Csáky István - Mörk Péter

Orlando Unix iskola

TARTALOM

Bevezetés

„A UNIX sokkal komplikáltabb -
az átlagos UNIX buherátornak
sose jut eszébe, hogy is hívják
ezen a héten a PRINT parancsot...”

(Ed Post: Az Igazi Programozó)

„Ez Unix! Ezt ismerem!” - kiáltott fel a dinoszaurusz-sokszorosító professzor tízéves kislánya az „Őslénypark” című filmben, mikor a történet vége felé leült a számítógéphez, hogy reteszelje az ajtót a fenyegetően közeledő őslények orra előtt. A filmből annyi mindenesetre igaz, hogy a Unix napjaink legelterjedtebb munkaállomásokon futó operációs rendszere. A mostanában korszerűnek számító egeres-ablakos-mutogatós operációs rendszerekhez képest a Unix kifejezetten primitívnek és elavultnak tűnik. Ha azonban közelebbről megvizsgáljuk, rájövünk hogy egyáltalán nem primitív, csak rettenetesen mogorva azokkal szemben, akik nem ismerik - ne várjuk tehát tőle, hogy első alkalommal barátságos legyen hozzánk. Ennek főleg történeti okai vannak: a Unix ugyanis azokban a történelem előtti időkben készült, amikor a „stupid user” (buta felhasználó) nevű állatfaj még nem volt honos a földön, legalább is nem olyan nagy tömegben mint manapság. Az Igazi Programozóknak (mert a Unixot természetesen Igazi Programozók írták) tehát nem kellett tekintettel lenniük arra, hogy az operációs rendszerüket rajtuk kívül még más is használni szeretné, így teljesen a saját ízlésüknek megfelelően alakították. Egyetlen szempont lebegett a szemük előtt: a hatékonyság; ennek érdekében még a kényelmet is feláldozták, ahogyan a Forma-1 versenyautóiban sem a vezető kényelme a fő szempont...

Egyre többen esküsznek rá, hogy a Microsoft Windows NT (mások szerint az OS/2) rövid időn belül kipusztítja majd a Unixot. Az ilyen véleményekkel legyünk óvatosak: a számítástechnika története mutat példákat arra, hogy bizonyos rendszerek meglepően nagy ellenállóképességgel bírnak, dacára annak, hogy a józan ész szerint már rég ki kellett volna halniuk (gondoljunk csak a FORTRAN-ra!) Az a nap tehát még nagyon messze van (ha ugyan eljön egyáltalán) amelyen a világ utolsó Unix alatt futó számítógépén is kidobják az operációs rendszert, ezért mindenképpen érdemes egy kicsit megbarátkoznunk a „modortalan öregúrral”.

A leckék áttanulmányozása után természetesen nem lesz belőlünk sem rendszergazda, sem Igazi Programozó, de annyit talán elérünk vele, hogy kikerülünk a „stupid user” kategóriából, néhány dolgot már magunk is meg tudunk csinálni, és talán már látni fogjuk, hogy a barátságtalan külső értékes belsőt takar.

Lássuk most, miről is lesz szó az egyes leckékben!

1. lecke
A legfontosabb dolgok, amiket a fájlokkal művelni lehet (másolás, törlés, mozgatás, stb.).

2. lecke
További fontos dolgok a fájlrendszerről.
Hozzáférési jogok értelmezése és megváltoztatása.

3. lecke
A shell.
Be- és kimenet átirányítása, processzek futtatása a háttérben.

4. lecke
Mit honnan lehet megtudni? A tudás forrásai.

5. lecke
Szövegszerkesztés Unix alatt (vi)

6. lecke
További hasznos tudnivalók a shellről.
Környezeti változók, a .cshrc fájl, alias parancs.
Terminál beállítások.

7. lecke
A hálózattal kapcsolatos legfontosabb parancsok és tudnivalók.

8. lecke
Keresés és szűrés. Grep, find, awk.
Reguláris kifejezések.

9. lecke
Archiválás. tar, gzip, compress, uuencode és uudecode.

10. lecke
OK, de mire jó ez az egész?
Tippek a Unix parancsok értelmes használatára.

11. lecke
A Unix shell humorérzéke.

Első lecke

„Szerintem a Unix olyan, mint a női lélek: bonyolult és csodálatos. (Az igazi buherátorok szerint csodálatosabb :-)”

(Csáky István)

A legfontosabb fájl- és könyvtárkezelő parancsok

Ebben a leckében az alábbi parancsokról lesz szó: cd, ls, cp, mv, rm, mkdir, rmdir, pwd, cat, more, head, tail, wc

A parancsok általános szintaxisa

A Unix rengeteg parancsot és kisebb-nagyobb hasznos segédprogramot tartalmaz; ezek közös jellemzője, hogy nevük - jóllehet néha nehezen felderíthető módon - utal feladatukra, és valamennyit a Unix parancssorában kell begépelni. (A parancssor elejét a prompt jelzi - ez rendszerint % vagy $ jel a sor elején, de bármi más is lehet, gépenként változik!)

A parancsokat úgy hajtjuk végre, hogy begépeljük a nevüket a parancssorba. A parancs neve után szóközökkel elválasztva kell megadni a paramétereket: ezek határozzák meg közelebbről, hogy a parancs mit is csináljon. A paraméterek közül azokat amelyek mínusz jellel kezdődnek, ezentúl kapcsolóknak fogjuk hívni, mivel ezekkel lehet bekapcsolni egy parancs különféle üzemmódjait. A kapcsolókból rendszerint egyszerre több is megadható, ilyenkor általában elegendő egy mínusz jelet kiírni és felsorolni utána a kapcsolókat (ezt nem minden rendszer teszi lehetővé!). Például az ls -l -a -r parancs megadható így is: ls -alr

Elsősorban PC-n nevelkedett felhasználóknak okozhat kellemetlen meglepetést a Unix szűkszavúsága. Mindig csak annyi üzenetet kapunk, amennyit a rendszer elengedhetetlenül szükségesnek tart. Ezért ne lepődjünk meg, ha egy parancs végrehajtása után semmi sem jelenik meg a képernyőn: ez azt jelenti hogy a parancs hibátlanul végrehajtódott. A Unix programok sikeres végrehajtás esetén soha nem jeleznek vissza, csak hiba esetén.

A Unix törekszik az egységességre, ennek szellemében minden interaktív parancsból ugyanúgy kell kilépni. (Interaktívnak nevezzük azokat a parancsokat, amelyek futás közben elvárják, hogy begépeljünk nekik valamit.) Tehát ha ki akarunk lépni egy ilyen programból, üssük le a <Ctrl-d> billentyűkombinációt.

Ha a parancssorban ütjük le a <Ctrl-d>-t, akkor kijelentkezünk a rendszerből (néhány gépen ez le van tiltva, helyette a logout parancsot kell használnunk). Ez összhangban van az előző alapelvvel, mert amikor be vagyunk jelentkezve, valójában egy programot futtatunk (ez a parancsértelmező, amit shell-nek vagy buroknak hívnak): kijelentkezéskor ebből a programból lépünk ki.

Ugyancsak alapelv, hogy a kis- és a nagybetűk különbözőnek számítanak. Ügyeljünk erre, és lehetőleg semmit ne gépeljük csupa nagybetűvel!

Fájlok

A Unix legfontosabb alapfogalma a fájl. Nem csak programokat és adatokat tárolunk a fájlokban, de a parancsok is futtatható fájlok, sőt még a terminálokra is ugyanúgy írunk, mint egy közönséges adatfájlba. Röviden: a Unixban minden fájl.

A fájlnevek maximális hossza általában 255 karakter, és szinte bármilyen karakterből állhat(nak). Néhány karakternek azonban különleges szerepe van, ezért célszerű a névadáshoz csak az angol ábécé betűit, a számokat, a pontot, valamint a mínusz és az aláhúzás jelet használni. A kis és nagybetűk a fájlnevekben különbözőnek számítanak, csakúgy mint a parancsokban. Lehetőleg ne használjuk a nevekben csillagot (*) és első karakterként pontot.

Listázás

Az ls (list) parancs megmutatja az aktuális könyvtárban található fájlok neveit. (Az aktuális alkönyvtárunk bejelentkezés után a munkakönyvtár, az un. „home directory”. Ide tehetjük a saját dolgainkat, például leveleinket, programjainkat, adatainkat, stb.)

A lista alapértelmezés szerint ábécé sorrendben sorolja fel a neveket, de alkalmas kapcsolók segítségével más sorrendet is előírhatunk. Például az

```
ls -t
```

parancs az utolsó módosítás dátuma szerint rendezve listázza ki a fájlok neveit úgy, hogy az utoljára módosított fájl lesz az első a listán. A -l kapcsoló un. hosszú (long) listát ad:

```
ls -l
-rw-r--r-- 1 owner user 14649 Sep 6 09:54 nevek.txt
```

A dátum és az idő megadja hogy a nevek.txt nevű fájl mikor volt utoljára módosítva. Az 14649-es szám a fájlban található karakterek számát jelenti. A user a csoport neve, amelyikbe a felhasználó tartozik. Az owner a fájl „tulajdonosát”, vagyis annak a felhasználónak az user-name-jét, aki a fájlt létrehozta. A „-rw-r--r--” kódsorozat a fájl elérési jogát mutatja: ezt a fájlt bárki elolvashatja, de csak a tulajdonos (owner) írhatja. Az elérési jogokról részletesen a második leckében lesz szó.

A kapcsolókból egyszerre több is megadható: például az ls -al parancs kilistázza valamennyi fájlt (all) beleértve azokat is, amelyek neve ponttal kezdődik, hosszú (long) formában.

Azt is előírhatjuk, hogy csak bizonyos fájlokat listázzon, például az

```
ls -l *c
```

kilistázza valamennyi c betűre végződő nevű fájlt. (A * karakter jelentése: „ezen a helyen bármilyen karakter(ek) állhatnak, tetszőleges számban; ez a szám akár nulla is lehet”). Jegyezzük meg, hogy mindig a kapcsolókat (-l) kell első helyen megadni, és a fájlnevet a másodikon.

A csillaghoz hasonló metakarakter a kérdőjel, jelentése „ezen a helyen bármilyen karakter állhat”. Így tehát az

```
ls a???
```

parancs az összes *a* betűvel kezdődő, négy karakter hosszúságú fájlt fogja kilistázni.

Az ls parancs alapértelmezésben nem mutatja meg azokat a fájlokat amelyeknek ponttal (.) kezdődik a nevük. Ha ezeket is látni szeretnénk, használjuk a -a kapcsolót:

```
ls -a
```

Másolás, törlés és átnevezés

Az mv parancs átnevez (move = mozgat) egy fájlt egy másik névre. Tehát az

```
mv régi_név új_név
```

parancs megszünteti a régi_név fájlt és létrehoz helyette egy új_név nevűt. Ha az új_név nevű fájl már létezik, akkor felülírja azt a régi_név fájl tartalmával, rendszerint anélkül, hogy megkérdezne minket; komolyan felül akarjuk-e írni az új_név fájlt.

Ha másolatot akarunk készíteni egy fájlról, akkor ezt a cp paranccsal tehetjük meg.

```
cp eredeti másolat
```

Ekkor létrejön egy másolat nevű fájl, amely méretben és tartalomban megegyezik az eredeti fájllal.

Ha egy fájlra már nincs szükségünk, letörölhetjük az rm (remove) paranccsal.

```
rm fájl1 fájl2 fájl3
```

Ez a parancs letörli a fájl1, fájl2 és fájl3 nevű fájlokat az aktuális alkönyvtárból.

FONTOS!

Ha egy fájlt letörlünk vagy felülírunk, akkor az megsemmisül és semmilyen eszközzel sem lehet visszaállítani!

Könyvtárak

A UNIX hierarchikus felépítésű, ami azt jelenti, hogy a fájlokat könyvtárakban tárolja. Egy könyvtárból alkönyvtárak nyílhatnak, amelyek ugyancsak tartalmazhatnak további fájlokat és alkönyvtárakat, és így tovább. Mindig van egy aktuális könyvtár, ahol éppen dolgozunk (working directory) - ez azt jelenti, hogy a fájlokra vonatkozó parancsok (ls, cp, mv, cat, stb.) az aktuális könyvtárban lévő fájlokra fognak vonatkozni.

Bejelentkezéskor a munkakönyvtárba kerülünk (home directory), de szabadon mozogtatunk az ebből nyíló alkönyvtárakban sőt, esetenként még más felhasználók alkönyvtáraiba is átmehetünk.

Alkönyvtárat az mkdir paranccsal tudunk létrehozni, belelépni pedig a chdir vagy cd paranccsal tudunk. Tehát az alábbi két parancs

```
mkdir alkonyvtar
cd alkonyvtar
```

létrehoz egy, az aktuális könyvtárból nyíló alkönyvtárat, és bele is lép. Ha paraméterek nélkül adjuk ki a cd parancsot, akkor visszajutunk a home könyvtárba. Ha most kiadjuk az ls -l parancsot, a fájlok listájában az imént létrehozott alkönyvtár is szerepelni fog:

```
drwxr-xr-x 2 owner group 32 Nov 22 24:32 alkonyvtar
```

A sor elején szereplő d betű jelzi, hogy alkönyvtárról van szó. A paraméterek nélküli ls parancs által előállított listáról sajnos nem tudjuk megállapítani, hogy melyik név jelent alkönyvtárat és melyik jelent fájlt.

Az rmdir parancs kitöröl egy alkönyvtárat, ha az üres. Ha a törölni kívánt alkönyvtárban fájlok vannak, azokat az rm paranccsal törölhetjük le. Az

```
rm -r könyvtár_név
```

parancs nemcsak a kijelölt könyvtárat törli le, hanem belőle nyíló valamennyi könyvtárat és alkönyvtárat is. Így egy mozdulattal eltakaríthatunk egy egész könyvtárstruktúrát, ezért hát legyünk óvatosak!

A gyökér könyvtárnak nincs neve és szülő könyvtára. A / jel jelenti a gyökeret (root), alkönyvtárai pedig az usr, home, stb. a /usr, /home, /stb. hivatkozással érhetők el. Ezt a hivatkozást elérési útvonalnak (pathname) hívják. Ez független attól az alkönyvtártól, amelyikben éppen tartózkodunk. A /usr alkönyvtárnak további alkönyvtárai vannak, például bin, etc, stb. Ezek elérési útvonala /usr/bin, /usr/etc és így tovább.

Ha egy fájlt az aktuális könyvtárból egy másik könyvtárba akarunk másolni vagy mozgatni, meg kell adnunk a cél könyvtár nevét is. Ha a művelet tárgyát képző fájl nincs az aktuális alkönyvtárban, akkor arról is meg kell adni, hogy melyik könyvtárban találjuk meg. Az alábbi parancs az aktuális könyvtárban lévő level1 fájlt átmásolja az (aktuális könyvtárból nyíló) mail könyvtárba:

```
cp level1 mail
```

A következő példa feltételezi, hogy éppen nem tudjuk, melyik könyvtárban vagyunk, és a gyökérből nyíló public/news alkönyvtárból szeretnénk „hazamásolni” (a „home directory”-nkba) a last-news nevű fájlt. Íme:

```
cp /public/news/last-news $HOME
```

A $HOME jel a munkakönyvtárunk rövid neve, így nem kell kiírnunk az általában tekintélyes hosszúságú (/public/users/dept/staff/.....) munka-könyvtár nevét. Egyes rendszerekben (az un. csh-ban) a $HOME jelnek van egy még rövidebb neve is: a ~ jel. Erről a Shellről szóló fejezetben lesz szó.

Az aktuális könyvtárnak is van egy rövidített neve, mégpedig a pont (.). A következő parancs a /usr/news/legujabb nevű fájt másolja át oda, ahol éppen tartózkodunk - vagyis az aktuális alkönyvtárba.

```
cp /usr/news/legujabb .
```

Végül két hasznos parancs: ha elfelejtettük hogy éppen melyik alkönyvtárban vagyunk, akkor a pwd (print working directory) paranccsal megtudakolhatjuk, a paraméterek nélküli cd parancs pedig hazavisz minket a munkakönyvtárunkba (home directory).

Műveletek szövegfájlokkal

A legegyszerűbb dolog, amit egy szövegfájllal csinálni lehet, hogy tartalmát megjelenítjük a képernyőn. Ezt a cat paranccsal érhetjük el.

```
cat fájl_név
```

Erre a fájl tartalma megjelenik a képernyőn. A cat parancsnak több fájlnevet is átadhatunk, ilyenkor egymás után jeleníti meg a fájlok tartalmát. Ha a szöveg több, mint amennyi egy képernyőn elfér, akkor leszalad a képről. Ilyenkor használjuk a more parancsot.

```
more fájl_név
```

A szövegből most egy képernyőnyi rész jelenik meg, a szóköz (space) billentyű lenyomására megmutatja a következő oldalt. Ha nem akarjuk végignézni az egész fájlt, üssük le a q betűt vagy a <Ctrl-c>-t.

Egy fájl első sorait a head programmal lehet megjeleníteni:

```
head -15 fájl_név
```

A fájl első tizenöt sorát listázza ki. Időnként szükséges, hogy ne a fájl elejét, hanem a végét tanulmányozzuk. Ezt a tail paranccsal tehetjük meg, ez a példa a szövegfájl utolsó három sorát mutatja meg:

```
tail -3 fájl_név
```

A wc (word count) parancs megszámolja a szövegfájl sorainak, szavainak és karaktereinek számát: eredményül e három számot írja ki a képernyőre.

```
wc fájl_név
```

A -l kapcsolót megadva csak a sorok számát írja ki, -w esetén csak a szavak számát, -c-re pedig csak a karakterek számát.

Feladatok

A feladatokat lehetőleg mindenki egyénileg oldja meg, és igyekezzen a felmerülő problémákat a lecke újraolvasásával és gondolkodással megoldani!

Ha nem sikerül, írja meg a problémát az orlando@mars.sztaki.hu címre![*]

A FELADATOK MEGOLDÁSÁT NEM KELL BEKÜLDENI!!!!

Mindenki saját belátása szerint gyakoroljon, és csak akkor írjon, ha segítségre van szüksége. A gyakran feltett kérdésekre a választ mindenkinek el fogjuk küldeni egy FAQ fájlban.

Első lépésként hozzunk létre egy egyszerű szövegfájlt! Ezt úgy tehetjük meg, hogy cat bemenetét nem egy fájlból, hanem a terminálról vesszük; kimenetét pedig nem a terminálra küldjük, hanem egy fájlba. (Minthogy a Unix a terminált is fájlnak nézi, ez nem jelent neki különösebb nehézséget.) Az átirányításról egy későbbi leckében lesz szó; most elégedjünk meg annyival, hogy a „proba1” nevű fájl létrehozásához a következőt kell begépelnünk a parancssorba:

```
cat >proba1
```

Ezután kezdjünk el gépelni; a sor végén üssük le az Entert, írjunk újabb sorokat, stb. Ha meguntuk a gépelést és a fájl létrehozását be akarjuk fejezni, akkor lépjünk ki a parancsból, azaz üssük le a <Ctrl-d> billentyűkombinációt.

Most pedig...

- Hozzunk létre a „proba1” fájlból egy-egy másolatot „proba2” és „probalkozas” néven!
- Hozzunk létre egy „masik1” nevű fájlt a fent ismertetett módon! (Azaz a cat paranccsal.) Írjunk bele valami mást, mint amit a proba1-be írtunk!
- A „masik1” fájlból készítsünk másolatot „masik-file” néven!

[*] Ez a cím már nem létezik! [a szerk.]

- Listázzuk ki egymás után a proba1 és masik1 fájlt, egyetlen cat paranccsal!
- Ismételjük meg az előző parancsot úgy, hogy kimenetét irányítsuk át egy nagy_file nevű fájlba, a > jel segítségével! Hány sort, hány karaktert és hány szót tartalmaz az így létrejött fájl? (Még a nagyon-nagyon kezdők se manuálisan számolják meg!)
- Nézzük meg, hogy most milyen fájlok vannak az alkönyvtárunkban!
- Ellenőrizzük le, hogy csakugyan a „home directory”-ban vagyunk-e? Ha nem, akkor térjünk vissza oda!
- Hozzunk létre egy „proba” és egy „masik” nevű alkönyvtárat a munkakönyvtárunkból nyílóan!
- Valamennyi „proba”-val kezdődő fájlt mozgassunk a „proba” alkönyvtárba, a „masik”-kal kezdődőeket pedig a „masik” alkönyvtárba (használjuk a * karaktert)!
- Lépjünk vissza a munkakönyvtárunkba (home directory) és adjuk ki az ls -R (nagy R betűvel!) parancsot! Mit tapasztalunk?
- Lépjünk be valamelyik általunk létrehozott alkönyvtárba (proba vagy masik) és listázzuk ki az ott található fájlok tartalmát a képernyőn! Hogyan lehet egyetlen paranccsal valamennyi fájl tartalmát kilistázni?

Második lecke

Ebben a leckében a következő parancsokról lesz szó: chmod, file, ln, du

Mi van még a fájlrendszerben?

Az első leckén való átvergődés után mindenki vérbeli Unix buherátornak képzelheti magát, de sajnos ez nem így van. A Unix fontos tulajdonsága, hogy többfelhasználós (multiuser), tehát biztosítania kell, hogy a felhasználók egymás fájljaihoz ne nyúlhassanak hozzá. Ennek elérése érdekében találták ki a Unix alkotói a fájl-elérési jogokat (file permissions). Ennek az a lényege, hogy a tulajdonos (user vagy owner) kontrollálja azt, hogy ki férhet hozzá az általa a gépen elhelyezett fájlokhoz, könyvtárakhoz.

A Unix lehetőséget biztosít arra is, hogy felhasználói csoportok tagjai fájlokat megosszanak egymás között, és a közös fájlok lehetnek a csoport (és nem egyetlen felhasználó) tulajdonában.

Háromféle fájl-elérési jog létezik: olvasási (read), írási (write) és futtatási (execute).

Ezeket a jogokat három osztályba osztva adhatjuk meg: magunkra (user, owner), a csoportunkra (group) és mindenki másra (world, others) nézve.

- Az olvasási jog lehetővé teszi, hogy a felhasználó a fájl tartalmát megnézhesse, könyvtárak esetén pedig, hogy azt kilistázhassa (ls parancs).
- Az írási jog a felhasználó számára engedélyezi a fájlok írását, módosítását, könyvtárak esetében pedig új fájlok létrehozását és a régiek törlését.
- Végül a futtatási jog lehetővé teszi a programok, burokprogramok (shell script-ek - amennyiben a fájl egy program) - futtatását; könyvtárak esetén az oda való belépést (cd parancs).

Hogy is néz ez ki a valóságban?

Itt az ideje, hogy ezt konkrétan is lássuk: listázzuk ki a könyvtárunkat (ls parancs; a „-l” és a „-g” kapcsolókkal, így kilistázzuk a fájl-elérési engedélyeket és a felhasználó csoportját is):

```
woland> ls -lg
-rw-r--r-- 1 csaky student 6242 Aug 28 1992 applied.rtf
woland>
```

Megjegyzés: Egyes Unix változatoknál (System V alapúak) elegendő csak a -l kapcsoló használata.

A legelső mező mutatja a fájl-elérési jogokat. (Mezőkön egymástól szóközökkel elválasztott karaktersorozatokat értünk.) A harmadik a fájl tulajdonosát, a negyedik pedig azt, hogy a felhasználó milyen csoportba tartozik. Az utolsó mező pedig a fájlnév.

Tehát a fájl tulajdonosa egy csaky nevű felhasználó, aki a student csoportba tartozik, a „-rw-r--r--” pedig a jogokat jelenti a tulajdonos, a csoport, végül pedig mindenki más számára.

Az első karakter („-”) a fájl típusát jelöli. A „-” azt jelenti, hogy ez egy „sima” fájl (nem könyvtár, stb.). A következő három karakter „rw-” jelentése, hogy a tulajdonos (jelen esetben csaky) olvashatja és írhatja a fájlt, de nem futtathatja. Ez így van jól, mivel ez a fájl nem program, így nincs is nagyon értelme futtatni.

A következő három karakter, „r--”, a csoportra vonatkozó jogokat definiálja. E szerint a student csoportba tartozó felhasználók olvashatják a fájlt, amint ezt a kis „r” betű mutatja.

Az utolsó három karakter „r--” jelentése pedig az, hogy az előbbiekben felsoroltakon (owner, group) kívül esők is olvashatják a fájlt.

Fontos itt megjegyezni, hogy az egyes fájl-elérési jogok függnek annak a könyvtárnak az elérési jogától is, amelyben megtalálhatóak. Például, még ha egy fájl elérési joga „-rwxrwxrwx” (tehát mindenki mindent csinálhat vele), mások nem férhetnek hozzá, ha a könyvtár amiben van, tiltja az olvasási és futtatási jogokat. Tehát a legegyszerűbb módja annak, hogy fájljainkat mások számára hozzáférhetetlenné tegyük az, hogy a munkakönyvtárunk hozzáférési jogait „-rwx------”-ra állítjuk: így senkinek sem lesz hozzáférése se a munkakönyvtárunkhoz, se pedig ahhoz ami benne van.

Az elérési jogok megváltoztatása

Az elérési jogok megváltoztatása a chmod paranccsal lehetséges a tulajdonos számára.

Szintaxisa pedig:

```
chmod [augo][+-][rwx] fájlnév...
```

Lássuk, hogyan kell értelmezni egy ilyen szintaxis leírást! Az első szó (chmod) nyilván a parancs neve. Utána következnek a kapcsolók (lásd: első lecke!). A szintaxisban egyszerre felsorolják az összes lehetséges kapcsolót; a zárójelben lévő kapcsolók közül meg lehet adni egyszerre egyet vagy többet is. A fájlnév utáni három pont azt jelenti, hogy egy sorban több fájlnevet is megadhatunk, ilyenkor a Unix mindegyik megadott fájlra végrehajtja a parancsot.

Az [augo] kapcsolókkal írjuk elő, hogy kinek adjuk a jogot. Adhatunk jogokat a tulajdonosnak (u - user), a csoportnak (g - group), másoknak (o - others) vagy pedig mindenkinek egyszerre (a - all). A [+-] azt jelenti, hogy adhatunk (+) vagy elvehetünk (-) jogokat. Végül pedig, hogy olvasási (r - Read), írási (w - Write) vagy futtatási (eXecute) jogot adunk. Néhány példa:

`chmod a+r applied.rtf` - olvasási jog mindenki számára

`chmod +r applied.rtf` - ugyanaz, mivel az alapértelmezés az „all”

`chmod go-rwx applied.rtf` - a tulajdonoson kívül senkinek semmi

Természetesen az igazi Unix buherátorok nem így változtatják meg az elérése jogokat, hanem egy szám megadásával. Például:

```
chmod 644 applied.rtf
```

Ennek a parancsnak a hatására az applied.rtf nevű fájl elérési joga „-rw-r--r--”-ra változik.

Ez a megadási mód sem bonyolult: az olvasási jog száma 4, az írásé 2, a futtatásé pedig 1. Ha ezeket összeadjuk, akkor megkapjuk a számot, amit be kell írnunk. Természetesen ez csak egyetlen szám, a jogokat pedig a parancsnak ennek a változatában mind három osztály számára be kell állítanunk. Tehát „rw-” a tulajdonosnak az 4+2 = 6, „r--” a csoportnak az 4 és „r- -” mindenki másnak az 4; együtt 644.

További fontos parancsok

Mint láthattuk, egy fájlra be lehet állítani a futtatási jogot is, de előfordulhat olyan eset, amikor nem tudjuk, hogy egy fájl program-e vagy sem. Ilyenkor segít a file parancs, ami megpróbálja megállapítani egy fájlról, hogy az milyen típusú (vájtfülűek számára általában a gép típusát is elárulja). Ezt úgy teszi, hogy a fájl első párszáz karakterét beolvassa és az alapján a fájlt megpróbálja besorolni az általa ismert kategóriákba. Az én gépem (NeXT) például a következő általános kategóriákat ismeri: adat, ASCII szöveg, C nyelvű forráskód, FORTRAN forráskód, assembler forráskód, troff kimeneti fájl, shell script (ejtése: szkript), C-shell script, futtatható program.

Lássunk egy példát:

```
woland> file *
d2u.c:          c program text
keybind.c:      c program text
life:           Mach-O executable
life.c:         c program text
life.dat:       ascii text
life1.dat:      ascii text
slice.c:        c program text
trans.c:        c program text
unarj:          directory
woland>
```

Ezen a könyvtáron látható, hogy egy programozó fájljait rejti, aki C-ben végzi munkáit. A life program, ami a life.c c-program fordítása, adatait valószínűleg a life.dat és a life1.dat fájlokból veszi, mivel ezek sima szöveget tartalmaznak.

Linkek

Van olyan eset, amikor az ls -l parancs igencsak furcsa dolgot ír ki, például:

```
lrwxrwxrwx 1 root 18 Dec 14 02:27 LocalApps -> /HD/NextStuff/Apps/@
```

Ez az eddig számunkra teljesen ismeretlen dolog a link. A link arra jó, hogy egy fájlnak több neve legyen, esetenként külön könyvtárban. A fenti példában azt, hogy a fájl (esetünkben könyvtár) link, onnan tudható, hogy a nevében egy kis nyíl van és utána a hely, ahol a valódi fájl található. Tehát a LocalApps a /HD/NextStuff/Apps alteregója. Ezt a fajta linket szimbolikus (symbolic vagy soft) linknek nevezzük. (Ha az ls paranccsal nem hosszú listát kérünk, akkor csak a LocalApps nevet fogjuk látni).

A link másik fajtája a hard link, aminek bár funkciója ugyanaz, az operációs rendszer másképp kezeli. Ennél a fajtájú linknél nem lehet kideríteni (illetve nagyon nehezen) azt, hogy melyik fájlra mutat, és az eredeti fájl tulajdonosa lesz az új fájl tulajdonosa is.

Linkeket az ln paranccsal hozhatunk létre (a forrás- és célnév természetesen könyvtár is lehet):

```
ln [ -s ] forrásnév célnév
```

Az ln kapcsolók nélkül hard linket készít, ha pedig megadjuk az „-s” kapcsolót, akkor szimbolikus linket.

Például:

Hard link készítése:

```
woland> ls -l egy
-rw-r--r-- 1 csaky student   12 Aug 5 14:20 egy
woland> ln egy ketto
woland> ls -lg egy ketto
-rw-r--r-- 2 csaky student   12 Aug 5 14:20 egy
-rw-r--r-- 2 csaky student   12 Aug 5 14:21 ketto
```

(Most már ezt is eláruljuk: a második mező jelenti a fájlra mutató linkek számát.)

Most az egy és a ketto ugyanaz a fájl - tehát, ha az egyiket módosítjuk a másik is módosul, mivel csak a név más, a fájl ugyanaz. Úgy is felfoghatjuk, hogy egy fájlnak egyszerre két neve is van.

Soft, avagy szimbolikus link készítése:

```
woland> ln -s egy ketto
woland> ls -lg ketto
lrwxrwxrwx 1 csaky student 12 Aug 5 14:25 ketto -> egy@
```

Vegyük észre, hogy a fájl-elérési jogok kezdő „-” jele most l betűre változott. Innen (is) látszik, hogy szimbolikus linkről van szó.

A linkek törlése természetesen a szokásos fájltörlés paranccsal (rm) végezhető. Törlés szempontjából döntő különbség van a hard linkek és a soft linkek között. Ha letörlünk egy fájlt, amelyre link kapcsolatok mutattak, akkor a hard linkkel létrehozott új fájl továbbra is megmarad és tartalma megegyezik a letörölt fájléval. A soft linkkel létrehozott fájlnév ugyan megmarad, de ha olvasni próbálunk a fájlból, akkor hibaüzenetet kapunk.

Mire jók a linkek? Például szeretnénk, hogy ugyanaz a fájl egyszerre több helyen is látható legyen, de mégse kelljen belőle annyi darab másolatot készíteni, ahány helyen szükség van rá. Egy egészen egyszerű alkalmazás: e tanfolyam szervezése során a jelentkezők e-mail címeit egy .addressbook nevű fájlban tartjuk. Ez a név (egy Unix buherátor számára) meglehetősen hosszú, de nem változtathatjuk meg, mert akkor nem találná meg a levelezőprogram. Mivel elég gyakran kell nézelődnünk ebben a fájlban, egy szimbolikus link segítségével létrehoztunk egy cim nevű linket; ezután valamennyi parancsban a cim nevet is használhatjuk a .addressbook helyett.

Mennyi helyet foglalunk a lemezen?

Nagyon hasznos dolog ha tudjuk, hogy mennyi a fájljaink által elfoglalt diszkterület. A Unixban természetesen mindenre van parancs, ez esetünkben a du.

Szintaxisa:

```
du [ -a ] [ -s ] könyvtárnév
```

Példa:

```
woland> du $HOME
10 /Users/student/csaky/.NeXT/services
2 /Users/student/csaky/.NeXT/Colors
1 /Users/student/csaky/.NeXT/.NextTrash
..
..
..
2 /Users/student/csaky/mail
6003 /Users/student/csaky/pin-ups
6798 /Users/student/csaky
woland>
```

Az eredmény szerint csaky felhasználó elég sok helyet foglal: 6798 kilobájtot (a du kilobájtban adja az eredményt) azaz majdnem 7 megabájtot. Ennek nagyrészét azonban a pin-up alkönyvtárban található ízléses aktfotók teszik ki (ami nem feltétlenül része programozói munkájának).

Az „a” kapcsolóval azt lehet beállítani, hogy a du minden fájlunkról adjon információt, ne csak a könyvtárakról. Az „s” kapcsoló pedig arra való, hogy du csak egyetlen számot írjon ki, a teljes helyfoglalást.

```
woland> du -s $HOME
6798
woland>
```

Melyik könyvtárban mi van?

A Unixban igencsak szerteágazó és kiterjedt könyvtárszerkezetet találunk. Van néhány könyvtár, amelyet hagyományosan ugyanúgy hívnak minden Unix változatban, és a bennük található programok és adatfájlok is ugyanahhoz a témához kapcsolódnak. Például:

```
/
```

A könyvtárstruktúra gyökere (root directory), minden más könyvtár innen nyílik, illetve itt található a vmunix (vagy unix) nevű futtatható program, ami nem más, mint az operációs rendszer magja (kernel).

```
/bin
```

Bináris programok könyvtára, itt vannak a legfontosabb végrehajtható programok (bináris formában).

```
/dev
```

Itt vannak az eszközfájlok (terminálok, diszkek, CD ROM olvasók, stb.)

```
/etc
```

Egyéb rendszerfájlok, rendszerint a rendszeradminisztrációs fájlok. A legtöbb esetben itt van a jelszavakat tároló fájl is.

`/lib`

Fontosabb függvénykönyvtárak - ezeket csak a programozók használják.

`/tmp`

Átmeneti tárolásra szolgáló alkönyvtár. Ezt az alkönyvtárat rendszeres időközönként törlik, ezért soha ne tároljunk itt semmi értékeset!

`/usr`

A felhasználói fájlrendszer: munkakönyvtárak, stb.

`/usr/adm`

Rendszeradminisztrációs fájlok.

`/usr/bin`

A felhasználók által használatos futtatható bináris fájlok.

`/usr/man`

Az online dokumentáció, amelyet a man paranccsal tudunk megjeleníteni.

`/usr/local`

Egyéb nyilvános dolgok (hírek, stb.).

`/usr/spool`

Kommunikációs programok munkakönyvtárai.

`/usr/spool/mail`

Itt van valamennyi felhasználó postaládája (mailbox). Elolvasni természetesen nem tudjuk más leveleit, de azt megnézhetjük, hogy valakinek van- e levél a postaládájában.

Feladatok

- Ha már van valamilyen fájlunk - remélhetőleg igen, hiszen az előző leckéből tudjuk, hogy hogyan kell csinálni egyet - elkezdhetjük a munkát. Vigyázat! A feladatokban több ízben is szerepel a fájl törlése, ezért keressünk valami értéktelen fájlt gyakorlatozásaink céljára!
- Állítsuk be a munkakönyvtárunk hozzáférésének jogát egy biztonságos értékre; (rwxr-xr-x) az ajánlott.
- Változtassuk meg a kiszemelt fájlunk hozzáférési jogát „rwxrwxrwx”-re, profibbak használják a jogok számmal történő megadását. A beállított jogokat ellenőrizzük le (ls parancs)!
- Most változtassuk meg a jogokat úgy, hogy csak mi tudjuk olvasni, írni és futtatni a fájlt! Ellenőrizzük le a beállítást!
- Most változtassuk meg a jogokat „-r--------”-ra, azaz olyanra, hogy még mi se tudjuk írni! Most próbáljuk meg letörölni a fájlt! Sikerül-e vagy sem? Ha igen miért, ha nem miért nem?
- Nézzük meg a fájl típusát!
- Nézzük meg a /bin/sh fájl típusát.
- Hozzunk létre egy egyszerű szövegfájlt az első leckében leírt módon. Az így elkészített fájlra csináljunk egy hard linket „kemeny” néven, és egy soft linket „puha” néven! Nézzük meg (a cat vagy a more segítségével), hogy csakugyan azonos-e a tartalmuk!

- Csináljuk egy teljes listát, és a listából állapítsuk meg, hogy a „kemeny” és a „puha” fájlok közül melyik készült hard linkkel és melyik soft linkkel!
- Töröljük most le azt a fájlt, amelyikről a linket készítettük!
- Nézzük meg, hogy léteznek-e még a „kemeny” és a „puha” nevű fájlok! Ha igen, próbáljuk meg kilistázni tartalmukat a cat paranccsal! Mi történik és miért?
- Nézzük meg, hogy eddigi munkánkkal mennyi helyet foglalunk!

Harmadik lecke

„Természetes párhuzam él a nők és a Unix shellek között. Mindkét csoportban vannak egyszerű, csúnyácska típusok (sh) és érettebb, minden igényt kielégítő típusok (tcsh). (-:
Egy igazi buherátor természetesen az sh-val és az sh-típusú nőkkel is megtalálja a nyelvet! :-)*”*

(Csáky István)

A Unix shell

A cím rém rövid, de e nélkül a program nélkül (a shell valójában csak egy program Unix alatt, de a funkciója kiemeli a többi program közül) a munka elképzelhetetlen. Ugyanis ez a program az amelyik elindul mikor beszállunk egy Unixos rendszerbe, ennek adjuk ki a parancsokat és ez válaszol nekünk. (Funkciója hasonló a DOS command.com-jáéhoz). Shellből több fajta létezik, mivel a Unix lehetőséget ad belső érzületünk kifejezésére és nem köt egy megadott shellhez. Két fő kategória van: az un. „Bourne shell” és a „C shell”, ezek neve sh és csh. Nevükön kívül szintaxisukban különböznek: a csh a C programoknál használatos szintaktikát követi. (Én a csh egy kibővített változatát a tcsh-t használom, természetesen az sh-nak is vannak bővített változatai: a ksh, a bash és a zsh). Hogy ki mit használ teljesen vallás kérdése - a különféle shellek szolgáltatásaikban egyenrangúak!

Mint már korábban olvashattuk, a Unix parancsok nem, vagy nagyon ritkán jelzik, ha jól működnek. Természetesen ez első látásra furcsán hat, azonban nagyon is értelmes magyarázata van. (Sokan szidják a Unixot bonyolultsága miatt, azonban Unixban minden az összehangoltságon alapul). Az, hogy a Unix nem jelzi a hiba nélkül futást, azért van, hogy a parancsok be- és kimenetei más parancsoknak átadhatóak legyenek. A legtöbb program a bemenetről (standard input - stdin) veszi az adatait és a kimenetre (standard output - stdout) küldi az eredményt. Rendszerint a bemenet a billentyűzet és a kimenet és a hibacsatorna a képernyő.

Például: a sort program rendezi a bemenetről beérkező sorokat:

```
woland> sort
gyufa
izzok
csoko
<Ctrl-d>
csoko
gyufa
izzok
woland>
```

Megjegyzés: a <Ctrl-d> a control (Ctrl) és a d billentyűk együttes leütését jelenti!

Így máris abc sorrendbe raktuk a bevásárlólistát. Az élet kis dolgaiban is nyugodtan számíthatunk a Unixra.

A bemenet és a kimenet átirányítása

Mi történik azonban akkor, ha mindig csak ezt a pár dolgot vesszük, ezért szeretnénk megtartani a listánkat? A shell lehetőséget ad arra, hogy az kimenetet átirányítsuk (redirection), és ehhez csupán a „>” jelet kell hogy használjuk. Nézzük, hogy is működik ez.

```
woland> sort > mit_vegyek
gyufa
izzok
csoko
<Ctrl-d>
woland>
```

Ugyanezzel a módszerrel - átirányítás - hoztunk létre fájlt az első leckében (ott a cat parancs kimenetét irányítottuk át). Az eredmény nem látható, hanem a mit_vegyek nevű fájlba került. Nézzük meg, hogy valóban ez történt-e?

```
woland> cat mit_vegyek
csoko
gyufa
izzok
woland>
```

A kimenet átirányítása csupán ennyi munkánkba kerül. Ugyanez vonatkozik a bemenet átirányítására is, ennek jele „<”.

Például: csináljunk egy aruk nevű fájlt, aminek tartalmát rendezzük:

```
woland> cat > aruk
izzok
gyufa
csoko
<Ctrl-d>
woland>
woland> sort < aruk
csoko
gyufa
izzok
woland>
```

Fontos jel még a „>>”, ami szintén a kimenet átirányítását jelenti; különbség közte és a szimpla átirányítójel között annyi, hogy ha ezt használjuk, a fájl amibe a kimenetet átirányítjuk, nem íródik felül. Ezzel a jellel a fájl tartalmához tudunk hozzáfűzni.

Megjegyzés: Természetesen a hibacsatornát is át lehet irányítani, erről bővebben az ötödik leckében olvashatunk majd!

Csövek

Arról már tudunk, hogy egy-egy program eredményét hogy lehet fájlba irányítani, azonban a lehetőség van arra, hogy egy program bemenetét egy másik programtól vegye. Erre találták ki Unixban a cső (pipe) fogalmát. Azt, hogy egy program kimenetét egy másik bemenetére szeretnénk átirányítani a „|" jellel jelezzük.

Vegyünk egy konkrét példát - szeretnénk a munkakönyvtárunkat fordított abc-sorrendben kinyomtatni. Eddigi tudásunkkal hogyan tudjuk ezt megvalósítani?

```
woland> ls -l >file_list
```

(Az igazi buherátor csak „bő" könyvtárlistát szeret olvasni.)

```
woland> cat file_list
total 8
drwxr-xr-x 2 csaky 1024 Jul 21 1992   Apps
drwxr-xr-x 8 csaky 1024 Oct 19 13:35  Library
drwxr-xr-x 3 csaky 1024 Jul 25 1992   Mailboxes
drwxr-xr-x 2 csaky 1024 Nov 9  19:05  NewFolder
-rw------- 1 csaky 5    Dec 6  16:46  dead.letter
drwx------ 2 csaky 1024 Jan 12 11:39  mail
drwxr-xr-x 2 csaky 1024 Jan 17 17:23  pin-ups
drwxr-xr-x 2 csaky 1024 Oct 5  09:02  tabla.draw~
woland> sort +7r < file_list

total 8
drwxr-xr-x 2 csaky 1024 Oct 5  09:02 tabla.draw~
drwxr-xr-x 2 csaky 1024 Jan 17 17:23 pin-ups
drwx------ 2 csaky 1024 Jan 12 11:39 mail
-rw------- 1 csaky 5    Dec 6  16:46 dead.letter
drwxr-xr-x 2 csaky 1024 Nov 9  19:05 NewFolder
drwxr-xr-x 3 csaky 1024 Jul 25 1992  Mailboxes
drwxr-xr-x 8 csaky 1024 Oct 19 13:35 Library
drwxr-xr-x 2 csaky 1024 Jul 21 1992  Apps
woland>
```

A sort kapcsolói csak azt jelentik, hogy a 8. mezőtől kezdjen rendezni (egyet ki kell vonni a mező sorszámából!), és hogy fordítva (r - reverse). A mező definícióját a második leckében találhatjuk meg!

Ezt a cső használatával egy sorban is el lehet intézni, és az eredmény ugyanaz:

```
woland> ls -l | sort +7r
```

Nagyon jó szolgálatot tesznek a csövek a hosszú könyvtárlisták nézegetésénél is. Az ls -l | more paranccsal oldalanként nézegethetjük a listát, így az nem fut le a képernyőről.

Processzek

A Unix egyik fő jellemzője amellett, hogy többfelhasználós, az, hogy egyszerre több program futtatására képes (multitasking). Ezt a lehetőségét már a shell használata során is élvezhetjük, tehát az igazi Unix buherátor nem éli át azt a lealacsonyító érzést, hogy várnia kell egy program lefutására ahhoz, hogy egy másikat indíthasson. (Hahó MS-DOS).

A ps paranccsal meg tudjuk nézni, hogy milyen processzeink (programjaink) futnak.

```
woland> ps
PID TT STAT TIME COMMAND
3902 p0 S 0:01 -csh (tcsh)
woland>
```

Amint látjuk, most a felhasználónak csak egy processze fut, mégpedig a shell (tcsh). A listán még egy nagyon fontos dolog szerepel, a processz egyedi azonosítója (PID - process ID), esetünkben 3902.

A futó processzet a shell job néven ismeri (a processz és a job ugyanazt jelentik!). A shell lehetőséget ad a jobok manipulálására, ezt job controlnak hívjuk; ami azt jelenti, hogy egyszerre több egymástól különböző dolgot csinálhatunk, és váltogathatjuk, hogy éppen melyik futását akarjuk figyelemmel kísérni. Például éppen szerkesztünk egy programot, de eszünkbe jut, hogy már meg kellett volna írni egy e-mailt; erre kínál megoldást a job control.

Előtérben (foreground) és háttérben (background) való futtatás

A jobok futhatnak előtérben és háttérben, előtérben azonban egyszerre csak egy job lehet. Az előtérben lévő job kapja a bemenetet a billentyűzetről és küldi a kimenetét a képernyőre (hacsak át nem irányítottuk). A háttérben futó jobok azonban semmilyen billentyűzetről jövő bemenetet sem kapnak meg. (Vigyázat, mert a képernyőre attól még írhatnak!)

A háttérben futtatás nagyon jól használható abban az esetben ha egy olyan programot futtatunk ami sokáig fut és semmi érdekeset nem produkál (pl. nagy fájlok tömörítése az előtérben - csak megrögzött DOS felhasználóknak ajánlott). Ebben az esetben nincs értelme arra várni, hogy a job lefusson, mikor közben valami mást is tudnánk csinálni (a számítógéppel). Futtassuk a háttérben!

A jobok lehetnek felfüggesztett állapotban (suspended); ez azt jelenti, hogy a jobot egy kis időre megállítjuk. Az ilyen job se az előtérben sem pedig a háttérben nem fut. Miután felfüggesztettünk egy jobot, a futtatását lehet folytatni tetszés szerint az előtérben, vagy akár a háttérben. A job ott folytatja a munkát, ahol előzőleg abbahagyta.

Figyelem! Egy job felfüggesztése nem ugyanaz, mint a megszakítása; ugyanis megszakítás esetén (Ctrl-c) a job már nem tudja a munkáját folytatni!

Vegyünk egy példát: (A yes parancs y karakterek végtelen sorozatát küldi a kimenetére.)

```
woland> yes
y
y
y
y
y
```

Ez így menne a végtelenségig, ha meg nem szakítanánk. Ha csak nem az y-ok végtelen sorát akarjuk látni (márpedig mi mást szeretnénk csinálni) irányítsuk át egy alkalmas helyre: /dev/null - egy ún. fekete lyuk: a beleirányított összes információt elnyeli.

```
woland> yes > /dev/null
```

Megy ez, de a job még mindig az előtérben fut. Hogy a háttérbe tegyük, a „&” jelet kell alkalmaznunk, ugyanis ez jelzi a Unix számára azt a vágyunkat, hogy a jobot a háttérben akarjuk futtatni.

```
woland> yes > /dev/null &
[1] 3954
woland>
```

Amint látjuk, a shell promptját visszakaptuk, két furcsa számmal együtt. Mik ezek, és a yes fut-e egyáltalán?

Az [1] a yes processz job számát jelenti a shell számára. A shell minden futó jobhoz rendel egy számot. Mivel még csak a yes-t futtatjuk, ezért annak száma 1. A másik szám a processz azonosítója (PID mező a ps-ben) - a processzre mindkét számmal utalhatunk.

Nézzük meg, hogy valóban fut-e a processzünk?

```
woland> jobs
[1] + Running yes > /dev/null
woland>
```

Igen, a processzünk fut. A ps paranccsal is megbizonyosodhatunk efelől.

```
woland> ps
PID TT STAT TIME COMMAND
3902 p0 S 0:01 -tcsh (tcsh)
3954 p0 R 6:00 yes
woland>
```

Most már tudunk jobot a háttérbe tenni, de valahogy le is kéne állítani, ha esetleg magától nem áll le (a yes-nél ez nagyon valószínű). Erre való a kill parancs, paramétere lehet a job száma vagy pedig a processz azonosítója. Használjuk:

```
woland> kill %1
```

vagy

```
woland> kill 3954
```

A % jellel jelezzük azt, hogy job számot adunk meg. Ha processz azonosítót adunk, akkor nincs szükség másra csak a processz azonosítójára. Lássuk, mi történt:

```
woland> jobs
[1] + Terminated yes > /dev/null
woland>
```

Mint látható, a jobot kilőttük, a következő job listában már nem is fog szerepelni.

Megjegyzés: Előfordulhat, hogy egy háttérben futó processzt sehogyan sem tudunk megállítani. Használjuk ilyenkor a „kill -KILL processz száma (vagy job száma)”, vagy „kill -9 processz száma (vagy job száma)” parancsot (a két parancs ekvivalens)! Ez a Unix számára azt jelenti, hogy mindenképp lője ki a processzt.

Van egy másik lehetőség is arra, hogy egy jobot a háttérben futtassunk, mégpedig az, hogy futását felfüggesztjük, majd a háttérbe téve futtatjuk tovább.

Indítsuk el:

```
woland> yes > /dev/null
```

Majd nyomjuk meg a <Ctrl-z>-t, amivel a job futását felfüggesztjük.

```
woland> yes > /dev/null
<Ctrl-z>
Stopped
woland>
```

A program futását ezután mind az előtérben, mind a háttérben folytathatjuk. Az előtérben való folytatásra az fg parancs szolgál, a háttérben futtatásra pedig a bg.

```
woland> fg
yes > /dev/null
<Ctrl-c>
woland> bg
[1] yes > /dev/null &
woland>
```

Természetesen mind az fg mind a bg parancsnak is megadhatunk job számot és így akár háttérben futó jobokat is vissza lehet hozni az előtérbe és fordítva.

```
woland> fg %2
```

vagy az egyszerűbb formát is használhatjuk:

```
woland> %2
```

Egyéb job vezérlő (control) parancsok

Még két parancsról kell ebben a fejezetben beszélnünk: ezek a nohup és az at. A nohup (jelentése no-hangup) arra szolgál, hogy egy processz azután is fusson, miután mi kiszálltunk a rendszerből; különben a háttérben futó processzek megállnak.

Szintaxisa: nohup <parancs>

Például:

```
nohup compress nagyon_nagy_program &
```

Ezután nyugodtan hazamehetünk, a processz le fog futni és mire másnap bejövünk, az eredmény látható lesz.

Megjegyzés: egyes Unix változatokban nincs nohup; az &-el indított processzek akkor is tovább futnak, ha kilépünk.

Az at paranccsal egy programot egy meghatározott időpontban futtathatunk (igazi Unix buherátorok nem használják; úgyis mindig ott vannak a gép mellett, úgyhogy a kellő időben ők maguk tudják futtatni az ilyen programokat). Az at parancsot úgy használjuk, hogy megadjuk az időpontot, amikor el akarjuk indítani a kérdéses programot, majd leütjük az Entert. Erre (a legtöbb gépen) megjelenik az at> prompt, ahova be kell gépelnünk az elindítandó programok neveit. A listát a <Ctrl-d> leütésével zárjuk. Ha a program(ok) futtatásakor nem vagyunk bejelentkezve, akkor a program által generált kimenetet a rendszer e-mailben elküldi nekünk.

Például:

```
at 8:00am jan 24
at> csenget
<Ctrl-d>
```

Ennek eredményeképp a gép január 24.-én reggel 8-kor csengetni fog (feltéve ha a csenget program valójában csenget, nem pedig perverz leveleket küld a bejelentkezve levő felhasználóknak).

Feladatok

- Nézzük meg milyen shellünk van (ps).
- Próbáljuk meg kilőni az aktuális shellünket (kill -KILL). Mi történt? Miért?
- Nézzük meg az aktuális dátumot és időt (date) és tegyük fel az at várólistájára a következő parancsot: echo "Haho"| Mail -s "greeting" orlando@mars.sztaki.hu úgy, hogy az öt perc múlva induljon!
- Nehéz! Fűzzük össze a /etc könyvtárban lévő fájlokat (cat) és ennek kimenetét adjuk át a szabványos Unix tömörítő programnak (compress), az eredményt irányítsuk egy fájlba (különben nagyon érdekes Unix élményben lesz részünk, mindenféle absztrakt karakterek jelennek meg a képernyőn)! Nézzük meg milyen tömörítési arányt sikerült elérnünk. (A du és az ls -l parancsok segítenek ebben!)
- Most ismételjük meg a feladatot a háttérben. Nézzük meg, hogy fut-e, és ha igen, akkor lőjük ki!
- Nézzük meg, hogy eddigi munkánkkal mennyi helyet foglalunk!

Negyedik lecke

„Ha már minden kísérleted csődöt mond,
olvasd el a használati utasítást”

(Murphy)

Információk a Unixról

Egy olyan bonyolult operációs rendszernél, mint amilyen a Unix, nagyon nagy szerepe van az írásos dokumentációnak. Ez a lecke arról szól, hogy hogyan és honnan lehet választ kapni a kérdéseinkre.

Angol nyelvű könyvek

Az információk legfontosabb forrása a rendszerrel együtt szállított dokumentáció. Minden valamire való Unixhoz félelmetes mennyiségű és súlyú dokumentáció érkezik (az írókat alighanem oldalszám szerint fizették), úgyhogy ahhoz is kell némi jártasság, hogy mikor melyik könyvet nyissuk ki. A „könyvek könyve” a Unix Reference Manual”. Ezt többnyire az alábbi fejezetekre osztják:

- A shellből kiadható parancsok.
- Rendszerhívások.
- A C nyelvű függvénykönyvtárak függvényei.
- Eszközkezelő programok (device drivers), adatátviteli formátumok.
- Fájlformátumok.
- Játékok és demók.
- Bárki számára hozzáférhető fájlok, táblázatok, TROFF makrók.
- Rendszeradminisztrációs és karbantartó programok.

E nyolc csoporton belül alcsoportokat is megkülönböztetünk.

A parancsokra való hivatkozásnál zárójelben azt is megadják, hogy a parancs leírása melyik kötetben található: ls(1), stb.

Általában a SUN gépek dokumentációit tartják legjobbnak, ezért a SUN dokumentációt bizonyos esetekben - kihasználva a Unix hordozhatóságát - más gépeknél is használhatjuk.

A rendszer dokumentációján kívül a Unixról számtalan további hosszabb-rövidebb, részletes és kevésbé részletes könyvet írtak már, ezek minden könyvtárban és könyvesboltban polcokat töltenek meg. Ha tudunk angolul, biztosan megtaláljuk az ízlésünknek, tudásunknak és pénztárcánknak megfelelő könyvet. A sok közül egy nagyon alapfokút említünk meg:

Peter Norton - Harley Hahn: Peter Norton's Guide to Unix (Bantam Books) (Ez a könyv úgy íródott, hogy a számítástechnikai analfabéták is megértsék.)

Magyar nyelvű könyvek

Magyar nyelven is megjelent a Unix-bibliának számító Kernighan-Pike könyv. Címe:

Brian W. Kernigham - Rob Pike: A Unix operációs rendszer (Műszaki könyvkiadó) Eredeti címe: The Unix programming environment.

Jelenleg is (1994 január) kapható Magyarországon, ára 800 forint körül van. Ha komolyan akarunk Unix-szal foglalkozni, akkor mindenképpen érdemes szert tennünk rá, mert hasznos és érdekes könyv, a hirdetéseket leszámítva mind a 350 oldala fontos dolgokat mond el a Unixról. Ez a legjobb magyar nyelven megjelent könyv a Unixról, amivel eddig találkoztunk.

Képernyőn megjeleníthető dokumentáció

Minden Unix rendszeren a Reference Manual első fejezetét felviszik a gépre is, ahol a man parancs segítségével lehet olvasgatni az egyes lapokat. Ha például kíváncsiak vagyunk rá, hogy mit ír a dokumentáció az sh programról, adjuk ki a man sh parancsot. Válaszul elönti a képernyőnket az sh program mindenre kiterjedő, részletes, és hat-nyolc oldalas leírása. (Komoly feladat az ilyen információözönből kikeresni azt az egy mondatot, amire történetesen szükségünk van.)

A man-ban minden parancs leírásával egy „lap" (Page) foglalkozik, még akkor is, ha egy ilyen lap esetenként több képernyőoldalt tesz ki. Nézzük meg egy példán, hogy a man lapjai milyen címszavakat tartalmaznak:

```
zeus% man sh

SH(1) Silicon Graphics SH(1)

NAME

sh, rsh - shell, the standard/restricted command programming
language
```

Ez alatt a címszó alatt egy sorban leírják, hogy a parancs mit csinál. Gyakori, hogy a logikailag összetartozó parancsokat egy lapon szerepeltetik a dokumentációban. (Mint például itt is).

```
SYNOPSIS

/bin/sh -acefhiknrstuvx [ args ]

/bin/rsh -acefhiknrstuvx [ args ]
```

Itt felsorolják, hogy hogyan lehet paraméterezni a parancsot, milyen kapcsolókat ismer, stb. (A szögletes zárójelek jelentéséről már volt szó a második leckében).

```
DESCRIPTION

sh is a command programming language that executes commands
read from a terminal or a file. sh is a restricted version
of the standard command interpreter sh; it is used to set
up login names and execution
```

Itt részletesen is ismertetik, hogy a parancs mit is csinál, hogyan és mire lehet használni, stb.

```
OPTIONS
```

Ez alatt a címszó alatt találjuk a parancs kapcsolóinak részletes ismertetését. Próbaképpen érdemes megnézni az ls parancs kapcsolóit.

```
FILES
/etc/profile
$HOME/.profile
```

Itt felsorolják azokat a fájlokat, amelyeknek valami közük van a parancshoz. (Például a program futás közben olvas valamelyikből, adatokat ment bele, vagy éppen a program működését lehet tartalmukkal befolyásolni.

```
SEE ALSO
cd(1), dup(2), echo(1), env(1), exec(2), fork(2), getopts(1),
getrlimit(2), intro(1), login(1), newgrp(1), pipe(2),
profile(4)
```

Bár első ránézésre nem látszik, de ez a legfontosabb címszó. Itt vannak felsorolva ugyanis azok a parancsok, amelyek kapcsolódnak a címszóban említett programhoz. Ha a fenti szöveg átolvasása után sem találtuk meg azt amit keresünk, akkor nézzük meg az itt felsorolt parancsokról szóló dokumentáció lapokat!

```
BUGS
```

Murphy szerint „Mindig marad egy huba” - a már felderített, de még ki nem javított hibákat itt sorolják fel.

Megjegyzés: Peter Norton szerint ha megkérdezünk egy Unix buherátort, hogy mit adjunk az édesanyánknak anyák napjára, ezt fogja válaszolni: „Nyomtasd ki neki az sh-ról szóló fejezetet a man-ból.”

A man parancs fájdalmas tulajdonsága, hogy a benne tárolt adatok a „mi mire való” elv alapján vannak szervezve, így egy kezdő, aki arra kíváncsi hogy „mit hogy kell megcsinálni” nehezen boldogul vele. A Unix programozó-barát és felhasználó-ellenes szemlélete kétségkívül a man parancsban ütközik ki a legélesebben.

A dokumentáció lapjait általában a more parancs segítségével listázzák ki. Ilyenkor kereshetünk is a szövegben, ha a --More- - felirathoz egy / jel után beírjuk a keresett szót. Ilyenkor a more parancs nem a következő oldalt jeleníti meg, hanem előre ugrik a szövegben annyit, hogy a keresett szót tartalmazó sor a képernyő legalsó sorába kerüljön. Jegyezzük meg azonban, hogy ezzel a módszerrel csak előrefele lehet keresni!

A whatis parancs segítségével egymondatos leírást kaphatunk arról, hogy a paraméterként megadott parancs mire való. A whatis program valójában nem csinál mást, mint kiírja a megadott parancs man lapjáról a NAME mező tartalmát.

Például:

```
zeus% whatis cat
cat (1) - concatenate and print files
zeus% whatis love
No manual entries seem appropriate for love.
```

Mi történik olyankor, ha nem arra vagyunk kíváncsiak, hogy egy parancs mit csinál, hanem adva van egy feladat, és azt akarjuk tudni, hogy melyik paranccsal lehet megoldani? Nos, ilyenkor legjobb, ha megkérdezünk egy buherátort. (Egy igazi buherátor természetesen ezt fogja válaszolni: „Bármelyikkel!") Némi támpontot adhat az, hogy a man adatbázisban a -k kapcsoló segítségével parancsnév helyett kereshetünk valamilyen kulcsszót is:

```
zeus% man -k 'execution time'
profile(2) - execution time profile
sprofile(2) - execution time profile or disjoint text spaces
```

Ugyanerre való az apropos parancs:

```
zeus% apropos close
ALcloseport (3A) - releases an audio port's resources
endclosedline (3G) - delimit the vertices of a closed line
CDclose (3) - closes a CD-ROM device
close (2) - close a file descriptor
closeobj (3G) - closes an object definition
.
.
.
```

Szerencsére vannak jóval fejlettebb megoldások: ilyen például a Silicon Graphics gépek insight nevű programja. Ez egy grafikus felületen futó hipertext rendszer, kereszthivatkozásokkal és példákkal, sajnos azonban csak grafikus terminálon lehet használni és természetesen csak a Silicon Graphics gépein.

Segítség elektronikus levélben

Kivétel nélkül minden Unixos gépnek van egy rendszergazdája, akinek írhatunk e-mailt, ha valamilyen kérdésünk van. Címe: root, de gyakran postmaster-nek vagy advisory-nak is hívják. Fontos szabály, hogy a világ összes rendszergazdája gyűlöli a buta kérdéseket, ezért csak akkor írjunk a root-nak, ha már minden egyéb kísérletünk csődöt mondott! A rendszergazda elvárja tőlünk, hogy előbb a manual átolvasásával, saját erőnkből próbáljuk meg megoldani a problémánkat. Másképp könnyen lehet, hogy egy túlságosan buta kérdésre válasz helyett az RTFM üzenetet kapjuk. (RTFM = Read The F***ing Manual!)

Feladatok

E lecke feladatai a man használatához kapcsolódnak, így sajnálatos módon csak az angolul értők tudják őket megoldani. (Nem is kell hangsúlyoznunk, hogy minden valamirevaló buherátor legalább annyit tud angolul, hogy a man-ból ki tudja bogarászni a szükséges információkat.) Ha tehát még nem tudunk angolul, akkor máris eggyel több érv szól amellett, hogy holnap reggel beiratkozzunk egy angol nyelvtanfolyamra...

A man segítségével adjunk választ a következő kérdésekre:

- Mit csinál a banner parancs? Próbáljuk ki!
- Mit csinál a sleep parancs?
- A harmadik leckében szó volt az at parancsról, amelynek segítségével egy megadott időpontban lehet végrehajtani egy programot. A végrehajtandó programok egy várólistára (queue) kerülnek. A várólistát meg lehet nézni, és a várakozó programokat meg végrehajtásuk előtt ki lehet venni a listából. Derítsük ki a manualból, hogy hogyan? Próbáljuk ki! (Tegyünk egy tetszőleges parancsot a várólistára az at paranccsal, majd töröljük ki, mielőtt a rendszer végrehajtaná!
- Olvassuk végig, hogy mit ír a manual az ls parancsról!

Ötödik lecke

„...az igazi programozó szerint az „azt kapod amit látsz” elv éppen olyan hibás az editoroknál, mint a nőknél. Nem, az igazi programozó az „ezt akartad, hát nesze” editort szereti - bonyolultat, titokzatosat, veszélyeset...”

(Ed Post: Az Igazi Programozó)

Szövegszerkesztés Unix alatt

Egy operációs rendszerben mindennapi feladat, hogy egy szövegfájl tartalmát meg kell változtatnunk. Ezt a célt szolgálják az editorok.

A Unix szabványos editora a vi, amelynek az az előnye, hogy minden rendszeren megtalálható. Az előnyök felsorolását ezzel ki is merítettük - mai szemmel nézve a vi rettenetesen elavult. A grafikus operációs rendszerek alatt futó szövegszerkesztőkkel (Word for Windows, Ami Pro, stb.) elkényeztetett felhasználónak a vi editortól olyan érzése támad, mintha késsel és villával próbálna megenni egy tál levest. Ha csak tehetjük, nagy ívben kerüljük el a vi-t!

A vi használatának elkerülésére a következő stratégiákat alkalmazhatjuk:

- Több más editor is létezik, például emacs, pico, joe. Nézzük meg a dokumentációban, vagy kérdezzük meg a rendszeradminisztrátortól, hogy ezek közül melyiket lehet használni.
- Ha valamilyen grafikus munkaállomáson dolgozunk, akkor ott kell lennie ablakos editornak. A Silicon Graphics gépeken jot a neve, a Sun-okon textedit, de biztosan könnyen elérhető helyen van. Ezek azonban csak grafikus terminálon futtathatók.
- Ha PC-ről jelentkezünk be, akkor nagyobb fájlok esetén érdemes ún. „file transfer” művelettel áthozni a fájlt DOS alá, ott a DOS editorral vagy akár valamilyen szövegszerkesztővel elvégezni a szükséges műveleteket, a kész fájlt pedig visszavinni a Unixra. A PC és a Unix közötti fájlátvitelről szintén az írásos dokumentációból, vagy a rendszeradminisztrátortól szerezhetünk információkat. A leggyakrabban használt program neve ftp (File Transfer Protocol), ennek használatáról a hálózatokról szóló leckében lesz szó.

Mégis érdemes a vi kezelését legalább alapszinten elsajátítani: ha netán kitennének minket egy lakatlan szigetre, és a sziget unixos gépén történetesen csak vi van, akkor ne álljunk ott tehetetlenül.

A vi editor

Először is meg kell értenünk a vi alapelveit.

Két üzemmódja van: szöveges mód és parancs mód. A parancs módban begépelt karaktereket parancsként értelmezi, például „karakter törlés” vagy „kurzor mozgatása jobbra”. Szöveges üzemmódban a bevitt karakterek bekerülnek a kurzor aktuális pozíciójába. Parancs módból úgy tudunk szöveges (text) üzemmódba váltani, hogy kiadjuk az a, C, i, o, O vagy R paran-

csok valamelyikét. Text módból az ESC billentyűvel jutunk parancs módba. Nagyon vigyázzunk, mert a képernyőn semmi sem utal rá, hogy éppen melyik üzemmódban vagyunk! Ez a fő ok, ami miatt az emberek ellenséges érzelmeket táplálnak a vi editorral szemben. Ne felejtsük el azonban, hogy a vi-t sok-sok évvel ezelőtt írták, amikor e terminálok még nem voltak olyan fejlettek mint napjainkban. Vigyázat, a kis- és nagybetűk különbözőnek számítanak!

Igyekezzünk fejből megtanulni a parancsokat; azt fogjuk tapasztalni, hogy minél több parancsot tudunk fejből, annál könnyebbé válik a vi kezelése. Egy parancsot mindenképpen tanuljunk meg, ez a :q! (a kettőspont és a felkiáltójel itt a parancs része).

Mozgás a szövegben

```
Ctrl u, Ctrl d
```

A kurzort a szövegben felfelé (up) vagy lefelé (down) viszi 12 sorral.

```
Ctrl f, Ctrl b
```

A kurzort a szövegben előre (forward) vagy hátra (back) viszi 12 sorral.

```
e
```

A kurzort a következő szó végére viszi.

```
b
```

A kurzort az előző szó elejére viszi.

```
Ctrl g
```

Kiírja a kurzort tartalmazó sor sorszámát.

```
G
```

A szerkesztett fájl végére ugrik.

```
n G
```

A fájl n-dik sorára ugrik.

```
$
```

Ugrás az aktuális sor végére.

```
/keresett_szó
```

A keresett_szó első előfordulási helyére ugrik.

```
n
```

A keresett_szó következő előfordulási helyére ugrik.

Szöveg beszúrás

```
i
```

A begépelt szöveget beszúrja (insert) a kurzor pozíciójához. Ha befejeztük a műveletet, üssük le az ESC billentyűt.

```
a
```

A begépelt szöveget a kurzor pozíciójától kezdve hozzáfűzi (append) a szerkesztett dokumentumhoz. Ha befejeztük a műveletet, üssük le az ESC billentyűt.

```
o
```

Egy üres sort hoz létre az aktuális sor alatt. Gépeljük be a kívánt szöveget, majd üssük le az ESC billentyűt.

```
O
```

Egy üres sort hoz létre az aktuális sor felett. Gépeljük be a kívánt szöveget, majd üssük le az ESC billentyűt.

Szöveg módosítása

```
x
```

Kitörli a kurzor pozícióján álló karaktert.

```
rc
```

Kicseréli (replace) a kurzor pozícióján álló karaktert c-re.

```
R
```

Felülírásos üzemmódba kapcsol. Üssük le az ESC billentyűt ha vissza akarunk térni parancs módba.

```
dd
```

Kitörli az aktuális sort.

```
J
```

A következő sort összefűzi az aktuális sorral.

```
C
```

Kijelöli az aktuális sor kurzortól jobbra eső részét (egy $ jelet rak a sor végére.) Amit ezután begépelünk, az felülírja a megjelölt részt.

```
u
```

Visszacsinálja az utolsó műveletet

```
U
```

Visszacsinálja az eredeti állapotot az aktuális sorban.

```
:i,j m p
```

Az i. és j. sorok közé eső részt (beleértve magát az i. és j. sort is) áthelyezi a p. sor után.

Kilépés a vi-ból

```
:w
```

Elmenti a szerkesztett szöveget.

```
:w név
```

„név” néven elmenti a szerkesztett szöveget.

```
:i,j w név
```

„név” néven elmenti az i. és j. sorok közé eső részt (beleértve magát az i. és j. sort is)

```
:q
```

Kilép, ha nem változtattunk a szövegen az utolsó mentés óta.

```
:q!
```

Mindenképpen kilép, az esetleges változtatások elmentése nélkül.

```
:wq
```

Elmenti a szerkesztett szöveget és kilép.

Egyéb

```
.
```

Megismétli az utolsó parancsot.

Feladatok

- Készítsünk magunknak „puskát” a vi legfontosabb parancsairól! Ehhez először mentsük ki e lecke szövegét egy szövegfájlba. Ezt minden levelezőprogrammal meg lehet tenni, általában az export parancs segítségével. Ha sehogyan sem boldogulunk, akkor használjuk a Unix mail programját. (Indítsuk el a mail programot, válasszuk ki ezt a levelet a sorszámának begépelésével, majd az s vi-doc paranccsal mentsük el a szöveget egy vi-doc nevű fájlba.
- Az így létrehozott szövegfájlt töltsük be a vi-ba! Töröljük ki a parancsok leírása előtti bevezető szöveget, és a feladatok leírását. Mentsük el a változtatásokat! Most lapozzunk vissza a fájl elejére, és szúrjunk be egy-egy üres sort minden parancs leírása elé!
- Az elkészült „puskát” ki is nyomtathatjuk, feltéve, hogy van nyomtató a rendszerben. A nyomtatási lehetőségekről és a nyomtató használatáról (hogy hívják a PRINT parancsot ezen a héten?), olvassuk el a dokumentációt, vagy kérdezzük meg a rendszeradminisztrátort!
- Nézzük meg, hogy az eddigi munkánkkal mennyi helyet foglalunk!

Hatodik lecke

„Az igazi Unix buherátor művész. Mint minden művész, ő is önön lényegét alkotja újjá az anyagban - ami jelen esetben a shell, annak változói és környezete.”

(Csáky István)

Bővebben az átirányításról

A harmadik leckében volt szó az átirányításról. Ebben a leckében pár hasznos példát mutatunk az átirányítás „haladóbb” alkalmazására. Bonyolultabb átirányításokhoz az sh (vagy sh típusú shell) alkalmazása javasolt, ezért a példák is az sh alkalmazásával készültek. (Akinek csh az alapértelmezés, az a Bourne shellt az sh beírásával indíthatja!)

Mit csináljunk például akkor, ha az stderr-t (a hibacsatornát) szeretnénk egy fájlba irányítani? Ehhez először is tudnunk kell, hogy az alapértelmezés szerinti csatornáknak (stdin, stdout, stderr), egy-egy szám a megfelelőjük; az stdin-é a 0, az stdout-é az 1 és az stderr-é a 2. Ezeknek a számoknak a használatával lehet bonyolultabb átirányítási feladatok megoldására kényszeríteni a Unixot. Például, ha a cat parancs hibajelzéseit át szeretnénk irányítani egy caterr nevű fájlba, a következőket kell tennünk:

```
sindbad> sh
sindbad% cat level 2>caterr
sindbad% cat caterr
cat: cannot open level
sindbad%
```

Mivel a level nevű fájl nem létezik, a cat program hibajelzést ad, amit mi a caterr nevű fájlba irányítottunk át.

Másik nagyon gyakran előforduló probléma, hogy egy program kimenetét és hibáit szeretnénk egy fájlban látni, de mivel a Unix a hibákat a képernyőre írja ki - abban az esetben is, ha a program a háttérben fut - átirányításra van szükség.

```
sindbad> sh
sindbad% program 1>errors 2>&1
sindbad%
```

A & jel azt jelenti, hogy nem egy fájlba irányítunk át, hanem egy másik csatornába, jelen esetben az stdout-ra.

A shell változók és a környezet

A shell részben programozási nyelv is, ezért tartalmaz változókat.

Figyelem! Az sh-ban (sh, ksh, bash, zsh) és a csh-ban (csh, tcsh) másként kell értéket adni a változóknak!

Az sh típusú shellek az „=” jelet használják:

```
sindbad% a="hello sh"
```

míg a csh típusúakban a set paranccsal kell értéket adni a változóknak:

```
woland> set a="hello csh"
```

A kiíratást mindkét shell esetében az echo parancs végzi, a változó neve előtt pedig egy $ jelnek kell szerepelnie:

```
sindbad% echo $a
hello sh
sindbad%
```

Ezek a shell belső változói - tehát csak az aktuális shellben ismertek, lokálisak. A shell lehetővé teszi a futtatandó programok környezetének (environment) beállítását is. A környezet azon változók csoportja, melyek minden programból elérhetőek, globálisak. A környezet beállításának szempontjából is különbözik az sh és a csh. A környezet beállítása sh-ban:

Állítsunk be valami hasznosat. A PAGER környezeti változó, amelyet például a man parancs használ megjelenítőprogramként, éppen megfelel erre a célra.

```
sindbad% PAGER="cat"
sindbad% export PAGER
```

Ennek kicsit kellemetlen az eredménye, mivel ha man-t kérünk, csak elrohan a szöveg a szemünk előtt, de példának megteszi! Azonban, ha képernyőnként szeretnénk látni, akkor állítsuk be a more-t nézegető programnak:

```
sindbad% PAGER="more"
sindbad% export PAGER
```

A környezeti változók beállításának csh-beli megfelelője a setenv parancs. A fenti példa csh-s megfelelője tehát:

```
woland> setenv PAGER cat
```

és

```
woland> setenv PAGER more
```

Amennyiben egy lokális vagy környezeti változóra nincs szükségünk, eltüntethetjük őket, mintha sosem lettek volna. Sh-ban unset változónév, csh-ban a lokális változókra unset változónév, a globálisokra pedig unsetenv változónév. Azt hogy milyen változók vannak beállítva úgy tudhatjuk meg, hogy a set és a setenv parancsok után nem írunk változónevet.

A PATH környezeti változó

Kiemelten kezeli a shell a PATH (csh-ban path) nevű változót, ugyanis ez a változó jelzi a shell számára, hogy hol kell keresnie a futtatandó programot. Például:

```
sindbad% echo $PATH
/bin:/usr/bin:/usr/local/bin:.
sindbad%
```

A fenti példa az sh-t mutatja, mindez csh-ban így néz ki:

```
woland> echo $path
/bin /usr/bin /usr/local/bin .
woland>
```

A különbség világosan látható.

Tehát a shell először megnézi, hogy a futtatandó program a /bin-ben, a /usr/bin-ben, a /usr/local/bin-ben végül pedig az aktuális könyvtárban (a „.” ezt jelzi) található-e?

Automatikus indítású shell scriptek

Minden shell képes arra, hogy indulásakor egy adott nevű shell scriptet elindítson, így a felhasználóknak lehetőségük van arra, hogy környezetüket személyiségükhöz igazítsák.

Ez a program az sh-nál a munkakönyvtárban (home dir) lévő „.profile”, csh esetében pedig a bejelentkezés esetén a „.login” és minden más csh indításnál a munkakönyvtárban lévő „.cshrc”. Ki-ki ebbe teheti az általa kedvelt változó- és környezet-beállításokat.

Egy minta .cshrc fájl

Lássuk miket lehet beállítani egy .cshrc fájlban.

```
woland> cat .cshrc
set history=100
alias h history
alias ls    ls -F
alias ll    'ls -l | more'
set prompt="`hostname` > "
woland>
```

Lássuk mit csinál ez a kis „program”. A set history arra szolgál, hogy a begépelt parancsainkat a rendszer megőrizze, hogy ha már egyszer begépeltünk valamit, ne kelljen ismét begépelni. A 100-as szám azt jelzi, hogy 100 parancsot őrizzen meg.

A h parancs most a history parancs ekvivalense - a history parancs az eddig begépelt utasításokat listázza ki.

A következő sorban az ls az ls -F-nek lesz az ekvivalense. Ezután minden ls parancs automatikusan ls -F lesz. (Nézzük meg a man-ban, hogy mit csinál az ls F kapcsolója).

Az ezutáni sor még érdekesebb, mivel arra példa, hogy egy utasítássorozatnak legyen egy rövid neve. Tehát egy bő lista kimenetét átadjuk a more nézegető programnak, hogy oldalanként lássuk a kimenetet.

Ezután a prompt beállítására láthatunk példát. A gép először lefuttatja a hostname programot, ami kiírja a gép nevét és azt berakja a promptba. A ` (visszafelé mutató egyes idézőjel) jelentése, hogy a program futásának eredményét helyettesítse be a hostname szó helyett.

Az alias parancs arra szolgál, hogy egy hosszabb utasítássorozatot egy parancs begépelésével indíthassunk vagy hosszú parancsnév esetén a program nevét Unixabbra alakítsuk (minél rövidebb annál jobb). A jelenleg definiált aliasokat a paraméter nélkül hívott alias parancs mutatja meg; egy- egy alias-t pedig az unalias paranccsal tudunk megszüntetni.

Terminál beállítások

Az egyik leghasznosabb dolog, amit az automatikusan elinduló .cshrc vagy .profile fájllal csinálhatunk, a terminálunk beállítása.

Bizonyára mindenki találkozott már azzal a kellemetlen jelenséggel, hogy egy terminálról való bejelentkezés után bizonyos billentyűk váratlan dolgokat művelnek. Ilyen például, hogy a visszatörlés (backspace) az utolsó karakter törlése helyett ^?-t ír ki, vagy törli a teljes sort.

A jelenség magyarázata a terminálok működésében keresendő. Amikor leütünk egy billentyűt, a terminálunk elküld egy kódot a számítógépnek. A Unix értelmezi ezt a kódot, és csinál valamit. Baj csak akkor van, ha a Unixunk rosszul értelmezi az általunk küldött kódokat: így keletkeznek a ^H, ^?, jelek és egyéb mellékhatások.

A terminál beállításait az stty paranccsal listázhatjuk ki. A -a kapcsolóval teljes listát kapunk. (Némely Unixnál ezt a kapcsolót másképp hívják - ilyenkor nézzük meg a manualt!)

```
zeus% stty -a
speed 9600 baud; line = 1;
intr = ^C; quit = ^\; erase = DEL; kill = ^U; eof = ^D; eol
<undef>; swtch = ^Z lnext = ^V; werase = ^W; rprnt = ^R; flush
= ^O; stop = ^S; start = ^Q -parenb -parodd cs8 -cstopb hupcl
cread clocal -loblk -tostop -ignbrk brkint ignpar -parmrk
-inpck istrip -inlcr -igncr icrnl -iuclc
ixon -ixany -ixoff
isig icanon -xcase echo echoe echok -echonl -noflsh
opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel tab3
zeus%
```

A listán funkciókat látunk (intr, quit, erase...) és az őket aktiváló kódokat (^C, ^\, DEL, ...) Ezek közül a legfontosabb az erase. Ez a funkció törli a kurzortól balra álló karaktert. A fenti példán értéke DEL, azaz a DEL billentyű leütésével lehet végrehajtani.

Ha a terminálunkon lévő DEL feliratú gomb nem a megfelelő kódot küldi, akkor nem a kívánt művelet történik. Ezt úgy küszöbölhetjük ki, hogy az erase funkcióra egy másik kódot definiálunk, például a ^H-t. Ezután a <Ctrl-h>-val lehet majd törölni. Másik (kényelmesebb megoldás, ha az erase funkciót ahhoz a kódhoz rendeljük hozzá, amit a DEL feliratú billentyűnk generál. Az erase funkció átdefiniálása ugyancsak az stty paranccsal történik. Tételezzük fel, hogy a backspace billentyűvel szeretnénk törölni, de ha leütjük ezt a billentyűt, akkor ^? jelenik meg a képernyőn. Gépeljük be, hogy:

```
stty erase "^?"
```

Ezután nyugodtan törölhetünk a backspace billentyűvel is. A többi funkció beállítása hasonlóan működik.

További fontos funkciók:

```
intr
```

Megszakítja a futó programot. Általában a ^C-vel lehet végrehajtani.

```
kill
```

Kitörli a parancssorba eddig begépelt szöveget (ha még nem ütöttük le az Entert.)

```
eof
```

Ez a jól ismert fájlvége jel. Ez szinte mindenhol <Ctrl-d>.

```
stop
```

Felfüggeszti a képernyőre írást.

```
start
```

Folytatja a stoppal befagyasztott képernyőre írást.

```
swtch
```

Az aktuális program futásának felfüggesztése (lásd harmadik lecke).

Természetesen nagyon fáradságos, ha minden paraméter legkedvezőbb beállítását nekünk kell kikísérletezni. Ezért definiáltak néhány beállítás-készletet, amelyekre a nevükkel lehet hivatkozni. Ilyen például a dec terminál; gyakran segít, ha kiadjuk az stty dec parancsot.

Feladatok

- Listázzuk ki a .profile vagy .cshrc fájlunk tartalmát! Valószínűleg találunk benne néhány alias utasítást, amelyeket a rendszeradminisztrátor helyezett el számunkra. Próbáljuk meg kitalálni, hogy melyik mire való!
- Találjuk ki, hogy mit csinálnak az alábbi alias definíciók!

  A „\!*” jel a paraméter helyettesítést jelenti, azaz mindent amit a cd után beírunk. Próbáljuk ki, és ha tetszik, építsük be a .cshrc vagy .profile fájlunkba!

```
woland> cat .cshrc
alias setprompt 'set prompt="$cwd > "
alias cd "chdir \!*; setprompt"
setprompt
woland>
```

  Az sh-ban nincsenek aliasok, helyettesítésük függvényekkel lehetséges. A fenti példa sh-ban:

```
sindbad% cat .profile
h()   {history}
ls()  {ls -F}
ll()  {ls -l | more}
sindbad%
```

Hetedik lecke

Az Internet hálózat felépítése

A világ számítógépeit összekötő hálózatok vezetékei keresztül-kasul átszelik az egész földgolyót. Több világméretű hálózat létezik, mi a továbbiakban az Internet hálózattal fogunk foglalkozni, mivel az oktatási intézményekben ez a legelterjedtebb.

Minden hálózatba kötött gépnek van egy neve; ez az ún. node name, például: woland.iit.uni-miskolc.hu. A névből általában következtetni lehet a gépet működtető intézetre és annak helyére (iit = Institute of Information Technology, uni-miskolc = Miskolci Egyetem, hu = Magyarország). Minden gépnek van egy telefonszám-szerű száma is, amit IP számnak hívunk. Ez valahogy így néz ki: 193.6.4.30 (a pontokkal elválasztott számok nagyjából megfelelnek a node név egyes elemeinek.) A hálózat bármelyik gépére be tudunk jelentkezni „fel tudjuk hívni" a telnet parancs segítségével (dolgozni természetesen csak akkor tudunk rajta, ha van username-ünk és érvényes jelszavunk.) A hívás fizikai szinten az IP szám segítségével történik. Az egyszerűség kedvéért - és hogy ne kelljen megerőltetni a szám-memóriánkat - a telnet parancs node név alapján is fel tud hívni egy gépet. Ilyenkor előbb kikeresi a „telefonkönyvből" a hívott gép IP számát; ez a telefonkönyv az ún. „domain name server", rövidítve DNS.

A gépek közötti kapcsolat a TCP/IP protokoll segítségével történik. Erről annyit kell tudnunk, hogy az adatokat nem egyszerre küldik el, hanem kis csomagokra (packet) bontják, majd ezeket egymás után elindítják a célállomás felé. Ha egy csomag útközben elvész, akkor elég csak a hiányzó csomagot újra küldeni. Ezért szokták az Internetet „csomagkapcsolt hálózat" (packet switching network) néven is emlegetni.

Az Interneten belül a legkülönfélébb szolgáltatások érhetők el, ezek közül a három legfontosabb az elektronikus levelezés, a fájl átvitel és az online kapcsolat (talk). Feltételezzük, hogy az elektronikus levelezést már mindenki ismeri, a fájl átvitelről és a talk-ról pedig e fejezet későbbi részeiben lesz szó.

Információk a hálózat gépeiről és felhasználóiról

Valamennyi parancs közül talán a finger adja a legtöbb információt. Paraméterek nélkül kiadva megmutatja hogy egy hoston rajtunk kívül ki van még bejelentkezve. Bármelyik hostot le lehet kérdezni a következő módon (a host név helyett írhatunk IP számot is):

```
finger @host_név
```

Ha valakinek nem ismerjük az username-jét, de tudjuk a nevét, a finger segítségével ki tudjuk deríteni. Írjuk be például:

```
finger Laszlo
```

Erre valami ilyen választ kapunk (legalább is Miskolcon):

```
Login name: kovacs          In real life: Kovacs Laszlo
Office: IIT
Directory: /usr/people/users/kovacs   Shell: /bin/tcsh
Never logged in.
No Plan.
```

```
Login name: cser            In real life: Cser Laszlo
Office: IIT
Directory: /usr/people/users/cser    Shell: /bin/tcsh
Never logged in.
No Plan.
Login name: szakal           In real life: Szakal Laszlo
Office: IIT, x2106
Directory: /usr/people/users/szakal   Shell: /bin/tcsh
Last login at Tue Aug 31 10:05 on /dev/tty
Plan:
Don't marry, be happy!
```

Amint látjuk, egészen sok információt kaptunk. A Login name: után szerepel az illető bejelentkezési neve (username). Az In real life: a felhasználó valódi nevét adja meg, az Office: a tanszéket, ahol dolgozik és a telefonszámot. Megtudjuk azt is, hogy hol található a felhasználó munkakönyvtára (Directory:), milyen parancsértelmezőt (Shell:) használ, mikor és melyik terminálról volt utoljára bejelentkezve (Last login...). Ha itt a „Never logged in" üzenet szerepel, az illető még soha nem volt bejelentkezve. Ha azt látjuk hogy „On since..." akkor a felhasználó jelenleg is a rendszerben tartózkodik.

Megjegyzés: A Last login... azt mondja meg, hogy a felhasználó az adott hoston mikor járt utoljára. Ha több host van összekapcsolva úgy, hogy a jelszó mindegyikre érvényes, akkor előfordulhat, hogy a felhasználó egy későbbi időpontban egy másik hostra is bejelentkezett. Ezt csak úgy tudjuk ellenőrizni ha sorban valamennyi hostot végignézzük. A levelekről adott információkra ez nem vonatkozik, mivel az összekapcsolt hostokon egy usernek csak egyetlen postafiókja van.

Bizonyos rendszerek (például a SUN) azt is elárulják, hogy a felhasználó mikor kapott utoljára levelet és mikor olvasta utoljára a leveleit. No unread mail - nincs elolvasatlan levél, New mail received at...- új levél érkezett... Mail last read at...- az utolsó levélolvasás időpontja.

A finger egy további bájos tulajdonsággal is rendelkezik: ha a kiindulási könyvtárunkban létrehozunk egy .plan és egy .project fájlt, akkor a finger a Plan: és Project: címszavak után ezek tartalmát is megjeleníti. Ha valamit közölni szeretnénk a rólunk érdeklődő felhasználókkal, azt e két fájl valamelyikébe kell beírnunk. (A .plan fájl egy életszagú alkalmazását láthatjuk az előző példán.)

A fingerhez hasonló a who parancs; ez valamivel szűkszavúbb:

```
varga   ttyq0 Sep 3 21:32
szabo   ttyq1 Sep 6 09:49
kovacs  ttyq3 Sep 6 09:32
lengyel ttyq5 Sep 6 11:04
```

A w parancs viszont azt is megmutatja, hogy a felhasználók éppen min dolgoznak:

```
User    tty  from    what
varga   q0   9:51am  -tcsh
kovacs  q2   9:17am  pine
lengyel q3   9:27am  xwsh -name winterm -na
```

A listából megállapítható, hogy a varga username-ű felhasználó a q0 nevű terminálról jelentkezett be tíz óra előtt kilenc perccel és jelenleg a parancsértelmező programot futtatja (command shell), azaz a monitorán a UNIX prompt látszik (%). kovacs a pine levelező-programon dolgozik, lengyel pedig a winterm nevű programot futtatja, ami nem más mint egy grafikus munkaállomáson használható shell ablak. (A fenti lista némileg egyszerűsített, a valóságos néhány további adatot is tartalmaz.)

Az rusers nem csak a mi hostunkra bejelentkezett felhasználó username-jét mutatja meg, hanem a hálózat azonos alegységén (szegmensén) lévő valamennyi gépet végignézi:

```
ind03.iit.uni-miskolc.hu cser
ind02.iit.uni-miskolc.hu toth toth toth
indvd.iit.uni-miskolc.hu wagner szakal vadasz
ind04.iit.uni-miskolc.hu stsmork
zeus.iit.uni-miskolc.hu  vadasz
indkl.iit.uni-miskolc.hu kovacs
```

Megjegyzés: A parancs természetesen nem a világ összes gépét nézi végig, hanem csak az azonos szegmensben lévőket. Általában 5-20 gép van egy szegmensben; ennél több gép esetén a művelet akár percekig is eltarthat! (Ilyenkor a <Ctrl-c> leütésével tudjuk megállítani a parancsot.)

Amint látjuk, egy hostra több felhasználó is bejelentkezhet egy időben. Sőt, ugyanaz a felhasználó bejelentkezhet ugyanarra a hostra több terminálról is.

Bármelyik hostról át tudunk jelentkezni bármely másikra, az rlogin paranccsal:

```
rlogin host_név
```

A rendszer ekkor ismét megkérdezi a jelszónkat, majd átjelentkezik a megadott hostra, amit a megváltozott promptból is láthatunk.

A last parancs megadja az utolsó néhány száz bejelentkezés tényét. Ha csak egy felhasználó adataira vagyunk kíváncsiak, akkor azt a következő módon tudhatjuk meg (tegyük fel, hogy a keresett felhasználó username-je winnie):

```
last winnie
```

Bejelentkezés egy távoli gépre

Erre szolgál a már említett telnet parancs. A telnettel történő bejelentkezést hívják „interaktív login”-nak - erre utal a bizonyos rendszereken belépéskor megjelenő „last interactive login” dátum és időpont. Ha például a piglet.forest.edu gépet akarjuk felhívni, akkor adjuk ki a

```
telnet piglet.forest.edu
```

Ha a parancs hibát jelez, akkor valószínűleg nem találja a telefonkönyvben a hívott gép IP számát. Ha tudjuk fejből az IP számot, azt is megadhatjuk a node név helyett. Ha minden jól megy, a vonal terheltségétől függően hosszabb-rövidebb várakozás után valami ilyesmi jelenik meg:

```
Trying piglet.forest.edu...
Connected to 19.54.285.892.
Escape character is '^]'.
login:
```

Megjegyzés: A piglet.forest.edu nevű gép a valóságban nem létezik, ezért senki ne próbálja felhívni. A lecke végén találunk néhány igazi nevet és IP számot, amelyeken nyugodtan gyakorolhatunk.

Az Escape character is '^]'. annyit tesz, hogy a kapcsolatot bármikor megszakíthatjuk a Ctrl és] billentyűk együttes lenyomásával. Ilyenkor nem a Unix prompthoz jutunk vissza, hanem a telnet parancs promtjához:

```
telnet>
```

Ha újra be akarunk kapcsolódni, akkor írjuk be a connect parancsot, egyébként pedig a quit parancsot. A további lehetőségekről a ? parancs ad rövid felvilágosítást.

Fájlok átvitele két gép között

Ha az ftp parancs segítségével jelentkezünk be egy távoli (remote) gépről, akkor fájlokat tudunk mozgatni a távoli és a helyi (local) gép között. Jegyezzük meg, hogy mindig az a helyi (local) gép, amelyiken elindítottuk az ftp programot, és az a távoli (remote), amelyikre bejelentkeztünk. Az ftp bejelentkezést hívják nem interaktív loginnak (non interactive login). A legtöbb gépre be lehet jelentkezni nem interaktívan, feltéve hogy az adott gépre van jelszavunk. Egyes gépeken van egy különleges username, az anonymous nevű, amelynek nincs jelszava és így bárki bejelentkezhet rá. (Jelszó helyett felkér minket, hogy adjuk meg az e-mail címünket: ezt illik megadni, de bármit beírhatunk.) Ha bejutottunk a gépre, a get és put parancsok segítségével fájlokat hozhatunk el, illetve vihetünk fel a távoli gépre.

Tegyük fel, hogy az előbb már említett (fiktív) piglet.forest.edu gépen van anonymous ftp szolgáltatás. A bejelentkezés a következőképpen történik:

```
orlando% ftp piglet.forest.edu
Connected to piglet.forest.edu.
220 cica FTP server (Version wu-2.1c(1) Mon Oct 18 10:56:22 EST 1993)
ready.
Name (piglet.forest.edu:stsmork): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
230-**
230-** You have reached piglet.forest.edu [123.789.290.287] at
230-** the Center for Innovative Computer Applications at
230-**
. .
. .
. .
```

Itt további információk olvashatók, majd megjelenik az ftp program promptja:

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

Az itt kiadható legeslegfontosabb parancsok:

```
binary
```

Bináris átvitelt állít be (CR/LF konverzió kikapcsolva).

```
ascii
```

ASCII text átvitelt állít be (CR/LF konverzió bekapcsolva).

```
hash
```

Az átvitel során minden átvitt egy kilobyte után kiír egy # jelet.

```
put fájlnév
```

Egy fájlt átmásol a lokális gépről a távoli gépre.

```
get fájlnév
```

Egy fájt átmásol a távoli gépről a lokális gépre.

```
bye
```

Kilép az ftp programból.

További nagyon fontos parancsok:

```
cd
```

Alkönyvtárat vált a távoli gépen.

```
lcd
```

Alkönyvtárat vált a lokális gépen.

```
ls
```

Kilistázza az aktuális alkönyvtárat a távoli gépen.

```
pwd
```

Kiírja a távoli gép aktuális alkönyvtárát.

```
?
```

Kilistázza az ftp program parancsait.

```
help
```

Rövid ismertetést ad a parancs működéséről.

Most belépünk a pub alkönyvtárba, beállítjuk az átvitel módját binárisra, bekapcsoljuk az átvitel jelző kereszteket, elhozunk egy fájlt, végül pedig kilépünk az ftp programból.

```
ftp> cd pub
250 CWD command successful.
ftp> binary
200 Type set to I.
ftp> hash
Hash mark printing on (1024 bytes/hash mark).
ftp> get
(remote-file) Where.to.find.Winnie
(local-file) winnie.txt
```

```
local: where.txt remote: Where.to.find.Winnie
200 PORT command successful.
150 Opening BINARY mode data connection for Where.to.find.Winnie
(6742 bytes)

226 Transfer complete.
6742 bytes received in 25.82 seconds (0.25 Kbytes/s)
ftp> bye
221 Goodbye.
orlando%
```

Az ftp-vel tehát bármely két gép között lehet fájlokat mozgatni, ha rendelkezünk a megfelelő jogosultságokkal (jelszavakkal). Az ftp egy különleges alkalmazásaként nyilvános adatbázisokból jelszó nélkül lehet freeware programokat letölteni.

Figyelem! A fájltranszfer művelet nagy terhelést jelent a hálózatnak, ezért a nyilvános adatbázisok felderítését és programok letöltését lehetőleg este vagy éjszaka végezzük!

Valós idejű kapcsolat két felhasználó között

Bármelyik bejelentkezve lévő felhasználó képernyőjére üzenetet tudunk küldeni a write parancs segítségével.

```
write username@host_név
```

Ehhez nyilván tudnunk kell az illető felhasználó username-jét és azt hogy éppen melyik hoston van bejelentkezve. Ha ugyanazon a hoston van mint mi, akkor a @ jel és az utána megadott host_név elhagyható.

Miután kiadtuk a parancsot, nem történik semmi. (Mint tudjuk, ez a Unixnál azt jelenti, hogy minden rendben.)

Gépeljük be az üzenetet, a sorok végén üssük le az Enter-t, majd - ha befejeztük az üzenetet - üssük le a <Ctrl-d> billentyűkombinációt. A begépelt szöveg erre megjelenik a megcímzett felhasználó terminálján képernyőjén, függetlenül attól hogy ott éppen mi van. Ha tehát a címzett éppen dolgozik valamin, akkor a képernyőjének a közepén egyszer csak megjelenik az üzenetünk (kivéve ha grafikus felhasználói felülettel dolgozik, ilyenkor egy ablakban jelenik meg a szöveg).

A másik felhasználó hasonló módon válaszolhat, de ha hosszabb beszélgetésbe akarunk kezdeni valakivel, akkor jobb, ha a talk parancsot használjuk. Ez oly módon köti össze a két gépet, hogy amit az egyik felhasználó begépel a billentyűzetén, az rögtön megjelenik a másik felhasználó monitorán. Így „beszélgetni” lehet, bár a párbeszéd sebességének határt szab a társalgó felek gépírás tudása. A parancsot így kell kiadnunk:

```
talk hívott_fél_username-je@hívott_fél_hostjának_neve
```

Ha beszélgetőpartnerünk velünk egy hoston van bejelentkezve, akkor a @ és a host_név elhagyható. A parancs kiadása után azonnal letörlődik a képernyő. Középen megjelenik egy szaggatott vonal, a felső sorban pedig egy üzenet ami arról tájékoztat hogy a rendszer próbálja felépíteni a kapcsolatot. A hívott fél képernyőjén ekkor megjelenik egy üzenet, hogy valaki talk-olni (beszélgetni) akar vele. A kapcsolat létrejöttéhez a hívott félnek is ki kell adnia a talk parancsot, a következő módon:

```
talk hívó_fél_username-je@hívó_fél_hostjának_neve
```

Ha mindketten begépelték a helyes parancsokat, a képernyő felső sorában megjelenik a [Connection established] üzenet. Ez azt jelenti, hogy a kapcsolat felépült, kezdhetünk „beszélgetni”. Amit mi írunk be, az a képernyő felső felén jelenik meg, a szaggatott vonal felett; partnerünk szövege pedig a vonal alatt. Ha be akarjuk fejezni a beszélgetést, üssük le a <Ctrl-c> billentyűkombinációt.

Megjegyzés: Ha valamilyen okból kifolyólag azt szeretnénk, hogy mások ne tudjanak velünk talkolni, vagy a write paranccsal üzenetet küldeni a képernyőnkre, akkor a mesg paranccsal letilthatjuk az üzenetek fogadását.

```
orlando% mesg n
```

E parancs letiltja az üzenetek fogadását, a mesg y pedig újra lehetővé teszi.

Egyéb hálózati segédprogramok

A Unix számos további hálózattal kapcsolatos segédeszközt is biztosít. Ilyen például a ping program (az /usr/etc vagy az /etc alkönyvtárban található) amely két számítógép közötti adatátvitel sebességét méri. (Csomagokat küld a másik gépnek és méri, hogy a csomagok mennyi idő alatt teszik meg az oda-vissza utat, valamint számolja, hogy hány csomag vész el.)

Feladatok

- A last és a tail segítségével írassuk ki a képernyőre két legrégebbi bejelentkezésünkről tárolt adatokat! (Használjuk a Unix operációs rendszer „cső” szolgáltatását!)
- A telnet program segítségével hívjuk fel az alábbi gépek valamelyikét, attól függően, hogy földrajzilag melyikhez vagyunk legközelebb! (Ha a telnet parancsunk név alapján nem ismeri fel a gépet, akkor próbálkozzunk az IP számmal.)

  mars.sztaki.hu 192.84.225.92 Magyarország
  consultant.micro.umn.edu 134.84.132.4 Észak-Amerika
  gopher.uiuc.edu 128.174.33.160 Észak-Amerika
  gopher.sunet.se 192.36.125.2 Észak-Európa
  gopher.chalmers.se 129.16.221.40 Svédország
  tolten.puc.cl 146.155.1.16 Dél-Amerika
  ecnet.ec 157.100.45.2 Ecuador
- Jelentkezzünk be gopher néven, ekkor a gép nem fog jelszót kérni tőlünk. Miután bejutottunk, egy világméretű információs rendszerben találjuk magunkat, ahol számtalan különböző témáról olvashatunk cikkeket. Az anyagok hierarchikusan vannak tárolva, a menükből almenük nyílnak, amelyekből újabb almenük nyílnak és így tovább. A menüpontok között a kurzormozgató nyilakkal mozoghatunk, belépni egy menübe, vagy megnézni egy dokumentumot pedig az Enter leütésével lehet.
- Lépjünk be az ftp.cica.indiana.edu nyilvános adatbázisba és nézzünk körül az alkönyvtárakban, hátha találunk minket érdeklő programot! Ha Magyarországról próbálkozunk, akkor a novell.aszi.sztaki.hu gépet hívjuk az előbbi helyett!!! (Megjegyzés: minden alkönyvtárban van egy index nevű fájl, ez tartalmazza az adott alkönyvtárban lévő fájlok neveit és egy-egy mondatban a feladatukat.)

- Nézzük meg, hogy ki van bejelentkezve a gépen; ha látunk valakit, akit személyesen is ismerünk, próbáljunk meg kapcsolatba lépni vele a talk parancs segítségével!
- Készítsünk .plan vagy .project fájlt magunknak! (Lehetőleg a vi editorral, hogy egy kicsit azt is gyakoroljuk.)

Nyolcadik lecke

Keresés szövegfájlokban

A legeslegfontosabb szűrő a grep, a Unix parancsok népszerűségi listáján mindjárt az ls után következik. A grep arra való, hogy segítségével megadott feltételeknek eleget tevő szavakat tartalmazó sorokat keressünk a szövegfájlokban. A szövegfájlokról annyit kell tudnunk, hogy sorokból állnak, a sorok pedig szavakból vagy más néven mezőkből. A sorok végén sorvége jel van, a szavakat (mezőket) pedig szóközök választják el egymástól.

A grep kimenetén kiírja az összes „találatot” - vagyis azokat a sorokat, amelyek tartalmaznak legalább egy, a feltételt kielégítő szót.

A szintaxis nagyon egyszerű, meg kell adni, hogy mit hol keressen:

```
grep mit_keressen hol_keresse
```

Például: barátaink és üzletfeleink nevet és e-mail címét egy .addressbook nevű szöveges fájlban tartjuk, és szeretnénk megnézni Kovacs nevű barátunk (vagy üzletfelünk) e-mail címét. A megoldás: „kigrepeljük” a szövegfájlból azokat a sorokat, amelyekben előfordul a „Kovacs” név.

```
orlando% grep Kovacs .addressbook
Kovacs Istvan pistike@badguys.hell.gov
Kovacsuzem h06789@kalapacs.uzem.com
orlando%
```

A nagy kezdőbetűs Kovacs - a Unixtól megszokott módon - nem azonos a nemecsekernős kovacs névvel. Ha azt szeretnénk, hogy a grep a keresés során ne különböztesse meg a kis- és nagybetűket, akkor a -i kapcsolót kell használnunk. A kapcsolókat a parancssorban az első argumentum (keresési minta) előtt kell megadni. A fenti példában a grep két sort írt ki, hiszen a „Kovacs” szó a „Kovacsuzem”-nek is része. Ha ezt el akarjuk kerülni, mert azt akarjuk, hogy a grep csak a teljes szavakat találja meg, akkor használjuk előszeretettel a -x kapcsolót.

Két további hasznos kapcsoló: az egyik (-n) hatására a grep a megtalált sorok elé kiírja a sorszámot is, a másikkal (-v) pedig fordított működésre lehet kapcsolni. Ilyenkor azokat a sorokat írja ki, amelyek NEM tartalmazzák a megadott mintát, a többit pedig lenyeli.

```
orlando% w | grep -v gyevi_biro
```

A fenti példa kilistázza az összes bejelentkezett felhasználót (w parancs), kivéve a gyevi_biro-t. A példán az is látszik, hogyan használhatjuk a grepet egy másik parancs kimenetének megszűrésére.

A hol_keresse mezőben használhatjuk a *, ?, stb. karaktereket, azaz a grep több fájlban is kutathat a feltételnek megfelelő sorok után. A félreértések elkerülése végett a sorok elé kiírja azt is, hogy melyik fájlban találta őket.

(Vigyázat! A grep nem rekurzív, tehát nem nézi meg azokat a fájlokat, amelyek a megadott könyvtárból nyíló alkönyvtárakban vannak.)

Reguláris kifejezések

Jó, jó, de hogy kell leírni azokat a bizonyos „megadott feltételek”-et, amelyek alapján a grep a keresést végzi? A Unix tervezői erre a célra alkották meg a reguláris kifejezéseket (regular expressions).

A dolog nagyon hasonló a * és ? karaktereket tartalmazó fájlnevekhez. A reguláris kifejezés egy olyan különleges karaktersorozat, amit a grep (és számos más) parancs mintaként értelmez. Ha egy szó megfelel a mintának, azt mondjuk rá, hogy „illeszkedik a reguláris kifejezésre”.

A legegyszerűbb eset, mikor a reguláris kifejezés nem tartalmaz semmilyen speciális karaktert. Az ilyen kifejezés csak önmagára illeszkedik. Vegyük példaként Csocsi és Vonyi egyik hal(l)hatatlan kétsorosát:

Romeo Julian topreng:
Fivere venne csak zokon a romancot!
Leverne rolam e rokon a zomancot.

Ezek után a

```
orlando% grep zokon romeo
```

Csak az első sort találja meg. Az alábbi parancs viszont mindkét sort kiírja:

```
orlando% grep '[zr]okon' romeo
```

mert a [zr]okon reguláris kifejezésre mind a rokon, mint a zokon szó illeszkedik. Ezek után lássuk, milyen feltételek szerepelhetnek a reguláris kifejezésekben:

`c`

Bármely közönséges karakter illeszkedik saját magára.

`\c`

Kikapcsolja a *c* speciális karakter speciális jelentését. Akkor használatos, ha történetesen épp speciális karaktereket szeretnénk keresni.

`^`

A mintát a sor elejére igazítja. Csak azok a sorok illeszkednek rá, amelyek a ^ jel utáni reguláris kifejezésre illeszkedő szóval kezdődnek.

`$`

Ugyanazt csinálja, mint az előző, annyi különbséggel, hogy a mintát a sor végére igazítja.

`.`

Az újsor kivételével minden karakter illeszkedik rá.

`[...]`

A szögletes zárójelek közé zárt karakterek bármelyike illeszkedik rá.

`[^...]`

A szögletes zárójelek közé zárt karakterek KIVÉTELÉVEL bármelyik karakter illeszkedik rá.

`[n-n]`

A megadott tartományon belül eső karakterek bármelyike illeszkedik rá.

`*`

A csillag előtt álló karakter akárhány előfordulása (nulla is!) illeszkedik rá.

Nézzünk pár példát! A reguláris kifejezéseket idézőjelek közé kell tenni; ennek magyarázatát a példák után találjuk.

```
orlando% grep '[tf]arka'
```

Kiírja az összes olyan sort, amelyben a tarka, vagy a farka szó előfordul.

```
orlando% grep '^\^' kalap
```

Kiírja az összes olyan sort a kalap nevű fájlból, amely ^ jellel kezdődik. (Figyeljük meg a ^ jel használatát! Az első jelenti azt, hogy az utána következő kifejezésnek a sor elején kell lennie, a második pedig maga a keresendő karakter, amelyet most \ jellel hatástalanítunk, hiszen különleges karakter.)

```
orlando% ls -l | grep '^d.......x'
```

Ez egy bonyolult, de nagyon praktikus példa. Az ls -l parancs kimenetéből azokat a sorokat írjuk ki, amelyek eleget tesznek a következő feltételeknek:

- d betűvel kezdődnek
- második-kilencedik karakterük bármi lehet
- tizedik karakterük x

Könnyű rájönni, hogy így azon alkönyvtárak listáját kapjuk, amelyekbe mindenki beléphet.

```
orlando% ls -l | grep '[^.xdh]$'
```

Megint az ls parancs kimenetében keresgélünk; ezúttal azokat a fájlokat szűrjük ki, amelyek NEM .xdh-ra végződnek.

Idézőjelek

Big problem: sajnos a reguláris kifejezések különleges karaktereit a shell is értelmezi, méghozzá a saját szabályai szerint. Ez alapjában véve hasznos tulajdonság, de nem most, ezért védekeznünk kell ellene. Ezt úgy tehetjük meg, hogy a reguláris kifejezést egyszeres normál idézőjelek (') közé zárjuk. A shell ekkor a idézőjelek közötti részt változatlan formában adja át a grep parancsnak.

Majdnem ugyanez történik, ha egyszeres idézőjelek helyett kétszerest (") használunk. A különbség annyi, hogy a shell ilyenkor megnézi, hogy van-e a stringben hivatkozás shell változóra. Ha van, akkor annak az értékét behelyettesíti és úgy adja tovább a kifejezést a grepnek.

Van egy harmadik fajta idézőjel is, a visszafele döntött idézőjel (`). Az ilyen jelek közé zárt kifejezést a shell megpróbálja parancsként lefuttatni és a végrehajtás eredménye kerül át a grephez.

Ennek szellemében:

```
orlando% cat >animals
$eger (ez egy gazdag eger)
fakutya
vasmacska
Microsoft mouse
<Ctrl-d>
```

Ezzel létre is hoztuk az „adatbázist”, amin most gyakorlatozni fogunk. A shell változók kezelésének kipróbálására hozzunk létre egy „eger” nevű változót:

```
orlando% set eger=mouse
orlando% grep '$eger' animals
$eger (ez egy gazdag eger)
orlando% grep "eger" animals
Microsoft mouse
orlando%
```

Látható, hogy míg az egyszeres idézőjeleknél a grep a $eger reguláris kifejezést kapta meg, addig a kétszeres idézőjelek használata esetén a shell változó értékét, azaz a mouse szót - ezért jelent meg a második grep parancs végrehajtása után a „Microsoft mouse” sor.

Nézzük most a ` jelet! A végrehajtandó parancs legyen az echo, írassuk ki vele a kutya szót, s ezt adjuk át a grepnek!

```
orlando% grep `echo kutya` animals
fakutya
orlando%
```

Fájlok keresése

Fájlok keresésére a Unixban find nevű program szolgál. Szintaxisa:

```
find keresési-útvonalak kifejezések
```

Nézzünk néhány példát!

Tegyük fel, hogy egy valami.o nevű fájlt keresünk, amely valahol a home directorynkban, vagy az abból nyíló alkönyvtárak egyikében van. Ezt így találhatjuk meg:

```
orlando% find $HOME -name valami.o -print
```

A -name kapcsoló után kell megadni a keresett fájl nevét. Természetesen nem egyértelmű nevet is megadhatunk a * és a ? segítségével, de ilyenkor a nevet ' jelek közé kell tenni. A -print kapcsoló azt mondja meg a find programnak, hogy ha talált olyan fájlt, ami megfelel a keresési feltételek, akkor írja ki a nevét a teljes elérési útjával együtt.

A keresés helyeként megadhatunk több könyvtárat is: ilyenkor mindegyiket végignézi, az összes alkönyvtárával együtt.

Egyszerre több keresési feltételt is megadhatunk, ehhez azonban zárójeleket kell használnunk, amelyeket meg kell védenünk attól, hogy a shell saját belátása szerint értelmezze őket. A következő példa megkeresi az aktuális könyvtárban és az abból nyíló alkönyvtárakban található .c-re és .o-ra végződő nevű fájlokat.

```
orlando% find . \( -name '/*.c' -o -name '*.o' \) -print
```

Az egész logikus, bár első ránézésre kissé kuszának tűnik. Derítsünk fényt a homályra: A pont (.) jelenti a keresési útvonalat, jelen esetben az aktuális könyvtárat. Több könyvtárat is megadhatunk, szóközökkel elválasztva. Ezután következik a zárójel, amit -a grepnél tanultak alapján- a \ jellel védünk meg a shelltől. A -name kapcsolók már ismertek. A -o mondja meg a findnak, hogy a két -name-val előírt feltételt hozza vagy kapcsolatba; azaz keresse meg mindazon fájlokat, melyek vagy az egyik, vagy a másik (vagy mindkét) feltételnek eleget tesznek.

Nézzünk most egy fokkal bonyolultabb példát. A korábbi leckékből már tudjuk, hogy a home directoryban lehet egy .plan nevű fájl, aminek tartalma megjelenik a képernyőn, ha valaki lefingerel minket. Nézzük végig, hogy kinek van ilyen .plan fájlja! A megtalált .plan fájlokat írassuk ki a képernyőre!

```
orlando% find /usr1/public/users -name '.plan' -print -exec cat {} \;
```

Feltételezzük, hogy a felhasználók home könyvtárai a /usr1/public/users alkönyvtárból nyílnak. A .plan nevet idézőjelek közé tettük, hogy a shell ne értse félre a pontot. Újdonság a -exec kapcsoló, az ez után megadott parancs hajtódik végre minden alkalommal, mikor a find talál valamit. Ebben az esetben minden megtalált .plan fájlnál a find átadja a .plan nevét elérési útvonalával együtt a cat parancsnak.

A paraméterlista a -exec kapcsolónál kezdődik és a pontosvesszőnél (;) ér véget. A {} szimbólummal lehet hivatkozni a find által megtalált fájlra. A cat-nak jelen esetben nincs paramétere, de ha az rm parancsot hajtanánk végre, megadhatnánk a -i kapcsolót, amire a shell minden megtalált fájl törlése előtt rákérdezne szándékunk komolyságára:

```
orlando% find /usr1/public/users -name '*.gif' -print -exec rm -i {} \;
```

Megjegyzés: e példához teljesen hasonló paraméterezésű find parancsot használnak a fasiszta típusú rendszergazdák a felhasználók alkönyvtáraiban található több megabyte-os .gif kiterjesztésű (általában pucér lányokat ábrázoló) digitalizált képek automatikus törlésére.

A parancs végrehajtása során melléktermékként több oldal hibaüzenetet kapunk, mivel a find megpróbál minden alkönyvtárba belelépni, és ha ez nem sikerül neki (mert az alkönyvtár le van tiltva), akkor a „Permission denied” üzenettel szórakoztat minket. Szerencsére a standard error csatornát - s vele együtt a hibaüzeneteket is - át lehet irányítani. Erre a célra most a /dev/null egység látszik a legalkalmasabbnak, ez ugyanis nyomtalanul elnyeli a neki küldött karaktereket. Keressük meg a gépen található összes C programot! A megoldás sh-ban így néz ki (A 2-es azt jelenti, hogy most kivételesen nem a standard outputot (1), hanem a standard error (2) csatornát irányítjuk át):

```
orlando% sh
$ find / -name '*.c' -print 2>/dev/null
$ <Ctrl-d>
orlando%
```

Az sh-t itt csak a keresés idejére indítottuk el, mert nem biztos, hogy az alapértelmezett shellünkben ugyanígy kell átirányítani a standard error csatornát (próbáljuk ki! Ha nem működik, nézzünk utána a manualban, hogyan kell csinálni!).

Keresés és csere

Nagyon gyakori művelet, hogy egy szövegben valamilyen szót szeretnénk egy másikra cserélni. Ezt a leggyorsabban a sed nevű programmal hajthatjuk végre, az alábbi szintaxis szerint:

```
orlando% sed 's/mit/mire/g' hol >hova
```

A fenti parancs végignézi a „hol” fájlt, kicseréli benne az összes „mit” szót „mire”-re és az eredményt a „hova” nevű fájlba (átirányítás-jel és fájlnév megadása nélkül a képernyőre) írja. A sed egy nagytudású szövegszerkesztő, de sajnos szinte lehetetlen kezelni, ezért csak a legelvetemültebb buherátoroknak javasoljuk, hogy parancsait elsajátítsák. A mindennapi életben elég, ha a fenti példát megjegyezzük, valamint azt, hogy „hol” és a „hova” fájlként SOHA ne adjuk meg ugyanazt a nevet!!

Mezők kiemelése a szövegfájl soraiból

A grep parancsnál említettük, hogy a szavakat mezőknek is hívjuk. Ha egy szövegfájlt táblázat szerű (mint amilyen az e-mail címeket tartalmazó .addressbook fájl), akkor megesik, hogy a sorokból csak bizonyos szavakat szeretnénk kiemelni. Erre az awk program használható. Az awk végigolvassa a megadott fájl sorait, és egy speciális programozási nyelven leírt műveleteket végez rajta. Ez nagyon misztikusan hangzik; itt csak azt mutatjuk meg, hogy hogyan lehet egy szövegfájl mezőit kinyomtatni. Íme:

```
orlando% awk '{print $1 $2}' .addressbook
```

Az idézőjelek között található a „program”, ami most egy print utasításból és két mezőhivatkozásból áll. Az awk a kimenetén kiírja a .addressbook fájl minden sorának első és második mezőjét, más szóval a táblázat első két oszlopát.

Az awk sokkal bonyolultabb, mint amire egy átlagos felhasználónak élete során szüksége van. A kíváncsi buherátor-jelölteknek ismét azt javasoljuk, hogy olvassák szorgalmasan az awk parancs man oldalát.

Feladatok

- Mit csinál a következő Unix parancsokból összerakott cső?

  ```
  % rm -rf `du -s * | sort -rn | head -1 | awk '{print $2}'`;
  ```

- A mail spooler fájlban a sor elején található „From” szó jelzi a levél elejét. Állapítsuk meg a grep és a wc segítségével, hogy hány darab levél van a postaládánkban!
- Mi történik, ha a sed-del végzett keresésnél ugyanazt a nevet adjuk meg „hol”-ként és „hova”-ként is? Miért?
- Írassuk ki a last parancs kimenetéből azoknak a felhasználóknak a username-jét és rendszerben eltöltött idejét, akik a tty1 terminálról jelentkeztek be! (Csak ezt a két adatot írassuk ki!)

Kilencedik lecke

„Az igazi buherátor soha nem veszíti el a fejét. Mindig van róla egy tartalék másolata mágnesszalagon.”

Adataink archiválása

Ha huzamosabb ideig dolgozunk Unix alatt, előbb vagy utóbb olyan sok fájlunk lesz, hogy már nem tudunk eligazodni köztük. További nehézséget jelent, ha archiválni szeretnénk az adatainkat, azaz a ritkán használt fájlokat összetömöríteni, hogy ne foglaljanak el annyi helyet.

Több fájl összefűzése egyetlen állományba

Az egyik legáltalánosabban használt archiváló eszköz a tar program. Eredeti feladata az, hogy a megadott fájlokat mágnesszalagra (streamer) írja, de alkalmas kapcsoló segítségével a mágnesszalag helyett fájlba is írathatunk vele. A tar nagyon egyszerű program, valójában nem csinál mást, mint a megadott fájlokat összefűzi egyetlen nagy fájlba. (Természetesen elhelyez néhány vezérlő információt is, másképp a „betarolt” adatokat később nem lehetne „kitarolni”.)

A tar program általános szintaxisa:

```
tar key [directory] [tapefile] [blocksize] [name...]
```

Az egyszerűség kedvéért a továbbiakban feltételezzük, hogy nem mágnesszalagra, hanem fájlba mentünk. (Mágnesszalagra mentéshez tudnunk kell egyet s mást a hardverről is.) A fájlok „betarolása” a következő módon történik:

```
tar cf tarfile file1, file2, ...
```

a c kapcsoló utasítja a programot, hogy új tarfile-t hozzon létre (create), az f kapcsoló jelzi, hogy a mentés fájlba történik (ha ezt nem adjuk meg, akkor automatikusan a mágnesszalagra kezd írni, még akkor is, ha nincs is streamerünk). A file1, file2, ... fájlok kerülnek be a tarfile-ba. Természetesen (mint mindenhol) itt is használhatjuk a * és ? karaktereket.

Néhány további hasznos kapcsoló:

```
t
```

megmutatja a tarfile-ban található fájlok neveit.

```
v
```

„beszédes” üzemmódra állítja a programot (ilyenkor a tar mindig kiírja, hogy mit csinál éppen. Ha a v kapcsolót a t-vel együtt használjuk, hosszú listát (long list, olyan mint amit az ls -l csinál) kapunk a tarfile (vagy a mágnesszalag) tartalmáról.

```
r
```

Új fájlokat fűz hozzá a már létező tarfile-hoz.

```
x
```

Kitarolja a tarfile-ben lévő fájlokat.

```
xw
```

Kitarolja a tarfile tartalmát, de minden fájlra rákérdez, hogy valóban létrehozza-e.

Nézzünk meg egy példát: az alábbi alkönyvtárban szeretnénk néhány fájt összefogni egyetlen állományba:

```
orlando% ls -l
total 410
-rwxr--r-- 1 stsmork iit 373    Jul 7 08:45 automail
-rw------- 1 stsmork iit 643    Jul 7 08:46 login.c
-rw------- 1 stsmork iit 643    Jul 7 08:45 logout.c
-rwx------ 1 stsmork iit 25     Jul 7 08:45 openwin
-rw-r--r-- 1 stsmork iit 1286   Jul 7 08:46 prg.c
-rwxr--r-- 1 stsmork iit 467    Jul 7 08:45 search
-rwx--x--x 1 stsmork iit 94208  Jul 7 08:45 unzip
-rwx--x--x 1 stsmork iit 110592 Jul 7 08:45 zip
```

Első lépésként „taroljuk be” az összes C nyelvű forrás fájlt egy source.tar nevű állományba. A következő sorban a már létező fájlhoz hozzáfűzünk két újat (a zip és unzip nevűeket), végül megnézzük a keletkezett .tar fájl tartalmát:

```
orlando% tar cf source.tar *.c
orlando% tar rf source.tar zip unzip
orlando% tar tvf source.tar
rw------- 321/113 643     Jul 7 08:46 1994 login.c
rw------- 321/113 643     Jul 7 08:45 1994 logout.c
rw-r--r-- 321/113 1286    Jul 7 08:46 1994 prg.c
rwx--x--x 321/113 110592  Jul 7 08:45 1994 zip
rwx--x--x 321/113 94208   Jul 7 08:45 1994 unzip
```

A kicsomagolás a következő módon történik:

```
orlando% tar xvf source.tar
x login.c, 643 bytes, 2 blocks
x logout.c, 643 bytes, 2 blocks
x prg.c, 1286 bytes, 3 blocks
x zip, 110592 bytes, 216 blocks
x unzip, 94208 bytes, 184 blocks
```

Tömörítés

Ha archiválni szeretnénk adatainkat, akkor célszerű tömöríteni őket. Erre a legegyszerűbb mód a (kissé fatengelyes) compress program használata. A compress mindössze egyetlen fájlt tud tömöríteni, így ha sok fájlt akarunk archiválni, akkor először be kell tarolnunk őket, majd a tar fájlt összenyomnunk. A sűrítés befejezte után az eredeti fájl letörlődik, helyette az összenyomott változat marad meg, .Z-re végződő névvel.

Folytatva az előző példát:

```
orlando% ls -l source.tar
-rw-r--r-- 1 stsmork iit 215040 Jul 7 08:48 source.tar
orlando% compress source.tar
orlando% ls -l source.tar.Z
-rw-r--r-- 1 stsmork iit 120811 Jul 7 08:48 source.tar.Z
```

Figyeljük meg az eredeti és a tömörített fájl mérete közötti különbséget!

A kicsomagolás az uncompress programmal történik. Tipikus eset, hogy egy archivált fájl neve valahogy így néz ki: valami.tar.Z - ebből következtethetünk arra, hogy az adatokat előbb „betarolták”, majd a tar fájlt összenyomták a compress segítségével. (A mérföld hosszúságú fájlnevek beírását elkerülhetjük a * karakter alkalmas használatával.)

```
orlando% uncompress so*Z
orlando% ls -l source.tar
-rw-r--r-- 1 stsmork iit 215040 Jul 7 08:48 source.tar
```

Természetesen több más tömörítőprogram is létezik, például a gzip/gunzip páros, vagy a PC-s világból jól ismert zip/unzip és arj programok. Ezek sokkal okosabbak mint a compress + tar páros, de nem minden rendszeren találhatók meg, ezért most nem beszélünk róluk. (A nyilvános adatbázisokban található programcsomagokat legtöbbször a tar és a compress programok segítségével archiválják).

Fájlok küldése e-mailben

Technikai okokból e-mailen általában nem tudunk bináris fájlokat küldeni, mert az átvitel legtöbb helyen hétbites. A megoldást az jelenti, hogy az e-mailben elküldendő bináris fájlokat alkalmas program segítségével átkonvertáljuk olyan formára, hogy az csak hétbites karaktereket tartalmazzon (pontosabban: olyan karaktereket, amelyek ASCII kódja kisebb mint 127).

Az intelligensebb levelezőprogramok a szöveg mellett bináris fájlokat is tudnak küldeni mellékletként (attachment). Ilyenkor a levelezőprogram automatikusan elvégzi a szükséges kódolást, de erről a címzett levelezőprogramjának is tudnia kell. Ha ez nem így van, akkor a címzett kénytelen kézzel dekódolni a bináris állományt.

Tételezzük fel, hogy ostoba levelezőprogramunk van és a bináris fájlokat kénytelenek vagyunk kézzel kódolni. Ezt az uuencode programmal tehetjük meg. Paraméterezése:

```
uuencode kódolandó_fájl dekódolt_fájl
```

A kimenet alapértelmezésben a standard outputra kerül, amit fájlba irányíthatunk a > segítségével. Kódoljuk el a korábbi példában létrehozott soruce.tar fájlt úgy, hogy a dekódolás után létrejött fájl neve src.tar legyen.

```
orlando% uuencode source.tar src.tar >source.tar.uu
orlando% ls -l source*
-rw-r--r-- 1 stsmork iit 215040 Jul 7 08:48 source.tar
-rw-r--r-- 1 stsmork iit 296302 Jul 8 09:01 source.tar.uu
```

Figyeljük meg, hogy a kódolt fájl valamivel hosszabb, mint az eredeti. Íme a kódolt fájl első néhány sora:

```
begin 644 src.tar
M``````````````````````C:6YC;'5D92`\<W1D:6\N:#X*"B-D969I;F4@
M5%)512`@,0HC9&5F:6YE($9!3%-%(#`*"D9)3$4@*F9P.PH*;6%I;B@I"GL*
M8VAA<B!C:"P@96YD(#T@5%)513L*;&]N9R!C=7)R96YT+"!N97AT.PH*(&EF
```

Világosan látszik, hogy ez emberi fogyasztásra alkalmatlan. A dekódolás az uudecode programmal történik:

```
orlando% uudecode source*uu
orlando% ls -l s*
-rw-r--r-- 1 stsmork iit 215040 Jul 7 08:48 source.tar
-rw-r--r-- 1 stsmork iit 296302 Jul 8 09:01 source.tar.uu
-rw-r--r-- 1 stsmork iit 215040 Jul 8 09:01 src.tar
```

Figyeljük meg, hogy a dekódolt program neve nem soruce.tar, hanem src.tar, mivel a kódoláskor így adtuk meg. Célszerű a kódolt fájl nevében valamilyen módon jelezni, hogy uuencode-olt fájlról van szó (általában odabiggyesztenek egy .uu-t a fájlnév végére). Gyakoriak az olyan archivált fájlok, mint pl.: valami.tar.Z.uu, amiből következtethetünk arra, hogy a fájlt milyen programok segítségével kell kicsomagolni.

Feladatok

- Készítsünk egy alkönyvtárat a home directorynkban és másoljunk bele néhány fájlt.
- Lépjünk be az alkönyvtárba és hozzunk létre két újabb alkönyvtárat, amibe ugyancsak tegyünk be valamilyen fájlokat.
- Visszatérve a home directoryba, tar-oljuk be az 1. pontban létrehozott alkönyvtárat.
- Nézzük meg az előbb létrehozott fájl tartalmát.
- Adjunk hozzá két újabb (tetszőleges) fájlt a 3. pontban létrehozott fájlhoz.
- Sűrítsük össze és uuencode-oljuk a fájlt.
- Töröljük le az 1. pontban létrehozott alkönyvtár struktúrát.
- Bontsuk ki a 6. pontban létrehozott archív fájlt. Vajon a kitarolás után meglesznek-e az alkönyvtárak, amiket a 2. pontban hoztunk létre?

Tizedik lecke

„A Unix az az operációs rendszer, melyet buherátorok írtak saját maguk és más buherátorok számára. Ezért a Unix jelszava: Az amatőrök dögöljenek meg!”

Tippek és trükkök

Ez a lecke azok számára készült, akik már egy kissé otthonosak a Unixban, ismerik a fontosabb parancsokat és kapcsolóikat, de tanácstalanok, hogy tulajdonképpen hogyan és mire is használják őket. Bátorításul bemutatunk néhány tippet.

Megkapta-e a címzett a levelünket?

A Unix a beérkező leveleket egy fájlban tárolja, melynek neve egyezik a felhasználó accountjának nevével. Ezek az inbox fájlok a legtöbb helyen a /usr/spool/mail alkönyvtárban vannak. (Kivétel a Silicon Graphics gépek, ahol a /usr/mail-ben. :-(A saját inboxunkon kívül természetesen senki másét nem tudjuk elolvasni, de az ls paranccsal meg tudjuk nézni, így láthatjuk, hogy levelezőpartnerünknek van-e levele a postaládájában. Ha ezt a vizsgálatot levelünk elküldése előtt és után is elvégezzük, meggyőződhetünk róla, hogy a levél valóban megérkezett. (Némely Unix rendszerek hajlamosak arra, hogy ha a címzett disk quota-ja kimerül, akkor a hozzá beérkező leveleket nyomtalanul eltüntetik.)

A Unix három különféle időpontot tárol a fájlokról, amelyet az ls parancs zárójelben megadott kapcsolóival lehet megtudni:

Last Modificaton - utolsó módosítás (write)

```
(ls -l)
```

Last Access - utolsó művelet (írás, olvasás, másolás, stb.)

```
(ls -lu)
```

Last Inode Change - utolsó inode módosítás

```
(create, chmod, stb.) (ls -lc)
```

E három időpont egymáshoz viszonyított értékéből levonhatók bizonyos - nem okvetlenül helytálló - következtetések. (Például: ha a Last Access későbbi, mint a Last Write, akkor levelünket valószínűleg elolvasták, stb.)

Parancsok végrehajtása bejelentkezés nélkül

Gyakori eset, hogy több gépre is van accountunk, be vagyunk jelentkezve az egyik helyen és valamiért szeretnénk egy parancsot végrehajtani a másik gépen. (Például szeretnénk kiadni egy w parancsot, hogy lássuk, ki milyen processzt futtat a másik gépen.) Ehhez be kellene

jelentkeznünk a telnet vagy az rlogin segítségével, beírni a jelszavunkat, begépelni a parancsot és végül kilépni. Van azonban egy kényelmesebb megoldás, az rsh parancs (remote shell). Az rsh a neki megadott parancsot nem nálunk, hanem a távoli gépen hajtja végre. Tegyük fel, hogy az ind12 gépen vagyunk és meg akarjuk nézni, ki milyen processzt futtat az ind03-on. Íme:

```
ind12% rsh ind03 w
```

Feltesszük, hogy az ind03 gépen ugyanaz a username-ünk, mint az ind12-n. Ha ez nem így van, akkor a parancsnak meg kell adnunk azt az username-t, amelyiken a parancsot végre szeretnénk hajtani. Ha például a gold.uni-miskolc.hu gépen szeretnénk végrehajtani egy last parancsot és ott valamilyen szeszély folytán xkrm5687 a username-ünk, akkor a megfelelő parancs így néz ki:

```
ind12% rsh gold.uni-miskolc.hu -l xkrm5687 -n last
```

Megjegyzés: Sajnos az rsh parancs kapcsolóit a különféle Unix rendszerekben kicsit másképp hívják. A mi példáink Silicon Graphics gépekre vonatkoznak; aki ettől eltérő gépen próbálkozik, az nézzen utána az rsh parancs kapcsolóinak a man-ban!

Ahhoz, hogy az rsh parancs csakugyan végre is hajtsa a kívánt parancsot a távoli gépen és ne a „Permission denied” sztereotip üzenettel térjen vissza, kell, hogy a távoli gépen legyen egy .rhosts fájlunk a kiindulási könyvtárunkban (home directory). A .rhost fájl írja le, hogy mely hostokról milyen username-vel lehet rsh-val belépni. Nézzük az ind12 gépen található alábbi minta .rhosts fájlt:

```
ind03
ind11
gold xkrm5687
suma1 stsmork
```

Ebből a következők derülnek ki. Az ind03 és ind11 gépekről be tudunk rsh-zni az ind12-re feltéve, hogy ugyanaz a username-ünk, mint az ind12-n. A gold-ról csak az xkrm5687 nevű user, a suma1-ről pedig csak stsmork tud távolról parancsokat végrehajtani az ind12-n lévő accountunkon, emennyiben ez a szándékuk.

VIGYÁZAT: Ha valakit felveszünk a .rhosts fájlba, akkor attól a rendszer - Unixban merőben szokatlan módon - nem fogja kérdezni a jelszót, ezért legyünk nagyon óvatosak, mikor .rhosts fájlt készítünk!

A .rhost fájl létrehozásának van egy további kényelmes következménye is: az rlogin parancs nem kérdezi meg a jelszót, mikor olyan gépre tépünk be, ahol a .rhosts fájlban meg van adva a gépünk neve. (Természetesen itt is feltétel, hogy a két gépen azonos legyen a username-ünk, vagy ha nem, akkor a másik username legyen feltüntetve a .rhosts fájlban.)

Talk beszélgetések logozása

A tee olyan Unix program, amely a bemenetére csövezett adatokat megjeleníti a képernyőn (standard kimeneten), és egyúttal a paraméterként megadott fájlba is kiírja. Ha például szeretnénk rögzíteni egy beszélgetés szövegét, amelyet a suma2 gépen bonyolítunk le, akkor a kapcsolat felvétele előtt tegyük a következőket:

```
suma1% telnet suma2 | tee duma.log
```

Ettől kezdve minden képernyőn megjelenő karakter egyúttal a duma.log fájlba is bekerül, egészen addig, míg a <Ctrl-d> leütésével ki nem lépünk a suma2 gépről. Miután így bejutottunk a suma2-re, a szokásos módon talkoljuk meg a beszélgetőpartnerünket és csevegjünk vele.

Később bármikor visszajátszhatjuk a beszélgetés szövegét a duma.log fájlból a more vagy a cat segítségével.

Foto ftp

A „foto ftp”-nek csúfolt eljárással olyan gépekről hozhatunk el bináris fájlokat, ahová valamilyen okból nem tudunk ftp-vel bejutni. (Ennek általában az az oka, hogy a két gép között annyira lassú a kapcsolat, hogy a hagyományos ftp program timeout hibával elakad.) A módszer a következő lépésekből áll:

- Jelentkezzünk be interaktívan a távoli gépre.
- A távoli gépen uuencode-oljuk be az áthozni kívánt fájlt.
- Kapcsoljuk be a logfájl szolgáltatást a lokális gépen. (Ha PC-ről jelentkeztünk be, akkor ez általában nem gond, mert minden terminál emulátor képes fájlba másolni a képernyőn megjelenő szöveget. Ha a lokális gép Unixos, akkor használhatjuk az előző pontban leírt módszert.)
- A cat segítségével írassuk ki a képernyőre a 2. pontban létre hozott uuencode-olt fájlt. (A 2. pont egyébként kihagyható, mert az uuencode program alapértelmezésben a képernyőre küldi kimenetét.)
- Lépjünk ki a távoli gépről és a lokális gépen keletkezett log fájlt uuedecode-oljuk ki.

Átjelentkezés egyik accountról a másikra

Előfordulhat, hogy valamilyen okból át akarunk jelentkezni egy másik accountra. (Pl. több accountunk is van ugyanazon a gépen). Ilyenkor elvileg újra be kellene telnet-elnünk, beírni a másik username-t, megadni a másik jelszót, stb. Van egyszerűbb út is; ha pl. éppen orlando vagyunk és stsmork szeretnénk lenni, akkor:

```
zeus% su - stsmork
```

A parancs megkérdezi stsmork jelszavát és - hacsak el nem gépeltük - átjelentkezik az stsmork accountra. A mínusz jel és az username közé KELL a szóköz karakter!! Az su parancs használatát a rendszer biztonsági okokból egy sulog nevű fájlban tárolja.

FIGYELEM! Soha ne adjuk ki paraméterek nélkül az su parancsot! Ez ugyanis a su root-nak felel meg, ami betörési kísérletnek számít, amit a rendszeradminisztrátorok rossznéven szoktak venni!

Tizenegyedik lecke

A Unix shell humorérzéke

A Unix operációs rendszer kimagasló intelligenciájának szórakoztató „mellékhatása”, hogy a shell program meglepően jó humorérzékkel bír. Mindössze a megfelelő parancsokat kell kiadnunk; ennek illusztrálására álljon itt néhány példa. (A példák a Minnesotai Egyetem gopher szerveréről származnak.)

```
% ar m God
ar: God does not exist

% "How would you rate Reagan's incompetence?
Unmatched ".

% [Where is Jimmy Hoffa?
Missing ].

% ^How did the sex change^ operation go?
Modifier failed.

% If I had a ( for every $ Congress spent, what would I have?
Too many ('s.

% make love
Make: Don't know how to make love. Stop.

% sleep with me
bad character

% got a light?
No match.

% man: why did you get a divorce?
man:: Too many arguments.

% ^What is saccharine?
Bad substitute.

% %blow
%blow: No such job.
```

```
% \(-
(-: Command not found.

% sh

$ PATH=pretending! /usr/ucb/which sense
no sense in pretending!

$ drink <bottle; opener
bottle: cannot open
opener: not found

$ mkdir matter; cat >matter
matter: cannot create
```