1. Bejelentkezés, terminálok

Bejelentkezés

A Linux, mint minden UNIX-szerű operációs rendszer, többfelhasználós rendszer. A bejelentkezéshez így szükséges egy *azonosító* és egy hozzá tartozó *jelszó*. Az azonosító általában nem a felhasználó ember neve, hanem egy (általában a névből származtatott, esetleg a felhasználó fantáziája által kreált), kisbetűkből (esetleg számokból) álló, szóközöket nem tartalmazó karaktersorozat. Az egyes felhasználók a jelszavukat tetszőlegesen megválaszthatják és bármikos megváltoztathatják (azonban hangsúlyozottan célszerű bonyolult, kis- és nagybetűket és spéci karaktereket tartalmazó jelszó váltasztása).

Az egyes számítógépeken futó Linux rendszerek lehetnek eltérő disztribúciók, eltérő verziószámmal, azonban a gyakorlatban ez nem nagyon jelent semmilyen lényeges különbséget. Jelenleg az SZTE fizikus tanszékein a következő verziók találhatóak meg:

- Debian Woody (`rcrb`, `hydra`, 19-es labor gépei) illetve Debian Sarge (`oberon`, `titan`, `thetys`, `vboo`, `xleo`)
- Redhat különböző verziószámai (`pluto`, `ariel`, `triton`, a hp notebook)
- Mandrake (`mira`)

Grafikus felületek

Mindegyik rendszernek grafikus bejelentkezési felülete van, itt rögtön megadhatjuk az azonosítónkat és a jelszavunkat. Érvényes azonosító-jelszó pár megadása – a sikeres bejelentkezés – után a rendszer egy grafikus felhasználói felületet (más néven *ablakkezelő*t vagy *operációs környezet*et) indít el. (Ha esetleg console-os, azaz nem grafikus bejelentkező felülettel találkozunk, akkor a bejelentkezés után a `startx` parancs kiadása után kapjuk meg a grafikus kezelői felületet) Ezek a modern grafikus felületek eléggé sablonos felépítésűek, hasonlóak az M$-rendszerekhez: a képernyő alján kis sáv (úgynevezett *panel*) mindenféle ikonokkal, bal alsó sarokban felgördülő menü, a képernyő nagyrészén (más néven *munkaasztalon* vagy *workspace*-en) ikonok elszórva, az egér működik, az ikonokra rá*kattint*va elindulnak mindenféle programok, stb. Néhány lényeges dolog ezekkel a grafikus felületekkel kapcsolatban:

- a bal alsó sarokban elhelyezett menüből a gépre telepített programok gyakorlatilag mind elérhetők, továbbá, munkánk befejeztével kijelentkezni is innen tudunk (az *Logout*, *Exit* vagy *Kijelentkezés* menüpontok segítségével, ennek a menüpontnak az elnevezése disztribúció-függő);
- a munkaasztalon és a panelen található többi ikon a gyakrabban használt programok gyorsabb elindítására szolgálnak;
- a munkaasztal tulajdonságait (a rajta levő ikonok eloszlását, a hátteret, stb.) megváltoztathatjuk: a munkaasztal üres (ikonoktól mentes) területén a jobb egérgombbal való *kattint*ással lejön egy menü, itt a megfelelő elemek (pl. új ikon hozzáadássa, opciók, tulajdonságok megváltoztatása) kiválasztásával a kívánt beállítás megváltoztatható.

Terminálok

Most a gyakorlatban (így, Linux-alapok tanulása címszó alatt) a grafikus környezet által felkínált rengeteg program közül csak *egyetlen* fajta programot használunk, az úgynevezett *terminál-emulátor*okat. Ezt a programot a bal alsó sarokban levő menüből, a panelről, vagy a munkaasztalról egyaránt indíthatjuk (mivel egyébként is elég gyakran használt program, alapértelmezésben is mindenhova kitették). Többfajta terminál-emulátor létezik, ezek közül a leggyakrabban használt a `xterm` vagy a `konsole` elnevezésű program. Az előbbi ikonja általában egy kis fekete monitor, míg az utóbbié egy kék monitor, előtérben egy kagylóval (mint a „Shell”-embéma). A tanszéki linuxokon általában a `KDE` elnevezésű grafikus környezet az alapértelmezett, ennél a `konsole` program mindig elérhető, de sokkal elterjedtebb, és nem PC-alapú UNIX-rendszereken is mindig megtalálható az `xterm` program. Az egyszerűbb grafikus felületeknél (windowmaker, blackbox, fluxbox ... stb.) az egérrel való jobb kattintással jönnek elő a lehetőségek. Ezen felületek egyszerű megjelésüknek köszönhetően sokkal gyorsabbak, mint „csicsás” társaik.

A terminálon keresztül parancsokat adhatunk ki az operációs rendszernek. Fontos tudni, hogy terminálból a rendszer minden funkciója elérhető, és egyes létfontosságú beállítások csak a terminálon, bizonyos parancsok kiadásával érhetők el. Terminál minden UNIX/Linux rendszeren van, akkor is, ha nincs grafikus környezet és akkor is, ha egyik gépről a másikra bejelentkezünk.

A terminálon keresztül a parancsokat egymás után, folyamatosan adhatjuk ki, a parancssorokat `Enter`-rel kell zárni. A parancssor elején egy *prompt* található, általában rendszerfüggő, hogy hogy néz ki (sőt, mi magunk is megváltoztathatjuk majd, ha nem tetszik). Általában valami ilyesmi:

```
gaspara@titan:~$
```

vagy:

```
[public@mira public]$
```

A továbbiakban az egyszerűség kedvéért ezt a prompt-ot mindig egy dollár-jellel (`$`) rövidítjük, valamint ezzel utalunk arra, hogy a dollár-jel utáni rész egy *parancs*, amit a terminál parancssorába kell beírni.

2. Egy pár parancs

A parancsokat az úgynevezett *héj*-program, másnéven *shell* várja és dolgozza fel. A Linux rendszereken a legelterjedtebb a `bash` nevezetű shell, a továbbiakban erről lesz szó.

Néhány parancs segítségével a rendszerről kaphatunk mindenféle információt. A `man` *`parancs`* parancs segítségével a többi parancs *manuel*-jét, azaz bővebb leírását illetve további lehetőségeit olvashatjuk el.

A `date` parancs kiirja az aktuális időt és dátumot:

```
$ date
Thu Jan 26 15:34:32 CET 2005
```

(a dátum megformázása angol nyelvű környezetben kissé furcsa: hét napja, hónap, nap, idő, milyen időszámítás van érvényben – CEST: nyári, CET: téli közép-európai idő – és év).

Ilyen egyszerű parancsok még az `w`, `who`, `finger`, `uname`, `pwd`, `uptime` parancsok is . Ezek közül az első három a bejelentkezett felhasználókról ír ki mindenfeéle adatot:

```
$ w
3:39pm up 36 days, 4:48, 3 users, load average:  0.08, 0.08, 0.08
USER      TTY   FROM          LOGIN@  IDLE  JCPU  PCPU WHAT
gaspara   pts/0 160.114.18.23 3:34pm 0.00s 0.08s 0.01s w
$ who
gaspara   pts/0 Oct 23 15:34 (160.114.18.23)
$ finger
Login      Name            Tty   Idle Login Time Where
gaspara    Gaspar Andras   pts/0 -    Sat 15:34  160.114.18.23
```

Az `uname` parancs megmondja, mi a használt operációs rendszer neve, a `-a` *kapcsoló*val kiegészítve (`uname -a`) valamivel informatívabb: kiírja az oprendszeren kívül a gép nevét, a *kernel* verzióját (itt ez a `2.4.19-4GB`), hogy mikor fordították a kernelt, milyen processzorra (ez itten a `i686`), stb:

```
$ uname
Linux
$ uname -a
Linux titan 2.4.18-1-686-smp #1 Wed Apr 14 18:42:49 UTC 2004 i686 GNU/Linux
```

A `pwd` (print working directory) kiírja, melyik könyvtárban vagyunk éppen:

```
$ pwd
/home/gaspara
```

Az `uptime` megmondja, mióta van bekapcsolva a gép (plusz még pár egyebet is: a mostani időpontot, hányan vannak bejelentkezve és mennyire van leterhelve átlagosan a gép):

```
$ uptime
3:47pm up 36 days, 4:57, 1 user, load average:  0.17, 0.09, 0.08
```

A terminálból kilépni az `exit` vagy `logout` parancsokkal, vagy a `Ctrl+D` billentyűkombináció lenyomásával lehet. Grafikus felületek esetén használhatjuk továbbá az ablak (általában jobb-felső) sarkában található kis X-et is a terminál-ablak bezárására.

A shell egyszerre több parancsot is fel tud dolgozni. Ekkor az egymást követő parancsokat pontosvesszővel kell elválasztani:

```
$ date ; w
3:39pm up 36 days, 4:48, 3 users, load average:  0.08, 0.08, 0.08
USER      TTY   FROM          LOGIN@  IDLE  JCPU  PCPU WHAT
gaspara   pts/0 160.114.18.23 3:34pm 0.00s 0.08s 0.01s w
Sat Oct 23 15:34:32 CEST 2004
```

A terminálból más programok is elindíthatóak. Például ha a mozilla nevű böngészőt nem találjuk a menüben vagy a munkaasztalon, a terminálból is elindíthatjuk:

```
$ mozilla
```

Fontos tudni, hogy a parancssort addig nem kapjuk vissza, amíg az elindított program le nem fut. Egyes alkalmazások esetén (például a fentinél) ez kimondottan bosszantó lehet: ennek kikerülése végett a programot *háttérben* is el lehet indítani: az elindítandó program neve után egy & jelet kell tenni, majd enter:

```
$ mozilla &
[1] 1234
$
```

Ekkor láthatóan visszakapjuk a parancssort, a shell pedig tudatja velünk, hogy a programot valóban a háttérben indítottuk el (az [1] jelzi, hogy ez az első ilyen háttérben indított programja a shell-nek, az azt követő szám pedig az újonnan indított program *folyamat-azonosító*ja, bővebben erről később...).

Szintén fontos tudni, hogy akár simán, akár ílymódon (háttérben) indítottunk el egy programot, a shell még kötődik ehhez az újonnan elindított programhoz. Az első esetben ez nyilvánvaló (nem is kapjuk vissza a parancssort), a második esetben kevésbé, de egy pár váratlan hibaüzenet – amit ez a háttérben levő program néha produkál – megjelenhet a további, ezen shell-ben végzett munkánk során.

Ha a terminált leállítjuk, akkor az összes alóla, akár háttérben akár simán indított program futása is megszakad! Egy web-böngésző kiölése azáltal, hogy az indító terminál-ablak jobb-felső sarkában található X-et véletlenül lenyomtuk, elég bosszantó tud lenni...!

A shell és a háttérben indítandó program közötti kapcsolatot kicsit bonyolultabb indítási paranccsal szakíthatjuk meg:

```
$ mozilla </dev/null >&/dev/null &
[1] 1234
$
```

Ekkor a shell ugyanúgy kiírja a két azonosítószámot, de már nincs kötődése a két programnak (a shell-nek és a mozillának) egymáshoz. (Hogy pontosan mi ez a kötődés, arról részletesebben később lesz szó.)

3. Könyvtárszerkezet, file-rendszer

A Linux rendszereken a file-ok és a könyvtárak szerkezete a hagyományos, UNIX rendszereken megszokott struktúrát követi. A UNIX rendszereken nincsenek a DOS-os, Windows-os környezetben megszokott „*meghajtók*” (*drive*-ok) és a hozzájuk kapcsolódó betűk (`A:`, `C:`, ...). Minden file és könyvtár a főkönyvtár (*root directory*) alól érhető el. A főkönyvtár jele `/`, az egyes könyvtárakat egymástól szintén a `/` jel választja el (hasonlóan, mint DOS-alapú rendszerek esetén a \ jel).

A főkönyvtárban hagyományosan csak egy tucatnyi könyvtár található, file-ok nem nagyon vannak benne. A fontosabb könyvtárak és tartalmuk:

- `/boot`: a rendszer indításához és felállásához szükséges file-ok (pl. a rendszermag, a *kernel* is itt található).
- `/bin`: a rendszer működéséhez alapvetően szükséges, mindenki által használható programok (kb. 50-100 program, itt van például a `date`, `uname`, `pwd` program is).
- `/sbin`: a rendszer működéséhez alapvetően szükséges, de csak a rendszergazda által indítható programok.
- `/home`: a felhasználók személyes adatait tároló könyvtár, ebben több alkönyvtár van, melynek neve általában megegyezik az egyes felhasználók azonosítójával.
- `/cdrom`, `/floppy`, `/usb`: a „cserélhető” eszközökön (CD, floppy, USB-tároló v. digitális fényképezőgép) levő file-ok ezeken a könyvtárakon keresztük érhetők el (ezen könyvtárak helye annyira nem szabványos, de a legtöbb Linux rendszernél ezek a megszokottak. Sok disztribúciónál ezek a `/mnt` könyvtáron belül találhatóak.).
- `/etc`: a rendszer illetve az egyes programok konfigurálását tároló file-ok helye.
- `/usr`: felhasználói programok és adatok, amik nem létfontosságúak a rendszer felállásához és karbantartásához. Maga a könyvtár több alkönyvtárat tartalmaz (a felépítése hasonló a főkönyvtáréhoz).
- `/dev`: speciális eszközök, meghajtó-programok vezérlésere szolgáló file-ok.

A UNIX rendszeren minden fizikai adattároló és adattovábbitő eszköz (wincseszterek, hangkártyák, modemek, egér, soros és párhuzamos csatlakozók, stb.) speciális file-ok formájában érhetőek el. Ezen file-okat megnyithatjuk (feltéve ha van hozzá jogunk), adatokat írhatunk bele, adatokat olvashatunk ki vagy mindkettőt egyszerre. Az adatok sorsa az eszköz tulajdonságaitól függ. Egy hangkártya esetében (`/dev/audio`) a kiírt adatnak megfelelő hangminta fog megszólalni a hangszórokon, a beolvasott adat meg a hangkártyára csatlakoztatott mikrofonon keresztül bejövő hang digitalizált formája.

Létezik pár „szolgáltatást” nyújtó file is ebben a könyvtárban, ezekhez nem tartozik semmilyen fizikai eszköz, az adatokat maga az operációs rendszer kezeli le:

- `/dev/zero`: csak olvasható file, csupa 0 jön belőle, a végtelenségig.
- `/dev/null`: olvasható és irható file. Olvasás során egy 0 hosszúságú, üres fileként viselkedik, írás során pedig a beleírt adatot elnyeli. Sehova nem írja ki, elnyeli.
- `/dev/urandom`: csak olvasható file, véletlen byte-ok sorozata jön ki belőle, a végtelenségig.

FHS

Linux rendszereknél az egyes könyvtárak tartalma elég jól szabványosított. Ezeket a szabványokat a *Filesystem Hierarchy Standard* foglalja magában, ezek az interneten elérhetőek. Az egyik leginkább elterjedt szabvány a Debian FHS-e, ez a `http://www.debian.org/doc/packaging-manuals/fhs/` címen érhető el.

Jogosultságok

A felhasználóknak a gépen található file-okhoz csak korlátozott hozzáférésük van. Egy felhasználó általában csak a saját könyvtárában (pl. `/home/gaspara`) hozhat létre új file-okat és könyvtárakat, illetve meglévő file-okat csak onnan törölhet le. Az összes többi file-t általában tudja olvasni, a programokat tudja futtatni, azonban ezeket törölni vagy a tartalmukat módosítani már nem tudja. Néhány spéci file-t (pl. a titkosított hálózati kommunikáció kulcsait, más felhasználók levelezéseit tároló file-okat) olvasni sem lehet az átlagfelhasználóknak.

Kicsit bővebben: a UNIX rendszereken vannak felhasználók. Ezen felül vannak úgynevezett *csoportok* (*groups*). A rendszergazda állítja be, hogy melyik csoportba kik tartoznak. Egy file-nak vagy könyvtárnak a *tulajdonosa* az a felhasználó, aki a file-t vagy könyvtárat létrehozta. File-okról vagy könyvtárakról a tulajdonosaik rendelkezhetnek, ők állítják be, hogy kinek milyen hozzáférése legyen az adott objektumhoz. Minden filehoz a tulajdonos hozzárendelhet egy csoportot is (azon csoportok bármelyikét, amiben a tulajdonos benne van), az egyes jogokat (olvasás, írás, a programoknál a futtatás jogát) beállíthatja a csoportra vonatkozólag is. A jogok harmadik csoportja az „egyéb" felhasználókra vonatkozik. Egy átlagos file-t általában a tulajdonosa írhatja és olvashatja, a többiek (az adott csoport tagjai es az egyéb felhasználók) csak olvashatják. A tulajdonos dönthet úgy, hogy a csoport tagjainak, vagy akár az összes felhasználónak megadja az írás-jogot is, ennek azonban értelemszerűen vannak kockázatai.

Az egyes jogok (olvasás, írás, futtatás) hasonló módon értelmezettek könyvtárakra is: a könyvtárba való írás a file-ok létrehozását, az olvasás a könyvtár tartalmának megtekintését, a futtatás a könyvtárban levő file-okhoz való hozzáférés jogát jelenti.

Listázás

Egy adott könyvtár tartalmát az `ls` ('list') paranccsal listázhatjuk ki. Ha van argumentuma a parancsnak, akkor az ott megadott könyvtárat listázza ki, egyébként pedig az aktuális könyvtárat.

Az `ls` parancs alapértelmezésben csak a file-ok nevét írja ki:

```
$ pwd
/home/gaspara
$ ls
Desktop    bin         iraf   labor  porto        test.cat
astronomy  index.html  kepek  mail   public_html  work
$
```

Az `ls -l` paranccsal részletesebb listát is lekérhetünk, ekkor kilistázza a file- és könyvtárneveken kívül a jogosultságokat, az egyes file-ok tulajdonosát és csoportját, a file-ok méretét és az utolsó módosítás időpontját is:

```
$ ls -l
total 757
drwx------   3 gaspara    hallg        240 2004-09-22 11:33 Desktop
drwxr-xr-x   2 gaspara    hallg        368 2004-09-22 11:33 astronomy
drwxr-xr-x   4 gaspara    hallg        251 2004-05-11 23:13 bin
-rw-r--r--   1 gaspara    hallg        582 2004-10-21 14:48 index.html
drwxr-xr-x   3 gaspara    hallg         80 2004-11-19 11:19 iraf
drwxr-xr-x   4 gaspara    hallg         96 2003-08-05 12:56 kepek
drwxr-xr-x   2 gaspara    hallg         48 2003-12-28 17:32 labor
drwx------   2 gaspara    hallg        456 2003-10-01 12:16 mail
drwxr-xr-x   2 gaspara    hallg        156 2004-09-28 17:32 porto
drwxr-xr-x   4 gaspara    hallg        184 2004-01-22 22:10 public_html
-rwxr-xr-x   1 gaspara    hallg         29 2005-01-20 09:11 test.cat
drwxr-xr-x   2 gaspara    hallg         48 2005-01-15 17:32 work
```

A legelső sor (`total ...`) a könyvtárban található file-ok teljes lemezterület foglalása (kilobyte-okban). Az első oszlop a jogosultságokat adja meg, a második a tulajdonost, a harmadik a csoportot, a negyedik a file méretét (könyvtároknál a méret az adott könyvtárban található információk leírására szolgáló lemezterület mérete), az ötödik a file vagy könyvtár utolsó módosításának időpontja, az utolso pedig a file neve.

A jogosultság-mező tíz karaktert tartalmaz. Az első a típus (sima file-oknál ez `-`, könyvtáraknál `d` mint *directory*), a maradék kilenc pedig három hármas blokk, a kb. egy oldallal korábban leírtaknak megfelelően: egy csoport az `rwx` karaktereket tartalmazhatja, vagy az adott karakter helyén egy `-` jelet. Az `r` az olvasás (*read*), a `w` az írás (*write*) míg az `x` a futtatás (*execution*) jogát, a `-` a megfelelő jog hiányát jelöli. Az első 3-as blokk a tulajdonosra, a második a csoportra míg az utolsó az egyebekre vontatkozik.

4. File-ok létrehozása, kiiratása, másolása, átnevezése, törlése

Editorok

File-okat legegyszerűbben file-szerkesztő programokkal (*editor*okkal) hozhatunk létre és illetve módosíthatjuk tartalmukat. Egy pár ilyen, Linux alól elérhető editor: `mcedit`, `nedit`, `kedit`. Az `mcedit` szöveges editor (a terminál-ablakban jelenik meg), a másik kettő grafikus (elindításakor feljön egy új ablak). Mindegyik editort az `mcedit filenév` (`nedit filenév`, `kedit filenév`) paranccsal is lehet indítani, egyébként pedig elég kultúráltak és egyszerűen használhatóak (pl. menü-vezéreltek, működik az egeres kijelölés, szólnak, ha módosított file-t nem mentettük el kilépés előtt, stb.).

Az `mcedit` nagyon hasonlít a DOS-os idők Norton Commander-ének editorához (ill. a többi, hasonló editorhoz: Volkov Commander, stb.), a billentyűkombinációk is megegyeznek:

- `F2`: mentés;
- `F3`: kijelölés;
- `F5`, `F6`: kijelölt blokk másolása, mozgatása;
- `F7`: keresés;
- `F4`: átírás (search & replace);
- `F9`: menü – innen további funkciók érhetőek el;
- `F10`: kilépés.

File-ok kiiratása

Parancssorból a legegyszerűbben a `cat` paranccsal lehet file-okat kiiratni, ez a file tartalmát a parancssor után, ömlesztve írja ki. Ha a file túl hosszú (gyk. 25 sornál hosszabb), akkor már kigördül az eleje. Nagyobb file-okat a `more`, `less` vagy `mcview` programokkal lehet kényelmesebben megnézni. Mindegyik itt felsorolt parancsnak egy argumentuma kell hogy legyen: a megtekintendő file neve.

Másolás

File-okat másolni a `cp` paranccsal lehet. általában két argumentuma van: a már létező file neve, és az új file neve. Ha több argumentumot adunk meg és a legutolsó egy létező könyvtár, akkor a többi argumentumban megadott file-okat ebbe a létező könyvtárba, az eredetivel azonos néven másolja be. Ha sikeres volt a másolás, a `cp` program nem ír ki semmit a terminálra, (azaz a megadott file(ok) létez(nek) és a cél-könyvtárra van írás-jogunk). Ha bármi miatt nem sikerül a másolás, a `cp` program kiírja a hiba okát.

Egy pár példa:

```
$ cp regifile ujfile
$ cp a.txt b.txt /home/gaspara/work/
$ cp asdfgh.txt akarmi.txt
cp:  cannot stat ‘asdfgh.txt’:  No such file or directory
$ cp a.txt /usr/bin
cp:  cannot create regular file ‘/usr/bin/a.txt’:  Permission denied
$ cp /home/gaspara /home/gaspara/szoveg
cp:  omitting directory ‘/home/gaspara’
```

Az utolsó három példa esetén a `cp` program nem tudta végrehajtani a kívánt másolást (az első esetben nem létező file-t akartunk volna másolni, a másodikban a cél-könyvtárba nincs jogunk írni, így oda másolni sem, a harmadik esetben pedig az argumentumok kombinációja nem megfelelő – ti. a `cp`-nek vagy két file-nevet, vagy sok file-nevet és egy könyvtárat lehet megadni, itt meg két könyvtárat adtunk meg).

Átnevezés

File-okat az `mv` paranccsal lehet átnevezni, vagy egy másik (már létező) könyvtárba átmozgatni. Használata teljesen megegyezik a `cp` parancséval: vagy két file-nevet adunk meg (ekkor az első argumentumban átadott file-t átnevezi a második argumetumban megadott névre), vagy sok file-t és végül egy könyvtárat (ekkor a file-okat átmozgatja a megadott könyvtárba, az eredeti file-ok pedig megszűnnek).

Törlés

File-okat az `rm` paranccsal törölhetünk. Az argumentumban megadott file-okat rendre letörli. Csak óvatosan!

Az `mc` program

A DOS-os időkben elterjedt Norton Commandernek (és újabb változatainak: Volkov Commander, DOS Navigator, Windows/Total Commander, stb.) is van UNIX-os megfelelője: a Midnight Commander. Ezt a `mc` paranccsal hívhatjuk be. Az `mc` program használata teljesen megegyezik a fenti DOS-os/Windows-os programok használatával. Egy pár gyakori billentyűkombináció:

- `Tab`: váltás a két panel között;
- `Up`, `Dn`, `PgUp`, `PgDn`, `End`, `Home`: mozgás a file-listákon;
- `Enter`: belépés az adott könyvtárba, programok elindítása, kilépés a könyvtárból (ha a `..`-ra megyünk rá), stb.;
- `F5`: file-ok másolása;
- `F6`: file-ok átnevezése vagy mozgatása;
- `F7`: új könyvtár létrehozása;
- `F8`: törlés;
- `F4`: file-ok szerkesztése, (tkp. a `mcedit`-et hívja meg);
- `F9`: menü – innen további funkciók érhetőek el;
- `Insert`: file-ok kijelölése másoláshoz, mozgatáshoz vagy törléshez;
- `+`, `-`: file-ok kijelölése (adott maszk, pl. kiterjesztés szerint) vagy kijelölés megszűntetése;
- `*`: a kijelölés invertálása (azaz ha nincs kijelölve egy file sem, akkor kijelöli az összeset, ha mindegyik ki volt jelölve, akkor megszűnteti a kijelölést);
- `Ctrl+S`: gyorskeresés az aktuális könyvtárban;
- `ESC`, majd `Enter`: az aktuális file nevének lemásolása a parancssorba;
- `ESC`, majd `P`: az előző begépelt parancs(ok) visszahívása (mint a shell-ben a felfele-nyíl);
- `F10`: kilépés.

Ezeknek a funkcióknak a nagyrésze, illetve számos további funkció a menüből (`F9`) is elérhető.

5. A `bc` program

A `bc` program egy egyszerű számológép, mely a begépelt kifejezéseket feldolgozza, majd az eredményt kiírja a képernyőre. Inditásakor kiír egy pár soros copyright-üzenetet, de utána azonnal kezdhetjük a számolást:

```
$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type ‘warranty’.
6*7
42
4*(6+7)
52
3^50
717897987691852588770249
quit
$
```

A *dőlt/aláhúzott* utasítások az általunk begépelt sorokat jelentik. A `bc` alapértelmezésben a négy alapműveletet (`+`, `-`, `*`, `/`) és a hatványozást (`^`) ismeri. A program által felhasznált értékes helyiértékek száma viszont tetszőleges, gyakorlatilag csak a memória mérete szab neki korlátot. A pontosságot (hogy a program hány értékes tizedesjegyre számoljon) a `scale=...` paranccsal állíthatjuk be:

```
$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type ‘warranty’.
4/3
1
scale=10
4/3
1.3333333333
```

Látható, hogy a pontosság alapesetben 0 tizedesjegy, a program minden eredményt csonkol (a nála kisebb abszolútértékű egész felé kerekít).

A `bc` program pár elemi függvényt is ismer, ezek eléréséhez azonban a `bc -l` parancsot kell kiadnunk (a shellből). Ezek a függvények: szinusz (`s(.)`), koszinusz (`c(.)`), arkusztangens (`a(.)`), exponenciális (`e(.)`), természetes logaritmus (`l(.)`), gyökvonás (`sqrt(.)`) valamint a Bessel-függvények (`j(.,.)`).

Egy pár példa:

```
$ bc -l
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type ‘warranty’.
scale=40
s(1)
.8414709848078965066525023216302989996225
sqrt(2)
1.4142135623730950488016887242096980785696
4*a(1)
3.1415926535897932384626433832795028841968
e(1)
2.7182818284590452353602874713526624977572
```

(Mivel $\mathrm{tg}\left(\frac{\pi}{4}\right) = 1$, ezért $\pi = 4 \cdot \mathrm{arctg}(1)$.)

A `bc` programban definiálhatunk változókat is, az értékadás a `scale` használatával egyező módon történik, valamint egy sorban több `bc`-utasítást is kiadhatunk, ekkor ezeket pontosvesszővel (`;`) kell elválasztani:

```
$ bc -l
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
scale=30 ; pi=4*a(1)
pi
3.141592653589793238462643383276
s(pi)
.000000000000000000000000000003
c(pi)
-1.000000000000000000000000000000
```

A `bc` program az eddig felsoroltaknál azonban jóval többet tud, például saját függvényeket is definiálhatunk, ciklusokat és elágazásokat is lehet beletenni:

```
$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
define square ( q ) { return(q*q); }
for ( i=1 ; i<=6 ; i++ ) { square(i); }
1
4
9
16
25
36
```

A `bc` szintaxisa nagyon hasonló a C nyelvhez. További információt a `bc` *kézikönyvéből* (*manual*-jából) lehet kapni, ezt a shell-ből kiadott

```
$ man bc
```

parancccsal lehet lekérni.

6. A kimenet és a bemenet átirányítása

A `bc` program a billentyűzetről olvasta a bemenetét (a képleteket), a kimenetét (az eredményeket) pedig a képernyőre írta ki. A shell lehetővé teszi mind a kimenet, mind a bemenet átirányítását: a billentyűzet helyett a bemenetet lehet file-ból is beolvastatni, a kimenetet pedig lehet file-ba is iratni.

Az átirányításhoz (*redirection*) a „csibecsőr” karaktereket (> ill. <) kell használni, a kisebb-jel (<) a bemetet irányítja az azt követő file-ból, a nagyobb-jel (>) pedig a kimenetet irányítja az azt követő file-ba.

Egy példa: csináljunk egy editorral egy kis file-t, ami néhány képletet tartalmaz, majd ezt irányítsuk bele a `bc`-be:

```
$ mcedit keplet.dat
```

```
6*7
120+17
scale=30; 4*a(1)
```

```
$ bc -l <keplet.dat
42
137
3.141592653589793238462643383276
$
```

Látható, hogy a `bc` kilépett, miután az utolsó képletet is feldolgozta (nem kellett nekünk kézzel a `quit` paranccsal, vagy `Ctrl+D`-vel kilépni), valamint nem írta ki azt a pár soros copyright-üzenetet .

A kimenet is átirányítható, például egy `eredmeny.dat` nevű file-ba:

```
$ bc -l <keplet.dat >eredmeny.dat
$ cat eredmeny.dat
42
134
3.141592653589793238462643383276
$
```

Ekkor a `bc` egyátalán *semmit* nem ír ki, rögtön visszaadja a parancssort. Hogy valóban kiszámolta és az `eredmeny.dat` file-ba le is tárolta az eredményt, arról az `eredmeny.dat` file kilistázásával vagy egyéb módon történő megtekintésével (lásd: `less`, `more`, `mcview`) meg lehet győződni! (és érdemes is...)

Természetesen lehetőség van csak a kimenet átirányítására is. A `bc` program erre nem igazán látványos példa, jobb példa a `cat`: az eredetileg a képernyőre írja ki a file tartalmát, a kimenetet átirányítva viszont tudunk a `cp`-hez hasonlóan másolni:

```
$ cat file
valami szoveg...
$ cat ujfile
cat:  ujfile:  No such file or directory
$ cat file >ujfile
$ cat ujfile
valami szoveg...
```

Ezzel az átirányítással a `/dev` könyvtárban levő file-okat is kezelhetjük. A `/dev/urandom` fentiekhez hasonló, a `/dev/audio`-ba történő átirányításával például sistergő hangot adathatunk ki:

```
$ cat /dev/urandom >/dev/audio
```

A `/dev/urandom` file-nak sosincs vége, így a sistergést csak a `cat` program megszakításával lehet leállítani (pl. a `Ctrl+C` billentyűkombinációval).

7. Shell-változók és az echo parancs

A bc programhoz hasonóan a shell-ben is definiálhatunk változókat, az értékadás teljesen hasonó, mint a bc esetében:

```
$ valtozo=ertek
$
```

Az értékadás során a shell nem ír ki semmit, rögtön visszakapjuk a parancssort. Fontos, hogy az egyenlőségjel elé és mögé egyaránt nem szabad szóközt rakni! A bc matematikai program, ott az egyes változókba számokat helyezhettünk, a shell-változókban gyakorlatilag bármilyen karaktersorozatot letárolhatunk.

A képernyőre szöveget vagy változók értékeit az echo paranccsal irathatjuk ki. Ez a parancs normál szöveg esetén semmi „extrát” nem csinál, csak „visszhangozza” a beírt szöveget:

```
$ echo qqriq torokcsaszar
qqriq torokcsaszar
$
```

Változókat a nevük elé írt dollár jellel irathatunk ki:

```
$ echo $valtozo
ertek
$
```

Ha az echo-val kiiratandó szöveg „csúnya” karaktereket tartalmaz (amiket a shell valamilyen módon feldolgoz: *, (,), [,], ?, {, }, #, ~, |, ;, <, >, ‘, %, &), a szöveget feltétlenül rakjuk dupla idézőjelek ("...") közé. A többi karaktert (betűk, számok, +, -, _, @, ,, :, ., /) minden további nélkük kiirathatjuk, akár idézőjelekkel, akár anélkül. Ha a szóközök számát is komolyan akarjuk vetetni, akkor is tegyük idézőjelek közé a szöveget (különben akárhány szóközt is írtunk a parancssorba az egyes szavak közé, az echo csak egyet fog tenni).

Ha a dollár-jel hatása alól is meg akarunk szabadulni (ti. ez a változók neve előtti karakter), akkor a szöveget tegyük szimpla idézőjelek ('...') közé. Ha nincs kedvünk idézőjelbe rakni ezeket a szövegeket, vagy magát az idézőjel-karaktert vagy a dollár jelet szeretnénk viszontlátni, a megfelelő csńya karakter elé tegyünk egy backslash-karaktert (\). Magát a backslash-t két backslash-sel irathatjuk ki.

Jó bonyolult ez az egész de hamar meg lehet szokni. . . egy pár példa:

```
$ echo szoveg
szoveg
$ echo a valtozo erteke:  $valtozo
a valtozo erteke:  ertek
$ echo "3*6"
3*6
$ echo "idezojelben valtozo:  $valtozo"
idezojelben valtozo:  ertek
$ echo 'szimpla idezojelben:  $valtozo'
szimpla idezojelben:  $valtozo
$ echo \\
\
$ echo \*, \#
*, #
$ echo elso    szo,       masodik          szo
elso szo, masodik szo
$ echo "elso    szo,       masodik          szo"
elso    szo,       masodik          szo
```

Az echo kimenetét is át lehet irányítani egy file-ba, ezzel például nagyon gyorsan kreálhatunk egysoros kis file-okat:

```
$ echo "valami szoveg" >filenev.txt
$ cat filenev.txt
valami szoveg
```

A változók értékeit (így, dollár-jellel hivatkozva) nem csak az `echo`-nak, hanem bármely más parancsnak is átadhatjuk. Például:

```
$ echo "6*7" | bc >valasz.dat
$ echo "valami szoveg" >filenev.txt
$ cat valasz.dat
42
$ cat filenev.txt
valami szoveg
$ file=valasz.dat
$ cat $file
42
$ file=filenev.txt
$ cat $file
valami szoveg
```

Itten a kiiratást az utolsó két esetben a `cat $file` parancs végezte el, azonban a két eset során a `$file` változó értéke más-más volt, így a `cat` más-más file tartalmát irta ki.

Változó értéke nem csak előre megadott karakterlánc lehet (mint ahogy az eddigi példákban), hanem egy program kimenetét is letárolhatjuk változóban. Azaz a program, ahelyett, hogy a képernyőre, vagy egy file-ba (a > után megadott file-ba) írná a tartalmát, sehova nem írja ki, hanem elhelyezi egy file-ban. A parancsot, aminek a kimenetét tárolni szeretnénk, *ferde* idézőjelek közé, vagy `$(...)`-be kell írni:

```
$ cat keplet.dat
6*7
$ bc <keplet.dat
42
$ valasz=`bc <keplet.dat`        # itt adunk értéket a változónak
$ echo $valasz                   # ellenőrzésképp kiiratjuk echo-val az értékét
42
```

Az `echo` a kiírt szöveg után mindig tesz egy sortörést. Ezt el lehet kerülni a `-n` kapcsoló megadásával.

8. Pipe-ok

A Linuxban alapesetben nincs olyan program, ami a prancssorba beírt képletet kiszámolná. Az `echo` és a `bc` összekombinálásával azonban ezt el tudjuk érni, csak valahogy rá kell venni a rendszert, hogy ami az `echo`-ból kijön, az ne a képernyőre és ne egy file-ba menjen, hanem a `bc`-program bementébe, amivel szintén tudatni kell, hogy most ne a billentyűzetről és ne is egy file-ból, hanem egy másik program kimenetéről olvassan.

Ezt az összeköttetést a shell-ben a pipe-karakterrel (`|`, egy függőleges vonal) érhetjük el. A pipe-jel előtti parancs kimenetét a shell átirányítja a pipe-jel utáni parancs bemenetébe:

```
$ echo "6*7" | bc
42
```

(A `6*7`-et itten idézőjelbe is kell tenni, mivel a szöveg tartalmaz csúnya karaktert, lásd a felsorolást egy szakasszal korábbról).

Természetesen a pipe-t a file-ba való átirányítással is kombinálhatjuk:

```
$ echo "6*7" | bc >valasz.dat
$ cat valasz.dat
42
```

A pipe-okat és a pipe-al összekapcsolt parancs-sorozatokat (az ún. pipeline-okat) előszeretettel alkalmazzák a UNIX-rendszerek programozása és használata során. Azaz lesz bőven példa az alkalmazásukra...

9. Egyszerű file-manipuláló parancsok és egyéb programok

9.1. A head és a tail parancs

A `head` parancs kiirja az argumentumában megadott file első 10 sorát. Ha nincs megadva file-név, akkor a billentyűzetről olvas (amit meg a `<`, `|` karakterekkel át lehet irányítani, hogy mégiscsak file-ból illetve egy másik program kimenetéről olvasson). Ha nem pont 10 sort akarunk kiiratni, a sork számát a `-n` kapcsolóval adhatjuk meg:

```
$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
$ head -n 2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
$ cat /etc/passwd | head -n 4
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
```

(Az `/etc/passwd` file tárolja a gépen az egyes felhasználók adatait.)
Hasonlóan működik a `tail` parancs is, csak nem az első, hanem az utolsó sorokat írja ki:

```
$ tail -n 5 /etc/passwd
tucsok:x:1208:110:Egerhazi Laszlo,,,:/ext/home/tucsok:/bin/bash
tcsako:x:1209:110:Csako Tamas,,,:/ext/home/tcsako:/bin/bash
elftamke.tag:x:1210:120::/ext/home/elftamke.tag:/bin/bash
messagebus:x:105:112::/var/run/dbus:/bin/false
apal:x:1211:110::/ext/home/apal:/bin/bash
+::::::
```

9.2. A `seq` parancs

A `seq` utasítással egy számtani sorozatot irathatunk ki a képernyőre. A `seq`-t háromféle módon hívhatjuk:

- `seq MAX`: ekkor a `seq` a számokat 1 és `MAX` között, egyesével írja ki;
- `seq MIN MAX`: ekkor a `seq` a számokat `MIN` és `MAX` között egyesével írja ki. Ha mégis az első argumentum a nagyobb, akkor csökkenő sorrendben történik a kiírás;
- `seq MIN LEPES MAX`: ekkor a `seq` `MIN` és `MAX` között, `LEPES`-enként növelve írja ki a számokat. Ha az első paraméter nagyobb, mint a harmadik, akkor a `LEPES` csak negatív lehet. Ellenkező esetben a program hibaüzenettel áll le.

Néhány példa:

```
$ seq 4
1
2
3
4
$ seq 3 6
3
4
5
6
$ seq 10 -1 7
10
9
8
7
$ seq 13 2 19
13
15
17
19
$ seq 19 2 13
$
$ seq 19 -2 13
19
17
15
13
```

A `seq` kimenetét is elhelyezhetjük egy változóban:

```
$ lista=`seq 10 2 18`
$ echo $lista
10 12 14 16 18
$ echo "$lista"
10
12
14
16
18
```

Itt is látható, hogy mi a különbség az idézőjelbe tett változó, és az idézőjel nélkül hivatkozott változó között! A ``lista=`seq 10 2 18` `` utasítás szórul szóra megőrizte a `seq` kimenetét, azonban ha egyszerűen, a változó nevének kiírásával hivatkozzuk meg, a shell átalakítja az újsorokat és a felesleges szóközöket *egyetlen* szóközzé! Hangsúlyozzuk, hogy összesen 3 fajta idézőjel létezik a shell-ben: ' (magyar billentyűn a Shift+1, angolon az enter mellett), ` (magyar billentyűn az altGr + 7, angolon az 1 mellett balra) illetve a " (magyar billentyűn a Shift + 2, angolon a Shift + Enter melletti billentyű).

9.3. A `wc` parancs

A `wc` parancs az argumentumaként megadott file sorait, szavait és karaktereit számolja össze. Ha nem adunk meg egy file-t sem argumentumnak, akkor a billentyűzetről (avagy pipe-ból, vagy <-vel átirányított file-ról) olvassa a leszámlálandó adathalmazt. A `wc` kimenete 3 szám, és – ha az előbbi módon file(ok)at számoltatunk – a file neve:

```
$ cat akarmi.txt
elso sor
ez a masodik sor
ez meg a harmadik !
$ wc akarmi.txt
      3      11      46 akarmi.txt
$ cat akarmi.txt | wc
      3      11      46
```

Ha csak a sorokat, szavakat vagy a karaktereket szeretnénk leszámlálni, a `wc`-t a `-l` (*lines*), `-w` (*words*) vagy a `-c` (*characters*) kapcsolóval kell meghívni:

```
$ cat akarmi.txt
elso sor
ez a masodik sor
ez meg a harmadik !
$ wc -l akarmi.txt
      3 akarmi.txt
$ cat akarmi.txt | wc -w
     11
$ wc -c <akarmi.txt
     46
```

9.4. A `date` parancs

A `date` parancs alpértelmezésben a pillanatnyi dátumot írja ki, az angolszász területeken megszokott (nálunk kissé idétlennek tűnő) formátumban:

```
$ date
Sun Dec 12 22:59:03 CET 2004
```

A kimeneti dátumot azonban meg tudjuk formázni, hogy a kedvünknek megfelelő formátumot alakítsuk ki. A formátum-leírást egy `+` karakterrel kell kezdeni (innen tudja a `date`, hogy mostan egy saját formátum szerint kell kiírnia a dátumot) , majd utána meg lehet adni a formátumot. A formátum tetszőleges szöveg lehet, azt normálisan kiírja a `date`, leszámítva a `%` jeleket. Ezután egy betűt kell megadni, a kiíratandó dátum-elem ettől a betűtől függ. Magát a százalék-jelet – ha formátumunkban erre van szükség – egy dupla `%%`-kal írathatjuk ki.

Egy pár példa:

```
$ date +%Y.%M.%d
2004.15.12
$ date +%H:%M:%S
23:15:53
$ date "+1970.  jan.  elseje eltelt masodpercek szama:  %s"
1970.  jan.  elseje eltelt masodpercek szama:  1106846672
$ date +%Z
CET
$ date +%z
+0100
```

Ez utóbbi kettő az aktuális időzónát mutatja (CET: Central European Time, azaz Közép-európai idő, KözEI, ami a greenwich-i világidőnél egy órával több).

A date-val nem csak az aktuális dátumot írathatjuk ki, hanem tetszőleges, más dátumot is. Ezt a tetszőleges dátumot a -d kapcsoló után kell írni, Y-M-D h:m:s formátumban (Y, M, D: év, hó, nap; h, m, s: óra, perc, másodperc). Amennyiben időpontot (h:m:s) nem adunk meg, az időpont automatikusan éjfél lesz. Ha megadunk időpontot is, akkor az egész dátumot idézőjelbe kell tenni (a D és a h közötti szóköz miatt). Például:

```
$ date -d 2001-1-1
Mon Jan 1 00:00:00 CET 2001
$ date -d "2001-1-1 11:22:33"
Mon Jan 1 11:22:33 CET 2001
$ date -d 2003-3-14 +%Z
CET
$ date -d 2003-5-14 +%Z
CEST
$
```

Természetesen a -d-vel megadott dátumot +...-tal meg is formázhatjuk. Ez utóbbira jó példa a fenti utolsó két parancs. Segítségével könnyen el lehet dönteni, hogy egy tetszőleges időpontban (pl. 2003. március 14-én vagy május 14-én) milyen időszámítás van érvényben (CET: Central European Time, KözEI; CEST: Central European Summer Time, közép-európai nyári időszámítás).

A %-ot követő egyes betűk jelentése:

%a	a hét napja (ang. röv)	%A	a hét napja (ang.)
%b	hónap (ang. röv)	%B	hónap (ang.)
%c	a teljes dátum		
%d	a hónap napja (1...31)	%D	dátum (hó/nap/év)
%e	a hónap napja (1...31)		
%h	lásd: %b	%H	óra (00...23)
		%I	óra (01...12)
%j	az év napja (001...366)		
%k	óra (0...23)		
%l	óra (1...12)		
%m	hónap (01...12)	%M	perc (00...59)
%n	újsor		
%p	AM vagy PM		
%r	idő (óra:perc:mp AM/PM)		
%s	időbélyeg	%S	másodperc (00...60)
%t	tabulátor	%T	idő (óra:perc:mp)
		%U	hét sorszáma (00...53, vasárnaptól)
		%V	hét sorszáma (01...53, hétfőtől)
%w	hét napja (0...6, 0: vasárnap)	%W	hét sorszáma (00...53, hétfőtől)
%x	helyi dátum-forma	%X	helyi idő-forma
%y	év utolsó két jegye	%Y	év
%z	időzóna (óra/perc)	%Z	zóna-kód

10. Internet

10.1. A `lynx` program

A `lynx` egy karakteres web-böngésző program. Használata:

```
$ lynx szerver.domain/konyvtar/alkonyvtar
```

azaz a parancs neve után a megnyitandó webcímet kell csak megadni. A megjelenítés csak karakteres, így képek illetve színek nem jelennek meg (helyettük maximum a kép neve). Navigálni az egyes mezők illetve oldalak között a szokásos billentyűkombinációkkal (nyilak, `PgUp`, `PgDn`, ...) lehet, a linkeket az `Enter`-rel lehet aktiválni.

A `lynx` parancs az interaktív használaton túl képes megadott oldalakat letölteni, és azok *forrását* képernyőre vagy file-ba kiiratni:

```
$ lynx --dump szerver.domain/konyvtar/alkonyvtar/file.ext
...
$ lynx --dump szerver.domain/konyvtar/alkonyvtar/file.ext >file.ext
```

10.2. A `wget` parancs

A `wget` parancs segítségével a `lynx` nem-interaktív módjához hasonlóan file-okat tölthetünk le és tárolhatunk le file-okban vagy irathatjuk ki a képernyőre. A `wget` inkább a letöltés folyamatára koncentrál, így alapesetben látható, hogy mi történik: melyik szerverre, IP-számra csatlakoztunk, és hogy a letöltés pillanatnyilag hol tart, mennyit töltött már le, és mennyi ideig tart még a letöltés...

A `wget` program kimenetét más file-ba vagy a képernyőre a `-O` (Output) kapcsolóval irányíthatjuk át, a letöltés közben megjelenő információkat pedig a `-q` (quiet) kapcsolóval lehet eltűnetetni:

```
$ ls -l *.html
$ wget astro.u-szeged.hu/index.html
--22:19:50-- http://astro.u-szeged.hu/index.html
           => `index.html'
Resolving astro.u-szeged.hu...  160.114.34.107
Connecting to astro.u-szeged.hu[160.114.34.107]:80...  connected.
HTTP request sent, awaiting response...  200 OK
Length:  501 [text/html]

100%[====================================>] 501         --.--K/s

22:19:50 (1.82 MB/s) - `index.html' saved [501/501]
$ ls -l *.html
-rw-r--r--    1 gaspara  hallg         501 Jan 21 21:49 index.html
$ wget -q astro.u-szeged.hu/index.html -O astro-index.html
$ ls -l *.html
-rw-r--r--    1 gaspara  hallg         501 Jan 21 21:49 index.html
-rw-r--r--    1 gaspara  hallg         501 Jan 21 21:51 astro-index.html
$ head astro-index.html
<html>
<head>
<title>Szegedi Csillagvizsgáló</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</head>

<frameset cols="120,497*" frameborder="NO" border="0" framespacing="0" rows="*">
<frame name="bal" scrolling="NO" src="bal.html" frameborder="NO" noresize
marginwidth="1" bordercolor="#006666">
<frame name="main" src="main_hu.html" frameborder="NO" bordercolor="#006666">
</frameset>
$
```

10.3. A `socket` parancs

Hálózati kapcsolatokat parancssori szinten a `socket` parancs segítségével kezdeményezhetünk illetve fogadhatunk.

Kapcsolat kezdeményezésekor (*connect*) a `socket` argumentumának meg kell adni azt, hogy melyik géphez hanyas porton szeretnénk csatlakozni. Ha a csatlakozás sikeres, a program a frissiben létrejött új hálozati kapcsolathoz hozzácsatolja a terminál bemenetét (a billentyűzetet, illetve `<`-rel való átirányítással egy file-t) és a kimenetét (a képernyőt, `>`-vel átirányítva egy file-t, vagy `|`-pal egy másik program bemenetét). Azaz, ami a másik géptől/programtól jön a hálózaton keresztül, azt kiírja a megfelelő kimenetre. Amit meg a bementről beolvas, azt a hálózaton keresztül továbbküldi a másik gépnek/programnak. Ha az adott gépen a megadott porton nem figyel egyetlen program sem, akkor nem is lehet kapcsolatot kezdeményezni és ezt a `socket` parancs tudatja.

Kapcsolat fogadásakor (*accept*) meg kell mondanunk, hogy melyik porton szeretnénk figyelni (*listen*) a bejövő kapcsolatokat. Ha ez a port már foglalt (egy már futó program, szerver már el kezdett figyelni ezen a porton), vagy nincs jogosultságunk az adott porton figyelni (felhasználóként csak az 1024, vagy annál nagyobb portok állnak rendelkezésünkre, rendszergazdaként azonban minden porton figyelhetünk), akkor a `socket` parancs hibaüzenettel leáll:

```
$ socket -s 137
socket:  server socket:  Permission denied
$ socket -s 6010
socket:  server socket:  Address already in use
$ socket -s 2233
[...]
```

Kapcsolat fogadása esetén a `socket` parancsot a `-s` kapcsolóval kell meghívni, majd a kapcsolatkezdeményezéssel megegyező módon, egy port-számot is meg kell adni:

`user@gep1: $ socket -s 2233`	`user@gep2: $ socket gep1 2233`
`szoveg`	*`szoveg`*
`masodik sor`	*`masodik sor`*
`ezt a masik kuldte`	`ezt a masik kuldte`
`ezt is`	`ezt is`
`ezt nem`	*`ezt nem`*

A `socket` parancs segítségével már meglevő a saját gépünkön vagy más gépen futó szolgáltatáshoz is hozzá lehet kapcsolódni.

Ha egy specifikus szolgáltatáshoz kapcsolódunk, akkor a kommunikáció során (azaz az elküldendő adatok megformázásakor és a kapott adatok értelmezésekor) a szolgáltatás által deklarált adat-formátumot, a szolgáltatás *protokoll*ját kell használni. Amennyiben a küldött adat nem a megfelelő formátumú, a szolgáltatást végző program visszajelez, vagy adott esetben a kapcsolatot is megszakíthatja.

A leggyakrabban használt, könnyen értelmezhető protokollok közé tartozik az e-mailek küldésekor és fogadásakor használt Simple Mail Transfer Protocol (SMTP), illetve a webezés során használt HyperText Transfer Protocol (HTTP). Egy adott gépen (szerveren) általában csak egy-egy program fut, ami ezeket a szolgáltatásokat végzi. Az SMTP a 25-ös, a HTTP a 80-as protot használja.

Egy pár példa:

- E-mail küldése. Egy e-mail küldése során egy szolgáltatóhoz (mail-server-hez) kell csatlakozni. A SMTP szerint „köszönni" (`HELO`) kell, majd meg kell mondani hogy kitől küldünk levelet (`MAIL FROM`), kinek (`RCPT TO`), majd magát a levéltörzset kell összeállítani (`DATA`). A folyamat végén, ha az összes levelet elküldtük, a kapcsolatot le kell bontani. A kapcsolat lebontására a kliens (mi, akik a levelet küldjük) ad egy parancsot (`QUIT`), amire a szerver megszakítja a kapcsolatot:

```
$ socket orion.fwall.u-szeged.hu 25
220 orion.fwall.u-szeged.hu ESMTP Exim 4.34 Thu, 27 Jan 2005 23:42:37 +0100
HELO orion.fwall.u-szeged.hu
250 orion.fwall.u-szeged.hu Hello titan.physx.u-szeged.hu [160.114.34.107]
MAIL FROM:<user@gep.u-szeged.hu>
250 Ok
RCPT TO:<valaki@freemail.hu>
250 Ok
DATA 354 End data with <CR><LF>.<CR><LF>
From:  <user@gep.u-szeged.hu>
Subject:  proba
To:  <valaki@freemail.hu>

leveltorzs!

.
250 Ok:  queued as 02D1626996D
QUIT
221 Bye
$
```

Az e-mail kiszolgálók általában csak egy adott címtarotmányon belülről jövő kapcsolatok esetén hajlandóak leveleket továbbítani külső e-mail címekre.

- Weblap letöltése. Egy adott weblap (pl. `http://astro.u-szeged.hu/index.html`) letöltéséhez a címben megjelölt szerverhez (itt `astro.u-szeged.hu`) kell csatlakozni a 80-as porton keresztül. A HTTP viszonylag egyszerű, gyakorlatilag a csatlakozás után meg lehet adni a letöltendő file (itt `/index.html`) nevét, majd a szerver visszaküldi a kívánt file-t és megszakítja a kapcsolatot:

```
$ socket astro.u-szeged.hu 80
GET /~szatmary/index.html
<html>
<head>
<title>Szegedi Csillagvizsgáló</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</head>

<frameset cols="120,497*" frameborder="NO" border="0" framespacing="0" rows="*">
<frame name="bal" scrolling="NO" src="bal.html" frameborder="NO" noresize
marginwidth="1" bordercolor="#006666">
<frame name="main" src="main_hu.html" frameborder="NO" bordercolor="#006666">
</frameset>
<noframes><body bgcolor="#FFFFFF">

</body></noframes>
</html>
$
```

Néhány szerver esetén nem ennyire egyszerű a `GET` parancs összeállítása, illetve pár egyéb információt is át kell adni (milyen böngészőnk van, a szerver-nevet meg kell ismételni, stb.). A legtöbb web-kiszolgáló azonban ilyen egyszerű parancsokra is már szépen reagál.

11. Ciklusok

Eddig az = egyenlőségjellel adtunk értéket a változóknak. A shell egyik hasznos szolgáltatása a ciklusok kezelése: segítségével el tudjuk érni, hogy egy adott blokkon (a *ciklusmagon*) belül egy általunk megadott változó az értékeit rendre egy halmazból vegye fel. A shell gondoskodik arról, hogy a ciklusmagban levő utasítások annyiszor lefussanak, mint amennyi eleme van ennek a halmaznak.

A ciklusszervező utasítás a `for`, a halmazt az `in` parancs után, pontosvesszővel lezárva kell megadni. A ciklusmagot a pontosvesszőt követően, a `do` és a `done` parancsok közé kell zárni. Maga a halmaz lehet helyben felsorolt halmaz, változó, több változó, vagy programkimenet is (ekkor a programot, az értékadásnál hasonlóan, ferde idézőjelek, vagy `$(...)` közé kell írni). Fontos tudni, hogy egy idézőjelen belüli blokkot, akárhány szóköz is van benne, egyetlen(!) halmazelemnek tekint a shell.

Egy pár példa az egyszerű ciklusokra:

```
$ for valtozo in A bb cc ; do echo ciklus:  $valtozo ; done
ciklus:  A
ciklus:  bb
ciklus:  cc
$ for valtozo in "A bb" cc ; do echo ciklus:  $valtozo ; done
ciklus:  A bb
ciklus:  cc
```

Egy bonyolultabb ciklus: a halmaz legyen az 1 és 10 közötti számok halmaza (ezt a `seq`-val generáljuk le), a ciklusmagban pedig számoljuk ki a változó négyzetét a `bc` program segítségével:

```
$ for i in `seq 10` ; do echo "$i*$i" | bc ; done
1
4
9
16
25
36
49
64
81
100
```

A halmazt a szokásos `*`-os keresési mitának megfelelően is megadhatjuk. Például, a

```
for szovegfile in *.txt ; do ...  ; done
```

ciklus esetén a `$szovegfile` változó a `txt` kiterjesztésű fileokon fog végigfutni. Itten vigyázni kell. Ha a `*.txt`-t idézőjelbe tesszük, a `*` elveszti ezt a spéci tulajdonságát és normál karakterként viselkedik. Hasonlóan, ha a shell nem talál egyetlen `txt` kiterjesztésű file-t sem, akkor is a `*.txt` karakterláncot fogja felvenni a ciklusváltozó (és a ciklusmag csak egyszer fut le). Ez kellemetlen tud lenni.

A ciklusmag lehet utasítások halmaza (ekkor az egyes utasításokat pontosvesszővel kell elválasztani), az egyes utasítások pedig lehetnek összetettek (azaz tartalmazhatnak pipe-okat, átirányításokat, változóértékadásokat, sőt, ciklusokat egymásba is lehet ágyazni).

12. A `bash` saját számológépe

A shell (mármint a `bash`) képes egyszerűbb műveletek elvégzésére is, így nem kell minden apróság kiszámítására a `bc` programot használni. A shell az összes, `$((...))` alakú kifejezésben a belső részt, mint aritmetikai kifejezést tekinti és kiszámolja:

```
$ echo $((6*7))
42
$ echo $((2*(4+7)-6))
16
$ i=13 ; echo $(($i*$i))
169
$ echo $((42%9))
6
$
```

Az aritmetikai kifejezésekben a négy alapműveletet, zárójelezést és a maradékképzés operátorát (`%`, lásd utolsó példa) használhatjuk. Az egyes számok helyett változókat is használhatunk, feltéve, hogy azok értéke szintén egy szám. A `bc`-vel szemben ez egy egyszerű számológép, azaz

- csak egész számokkal tud műveleteket végezni, illetve
- a pontossága véges, $\pm 2 \cdot 10^9$ (gyk. $\pm(2^{31}-1)$) közötti számokkal tud dolgozni.

Ezen hátrányok mellett van egy nagy előnye: sokkal gyorsabb, mintha a `bc`-t használnánk.

Annak ellenére, hogy ez a számológép nem ismeri a törteket, két egész számot leoszthatunk egymással, akkor is, ha nem egymásnak az osztói. Ebben az esetben a számológép a kapott eredményt *csonkolja*, azaz elhagyja a tizedespont utáni részeit és egy egész számot ad vissza. A csonkolás pozitív számok esetében a *lefelé kerekítésnek*, negatív számok esetén pedig a *felfelé kerekítésnek* felel meg:

```
$ echo $((12/9))
1
$ echo $((26/9))
2
$ echo $(((-12)/9))
-1
$ echo $(((-26)/9))
-2
```

Amennyiben a zárójelben megadott kifejezés nem szabályos (szintakikai hibát, tizedesszámokat(!) vagy betűket tartalmaz), a shell hibaüzenetet ír ki, és az a paracs, amiben ez a hibás `$((...))` kifejezés szerepel, nem is fut le.

13. Feltételek: elágazások és feltételes ciklusok

13.1. A `while` ciklus

A `for` ciklus addig futott, míg a paraméterként megadott változó végig nem járta a szintén paraméterként megadott halmaz összes elemét. Lehetőség van úgynevezett *feltételes* ciklus készítésére is. Ezesetben a ciklusmag addig fut, amíg a megadott feltétel igaz. Ha már a ciklusba való belépéskor hamis a feltétel, akkor a ciklus egyszer sem fut le. Ha nem gondoskodunk arról, hogy a feltétel egyszercsak hamissá váljon, a ciklus sosem fog leállni (vagyis, csak a megfelelő megszakító parancsok – ld. később – vagy billentyűkombinációk – pl. `Ctrl+C` – hatására lehet leállítani).

Ezt a feltételes ciklust a `while` ciklusszervező utasítással lehet létrehozni. A formátuma:

```
$ while [ feltétel ] ; do ...; done
```

A ciklusmag – a `for`-hoz hasonlóan a `do` és a `done` közötti parancs(sorozat) – addig fut le, amíg a *`feltétel`* igaz. A feltétel többféle lehet:

- `"string1" == "string2"`: ez igaz, ha a két karakterlánc megegyezik. Az egyes karakterláncokban használhatunk változókat is (szokásosan, dollár-jelekkel), valamint ha a karakterláncok csúnya karaktereket tartalmaznak, feltétlenül idézőjelbe kell tenni azokat, egyébként nem muszáj.
- `"string1" != "string2"`: ez a feltétel igaz, ha a két karakterlánc nem egyezik meg.
- `"string1" \> "string2"` illetve `"string1" \< "string2"`: ezek a feltételek rendre akkor igazak, ha az első karakterlánc később illetve előbb található meg az ABC-ben (a szokásos lexikális rendezés szerint), mint a második karakterlánc. A `<` és `>` karakterek eredetileg ugye az átirányításra szolgáltak, itt tehát el kell vonatkoztatni ettől a jelentésüktől. Azaz vagy dupla idézőjelbe kell tenni, vagy egy backslash (\) karaktert kell tenni elé (lásd még az `echo`-nál írtakat).
- `-e "file"`: igaz, ha a `file` mint könyvtárbejegyzés létezik. Azaz a `file` egy létező file, könyvtár, szimbolikus link, vagy speciális file neve.
- `-f "file"`, `-d "könyvtár"`: igaz, ha a `file` mint normális adatfile létezik, illetve igaz, ha a `könyvtár` mint normális könyvtár létezik.
- `A -eq B`, `A -ne B`, `A -gt B`, `A -ge B`, `A -lt B`, `A -le B`: ezek a feltételek akkor igazak, ha az `A` és `B` *egész* számok között rendre az egyenlőség (equal to), nem egyenlőség (not equal), nagyobb (greater than), nagyobb vagy egyenlő (greater or equal), kisebb (less than) illetve kisebb vagy egyenlő (less or equal) relációk teljesülnek. Ezekkel az összehasonításokkal csak egész számok hasonlíthatóak össze, tizedestörtek és karakterláncok nem (ez előbbi összehasonlításra használható a `bc` program, utóbbira meg a fentebb említett `==`, `!=`, ... relációk).

Ezeken kívül számos más feltétel létezik, mivel még speciális file-teszteléseket (pl. hogy egy file-ra van-e futtatási jogunk, mi magunk vagyunk-e a tulajdonosa, stb.), illetve időrendi összehasonlításokat (pl. hogy két file közül melyiket módosították később) is végezhetünk. Lásd még: `man bash`.

Feltételeket össze is lehet kapcsolni:

- `felt1 -a felt2`: igaz, ha a `felt1` és `felt2` feltételek közül mindkettő igaz (logikai és – and – kapcsolat).
- `felt1 -o felt2`: igaz, ha a `felt1` vagy `felt2` feltételek közül legalább az egyik igaz (logikai vagy – or – kapcsolat).
- `! felt`: igaz, ha a `felt` feltétel hamis (logikai tagadás).
- `\( felt \)`, zárójelezés: igaz, ha a `felt` igaz. Ezzel tulajdonképpen a bonyolultabb, fenti és-t, vagy-ot vagy tagadást tartalmazó feltételek csoportosíthatóak (hasonlóan, mint a matematikai kifejezések zárójelezésénél). A nyitó és csukó kerek zárójel csúnya karakter, azaz ennek a jelentését a – karakterlánc összehasonlításnál látottakhoz hasonlóan – vagy egy backslash (\), vagy idézőjelek segítségével felül kell bírálni.

13.2. Az `if ... else` elágazás

Elágazások szervezésére az `if` parancs használható:

```
$ if [ feltétel ] ; then ...; fi
```

Amennyiben a ***feltétel*** igaz, a `then` és `fi` között levő blokk pontosan egyszer fut le. Ha a ***feltétel*** nem igaz, a blokkban levő parancs(ok) nem fut(nak) le.

Kicsit összetettebb az *egyébként* (else) ágat is tartalmazó feltétel:

```
$ if [ feltétel ] ; then ...(1) ; else ...(2) ; fi
```

Ebben az esetben, ha a ***feltétel*** igaz, az első (`...(1)`) utasítás-sorozatot hajtja végre a shell, ellenkező esetben, azaz ha a ***feltétel*** nem igaz, a második (`...(2)`) utasítás-sorozat fut le.

Például egy `if-else` szerkezettel könnyen kideríthetjük, hogy milyen időszámítás – téli vagy nyári – van most érvényben:

```
$ if [ `date +%Z` == "CET" ] ; then echo "Teli" ; else echo "Nyari" ; fi
```

13.3. További példák

Feltételes ciklusokat és elágazásokat egyszerű parancssorban ritkán használnak (a `for` ciklus sokkal gyakoribb), nehéz is efféle egysoros példát találni, illetve ami van, az kissé erőltetett:

```
$ f="/usr/bin" ; if [ -e $f ] ; then echo "$f letezik" ; \
        else echo "$f nem letezik" ; fi
/usr/bin létezik
$ f="/qqriq/xx" ; if [ -e $f ] ; then echo "$f létezik" ; \
        else echo "$f nem letezik" ; fi
/qqriq/xx nem létezik
$ f="/usr/bin" ; if [ -d $f ] ; then echo "$f egy könyvtár" ; \
        else echo "$f nem könyvtár" ; fi
/usr/bin egy könyvtár
$ f="/etc/issue" ; if [ -d $f ] ; then echo "$f egy könyvtár" ; \
        else echo "$f nem könyvtár" ; fi
/etc/issue nem könyvtár
$ if [ 10 -le 20 ] ; then "10 kisebb, mint 20!" ; fi
10 kisebb, mint 20!
$ if [ $((6*9)) -eq 42 ] ; then "A kérdésre a válasz:  42" ; fi
$

$ a=10; while [ $a -lt 18 ] ; do echo $a ; a=$(($a+2)) ; done
10
12
14
16
$
```

Több ciklust, feltételt és elágazást tetszőleges mélységig egymásba ágyazhatunk. Arra figyelni kell, hogy az `if`-et `fi`-vel, míg a ciklusokat `done`-nal kell lezárni, különben nagy kavarodás lesz...

14. Shell-programok

A fentiekhez hasonó, egysoros ciklusok elég ritkán használatosak a gyakorlatban, a ciklusmag mindig azért kicsit bonyolultabb. általában is, néha egy rakat utasítást azonos formában, többször szeretne az ember végrehajtatni. Ennek a megkönnyítésére találták ki a shell-programokat, avagy *shell-szkripteket*, amik nem mások, mint shell-parancsok együttese. Mindent, amit a shell-nek parancsként kiadhatunk, összefoglalhatjuk egy utásítás-gyűjteménybe, és később, bármikor lefuttathatjuk.

Utasítás-gyűjteményt bármely editorral létrehozhatunk, majd az így kapott file-t a `.` „paranccsal" lefuttathatjuk:

```
$ mcedit utasitasok
```

```
date
w
```

```
$ .  utasitasok
3:39pm up 36 days, 4:48, 3 users, load average:  0.08, 0.08, 0.08
USER      TTY   FROM          LOGIN@  IDLE  JCPU  PCPU WHAT
azonosito pts/0 szofi.elte.hu 3:34pm 0.00s 0.08s 0.01s w
Sat Oct 23 15:34:32 CEST 2004
```

```
$ mcedit negyzetszamok
```

```
echo "Az első öt négyzetszám:"
for i in `seq 5` ; do
        echo "$i*$i" | bc
done
```

```
$ .  negyzetszamok
Az első öt négyzetszám:
1
4
9
16
25
```

A fenti második példa tanulsága a ciklusokra vonatkozik. Nem szükséges a ciklusmagot egy sorba írni: az utasítások elválasztásához nem csak a pontosvessző használható, hanem az egyes utasításokat külön-külön sorba is lehet tenni. Sőt, a ciklus-halmaz és a `do` közötti pontosvessző helyett is rakhatunk sortörést.

Az `.` kiírása (az utasítás-gyűjtemény neve előtt) nem túl elegáns megoldás. Két változtatással már igazi programként tudjuk kezelni a szkriptet. Az első, hogy a szkript első sora legyen `#!/bin/sh`. Pont így, betűről betűre (azaz nem lehet benne szóköz, és a legvégén sortörésnek kell lennie):

```
$ mcedit negyzetszamok
```

```
#!/bin/sh
echo "Az első öt négyzetszám:"
for i in `seq 5` ; do
        echo "$i*$i" | bc
done
```

Ezután tudatni kell a rendszerrel, hogy ez a file (`negyszetszamok`) immáron nem csak egy sima szövegfile, hanem egy *program*, azaz futtatási jogot kell rá adni. Az operációs rendszer csak azokat a file-okat tekinti programnak, csak azokat hajlandó lefuttatni, aminek van futtatási joga. Viszont ezt a file-t mi hoztuk létre, azaz mi vagyunk a tulajdonosai, s így, mint tulajdonosok, megváltoztathatjuk a jogsultságokat:

```
$ chmod +x negyzetszamok
```

A `chmod` parancsról bővebben később, ez az utasítás szolgál a jogosultságok megváltoztatására. A `+` utal arra, hogy jogokat *adunk*, az `x` pedig a futtatásra (*execution*) vonatkozik. Ezután már csak a file nevét kell megadni.

Az így kezelt file-t már rendesen le tudjuk futtatni, mintha egy „igazi” parancs lenne:

```
$ negyzetszamok
Az első öt négyzetszám:
1
4
9
16
25
```

14.1. Néhány egyszerű program, feltételes ciklusok és elágazások nélkül

Az eddigiekben leírtak alapján már összetettebb, látványosabb programokat is tudunk készíteni.

Sorok átlagos hossza

```
$ mcedit sorhossz
```

```
#!/bin/sh
file="akarmi.txt"
sorok=`cat $file | wc -l`
hossz=`cat $file | wc -c`
atlag=`echo "scale=2;$hossz/$sorok" | bc`
echo "A sorok átlagos hossza:  $atlag karakter"
```

```
$ chmod +x sorhossz
$ cat akarmi.txt
elso sor
ez a masodik sor
ez meg a harmadik !
$ sorhossz
A sorok átlagos hossza:  15.33 karakter
```

Faktoriálisok

```
$ mcedit faktor
```

```
#!/bin/sh
f=1
for i in `seq 6` ; do
        f=`echo "$f*$i" | bc`
        echo $f
done
```

```
$ chmod +x faktor
$ faktor
1
2
6
24
120
720
```

Fibonacci-számok

```
$ mcedit fibonacci
```

```
#!/bin/sh
a=1 ; b=1
for i in `seq 6` ; do
        echo $a
        c=`echo $a+$b | bc`
        a=$b ; b=$c
done
```

```
$ chmod +x fibonacci
$ fibonacci
1
1
2
3
5
8
```

Pascal-háromszög

```
$ mcedit pascal
```

```
#!/bin/sh
str="1"
for k in `seq 10` ; do
        echo $str
        next=""
        p=0
        for i in $str ; do
                next="$next $(($i+$p))"
                p=$i
        done
        str="$next 1"
done
```

```
$ chmod +x pascal
$ pascal
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
```

14.2. Speciális változók

A változók értékére az eléjük írt dollár-karakterrel lehet hivatkozni. Shell-programokban a dollár-jel után mást is írhatunk, ami kimondottan már a futtatáshoz kapcsolódik, külön, parancssorból nem szokás ezeket használni:

- `$1`, `$2`, . . . `$9`: parancssori argumentumok,

- `$#`: a parancssori argumentumok száma,
- `$@`: a teljes argumentumlista,
- `$?`: az előző parancs *kilépési kódja*

(ezeken kívül még van pár egyéb speciális változó, ezeket azonban ritkábban használják... lásd még: `man bash`).

```
$ mcedit parancs
```

```
#!/bin/sh
echo "Az első argumentum: $1"
echo "A második: $2"
echo "A harmadik: $3"
echo "A teljes argumentumlista: $@"
echo "Az argumentumok száma: $#"
```

```
$ chmod +x parancs
```

```
$ parancs aaa qqriq xyz akarmi
Az első argumentum: aaa
A második: qqriq
A harmadik: xyz
A teljes argumentumlista: aaa qqriq xyz akarmi
Az argumentumok száma: 4
```

```
$ parancs egyparameter
Az első argumentum: egyparameter
A második:
A harmadik:
A teljes argumentumlista: egyparameter
Az argumentumok száma: 1
```

```
$ parancs akarmi "ez egy lesz!" ez-a-harmadik
Az első argumentum: akarmi
A második: ez egy lesz!
A harmadik: ez-a-harmadik
A teljes argumentumlista: akarmi ez egy lesz! ez-a-harmadik
Az argumentumok száma: 3
```

A kilépési kóddal a legutoljára lefutott program sikerességéről kaphatunk tájékoztatást. Ha a program sikeresen lefutott, a `$?` változó értéke 0 lesz, egyébként egy 0-tól különböző kis egész szám (néhány program esetén ez a szám a hiba jellegére utal):

```
$ echo "valmi" >szoveg.txt
$ cat szoveg.txt
valami
$ echo $?
0
$ cat dhksdgjdkgl.txt
cat: dhksdgjdkgl.txt: No such file or directory
$ echo $?
1
```

A `while` és az `if` parancsok egy kicsit „trükkösebben” működnek, mint ahogy néhány szakasszal korábban írtuk. Az `if` és a `; then` illetve a `while` és a `; do` közötti résznek tulajdonképpen egy programnak kell lennie. A shell lefuttatja ezt a programot és elágazás igaz blokkja illetve a ciklus addig hajtódik végre, amikor/amíg ez a program sikeresen lefut(ott), azaz a kilépési kódja nulla. A `[` karakter pedig nem más, mint egy feltétel-kiértékelő program (néhány UNIX rendszer alatt, pl. a NetBSD-nél létezik is, mint külön program: `/usr/bin/[`, linux alatt a `bash` dolgozza fel ezeket a feltételes kifejezéseket is). Azaz ha a feltétel igaz, a kilépési kódja 0, ha hamis *vagy* hibás, a kilépési kódja nullától különböző.

14.3. Néhány program, feltételes ciklusokkal és/vagy elágazásokkal

Legnagyobb közös osztó

A legnagyobb közös osztót az *euklideszi algoritmus* segítségével számolhatjuk ki. Tegyük fel, hogy A és B pozitív egész számok, valamint A nagyobb, mint B. Ekkor két eset lehetséges:

- Ha B osztója A-nak, akkor A és B legnagyobb közös osztója B.
- Ha B nem osztója A-nak, akkor A és B legnagyobb közös osztója megegyezik B és $A \bmod B$ legnagyobb közös osztójával, és $A \bmod B$ nyilván kisebb lesz, mint B.

Például ha 42 és 15 legnagyobb közös osztóját számolnánk ki, akkor: 42 a nagyobb, 15 nem osztja 42-t. 42-nek a 15-tel vett osztási maradéka 12. 12 nem osztja 15-öt, 15-nek a 12-vel vett osztási maradéka 3. 3 már osztja 12-t, azaz 12 és 3 legnagyobb közös osztója 3. Ez viszont megegyezik 12 és 15 legnagyobb közös osztójával, ami megegyezik 42 és 15 legnagyobb közös osztójával:

$$LNKO(42, 15) \rightarrow 42, 15, 12, \boxed{3}$$

Hasonlóan például 137 és 42 legnagyobb közös osztója:

$$LNKO(137, 42) \rightarrow 137, 42, 11, 9, 2, \boxed{1}$$

És egy program, ami ezt ki is számolja:

```
$ mcedit lnko
```

```
#!/bin/sh
a=$1
b=$2
while [ $(($a%$b)) -ne 0 ] ; do
        c=$(($a%$b))
        a=$b
        b=$c
done
echo $b
```

```
$ chmod +x lnko
$ lnko 42 15
3
$ lnko 137 42
1
```

Mi lenne akkor, ha valaki úgy használná, hogy nem adja meg az egyik, vagy másik argumentumot vagy megadja ugyan, de az nem egy szám? (Ki lehet próbálni...) Egy kis módosítással felhasználóbaráttá (és bolondbiztossá) lehet tenni a programot:

```
$ mcedit lnko
```

```
#!/bin/sh
a=$1
b=$2
[ "$a" -gt 0 -a "$b" -gt 0 -a $# -eq 2 ] >&/dev/null
if [ $?  -ne 0 ] ; then
        echo "használat:  lnko n1 n2 (n1 és n2 pozitív egészek)
        exit 1
fi
while [ $(($a%$b)) -ne 0 ] ; do
        c=$(($a%$b))
        a=$b
        b=$c
done
echo $b
```

Itt az első `[...]` blokkban levő feltétel igaz, ha az első argumentum is és a második argumentum is nagyobb, mint 0 és az argumentumok száma kettő. Ekkor a `[` parancs kilépési kódja 0 lesz. Ha nem igaz a feltétel *vagy* a feltétel szintaktikailag hibás (ez abban az esetben lép fel, ha `$a` vagy `$b` nem egy egész szám), akkor a kilépési kód nem lesz 0. Amennyiben a kifejezés szintaktikailag hibás, a `[` parancs kiír egy hibaüzenetet. Ezt a hibaüzenetet a `[...]` utáni `&>/dev/null` átirányítással tüntetjük el. A `if` utasításban már egyszerűen ennek a `[...]` parancsnak a kilépési kódját vizsgáljuk. Ha ez nem nulla (`$?  -ne 0`), akkor a feltétel igaz, kiírjuk a használati utasítást és kilépünk a programból. Az exit utáni 1-es azt okozza, hogy a mi programunknak is 1 (azaz gyk. nem 0) legyen a kilépési kódja, így ha valaki ezt egy másik programból vagy parancssorból használná, ezen az úton (a `$?` használatával) tájékozódhat a program lefutásának sikerességéről.

Hét napja

Azt, hogy egy adott dátum (pl. 2004. december 10-ike) a hét melyik napjára esik, az alábbi módon határozhatjuk meg. Jelölje Y, M és D rendre az évet, hónapot és a napot (ennél a példánál $Y = 2004$, $M = 12$ és $D = 10$), valamint legyen $Y' = Y$ és $M' = M$, ha $M \geq 3$, illetve $Y' = Y - 1$ és $M' = M + 12$, ha $M = 1$ vagy $M = 2$. Ekkor az $Y.M.D$ dátummal jelölt nap a hét

$$W = 1 + \left(\left[\frac{5 \cdot Y'}{4} \right] + \left[\frac{13 \cdot (M' + 1)}{5} \right] + 3 + D + \Delta \right) \bmod 7$$

napjára esik ($W = 1$: hétfő, $W = 7$: vasárnap), ahol

$$\Delta = 2 - \left[\frac{Y'}{100} \right] + \left[\frac{Y'}{400} \right],$$

amennyiben a Gregorián naptárrendszer van érvényben. $[x]$ jelöli x egészrészét.

```
$ mcedit hetnapja
```

```
#!/bin/sh
iy=$1
im=$2
id=$3
d=$id
if [ $im -le 2 ] ; then
        y=$(($iy-1))
        m=$(($im+12))
else
        y=$iy
        m=$im
fi
delta=$((2-($y/100)+($y/400)))
wd=$((1+((5*$y)/4+(13*($m+1))/5+3+$d+$delta)%7))
echo $wd
```

```
$ chmod +x hetnapja
$ hetnapja 2004 12 10
5
$ hetnapja 2004 1 1
4
```

Ennél a kis programnál kihasználtuk azt, hogy az egészrész-képzés csak két egész szám osztását követően jelenik meg, azaz rögtön ki lehet használni a `$((...))` azon tulajdonságat, hogy két egész szám osztásakor rögtön lefele kerekít. Mivel ezeknél az osztásoknál mindig pozitív mennyiségeket kell osztani, ezért a lefele kerekítés, az egészrész-képzés és a csonkolás ekvivalens műveletek.

A programot a fentiebbi, legnagyobb közös osztós példához hasonlóan tehetjük felhasználóbaráttá. Érdemes továbbá kiegészíteni még egy teszteléssel: a Gregorián naptárat 1582. október 14-én vezették be. Ez előtt a Δ-t 0-nak kell venni, utána pedig a fenti formulát kell alkalmazni.