Programozási alapismeretek 1. gyakorlat emlékeztető

Riskó Gergely <progalap@risko.hu>

ELTE IK PSZT, 2007.

Követelmények és házi feladatok, órarend, ZH időpontok, adminisztráció, a jegyzet új verziói: http://www.gergely.risko.hu/oktatas-2007-pa1.hu.html

A jegyzet során feltételezzük, hogy bizonyos alapvető fogalmakkal a hallgató tisztában van. Pl. tudja, hogy mi a különbség egy szöveg fájl és egy bináris fájl között és ezért érti, hogy miért nem szerkeszthet Microsoft Worddel szövegfájlokat. Az alapvető informatikai ismeretekről remek szakkönyvek állnak rendelkezésre.

1. A UNIX és a GNU/Linux

Röviden: a UNIX egy olyan operációs rendszer, ami nagyon régen készült. A GNU/Linux pedig ennek egy utánzata, amely a szabad szoftver mozgalom keretein belül készül(t). Külön érdekessége, hogy a félév során meg fogjuk tanulni haladó felhasználói szinten kezelni.

Bővebben:

http://en.wikipedia.org/wiki/UNIX
http://en.wikipedia.org/wiki/GNU
http://en.wikipedia.org/wiki/Linux
http://www.gnu.org/
http://www.fsf.org/
http://www.kernel.org/

Nem kell feltelepíteni ilyen operációs rendszert senkinek otthon, habár nyugodtan megpróbálkozhat vele szabad idejében[1]. Minden megtanulható és elvégezhető az egyetem számítógépein a Lovardában[2], illetve a Pandorán[3].

Továbbá nem kari, hanem egyetemi szinten is lehetőség van GNU/Linux rendszerre azonosítót kérni. Ezen a gépen további szolgáltatások, pl. az egyetem hírcsoportjai (ELTE NEWS) is elérhetőek: http://www.caesar.elte.hu/.

Szintén az ELTE Informatikai Igazgatósága[4] üzemeltet egy levelezőlista szolgáltatást, amin megtalálhatóak az egyes szakok levelezőlistái is, nézzünk körül ezen a szolgáltatáson[5]. Ajánlott figyelemmel követni a `progmat`[6] és a `proginf`[7] elnevezésű listák történéseit.

[1] Legkönnyebben talán az Ubuntuval fog boldogulni: http://www.ubuntu.com/

[2] A Lovarda egy gépterem, ami a földszinten van a déli tömbben és minden gépen van Windows és GNU/Linux is.

[3] A Pandora egy szerverszámítógép, ahova bármely hallgató bejelentkezhet és az interneten elérhető a pandora.inf.elte.hu címen

[4] http://iig.elte.hu/

[5] http://listbox.elte.hu/

[6] https://listbox.elte.hu/mailman/listinfo/progmat

[7] https://listbox.elte.hu/mailman/listinfo/proginf

2. Alapvető parancsok, műveletek

2.1. Bejelentkezés egy GNU/Linux-os gépre

Mindenki rendelkezik a Pandorára egy felhasználói névvel és jelszóval, ezzel tud bejelentkezni a Lovarda helyi számítógépeire akár a Windows-ba, akár a GNU/Linux-ba értelemszerű módon, kicsit bonyolultabb a helyzet, ha a Pandorát vagy a Caesart akarja használni, ugyanis ezek elé fizikailag nem tud leülni, csak interneten keresztül tudja elérni ún. SSH[8] kliensprogrammal, amilyen pl. GNU/Linux-on a `ssh` vagy Windows-on a PuTTY[9].

A megadandó számítógépnév (host name) a pandora.inf.elte.hu, a port a 22-es, a kiválasztandó protokoll az SSH, a „window→translation” menüpontban a „character set”-et állítsuk „UTF-8”-ra! A beállításokat el is lehet menteni!

GNU/Linux alól a Pandorára való bejelentkezés az `ssh usernév@pandora.inf.elte.hu` paranccsal lehetséges.

Minden esetben ügyeljünk a felhasználói név és a jelszó pontos megadására, a kis- és nagybetűk közti különbségre!

Sikeres bejelentkezés után megjelennek a rendszergazda éppen aktuális üzenetei, majd elindul a shell, a parancsértelmező. Ez a program fogja a továbbiakban minden parancsunkat feldolgozni, lényegében a félév GNU/Linux-os része ezen program megismeréséről fog szólni.

Kijelentkezni a `logout` paranccsal lehet:

```
errge@home:~$ ssh errge@pandora.inf.elte.hu
errge@pandora.inf.elte.hu's password:
Last login: Tue Sep  4 23:47:33 2007 from catv-5063052c.catv.broadband.hu
Linux pandora 2.6.16-hardened-r10-pandora #1 Fri Jul 21 14:17:46 CEST 2006 i686 GNU/Linux

* Van lehetőség authentikált smtp servert használni
pandorás azonosítóval (SSL kell). Server: smtp.inf.elte.hu, port 465

* Oracle adatbázis elérése lehetséges sqlplus cliensel a pandora-n.
Használat: sqlplus username@oradb v sqlplus username@ablinux

* Meglevo szovegfajlokat az iconv paranccsal lehet konvertalni:
iconv -f ISO-8859-2 -t UTF-8 <regi_iso.txt >uj_utf8.txt
iconv -f UTF-8 -t ISO-8859-2 <regi_utf8.txt >uj_iso.txt


UTF-8 ekezetteszt: áéíóöőűÁÉÍÓÖŐŰÜŰ
errge@pandora:~$ logout
Connection to pandora.inf.elte.hu closed.
errge@home:~$
```

Egy bejelentkezés és egy kijelentkezés

Az aláhúzott rész a sor elején a shell által adott prompt, ahol várja az utasításunkat, az utána következő részt kell begépelni, majd entert ütve a számítógép válasza látható a következő promptig.

2.2. Dokumentáció

Minden tárgyalásra kerülő parancsról kérhetünk segítséget a `man 1 parancsnév` segítségével. A fontosabb parancsok segítőlapja, illetve hibaüzenetei magyarul jelenhetnek meg a Pandorán, ha ez minket zavar[10], akkor adjuk ki az `export LANGUAGE=C` parancsot! Hamarosan megtanuljuk beállítani, hogy ez minden belépésünkkor automatikusan megtörténjen.

[8] http://en.wikipedia.org/wiki/Secure_Shell
[9] http://en.wikipedia.org/wiki/PuTTY
[10] Ugye zavar!

2.3. Könyvtárak, fájlok

Adatainkat fájlokban tárolhatjuk, melyeket könyvtárakba rendezhetünk. Minden fájlnak van egy neve, a fájl rejtett, ha első betűje pont. A rejtett fájlok néhány esetben speciálisan viselkednek (pl. automatikusan nem listázódnak, illetve a minden fájlra vonatkozó parancsok rájuk nem vonatkoznak), de egyébként teljesen hagyományosak. A fájlkiterjesztések használata nem annyira megszokott és magától értetődő, mint Windowsban, de azért elterjedt.

A GNU/Linux rendszerekben minden állomány egy közös fában van, a fa építőelemeit a `/` jellel választjuk el, a fa gyökerének a neve szintén a `/`. Minden egyes pillanatban a shellünk a fa valamely csomópontjában áll, ezt hívjuk az aktuális könyvtárnak vagy munkakönyvtárnak. Ez a `pwd` paranccsal lekérdezhető, a `cd` paranccsal változtatható és erre a műveletre olyan hétköznapilag fogunk hivatkozni, minthogy „menjünk bele a könyvtárba”.

Azért célszerű ez a fogalom, mert a fájlok nevét az aktuális könyvtárhoz viszonyítva is megadhatjuk (ha nem a `/` jellel kezdünk), és így a saját könyvtáramban lévő `jegyzet.pdf`-re nem kell mindig a bonyolult `/h/e/errge/jegyzet.pdf` módon hivatkoznom, hanem ha az aktuális munkakönyvtáram a `/h/e/errge`, akkor egyszerűen `jegyzet.pdf`-et is írhatok.

Az munkakönyvtárunk neve a felhasználói név és a gépnév után a promptban mindig látszik.

Most lássunk jópár parancsot egy példán keresztül, a parancsok egy részét majd részletesebben is tárgyaljuk, de a már bemutatott `man` paranccsal a dokumentációjuk önállóan is olvasható. Figyeljünk fel a `..` és a `.` használatára!

```
1  errge@pandora:~$ mkdir progalap-pelda
2  errge@pandora:~$ cd progalap-pelda
3  errge@pandora:~/progalap-pelda$ mkdir konyvtar
4  errge@pandora:~/progalap-pelda$ echo hello world >fajl
5  errge@pandora:~/progalap-pelda$ ls
6  fajl  konyvtar
7  errge@pandora:~/progalap-pelda$ cp fajl masolat
8  errge@pandora:~/progalap-pelda$ ls
9  fajl  konyvtar  masolat
10 errge@pandora:~/progalap-pelda$ cd konyvtar
11 errge@pandora:~/progalap-pelda/konyvtar$ ls
12 errge@pandora:~/progalap-pelda/konyvtar$ cp ../masolat megegymasolat
13 errge@pandora:~/progalap-pelda/konyvtar$ ls
14 megegymasolat
15 errge@pandora:~/progalap-pelda/konyvtar$ cd ..
16 errge@pandora:~/progalap-pelda$ ls .
17 fajl  konyvtar  masolat
18 errge@pandora:~/progalap-pelda$ pwd
19 /h/e/errge/progalap-pelda
20 errge@pandora:~/progalap-pelda$ cd ~
21 errge@pandora:~$ ls progalap-pelda
22 fajl  konyvtar  masolat
23 errge@pandora:~$ pwd
24 /h/e/errge
25 errge@pandora:~$ cd /h/e/errge/progalap-pelda/
26 errge@pandora:~/progalap-pelda$ cd ..
27 errge@pandora:~$ rm progalap-pelda
28 rm: cannot remove ‘progalap-pelda’: Is a directory
29 errge@pandora:~$ rmdir progalap-pelda
30 rmdir: progalap-pelda: Directory not empty
31 errge@pandora:~$ rm -r progalap-pelda
```

A példában szereplő `echo` dolga, hogy kiírja a paramétereit, az utána szereplő jellel ez a kiírás egy fájlba irányítható. A különböző fajta átirányításokról még részletesen lesz szó.

Az `rmdir` parancs csak üres könyvtárat töröl, míg az `rm` parancs alapértelmezés szerint nem foglalkozik könyvtárakkal, ha könyvtárat adunk meg neki, hibával tér vissza. Ugyanakkor a `-r` opcióval rekurzívan működik, azaz az egész megadott könyvtárstruktúrát bejárva, minden fájlt és könyvtárat töröl, végül a megadottat is. Ez fontos jellemzője lesz a többi parancsunknak is, hogy a megfelelő opciók megtalálásával a program működése nagyban befolyásolható. Az opciókról minden esetben tartalmaz részletes leírást a parancs man oldala.

Láthatjuk, hogy amikor a saját könyvtárunkba belépünk (`/h/e/errge` az esetemben), akkor a promptban munkakönyvtárként a `~` jelenik meg, nem pedig a `/h/e/errge`. Ez azért van, mert a shellben a `~` rövidíti a saját könyvtárunkat, ezt mi is használhatjuk, így a `cd ~/progalap-pelda` helyett a `cd /h/e/errge/progalap-pelda/` parancs is szerepelhetne.

A Pandorán minden felhasználóhoz tartozik egy nyilvános könyvtár is, az én esetemben ez a `/h/public/e/errge`, ami mindenki más által böngészhető. Próbáljuk ki, nézzük meg mások publikus könyvátrait, menjünk oda cd-vel, listázzunk, nézelődjünk: fájlokat a `cat` paranccsal jeleníthetünk meg a képernyőn! A publikus könyvtáron belül van egy `public_html` nevű könyvtár, aminek a tartalma weblapként jelenik meg a http://people.inf.elte.hu/errge/ címen.

Próbáljuk ki az `mc` parancsot is, ennek hatására megjelenik egy modern fájlkezelő alkalmazás (mint amilyen Windowson a Total Commander, vagy a Far Manager).

A fájlaink és könyvtáraink neveibe sose tegyünk speciális karaktereket (mint amilyenek az ékezetek, próbáljuk meg a szóközt is kerülni, bár arról fogunk szót ejteni, hogy mire kell ügyelni szóköz használatakor).

2.4. Editorok

Munkánk során sokszor lesz szükség arra, hogy szövegfájlokat szerkesszünk, ezt tehetnénk a saját gépünkön is, ugyanis hamarosan megláthatjuk, hogyan másolhatunk fájlokat a Pandorára, de ez rendkívűl kényelmetlen lenne hosszútávon, mindenképp érdemes megtanulni legalább egy editort használni. Egyébként is, egy programozó ideje nagy részét UNIX gépek előtt tölti, így az ezen a rendszeren működő editorokat, de legalább egyet alaposan ismernie kell, különben reménytelenül lassan fog csak tudni dolgozni.

A kurzus ideje alatt a `vim` vagy az `emacs` valamelyikének elsajátítását ajánlom. Ha egyiket sem ismeri még a hallgató, akkor inkább az `emacs` megismerését javaslom, ugyanis ehhez sokkal több kényelmi funkció érhető el és határtalanul testreszabható, programozható.

Mindkét editorhoz elérhető rengeteg dokumentáció, bevezető, ezeket Google-vel keressük meg! Amennyiben ebbe mégsem akarunk időt fektetni, akkor használjuk az `mcedit fájlnév` parancsot fájlok szerekesztésére, ez azonnal, intuitíven működtethető.

Gyakorlásként készítsük el a saját könyvátrunkban a `.profile` nevű állományt és írjunk bele egy sort, méghozzá az `export LANGUAGE=C` sort. Ne felejtsünk a sor végére entert tenni! Jól nevelt ember, jól nevelt szövegfájlai közt nincs olyan, aminek az utolsó karaktere nem újsor. Például azért, mert az ilyen fájlokat `cat`-tel megjelenítve a következő promptunk (az újsor híján) nem sor elején kezdődik. Továbbá az utolsó sordobás hiánya okozhat egyéb kellemetlenségeket is különböző programoknál (olyan alapvetőeknél is, mint a `grep` vagy a `sed`). Mint az már sejthető, a `.profile` állomány fut le minden bejelentkezéskor, mielőtt az első promptunk megjelenne. Ezekszerint ebbe a fájlba tehetünk emlékeztetőket is magunknak, amik minden alkalommal megjelennek belépéskor. Próbáljuk ezt ki, használjuk az `echo` parancsot!

Ilyen egyszerű feladatot elvégezhetünk persze editor használata nélkül is, hiszen az `echo export LANGUAGE=C >~/.profile` paranccsal is elkészül a fájl és a tartalma is megfelelő. Ugyanakkor ezzel a módszerrel a fájl korábbi tartalma figyelmeztetés nélkül elvész, míg egy editorban látjuk, hogy hoppá, ebben a fájlban már van valami.

2.5. Levelezés, PINE

A Pandorás felhasználónkhoz `felhasznalonev@inf.elte.hu` alakú email cím kapcsolódik, ezért fedezzük fel a `pine` program használatát is, ezzel olvashatjuk el az érkezett leveleket és küldhetünk újakat tetszőleges emailcímre a Pandorás azonosítónkról. Ez az `mcedit`-hez hasonlóan a képernyőn megjelenő információk segítségével azonnal használható, figyeljünk fel arra, hogy az éppen használható billentyűk jelentéséről a képernyő alján mindig megjelenik információ! Ha egy billentyűkombináció `^X`-ként van jelölve, ez a `Ctrl-X`-et jelenti.

Fájlokat csatolni egy levélhez az új levél foglamazása (`Compose Message`) közben lehet az `Attchmnt` sorban a `^J` lenyomásával.

2.6. Fájlok másolása gépek között

Tegyük fel, hogy egy másik, szintén internetre kapcsolt GNU/Linux gépre vagy gépről akarunk átmásolni egy fájlt, ekkor az `scp` parancsot használhatjuk:

Az scp használata

```
1 errge@home:~/tmp$ scp valami.zip errge@pandora.inf.elte.hu:
2 errge@pandora.inf.elte.hu's password:
3 valami.zip                                   100%  193KB 192.6KB/s   00:00
4 errge@home:~/tmp$ cd ~/bpkonyv/
5 errge@home:~/bpkonyv$ scp -r errge@pandora.inf.elte.hu:/h/public/f/fa/public_html/ps .
6 errge@pandora.inf.elte.hu's password:
7 f.ps                                         100%   55KB  54.7KB/s   00:00
8 elemi.ps                                     100%  747KB 373.7KB/s   00:02
9 ...
```

Mindig először a forrásállományt kell megadni, majd a célt. A célban vagy a forrásban kettősponttal elválaszthatjuk a cél-, illetve forrásgép nevét a gépen való elérési úttól, ha nem használunk kettőspontot, akkor a helyi gépen értjük az elérési utat. A gépnév előtt, az `ssh`-hoz hasonlóan az at (`@`) jellel, mint elválasztóval a felhasználói nevünket is megadhatjuk. A `-r` kapcsolót használva tudtunk egész könyvtárat másolni, ugyanis a példában szereplő `ps` az egy könyvtár neve volt.

Amennyiben az a gép, ami előtt ülünk, Windows operációs rendszert futtat, akkor a már említett PuTTY projekt weblapjáról letölthető[11] a `pscp.exe` nevű program, ami az `scp`-vel teljesen azonos módon működik, csupán `pscp` a neve, tehát a megtanult módon annak segítségével másolhatunk Windowsról is fájlokat.

Amennyiben ennél kényelmesebb módszerre vágyunk, akkor próbáljuk ki a WinSCP[12] nevű programot! Ez az `mc`-hez hasonlóan, egyik ablakban mutatja a saját gépünk tartalmát és egy másikban a távoli gépét (pl. Pandora) és a megszokott billentyűkkel (F5, F6, F8, stb.) végezhetjük el a kívánt műveleteket.

3. További egyszerű parancsok, opciók

Mielőtt megnéznék, hogy hogyan lehet apró parancsokból hatalmas, mindent elvégző, kávéfőző pipeline-okat (avagy csővezetékeket[13]) építeni, nézzünk még pár új parancsot, illetve már ismert parancsokhoz új opciókat. Természetesen nem fogjuk megnézni az összes létező opciót, vagy parancsot. Referenciaként a man oldalakat lapozgassuk, ne ezt a jegyzetet!

[11] http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

[12] http://winscp.net/eng/download.php

[13] Ugyanis ez a magyar neve. Pár könyv használja, mi is néha, de nem mindenki érti meg, ha mondjuk neki, hogy csővezeték.

3.1. cat

A `cat` parancs az argumentumokban felsorolt fájlokat kiírja a képernyőre, opciói:

- `-n`: sorszámozás
- `-b`: a nem üres sorok sorszámozása
- `-E`: a sorvégek megjelölése `$` jellel, így láthatóvá válnak a sorvégi láthatatlan karakterek[14]

Próbáljuk ki ezeket az opciókat egy–két szövegfájlon, amiket megírtunk a kedvenc editorunkkal![15]

3.2. ls

- `-a`: a rejtett, ponttal kezdődő bejegyzések megjelenítése
- `-A`: mint `-a`, de nem listázza a mindig jelenlévő `.` és `..` bejegyzést
- `-l`: részletes lista soronként egy fájllal (bejegyzéssel), a mezők:
  - fájltípus (1 karakter) és jogosultságok (9 karakter) leírása
  - hardlinkek száma
  - a fájl tulajdonosa
  - a fájlhoz rendelt csoport
  - a fájl mérete
  - az utolsó módosítás ideje
  - a fájl neve
- `-d`: a könyvtárak egy bejegyzésként való megjelenítése a tartalmuk helyett

  Ha a paraméter listában megadunk egy könyvtárat, akkor az `ls` alapértelmezés szerint a könyvtár tartalmát listázza, hiszen így működik az intuíciónknak megfelelően az `ls /h/e` parancs. Ugyanakkor ha meg szeretnénk tudni, hogy milyen is a jogosultságok beállítása a publikus és a privát könyvtárunkon, mi lehet a különbség, akkor meg az tűnik logikusnak, hogy kiadjuk az `ls -l ~ /h/public/e/errge` parancsot, ami azonban ilyenkor a két említett könyvtár tartalmát listázza, mit sem mondva magukról a könyvtárakról, ilyenkor kell használni ezt az opciót.

```
1 errge@pandora:~$ ls -ld ~ /h/public/e/errge
2 drwx------ 23 errge progterv 4096 2007-09-09 23:47 /h/e/errge
3 drwxr-xr-x  5 errge progterv  117 2007-09-09 21:22 /h/public/e/errge
```

A jogosultságokról, tulajdonosokról, csoportokról és a hardlinkekről még lesz szó.

Vegyük észre a példában, hogy az `ls -l -d ...` parancs `ls -ld ...` formában is kiadható, azaz az opciók összevonhatók. Ez igaz minden programra (parancsra), a rövid opciónevek tekintetében. Van ugyanis sok opciónak hosszú neve, a `-d`-é például a `--directory`, míg a `-l`-nek nincs hosszú neve. Vannak olyan ritkán használt opciók is, amiknek csak hosszú neve van, illetve vannak olyan hosszú nevű opciók is, amiknek plusz argumentumot is kell adni, például:

[14] Egy kultúrált ember fájlaiban nincsenek a soros végén fölösleges szóközök vagy tabok (azaz whitespacek).
[15] Ami mostanra ugye az `emacs` és nem a `vim`... ;)

```
errge@pandora:~$ ls -l --full-time righaat.txt
-rw-r--r-- 1 errge progterv 11561 2007-03-21 14:29:33.599879779 +0100 righaat.txt
errge@pandora:~$ ls -l --time-style=+%Y%m%d-%H%M%S righaat.txt
-rw-r--r-- 1 errge progterv 11561 20070321-142933 righaat.txt
```

Az utolsó példában látható rejtélyesnek tűnő kifejezés formáját a `date` parancs tárgyalásakor majd tisztázzuk, vagy a `man 1 date` súgó használatával már most megismerkedhetünk velük.

Rövid névvel is rendelkező opció is várhat argumentumot, azonban az egyenlőségjelet nem szabad kitenni a rövid opciónév után:

```
errge@pandora:~/public$ ls -w 20
asm2
local
nevnap.tar
public_html
recept.txt
xor
xor.asm
xor.o
errge@pandora:~/public$ ls -w
ls: option requires an argument -- w
Try 'ls --help' for more information.
errge@pandora:~/public$ ls -w=30
ls: invalid line width: =30
errge@pandora:~/public$ ls -w30
asm2         recept.txt
local        xor
nevnap.tar   xor.asm
public_html  xor.o
errge@pandora:~/public$ ls --width=30
asm2         recept.txt
local        xor
nevnap.tar   xor.asm
public_html  xor.o
```

Próbáljuk ki, hogy mi történik, ha teljesen rövid `-w30` forma mellé még egy `-d` opciót is szeretnénk besűríteni! A `-w30` kifejezésben hova kell ekkor tennünk az `d`-t, hogyha az argumentumok közt könyvtárakat sorulunk fel, akkor azoknak ne listázza a tartalmát? Olvassunk utána, hogy mit is csinál a `-w` kapcsoló és egyben értsük meg a `-1` kapcsoló használatát is! Lehet különbség az `ls -w1`, illetve `ls -1` kimenete között?

3.3. A more és a nála is több less

Ezzel a két paranccsal hosszú fájlokat tekinthetünk meg olyan terminálon, aminek a magassága jóval kisebb, mint a fájl sorainak száma. A `less`-ből a `q` gombbal lehet kilépni, a `more` automatikusan kilép, amikor a fájl végére érünk a lapozásban. További információ a man oldalaikban található, a `less` sokkal többmindenre képes (pl. kurzor gombokkal könnyedén visszafele is lapozható a megjelenített dokumentum). A `man` parancs is a `less`-t használja alapértelmezés szerint a Pandorán, de ez a `PAGER` környezeti változóval megváltoztatható.[16]

3.4. wc (word count)

Segítségével megszámolhatjuk szövegfájlainkban lévő karakterek, szavak és sorok számát:

[16] A környezeti változókról később még lesz szó.

```
errge@home:~/docs/elte/progalap1$ wc jegyzet.tex
  492  2515 22220 jegyzet.tex
errge@home:~/docs/elte/progalap1$ wc -w jegyzet.tex Makefile
 2444 jegyzet.tex
   33 Makefile
 2477 total
errge@home:~/docs/elte/progalap1$ wc -c jegyzet.tex Makefile
21448 jegyzet.tex
  301 Makefile
21749 total
errge@home:~/docs/elte/progalap1$ wc -l jegyzet.tex Makefile
  470 jegyzet.tex
   15 Makefile
  485 total
errge@home:~/docs/elte/progalap1$ wc -w jegyzet.tex Makefile
 2444 jegyzet.tex
   33 Makefile
 2477 total
errge@home:~/docs/elte/progalap1$ wc jegyzet.tex Makefile
  470  2444 21448 jegyzet.tex
   15    33   301 Makefile
  485  2477 21749 total
errge@home:~/docs/elte/progalap1$ wc -wc jegyzet.tex
 2577 22752 jegyzet.tex
```

Amennyiben több fájlt adunk meg, akkor összegzést is kapunk, a különböző bemutatott opciókkal pedig válogathatunk, hogy a három lehetséges mennyiség (sorok, szavak, karakterek) közül melyikek jelenjenek meg.[17]

3.5. sort

A `sort` parancs segítségével fájlok sorai ABC (vagy a `-n` opcióval numerikus) sorrendbe rendezhetőek, a `-r` kapcsolóval pedig a fordított sorrend érhető el.

```
errge@pandora:~/tmp$ cat proba
2
z
10
a
errge@pandora:~/tmp$ sort proba
10
2
a
z
errge@pandora:~/tmp$ sort -n proba
a
z
2
10
errge@pandora:~/tmp$ sort -nr proba
10
2
z
a
```

[17] Az opciók sorrendjének variálása nem változtat a kimenet sorrendjén, az mindig sorok, szavak, karakterek sorrendű, tehát a `wc -wl ...` parancs ugyanazt csinálja, mint a `wc -lw ...` parancs.

3.6. reset

Ha a képernyőre nem megjeleníthető karaktereket írunk (pl. bináris fájl megjelenítésével, mondjuk a `cat /bin/ls` paranccsal), azzal akarutunkon kívűl is „megbolondulhat” a terminálunk és a megjelenő prompt is szemétnek tűnik ezekután. Ilyenkor vakon adjuk ki a `reset` parancsot, aminek hatására minden visszaáll a régi kerékvágásba.

3.7. gzip

Régóta ismertek algoritmusok[18] fájlok méretének a csökkentésére, a bennük lévő redundanciák csökkentésével.

A `gzip` a Huffman kódolás és az LZ77 felhasználásával tömörít fájlokat. Fájlok tömörítése a `gzip fájlnév` paranccsal, kibontásuk a `gunzip fájlnév.gz` paranccsal érhető el. Tömörítéskor és kibontáskor is a kiindulási fájl törlődik és helyette csak a kibontott vagy a tömörített marad meg. Ettől eltérő viselkedést a `-c` kapcsolóval érhetünk el, azonban ekkor a kimenet a standard outputra érkezik és annak átirányításáról gondoskodnunk kell, erről még lesz szó.

3.8. tar

Az előbb említett `gzip` parancs csupán arra való, hogy egy-egy fájlt tömörebb formára hozzunk és a továbbiakban úgy tároljuk. Több fájl, illetve egész könyvtárstruktúrák egy fájlba való ábrázolására, majd később ezen fájlból a könyvtárstruktúra visszaállítására a `tar` parancs való.

Lássuk[19]:

```
errge@pandora:~/tmp$ find test
test
test/file
test/subdir
test/subdir/file2
errge@pandora:~/tmp$ tar -c -f test.tar test
errge@pandora:~/tmp$ rm -r test
errge@pandora:~/tmp$ ls -l
total 12
-rw-r--r-- 1 errge progterv 10240 2007-09-16 23:54 test.tar
errge@pandora:~/tmp$ gzip test.tar
errge@pandora:~/tmp$ ls -l
total 4
-rw-r--r-- 1 errge progterv 204 2007-09-16 23:54 test.tar.gz
errge@pandora:~/tmp$ gunzip test.tar.gz
errge@pandora:~/tmp$ tar -x -f test.tar
errge@pandora:~/tmp$ ls -l
total 12
drwxr-xr-x 3 errge progterv    30 2007-09-16 23:53 test
-rw-r--r-- 1 errge progterv 10240 2007-09-16 23:54 test.tar
errge@pandora:~/tmp$ find test
test
test/file
test/subdir
test/subdir/file2
errge@pandora:~/tmp$ tar -t -f test.tar
test/
test/file
test/subdir/
test/subdir/file2
```

[18] http://en.wikipedia.org/wiki/Lossless_data_compression

[19] A `find test` parancs itt csak kiírja a `test` nevű könyvtár tartalmát rekurzívan.

Mint az a példában látható, a tar parancsnak három fontos módja van, amit a megfelelő opciók megadásával lehet kiválasztani. Az első a csomagolványok létrehozása (a -c), a második a csomagolványok kibontása (a -x), a harmadik pedig a csomagolványok tartalmának kiiratása (a -t). A csomagolvány fájl neve, amivel a műveletet végrehajtjuk, mindig a -f opcióval adtható meg az opció argumentumaként. Valamint csomagolvány létrehozásakor még kötelező megadni az egész parancs argumentumában a csomaghoz hozzáadandó fájlok és/vagy könyvtárak nevét.

Látható, hogy a csomagolás helymegtakarítást nem okoz, sőt, általában a fájlokat kicsomagolt állapotban a fájlrendszer hatékonyabban tudja tárolni, mint a tar, azonban a gzip segítségével a csomagolványt be is tömöríthetjük és így már a windowsból jól ismert zip fájlokkal ekvivalens eredményt érhetünk el. Az így létrejövő .tar.gz kiterjesztésű fájlokat szokás egyszerűen csak .tgz kiterjesztéssel ellátni. Ezt a részét a jegyzetnek a gunzip parancs is olvasta és ezért .tgz kibontásakor a kibontott fájlt .tar kiterjesztéssel látja el.

Mivel nagyon sokszor a tar-t és a gzip-et az itt leírt módon együtt kell használni, erre külön opciót is bevezettek a tar-ban (a -z-t). Ezt megadva, az elkészülő csomagolvány tömörítve is lesz automatikusan a gzip meghívásával, illetve használhatjuk ezt a kapcsolót kibontáskor is és akkor emiatt a tar rögtön tudja, hogy a megadott csomagolványt először a gzip-pel ki is kell bontani.

3.9. tail és head

E parancsok segítségével kiírható a fájlok eleje vagy vége. Alapértelmezés szerint mindkettő az első/utolsó 10 sorral dolgozik, de a -n opció ügyes megválasztásával ez befolyásolható:

```
errge@pandora:~$ for i in $(seq 1 25); do echo $i >>szamok; done
errge@pandora:~$ head -n 3 szamok
1
2
3
errge@pandora:~$ tail -n 3 szamok
23
24
25
```

Próbáljuk ki, hogy mi a head -n -3 szamok, tail -n -3 szamok, head -n +3 szamok és tail -n +3 szamok parancsoknak mi a hatása és jegyezzük meg ezeket a lehetőségeket, jól jöhet, ha tudunk róluk!

3.10. cut

Egy fájl minden sorának bizonyos részeit vághatjuk ki ezzel a paranccsal.

A kivágandó rész megadható a karakterek pozícióinak specifikálásával vagy úgynevezett mezők használatával. A mezők arra használhatók, hogy bizonyos elválasztókarakter mentén olyan szeleteket is kezelni tudjunk, amelyben a betűk száma nem azonos.

Amennyiben a -c <lista> (avagy --characters=<lista>) opciót használjuk, akkor karakterpozíciókról beszélünk, amennyiben pedig a -f <lista> (avagy --fields=<lista>) kapcsolót, akkor mezőkről. Mindkét esetben a lista vesszővel elválasztott elemekből épül fel, ahol az egyes elemek alakja N, N-, N-M vagy -M lehet, itt az N és az M számok, a jelentés pedig értelemszerű. A -f esetében a mezőket a TAB karakter választja el, de ez megváltoztatható a -d (hosszúneve: --delimeter) használatával, ami argumentumként egyetlen karaktert vár, a kívánt elválasztókaraktert.[20]

[20] Az intervallumok sorrendjének variálása a wc-hez hasonlóan nem hoz más eredményt, mint a növekvő sorrendben való megadásuk.

A --complement opció megadásával a vágás minden sorra külön-külön vett pontos ellentéte hajtódik végre.

A cut

```
errge@pandora:~$ vi test
errge@pandora:~$ cat test
egy-maganal volt a gyilkos fegyver
ketto-csipkebokor vesszo
errge@pandora:~$ cut -c4-10 test
-magana
to-csip
errge@pandora:~$ cut -d- -f1 test
egy
ketto
errge@pandora:~$ cut -d" " -f1,3- test
egy-maganal a gyilkos fegyver
ketto-csipkebokor
errge@pandora:~$ cut -d" " -f1,4- test
egy-maganal gyilkos fegyver
ketto-csipkebokor
```

3.11. uniq

Ha egy fájlt a cat helyett a uniq-kal iratunk ki, akkor a fájlban egymásután többször szereplő teljesen azonos sorok csak egyszer jelennek meg.

Opciói:

- -u: csak az egyedi sorok megjelenítése
- -d: csak az ismétlődő sorok (egyszer való) megjelenítése
- -c: ezzel az opcióval a sorok elé kiírja, hogy az a sor mennyiszer fordult elő egymásután

Ezt a parancsot programjaink kimeneteinek szépítésére, illetve a különböző számlálásoknál az azonos találtok többször számolásának elkerülésére használhatjuk majd.

3.12. fgrep

Ez a parancs karakterláncra szűr az input fájlban. A parancs első argumentuma mindig a keresett karakterlánc, majd a fájlnevek felsorolása jön. Pl.: fgrep almafa fajl1 fajl2 fajl3.

Az -x opcióval megköthetjük, hogy a megadott karakterlánc ne a sorban része legyen csupán, hanem a teljes sort töltse ki.

A kimeneten az illeszkedő sorok jelennek meg. Amennyiben több fájlt adtunk meg, akkor az fgrep azt is kiírja, hogy melyik fájlban van az abban a sorban szereplő találat. Próbáljuk ki a szóbajövő eseteket (sok fájl, egy találat; sok fájl, sok találat; nincs találat; egy fájl, stb.!

3.13. cp

Fájlok másolása kétféle alakban:

- cp FORRÁS CÉL, ahol a FORRÁS egyelemű,
- cp FORRÁS... CÉL, ahol a FORRÁS... sok paraméter is lehet és mindet másolja, de így a CÉL kötelezően könyvtár.

Mindkét esetben megadható a -a kapcsoló a parancs elején, hogy a FORRÁS részben lévő könyvtárakat egy az egyben másolja, így teljesen (jogosultságokra is) egyező rekurzív másolat készíthető.

3.14. mv

Az mv a cp-vel egyező két formában működik, azonban mindig eltávolítja a FORRÁS-t, amikor már készen van a CÉL. [21] Könyvtárakkal mindenféle opció nélkül is hajlandó foglalkozni, így -a opciója nincs is.

3.15. last

A számítógépre történt belépéseket listázhatjuk ezzel a paranccsal. Amennyiben egy belépés távoli volt[22], akkor a last megpróbálja azt is megmutatni, hogy melyik számítógépről történt ez a bejelentkezés. Azonban amikor a kimeneti formátumát kitalálták, a számítógépek elnevezése az interneten még jóval rendezettebb és rövidebb neveket eredményező volt. Így csak 16 karaktert tartottak fent, ami ma már kevés. Ráadásul a gépneveknek általában a vége az érdekes, hiszen ebből lehet következtetni arra, hogy valaki az épületben van-e, érdemes-e megpróbálni megkeresni, stb. Ugyanakkor a last parancs pont a gépnév végét vágja le, ha nem fér el a 16 karakterben. Ezért mi használni fogjuk mindig a -a kapcsolót, ami a gépnevet a kimenet legvégére viszi és így ott (szinte) korlátlan hely áll rendelkezésre a megjelenítéshez.

Amennyiben csak valakiknek a belépéseire vagyunk kíváncsiak, akkor soroljuk fel ezeket a felhasználókat a parancs végén argumentumokként.

Az első oszlopban található a felhasználó neve, ez sosem hosszabb 8 betűnél és biztosan nem tartalmaz szóközt vagy TAB karaktert. Utána következik a 10. karaktertől kezdve (12 karakter hosszan) a belépéshez felhasznált terminál, távoli bejelentkezésekkor ez valami fizikailag nem létező eszközön foglalt (a mi szempontunkból véletlenszerű sorszámú) képernyő. A 23. karakterpozíciótól 36 karakteren keresztül a bejelentkezés kezdeti dátuma és a gépen töltött időintervallum szerepel.[23] A 61. karaktertől kezdődik ezekután a bejelentkezéshez használt gép neve.

A last alapértelmezés szerint, majd a -a kapcsolóval

```
1  errge@pandora:~$ last
2  kandras  pts/3        g3kthljf0y.adsl. Mon Sep 17 01:18 - 01:28  (00:09)
3  nct      pts/1        fw.tigra.hu      Mon Sep 17 01:16 - 01:41  (00:24)
4  kandras  pts/3        g3kthljf0y.adsl. Mon Sep 17 01:09 - 01:14  (00:04)
5  ...
6  szabogab pts/0        dsl51b642e3.pool Sat Sep  1 06:38 - 06:44  (00:05)
7
8  wtmp begins Sat Sep  1 06:38:58 2007
9  errge@pandora:~$ last -a
10 kandras  pts/3        Mon Sep 17 01:18 - 01:28  (00:09)     g3kthljf0y.adsl.datanet.hu
11 nct      pts/1        Mon Sep 17 01:16 - 01:41  (00:24)     fw.tigra.hu
12 kandras  pts/3        Mon Sep 17 01:09 - 01:14  (00:04)     g3kthljf0y.adsl.datanet.hu
13 ...
14 szabogab pts/0        Sat Sep  1 06:38 - 06:44  (00:05)     dsl51b642e3.pool.t-online.hu
15
16 wtmp begins Sat Sep  1 06:38:58 2007
```

[21] Vagy, ha lehetséges, akkor egylépésben csinálja ezt a kettő dolgot.

[22] A Pandorán és egyéb szervergépeken kevés, rendszergazdai művelettől eltekintve minden belépés ilyen természetesen.

[23] Ezen rész értelmezése ember számára könnyű, azért az fgrep segítségével keressünk rá egy + jelet tartalmazó sorra, hogy arra is lássunk példát, ha egy bejelentkezés több, mint egy napig él. Ugyanakkor programmal eldönteni, hogy valaki be volt-e jelentkezve ekkor, meg akkor nehézkes ezen információ alapján, erre a -t opció való, nézzük meg a man oldalban a használatát!

4. Parancsfájlok

Nyilvánvaló lehet, hogy a parancsértelmező viszonylag mély tárgyalásába nem mennénk bele, ha a cél az lenne, hogy az olvasó tudjon kiadni parancsokat a terminálon. Hiszen kinek lenne kedve egy egyszeri eset kedvéért (főleg kezdőként, amíg lassan megy) bonyolult dolgokat átgondolni.

Azért kell mindezt megérteni, mert a parancsainkat beírhatjuk egy fájlba és azt a fájlt bármennyiszer futtathatjuk később is. A parancsfájlainkat elkészíthetjük kényelmesen egy editorral, kipróbálhatjuk, hogy mi történik, javíthatunk, azok megmaradnak (valamint gyakorlatvezetőknek is így adhatjuk be a feladatainkat).

Habár nem kötelező, a parancsfájlainkat mindig kezdjük a `#!/bin/bash` sorral és utána egy új sorral, valamint adjunk nekik `.sh` kiterjesztést pl.:

test.sh és a futtatása

```
errge@pandora:~$ cat test.sh
#!/bin/bash

echo Ez a parancsfajl nem csinal semmi erdekeset, csupan
echo a kepernyore irkal, de ket paranccsal, mindketto lefut,
# kommentek így helyezhetőek el a sor végéig a #-tól kezdődően
echo sot, mindharom, ha meg ezt a sort is latjuk.
errge@pandora:~$ bash test.sh
Ez a parancsfajl nem csinal semmi erdekeset, csupan
a kepernyore irkal, de ket paranccsal, mindketto lefut,
sot, mindharom, ha meg ezt a sort is latjuk.
errge@pandora:~$
```

5. Csatornák

A shell a működése közben nyitva tart három csatornát, ahonnan olvashat (ebből van egy), illetve ahova írhat (ebből van kettő):

- standard bemenet (0): alapértelmezés szerint a terminál billentyűzete
- standard kimenet (1): alapértelmezés szerint a terminál képernyője
- standard hibacsatorna (2): alapértelmezés szerint a terminál képernyője

Az eddig említett legtöbb parancs, ha nem adnak meg neki fájlnév argumentumokat, akkor a standard inputról olvas, ezért van az, hogy a `cat` „lefagy”, ha önmagában futtatjuk. Igazából bemenetre vár. A bemenet végét billentyűzeten a `Ctrl-D`-vel jelezhetjük, a fájloknak egyszerűen pedig van végük.

A parancsok a normális működésük eredményét írják a kimeneti csatornára és a hibaüzeneteket (mint pl. nem megfelelő használat) írják a hibacsatornára.

5.1. Átirányítások

A standard bemenet fájllal helyettesíthető, ha parancssor végére a `<fájlnév` karaktersorozatot írjuk (vagy még formálisabban a `0<fájlnév` karaktersorozatot), illetve a kimenet új fájlba (avagy régi felülírottjába) irányítható, ha a végére a `>fájlnév` (ami megegyezik az `1>fájlnév` hatásával) sztringet írjuk, a standard errort a `2>fájlnév` irányítja át. A `>` helyett a `>>` is állhat, ekkor létező fájl esetén felülírás helyett hozzáírás történik.

Megjegyezzük, hogy nem szabad ugyanabba a fájlba egyszerre átirányítani két különböző csatornát az `1>pelda 2>pelda` parancsvéggel, ugyanis ekkor mindenféle versenyhelyzetek léphetnek fel és a kimenet erősen kevert vagy akár veszteséges is lehet, erre van egy külön formula: `>pelda 2>&1` (fontos a sorrend, fordítva nem működik).

5.2. Pipeline-ok

Megoldható az is, hogy két vagy több parancsot indítsunk el egyszerre, egy parancssorral és mindig az előző parancs kimenete adja a következő parancs bemenetét. Ez lesz az az elem, ami az eddig felsorolt unalmas lehetőségeknek hatalmas erőt ad. Ez volt az az ötlet, amivel nagyon hosszú időre sikerült megoldást adni viszonylag nehezen megfogalmazható, előtte külön-külön programokat igénylő feladatokra is.

A parancsok közé a `|` jelet kell tenni, minden parancsrész elindul párhuzamosan és feldolgozzák egymás be- és kimeneteit, a csővezeték első és utolsó parancsának be- és kimenete a `<`, `>` és `>>` jelekkel a fent leírt módon átirányítható.

Pl. hány különböző felhasználó jelentkezett be eddig ebben a hónapban a Pandorára?

```
1 errge@pandora:~$ last | head -n -2 | cut -c1-8 | sort | uniq | wc -l
2 1137
3 errge@pandora:~$
```

További feladatok:

- Hányszor jelentkezett be `bzsr` a Pandorára?
- Mennyi b kezdőbetűs felhasználó jelentkezett be a Pandorára?
- Számoljuk meg a bejelentkezési könyvtárban lévő directorykat!
- Az `adatok` fájl minden sora egy hallgató nevét, lakhelyét, születési dátumát, emailcímét és szakját tartalmazza; ezeket egymástól `:`-tal elválasztva.
  - Mondjuk meg, hogy mely városokban laknak a hallgatók!
  - Adjuk meg a fizikusok számát!
  - Készítsünk névsort a proginfósokról!
- A `VB` fájlban tároljuk az eddigi labdarúgó VB győzteseinek nevét, az évszámmal és a rendező ország nevével együtt.
  - Hány különböző nyertes van?
  - Ki nyert 1966-ban?
  - Kik tudtak többször is nyerni?
  - Melyik ország nyert legtöbbször és hányszor?
- Hány olyan sor van a `mondatok` nevű fájlban, amiben a tej és a vaj szó együtt szerepel?
- Készítsük el parancssorral a `3legtobb` nevű fájlt, amiben a Pandorára ebben a hónapban legtöbbször bejelentkezett felhasználók közül az első három szerepel, a bejelentkezési számaikkal együtt, csökkenő sorrendben.

6. Parancsfeldolgozás

Amikor egy parancsot kiadunk a shellünkben, akkor azt a parancsértelmező először is a benne szereplő (védetlen) szóközök (és tabok) mentén szavakra vágja, majd végrehajt egy elég bonyolult szabályrendszer alapján különböző kifejtéseket.

Fontos már most megérteni, hogy az parancssor kifejtését a shell végzi, nem pedig az egyes programok (és így a hamarosan bevezetendő, más operációsrendszerekből már jól ismert joker-karakterek értelmezését sem kell újra és újra megírni).

Mivel ezek a szabályok tényleg bonyolultak, először nézzük meg, hogyan fogjuk őket tesztelni. A legtöbb rejtélyes hiba biztos, hogy a kifejtések körül lesz minden olyan hallgatónak, aki először találkozik Unix rendszerrel.[24] A tesztelésben a `:` parancs fog segíteni, aminek lényege, hogy nem csinál semmit, viszont gyorsan be lehet gépelni, illetve a `set -x` kiadása után a végrehajtott parancsok előtt megjelenik, hogy a shell hogyan értette őket. Ez az ami érdekes most, maga a `:` futása nem lesz túl érdekes, mert mint említettem, nem csinál semmit. Ha befejeztük a próbálkozást és zavar a túl sok megjelenő információ minden parancs előtt, akkor a `set +x` hatástalanítja ezt a nyomkövető beállítást.

Kifejtések nyomkövetése

```
errge@pandora:~/tmp$ A="x y"
errge@pandora:~/tmp$ ls -l
total 0
-rw-r--r-- 1 errge progterv 0 2007-09-18 00:23 fajl1
-rw-r--r-- 1 errge progterv 0 2007-09-18 00:23 fajl2
drwxr-xr-x 2 errge progterv 6 2007-09-18 00:23 szokoz a konyvtar neveben
-rw-r--r-- 1 errge progterv 0 2007-09-18 00:24 xxx
errge@pandora:~/tmp$ set -x
errge@pandora:~/tmp$ : * $A
+ : fajl1 fajl2 'szokoz a konyvtar neveben' xxx x y
errge@pandora:~/tmp$ : * "$A"
+ : fajl1 fajl2 'szokoz a konyvtar neveben' xxx 'x y'
errge@pandora:~/tmp$ set +x
```

A kimeneten (a kezdő plusz jel után) látható, hogy a `:` parancsnak milyen argumentumai lesznek és az aposztrófokra figyelve az is megállapítható, hogy az első esetben hat argumentuma van, míg a második esetben csak öt.

6.1. Speciális karakterek levédése (Quoting)

A parancssorkifejtés során a következő karaktereknek van speciális jelentésük:
`$ ‘ " \ ’ # ! { } * ? ~ | & ; ( ) < > space tab newline`

Minden karakter speciális jelentése három féle módon vehető el:

- bármely (akár nem speciálisé is) egy karakteré a karaktert megelőző backslashsel. Egyszerűen arról van szó, hogyha a shell találkozik egy védetlen backslashsel, akkor törli és a következő egyetlen egy karakternek viszont semmiképp nem tulajdonít speciális jelentést, pl. `\$ \* \\ \’`

- egész karaktercsoporté a karaktereket körülvevő idézőjellel: `"mindenfele: \$*\"&|<\‘"`

  Az idézőjel nem védi a `$`, `‘` (backtick), `\` és `"` jeleket, minden mást azonban igen. Az említett négy jel úgy érhető el az idézőjeleken belül, ha backslasht teszünk eléjük. Idézőjelen belül a backslash egyébként nem viselkedik speciálisan.

  Ha a shellt interaktívan futtatjuk, tehát billentyűzetről gépeltük be a parancsot és nem pedig parancsfájlból hajtódik végre, akkor még a `!` sincs védve. Ez a speciális viselkedés a `set +H` paranccsal interaktív módban is kikapcsolható. Ha most beírjuk ezt a `.profile` fájlunkba, később kevesebb gondunk lesz.

[24] Sőt, azoknak is, akik 10 éve írnak shell programokat.

- egész karaktercsoporté a karaktereket körülvevő aposztróffal: `'mindenfele: $*"&|<‘'`

  A védés ilyenkor teljesen mechanikus, a következő aposztrófnál ér véget, bármi is volt előtte, tehát aposztrófot ezen belül nem lehet használni még backslashsel sem (sűrű hiba!). Viszont az aposztróf leírható mindenféle környezeten kívűl a `\'` segítségével és így tulajdonképpen megjegyezhető az az ökölszabály, hogy aposztrófon belül aposztrófot lehet írni a `'\''` karaktersorozattal, ugyanis az első apsztróf (átmenetileg) lezárja a védő környezetet, majd backslashsel lerakjuk a kívánt jelet és megnyitjuk újra a védést. Érdekesség, hogy erre a módszerre a shelltől való puskázással is rájöhetünk:

```
1 errge@pandora:~$ set -x
2 errge@pandora:~$ : "gergely's apple"
3 + : 'gergely'\''s apple'
```

Ha úgy ütünk entert egy parancssor végén, hogy az utolsó kettő védés közül valamelyik még nyitva van, akkor megjelenik egy kacsacsőr, ami további inputot vár, ilyenkor ha nem tudjuk kibogozni, hogy mit is akarunk, `Ctrl-C`-vel visszakaphatjuk a promptot és semmi nem kerül végrehajtásra.

6.2. A kifejtés menete (Expansion)

Miután a shell a parancssort végigolvasta és olvasás közben az előbb leírt védési szabályok alapján szavakra bontotta azt, az itt leírt módszerekkel, az alábbi sorrendben tovább kifejti a szavakat.

A kifejtés megakadályozható levédéssel.

6.2.1. Kapocs kifejtés (Brace expansion)

Teljesen mechanikus (azaz a speciális karaktereket figyelmenkívűlhagyó) direktszorzó, ami különálló szavakat generál és intervallumokat is kibont. Minden eredmény elé odateszi a prefixet, ami a nyitó kapcsoszárójelet megelőzte, mögé pedig a postfixet, ami a záró kapcsost követte.

`A kapocs kifejtés, egymásba ágyazni is szabad...`

```
1 errge@pandora:~$ set -x
2 errge@pandora:~$ : e{1,2,3{1..7}}u
3 + : e1u e2u e31u e32u e33u e34u e35u e36u e37u
```

6.2.2. Tilde kifejtés (Tilde expansion)

Ha egy szó elején van (védetlen) `~`, akkor azt az első (védetlen) `/`-ig az ún. tildeprefix követi. Ha nincs per a szóban, akkor az egész tilde utáni rész a tildeprefix. A shell a tilde és a tildeprefix helyére behelyezi a tildeprefixben megemlített felhasználó saját könyvtárának abszolút elérési útját, vagy ha a tildeprefix üres, akkor az aktuális felhasználóét.

A tildeprefix az új szó elején túl még változóértékadáskor is használható egyenlőségjel vagy kettőspont után, ez később még (pl. a `PATH` átállításánál) praktikus lesz.

6.2.3. Paraméterek, aritmetikai kifejezések kifejtése és parancsok behelyettesítése (Parameter, arithmetic expansion and command substitution)

Erről később részletesen beszélünk a paramétereknél, változóknál. Fontos, hogy a `"` védőkarakter ezt a kifejtést nem akadályozza meg (és pont ezért szeretjük), valamint az, hogy ez a három kifejtés egy menetben történik, balról jobbra, a kifejtett paraméter már nem lesz újból kifejtve.

Arról lehet felismerni a kifejtések ezen csoportját, hogy mindegyiket a `$` jel vagy a ‘ („fordított aposztróf”) jel vezeti be. A kifejtések ezen csoportjára igaz, hogyha idézőjellel vannak körülvéve (azaz a `"` védi őket), akkor bármi is a kifejtés eredménye, a végeredmény egy szó lesz, azaz az ilyen részekre a következő pont nem vonatkozik.

6.2.4. Szavak szétbontása újra (Word splitting)

Amennyiben az előző pontban említett kifejtések között van olyan, ami nem volt idézőjelek között, akkor ezeket a shell újra szavakra bontja a szóközök, tabok és újsorok mentén.

6.2.5. Elérési utak kifejtése, globbing (Pathname expansion)

Ha a shell valamelyik szóban `*`, `?` vagy `[` jelet talál, akkor azt a szót mintának értelmezi és kicseréli a mintára illeszkedő fájlokra (mint új paraméterekre) abc sorrendben, vagy nem végez cserét, ha nincs illeszkedő fájlrendszerbejegyzés.

Globbing példák

```
1  errge@pandora:~/tmp$ ls -l
2  total 0
3  drwxr-xr-x 2 errge progterv 6 Sep 18 08:12 A
4  -rw-r--r-- 1 errge progterv 0 Sep 18 00:23 fajl1
5  drwxr-xr-x 2 errge progterv 6 Sep 18 00:23 szokoz a konyvtar neveben
6  -rw-r--r-- 1 errge progterv 0 Sep 18 00:24 xxx
7  errge@pandora:~/tmp$ set -x
8  errge@pandora:~/tmp$ : *[[:digit:]]
9  + : fajl1
10 errge@pandora:~/tmp$ : [[:lower:]]*
11 + : fajl1 'szokoz a konyvtar neveben' xxx
12 errge@pandora:~/tmp$ : [[:upper:]]*
13 + : A
14 errge@pandora:~/tmp$ : *[[:space:]]*
15 + : 'szokoz a konyvtar neveben'
16 errge@pandora:~/tmp$ : *[[:space:][:digit:]x]*
17 + : fajl1 'szokoz a konyvtar neveben' xxx
```

A mintában szereplő speciális karakterek a következők lehetnek:

- `*`: bármilyen karakterláncot, az üreset is helyettesíti
- `?`: egyetlen karaktert helyettesít, de abból bármilyet
- `[...]`: A `...` helyén felsorolt karakterek közül bármelyik egyet helyettesíti, pl. `[ahq]`. Régi könyvek előszeretettel említik, hogy megadhatóak intervallumok és ezért a kisbetűk így illeszthetők: [a-z]. Azonban ez újabb rendszereken megbízhatatlan, ugyanis újabb telepítéseknél az `A` betűt nem a `B` betű követi a rendezési sorrendben, hanem az `a` és így a `a`-t nem a `b`, hanem a `B`. Át lehet ezt állítani, de inkább az ajánlható, hogy a szögletes zárójelen belül ezeket a megjelölőket használjuk egész karaktercsoportok helyére (igen, ekkor összesen két szögletes zárójelre is szükség van a kifejezés két szélén):
  - `[:alnum:]`: számjegyek és betűk,
  - `[:alpha:]`: betűk,
  - `[:upper:]`: nagybetűk,
  - `[:lower:]`: kisbetűk,
  - `[:digit:]`: számok,

– `[:space:]`: bármilyen szóköz (akár tab vagy újsor is).

Egy minta csak akkor illeszkedik a rejtett fájlokra, ha a minta a `.` karakterrel kezdődik. A nyitó szögletes zárójel után tett `^` negációt jelent, az ilyen mintaelem pont a nem felsorolt karakterekre illeszkedik.

6.3. A parancs végrehajtása

Miután minden kifejtés megtörtént, a shell rendelkezik szavak egy hosszú listájával, ezek közül az első a parancs neve, amit a `PATH` környezetváltozó segítségével megkeres, majd lefuttatja paraméterként átadva az összes többi listaelemet, minden elem egy különálló paraméter[25]. Az, hogy melyik paramétert értelmezi a program opcióként és melyiket argumentumként, teljesen a programon áll, nem is kell, hogy így csoportosítsa a paramétereket, ez csak egy szokás, a shell szemantikailag nem értelmezi a kifejtésnél mélyebben az eredményt.

Ezért van az is, hogy pl. ha a `-bar` nevű fájlt akarjuk letörölni, akkor bajban vagyunk:

```
errge@pandora:~/tmp$ echo hello world >-bar
errge@pandora:~/tmp$ rm -bar
rm: invalid option -- b
Try ‘rm ./-bar’ to remove the file ‘-bar’.
Try ‘rm --help’ for more information.
errge@pandora:~/tmp$ rm -- -bar
errge@pandora:~/tmp$ ls -- -bar
ls: -bar: No such file or directory
```

Az `rm -bar` nem működhet, hiszen az rm ezt úgy érti, hogy biztosan meg akartuk neki adni a következő opciókat: `b`, `a`, `r`. `b` opciója nincs is, ezért hibát üzen és segít, azt mondja, hogy használjuk az `rm ./-bar` parancsot, ami valóban működne, azonban ha a `-` karakterrel kezdődő fájl elérési út kiterjesztésének az eredménye (mert kiadtuk az `rm *` parancsot ebben a könyvtárban), akkor nem tudjuk hogyan elé tenni minden ilyen fájlnévnek a `./`-t.

Ezért egy általánosabb megoldás és minden fontosabb paranccsal működik, hogy a fájlnév paraméterek elé, az utolsó opció után teszünk egy külön paraméterként egy `--`-t, ami azt jelzi a parancsnak, hogy köszönjük szépen, több opció már nem, csak argumentumok jönnek.

7. Paraméterek, környezetváltozók

Minden paraméter egy értéket tárol. Paraméter lehet egy szám, egy név[26], illetve egyes speciális karakterek is. Változónak hívunk minden olyan paramétert, amit név azonosít, szokás, hogy a változóneveink csupa nagybetűkből és esetleges aláhúzásokból állnak, de ez nem kötelező.

Amennyiben egy paraméter szám, akkor az egy pozícionális paraméter, ezeknek parancsfájlok esetén van jelentősége. A shell bennük tárolja el a parancsfájl hívásakor megadott paramétereket, nulladikként a parancsfájl hívási nevét. Előrevetítve, hogy paraméter értékét a parancsainkban a `$paraméter` formával fogjuk kifejteni, álljon itt egy példa:

```
errge@home:~$ cat test.sh
#!/bin/bash

echo fajlnev: $0, elso paramter: $1, 10. paramter: ${10}
errge@home:~$ bash test.sh egy ketto harom negy ot hat het nyolc kilenc tiz tizenegy
fajlnev: test.sh, elso paramter: egy, 10. paramter: tiz
```

[25] Kivéve azokat a paramétereket, amiket a csatornák átirányítgatására használunk.

[26] Név alatt az olyan szavakat értjük, amik csak alfanumerikus karaktereket és aláhúzást tartalmaznak, valamint nem kezdődnek számmal.

A változók értéket (nevükhöz híven) meg is változtathatjuk a `változónév=újérték` parancs kiadásával[27], az egyéb paraméterek értékét nem változtathatjuk meg. Az értékadás megtörténte előtt végrehajtódnak az `újérték`-en a 6.2.2 és 6.2.3 pontokban leírt kifejtések. Vannak olyan környezetváltozók, amik nem csak saját magunk számára értelmesek, hanem a rendszer saját maga is használja, ilyenre már láttunk példát, a `LANGUAGE` esetében, ezzel állíthattuk be az általunk kívánatos nyelvet. Továbbiak[28]:

- `PATH`: programok keresési útvonala, a könyvtárnevek kettősponttal elválasztva, a Pandorán, ha van `~/bin` könyvtárunk, akkor az automatikusan a `PATH` elejére kerül[29],
- `USER`: a felhasználó belépési neve,
- `HOME`: a felhasználó munkakönyvátárának elérési útja,
- `PS1`: a prompt formája (ld. „PROMPTING” fejezet a bash manlapjában).

Természetesen a `USER` és `HOME` környezetváltozók birizgálása nem jelent biztonsági kockázatot senkire, attól, hogy oda más valakit vagy más valaki munkakönyvtárát írjuk, még nem fog a rendszer minket más jogosultságokkal kezelni, csupán a mi életünk lesz kényelmetlenebb, mert pl. a tilde kifejtés rosszul fog működni.

Paraméter lehet még a `#` is, ami a pozícionális paraméterek közül a létező legnagyobb számértékét adja meg, a `@` és a `*`, amik mindketten az összes pozícionális paraméter értékét jelentik. Különbség az utóbbi kettő között akkor van, ha valamely kifejtendő szövegben a `$@`, illetve a `$*` idézőjelek közt van, előbbit ugyanis ekkor a `"$1" "$2" "$3"...`[30], utóbbit pedig a `"$1 $2 $3..."` értékké fejti ki a shell. A `?` paraméter a legutóbbi parancs visszatérési értékét, a `$` paraméter az éppen futó parancsértelmező processz-azonosítóját tartalmazza, habár ezekről még nem tanultunk, próbáljuk meg kiiratni őket!

8. Bonyolult kifejtések

Ideje visszatérni arra, amit az előző fejezetben későbbre halasztottunk, nevezetesen a paraméterek, a változók, az aritmetikai kifejezések és a parancsok behelyettesítésének tárgyalására. Ez a téma szinte kimeríthetetlen, a jegyzet nem referencia, sokkal részletesebb és mélyebb leírás található a `bash` manoldalában!

8.1. Paraméter behelyettesítése

Amennyiben egy paraméter értékét szeretnénk a neve alapján parancssorunkba illeszteni, akkor azt megtehetjük a `${paraméternév}` kifejezéssel, amiből a kapcsoszárójelek elhagyhatóak, ha a zárót nem követi olyan karakter, ami a paraméter részét is képezhetné[31].

8.2. Parancsok behelyettesítése

A `$(parancs)` (új forma) vagy `‘parancs‘` (régi forma) hatására kifejtés közben a shell végrehajtja a parancsot, majd a kimenetét beilleszti a kifejezés helyére. Emlékeztetünk rá, hogy

[27] Megszüntetni változót a `unset változónév` paranccsal lehet, a `változónév=` parancs ugyanis csak üresre állítja a változóhoz rendelt értéket, nem szünteti meg a változót magát.

[28] Mégtöbb információ fellelhető a `man 1 bash` paranccsal előhozott segítőlap „Shell Variables” szekciójában.

[29] Az én `PATH` környezetváltozóm a Pandorán:
`/h/e/errge/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games`

[30] És ez különösen jól használható néha, pl. egy egyszerű tömörítőszkript így nézhetne ki: `tar -c -z -f $@`

[31] Vajon miért működik az `echo $0alma` parancs, de nem az `echo $PATHalma`?

mivel ezután a kifejtés után jön még a 6.2.4 és a 6.2.5, ezért a behelyettesített kimenet még további kifejtéseknek eshet áldozatul, ha nincs idézőjellel védve.

Ezt a lehetőséget nagyon sokszor alkalmazzuk az `expr` paranccsal, aminek segítségével egyszerű aritmetikai kifejezések értékelhetőek ki. Pl. az `expr 4 + 5 - 4 / 2` parancs eredménye `7`. Mitől függ az `expr 7 * 5` parancs eredménye? Fontos, hogy az `expr` parancsnak az argumentumokat és operátorokat külön-külön paraméterként kell megadni, ugyanis ebbe a programba csak az alapműveletek megvalósítását építették bele, a parancssor darabokra szedését egyszerűen a shellre bízzák. Nem működik ezért pl. az `expr 7-5+6` parancs.

Mindenféle kifejtések

```
errge@pandora:~/tmp$ ls
A  fajl1  fajl2  output  szokoz a konyvtar neveben  xxx
errge@pandora:~/tmp$ set -x
errge@pandora:~/tmp$ : *
+ : A fajl1 fajl2 output 'szokoz a konyvtar neveben' xxx
errge@pandora:~/tmp$ A=*
+ A='*'
errge@pandora:~/tmp$ : $A
+ : A fajl1 fajl2 output 'szokoz a konyvtar neveben' xxx
errge@pandora:~/tmp$ A="ket szo"
+ A='ket szo'
errge@pandora:~/tmp$ : $A
+ : ket szo
errge@pandora:~/tmp$ : "$A"
+ : 'ket szo'
errge@pandora:~/tmp$ : 7 * 5
+ : 7 A fajl1 fajl2 output 'szokoz a konyvtar neveben' xxx 5
errge@pandora:~/tmp$ : $((7*5))
+ : 35
errge@pandora:~/tmp$ A=2
+ A=2
errge@pandora:~/tmp$ A=$(expr $A + 1)
++ expr 2 + 1
+ A=3
errge@pandora:~/tmp$ A=$(($A+1))
+ A=4
errge@pandora:~/tmp$ : $(ls -w$A)
++ ls -w4
+ : A fajl1 fajl2 output szokoz a konyvtar neveben xxx
errge@pandora:~/tmp$ : "$(ls -w$A)"
++ ls -w4
+ : 'A
fajl1
fajl2
output
szokoz a konyvtar neveben
xxx'
errge@pandora:~/tmp$ : $((2**16))
+ : 65536
errge@pandora:~/tmp$ set +x
errge@pandora:~/tmp$ cat test.sh
#!/bin/bash

set -x

: $* $@ "$*" "$@"
errge@pandora:~/tmp$ bash test.sh egy\ \ param ket ha
+ : egy param ket ha egy param ket ha 'egy  param ket ha' 'egy  param' ket ha
```

8.3. Aritmetikai kifejezések

Az `expr` megjelenése után, felismerve, hogy azt mennyien használják, annak funkcióit a shellbe is beépítették, a `$((kifejezés))` forma kifejtéskor a kifejezés aritmetikai értékére cserélődik. Erről bővebben a shell dokumentációjának „ARITHMETIC EVALUATION” fejezete szól.

8.4. Feladatok

- Írj olyan parancsfájlt feladat1 néven, ami az első paramétereként kapott fájlt felülírja ugyanezen fájl azon soraival, melyekben szerepel az alma szó. Feltesszük, hogy a parancsfájlnak van paramétere, az tényleg közönséges fájl, továbbá a parancsfájlt olyan könyvtárban futtatjuk, amelyikre van írási jog.
- Írj parancsfájlt feladat2 néven, aminek az első paramétere egy fájlnév. Ennek a fájlnak midnen sorában egy létező könyvtár neve van. A könyvtárnevekben nincs szóköz. A parancsfájl adjon ezekről a könyvtárakról fájllistát (a benne lévő fájlok neveit írja ki).
- Írj parancsfájlt feladat3 néven, ami megadja, hogy az aktuális könyvtárban hány közönséges fájl van.

9. Jogosultságok

A Unix három jogosultságfajtát ismer:

- `r` (4), olvasási: a fájl olvashatóságát, illetve egy könyvtár tartalmának (egy mélységű) listázhatóságát szabja meg,
- `w` (2), írási: a fájl módosíthatóságát, illetve a könyvtárban közvetlenül előforduló fájlok létrehozhatóságát, törölhetőségét, átnevezhetőségét szabja meg,
- `x` (1), futtatási: a fájl futtathatóságát, illetve a könyvtár alatti fájlrendszerrész elérhetőségét szabja meg.

Minden fájl esetében a felhasználók 3 diszjunkt halmazra oszlanak és erre a három halmazra külön-külön állítható, hogy a fenti jogosultságokból melyikkel rendelkeznek, ez a három rész:

- tulajdonos: a fájl tulajdonosa, egy konkrét felhasználó,
- csoport: a fájlhoz rendelt csoportba tartozó felhasználók, kivéve a tulajdonost,
- mindenki más.

Egy fájl tulajdonosa, illetve a fájlhoz rendelt csoport az `ls -l` hosszú kimenetében található, a 3. és a 4. oszlopban. A tulajdonos megváltoztatására csak a rendszergazdának van joga (a `chown` paranccsal), a saját fájlainkról nem mondhatunk le[32]. A fájlhoz rendelt csoportot átállíthatjuk a `chgrp újcsoportnév fájlnév` paranccsal, amennyiben az adott csoportnak tagjai vagyunk. Az, hogy milyen csoportoknak vagyunk tagjai, az `id` vagy a `groups` paranccsal tekinthető meg.

Az `ls -l` listájában a fentiek alapján már lehet értelmezni a 9 karaktert, ami a jogosultságokat jelzi, 3 csoportra kell osztani őket, az első 3 karakter a tulajdonosra vonatkozó jogok, a második három a fájlhoz rendelt csoportba tartozó felhasználókra vonatkozó jogok, a harmadik részben leírtak vonatkoznak mindenki másra. Mindhárom részben azok a jogok érvényesek,

[32] Hiszen azzal összezavarnánk a `quota` parancsot!

amik helyén a neki megfelelő betű (`r`, `w` vagy `x`) áll és azok nincsenek engedélyezve, amik helyén kötőjel áll.

Egy fájlra vonatkozó jogokat a `chmod jogok fájlnév...` paranccsal lehet változtatni, ahol a jogokat többféleképpen is megadhatjuk. Az első lehetőség, hogy a teljes új jogosultságrendszert kiszámoljuk, méghozzá négy számjegy formájában, amiből az utolsó három számít, az első mindig 0[33], pl. a `0517` az `r-x--xrwx` jogokat jelenti. A második lehetőség, hogy a jogoknál csak a jelenlegi jogrendszerhez képest kívánt változást írjuk le, `oszlop+-változás` formában, ahol az `oszlop` helyén három karakter (`u`(ser), `g`(roup) vagy `o`(ther)) valamilyen kombinációja állhat, annak megfelelően, hogy melyik oszlopo(ka)t szeretnénk változtatni, a `+-` helyén a `+` vagy `-` állhat, attól függően, hogy jogosultságot adni vagy elvenni szeretnénk, legvégül a `változás` helyén az `r`, `w` és `x` betűk valamilyen kombinációja állhat, aszerint, hogy milyen jogosultságot adunk vagy veszünk el.

Próbálgassuk ki ezeket a jogosultságokat, különösképp legyünk figyelemmel a könyvtárakra vonatkozó jogosultságokra, nézzük meg, hogy mi történik, ha egy könyvtárunkra nincs `x` jog, `r` jog, illetve `w` jog!

A jogosultságok köréből a leggyakrabban használt parancs a `chmod a+x parancs.sh`, amivel a `parancs.sh` nevű scriptfájlunk (azaz shell parancsokat tartalmazó programunk) futtathatóvá válik és így a későbbiekben csupán a nevével, mint paranccsal elindíthatjuk, nem kell eléírni, hogy `bash`. Természetesen, ha nincs a `PATH`-ben, akkor a relatív vagy az abszolút elérési útjával együtt kell megadnunk.

10. Parancskonstrukciók

10.1. Egyszerű parancsok

A bash shellben[34] már tudunk egyszerű parancsokat írni, amiknek a formája a következő:
`<változó értékadások> <parancs> <opciók és argumentumok> <átirányítások>`
Például: `LANG=hu_HU LC_CTYPE=hu_HU man ls`

A változók csak a futatott parancs környezetében változnak meg, a shell későbbi parancsai már a változók régi értékeivel kerülnek feldolgozásra:

```
1 errge@home:~$ B=almafa bash
2 errge@home:~$ echo $B
3 almafa
4 errge@home:~$ exit
5 errge@home:~$ echo ${B?}
6 bash: B: parameter null or not set
```

Vegyük észre, hogy milyen hasznos apróságot tartalmaz a példa: ha egy változó kifejtését a kapcsoszárójelen belül a kérdőjel követ, akkor amennyiben a változó igazából nem is létezik, akkor a shell hibát ad, ahelyett, hogy egyszerűen az üres sztringgel helyettesítené. Ráadásul, ha ez egy szkript közepén fordul elő, akkor az rögtön leáll és a hiba nem gyűrűzik tovább. Parancsfájlainkban szinte minden változóbehelyettesítést lecserélhetünk ilyenre és cserébe sokkal könnyebben és gyorsabban fény fog derülni az elkövetett hibákra.

A futtatott parancs környezete nem örökli a shellünk indulásakor nem definiált, azóta saját magunk által (értékadással) létrehozott változókat. Ha mégis ezt szeretnénk, akkor az `export`

[33] Aki nem hiszi, járjon utána, ezzel sokat tanulhat, de nekünk ez így elég...

[34] Apropó: a jegyzetben mindenhol a Bourne-Again SHell-el foglalkozunk. Többféle parancsértelmező létezik, ha valami nem működik, ellenőrízzük, hogy a rendszerünkön a bash fut, pl. úgy, hogy kiiratjuk a `BASH_VERSION` változót, vagy a `0` paramétert.

parancsot kell használnunk. A futtatott programok, shellek által végzett változó beállítások sosem befolyásolják a szülő processz környezetét.

```
errge@home:~$ echo ${A?}
bash: A: parameter null or not set
errge@home:~$ A=korte
errge@home:~$ echo ${A?}
korte
errge@home:~$ bash
errge@home:~$ echo ${A?}
bash: A: parameter null or not set
errge@home:~$ C=szilva
errge@home:~$ echo ${C?}
szilva
errge@home:~$ exit
errge@home:~$ echo ${C?}
bash: C: parameter null or not set
errge@home:~$ echo ${A?}
korte
errge@home:~$ export A
errge@home:~$ bash
errge@home:~$ echo ${C?}
bash: C: parameter null or not set
errge@home:~$ echo ${A?}
korte
```

Nem volt szó még részletesen arról, hogy minden parancs visszaad egy értéket, ami egy egész szám (0 és 255 között). Amennyiben igazságértéket akarunk ezekhez rendelni, azt úgy szokás, hogy a 0 jelenti az igazat és minden más a hamisat. Ha kíváncsiak vagyunk a legutoljára lefutott parancs visszatérési értékre, akkor a `?` paramétert (azaz pl. az `echo $?` parancsot) használhatjuk. Mi a saját parancsfájlainkból az `exit` parancsnak adott argumentummal jelezhetjük, hogy mi legyen a visszatérési érték, míg számos parancs tömören megfogalmazza mondanivalóját ebben a számban. Pl. a `grep` csak akkor ad vissza igazat (azaz 0-t), ha talált legalább egy illeszkedő sort, míg a `cmp`, ami fájlok összehasonlítására használható, hamissal jelzi, ha a két összehasonlítandó fájlban különbség mutatkozott.

Visszatérési értékek

```
errge@pandora:~/tmp$ cat test.sh
#!/bin/bash

exit $((4**2))
errge@pandora:~/tmp$ chmod a+x test.sh
errge@pandora:~/tmp$ ./test.sh
errge@pandora:~/tmp$ echo $?
16
errge@pandora:~/tmp$ echo $?
0
errge@pandora:~/tmp$ fgrep zsh test.sh ; echo $?
1
errge@pandora:~/tmp$ fgrep bash test.sh ; echo $?
#!/bin/bash
0
errge@pandora:~/tmp$ fgrep bash test.sh >/dev/null ; echo $?
0
```

10.2. Pipeline-ok

Az egyszerű parancsok csővezetékekké szervezhetőek a `|` jellel, az egész struktúra visszatérési értéke az utolsó, jobboldali parancs visszatérési értéke lesz. Ha a pipeline első parancsa előtt szerepel a `!` karakter (szóközzel elválasztva a parancstól), akkor a visszatérési érték negálódik, tehát 0 lesz, ha nem 0 volt és nem lesz 0, ha 0 volt. A csővezetékek szervezésének módjára részletesen kitértünk az 5.2 fejezetben.

Pipeline-ok készítése közben hasznos, ha ismerjük a `tee` parancsot, nyissuk fel a man oldalát és barátkozzunk meg vele most!

10.3. Listák

A csővezetékeket eddig mindig új sorba írtuk, azonban a pontosvessző használatával listába is fűzhetjük őket, ekkor a végrehajtás szekvenciális. A pontosvessző helyett az `&&` és a `||` operátor is használható, az első hatására a hátsó parancs csak akkor fut, ha az elülső igazzal tért vissza, míg a második esetében pont fordítva, a hátsó parancs csak akkor fut, ha az elülső meghiúsult. A visszatérési értéke az összetételnek pedig `&&` esetén csak akkor 0, ha mindkét rész 0-val tért vissza, `||` esetén 0, bármelyik rész 0-val tért vissza.

Mindennek a bemutatásához felhasználom a `true` és `false` beépített parancsokat, amik nem csinálnak semmit, csupán 0-át, illetve 1-et adnak vissza:

```
errge@home:~$ true && echo $?
0
errge@home:~$ false && echo $?
errge@home:~$ true || echo $?
errge@home:~$ false || echo $?
1
```

Mindenképp tanulmányozzuk a nagyon hasznos `test` parancs dokumentációját, rengeteg kapcsolójával szinte mindenféle aritmetikai, sztring egyenlőségi és fájlrendszerbeli vizsgálat elvégezhető. Az eredményt mindig a visszatérési értékben jelzi, így programok írásakor jól használható.

A csővezetéket építő `|` jel mindig erősebben köt, mint a `;`, `&&` vagy `||` bármelyikéé, amik azonos precedenciával bírnak. Ez elég meglepő, más programozási nyelvekben ez nem szokás:

```
errge@home:~$ true || { false && false ;}; echo $?
0
errge@home:~$ true || false && false; echo $?
1
```

10.4. Összetett parancsok

Az összetett parancsok listákból épülnek fel és a program logikai vezérlését, a szokásostól eltérő végrehajtási sorrend kijelölését teszik lehetővé. Természetesen a `bash` sokkal többféle parancsösszetételt ismer, mint amit mi megtanulunk.

10.4.1. `( lista )` és `{ lista ; }`

Listák csoportosítására szolgáló parancsok, így bírálhatjuk felül a számunkra nem kedvező zárójelezési sorrendet. A kapcsoszárójeles formula belsejét mindig újsorral (vagy pontosvesszővel) kell lezárni és a nyitó- és zárójelei körül szóközöket kell hagyni, ez a funkció tényleg csak a végrehajtási sorrend megadására való. A kerezárójeles kifejezés viszont egy új parancsértelmezőben

hajtja végre a listában felsorolt parancsokat, majd kilép és a végrehajtás a zárójel után folytatódik. Erre akkor lehet szükség, ha nem szeretnénk, hogy a zárójelezett rész környezetének változásai (pl. munkakönyvtár váltás, változók módosítása) a lefutásuk után érvényben maradjanak. Mindkét parancsösszetétel visszatérési értéke az általuk végrehajtott lista visszatérési értéke.

Vegyük észre, hogy az eddig megtanultakkal már lényegében tudunk elágazásokat írni, pl. parancsok helyes meghívását tömören ellenőrízni.

```
errge@pandora:~/tmp$ cat test.sh
#!/bin/bash

[ "$#" -eq 2 ] || { echo "Ezt a parancsot két paraméterrel kell hívni!"; exit 10; }
echo "Helyes használat, első paraméter: $1, második paraméter: $2"
errge@pandora:~/tmp$ ./test.sh ; echo $?
Ezt a parancsot két paraméterrel kell hívni!
10
errge@pandora:~/tmp$ ./test.sh a; echo $?
Ezt a parancsot két paraméterrel kell hívni!
10
errge@pandora:~/tmp$ ./test.sh a b; echo $?
Helyes használat, első paraméter: a, második paraméter: b
0
errge@pandora:~/tmp$ ./test.sh a b c; echo $?
Ezt a parancsot két paraméterrel kell hívni!
10
errge@pandora:~/tmp$ ./test.sh a "b c"; echo $?
Helyes használat, első paraméter: a, második paraméter: b c
0
errge@pandora:~/tmp$ vi test.sh
errge@pandora:~/tmp$ cat test.sh
#!/bin/bash

[ "$#" -eq 2 ] || {
  echo "Ezt a parancsot két paraméterrel kell hívni!"
  exit 10
}
echo "Helyes használat, első paraméter: $1, második paraméter: $2"
```

A példában használt `[ kifejezés ]` pontosan megfelel annak a működésnek, mint amit a `test kifejezés` jelent. Ez nem speciális shell utasítás, létezik a `/usr/bin/[` fájl is.

10.4.2. `(( kifejezés ))` és `[[ kifejezés ]]`

Ezzel a két parancskonstrukcióval nem foglalkozunk, bizonyos aritmetikai, illetve egyéb kifejezésekre vonatkozó vizsgálatok írhatóak fel velük külső parancsok (mint amilyen a `test` vagy az `expr`) használata nélkül. Vigyázzunk velük, mert szintaktikájuk habár hasonló, mint az említett parancsoké, vannak azért különbségek, inkább kerüljük őket, ha nincs kedvünk alaposan elolvasni a `bash` man oldalának ide vonatkozó részeit.

10.4.3. `for ... in ...; do lista ; done`

A `for` programkonstrukcióval ciklusokat szervezhetünk, az `in` után szereplő listát a shell először kifejti, majd végrehajtja a ciklusmagot annyiszor, ahány eleme van a kifejtésnek, minden egyes alkalommal a `for` után lévő névbe, mint környezetváltozóba téve az aktuális értéket. Amennyiben a kifejtés eredménye üres, akkor egy parancsot sem hajt végre és a visszatérési értéke a `for`-nak 0, egyébként a visszatérési érték az utoljára végrehajtott parancsé.

Amennyiben az `in` részt elhagyjuk, akkor a pozícionális paramétereken megy végig a shell.

A `seq` nagyon hasznos `for` ciklusokhoz szervezéséhez, ugyanis ez intervallumokat tud kiírni, így parancsbehelyettesítéssel kombinálva számlálóciklusokat írhatunk.

A for ciklus

```
errge@pandora:~/tmp$ cat test.sh
#!/bin/bash

for param ; do echo "paraméter: $param"; done

for x in 1 2 5
do
  echo -n "x: $x "
done
echo

for x in $(seq 1 100)
do
  echo $(($x**2))
done
errge@pandora:~/tmp$ ./test.sh p1 p2 | head
paraméter: p1
paraméter: p2
x: 1 x: 2 x: 5
1
4
9
16
25
36
49
```

10.4.4. `case ... in ...|... ) lista ;; ... esac`

```
errge@pandora:~/tmp$ cat test.sh
#!/bin/bash

i=1
for param
do
  echo -n "$i. paraméter: "
  case $param in
    [[:lower:]]*)
      echo -n "kisbetuvel kezdodik, "
      ( exit 1 ) ;;
    [[:upper:]]*)
      echo -n "nagybetuvel kezdodik, "
      ( exit 2 ) ;;
    [0-9]*)
      echo -n "szammal kezdodik, "
      ( exit 3 ) ;;
  esac
  echo "retval: $?"
  i=$(($i+1))
done
errge@pandora:~/tmp$ ./test.sh Bdsfg 4sdf aeetr ?234
1. paraméter: nagybetuvel kezdodik, retval: 2
2. paraméter: szammal kezdodik, retval: 3
3. paraméter: kisbetuvel kezdodik, retval: 1
4. paraméter: retval: 0
```

A `case` paranccsal sok ágú elágazásokat szervezhetünk. A `case` után kell megmondani, hogy milyen szót próbáljon meg illeszteni a shell, majd az `in` utáni részek többszöri ismétlésével megadható, hogy milyen mintára való sikeres illesztés esetén milyen utasításlista hajtódjon végre. Mind a minták, mind a szó kifejtésre kerül, de csak a 6.2.2 és a 6.2.3 szerint. A mintákban szereplő speciális védtelen karaktereket úgy viselkednek a szóra való illesztés közben, mint azt a 6.2.5-ban láttuk. Az első illeszthető ág kerül futtatásra és a visszatérési érték az ottani utolsó parancs visszatérési értéke. Ha egy ág sem illeszthető, akkor a visszatérési érték 0.

Gondoljuk meg, hogy miért volt szükség a mintában az `exit` utasítások zárójelbe tételére? Megfelelő lett volna zárójelezésre a `{ ...; }` formula?

10.4.5. `if lista; then lista; elif lista; then lista; ... else lista; fi`

Ennek a szerkezetnek az `elif` ágai elhagyhatóak, illetve bármennyi lehet belőlük és az `else` ág sem kötelező. Amikor az első `if` vagy `elif` utáni lista igaz értékkel tér vissza, akkor a hozzá tartozó `then`-ben foglalt parancsok végrehajtásra kerülnek, majd az egész `if` szerkezet visszatérési értéke felveszi az utoljára végrehajtott parancsét. Ha egyik ág feltétel listája sem tér vissza igazzal a kiértékelések során, akkor az `else`ághoz tartozó lista kerül végrehajtásra, a visszatérési értékre ugyanaz a szabály vonatkozik ekkor is.

```
1  #!/bin/bash
2
3  if test $(($1%3)) -eq 0
4  then
5    echo A $1 oszthato 3-mal.
6  else
7    echo A $1 nem oszthato 3-mal.
8  fi
```

10.4.6. `while lista; do lista; done`

A `while` ciklus esetében a `for`-ral ellentétben nem előre meghatározzuk a ciklus által bejárandó elemek sorozatát, hanem egy parancsot írunk, amit minden ciklusmag futtatása előtt kiértékel a shell a végrehajtást csak akkor folytatja, ha a kiértékelés eredménye igaz.

Mivel a `for` ciklus `in` opciójának elhagyása nem széleskörben ismert lehetőség, ezért az összes paraméter bejárását egészen máshogy, `while` ciklussal szokták megoldani, méghozzá `shift` utasítással, ami minden (1-nél nagyobb) pozícionális paramétert előrébb csúsztat eggyel és így természetesen az 1 paraméter értéke elveszik.

Vigyázzunk, mert a `while` és `shift` ilyen módon való használata lerombolja a paraméterlistánkat, csak egyszer lehet elsütni shellkörnyezetenként. Szerencsére új shellkörnyezetet egyszerűen indíthatunk a már tanult zárójelezéssel. Arra is figyeljünk, hogy ne a `[ "$1" != ""]` ciklusfeltételt használjuk, hiszen lehet valamelyik paraméter az üres sztring.

```
1  #!/bin/bash
2
3  p=0
4  # rossz: while [ "$1" != "" ]
5  # jo, csak bonyolult, magyarazzuk meg: while ( : ${1?} ) 2>/dev/null
6  ( while [ $# -ne 0 ]
7  do
8    p=$(($p+1))
9    shift
10 done
11 echo Parameterek szama: $p )
12 echo Elso parameter: $1
```

10.4.7. Gyakorlás

- Írjuk ki az első paramétert függőlegesen, illetve megfordítva!

```
#!/bin/bash

H=$(echo -n "$1" | wc -c)
for i in $(seq $H)
do
  echo "$1" | cut -c $i
done

for i in $(seq $H -1 1)
do
  echo -n "$(echo "$1" | cut -c $i)"
done
echo
```

- Az aktuális könyvtárban lévő fájlokat rendezzük soraik száma szerint!

```
#!/bin/bash

for i in *
do
  if test -f "$i"
  then
    wc -l "$i"
  fi
done | sort -n | cut -d" " -f2-
```

Feltéve legalább két fájl létezését, így is meg lehet oldani a feladatot:
`wc -l * 2>/dev/null | head -n-1 | sort -n | sed 's/^ *//' | cut -d\  -f2-`

A `wc` helyett a `grep`-et használva nem kell feltenni legalább két fájl létezését.[35]

Figyeljük meg, hogy mindegyik esetben a fájlneveket a két oszlopos, szóközzel elválasztott listából nem a `cut -d" " -f2`, hanem a `cut -d" " -f2-` utasítással nyertük ki, ugyanis a fájlnevekben előfordulhatnak szóközök, amik az első parancs esetében már a harmadik oszlop elejét jelentik, tehát nekünk az összes oszlopra szükségünk van a másodiktól kezdődően, nem csupán a másodikra.

- Mondjuk meg az aktuális könyvtárban lévő közönséges fájlok méretének összegét!

```
#!/bin/bash

sum=0
for i in *
do
  if test -f "$i"
  then
    sum=$(($sum+$(ls -l "$i" | tr -s ' ' | cut -d' ' -f5)))
  fi
done

echo Osszeg: $sum
```

[35] `grep -dskip -Hsc '.*' * | sed 's/^\(.*\):\(.*\)$/\2 \1/' | sort -n | cut -d\  -f2-`

Ebben a példában azt kell megfigyelni, hogy a parancsbehelyettesítés hamarabb végrehajtódik, mint az aritmetika kiértékelése, valamint az aritmetika kiértékelése közben a változóbehelyettesítések megtörténnek és így az összeadás valóban helyes.

11. Szövegek átalakítása és illesztése

A reguláris kifejezések olyan karaktersorozatként megadott minták, amik sztringek egy egész halmazát írják le önmaguk. Pl. az a reguláris kifejezés, hogy `.*a` egy végtelen halmazát írja le a karakterláncoknak, ugyanis erre a mintára az összes olyan karaktersorozat illeszkedik, ami az `a` betűre végződik. Ne keverjük össze ezeket a kifejezéseket a shell által a `case` utasításnál, illetve fájlnevek illesztésekor használt globbinggal(6.2.5)!

A reguláris kifejezések használatát újabban közvetlenül a `bash` is támogatja, azonban a shell ezen része még most is kialakulóban van, évről–évre változik, hogy mi mit jelent, így ennek a tárgyalásával nem foglalkozunk, hanem a régóta létező és kiforrott `grep`, `sed` és `find` parancsok reguláris kifejezéseit nézzük majd meg.

Megjegyezzük, hogy ezen kifejezések értelmezése, az illeszkedő halmaz meghatározása, illetve egy konkrét sztringről a halmazba tartozás eldöntése nagyon komoly információtechnológiai problémák, a témakör a formális nyelvek és automaták nevet viseli. Ezek az ismeretek szükségesek fordítóprogramok, illetve olyan (egyszerűnek tűnő) interpreterek készítéséhez is, mint amilyen a `bash` shellünk.

11.1. `tr`

A reguláris kifejezések tárgyalása előtt egy egyszerű, de mégis nagyon sok feladat elvégzését lehetővé tévő programra, a `tr`-re hívjuk fel a figyelmet pár példán keresztül. Az olvasó a `man 1 tr` paranccsal maga elolvashatja a részletes tudnivalókat erről a programról.

tr

```
1  errge@pandora:~$ echo "a  b c          d" | tr -d " "
2  abcd
3  errge@pandora:~$ echo "a  b c          d" | tr -s " "
4  a b c d
5  errge@pandora:~$ echo "BalmafaE" | tr '[:lower:]' '[:upper:]'
6  BALMAFAE
7  errge@pandora:~$ echo "bALMAFAe" | tr '[:upper:]' '[:lower:]'
8  balmafae
9  errge@pandora:~$ cat test
10 egy-maganal volt a gyilkos fegyver
11
12
13
14
15 ketto-csipkebokor vesszo
16 errge@pandora:~$ cat test | tr -s '\n'
17 egy-maganal volt a gyilkos fegyver
18 ketto-csipkebokor vesszo
19 errge@pandora:~$
```

A man oldal elolvasása után már érteni fogjuk, hogy miért működik a `\n`-es utolsó példa, de miért is kell a `\n`-t aposztrófok közé tenni? Idézőjelekkel is működne ez a parancssor aposztrófok helyett? Gondolkodjuk el ezeken a kérdéseken, majd ellenőrízzük álláspontunkat egy számítógép előtt!

11.2. A reguláris kifejezések

A reguláris kifejezések egyszerű építőelemekből és ezeken értelmezett műveletekből állnak, akár csak az aritmetikai kifejezések. A legegyszerűbb ilyen építőelemek a kis- és nagybetűk, valamint a számok, amik egyszerűen önmagukat jelentik, mint minták. A `.` jelentése már speciális, vele egy tetszőleges karaktert jelölünk a mintában.

11.2.1. Karakterosztályok, a szögletes zárójel

Karakterosztályt úgy hozhatunk létre, hogy egy szögletes zárójelpáron belül felsoroljuk a karaktereket. Egy ilyen építőelem bármilyen olyan karakterre illeszkedik, ami fel van sorolva, illetve bármilyen olyanra, ami nincs fel sorolva, ha a felsorolást a `^` karakterrel kezdjük. Például a `[0123456789]` egy tetszőleges számjegyre, míg a `[^.[:lower:]]` bármire, ami nem pont vagy kisbetű illeszkedik. A szögletes zárójelen belül használhatjuk a már megismert intervallumos kifejezéseket, illetve osztályneveket.

11.2.2. Sztring elejének és végének jelölése

A `^` és a `$` karakterek az üres karakterláncra illeszkednek, de csak a sztring elején, illetve végén. Azaz, ha a reguláris kifejezésünkben a két jel valamelyikét használjuk, azzal azt mondjuk, hogy a kifejezés csak akkor illeszkedik, ha a jel helyén van a karakterlánc eleje, illetve vége.

11.2.3. A backslash jelentése

A backslash (\) jel többféle szerepet tölt be. Egyrészt az ismeretett speciális karakterek (`.`, `*`, `^`, stb.) elé írva azok elvesztik speciális jelentésüket, másrészt pár egyébként saját magát jelentő jel pont attól nyer speciális jelentést, hogyha előtte a backslash szerepel. Pl. a `\<`, illetve `\>` olyan minta, ami csak szó elején, illetve végén illeszkedik az üres karakterláncra. Nézzünk utána további ilyen hasznos jelölőknek a `grep` man oldalában (`\w`, `\W`, `\b`, `\B`)!

11.2.4. Ismétlés

Egy minta után tett `\?` a mintát opcionálissá, a `\+` kötelezővé, de több ismétlődést is megengedővé, a `*` opcionálissá és több ismétlődést is megengedővé teszi. Figyeljük meg, hogy a `*` a sűrűn használt művelet, ezért azt nem kell, hogy \ előzze meg.

Lehetőség van pontos ismétlődésszám megadására is a `\{` és `\}` operátorok használatával, további információért olvassuk el a man oldalt!

11.2.5. Konkatenáció

A bemutatott építőelemeket egymásután írhatjuk, ekkor olyan mintát hozunk létre, ami abban az esetben illeszkedik, ha az egymásután írás sorrendjében illeszthetőek a részek a karakterlánc egymásutáni részeire. Pl. az `ab*.` reguláris kifejezés illeszkedik az `dabbbd` karakterláncra, de nem az `acb`-re, ugyanis az előbbinek van olyan része (az `abbbd`), ami felvágható úgy, hogy az első részre az `a`, a másodikra a `b*`, a harmadik a `.` minta illeszkedik.

11.2.6. Alternáció

Ha egy reguláris kifejezésben két (vagy több) minta közé a `\|` jelet tesszük, azzal kifejezhetjük, hogy a felsorolt minták közül pontosan az egyiknek kell illeszkednie.

11.2.7. Precedencia

Az előző három operátor precedenciája a megismerésük sorrendjével azonos, azaz az ismétlést előíróak kötnek a legerősebben és az alternáció a legkevésbé. Ezt a szabályt zárójelezéssel, méghozzá a `\(` és `\)` zárójelek használatával bírálhatjuk felül. Vegyük észre, hogy erre olyan ritkán van szükség, hogy a reguláris kifejezésekben a zárójelezést backslash kell, hogy megelőzze.

Amennyiben zárójelezést használunk, akkor a `\n` (ahol `n` egy számjegy) kifejezés az első 9 zárójelezett részre illesztett sztring valamelyikét jelenti.

11.2.8. Példák

minta	illeszkedő karakterláncok
`.*`	bármi
`^.*$`	bármi
`^[[:upper:]]\{7\}\.ELTE$`	az XXXXXXX.ELTE alakú hallgatói azonosítók
`[[:upper:]]\{7\}\.ELTE`	hallgatói azonosítót tartalmazó bármilyen sztring
`...`	legalább három hosszú sztring
`^\(.\+\)\1$`	két azonos sztringből álló konkatenációk
`^\(.*\)\1$`	mint előbb, de az üres sztring is
`^[[:upper:]]\{7\}\(:[0-9]\+\)*$`	{XYZXYZQ.ELTE:12:0,QWERTYU.ELTE,...}

11.3. grep

A `grep`, az `fgrep`-hez hasonlóan bizonyos kritériumoknak megfelelő sorokat ír a képernyőre fájlokból vagy a standard bemenetről. A különbség, hogy a `grep` esetén a kritérium nem csupán az lehet, hogy egy részkarakterlánc forduljon elő a sorban, hanem egy tetszőleges reguláris kifejezést próbálhatunk meg illeszteni minden sorra.

Fontos opciók a `-c`, `-v`, `-h`, `-H`, `-x`, `-i`, `-L`, `-l`, `-q`, `-o`, `-A`, `-B` és `-C`, ezek jelentését mindenképp nézzük meg a dokumentációban!

A `grep` programhoz a man oldalán túl egy jobb dokumentáció is elérhető ún. texinfo formátumban, ami a jegyzet írásának pillanatában (az elvhű Debian fejlesztők és a szintén elvhű FSF miatt) a Pandorán nem érhető el, azonban megtalálható a http://www.gnu.org/software/grep/doc/grep.html címen, illetve egy PDF verziója a jegyzet weblapjáról is letölthető (http://www.gergely.risko.hu/progalap1/manpages/grep-info.pdf).

Gyakorlásképp írjuk ki az alma.txt fájl azon sorait, melyekben

- van számjegy,[36]
- csak számjegy van,[37]
- a sor elején 2 egyező karakter van,[38]
- van `xyxy` (vagy `xxxx`) alakú szó,[39]
- van csupa nagy betűből álló szó (tehát rövidítés),[40]
- van legalább 4, de nincs 8 hosszú szó.[41]

[36] `cat alma.txt | grep '[0-9]'`
[37] `cat alma.txt | grep '^[0-9]\+$'`
[38] `cat alma.txt | grep '^\(.\)\1'`
[39] `cat alma.txt | grep '\<\(\w\w\)\1\>'`
[40] `cat alma.txt | grep '\<[[:upper:]]*\>'`
[41] `cat alma.txt | grep '\<\w\{4,\}\>' | grep -v '\<\w\{8\}\>'`

Írjunk egy parancsfájlt, ami eldönti a megadott paraméterekről, hogy azok szabályos hallgatói azonosítók-e! Adjon vissza annyit, amennyi szabálytalan van köztük, azaz pont igazat adjon vissza, ha mind szabályos. Nem kell azzal törődni, hogyha 256 szabálytalannal hívják meg a programot, akkor újra igazat ad vissza, 257-nél meg 1-et, tegyük fel, hogy ennyi paramétert nem adnak neki soha!

11.4. sed, a stream editor

A `sed` programmal előre megadott parancsokkal szerkeszthetünk szöveges folyamokat. A legnépszerűbb felhasználása természetesen az, hogy szűrőként építjük a parancssorunkba és a szerkesztéseket ekkor a standard bemeneten végzi és a standard kimenetre írja. Paraméterek megadásával azonban az is elérhető, hogy a bemenetetét már létező fájlokból olvassa[42], sőt a `-i` paraméterrel a megadott fájlokat helyben szerkeszti[43]. A `sed` programozási nyelvén szinte bármilyen probléma (kellően csúnyán és bonyolultan) megoldható[44] így az összes többi programhoz hasonlóan csak egy töredékét tudjuk ismertetni a lehetőségeknek.

A man oldalon kívűl egy jobb dokumentáció is elérhető, ugyanúgy texinfo formátumban, mint a `grep`-hez. A Pandorán az `info sed` hozza elő ezt a dokumentációt, de megtalálható a http://www.gnu.org/software/sed/manual/ címen is, illetve egy PDF verziója a jegyzet weblapjáról letölthető (http://www.gergely.risko.hu/progalap1/manpages/sed-info.pdf).

11.4.1. A sed működése

A `sed` kezdetben két üres tárterülettel (bufferrel) indul, a minta bufferrel és a tároló bufferrel. A tároló bufferrel a jegyzet nem foglalkozik, trükkösebb `sed` programoknál van rá szükség.

Az input minden egyes sorára a következő műveletsort hajtja végre:

- beolvassa a sort a bemenetről és levágja a sorvég jelet, ha van;
- az így kapott sort a minta bufferbe teszi (a régi tartalmat törölve);
- végrehajtja azokat a parancsokat, amiknek a feltétele igaz;
- ha a `-n` opció nem volt megadva, akkor kiírja a minta buffer tartalmát (kiírva a korábban elvett újsor jeleket is).

Ha eléri a fájl végét, a `sed` kilép.

11.4.2. Feltételek, avagy címek

Mint az a végrehajtási cikluson látszik, minden parancshoz megadható feltétel, amit címnek is szokás hívni, mert az adott feltétel megcímez bizonyos sorokat, amikre a feltételhez tartozó parancsok végrehajtódnak. Cím megadása után `{ parancs1 ; parancs2 ; ...}` módon több parancs is felsorolható, elkerülve így a feltétel állandó ismételgetését. Az üres feltétel az igazat jelenti, tehát ha nincs megadva cím, akkor a parancs minden sorra végrehajtódik.

Ha van megadva feltétel, akkor csak a feltételt teljesítő sorokon hajtódnak végre a parancsok.

Két címet vesszővel elválasztva intervallum is megadható, ami ott kezdődik, ahol a vessző előtti feltétel igaz és ott végződik, ahol a vessző utáni feltétel igaz. Beleértjük az intervallumba a határokat is. A záró határ lehet `+N` alakú is, ahol `N` egy szám, ekkor ez azt a sort jelenti, amit kezdő határhoz `N` sort hozzáadva kapunk.

[42] Amit persze egyszerűen egy elé írt `cat`-tel is megoldhatunk.

[43] természetesen közben létrehoz egy átmeneti fájlt, de ezzel nekünk nem kell foglalkoznunk

[44] A texinfo dokumentáció ad is példát a `tac`, `head`, `tail`, `uniq`, `wc` és `cat` megvalósítására `sed`-ben.

Feltétel tagadása a feltételt követő felkiáltójellel lehetséges.
Fontosabb feltételtípusok:

- `N` (ahol `N` egy (akár többjegyű) szám): csak az `N`-edik sor feldolgozásakor igaz;
- `$`: az utolsó sor feldolgozásakor igaz csak;
- `/regkif/`: csak akkor igaz, ha a `regkif` által leírt (a per jeleket levédve tartalmazó) reguláris kifejezés illeszkedik az aktuálisan sorra.

11.4.3. Egyszerű `sed` parancsok

- `q`: kiírja az aktuális minta buffert[45], majd kilép;
- `d`: törli az aktuális minta buffert és rögtön a következő sor végrehajtásával folytatja;
- `p`: kiírja az aktuális minta buffert (`-n` használata esetén jön jól);
- `n`: lecseréli a minta buffert a következő input sorra, de előtte kiírja, ha az automatikus kiírás a `-n`-nel nincs letiltva;

```
errge@pandora:~/tmp$ seq 1 3 | sed ''
1
2
3
errge@pandora:~/tmp$ seq 1 3 | sed -n ''
errge@pandora:~/tmp$ seq 1 3 | sed -n 'p'
1
2
3
errge@pandora:~/tmp$ seq 1 3 | sed '2d'
1
3
errge@pandora:~/tmp$ seq 1 7 | sed '/2/,+3d'
1
6
7
errge@pandora:~/tmp$ seq 1 7 | sed '2,/5/!d'
2
3
4
5
errge@pandora:~/tmp$ seq 1 7 | sed '3q'
1
2
3
errge@pandora:~/tmp$ seq 1 3 | sed -n 'p;1n'
1
3
errge@pandora:~/tmp$ seq 1 3 | sed -n 'p;1d'
1
2
3
errge@pandora:~$ seq 1 4 | sed '${p;p;p}' | tr '\n' ' ' ; echo
1 2 3 4 4 4 4
```

[45] Ha a `-n` opció nincs megadva.

11.4.4. Az s/MIT/MIRE/OPCIÓK parancs

Ez az a parancs, ami közismert a sed-ből, a legtöbb felhasználója a sed-nek az előbb említett parancsokat, sőt a címzések lehetőségét nem is ismeri. Végrehajtásakor a minta buffer első olyan része, amely illeszkedik a MIT reguláris kifejezésre kicserélődik a MIRE részre. Bármely részbe úgy írhatunk önmagát jelentő pert, ha backslash-sel levédjük.

A MIRE részben szerepelhet a már ismert \1, \2, stb. értékeken kívűl az & jel is, aminek jelentése a teljes MIT részre illeszkedő karakterlánc. Ezekből az is következik, hogyha \ vagy & jelet szeretnénk használni a MIRE részben, akkor azt backslash-sel le kell védeni.

A különböző opciókat csak egymásután kell írni. A fontosabbak:

- N, ahol N egy (akár többjegyű) szám: nem az első, hanem az N. illeszkedő rész cseréje;
- g: az összes illeszkedő rész cseréje, nem csak az elsőé;
- i: a minta illesztése a kis- és nagybetű különbségek figyelmenkívűl hagyásával.

Fontos megjegyezni, hogy a reguláris kifejezések mindig mohók, ami azt jelenti, hogy addig illesztenek, amíg csak tudnak, pl. a alma:szilva:mogyoro sztringre illeszkedik a ^\(.*\):.*$ kifejezés, méghozzá úgy, hogy a \1 „értéke” alma:szilva lesz és nem csak alma.

11.4.5. Escape szekvenciák

Eddig a sed programokban a backslash szerepe az volt, hogy a speciális jelentéssel bíró karaktereket védje (pl. \^ vagy *). Azonban a backslash-t lehet használni ún. escape szekvenciák létrehozására is, itt pont a backslash miatt nyer speciális szerepet egy jel. Pl. a \n jelentése egy újsor karakter, míg a \t jelentése egy tabulátor, a \xXX jelentése (ahol XX egy hexadecimális szám) pedig az XX kódú karakter. A sed programjainkba ezeket a speciális karaktereket egyszerűen be is írhatnánk escape szekvenciák használata nélkül, azonban ez elrejtené őket, kevésbé lenne átlátható a kód, illetve a ASCII 0-ás karaktert nem egyszerű feladat beírni egy szkriptbe.

Az említett három szekvenciát az echo is támogatja, azaz pl. így egyszerre tudunk kiírni több soros szöveget is, de először a -e kapcsolóval engedélyeznünk kell az értelmezésüket.

További információ a sed texinfo dokumentációjában az „Escapes” fejezetben, illetve a bash-builtins man oldal echo fejezetében található.

11.4.6. Gyakorló feladatok

- Írjunk parancsfájlt, ami összeadja a standard bemenetén soronként kapott számokat![46]
- A barack.txt fájlban ha a sor elején 2 egyező karakter van, azokat cseréljük a sor végén lévő jelre.[47]
- A cut tárgyalásakor megjegyeztük, hogy hiába cseréljük a (mondjuk kettősponttal elválasztott) mezők felsorolását az opciólistában, attól a kiíráskori sorrendjük nem változik. Hogyan cserélhetnénk akkor meg egy kimenetben minden sor 1. és 2. oszlopát, feltéve, hogy a fájl minden sora két oszlopos?[48]
- A 3. oszlopot és az utolsót, feltételezésekkel nem élve?[49]
- Adjuk meg a recept.txt fájl leggyakoribb betűjét, ha a kis- és nagybetűk között nem teszünk különbséget![50]

[46] `echo $(($(tr '\n' + | sed 's/+$//')))`

[47] `sed -i 's/^\(.\)\1\(.*\)\(.\)$/\3\2\3/' barack.txt`

[48] `sed 's/^\(.*\):\(.*\)$/\2:\1/'`

[49] `sed 's/^\(\([^:]*:\)\{2\}\)\([^:]*\):\(.*\):\(.*\)$/\1\5:\4:\3/'`

[50] `sed 's/./&\n/g' barack.txt | tr '[:upper:]' '[:lower:]' | grep '[[:lower:]]' | sort | uniq -c | sort -n | tail -n1 | sed 's/[^a-z]//g'`

12. find

A fájlok interaktív listázására az `ls` parancs kitűnően megfelelt, azonban parancsfájlokban való használatra nem volt túlságosan alkalmas. Nehézkes volt pl. a méret oszlop megszerzése, előtte a szóközöket `tr`-rel össze kellett vonni, aztán le kellett számolni, hogy pontosan melyik oszlopra van szükségünk, a szóközös fájlok kezelése pedig extra figyelmet igényelt.

A `find` ezekre a problémákra nyújt sokkal jobb megoldást, itt röviden mutatjuk be és most is van a man oldalon kívűl texinfo dokumentáció is, ami az `info find` paranccsal, illetve HTML[51] vagy PDF[52] formában is elérhető.

Használata: `find KÖNYVTÁRAK-ÉS-FÁJLOK... OPCIÓK TESZTEK-ÉS-TEVÉKENYSÉGEK...`

Vegyük észre, hogy a `find` kilóg a többi parancsunk sorából, mivel ez a könyvtárak és fájlok nevét, amin dolgoznia kell a parancssor elején várja és máshol nem is fogadja el. Alapértelmezés az aktuális könyvtár, így a `find . -opc1 -opc2 ...` helyett `find -opc1 -opc2 ...` is írható. Azért is kilóg a `find` a többi parancsunk közül, mert az opciók nevét egész szavas, hosszú opciók esetén is csak egy `-` jel kell és szabad, hogy megelőzze.

Az opciók (options) a `find` teljes lefutását befolyásolják, így azokat közvetlenül az argumentumok után kell felsorolni, míg a tesztek (tests) és a tevékenységek (actions) egyszerűen igaz vagy hamis értékkel bírnak minden fájlrendszerbejegyzésre (továbbiakban fájlra) nézve. Az operátorok segítségével a tesztek és a tevékenységek összekapcsolhatóak, így nagyobb logikai kifejezések írhatóak. A tevékenységeknek a tesztekkel ellentétben mindig van mellékhatásuk is (pl. letörölnek egy fájlt, kiírják a nevét vagy méretét, stb.).

A find alapértelmezés szerint rekurzívan listáz és csak a megtalált fájlrendszerbejegyzések nevét írja ki, soronként egyet. A rekurzív listázás maximális és minimális mélysége a `-maxdepth` és `-mindepth` argumentumot váró opciókkal állítható, pl. a `find -maxdepth 1 -mindepth 1` parancs pontosan ugyanazt csinálja, mint a már ismert `ls -1`.

Fontos opció még a `-regextype`, mivel a find alapértelmezés szerint más fajta reguláris kifejezéseket használ, mint amilyeneket a `sed`-nél és a `grep`-nél megtanultunk, ezért ha olyan parancssort használunk, ami reguláris kifejezéseket használ, akkor a parancssor `OPCIÓK` részében adjuk meg a `-regextype posix-basic` opciót.

12.1. Tesztek (tests)

A számmal paraméterezhető tesztek esetén a szám megadható `+N` vagy `-N` formában is, ahol az előbbi az `N`-nél szigorúan nagyobb, míg utóbbi a kisebb értékeket jelenti. A fontosabb tesztek:

- `-false`: mindig hamis,
- `-true`: mindig igaz,
- `-empty`: üres könyvtár vagy fájl,
- `-group GROUP`: a fájl tulajdonosi csoportja `GROUP`,
- `-user USER`: a fájl tulajdonosa `USER`,
- `-name MINTA`: a fájl neve illeszkedik a `MINTÁ`-ra globbing értelemben (6.2.5)[53],
- `-iname MINTA`: mint az előző, de a kis- és nagybetű különbségeket figyelmen kívűl hagyva,
- `-wholename MINTA`: a fájl teljes elérési útja illeszkedik a `MINTÁ`-ra, mint előbb,

[51] A http://www.gnu.org/software/findutils/manual/find.html címen.
[52] A http://www.gergely.risko.hu/progalap1/manpages/find-info.pdf címen.
[53] A rejtett fájlokat is figyelembe veszi.c

- `-iwholename MINTA`,
- `-regex REGEX`: az `REGEX` reg. kifejezés illeszkedik a fájl teljes elérési útjának egészére[54],
- `-size Nc`: a fájl mérete `N` bájt,
- `-type c`: a fájl típusa `c`, ahol a `c` lehet `d` és ekkor könyvtárat jelent vagy `f` és ekkor hagyományos fájlt.

12.2. Tevékenységek (actions)

A fontosabb tevékenységek:

- `-print`: mindig igaz, és kiírja az aktuális fájlnevet (ez alapértelmezés, ha nincs más tevékenység megadva),
- `-printf FORMA`: mindig igaz, és kiírja a fájl tulajdonságait a megadott `FORMÁ`-nak megfelelően, amiben a következő speciális karakterek lehetnek:
  - `\n`: újsor karakter, ui. nincs új sor a `-printf`-fel való kiírás végén automatikusan,
  - `%s`: a fájl mérete bájtokban,
  - `%f`: a fájl neve,
  - `%p`: a fájl elérési útja,
  - `%%`: egy `%` karakter.
- `-exec PARANCS ;`: a PARANCS-ot végrehajtja (és a parancs visszatérési értéke ezen tevékenység igazságértéke), úgy, hogy a PARANCS-ban előforduló `{}` jelsorozatokat az aktuális fájl elérési útjára cseréli. Ne feledkezzünk meg arról, hogy a `;` speciális a shell számára, így azt le kell védeni, hogy a `find` egyáltalán megkapja, mint paramétert.[55]

  Pl. letörölhetjük az aktuális könyvtár összes olyan fájlát, ami legalább egy `a` betűt tartalmaz a következő paranccsal:
  `find -maxdepth 1 -mindepth 1 -type f -name '*a*' -exec rm {} \;`

Példa, fájlméretek összeadása:

```
1  errge@pandora:~/tmp$ cat sum.sh
2  #!/bin/bash
3
4  echo $(($(tr '\n' + | sed 's/+$//')))
5  errge@pandora:~/tmp$ find -maxdepth 1 -mindepth 1  -type f -printf '%s:%p\n'
6  0:./fajl1
7  0:./.a
8  36:./output
9  17:./xxx
10 125:./rendez-sorszerint.sh
11 2:./x  b
12 117:./sdfg:sdfg
13 177:./megfordit.sh
14 284:./test.sh
15 69:./sum.sh
16 82:./barack.txt
17 errge@pandora:~/tmp$ find -maxdepth 1 -mindepth 1  -type f -printf '%s\n' | ./sum.sh
18 909
```

[54] Tehát ha az `a` vagy `b` karaktert az elérési útban bárhol tartalmazó fájlokra vagyunk kíváncsiak, akkor a `-regex '[ab]'` teszt helyett a `-regex '.*[ab].*'` tesztet kell használni.

[55] Rendszergazdajelöltek mindenképp nézzék meg az `-execdir` tevékenységet is!

12.3. Operátorok

Az ismeretett teszteket és tevékenységeket a következő operátorokkal köthetjük össze (csökkenő precendencia sorrendben):

- `( find-kifejezés )`: zárójelezés, a helyes precedencia-sorrend kikényszerítésére. Ne feledkezzünk meg róla, hogy a zárójeleket a shell elől levédjük, hiszen ő is használ zárójeleket! Például a `find ( -print )` parancs helytelen, míg a `find \( -print \)` helyes. Ugyanakkor ne gondoljuk, hogy a problémát megoldhatjuk az egész kifejezés aposztrófok közé írásával, ugyanis a paraméterek szavakra bontására a `find` önállóan nem képes, azt a shellnek kell elvégeznie;
- `! find-kifejezés`: tagadás, a kifejezés igazságértékének megfordítása. Gondoljuk meg, hogy mit csinál a `find -print -print` és mit csinál a `find \! -print -print` parancs;
- `find-kif1 find-kif2` vagy `find-kif1 -a find-kif2`: kifejezések logikai konjunkciója (ése). A shellhez hasonlóan a find lusta, ha az első kifejezés meghiúsul, akkor a másodikat már meg sem próbálja. Vegyük észre azt is, hogy ez az operátor az alapértelmezett, ha nem írunk semmit a tevékenységeink és teszteink közé. Ezért kapjuk azt a természetes működést, hogy a `-type f -size 5 -owner errge` opciósorozat a jegyzet szerzőinek 5 bájtos fájlait jelenti, anélkül, hogy `-a` operátorokat is be kellene szúrnunk;
- `find-kif1 -o find-kif2`: diszjunkció (vagy) operátor, szintén rövidzár kiértékeléssel.

12.4. Gyakorló feladatok

- Listázzuk ki az aktuális könyvtár összes reguláris fájlának nevét!

```
1  errge@pandora:~/tmp/A$ find . -maxdepth 1 -type f
2  ./xxx
3  ./x b
4  ./.a
5  errge@pandora:~/tmp/A$ find . -maxdepth 1 -type f \! -name '.*'
6  ./xxx
7  ./x b
8  errge@pandora:~/tmp/A$ find . -maxdepth 1 -type f \! -name '.*' -printf '%f\n'
9  xxx
10 x b
```

- Add meg a bejelentkezési könyvtár bármely mélységében lévő `.html` dokumentumok közül a legnagyobb méretűnek a nevét![56]
- Add meg az összes említett `.html` dokumentumok közül azokat a méretükkel együtt, amelyek legalább 100 bájt hosszúak![57]
- Írj parancssort, ami a `DIR` környezetváltozóban adott könyvtárban lévő alkönyvtárak közül kiírja azok nevét, melyekben van legalább 5 közönséges fájl.[58]
- Írj parancssort, ami az aktuális könyvtárban lévő összes `.txt` kiterjesztésű fájl tartalmát a `nagy.sum` nevű fájlba írja, úgy, hogy minden fájlt megelőz egy egysoros fejléc, ami a fájl nevét tartalmazza.[59]

[56] `find ~ -name '*.html' -printf '%s:%f\n' | sort -n | tail -n1 | cut -d: -f2-`

[57] `find ~ -name '*.html' -size +99c -printf '%s %f\n'`

[58] `find $DIR -mindepth 2 -maxdepth 2 -type f -printf '%h\n' | uniq -c | grep -v '^ *[1-4] ' | cut -d/ -f2`

[59] `find . -maxdepth 1 -type f -name '*.txt' -printf 'Nev: %f\n' -exec cat {} \; >nagy.sum`

- Írj parancssort, ami az aktuális könyvtárban közvetlenül lévő fájlok közül kiírja azok nevét, melyek alakja `unixPQR`, ahol `P`, `Q` és `R` számjegyek, és még az is teljesül, hogy `PQR`, mint 3-jegyű szám hárommal osztható!

  A feladat első megoldása során egy sokszor használt, haladónak nevezhető trükkhöz folyamodunk. Azt fogjuk csinálni, hogy program által előállítunk olyan parancsokat, amik pont a kívánt feladatot érik el. Ezt a parancsokat tartalmazó kimenetet utána már csak bele kell irányítanunk a parancsfeldolgozóba, hogy az végre is hajtsa!

```
errge@pandora:~/tmp/A$ find -maxdepth 1 -name 'unix[0-9][0-9][0-9]' -type f
./unix123
./unix133
./unix000
./unix998
errge@pandora:~/tmp/A$ find -maxdepth 1 -name 'unix[0-9][0-9][0-9]' -type f \
  -printf '[ $(($(echo %f | cut -c5-) %% 3)) -eq 0 ] && echo %f\n'
[ $(($(echo unix123 | cut -c5-) % 3)) -eq 0 ] && echo unix123
[ $(($(echo unix133 | cut -c5-) % 3)) -eq 0 ] && echo unix133
[ $(($(echo unix000 | cut -c5-) % 3)) -eq 0 ] && echo unix000
[ $(($(echo unix998 | cut -c5-) % 3)) -eq 0 ] && echo unix998
errge@pandora:~/tmp/A$ find -maxdepth 1 -name 'unix[0-9][0-9][0-9]' -type f \
  -printf '[ $(($(echo %f | cut -c5-) %% 3)) -eq 0 ] && echo %f\n' | bash
unix123
unix000
errge@pandora:~/tmp/A$
```

  Egy új szintaktikai apróságot is tanultunk, ha a sor végére rakunk egy backslasht és rögtön újsort kezdünk, akkor azzal levédtük az újsor karaktert, azaz nem jelenti a parancs végét többé és így a parancsunkat a következő sorban tudjuk folytatni. A való életben persze erre nincs szükség sűrűn, hiszen bármilyen hosszú sorokat be tudunk gépelni, azonban nyomtatásban szükség lehet rá, illetve a saját programkódjaink is átláthatóbbak, ha nincsenek benne nagyon-nagyon hosszú sorok.

- A második megoldás egy parancsfájl lesz, ami szokványosabb és egyáltalán nem épít a `find`-ra, egy `for unix[0-9][0-9][0-9]` ciklust fog használni és a ciklusmag mindig csak azt vizsgálja, hogy jó (3-mal osztható) sort vizsgálunk éppen.

```
#!/bin/bash
shopt -s nullglob
for i in unix[0-9][0-9][0-9]
do
  if [ -f $i ] && [ $(($(echo $i | cut -c 5-) % 3)) -eq 0 ]
  then
    echo $i
  fi
done
```

  A parancsfájl első parancsának hatására amennyiben egy globbing (6.2.5) nem illeszkedik, akkor nem a nyers minta, hanem az üres sztring a kifejtés eredménye és így a `for` ciklus nem csinál semmit. Más szóval az a sor csak azért kell, hogy a parancsfájl működjön `unixPQR` fájlt egyáltalán nem tartalmazó könyvtárakban is. Persze ez a probléma máshogy is kezelhető, pl. a ciklusmagban ellenőrízhetnénk, hogy a `$i` a `unix[0-9][0-9][0-9]` sztring e és ha igen, akkor rögtön kiléphetnénk egy `exit 0` utasítással. Írjuk át ilyenre a parancsfájlt és vegyük ki az `shopt` kezdetű sort, majd ellenőrízzük egy pl. üres könyvtárban, hogy ekkor valóban semmilyen hiba nem történik!

13. read

A read-del a standard bemenetről olvashatunk egy sort és ezt ciklusba szervezve olyan feladatokat, amiket csak különböző trükkökkel tudtunk megoldani, szépen, átláthatóan programozhatunk le shellben. A read parancsot nem lehet külön programként megvalósítani, csak a shell részeként, ezért arról (és a többi bash-be épített parancsról) a man 7 builtins paranccsal kérhetünk részletesebb információt.

Legegyszerűbb és az általunk egyetlen megemlített felhasználási módja a read KÖRNYVÁLT forma, aminek hatására az olvasott sor bekerül a KÖRNYVÁLT valtozóba és a visszatérési érték 0, amennyiben még nem értük el a fájl végét. Ha elértük, akkor a KÖRNYVÁLT érétéke üres lesz és a read visszatérési értéke nem 0.

Fontos megjegyezni, hogy a while read ciklusok sokszor szükségesek, de mindig érdemes elgondolkodni, ugyanis ha ki tudjuk kerülni a használatukat, akkor általában a programunk erőforrásigénye (és így a futási idő is) radikálisan lecsökken.

13.1. Példák

A standard inputon érkező számok összeadása (sum.sh) elég trükkös volt korábban. Most azonban könnyebben megoldhatjuk: a sum környezetváltozót lenullázunk, majd egy while ciklussal kombinált read utasítással soronként beolvassuk a számokat és mindet hozzáadjuk a sum aktuális értékéhez, melyet a ciklus után kiírunk.

sum.sh

```
1  #!/bin/bash
2
3  sum=0
4  while read SOR
5  do
6    sum=$(($sum+$SOR))
7  done
8  echo $sum
```

Vegyük észre, hogy a ciklusfeltételül megadott read akkor ad először hamisat vissza, amikor az input fájl végét elérte és így pont jókor lép ki.

Készítsünk egy következő programot, ami az első paraméterként kapott fájl sorait egyesével bekeretezi pont akkora kerettel, amibe belefér, majd egymás alatt kiírja a keretezett szövegeket. Ügyeljünk a szóközöket tartalmazó paraméterekre! Így nézzen ki:

```
1  errge@pandora:~/tmp/A$ cat szilva.txt
2  szilva
3  korte      dio
4  mogyoro
5  errge@pandora:~/tmp/A$ ./keretez.sh szilva.txt
6  +------+
7  |szilva|
8  +------+
9  +--------------+
10 |korte      dio|
11 +--------------+
12 +-------+
13 |mogyoro|
14 +-------+
```

A megoldásban egyszerűen soronként olvassuk az inputot, majd minden sorhoz elkészítjük a keret felső sorát, a középső sorát a tartalommal és az alsó sorát!

keretez.sh

```
#!/bin/bash

cat "$1" | while read SOR
do
  echo -n + ; echo "$SOR" | sed 's/./-/g;s/$/+/'
  echo "|$SOR|"
  echo -n + ; echo "$SOR" | sed 's/./-/g' | sed 's/$/+/'
done
```

Avagy: `cat $1 | sed -n 'p;p;p' | sed '2~3!{s/./-/g;s/^/+/;s/$/+/;};2~3{s/^/|/g;s/$/|/g}'`

A `read` parancsnak több változónevet megadva az olvasott sort az `IFS` környezetváltozóban lévő karakterek szerint[60] szétválasztja és az előfordulásuk sorrendjében hozzárendeli a felsorolt változókhoz, az utolsó környezetváltozóba kerül a sor megmaradt része. Tehát az eddig látott `while read VÁLTOZÓ` alak egyszerűen ennek az esetnek speciális előfordulása, ahol az egész sor „hátramaradtnak" számít.[61]

Felhasználva ezt szép kódot lehet írni, kényelmesen és tökéletesen (a dupla szóközök is megmaradnak) feldolgozható az `ls` kimenete:

```
errge@pandora:~/tmp$ ls -l --time-style=long | tail -n +2 | \
  while read JOGOK LINKEK TULAJ CSOPORT MERET DATUM IDO NEV ; do echo "$MERET:$NEV" ; done
6:alma korte
0:atmenetifajl
2:x  b
```

Mégsem szokás ezt a lehetőséget felhasználni, hiszen az egész sor később is feldolgozható a már tanult `tr` és `cut` parancsokkal, és ugyanez a hatás elérhető. Mi sem ezért említjük ezt meg, hanem azért, mert ebből következik a `read`-nek egy idegesítő mellékhatása: gondoljuk át azt az esetet, ha egy sor szóközökkel (az elválasztó karakterrel) kezdődik, pl. be szeretnénk keretezni magát a `keretez.sh`-t! Ezek a kezdő szóközök eltűnnek, hiszen az első oszlop csak utánuk kezdődik és attól kezdve kell mindent elrakni a `SOR` nevű változóba. Az utolsó oszloptól már megmaradnak az egymásutáni szóközök, de előtte nem, pont ezért működött az `ls` feldolgozásai is. Próbáljuk ki, a keretezésben nem lesznek meg a sorkezdő szóközök! A hiba úgy orvosolható, ha az `IFS`-t csak a `read` futtatásának az idejére üresre állítjuk, azaz az első sort erre módosítjuk: `cat "$1" | while IFS="" read SOR`

Azt is figyeljük meg, hogy milyen fontos, hogy a `$SOR` változóbehelyettesítéseket idézőjellel védjük! Egyrészt emiatt lesz a keret helyes sok szóköz előfordulásának esetén is, másrészt a keret közepének első és utolsó karaktere shell környezetben védés nélkül a csővezetéképítést jelenti, azaz védés nélkül szintaktikai hibákat kapunk (amik ellenséges szövegfájl feldolgozásakor akár általunk nem kívánt parancsok nevünkben való végrehajtását is eredményezhetik).

14. tempfile

Ezzel a (nem minden unixon megtalálható, széles körben sajnos még nem elterjedt) paranccsal egyszerűen hozhatunk létre átmeneti fájlokat az erre szolgáló `/tmp` könyvtárban.

Példaként tükrözzük függőlegesen az inputot, sorainak felcserélésével! Írjuk elé a sorok számát! Az első megoldás egészen egyszerűen a `tac | cat -n` pipeline, de most egy pillanatig tételezzük fel, hogy se a `tac` program, se a `cat -n` paramétere nem létezik.

[60] Az `IFS` alapértelmezés szerint csak a szóközt tartalmazza, és vigyázzunk az állítgatásakor, mert nagyon sok más parancsot is befolyásol, legjobb, ha csak a `read` idejére állítjuk át, ha ezt a különleges utat választjuk.

[61] Ha a `VÁLTOZÓ`-t is elhagyjuk, akkor a `REPLY`-ban lesz az egész válasz, mint hátramaradt rész.

Akkor a megoldás az lehetne, hogy az egész inputot kiírjuk egy fájlba és utána egy ciklussal feldolgozzuk a fájlt visszafele. Mondjuk így:

visszafeleszamoz.sh

```
#!/bin/bash

cat >atmenetifajl

SOROK=$(cat atmenetifajl | wc -l)
for i in $(seq $SOROK)
do
  echo -n "$i "
  tail -n $i atmenetifajl | head -n1
done
rm atmenetifajl
```

Problémát okoz, ha van `atmenetifajl` nevű fájlunk az aktuális könyvtárban, illetve az egész szkript nem működik, ha az aktuális könyvtárban nem tudunk fájlt létrehozni pl. jogosultság hiányában. Ezért az ehhez hasonló átmeneti fájlok tárolására készítették van a `/tmp` könyvtárat, ahova mindenki „szemetelhet” és azt időnként letörlik a rendszerek gazdái. Óvatosan kell bánni ezzel, mivel ezt a könyvtárat mindenki közösen használja. A `cat >/tmp/atmenetifajl` megoldást pl. két különböző felhasználó egyszerre futtatva problémát okozhat egymásnak. Illetve az így létrejövő fájl más által is olvasható feldolgozás közben. Sőt, ha valaki előre megtudja, hogy mit fog a programunk csinálni akkor ügyesen még tetszőleges fájlunkat le is letörölheti.

A helyes megoldás valahogy így néz ki:

visszafeleszamoz.sh

```
#!/bin/bash

TMPFILE=$(tempfile)
cat >$TMPFILE

SOROK=$(cat $TMPFILE | wc -l)
for i in $(seq $SOROK)
do
  echo -n "$i "
  tail -n $i $TMPFILE | head -n1
done
rm $TMPFILE
```

Persze ez a megoldás nem túl hatékony, nagyon sokszor végigolvassa a fájlt, hogy kiírja az utolsó `i` darab sort, pedig igazából egyszer elég lenne, csak mindent el kellene tárolni és aztán visszafele kiírni. A `tac` így működik, de mi is tudunk ilyen okosabb és hatékonyabb, átmeneti fájl nélküli megoldást készíteni, felhasználva a shell tömbjeit, melyekkel egy néven, de különböző indexekkel tárolhatunk sok adatot, hasonlóan a matematika vektor fogalmához. A tömbökről bővebb információ a `man 1 bash` paranccsal kapható, az `Arrays` szóra kell keresni!

visszafeleszamoz-tomb.sh

```
#!/bin/bash

i=0
while read SOR[i]
do
    i=$(($i+1))
done

for j in $(seq $(($i-1)) -1 0)
do
  echo $(($i-$j)) ${SOR[$j]}
done
```

15. A UNIX fájlrendszere

Felhasználói szemszögből megismertük, hogy a legegyszerűbb fájlrendszerbejegyzésekkel hogyan kell dolgoznunk, kezeltünk fájlokat és könyvtárakat. Érdekes kérdés, hogy hogyan biztosítja a rendszer ezeket az egyszerű objektumokat. Hiszen neki csak egy nagy összefüggő adatterülete van, a háttértároló. Mi pedig kis (és nagy) fájlokat hozunk létre, amiket könyvtárakba teszünk, teljesen össze-vissza némelyiknek megnöveljük (illetve lecsökkentjük) a méretét, másolgatjuk és áthelyezzük őket. Az informatika ezen területe állandóan fejlődik, jelenleg is aktívan kutatják, újabb és újabb fájlrendszereket hoznak létre, amik egyre jobban oldják meg a feladatokat.

Ugyanakkor kialakult egy elgondolás, ami alapján szervezik a fájlrendszereket. Újabban nem minden fájlrendszer követi ezt, de azok is úgy viselkednek a külső szemlélő számára, mintha követnék, mivel programok sokasága ráépült erre a régen kitalált működési elvre.

Az elgondolás alapja, hogy minden egyes fájlhoz létrehozunk egy `inode` nevű adatstruktúrát, amit egy azonosítóval, az `i-number`-rel (avagy `ino`-val) címzünk egy nagy táblázatban. A nevén kívűl minden adatot (a bejegyzés típusát, a méretét, a jogosultságokat, a tulajdonost és a csoportot, a különböző időbélyegeket, a fájlra mutató fájlnevek számát, a tárolt adatok háttértárolón való helyét) ez az adatszerkezet tárol a fájlról.

Ezzel az egy ötlettel készen is vagyunk, már csak annyit kell hozzátenni, hogy a könyvtárak pedig olyan egyszerű fájlok, amik albejegyzések nevét rendelik hozzá `ino` számokhoz. Ezekután tetszőleges fájl `inode` struktúrája megkereshető a gyökérkönyvtártól kiindulva (aminek az `ino` száma fix). Megállapodtak abban is, hogy minden könyvtár első és második bejegyzése a `.` és `..`, hogy az relatív elérési utak is gyorsan értelmezhetőek legyenek.

15.1. Hardlinkek

A rendszer nem tiltja, hogy egy adott `ino` számra több könyvtárból is hivatkozást helyezzünk el, azonban az utolsó hivatkozás törlése után a fájl már nem lesz elérhető és az ő adatai bármikor felülírhatók. Ezért kell a számláló az `inode` struktúrába, hogy tudjuk, hogy az mikor válik nullává. Ha egy `inode`-ra több fájlnév is hivatkozik, akkor ezeket a fájlokat szokás egymás hardlinkjeinek hívni. Nincs közülük elsődleges, meg másodlagos, hiszen sehogy nem tudunk különbséget tenni köztük, hogy ki hardlinkje kinek, ráadásul ha bármelyik mentén módosítjuk a fájlt vagy tulajdonságait, azzal a háttértároló megfelelő adatterületét, illetve az inode struktúrát módosítjuk, tehát a módosítás a másik néven keresztül is látszódni fog. Az elmondottakból az is következik, hogy annak megállapítása, hogy egy fájlra vannak-e hardlinkek könnyű feladat (nagyobb-e a számlálója, mint 1), viszont annak megállapítása, hogy hol vannak ezek a hardlinkek az egész fájlrendszer fájának teljes átkutatását igényli, mivel egy utolsó eldugott kis könyvtárban is lehetnek hivatkozások az adott `ino` számra.

Az `ls -li` paranccsal a fájllistában láthatóvá válnak a sorok elején az `ino` számok, az `ln FORRÁS CÉL` paranccsal pedig létrehozhatunk hardlinkeket. Könyvtárak hardlinkelését csak a rendszergazda végezheti el, de neki sem tanácsos, ugyanis ezzel a fájlrendszer fastruktúrját tetszőleges gráffá változtathatja, aminek kezelése később nagyon elbonyolodhat.

Az üres könyvtáraknak kettő a számlálójuk, hiszen egy hivatkozás van maguktól (`.`), egy pedig a szülőkönyvtárban, míg a hagyományos fájlok számlálójának kiindulóértéke 1:

```
1 errge@pandora:~/tmp/x$ ls
2 errge@pandora:~/tmp/x$ ls -lid .
3 1363013114 drwxr-xr-x 2 errge progterv 6 2007-10-25 00:14 .
4 errge@pandora:~/tmp/x$ echo valami >fajl1
5 errge@pandora:~/tmp/x$ ls -li
6 total 4
7 1366509377 -rw-r--r-- 1 errge progterv 7 2007-10-25 00:15 fajl1
```

Minden egyes újonnan létrehozott hardlinkkel 1-el nő:

```
errge@pandora:~/tmp/x$ ln fajl1 hardlink
errge@pandora:~/tmp/x$ ls -li
total 8
1366509377 -rw-r--r-- 2 errge progterv 7 2007-10-25 00:15 fajl1
1366509377 -rw-r--r-- 2 errge progterv 7 2007-10-25 00:15 hardlink
```

Új alkönyvtár létrehozásával, a tartalmazó könyvtár számlálója eggyel nő, hiszen az új könyvtár `..` bejegyzése pont rá hivatkozik:

```
errge@pandora:~/tmp/x$ mkdir aldir
errge@pandora:~/tmp/x$ ls -lid .
1363013114 drwxr-xr-x 3 errge progterv 45 2007-10-25 00:15 .
errge@pandora:~/tmp/x$ mkdir aldir2
errge@pandora:~/tmp/x$ ls -lid .
1363013114 drwxr-xr-x 4 errge progterv 58 2007-10-25 00:15 .
```

Mint említettük, a tárterület közös:

```
errge@pandora:~/tmp/x$ ln hardlink aldir/xxx
errge@pandora:~/tmp/x$ echo felulir >aldir/xxx
errge@pandora:~/tmp/x$ cat fajl1
felulir
errge@pandora:~/tmp/x$ ls -li
total 8
1616090068 drwxr-xr-x 2 errge progterv 6 2007-10-25 00:16 aldir
1774943086 drwxr-xr-x 2 errge progterv 6 2007-10-25 00:15 aldir2
1366509377 -rw-r--r-- 3 errge progterv 8 2007-10-25 00:16 fajl1
1366509377 -rw-r--r-- 3 errge progterv 8 2007-10-25 00:16 hardlink
```

Azt is láthatjuk az `ls` kimenetéből, hogy a `fajl1` és `hardlink` nevű fájlok egymás hardlinkjei, viszont ez nem minden, mert összesen három link van arra a területre, azonban ha nem tudjuk, hogy az `aldir` nevű könyvtárunk `xxx` fájla az, akkor elég nehéz megtalálni. Persze a saját fájlaink hardlinkjei biztos a saját részünkben lesz a Pandora gépen és tudjuk a számát is (1366509377), így `find`-dal megtalálható:

```
errge@pandora:~/tmp/x$ find ~ -inum 1366509377
/h/e/errge/tmp/x/fajl1
/h/e/errge/tmp/x/hardlink
/h/e/errge/tmp/x/aldir/xxx
```

15.2. Szimbolikus linkek

Az, hogy egy tárterületre több néven is lehet hivatkozni olyan közkedvelt lett, hogy egy másik ilyen lehetőséget is létrehoztak, a szimbolikus linkeket (symlinkeknek és szoftlinkeknek is szokás nevezni őket). Ezek sokban különböznek a hardlinkektől, ők egyszerűen csak egy speciális fájltípussal rendelkező fájlok, amiknek a tartalma a mutatott fájl elérési útja. Egy szimbolikus link célja meg lehet adva abszolút elérési úttal (`/` jellel kezdődően) vagy relatíven. A relatív linkek azért jók, mert egész könyvtárakat mozgatva, tömörítve, kibontva helyesek maradnak az új helyükön is.

A symlinkek esetében látható, hogy meg lehet mondani, hogy ki symlink kire, viszont azt továbbra is csak teljes kereséssel lehet megállapítani, hogy egy fájlra vannak-e szimbolikus linkek. Ha egy fájlt letörlünk, amire szimbolikus linkek voltak, akkor azok a linkek értelmetlenné válnak, hiszen a hivatkozott fájl már nem található meg.

A hardlinkekkel ellentétben, a szimbolikus linkek könyvtárakra is hivatkozhatnak, illetve más háttértárolón is lehet a céljuk. Ugyanis annak ellenére, hogy a unix rendszerekben látszólag minden egy közös gyökérkönyvtárban van, bizonyos könyvtárakra a rendszergazda kérheti egy másik fájlrendszer (és így háttértároló) csatlakoztatását. Ezután a folyamat után a könyvtár normál tartalma láthatatlanná válik és a rendszer úgy csinál, mintha a felcsatlakoztatott egység fájlai lennének ott. Természetes, hogy hardlinkek nem lehetnek különböző fájlrendszerek között, hiszen ugyanaz az `ino` szám ki lehet osztva mindkét adattárolón.

Szoftlinkeket az `ln -s FORRÁS SYMLINK` paranccsal hozhatunk létre, `ls -l`-lel való listázáskor pedig egy nyíl jelzi, hogy ilyen fájlról van szó. A `find` parancs `-type l` tesztje akkor és csak akkor igaz, ha a szóban forgó fájl egy szimbolikus link.

```
errge@pandora:~/tmp/x/aldir2$ ls -l
total 0
errge@pandora:~/tmp/x/aldir2$ echo mogyoro >alma
errge@pandora:~/tmp/x/aldir2$ ln -s alma korte
errge@pandora:~/tmp/x/aldir2$ ls -l
total 4
-rw-r--r-- 1 errge progterv 8 2007-10-25 00:45 alma
lrwxrwxrwx 1 errge progterv 4 2007-10-25 00:46 korte -> alma
errge@pandora:~/tmp/x/aldir2$ find -type l
./korte
errge@pandora:~/tmp/x/aldir2$ cat korte
mogyoro
errge@pandora:~/tmp/x/aldir2$ echo atir >korte
errge@pandora:~/tmp/x/aldir2$ cat alma
atir
errge@pandora:~/tmp/x/aldir2$ rm alma
errge@pandora:~/tmp/x/aldir2$ ls -l
total 0
lrwxrwxrwx 1 errge progterv 4 2007-10-25 00:46 korte -> alma
errge@pandora:~/tmp/x/aldir2$ cat korte
cat: korte: No such file or directory
errge@pandora:~/tmp/x/aldir2$ ln -s korte mogyoro
errge@pandora:~/tmp/x/aldir2$ ln -s mogyoro alma
errge@pandora:~/tmp/x/aldir2$ ls -l
total 0
lrwxrwxrwx 1 errge progterv 7 2007-10-25 00:48 alma -> mogyoro
lrwxrwxrwx 1 errge progterv 4 2007-10-25 00:46 korte -> alma
lrwxrwxrwx 1 errge progterv 5 2007-10-25 00:48 mogyoro -> korte
errge@pandora:~/tmp/x/aldir2$ cat alma
cat: alma: Too many levels of symbolic links
errge@pandora:~/tmp/x/aldir2$ cat korte
cat: korte: Too many levels of symbolic links
errge@pandora:~/tmp/x/aldir2$ cat mogyoro
cat: mogyoro: Too many levels of symbolic links
```

Persze mint látható szimbolikus linkekből egy kör is létrehozható, ami végülis nem hivatkozik semmire. Az ebből fakadó problémákat úgy oldották meg, hogy bizonyos számú „átirányítás” után a rendszer magja nem folytatja a fájlnévhez tartozó `inode` megkeresését, hanem hibaüzenettel leáll.

Írjunk szkriptet, ami kideríti, hogy mennyi a mi rendszerünkön ez a limit!

```
errge@pandora:~/tmp$ bash ~/tmp/symlink-limit.sh
6 melysegu symlink lancot mar nem bir.
errge@pandora:~/tmp$
```

A szkript létrehoz egy `symlink-teszt` nevű alkönyvtárat és abban addig hoz létre nem kört tartalmazó, csak egyre hosszabb láncot, amíg sikerül a feloldás:

```
#!/bin/bash

mkdir symlink-teszt
cd symlink-teszt
echo tartalom >fajl
i=1
ln -s fajl symlink1
while [ "tartalom" = "$(cat symlink$i 2>/dev/null)" ]
do
  ln -s symlink$i symlink$(($i+1))
  i=$(($i+1))
done
echo "$i melysegu symlink lancot mar nem bir."
cd ..
rm -rf symlink-teszt
```

15.3. Szimbolikus linkek kibogozása (nehéz)

A 6 az nem túl sok és az otthoni rendszeremen is csak 9 ez az érték. Gyakorlásképp írjunk egy másik szkriptet, ami bármekkora láncnak megkeresi a végét, azaz kimenete az, amire a lánc vége mutat. Figyeljünk arra, hogy ne kerüljön végtelen ciklusba, csak azért, mert belefut egy körbe, hanem ekkor illedelmesen közölje (a hibacsatornán), hogy ez bizony egy kör, így nincs vége és a visszatérési értéke ekkor 1 legyen.

Első gondolatunk az lehetne, hogy megjegyezzük az első szimbolikus link fájlnevét és utána ha bármikor ezzel találkozunk mégegyszer, az azt jelenti, hogy körbe jutottunk. Azonban ez a stratégia csődöt mond, ha nem a kör közepén indítják el a szkriptet, hanem annak egy bevezető „nyelén", hiszen arra a nyélre már soha nem fogunk visszajutni és így a végtelenségig kőrözünk. Szintén rossz ötlet a fájlnak a nevét megjegyezni, mivel az sokféleképp előfordulhat a szimbolikus linkekben, a különböző relatív nevekkel (`../aldir/../aldir/f1`, stb.), valamint a symlinkek közül némelyikek lehetnek hardlinkek egymásra, akkor sem egyezik a nevük, mégis ugyanazt a láncszemet jelentik. Jegyezzük meg inkább az `ino` számot!

A kör megtalálásához egy helyes stratégiát adott Robert W. Floyd, ez a sokat eláruló „a teknős és a nyúl" nevet viseli.[62] A megoldás lényege, hogy elindítjuk a két állatot a startból, de a nyúl mindig kétszer annyit lép. Akkor és csak akkor találkoznak újra (fognak ugyanazon az `ino` számon tanyázni), ha kör van a struktúrában. Egyébként pedig a nyúl eléri a lánc végét és az ott található fájlnév[63] az eredmény.

Már most látható, hogy ez az eddigi legkomolyabb shell vállalásunk. Ilyen algoritmikus problémát talán már nem is kényelmes shellben megoldani, viszont mivel nagyon szorosan kapcsolódik a feladat az operációs rendszerhez, valami magasabb szintű programozási nyelven is nehézségekbe ütköznénk, ugyanis ott annak kiderítése lenne pl. nehézkes, hogy mi szoftlink és mi nem vagy minek mi az `ino` száma.

A program bonyolultságának további növekedését elkerülve feltételezzük, hogy a kérdéses szoftlinkek mind egy könyvtárban vannak. A problémát az okozná, hogyha egy link mondjuk a `../valami`-re mutat, az pedig a `./valami2`-re, akkor a `valami2` nevű fájlt nyilván a `..`-ban kellene keresni, mi azonban a `.`-ban keresnénk, hiszen a mutatott név a `./valami2`. Talán a legegyszerűbben ez úgy orvosolható, ha készítünk egy programot, ami első paramétereként egy relatív vagy abszolút útvonalat vár és ha relatívat kap, akkor átalakítja abszolúttá pl. úgy,

[62] http://en.wikipedia.org/wiki/Cycle_detection

[63] Ami persze lehet nem létező is, de a lényeg, hogy már biztosan nem egy létező szoftlink neve.

hogy elé írja a `pwd` parancs kimenetét. Ezt felhasználva már a programban tudjuk mindig teljes elérési úttal követni a fájlneveket.

veget-keres.sh

```
#!/bin/bash

if [ "$#" -ne "1" ] || ! [ -h "$1" ]
then
  echo "Hasznalat: $0 <kibogozando-symlink>"
fi

NYUL_F=$1
TEKNOS_F=$1
NYUL_INO=$(find "$1" -printf '%i')
TEKNOS_INO=$(find "$1" -printf '%i')

while true
do
  # lepjen a nyul 1-et es nezze meg, hogy nincs-e vege
  NYUL_F=$(find "$NYUL_F" -printf '%l')
  [ -h "$NYUL_F" ] || { echo $NYUL_F; exit 0; }
  NYUL_INO=$(find "$NYUL_F" -printf '%i')
  # lepjen a nyul meg 1-et es nezze meg, hogy nincs-e vege
  # ez csak az elozo 3 sor copy&paste-je
  NYUL_F=$(find "$NYUL_F" -printf '%l')
  [ -h "$NYUL_F" ] || { echo $NYUL_F; exit 0; }
  NYUL_INO=$(find "$NYUL_F" -printf '%i')

  # lepjen a teknos 1-et
  # ez az elozo harom sor s/NYUL/TEKNOS/g csereje a 2. sor nelkul
  TEKNOS_F=$(find "$TEKNOS_F" -printf '%l')
  TEKNOS_INO=$(find "$TEKNOS_F" -printf '%i')

  # ha egyezik a ket INO, akkor kor van
  if [ "$TEKNOS_INO" = "$NYUL_INO" ]
  then
    echo kor van >&2
    exit 1
  fi
done
```

Stressztesztként létrehoztam egy 999 → 998 → . . . → 001 szoftlink láncot, amit ki is bogoz:

```
errge@pandora:~/tmp/x/aldir2/x$ bash ../veget-keres.sh 999
001
errge@pandora:~/tmp/x/aldir2/x$ ln -s 681 001
errge@pandora:~/tmp/x/aldir2/x$ bash ../veget-keres.sh 999
kor van
```

Ha esetleg megoldjuk, hogy a könyvtárak közötti szoftlinkek is működjenek, akkor figyeljünk arra, hogy könnyen átléphetünk egy fájlrendszer határt és ekkor ha hatalmas pechünk van, akkor két `ino` szám akkor is egyezhet, ha azok egymástól teljesen független fájlrendszeren lévő, csak azonos számmal rendelkező fájlok. Ez ellen úgy védekezhetünk, hogy a `TEKNOS_INO` és `NYUL_INO` környezetváltozókban a `find %D:%i` formájú `printf` eredményét írjuk. A `%d` egy olyan szám, ami adattárolónként biztosan különböző és így ők ketten kettősponttal elválasztva már biztosan egy teljesen egyedi azonosítót képeznek az operációs rendszer minden fájlára.

15.4. FIFO-k, eszközfájlok, socketek

A unix rendszerben majdnem minden ábrázolható fájlként, ezért még további speciális fájltípusok is vannak, amelyeknek ismerete nem szükséges egyetlen felhasználó számára sem. Azonban minden rendszergazdának tisztában kell lennie ezekkel a fogalmakkal, nélkülük nem boldogulhat.

16. Processzek

...

17. Web automatizáció

Egyre több feladatot intézünk egyetlen webböngésző segítségével. Így érhetjük el a menetrendeket, a közösségi oldalakon az üzeneteinket, így kereshetünk a weben a googlevel, így olvashatjuk a wikipediat. Sokan az elektronikus levelezésüket sem valamely erre a célra kifejlesztett programmal, mint pl. a GNU/Emacs Gnus moduljával intézik, hanem a rendszergazda által telepített webmailt használják inkább.

Ez a megközelítés egészen addig működőképes, amíg nem szeretnénk a weblap által nyújtott legegyszerűbb funkcióknál mi egy kicsit többet vagy mást csinálni. Például elképzelhető, hogy le szeretnénk menteni az üzeneteinket, hogy mindig meg legyenek, akkor is, ha nincs internet, vagy egy képgaléria nem a böngészőben szeretnénk átlapozni, hanem az egészet letöltenénk és a repülőn a laptopunkon majd egy rendes képnézegetőprogrammal megnéznénk az összes képet.

Ezeket az igényeket az oldalak üzemeletetői nem szokták kielégíteni, mivel egyrészt számukra némelyik üzletrontó lenne, némelyikre pedig egyszerűen nincs igény (ki törődik manapság azzal, ha az informatika forgatagában elveszik pár levele?).

Némi shell és HTML (szélsőséges esetben Javascript) tudással és a `wget` program felhasználásával azonban mi magunk megoldhatjuk ezeket a feladatokat. Pár példán keresztül bemutatjuk, hogy hogyan.

17.1. `wget`

Újabban megtalálható a GNU/Linux rendszereken egy `wget` nevű segédprogram, amivel a weben megtalálható fájlok letöltése végezhető el. Például ezen jegyzet aktuális példányát letölti a `wget http://www.gergely.risko.hu` parancs az aktuális könyvtárba.

```
1  errge@home:~$ wget http://www.gergely.risko.hu/progalap1/jegyzet.pdf
2  --23:18:30--  http://www.gergely.risko.hu/progalap1/jegyzet.pdf
3  => ‘jegyzet.pdf’
4  Resolving www.gergely.risko.hu... 62.112.193.66
5  Connecting to www.gergely.risko.hu|62.112.193.66|:80... connected.
6  HTTP request sent, awaiting response... 200 OK
7  Length: 621,071 (607K) [application/pdf]
8
9  100%[=============================================================>] 621,071      950.58K/s
10
11 23:18:31 (945.29 KB/s) - ‘jegyzet.pdf’ saved [621071/621071]
12
13 errge@home:~$ rm jegyzet.pdf
14 errge@home:~$ wget -q http://www.gergely.risko.hu/progalap1/jegyzet.pdf
15 errge@home:~$ ls -l jegyzet.pdf
16 -rw-r--r-- 1 risko risko 621071 Oct 25 09:24 jegyzet.pdf
```

A letöltés közben mindenféle üzeneteket ír ki (a standard hibacsatornára), amiket a `-q` opcióval letilthatunk. A `-O célfájl` opcióval beállítható a kimeneti fájl, illetve célfájlként a `-` karaktert megadva a standard kimenetre írja a letöltött fájlt a `wget`. Más opciók megadásával elérhető pl. az is, hogy teljes webstruktúrákat letöltsünk (a linkek automatikus követésével), a man oldal most is bő információforrás.

17.2. Google Fight

A http://www.googlefight.com/ oldalon ha megadunk két kulcsszót, akkor az oldal mindkettőre rákeres a Googlén és megmutatja nekünk, hogy melyikhez mennyi találat volt. Kissé igénytelenül, de használhatjuk ezt a szolgáltatást arra, hogy eldöntsünk egyszerű helyesírási kérdéseket, pl., hogy az „enthusiastic” szóban valóban van-e h betű. Ugyanis feltehetjük, hogy az emberek többsége helyesen írja ezt a szót, és így az elterjedtebb írásmód lesz a helyes. Persze némely esetben az ilyen gondolkodás tévútra is vihet.

Talán kicsit érdekesebb felhasználása ennek a weboldalnak annak eldöntése, hogy vajon az interneten Newtonról vagy Einsteinről írnak többet.

Mindenesetre valósítsuk meg ezt a szolgáltatást shell scriptben:

```
errge@pandora:~$ ./google-fight.sh vajon valyon
3420000:vajon
82200:valyon
vajon nyert
errge@pandora:~$ ./google-fight.sh Newton Einstein
62400000:Newton
41100000:Einstein
Newton nyert
```

Először is azt kell ehhez észrevennünk, a webböngészőnkben próbálkozva, hogy a Googlének a http://www.google.com/search?q=newton URL-t kell megadni ahhoz, hogy a „newton” sztringre keressen, az eredményoldalon pedig látszik, hogy hány találat van. Mentsük le ezt a példát a `wget -O proba-google http://www.google.com/search?q=newton` paranccsal. Az első probléma, amivel szembesülünk, hogy a `wget` hibát ad vissza, ugyanis a Google nem szeretné, ha így használnák a keresés funkciót, mivel magasszintű programozási nyelvekből könnyebben kezelhető interfészt is biztosítanak az automatikus keresésekhez. A weben minden letöltés alkalmával elküldik a böngészők, hogy pontosan milyen milyen operációs rendszeren, milyen verziójuk fut. Ezt használja ki a Google, ha meglátja, hogy mi a `wget`-tel próbálkozunk, akkor megtagadja a lekérdezést. Szerencsére a `-U ""` opcióval utasíthatjuk a `wget`-et, hogy ne küldjön ilyen információkat. Töltsük így le a keresés eredményét és vegyük szemügyre a létrejövő `proba-google` nevű fájlt!

A következő feladatunk, hogy kitalálunk olyan parancssorozatot, ami kinyeri a szükséges számértéket ebből a fájlból. Ez a rész természetesen feladatonként nagyon eltérő, de a tanult unix eszközök alkalmazásával mindig célt érhetünk. Jelen esetben:

```
errge@pandora:~$ cat proba-google | sed 's/<font size=-1>/\n/g' | grep '^Results' \
  | cut -d\> -f 6 | cut -d\< -f1
62,200,000
errge@pandora:~$ cat proba-google | sed 's/<font size=-1>/\n/g' | grep '^Results' \
  | cut -d\> -f 6 | cut -d\< -f1 | sed s/,//g
62200000
```

Vegyük észre, hogy gyakorlatilag készen vagyunk, már csak meg kell írni a szkriptet, ami mindezt összefogja, paramétereket dolgoz fel és ellenőríz és segítséget ad, valamint megállapítja, hogy melyik szám a nagyobb és győztest hirdet.

google-fight.sh

```
#!/bin/bash

if [ "$#" -ne 2 ]
then
  echo "Usage: $0 <string1> <string2>" >&2
  exit 1
fi

# az elso sztring lekerese, a talaltok szamanak kibanyaszasa
A=$(wget -q -O - -U xxx "http://www.google.com/search?q=$1" | sed 's/<font size=-1>/\n/g' \
  | grep '^Results' | cut -d\> -f 6 | cut -d\< -f1 | sed s/,//g)
# a masodik sztring
B=$(wget -q -O - -U xxx "http://www.google.com/search?q=$2" | sed 's/<font size=-1>/\n/g' \
  | grep '^Results' | cut -d\> -f 6 | cut -d\< -f1 | sed s/,//g)

# ha valamelyikre nincs talalat, akkor legyen 0
if [ "$A" = "" ]; then A="0"; fi
if [ "$B" = "" ]; then B="0"; fi

# megjelenites
echo "$A:$1"
echo "$B:$2"

if [ $A -gt $B ]
then
  echo "$1 nyert"
elif [ $A -lt $B ]
then
  echo "$2 nyert"
else
  echo "Dontetlen"
fi
```

Gondolkodjunk el azon, hogy hogyan tudnánk megoldani, hogy több paramétert is meg lehessen adni, bármennyit. Mindegyikre keressünk rá és írjuk ki az értékeket hozzájuk, majd adjuk meg a vesztest és a győztest!

Látható, hogy ezek a megoldások bizonyos szempontból nagyon szörnyűek. Ugyanis ha a Google egy kicsit megváltoztatja azt, ahogy a keresés HTML eredménye kinéz, akkor a programunk már nem működik. És miért ne változtatná, hiszen ő nem számít arra, hogy mi ezt programból használjuk. Kicsit javíthatunk a helyzeten, ha a html-ekből információt (a pontszámot) kigyűjtő parancssort külön fájlba tesszük és a `google-fight.sh`-ból csak használjuk. Ugyanis ekkor külön tudjuk tesztelni és „fejleszteni” azt a részt, amennyiben valami nem működne. A programunk többi része, ha már egyszer az a belső hosszú parancs jó értéket ad vissza, nem kell, hogy változzon.

17.3. Esnips képletöltés

Kicsit bonyolultabb a helyzet akkor, ha az automatizálandó letöltés személyünkhöz kötött (pl. jelszóval be kell lépni). Ezek az oldalak úgy működnek, hogy elhelyeznek egy ún. cookie-t (egy nagyon nagy véletlen számot) a böngészőnkben és ahogy újabb képekre kattintunk, az a cookie mindig visszaküldésre kerül. Így a szerver tudja ellenőrízni, hogy mi valóban jogosultak vagyunk a kép letöltésére, hiszen birtokában vagyunk egy olyan cookienak, amit csak úgy kaphattunk meg, ha korábban már beírtuk a jelszavunkat.

A www.esnips.com esetében nem felhasználói névvel és jelszóval kell belépni, hanem emailben szoktam URL-eket kapni azoktól az emberektől, akik meg akarják mutatni egy albumokat.

Ez a „meghívó” webcím, amit az esnips legyárt és emailben elküld nekem csak az adott albumhoz ad hozzáférést.

A `wget`-nek vannak olyan opciói, amiket, ha megadjunk, akkor az említett cookiek kilépéskor elmentésre kerülnek egy fájlba, majd újboli elindításkor pedig betöltésre és így a weboldal úgy érzi, mintha egy böngésző megjegyezte volna őket, ahogy azt eredetileg kitalálták.

Az esnips képletöltést végző program kommentekkel ellátva:

```
#!/bin/bash

[ "$1" = "" ] && {
  echo download URL must be given
  exit 1
}

# fuggveny, ami a kis kepes listabol megcsinalja a nagy kepek letolto parancsait
listimgs () {
  # kikiserletezes es gondolkodas eredmenye
  # ha az esnips valtozik, ez elromlik
  cat tmp.html | fgrep /doc/ | grep CommandLink | cut -d"'" -f4-6 | \
  sed "s,^/doc/\(.*\)/\(.*\)'.*,wget $MY_WGET -O \2.jpg http://www.esnips.com/nsdoc/\1,"
}

URL=$1
# ezek az opciok kellenek ahhoz, hogy megtartsa a cookiekat a cook.txt-ben
MY_WGET="--keep-session-cookies --load-cookies cook.txt --save-cookies cook.txt"

# ha mar a NEXTPAGE ures lesz, akkor ez a feltetel igaz
while [ "http://www.esnips.com/" != "$URL" ]
do
  # letoltes atmeneti fajlba
  wget $MY_WGET -O tmp.html $URL
  # lefuttatjuk az aktualis oldal kepletolto parancsait
  listimgs | bash
  # ez a grep es cut csak akkor ad vissza valamit, ha meg van az oldalon Next
  # link, azaz ez nem az utolso lap a kepgaleriabol
  NEXTPAGE=$(cat tmp.html | grep Next | cut -d"'" -f2)
  # a kovetkezo lap URL-je igy kaphato
  URL=http://www.esnips.com/$NEXTPAGE
done
rm tmp.html cook.txt
```

Csak 18 kódsor és mégis működik:

```
errge@home:~$ cat esnipsget.sh | grep -v '^$' | grep -v '#.*' | wc -l
18
errge@home:~$ chmod a+x esnipsget.sh
errge@home:~$ ./esnipsget.sh \
  'http://www.esnips.com/fm/7f85391b-f3a3-4648-af78-3a0b315134f4/?v=458337&source=ws'
...
errge@home:~$ ls
DSC_4833.jpg  DSC_4845.jpg  DSC_4854p1.jpg  DSC_4863.jpg  DSC_4873.jpg  DSC_4882.jpg
DSC_4834.jpg  DSC_4846.jpg  DSC_4854p.jpg   DSC_4864.jpg  DSC_4874.jpg  DSC_4884.jpg
DSC_4835.jpg  DSC_4847.jpg  DSC_4855.jpg    DSC_4865.jpg  DSC_4875.jpg  esnipsget.sh
DSC_4836.jpg  DSC_4848.jpg  DSC_4856.jpg    DSC_4866.jpg  DSC_4876.jpg
DSC_4838.jpg  DSC_4849.jpg  DSC_4858.jpg    DSC_4867.jpg  DSC_4877.jpg
DSC_4839.jpg  DSC_4850.jpg  DSC_4859.jpg    DSC_4869.jpg  DSC_4878.jpg
DSC_4842.jpg  DSC_4852.jpg  DSC_4860.jpg    DSC_4870.jpg  DSC_4879.jpg
DSC_4843.jpg  DSC_4853.jpg  DSC_4862.jpg    DSC_4871.jpg  DSC_4881.jpg
```

17.4. IWIW üzenetek archiválása

Lássuk, hogyan lehetne az IWIW-es üzeneteinket letölteni és emészthető formában kiírni! Először határozzuk el, hogy milyen formátumot szeretnénk kapni a weben való olvasgatás helyett!

```
From iwiw
From: feladó
Date: dátum
Subject: az üzenet tárgya

Az üzenet szövege

From iwiw
From: a 2. üzenet feladója
Date: hozzá a dátum
Subject: ...
...
```

Ebben a formátumban az a pláne, hogyha a kiadjuk a `pine -f /h/.../iwiwfajl -i` parancsot, akkor pine-ból olvashatjuk az iwiw-ről letöltött üzeneteinket. Ezt a levéltárolási formát Berkley mailboxoknak, vagy röviden mbox[64]-oknak nevezik.

Első tevékenységként be kellene lépnünk az iwiw-re `wget`-tel, hogy megkapjuk a cookiet, amivel később az leveleket egyesével lekérhetjük. Az, hogy ezt hogy csináljuk természetesen teljesen iwiw specifikus, megnézve a belépő oldal forrását, azt látjuk, hogy ún. POST kéréssel kell belépni. Erre a `wget --post-data` kapcsolója ad lehetőséget, ahol az adatot úgy kell megadni, hogy minden mezőt a `&`, a mező nevét és értékét pedig az `=` választja el.[65]

A szkriptünk eleje tehát így néz ki:

iwiw-letolt.sh

```
#!/bin/bash

EMAIL=idekell@beirniazemail.cimet
PASSWORD=idekellirniajelszot

wget -q -O /dev/null --keep-session-cookies --save-cookies cookies.txt \
  --post-data="email=$EMAIL&password=$PASSWORD" \
  'http://www.iwiw.hu/pages/user/login.jsp?method=Login'
```

Lássuk, hogy lehetne kinyerni azt, hogy hány oldalon keresztül listázza a wiw a leveleinket:

```
errge@pandora:~/iwiwget$ chmod a+x iwiw-letolt.sh
errge@pandora:~/iwiwget$ ./iwiw-letolt.sh
errge@pandora:~/iwiwget$ cat cookies.txt
www.iwiw.hu     FALSE   /pages/main/    FALSE   1214309909      lastHit 1194309909160
www.iwiw.hu     FALSE   /pages/user/    FALSE   1214309909      httpslogin      0
www.iwiw.hu     FALSE   /pages/user/    FALSE   1214309909      email   iwiw@risko.hu
www.iwiw.hu     FALSE   /pages/user/    FALSE   1214309909      forgetEmail     0
www.iwiw.hu     FALSE   /pages/user/    FALSE   1209861909      autoLoginLimited        0
www.iwiw.hu     FALSE   /       FALSE   0       JSESSIONID      1
www.iwiw.hu     FALSE   /       FALSE   0       cachedSID       1194309909119_...
errge@pandora:~/iwiwget$ wget -q --load-cookies cookies.txt -O - \
  'http://www.iwiw.hu/pages/message/inbox.jsp?page=0'  | grep pg_selector | \
  sed 's/page=/\n/g' | tail -n1 | cut -d'"' -f1
4
```

[64] http://en.wikipedia.org/wiki/Mbox

[65] Az, hogy ez miért van így, messze túl mutatna a jegyzet keretein, HTML specin, vagy tetszőleges szakirodalom felhasználásával megtanulhatjuk mindezt.

Tehát egy egyszerű `for` és `seq` kombinációval végig fogunk tudni menni az oldalakon, amiken az üzeneteink listázva vannak. Ez a lista úgy néz ki, ha megtekintjük az oldal forrását, hogy minden üzenetnek van egy azonosítószáma, amire linkeket generál a szerver. Tehát nekünk ezeket a linkeket kell kiválogatni és az előző példa képeihez hasonlóan letölteni őket. Ha ez készen van, akkor már csak egy kis formai átalakítás lesz hátra, hogy mbox formátumot kapjunk.

Lássuk tehát azt a `for` ciklust, ami végigmegy az oldalakon és minden oldalon letölti az ott található üzeneteket (mindegyiket a saját `sorszáma.html` nevű fájlba):

```
PAGES=$(wget $MY_WGET -q -O - 'http://www.iwiw.hu/pages/message/inbox.jsp?page=0' \
  | grep pg_selector | sed 's/page=/\n/g' | tail -n1 | cut -d'"' -f1)

if [ "$PAGES" = "" ]; then PAGES=0; fi

OLVASURL=http://www.iwiw.hu/pages/message/messageread.jsp
for i in $(seq 0 $PAGES)
do
  wget $MY_WGET -q -O - "http://www.iwiw.hu/pages/message/inbox.jsp?page=$i" \
    | tr '?' '\n' | grep messageID | sed 's/.*messageID=//' | \
    sed "s,\([0-9]*\).*,wget -O \1.html $MY_WGET '$OLVASURL?messageID=\1',"
done | bash
```

Készen van tehát az a programunk, ami egy könyvtárban létrehozza az összes `.html` fájlt, ami a leveleinket tartalmazza. Foglaljuk ezt össze és csináljuk meg, hogy létrehozzon egy `tmp` nevű könyvtárat kezdetben és oda szemeteljen:

```
#!/bin/bash

if [ -e tmp ]
then
  echo "Mar letezik tmp nevu fajlbejegyzes"
  exit 1
else
  mkdir tmp
  cd tmp
fi

EMAIL=usernev
PASSWORD=jelszo
MY_WGET="--keep-session-cookies --load-cookies cook.txt --save-cookies cook.txt"

echo "Logging in..."
wget -q -O /dev/null $MY_WGET --post-data="email=$EMAIL&password=$PASSWORD" \
  'http://www.iwiw.hu/pages/user/login.jsp?method=Login'

echo "Getting number of pages..."
PAGES=$(wget $MY_WGET -q -O - 'http://www.iwiw.hu/pages/message/inbox.jsp?page=0' \
  | grep pg_selector | sed 's/page=/\n/g' | tail -n1 | cut -d'"' -f1)
if [ "$PAGES" = "" ]; then PAGES=0; fi

echo "Getting message list..."
OLVASURL=http://www.iwiw.hu/pages/message/messageread.jsp
for i in $(seq 0 $PAGES)
do
  wget $MY_WGET -q -O - "http://www.iwiw.hu/pages/message/inbox.jsp?page=$i" \
    | tr '?' '\n' | grep messageID | sed 's/.*messageID=//' \
    | sed "s,\([0-9]*\).*,wget -O \1.html $MY_WGET '$OLVASURL?messageID=\1',"
done | bash
```

Végül hozzuk valamennyire mbox formátumra a html fájlokat:

```
for i in tmp/*.html
do
  from=$(cat $i | tr -d '\n' | sed 's/.*class="">//' |  cut -d\< -f1)
  subject=$(cat $i | tr -d '\n' | sed 's/.*T.ma: <\/span>[ \t]*//' | cut -d\< -f1)
  date=$(cat $i | tr -d '\n' | sed 's/.*d.tuma: <\/span>[ \t]*//' | cut -d\< -f1)

  echo "From iwiw Thu Jan  1 00:00:00 CET 1970"
  echo "From: $from"
  echo "Date: $date"
  echo "Subject: $subject"
  echo
  cat $i | grep -A1 'caption">.zenet sz.vege' | tail -n1 | links -dump
  echo
  echo
done
```

Nem részletezzük ennek magyarázatát, több okból:

- mire az olvasóhoz jut ez a kód, valószínűleg az iwiwesek már változtattak valamit és az egész nem is működik,
- az összes létező mboxokra és emailekre vonatkozó szabványt megszegi, amit csak lehet, nem foglalkoztunk az ékezetekkel, a dátumok átalakításával,
- a Pine ezért elég hibásan, de kezeli az eredményt, a koncepció életképessége látható,
- a szöveges böngésző (`links`) dump parancsa minden képet lenyel, így a levelek egy jó része, a mosolygók, az ajándék szó, az iwiw szó, meg minden, amit az okos wiwesek képekkel helyettesítenek elveszik,
- végül ez a részfeladat valószínűleg sokkal korrektebbül megoldható valami olyan nyelven, ahol mboxok, dátumok, emailek és a különböző karakterkonverziók (ékezetek) kezelésére kész programmodulok vannak.

Az is egy megoldás lehet, hogy ezt az utolsó lépést nem csináljuk, meghagyjuk a leveleinket htmlként, és egyszerűen webböngészővel nézegetjük a html-eket és így a wiwes leveleinket.

Mindezen problémák jól mutatják, hogy az ebben a fejezetben ismeretett vészmegoldásokat igazából sosem szabad használni, hanem az adat tulajdonosát rá kell venni, hogy a szóban forgó adatot normális külalakkal szolgáltassa. A fényképeket egy nagy zip fájlban is az album mellett, a wiw pedig biztosítson megoldást a tulajdon leveleink archiválására. Sajnos amíg ebben ellenérdekeltek és ez feloldhatatlan, addig is jobban járhatunk az ilyen félmegoldásokkal, mintha egyszerűen lemondunk pl. a biztonsági mentésről, a leveleink `grep`-pel való kereshetőségéről, az offline elérhetőségről és a többi előnyről, amit a web, mint platform mind elvesz tőlünk.