ext2 (rw)
/dev/mapper/voyager-home on /home type xfs (rw)
securityfs on /sys/kernel/security type securityfs (rw)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,noexec,nosuid,nodev)
/dev/scd0 on /media/cdrom0 type iso9660 (rw,nosuid,nodev,user=pasztor)
```

3.2.3. Lemezhasználtság

Egy fájlrendszer életében egy fontos információ, hogy mennyi adat fér rá, mennyi van rajta az adott pillanatban, és mennyi szabad. Ennek megmutatására a **df**[4] parancs használható. Ha nem adunk neki semilyen paramétert, akkor a választ rendszerint kilobájtokban adja meg. Ha egy könyvtárnevet is megadunk neki, akkor csak az ahhoz a lemezhez/eszközhöz tartozó használtsági információkat mutatja meg. Használatos még a **-h** kapcsoló is, amely a „human-readable" rövidítés alapján az egyes információkat olvasható méretűre kerekítve adja meg mér-

[3] gyakorlatilag bármelyik alá, de csak óvatosan, mert a Unix típusú rendszerek az eredeti filozófiájuk szerint a rendszergazda kezébe megadják a lehetőséget, hogy „önmagát lábonlőhesse"!

[4] neve a *disk free* szavakból jön

tékegységgel együtt. Vagyis néhány Megabájtos lemez/foglaltság esetén Megabájtban írja ki az információt, több/több száz Gigabájtos lemez esetén Gigabájtban...

Másik hasznos parancsunk a **du**[5], amely azt mondja meg, hogy egy könyvtárstruktúrán belül mennyi lemezterület van elfoglalva.

Példák:

```
pasztor@voyager:~$ df
Fájlrendszer           1K-blokk   Foglalt    Szabad Fo.% Csatl. pont
/dev/mapper/voyager-ustab
                        5232640   4477436    755204  86% /
varrun                  1015408       124   1015284   1% /var/run
varlock                 1015408         4   1015404   1% /var/lock
udev                    1015408        68   1015340   1% /dev
devshm                  1015408       112   1015296   1% /dev/shm
lrm                     1015408     28628    986780   3% /lib/modules/2.6.24-2-generic/vo
/dev/sda1                180639     39666    131336  24% /boot
/dev/mapper/voyager-home
                        2086912    216140   1870772  11% /home
/dev/scd0                712508    712508         0 100% /media/cdrom0
pasztor@voyager:~$ df -h
Fájlrendszer            Méret  Fogl. Szab. Fo.% Csatl. pont
/dev/mapper/voyager-ustab
                       5,0G  4,3G  738M  86% /
varrun                 992M  124K  992M   1% /var/run
varlock                992M  4,0K  992M   1% /var/lock
udev                   992M   68K  992M   1% /dev
devshm                 992M  112K  992M   1% /dev/shm
lrm                    992M   28M  964M   3% /lib/modules/2.6.24-2-generic/volatile
/dev/sda1              177M   39M  129M  24% /boot
/dev/mapper/voyager-home
                       2,0G  212M  1,8G  11% /home
/dev/scd0              696M  696M     0 100% /media/cdrom0


pasztor@voyager:~$ du
4 ./.gnome2/keyrings
28 ./.gnome2/accels
4 ./.gnome2/share/fonts
...
4 ./.vim/colors
8 ./.vim
209028 .
pasztor@voyager:~$ du -h
4,0K ./.gnome2/keyrings
28K ./.gnome2/accels
4,0K ./.gnome2/share/fonts
...
4,0K ./.vim/colors
8,0K ./.vim
205M .
```

[5]neve a *disk usage* szavakból jön

3.2.4. Az fstab

Az fstab, ahogyan arra a neve is utal, egy táblázat. Az elsődleges célja, hogy leírja, hogy mely eszközöket mely könyvtárak alá kell bemountolni a bootfolyamat során.

Egy példa fstab:

```
proc                    /proc           proc            defaults        0 0
/dev/mapper/io-root     /               xfs             defaults        0 1
/dev/hda1               /boot           ext2            defaults        0 2
/dev/mapper/io-home     /home           xfs             usrquota        0 2
/dev/scd0               /media/cdrom0   udf,iso9660 user,noauto     0 0
/dev/scd1               /media/cdrom1   udf,iso9660 user,noauto     0 0
```

Tartalmazhat olyan opciókat is egy-egy sor, amely arra utal, hogy a boot folyamat során ne kerüljön automatikusan mountolásra egy eszköz. Ez hasznos lehet olyankor, amikor valamilyen eszközt sűrűn kell csatlakoztatni, esetleg sok különféle opcióval, mert ha szerepel hozzá való bejegyzés az fstab-ban, akkor a **mount** parancsnak elég csak a célkönyvtárat megadni, minden egyéb paramétert/opciót innen vesz.

Az fstab oszlopai:

eszköznév Annak az eszköznek a neve, amit fel kell csatolni.

könyvtárnév Annak a könyvtárnak a neve, ahova az eszközt fel kell csatlakoztatni.

fájlrendszer Milyen típusú fájlrendszert kell/lehet feltételezni az eszközön.

opciók Milyen beállításokkal kell az eszközt csatlakoztatni. Ill. kiegészítő opció a *noauto*, amely jelzi, hogy a boot folyamat során nem kell csatlakoztatni az eszközt, valamint a *user*, amely jelzi, hogy felhasználó is végrehajthatja a csatlakoztatást.

dumpszám Régebbi típusú backup-eszközöknél volt használatos.

passno Ez a sorszám adja meg, hogy a bootfolyamat során, hanyadik körben kell a fájlrendszer épségét ellenőrizni. A **0**-t olyan speciális esetekre szokás használni, amit nem kell ellenőrizni, az **1**et tipikusan a / fájlrendszert tartalmazó eszközre szokás megadni, hogy ez minden egyéb előtt legyen ellenőrizve, és általában minden más eszközt a **2**. körben szokás ellenőrizni.

3.3. Fájlrendszer

3.3.1. i-node tábla

Egy unixos fájlrendszer legfontosabb eleme az i-node táblázat. Tételezzünk fel, egy ilyesmi táblázatot a példa kedvéért:

i-node	t	uid	gid	jogok	ln	ATime	CTime	MTime	poz.
414	d	1000	1000	0755	2	2008-01-09 22:28	01-07 22:20	2008-01-07 22:20	...
420	d	1000	1000	0755	2	2008-01-09 09:57	01-08 20:07	2007-12-22 12:11	...
2468852	d	1000	1000	0755	2	2008-01-09 22:17	01-08 20:07	2007-12-25 16:31	...
3145981	d	1000	1000	0755	2	2008-01-09 22:09	01-08 20:07	2007-12-22 12:11	...
4235587	d	1000	1000	0755	2	2008-01-09 09:57	01-08 20:07	2007-12-22 12:11	...
5252700	d	1000	1000	0755	7	2008-01-09 22:09	01-09 22:09	2008-01-09 22:09	...
5259712	d	1000	1000	0755	2	2008-01-09 22:27	01-09 22:27	2008-01-09 22:27	...
6292152	d	1000	1000	0755	2	2008-01-09 09:57	01-08 20:07	2007-12-22 12:11	...
7340894	d	1000	1000	0755	2	2008-01-09 09:57	01-08 20:07	2007-12-22 12:11	...
...	...	...	...	...	...	...	...	...	...

A táblázatnak vannak speciális oszlopai, amelyek egyéb, nem reguláris fájlok/könyvtárak esetén érdekesek. Ezeket nincsenek most a példabeli táblázatunkban.

A táblázat oszlopainak jelentése:

i-node Az i-node sorszáma

típus Az i-node típusa: fájl, könyvtár, symlink, blokkos/karakteres eszköz, pipe, socket.

uid A fájl/könyvtár/stb. tulajdonosának uidje.

gid A fájl/könyvtár/stb. tulajdonos csoportjának id-je.

jogok A fájlhoz/könyvtárhoz/stb.hez való hozzáférési jogosultság

ln Hány hardlink van a fájlra/könyvtárra/stb.

ATime Az utolsó hozzáférés időpontja. Pl. megnyitás olvasásra, írásra, ha könyvtár, akkor belelépés időpontja, stb.

CTime A létrehozás időpontja.

MTime Az utolsó módosítás időpontja.

poz A fájl/könyvtár adatainak a fájlrendszer adatterületén való elhelyezkedését megadó információk. Ennek segítségével állapítható meg egy fájl mérete is[6].

Ahogyan azt a unix rendszerektől megszokhattuk általában a saját kezelésére valamilyen numerikus azonosítóval (*id-vel*) azonosítja az objektumokat – legyen az felhasználó, csoport, vagy épp fájl – de a felhasználó felé, mindíg kínál lehetőséget, hogy névvel hivatkozzunk rá. Nincs ez másképp a fájloknál sem. Ahogyan általában az operációs rendszereknél megszoktuk, a fájlok könyvtárakba szervezhetőek, és minden fájlrendszeren található egy gyökérkönyvtár. Fájlrendszertípusonként eltér a konkrét szám, de annyi közös a Unix-típusú fájlrendszerekben, hogy ezt mindíg egy speciális sorszámú i-node-hoz köti. A könyvtárak, így gyakorlatilag speciális fájlokká avanzsálnak, amelyek egy az alábbihoz hasonló egyszerű táblázat információit

[6]Érdekesség a Unix-típusú fájlrendszereknél, hogy a fájl mérete nem feltétlenül egyezik meg a lemezről elfoglalt adatterület mennyiségével. Lásd.: sparse-fájl

tárolják:

i-node	név
414	Examples
420	Videók
2468852	Munkaasztal
3145981	Sablonok
4235587	Nyilvános
5252700	Dokumentumok
5259712	TeX
6292152	Zene
7340894	Képek
...	...

3.3.2. Fájlkezelő parancsok

Az alapvető parancsok a fájlrendszerben közlekedéshez / információszerzéshez:

cd Könyvtárváltás parancsa. Ha paraméter nélkül adjuk ki a home-könyvtárunkba lép. Ha adunk neki 1 db. paramétert, akkor megpróbál belépni abba a könyvtárba. Nevét a **C**hange **D**irectory szavak után kapta.

pwd Kiírja, hogy aktuálisan melyik könyvtárban vagyunk. Nevét a **P**rint **W**orking **D**irectory szavak után kapta.

mkdir Könyvtár létrehozása. Nevét a **mak**e **dir**ectory szavak után kapta.

rmdir Könyvtár törlése. Nevét a **rem**ove **dir**ectory szavak után kapta.

ls Kilistázza az adott könyvtárban levő bejegyzéseket. Részleteket lásd később. Nevét a **lis**t szó után kapta.

cat Kiírja a terminálra/standard kimenetére a paramterül megadott fájl tartalmát. Ha nem kap paramétert, akkor a standard bemenetét írja vissza. Nevét a **cat**alog szó után kapta.

cp Fájl másolása. Nevét a **c**o**p**y szó után kapta.

mv Fájl mozgatása/átnevezése. Nevét a **mov**e szó után kapta.

ln Link készítése. Nevét a **lin**k szó után kapta.

rm Fájl törlése. Nevét a **rem**ove szó után kapta.

touch Fájl vagy könyvtár utolsó módosítási dátumának átállítása az aktuális időpontra. Ha nem létezett korábban, akkor létrehoz egy a paraméterben megadott nevű üres – *0 hosszú* – fájlt.

find Fájlok keresése megadott feltételek alapján.

Az ls

Az ls parancs számos kapcsolóval rendelkezik. Amikor terminálról használjuk, nagyjából így néz ki az általa adott lista:

```
pasztor@voyager:~$ ls
Dokumentumok  Képek        Nyilvános  TeX     Zene
Examples      Munkaasztal  Sablonok   Videók
```

Ha azt szeretnénk, hogy egy sorban, csak egy név szerepeljen, akkor a **-1** kapcsolót használhatjuk. Előfordulhat, hogy valamilyen oknál fogva kíváncsiak vagyunk a fájlokhoz kapcsolt i-node számokra is. Ekkor a **-i** kapcsoló lehet a segítségünkre:

```
pasztor@voyager:~$ ls -i
5252700 Dokumentumok  2468852 Munkaasztal  5259712 TeX
    414 Examples      4235587 Nyilvános        420 Videók
7340894 Képek         3145981 Sablonok     6292152 Zene
```

Azonban mégis a leggyakoribb kapcsolója, amit használni szoktunk, a **-l** kapcsoló, amely részletes listát ad a fájlokról:

```
pasztor@voyager:~$ ls -l
összesen 40
drwxr-xr-x 7 pasztor pasztor 4096 2008-01-09 22:09 Dokumentumok
lrwxrwxrwx 1 pasztor pasztor   26 2008-01-07 22:20 Examples -> /usr/share/example-conten
drwxr-xr-x 2 pasztor pasztor    6 2007-12-22 12:11 Képek
drwxr-xr-x 2 pasztor pasztor   23 2007-12-25 16:31 Munkaasztal
drwxr-xr-x 2 pasztor pasztor    6 2007-12-22 12:11 Nyilvános
drwxr-xr-x 2 pasztor pasztor    6 2007-12-22 12:11 Sablonok
drwxr-xr-x 2 pasztor pasztor 4096 2008-01-10 00:25 TeX
drwxr-xr-x 2 pasztor pasztor    6 2007-12-22 12:11 Videók
drwxr-xr-x 2 pasztor pasztor    6 2007-12-22 12:11 Zene
```

A részletes lista oszlopai sorban:

drwxr-xr-x A bejegyzés típusa: -:Reguláris fájl, **d**:Könyvtár, **b**:Blokkos eszköz, **c**:Karakteres eszköz, **l**:symlink, **s**:Socket, **p**:Pipe
Valamint a bejegyzésre vonatkozó jogosultságok: **R**ead, **W**rite, e**x**ecute.
Az **x** helyén ez első két oszlopban állhat **s**, illetve a harmadik oszlopban **t**, amely a setuid, setgid, és sticky jogosultságokat jelzi.

2 Az adott bejegyzésre mutató linkek száma.

pasztor A tulajdonos felhasználó

pasztor A tulajdonos csoport

4096 Méret

01-10 00:25 Utolsó módosítás időpontja

TeX A bejegyzés neve

Használatos még a **-n** kapcsoló, amely csak a **-l**-el együtt értelmes. Ilyenkor a listázásnál az uid és gid-et nem helyettesíti be a felhasználó/csoport nevével, hanem az eredeti numerikus értéket írja ki.

A **-r** kapcsoló rekurzív módban listáztat. Vagyis sorban bejárja az alkönyvtárakat is, és azokat is kilistázza.

A **-a** és **-A** kapcsoló a rejtett fájlok és könyvtárak kilistázására szolgál. A unixban a „rejtett” tulajdonság nem egy fájlattributum. Akkor lesz egy fájl/könyvtár rejtett, ha a neve `.`-al kezdődik. Minden könyvtárban található egy `.` és egy `..` nevű könyvtárra való hivatkozás. A **-A** használatakor ez a kettő nem listázódik ki. A `.` az aktuális könyvtárat, magát hivatott szimbolizálni, a `..` pedig a szülőkönyvtárat. Ha léterhozunk egy könyvtárat, a linkszáma

rögtön kettő lesz, hiszen a szülőkönyvtár, amiben létrehoztuk tartalmazza az i-nodejára az egyik hivatkozást, a másikat pedig önmaga a . nevű bejegyzésével. A könyvtárak másik ebből levezethető tulajdonsága, hogy ha pl. 5 alkönyvtára van, akkor a linkszáma 7 lesz. Az első kettő ugye már adott (a szülőkönyvtárból mutató bejegyzés és a saját . nevű bejegyzésünk), a többi ötöt pedig az alkönyvtárakban szereplő .. nevű bejegyzések adják.

A cp

Lehetésges használati módjai:

- Két paraméter: egy forrás és egy célfájl. Vagyis, hogy milyen nevű fájlt másoljon, és a másolatnak mi legyen a neve.
- több paraméter: egy vagy több forrásfájl és egy célkönyvtár. A forrásfájlok mindegyikét megadja, az utolsó paraméterként megadott nevű könyvtárba. Ebben az esetben az utolsó paraméternek, mindig egy már létező könyvtárnak kell lennie.

A **cp** parancs kapcsolói közül a **-r** szintén a rekurziót kapcsolja be. Ebben az esetben az a hatása, hogy ne csak egy fájlt másoljon át, hanem egy teljes könyvtárstruktúrát.

A másik hasznos kapcsoló a **-a**, amely az archiváló módot kapcsolja be. Ilyenkor nem csak arra képes, hogy rekurzívan másoljon át egy könyvtársturktúrát, de megőrzi a fájlok tulajdonosát is[7], valamint a hozzátartozó időbélyegeket (atime,ctime,mtime) is.

Az mv

Fájl vagy könyvtár átnevezésére, vagy másik könyvtárba való átmozgatásra szolgál. Ha ugyanazon a fájlrendszeren belül tesszük, akkor egy egész könyvtárat, minden benne lévő bejegyzéssel képes átmozgatni, hiszen ilyenkor csak a forráskönyvtárban kell kitörölni az adott i-nodera vonatkozó utalást és a célkönyvtárban pedig létrehozni. Másik fájlrendszerre nem lehet átmozgatni ilyen egyszerűen egy könyvtárstruktúrát, hiszen minden fájlrendszerhez külön-külön i-node táblázat tartozik.

Az ln

Az **ln** parancs link létrehozására szolgál. Ha a **cp**hez hasonlóan egy forrás és egy célfájlnevet adunk meg neki, akkor egy hardlinket hoz létre. A hardlink annyit jelent, hogy ugyanarra az i-nodera több különböző könyvtárbejegyzés is vonatkozik. Ezeknek a számát mutatja az **ls** parancsnál a linkszám oszlop a részletes listázásnál. Abból következően, hogy az i-node táblázat fájlrendszerenként adott, ezért hardlinket is, csak fájlrendszeren belül tudunk létrehozni.

Ha a **-s** kapcsolót is használjuk, akkor nem hard-, hanem szimbolikus linket hoz létre. Ilyenkor egy újabb i-node jön létre, de az operációs rendszer nyilvántartja, hogy amikor az adott fájlt kellene megnyitnia, akkor az adott területen egy célfájl neve van, és ehelyett azt a fájlt nyitja meg. Ez a szimbolikus link.

Az rm

Az rm parancs fájl törlésére szolgál. Szintén rendelkezik **-r** kapcsolóval amely egy teljes könyvtárstruktúra rekurzív törlésére szolgál. Valamint szokás még a **-f**[8] kapcsolót is használni, amellyel a törlésre való kérést „nyomatékosíthatjuk”.

[7] ezt csak rendszergazdaként kiadva tudja megtenni
[8] force

A find

A find parancs egy megadott könyvtárban, megadott feltételeknek megfelelő fájlok megkeresésére szolgál. A megtalált fájlokkal több művelet is végezhető: Kiiratható a nevük, vagy parancs futtatható az adott fájlnevet paraméterül adva neki. Az első paramétere opcionálisan egy könyvtárnév. Ha nem adunk meg külön könyvtárnevet, akkor az aktuális könyvtárban kezdi a feltételeknek megfelelő fájlokat keresni. A keresés az alkönyvtárakban mindíg rekurzív módon történik.

Lehetséges feltételek:

-mindepth A paraméterül megadott mélységben és mélyebben keres

-maxdepth A megadott paraméternél mélyebben nem keres.

-type A megadott típusú bejegyzéseket keres csak: **f**: reguláris fájl, **d**: könyvtár, **l**: szimbolikus link, **b**: blokkos eszköz, **c**: karakteres eszköz, **s**: socket, **p**: pipe

-mtime legalább a paraméterben megadott napja lett utoljára módosítva a fájl, ha egy + jelet teszünk a napok száma elé, ill. ha egy **-**t, akkor legfeljebb. Vagyis kevesebb mint n napja lett módosítva a fájl.

-xdev Nem megy olyan alkönyvtárakba, ahova már másik fájlrendszer van felmountolva.

-uid A paraméterben uid-del megadott tulajodonosú fájlokat keresi

-gid A paraméterben gid-del megadott tulajodonos-csoportú fájlokat keresi

-name A paraméterben megadott mintára illeszkedő nevű fájlokat keresi

-iname A paraméterben megadott mintára illeszkedő nevű fájlokat, **case insensitive** módon keresi

-perm A fájl jogosultságaira vonatkozó feltételeket adhatunk meg

-size A fájl méretére vonatkozó feltételt adhatunk meg. Pl. a +5M arra szolgál, hogy az 5 megabájtnál nagyobb fájlokat keresse csak.

3.3.3. Speciális eszközök

A fájlrendszerben már ismerjük a fájl, könyvtár és szimbolikus linkek mibenlétét. Azonban van még néhány speciális eszköz amelyet megtalálhatunk a rendszerünkön. Ezen speciális eszközök a „named pipe", a „socket", a karakteres eszköz valamint a blokkos eszköz.

Named Pipeok

Ezek arra jók, hogy ha különböző időpontokban indítunk el parancsokat, amelyek kimenetét és bemenetét szeretnénk összekapcsolni. Ahogyan a parancsértelmezőben erre látni fogunk példát erre tipikusan a **parancs1** | **parancs2** módon kiadott parancsokat szoktuk használni. De van amikor a parancs2-t már korábban elindítottuk, és korábban elkezdené várni az inputját, vagy a parancs1-et indítottuk jóval korábban, és ő már „gyártja" az outputját. Ezesetben ha egy ilyen *named pipe*ba irányíthatjuk a **parancs1** kimenetét, és később innen vehetjük a **parancs2** bemenetét. Az **ls** parancs részletes kimenetében ezek a named pipe-ok **p** betűvel kezdődnek. Egy ilyen példa a `/dev/initctl` amelyen keresztül az init parancs várja a rendszer futása közben az olyan parancsokat, amelyek őt vezérlik.

```
pasztor@voyager:~$ ls -l /dev/initctl
prw------- 1 root root 0 2008-01-08 20:00 /dev/initctl
```

Socketek

A socketek a named pipeokkal szemben kétirányú kapcsolatot tesznek lehetővé programok között. Ahogyan azt a hálózatoknál/TCP-nél megszokhattuk a socket egy kapcsolódási pont egy gépen futó programhoz. Míg tcp-nél a gép ipcíme és a tcp portszáma azonosította be, hogy mely socketről is beszélünk, addig az ún. „Unix Socketek"nél a kommunikáció, mindenképpen a gépen belül zajlik, a port helyett pedig a fájlnév azonosítja a socketet. Tipikus alkalmazások amelyek nem csak TCP-n, de unix socketen is elérhetők:

- X11-szerver[9]
- PostgreSQL szerver
- MySQL szerver
- ...

A socketek az **ls** parancs részletes kimenetében arról vehetők észre, hogy a sor `s` betűvel kezdődik.

Karakteres eszközök

A „szekvenciális" hozzáférésű/karakteres eszközökhöz nyújtanak hozzáférést ezek a speciális i-node bejegyzések. Ilyen speciális eszközök lehetnek akár a terminál, egy hangkártya, egér vagy bármi más, ahonnan bájtok/karakterek sorozatát olvassuk be. Az eszközöket a unix rendszerek két számmal azonosítják be: Egy major és egy minor eszközszám mondja meg az operációs rendszer számára, hogy mely eszközről van szó. Az i-node táblában ez a számpár van eltárolva.

Fájlnév	Célja
/dev/null	Bármit írunk bele „elveszik"[10].
/dev/full	Bármit próbálunk beleírni, olyan hibaüzenetet ad vissza, hogy az eszköz megtelt.
/dev/zero	Végtelensok 0 ascii kódú karaktert (0 értékű bájtot) tudunk belőle kiolvasni[11].
/dev/stdin	A standard bemenetünk fájlnévvel megadva
/dev/stdout	A standard kimenetünk fájlnévvel megadva
/dev/stderr	A standard hibacsatornánk fájlnévvel megadva
/dev/random	Véletlen bájtok sorozata olvasható belőle

Blokkos eszközök

A blokkos eszközök olyan fizikai eszközökhöz nyújtanak hozzáférést, ahol „blokkosan" vannak szervezve az adatok. Pl. egy merevlemez, vagy egy partíció, ahol szektoronként vagyis blokkonként tudjuk elérni az adatokat. További jellemzője, hogy fix mérete van egy-egy ilyen eszköznek, és az eszköz tetszőleges pontjáról tudunk adatokat kérni annélkül, hogy az előtte levő adatokat is be kellene olvassuk.

Erre példa pl. a már tárgyalt lemezek, partíciók és optikai lemezek eszközei: `/dev/hda`, `/dev/hdb`, ..., `/dev/sda`, `/dev/sdb`, ..., `/dev/scd0`, ...

3.3.4. Jogosultságok

A Unixban a jogosultságok 3 részre tagolódnak: A felhasználó jogosultságai, a csoport jogosultságai és mindenki más jogosultságai. Innen jön a fájlokra kiosztható jogosultsághármasnál az `rwxrwxrwx` tagolódása. A kiosztható jogosultságok pedig:

[9] Ez az a szolgáltatás, amely a VGA kártyákat vezérli, és a grafikus programok monitoron való megjelenítéséért felel

r Read

w Write

x eXecute

A jogosultságok kettes számrendszerbeli számokra is leképezhetők, mivel egy jogosultságot az adott obejktum vagy megkap, vagy nem kap meg. Így pl. a fájl tulajdonosának ha van `rw` joga, akkor az pl. leírható így is: *110*. Vegyük észre, hogy ha a jogosultsághármasokkal, ha megtartjuk ezt a tagolódást, akkor a számok a **000**-tól a **111**-ig terjednek, aholis a kettes számrendszerbeli **111** az a 7-es számot jelenti a „hétköznapi”[12] számolásaink szerint. Azt is megfigyelhetjük, hogy ha ezek az egy egy objektumra[13] vonatkozó jogosultságok egy ilyen `0`-tól 7-ig terjedő számok, azt a hétköznapokban 8-as számrendszerbeli számoknak hívjuk. Vagyis egy fájl jogosultsága 3 db. 8-as számrendszerbeli számjeggyel fejezhető ki. Pl. Ha a részletes listázásban így néz ki egy fájl jogosultsága: `rwxr-x--`, ami annyit jelent, hogy a fájl tulajdonosa tudja írni, olvasni, és futtatni is, de a tulajdonos csoport már nem tudja módosítani, aki pedig se nem tulajdonosa és a tulajdonoscsoportban sincs benne, az semmit nem tud vele tenni. Ez az jogosultság kifejezhető a `750` értékkel is.

Könyvtárak esetén a „végrehajtási jogosultság” azt jelenti, hogy beléphetünk az adott könyvtárba. A könyvtár részletes kilistázásához is kevés az olvasási jog. Ha egy könyvtárra csak olvasási jogunk van, akkor a benne lévő bejegyzések nevét tudjuk csak kideríteni, de azokról részletes információt már nem kapunk. Ha pedig csak végrehajtási jogosultságunk van, akkor beléphetünk, ott fájlokat megnyithatunk, de tudnunk kell a fájl nevét amit meg akarunk nyitni, mert kilistázni egyáltalán nem fogjuk tudni a könyvtárat. Lásd a példát:

```
pasztor@voyager:~$ mkdir teszt
pasztor@voyager:~$ touch teszt/tesztfile
pasztor@voyager:~$ chmod 600 teszt
pasztor@voyager:~$ ls -ld teszt
drw------- 2 pasztor pasztor 22 2008-01-20 17:44 teszt
pasztor@voyager:~$ ls -la teszt/
összesen 0
?--------- ? ? ? ?                ? teszt/.
?--------- ? ? ? ?                ? teszt/..
?--------- ? ? ? ?                ? teszt/tesztfile
pasztor@voyager:~$ cd teszt
bash: cd: teszt: Hozzáférés megtagadva
pasztor@voyager:~$ chmod 300 teszt
pasztor@voyager:~$ ls -ld teszt
d-wx------ 2 pasztor pasztor 22 2008-01-20 17:44 teszt
pasztor@voyager:~$ ls -la teszt
ls: teszt: Hozzáférés megtagadva
pasztor@voyager:~$ cd teszt
pasztor@voyager:~/teszt$ ls -la
ls: .: Hozzáférés megtagadva
```

speciális jogosultságok

A unix rendszerekben merültek fel problémák, amelyek megoldására speciális jogosultságokat vezettek be. Egy probléma például, hogy ha valaki meg akarja változtatni a jelszavát, akkor

[12] a hétköznapokban a 10-es számrendszerrel számolunk

[13] obejktum alatt a felhasználót, a csoportot, meg a „mindenki más” értendő ebben a környezetben

a passwd és shadow fájlt nem tudja felülírni, sőtt ez utóbbit olvasni sem. A megoldás az, hogy a rendszeren a **passwd** nevű parancs a futása közben a fájl tulajdonosának, vagyis a rendszergazdának a jogosultságaival bír. Így teljesen biztonságos a rendszer, hiszen a nem-rendszergazda felhasználók csak futtatni tudják a parancsot, módosítani nem, így a működésének az olyan „mellékhatásait” sem tudják kiküszöbölni, hogy az eredeti jelszavuk ellenőrzése nélkül változtassa meg a jelszavukat, vagy hogy ne tudja egy ember más jelszavát megváltoztatni. Ugyanígy működik a **mount** parancs is. A parancsot tartalmazó fájl szintén a *root* birtokában van. A rendszergazda így bármit fel tud mountolni, de ha egyéb felhasználó próbál ilyet tenni, akkor előbb leelenőrzi az fstab-ot, hogy az adott csatolásra van-e engedélye a **user** opcióval, ezekután pedig elvégzi a csatlakoztatást, hiszen a futása ideje alatt, és csakis addig, rendszergazdai jogosultságokkal rendelkezik a folyamat, de csakis az az egy adott folyamat[14]. Ennek a végrehajtási jogosultságot módosító jogosultságnak a neve *setuid*.

Ugyanennek a problémának a csoportokra meglévő alternatívája, amikor egy adott csoport jogosultságait szeretnénk a parancsot futtató felhasználó számára lehetővé tenni a parancs futásának idejére. Tipikus példa ennek használatára a játékok: Az eredménytábla egy olyan fájlban van, amelyet csak a games csoport tud módosítani. Így amikor megnézzük 1-1 a unix rendszerünkön levő játék eredménytábláját, akkor nem tudja senki beírni magát oda annélkül, hogy ne valósan az adott játékban elért helyezéssel vívta volna ki a pozícióját. Ennek a jogosultságnak a neve *setgid*.

Ugyanezt a setgid jogosultságot könyvtárakra alkalmazva azt az eredményt érhetjük el, hogy a könyvtárba ha fájlt hoz létre valaki, akkor a fájl tulajdonos csoportja nem a fájlt létrehozó felhasználó elsődleges csoportja lesz, hanem a könyvtár tulajdonoscsoportja.

```
pasztor@voyager:~$ mkdir tesztk1
pasztor@voyager:~$ mkdir tesztk2
pasztor@voyager:~$ chgrp admin tesztk2
pasztor@voyager:~$ chmod g+ws tesztk2
pasztor@voyager:~$ touch tesztk1/tesztfile
pasztor@voyager:~$ touch tesztk2/tesztfile
pasztor@voyager:~$ ls -l tesztk*
tesztk1:
összesen 0
-rw-r--r-- 1 pasztor pasztor 0 2008-01-20 19:46 tesztfile

tesztk2:
összesen 0
-rw-r--r-- 1 pasztor admin 0 2008-01-20 19:47 tesztfile
```

A unix rendszerekben az ideiglenes fájlok a /tmp könyvtárba kerülnek függetlenül attól, hogy mely felhasználó ideiglenes fájljai. Ahhoz, hogy bárki idetehesse a fájlját, és törölhesse is innen, minden jogosultságot meg kell adni a könyvtárra, mindeki számára. A megoldás erre az, hogy ebben a könyvtárban mindenki csak a saját fájljai fölött rendelkezzen befolyással, egy új jogosultságot vezettek be a könyvtárakra. Ennek a jogosultságnak a neve *sticky-bit*.

A kiegészítő jogosultságokat ha a setuid, setgid, sticky-bit szerint sorbatesszük, akkor ez is egy jogosultsághármast ad, vagyis ezeknek a jelemzői is kifejezhetők egy 8-as számrendszerbeli számmal. Pl. ha *setuid*es valami, akkor a 4-essel, ha *setgides*, akkor a 2-essel, ha pedig *sticky-bit*es, akkor a 1-essel. Megegyezés szerint ha egy fájlra, vagy könyvtárra vonatkozik speciális jogosultság, akkor azt a többi jogosultság elé szokás írni. Vagyis a /tmp könyvtár jogosultságait pl. az 1777 szám írja le, még a **/usr/bin/passwd** fájlban levő parancsét a 4755.

[14]tehát a felhasználó nem lesz ettől átmeneti időre rendszergazda, és nem fog tudni rendszergazdai teendőket végezni pl. egy másik terminálon

```
pasztor@voyager:~$ ls -ld /tmp
drwxrwxrwt 14 root root 4096 2008-01-20 19:34 /tmp
```

Jogosultságkezelő parancsok

A **chgrp** paranccsal megváltoztathatjuk egy fájl vagy könyvtár tulajdonoscsoportját. Használata a következő:
chgrp *csoportnév fájl/könyvtárnév/nevek*
Ha a **-R** kapcsolót is használjuk miközben könyvtárnevet adunk paraméterül, akkor a működése „rekurzív" lesz, vagyis nem csak a könyvtár tulajdonoscsoportját változtatja meg hanem a benne lévő fájlokét, alkönyvtárakét, az alkönyvtáraiban levőkét, sít.

A **chown** paranccsal megváltoztathatjuk egy fájl vagy könyvtár tulajdonosát. Használata a következő:
chown *felhasználónév fájl/könyvtárnév/nevek*
A **chgrp**-hez hasonló módon itt is használhatjuk a **-R** kapcsolót. A **chown** parancsot használhatjuk a következő formában is:
chown *felhasználónév.csoportnév fájl/könyvtárnév/nevek*
Ilyenkor egyszerre tudjuk a tulajdonos felhasználót, és a tulajdonoscsoportot átállítani.

A fájlok jogosultságát a chmod paranccsal tudjuk állítani. A **-R** kapcsoló hatása itt is a megszokott. Használatának lehetséges módjai: **chmod** *szám fájl/könyvtárnév/nevek*
chmod *jogváltozás fájl/könyvtárnév/nevek*
Itt a *szám* a jogosultságoknál már tárgyalt módon számmal leírt jogosultságot takarja. A *jogváltozás* egy másik lehetséges módja a parancs használatának. Ilyenkor nem azt írjuk le, hogy egy-egy fájl/könyvtár jogosultságát mire állítsa be, hanem azt, hogy hogyan változtassa meg. Ez akkor lehet érdekes, ha van egy könyvtárban pl. sok fájl, de az „egyéb" felhasználói körtől[15] szeretnénk a jogosultságokat elvenni. Ez pl. a **chmod -R o-rwx .** paranccsal leírható, ahol az **o** azt *other* szó rövidítése, az **rwx** pedig az elveendő jogosultságok rövidítése, a **-** pedig jelzi hogy elvenni, és nem hozzáadni kell a jogosultságot, **.** pedig az aktuális könyvtár nevét szimbolizálja.

Az ilyen esetben használható rövidítések:

- a (all)mindenki
- u (user)a tulajdonos felhasználó
- g (group)a tulajdonos csoport
- o (other)mindenki más

A jogosultságokra használható rövidítések:

- r read
- w write
- x execute
- s setuid/setgid[16]
- t sticky bit

[15]vagyis aki se nem a tulajdonosa, és a tulajdonoscsoportnak sem tagja
[16]attól függ a jelentése, hogy csoport vagy tulajdonos jogosultságát kell változtatni

3.4. A Unix könyvtárai

Egy átlagos Unix rendszeren a megfelelő könyvtárban mindig hasonló célokra való fájlokat találunk, függetlenül attól, hogy épp egy Linux, valamilyen BSD, Solaris, vagy más egyéb Unix rendszerrel van dolgunk.

A Unix-típusú rendszereknél ezzel kapcsolatban már el is indult egy szabványosítási folyamat, melynek az eredményei a *Filesystem Hieararchy Standard* nevű dokumentumban[17] található meg, amely jelen pillanatban a *2.3*-as verziószámnál tart.

3.4.1. /boot

A `/boot` könyvtárban található minden unix rendszeren az operációs rendszer kernele, a hozzá tartozó initrd, ...

3.4.2. /bin, /sbin, /lib

A `/bin` és a `/sbin` könyvtárban találhatók a fontosabb parancsok, amelyek a rendszer fontos részét képezik. Az hogy egy parancs a `/bin` vagy a `/sbin` könyvtárba kerül, attól függ, hogy általános célú parancs, vagy kifejezetten csak a rendszergazda használja. A **mount**-olást pl. bárki elvégezheti a megfelelő `/etc/fstab`-beli engedélyekkel ezért az a `/bin` könyvtárba kerül, de a hálózati interfészeket csak a rendszergazda konfigurálhatja, így a **ifconfig** parancs már a `/sbin` könyvtárba fog kerülni.

Mivel minden parancs sok olyan dolgot használ, ami általánosabb művelet, mint pl. egy fájl megnyitása, vagy egy a **gzip** parancsban használt algoritmus szerinti tömörített tartalom ki- és betömörítése, ezért az ilyen programkódokat ún. *shared library*kba, vagy magyarul *osztott (függvény)könyvtárakba* szokás tenni. Az itt levő fájloknak tipikusan `.so` a kiterjesztése a *Shared Object*-re utalandó[18]. Ezen osztottkönyvtáraknak a helye egy Unix rendszerben a `/lib` könyvtár[19].

3.4.3. /etc

A rendszer és rendszerszolgáltatások konfigurációs állományai, adminisztratív fájlja mint pl. `fstab`, `passwd`, `shadow`, `group`, `gshadow`,... találhatók itt.

3.4.4. /tmp

A felhasználók ide tehetik az átmeneti állományaikat. Mivel ez a hely szigorúan átmeneti állományok tárolására szolgál, számíthatunk arra, hogy a rendszer újraindulásakor az ide elhelyezett állományok törlődnek. Pont ezért némelyik linuxdisztribúció a `/tmp`-t nem a lemezen, hanem a memóriában tartja, hogy az ideiglenes állományok tartalmához minél gyorsabban férjenek hozzá a programok.

3.4.5. /home, /root

A felhasználók home könyvtárai a `/home` könyvtár alá kerülnek be. Általában a `/home/`*loginnév* formában szokták a rendszereken a felhasználói home könyvtárakat kiosztani, de vannak rendszerek, ahol többszáz/többezer felhasználónak van home könyvtára. Ilyen esetekben előfor-

---

[17] http://www.pathname.com/fhs/

[18] alternatív rendszerekben ugyanezen célra szolgáló fájloknak `.dll` a kiterjesztése

[19] A `/lib/modules` alá kerül be minden menetközben betölthető kernelmodul, innen „válogatódnak össze" azok is, amelyek az initrd generálásakor az *initrd*be kerülnek bele.

dulhat, hogy ezen belül még egy szintet létrehoznak, pl. a loginnév kezdőbetűje szerint: `/home/`*loginnév-első-betűje*`/`*loginnév*.

Sok rendszeren szokták azt a módszert is alkalmazni, hogy a `/home` könyvtárat külön partícióra/eszközre teszik, hogy ha a felhasználók az adataikkal túl sok helyet emésztenének fel, az ne zavarja a rendszer működését. Olyan szempontból is hasznos lehet, hogy ha egy rendszeren felhasználónként kvóta van bevezetve az eltárolható adatok mennyiségét illetően, akkor a kvóta-információkat elég csak a **/home**-hoz tartozó eszközön nyilvántartani. Szintén értelmes ok lehet a `/home` különválasztására, ha egy rendszerben nagyon szigorúan veszik a biztonságot, és ilyenkor a `/tmp` és a `/home` olyan eszközökön van, amelyek a **noexec** opcióval vannak felmountolva, és így egy fájlra hiába van végrehajtási jogosultság, azt nem engedi ilyenkor futtatni, vagyis a felhasználók saját maguk semmilyen módon nem tudnak a rendszerre programokat (vagy akár trójaiakat, vírusokat, stb.) feltelepíteni.

A rendszergazda home könyvtára minden rendszeren a `/root` szokott lenni. Ennek egyik magyarázata az, hogy ha a rendszer valamiért nem képes a szokásos módon elindulni, akkor a rendszergazda egyfelhasználós módban amikor bejelentkezik be tudjon lépni akkor is, amikor az esetlegesen külön partícióra tett `/home` még nincs felcsatlakoztatva. Egy ilyen tipikus hiba tud lenni, ha a `/home`-on annyi adat van, hogy külön merevlemezre került, és pont ez a merevlemez hibásodott meg. Egy ilyen esetben a rendszergazda zavartalanul be tud lépni, és kideríteni a hiba okát. Pont ezért a `/root` könyvtárat nem szokás különválasztani a gyökérkönyvtárt tartalmazó eszköztől.

3.4.6. /usr, /usr/bin, /usr/sbin, /usr/lib

A `/usr` könyvtár alá a rendszer nagyobb része kerül. A `/usr` alatt úgyanúgy található `bin`, `sbin`, `lib` könyvtár mint a gyökérkönyvtárban, de azok a programok, amelyek a rendszer indulása közben, illetve annak egy korai szakaszában nem szükségesek, azok ide kerülnek feltelepítésre. Tipikusan a grafikus felületet adó X11 szerver, a nagyobb szolgáltatások (pl. SQL szerver, levelezőszerver, nyomtatási alrendszer, SSH szerver, ...), grafikus felületű alkalmazások, egyéb kiegészítő alkalmazások a gyökérkönyvtárbeli struktúra helyett a `/usr` alatt helyezkednek el. Ennek az a másik nagy előnye, hogy ha a `/usr` könyvtárat külön eszközre/partícióra tesszük, akkor azon fájlok, amelyeknek ténylegesen a `/`-t tartalmazó eszközön kell lenniük 50-200 MByte méretűre csökkenthetők, még egy teljes asztali rendszer esetén is.

3.4.7. /usr/src

A `/usr/src` könyvtárat arra szokás használni, hogy a rendszergazda ide teszi azon programok forráskódjait, amelyeket nem a rendszere által nyújtott szoftverként telepít fel, hanem forráskódként tölti le/szerzi be, és abból fordítja le a futáskész végrehajtható állományokat, majd telepíti fel. Egy tipikus példa erre, hogy manapság egyre több Unix rendszer érhető el szabad szoftverként[20] amely definíció szerint[21] lehetőséget ad a rendszer módosítására. Megesik, hogy egy-egy rendszeradminisztrátor az operációs rendszer kernelét további biztonságot növelő *patch*ekkel, vagy hivatalosan még el nem fogadott fejlesztői kiegészítésekkel változtatja meg, és fordítja le újra a forráskódból[22].

[20] pl. számos GNU/Linux disztribúció, BSD változatok, openSolaris
[21] http://www.gnu.org/philosophy/free-sw.html
[22] UTSL = Use The Source Luke

3.4.8. /usr/local

A `/usr/local` alatt a `/` és `/usr` könyvtárakhoz hasonlóan található egy struktúra, amelyben van `bin`, `sbin`, `lib` alkönyvtár. A különbség a többihez képest, hogy itt a rendszer szerves részét **nem** képező szoftverek komponensei találhatók. Pl. Ha rendszeradminisztrátor letöltött egy szoftvert, ami nem volt eleve megtalálható az operációs rendszer részeként telepíthető formában[23] és forráskódból fordította le, majd telepítette fel, akkor az minden valószinűség szerint (ha a rendszergazda **is** betartotta az *FHS*t) ide került.

3.4.9. /var, /var/spool, /var/log

A `/var` könyvtár a nevét a *variable* vagyis változó szó után kapta, utalván arra, hogy ide tipikusan olyan fájlok kerülnek, amelyeknek várhatóan rendszeresen megváltozik a tartalma. Ezek közül is két fontosabbat emelnék ki.

Az egyik a `/var/spool` ahova az ún. spoolfájlok kerülnek. Például ha kinyomtatunk egy dokumentumot, weboldalt, képet, ..., akkor abból először egy a nyomtató által is értelmezhető valami generálódik[24], majd a nyomtatást kezelő alrendszer hogy ne összeömlesztve kerüljenek kinyomtatásra a nyomtatási feladatok, azokat sorban egymás után adja át a nyomtatónak a *spoolfájl*ból felolvasva.

A másik kiemelt alkönyvtár a `/var/log` ahova a rendszer naplózó alrendszere a rendszerben futó programok által küldött naplózüzeneteket menti a konfigurációjában meghatározott módon. Mivel a rendszer hibakeresése, vagy egy internetre kötött fontos szolgáltatásokat ellátó gépen egy-egy biztonsági incidens után a történtek pontos visszakövetése szempontjából fontos hogy mi is történt/történik egy rendszerben, szokás ezt külön eszközre/partícióra tenni, még akár akkor is, ha már a `/var`t külön eszközre tették.

3.4.10. /opt

Bármily ideális állapot felé is tart világunk léteznek nem szabad szoftverek[25], amelyeknek a használatát nem tudjuk másképp megkerülni. Rendszerint ezek a *proprietrary software*-ek szokták magokat a `/opt` könyvtár alatt megtalálható struktúrába telepíteni.

[23]erre valók a szoftvercsomagok, pl. a debian alapú rendszerek a szoftvercsomagjaikat `.deb` kiterjesztésű fájlokba szokták tenni

[24]tipikusan PostScript dokumentum

[25]az angol proprietrary software kifejezésre nincs elterjedt és a köz által elfogadott magyar kifejezés még

4. fejezet

Folyamatok

4.1. A processzlista

A Unix rendszerek, ahogy szinte mindenről táblázatot vezetnek (pl. passwd/felhasználók, i-node tábla, ...), úgy az éppen aktuálisan futó folyamatok nyilvántartására is egy táblázatot vezet az operációs rendszer. Mint ahogyan a passwd-ben a felhasználóneveknek megfeleltet egy sorszámot, úgy a processzlistában is minden futó folyamatot(processzt) egy sorszámmal azonosít. Ez a sorszám a **PID**, ami a *Process ID* rövidítése.

Tekintsük át, hogy milyen információkat tart nyilván a rendszer az egyes folyamatokról és miért.

4.1.1. A parancs

Minden egyes folyamathoz alapfontosságú tudni, hogy mi is a hozzátartozó végrehajtandó kód, vagyis a hozzátartozó parancs/végrehajtható fájl.

4.1.2. Paraméterek

Szinte minden programnak/parancsnak megadhatunk a működését befolyásoló paramétereket a parancssorában. Ezt a listát eltárolja az operációs rendszer, így pl. amikor megnézzük a futó folyamataink listáját egy-egy processznél nem csak azt látjuk, hogy melyik parancs fut, de azt is, hogy az a példány milyen paraméterezéssel lett elindítva.

4.1.3. Szülő folyamat

Mint ahogyan a fájlok is könyvtárakba-alkönyvtárakba vannak szervezve, úgy a folyamatok is egy faszerkezetbe vannak szervezve. Vagyis minden folyamatnak van egy szülője, kivéve azt amelyik a "gyökér" szerepét látja el, vagyis minden folyamat elindulása hozzá vezethető vissza. Ez a folyamat az **init** parancs, ami elsőként indul az operációs rendszer betöltődése után.

Hogy ez a hierarchia visszakövethető legyen, a folyamatokhoz a szülő folyamat **PID**-je is el van tárolva. Ezt rendszerint **PPID** néven szokás illetni, ahol az első **P** a *Parent* szó rövidítése.

4.1.4. Környezeti változók

Egy egy parancs működését nem csak a számára átadott paraméterek befolyásolhatják, hanem a környezeti változók[1] is. Ennek tipikus példája a **PATH** amely meghatározza, hogy ha a parancs egy gyerekprocesszt indít, akkor a hozzátartozó végrehajtható állományt mely

[1] Angol neve: environment variables

könyvtárakban kell keresni[2]. De a segítségünkre lehet a **HOME** környezeti változó is, amely segít kitalálni, hogy mely könyvtár a homekönyvtárunk[3].

Másik hasznos környezeti változó pl. a **LANG** és az **LC_ALL** amely az egyes parancsok számára meghatározza, hogy az hibaüzeneteiket, visszajelzéseiket milyen nyelven közöljék a felhasználóval, így akár ugyanazon a számítógépen felhasználónként, vagy akár bejelentkezésenként eltérhet, hogy milyen nyelven kommunikál vele a rendszer[4].

4.1.5. Tulajdonosi jogosultságok

A parancs futásakor az operációs rendszer feladata eldönteni, hogy amikor az igénybe vesz egy szolgáltatást, vagy végrehajtana a rendszeren valamilyen módosítást (pl. hálózati interfész konfigurálása), akkor az engedély a számára megadható-e. Ehhez folyamatonként kell nyilvántartani, hogy ki a hozzá tartozó felhasználó, illetve a folyamat indításakor mi volt a hozzátartozó elsődleges csoport[5], illetve milyen csoportokba tartozott bele a parancs indításakor[6].

4.1.6. Effektív jogosultságok

Az *Effektív jogosultságok* eltárolására a *setuid*-es és *setgid*-es parancsok miatt van szükség. Tipikus példa erre a **passwd** parancs, ahol a *root* jogaira van szükség az új jelszó eltárolásához, de azt is tudunk kell, hogy ki a *valódi tulajdonosa* a folyamatnak, hogy az ő jelszavát ellenőrizzük le és változtassuk meg.

4.1.7. Prioritás

A Unix rendszereken különböző prioritású folyamatok futhatnak. Ha egy interaktív program fut, mint például egy levelezőprogram, ahol szeretnénk ha a levélbe begépelt betűket azonnal viszontlátnánk a monitoron, ott szeretnénk, hogy amikor lehetősége van futni a programak minél hamarabb átadná neki a vezérlést az operációs rendszer. Az ilyen folyamatok pl. tipikusan nem sok utasítást hajtanak végre, de a „felhasználói élmény” szempontjából fontos, hogy amint megvan a friss input, azonnal cselekedjen annak megfelelően (vagyis az épp gépelt levélben jelenjen meg a leütött betű).

Az ún. *batch* típusú feladatok pedig pont a másik végletet képviselik, ahol az input már előre rendelkezésre áll, legroszabb esetben is csak a lemezvezérlőre kell várni, de utána sok processzorutasításon keresztül a kapott adatokkal számol a program. Például szoltálnak erre a konverziós parancsok. Pl. egy `.bmp`-t `.jpg`-é alakító parancsnál nem bánjuk ha az operációs rendszer időnként elveszi tőle a vezérlést és ezáltal $0.125s$-al később készül el a képünk, mert sokkal jobban zavarna ha közben a zenelejátszónk „akadna”.

4.1.8. Aktuális könyvtár

Amikor nem a gyökérkönyvtárból kiindulva teljes nevével adunk meg egy megnyitandó fájlt, akkor az operációs rendszer az alapján tudja, hogy a folyamat a hivatkozott fájlt hol keresse,

[2]Kivétel ez alól az a ritka eset, amikor egy parancsot a gyökérkönyvtárból kiindulva a teljes nevével adunk ki

[3]A Unix könyvtárai fejezetben volt példa arra, hogy nem feltétlenül következtethető ki egyszerűen a loginnevünkből

[4]szemben más alternatív rendszerekkel, ez alapszolgáltatásnak tekinthető a rendszer logikus tervezésének köszönhetően a kezdetek óta

[5]Ennek fájlok létrehozásakor van jelentősége a nem setgid-es könyvtárakban.

[6]Ennek akkor van szerepe, ha pl. egy fájlt csak egy bizonyos csoport tagjai írhatnak/olvashatnak/hajthatnak végre, és a folyamat pont ezt tenné egy adott fájllal

hogy nyilvántartja mi az aktuális könyvtára. Ezt változtatja a parancsértelmező futó folyamatában a parancsértelmező számára szóló **cd** belső parancs, valamint az elindított folyamatok felé az aktuális könyvtár is öröklődik.

4.1.9. Root könyvtár

A Unix rendszerekben létezik egy **chroot** parancs, amely úgy indít el egy alfolyamatot, hogy előtte megváltoztatja a folyamat által látott / könyvtárat annak a paraméterben megadott alkönyvtárára. Ezt a lehetőséget tipikusan arra szokták használni, hogy a rendszerünkön olyan szolgáltatást futtassunk, amelyről tudjuk, hogy *sérülékeny kódja* van. Ilyenkor az adott szolgáltatáshoz egy miniális összetevőkkel rendelkező Unix rendszert `/bin`, `/sbin`, `/lib`, ... feltelepítünk egy alkönyvtárba, és ott futtatjuk. Ha fel is törték a szolgáltatást nyújtó programot, az ezáltal okozható károk minimalizálhatók így, hiszen a folyamat és alfolyamatai ezt a könyvtárt tekintik gyökérkönyvtáruknak, ezáltal a valódi rendszer többi könyvtárához nem férnek hozzá, annál kijjebb „nem látnak”.

Ugyanezt a technológiát használják a Unix rendszerek telepítőprogramjai is, amelyek a legtöbb esetben a memóriába betöltődött rendszerként futnak, megformázzák a megfelelő partíciókat, felcsatlakoztatják egy alkönyvtárba, majd a végleges beállításokat elvégző parancsok már a készülő rendszeren belül, a leendő /-t /-nek látva futnak le.

4.1.10. Egyéb erőforrások

Amit még a folyamatokról nyilván kell tartani:

- A folyamat által használt memóriaterületek.
- A folyamat által felhasznált shared libraryk. Ha egy shared library-t már egyetlen folyamat sem használ, akkor az általa foglalt memóriaterületet fel lehet szabadítani hasznos adatok számára.
- A folyamat által nyitva tartott fájlok. Hasznos tudni, hogy ha egy futó folyamat még nyitva tart egy fájlt, és mi a folyamat futása közben azt töröltük, attól még az operációs rendszer a hozzá tartozó területet nem szabadítja fel a fájlrendszerből a folyamat befejeződésének vége előtt. Egy példa: Ha CD-t írunk képfájlból, és az írás közben töröljük a képfájlt, akkor attól az íróprogram még rendesen be tudja fejezni a lemezt emiatt. Viszont ennek a mellékhatása, hogy ha akkut helyszűke miatt töröltük le a képfájlt, attól még a CDírás befejeződése előtt nem fogjuk visszakapni a hozzátartozó lemezterületet.

4.2. A folyamatok kezelésére szolgáló parancsok

4.2.1. ps

Az aktuálisan futó folyamatok és tulajdonságaik kilistázására szolgál. A *BSD* és *SystemV* típusú rendszereknél eltérő a paraméterezése. A Linux rendszereken ezt úgy hidalták át, hogy ha a --t kiírjk a kapcsolók elé, akkor a kapcsolókat a *SystemV*ban megszokottak szerint értelmezi, ha pedig nem írjuk ki, akkor a *BSD* rendszereken megszokott módon értelmezi.

Ha nem adunk neki kapcsolókat akkor csak az adott terminálról a felhasználó által elindított folyamatokat listázza ki. Ha a többi terminálról, és a többi felhasználó folyamatait is látni szeretnénk, akkor a **ps aux** illetve a **ps -ef** paraméterezésekkel érhetjük el a kívánt hatást.

A *BSD* stílusú paraméterezésben hasznos lehet még **ps auxwf** módon kiadni a parancsot, ahol a **w** kapcsoló hatására a teljes parancsot, az összes paraméterével kilistázza, függetlenül a

terminálunk szélességétől, valamint a **f** kapcsoló[7] a folyamatok kilistázásánál jelekkel mutatja a gyerek-szülő viszonyokból levezethető faszerkezetet.

4.2.2. top

A rendszeren épp aktuálisan futó folyamatok kilistázása. Folyamatosan újrasorrendezi a listát aszerint, hogy melyik terheli jobban a rendszert. Külön módosítás nélkül a sorbarendezés szempontja a CPU-használat. Hasznos lehet arra, hogyha úgy látjuk, hogy „akad” a rendszerünk, akkor megtalálhassuk a „bűnöst”.

Egy rendszer lassúságát nem csak sok CPU használatot igénylő folyamatok okozhatják. Tipikus problémája a Firefoxnak[8], hogy időnként irreálisan sok memóriát használ fel. Ennek detektálásában segíthet ha a top-ot rávesszük, hogy a memóriahasználat szerint rendezze sorba a folyamatokat. Így például, ha a Firefox volt a lassulás okozója, akkor sok esetben elég csak bezárni a programot, aminek hatására az általa használt memóriaterület felszabadul, majd ezután újra elindítani, hogy tiszta lappal kezdjen[9].

4.2.3. kill

A processzek egymással nem csak adatokat tudnak küldeni *IPC*vel, vagy *socket*eken keresztül, de létezik a kommunikációjuknak egy szegényesebb formája, amikor ún. *szigánál*okat küldenek egymásnak. A legtöbb szignál a processz befejeződést kéri a programtól, habár létezik a felfüggesztésre és folytatódásra irányuló kérelem is.

Szignálokat a kill paranccsal tudunk a processzeknek küldeni. Használatának lehetséges módjai:

- **kill** *PID*
- **kill** -*szignálneve PID*
- **kill** -*szignálsorszáma PID*

Amikor nem definiáljuk hogy milyen szignált küldjön a parancs, olyankor a **TERM**[10] szignált fogja elküldeni, amely egy arra irányuló kérelem, hogy fejeződjön be a program.

Előfordulhat, hogy a program valamilyen módon hibás[11] és nem hajlandó, vagy nem tud kilépni. Ilyenkor **KILL** szignált szokás küldeni neki[12]. Ez az egy szignál különleges, annyiban hogy ezt nem a program kezeli le önként, hanem az operációs rendszer fogja a program futását befejezni, és az erőforrásait felszabadítani (pl. nyitott fájlok, hálózati kapcsolatok lezárása, memória felszabadítása, ...).

Szintén általános használatnak örvend a **HUP** szignál[13], amelyre a Unix rendszerekben az általános elveket követő (rendszer- és hálózati)szolgáltatások úgy reagálnak, hogy újraolvassák a konfigurációs fájljaikat, és annélkül hogy egy pillanatra is megszakítanák a működésüket folytatódnak, de immár a friss konfigurációnak megfelelően.

---

[7] A *forest* szó rövidítése.

[8] A fejlesztők jelenleg készülő 3.0-s verzióhoz igérnek ilyen természetű javulást.

[9] A probléma nem annyira zavaró, mint elsőre hangzik. A sok memória felhasználást olyan gépen kell érteni, amit a gazdája nem szokott újraindítani és már napokkal/hetekkel korábban indította el a felhasználó a Firefoxát.

[10] A **TERM** szignál sorszáma a 15-ös

[11] Ahogyan a mondás tartja tökéletes program nem létezik, csak olyan amit már nem fejlesztenek

[12] A **KILL** szignál sorszáma a 9-es.

[13] A **HUP** szignál sorszáma az 1-es.

4.2.4. jobs, fg, bg

A legtöbb parancsértelmező definiálja a *job* fogalmát, ami ezesetben annyit jelent, hogy olyan gyerekfolyamat, amely a parancsértelmezőből került elindításra. Azonban a multitask tulajdonságnak, mint tervezési szempontnak köszönhetően a Unixos parancsértelmezők a kezdetektől fogva fel vannak készítve ezeknek a kezelésére.

A mivel a *job* fogalma a parancsértelmezőkhöz köthető ezért a hozzátartozó parancsok sem valódi parancsok, vagyis a `/bin`, `/sbin`, ... könyvtárakban nem találhatók meg, hanem a parancsértelmező belső parancsai[14]. Azonban a legtöbb parancsértelmezőben található egy **jobs** nevű beépített parancs amely a jobjainkat kilistázza.

A parancsértelmező ahhoz is segítséget nyújt, hogy a *job*ok futását felfüggesszük, vagy éppen újra elindítsuk előtérben vagy háttérben. A háttérben továbbfolytatásra a **bg** beépített parancs szolgál, az előtérben folytatásra pedig a **fg** parancs.

Az előtérben legfeljebb egyetlen job futhat! Ez azért van, mert minden egyes parancsértelmező egy hozzátartozó ún. *virtuális terminálról* lett elindítva. Az előtérben futó job fogja megkapni a terminálon bejövő inputeseményeket[15]. A háttérben futó folyamatok így nem tudnak interaktívan bemenethez jutni, ellenben ami a kimenetükre kerül azt a parancsértelmező „gondolkodás nélkül" továbbítja a terminálunkra. Amikor egy *job* sem fut előtérben, olyankor a parancsértelmező fogja feldolgozni a bejövő billentyűzeteseményeket, és az `enter` billentyű leütésének hatására kezdi értelmezni és futtatni az begépelt parancsot.

4.2.5. export

Ahogyan már láttuk, minden egyes folyamathoz tartoznak környezeti változók. A parancsértelmező azon környezeti változókat, amelyeket nem a *szülőjétől* örökölt, alapértelmezetten csak belső használatra szánja. Ha azt szeretnénk, hogy egy újonnan bevezetett környezeti változót a parancsértelmezőből indított gyerekfolyamatok is megkapjanak környezeti változójukként, akkor az **export** paranccsal tudunk erről gondoskodni. Használatának módja:
export *környezetiváltozó_neve*
Ahol a *környezeti változó neve* annak a környezeti változónak a neve, amelyet *exportálni* szeretnénk.

A környezeti változók listáját az aktuális értékükkel együtt az **env** parancs írja ki. Ha csak egy adott környezeti változó értékére vagyunk kíváncsiak akkor használhatjuk a következő formát is:
echo $*környezetiváltozó_neve*
Például a home könyvtárunk helyének megtudakolására az **echo $HOME** parancsot.

4.2.6. nice

Ahogyan már szóbakerült a processzekhez prioritást is rendel az operációs rendszer. Az interaktív ütemezést igénylő programok nem igényelnek külön módosítást, azonban ha valamit csökkentett prioritással szeretnénk futtatni, akkor a **nice** parancs lehet segítségünkre amely a *batch* típusúan ütemezendő folyamatok elindítására szolgál.

[14]Tipikusan a kill parancs is a parancsértelmezőkben beépítve található meg. Ha az igazi **kill** parancsot akarjuk futtatni, akkor a teljes nevével kell rá hivatkozzunk: **/bin/kill**.

[15]Pl. a billentyűleütéseket. De lehet épp egérkattintás is, ahogyan azt az **mc** parancsnál láttuk, hogy az egérrel is rá tudunk kattintani a menüsorára, fusson akár valódi terminálablakunkban, vagy egy távoli gépen, ahova az ssh vagy putty programok segítségével jutottunk be.

5. fejezet

A parancsértelmező

A unixok történelmében a parancsértelmezők fejlődésénél két nagyobb irányzat volt. Az egyik a *Korn Shell*-féle ág, a másik a *C Shell* féle ág. Az előbbi neve a **ksh** utóbbié a **csh**. A **ksh** ág elsődleges fejlesztési szempontja a minél kényelmesebb interaktív használat volt, míg a **csh** ág szempontjai közé inkább a jó programozhatósági lehetőségek beépítése tartozott.

A **csh** ág alá sorolható be a későbbi **tcsh** vagy akár a **zsh**.

Azonban a **ksh** felhasználóbarátságával nem mindenki volt megelégedve így születetett a *Bourne Shell*, vagy születési nevén **bsh**, majd miután ez még mindig nem annyira volt közkedvelt a *GNU projekt* keretein belül kifejlesztették a *Bourne Again Shell*-t, vagyis a **bash**t. Ebben már ilyen (a megjelenésekor újdonságnak számító) szolgáltatások jelentek meg mint a ...

- command completion, ami a tab billentyű hatására befejezi a parancsot ha az egyértelműen befejezhető, vagy a tab két egymás utáni leütésére a lehetséges befejezéseket jeleníti meg
- filename completion, ami a tab billentyű hatására ha parancs paramétereket írunk, akkor befejezi a fájl nevét, ha az egyértelműen befejezhető, illetve a tab két egymás utáni leütésére a lehetséges befejezéseket jeleníti meg.

De a parancssor kényelmes szerkesztésére, vagy korábbi parancsokban való keresésre is lehetőséget ad, ugyanakkor a **csh** ággal összehasonlítható eszköztára van az egyszerűbb programok, rutinfeladatok megírásához is.

5.1. Jobvezérlés

5.1.1. Vezérlőbillentyűk

A jobok vezérlésére használhatók a folyamatoknál már megismert parancsok is (**jobs**, **fg**, **bg**, **kill**), de szükség lehet arra, hogy menetközben interaktívan változtassunk a folyamatok állapotán. Ahogyan a parancsokra is általánosan érvényes a Unix típusú rendszerekben, hogy milyen parancs mit csinál, úgy ezek a billentyűkombinációk is általános érvényűnek tekinthetőek.

Az éppen előtérben futó folyamat megszakítására vonatkozó igényünket a CTRL-**c** billentyűkombinációval jelezhetjük a parancsértelmező számára[1]. A CTRL és valamilyen egyéb billentyű együttes lenyomását a kézikönyvek tipikusan a ^ szimbólummal szokták jelölni: vagyis a CTRL-**c**-t ^C-ként jelölik.

[1] A valóságban ilyenkor egy INT szignált küld a folyamatnak, amit az lekezel. Nem minden esetben kilépéssel, mert bizonyos szövegszerkeszőprogramok (**pico**, **nano**) pl. a vágólapra másolás funkcióra használják ezt a billentyűzetkombinációt.

Az előtérben futó folyamat felfüggesztésére vonatkozó kérésünket a CTRL-**z**-vel jelezhetjük. Ha az előtérben futó folyamat számára fájl-vége jelzést szeretnénk küldeni a terminálunkról, akkor pedig a CTRL-**d** billentyűkombinációt használhatjuk.

5.1.2. Vezérlő operátorok

Előfordulhat hogy egy parancsot eleve úgy szeretnénk elindítani, hogy a háttérben fusson. Ilyenkor a következő formában kell kiadni:
PARANCS **&**

A másik rendszeresen használt eset, amikor úgy szeretnénk elindítani egy parancsot, hogy közvetlen a befejeződése után induljon el rögtön egy másik parancs. Ilyenkor a **;** operátort használjuk:
PARANCS1 **;** PARANCS2

5.1.3. Átirányítások

A parancsértelmező által elindított alfolyamatok alapértelmezésben arról a terminálról kapják a bemenetüket amelyen maga a parancsértelmező is fut, valamint a kimenetük és a hibajelzéseik azon a terminálon jelennek meg. Ezt a három *stream*et így standard bemenetnek(*stdin*), standard kimenetnek(*stdout*) és standard hibacsatornának(*stderr*) szokás hívni.

Azonban szükség lehet arra, hogy egy program ne a terminálról kapja a bemenetét, vagy hoszabb kimenetet ad, amit később szeretnénk feldolgozni, ilymódon nem a terminálon szeretnénk viszontlátni azt. Esetleg egyszerűen csak nem szeretnénk látni a küldött hibaüzeneteket.

Ezen problémákat szinte minden parancsértelmező ugyazokkal az operátorokkal oldja meg:

> Kimenet átirányítása. Használatának általános módja: *parancs >kimenetifájlneve*
A parancs standard kimenett a megadott nevű fájlba irányítja. Ha korábban volt benne valami, akkor először kiüríti.

>> Kimenet átirányítása. Hatása és használatának módja hasonló mint az előzőé, de nem üríti ki a fájlt, ahova a kimenet kerül, hanem a végéhez hozzáírja a parancs által előállított kimenetet.

< Bemenet átirányítása. Használatának módja: *parancs <bemenetifájlneve*
A parancs a standard bemenetét a megadott nevű fájlból fogja kapni. A fájl végén kap egy fájlvége jelet, mintha csak egy CTRL-**d** billentyűkombinációt nyomtunk volna.

<< „Here is the document”. Használatának módja: *parancs <<végszó*
A parancsértelmező nem adja át közvetlenül a terminált a gyerekfolyamatnak, hanem először saját maga értelmezi, és ha megtalálja valamelyik sor tartalmaként a *végszó*t, akkor onnantól ismét parancsként kezdi értelmezni a terminálról jövő információkat az alfolyamatnak pedig küld egy fájlvége jelet. Tipikusan akkor van értelme, ha egyetlenegy scriptfájlban szeretnénk minden előre elvégzendő feladatot leírni valamilyen automatizmushoz, és már azt is előre tudjuk milyen bemenetet kell átadni egy programnak, de külön fájlokat nem áll módunkban vagy szándékunkban mellékelni.

2 > A *stderr* átirányítása fájlba. Előfordulhat, hogy nem szeretnénk látni egy adott program által kiírt hibaüzeneteket, vagy azokat később szeretnénk visszaolvasni. Ilyenkor ezeket egy fájlba át tudjuk irányítani[2].
A 2-es onnan jön, hogy a folyamatok által nyitva tartott fájlokat is egy sorszámmal azonosítja be a unix rendszer. Azonban bármilyen unixos környezetben induló parancs

[2]Ha nem szeretnénk látni valamilyen kimenetet, akkor tipikusan a `/dev/null` eszközbe szokás átirányítani

az indulásakor számíthat arra, hogy a következő három fájldeszkriptor már nyitva van: 0 *stdin*, 1 *stdout*, 2 *stderr*.

| Pipe. A pipeokat olyan esetben használjuk, amikor az egyik program kimenetét szeretnénk ha azonnal (köztes fájl használata nélkül) megkapna egy másik program a bemenetére. Ilyenkor őket egy ún. *pipe*-pal kötjük össze. Használata:
PARANCS1|PARANCS2

2 > &1 A *stderr* összefésülése a *stdout*tal. Ezt például szokás olyan esetben használni, amikor egy program üzen *normál* üzeneteket és hibaüzeneteket is, azonban a teljeset meg szeretnénk őrizni későbbi vizsgálódás/kielemzés céljából, vagy átadni egy másik program bemenetére az egészet.

5.2. Programozási lehetőségek

Ahogy a bevezető is utalt rá, a **bash** számos programozásból ismert fogalmat és eszközt kínál a felhasználóinak. Ezek egyike a környezeti változók használatának lehetősége.

A másik fontos dolog, amiről tudnunk kell a visszatérési érték fogalma. A parancsok a lefutásukkal nem csak kimenetet tudnak eredményezni. A befejeződésekor egy visszatérési értéket is visszaad, amely egy egy bájton eltárolt előjel nélküli egész szám. Vagyis az értéke 0 és 255 közt lehet. A visszatérési érték a következő jelentéssel bír: ha 0-t ad vissza, akkor evvel jelzi a hibamentes lefutást egy program. Ha más értéket ad vissza, akkor evvel hibát jelez. Bizonyos programok a hiba okát szokták a visszatérési értékkel jelezni, így különböző hibaokokhoz különböző visszatérési értéket adnak. Ez parancsonként/programonként eltérő, így ha erről konkrétat akarunk tudni, mindig az adott program kézikönyvében nézzünk utána a pontos jelentésnek.

A visszatérési érték azért is fontos, mert evvel a parancsértelmező számára egy igaz/hamis jelentéssel is bír.

5.2.1. Feltételes elágazások

A feltételes elágazások legegyszerűbb módja amikor csak egy parancs lefutásának sikerétől függően kell egy másikat elindítani. Erre a || és **&&** operátorok szolgálnak. Használatuk:

- Ha azt szeretnénk, hogy az első parancs sikeres befejezése esetén (0 visszatérési érték) indítsa el a másodikat, akkor a következő forma használatos:
  PARANCS1**&&**PARANCS2
- Ha azt szeretnénk, hogy az első parancs sikertelen befejezése esetén (0-től eltérő visszatérési érték) indítsa el a másodikat, akkor a következő forma a használatos:
  PARANCS1||PARANCS2

Elképzelhető olyan eset, amikor az adott parancs befejeződésének mindkét esetére más-más parancsokat szeretnénk lefuttatni, vagy több parancsot akarunk lefuttatni a befejeződése után, ha a feltételünk teljesült. Ilyenkor az **if then else** szerkezetet szokás használni:

```
if feltetelparancs; then
parancsigen1
parancsigen2
...
else
```

```
parancsnem1
parancsnem2
...
fi
```

Ebben az esetben a `feltetelparancs` azt a parancsot jelenti, aminek az igaz-hamis végeredménye alapján döntünk, hogy a *then* vagy az *else* ágat használjuk, a `parancsigen1` az első parancs a *then* ágban, ...

5.2.2. Számlálásos ciklus

A klasszikus értelemben vett számlálásos ciklust a parancsértelező nem ad a számunkra. Ellenben van olyan, ami egy halmaz minden elemén egyszer végrehajtja a ciklusmagot. Egy ilyen ciklushoz három dolog kell:

- ciklusváltozó
- halmaz elemekkel
- ciklusmag

Ezek után így fog kinézni egy ilyen ciklus a **bash**ban:

```
for i in a b c d ;
do
parancs1
parancs2
...
done
```

Aholis `i`-vel jelöltem a ciklusváltozót `a b c d` a halmaz elemei, és a `parancs1 parancs2 ...` parancsok vannak a ciklusmagban.

Ha valaki mégis ragaszkodna egy olyan ciklushoz, ahol a ciklusváltozó az $1..n$ halmaz elemein megy végig, akkor segítségére lehet a **seq** parancs, aminek ha egy n paramétert adunk, akkor 1-től n-ig elszámol. Ha pedig egy k és v paramétert, akkor pedig k-tól v -ig. Ha pedig nem egyesével, vagy nem növekvően szeretnénk számolni, akkor adhatunk neki 3 paramétert, amikor második paraméterként a lépésközt kell megadjuk:

```
pasztor@voyager:~$ for i in ‘seq 1 5‘ ; do echo $i ; done
1
2
3
4
5
pasztor@voyager:~$ for i in ‘seq 6 -1 3‘ ; do echo $i ; done
6
5
4
3
```

A számlálásos ciklus az elöltesztelősök közé tartozik, ti. ha a halmaz, amelynek az elemein végig kellene menjen egy üreshalmaz, akkor a ciklusmag nem hajtódik végre egyszer sem.

5.2.3. Elöltesztelős ciklus

Egy elöltesztelős ciklushoz két dologra van szükségünk: Egy tesztfeltételre, és egy ciklusmagra, amit végrehajtunk. Mivel a visszatérési értékkel tudnak jelezni a parancsok a parancsértelmező számára egy igaz-hamis jellegű értéket, így a tesztfeltétel gyakorlatilag az, hogy a feltételként megadott parancs hibátlanul fut-e le. Általánosságban ez így néz ki:

```
while feltetel;
do
parancs1
parancs2
...
done
```

A példa kedvéért említsük meg a **pidof** parancsot, amelynek ha mondunk egy parancsnevet, akkor megmondja, hogy milyen *pid*del, vagy *pid*ekkel fut ilyen folyamat. Ha nem fut ilyen, akkor nem ír ki semmit sem, és 0-tól eltérő hibakódot ad. Tegyük fel elindítottunk egy konverziót a számítógépünkön, ami lefoglalja, de azt szeretnénk hogy ahogyan befejezte a munkát kapcsolódjon is ki, mert elmentünk otthonról. Ilyen esetben pl. egy ilyen összetetteb parancs lehet a segítségünkre:

```
while pidof mencoder >/dev/null ; do sleep 10 ; done ; halt
```

Figyeljük meg, hogy közben a pidof által rendszeresen kiírt pid-et elküldjük a `/dev/null`ba, különben azt tapasztalnánk, hogy tele van a képernyőnk az adott piddel. A sleep 10-et azért kell a ciklusmagba tenni, mert ha valami nagyon rövid idő alatt lefutó dolgot adnánk meg, akkor a mencoderrel nem is jutna ideje a processzornak foglalkoznia, mert állandóan a feltételt ellenőrizgetné csak. Valamint használjuk a **while** ciklus után a **;** operátort is, azért hogy a ciklusból amint kilép, állítsa le a gépünket, ahogyan azt szerettük volna.

5.2.4. Többágú elágazás

Előfordulhat olyan eset is, amikor egy változó tartalmától függően szeretnénk végrehajtani valamit, de a változó különféle értékeket vehet fel, és nem egy egyszerű két irányú elágazásról van szó. Tény, hogy az ilyen eseteket is leírhatnánk több egymás utáni **if**-es elágazást használva, csak épp olvashatatlan lenne a „kód".

Ezen esetekre találták ki a **case** elágazást, ahol akár mintákat is megadhatunk a fájlneveknél megszokott ***** és **?** helyettesítőkaraktereket használva.

```
case kifejezes in
  minta1)
parancs1.1
parancs1.2
;;
  minta2)
parancs2.1
parancs2.2
;;
...
  *)
parancse.1
parancse.2
;;
esac
```

A kifejezés helyére tipikusan egy változó értékét szokás behelyettesíteni. Pl. a szolgáltatásokat indító scriptek, amelyeket jellemzően **start**, **stop**, **restart**, **reload**, **status** paraméterekkel lehet általában meghívni a `$1` speciális változót használják, ami a scriptnek átadott első paramétert jelenti. Egy ilyen script váza általában nagyjából így néz ki:

```
case "$1" in
    start)
        a_szolgaltatast_elindito_parancs
        visszajelzes_a_sikeres_elinditasrol
        ;;
    stop)
        a_szolgaltatast_leallito_parancs
        visszajelzes_a_sikeres_leallitasrol
        ;;
    restart)
        a_szolgaltatast_leallito_parancs
        visszajelzes_a_sikeres_leallitasrol
        sleep 1
        a_szolgaltatast_elindito_parancs
        visszajelzes_a_sikeres_elinditasrol
        ;;
    reload)
        kill -hup `pidof szolgaltatas`
        ;;
    status)
        a_szolgaltatas_ellenorzese
        ;;
    *)
        hasznalati_utasitas_kiirasa
        exit 1
        ;;
esac
```

Figyeljük meg azt az érdekes részletet, hogy ha nem megfelelő paraméterekkel hívunk meg egy ilyen parancsot, akkor a használati utasítás után, egy 1-es hibakódot adnak vissza jellemzően, jelezvén, hogy nem volt valami rendben.

5.2.5. Feltételek

A vezérlési szerkezetek amelyeket a parancsértelmező nyújt, önmagukban még nem lennének elegek ahhoz, hogy valamire való scripteket írjunk. Ezért az egyik leghasznosabb parancsunk a test, amellyel elenőrizhetünk fájlokat bizonyos szempontok szerint, összehasonlíthatunk két értéket, stb. Néhány a használati lehetőségei közül:

- `test -r fájlnév` ellenőrzi, hogy a paraméterül adott fájl olvasható-e.
- `test -w fájlnév` ellenőrzi, hogy a paraméterül adott fájl írható-e.
- `test -x fájlnév` ellenőrzi, hogy a paraméterül adott fájl végrehajtható-e.
- `test -e név` ellenőrzi, hogy a paraméterül adott fájl/könyvtár/... létezik-e.
- `test -d név` ellenőrzi, hogy a paraméterül adott név könyvtár-e.

- `test -f név` ellenőrzi, hogy a paraméterül adott név fájl-e.
- `test -L név` ellenőrzi, hogy a paraméterül adott név symlink-e.
- `test -b név` ellenőrzi, hogy a paraméterül adott név blokkos eszköz-e.
- `test -c név` ellenőrzi, hogy a paraméterül adott név karakteres eszköz-e.
- `test -p név` ellenőrzi, hogy a paraméterül adott név named pipe-e.
- `test -S név` ellenőrzi, hogy a paraméterül adott név socket-e.
- `test -n szöveg` ellenőrzi, hogy a **szöveg** hossza nullánál nagyobb-e.
- `test szöveg1 = szöveg2` ellenőrzi, hogy a **szöveg1** megegyezik-e a **szöveg2**-vel.
- `test szöveg1 != szöveg2` ellenőrzi, hogy a **szöveg1** eltér-e a **szöveg2**-től.
- `test szám1 -eq szám2` ellenőrzi, hogy a **szám1** egyenlő-e a **szám2**-vel.
- `test szám1 -lt szám2` ellenőrzi, hogy a **szám1** kisebb-e a **szám2**-nél.
- `test szám1 -gt szám2` ellenőrzi, hogy a **szám1** nagyobb-e a **szám2**-nél.
- `test szám1 -le szám2` ellenőrzi, hogy a **szám1** kisebb vagy egyenlő-e a **szám2**-vel.
- `test szám1 -ge szám2` ellenőrzi, hogy a **szám1** nagyobb vagy egyenlő-e a **szám2**-vel.
- `test szám1 -ne szám2` ellenőrzi, hogy a **szám1** nemegyenlő-e a **szám2**-vel.

A **test** parancs a `[`-ként is elérhető, csak ilyenkor az utána álló kifejezést egy `]`-al le kell zárni. Ez a két parancs tartalmilag megegyezik, de utóbbi egy öszetettebb **bash**programot megnézve olvashatóbb:

```
# Ha nem létezik a script beállításait tartalmazó fájl, kilépünk hibakóddal.
[ -r /etc/parancsom/beallitasai ] || exit 1
```

5.2.6. Változók, behelyettesítések

A változók, ahogyan a programozásra szánt nyelvekben is értékek eltárolására szolgál. Azonban nem céltalanul szokásunk ezeket eltárolni, időnként szükségünk van rá, hogy tudjuk mennyi is az annyi. A változók behelyettesítése a `$` jel után írt változónévvel történik. Előfordulhat, hogy egy parancsnak úgy szeretnénk egy paramétert átadni, hogy közvetlen utána már más szöveggel folytatnánk a paramétert, és nem szeretnénk szóközt tenni. Ilyenkor a következő formula használatos: `${változónév}`

```
pasztor@voyager:~$ echo A home könyvtáram a ${HOME}.
A home könyvtáram a /home/pasztor.
```

A parancsok a paramétereiket „egyesével” kapják meg. Egy paraméterbe kerülhet szóköz is, de a parancsértelmező a paramétereket a szóközök mentén vágja szét, és adja következő paraméterként. Ha mégis olyan paramétert szeretnénk átadni egy parancsnak, amely szóközöket is tartalmaz, akkor idézőjelekkel körbe kell zárni a paramétert. Az idézőjelekből is van többfaja: `'`, `"`, `` ` ``. A hatásuk is eltérő:

' A közé zárt kifejezést pontosan idézi, semmilyen helyettesítést nem végez el belül.

" A közé zárt kifejezésben a változókat, és a ‘-os kifejezéseket behelyettesíti.

‘ A közé zárt kifejezést, mint parancsot értelmezi, és amit a parancs a standard outputjára kiír, azt az értéket fogja behelyettesíteni.

Íme mindez a gyakorlatban:

```
pasztor@voyager:~$ echo '$HOME'
$HOME
pasztor@voyager:~$ echo "$HOME"
/home/pasztor
pasztor@voyager:~$ echo A usernevem: `whoami`
A usernevem: pasztor
```

Mivel a parancsoknak szoktunk átadni paramétereket, úgy a **bash**ban megírt programjaink is tudnak paramétereket átvenni. Ezek speciális változókban vannak:

$1 Az első átadott paraméter

$2 A második átadott paraméter

$... ...

$* Minden átadott paraméter

$# Az átadott paraméterek száma

$0 A parancs neve, amilyen néven el lett indítva

$? A legutoljára lefuttatott parancs visszatérési értéke

$$ A shell saját *PID*je

A **shift** beépített parancs hatása pedig az, hogy a legelső átadott paramétert eldobja, ezáltal eggyel csökkentve a paraméterek számát. Ha nincs több eldobható paraméter, akkor hamis visszatérési értéket ad. Ha pl. fel akarjuk dolgozni, hogy milyen paraméterekkel, kapcsolókkal hívták meg a **bash**ban írt programunkat, akkor egy ilyen szerkezettel könnyedén fel tudjuk dolgozni a kapott paramétereket:

```
p=$1
while shift;
do
...
p=$1
done
```

5.2.7. Függvények

Ha összetettebb részfeladatokat akarunk használni a *script*jeinkben, erre is ad lehetőséget a **bash**:

```
fvnev () {
parancsok
}
```

Természetesen a függvényeinknek ugyanúgy adhatunk át paramétereket, mint a komplett parancsoknak. Ilyenkor a függvények a nekik átadott paramétereket látják a `$1`, `$2`, ... speciális változókban.

6. fejezet

Unixos segédprogramok

A scriptjeink írásához már az eddig megismert eszközök is sokat segítenek, de van néhány igazi „klasszikus”, amely nem hiányozhat az eszköztárunkból.

6.1. grep

A grep parancs arra használható, hogy szavakat keressünk szövegfájlokban. A legegyszerűbb használati módja, amikor így hívjuk meg: `grep szó`. Ilyenkor a *standard input*járól veszi a szöveget, és csak azokat a sorokat adja át a *standard output*jára, amiben szerepel a paraméterül megadott szó.

Előfordulhat, hogy csak arra vagyunk kíváncsiak, hogy szerepel-e egy szó egy fájlban. Ilyenkor használhatjuk a **-q** kapcsolót, és az első találat után kilép, és nem olvassa tovább a fájlt. Ezt a módot tipikusan scriptekben szokták használni pl. egy **if** feltételéül, tudniilik, amikor egyáltalán nem volt találat, akkor a **grep** 0-tól eltérő *hibakódot* ad vissza.

Ha több paramétert is adunk neki, akkor a másodiktól kezdve már fájlnevekként tekinti őket, és nem a *standard input*járól veszi a szöveget, hanem a megnevezett fájlból/fájlokból. Ha több fájlnevet adtunk meg neki akkor a sorok elejére azt is kiírja hogy az adott sor melyik fájlból származik.

Ha arra is kíváncsiak vagyunk, hogy a fájl hanyadik sorában találta meg az adott szót, akkor a **-n** kapcsolóval tudjuk rábírni a **grep**et, hogy a sorok elejére a sorszámot[1] is írja ki.

Ha az olyan sorokra vagyunk kíváncsiak, amelyekben **nem** szerepel az adott szó, akkor a **-v** kapcsolóra van szükségünk.

6.2. sed

A **sed** parancs legfontosabb funkciója, hogy a bemenetét úgy küldi át a kimenetére, hogy közben apróbb módosításokat tud rajta végezni. A leggyakoribb ilyen módosítás, amikor egy szót kicseréltetünk egy másikra: `sed 's/cica/macska/'`

```
pasztor@voyager:~$ echo A cica megette az egeret. | sed s/cica/macska/
A macska megette az egeret.
```

Az ilyenkor adott feladatoknál az `s` a *szubsztitúció* szó rövidítése. A második `/` jel után további a működést módosító kapcsolók adhatók meg. A **g** gondoskodik róla, hogy egy-egy sorban minden előfordulást kicseréljen ne csak a legelsőt, a **i** pedig arról, hogy a kis és nagybetűk közti különbségtől tekintsen el a mintaillesztéskor. Példa[2]:

[1] Ilyenkor az eredeti fájlbeli sor számát írja ki.

[2] A példában felhasználtuk azt, hogy ha egy \ jelet írunk a parancssorba, akkor utána egy > el jelzett prompt után folytathatjuk a parancsot.

```
pasztor@voyager:~$ echo A Cica megette az egeret. | sed s/cica/macska/
A Cica megette az egeret.
pasztor@voyager:~$ echo A Cica megette az egeret. | sed s/cica/macska/i
A macska megette az egeret.
pasztor@voyager:~$ echo A cica megfogta az egeret. \
> Az egér így végezte a Cica gyomrában. | sed s/cica/macska/i
A macska megfogta az egeret. Az egér így végezte a Cica gyomrában.
pasztor@voyager:~$ echo A cica megfogta az egeret. \
> Az egér így végezte a cica gyomrában. | sed s/cica/macska/g
A macska megfogta az egeret. Az egér így végezte a macska gyomrában.
pasztor@voyager:~$ echo A cica megfogta az egeret. \
> Az egér így végezte a Cica gyomrában. | sed s/cica/macska/g
A macska megfogta az egeret. Az egér így végezte a Cica gyomrában.
pasztor@voyager:~$ echo A cica megfogta az egeret. \
> Az egér így végezte a Cica gyomrában. | sed s/cica/macska/ig
A macska megfogta az egeret. Az egér így végezte a macska gyomrában.
```

6.3. cut

A **cut** parancsot arra használjuk, hogy egy szöveg soraiból bizonyos részeket tudjunk meg. A kivágandó részt többféleképp is meg tudjuk fogalmazni. Például a kivágandó karaterekre hivatkozással:

```
pasztor@voyager:~$ ls -l | cut -c 1,54-

d Dokumentumok
l Examples -> /usr/share/example-content
d Képek
d Munkaasztal
d Nyilvános
d Sablonok
- screenshot1.bmp
d TeX
d Videók
d Zene
```

A másik módszer, hogy megadjuk mely oszlopokat kérjük. Ilyenkor azt is meg kell adjuk, hogy az oszlopokat milyen karakterek választják el egymástól. Erre jó példa a `passwd` fájl, ahol :-tal elválasztott oszlopok vannak. Pl. egy utasítással megtudhatom, hogy mi a *login*nevem, a hozzátartozó *UID*, *GID*, és *home könyvtár*:

```
pasztor@voyager:~$ grep ‘whoami‘ /etc/passwd | cut -d : -f 1,3,4,6
pasztor:1000:1000:/home/pasztor
```

6.4. head

A fájl elejének a kiírására szolgál. Hasonlóan a **grep**hez, ha nem adunk meg neki fájlnevet, akkor a *standard input*ról veszi az adatait. Ha a sorok számára vonatkozó paramétert sem adunk meg neki, akkor az első 10 sort írja ki. Szintén hasonló abban is a **grep**hez, hogy több fájlt is meg lehet neki adni paraméterül.

```
pasztor@voyager:~$ head -2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
root@voyager:~# head -2 /etc/group /etc/gshadow
==> /etc/group <==
root:x:0:
daemon:x:1:

==> /etc/gshadow <==
root:*::
daemon:*::
```

6.5. tail

A fájl végének kiírására szolgál. Teljesen hasonló a **head**hez a paraméterezése. Annyi eltérés van, hogy a sorok számánál adhatunk meg úgy értéket is, hogy ne a fájl utolsó n sorát írja ki, hanem az n. sortól kezdve írja ki a sorait.

```
pasztor@voyager:~$ seq 1 10 | tail -2
9
10
pasztor@voyager:~$ seq 1 10 | tail -n +8
8
9
10
```

A tail-t még egy dologra szokták használni a **-f** kapcsolójának[3] köszönhetően: Ha egy fájlnak kíváncsi vagyunk a végére, de szeretnénk látni, ha hozzáíródik valami. Ilyenek tipikusan a rendszer naplófájljai. Ha szeretnénk követni, hogy épp mi történik a rendszerünkön, akkor használhatjuk a `tail -f /var/log/syslog` parancsot.

6.6. wc

A **wc** parancs a nevét a *word count* szavak után kapta. Működése szabályozható kapcsolókkal. Ha kapcsolókat nem kap akkor a sorok, szavak és betűk/karakterek számát számolja össze egy fájlban/fájlokban/a bemenetén.

Hogy miben számoljon össze teljesen hasonlóan vezérelhető, mint a többi program esetén: ha nem kap fájlnevet, akkor a bemenetéről számolja ezeket össze. Ha pedig több fájlnevet kap, akkor kiírja, hogy egyesével melyikben mennyi van, majd végül egy összesítést.

Ha nem szeretnénk egy fájlról mindhárom adatot tudni, akkor használhatjuk a kapcsolóit:

-l A sorokat számolja össze.

-w A szavakat számolja össze.

-c A betűket számolja össze.

[3] A *forever* szó rövidítése.

```
pasztor@voyager:~$ wc /etc/passwd
  32   48 1478 /etc/passwd
pasztor@voyager:~$ wc -l /etc/passwd /etc/group
  32 /etc/passwd
  61 /etc/group
  93 összesen
```

6.7. sort

Fájl sorainak ABC szerinti sorbarendezésére szolgál. Használata a korábbiakéhoz hasonló: Ha nem kap fájlnevet, akkor a *standard input*ról veszi a sorbarendezendő sorokat, ha kap akkor onnan. A sorbarendezett adatokat a *standard kimenet*ére írja ki.

```
pasztor@voyager:~$ sort <<VEGE
> Bicaj
> Kerékpár
> Jármű
> VEGE
Bicaj
Jármű
Kerékpár
```

6.8. uniq

Egy fájlból kiszűri az egymás után következő egyforma sorokat. Használata, mint a korábbiaké...

```
pasztor@voyager:~$ uniq <<VEGE
> Bicaj
> Bicaj
> Kerékpár
> Bicaj
> VEGE
Bicaj
Kerékpár
Bicaj
```

Ha azt szeretnénk, hogy ne csak az egymás után következő sorokkal csinálja meg, hogy egy sor csak egyszer szerepeljen, akkor érdemes előtte a **sort** paranccsal sorbarendezetni a sorokat, mert így az egyforma sorok egymás után kerülnek.

7. fejezet

Virtuális Memóriakezelés

A modern processzorok ahogy megjelentek, megjelent bennük az ahhoz szükséges tudás, hogy az operációs rendszer az alkalmazások számára rugalmasan tudja a memóriát kezelni. Ez a gyakorlatban annyit tesz, hogy ha egy alkalmazásnak több vagy kevesebb memóriára van szüksége, mint amennyi épp a rendelkezésre áll, akkor ezt jelezheti az operációs rendszer felé. Azonban a memóriában több folyamat is fut párhuzamosan. Mint ahogyan a fájlok a lemezen „töredékenként” vannak, úgy az alkalmazások adatterülete a memóriában is szét van szórva. Az operációs rendszer beállítja a memóriavezérlő egységet[1], mert az már az ő feladata, hogy az alkalmazásnak a szétszórt memóriadarabkákat egy összefüggő egészként mutassa a valódi memóriára leképezve.

Természetesen olyan eset is előfordulhat, hogy egy alkalmazásnak több memória kell, mint amennyi épp szabadon elérhető a valódi memóriából. Ilyen eseteket is le lehet kezelni virtuális memória használatával. A dolog lényege annyi, hogy a merevlemez egy adott területét virtuális memóriaként használja az operációs rendszer, és a memóriából nem használt darabokat ide „kipakol”. Ha egy alkalmazásnak olyan memóriaterületre van szüksége, ami nincs a fizikai memóriában, akkor a **MMU** egy megszakítást generál, amellyel visszakerül a vezérlés az operációs rendszerhez, ezáltal annak lehetősége nyílik arra, hogy a memóriából régen/nem használt területet tegyen ki a merevlemezre, majd az így felszabaduló helyre betöltse az épp igényelt adatrészt a fizikai memóriába beolvassa, ezután az **MMU**t újrafelprogramozza, az épp aktuális helyzetnek megfelelően. Ilymódon az alkalmazás az egész háttérben történő dolgokból semmit nem vesz észre, számára ez az egész *transzparens*.

A memóriaterület kiterjesztésére rendszerint egy külön partíciót szokás használni a merevlemezen, mert egy a fájlrendszeren ezt egy fájlban tárolva, könnyen *töredezett*é válhatna. Ez sokat rontana annak hatékonyságán. Erre szolgál az ún. *swap*partíció.

Ahogyan a partíciók fájlrendszerrel való formázására, csatlakoztatására, leválasztására is vannak **mkfs**, **mount**, **umount** parancsok, úgy a *swap*pelésre előkészítésre van az **mkswap** parancs, a *swap* aktiválására a **swapon** parancs és a *swap* inaktiválására a **swapoff** parancs.

Programok memóriahasználásának felderítésénél hasznunkra lehet a már ismert **top** parancs. A memória információinak megtudakolásában pedig a **free** parancs van a segítségünkre, amely megmondja mennyi memóriánk van, abból mennyi van használatban, mennyi swapünk van és abból mennyi van használatban:

```
pasztor@voyager:~$ free
             total       used       free     shared    buffers     cached
Mem:       2031060    2000628      30432          0         20     784264
-/+ buffers/cache:    1216344     814716
Swap:      4194296      38400    4155896
```

[1] **MMU**

8. fejezet

VI és regexpek

8.1. Reguláris kifejezések

A legtöbb esetben pl. a **grep**-pel nem csak egyszerű szavakra kereshetünk, hanem sokkal összetettebb feltételeket is meghatározhatunk, ha ismerjük a reguláris kifejezéseket. Íme az alapok:

betűk Önmaguk megfelelői

. Egy tetszőleges betű

[ ] Halmaz. A [és] jel közt felsorolt betűk bármelyike. Ha a halmazt a ^ jellel kezdjük akkor az azt jelenti, hogy a halmaz komplementerére fog illeszkedni a kifejezés.

* Az előtte álló kifejezés 0 vagy többször fordulhat elő

+ Az előtte álló kifejezés 1 vagy többszor fordulhat elő

? Az előtte álló kifejezés 0 vagy 1szer fordulhat elő.

( ) A benne lévő kifejezések csoportosítása. Ha használjuk belül a | jelet is, az azt jelenti hogy több különböző alternatívát adunk meg. Pl. a `(Ádám|Éva)` kifejezés mind az *Ádám*, mind az *Éva* szavakra illeszkedik.

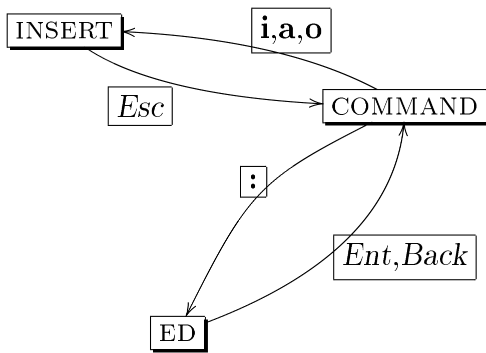
8.2. Az alap vi bemutatása

A vi használatában három üzemmód van:

insert mode Beszúró üzemmód. Szabadon gépelhetünk szöveget, ez bekerül a dokumentumba

command mode Különböző parancsokat adhatunk ki (pl. kurzor alatti karakter törlése **x**, sor törlése **dd**, beszúrás **i**, utána-írás/*append* **a**, új-sor kezdés **o**)

ed mode Különböző az ed szövegszerkesztőben megszokott parancsokat adhatunk ki (pl. fájlba írás **:w**, fájl beolvasása **:r***fájlnév*, helyettesítés **:s**)



8.1. ábra. Váltás a vi módjai közt

8.2.1. A command mód fontosabb parancsai

x Annak a karakternek a kitörlése, amin a kurzor áll

d Sor, vagy sorok törlése

y Sor, vagy sorok vágólapra helyezése

p Vágólap beillesztése a kurzor után

i Beszúró módba váltás, a kurzor helyére szúr be

a Beszúró módba váltás, a kurzor utáni helyre kezd beszúrni

o Beszúró módba váltás, új sort kezd, a kurzor utáni sorban

w Következő szóra ugrás

c Szövegrész cseréje

/ Reguláris kifejezés keresése.

? Reguláris kifejezés keresése a szövegben visszafele haladva.

n A következő találat megkeresése, az előzőleg már megadott regexp-hez.

h l j k Balra, Jobbra, Le, Fel mozgás a szövegben. (Általában a kurzorbillentyűk ugyanúgy használhatók)

~ Megcseréli az aktuális karaktert, ha az kisbetűs volt annak nagybetűs változatára, ill. ha nagybetűs volt, akkor a kisbetűs változatára.

. Megismétli a legutolsó vi parancsot

Megj.: a p,i,a,o parancsoknak van nagybetűs változata is. A vi „gondolkodásmódja", amikor parancsot adunk ki:
[Multiplikációs szám]PARANCS**[[Multiplikációs szám] Argumentum]**
Pl. törlés *(d)*, vagy vágólapra másolás *(y)* esetén tudatni kell a vi-jal, hogy „mit" szeretnénk kivágni. Pl. ha szeretnénk a kurzortól kezdődő 14 betűt a vágólapra tenni, akkor a következő billentyűket kell leütnünk: **d14**SPACE
További példák:

2×5 sor törlése	**2d5d**
5 sor vágólapra tétele	**5yy** vagy **y5y**
14 karakter törlése	**d14␣** vagy **14d␣** vagy **14x**
5 szó vágólapra tétele	**y5w** vagy **5yw**
2 szó kicserélése a 'broaf' szövegre	**c2wbroaf** ESC

Megjegyzések:

- ESC*-re azért van szükség, mert a c parancs is beszúró módba tesz.*
- A nagybetûs változatok:

	kisbetûs eset	nagybetûs eset
i/I	beszúr	sor elejére szúr be
o/O	következő sorba kezd új sort	előző sorba kezdi az új sort
p/P	utána beilleszt	aktuális pozíciótól kezdve beilleszt
a/A	a kurzor után beilleszt	a sor végére illeszt be

8.2.2. Az ed mód fontosabb parancsai

r fájl beolvasása. A kurzor utáni sorban kezdve beszúrja a megadott fájl tartalmát.

w fájl elmentése. Ha fájlnevet is megadunk, akkor a megadott fájlnévbe ment. Ha előtte megadjuk, hogy csak bizonyos sorokra vonatkozzon a parancs, akkor csak azokat a sorokat menti el. Pl. a kurzor sorát, és az utána következõ még 4 sort (tehát összesen 5 sort) szeretnénk egy másik fájlba elmenteni, akkor a kiadandó parancs: **:,+4w masikfile.txt**

q Kilépés. Ha megváltozott a fájl, és nem mentettük azóta, akkor nem lép ki, hanem figyelmeztet.

wq Elmenti a fájlt, és utána lép ki.

q! Mindenféleképp kilép.

w! Ha a fájl írásvédett, akkor is megpróbálja felülírni.

s csere *(szubsztitúció)*. Reguláris kifejezést adhatunk meg cserére, és a perl-ben megszokott módon lehet kifejezéseket megadni. Részleteket lásd késõbb. Néhány példa:

Az aktuális sorban a joe szót cseréljük ki vim-re.	**:s/joe/vim/**
Az aktuális sorban az összes joe szót cseréljük ki vim-re.	**:s/joe/vim/g**
Az aktuális és köv 2 sorban a sor elsõ joe-ját cseréljük ki vim-re.	**:,+2s/joe/vim/**
Az elsõ sortól az utolsó előttiig a sor első joe-ját cseréljük ki vim-re.	**:1,$-1s/joe/vim/**

8.3. A VIM-ben megjelent új parancsok

V *Visual Line*: Sorok kijelölésére szolgál, és a megadott parancsot az ily-módon kijelölt sorokra hajtja végre. Pl. Ha nem tudjuk fejbõl, hogy hány sort szeretnénk a vágólapra tenni, akkor csak kijelöljük õket, és utána y-t ütünk.

^**V** *Visual Block*: Hasonló az előzőhöz, de itt nem sorokat, hanem téglalap alakú területeket tudunk a szövegben kijelölni.

< Kijjebhúzza a kijelölt programblokkot.

> Beljebhúzza a kijelölt programblokkot.

8.4. A VIM-ben megjelent új ed-szerű parancsok

:sp *Split screen*: Vízszintesen kétfelé osztja a képernyőt, és ha fájlnevet is megadunk, akkor a megadott fájlnevet betölti a képernyő másik felébe.

:vsplit *Vertical Split screen*: Ugyanaz mint az előző, csak függőlegesen vágja ketté a képernyőt.

A szétosztott képernyők esetén a mozgás a képernyőrészek közt a CTRL-**w** -vel történik. Figyelem: Ez csak command módban használható!
Használata: Ha még egyszer leütjük, akkor ciklikusan a következõre ugrik. Ha megadott irányba akarunk elmozdulni, akkor a már megszokott módon leütjük a **h**, **j**, **k**, **l** billentyűk valamelyikét, vagy a kurzormozgató nyilak valamelyikét. Ha az aktuális képszeletet szeretnénk növelni, akkor a +, ill ha csökkenteni szeretnénk akkor a - billentyűket kell leütni. Itt is ugyanúgy üthetünk elõtte vagy a CTRL-**w** után számokat, hogy adott számszor hajtsa végre a mûveletet (, vagyis ne csak 1 sorral növekedjen/csökkenjen az ablak mérete, hanem pl. 5-el: CTRL-**w 5+**).

8.5. Makrózás a vim-ben

A vim a makrókat az angol abc 26 betűjével azonosítja. Ha egy makrót rögzíteni akarunk, akkor üssük le a **q** betűt, majd a makró *nevét*. Az ezután következő vi parancsokat a vim megjegyzi. Ha a makrónkat be akarjuk fejezni, akkor adjuk ki ismét a **q** parancsot. Ha a státuszsor be van kapcsolva, akkor a makró rögzítése közben a státuszsorban a *recording* felirat látható. A makrót tartalmazó vi parancsokat elő akarjuk hívni, akkor a **@** jelet és a makró nevét kell leütni. Ha a makrót egymás után többször akarjuk végrehajtani, akkor a **@** előtt üssük le a számot is, ahányszor végre akarjuk hajtani.

8.6. Néhány hasznos alapbeállítás, .vimrc

syntax on Syntax highlighting bekapcsolása. Automatikusan megpróbálja felismerni a szerkesztett fájl típusát, és az ehhez tartozó szintaxismodult betölti, és ennek megfelelõen „színezi a szöveg részeit.

filetype plugin on Ha felismerte a fájltípust, akkor ennek megfelelően betölt néhány ehhez tartozó alapbeállítást. Pl. C fájlok esetén automatikusan indentel gépelés közben. Gyakorlatban ki kell próbálni, pl. egy **switch** elágazásnál.

set showcmd A részlegesen begépelt parancsot jelzi a státuszsorban

set showmatch Ha épp zárójelet lezárunk, akkor egy pillanatra megmutatja a párját

set ignorecase Keresésnél nem számít a kis és nagybetû különbsége

set incsearch Inkrementális keresés

set ruler A státuszsorban, mindig mutatja a kurzor pozícióját, ill. hogy a szövegben hol állunk

set history=50 A vi-ban kiadott parancsok pufferét 50 parancsra visszamenõleg megõrzi

set ts=8 sts=4 sw=2 Indentelésnél 8 space-t cserél ki egy tab-ra *(tab space)*, egy tab leütésre 4 space-t tesz ki *(soft tab space)*, 2 space-el húzza beljebb a szöveget, ahol be kell húzni *(space width)*

set tw=0 Nem töri el a sorokat. Beállítható, hogy hány karakter széles legyen a szöveg *(text width)* szerkesztés közben, és ha szélesebbre nyúlik amit gépeltünk, azt eltöri a megadott szélességnek megfelelõen. Pl. ha levelezőből külső editornak használjuk, akkor érdemes egy **:set tw=76** paranccsal kezdeni.

8.7. További vim információk

A vimről további információkat a [4] **vimtutor** programból szerezhetünk, valamint a vimben van *online* súgó is, ami a **:help** parancsal érhető el. Ha a súgó hivatkozhat más sugó oldalakra is. Ezek |jelek| közt vannak. Ha az adott témát akarjuk inkább olvasni, akkor üssük le a CTRL-**]** -t. Ha vissza akarunk menni az elõzõ témára, akkor a CTRL-**t** billentyűkombinációt üssük le. Ha egy adott témát akarunk olvasni, akkor azt elérhetjük közvetlenül a **:help** *témanév* parancsal.

Ábrák jegyzéke

3.1. Egy példa partícionálásra 8

8.1. Váltás a vi módjai közt 44

Példák jegyzéke

(P)ATA lemezek eszközneveire példák, 7

A /tmp, mint sticky-bites könyvtár, 20

df parancsra példa, 11
du parancsra példa, 11

Egyéb példák behelyettesítésre a **bash**ban, 37

felhasználókezelő parancsokra példák, 6
fstab példa, 12

group fájl tartalmára példa, 4

head példa, 39

i-node tábla, 13

könyvtár jogosultságai, 19

ls -i, 15
ls parancs, 14

mount parancsra példa, 10

Named pipe példa, 17

Példa a free parancsra, 42
partíciós séma-példa, 8
passwd fájl tartalmára példa, 4

SCSI eszközök neveire példák, 8
sed használatra példa, 38
setgid-es könyvtárra példa, 20
sort parancs használatára példa, 41
Számlálásos ciklusra példa, 33

tail példa, 40
test parancsra példa, 36

uniq parancs használatára példa, 41

Változó behelyettesítésére példa, 36
vi alap példák, 44
vi regexp példák, 45

wc parancs használatára példa, 40
while ciklusra összetett példa, 34

Irodalomjegyzék

[1] http://hup.hu/ Hungarian Unix Portal

[2] Brian W. Kernigham, Rob Pike: A UNIX Operációs rendszer

[3] Orlando Unix Iskola http://www.cab.u-szeged.hu/local/doc/UNIX/orlando/bev.html

[4] Vi IMproved tutorial